z/VM
7.4

OpenExtensions Commands Reference

IBM

Note:

Before you use this information and the product it supports, read the information in "Notices" on page 549.

Contents

Figures

x

Tables

About This Document

This document describes the IBM z/VM OpenExtensions shell commands and utilities. These commands and utilities provide z/VM users with application development tools and an interactive shell interface based on open systems standards. Using the OpenExtensions shell, you can enter shell commands, write shell scripts, and work with OpenExtensions byte file system (BFS) files.

This document also describes a subset of CMS commands known as OPENVM commands that you can use to obtain OpenExtensions services.

Intended Audience

This information is for anyone who plans to develop OpenExtensions applications and needs detailed reference information about OpenExtensions shell commands and utilities and CMS OPENVM commands.

Conventions Used in This Document

The following conventions are used in this document.

Escape Character Notation

When you see the following notation:

enter <EscChar-C>

it should be interpreted as:

type the EscChar, which by default is the ¢ (cent sign) and then type the **C** character.
Press ENTER after typing these characters.

Note: To change the escape character to something other than the cent sign, see the BPX1TSX service in *z/VM: OpenExtensions Callable Services Reference*.

Case-Sensitivity

The OpenExtensions shell commands and CMS OPENVM commands are case-sensitive and distinguish characters as either uppercase or lowercase. Therefore, FILE1 is not the same as file1.

Typography

The following typographic conventions are used:

Style	Use
BOLD	Bold uppercase is used for all command names (OPENVM SHELL) except the shell commands, statements (CLINKNAME), and references to a key that you would press (ENTER).
bold	Bold lowercase is used for shell commands (make).
variable	Lowercase italics is used to indicate a variable.
VARIABLE	Uppercase italics is used to indicate a shell environment variable.
example font	Example font is used to indicate file specifications (.profile, XEDIT PROFILE), directory names (/usr/lib/nls/charmap), and verbatim user input.

Syntax, Message, and Response Conventions

The following topics provide information on the conventions used in syntax diagrams and in examples of messages and responses.

How to Read Syntax Diagrams

Special diagrams (often called *railroad tracks*) are used to show the syntax of external interfaces.

To read a syntax diagram, follow the path of the line. Read from left to right and top to bottom.

- The ►►—— symbol indicates the beginning of the syntax diagram.

- The ——► symbol, at the end of a line, indicates that the syntax diagram is continued on the next line.

- The ►—— symbol, at the beginning of a line, indicates that the syntax diagram is continued from the previous line.

- The ——►◄ symbol indicates the end of the syntax diagram.

Within the syntax diagram, items on the line are required, items below the line are optional, and items above the line are defaults. See the examples in .

Table 1. Examples of Syntax Diagram Conventions	
Syntax Diagram Convention	**Example**
Keywords and Constants A keyword or constant appears in uppercase letters. In this example, you must specify the item KEYWORD as shown. In most cases, you can specify a keyword or constant in uppercase letters, lowercase letters, or any combination. However, some applications may have additional conventions for using all-uppercase or all-lowercase.	►► KEYWORD ►◄
Abbreviations Uppercase letters denote the shortest acceptable abbreviation of an item, and lowercase letters denote the part that can be omitted. If an item appears entirely in uppercase letters, it cannot be abbreviated. In this example, you can specify KEYWO, KEYWOR, or KEYWORD.	►► KEYWOrd ►◄
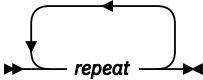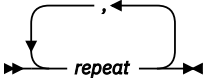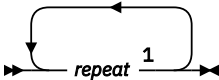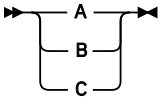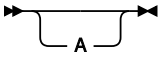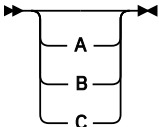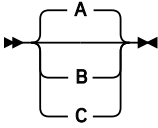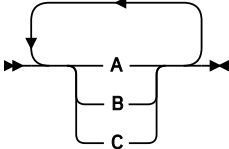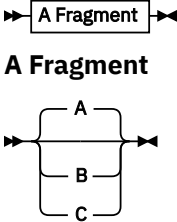
Table 1. Examples of Syntax Diagram Conventions (continued)	
Syntax Diagram Convention	**Example**
Symbols You must specify these symbols exactly as they appear in the syntax diagram.	***** Asterisk **:** Colon **,** Comma **=** Equal Sign **-** Hyphen **( )** Parentheses **.** Period
Variables A variable appears in highlighted lowercase, usually italics. In this example, *var_name* represents a variable that you must specify following KEYWORD.	►►— KEYWOrd —— *var_name* —►◄
Repetitions An arrow returning to the left means that the item can be repeated. A character within the arrow means that you must separate each repetition of the item with that character. A number (1) by the arrow references a syntax note at the bottom of the diagram. The syntax note tells you how many times the item can be repeated. Syntax notes may also be used to explain other special aspects of the syntax.	 Notes: [1] Specify *repeat* up to 5 times.
Required Item or Choice When an item is on the line, it is required. In this example, you must specify A. When two or more items are in a stack and one of them is on the line, you must specify one item. In this example, you must choose A, B, or C.	►►— A —►◄
Optional Item or Choice When an item is below the line, it is optional. In this example, you can choose A or nothing at all. When two or more items are in a stack below the line, all of them are optional. In this example, you can choose A, B, C, or nothing at all.	

Table 1. Examples of Syntax Diagram Conventions (continued)	
Syntax Diagram Convention	**Example**
Defaults When an item is above the line, it is the default. The system will use the default unless you override it. You can override the default by specifying an option from the stack below the line. In this example, A is the default. You can override A by choosing B or C.	
Repeatable Choice A stack of items followed by an arrow returning to the left means that you can select more than one item or, in some cases, repeat a single item. In this example, you can choose any combination of A, B, or C.	
Syntax Fragment Some diagrams, because of their length, must fragment the syntax. The fragment name appears between vertical bars in the diagram. The expanded fragment appears in the diagram after a heading with the same fragment name. In this example, the fragment is named "A Fragment."	 **A Fragment**

Examples of Messages and Responses

Although most examples of messages and responses are shown exactly as they would appear, some content might depend on the specific situation. The following notation is used to show variable, optional, or alternative content:

xxx
> Highlighted text (usually italics) indicates a variable that represents the data that will be displayed.

[ ]
> Brackets enclose optional text that might be displayed.

{ }
> Braces enclose alternative versions of text, one of which will be displayed.

|
> The vertical bar separates items within brackets or braces.

...
> The ellipsis indicates that the preceding item might be repeated. A vertical ellipsis indicates that the preceding line, or a variation of that line, might be repeated.

Where to Find More Information

For information about using the OpenExtensions shell and setting up OpenExtensions facilities on z/VM, see the *z/VM: OpenExtensions User's Guide*.

For a list of other z/VM publications, see the "Bibliography" on page 553.

Links to Other Documents and Websites

The PDF version of this document contains links to other documents and websites. A link from this document to another document works only when both documents are in the same directory or database, and a link to a website works only if you have access to the Internet. A document link is to a specific edition. If a new edition of a linked document has been published since the publication of this document, the linked document might not be the latest edition.

How to provide feedback to IBM

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. See How to send feedback to IBM for additional information.

Summary of Changes for z/VM: OpenExtensions Commands Reference

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line (|) to the left of the change.

SC24-6297-74, z/VM 7.4 (September 2024)

This edition supports the general availability of z/VM 7.4. Note that the publication number suffix (-74) indicates the z/VM release to which this edition applies.

SC24-6297-73, z/VM 7.3 (September 2023)

This edition includes terminology, maintenance, and editorial changes.

SC24-6297-73, z/VM 7.3 (September 2022)

This edition supports the general availability of z/VM 7.3. Note that the publication number suffix (-73) indicates the z/VM release to which this edition applies.

SC24-6297-01, z/VM 7.2 (September 2020)

This edition supports the general availability of z/VM 7.2.

Chapter 1. OpenExtensions Shell Commands

This chapter is an introduction to the OpenExtensions shell command descriptions.

Note: Although the POSIX standard distinguishes between a command and a utility, this book uses the term **command** for both.

Reading the Command Descriptions

Each shell command appears in alphabetic order.

The description for each command is divided into several sections, which are explained in the following paragraphs. Some of these sections apply only to a few command descriptions. Also, some command descriptions include special sections that are not explained here.

Format Section

The *Format* section provides a quick summary of the command's format, or syntax. The syntax was chosen to conform to general UNIX® usage. For example, here is the format of the **ls** command:

```
ls [-AabCcdFfgiLlmnopqRrstuWx1][pathname ...]
```

The format takes the form of a command line as you might type it into the system; it shows what you can type in and the order in which you should do it. The parts enclosed in square brackets are *optional*; you can omit them if you choose. Parts outside the square brackets *must* be present for the command to be correct.

The format begins with the name of the command itself. Command names always appear in **bold** font.

After the command name comes a list of options, if there are any. A typical OpenExtensions shell command option consists of a dash (**–**) followed by a single character, usually an uppercase or lowercase letter. For example, you might have **–A** or **–a**.

Note: The case of letters is important; for example, in the format of **ls**, **–a** and **–A** are *different* options, with different effects.

If you are going to specify several options for the same command, you can put all the option characters after the same dash. Or you can put each option after its own dash. Or you can rearrange the order of options. For example,

```
ls -A -a
ls -Aa
ls -a -A
ls -aA
```

are all equivalent.

The format line shows options in **bold** font. In the description of **ls**, all options are shown in one long string after the single dash. But another common option form is:

```
-x value
```

where **–x** is a dash followed by a character, and *value* provides extra information for using that option. For example, here are the formats of the **sort** command, which takes unsorted input and sorts it:

```
sort [-cmu]
[-o outfile]
[-t char]
[-y[n]]
[-zn]
[-bdfiMnr]
[-k startpos[,endpos] ] ...
[file ...]
```

1

```
sort [-cmu]
[-o outfile]
[-tchar]
[-yn]
[-zn]
[-bdfiMnr]
[+startposition[-endposition]] ...
[file ...]
```

You can see that there are two possibilities here; you would need to choose which of the two versions of **sort** met your requirements. In either possibility, however, we have the option:

```
-o outfile
```

This option tells the **sort** command where to save its sorted output. The form of the option is **–o**, followed by a space, followed by *outfile*. In a command format, anything appearing in *italic* font is a *placeholder* for information that you are expected to supply. Sometimes after the format, the kind of information expected in place of the placeholder is explained. In our **sort** example, *outfile* stands for the name of a file where you want **sort** to store its output. For example, if you wanted to store the output in the file **sorted.dat**, you would specify:

```
sort -o sorted.dat
```

(followed by the rest of the command).

The format for **sort** also contains an option of the form:

```
-tchar
```

This is similar to the option form we were just discussing, except that there is no space between the **–t** and *char*. As before, *char* in italics is a placeholder; in this case, it stands for any single character. If you want to use the **–t** option for **sort**, you just type **–t** followed immediately by another character, as in:

```
sort -t:
```

In this case, we use a colon (`:`) in the position of the placeholder *char*.

The end of the **sort** format is:

```
[file ...]
```

This means a list of one or more file names; the ellipsis (`...`) stands for repetitions of whatever immediately precedes it. Because there are square brackets around the previous list, you can omit the list if you like.

The format of **ls** ended in:

```
[pathname ...]
```

As you might guess, this means that an **ls** command can end with an optional list of one or more path names. What's the difference between this and our **sort** example? A path name (specified with *pathname*) can be the name of either a file or a directory; a file name (specified with *file*) is always the name of a file.

The order of items on the command line is important. When you type a command line, you should specify its parts in the order they appear in the command format. The exceptions to this are options marked with a dash (**–**); they do not have to be given in the exact order shown in the format. However, all the **–** options must appear in the correct *area* of the command line. For example, you can specify:

```
ls -l -t myfiles
ls -t -l myfiles
```

but you will not get correct results if you specify:

```
ls myfiles -l -t              ***incorrect***
```

or:

```
ls -l myfiles -t          ***incorrect***
```

and so on. If you enter the last example, for instance, **ls** interprets **–t** as the path name of a file or directory, and the command will try to list the characteristics of that item.

As a special notation, most OpenExtensions shell commands let you specify two dashes (**––**) to separate the options from the nonoption arguments; **––** means: "There are no more options." Thus, if you really have a directory named **–t**, you could specify:

```
ls –– –t
```

to list the contents of that directory.

Description Section

The *Description* section describes what the command does. For a particularly complex command, this section may be divided into a large number of subsections, each dealing with a particular aspect of the command.

The Description section often mentions the *standard input* (**stdin**) and the *standard output* (**stdout**). The standard input is usually the workstation keyboard; the standard output is usually the display screen. The process of *redirection* can change this. Redirection is explained in the *z/VM: OpenExtensions User's Guide*.

The shell differentiates between hex, octal, and decimal as follows:

- Any number that starts with 0x is hex.
- Any number that starts with 0 is octal.
- Any number that does not start with 0x or 0 is decimal.

Inside the Description section, the names of files and directories are presented in **bold** font. The names of environment variables are also presented in **BOLD** font, capitalized.

Options Section

The *Options* section describes each of the options used by the command.

Examples Section

The *Examples* section is present in many command descriptions, giving examples of how the OpenExtensions shell can be used. This book tries to give a mix of simple examples that show how the commands work on an elementary level and more complex examples that show how the commands can perform complicated tasks.

Before you try to run any of the examples in this book, you need to know that the OpenExtensions shell uses the EBCDIC *Latin1/Open System Interconnection Code Page* 01047. Characters entered on a workstation keyboard and passed to the shell by VM do not have the same hexadecimal encoding as the code page the shell uses. You may need to customize your keyboard so that those characters have the encoding the shell uses. See the *z/VM: OpenExtensions User's Guide* for more information about code page conversion, about using a keyboard with customized characters, and for a copy of code page 01047.

Environment Variables Section

The *Environment Variables* section lists the environment variables that affect the command, if any, and describes the purposes that those variables serve. For example, the **ls** command description lists two environment variables— **COLUMNS** and **TZ**—and informs you that **COLUMNS** is the terminal width and that **TZ** contains information about the local time zone.

Localization Section

The *Localization* section describes how the locale-related environment variables affect the behavior of the command. These environment variables allow you to access *locale* information, including alternate character sets; alternate numeric, monetary, and date and time formats; and foreign language translations of common messages. Locales make it easier for users around the world to use the shell and utilities.

The OpenExtensions Shell and Utilities supports the IBM-supplied locales listed in . User-generated locales using code page 1047 are also supported.

Table 2. Locales Supplied by OpenExtensions

Country	Language	Locale Name
Belgium	Dutch	Nl_BE.IBM-1047
Belgium	French	Fr_BE.IBM-1047
Canada	French	Fr_CA.IBM-1047
Denmark	Danish	Da_DK.IBM-1047
Finland	Finnish	Fi_FI.IBM-1047
France	French	Fr_FR.IBM-1047
Germany	German	De_DE.IBM-1047
Iceland	Icelandic	Is_IS.IBM-1047
Italy	Italian	It_IT.IBM-1047
Japan	English	En_JP.IBM-1027
Japan	Japanese	Ja_JP.IBM-939
Japan	Japanese	Ja_JP.IBM-1027
Netherlands	Dutch	Nl_NL.IBM-1047
Norway	Norwegian	No_NO.IBM-1047
Portugal	Portuguese	Pt_PT.IBM-1047
Spain	Spanish	Es_ES.IBM-1047
Sweden	Swedish	Sv_SE.IBM-1047
Switzerland	French	Fr_CH.IBM-1047
Switzerland	German	De_CH.IBM-1047
United Kingdom	English	En_GB.IBM-1047
United States	English	En_US.IBM-1047

For more information on locales, see *XL C/C++ for z/VM: User's Guide*.

Files Section

The *Files* section of the command lists any supplementary files (files not specified on the command line) that are referenced. Such files usually provide information the command needs; the command accesses these files during its operation. If the files cannot be found, the command issues a message to this effect.

Files documented in this section may be temporary files, output files, databases, configuration files, and so on.

The C/C++ runtime library supports a file naming convention of // (the file name can begin with exactly two slashes). However, the OpenExtensions Shell and Utilities *does not* support this convention. Do not use this convention (//) unless it is specifically indicated (as in the description for the **c89** command). The OpenExtensions Shell and Utilities does support the POSIX file naming convention, where the file name can be selected from a set of character values that excludes the slash and the null character.

The OpenExtensions Shell and Utilities supports the concept of the fully qualified file system root. This allows you to operate on and reference files that reside in file systems that are not part of your file system hierarchy. For a full description of the fully qualified file system root concept, refer to "Understanding Byte File System (BFS) Path Name Syntax" on page 368.

Usage Notes Section

The *Usage Notes* section gives additional notes for those using the shell. The purpose of this section is to provide important information that the reader should not overlook.

Exit Values Section

The *Exit Values* section presents the error messages that the shell may display, along with a description of what caused the message and a possible action you can take to avoid getting that message. Occasionally, this section refers you to another command description for more information on an error message.

This section also contains information about the exit status returned by the command. You can test this status to determine the result of the operation that the command was asked to perform.

Limits Section

The *Limits* section lists any restrictions on the operation of the shell. Some limits are implicit rather than explicit and may be lower than the explicitly stated limit.

Portability Section

The *Portability* section includes two types of information:

- Availability of a version of the command on existing UNIX systems (System V, BSD).
- Compatibility with industry standards—for example, the POSIX.2 Draft Standard or the X/Open Portability Guide, Issue 4 (XPG4).

Related Commands

The *Related Commands* section refers to other command descriptions that may contain information relevant to the command description you have just read. For example, consider the **head** command; by default, **head** displays the first 10 lines of each file given on the command line. Its *Related Commands* section refers you to **tail**, the command that displays the last 10 lines of a file.

Default File Permissions

When a shell command creates a new file (other than character special files), the default permission settings assigned to the file are **read** and **write** for the owner, and **read** for group and other. The exceptions to this rule are:

c89/cxx
Read, write, and execute for owner, group, and other

ln
Read, write, and execute for owner, group, and other (for **ln -s**)

cp
The permissions of the file that was copied.

These default permissions are subject to filtering by the **umask** in effect.

POSIX Conformance: The OpenExtensions shell is based on the KornShell that originated on a UNIX system. As implemented in OpenExtensions, this shell conforms to POSIX standard 1003.2-1992.

alias — Display or create a command alias

```
alias [–x] [name[=value] ...]
alias –t [name ...]
```

Purpose

When the first word of a shell command line is not a shell keyword, **alias** causes the shell to check for the word in the list of currently defined *aliases*. If it finds a match, the shell replaces the alias with its associated string value. The result is a new command line that might begin with a shell function name, a built-in command, an external command, or another alias.

When the shell performs alias substitution, it checks to see if the associated string value (specified by *value*) ends with a blank. If so, the shell also checks the next word of the command line for aliases. The shell then checks the new command line for aliases and expands them, following these same rules. This process continues until there are no aliases left on the command or recursion occurs in the expansion of aliases.

Calling **alias** without parameters displays all the currently defined aliases and their associated values. Values appear with appropriate quoting so that they are suitable for reinput to the shell.

Calling **alias** with parameters of the form *name=value* creates an alias for each function name you specify as *name* with the given string you specify as *value*.

If you are defining an alias where *value* contains a backslash character, you must precede it with another backslash. The shell interprets the backslash as the escape character when it performs the expansion. If you use double quotation marks to enclose *value*, you must precede each of the two backslashes with an additional backslash, because the shell escapes characters—that is, the shell does not interpret the character as it normally does—both when assigning the alias and again when expanding it.

To avoid using four backslashes to represent a single backslash, use single quotation marks rather than double quotation marks to enclose *value*, because the shell does not escape characters enclosed in single quotation marks during assignment. As a result, the shell escapes characters in single quotation marks only when expanding the alias.

Calling **alias** with *name* without any value assignment displays the function name (*name*) and its associated string value (*value*) with appropriate quoting.

Options

alias supports the following options:

–t

> Makes each *name* on the command line a *tracked alias*. Unlike regular aliases, tracked aliases do not shorten typing or provide command synonyms. Instead, the set of tracked aliases help shell performance by acting as a lookaside list or cache for commands that otherwise would be searched for on the search path ($PATH). Each entry in this cache maps a tracked alias to the path name of its corresponding executable file.
>
> When you establish tracked alias *name*, the shell resolves *name* (it finds out where its executable file is according to the search rule specified by $PATH) and sets up a mapping between *name* and the path name of its executable file.
>
> When you run a command that is a tracked alias, the shell does not search $PATH to find it. Instead, it determines the path name through lookup in the list of tracked aliases.
>
> Running **alias –t** without any specified names displays all currently defined tracked aliases with appropriate quoting.

When you enter the command **set –h**, each subsequent command you use in the shell automatically becomes a tracked alias.

–x

Marks each alias *name* on the command line for export. If you specify **–x** without any names on the command line, **alias** displays all exported aliases. Only exported aliases are passed to a shell that runs a shell script.

There are several aliases built into the shell:

```
alias functions="typeset –f"
alias hash="alias –t"
alias history="fc –l"
alias integer="typeset –i"
alias nohup="nohup"
alias r="fc –s"
alias stop="kill -STOP"
alias suspend="stop \$\$"
```

You can change or remove any of these aliases.

Examples

The command:

```
alias ls="ls -C"
```

defines `ls` as an alias. From this point onward, when you issue an `ls` command, it produces multicolumn output by default.

Localization

alias uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Usage Notes

1. This command is built into the shell.
2. Instead of using the **–x** option, you should define nonexported aliases in the **sh** command's **ENV** variable.

Exit Values

Possible exit values are:

0

Successful completion

1

Failure because an alias could not be set

2

Failure because of an incorrect command-line option

If you define **alias** to determine the values of a set of names, the exit value is the number of those names that are not currently defined as aliases.

Portability

UNIX KornShell, POSIX.2

alias is a built-in shell command.

The **–t** and **–x** options are extensions to the POSIX standard.

Related Commands

cd, **fc**, **let**, **nohup**, **set**, **sh**, **typeset**, **unalias**

ar — Create or maintain library archives

> ar **–d**[**–Ilv**] [**–F** *format*] *archive member*...
> ar **–m**[**–abIilsv**] [*posname*] [**–F** *format*] *archive member* ...
> ar **–p**[**–Ilsv**] [**–F** *format*] *archive member*...
> ar **–q**[**–clsv**] [**–F** *format*] *archive member* ...
> ar **–r**[**abcIilsuv**] [**–F** *format*] [*posname*] *archive member* ...
> ar **–t**[**Ilsv**] [**–F** *format*] *archive*[*member*...]
> ar **–u**[**–abcIklsv**] [**–F** *format*] [*posname*] *archive member* ...
> ar **–x**[**–CIlsTv**] [**–F** *format*] *archive* [*member*...] ...

Purpose

ar maintains archive libraries. The archive library is a collection of files, usually object files. Using **ar**, you can perform various operations on archive libraries, such as creating a new library, adding members to an existing library, deleting members from a library, extracting members from a library, and printing a table of contents for a library.

A library member is an arbitrary file. Normally, these files are object files or side files, suitable for use by a linkage editor. If any members of a library are object files, **ar** creates and maintains an external symbol index for link-editing.

Member names in an archive are only the final component of any path name. When creating a new library member (*member*) as given on the command line, **ar** uses the full path name given. When storing the member name in the library, or comparing a member name, **ar** uses only the final component.

Options

The format shows the seven main functions of **ar**, which are defined as follows:

–d

Deletes each named *member* from the archive and regenerates the symbol table.

–m

Moves the named archive member in the archive. The new position is specified by **–a**, **–b**, **i**, or *posname*. If a location is not specified, the member is moved to the end of the archive.

–p

Displays each *member* specified to the standard output (**stdout**). If you did not specify any members, **ar** displays all members.

–q

Quickly appends the specified *file* to the archive. With this option, **ar** does not check to see if *file* is already a member of the archive.

–r

Replaces or adds *file* to *archive*. If *archive* does not exist, **ar** creates it and prints a message. When **ar** replaces an existing member, the archive order is not changed. If *file* is not replacing a member, it is added to the end of the archive unless **–a**, **–b**, or **–i** is used. This option regenerates the symbol table.

–t

Displays a table of contents that lists members, or every member if *member* is not specified. **ar** prints a message for each member it doesn't find. By default, **ar** prints the member name for all selected members. With the verbose (**–v**) option, **ar** prints more information for all selected members.

–x

Extracts each specified *member* from the archive and copies it to a file. If *member* is specified as a full path name, it is copied to that path name. If no *member* is specified, all members are extracted. The archive remains unchanged.

The following options change the behavior of the main functions:

–a
Places *file* in the archive after the member specified by *posname*. If no member is named, *file* is added to the end of the archive.

–b
Places *file* in the archive before the member specified by *posname*. If no member is named, *file* is placed at the beginning of the archive.

–C
Prevents **ar** from overwriting existing files with extracted files. This option is used only with extraction (**–x**).

–c
Suppresses the message normally printed when **ar** creates a new archive file. You can use this only in conjunction with the **–r** and **–q** options.

–F *format*
Specifies the format of the archive library. On OpenExtensions, this is ignored.

–I
Ignores the case of letters when searching the archive for specified member names. Normally, the case is significant.

–i
Inserts *file* into the archive before the member specified by *posname*. If *posname* isn't specified, **ar** inserts *file* at the beginning of the archive. This option is the same as **–b**.

–l
This option is ignored. It requests that temporary files generated by **ar** be put in the directory rather than in the default temporary file directory. It is provided for backward compatibility with other versions of **ar**. See the FILES section for more details about temporary files.

–s
Regenerates the external symbol table regardless of whether the command modifies the archive.

–T
When used with **–x**, allows extraction of members with names longer than the file system supports. Normally this is an error, and **ar** does not extract the file. Most file systems truncate the file name to the appropriate length.

–u
Replaces the archive member only if the *member* file's modification time is more recent than the archive member time. **–u** implies **–r**, so it is not necessary to specify **–r** also (**–ru** and **–u** are exactly equivalent).

–v
Gives verbose output. With **–d**, **–q**, **–r**, and **–x**, this option prints the command letter and the member name affected before performing each operation. With **–t**, **ar** prints more information about archive members using a format similar to `ls -l`. With **–p**, **ar** writes the name of the member to **stdout**, before displaying the contents of the file.

Operands

archive
Specifies the path name of the archive file.

member
Specifies the path name of the file that is to be acted upon (placed, deleted, searched for, and so on) in the archive library.

Environment Variables

ar uses the following variable:

TMPDIR

The path name of the directory being used for temporary files. If it is not set, OpenExtensions uses **/tmp**.

Localization

ar uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**

See Appendix C, "Localization," on page 477 for more information.

Files

The **ar** command creates temporary files in the working directory and in the directory named by the **TMPDIR** environment variable. These files are intermediate versions of the archive file being created or updated. Consequently, they normally are the same size as the archive file being manipulated.

Usage Notes

ar may be used to store multiple versions of the same object file within one archive library. This is useful if you are providing an archive library which may be used to resolve references from code compiled with various compiler options. These options cause differences in the object files which must be matched with the archive library member attributes. Attributes for **ar** are: AMODE and XPLINK.

ar will store the attribute information for every entry in the symbol table. The linkage editor will use the attribute information to resolve external references with the appropriate archive library member. Because archive library member names are only the final component of the path name, these member names must be unique for the different object file versions.

Within the external symbol table, all symbols for a given member are kept together. Symbols of more recently added or modified members are located before symbols of older (not as recently modified) members in the archive. The modification time of an archive member determines its relative age.

Side files (normally those created with link-editing a DLL) can be made members of an archive file. When the linkage editor processes such an archive file, it will normally read in all such side files so that archives can be used for resolving symbol references in DLLs. For more information about resolving external references, see *z/OS MVS Program Management: User's Guide and Reference*.

You will want to establish a naming convention for the object files, and change your build procedures to generate the correct names. For example, if your archive contains 3 versions of myfuncs.o, you could generate names:

```
myfuncs.o    AMODE(31), non-XPLINK
myfuncsX.o   AMODE(31), XPLINK
myfuncs64.o  AMODE(64) (AMODE(64) always forces XPLINK)
```

Your make file might generate commands such as these:

```
c89 -c myfuncs.c
c89 -Wc,xplink -o myfuncsX.o -c myfuncs.c
c89 -Wc,LP64 -o myfuncs64.o -c myfuncs.c
ar  -ruv libmyfuncs.a myfuncs.o myfuncsX.o myfuncs64.o
```

To display the attributes of the symbols within a object file or an archive library of object files, use nm — Display symbol table of object, library, or executable files.

Exit Values

Possible exit status values are:

0
   Successful completion

1
   Failure due to any of the following:

   - Inability to create the extracted file
   - An error writing to the extracted file
   - The requested module not found on appending
   - An error opening the module on appending
   - An incorrect module on appending
   - Inability to access the module on appending
   - A module not found on table or extraction

2
   Incorrect command-line arguments or options

Portability

POSIX.2, X/Open Portability Guide, UNIX systems

For backward compatibility, you can omit the dash (**–**) preceding the options if the options appear only as the first argument after the command name.

The following options are XPG extensions to the POSIX standard: **–a**, **–b**, **–C**, **–i**, **–l**, **–m**, **–q**, **–s**, and **–T**.

The **–F** and the **–I** options are an extension to the POSIX and XPG standards.

Examples

1. To add a member **fioacc.o** to the archive file **/u/turner/bin/cliserpgm.a**, specify:

```
ar -rc /u/turner/bin/cliserpgm.a fioacc.o
```

2. To display the members of the archive file **/u/turner/bin/cliserpgm.a**, specify:

```
ar -tv /u/turner/bin/cliserpgm.a
```

3. To delete the member **repgen.o** from the archive file **/u/turner/bin/cliserpgm.a** and regenerate the external symbol table for the archive, specify:

```
ar -ds /u/turner/bin/cliserpgm.a repgen.o
```

Related Commands

c89, cxx, make, nm

awk — Process programs written in the awk language

---

awk [–F *ere*] [**–f** *prog*] [**–v** *var=value* ...] [*program*] [*var=value* ...] [*file* ...]

Purpose

awk is a file-processing language that is well suited to data manipulation and retrieval of information from text files. If you are unfamiliar with the language, you may find it helpful to read the **awk** information in *z/VM: OpenExtensions User's Guide* before reading the following material.

An **awk** program consists of any number of user-defined functions and *rules* of the form:

```
pattern {action}
```

There are two ways to specify the **awk** program:

- Directly on the command line. In this case, *program* is a single command-line argument, usually enclosed in single quotation marks (`'`) to prevent the shell from attempting to expand it.
- By using the **–f** *prog* option.

You can specify *program* directly on the command line only if you do not use any **–f** *prog* arguments.

Options

awk recognizes the following options:

–F *ere*
  Is an extended regular expression to use as the field separator.

–f *prog*
  Runs the **awk** program contained in the file *prog*. When more than one **–f** option appears on the command line, the resulting program is a concatenation of all programs you specify.

–v *var=value*
  Assigns *value* to *var* before running the program.

Files that you specify on the command line with the *file* argument provide the input data for **awk** to manipulate. If you specify no files or you specify a dash (**–**) as a file, **awk** reads data from standard input.

You can initialize variables on the command line using:

```
var=value
```

You can intersperse such initializations with the names of input files on the command line. **awk** processes initializations and input files in the order they appear on the command line. For example, the command:

```
awk -f progfile a=1 f1 f2 a=2 f3
```

sets *a* to 1 before reading input from *f1* and sets *a* to 2 before reading input from *f3*.

Variable initializations that appear before the first *file* on the command line are performed immediately after the **BEGIN** action. Initializations appearing after the last *file* are performed immediately before the **END** action. For more information on **BEGIN** and **END,** see "Patterns" on page 21.

The **–v** option lets you assign a value to a variable before the **awk** program begins execution (that is, before the **BEGIN** action). For example, in:

```
awk -v v1=10 -f prog datafile
```

awk assigns the variable *v1* its value before the **BEGIN** action of the program (but after default assignments made to such built-in variables as **FS** and **OFMT**; these built-in variables have special meaning to **awk**, as described later).

awk divides input into *records*. By default, newline characters separate records; however, you can specify a different record separator if you want. For more information, see the description of the **RS** variable ("Input" on page 17).

One at a time, and in order, **awk** compares each input record with the pattern of every rule in the program. When a pattern matches, **awk** performs the action part of the rule on that input record. Patterns and actions often refer to separate *fields* within a record. By default, white space (usually blanks, newlines, or horizontal tab characters) separates fields; however, you can specify a different field separator string using the **–F** *ere* option (see "Input" on page 17).

You can omit the *pattern* or *action* part of an **awk** rule (but not both). If you omit *pattern*, **awk** performs the *action* on every input record (that is, every record matches). If you omit *action*, **awk** writes every record matching the *pattern* to the standard output.

awk considers everything after a *#* in a program line to be a comment. For example:

```
# This is a comment
```

To continue program lines on the next line, add a backslash (\) to the end of the line. Statement lines ending with a comma (,), double or-bars (||), or double ampersands (&&) continue automatically on the next line.

Variables and Expressions

There are three types of *variables* in **awk**: *identifiers*, *fields*, and *array elements*.

An *identifier* is a sequence of letters, digits, and underscores beginning with a letter or an underscore. These characters must be from the POSIX portable character set. (Data can come from other character sets.)

For a description of *fields*, see "Input" on page 17.

Arrays are associative collections of values called the *elements* of the array. Constructs of the form:

```
identifier[subscript]
```

where *subscript* has the form *expr* or *expr,expr,...,* refer to array elements. Each such *expr* can have any string value. For multiple *expr* subscripts, **awk** concatenates the string values of all *expr* arguments with a separate character **SUBSEP** between each. The initial value of **SUBSEP** is set to \042 (code page 01047 field separator).

We sometimes refer to fields and identifiers as *scalar variables* to distinguish them from arrays.

You do not declare **awk** variables, and you do not need to initialize them. The value of an uninitialized variable is the empty string in a string context and the number 0 in a numeric context.

Expressions consist of constants, variables, functions, regular expressions, and *subscript-in-array* conditions (described Subscript in Array) combined with operators. Each variable and expression has a string value and a corresponding numeric value; **awk** uses the value appropriate to the context.

When converting a numeric value to its corresponding string value, **awk** performs the equivalent of a call to the **sprintf()** function (see "Built-In String Functions" on page 19) where the one and only *expr* argument is the numeric value and the *fmt* argument is either %d (if the numeric value is an integer) or the value of the variable **CONVFMT** (if the numeric value is not an integer). The default value of **CONVFMT** is %.6g. If you use a string in a numeric context, and **awk** cannot interpret the contents of the string as a number, it treats the value of the string as zero.

Numeric constants are sequences of decimal digits.

String constants are quoted, as in "a literal string". Literal strings can contain the following escape sequences:

Table 3. Escape Sequences in awk Literal Strings

Escape Sequence	Character
\a	Audible bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
ooo	Octal value *ooo*
\x*dd*	Hexadecimal value *dd*
\\/	Slash
\"	Quote
c	Any other character *c*

awk supports full regular expressions. (See Appendix B, "Regular Expressions (regexp)," on page 471 for more information.) When **awk** reads a program, it compiles characters enclosed in slash characters (/) as regular expressions. In addition, when literal strings and variables appear on the right side of a ~ or **"!~"** operator, or as certain arguments to built-in matching and substitution functions, **awk** interprets them as dynamic regular expressions.

Note: When you use literal strings as regular expressions, you need extra backslashes to escape regular expression metacharacters, because the backslash is also the literal string escape character.

For example the regular expression:

```
/e\.g\./
```

when written as a string is:

```
"e\\.g\\."
```

Subscript in Array: **awk** defines the *subscript-in-array* condition as:

```
index in array
```

where *index* looks like *expr* or *(expr,...,expr)*. This condition evaluates to 1 if the string value of *index* is a subscript of *array*, and to 0 otherwise. This is a way to determine if an array element exists. When the element does not exist, the subscript-in-array condition does not create it.

Symbol Table

You can access the symbol table through the built-in array **SYMTAB. SYMTAB[*expr*]** is equivalent to the variable named by the evaluation of *expr*. For example, **SYMTAB[*"var"*]** is a synonym for the variable *var*.

Environment

An **awk** program can determine its initial environment by examining the **ENVIRON** array. If the environment consists of entries of the form *name=value,* then **ENVIRON**[*name*] has string value "*value*". For example, the following program is equivalent to the default output of **env**:

```
BEGIN {
      for (i in ENVIRON)
             printf("%s=%s\n", i, ENVIRON[i])
```

```
        exit
    }
```

Operators

awk follows the usual precedence order of arithmetic operations, unless overridden with parentheses; a table giving the order of operations appears later in this section.

The unary operators are +, -, ++, and - -, where you can use the ++ and - - operators as either postfix or prefix operators, as in C. The binary arithmetic operators are +, -, *, /, %, and ^.

The conditional operator

```
expr ? expr1 : expr2
```

evaluates to the *expr1* if the value of *expr* is nonzero, and to *expr2* otherwise.

If two expressions are not separated by an operator, **awk** concatenates their string values.

The tilde operator (**~**) yields **1** (true) if the regular expression on the right side matches the string on the left side. The operator **!~** yields 1 when the right side has no match on the left. To illustrate:

```
$2  ~ /[0-9]/
```

selects any line where the second field contains at least one digit. **awk** interprets any string or variable on the right side of **~** or **!~** as a dynamic regular expression.

The relational operators are <, <=, >, >=, ==, and !=. When both operands in a comparison are numeric, **awk** compares their values numerically; otherwise, it compares them as strings. An operand is numeric if it is an integer or floating-point number, if it is a field or **ARGV** element that looks like a number, or if it is a variable created by a command-line assignment that looks like a number.

The Boolean operators are || (or), && (and), and ! (not). **awk** uses *short-circuit evaluation* when evaluating expressions. With an && expression, if the first operator is false, the entire expression is false and it is not necessary to evaluate the second operator. With an || expression, a similar situation exists if the first operator is true.

You can assign values to a variable with:

```
var = expr
```

If *op* is a binary arithmetic operator, *var op= expr* is equivalent to *var = var op expr*, except that *var* is evaluated only once.

See Table 4 on page 16 for the precedence rules of the operators.

Table 4. The Order of Operations for awk

Operators	Order of Operations
(A)	Grouping
$*i* V[a]	Field, array element
V++ V- - ++V - -V	Increment, decrement
A^B	Exponentiation
+A -A !A	Unary plus, unary minus, logical NOT
A*B A/B A%B	Multiplication, division, remainder
A+B A-B	Addition, subtraction
A B	String concatenation

Table 4. The Order of Operations for awk (continued)

Operators	Order of Operations
A<B A>B A<=B A>=B A!=B A=.=B	Comparisons
A ~B A! ~B	Regular expression matching
A in V	Array membership
A && B	Logical AND
A || B	Logical OR
A ? B : C	Conditional expression
V=B V+=B V-=B V*=B V/=B V%=B V^=B	Assignment

Note:

1. A, B, and C are any expression.

2. *i* is any expression yielding an integer.

3. V is any variable.

Command-Line Arguments

awk sets the built-in variable **ARGC** to the number of command-line arguments. The built-in array **ARGV** has elements subscripted with digits from zero to **ARGC**-1, giving command-line arguments in the order they appeared on the command line.

The **ARGC** count and the **ARGV** vector do not include command-line options (beginning with -) or the program file (following **–f**). They do include the name of the command itself, initialization statements of the form $var=value$, and the names of input data files.

awk actually creates **ARGC** and **ARGV** before doing anything else. It then "walks through" **ARGV,** processing the arguments. If an element of **ARGV** is an empty string, **awk** skips it. If it contains an equals sign (=), **awk** interprets it as a variable assignment. If it is a minus sign (-), **awk** immediately reads input from the standard input until it encounters the end of the file. Otherwise, **awk** treats the argument as a file name and reads input from that file until it reaches the end of the file.

Note: awk runs the program by "walking through" **ARGV** in this way; thus, if the program changes **ARGV,** **awk** can read different files and make different assignments.

Input

awk divides input into records. A *record separator character* separates each record from the next. The value of the built-in variable **RS** gives the current record separator character; by default, it begins as the newline (\n). If you assign a different character to **RS, awk** uses that as the record separator character from that point on.

awk divides records into fields. A *field separator string*, given by the value of the built-in variable **FS**, separates each field from the next. You can set a specific separator string by assigning a value to **FS**, or by specifying the **–F** *ere* option on the command line. You can assign a regular expression to **FS**. For example:

```
FS = "[,:$]"
```

says that commas, colons, or dollar signs can separate fields. As a special case, assigning **FS** a string containing only a blank character sets the field separator to white space. In this case, **awk** considers any sequence of contiguous space or tab characters a single field separator. This is the default for **FS**.

However, if you assign **FS** a string containing any other character, that character designates the start of a new field. For example, if we set FS=\t (the tab character),

```
texta \t textb \t  \t  \t textc
```

contains five fields, two of which contain only blanks. With the default setting, this record only contains three fields, since **awk** considers the sequence of multiple blanks and tabs a single separator.

The following list of built-in variables provides various pieces of information about input:

```
NF            Number of fields in the current record
NR            Number of records read so far
FILENAME      Name of file containing current record
FNR           Number of records read from current file
```

Field specifiers have the form $n, where *n* runs from 1 through **NF**. Such a field specifier refers to the *n*th field of the current input record. **$0** (zero) refers to the entire current input record.

The **getline** function can read a value for a variable or **$0** from the current input, from a file, or from a pipe. The result of **getline** is an integer indicating whether the read operation was successful. A value of 1 indicates success; 0 indicates that the end of the file was encountered; and -1 indicates that an error occurred. Possible forms for **getline** are:

getline
> Reads next input record into $0 and splits the record into fields. **NF, NR,** and **FNR** are set appropriately.

getline *var*
> Reads the next input record into the variable *var*. **awk** does not split the record into fields (which means that the current *$n* values do not change), but sets **NR** and **FNR** appropriately.

getline <*expr*
> Interprets the string value of *expr* to be a file name. **awk** reads the next record from that file into **$0**, splits it into fields, and sets **NF** appropriately. If the file is not open, **awk** opens it. The file remains open until you close it with a **close** function.

getline *var* **<*expr***
> Interprets the string value of *expr* to be a file name, and reads the next record from that file into the variable *var*, but does not split it into fields.

expr **| getline**
> Interprets the string value of *expr* as a command line to be run. **awk** pipes output from this command into **getline**, and reads it into *$0*, splits it into fields, and sets **NF** appropriately. See "System Function" on page 21 for additional details.

expr **| getline** *var*
> Runs the string value of *expr* as a command and pipes the output of the command into **getline**. The result is similar to **getline** *var <expr*.

You can have only a limited number of files and pipes open at one time. You can close files and pipes during execution using the **close**(*expr*) function. The *expr* argument must be one that came before | or after < in **getline**, or after > or >> in **print** or **printf**.

For a description of **print** and **printf**, see "Output" on page 22. If the function successfully closes the pipe, it returns zero. By closing files and pipes that you no longer need, you can use any number of files and pipes in the course of running an **awk** program.

Built-In Arithmetic Functions

atan2(*expr1*, *expr2*)
> Returns the arctangent of *expr1*/*expr2* in the range of -π through π.

exp(*expr*), log(*expr*), sqrt(*expr*)
> Returns the exponential, natural logarithm, and square root of the numeric value of *expr*. If you omit (*expr*), these functions use *$0* instead.

int(*expr*)
> Returns the integer part of the numeric value of *expr*. If you omit (*expr*), the function returns the integer part of *$0*.

rand()
> Returns a random floating-point number in the range 0 through 1.

sin(*expr*), cos(*expr*)
> Returns the sine and cosine of the numeric value of *expr* (interpreted as an angle in radians).

srand(*expr*)
> Sets the seed of the **rand** function to the integer value of *expr*. If you omit (*expr*), **awk** uses the time of day as a default seed.

Built-In String Functions

len = **length** *(expr)*
> Returns the number of characters in the string value of *expr*. If you omit (*expr*), the function uses *$0* instead. The parentheses around *expr* are optional.

n = **split***(string, array, regexp)*
> Splits the *string* into fields. *regexp* is a regular expression giving the field separator string for the purposes of this operation. This function assigns the separate fields, in order, to the elements of *array*; subscripts for `array` begin at 1. **awk** discards all other elements of *array*. **split** returns the number of fields into which it divided *string* (which is also the maximum subscript for *array*). *regexp* divides the record in the same way that the **FS** field separator string does. If you omit *regexp* in the call to **split**, it uses the current value of **FS**.

str = **substr***(string, offset, len)*
> Returns the substring of *string* that begins in position *offset* and is at most *len* characters long. The first character of the string has an *offset* of 1. If you omit *len*, **substr** returns the rest of *string*.

pos = **index***(string, str)*
> Returns the position of the first occurrence of *str* in *string*. The count is in characters. If **index** does not find *str* in *string*, it returns 0.

pos = **match***(string, regexp)*
> Searches *string* for the first substring matching the regular expression *regexp*, and returns an integer giving the position of this substring counting from 1. If it finds no such substring, **match** returns zero. This function also sets the built-in variable **RSTART** to *pos* and the built-in variable **RLENGTH** to the length of the matched string. If it does not find a match, **match** sets **RESTART** to 0, and **RLENGTH** to -1. You can enclose *regexp* in slashes or specify it as a string.

n = **sub***(regexp, repl, string)*
> Searches *string* for the first substring matching the regular expression *regexp*, and replaces the substring with the string *repl*. **awk** replaces any ampersand (&) in *repl* with the substring of *string* which matches *regexp*. You can suppress this special behavior by preceding the ampersand with a backslash. If you omit *string*, **sub** uses the current record instead. **sub** returns the number of substrings replaced (which is 1 if it found a match, and 0 otherwise).

n = **gsub***(regexp, repl, string)*
> Works the same way as **sub**, except that **gsub** replaces all matching substrings (global substitution). The return value is the number of substitutions performed.

str = **sprintf***(fmt, expr, expr...)*
> Formats the expression list *expr, expr, ...* using specifications from the string *fmt*, and then returns the formatted string. The *fmt* string consists of conversion specifications that convert and add the next *expr* to the string, and ordinary characters that **sprintf** simply adds to the string. These conversion specifications are similar to those used by the ANSI C standard.

Conversion specifications have the form

```
%[-][0][x][.y]c
```

where

-

Left-justifies the field; default is right justification.

0

(Leading zero) prints numbers with leading zero.

x

Is the minimum field width.

y

Is the precision.

c

Is the conversion character.

In a string, the precision is the maximum number of characters to be printed from the string; in a number, the precision is the number of digits to be printed to the right of the decimal point in a floating-point value. If *x* or *y* is ⋆ (asterisk), the minimum field width or precision is the value of the next *expr* in the call to **sprintf**.

The conversion character *c* is one of following:

d

Decimal integer

i

Decimal integer

o

Unsigned octal integer

x,X

Unsigned hexadecimal integer

u

Unsigned decimal integer

f,F

Floating point

e,E

Floating point (scientific notation)

g,G

The shorter of **e** and **f** (suppresses nonsignificant zeros)

c

Single character of an integer value; first character of string

s

String

The lowercase **x** specifies alphabetic hex digits in lowercase, whereas the uppercase **X** specifies alphabetic hex digits in uppercase. The other uppercase-lowercase pairs work similarly.

n = **ord***(expr)*

Returns the integer value of first character in the string value of *expr*. This is useful in conjunction with *%c* in **sprintf**.

str = **tolower***(expr)*

Converts all letters in the string value of *expr* into lowercase, and returns the result. If you omit *expr*, **tolower** uses *$0* instead. This function uses the value of the locale or the LC_CTYPE environment variable.

str = **toupper***(expr)*

Converts all letters in the string value of *expr* into uppercase, and returns the result. If you omit *expr*, **toupper** uses *$0* instead. This function uses the value of the locale or the LX_CTYPE environment variable.

System Function

***status* = system**(*expr*)

> Runs the string value of *expr* as a command. For example, **system**(*"tail " $1*) calls the **tail** command, using the string value of *$1* as the file that **tail** examines. The standard command interpreter runs the command as discussed in "Portability Section" on page 5 and the exit status returned depends on that command interpreter.

User-Defined Functions

You can define your own functions using the form:

```
function name(parameter-list) {
       statements
}
```

A function definition can appear in the place of a *pattern* {*action*} rule. The *parameter-list* argument contains any number of normal (scalar) and array variables separated by commas. When you call a function, **awk** passes scalar arguments by value, and array arguments by reference. The names specified in *parameter-list* are local to the function; all other names used in the function are global. You can define local variables by adding them to the end of the parameter list as long as no call to the function uses these extra parameters.

A function returns to its caller either when it runs the final statement in the function, or when it reaches an explicit **return** statement. The return value, if any, is specified in the **return** statement (see "Actions" on page 21).

Patterns

A *pattern* is a regular expression, a special pattern, a pattern range, or any arithmetic expression.

BEGIN is a special pattern used to label actions that **awk** performs before reading any input records. **END** is a special pattern used to label actions that **awk** performs after reading all input records.

You can give a pattern range as:

```
pattern1,pattern2
```

This matches all lines from one that matches *pattern1* to one that matches *pattern2*, inclusive.

If you omit a pattern, or if the numeric value of the pattern is nonzero (true), **awk** runs the resulting action for the line.

Actions

An *action* is a series of statements ended by semicolons, newlines, or closing braces. A *condition* is any expression; **awk** considers a nonzero value true, and a zero value false. A *statement* is one of the following or any series of statements enclosed in braces:

```
# expression statement, e.g. assignment
expression
```

```
# if statement
if (condition)
       statement
[else
       statement]
```

```
# while loop
while (condition)
       statement
```

```
# do-while loop
do
```

```
        statement
while (condition)
```

```
# for loop
for (expression1; condition; expression2)
        statement
```

The **for** statement is equivalent to:

```
expression1
while (condition) {
        statement
        expression2
}
```

The **for** statement can also have the form:

```
for (i in array)
        statement
```

awk runs the statement (specified with the *statement* argument) once for each element in *array*; on each repetition, the variable *i* contains the name of a subscript of *array*, running through all the subscripts in an *arbitrary* order. If *array* is multidimensional (has multiple subscripts), *i* is expressed as a single string with the **SUBSEP** character separating the subscripts.

The statement **break** exits a **for** or a **while** loop immediately. **continue** stops the current iteration of a **for** or **while** loop and begins the next iteration (if there is one). **next** ends any processing for the current input record and immediately starts processing the next input record. Processing for the next record begins with the first appropriate rule. **exit**[(*expr*)] immediately goes to the **END** action if it exists; if there is no **END** action, or if **awk** is already running the **END** action, the **awk** program ends. **awk** sets the exit status of the program to the numeric value of *expr*. If you omit (*expr*), the exit status is 0. **return** [*expr*] returns from the execution of a function.

If you specify an *expr*, the function returns the value of the expression as its result; otherwise, the function result is undefined. **delete** *array*[*i*] deletes element *i* from the given *array*. **print** *expr*, *expr*, ... is described in "Output" on page 22. **print***f fmt*, *expr*, *expr*, ... is also described in "Output" on page 22.

Output

The **print** statement prints its arguments with only simple formatting. If it has no arguments, it prints the entire current input record. **awk** adds the output record separator **ORS** to the end of the output that each **print** statement produces; when commas separate arguments in the **print** statement, the output field separator **OFS** separates the corresponding output values. **ORS** and **OFS** are built-in variables, whose values you can change by assigning them strings. The default output record separator is a newline, and the default output field separator is a space.

The variable **OFMT** gives the format of floating-point numbers output by **print**. By default, the value is %.6g; you can change this by assigning **OFMT** a different string value. **OFMT** applies only to floating-point numbers (ones with fractional parts).

The **printf** statement formats its arguments using the *fmt* argument. Formatting is the same as for the built-in function **sprintf**. Unlike **print**, **printf** does not add output separators automatically. This gives the program more precise control of the output.

The **print** and **printf** statements write to the standard output. You can redirect output to a file or pipe.

If you add *>expr* to a **print** or **printf** statement, **awk** treats the string value of *expr* as a file name, and writes output to that file. Similarly, if you add *>>expr*, **awk** sends output to the current contents of the file. The distinction between > and >> is important only for the first **print** to the file *expr*. Subsequent outputs to an already open file append to what is there already.

You cannot use such ambiguous statements as:

```
print a > b c
```

Use parentheses to resolve the ambiguity.

If you add /*expr* to a **print** or **printf** statement, **awk** treats the string value of *expr* as an executable command and runs it with the output from the statement piped as input into the command.

As mentioned earlier, you can have only a limited number of files and pipes open at any time. To avoid going over the limit, use the **close** function to close files and pipes when you no longer need them.

print and **printf** are also available as functions with the same calling sequence, but no redirection.

Examples

1. The following example:

```
awk '{print NR ":" $0}' input1
```

outputs the contents of the file **input1** with line numbers added before to each line.

2. The following is an example using *var=value* on the command line:

```
awk '{print NR SEP $0}' SEP=":" input1
```

awk can also read the program script from a file as in the command line:

```
awk -f addline.awk input1
```

which produces the same output when the file **addline.awk** contains:

```
{print NR ":" $0}
```

3. The following program appends all input lines starting with January to the file **jan** (which may or may not exist already), and all lines starting with February or March to the file **febmar**:

```
/^January/ {print >> "jan"}
/^February|^March/ {print >> "febmar"}
```

4. This program prints the total and average for the last column of each input line:

```
        {s += $NF}
END     {print "sum is", s, "average is", s/NR}
```

5. The next program interchanges the first and second fields of input lines:

```
{
        tmp = $1
        $1 = $2
        $2 = tmp
        print
}
```

6. The following inserts line numbers so that output lines are left-aligned:

```
{printf "%-6d: %s\n", NR, $0}
```

7. The following prints input records in reverse order (assuming sufficient memory):

```
{
        a[NR] = $0 # index using record number
}
END {
        for (i = NR; i>0; --i)
                print a[i]
}
```

8. The following program determines the number of lines starting with the same first field:

```
{
        ++a[$1] # array indexed using the first field
}
END {    # note output will be in undefined order
```

```
        for (i in a)
                print a[i], "lines start with", i
}
```

You can use the following program to determine the number of lines in each input file:

```
{
        ++a[FILENAME]
}
END {
        for (file in a)
                if (a[file] == 1)
                        print file, "has 1 line"
                else
                        print file, "has", a[file], "lines"
}
```

9. The following program illustrates how you can use a two-dimensional array in **awk**. Assume the first field of each input record contains a product number, the second field contains a month number, and the third field contains a quantity (bought, sold, or whatever). The program generates a table of products versus month.

```
BEGIN   {NUMPROD = 5}
{
        array[$1,$2] += $3
}
END     {
        print "\t Jan\t Feb\tMarch\tApril\t May\t" \
              "June\tJuly\t Aug\tSept\t Oct\t Nov\t Dec"
        for (prod = 1; prod <= NUMPROD; prod++) {
                printf "%-7s", "prod#" prod
                for (month = 1; month <= 12; month++){
                        printf "\t%5d", array[prod,month]
                }
                printf "\n"
        }
}
```

10. As the following program reads in each line of input, it reports whether the line matches a predetermined value:

```
function randint() {
        return (int((rand()+1)*10))
}
BEGIN   {
        prize[randint(),randint()] = "$100";
        prize[randint(),randint()] = "$10";
        prize[1,1] = "the booby prize"
}
{
        if (($1,$2) in prize)
                printf "You have won %s!\n", prize[$1,$2]
}
```

11. The following example prints lines, the first and last fields of which are the same, reversing the order of the fields:

```
$1==$NF {
        for (i = NF; i > 0; --i)
                printf "%s", $i (i>1 ? OFS : ORS)
}
```

12. The following program prints the input files from the command line. The **infiles** function first empties the passed array and then fills the array. The extra parameter *i* of **infiles** is a local variable.

```
function infiles(f,i) {
        for (i in f)
                delete f[i]
        for (i = 1; i < ARGC; i++)
                if (index(ARGV[i],"=") == 0)
                        f[i] = ARGV[i]
}
BEGIN   {
        infiles(a)
        for (i in a)
```

```
                print a[i]
        exit
}
```

13. Here is the standard recursive factorial function:

```
function fact(num) {
        if (num <= 1)
                return 1
        else
                return num * fact(num - 1)
}
{ print $0 " factorial is " fact($0) }
```

14. The following program illustrates the use of **getline** with a pipe. Here, **getline** sets the current record from the output of the **wc** command. The program prints the number of words in each input file.

```
function words(file,   string) {
        string = "wc " fn
        string | getline
        close(string)
        return ($2)
}
BEGIN   {
        for (i=1; i<ARGC; i++) {
                fn = ARGV[i]
                printf "There are %d words in %s.",
                    words(fn), fn
        }
}
```

Environment Variables

PATH
Contains a list of directories that **awk** searches when looking for commands run by **system**(*expr*), or input and output pipes.

Any other environment variable can be accessed by the **awk** program itself.

Localization

awk uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_NUMERIC**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
Successful completion

1
Any of the following errors:

- Parser internal stack overflow
- Syntax error
- Function redefined

- Internal execution tree error
- Insufficient memory for string storage
- Unbalanced parenthesis or brace
- Missing script file
- Missing field separator
- Missing variable assignment
- Unknown option
- Incorrect character in input
- Newline in regular expression
- Newline in string
- EOF in regular expression
- EOF in string
- Cannot open script file
- Inadmissible use of reserved keyword
- Attempt to redefine built-in function
- Cannot open input file
- Error on **print**
- Error on **printf**
- Getline in **END** action was not redirected
- Too many open I/O streams
- Error on I/O stream
- Insufficient arguments to **printf** or **sprintf()**
- Array cannot be used as a scalar
- Variable cannot be used as a function
- Too many fields
- Record too long
- Division (/ or %) by zero
- Syntax error
- Cannot assign to a function
- Value required in assignment
- Return outside of a function
- Can delete only array element or array
- Scalar cannot be used as array
- **SYMTAB** must have exactly one index
- Impossible function call
- Function call nesting level exceeded
- Wrong number of arguments to function
- Regular expression error
- Second parameter to "split" must be an array
- **sprintf** string longer than allowed number of characters
- No open file name
- Function requires an array
- Is not a function

- Failed to match
- Incorrect collation element
- Trailing \ in pattern
- Newline found before end of pattern
- More than 9 \(\) pairs
- Number in [0–9] incorrect
- [ ] imbalance or syntax error
- ( ) or \(\) imbalance
- { } or \{ \} imbalance
- Incorrect endpoint in range
- Out of memory
- Incorrect repetition
- Incorrect character class type
- Internal error
- Unknown *regex* error

When an **awk** program ends because of a call to **exit()**, the exit status is the value passed to **exit()**.

Limits

Most constructions in this implementation of **awk** are dynamic, limited only by memory restrictions of the system.

The maximum record size is guaranteed to be at least LINE_MAX as returned by **getconf**. The maximum field size is guaranteed to be LINE_MAX, also.

The parser stack depth is limited to 150 levels. Attempting to process extremely complicated programs may result in an overflow of this stack, causing an error.

Input must be text files.

Portability

POSIX.2 X/Open Portability Guide, UNIX systems

The **ord** function is an extension to traditional implementations of **awk**. The **toupper** and **tolower** functions and the **ENVIRON** array are in POSIX and the UNIX System V Release 4 version of **awk**. This version is a superset of New AWK, as described in *The AWK Programming Language* by Aho, Weinberger, and Kernighan.

The *standard command interpreter* that the *system* function uses and that **awk** uses to run pipelines for **getline**, **print**, and **printf** is system-dependent. On OpenExtensions, this interpreter is always **/bin/sh**.

Related Commands

ed, sed, regexp (see).

basename — Return the nondirectory components of a path name

---

basename *name* [*suffix*]

Purpose

The **basename** command strips off the leading part of a path name, leaving only the final component of the name, which is assumed to be the file name. To accomplish this, **basename** first checks to see if *name* consists of nothing but slash (/) characters. If so, **basename** replaces *name* with a single slash and the process is complete. If not, **basename** removes trailing slashes. If slashes still remain, **basename** strips off all leading characters up to and including the final slash. Finally, if you specify *suffix* and the remaining portion of *name* contains a suffix that matches *suffix*, **basename** removes that suffix.

Examples

The command:

```
basename src/dos/printf.c .c
```

produces:

```
printf
```

Localization

basename uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
    Successful completion

1
    Failure due to any of the following:

- Unknown command-line option
- Incorrect number of arguments

Portability

POSIX.2, X/Open Portability Guide, UNIX systems

Related Commands

dirname

bc — Use the arbitrary-precision arithmetic calculation language

bc [−i] [−l] [*file ...*]

Purpose

bc is a programming language that can perform arithmetic calculations to arbitrary precision. You can use it interactively, by entering instructions from the terminal. It can also run programs taken from files.

The *file* arguments you specify on the command line should be text files containing **bc** instructions. **bc** runs the instructions from those files, in the order that they appear on the command line, and then runs instructions from the standard input. **bc** ends when it runs a **quit** instruction or reaches the end of the file on standard input.

bc is a simple but complete programming language with a syntax reminiscent of the C programming language. This version of **bc** is a superset of the standard language available on most systems. It has a number of additional features intended to make the language more flexible and useful. Features unique to this implementation are noted.

Input consists of a series of instructions that assign values to variables or make calculations. It is also possible to define subprograms called *functions,* which perform a sequence of instructions to calculate a single value.

bc displays the result of any line that calculates a value, but does not assign it to a variable. For example, the instruction:

```
2+2
```

displays:

```
4
```

By default, **bc** displays the result of any evaluated instruction followed by a newline. **bc** also saves the last value displayed in a special variable `.` (dot), so that you can use it in subsequent calculations.

Options

bc recognizes the following options.

−i

Puts **bc** into interactive mode with a displayed prompt. In this mode, **bc** displays a prompt, which is ":"—waiting for input. In addition, it handles errors somewhat differently. Normally, when **bc** encounters an error while processing a file, the interpreter displays the error message and exits. In interactive mode, the interpreter displays the message and returns to the prompt mode to allow debugging.

−l

Loads a library of standard mathematical functions before processing any other input. This library also sets the *scale* to 20. For a description of the functions in the **−l** library, see "Built-In Functions" on page 39.

Numbers

Numbers consist of an optional minus (-) sign or an optional plus (+) sign followed by a sequence of zero or more digits, followed by an optional decimal point (.), followed by a sequence of zero or more digits. Valid digits are 0 through 9, and the hexadecimal digits A through F. The uppercase letters represent the

values from 10 through 15. There must be at least one digit, either before or after the decimal point. If not, **bc** interprets the decimal point as the special variable . (mentioned earlier).

A number can be arbitrarily long and can contain spaces. Here are some valid numbers with an input base of 10:

```
0    0.    .0    -3.14159    +09.    -12    1 000 000
```

Here are some valid numbers with an input base of 16 (*ibase*=16):

```
0    FF    FF.3    -10.444    A1
```

See "Bases" on page 31 for more information.

A final point is that you cannot break up numbers with commas; you can write 1000000 or 1 000 000, but 1,000,000 results in an error message.

Identifiers

Identifiers can include sequences containing any number of letters, digits, or the underscore (_) character but must start with a lowercase letter. Spaces are not allowed in identifiers.

In the POSIX locale, valid identifiers can include sequences containing any number of letters, digits, or the underscore (_) character but must start with a lowercase letter, as defined by the current locale.

For other locales, the character map for that locale determines which characters are valid in an identifier. If you want identifiers to be portable between locales, use characters from the POSIX character set. The use of identifiers longer than one character is an extension of this implementation. Identifiers are used as names for variables, functions, or arrays:

- A *variable* holds a single numeric value. You can declare variables that are local to a function using the **auto** statement (see "Functions" on page 37). All other variables are *global* and you can use them inside any function or outside all functions. You do not need to declare global variables. **bc** creates variables as it requires them, with an initial value of zero. (Remember that there is also the special variable . [dot], which contains the result of the last calculation.)

- A *function* is a sequence of instructions that calculates a single value. A list of zero or more values enclosed in parentheses always follow a function name, as in **my_func(3.14159)**. See "Functions" on page 37.

- An *array* is a list of values. Values in the list are called *elements* of the array. These elements are numbered, beginning at zero. We call such a number a *subscript,* or *index,* of the array. Subscripts always appear in square brackets after the array. For example, **a[0]** refers to element zero in the array **a**. The first element of the array always has the subscript 0. If a subscript value is a floating-point number, the fractional part is discarded to make the subscript into an integer. For example, the following expressions all refer to the same element:

```
a[3]    a[3.2]    a[3.999]
```

The maximum number of elements in a **bc** array is in the range from 0 to {*BC_DIM_MAX*}-1 inclusive. Unlike with many languages, you don't need to declare the size of an array. Elements are created dynamically as required, with an initial value of zero.

Since parentheses always follow function names and square brackets always follow array names, **bc** can distinguish between all three types of names—variable names, function names, and array names. Therefore, you can have variables, functions, and arrays with the same name. For example, *foo* may be a variable whereas **foo()** is a function and **foo[ ]** is an array.

Built-In Variables

bc has a number of built-in variables that are used to control various aspects of the interpreter. These are described in the following sections.

Scale

The *scale value* is the number of digits to be retained after the decimal point in arithmetic operations. For example, if the scale is 3, each calculation retains at least three digits after the decimal point. This means that:

```
5 / 3
```

has the value:

```
1.666
```

If −l is specified, the scale is set to 20; otherwise, the default scale is zero.

The variable *scale* holds the current scale value. To change scales, assign a new value to *scale*, as in:

```
scale = 5
```

Since *scale* is just a regular **bc** variable, it can be used in the full range of **bc** expressions.

The number of decimal places in the result of a calculation is affected not only by the scale, but also by the number of decimal places in the operands of the calculation. This is discussed in detail in "Arithmetic Operations" on page 32.

There is also a function **scale**, which can determine the scale of any expression. For example, **scale(1.1234)** returns the result 4, which is the scale of the number 1.1234. The result of the **scale** function is always an integer (that is, it has the scale of 0).

The maximum value for **scale** is given by the configuration variable {*BC_SCALE_MAX*} and the minimum value is 0.

Bases

bc lets you specify numbers in different bases—for example, octal (base 8) or hexadecimal (base 16). You can input numbers in one base and output them in a different base, simplifying the job of converting from one base to another. **bc** does this using the built-in variables *ibase* and *obase*.

ibase is the base for input numbers. It has an initial value of 10 (normal decimal numbers). To use a different base for inputting numbers, assign an integer to *ibase*, as in:

```
ibase = 8
```

This means that all future input numbers are to be in base 8 (octal). The largest valid input base is 16, and the smallest valid input base is 2. There is no mechanism provided to represent digits larger than 15, so bases larger than 16 are essentially useless. When the base is greater than 10, use the uppercase letters as digits. For example, base 16 uses the digits 0 through 9, and A through F. The digits are allowed in any number, regardless of the setting of *ibase* but are largely meaningless if the base is smaller than the digit. The one case where this is useful is in resetting the input base to 10. The constant A always has the value 10 no matter what *ibase* is set to, so to reset the input base to 10, type:

```
ibase = A
```

obase is the base in which numbers are output. It has an initial value of 10 (normal decimal numbers). To change output bases, assign an appropriate integer to *obase*.

If the output base is 16 or less, **bc** displays numbers with normal digits and hexadecimal digits (if needed). The output base can also be greater than 16, in which case each *digit* is printed as a decimal value and digits are separated by a single space. For example, if *obase* is 1000, the decimal number 123 456 789 is printed as:

```
123 456 789
```

Here, the digits are decimal values from 0 through 999. As a result, all output values are broken up into one or more *chunks* with three digits per chunk. Using output bases that are large powers of 10, you can

arrange your output in columns; for example, many users find that 100 000 makes a good output base, because numbers are grouped into chunks of five digits each.

Long numbers are output with a maximum of 70 characters per line. If a number is longer than this, **bc** puts a backslash (\) at the end of the line, indicating that the number is continued on the next line.

Internal calculations are performed in decimal, regardless of the input and output bases. Therefore the number of places after the decimal point are dictated by the scale when numbers are expressed in decimal form.

The maximum value for **obase** is given by the configuration variable {*BC_BASE_MAX*}.

Arithmetic Operations

bc provides a large number of arithmetic operations. Following standard arithmetic conventions, some operations are calculated before others. For example, multiplications take place before additions unless you use parentheses to group operations. Operations that take place first are said to have a higher *precedence* than operations that take place later.

Operations also have an *associativity*. The associativity dictates the order of evaluation when you have a sequence of operations with equal precedence. Some operations are evaluated left to right, whereas others are evaluated right to left. The following list shows the operators of **bc** from highest precedence to lowest.

bc Operator	Associativity
( )	Left to right
Unary ++ --	Not applicable
Unary - !	Not applicable
^	Right to left
* / %	Left to right
+ -	Left to right
= ^= *= /= %= +=	Right to left
== <= >= != < >	None
&&	Left to right
||	Left to right

Note: bc's order of precedence is not the same as C's. In C, the assignment operators have the lowest precedence.

The following list describes what each operation does. In the descriptions, A and B can be numbers, variables, array elements, or other expressions. V must be either a variable or an array element.

(A)
Indicates that this expression—A—should be evaluated before any other operations are performed on it.

-A
Is the negation of the expression.

!A
Is the logical complement of the expression. If A evaluates to zero, !A evaluates to 1. If A is not zero, !A evaluates to zero. This operator is unique to this version of **bc**.

++V
Adds 1 to the value of V. The result of the expression is the new value of V.

--V
Subtracts 1 from the value of V. The result of the expression is the new value of V.

V++
    Adds 1 to the value of V, but the result of the expression is the old value of V.

V--
    Subtracts 1 from the value of V, but the result of the expression is the old value of V.

A ^ B
    Calculates A to the power B. B must be an integer. The scale of the result of A^B is:

```
min(scale(A) * abs(B), max(scale, scale(A)))
```

    where **min** calculates the minimum of a set of numbers and **max** calculates the maximum.

A * B
    Calculates A multiplied by B. The scale of the result is:

```
min(scale(A) + scale(B), max(scale, scale(A), scale(B)))
```

A / B
    Calculates A divided by B. The scale of the result is the value of *scale*.

A % B
    Calculates the remainder from the division of A by B. This is calculated in two steps. First, **bc** calculates A/B to the current scale. It then obtains the remainder through the formula:

```
A - (A / B) * B
```

    calculated to the scale:

```
max(scale + scale(B), scale(A))
```

A + B
    Adds A plus B. The scale of the result is the maximum of the two scales of the operands.

A-B
    Calculates A minus B. The scale of the result is the maximum of the two scales of the operands.

The next group of operators are all *assignment* operators. They assign values to objects. An assignment operation has a value: the value that is being assigned. Therefore, you can write such operations as a=1+(b=2). In this operation, the value of the assignment in parentheses is 2 because that is the value assigned to b. Therefore, the value 3 is assigned to a. The possible assignment operators are:

V = B
    Assigns the value of B to V.

V ^= B
    Is equivalent to V=V^B.

V *= B
    Is equivalent to V=V*B.

V /= B
    Is equivalent to V=V/B.

V %= B
    Is equivalent to V=V%B.

V += B
    Is equivalent to V=V+B.

V -= B
    Is equivalent to V=V-B.

The following expressions are called *relations*, and their values can be either true (1) or false (0). This version of **bc** lets you use the relational operators in any expression, not just in the conditional parts of **if**, **while**, or **for** statements. These operators work exactly like their equivalents in the C language. The result of a relation is 0 if the relation is false and 1 if the relation is true.

A == B
   Is true if and only if A equals B.

A <= B
   Is true if and only if A is less than or equal to B.

A >= B
   Is true if and only if A is greater than or equal to B.

A != B
   Is true if and only if A is not equal to B.

A < B
   Is true if and only if A is less than B.

A > B
   Is true if and only if A is greater than B.

A && B
   Is true if and only if A is true (nonzero) and B is true. If A is not true, the expression B is never evaluated.

A || B
   Is true if A is true or B is true. If A is true, the expression B is never evaluated.

Comments and White Space

A *comment* has the form:

```
/* Any string */
```

Comments can extend over more than one line of text. When **bc** sees /* at the start of a comment, it discards everything up to */. The only effect a comment has is to indicate the end of a token. As an extension, this version of **bc** also provides an additional comment convention using the # character. All text from the # to the end of the line is treated as a single blank, as in:

```
2+2 # this is a comment
```

bc is free format. You can freely insert blanks or horizontal tab characters to improve the readability of the code. Instructions are assumed to end at the end of the line. If you have an instruction that is so long you need to continue it on a new line, put a backslash (\) as the very last character of the first line and continue on the second, as in:

```
a = 2\
  + 3
```

The \ indicates that the instruction continues on the next line, so this is equivalent to:

```
a = 2 + 3
```

Instructions

A **bc** instruction can be an expression that performs a calculation, an assignment, a function definition, or a statement. If an instruction is not an assignment, **bc** displays the result of the instruction when it has completed the calculation. For example, if you enter:

```
3.14 * 23
```

bc displays the result of the calculation. However, with:

```
a = 3.14 * 23
```

bc does not display anything, because the expression is an assignment. If you do want to display the value of an assignment expression, simply place the expression in parentheses.

The following list shows the instruction forms recognized by **bc**:

expression
Calculates the value of the *expression*.

"string"
Is a string constant. When **bc** sees a statement of this form, it displays the contents of the string. For example:

```
"Hello world!"
```

tells **bc** to display Hello world! A newline character is *not* output after the string. This makes it possible to do things like:

```
foo = 15
"The value of foo is "; foo
```

With these instructions, **bc** displays

```
The value of foo is 15
```

statement ; statement ...
Is a sequence of statements on the same line. In **bc**, a semicolon (;) and a newline are equivalent. They both indicate the end of a statement. **bc** runs these statements in order from left to right.

{statement}
Is a brace-bracketed statement. Braces are used to group sequences of statements together, as in:

```
{
   statement
   statement
      ...
}
```

Braces can group a series of statements that are split over several lines. Braces are usually used with control statements like **if** and **while**.

break
Can be used only inside a **while** or **for** loop. break ends the loop.

for (initexp ; relation ; endexp) statement
Is equivalent to:

```
initexp
while (relation) {
    statement
    endexp
}
```

where *initexp* and *endexp* are expressions and *relation* is a relation. For example:

```
a = 0
for (i = 1; i <= 10; ++i) a += i
```

is equivalent to the **while** example given earlier.

Note: All three items inside the parentheses must be specified. Unlike C, **bc** does not let you omit any of these expressions.

if (relation) statement
Tests whether the given *relation* is true. If so, **bc** runs the *statement*; otherwise, **bc** skips over the *statement* and goes to the next instruction. For example:

```
if ((a%2) == 0) "a is even"
```

displays a is even if a has an even value.

if (*relation*) *statement1* else *statement2*

Is similar to the simple **if** statement. It runs *statement1* if *relation* is true and otherwise runs *statement2*. It may be used as follows:

```
if ((a%2) == 0) "a is even" else "a is odd"
```

Note: There is no statement separator between "a is even" and the **else** keyword. This differs from the C language.

Here is another example:

```
if (a<10) {
        "a "
        "is "; "less than 10 "
        a
} else {
        "a is"
        " greater than 10 "
        a
}
```

Note: The braces must be on the same line as the **if** and the **else** keywords. This is because a new line or a semicolon right after (*relation*) indicates that the body of the statement is null. One common source of errors in **bc** programs is typing the statement body portion of an **if** statement on a separate line. If **–i** is used, the interpreter displays a warning when **if** statements with null bodies are encountered.

while (*relation*) *statement*

Repeatedly runs the given *statement* while *relation* is true. For example:

```
i = 1
a = 0
while (i <= 10) {
    a += i
    ++i
}
```

adds the integers from 1 through 10 and stores the result in a.

If *relation* is not true when **bc** encounters the **while** loop, **bc** does not run *statement* at all.

print *expression , expression ...*

Displays the results of the argument expressions. Normally, **bc** displays the value of each expression or string it encounters. This makes it difficult to format your output in programs. For this reason, the OpenExtensions shell version of **bc** has a **print** statement to give you more control over how things are displayed. **print** lets you display several numbers on the same line with strings. This statement displays all its arguments on a single line. A single space is displayed between adjacent numbers (but not between numbers and strings). A **print** statement with no arguments displays a newline. If the last argument is null, subsequent output continues on the same line. Here are some examples of how to use **print**:

```
/* basic print statement */
print "The square of ", 2, "is ", 2*2
The square of 2 is 4
```

```
/* inserts a space between adjacent numbers */
print 1,2,3
1 2 3
```

```
/* note - no spaces */
print 1,"",2,"",3
123
```

```
/* just print a blank line */
print
```

```
/* two statements with output on same line */
print 1,2,3, ; print 4, 5, 6
1 2 3 4 5 6
```

quit

Ends **bc**. In other implementations of **bc**, the interpreter exits as soon as it reads this token. This version of **bc** treats **quit** as a real statement, so you can use it in loops, functions, and so on.

sh ...

Lets you send a line to the system command interpreter for execution, as in:

```
sh ls -al
```

This command passes everything from the first nonblank character until the end of the line to the command interpreter for execution.

void *expression*

Throws away, or "voids," the result of the evaluation of *expression* instead of displaying it. This instruction is useful when using ++ and -- operators, or when you want to use a function but don't want to use the return value for anything. For example:

```
void foo++
```

increments foo but does not display the result. The **void** statement is unique to this version of **bc**.

Several other types of statements are relevant only in function definitions. These are described in the next section.

Functions

A function is a *subprogram* to calculate a result based on *argument* values. For example, the following function converts a temperature given in Fahrenheit into the equivalent temperature in Celsius:

```
define f_to_c(f) {
    return ((f-32) * 5 / 9)
}
```

This defines a function named **f_to_c()** that takes a single argument called f. The *body* of the function is enclosed in brace brackets. The opening brace must be on the same line as the **define** keyword. The function body consists of a sequence of statements to calculate the *result* of the function. An expression of the form:

```
return (expression)
```

returns the value of *expression* as the result of the function. The parentheses around the expression are optional.

To activate the subprogram you use a *function call*. This has the form:

```
name(expression,expression,...)
```

where **name** is the name of the function, and the *expression*s are argument values for the function. You can use function call anywhere you might use any other expression. The value of the function call is the value that the function returns. For example, with the function **f_to_c()**, described earlier, **f_to_c(41)** has the value 5 (since 41 Fahrenheit is equivalent to 5 Celsius).

The general form of a function definition is:

```
define name(parameter,parameter,...) {
    auto local, local, ...
    statement
    statement
        ...
}
```

Each *parameter* on the first line can be a variable name or an array name. Array names are indicated by putting square brackets after them. For example, if **cmpvec** is a function that compares two vectors, the function definition might start with:

```
define cmpvec(a[],b[]) {
```

Parameters do not conflict with arrays or variables of the same name. For example, you can have a parameter named a inside a function, and a variable named a outside, and the two are considered entirely separate entities. Assigning a value to the variable does not change the parameter and vice versa. All parameters are *passed by value*. This means that a copy is made of the argument value and is assigned to the formal parameter. This also applies to arrays. If you pass an array to a function, a copy is made of the whole array, so any changes made to the array parameter do not affect the original array.

A function may not need any arguments. In this case, the **define** line does not have any parameters inside the parentheses, as in:

```
define f() {
```

The **auto** statement declares a sequence of *local* variables. When a variable or array name appears in an **auto** statement, the current values of those items are saved and the items are initialized to zero. For the duration of the function, the items have their new values. When the function ends, the old values of the items are restored.

However, **bc** uses dynamic scoping rules, unlike C which uses lexical scoping rules. See "Usage Notes" on page 41 for more details.

For example:

```
define addarr(a[],l) {
    auto i, s
    for (i=0; i < l; ++i) s += a[i]
    return (s)
}
```

is a function that adds the elements in an array. The argument l stands for the number of elements in the array. The function uses two local names: a variable named *i* and a variable named *s*. These variables are "local" to the function **addarr** and are unrelated to objects of the same name outside the function (or in other functions). Objects that are named in an **auto** statement are called *autos*. Autos are initialized to 0 each time the function is called. Thus, the sum *s* is set to zero each time this function is called. You can also have local arrays, which are specified by placing square brackets after the array name in the **auto** statement.

```
define func_with_local_array() {
        auto local_array[];
        for(i=0; i<100; i++) local_array[i] = i*2
}
```

This example defines a local array called **local_array**. Local arrays start out with no elements in them.

If a function refers to an object that is not a parameter and not declared **auto**, the object is assumed to be *external*. External objects may be referred to by other functions or by statements that are outside of functions. For example:

```
define sum_c(a[ ],b[ ],l) {
    auto i
    for (i=0; i < l; ++i) c[i] = a[i] + b[i]
}
```

refers to an external array named **c**, which is the element-by-element sum of two other arrays. If **c** did not exist prior to calling **sum_c**, it is created dynamically. After the program has called **sum_c**, statements in the program or in functions can refer to array **c**.

Functions usually require a `return` statement. This has the form:

```
return (expression)
```

The argument *expression* is evaluated and used as the result of the function. The expression must have a single numeric value; it cannot be an array.

A **return** statement ends a function, even if there are more statements left in the function. For example:

```
define abs(i) {
    if (i < 0) return (-i)
    return (i)
}
```

is a function that returns the absolute value of its argument. If *i* is less than zero, the function takes the first **return**; otherwise, it takes the second.

A function can also end by running the last statement in the function. If so, the result of the function is zero. The function **sum_c** is an example of a function that does not have a **return** statement. The function does not need a **return** statement, because its work is to calculate the external array **c**, not to calculate a single value. Finally, if you want to return from a function, but not return a value you can use **return()** or simply **return**. If there are no parameters to the **return** statement, a default value of zero is returned.

Built-In Functions

bc has a number of built-in functions that perform various operations. These functions are similar to user-defined functions. You do not have to define them yourself, however; they are already set up for you. These functions are:

length(*expression*)
> Calculates the total number of decimal digits in *expression*. This includes digits both before and after the decimal point. The result of **length()** is an integer. For example, **length(123.456)** returns 6.

scale(*expression*)
> Returns the scale of *expression*. For example, **scale(123.456)** returns 3. The result of **scale()** is always an integer. Subtracting the scale of a number from the length of a number lets you determine the number of digits before the decimal point.

sqrt(*expression*)
> Calculates the square root of the value of *expression*. The result is truncated in the least significant decimal place (not rounded). The scale of the result is the scale of *expression*, or the value of **scale()**, whichever is larger.

You can use the following functions if **−l** is specified on the command line. If it is not, the function names are not recognized. There are two names for each function: a full name, and a single character name for compatibility with POSIX.2. The full names are the same as the equivalent functions in the standard C math library.

arctan(*expression*) or a(*expression*)
> Calculates the arctangent of *expression*, returning an angle in radians. This function can also be called as **atan**(*expression*).

bessel(*integer*,*expression*) or j(*integer*,*expression*)
> Calculates the Bessel function of *expression*, with order *integer*. This function can also be called as **jn**(*integer*,*expression*).

cos(*expression*) or c(*expression*)
> Calculates the cosine of *expression*, where *expression* is an angle in radians.

exp(*expression*) or e(*expression*)
> Calculates the exponential of *expression* (that is, the value **e** to the power of *expression*).

ln(*expression*) or l(*expression*)
> Calculates the natural logarithm of *expression*. This function can also be called as **log**(*expression*).

sin(*expression*) or s(*expression*)
> Calculates the sine of *expression*, where *expression* is an angle in radians.

Examples

1. Here is a simple function to calculate the sales tax on a purchase. The amount of the purchase is given by *purchase*, and the amount of the sales tax (in per cent) is given by *tax*.

```
define sales_tax(purchase,tax) {
    auto old_scale
    scale = 2
    tax = purchase*(tax/100)
    scale = old_scale
    return (tax)
}
```

For example:

```
sales_tax(23.99,6)
```

calculates 6% tax on a purchase of $23.99. The function temporarily sets the scale value to 2 so that the monetary figures have two figures after the decimal point. Remember that **bc** truncates calculations instead of rounding, so some accuracy may be lost. It is better to use one more digit than needed and perform the rounding at the end. The **round2** function, shown later in this section, rounds a number to two decimal places.

2. Division resets the scale of a number to the value of *scale*. You can use this to extract the integer portion of a number, as follows:

```
define integer_part(x) {
        # a local to save the value of scale
        auto old_scale
        # save the old scale, and set scale to 0
        old_scale = scale; scale=0
        # divide by 1 to truncate the number
        x /= 1
        # restore the old scale
        scale=old_scale
        return (x)
}
```

3. Here is a function you can define to return the fractional part of a number:

```
define fractional_part(x) {return (x - integer_part(x))}
```

4. The following function lets you set the scale of number to a given number of decimal places:

```
define set_scale(x, s)
    {   auto os
        os = scale
        scale = s
        x /= 1
        scale = os
        return (x)   }
```

You can now use **set_scale()** in a function that rounds a number to two decimal places:

```
define round2(num) {
        auto temp;
        if(scale(num) < 2) return (set_scale(num, 2))
        temp = (num - set_scale(num, 2)) * 1000
        if(temp > 5) num += 0.01
        return (set_scale(num,2))
}
```

This is a very useful function if you want to work with monetary values. For example, you can now rewrite **sales_tax()** to use **round2()**:

```
define sales_tax(purchase,tax) {
    auto old_scale
    scale = 2
    tax = round2(purchase*(tax/100))
    scale = old_scale
    return (tax)
}
```

5. Here is a function that recursively calculates the factorial of its argument:

```
define fact (x) {
        if(x < 1) return 1
        return (x*fact(x-1))
}
```

You can also write the factorial function iteratively:

```
define fact (x) {
        auto result
        result = 1
        while(x>1) result *= x--
        return (result)
}
```

With either version, **fact(6)** returns 720.

6. Here is another recursive function, that calculates the *n*th element of the Fibonacci sequence:

```
define fib(n) {
        if(n < 3) {
                return (1)
        } else {
                return (fib(n-1)+fib(n-2))
        }
}
```

Usage Notes

1. Unlike the C language, which uses lexical scoping rules, **bc** uses dynamic scoping, which is most easily explained with an example:

```
a=10
define f1() {
        auto a;
        a = 13;
        return (f2())
}
define f2() {
        return (a)
}
f1()
13
f2()
10
```

If **f1()** is called, **bc** prints the number 13, instead of the number 10. This is because **f1()** hides away the old (global) value of a and then sets it to 13. When **f2()** refers to a, it sees the variable dynamically created by **f1()** and so prints 13. When **f1()** returns, it restores the old value of a. When **f2()** is called directly, instead of through **f1()**, it sees the global value for a and prints 10. The corresponding C code prints 10 in both cases.

2. Numbers are stored as strings in the program and converted into numbers each time they are used. This is important because the value of a "constant" number may change depending on the setting of the *ibase* variable. For example, suppose the following instructions are given to **bc**:

```
define ten() {
        return (10)
}
ten()
10
ibase=16
ten()
16
```

In this example, when the base is set to 10, **ten()** returns the decimal value 10. However, when the input base is changed to 16, the function returns the decimal value 16. This can be a source of confusing errors in **bc** programs.

3. The library of functions loaded using the −l option is stored in the file /usr/lib/lib.b under your root directory.

Files

/usr/lib/lib.b
File containing the library of functions loaded with **−l**

Localization

bc uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See for more information.

Exit Values

Possible exit status values are:

0
Successful completion

1
Failure due to any of the following errors:

- Break statement found outside loop
- Parser stack overflow
- Syntax error
- End of file in comment
- End of file in string
- Numerical constant is too long
- String is too long
- Unknown option
- Empty evaluation stack
- Cannot pass scalar to array
- Cannot pass array to scalar
- Incorrect array index
- Built-in variable cannot be used as a parameter or auto variable
- *name* is not a function
- Incorrect value for built-in variable
- Shell command failed to run
- Division by 0
- Incorrect value for exponentiation operator
- Attempt to take square root of negative number
- Out of memory

Limits

The parser stack depth is limited to 150 levels. Attempting to process extremely complicated programs may result in an overflow of this stack, causing an error.

Portability

POSIX.2, UNIX systems

The following are extensions to the POSIX standard:

- The **–i** option
- The &&and || operators
- The **if … else …** statement
- identifiers of more than one character or containing characters outside the POSIX character set
- The **print** statement
- The **sh** statement
- The optional parentheses in the **return** statement

bg — Move a job to the background

bg [*job...*]

Purpose

bg runs one or more jobs in the background. The job IDs given on the command line identify these jobs, which should all be ones that are currently stopped. If you do not specify any job IDs, **bg** uses the most recently stopped job.

bg works only if job control is enabled; see the **–m** option of **set** for more information. Job control is enabled by default in the OpenExtensions Shell.

Localization

bg uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
> Successful completion.

>0
> Failure because a *job* argument is incorrect or there is no current job.

If an error occurs, **bg** exits and does not place the job in the background.

Portability

POSIX.2 User Portability Extension, UNIX systems.

Related Commands

fg, **jobs**, **set**

break — Exit from a for, select, while, or until loop in a shell script

```
break [number]
```

Purpose

break exits from a **for**, **select**, **while**, or **until** loop in a shell script. If *number* is given, **break** exits from the given number of enclosing loops. The default value of *number* is 1.

Usage Notes

This is a special built-in command of the shell.

Exit Values

break always exits with an exit status of 0.

Portability

POSIX.2, X/Open Portability Guide.

break is a special built-in shell command.

Related Commands

continue, **sh**

c89/cxx — Compile C/C++ source code and create an executable file

```
c89/cxx [-cEgOsV]
        [-D name[=value]]... [-U name]...
        [-W phase,option[,option]...]...
        [-o outfile]
        [-I directory]... [-L directory]...
        [file.c]... [file.a]...
        [file.o]...[file.x]...
        [-l libname]...
```

Purpose

The **c89** and **cxx** commands compile and build C/C++ programs. They are the OpenExtensions interface to the IBM C/C++ compilers:

- **c89** can invoke the IBM XL C/C++ for z/VM compiler, the IBM C/C++ for z/VM compiler, or the IBM C for VM/ESA compiler. See usage note "1" on page 50.
- **cxx** invokes the IBM XL C/C++ for z/VM compiler or the IBM C/C++ for z/VM compiler (whichever is installed).

When you issue **c89/cxx**, the utility passes information about the application program and the compiler options to the compiler for processing. First, **c89/cxx** performs the compilation phase (including preprocessing) by compiling all operands of the *file.c* form. The result of each compile step is a *file.o* file. If all compilations are successful, or if only *file.o* and no *file.c* files are specified, **c89/cxx** proceeds to the module build phase:

- **c89** by default invokes the Program Management binder. However, the C prelinker can be invoked by specifying the **-W b,p** option.
- **cxx** always invokes the Program Management binder.

In the module build phase, **c89/cxx** combines all *file.o* files from the compilation phase along with any *file.o* and *file.x* operands that were specified. Any *file.a* and **-l** *libname* operands that were specified are also used.

The output of the module build phase is an executable file. For **c89/cxx** to produce an executable file, you must specify at least one operand of the *file.c* or *file.o* form (or corresponding CMS native record file), or at least one operand of the *file.a* form.

c89/cxx can be invoked from the shell as a utility or from CMS as a CMS command.

For more information on how to manage your C source code, see *z/VM: OpenExtensions Advanced Application Programming Tools*.

Options

c89/cxx recognizes the following options.

-c
> Specifies that only compilations be done. If the source file is a BFS file, the object file is written to the working directory. If the source file is a CMS record file, the object file is written to the A-disk with the name *file* TEXT.

-D *name*[*=value*]
> Defines a C/C++ macro for use in compilation. If only *name* is provided, a value of 1 is used for the macro it specifies. For information about macros that **c89/cxx** automatically defines, see usage note "6" on page 50. Also, for related information, see usage note "10" on page 51.

-E

Specifies that C/C++ source produced by the compiler preprocessor phase be copied to **stdout**. Compilation into object and link-edit are not done. If **c89/cxx** is invoked from CMS and the original C/C++ source resides in the byte file system, then the generated C/C++ source is placed in the directory in which the original C/C++ source resides. If **c89/cxx** is invoked from CMS and the original C/C++ source resides on an accessed file mode, then the generated C/C++ source is placed on the user's A-disk.

-g

Specifies that compilation is to produce an object file that includes symbolic information, which is required for source-level debugging.

-I *directory*

Specifies the directories to be used during compilation in searching for *include files* (also called *header files*).

Absolute path names specified on **#include** directives are searched exactly as specified. The directories specified using the **-I** option or from the usual places are not searched.

If absolute path names are not specified on **#include** directives, then the search order is as follows:

- Include files enclosed in double quotation marks (") are first searched for in the directory of the file containing the **#include** directive. Include files enclosed in angle-brackets (< >) skip this initial search.
- The include files are then searched for in all directories specified by the **-I** option, in the order specified.
- Finally, the include files are searched for in the usual places. (See usage note "5" on page 50 for a description of the usual places.)

CMS files can explicitly be specified on **#include** directives. You can indicate this by specifying a leading double slash (//). For example, to include the include file DEF H that is on a CMS minidisk, code your C/C++ source as follows:

```
#include <//def.h>
```

CMS include files are handled according to C/C++ compiler conversion rules (see usage note "5" on page 50). When specifying an **#include** directive with a leading double slash, the file search follows the CMS access search order. This means that when you explicitly specify a CMS file, any directory names specified on the **-I** option are ignored.

-L *directory*

Specifies the directories to be used to search for archive libraries specified by the **-l** operand. The directories are searched in the order specified, followed by the usual places. You cannot specify a CMS file as an archive library directory.

For information on specifying C370LIB libraries, see the description of the **-l** *libname* operand. Also see usage note "7" on page 51 for a description of the usual places.

-O

Specifies that compilation be done with the C/C++ compiler level 1 optimization and selective inlining techniques. The defaults are no optimization and no inlining. If you compile and build your C/C++ program using the **-O** option, you cannot take advantage of source-level debugging.

In addition to using optimization techniques, you may want to control writable strings by using the **#pragma strings (readonly)** directive.

-o *outfile*

Specifies where **c89/cxx** is to write the executable file. The file **a.out** is the default when the source file is a BFS file, and is written to the working directory. If the source file is a CMS record file, the default is to write the executable file to the A-disk with the name *file* MODULE.

Also see usage note "4" on page 50 for related information.

-s

Specifies that compilation produces a *file.o* file that does *not* include symbolic information. This is the default behavior for **c89/cxx**.

-U *name*

Undefines a C/C++ macro specified with *name*. This option affects only macros defined by the **-D** option, including those automatically specified by **c89/cxx**. For information about macros that **c89/cxx** automatically defines, see usage note "6" on page 50. Also, for related information, see usage note "10" on page 51.

-V

This verbose option produces and directs output to **stdout** as compiler listings and (for **c89** only) prelinker listings. Error output continues to be directed to **stderr**. If **c89/cxx** is invoked from CMS, and if the source resides in a byte file system directory, then the output is placed in the directory where the source was found. If **c89/cxx** is invoked from CMS, and if the source resides on an accessed file mode, then the output is placed on the user's A-disk.

-W *phase,option*[,*option*]*...*

Specifies options to be passed to the compile or module build phases of **c89/cxx**. Phase **0** or **c** specifies the compile phase, and phase **b** specifies the module build phase.

- When using **c89** to invoke the IBM XL C/C++ for z/VM compiler or the IBM C/C++ for z/VM compiler, or when using **cxx**, the module build phase always uses the Program Management binder. To pass options to the binder, the first module build phase option must be **b**.

- When using **c89** to invoke the IBM C for VM/ESA compiler, the module build phase includes prelinker processing, the loading of the resulting CMS TEXT file using the CMS **LOAD** command, and the creation of the module file by the CMS **GENMOD** command.

  To pass options to the prelinker, the first module build phase option must be **p**. For example, to write the prelink map to **stdout,** specify:

  ```
  c89 -W b,p,map file.c
  ```

  To pass options to the **LOAD** command, the first module build phase option must be **l**.

  To pass options to the **GENMOD** command, the first module build phase option must be **g**.

  To use the Program Management binder (instead of the prelinker, **LOAD**, and **GENMOD**) and pass options to it, the first module build phase option must be **b**. For example:

  ```
  c89 -W b,b,NOTERM file.c
  ```

You *cannot* use **-W** to override the compiler options that correspond to **c89/cxx** options, with the exception of the listing options (corresponding to **-V**) and inlining options (corresponding to **-O**).

For the prelinker, **c89** uses the following options, all of which can be overridden using the **-W** option with the exception of OE.

```
DUP                   NONCAL
OE                    NOMAP
NER                   NOUPCASE
NOMEMORY              NOLIB
```

For the CMS LOAD command, **c89** uses the default options except for RLDSAVE, NOAUTO, and NOMAP.

For the CMS GENMOD command, **c89** uses the default options except for NOMAP.

For the Program Management binder (CMS BIND command), **c89/cxx** uses the default options except for CASE MIXED.

Notes:

1. Most compiler and prelinker options have a positive and negative form. The negative form is the positive with a NO added before (as in XREF and NOXREF). The same is true for **LOAD** and **GENMOD**.

2. The IBM XL C/C++ for z/VM compiler is described in *XL C/C++ for z/VM: User's Guide*. The IBM C/C++ for z/VM compiler is described in *C/C++ for z/VM: User's Guide*, SC09-7625-00. The IBM C for VM/ESA compiler is described in *C for VM/ESA: User's Guide*, SC09-2152-00.

3. The Program Management binder is described in *z/VM: Program Management Binder for CMS*.

4. The prelinker is described in *z/VM: Language Environment® User's Guide*.

5. The CMS module build process is described in *z/VM: CMS Application Development Guide*.

Operands

c89/cxx recognizes the following operands:

Note: You can specify a CMS record file system file identifier by preceding the file name with a double slash (**//**).

file.c
> Specifies the name of a C/C++ source file to be compiled. The form for a C source file is *file*.c; the form for a C++ source file is *file*.cpp or *file*.cxx. You can specify a CMS file, but it must have a file type of C, CPP, or CXX.
>
> The object file is written in the working directory and is named *file.o*. If a CMS native record file name is specified, the object file is named *file* MODULE A. See usage note "4" on page 50 for related information.

file.a
> Specifies the name of an archive file, as produced by the **ar** command, to be used during the module build phase.

file.o
> Specifies the name of a C/C++ object file, produced by **c89/cxx**, to be used in the module build phase. You can specify a CMS file, but it must have a file type of TEXT.

file.x
> Specifies the name of a definition side-deck produced during the **c89** link-editing phase when creating a Dynamic Link Library (DLL). You can specify a CMS file, but it must have a file type of EXP. For additional information, see usage note "12" on page 51.

-l *libname*
> Specifies the name of an archive library. **c89/cxx** searches for the file **lib***libname***.a** in the directories specified on the **-L** option and then in the usual places. The first occurrence of the archive library is used. For a description of the usual places, see usage note "7" on page 51.
>
> You can specify a CMS file, but it must have a file type of TXTLIB. *libname* is used directly without prefixing it with **lib**. If only **//***libname* is specified, the file type txtlib is assumed. The CMS native record file specified must be a C370LIB object library. For more information about the Object Library Utility, see *XL C/C++ for z/VM: User's Guide*.

Files

libc.a
> C/C++ function library (see usage note "7" on page 51).

libm.a
> C/C++ math function library (see usage note "7" on page 51).

libl.a
> **lex** function library.

liby.a
> **yacc** function library.

/usr/include
> The usual place to search for include files (see usage note "5" on page 50).

/lib
> The usual place to search for library functions (see usage note "7" on page 51).

/usr/lib
> The usual place to search for library functions (see usage note "7" on page 51).

Usage Notes

1. By default, the **c89** command invokes the IBM XL C/C++ for z/VM compiler or the IBM C/C++ for z/VM compiler (whichever is installed) to compile C or C++ source. If you had previously set **c89** to invoke the IBM C for VM/ESA compiler and want to change to the IBM XL C/C++ for z/VM compiler or the IBM C/C++ for z/VM compiler, you can issue the following command to specify the C/C++ compiler module (CBXFINIT) on the _CNAME environment variable:

   ```
   globalv select cenv setlp_cname cbxfinit
   ```

   To use the IBM C for VM/ESA compiler, you can specify the C compiler module (CBC310) by issuing the following command:

   ```
   globalv select cenv setlp_cname cbc310
   ```

   The **cxx** command always invokes CBXFINIT MODULE and does not look at the _CNAME environment variable.

2. To be able to specify an operand that begins with a dash (-), before specifying any other operands that do not, you must use the double dash (- -) end-of-options delimiter. This also applies to the specification of the **-l** operand.

3. When invoking **c89/cxx** from the shell, any option-arguments or operands specified that contain characters with special meaning to the shell must be escaped. For example, some **-W** option-arguments contain parentheses.

   To escape these special characters, either enclose the option-argument or operand in double quotation marks, or precede each character with a backslash.

4. Some **c89/cxx** behavior applies only to files (and not to CMS native record files).

   - The **-o** option does not allow a file of the form *file.c* to be specified.
   - If the compilation is not successful, the corresponding *file.o* file is always removed.

5. Minidisks and SFS directories in the CMS file system search order are used as the usual place to resolve compiler include files during compilation. Searching here for include files is automatic.

   Because the include files are CMS files, the C/C++ compiler uses conversion rules to transform the **#include** preprocessor directive specification into a CMS file name. This transformation strips any directory name on the **#include** directive, and then takes the first 8 or less characters up to the first dot (.).

   Therefore, if an application programmer specifies an **#include** directive with a relative path name having the same file name as a system include file (CMS file) and the user include file cannot be found, the system include file is found instead.

   For consistency with other implementations, **c89/cxx** searches the directory **/usr/include** as the usual place, just prior to searching the CMS minidisks or SFS directories.

6. **c89/cxx** automatically defines the following POSIX feature test macros:

   ```
   errno=(*__errno())
   _OPEN_DEFAULT=1
   __OPEN_VM=1
   ```

   c89/cxx adds the macro definition only after processing the **c89/cxx** command string. You can override the macro by specifying **-D** or **-U** options for it on the **c89/cxx** command string.

   The __OPEN_VM macro is used internally in the compiler and does not change any of the standard feature macros. The _OPEN_DEFAULT macro defines the level of POSIX feature test macros used in FEATURES H.

7. The usual place for the **-L** option search is the **/lib** directory followed by the **/usr/lib** directory. For consistency with other implementations, the archive libraries **libc.a** and **libm.a** exist as files in the **/usr/lib** directory. However, the library functions are not contained in them. Instead, CMS files installed with Language Environment® are used as the usual place to resolve library functions in the final step of the link-editing phase.

8. Because archive library files are searched when their names are encountered, the placement of **-l** operands and *file.a* operands is significant. You may have to specify a library multiple times on the **c89/cxx** command string, if subsequent specification of *file.o* files requires that additional symbols be resolved from that library.

9. Normally, options and operands are processed in the order read (from left to right). Where there are conflicts, the last specification is used (such as with **-g** and **-s**). However, some **c89/cxx** options will override others, regardless of the order in which they are specified. The option priorities, in order of highest to lowest, are as follows:

   -E
   Overrides **-O** and **-V**, **-c**, **-g** and **-s**.

   -g
   Overrides **-O** and and **-s**.

   -s
   Overrides **-g** (the last one specified is honored).

   -O -V, -c
   All are honored if not overridden.

10. For options that have option-arguments, the meaning of multiple specifications of the options is as follows:

    -D
    All specifications are used. If the same name is specified on more than one **-D** option, only the first definition is used.

    -U
    All specifications are used. The name is *not* defined, regardless of the position of this option relative to any **-D** option specifying the same name.

    -I
    All specifications are used. If the same directory is specified on more than one **-I** option, the directory is searched only the first time.

    -L
    All specifications are used. If the same directory is specified on more than one **-L** option, the directory is searched only the first time.

    -W
    All specifications are used. All options specified for a phase are passed to it, as if they were concatenated together in the order specified.

    -o
    The output file used will be the one specified on the last **-o** option.

11. The C/C++ runtime library supports a file naming convention of // (the file name can begin with exactly two slashes). **c89/cxx** indicates that the file naming convention of // can be used.

    However, OpenExtensions *does not* support this convention. Do not use this convention (//) unless it is specifically indicated (as here in **c89/cxx**). OpenExtensions does support the POSIX file naming convention where the file name can be selected from the set of character values excluding the slash and the null character.

12. A *file.x* definition side-deck contains link-editing phase IMPORT control statements naming symbols that are exported by a DLL. The definition side-deck is subsequently used during the link-editing phase of an application that is to use the DLL.

To create a definition side-deck, you must specify the **dll** option as a linkage editor option during the **c89** link-editing phase when creating the DLL. Also, you must use either the C compiler option **exportall** or the C compiler directive **#pragma export**. For example:

```
c89 -o outdll -W c,expo,dll -W b,p,dll file.c
```

The definition side-deck is written to the working directory and is named [*var*][outdll.x/*var*]. If a file identifier of **//outdll** is specified, the definition side-deck is named //outdll.EXP. If the output file specified already has a suffix, that suffix is replaced.

To subsequently use *file.x* definition side-decks, specify them along with any other *file.o* object files specified for the **c89** link-editing phase. For example:

```
c89  -omyappl myappl.o outdll.x
```

To run an application that is link-edited with a definition side-deck, the DLL that was created along with the definition side-deck must be made available. When the DLL resides in the BFS, it must be in either the working directory or a directory named on the LIBPATH environment variable. Otherwise, it must be a file residing on a minidisk or SFS directory accessed in the current CMS search order.

For more information about DLLs, see the *z/VM: CMS Application Development Guide*.

Localization

c89/cxx uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

0

Successful completion.

1

Failure due to incorrect specification of the arguments passed to **c89/cxx**.

2

Failure processing archive libraries:

- Archive library was not in any of the library directories specified.
- Archive library was incorrectly specified, or was not specified, following the **-l** operand.

3

Compilation, prelink, or build step was unsuccessful.

4

Error when preparing to call the compiler, prelinker, or module build commands for one of the following reasons:

- The file or CMS native record file name specified is incorrect.
- The file or CMS native record file name cannot be opened.

5

Dynamic allocation error, when preparing to call the compiler, prelinker, or module build commands due to an error being detected in the allocation information.

6

Error copying the file from a temporary CMS file to a BFS file.

8
   Error creating a temporary input CMS file for the compiler, prelinker, or module build commands.

Portability

POSIX.2.

The **-V** option is an extension of the POSIX standard.

Related Commands

ar, **lex**, **make**, **strip**, **yacc**

cat — Concatenate and display a text file

---

cat [**–su**] [**–v** [**et**]] [*file …*]

Purpose

cat displays and concatenates files. It copies each *file* argument to the standard output (**stdout**). If you specify no files or specify a dash (**–**) as a file name, **cat** reads the standard input (**stdin**).

Options

cat recognizes the following options:

–e
> Displays a $ character at the end of each line. This option works only if you also specify **–v**.

–s
> Does not produce an error message if **cat** cannot find or read a specified file.

–t
> Displays tabs as ^I. This option works only if you also specify **–v**.

–u
> Does not buffer output.

–v
> Displays all characters including those that are unprintable characters. If the character is unprintable, one of the following three representations is used:
>
> - M–X is used for character X if the significant bit is set.
> - ^X is used for the control character X (for example, ^A for CTRL-A).
> - \xxx represents a character with the octal value xxx.
>
> The \xxx form is used if neither of the other representations can be used.

Localization

cat uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
> Successful completion.

1
> Failure due to any of the following:
>
> - An incorrect command-line argument.
> - Inability to open the input file.

- End of the file detected on the standard output.
- The input file is the same as the output file.

2

    An incorrect command-line argument.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–e**, **–s**, **–t**, and **–v** options are extensions of the POSIX standard.

Related Commands

cp, **mv**

cd — Change the working directory

```
cd [directory]
cd old new
cd −
```

Purpose

The command **cd** `directory` changes the working directory of the current shell execution environment (see **sh**) to *directory*. If you specify *directory* as an absolute path name, beginning with /, this is the target directory. **cd** assumes the target directory to be the name just as you specified it. If you specify *directory* as a relative path name, **cd** assumes it to be relative to the current working directory.

Two special symbols are also supported:

. (dot)
    Represents the current directory

.. (dot dot)
    Represents the parent of the current directory.

If the variable **CDPATH** is defined in the shell, the built-in **cd** command searches for a relative path name in each of the directories defined in **CDPATH**. If **cd** finds the directory outside the working directory, it displays the new working directory.

Use colons to separate directories in **CDPATH**. In **CDPATH**, a null string represents the working directory. For example, if the value of **CDPATH** begins with a separator character, **cd** searches the working directory first; if it ends with a separator character, **cd** searches the working directory last.

In the shell, the command **cd -** is a special case that changes the current working directory to the previous working directory by exchanging the values of the variables **PWD** and **OLDPWD**.

Note: Repeating this command toggles the current working directory between the current and the previous working directory.

Calling **cd** without arguments sets the working directory to the value of the **HOME** environment variable, if the variable exists. If there is no **HOME** variable, **cd** does not change the working directory.

The form **cd** *old new* is an extension to traditional implementations of **sh**. The shell keeps the name of the working directory in the variable **PWD**. The **cd** command scans the current value of **PWD** and replaces the first occurrence of the string *old* with the string *new*. The shell displays the resulting value of **PWD**, and it becomes the new working directory.

If either directory is a symbolic link to another directory, the behavior depends on the setting of the shell's **−o** logical option. See "set — Set or unset command options and positional parameters" on page 273 for more information.

Environment Variables

CDPATH
    Contains a list of directories for **cd** to search in when *directory* is a relative path name.

HOME
    Contains the name of your home directory. This is used when you do not specify *directory* on the command line.

OLDPWD
    Contains the path name of the previous working directory. This is used by **cd −**.

PWD

> Contains the path name of the current working directory. This is set by **cd** after changing to that directory.

Localization

cd uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

> Successful completion

1

> Failure due to any of the following:
>
> - No **HOME** directory
> - No previous directory
> - A search for *directory* failed
> - An *old*-to-*new* substitution failed

2

> An incorrect command-line option

Messages and Return Codes

Possible error messages include:

***dir* bad directory**

> **cd** could not locate the target directory. This does not change the working directory.

Restricted

> You are using the restricted version of the shell (for example, by specifying the **–r** option for **sh**). The restricted shell does not allow the **cd** command.

No HOME directory

> You have not assigned a value to the **HOME** environment variable. Thus, when you run **cd** in order to return to your home directory, **cd** cannot determine what your home directory is.

No previous directory

> You tried the command **cd –** to return to your previous directory; but there is no record of your previous directory.

Pattern *old* not found in *dir*

> You tried a command of the form **cd** *old new*. However, the name of the working directory *dir* does not contain any string matching the regular expression *old*.

Portability

POSIX.2, X/Open Portability Guide.

All UNIX systems feature the first form of the command.

In the OpenExtensions shell implementation of this command, all forms are built into the shell.

The **cd** *old new* form of the command is an extension of the POSIX standard.

Related Commands

set, **sh**

chgrp — Change the group owner of a file or directory

---

chgrp [**–fR**] *group pathname ...*

Purpose

chgrp sets the group ID to *group* for the files and directories named by the *pathname* arguments. *group* can be a group name, from a group database, or it can be a numeric group ID (GID).

Note: chgrp can be used only by the file owner or a superuser. The file owner must have the new group as his or her group or one of the supplementary groups.

Options

chgrp accepts two options:

–f

> Does not issue an error message if **chgrp** cannot change the group ID. In this case, **chgrp** always returns a status of 0.

–R

> If a *pathname* on the command line is the name of a directory, **chgrp** changes the group ID of all files and subdirectories in that directory. If **chgrp** cannot change some file or subdirectory in the directory, it continues to try to change the other files and subdirectories in the directory, but exits with a nonzero status.

Localization

chgrp uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

> You specified **–f**, or **chgrp** successfully changed the group ownership of all the specified files and directories.

1

> Failure due to any of the following:
>
> - Inability to access a specified file.
> - Inability to change the group of a specified file.
> - An irrecoverable error was encountered when you specified the **–R** option.

2

> Failure due to any of the following:
>
> - The command line contained an unknown option or too few arguments.
> - **chgrp** did not recognize the specified *group*.

Portability

POSIX.2, UNIX systems.

The **–f** option is an extension of the POSIX standard.

Related Commands

chmod, **chown**

chmod — Change the mode of a file or directory

> **chmod** [**–fR**] *mode pathname ...*

Purpose

chmod changes the access permissions, or *modes,* of the specified file or directory. Modes determine who can read, write, or search a directory.

Note: chmod can be used only by the file owner or a superuser.

Options

chmod accepts two options:

–f

> Does not issue error messages concerning file access permissions, even if **chmod** encounters such errors.

–R

> If you specify a directory as a path name on the command, **chmod** changes the access permissions of all files and subdirectories under that directory.

You can specify the *mode* value on the command line in either symbolic form or as an octal value.

The symbolic form of the *mode* argument has the form:

```
[who] op permission [op permission ...]
```

The *who* value is any combination of the following:

u

> Sets all owner (user or individual) permissions.

g

> Sets all group permissions.

o

> Sets all other permissions.

a

> Sets all permissions (owner, group, and other); this is the default. If a *who* value is not specified, the default is **a** and the file creation mask is applied.

The *op* part of a symbolic mode is an operator that tells **chmod** to turn the permissions on or off. The possible values are:

+

> Turns on a permission.

-

> Turns off a permission.

=

> Turns on the specified permissions and turns off all others (owner, group, or other) for the specified *who*.

The *permission* part of a symbolic mode is any combination of the following:

r

> Read permission. If this is off, you cannot read the file.

x

Execute permission for a file. If this is off, you cannot run the file. Search permission for a directory. If this is off, you cannot search the directory.

X

Search permission for a directory; or execute permission for a file only when the current mode has at least one of the execute bits set.

Notes:

1. When using the **–R** option, you can turn on search permission for all directories without changing the execute permission for all regular files.

2. Using **X** on **chmod** is not displayed as **X** on **ls**. A file cannot choose between **x** and **X** as the execute permission. **X** is determined at the time of the **chmod**.

w

Write permission. If this is off, you cannot write to the file.

s

If in owner permissions section, the *set-user-ID* bit is on; if in group permissions section, the *set-group-ID* bit is on.

t

This represents the *sticky bit*. The *sticky bit* can be set, but OpenExtensions will take no action based on its setting.

You can specify multiple symbolic names if you separate them with commas. For example, you can specify the same *who* when you have multiple groups, which are processed left to right:

```
chmod a=,u=rwx
```

Absolute modes are octal numbers specifying the complete list of attributes for the files; you specify attributes by ORing together these bits.

```
4000   Set-user-ID bit
2000   Set-group-ID bit
1000   Sticky bit
0400   Owner read
0200   Owner write
0100   Owner execute/search (or list directory)
0040   Group read
0020   Group write
0010   Group execute/search
0004   Other read
0002   Other write
0001   Other execute/search
```

Examples

```
chmod -w orgcht
```

removes write permission from **orgcht**.

```
chmod a=rwx aprsal
```

turns on read, write, and execute permissions, and turns off the set-user-ID bit, set-group-ID bit, and sticky-bit attributes. This is equivalent to chmod 0777 aprsal.

Localization

chmod uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**

- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Usage Notes

For a mounted external link, the actual access permission set is the combination of the permission set for the linked object and the permission set for the link itself. If you are changing the access permissions for an external link, you may have to change both of these permission sets. Specifying the **chmod** command with the name of the external link changes only the permissions for the link. To change permissions for the linked object, specify the name of the external link with a closing slash (/) or specify its fully qualified pathname.

Exit Values

Possible exit status values are:

0

> Successful completion

1

> Failure due to any of the following:
>
> - Inability to access a specified file
> - Inability to change the modes on a specified file
> - Inability to read the directory containing the item to change
> - An irrecoverable error was encountered when using the **–R** option

2

> Failure due to any of the following:
>
> - Missing or incorrect *mode* argument
> - Too few arguments

Messages and Return Codes

Possible error messages include:

irrecoverable error during -R option
> The **–R** option was specified, but some file or directory in the directory structure was inaccessible. This may happen because of permissions.

read directory *name*
> Read permissions are not on the specified directory.

Portability

POSIX.2, X/Open Portability Guide.

The **–f** option and the **t** permission are extensions of the POSIX standard.

Related Commands

ls, **umask**

chown — Change the owner or group of a file or directory

chown [**–fR**] *owner*[**:** *group*] *pathname ...*

Purpose

chown sets the user ID (UID) to *owner* for the files and directories named by *pathname* arguments. *owner* can be a user name from the user profile, or it can be a numeric user ID.

If you include a *group* name—that is, if you specify *owner* followed immediately by a colon (:) and then *group* with no intervening spaces, such as *owner:group*—**chown** also sets the group ID to *group* for the files and directories named.

Note: chown can be used only by a superuser.

Options

chown accepts the following options:

–f

> Does not issue an error message if **chown** cannot change the owner. In this case, **chown** always returns a status of zero. Other errors may cause a nonzero return status.

–R

> If *pathname* on the command line is the name of a directory, **chown** changes all the files and subdirectories in that directory to belong to the specified *owner* (and *group*, if *:group* is specified). If **chown** cannot change some file or subdirectory in the directory, it continues to try to change the other files and subdirectories in the directory, but exits with a nonzero status.

Localization

chown uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Usage Notes

For a mounted external link, both the linked object and the link itself have an owner and group ID. However, true ownership of the external link rests with the linked object. Specifying the **chown** command with the name of the external link changes only the owner or group ID of the link. To change the owner or group ID of the linked object, specify the name of the external link with a closing slash (/) or specify its fully qualified pathname.

Exit Values

Possible exit status values are:

0

> You specified **–f**, or **chown** successfully changed the ownership of all the specified files and directories.

1

Failure due to any of the following:

- Inability to access a specified file.
- Inability to change the owner of a specified file.
- Inability to read the directory containing the directory entry of the file.
- An irrecoverable error was encountered when using the **–R** option.

2

Failure due to any of the following:

- The command line contained an incorrect option.
- The command line had too few arguments.
- An owner was specified with a user ID that the system did not recognize.

Portability

POSIX.2, UNIX systems. The **–f** option is an extension of the POSIX standard.

Related Commands

chgrp, **chmod**

cksum — Calculate and write checksums and byte counts

cksum [**–ciprt**] [*file ...*]

Purpose

cksum calculates and displays a checksum for each input *file*. A *checksum* is an error-checking technique used by many programs as a quick way to compare files that have been moved from one location to another to ensure that no data has been lost. It also displays the number of 8-bit bytes in each *file*.

If you do not specify any files on the command line, or if you specify – as the file name, **cksum** reads the standard input.

The output has the form:

```
checksum     bytecount     filename
```

Options

cksum can calculate checksums in a variety of ways. The default is compatible with the POSIX.2 standard. You can specify other algorithms with the following options. The POSIX standard does not recognize these algorithms; the OpenExtensions shell provides them for compatibility with the UNIX **sum** command.

–c

    Uses a standard 16-bit cyclic redundancy check (CRC-16).

–i

    Uses the CCITT standard cyclic redundancy check (CRC-CCITT). Data communication network protocols often use a cyclic redundancy check to ensure proper transmission. This algorithm is more likely to produce a different sum for inputs—the only difference is byte order.

–p

    Uses the POSIX.2 checksum algorithm. This is the default.

–r

    enables the use of an alternate checksum algorithm that has the advantage of being sensitive to byte order.

–t

    Produces a line containing the total number of bytes of data read as well as the checksum of the concatenation of the input files.

Localization

cksum uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

Successful completion

1

Failure due to any of the following:

- Inability to open input file
- An error reading the input file

2

Unknown command-line option

Portability

POSIX.2, X/Open Portability Guide.

All the listed options are extensions of the POSIX standard.

Related Commands

cmp, **diff**, **ls**, **wc**

cmp — Compare two files

---

cmp [–blsx] *file1 file2* [*seek1*[*seek2*]]

Purpose

cmp compares two files. If either file name is –, **cmp** reads the standard input for that file. By default, **cmp** begins the comparison with the first byte of each file. If you specify *seek1* and/or *seek2*, **cmp** uses it as a byte offset into *file1* or *file2* (respectively), and comparison begins at that offset instead of at the beginning of the files. The comparison continues (1 byte at a time) until a difference is found, at which point the comparison ends and **cmp** displays the byte and line number where the difference occurred. **cmp** numbers bytes and lines beginning with 1.

Options

cmp supports the following options:

–b
> Compares single blocks at a time. Normally, **cmp** reads large buffers of data into memory for comparison.

–l
> Causes the comparison and display to continue to the end; however, **cmp** attempts no resynchronization. **cmp** displays the byte number (in decimal) and the differing bytes (in octal) for each difference found.

–s
> Suppresses output and returns a nonzero status if the files are not identical.

–x
> Displays the differing bytes shown by the **–l** option in hex; normally **cmp** displays them in octal.

Localization

cmp uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
> The files were identical.

1
> The files were not identical.

2
> Failure because of an error opening or reading an input file.

Messages and Return Codes

Possible error messages include:

EOF on *filename*
> **cmp** reached the end of the file on the specified file before reaching the end of the file on the other file.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–b** and **–x** options and the *seek* pointers are extensions of the POSIX standard.

Related Commands

comm, **diff**, **uniq**

cms — Enter a CMS command from the shell

---

cms *cms_command_string*

Purpose

The **cms** built-in command allows any CP or CMS command to be executed from the shell environment. Abbreviations, synonyms and EXECs are respected. The CMS line-mode output of the command is written to standard output, while any full screen interactions performed by the command will interact directly with the user's console. The CP line-mode output is written to the virtual machine console.

The *cms_command_string* is the CMS command, including any operands or options, following the syntax of the command. CMS command syntax includes the characters '*****', '**)**' and '**(**'. Therefore, these characters must be enclosed in single or double quotation marks to prevent the shell from interpreting them. The *cms_command_string* is limited to 238 characters in length. Any characters that are included in the *cms_command_string* past the first 238 characters are ignored.

Note: OpenExtensions C or C++ applications that reside on minidisks or in CMS shared file system directories cannot be executed directly with the cms built-in command. You must create an external link by using the OPENVM CREATE EXTLINK command to point to the application program. Then use the BFS path name of the application program to invoke it.

Exit Values

Possible exit status values are:

0

Successful completion

≠0

Failure due to a problem encountered by the command or the CMS command processor. The returned value is the return code from the command.

Portability

cms is a built-in shell command.

cmsfile — Redirect contents of standard input

<div style="border:1px solid">

cmsfile[−a][−f infile]outfile

</div>

Purpose

cmsfile is a shell pipe stage to redirect the standard input stream to an externally linked CMSDATA file. **cmsfile** can also be used as a shell command by specifying the **—f** option. This command is designed to serve as a substitute for the **">"** and **">>"** redirection functions, which do not support external links.

Options

cmsfile supports the following options:

−a

    Appends the input data to the CMS data file outfile.

−f

    Specifies a filepath, infile, to be read as input in place of the shell's standard input stream (STDIN). **cmsfile** is used as a stand-alone shell command when this argument is used.

Examples

1. The pipes;

   ```
   cat a.b > cmsfile ofile
   cat a.b >> cmsfile ofile
   cat > cmsfile ofile
   ```

   are examples of invalid pipe because they use incorrect redirection operators.

2. The pipes;

   ```
   tar -c * | compress | cmsfile ofile.tar
   cat a.b | cmsfile ofile
   ```

   are examples of pipes correctly using the last pipe stage to redirect the standard input stream to the external link.

3. Use the OPENVM CREATE EXTL command to create external cmsdata links. Use the option string on the cmsdata version of OPENVM CREATE EXTL to specify the format that the output file on CMS will have. For example:

   ```
   OPENVM CREATE EXTL ofile.tar CMSDATA OFILE TAR A,&&B
   ```

Localization

cmsfile uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Usage Notes

1. The output file, outfile, must be an externally linked CMSDATA file. If the file does not exist it will be created. If the file already exists then, using the **-a** argument will cause the input file/stream to be appended as a new record at the end of the file. Otherwise, the existing file will be removed prior to writing the first character of the input file/stream.

2. The **-f** argument cannot be used to redirect the standard input, standard output, or standard error streams.

3. **cmsfile** cannot be used as the target of redirection using the redirection operators ">", ">>", "<>", ">|", ">&", or ">&-".

4. Using **cmsfile** as a stand-alone command has performance advantages over using **cmsfile** as a pipe stage, when copying a file to CMS. The command **cmsfile —f a.b cms.output** eliminates a great deal of the I/O and processor time required to execute the equivalent shell pipe **cat a.b | cmsfile cms.output**.

5. The **ls** command is called by **cmsfile** during processing. Therefore, **ls** messages may appear during **cmsfile** processing. These messages normally appear when the output external link is missing.

6. The format of the CMS file created is determined by options specified on the OPENVM CREATE EXTL command used to define the external link between BFS and CMS.

Exit Values

Possible exit status values are:

0
> Successful completion.

1
> Failure due to any of the following:
> - Inability to open the input file
> - The input file is the same as the output file
> - The output file is not an externally linked cmsdata file
> - Inability to open the output file

2
> An incorrect command-line argument.

Messages and Return Codes

Possible error messages include:

Cannot allocate buffer
> There is not enough memory to allow **cmsfile** to set up one or more internal buffers.

External link name was not found
> The file, ofile, does not exist, or is not an externally linked cmsdata file.

Pipe() failed
> The C pipe() function failed while initializing an unnamed pipe between the **cmsfile** and the **ls** commands.

Cannot determine PATH_MAX
> **cmsfile** could not determine the value of the system PATH_MAX environment variable.

The file referred to is an external link
> An error has occurred while opening the output file. This is probably due to an error in the access mode or one or more keyword parameters in the OPENVM CREATE EXTLINK command used to link the output file, **ofile**, to it's CMS file.

Portability

None; this is a z/VM specific command/utility.

: (colon) — Do nothing, successfully

: [*argument* **...**]

Purpose

The **:** (colon) command is used when a command is needed, as in the **then** condition of an **if** command, but nothing is to be done by the command. This command simply yields an exit status of zero (success). This can be useful, for example, when you are evaluating shell expressions for their side effects.

Examples

```
: ${VAR:="default value"}
```

sets *VAR* to a default value if and only if it is not already set.

Usage Notes

This command is built into the shell.

Exit Values

Since this command always succeeds, the only possible exit status is:

0
  Successful completion.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

sh, **true**

comm — Show and select or reject lines common to two files

comm [**−123**] *file1 file2*

Purpose

comm locates identical lines within files sorted in the same collating sequence, and produces three columns; the first contains lines found only in the first file, the second lines only in the second file, and the third lines that are in both files.

Options

−1

    Suppresses lines that appear only in *file1*

−2

    Suppresses lines that appear only in *file2*

−3

    Suppresses lines that appear both in *file1* and *file2*

The options suppress individual columns. Thus, to list only the lines common to both files, use:

```
comm -12
```

To find lines unique to one file or the other, use:

```
comm -3
```

Observe that `comm  -123` displays nothing.

Localization

comm uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

    Successful completion

1

    Failure because of an error opening or reading an input file

2

    Failure that generated a usage message, such as naming only one input file.

Incorrect command-line options are reported but do not affect the exit status value.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

cmp, **diff**, **sort**, **uniq**

command — Run a simple command

```
command [–p] command-name[ argument...]
command [–V | –v] command-name
```

Purpose

command causes the shell to suppress its function lookup and execute the given command name and arguments as though they made up a standard command line. In most cases, if *command-name* is not the name of a function, the results are the same as omitting command. If, however, *command-name* is a special built-in utility (see sh), some unique properties of special built-ins do not apply:

- A syntax error in the utility does not cause the shell running the utility to abort.
- Variable assignments specified with the special built-in utility do not remain in effect after the shell has run the utility.

Options

command supports the following options:

–p

    Searches for *command-name* using the default system variable **PATH**.

–v

    Writes a string indicating the path name or command that the shell uses to invoke *command-name*.

–V

    Writes a string indicating how the shell interprets *command-name*. If *command-name* is a utility, regular built-in utility, or an implementation-provided function found using the **PATH** variable, the string identifies it as such and includes the absolute path name. If *command-name* is an alias, function, special built-in utility, or reserved word, the string identifies it as such and includes its definition if it is an alias.

Examples

Typically, you use command when you have a command that may have the same name as a function. For example, here is a definition of a cd function that not only switches to a new directory, but also uses ls to list the contents of that directory:

```
function cd {
    command cd $1
    ls
}
```

Inside the function, we use command to get at the real cd. If we didn't do this, the cd function would call itself in an infinite recursion.

Environment Variables

PATH

    Contains a list of directories for command to use when searching for *command-name* except as described under the –p option.

Localization

command uses the following localization environment variables:

- **LANG**

- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

If you specified −v, possible exit status values are:

0
    Successful completion.

1
    command could not find *command-name*, or an error occurred.

2
    Failure due to incorrect command-line argument.

If you did not specify −v, possible exit status values are:

126
    command found *command-name,* but failed to invoke it.

127
    An error occurred in the command utility or it could not find *command-name.*

Otherwise, the exit status of command is the exit status of *command-name.*

Portability

POSIX.2.

Related Commands

sh

compress — Use Lempel-Ziv compression

```
compress [–DdfVv] [–b bits] [file...]
compress [–cDdfVv] [–b bits] [file]
```

Purpose

compress uses the Lempel-Ziv compression techniques to compress data in a file or from the standard input. Each file in the input file list is replaced by the compressed form. The compressed file has the same name as the input file but with a .Z suffix. For example, the command `compress myfile.abc` replaces the file named myfile.abc with the compressed form named myfile.abc.Z. If you do not specify any input files, **compress** reads data from the standard input and writes the compressed result to the standard output.

If the .Z file already exists and you did not specify the **-f** option, **compress** issues an error message and ends without replacing the file.

compress uses the modified Lempel-Ziv algorithm. It first replaces common substrings in the file by 9-bit codes starting at 257. After it reaches code 512, **compress** begins with 10-bit codes, and continues to use more bits until it reaches the limit set by the **–b** option. After attaining the limit, **compress** periodically checks the compression ratio. If the ratio is increasing, **compress** continues to use the existing code dictionary. However, if the compression ratio decreases, **compress** discards the table of substrings and rebuilds it from scratch. This allows the algorithm to compensate for certain files, such as archives, where individual components have different information content profiles.

This implementation of **compress** is limited to a maximum of 16-bit compression.

Options

compress accepts the following options:

–b

Limits the maximum number of bits of compression to the value *bits*. This value may be an integer from 9 to 16. The default is 16.

–c

Writes the output to the standard output. When you use this option, you can specify only one file on the command line.

–D

Allows an extra degree of compression to be done for files such as sorted dictionaries where subsequent lines normally have many characters in common with the preceding line.

–d

Uncompresses input files instead of compressing them. This works by overlaying the **compress** program with the **uncompress** program. Uncompressing files this way is slower than using **uncompress** directly.

–f

Forces compression even if the resulting file is larger or the output file already exists. When you do not specify this option, files which will be larger after compression are not compressed. **compress** does not print an error message if this happens.

–V

Prints the version number of **compress**.

–v

Prints statistics giving the amount of compression achieved. Statistics give the name of each file compressed and the compression ratio, expressed as a percentage. If the file resulting from compression is larger than the original, the compression ratio is negative.

Localization

compress uses the following localization environment variables:

• **PATH**

See for more information.

Exit Values

Possible exit status values are:

0

Successful completion.

1

Failure due to any of the following:

- Missing or unsupported number of bits after **–b** option
- Failed to execute **uncompress**
- Dictionary option - same count of string exceeded
- Cannot use stat function to get file status information
- Input file not a regular file
- Input file has other links
- Inability to find a file
- Inability to open an input file for reading
- Inability to create or open an output file
- Read error occurred on an input file
- Write error occurred on an output file
- Incorrect command-line option
- No space left on target device
- Insufficient memory to hold the data to be compressed or compression tables

2

Failure due to the following:

- One or more files were not compressed because the compressed version was larger than the original

Messages and Return Codes

Possible error messages include:

compress: Option –b argument missing

You have specified **–b** but did not specify the *bits* argument that must follow.

Bits must be between 9 and 16

The **–b** *bits* option was specified but the *bits* argument was not an integer between 9 and 16.

tempfile already exists; *name*

The temporary file used for compression output already exists. The file must be erased before **compress** can be used.

***name* already exists; not overwritten**

The output file *name* already exists. Specify the **–f** option to force **compress** to overwrite the file.

name not a regular file: unchanged
> _name_ does not refer to a byte file system file. It refers to a directory, socket, pipeline, device, or the standard I/O.

compress: (-D) same count exceeded - aborting
> The maximum count of 255 successive identical characters has been exceeded. The compression has been aborted.

cannot allocate buffer
> There is insufficient memory to create one or more internal buffers used for compression.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

uncompress, **zcat**

continue — Skip to the next iteration of a loop in a shell script

---

continue [*n*]

Purpose

continue skips to the next iteration of an enclosing **for**, **select**, **until**, or **while** loop in a shell script. If a number *n* is given, execution continues at the loop control of the *n*th enclosing loop. The default value of *n* is 1.

Usage Notes

This command is built into the shell.

Exit Values

Possible exit values are:

0
    Successful completion

1
    The value of *n* given was not an unsigned decimal greater than 0.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

break, **sh**

cp — Copy a file

> **cp** [**–fimp**] *file1 file2*
> **cp** [**–fimp**] *file ... directory*
> **cp –R** [**–fimp**] *source... directory*
> **cp –r** [**–fimp**] *source... directory*

Purpose

cp copies files to a target named by the last argument on its command line. If the target is an existing file, **cp** overwrites it; if it does not exist, **cp** creates it. If the target file already exists and does not have write permission, **cp** denies access and continues with the next copy.

If you specify more than two path names, the last path name (that is, the target) must be a directory. If the target is a directory, **cp** copies the sources into that directory with names given by the final component of the source path name.

Options

cp accepts the following options:

–f

   Attempts to replace files that do not have write permission.

–i

   Asks you if you want to overwrite an existing file, whether or not the file is read-only.

–m

   Sets the modification and access time of each destination file to that of the corresponding source file. Normally, **cp** sets the modification time of the destination file to the present.

–p

   Preserves the modification and access times (as the **–m** option does); in addition, it preserves file mode, owner, and group owner, if possible.

–R

   "Clones" the source trees. **cp** copies all the files and subdirectories specified by *source...* into *directory*, making careful arrangements to duplicate special files (FIFO, character special).

–r

   "Clones" the source trees, but makes no allowances for special files (FIFO, character special). Consequently, **cp** attempts to read from a device rather than duplicate the special file. This is similar to, but less useful than, the preferred **–R**.

Localization

cp uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

Successful completion.

1

Failure due to any of the following:

- An argument had a trailing slash (/) but was not a directory.
- Inability to find a file.
- Inability to open an input file for reading.
- Inability to create or open an output file.
- A read error occurred on an input file.
- A write error occurred on an output file.
- The input and output files were the same file.
- An irrecoverable error when using **–r** or **–R**.
- Possible irrecoverable **–r** or **–R** errors include:
  - Inability to access a file.
  - Inability to change permissions on a target file.
  - Inability to read a directory.
  - Inability to create a directory.
  - A target that is not a directory.
  - Source and destination directories are the same.

2

Failure due to any of the following:

- An incorrect command-line option.
- Too few arguments on the command line.
- A target that should be a directory but isn't.
- No space left on target device.
- Insufficient memory to hold the data to be copied.
- Inability to create a directory to hold a target file.

Messages and Return Codes

Possible error messages include:

cannot allocate target string
cp has no space to hold the name of the target file. Try to release some memory to give **cp** more space.

***name* is a directory (not copied)**
You did not specify **–r** or **–R**, but one of the names you asked to copy was the name of a directory.

***target name*?**
You are attempting to copy a file with the **–i** option, but there is already a file with the target name. If you have specified **–f**, you can write over the existing file by typing y and pressing <Enter>. If you do not want to write over the existing file, type n and press <Enter>. If you did not specify **–f** and the file is read-only, you are not given the opportunity to overwrite it.

source *name* and target *name* are identical
The source and the target are actually the same file (for example, because of links). In this case, **cp** does nothing.

> **unreadable directory** *name*
>> **cp** cannot read the specified directory—for example, because you do not have appropriate permission.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–f** and **–m** options are extensions of the POSIX standard.

Related Commands

cat, **cpio**, **mv**, **rm**, **ln**

cpio -- Copy in/out file archives

```
cpio -o [-aBcvyz] [-C blocksize] [-O file] [-V volpat]
cpio -i [-BbcdfmrsStuvqyz] [-C blocksize] [-I file] [-V volpat] [pattern ...]
cpio -p [-aBdlmruv] directory
```

Purpose

cpio reads and writes files called **cpio** *archives*. A **cpio** archive is a concatenation of files and directories preceded by a header giving the file name and other file system information. With **cpio**, you can create a new archive, extract contents of an existing archive, list archive contents, and copy files from one directory to another.

Options

Every call to **cpio** must specify one and only one of the following *selector* options:

-i

> Reads an existing archive (created with the **-o** option) from the standard input. Unless you specify the **-t** option, **cpio** extracts all files matching one or more of the given *pattern* arguments from the archive. Patterns are the same as those used by file name generation (see **sh**). When you do not specify a *pattern* argument, the default pattern * is used; as a result, **cpio** extracts all files.

-o

> Writes a new archive to the standard output, using the list of files read from the standard input. Such a list might be produced by the **ls** or **find** commands. For example:

```
ls . | cpio -o >arch
```

> uses **ls** to list the files of the working directory and then pipes this list as input to **cpio**. The resulting archive contains the contents of all the files, and is written to **arch**.

-p

> Is shorthand for:

```
cpio -o | (cd directory; cpio -i)
```

> where **cpio -i** is performed in the given *directory*. You can use this option to copy entire file trees.

Consult the syntax lines to determine which of the following additional options can be applied with a particular selector option:

-a

> Resets the access time (of each file accessed for copying to the archive) to what it was before the copy took place.

-B

> Uses buffers of 5120 bytes for input and output rather than the default 512-byte buffers.

-b

> causes 16-bit words to be swapped within each longword and bytes to be swapped within each 16-bit word of each extracted file. This facilitates the transfer of information between different processor architectures. This is equivalent to specifying both the **-s** and **-S** options.

-C *blocksize*

> Sets the buffer size to a specified blocksize, rather than the default 512-byte buffers.

-c

> Reads and writes header information in ASCII form. Normally, **cpio** writes the header information in a compact binary format. This option produces an archive more amenable to transfer through nonbinary

streams (such as some data communication links) and is highly recommended for those moving data between different processors.

–d

Forces the creation of necessary intermediate directories when they do not already exist.

–f

Inverts the sense of pattern matching. More precisely, **cpio** extracts a file from the archive if and only if it does *not* match any of the *pattern* arguments.

–I *file*

Causes input to be read from the specified file, rather than from **stdin.**

–l

Gives permission to create a link to a file rather than making a separate copy.

–m

Resets the modification time of an output file to the modification time of the source file. Normally, when **cpio** copies data into a file, it sets the modification time of the file to the time at which the file is written.

–O *file*

Causes output to be written to the specified file, rather than to **stdout.**

–q

Assumes all created files are text. This means that any \r (carriage return) characters are stripped, and only the \n (newlines) are retained.

It is not advisable to use the -q option for converting text to a system-independent format, since that would require all files to be read twice.

–r

Provides an interactive mechanism for selecting and renaming particular files. For each file processed, **cpio** displays the name before copying it to its new location. At this point, you can type in a new name for the file. If you enter an empty line, the file is skipped.

–S

For portability reasons, swaps pairs of 16-bit words within longwords (a 32-bit or 64-bit word) only when extracting files. This option does not affect the headers.

–s

For portability reasons, swaps pairs of bytes within each 16-bit word only when extracting files. **–s** does not affect the headers.

–t

Prevents files extraction, producing instead a table of file names contained in the archive. See the description of the **–v** option.

–u

Copies an archive file to a target file even if the target is newer than the archive. Normally, **cpio** does not copy the file.

–V *volpat*

Provides automatic multivolume support. **cpio** writes output to files, the names of which are formatted using *volpat*. The current volume number replaces any occurrence of # in *volpat*. When you invoke **cpio** with this option, it asks for the first number in the archive set, and waits for you to type the number and a carriage return before its precedes with the operation. **cpio** issues the same sort of message when a write error or read error occurs on the archive; the reasoning is that this kind of error means that **cpio** has reached the end of the volume and should go on to a new one.

–v

Provides more verbose information than usual. **cpio** prints the names of files as it extracts them from or adds them to archives. When you specify both **–v** and **–t**, **cpio** prints a table of files in a format similar to that produced by the `ls –l` command.

–y

When used with **–V**, does not ask for a volume number to begin with, but does ask if it gets a read or write error.

−z

Performs Lempel-Ziv compression. Output is always a 16-bit compression. On input, any compression up to 16-bit is acceptable.

File Formats

A **cpio** archive consists of the concatenation of one or more member files. Each member file contains a header (as described later in this command description) optionally followed by file contents (as indicated in the header). The end of the archive is indicated by another header describing an (empty) file named **TRAILER!!!**.

There are two types of **cpio** archives, differing only in the style of the header. By default, **cpio** writes archives with binary headers.

The information in ASCII archive headers is stored in fixed-width, octal (base 8) numbers, zero-padded on the left. Table 5 on page 87 gives the order and field width for the information in the ASCII header:

Table 5. ASCII Header Format for a cpio File

Width	Field Name	Meaning
6	*magic*	Magic number "070707"
6	*dev*	Device where file resides
6	*ino*	I-number of file
6	*mode*	File *mode*
6	*uid*	Owner user ID (UID)
6	*gid*	Owner group ID (GID)
6	*nlink*	Number of links to the file
6	*rdev*	Device major or minor for a special file
11	*mtime*	Modification time of the file
6	*namesize*	Length of the file name
11	*filesize*	Length of the file to follow

Most of this information is compatible with that returned by the UNIX **stat** function. After this information, *namesize* bytes of the path name is stored. *namesize* includes the null byte of the end of the path name. After this, *filesize* bytes of the file contents are recorded.

Binary headers contain the same information in 2-byte (short) and 4-byte (long) integers as shown in Table 6 on page 87.

Table 6. Binary Header Format for a cpio File

Bytes	Field Name	Meaning
2	*magic*	Magic number "070707"
2	*dev*	Device where file resides
2	*ino*	I-number of file
2	*mode*	File *mode*
2	*uid*	Owner user ID (UID)
2	*gid*	Owner group ID (GID)
2	*nlink*	Number of links to the file
2	*rdev*	Device major or minor for a special file

Table 6. Binary Header Format for a cpio File (continued)

Bytes	Field Name	Meaning
4	*mtime*	Modification time of the file
2	*namesize*	Length of the file name
2	*reserved*	Two bytes of reserved space
4	*filesize*	Length of the file to follow

After this information comes the file name (with *namesize* rounded up to the nearest 2-byte boundary). Then the file contents appear as in the ASCII archive. The byte ordering of the 2- and 4-byte integers in the binary format is machine-dependent, and thus portability of this format is not easily guaranteed.

Compressed **cpio** archives are exactly equivalent to the corresponding archive being passed to a 16-bit **compress** utility.

Usage Notes

1. The byte and word swapping done by the **–b**, **–S**, and **–s** options is effective only for the file data written. With or without the **–c** option, header information is always written in a machine-invariant format.

2. The cpio utility is scheduled to be withdrawn from XPG; for standards compatibility, you should use **pax**.

Exit Values

Possible exit status values are:

0

Successful completion.

1

Failure due to any of the following:

- An incorrect option
- Incorrect command-line arguments
- Out of memory
- Compression error
- Failure on extraction
- Failure on creation

Portability

X/Open Portability Guide, non-Berkeley UNIX systems after Version 7.

The **–q**, **–V**, **–y**, and **–z** options are specific to the OpenExtensions shell.

Related Commands

cp, **dd**, **find**, **ls**, **mv**, **pax**, **tar**

cut — Cut out selected fields from each line of a file

```
cut −b list [−n] [file...]
cut −c list [file...]
cut −f list [−d char] [−s] [file...]
```

Purpose

cut reads input from files, each specified with the *file* argument, and selectively copies sections of the input lines to the standard output. If you do not specify any *file*, or you specify a file named −, **cut** reads from standard input.

Options

cut accepts the following options:

−b *list*
> Invokes byte position mode. After this comes a list of the byte positions you want to display. This list may contain multiple byte positions, separated by commas (**,**) or blanks or ranges of positions separated by dashes (−). Since the list must be a single argument, shell quoting is necessary if you use blanks. You can combine these to allow selection of any byte positions of the input.

−c *list*
> Invokes character-position mode. After this comes a list of character positions to retain in the output. This list can contain many character positions, separated by commas (**,**) or blanks or ranges of positions separated by a dash (−). Since the list must be a single argument, shell quoting is necessary if you use blanks. You can combine these to allow selection of any character positions of the input.

−d *char*
> Specifies *char* as the character that separates fields in the input data; by default, this is the horizontal tab.

−f *list*
> Invokes field delimiter mode. After this comes a list of the fields you want to display. You specify ranges of fields and multiple field numbers in the same way you specify ranges of character positions and multiple character positions in **−c** mode.

−n
> Does not split characters. If the low byte in a selected range is not the first byte of a character, **cut** extends the range downward to include the entire character; if the high byte in a selected range is not the last byte of a character, **cut** limits the range to include only the last entire character before the high byte selected. If **−n** is selected, **cut** does not list ranges that do not encompass an entire character, and these ranges do not cause an error.

−s
> Does not display lines that do not contain a field separator character. Normally, **cut** displays lines that do not contain a field separator character in their entirety.

Examples

```
cd /bin
ls −al | cut −c 42−48,54−66
```

prints a directory listing containing file creation dates and file names of files in the working directory.

Localization

cut uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

Successful completion

1

Failure due to any of the following:

- Cannot open the input file
- Out of memory

2

Failure due to any of the following:

- An incorrect command-line argument
- You did not specify any of **–b**, **–c**, or **–f**
- You omitted the *list* argument
- Badly formed *list* argument

Portability

POSIX.2, X/Open Portability Guide, UNIX System V.

Related Commands

paste, **uname**

date — Display the date and time

> **date** [**–cu**] [**+***format*]

Purpose

date displays the operating system's idea of the current date and time.

The following example shows the default format of the date:

```
Wed Feb 26 14:01:43 EST 1986
```

Options

date accepts the following options:

–c
> Displays the date and displays the time according to Greenwich Mean Time (Coordinated Universal Time) using *CUT* as the time zone name.

–u
> Displays the date and displays the time according to Greenwich Mean Time (Coordinated Universal Time) using *GMT* as the time zone name.

If the argument to **date** begins with a + character, **date** uses *format* to display the date. **date** writes all characters in *format*, with the exception of the % and the character that immediately follows it, directly to the standard output. After **date** exhausts the *format* string, it outputs a newline character. The % character introduces a special format field similar to the **printf()** function in the C library. **date** recognizes the following field descriptors:

%A
> The full weekday name (for example, Sunday).

%a
> The three-letter abbreviation for the weekday (for example, Sun).

%B
> The full month name (for example, February).

%b
> The three-letter abbreviation for the month name (for example, Feb).

%C
> The first two digits of the year (00 to 99).

%c
> The local representation of the date and time (see %D and %T).

%D
> The date in the form *mm*/*dd*/*yy*.

%d
> The two-digit day of the month as a number (01 to 31).

%e
> The day of the month in a two-character, right-justified, blank-filled field.

%H
> The two-digit hour (00 to 23).

%h
> The three-letter abbreviation for the month (for example, Feb). The %h is a synonym for %b.

%I
> The hour in the 12-hour clock representation (01 to 12).

%j
> The numeric day of the year (001 to 366).

%M
> The minute (00 to 59).

%m
> The month number (01 to 12).

%n
> The newline character.

%p
> The local equivalent of a.m. or p.m.

%r
> The time in a.m.–p.m. notation (11:53:29 a.m.).

%S
> The seconds (00 to 61). There is an allowance for two leap seconds.

%T
> The time (14:53:29).

%t
> A tab character.

%U
> The week number in the year, with Sunday being the first day of the week (00 to 53).

%W
> The week number in the year, with Monday being the first day of the week (00 to 53).

%w
> The weekday number, with Sunday being 0.

%X
> The local time representation (see %T).

%x
> The local date representation (see %D).

%Y
> The year.

%y
> The two-digit year.

%Z
> The time zone name (for example, EDT).

%%
> A percent-sign character.

The **date** command also supports the following modified field descriptors to indicate a different format as specified by the locale indicated by **LC_TIME**. If the current locale does not support a modified descriptor, **date** uses the unmodified field descriptor value.

%EC
> The name of the base year (period) in the current locale's alternate representation.

%Ec
> The current locale's alternate date and time representation.

%Ex
> The current locale's alternate date representation.

%EY
> The full alternate year representation.

%Ey
   The offset from %EC (year only) in the current locale's alternate representation.

%Od
   The day of the month using the current locale's alternate numeric symbols.

%Oe
   The day of the month using the current locale's alternate numeric symbols.

%OH
   The hour (24-hour clock) using the current locale's alternate numeric symbols.

%OI
   The hour (12-hour clock) using the current locale's alternate numeric symbols.

%OM
   The minutes using the current locale's alternate numeric symbols.

%Om
   The month using the current locale's alternate numeric symbols.

%OS
   The seconds using the current locale's alternate numeric symbols.

%OU
   The week number of the year (0–53) (with Sunday as the first day of the week) using the current locale's alternate numeric symbols.

%OW
   The week number of the year (0–53) (with Monday as the first day of the week) using the current locale's alternate numeric symbols.

%Ow
   The weekday as a number using the current locale's alternate numeric symbols (Sunday=0).

%Oy
   The year (offset from %C) using the current locale's alternate numeric symbols.

Examples

The command:

```
date '+%a %b %e %T %Z %Y'
```

produces the date in the default format—as shown at the start of this command description.

Environment Variables

TZ
   Gives the time zone for **date** to use when displaying the times. This is ignored if you specify either the **–c** or the **–u** option. See **TZ** in the **sh** command environment variable list.

Localization

date uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
　Successful completion.

>0
　Failure due to any of the following:

- An incorrect command line option.
- Too many arguments on the command line.
- A bad date conversion.
- A formatted date that was too long.
- You do not have permission to set the date.

Messages and Return Codes

Possible error messages include:

Bad format character *x*
　A character following "%" in the *format* string was not in the list of field descriptors.

No permission to set date
　The system has denied you the right to set the date.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–c** option is an extension of the POSIX standard.

Related Commands

touch

dd — Convert and copy a file

> dd [bs=*size*] [cbs=*size*] [conv=*conversion*] [count=*n*] [ibs=*size*] [if=*file*]
>   [imsg=*string*] [iseek=*n*] [obs=*size*] [of=*file*] [omsg=*string*]
>   [seek=*n*] [skip=*n*]

Purpose

dd reads and writes data by blocks. It is frequently used for such devices as tapes that have discrete block sizes, or for fast multisector reads from disks. **dd** performs conversions to accommodate nonprogrammable terminals, which require deblocking, conversion to and from EBCDIC, and fixed-length records.

dd processes the input data as follows:

1. **dd** reads an input block.
2. If this input block is smaller than the specified input block size, **dd** pads it to the specified size with null bytes. When you also specify a **block** or **unblock** conversion, **dd** uses spaces instead of null bytes.
3. If you specified **bs=***s* and requested no conversion other than **sync** or **noerror**, **dd** writes the padded (if necessary) input block to the output as a single block and omits the remaining steps.
4. If you specified the *swab* conversion, **dd** swaps each pair of input bytes. If there is an odd number of input bytes, **dd** does not attempt to swap the last byte.
5. **dd** performs all remaining conversions on the input data independently of the input block boundaries. A fixed-length input or output record may span these boundaries.
6. **dd** gathers the converted data into output blocks of the specified size. When **dd** reaches the end of the input, it writes the remaining output as a block (without padding if **conv=***sync* is not specified). As a result, the final output block may be shorter than the output block size.

Options

bs=*size*
> Sets both input and output block sizes to *size* bytes. You can suffix this decimal number with **w**, **b**, **k**, or **x** *number*, to multiply it by 2, 512, 1024, or *number,* respectively. You can also specify *size* as two decimal numbers (with or without suffixes) separated by **x** to indicate the product of the two values. Processing is faster when **ibs** and **obs** are equal, since this avoids buffer copying. The default block size is 1B. **bs=***size* supersedes any settings of **ibs=***size* or **obs=***size*.
>
> If you specify **bs=***size* and you request no other conversions than **noerror**, **notrunc**, or **sync**, **dd** writes the data from each input block as a separate output block; if the input data is less than a full block and you did not request **sync** conversion, the output block is the same size as the input block.

cbs=*size*
> sets the size of the conversion buffer used by various **conv** options.

conv=*conversion*[, *conversion*, ...]
> **Note:** To copy a file and convert between code page 01047 (used in the OpenExtensions shell) and ASCII, use the CMS COPYFILE command, *not* the **dd** command. The **ascii**, **ebcdic**, and **ibm** conversion options are provided for compatibility purposes only.
>
> *conversion* can be any one of the following:
>
> **ascii**
> > Converts EBCDIC input to ASCII for output. This is 8-bit extended US ASCII. **dd** copies **cbs** bytes at a time to the conversion buffer, maps them to ASCII, strips trailing blanks, adds a newline, and copies this line to the output buffer.

dd

> **block**
>> Converts variable-length records to fixed-length records. **dd** treats the input data as a sequence of variable-length records (each terminated by a newline or an EOF character) independent of the block boundaries. **dd** converts each input record by first removing any newline characters and then padding (with spaces) or truncating the record to the size of the conversion buffer. **dd** reports the number of truncated records on the standard error. You must specify **cbs=**_size_ with this conversion.
>
> **convfile**
>> Uses **convfile** as a translation table if it is not one of the conversion formats listed here and it is the name of a file of exactly 256 bytes.
>>
>> You can perform multiple conversions at the same time by separating arguments to **conv** with commas; however, some conversions are mutually exclusive (for example, **ucase** and **lcase**).
>>
>> **Note:** When you specify one or more of the character set conversions (**ascii**, **ebcdic**, **ibm**, or **convfile**), **dd** assumes that all characters are singlebyte characters, regardless of the locale.
>
> **ebcdic**
>> Converts ASCII input to EBCDIC for output. **dd** copies a line of ASCII to the conversion buffer, discards the newline, pads it out with trailing blanks to **cbs** bytes, maps it to EBCDIC, and copies it to the output buffer.
>
> **ibm**
>> Like **ebcdic**, converts ASCII to EBCDIC; however, **ibm** ignores the top (eighth) bit.
>
> **lcase**
>> Converts uppercase input to lowercase.
>
> **noerror**
>> Ignores errors on input.
>
> **notrunc**
>> Does not truncate the output file. **dd** preserves blocks in the output file that it does not explicitly write to.
>
> **swab**
>> Swaps the order of every pair of input bytes. If the current input record has an odd number of bytes, this conversion does not attempt to swap the last byte of the record.
>
> **sync**
>> Specifies that **dd** is to pad any input block shorter than **ibs** to that size with NUL bytes before conversion and output. If you also specified _block_ or _unblock_, **dd** uses spaces instead of null bytes for padding.
>
> **ucase**
>> Converts lowercase input to uppercase.
>
> **unblock**
>> Converts fixed-length records to variable-length records by reading a number of bytes equal to the size of the conversion buffer, deleting all trailing spaces, and appending a newline character. You must specify **cbs=**_size_ with this conversion.

count=_n_
> Copies only _n_ input blocks to the output.

ibs=_size_
> Sets the input block _size_ in bytes. You specify it in the same way as with the **bs** option.

if=_file_
> Reads input data from _file_. If you don't specify this option, **dd** reads data from the standard input.

imsg=_string_
> Displays _string_ when all data has been read from the current volume, replacing all occurrences of %d in _string_ with the number of the next volume to be read. **dd** then reads and discards a line from the controlling terminal.

iseek=*n*

seeks to the *n*th block of the input file. The distinction between this and the **skip** option is that **iseek** does not read the discarded data. There are some devices, however, such as tape drives and communication lines, on which seeking is not possible, so only **skip** is appropriate.

obs=*size*

Sets the output block *size* in bytes. You specify it in the same way as the **bs** value. The size of the destination should be a multiple of the value chosen for *size*. For example, if you choose **obs=**10K, the destination's size should be a multiple of 10K.

of=*file*

Writes output data to *file*. If you don't specify this option, **dd** writes data to the standard output. **dd** truncates the output file before writing to it, unless you specified the **seek=*n*** operand. If you specify **seek=*n***, but do not specify **conv=***notrunc*, **dd** preserves only those blocks in the output file over which it seeks. If the size of the seek plus the size of the input file is less than the size of the output file, this can result in a shortened output file.

omsg=*string*

Displays *string* when **dd** runs out of room while writing to the current volume. Any occurrences of %d in *string* are replaced with the number of the next volume to be written. **dd** then reads and discards a line from the controlling terminal.

seek=*n*

Initially seeks to the *n*th block of the output file.

skip=*n*

Reads and discards the first *n* blocks of input.

Examples

Entering:

```
dd if=in of=out conv=ascii cbs=80 ibs=6400 obs=512
```

converts 80-byte fixed-length EBCDIC card images in 6400-byte input blocks to variable-length ASCII lines, 512 bytes to the output block.

Localization

dd uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

Successful completion

1

Failure due to any of the following:

- I/O errors on read/write
- Incorrect command-line option

2

Failure resulting in a usage message such as:

- An option that should contain **=** does not
- Unknown or incorrect command-line option

Messages and Return Codes

Possible error messages include:

badly formed number *number*
A value specified as a number (for example, a block size) does not have the form of a number as recognized by **dd**. For example, you may have followed the number with a letter that **dd** does not recognize as a block-size unit (**w**, **b**, **k**).

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **conv=ascii**, **conv=ebcdic**, **conv=ibm**, **conv=convfile**, **iseek**, **imsg**, and **omsg** options plus the **w** suffix described in the **bs=** option are all extensions of the POSIX standard.

Related Commands

cp, **cpio**, **mv**, **tr**

diff — Compare two text files and show the differences

diff [-befHhimnrsw] [-C *n*] [-c[*n*]] [-D*ifname*] *path1 path2*

Purpose

The **diff** command attempts to determine the minimal set of changes needed to convert a file whose name is specified by the *path1* argument into the file specified by the *path2* argument.

If either (but only one) file name is **–**, **diff** uses a copy of the standard input for that file. If exactly one of *path1* or *path2* is a directory, **diff** uses a file in that directory with the same name as the other file name. If both are directories, **diff** compares files with the same file names under the two directories; however, it does not compare files in subdirectories unless you specify the **–r** option. When comparing two directories, **diff** does not compare character special files, or FIFO special files with any other files.

By default, output consists of descriptions of the changes in a style like that of the **ed** text editor. A line indicating the type of change is given. The three types are a (append), d (delete), and c (change). The output is symmetric: A delete in *path1* is the counterpart of an append in *path2*. **diff** prefixes each operation with a line number (or range) in *path1* and suffixes each with a line number (or range) in *path2*. After the line giving the type of change, **diff** displays the deleted or added lines, prefixing lines from *path1* with < and lines from *path2* with >.

Options

Options that control the output or style of file comparison are:

–b
> Ignores trailing blanks and tabs and considers adjacent groups of blanks and tabs elsewhere in input lines to be equivalent.

–C *n*
> Is equivalent to **-c***n*.

–c[*n*]
> Shows *n* lines of context before and after each change. The default value for *n* is 3. **diff** marks lines removed from *path1* with **–**, lines added to *path2* with **+**, and lines changed in both files with ! .

–D*ifname*
> Displays output that is the appropriate input to the C preprocessor to generate the contents of *path2* when *ifname* is defined, and the contents of *path1* when *ifname* is not defined.

–e
> writes out a script of commands for the **ed** text editor, which converts *path1* to *path2*. **diff** sends the output to the standard output.

–f
> Writes a script similar to the one produced under **–e** to standard output, but does not adjust the line numbers to reflect earlier editing changes; instead, they correspond to the line numbers in *path1*.

–H
> Uses the half-hearted (**–h**) algorithm only if the normal algorithm runs out of system resources.

–h
> Uses a fast, half-hearted algorithm instead of the normal **diff** algorithm. This algorithm can handle arbitrarily large files; however, it is not particularly good at finding a minimal set of differences in files with many differences.

–i
> Ignores the case of letters when doing the comparison.

diff

–m
> Produces the contents of *path2* with extra formatter request lines interspersed to show which lines were added (those with vertical bars in the right margin) and deleted (indicated by a ⋆ in the right margin).

–n
> Is accepted for compatibility, but performs no function.

–r
> Compares corresponding files under the directories, and recursively compares corresponding files under corresponding subdirectories under the directories. You can use this option when you specify two directory names on the command line.

–s
> Compares two directories, file by file, and prints messages for identical files between the two directories.

–w
> Ignores white space when making the comparison.

Examples

The following example illustrates the effect of the **–c** option on the output of the **diff** command. The following two files, **price1** and **price2**, are compared with and without the use of the **–c** option.

The contents of **price1** are as follows:

```
Company X Price List:
$  0.39  -- Package of Groat Clusters
$  5.00  -- Candy Apple Sampler Pack
$ 12.00  -- Box of Crunchy Frog Chocolates
$ 15.99  -- Instant Rain (Just Add Water)
$ 20.00  -- Asparagus Firmness Meter
$ 25.00  -- Package of Seeds for 35 Herbs
$ 30.00  -- Child's Riding Hood (Red)
$ 35.00  -- Genuine Placebos
$ 45.00  -- Case of Simulated Soy Bean Oil
$ 75.88  -- No-Name Contact Lenses
$ 99.99  -- Kiddie Destructo-Bot
$125.00  -- Emperor's New Clothes
```

The contents of **price2** are as follows:

```
Company X Price List:
$  0.39  -- Package of Groat Clusters
$  5.49  -- Candy Apple Sampler Pack
$ 12.00  -- Box of Crunchy Frog Chocolates
$ 15.99  -- Instant Rain (Just Add Water)
$ 17.00  -- Simulated Naugahyde cleaner
$ 20.00  -- Asparagus Firmness Meter
$ 25.00  -- Package of Seeds for 35 Herbs
$ 30.00  -- Child's Riding Hood (Red)
$ 35.00  -- Genuine Placebos
$ 45.00  -- Case of Simulated Soy Bean Oil
$ 75.88  -- No-Name Contact Lenses
$ 99.99  -- Kiddie Destructo-Bot
```

The command:

```
    diff price1 price2
```

results in the following output:

```
4c4
< $  5.00  -- Candy Apple Sampler Pack
--->
 $  5.49  -- Candy Apple Sampler Pack
6a7
> $ 17.00  -- Simulated Naugahyde cleaner
14d14
< $125.00  -- Emperor's New Clothes
```

The addition of the **–c** option, as in:

```
diff -c price1 price2
```

results in the following output:

```
*** price1 Wed Mar 04 10:08:40 1993
--- price2 Wed Mar 04 10:09:10 1993
***************
*** 1,9 ****
Company X Price List:

    $  0.39  -- Package of Groat Clusters
!   $  5.00  -- Candy Apple Sampler Pack
    $ 12.00  -- Box of Crunchy Frog Chocolates
    $ 15.99  -- Instant Rain (Just Add Water)
    $ 20.00  -- Asparagus Firmness Meter
    $ 25.00  -- Package of Seeds for 35 Herbs
    $ 30.00  -- Child's Riding Hood (Red)
--- 1,10 ----
    Company X Price List:

    $  0.39  -- Package of Groat Clusters
!   $  5.49  -- Candy Apple Sampler Pack
    $ 12.00  -- Box of Crunchy Frog Chocolates
    $ 15.99  -- Instant Rain (Just Add Water)
+   $ 17.00  -- Simulated Naugahyde cleaner
    $ 20.00  -- Asparagus Firmness Meter
    $ 25.00  -- Package of Seeds for 35 Herbs
    $ 30.00  -- Child's Riding Hood (Red)
***************
*** 11,14 ****
    $ 45.00  -- Case of Simulated Soy Bean Oil
    $ 75.88  -- No-Name Contact Lenses
    $ 99.99  -- Kiddie Destructo-Bot
-   $125.00  -- Emperor's New Clothes
--- 12,14 ----
```

diff –c marks lines removed from **price1** with **–**, lines added to **price1** with **+** and lines changed in both files with !. In the example, **diff** shows the default three lines of context around each changed line. One line was changed in both files (marked with !), one line was added to **price1** (marked with **+**), and one line was removed from **price1** (marked with **–**).

Note: If there are no marks to be shown in the corresponding lines of the file being compared, the lines are not displayed. Lines 12 to 14 of **price2** are suppressed for this reason.

Localization

diff uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**

See for more information.

Exit Values

Possible exit status values are:

0

   No differences between the files compared.

1

   diff compared the files and found them to be different.

2
> Failure due to any of the following:
>
> - Incorrect command-line argument
> - Inability to find one of the input files
> - Out of memory
> - Read error on one of the input files

4
> At least one of the files is a binary file containing embedded NUL (\0) bytes or newlines that are more than LINE_MAX bytes apart.

Messages and Return Codes

Possible error messages include:

Binary files *filename* and *filename* differ
> The two specified files are binary files. **diff** has compared the two files and found that they are not identical. With binary files, **diff** does not try to report the differences.

file *filename*: no such file or directory
> The specified *filename* does not exist. *filename* was either typed explicitly, or generated by **diff** from the directory of one file argument and the basename of the other.

Files *file1* and *file2* are identical
> The **–s** option was specified and the two named files are identical.

Common subdirectories: *name* and *name*
> This message appears when **diff** is comparing the contents of directories, but you have not specified **–r**. When **diff** discovers two subdirectories with the same name, it reports that the directories exist, but it does not try to compare the contents of the two directories.

Insufficient memory (try diff –h)
> **diff** ran out of memory for generating the data structures used in the file differencing algorithm (see "Limits" on page 102). The **–h** option of **diff** can handle any size file without running out of memory.

Internal error—cannot create temporary file
> **diff** was unable to create a working file that it needed. Ensure that you either have a directory /**tmp** or that the environment contains a variable **TMPDIR,** which names a directory where **diff** can store temporary files. Also, ensure that there is sufficient file space in this directory.

Missing *ifdef* symbol after -D
> You did not specify a conditional label on the command line after the **–D** option.

Only one file may be –
> Of the two input files normally found on the command line of **diff**, only one can be the standard input.

Too many lines in *filename*
> A file of more than the maximum number of lines (see "Limits" on page 102) was given to **diff**.

Limits

The longest input line is 1024 bytes. Except under **–h**, files are limited to INT_MAX lines. INT_MAX is defined in **limits.h**.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–D**, **–f**, **–H**, **–h**, **–i**, **–m**, **–s**, and **–w** options, and the *n* argument to the **–c** option, are extensions of the POSIX standard.

Related Commands

J. W. Hunt and M. D. McIlroy, "An Algorithm for Differential File Comparison", *Computing Science Technical Report* 41 (Bell Telephone Laboratories).

cmp, **comm**

dirname — Return the directory components of a path name

> **dirname** *pathname*

Purpose

dirname deletes the trailing part of a file name. The result is the path name of the directory that contains the file. This is useful in shell scripts.

Note: dirname makes no attempt to validate the path name; for validation, use **pathchk**.

dirname follows these rules:

1. If *pathname* is //, return it.
2. Otherwise, if it is all slashes, return one slash.
3. Otherwise, remove all trailing slashes.
4. If there are no slashes remaining in *pathname,* return period (.).
5. Otherwise, remove trailing nonslash characters.
6. If the remaining string is //, return it.
7. Otherwise, remove any trailing slashes.
8. If the resulting string is empty, return period (.).
9. Otherwise, return the resulting string.

Examples

The command:

```
dirname src/lib/printf.c
```

produces:

```
src/lib
```

Localization

dirname uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

The only possible exit status value is:

0
 Successful completion

1
 Failed

2
  Unknown command-line option

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

basename, **pathchk**

. (dot) — Run a shell file in the current environment

> *. file* [*argument ...*]

Purpose

. (**dot**) runs a shell script in the current environment and then returns. Normally, the shell runs a command file in a subshell so that changes to the environment by such commands as **cd**, **set**, and **trap** are local to the command file. The **.** (**dot**) command circumvents this feature.

If there are slashes in the file name, **.** (**dot**) looks for the named file. If there are no slashes **.** (**dot**) uses the search **PATH** variable to find *file*. This may surprise some people when they use dot to run a file in the working directory, but their search rules are not set up to look at the working directory. As a result, the shell doesn't find the shell file. If you have this problem, you can use:

```
. ./file
```

This indicates that the shell file you want to run is in the working directory. Also, the file need not be executable, even if it is looked for on the **PATH**. If you specify an argument list *argument ...*, **.** (**dot**) sets the positional parameters to this list before execution.

Environment Variables

PATH
  Contains a list of directories that **.** (**dot**) searches when attempting to find *file*.

Usage Notes

This command is built into the shell.

Exit Values

Possible exit status values are:

1
  The path search failed or *file* is unreadable

2
  Failure because of an incorrect command-line option

Otherwise, the exit status is the exit status of the last command run from the script.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

cd, **set**, **sh**, **trap**

echo — Write arguments to standard output

---

echo *argument...*

Purpose

echo writes its arguments, specified with the *argument* argument, to standard output. **echo** accepts these C-style escape sequences:

\a
> Bell (accepted but has no effect)

\b
> Backspace

\c
> Removes any following characters, including **\n** and **\r**.

\f
> Form feed

\n
> Newline

\r
> Carriage return

\t
> Horizontal tab

\v
> Vertical tab

\0*num*
> The byte with the numeric value specified by the zero to three-digit octal *num*.

****
> Backslash

echo follows the final argument with a newline unless it finds \c in the arguments. Arguments are subject to standard argument manipulation.

Examples

1. One important use of **echo** is to expand file names on the command line, as in:

   ```
   echo *.[ch]
   ```

   This displays the names of all files with names ending in **.c** or **.h**—typically C source and include (header) files. **echo** displays the names on a single line. If there are no file names in the working directory that end in **.c** or **.h**, **echo** simply displays the string ***.[ch]**.

2. **echo** is also convenient for passing small amounts of input to a filter or a file:

   ```
   echo 'this is\nreal handy' > testfile
   ```

Usage Notes

echo is provided as both an external utility and as a shell built-in.

Localization

echo uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

echo always returns the following exit status value:

0
    Successful completion

Portability

POSIX.2, X/Open Portability Guide, UNIX system V.

The POSIX.2 standard does not include escape sequences, so a strictly conforming application cannot use them. **printf** is suggested as a replacement.

Related Commands

sh

ed — Use the ed line-oriented text editor

```
ed [−bsx] [−p prompt] [file]
```

Purpose

ed is a text editor that lets you manipulate text files interactively. **ed** reads the text of a file into memory and stores it in an area called a *buffer*. Various subcommands let you edit the text in the buffer. Finally, you can write the contents of the buffer back out to the file, thereby overwriting the old contents of the file.

red is a restricted version of **ed**. It is intended to *protect* the novice user by disallowing the ! command and the ability to access files found anywhere but the working directory.

Options

ed supports the following options:

−b
> Lets you edit larger files by restricting the amount of memory dedicated to paging. This frequently makes **ed** run slower.

−p *prompt*
> Displays the given *prompt* string prompting you to input a subcommand. By default, **ed** does not usually prompt for subcommand input. See the description of the **P** subcommand for more on subcommand prompting ("Subcommands" on page 110).

−s
> Puts **ed** into a *quiet* mode, in which **e**, **E**, **r**, and **w**, subcommands do not display file size counts; the **q** and **e** subcommands do not check buffer modification; and ! is not displayed after calling the shell to run a subcommand. This mode is particularly useful when you invoke **ed** from within a shell script.

−x
> Runs an **X** subcommand to handle encrypted files properly. See the description of the **X** subcommand for more details ("Subcommands" on page 110).

If the optional *file* argument is present on the command line, **ed** reads the specified *file* into the editor by simulating an **e** *file* subcommand.

Addresses

You can prefix subcommands in **ed** with zero, one, or two addresses. These addresses let you refer to single lines or ranges of lines in the buffer. You do not need to specify addresses for certain subcommands that use default addresses. Consult the description for a particular subcommand. You can construct each address out of the following components:

.
> The single *dot* character represents the *current line* in the buffer. Many subcommands set the *current line*; for example the **e** command sets it to the last line of the new file being edited.

$
> This is a shorthand notation for the last line in the buffer.

n
> The number *n* refers to the *n*th line in the buffer.

/regexp/
> This searches for a line containing a string that matches the regular expression, `regexp` (for information on regular expressions, see Appendix B, "Regular Expressions (regexp)," on page 471). The search begins at the line immediately following the current line. It proceeds *forward* through the

buffer; if **ed** reaches the end of the buffer without finding a match, it wraps around to the first line of the buffer and continues the search. If **ed** does not find a match, the search ends when it reaches the original current line. If it does find a match, the address */regexp/* refers to the first matching line. If you omit *regexp*, the last used regular expression becomes the object of the search. You can omit the trailing */*. Within *regexp*, \/ represents a literal slash and not the *regexp* delimiter.

?*regexp*?
> This is similar to the previous address form, except that the search goes *backward* through the buffer. If the search reaches the first line in the buffer without finding a match, **ed** wraps around and continues searching backward from the last line in the buffer. If you omit *regexp*, the last used regular expression becomes the object of the search. You can omit the trailing *?*. Within *regexp*, \? represents a literal question mark and not the *regexp* delimiter.

'*l*
> The address is the line marked with the mark name *l*. The name *l* must be a lowercase letter set by the **k** subcommand.

You can combine these basic addresses with numbers using the **+** and **−** operators, with the usual interpretation. Missing left operands default to . (dot); missing right operands default to 1. Missing right operands also have a cumulative effect; so an address of − − refers to the current line number less two.

You can specify address ranges in the following ways:

a1,a2
> Specifies a range of addresses from address *a1* to address *a2*, inclusive. If you omit *a1* and *a2* (that is, the comma alone is specified), this is equivalent to the range 1,$.

a1;a2
> Is similar to the previous form except that **ed** resets the current line after calculating the first address, *a1*, so that the second address, *a2*, is relative to *a1*. If you omit *a1* and *a2* (that is, the semicolon alone is specified), this is equivalent to .;$. If you specify only *a1* and the command requires both *a1* and *a2*, the command operates as though you specified a range of:

```
a1;. command
```

>
> Is equivalent to .,.+22 (that is, page forward), except that it never attempts to address any line beyond $.

<
> Is equivalent to .−22,. (that is, page backward), except that it never addresses any line before line 1.

Subcommands

An **ed** command has the form [address] *command*.

All commands end with a newline; you must press <Enter>. Most commands allow only one command on a line, although you can modify commands by appending the **ln**, **n**, and **p** commands.

Subcommands generally take a maximum of zero, one, or two addresses, depending upon the particular subcommand. In the following descriptions, we show commands with their default addresses (that is, the addresses used when you don't specify any addresses) in a form that shows the maximum number of permitted addresses for the command. In any of the subcommands that take a *file* argument, *file* can be a path name or:

```
!command-line
```

If you use the **!** form, **ed** runs the given command line, reading its standard output or writing its standard input, depending on whether the **ed** command does reading or writing.

ed accepts the following subcommands:

.a
> Appends text *after* the specified line. Valid addresses range from 0 (text is placed after the last line of the buffer, before the first line) to $ (text is placed after the last line of the buffer). **ed** reads lines of

text from the workstation until a line consisting solely of an unescaped **.** (dot) is entered. **ed** sets the current-line indicator to the last line appended.

.,.c

Changes the addressed range of lines by deleting the lines and then reading new text in the manner of the **a** or **i** subcommands.

.,.d

Deletes the addressed range of lines. The line after the last line deleted becomes the new current line. If you delete the last line of the buffer, **ed** sets the current line to the new last line. If no lines remain in the buffer, it sets the current line to 0.

E[*file*]

Is similar to the **e** command, but **ed** gives no warning if you have changed the buffer.

e [*file*]

Replaces the contents of the current buffer with the contents of *file*. If you did not specify *file*, **ed** uses the *remembered* file name, if any. In all cases, the **e** subcommand sets the *remembered* file name to the file that it has just read into the buffer. **ed** displays a count of the bytes in the file unless it is in *quiet* mode. If you have changed the current buffer since the last time its contents were written, **ed** warns you if you try to run an **e** subcommand, and does not run the subcommand. If you enter the **e** subcommand a second time, **ed** goes ahead and runs the command.

f [*file*]

Changes the *remembered* file name to *file*. **ed** displays the new *remembered* file name. If you do not specify *file*, **ed** displays the current *remembered* file name.

1,$G/*regexp*/

Is similar to the **g** command except that when **ed** finds a line that matches *regexp*, it prints the line and waits for you to type in the subcommand to be run. You cannot use the **a**, **c**, **i**, **g**, **G**, **v**, and **V** subcommands. If you enter &, the **G** subcommand reruns the last subcommand you typed in. If you just press <Enter>, **G** does not run any subcommand for that line.

1,$g/*regexp*/*command*

Performs *command* on all lines that contain strings matching the regular expression *regexp*. This subcommand works in two passes. In the first pass, **ed** searches the given range of lines and marks all those that contain strings matching the regular expression *regexp*. The second pass actually performs *command* on those lines. You cannot use **!**, **g**, **G**, **V**, or **v** as *command*. *command* consists of one or more **ed** subcommands, the first of which must appear on the same line as the **g** subcommand. All lines of a multiline command list, except the last, must end with a backslash (\). If *command* is empty, **ed** assumes it to be the **p** subcommand. If no lines match *regexp*, **ed** does not change the current line number; otherwise, the current line number is the one set by the last subcommand in *command*. Instead of the slash (/) to delimit *regexp*, you can use any character other than space or newline.

H

Tells **ed** to display more descriptive messages when errors occur. If **ed** is already printing descriptive messages, **H** returns to terse error messages. Normally, **ed** indicates error messages by displaying a ?. When you turn on descriptive error messages with this subcommand, **ed** also displays the descriptive message for the most recent ? message.

h

Provides a brief explanation of the last error that occurred. This does not change the current line number.

.i

Works similarly to the **a** subcommand, except that **ed** places the text *before* the addressed line. Valid addresses range from line 1 to $ (the last line). **ed** sets the current line number to the last inserted line.

.,.+1j

Joins a range of lines into one line. To be precise, the **j** command removes all newline characters from the addressed range of lines, except for the last one. **ed** sets the current line number to the resulting combined line.

ed

.kl

Marks the addressed line with the mark name **l**, which is a single lowercase letter of the alphabet. This lets you refer to a marked line with the construct **'l**. This is called an *absolute address,* because it always refers to the same line, regardless of changes to the buffer.

.,.l

Displays the addressed range of lines, representing nonprintable (control) characters in a visible manner. **ed** sets the current line to the last line so displayed. You can append this subcommand to most other commands, to check on the effect of those subcommands.

.,.m*a*

Moves the addressed lines to the point immediately following the line given by the address *a*. The address *a* must not be in the range of addressed lines. If address *a* is 0, **ed** moves the lines to the beginning of the buffer. The last line moved becomes the new current line.

.,.n

Displays the addressed lines in a way similar to the **p** command, but **ed** puts the line number and a tab character at the beginning of each line. The last line displayed becomes the new current line. You can append **n** to any subcommand (except for **E**, **e**, **f**, **Q**, **r**, **w**, or **!**) so that you can check on the effect that the subcommands had.

P

Turns on subcommand prompting if it is not already on. If you specified the **–p** *prompt* option on the **ed** command line, **ed** displays the *prompt* string whenever it is ready for you to type in another subcommand. If you did not include the **–p** option, **ed** uses the * character as a prompt. If subcommand prompting is currently turned on, issuing the **P** subcommand turns it off.

.,.p

Displays (prints) the addressed lines. The last line displayed becomes the new current line. You can append **p** to most subcommands, so that you can check on the effect that the subcommands had.

You can append **p** to any subcommand (except for **E**, **e**, **f**, **Q**, **r**, **w**, or **!**) so that you can check on the effect that the subcommands had.

Q

Quits unconditionally, without checking for buffer changes.

q

Causes the editor to exit. If you have made changes to the buffer since the last save and you try to quit, **ed** issues a warning. Entering the **q** subcommand again lets you quit, regardless of unsaved changes.

$r [*file*]

Reads the contents of the *file* into the buffer after the addressed line. If you do not specify *file*, **ed** uses the *remembered* file name; if no *remembered* file name exists, *file* becomes the new *remembered* name. The **r** subcommand displays the number of bytes read from *file* unless you specified the **–s** option. The last line read from the file becomes the new current line. If *file* is replaced by **!**, the rest of the line is considered a shell command line, the output of which is to be read.

.,.s/*regexp*/*new*/[*flags*]

Searches the specified range of lines for strings matching the regular expression *regexp*. Normally the **s** subcommand replaces the first such matching string in each line with the string *new*. The **s** subcommand sets the current line to the last line on which a substitution occurred. If **ed** makes no such replacements, **ed** considers it an error.

flags can be one of the following:

n

Replaces the *n*th matching string in the line instead of the first one.

g

Replaces *every* matching string in each line, not just the first one.

l

Displays the new current line in the format of the **l** subcommand.

n

Displays the new current line in the format of the **n** subcommand.

p

Displays the new current line in the format of the **p** subcommand.

You can use any single printable character other than space or newline instead of / to separate parts of the subcommand provided that you use the same character to delimit all parts of the subcommand. You can omit the trailing delimiter.

You can include a newline in the *new* string by putting a \ immediately in front of the newline. This is a good way to split a line into two lines. If *new* consists only of the % character, **s** uses the *new* string from the previous **s** command. If & appears anywhere in *new*, **ed** replaces it with the text matching the *regexp*. If you want *new* to contain a literal ampersand, or percent sign, put a backslash (\) in front of the & or % character.

.,.t*a*

Copies the addressed lines to the point *after* the line given by the address *a*. The address *a* must not fall in the range of addressed lines. If address *a* is 0, **ed** copies the lines to the beginning of the buffer. This sets the current line to the last line copied.

u

Rolls back the effect of the last subcommand that changed the buffer. For the purposes of **u**, subcommands that change the buffer are: **a**, **c**, **d**, **g**, **G**, **i**, **j**, **m**, **r**, **s**, **t**, **v**, **V**, and (of course) **u**. This means that typing **u** repeatedly switches the most recent change back and forth. This subcommand sets the current line number to the value it had immediately before the subcommand being undone started.

1,$V/*regexp*/

Is similar to the **G** subcommand, except that this subcommand gives you the chance to edit only those lines that do *not* match the given regular expression.

1,$v/*regexp*/*commands*

Is similar to the **g** (global) command, except that **ed** applies the given *commands* only to lines that do *not* match the given regular expression.

1,$W [file]

Is similar to the **w** subcommand, except that this command appends data to the given *file* if the file already exists.

1,$w [*file*]

Writes the addressed lines of the buffer to the named *file*. This does not change the current line. If you do not provide *file*, **ed** uses the *remembered* file name; if there is no *remembered* file name, *file* becomes the *remembered* name. If the output file does not exist, **ed** creates it. **ed** displays the number of characters written unless you had specified the **–s** option.

X

Prompts you to enter an *encryption key*. All subsequent **e**, **r**, and **w** subcommands use this key to decrypt or encrypt text read from or written to files. To turn encryption off, issue an **X** subcommand and press <Return> in response to the prompt for an encryption key.

!*command*

Runs *command* as if you typed it to your chosen command interpreter. If *command* contains the % character, **ed** replaces it with the current *remembered* file name. If you want a subcommand to contain a literal %, put a backslash (\) in front of the character. As a special case, typing **!!** reruns the previous *command*.

$=

Displays the line number of the addressed line. This does not change the current line.

.+1.,.+1

If you supply zero, one, or two addresses without an explicit subcommand, **ed** displays the addressed lines in the mode of the last print subcommand: **p**, **l**, or **n**. This sets the current line number to the last line displayed.

Environment Variables

COLUMNS
Contains the terminal width in columns. **ed** folds lines at that point. If it is not set, **ed** uses the appropriate value from the **TERMINFO** database or if that is not available, it uses a default of 80.

HOME
Contains the path name of your home directory.

SHELL
Contains the full path name of the current shell.

TMPDIR
Is the path name of the directory being used for temporary files. If it is not set, the OpenExtensions shell uses **/tmp**.

Files

/tmp/e*
This is the *paging file*. It holds a copy of the file being edited. You can change the directory for temporary files using the environment variable **TMPDIR**.

ed.hup
ed writes the current buffer to this file when it receives a hangup signal.

Localization

ed uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**

See for more information.

Exit Values

Possible exit status values are:

0
Successful completion.

1
Failure because of any of the following:

- Addressed line out of range.
- Only one file name is allowed.
- No space for the line table.
- Temporary file error.
- Badly constructed regular expression.
- No remembered regular expression.
- File read error.
- Out of memory.
- Unknown command.
- Command suffix not permitted.
- No match found for regular expression.
- Wrong number of addresses for the subcommand.

- Not enough space after the subcommand.
- The name is too long.
- Badly formed name.
- Subcommand redirection is not permitted.
- Restricted shell.
- No remembered file name.
- The mark name must be lowercase.
- Undefined mark name.
- **m** and **t** subcommands require a destination address.
- The destination cannot straddle source in **m** and **t**.
- A subcommand not allowed inside **g**, **v**, **G**, or **V**.
- The **x** subcommand has become X (uppercase).
- The global command is too long.
- Write error (no disk space).

2
Usage error.

Messages and Return Codes

The error messages are issued only if **h** or **H** subcommands are used after **ed** outputs ?. Possible error messages include:

Destination cannot straddle source in m and t
The range of lines being moved or copied by **m** or **t** cannot include the destination address.

Global command too long
There is a limit on the length of a global instruction (**g** or **v**). See <u>"Limits" on page 116</u> for this limit.

'm' and 't' require destination address
You must follow the **m** or **t** subcommands with an address indicating where you want to move or copy text. You omitted this address.

No remembered file name
You tried to run a subcommand that used a remembered file name (for example, you used **w** to write without specifying an output file name). However, there is no remembered file name at present. Run the subcommand again, but specify a file name this time.

Restricted shell
The command line invoked the restricted form of **ed**, but you tried an action that was not allowed in the restricted editor (the **!** subcommand).

Temporary file error
You ran out of space on disk or encountered other errors involving the page file stored in the temporary file.

Warning: file not saved
You entered a subcommand to quit editing the current file, for example, **q** or **e** to edit a new file; however, you have changed the file since the last time you saved it. **ed** is suggesting that you save the file before you exit it; otherwise, your recent changes will be lost. To save the file, use the **w** command. If you really do not want to save the recent changes, use **q** to quit or **e** to edit a new file.

?file
An error occurred during an attempt to open or create *file*. This is applicable to the **e**, **r**, and **w** subcommands.

?
An unspecified error occurred. Use the **h** or **H** subcommand for more information. If the input to **ed** comes from a script rather than from a workstation, **ed** exits when any error occurs.

Limits

ed allows a limit of 1024 bytes per line and 28,000 lines per file. It does not allow the NUL ('\0') character. The maximum length of a global command is 256 characters, including newlines.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The addresses **<** and **>**, the **–b** and **–x** options, and the **W** and **X** subcommands are extensions of the POSIX standard.

Related Commands

awk, **diff**, **grep**, **sed**, **env**, **regexp** (see Appendix B, "Regular Expressions (regexp)," on page 471)

env — Display environments, or set an environment for a process

> **env [–i] [*variable=value* ...] [*command argument* ...]**
> **env [–] [*variable=value* ...] [*command argument* ...]**

Purpose

If you enter **env** with no arguments, it displays the environment that it received from its parent (presumably the shell).

Arguments of the form *variable=value* let you add new variables or change the value of existing variables of the environment.

If you specify *command*, **env** calls *command* with the arguments specified with the *argument* argument that appears on the command line, passing the accumulated environment to this command. The *command* is run directly as a program found in the search **PATH,** and is not interpreted by a shell.

Options

The **env** command recognizes the following two options, both of which have the same effect.

–i

Specifies that the environment inherited by **env** not be used.

–

Specifies that the environment inherited by **env** not be used.

Examples

Compare the output of the following two examples:

```
env foo=bar env
env -i foo=bar env
```

Environment Variables

PATH

Contains a list of directories to search when attempting to find *command*.

Localization

env uses the following localization environment variables:

• **LANG**
• **LC_ALL**
• **LC_CTYPE**
• **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

Successful completion.

1
> Failure due to any of the following:
>
> - Not enough memory
> - Name is too long

2
> Incorrect command-line argument.

126
> **env** found *command* but could not invoke it.

127
> **env** was unable to find *command*.

Messages and Return Codes

Possible error messages include:

Too many environment variables
> The maximum number of environment variables that can be specified in a single **env** command is 512.

Portability

POSIX.2, X/Open Portability Guide, UNIX system V.

printenv on Berkeley UNIX systems works like **env**.

Related Commands

sh

eval — Construct a command by concatenating arguments

---

eval [*argument ...*]

Purpose

The shell evaluates each argument as it would for any command. **eval** then concatenates the resulting strings, separated by spaces, and evaluates and executes this string in the current shell environment.

Examples

The command:

```
for a in 1 2 3
do
        eval x$a=fred
done
```

sets variables *x1*, *x2*, and *x3* to `fred`. Then:

```
echo $x1 $x2 $x3
```

produces:

```
fred fred fred
```

Usage Notes

This command is built into the shell.

Exit Values

The only possible exit status value is:

0
    You specified no arguments or the specified arguments were empty strings.

Otherwise, the exit status of **eval** is the exit status of the command that **eval** runs.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

exec, **sh**

exec — Run a command and open, close, or copy the file descriptors

> **exec** [*command_line*]

Purpose

The *command_line* argument for **exec** specifies a command line for another command. **exec** runs this command without creating a new process. Some people picture this action as *overlaying* the command on top of the currently running shell. Thus, when the command exits, control returns to the parent of the shell.

Input and output redirections are valid in *command_line*. You can change the input and output descriptors of the shell by giving only input and output redirections in the command. For example:

```
exec 2>errors
```

redirects the standard error stream to **errors** in all subsequent commands ran by the shell.

If you do not specify *command_line*, **exec** simply returns a successful exit status.

Usage Notes

This is a special built-in command of the shell.

Exit Values

If you specify *command_line*, **exec** does not return to the shell. Instead, the shell exits with the exit status of *command_line* or one of the following exit status values:

1–125
    A redirection error occurred.

126
    The command in *command_line* was found, but it was it was not an executable utility.

127
    The given *command_line* could not be run because the command could not be found in the current **PATH** environment.

If you did not specify *command_line*, **exec** returns with an exit value of zero.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

sh

exit — Return to the parent process from which the shell was called or to CMS

exit [*expression*]

Purpose

exit ends the shell.

The value of expression should be between 0 and 255. The **EXIT** trap is raised by the **exit** command, unless **exit** is being or called from inside an **EXIT** trap.

If you have a shell background job running, you cannot exit from the shell until it completes.

Exit Values

exit returns the value of the arithmetic expression specified by the *expression* argument to the parent process as the exit status of the shell. If you omit *expression*, **exit** returns the exit status of the last command run.

Portability

exit is a special built-in shell command.

Related Commands

The **exit()** ANSI C function, the _exit callable service, and the **_exit()** POSIX C function are unrelated to the **exit** shell command.

export — Set the export attributes for variables, or show currently exported variables

```
export [name [=value] ...]
export –p
```

Purpose

export marks each variable *name* so that the current shell makes it automatically available to the environment of all commands run from that shell. Exported variables are thus available in the environment to all subsequent commands. Several commands (for example, **cd** and **date**) look at environment variables for configuration or option information.

Variable assignments of the form *name=value* assign *value* to *name* as well as marking *name* for export.

Calling **export** without arguments lists, with appropriate quoting, the names and values of all variables in the format *Variable="value"*. If you reinput this format to another shell, variables are assigned appropriately but not exported. The **–p** option lists variables in a format suitable for reinput to the shell (see the description of the **–p** option).

Options

export recognizes the following option:

–p

Lists variables in the form:

```
export name="value"
```

suitable for reinput to the shell.

Usage Notes

This is a special built-in shell command.

Exit Values

Possible exit status values:

0

Successful completion

1

Failure due to incorrect command-line argument

2

Failure, usually due to incorrect an incorrect command-line argument, that results in a usage message

Portability

POXIS.2, X/Open Portability Guide.

Assigning a value to *name*, and the behavior given for calling **export** with arguments are extensions of the POSIX standard.

Related Commands

cd, **date**, **set**, **sh**, **typeset**

expr — Evaluate arguments as an expression

---

expr *expression*

Purpose

The set of arguments passed to **expr** constitutes an expression to be evaluated. Each command argument is a separate token of the expression. **expr** writes the result of the expression on the standard output. This command is primarily intended for arithmetic and string manipulation on shell variables.

Operators explained together have equal precedence; otherwise, they are in increasing order of precedence. **expr** stores an expression as a string and converts it to a number during the operation. If the context requires a Boolean value, a numeric value of 0 (zero) or a null string ("") is *false,* and any other value is *true.* Numbers have an optional leading sign, followed by either a hexadecimal, an octal, or a decimal number. The shell differentiates between hex, octal, and decimal as follows:

- Any number that starts with 0x is hex.
- Any number that starts with 0 is octal.
- Any number that does not start with 0x or 0 is decimal.

Numbers are manipulated as long integers.

expr1 | expr2
 Results in the value *expr1* if *expr1* is true; otherwise, it results in the value of *expr2*.

expr1 & expr2
 Results in the value of *expr1* if both expressions are true; otherwise, it results in 0.

expr1 <= expr2 | expr1 < expr2 | expr1 = expr2 | expr1 != expr2 | expr1 >= expr2 | expr1 > expr2
 If both *expr1* and *expr2* are numeric, **expr** compares them as numbers; otherwise, it compares them as strings. If the comparison is true, the expression results in 1; otherwise, it results in 0.

expr1 + expr2 | expr1 − expr2
 Performs addition or subtraction on the two expressions. If either expression is not a number, **expr** exits with an error.

*expr1 * expr2 | expr1 / expr2 | expr1 % expr2*
 Performs multiplication, division, or modulus on the two expressions. If either expression is not a number, **expr** exits with an error.

*expr1 : re | **match** expr1 re*
 matches the regular expression *re* against *expr1* treated as a string. The regular expression is the same as that accepted by **ed,** except that the match is always anchored—that is, there is an implied leading ^. Therefore, **expr** does not consider ^ to be a metacharacter. If the regular expression contains \(...\), **\)** and it matches at least part of *epr1,* **expr** results in only that part; if there is no match, **expr** results in 0. If the regular expression doesn't contain this construct, the result is the number of characters matched. The function **match** performs the same operation as the colon operator.

substr *expr1 expr2 expr3*
 Results in the substring of *expr1* starting at position *expr2* (origin 1) for the length of *expr3*.

index *expr1 expr2*
 Searches for any of the characters in *expr2* in *expr1* and results in the offset of any such character (origin 1), or 0 if no such characters are found.

length *expr1*
 Results in the length of *expr1*.

(*expr***)**
 Groups expressions.

Examples

```
fname=src/fn_abs.c
expr $fname : '.*_\(.*\)\.c'
```

returns abs.

```
a=`expr $a + 1`
```

adds 1 to the value of the shell variable *a*.

Localization

expr uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
> The result of *expression* is true.

1
> The result of *expression* is false.

2
> Failure due to any of following:
>
> - Not enough memory.
> - Command line syntax error.
> - Too few arguments on the command line.
> - Incorrect regular expression.
> - Regular expression is too complicated.
> - Nonnumeric value found where a number was expected.

Messages and Return Codes

Possible error messages include:

internal tree error
> Syntax errors or unusual expression complexity make it impossible for **expr** to evaluate an expression. If an expression has syntax errors, correct them; if not, simplify the expression (perhaps by breaking it into parts).

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

In the shell, **let** largely supersedes this command.

match, **substr**, **length**, and **index** are undocumented on all UNIX systems, though they do appear to exist there. They are extensions of the POSIX standard.

Related Commands

ed, **let**, **sh**, **test**, **regexp** (see Appendix B, "Regular Expressions (regexp)," on page 471)

false — Return a nonzero exit code

```
false [argument ...]
```

Purpose

The `false` command simply returns an exit status value of 1 (failure). This can be useful in shell scripts.

Usage Notes

This command is provided as both an external utility and a shell built-in.

Exit Values

`false` always returns an exit status value of 1.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

sh

fc, history, r -- Process a command history list

```
fc [-r] [-e editor] [first[last]]
fc -l [-nr] [first[last]]
fc -s [old=new] [specifier]
```

Purpose

fc displays, edits, and reenters commands that have been input to an interactive shell. **fc** stands for "fix commands." If the variable **HISTSIZE** is not defined, 128 commands are accessible. The number of commands that are accessible is determined by the **HISTSIZE** variable.

The shell stores these commands in a history file. When the **HISTFILE** environment variable is defined as the name of a writable file, the shell uses this as the history file. Otherwise, the history file is **$HOME /.sh_history**, if **HOME** is defined and the file is writable. If the **HOME** variable is not defined, or the file is not writable, the shell attempts to create a temporary file for the history. If a temporary file cannot be created, the shell does not keep a history file.

Note: A shell shares history (commands) with all shells that have the same history file. A login shell truncates the history file if it is more than **HISTSIZE** lines long.

Normally, the shell does not keep a history of commands run from a profile file or the **ENV** file. By default, however, it begins recording commands in the history file when it encounters a function definition in either of these setup files. This means that the **HISTSIZE** and **HISTFILE** variables must be set up appropriately before the first function definition. If you do not want the history file to begin at this time, use:

```
set -o nolog
```

For further information, see **sh** and **set**. Any variable assignment or redirection that appears on the **fc** command line affects both the **fc** command itself and the commands that **fc** produces.

The first form of the **fc** syntax puts you into an editor with a range of commands to edit. When you leave the editor, **fc** inputs the edited commands to the shell.

The first and last command in the range are specified with *first* and *last*. There are three ways to specify a command.

- If the command specifier is an unsigned or positive number, **fc** edits the command with that number.
- If the command specifier is a negative number –*n*, **fc** edits the command that came *n* commands before the current command.
- If the command specifier is a string, **fc** edits the most recent command beginning with that string.

The default value of *last* is *first*. If you specify neither *first* nor *last*, the default command range is the previous command entered to the shell.

Options

fc recognizes the following options:

–e *editor*
> Invokes *editor* to edit the commands. If you do not specify the **–e** option, **fc** assumes that the environment variable **FCEDIT**, if defined, contains the name of the editor for **fc** to use. If **FCEDIT** is not defined, **fc** invokes **ed** to edit the commands.
>
> **Note: ed** is the only supported interactive editor.

–l
> Simply displays the command list. This option does not edit or reenter the commands. If you omit *last* with this option, **fc** displays all commands from the one indicated by *first* through to the previous

command entered. If you omit both *first* and *last* with this option, the default command range is the 16 most recently entered commands.

–n

Suppresses command numbers when displaying commands.

–r

Reverses the order of the commands in the command range.

–s

Reenters exactly one command without going through an editor. If a command *specifier* is given, **fc** selects the command to reenter as described earlier; otherwise, **fc** uses the last command entered. To perform a simple substitution on the command before reentry, use a parameter of the form *old=new*. The string *new* replaces the first occurrence of string *old*. **fc** displays the (possibly modified) command before reentering it.

Environment Variables

FCEDIT

Contains the default editor to be used if none is specified with the **–e** option.

HISTFILE

Contains the path name of the history file.

HISTSIZE

Gives the maximum number of previous commands that are accessible.

Files

/tmp

Used to store temporary files. You can use the **TMPDIR** environment variable to dictate a different directory to store temporary files.

/.sh_history

This default history file is created.

Localization

fc uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

Usage Notes

This command is built into the shell. **r** is a built-in alias for **fc –s**. **history** is a built-in alias for **fc –l**.

Exit Values

Possible exit status values are:

0

If you specified **–l**, this indicates successful completion.

1

Failure due to any of the following:

- Missing history file
- Inability to find the desired line in the history file
- Inability to create temporary file

2

An incorrect command-line option or argument

If **fc** runs one or more commands, the exit status of **fc** is the exit status of the last run command.

Messages and Return Codes

Possible error messages include:

Cannot create temporary file

fc must create a temporary file to do some operations, such as editing. It prints this message when it cannot create its temporary file—for example, because the disk is full.

No command matches *string*

You asked to edit a command beginning with a particular *string*, but there was no such command in the history file.

Portability

POSIX.2.

Related Commands

alias, **ed**, **print**, **read**, **sh**

fg — Bring a job into the foreground

```
fg [%job-identifier]
```

Purpose

fg restarts a suspended job or moves a job from the background to the foreground. To identify the job, you give a *job-identifier* (preceded by %) as given by the jobs command.

If you do not specify *job-identifier,* fg uses the most recent job to be suspended (with the kill command) or placed in the background (with the bg command).

On POSIX, fg is available only if you have enabled job control.

Localization

fg uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
Successful completion

>0
No current job

Messages and Return Codes

Possible error messages include:

Not a stopped job
Job was not stopped.

Portability

POSIX.2 User Portability Extension.

Related Commands

bg, jobs, kill, ps

find — Find a file meeting specified criteria

find *path ... expression*

Purpose

find searches a given file hierarchy specified by *path*, finding files that match the criteria given by *expression*. Each directory, file, or special file encountered in the hierarchy is "passed through" *expression*, and if a match is found, then an action defined by *expression* occurs.

find builds *expression* from a set of *primaries* and *operators*. A *primary* does one of the following:

- Defines some trait to be matched, such as the audit mask of a file.
- Controls some other aspect of the behavior of **find**, such as how **find** traverses the file hierarchy or what **find** does when a match occurs.

An *operator* modifies how **find** interprets a *primary* or set of *primaries*. For example, an *operator* might invert the meaning of a *primary,* or an *operator* might be used to specify the logical OR of two *primaries*. The juxtaposition of two *primaries* is an implied *operator* in a way that it implies a logical AND of two *primaries*.

Operators

find recognizes the following operators:

–a
> Used between primaries for a logical AND. You can omit this operator to get the same result, since logical AND is assumed when no operator is used between two primaries.

–o
> Used between primaries for a logical OR.

!
> Precedes an expression in order to negate it.

You can group primaries and operators using parentheses. You must delimit all primaries, operators, numbers, arguments, and parentheses with white space. Each *number* noted in the primary list is a decimal number, optionally preceded by a plus or minus sign. If a number is given without a sign, **find** tests for equality; a plus sign implies "greater than" or "older than," and a minus sign implies "less than" or "newer than".

Primaries

find recognizes the following list of primaries for defining match criteria. Whenever *number* is used as a primary argument, it is interpreted as a decimal integer that is optionally preceded by a plus (+) or minus (-) sign as follows:

+*number*
> More than *number*

number
> Exactly *number*

-*number*
> Less than *number*.

Primary arguments:

–aaudit *auditmask*
> The **-aaudit** primary is used to match the auditor audit bits. See **–audit** *auditmask*.

–audit *auditmask*

The **-audit** primary is used to match the user audit bits. *auditmask* can be in octal or in symbolic form. The mask can be preceded by a **-** character (as in the **perm** primary), but it is ignored. Symbolic mode is an *operation=condition* list, separated by commas:

```
[rwx]=[sf]
```

where:

=sf

Success or failure on any of **rwx**

r=s

Success on **read**

r=s, x=sf

Success on **read** or **exec**, failure on **exec**

r, w=s

Incorrect

x

Incorrect

Note: Audit bits can be set by only the callable service BPX1CHA. See *z/VM: OpenExtensions Callable Services Reference* for more information.

–atime *number*

Matches if someone has accessed the file in the past *number* 24-hour periods.

–ctime *number*

Matches if someone has changed the attributes of the file in the past *number* 24-hour periods.

–group *name*

Matches if the group owner is *name*. If *name* is not a valid group name, it is treated as a group ID.

–inum *number*

Matches if the file has inode number *number*.

–links *number*

Matches if there are *number* links to the file.

–mtime *number*

Matches if someone has modified the file in the past *number* 24-hour periods.

–nogroup

Matches if no defined group owns the file.

–nouser

Matches if no defined user owns the file.

–perm[-]*mask*

By default, matches if the permissions on the file are identical to the ones given in *mask*. You can specify *mask* in octal or in symbolic mode (see **chmod**). If you use symbolic mode, **find** assumes that you begin with no bits set in *mask*, and that the symbolic mode is a recipe for turning the bits you want on and off. A leading minus sign (-) is special. It means that a file matches if at least all the bits in *mask* are set. As a result, with symbolic mode, you cannot use a *mask* value that begins with a minus sign (-).

If you use octal mode, **find** uses only the bottom 12 bits of *mask*. With an initial minus sign (-), **find** again matches only if at least all the limits in *mask* are set in the file permissions lists.

–size *number***[c]**

Matches if the size of the file is *number* blocks long, where a block is 512 bytes. If you include the suffix *c*, the file size is *number* bytes.

–type *c*

Matches if the type of the file is the same as the type given by the character *c*. Possible values of the character are:

b
   — Block special

c
   — Char-special

d
   — Directory

f
   — Regular file

l
   — Symbolic link

n
   — Network file

p
   — FIFO (named pipe)

s
   — Socket

–user *name*
   Matches if the owner of the file is *name. name* can also be a user ID number.

find recognizes the following primaries that control the actions taken when a match occurs:

–cpio *cpio-file*
   Writes the file found to the target file *cpio-file* in **cpio** format. This is equivalent to:

   ```
   find ... | cpio -o >cpio-file
   ```

   This primary matches if the command succeeds.

–exec *command* **;**
   Takes all arguments between **–exec** and the semicolon as a command line, replacing any argument that is exactly {} (that is, the two brace characters) with the current file name. It then executes the resulting command line, treating a return status of zero from this command as a successful match, nonzero as failure. You must delimit the terminal semicolon with white space.

   Note: The semicolon is a shell metacharacter. To use it in *expression*, you must quote it.

–name *pattern*
   Compares the current file name with *pattern*. If there is no match, the expression fails. The pattern uses the same syntax as file name generation (see **sh**). It matches as many trailing path name components as specified in *pattern*.

–ncpio *cpio-file*
   Writes the file found to the target file *cpio-file* in **cpio –c** format. This is equivalent to:

   ```
   find ... | cpio -oc >cpio-file
   ```

   This primary matches if the command succeeds.

–newer *file*
   Compares the modification date of the found file with that of the *file* given. This matches if someone has modified the found file more recently than *file*.

–none
   Indicates that some action has been taken; thus **find** does not invoke the default **–print** action.

–ok *command***;**
   Is similar to **–exec**, but before **find** executes the command, it displays the command to confirm that you want to go ahead. **find** executes the command line only if your input matches the expression for "yes" (yes and no expressions are defined in **LC_MESSAGES**). If you type the expression for "no", the primary does not match. You must delimit the terminal semicolon with white space.

   Note: The semicolon is a shell metacharacter. To use it in *expression*, you must quote it.

–print
　　Displays the current file name.

find recognizes the following primaries that control file hierarchy traversal:

–depth
　　Processes directories after their contents.

–follow
　　Follows symbolic and Mount External links.

–level *number*
　　Does not descend below *number* levels.

–prune
　　Stops searching deeper into the tree at this point. **–prune** has no effect if **–depth** is also specified.

–xdev
　　Does not cross device boundaries from the root of the tree search.

Examples

1. To find all files with a suffix of **.c** that have the audit mode set to **rwx** (read, write, execute), enter:

```
find / –name "*.c" –audit rwx=sf
```

2. To find all files with a suffix of **.c** and audit mode bits set to 777 (**rwx**), enter:

```
find / –name "*.c" –audit 777
```

Environment Variables

Path
　　Determines the location of the *command* specified with the **–exec** or **–ok** primaries.

Localization

find uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**

See for more information.

Exit Values

Possible exit status values are:

0
　　Successful completion

1
　　Failure due to any of the following:

- Not enough memory
- Missing option
- Incorrect character specified after **–type**
- Inability to get information on a file for **–newer**
- Incorrect permissions for **–perm**

- Inability to open a file for the **–cpio** option
- Unknown user or group name
- Inability to access the **PATH** variable
- Cannot run a command specified for **–exec** or **–ok**
- Syntax error
- Stack overflow caused by an expression that is too complex

2

Failure due to one of the following:

- Incorrect command-line option
- Not enough arguments on the command line
- Missing option
- Argument list that is not properly ended

Messages and Return Codes

Possible error messages include:

bad number specification in *string*
You specified an option that takes a numeric value (for example, **–atime**, **–ctime**) but did not specify a valid number after the option.

cannot stat file *name* **for -newer**
You used a **–newer** option to compare one file with another; however, **find** could not obtain a modification time for the specified file. Typically, this happens because the file does not exist or you do not have appropriate permissions to obtain this information.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Most UNIX systems do not have a default action of **–print**; hence, they do not need the **–none** option. The **–a** operator is undocumented on many UNIX systems. The **–aaudit**, **–audit**, **–cpio**, **–follow**, **–level**, **–ncpio**, and **–none** primaries are extensions of the POSIX standard. The **aaudit** and **audit** options are unique to the OpenExtensions shell.

Related Commands

chmod, **cpio**, **sh**

fold — Break lines into shorter lines

---
fold [–bs] [–w *width*] [–*width*] [*file...*]

Purpose

fold reads the standard input, or each *file*, if you specify any. Each input line is broken into lines no longer than *width* characters. If you do not specify *width* on the command line, the default line length is 80. The output is sent to the standard output.

Options

fold recognizes the following options:

–b
> Specifies *width* in bytes rather than in column positions; that is, **fold** does not interpret tab, backspace, and carriage return characters.

–s
> Breaks each line at the last blank within *width* column positions. If there is no blank that meets the requirement, **fold** breaks the line normally.

–w *width*
> Specifies a maximum line length of *width* characters.

–*width*
> is identical in effect to **–w** *width*.

Localization

fold uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
> Successful completion.

1
> Failure because the input file could not be opened.

2
> Invalid command-line option or a missing *width* argument.

Portability

POSIX.2, 4.2BSD

The **–*width*** option is an extension of the POSIX standard.

Related Commands

pr

getconf — Get configuration values

```
getconf [–a] system_var
getconf [–a] path_var pathname
```

Purpose

getconf writes the value of a configuration variable to the standard output. You can specify the configuration variable using one of the forms listed in the Format section. If you use the first form, **getconf** writes the value of the variable *system_var*. If you use the second form, **getconf** writes the value of the variable *path_var* for the path name given by *pathname*. The **–a** option prompts **getconf** to display all current configuration variables, and their values, to standard output.

getconf writes numeric values in decimal format and nonnumeric values as simple strings. If the value is undefined, **getconf** writes the string undefined to the standard output.

Options

getconf recognizes the following option:

–a
> Writes out all the configuration variables for the current system, and their values, to standard output. Path variables are written based on a path name of dot (**.**).

Configuration Variables

You can use the second form of **getconf** to find the value of the following POSIX.1-1990 standard configuration variables for the specified *path name*:

LINK_MAX
> Specifies the maximum number of links that this file can have.

MAX_CANON
> Specifies the maximum number of bytes in the workstation's canonical input queue (before line editing).

MAX_INPUT
> Specifies the space available in the workstation's input queue.

NAME_MAX
> Specifies the largest file name size.

PATH_MAX
> Specifies the maximum number of bytes in a path name.

PIPE_BUF
> Specifies the largest atomic write to a pipe.

_POSIX_CHOWN_RESTRICTED
> Specifies the restrictions that apply to file ownership changes.

_POSIX_NO_TRUNC
> If set, it is an error for any path name component to be longer than NAME_MAX bytes.

_POSIX_VDISABLE
> Specifies that processes are allowed to disable ending special characters.

You can use the first form of **getconf** to find the value of the following POSIX.1-1990 standard configuration variables:

ARG_MAX
Specifies the maximum length of arguments for running a program, including environment data.

CHILD_MAX
Specifies the maximum number of simultaneous processes allowed per real user.

CLK_TCK
Specifies the number of intervals per second in the machine clock.

NGROUPS_MAX
Specifies the number of simultaneous group IDs per process.

OPEN_MAX
Specifies the maximum number of open files at any time per process.

PATH
Specifies the standard **PATH** setting.

_CS_PATH
Specifies the standard **PATH** setting.

STREAM_MAX
Specifies the number of streams that one process can have open at one time.

TZNAME_MAX
Specifies the maximum number of bytes supported for the name of a time zone (not of the **TZ** variable).

_POSIX_ARG_MAX
Specifies the minimum conforming value for ARG_MAX.

_POSIX_CHILD_MAX
Specifies the minimum conforming value for CHILD_MAX.

_POSIX_JOB_CONTROL
Specifies the POSIX job control supported.

_POSIX_LINK_MAX
Specifies the minimum conforming value for LINK_MAX.

_POSIX_MAX_CANON
Specifies the minimum conforming value for MAX_CANON.

_POSIX_MAX_INPUT
Specifies the minimum conforming value for MAX_INPUT.

_POSIX_NAME_MAX
Specifies the minimum conforming value for NAME_MAX.

_POSIX_NGROUPS_MAX
Specifies the minimum conforming value for NGROUPS_MAX.

_POSIX_OPEN_MAX
Specifies the minimum conforming value for OPEN_MAX.

_POSIX_PATH_MAX
Specifies the minimum conforming value for PATH_MAX.

_POSIX_PIPE_BUF
Specifies the minimum conforming value for PIPE_BUF.

_POSIX_SAVED_IDS
Specifies that processes have saved set-user-ID and saved set-group-ID bits set.

_POSIX_SSIZE_MAX
Specifies the value that can be stored in an object of type *ssize_t*.

_POSIX_STREAM_MAX
Specifies the minimum conforming value for STREAM_MAX.

_POSIX_TZNAME_MAX
Specifies the minimum conforming value for TZNAME_MAX.

_POSIX_VERSION
Specifies the version of POSIX adhered to in this release.

You can use the first form of **getconf** to find the value of the POSIX.2 standard configuration variables:

BC_BASE_MAX
Specifies the maximum *ibase* and *obase* values for the **bc** command.

BC_DIM_MAX
Specifies the maximum number of elements permitted in a **bc** array.

BC_SCALE_MAX
Specifies the maximum *scale* size allowed in **bc**.

BC_STRING_MAX
Specifies the maximum number of characters in a string in **bc**.

COLL_WEIGHTS_MAX
Specifies the maximum number of weights assignable to an entry of the **LC_COLLATE order** keyword.

EXPR_NEST_MAX
Specifies the maximum number of expressions that you can nest inside parentheses in an expression evaluated by **expr**.

LINE_MAX
Specifies the maximum number of bytes that a utility can accept as an input line (either from the standard input or a text file) when the utility takes text files as input. This number includes the trailing <newline>.

RE_DUP_MAX
Specifies the maximum number of repeated occurrences of a regular expression when using the interval notation $\backslash\{m,n\backslash\}$ (see Appendix B, "Regular Expressions (regexp)," on page 471).

POSIX2_C_BIND
Indicates if the system supports the C Language Bindings Option.

POSIX2_C_DEV
Indicates if the system supports the C Language Development Utilities Option.

POSIX2_FORT_DEV
Indicates if the system supports the FORTRAN Development Utilities Option.

POSIX2_FORT_RUN
Indicates if the system supports the FORTRAN Runtime Utilities Option.

POSIX2_LOCALEDEF
Indicates if the system supports the creation of locales.

POSIX2_SW_DEV
Indicates if the system supports the Software Development Utilities Option.

POSIX2_CHAR_TERM
Indicates if the system supports at least one terminal type capable of all operations necessary for the User Portability Utilities Option. This parameter name is correct only on if POSIX2_UPE is on.

POSIX2_UPE
Indicates if the system supports the User Portability Utilities Option.

POSIX2_VERSION
Specifies the version of POSIX.2 adhered to in this release.

POSIX2_BC_BASE_MAX
Specifies the minimum conforming value for BC_BASE_MAX.

POSIX2_BC_DIM_MAX
Specifies the minimum conforming value for BC_DIM_MAX.

POSIX2_BC_SCALE_MAX
Specifies the minimum conforming value for BC_SCALE_MAX.

POSIX2_BC_STRING_MAX
Specifies the minimum conforming value for BC_STRING_MAX.

POSIX2_COLL_WEIGHTS_MAX
> Specifies the minimum conforming value for EQUIV_CLASS_MAX.

POSIX2_EXPR_NEST_MAX
> Specifies the minimum conforming value for EXPR_NEST_MAX.

POSIX2_LINE_MAX
> Specifies the minimum conforming value for LINE_MAX.

POSIX2_RE_DUP_MAX
> Specifies the minimum conforming value for RE_DUP_MAX.

This implementation of **getconf** also recognizes the following non-POSIX-conforming name:

_CS_SHELL
> Specifies the default shell (command interpreter).

Examples

```
getconf OPEN_MAX
getconf NAME_MAX /dir
```

Localization

getconf uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Usage Notes

1. The **–a** option does not display values for MAX_CANON, MAX_INPUT, and POSIX_VDISABLE path variables. This is because they are terminal file variables, and are not based on a path name of dot (.). The second form of the **getconf** command should be used to display the values of these variables.

Exit Values

Possible exit status values are:

0
> The specified *parameter_name* was valid and **getconf** displayed its value successfully.

>0
> An error occurred.

Portability

POSIX.2.

_CS_SHELL is an extension of the POSIX standard. Some symbols are supported only on systems that support POSIX.2.

Related Commands

bc, **expr**, **sh**, **regexp** (see Appendix B, "Regular Expressions (regexp)," on page 471)

getopts — Parse utility options

getopts *opstring name* [*arg ...*]

Purpose

getopts obtains options and their arguments from a list of parameters that follows the standard POSIX.2 option syntax (that is, single letters preceded by a hyphen (—) and possibly followed by an argument value). Typically, shell scripts use **getopts** to parse arguments passed to them. When you specify arguments with the *arg* argument on the **getopts** command line, **getopts** parses those arguments instead of the script command line (see **set**).

The *opstring* argument gives all the option letters that the script recognizes. For example, if the script recognizes **–a**, **–f**, and **–s**, *opstring* is afs. If you want an option letter to be followed by an argument value or group of values, put a colon after the letter, as in a:fs. This indicates that getopts expects the **–a** option to have the form **–a** *value*. Normally one or more blanks separate *value* from the option letter; however, **getopts** also handles values that follow the letter immediately, as in **–a***value*. *opstring* cannot contain a question mark (?) character.

name on the **getopts** command line is the name of a shell variable. Each time you invoke **getopts**, it obtains the next option from the positional parameters and places the option letter in the shell variable *name*.

getopts places a question mark (?) in *name* if it finds an option that does not appear in *opstring*, or if an option *value* is missing.

Each option on the script command line has a numeric *index*. The first option found has an index of 1, the second has an index of 2, and so on. When **getopts** obtains an option from the script command line, it stores the index of the script in the shell variable **OPTIND**.

When an option letter has a following argument (indicated with a : in *opstring*), **getopts** stores the argument as a string in the shell variable **OPTARG**. If an option doesn't take an argument, or if **getopts** expects an argument but doesn't find one, **getopts** unsets **OPTARG**.

When **getopts** reaches the end of the options, it exits with a status value of 1. It also sets *name* to the character ? and sets **OPTIND** to the index of the first argument after the options. **getopts** recognizes the end of the options by any of the following:

- Finding an argument that doesn't start with **–**
- Finding the special argument **––**, marking the end of options
- Encountering an error (for example, an unrecognized option letter)

OPTIND and **OPTARG** are local to the shell script. If you want to export them, you must do so explicitly. If the script invoking **getopts** sets **OPTIND** to 1, it can call **getopts** again with a new set of parameters, either the current positional parameters or new *arg* values.

By default, **getopts** issues an error message if it finds an unrecognized option or some other error. If you do not want such messages printed, specify a colon as the first character in *opstring*.

Examples

Following is an example of using getopts in a shell script:

```
# Example illustrating use of getopts builtin. This
# shell script would implement the paste command,
# using getopts to process options, if the underlying
# functionality was embedded in hypothetical utilities
# hpaste and vpaste, which perform horizontal and
# vertical pasting respectively.
#
```

```
paste=vpaste     # default is vertical pasting
seplist=" " # default separator is tab

while getopts d:s o
do      case "$o" in
        d)      seplist="$OPTARG";;
        s)      paste=hpaste;;
        [?])  print >&2 "Usage: $0 [-s] [-d seplist] file ..."
              exit 1;;
        esac
done
shift $OPTIND-1

# perform actual paste command
$paste -d "$seplist" "$@"
```

Environment Variables

getopts uses the following environment variables:

OPTARG
  Contains the value of the option argument found by **getopts**.

OPTIND
  Contains the index of the next argument to be processed.

Localization

getopts uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Usage Notes

This command is a built-in shell command.

Exit Values

Possible exit status values are:

0
  getopts found a script command line with the form of an option. This happens whether or not it recognizes the option.

1
  getopts reached the end of the options, or an error occurred.

2
  Failure because of an incorrect command-line option.

Portability

On UNIX systems, **getopts** is built into both the KornShell and Bourne shell.

Related Commands

sh

grep — Search a file for a specified pattern

grep [–bcEFilnqsvx] [–e *pattern*]... [–f *patternfile*]... [*pattern*] [*file ...*]

Purpose

grep –F searches files for one or more *pattern* arguments. It does not use regular expressions; instead, it does direct string comparison to find matching lines of text in the input. **grep** uses standard string search functions. The search stops after a null character is encountered. **grep** should not be used on lines that contain embedded null characters.

grep –E works similarly, but uses *extended* regular expression matching. This is described in Appendix B, "Regular Expressions (regexp)," on page 471. If you include special characters in patterns typed on the command line, escape them by enclosing them in single quotation marks to prevent inadvertent misinterpretation by the shell or command interpreter. To match a character that is special to **grep –E**, put a backslash (\) in front of the character. It is usually simpler to use **grep –F** when you don't need special pattern matching.

grep combines the functions of the UNIX commands **egrep** and **fgrep**. If you do not specify either **–E** or **–F**, **grep** behaves like **grep –E** but matches *basic* regular expressions instead of extended ones.

You can specify a pattern to search for with either the **–e** or **–f** option. If you specify neither option, **grep** takes the first nonoption argument as the pattern for which to search. If **grep** finds a line that matches a pattern, it displays the entire line. If you specify multiple input files, the name of the current file precedes each output line.

Options

grep accepts all of the following options:

–b
> Precedes each matched line with its file block number.

–c
> Displays only a count of the number of matched lines and not the lines themselves.

–E
> Matches using extended regular expressions.

–e *pattern*
> Specifies one or more patterns separated by newlines for which **grep** is to search.
>
> You can indicate each pattern with a separate **–e** option character, or with newlines within pattern. For example, the following two commands are equivalent:
>
> ```
> grep –e pattern_one –e  pattern_two file
> grep –e 'pattern_one pattern_two' file
> ```

–F
> Matches using fixed strings.

–f *patternfile*
> Reads one or more patterns from *patternfile*. Patterns in *patternfile* are separated by newlines.

–i
> Ignores the case of the strings being matched.

–l
> Lists only the file names that contain the matching lines.

–n
> Precedes each matched line with its fileline number.

−q

    Suppresses output and simply returns appropriate return code.

−s

    Suppresses the display of any error messages for nonexistent or unreadable files.

−v

    Complements the sense of the match—that is, displays all lines *not* matching a pattern.

−x

    Requires a string to match an entire line.

Examples

To display every line mentioning an astrological element:

```
grep -E "earth|air|fire|water" astro.log
```

Localization

grep uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

    The command found at least one match for *pattern*.

1

    The command found no matches for *pattern*.

2

    Failure due to any of the following:

    - **−e** option was missing *pattern*.
    - **−f** option was missing *patternfile*.
    - Out of memory for input or to hold a pattern.
    - *patternfile* could not be opened.
    - Incorrect regular expression.
    - Incorrect command-line option.
    - The command line had too few arguments.
    - The input file could not be opened.

    If the program fails to open one input file, it tries to go on to look at any remaining input files, but it returns 2 even if it succeeds in finding matches in other input files.

Messages and Return Codes

Possible error messages include:

input lines truncated—result questionable
  One or more input lines were longer than **grep** could handle; the line has been truncated or split into two lines, if possible. This message does not affect the exit status.

out of space for pattern *string*
  grep did not have enough memory available to store the code needed to work with the given pattern (regular expression). The usual cause is that the pattern is very complex. Make the pattern simpler, or try to release memory so that **grep** has more space to work with.

Limits

The longest input record (line) is restricted by the system variable LINE_MAX. It is always at least 2048 bytes. Longer lines are treated as two or more records.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–b** option is an extension of the POSIX standard.

Related Commands

ed, **find**, **regexp** (see)

head — Display the first part of a file

```
head [−bcklmn num] [file ...]
head  [−num]  [file ...]
```

Purpose

By default, **head** displays the first 10 lines of each file given on the command line. If you do not specify *file*, **head** reads the standard input.

Options

head recognizes the following options:

−b *num*
> Displays the first *num* blocks (a block is 512 bytes) of each file.

−c *num*
> Displays the first *num* bytes of each file.

−k *num*
> Displays the first *num* kilobytes (1024 bytes) of each file.

−l *num*
> Displays the first *num* lines of each file.

−m *num*
> Displays the first *num* megabytes of each file.

−n *num*
> Displays the first *num* lines of each file.

−*num*
> Displays the first *num* lines of each file.

Localization

head uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
> Successful completion

1
> Failure due to any of the following:
>
> - Inability to open an input file
> - Read error on the standard input
> - Write error on the standard output

2
> Failure due to any of the following:
>
> - Unknown command-line option
> - Missing or incorrect *num* in an **–n** option

Messages and Return Codes

Possible error messages include:

Badly formed line or character count *num*
> The value *num*, following a **–b**, **–c**, **–k**, **–l**, **–m**, or **–n** option, was not a valid number.

Portability

POSIX.2, X/Open Portability Guide.

This program originated with Berkeley Software Distribution (BSD) and is a frequent add-on to UNIX systems.

The POSIX.2 standard includes only the **–n** *num* and **–***num* options, though it considers the latter obsolete.

Related Commands

cat, **sed**, **tail**

iconv — Convert characters from one code set to another

```
iconv [–sc] –f oldset –t newset [file ...]
iconv –l[–v]
```

Purpose

iconv converts characters in *file* (or from standard input if no *file* is specified) from one code page set to another. The converted text is written to standard output. The code sets supported are system-dependent; check the documentation for your system's **iconv()** function. See the C/C++ documentation for more information about the code sets supported for this command.

If the input contains a character that is not valid in the source code set, **iconv** replaces it with the byte 0xff and continues, unless the **–c** option is specified.

If the input contains a character that is not valid in the destination code set, behavior depends on the system's **iconv()** function.

Options

iconv recognizes the following options:

–c
> Characters containing conversion errors are not written to the output. By default, characters not in the source character set are converted to the value 0xff and written to the output.

–f *oldset*
> Specifies the current code set of the input.

–l
> Lists code sets in the internal table.

–s
> Suppresses message that would be issued in the situation when exit value 2 is returned.

–t *newset*
> Specifies the destination code set for the output.

–v
> Specifies verbose output.

Localization

iconv uses the following localization environment variable:

• **LC_CTYPE**

See Appendix C, "Localization," on page 477 for more information.

Examples

1. To convert the file **words.txt** from the IBM-1047 standard code set to the ISO 8859-1:1987 standard code set and store it in **converted**:

```
iconv -f IBM-1047 -t ISO8859-1 words.txt > converted
```

Exit Values

Possible exit status values are:

0

Successful completion.

1

Failure because of any of the following:

- Insufficient memory
- Inability to open the input file
- Incorrect or unknown option

2

Input contained a character sequence that is not permitted in the source code set.

Portability

X/Open Portability Guide 4.0.

–v is an extension to the POSIX.2 standard. The **–c**, **–l**, and **–s** options are extensions to the XPG standard.

id — Return the user identity

```
id [user]
id −G [−n] [user]
id −g [−nr] [user]
id −u [−nr] [user]
```

Purpose

Entering **id** without arguments displays the user name and group affiliations of the invoking process that enters the command. Specifying a *user* argument on the command line displays the same information for the given user instead of the person invoking **id**. In this case, you require appropriate permissions.

The output has the format:

```
uid=runum(username) gid=rgnum(groupname)
```

where *runum* is the user's real user ID (UID) number, *username* is the user's real user name, *rgnum* is the user's real group ID (GID) number, and *groupname* is the user's real group name.

A user's real and effective IDs may differ. In this case, there may be separate entries for effective user ID (UID) with the format:

```
euid=eunum(euname)
```

where *eunum* is the effective user ID number and *euname* is the effective user name. An entry for effective group ID has the format:

```
egid=egnum(egname)
```

where *egnum* is the effective group ID number and *egname* is the effective group name.

Options

id recognizes the following options:

−G
   Displays all different group IDs (effective, real, and supplementary) as numbers separated by spaces.

−g
   Displays only the effective group ID number.

−n
   With **−G**, **−g**, or **−u**, displays the name rather than the number.

−r
   With **−g** or **−u**, displays the real ID rather than the effective one.

−u
   Displays only the effective user ID number.

Localization

id uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

- **LC_NUMERIC**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
    Successful completion

1
    You specified an incorrect user with the −*u* option.

2
    Failure due to an incorrect command-line argument, or the wrong number of command-line arguments.

Portability

POSIX.2, X/Open Portability Guide, UNIX system V.

Related Commands

logname

jobs — Return the status of jobs in the current session

---

jobs [−l|**−p**] [*job-identifier...*]

Purpose

jobs produces a list of the processes in the current session. Each such process is numbered for easy identification by **fg** or **kill**, and is described by a line of information:

```
[job-identifier]    default    state    shell_command
```

job-identifier
> Is a decimal number that identifies the process for such commands as **fg** and **kill** (preface *job-identifier* with **%** when used with these commands).

default
> Identifies the process that would be the default for the **fg** and **bg** commands (that is, the most recently suspended process). If *default* is a **+**, this process is the default job. If *default* is a **−**, this job becomes the default when the current default job exits. There is at most one **+** job and one **−** job.

state
> Shows a job as:

> **Running**
>> If it is not suspended and has not exited

> **Done**
>> If it exited successfully

> **Done(exit status)**
>> If it exited with a non-zero exit status

> **Stopped (***signal***)**
>> If it is suspended; *signal* is the signal that suspended the job

shell_command
> Is the associated shell command that created the process.

Options

jobs recognizes the following options:

−l
> Displays the process group ID of a job (before *state*).

−p
> Displays the process IDs of all processes.

The **−l** and **−p** options are mutually exclusive.

Localization

jobs uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
> Successful completion.

2
> Failure due to an incorrect command-line argument.

Portability

POSIX.2 User Portability Extension.

join — Join two sorted, textual relational databases

---

join [-a *n*] [-e *string*] [-o *list*] [-t *c*] [-v *n*] [-1 *n*] [-2 *n*] *file1 file2*
join [-a *n*] [-e *string*] [-j[*n*] *m*] [-o *list*] [-t *c*] *file1 file2*

Purpose

join joins two databases. It assumes that both *file1* and *file2* contain textual databases in which each input line is a record and that the input records are sorted in ascending order on a particular join key field (by default the first field in each file). If you specify **–** in place of *file1* or *file2*, **join** uses the standard input for that file. If you specify **– –** in place of both *file1* and *file2*, the output is undefined.

Conceptually, **join** computes the Cartesian product of records from both files. By default, spaces or tabs separate input fields and **join** discards any leading or trailing white space. (There can be no white-space-delimited empty input fields.) It then generates output for those combined records in which the join key field (the first field by default) matches in each file. The default output for **join** is the common join key field, followed by all the other fields in *file1*, and then all the other fields in *file2*. The other fields from each file appear in the same order they appeared in the original file. The default output field separator is a space character.

Options

Options to **join** are as follows:

–a *n*
> Produces an output line for lines that do not match in addition to one for a pair of records that does match. If you specify *n* as one of 1 or 2, **join** produces unpaired records from only that file. If you specify both **–a** 1 and **–a** 2, it produces unpaired records from both files.

–e *string*
> Replaces an empty field with *string* on output.

–j[*n*] *m*
> Uses field number *m* as the join key field. By default, the join key field is the first field in each input line. As with the **–a** option, if *n* is present, this option specifies the key field just for that file; otherwise, it specifies it for both files.

–o *list*
> Specifies the fields to be generated. You can specify each element in *list* as either *n.m*, where *n* is a file number (1 or 2) and *m* is a field number, or as 0 (zero), which represents the join field. You can specify any number of output fields by separating them with blanks or commas. The POSIX-compatible version of this command (first form in the syntax) requires multiple output fields to be specified as a single argument; therefore, shell quoting may be necessary. **join** generates the fields in the order you list them.

–t *c*
> Sets the field separator to the character *c*. Each instance of *c* introduces a new field, making empty fields possible.

–v *n*
> Suppresses matching lines. If you specify *n* as one of 1 or 2, **join** produces unpaired records from only that file. If you specify both **–v** 1 and **–v** 2, it produces unpaired records from both files. This does not suppress any lines produced using the **–a** option.

–1 *n*
> Uses the *n*th field of *file1* as the join key field.

–2 *n*
> Uses the *n*th field of *file2* as the join key field.

Localization

join uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

Successful completion

1

Failure due to any of the following:

- Incorrect syntax
- The wrong number of command-line arguments
- Inability to open the input file
- Badly constructed output list
- Too many **–o** options on the command line

2

Failure due to an incorrect command-line argument

Messages and Return Codes

Most diagnostics deal with argument syntax and are self-explanatory. For example:

Badly constructed output list at *list*
Indicates that the list for a **–o** option did not have the proper syntax.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

POSIX considers the **–j** option to be obsolete.

Related Commands

awk, **comm**, **cut**, **paste**, **sort**

kill — End a process or job, or send it a signal

> **kill –l** *exit_status***]**
> **kill [–s** *signal_name***] [***pid***...] [***job-identifier***...]**
> **kill [–***signal_name***] [***pid***...] [***job-identifier***...]**
> **kill [–***signal_number***] [***pid***...] [***job-identifier***...]**

Purpose

kill ends a process by sending it a signal. The default signal is **SIGTERM**.

Options

You can specify the following options on the command line:

–l
> Displays the names of all supported signals. If you specify *exit_status*, and it is the exit code of a ended process, **kill** displays the ending signal of that process.

–s *signal_name*
> sends the signal *signal_name* to the process instead of the **SIGTERM** signal. When using the **kill** command, do not use the first three characters (*SIG*) of the *signal_name*. Enter the *signal_name* with uppercase characters. For example, if you want to send the **SIGABRT** signal, enter:

```
kill -s ABRT pid
```

–*signal_name*
> (Obsolete.) Same as **–s** *signal_name*.

–*signal_number*
> (Obsolete.) A positive integer representing the signal to be used instead of **SIGTERM** as the *sig* argument in the effective call to **kill**.

The relationship between the *sig* value and integer values is shown as follows:

signal_number
> *signal_name*

0
> 0

1
> **SIGHUP**

2
> **SIGINT**

3
> **SIGQUIT**

6
> **SIGABRT**

9
> **SIGKILL**

14
> **SIGALRM**

15
> **SIGTERM**

The effects of specifying any *signal_number* other than those listed in the table is undefined.

Operands

kill recognizes the following operands:

job-identifier
Is the job identifier reported by the shell when a process is started with **&**. It is one way to identify a process. It is also reported by the **jobs** command. When using the job identifier with the **kill** command, the job identifier must be prefaced with a percent (%) sign. For example, if the job identifier is 2, the **kill** command would be entered as follows:

```
kill -s KILL %2
```

pid
Is the process ID that the shell reports when a process is started with **&**. You can also find it using the **ps** command. The *pid* argument is a number that may be specified as octal, decimal, or hex. Decimal process IDs are reported with default actions. **kill** supports negative values for *pid*.

If *pid* is negative but not -1, the signal is sent to all processes whose process group ID is equal to the absolute value of *pid*. The negative *pid* is specified in this way:

```
kill -s KILL -- -nn
```

where *nn* is the process group ID and may have a range of 2 to 7 digits (*nn* to *nnnnnnn*).

```
kill -s KILL -- -9812753
```

The format must include the - -   - before the *nn* in order to specify the process group ID.

If *pid* is 0, the signal is sent to all processes in the process group of the invoker.

The process to be killed must belong to the current user, unless the current user is the superuser.

Localization

kill uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
Successful completion.

1
Failure due to one of the following:

- The job or process did not exist
- There was an error in command-line syntax

2
Failure due to one of the following:

- Two jobs or processes did not exist
- Incorrect command-line argument
- Incorrect signal

>2
　　Tells the number of processes that could not be killed.

Messages and Return Codes

Possible error messages include:

***job-identifier* is not a job**
　　You specified an incorrect ID.

***signal_name* is not a valid signal**
　　You specified a noninteger signal for **kill**, or you specified a signal that is outside the range of valid signal numbers.

Portability

POSIX.2, X/Open Portability Guide.

Related Commands

jobs, **ps**, **sh**

let — Evaluate an arithmetic expression

```
let expression ... ((expression))
```

Purpose

let evaluates each arithmetic *expression* from left to right, using long integer arithmetic with no checks for overflow. No output is generated; the exit status is 0 if the last *expression* argument has a nonzero value, and 1 otherwise.

The following two lines are equivalent: the second form avoids quoting and enhances readability. These two forms are extensions to the POSIX standard. The ((expression)) form can be entered only if the shell is running in korn mode; in other words, set -o korn has been entered.

```
let "expression"
((expression))
```

The POSIX version of this command is as follows:

```
$((expression))
```

Expressions consist of named variables, numeric constants, and operators. See "Arithmetic Substitution" on page 286.

Examples

Examples of the three forms of the **let** command are as follows:

```
let a=7
echo $a
```

produces:

```
    7
```

```
echo $((a=7*9))
```

produces:

```
    63
```

```
set -o korn
((a=3*4))
echo $a
```

produces:

```
    12
```

Usage Notes

This command is built into the shell.

Exit Values

Possible exit status values are:

0

The last argument evaluated to a nonzero value.

1

The last argument evaluated to a zero value, or the expression contained a syntax error or tried to divide by zero.

Portability

POSIX.2. The POSIX version of this command is **$((***expression***)).**

Related Commands

expr, **sh**, **test**

lex — Generate a program for lexical tasks

```
lex [-achlntTv] [-o file.c] [-P proto] [-p prefix] [file.l ...]
```

Purpose

lex reads a description of a lexical syntax, in the form of regular expressions and actions, from *file.l*, or the standard input if no *file.l* is provided or if the file is named **–**. It produces a set of tables that, together with additional prototype code from **/etc/yylex.c**, constitute a lexical analyzer to scan those expressions. The resulting recognizer is suitable for use with **yacc**. You can find detailed information regarding the use of **lex** in *z/VM: OpenExtensions Advanced Application Programming Tools*.

For a description of the typedefs, constants, variables, macros, and functions in the table file, which can be used to access the lexical analyzer's variables or to control its operations, see *z/VM: OpenExtensions Advanced Application Programming Tools*.

Options

lex recognizes the following options:

–a

Generates 8-bit tables instead of 7-bit tables. On systems with 8-bit character sets (such as this one), this option is always enabled.

–c

Generates C code. Because this is the default, this option is provided only for compatibility with other implementations.

–h

Prints a brief list of the options and quits.

–l

Suppresses **#line** directives in the generated code.

–n

Suppresses the display of table sizes by the **–v** option. If you did not specify **–v** and there are no table sizes specified in *file.l*, **lex** behaves as though you specified **–n**.

–o *file.c*

Writes the lexical analyzer (internal state tables) onto the named output file, instead of the default file **lex.yy.c**.

–P *proto*

Uses the named code file, instead of the default prototype file **/etc/yylex.c**.

–p *prefix*

Uses the given prefix instead of the prefix **yy** in the generated code.

–T

Writes a description of the analyzer onto the file **l.output**.

–t

Writes the lexical analyzer onto standard output, instead of the file **lex.yy.c**.

–v

Displays the space used by the various internal tables. Normally **lex** displays these statistics on the standard output, but if you also specified the **–t** option, it displays them on the standard error. If you did not choose this option and *file.l* specifies table sizes, **lex** still displays these statistics unless you specified the **-n** option.

The LEX library contains a number of functions essential for use with **lex**. These functions are described in *z/VM: OpenExtensions Advanced Application Programming Tools*. The actual library to use depends on your system and compiler. For OpenExtensions programs, you should use **-1l**.

Some **lex** programs can cause one or more tables within **lex** to overflow. These tables are the NFA, DFA, and move tables; **lex** displays an appropriate message if an overflow occurs. You can change table sizes by inserting the appropriate line into the *definition* section of the **lex** input, with the number *size* giving the number of entries to use. This is shown in Table 7 on page 163.

Table 7. Internal Table Sizes

Line	Table Size Affected	Default
%e*size*	Number of NFA entries	1000
%n*size*	Number of DFA entries	500
%p*size*	Number of move entries	2500

You can often reduce the NFA and DFA space to make room for more move entries.

Locale

A *locale* is the subset of a user's environment that depends on language and cultural conventions. A locale defines such things as the definition of characters, and the collation sequence of those characters. POSIX.2 defines a POSIX locale, which is essentially USASCII.

Since **lex** generates code that is then compiled before being executed, it is difficult for **lex** to act properly on collation information. The POSIX.2 standard therefore does not require **lex** to accept any locales other than the POSIX locale. **lex** accepts regular expressions in this locale only.

Files

l.output
Scanner machine description

lex.yy.c
Tables and action routines

/etc/yylex.c
The prototype **lex** scanner

/usr/lib/libl.a
lex function library

Localization

lex uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
Successful completion

1

    Failure because of any of the following:

- Inability to create an output file
- Inability to open the file
- Missing output file name after **-o**
- Missing prefix after **-p**
- No **lex** rules
- No memory for DFA moves
- Out of NFA state space
- Out of DFA move space
- Out of DFA state space
- Push-back buffer overflow
- Read error on file
- Table too large for machine
- Too many character classes
- Too many translations
- Unknown option
- Write error on file
- Incomplete %{ declaration
- Token buffer overflow

Limits

The parser stack depth is limited to 150 levels. Attempting to process extremely complicated syntaxes may result in an overflow, causing an error.

Portability

POSIX.2, UNIX systems.

The **–a**, **–h**, **–l**, **–o**, **–p**, **–P**, and **–T** options are extensions of the POSIX standard.

Related Commands

yacc (see *z/VM: OpenExtensions Advanced Application Programming Tools*)

ln — Create a link to a file

---

ln [–fiRrs] *old new*
ln [–fiRrs] *old old ... dir*

Purpose

ln creates a link to an existing file or set of files. A *link* is a new directory entry that refers to the same file. This entry can be in the same directory that currently contains the file or in a different directory. The result is that you get a new path name that refers to the file. You can access the file under the old path name or the new one. Both path names are of equal importance. If you use **rm** to remove either name, the other one still remains and the file contents are still available under that name. The contents of the file do not disappear until you remove the last link.

A file can have any number of links to it. Thus you can establish any number of different path names for any file.

In the first form given in the syntax, *new* becomes a new path name for the existing file *old*. In the second form, **ln** creates entries for all the *old* files under the directory *dir*. For example:

```
ln yourdir/* mydir
```

creates links under **mydir** to all the files under **yourdir**. The files have the same names under **mydir** that they had under **yourdir**. **ln** always assumes this directory form when the last operand on the command line is the name of a directory. In this case, none of the *old* names can be a directory.

There could already be a file with the same name as the link you are trying to set up: a *conflicting* path name. To deal with a conflicting path name, **ln** follows these steps:

- If you have specified **–i**, **ln** writes a prompt to standard error to ask if you want to get rid of the conflicting path name. If you answer affirmatively, **ln** attempts to remove it.

- Otherwise, if you have specified **–f**, **ln** attempts to remove the existing file without a warning.

- Otherwise, **ln** prints a diagnostic message.

- **ln** gets to this point if it is going to get rid of the conflicting path name. It therefore attempts to get rid of the conflicting path name in the same way that **rm** does. **ln** deletes the file associated with the path name if this path name is the last link to the file. If **ln** can't get rid of the conflicting path name, it does not attempt to establish the new link; it simply prints an error message on the standard error and goes on to process any other files.

- If **ln** successfully gets rid of the conflicting path name, it then establishes the link.

Options

ln recognizes the following options:

–f

Gets rid of any conflicting path names without asking you for confirmation.

–i

Checks with you before getting rid of conflicting path names. You must not specify both **–f** and **–i**.

–R

Links files recursively. That is, you can link an entire hierarchy of subdirectories at once.

–r

Is identical to **–R**.

–s

Creates a symbolic link.

The locale settings for **LC_COLLATE**, **LC_CTYPE**, and **LC_MESSAGES** affect the program's interpretation of what constitutes a "yes" answer when **ln** asks if you want to get rid of a conflicting path name.

Localization

ln uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

All requested links were established successfully.

1

Failure due to any of the following:

- An argument had a trailing / but was not the name of a directory.
- A file could not be found.
- An input file could not be opened for reading.
- An output file could not be created or opened for output.
- The new link file already exists.
- A link could not be established.
- A read error occurred on an input file.
- A write error occurred on an output file.
- The input and output files were the same file.
- Inability to access a file when using **–r**.
- Inability to read a directory when using **–r**.
- Inability to create a directory when using **–r**.
- A target is not a directory when using **–r**.
- Source and destination directory are the same when using **–r**.

2

Failure due to any of the following:

- Incorrect command-line option.
- Too few arguments on the command line.
- A target that should be a directory but isn't.
- No space left on target device.
- Out of memory to hold the data to be copied.
- Inability to create a directory to hold a target file.

Messages and Return Codes

Possible error messages include:

link to target *name* **failed**
> **ln** could not establish the link to the given file or directory. This may be because you do not have appropriate permissions, or because the target did not exist.

source *name* **and target** *name* **are identical**
> The source and the target are actually the same file (for example, because of links, on UNIX systems). In this case, **ln** does nothing.

target directory *name* **on different file system than source** *name*
> You cannot establish a normal link between files that are two different file systems.

target *name* **must be a directory**

cannot find file *name*

target file *name* **already exists**

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Only the **–f** option is part of the POSIX standard.

Related Commands

cp, **locale**, **mv**, **rm**

locale — Get locale-specific information

```
locale [–a/–m]
locale [–ck] name ...
```

Purpose

locale displays information about the current locale and all locales accessible to the current application. **locale** searches directory **/usr/lib/nls/locale** for all the compiled locales.

Invoking **locale** with no options or operands displays the values of the **LANG** and **LC_*** environment variables. If a **LC_*** variable is not set or is overridden by **LC_ALL**, **locale** displays its implied value in double quotation marks.

The operand *name* can be a category name, keyword name, or the reserved name `charmap`. If it is a category name, **locale** selects the given category and all keywords within it for output. If *name* is a keyword name, **locale** selects the given keyword and its category for output. If *name* is `charmap`, **locale** displays the name of the charmap used on the LOCALDEF utility when the locale was created. For information about LOCALDEF, see *XL C/C++ for z/VM: User's Guide*.

Options

locale recognizes the following options:

–a

   Displays information about all accessible locales including **POSIX**, the POSIX locale.

–c

   Displays the names of selected categories.

–k

   Displays the names of selected keywords. If you do not specify the **–k** option, **locale** displays the values of selected keywords but not their names. With **–k**, strings are written in an unambiguous form using the `escape` character from the current locale.

–m

   Displays a list of all available charmaps.

Examples

In the following examples, let's assume that locale environment variables are set as follows:

```
LANG=locale_x
LC_COLLATE=locale_y
```

1. The command:

   ```
   locale
   ```

   produces the following output:

   ```
   LANG=locale_x
   LC_CTYPE="locale_x"
   LC_COLLATE=locale_y
   LC_TIME="locale_x"
   LC_NUMERIC="locale_x"
   LC_MONETARY="locale_x"
   LC_MESSAGES="locale_x"
   LC_ALL=
   ```

2. The command:

```
LC_ALL=POSIX locale -ck decimal_point
```

produces:

```
LC_NUMERIC
decimal_point="."
```

3. The following command shows an application of **locale** to determine whether a user supplied response is affirmative:

```
if printf "s%\n" "$response" | grep -Eq "$(locale yesexpr)"
then
        affirmative processing goes here
else
        nonaffirmative processing goes here
fi
```

Localization

locale uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See for more information.

Exit Values

Possible exit status values are:

0
Successful completion.

1
An error occurred.

2
A usage message was printed.

Portability

POSIX.2, UNIX system V.

Related Commands

LOCALDEF utility

logger — Log messages

logger [**–IisTu**] [**–d** *dest*] [**–f** *filename*] [**–p** *priority*] [**–t** *tag*] *string...*

Purpose

logger saves a message in the console log; the message consists of the *string* operand on the command line. Some options of **logger** may be in effect by default; if they are on by default, they cannot be disabled.

The **–u** and **–i** options are in effect by default, so all messages from **logger** are prefixed by process ID and user login user name.

If there is no message specified on the command line, the standard input is read; each line of standard input is treated as a log message. If **–f** *filename* is specified, the file is read instead of the standard input.

Options

logger recognizes the following options:

–d *destination*
> CMS uses the TELL command to transmit your log message to the place specified by *destination*. Any single-token value suitable for use in a TELL command may be used for *destination*. If you do not specify a destination, CMS uses TELL OP, sending your log message to the system operator.
>
> **Note:** This option works on OpenExtensions; however, since it is system-specific, it may or may not actually work on another system.
>
> For more information on *destination*, see the TELL or NAMES command in *z/VM: CMS Commands and Utilities Reference*.

–f *filename*
> Reads log messages from the file *filename* rather than from the standard input.

–I
> Adds the parent process ID (PPID) of **logger** to the message.

–i
> Adds the process ID (PID) of **logger** to the message. This option is in effect by default, so all messages from **logger** are prefixed by the PID.

–p *priority*
> The *priority* is ignored on VM.
>
> **Note:** This option works on OpenExtensions; however, since it is system-specific, it may or may not actually work on another system.

–s
> Overrides any destination options and causes logging to the standard error output.

–T
> Adds a time stamp (%x %X format, per date) to the message. This time stamp is always in the POSIX locale, no matter the locale of the message.

–t *tag*
> Adds *tag* to the start of the message.

–u
> Adds the login name of the controlling terminal to the message. This option is in effect by default, so all messages from **logger** are prefixed by the login name.

Localization

logger uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
    Successful completion

>0
    An error occurred.

Messages and Return Codes

Possible error messages include:

-f *filename* **invalid if message given**
    Both a file name and message was specified; only one is allowed.

file *filename***: system error**
    The file specified by **–f** *filename* could not be opened.

Formatted log message too long -- limit LINE_MAX (*number***)**
    The log message specified was longer than the limit specified by **LINE_MAX**.

Unknown option *option*
    You specified an incorrect option to **logger**.

Portability

POSIX.2.

All the options are extensions.

logname — Return a user's login name

```
logname
```

Purpose

logname returns the user ID of the person who enters the command. **logname** returns your login name, which is your z/VM logon ID. It is displayed as all lowercase letters, regardless of how it was entered.

More precisely, it displays the current value of the **LOGNAME** environment variable; when you sign on, this is automatically set to your login name.

Localization

logname uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
    Successful completion

1
    logname could not determine the login name.

Environment Variables

LOGNAME
    Contains your user name.

Portability

POSIX.2, X/Open Portability Guide, UNIX system V.

Related Commands

env, **id**

lp — Send a file to a printer

lp [–cmsw] [–d *dest*] [–n *number*] [–o *printer-option*] [–t *title*] [*file...*]

Purpose

lp prints one or more input files on a printer. If you do not specify any files on the command line, or if you specify a file name of –, **lp** reads and prints the standard input. The files are printed in the same order that they are specified on the command line.

Options

lp supports the following options.

–c
> Immediately makes a copy of the files to be printed. This ensures that the version of the file that exists when the print request is made is the version printed. On OpenExtensions, this option is always in effect, whether it was specified or not.

–d *dest*
> Specifies *dest* as the output device. **–d** takes precedence over the **LPDEST** environment variable, which in turn takes precedence over the **PRINTER** environment variable.

> The *dest* is a comma-separated list of three items, *destination*, *class*, and *forms*. These items are defined as follows:

> **destination**
>> This item can take one of these forms:

>> **node.user**
>>> The print file is sent to this user at this node.

>> **user**
>>> The print file is sent to this user on your node.

>> ***nick***
>>> The print file is sent to the user defined by the nickname *nick* in your NAMES file.

> **class**
>> The class to which your virtual printer should be spooled

> **forms**
>> The forms for which your virtual printer should be spooled.

–m
> This option is not implemented.

–n *number*
> Prints *number* copies of each input file (the default is 1 copy).

–o *printer-option*
> This option is not implemented.

–s
> This option is not implemented.

–t
> This option is not implemented.

–w
> This option is not implemented.

Environment Variables

LPDEST
　　Names the output device. This variable takes precedence over **PRINTER**.

PRINTER
　　Names the output device if **LPDEST** is not defined.

Examples

1. The following sends a previously formatted file to a VM printer:

```
lp filename
```

　You can specify more than one file name with the command.

2. Either of the following prints the file **temp.prt** using the default printer destination and specifying class c (where c is the locally designated class for confidential information):

```
lp -d ,c temp.prt

lp -d,c temp.prt
```

　The parameters on the **-d** option are positional, so if you omit a destination, you must still include the comma.

Localization

lp uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See for more information.

Exit Values

Possible exit status values are:

0
　　Successful completion

>0
　　An error occurred.

Portability

POSIX.2, X/Open Portability Guide.

ls — List file and directory names and attributes

```
ls [–AabCcDdFfgiLlmnopqRrstuWx1] [pathname …]
```

Purpose

ls lists files and directories. If *pathname* is a file, **ls** displays information on the file according to the requested options. If it is a directory, **ls** displays information on the files and subdirectories therein. You can get information on a directory itself using the **–d** option.

If you do not specify any options, **ls** displays only the file names. When **ls** sends output to a pipe or a file, it writes one name per line; when it sends output to the terminal, it uses the **–C** (multicolumn) format.

Options

ls displays at least the file name; you can request more information with the following options:

–A
> Lists all entries including those starting with periods (.).

–a
> Lists all entries including those starting with a period (.).

–b
> Displays nonprintable characters as octal bytes with the form *ooo*.

–C
> Puts output into columns, sorted vertically; this is the default output format to the terminal.

–c
> Uses the time of the last change of the file's attributes for sorting (**–t**) or displaying (**–1**).

–D
> Displays requested information about directories only.

–d
> Does not display the contents of named directories, but information on the directories themselves.

–F
> Puts a **/** after each directory name, a ***** after every executable file, a **|** after every FIFO file, a **@** after every symbolic link, and a **=** after every socket. It also puts an & character after an external link name.

–f
> Forces the *pathname* argument to be a directory; turns off sorting. **ls** gives the ordered list of file names in a directory file. The directory file is read and the file names are listed in the same order as they are returned. The contents of a directory file are shown.

–g
> Does not display group ID numbers.

–i
> Displays file serial (inode) numbers along with file names.

–L
> Follows symbolic links. Symbolic links are automatically followed unless the **–g –1**, **–n**, or **–o** option is specified. The **–L** option forces symbolic links to be followed even when these other options are specified.

–1
> Displays permissions, links, owner, group, size, time, name; see "Long Output Format" on page 176.

–m
> Displays names in a single line, with commas separating names.

−n

Displays user ID and group ID numbers.

−o

Displays only the user ID of the owner.

−p

Puts / after directory names.

−q

Displays nonprintable characters as ?.

−R

Lists subdirectories recursively.

−r

Sorts in reverse of usual order; you can combine this with other options that sort the list.

−s

Displays size in blocks, after the file serial (inode) number, but before other information.

−t

Sorts by time. By default, this option sorts the output by the modification times of files. You can change this with the **−c** and **−u** options.

−u

Uses the last access time for sorting (**−t**) or displaying (**−1**).

−W

Displays the audit bits of the file.

−x

Puts output into sorted columns, with output going across the rows.

−1

Forces output to be one entry per line.

Notes:

1. When you specify options that are mutually exclusive (for example, **−c** and **−u**), the option that appears last on the command line is used.

2. The owning user and group values are user and group names, with these exceptions:

   - There is not a user in the CP directory who currently has the UID that is the owning UID for the file.
   - The user entering this command does not have authorization to query user database information for other users.

   In either of these cases the values displayed will be the UID and GID.

Long Output Format

The output from **ls −l** summarizes all the most important information about the file on a single line. If the specified *pathname* is a directory, **ls** displays information on every file in that directory (one file per line). It precedes this list with a status line that indicates the total number of file system blocks occupied by files in the directory (in 512-byte chunks). Here is a sample of the output along with an explanation:

```
total 11
drwxr-xr-x      3 root    sys1     0 Mar 12 19:32 tmp
drwxrwxrwx      4 root    sys1     0 Mar 12 19:32 usr
drwxr-xr-x      2 root    sys1     0 Mar 12 19:32 bin
-rwxr--r--      1 root    sys1   572 Mar 12 19:32 foo
-rwxr--r--      1 root    sys1   640 Mar 12 19:33 abc
```

The first character identifies the file type:

-

Regular file

b

Block special file

c

Character special file

d

Directory

E

External link

l

Symbolic link

p

FIFO

s

Socket file

The next 9 characters are in three groups of 3; they describe the permissions on the file. The first group of 3 describes owner permissions; the second describes group permissions; the third describes other (or "world") permissions. Characters that may appear are:

r

Permission to read the file

w

Permission to write on the file

x

Permission to execute the file or permission to search the directory.

The following characters appear only in the execute permission (x) position of the output.

S

Same as s, except that the execute bit is off.

s

If in owner permissions section, the set-user-ID bit is on; if in group permissions section, the set-group-ID bit is on. The execute bit is also on.

T

Same as t, except that the execute bit is off.

t

The sticky bit is on. The execute bit is also on.

You can set permissions with the **chmod** command.

After the permissions are displayed, **ls** displays the following (using the preceding example), in order:

- The number of links to the file.
- The name of the owner of the file or directory.
- The name of the group that owns the file or directory.
- The size of the file, expressed in bytes.
- For a file, the date and time the file was last changed; for a directory, when it was created. The **–c** and **–u** options can change which time value is used. If the date is more than 6 months old or if the date is in the future, the year is shown instead of the time.
- The name of the file or directory.

If **ls –W** is issued, an additional 6 characters, in two groups of 3, follow the original 10 characters. The first group of 3 describes the user-requested audit information; the second group describes auditor-requested audit information.

```
total 11
drwxr-xr-x  fff---    3 root   sys1    0 Mar 12 19:32 tmp
```

```
drwxrwxrwx  fff---    4 root   sys1    0 Mar 12 19:32 usr
drwxr-xr-x  fff---    2 root   sys1    0 Mar 12 19:32 bin
-rwxr--r--  fff---    1 root   sys1  572 Mar 12 19:32 foo
-rwxr--r--  fff---    1 root   sys1  640 Mar 12 19:33 abc
```

Note: Audit bits can be set only by the callable service BPX1CHA. See *z/VM: OpenExtensions Callable Services Reference* for more information.

Environment Variables

COLUMNS
Contains the terminal width in columns. **ls** uses this value to determine the number of output columns to write using the **–C** option.

TZ
Contains the time zone to be used when displaying date and time strings.

Localization

ls uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**

See for more information.

Usage Notes

For a mounted external link, the output for options **–g –l**, **–n**, **–o**, and **–W** provides information about the link itself, not the linked object. In the output from the **–l**, option, the fully qualified pathname of the external link target is displayed following the name of the external link (the name of the file or directory). To get information on the target of the external link, you must reissue the command with one of the following changes:

- Include the **–L** option
- Include a closing slash (/) following the name of the external link
- Specify the fully qualified pathname of the external link

Exit Values

Possible exit status values are:

0
Successful completion

1
Failure due to any of the following:

  - Out of memory
  - Inability to find a file's information
  - Too many directories
  - File or directory not found
  - Specified on the command line

2
Incorrect command-line option

Messages and Return Codes

Possible error messages include:

File or directory *name* **is not found**
> The requested file or directory does not exist.

Cannot allocate memory for sorting
> To sort its output, **ls** needs to allocate memory; this message says that there was not enough memory for the sorting operation.

Too many directory entries in *dir*
> This message appears only when **ls** runs out of dynamically allocated memory.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–A**, **–b**, **–f**, **–g**, **–L**, **–m**, **–n**, **–o**, **–p**, **–s**, **–W**, and **–x** options are extensions of the POSIX standard.

mailx — Send or receive electronic mail

---

mailx [–efHiNn] [–u *user***] [***filename***]**
mailx [–FinU] [–h *number***] [–r** *address***] [–s** *subject***]** *user ...*

Purpose

mailx helps you read electronic mail messages. It can also send messages to users on your system, but it has no built-in facilities for sending messages to other systems.

The command line:

```
mailx [options] user user user ...
```

sends a mail message to the given users. If you do not specify any users on the command line, **mailx** lets you read incoming mail (interactively); however, see the environment variable ('sendmail').

This description of **mailx** is divided into several sections:

- Options
- General overview
- Command-mode subcommands
- Input-mode subcommands
- Startup files
- Example
- Environment variables
- Files
- Exit values
- Portability
- Related Information

If you are unfamiliar with electronic mail systems, first read "General Overview" and come back to the "Options" section when you have a grasp of how **mailx** works.

The **mailx** utility invokes another program, **/usr/lib/tsmail**, to transmit mail to other users, and **tsmail** is a set-user-ID program. If your VM user ID is not authorized to run set-user-ID programs, then you cannot use **mailx** to send mail to other users, but other **mailx** functions will still work. Authorizations for set-user-ID programs are controlled in the CP directory and in CP's configuration file, SYSTEM CONFIG. For more information about set-user-ID authorization, see *z/VM: CP Planning and Administration*.

Options

You can use the following options ***when reading messages:***

–e
   Checks to see if you have any messages waiting to be read. With this option, nothing is displayed. If you have waiting messages, **mailx** exits with a successful status return; otherwise, **mailx** exits with a failure return.

–f *filename*
   Looks for messages in the specified file instead of in your current mailbox. If you do not specify *filename*, **mailx** reads messages from $HOME/mbox.

–H
   Displays only the header summary of a message.

–N

Does not display the header summary of messages.

–u *user*

Looks for messages in the system mailbox of the specified user. This works only if you have read permission on the user's system mailbox.

You can use the following options **only** *when sending messages:*

–F

Records your message in a file with the same name as the first user specified on the command line. This option overrides the `record` variable, if it has been set. See "Environment Variables" on page 192 for more on the `record` variable.

–h *number*

Indicates how many "hops" a message has already made from one machine to another (in a network of machines). This option is not intended for most users; network mail software uses the option to prevent *infinite loops* (the same message cycling through a sequence of machines without ever getting to its intended destination).

–r *address*

Passes the given address to network mail software. If this option is present, it disables all input mode commands. Again, this option is not intended for most users.

–s *subject*

Uses the given *subject* string in the `Subject` heading line of the message. If the subject contains spaces or tab characters, the string should be enclosed in double quotation marks or single quotation marks. If you specify this option on the command line, **mailx** does not prompt you to enter a subject line when you type in the text of the message.

–U

Converts the address from UNIX-to-UNIX Copy Program (UUCP) style to Internet Protocol standards. This option overrides the effect of the *conv* environment variable.

This option is not supported with OpenExtensions.

You can use the following options *for both sending and reading messages:*

–i

Ignores interrupts (for example, from pressing <Break> or <Ctrl-c>). Also see the description of the *ignore* environment variable in "Environment Variables" on page 192.

–n

Does not initialize your **mailx** session from the system's **/etc/mailx.rc** file. For more information about this file, see "Startup Files" on page 191.

General Overview

We will begin by describing the *default* behavior of **mailx**.

The simplest command to send a message is:

```
mailx address address address ...
```

where each `address` names someone who is to receive the message. The simplest kind of address is the *login name* of someone else who uses your OpenExtensions shell.

You can also send messages as input to commands. To do this, use an address that consists of an "pipe symbol" (|) followed by a command line that invokes the appropriate command; enclose this whole address in single quotation marks. For example:

```
mailx robin '|cat>save'
```

mails a message to `robin` and also copies the message into a file called **save**.

After you type in the command to send a message, **mailx** asks you to enter the subject of the message (a brief description of what the message is about), and then lets you type in the text of the message. Your

message can consist of any number of lines, and may include blank lines. When you finish entering the message, type a line consisting only of a tilde (**~**), followed by a period (**.**); then press <Enter>. This tells **mailx** that the message is ready to be sent.

mailx puts the completed message into a file called the recipient's *system mailbox*. The message stays in the system mailbox until the recipient asks to read the message. At that point, the message is obtained from the system mailbox and displayed on the recipient's workstation. The message is then saved in the recipient's *personal mailbox.* Since this is usually a file named **mbox** in the recipient's home directory, we use the name *mbox* to represent the personal mailbox and *mailbox* for a system mailbox.

The simplest way to read incoming messages is to type the command **mailx** (with no addresses on the command line). This starts an *interactive session* in which **mailx** lets you read your mail and perform other operations. For example, you can display new messages, delete old ones, reply to messages, or forward them to someone else, and so on. When you are performing operations in this way, you are in *command mode.* When you are typing in the text of a message, you are in *input mode*.

A message consists of a sequence of *header lines* followed by the body of the message. The header lines tell who sent the message, the time and date that the message was sent, the subject of the message, and so on. **mailx** automatically creates header lines. Some of the common header lines are:

Cc: *name name ...*
> Stands for "carbon copies". This indicates that copies of this message are to be sent to the specified recipients. The names of these recipients appear in the header lines of everyone receiving the message.

Bcc: *name name ...*
> Stands for "blind carbon copies." This is similar to Cc:, but the names of people receiving carbon copies do not appear in the header lines of the message. Recipients do not know that these people received a copy of the message.

Subject: *text*
> Gives the subject of the message.

To: *name name ...*
> Gives the names of people who were sent the message directly.

All messages are in one of the following states:

deleted
> You used a **delete**, **dp**, or **dt** command to delete the message, or you saved it using a **Save** or **save** command and the variable *keepsave* was not set. When **mailx** quits, messages in this state are deleted.

new
> The message is in the system mailbox and you have not yet read it or otherwise changed its state. When **mailx** quits, messages in this state are kept in your system mailbox.

preserved
> You used a **preserve** command on the message. When **mailx** quits, messages in this state are kept in their current locations.

read
> You used one of the following commands on the message:

~F	**copy**	**Print**	**type**
~f	**mbox**	**print**	**undelete**
~M	**next**	**top**	
~m	**pipe**	**Type**	

> or you used **delete**, **dp**, or **dt** on the preceding message and the *autoprint* environment variable was set. When **mailx** quits and you are in your system mailbox, **read** messages are kept in your personal mailbox—unless the variable *hold* is set, in which case, **read** messages are kept in your system

mailbox. If you are in your personal or a secondary mailbox when **mailx** quits, **read** messages are kept in their current location.

unread
You have run more than one **mailx** session with the message in the system mailbox and you have not read it or otherwise changed its state. When **mailx** quits, messages in this state are kept in your system mailbox.

Command-Mode Subcommands

The standard format of a command-mode subcommand is:

```
[subcommand][refs][arguments]
```

If no **subcommand** is specified, **p[rint]** is assumed.

The *refs* argument indicates the messages to which you want to apply the **subcommand**. **mailx** numbers incoming messages sequentially as they are received. The easiest way to refer to a message is to give its number. For example, the subcommand:

```
p 3
```

displays message number 3. At any point in a **mailx** session, there is one message that is considered the *current message.* This is the message you most recently did something with (for example, the one you most recently read). If you omit the *refs* argument in a subcommand that uses *refs*, the subcommand works with the current message.

You can also use special notations as the *refs* value, as shown in Table 8 on page 183.

Table 8. Reference Notations

refs	**Meaning**
n	Message number *n*
n-m	Messages *n* through *m*
.	The current message
^	The first undeleted message (or first deleted message for *undelete*)
$	The last message
*	All messages
+	Next message
-	Previous message
'*user*'	All messages from *user*
/*string*	All messages with *string* in the subject line (the case of characters in *string* is ignored)
:d	All deleted messages
:n	All new messages
:o	All old messages
:r	All messages that have already been read
:u	All unread messages

Several *refs* arguments may be specified for the same subcommand, separated by spaces. For example:

```
p alice lewis
```

displays all messages from `alice` plus all messages from `lewis`.

The arguments allowed at the end of a command-mode subcommand depend on the subcommand itself. If a subcommand allows a file name as an argument, you can use the usual file name generation characters in the file name (see **sh**).

The following list shows the subcommands recognized in command mode. In every subcommand name, some characters are enclosed in square brackets. These characters are optional. For example, the [**p**]**rint** command may be given as **print** or **p**.

?
> Displays a summary of command-mode subcommands

=
> Displays the current message number

a[lias] [*alias* [*name* ...]]
> Sets up an address *alias*. If you enter a subcommand to send mail to the given *alias*, the messages are actually sent to the given list of names. For example, you might enter the subcommand:
>
> ```
> alias joe jsmith
> ```
>
> From this point onward, you can address messages to `joe` and they are sent to `jsmith`. You may also set up an alias for several people, as in:
>
> ```
> alias choir soprano alto tenor bass
> ```
>
> After you have done this, you can send messages to `choir` and they are sent to the names that follow `choir` in the command. Entering the **alias** subcommand without any arguments displays a list of the currently defined aliases.
>
> **Note:** Aliases entered interactively remain in effect only until the end of the current interactive session.
>
> To make an alias permanent, include the **alias** subcommand in your startup file. See "Startup Files" on page 191. See also **group**.

alt[ernates] *name*
> Lists a set of alternate names for your own login name. This is useful for people who login under several different names. When you reply to a message, **mailx** usually sends your reply to the author of the message and all the recipients as well; however, it does not send the message to any of your alternate login names. You don't have to worry about sending mail to yourself.
>
> Specifying alternates without names displays your list of currently defined alternate names.

cd *directory*
> Makes *directory* your new working directory. If no *directory* is specified, **cd** goes to your **HOME** directory.

ch[dir] *directory*
> Is the same as **cd**.

c[opy] [*filename*]
> Copies the current message into the specified file. If the file does not already exist, it is created. If no *filename* is specified, your **mbox** file is used.
>
> This operation does not mark the message as "saved"; if it was previously unread, it is still regarded as an unread message. Thus the original message remains in your system *mailbox*. See also **save**.

c[opy] *refs filename*
> Copies the messages referred to by *refs* into the given file. The *filename* must be specified. If the file does not already exist, it is created. As with the previous form of **copy**, the messages are not marked as "saved".

C[opy] [*refs*]
> Is similar to the **copy** command, except that the messages referred to are saved in a file the name of which is derived from the author of the first message referred to. The name of the file is the author's name, stripped of any network addressing. If the *folder* environment variable is set, the file is saved

to the specified directory. The copied messages are not marked as "saved". If *refs* is not specified, the current message is copied.

d[elete] [*refs*]

Deletes the specified messages from your system *mailbox*. If *refs* is not specified, the current message is deleted. After a delete operation, the current message is set to the message after the last message deleted. Deleted messages are not thrown away until you end your session with the current *mailbox* (see **quit** and **file**). Until then, they can be undeleted (see **undelete**).

di[scard] [*header...*]

Does not display the given *header* fields when displaying a message. For example:

```
discard References
```

tells **mailx** not to display the References line at the beginning of any mail message. These header lines are retained when the message is saved; they are just not shown when the message is displayed. See also **ignore** and **retain**.

dp [*refs*]

Deletes the specified messages and then displays the message after the last message deleted.

dt [*refs*]

Is the same as the **dp** subcommand.

ec[ho] *string ...*

Echoes the given *strings* (like the **echo** subcommand).

e[dit] [*refs*]

Lets you edit the messages specified by *refs*. The messages are stored in a temporary file and an editor is invoked to let you edit the file. The default editor is **ed**, but you can change this using the **EDITOR** environment variable (see "Environment Variables" on page 192).

ex[it]

Quits **mailx** without changing the system *mailbox*. Contrast this with **quit**.

fi[le] [*filename*]

Quits the system mailbox (as if a **q[uit]** subcommand were run) and then reads in the specified file as the new mailbox to examine. If no *filename* is specified, the default is your current *mailbox*.

Several special strings can be used in place of *filename*:

%

Your system *mailbox*

%user

The system *mailbox* for user

#

The previous file

&

Your *mbox* (personal mailbox)

+file

The named file in the **folder** directory

fold[er] [*filename*]

Is the same as the **file** subcommand.

folders

Displays the names of the files in the directory given by the $folder$ variable; see "Environment Variables" on page 192.

F[ollowup] [*refs*]

Replies to the first message given in *refs*; **mailx** sends this reply to the authors of every message given in *refs*. The Subject line is taken from the first message in *refs*. Your reply is automatically saved in a file which derives its name from the author of the message to which you are replying.

To create your reply, **mailx** puts you into input mode, where you can use all of the input mode commands.

fo[llowup] [*ref*]

Replies to the specified message; if no message *ref* is given, you reply to the current message. Your reply is automatically saved in a file which derives its name from the author of the message to which you are replying. This overrides the *record* environment variable if *record* is set; see "Environment Variables" on page 192.

To create your reply, **mailx** puts you into input mode, where you can use all of the input mode commands.

f[rom] [*refs*]

Displays the header summary for the specified messages. If *refs* is not given, the current message is used.

g[roup] [*alias* [*name* ...]]

Is the same as the **alias** command.

h[eaders] [*ref*]

Displays the headers of a screenful of messages surrounding the message given by *ref*. The number of lines in a screen is given by the *screen* environment variable.See *screen*.

hel[p]

Displays a summary of the command-mode subcommands.

ho[ld] [*refs*]

Retains the specified messages in your system *mailbox*. For example, you might decide to **ho**ld a message if you read it, but decide not to act upon it immediately. If *refs* is not specified, the current message is held. If any of the specified messages have been marked as deleted, the **hold** subcommand overrides that and still retains the messages. Subsequent **delete**, **dp**, and **dt** commands during the same **mailx** session can delete files marked for retention. See also **preserve**, and the environment variables *hold* and *keepsave*.

i[f] *code mailx_subcommands* [el[se] *mailx_subcommands*] [en[dif]]

Is primarily intended for use in startup files; see "Startup Files" on page 191 for information. The *code* must be the character r or s. If it is r, the first set of *mailx subcommands* are executed if **mailx** is in *receive mode*, and the second set if **mailx** is in *send mode*. If *code* is s, the opposite is true. The **else** part is optional.

ig[nore] [*header* ...]

Is the same as the **discard** subcommand.

l[ist]

Displays the names of all command-mode subcommands.

m[ail] *address* ...

Sends a message to the specified recipients. **mailx** goes into input mode to let you enter the text of the message.

mb[ox] [*refs*]

Indicates that the given messages are to be saved in your *mbox* (personal mailbox) when **mailx** quits normally (that is, through the **quit** command as opposed to **exit**).

n[ext] [*refs*]

Goes to the next message in the mailbox that appears in the list of *refs*. For example:

```
n user
```

goes to the next message from the specified *user*.

pi[pe] [[*refs*] *command*]

Pipes the messages given by *refs* through the specified shell *command*. These messages are considered read. If *refs* is not specified, the current message is used. If no *command* is specified, **mailx** uses the command specified by the *cmd* environment variable; see "Environment Variables" on page 192. If the *page* environment variable has a value, a form feed character is sent into the pipe after every message.

The subcommand | [*refs*] [*command*] is equivalent to **pipe**.

pre[serve] [*refs*]

Is the same as the **hold** subcommand.

P[rint] [*refs*]

Displays the specified messages on the screen. If *refs* is not specified, the current message is displayed. All header fields are displayed; the **discard** and **ignore** subcommands do not affect **Print**.

p[rint] [*refs*]

Displays the specified messages on the screen. If *refs* is not specified, the current message is displayed. Header fields specified by **discard** and **ignore** subcommands are not displayed. If the `crt` variable is set to an integer, messages with more lines than that integer are "paginated" using the command specified by the *PAGER* variable. For more information, see "Environment Variables" on page 192.

q[uit]

Ends a **mailx** session. This is the usual method to leave **mailx**. Messages that have been read but not saved or deleted are stored in your *mbox* (personal mailbox). Messages that are still unread are retained in your system *mailbox*. Messages that have been deleted or explicitly saved in other files are discarded. Typing the end-of-file character has the same effect.

R[eply] [*refs*]

Sends a reply to the authors of each of the messages specified by *refs*. If *refs* is not specified, the current message is used. The `Subject` line of the reply message is taken from the first message in *refs*. If the `record` environment variable is set to a file name, your reply message is appended to the end of that file.

Normally, you use **Reply** if you just want to send your reply to the author of a message, and **reply** if you want to send your reply to the author and all recipients. If set, the `flipr` environment variable reverses the meanings of the **R** and **r** commands. See "Environment Variables" on page 192.

r[eply] [*ref*]

Sends a reply to the author of a specific message, and all other recipients of the message. If *ref* is not specified, **mailx** replies to the current message. If the `record` environment variable is set to a file name, your reply message is appended to the end of that file.

R[espond] [*refs*]

Is the same as the **Reply** subcommand.

r[espond] [*ref*]

Is the same as the **reply** subcommand.

ret[ain] [*header* ...]

Is the opposite of the **discard** subcommand. It tells **mailx** to display the given *header* fields when displaying a message. The comparison of *header* fields is not case sensitive. You can use **retain** to override existing **discard** and **ignore** commands. If you do not specify any *header* fields, **retain** displays a list of currently retained header fields.

S[ave] [*refs*]

Saves the specified messages in a file the name of which is taken from the author of the first message (the file name is the author's name, without any attached network addressing). If the `folder` variable is set, the file is saved to the specified directory.

s[ave] [*refs*][*filename*]

Saves the specified messages in the given file. If *refs* is not given, the current message is saved. The file is created if it doesn't already exist. If you do not specify *filename*, **mailx** saves the messages in *mbox* (your personal mailbox). A message that has been saved with **s**ave is normally deleted from *mailbox* when **mailx** ends (see **quit**); but see the variables `hold` and `keepsave`.

se[t] *name*

Defines a variable with the given *name* and assigns it a null value. If you omit *name*, **set** displays a list of all defined variables and their values.

se[t] *name=value*
>   Defines a variable with the given *name* and assigns it the given *value*, which may be a string or a number.

se[t] no*name*
>   Is the same as the **unset** *name* subcommand.

sh[ell]
>   Invokes the shell given by the **SHELL** environment variable.

si[ze] [*refs*]
>   Displays the size in bytes of each of the specified messages. If no *refs* are specified, the current message is used.

so[urce] *file*
>   Reads the specified text *file*, executes its contents as command-mode subcommands, and then returns to read more commands from the original source.

to[p] [*refs*]
>   Displays the first few lines of each of the specified messages. If *refs* is not specified, the current message is used. If the `toplines` variable has a numeric value, that many lines are displayed from each message; otherwise, five lines are displayed from each message.

tou[ch] [*refs*]
>   "Touches" the specified messages, making them appear to have been read. This means that when you **quit mailx**, the messages are saved in your *mbox* (personal mailbox) if they are not deleted or explicitly saved in another file. If *refs* is not specified, the current message is touched.

T[ype] [*refs*]
>   Is the same as the **Print** subcommand.

t[ype] [*refs*]
>   Is the same as the **print** command.

una[lias] [*alias*[*name* ...]]
>   Deletes specified alias names.

u[ndelete] [*refs*]
>   Restores previously deleted messages. When messages are deleted, they are not discarded immediately; they are just marked for deletion and are actually deleted when **mailx** ends. Until **mailx** ends, you can use **undelete** to restore the specified messages. You cannot **undelete** messages deleted in previous sessions. If you do not specify *refs*, this command restores the first deleted (but not yet undeleted) message following the current message; if no such message exists, it restores the last deleted (but not yet undeleted) message preceding the current message. If the `autoprint` variable is set, the last restored message is displayed. This is the only subcommand that lets you give a *ref* to a message that has been deleted.

U[nread] [*refs*]
>   Marks the specified messages as unread.

uns[et] *name* ...
>   Discards the specified variables.

ve[rsion]
>   Displays version information about **mailx**.

v[isual] [*refs*]
>   Edits the specified messages with a screen editor. If *refs* is not specified, the current message is edited. The messages are saved in a temporary file and the screen editor is invoked to edit that file. The editor used is given by the **VISUAL** variable; see "Environment Variables" on page 192.

w[rite] [*refs*] *filename*
>   Writes the specified messages into the given file. If *refs* is not specified, the current message is written. **write** is the same as **save**, except that it does not write out the header lines and the blank line at the end of the message.

x[it]
>   Is the same as the **exit** command.

z+
> Scrolls the header display forward one screenful.

z−
> Scrolls the header display backward one screenful.

! *command*
> Executes the given shell *command*. For example:

```
!ls
```

> lists all files in the current directory. The shell that will be used to run the command is given by the **SHELL** environment variable; see "Environment Variables" on page 192.

#*comment*
> Specifies that **mailx** should ignore everything from the **#** to the end of the line. This is useful for putting comments into startup files.

?
> Is the same as the **help** command (it displays a summary of the command-mode subcommands).

Input-Mode Subcommands

You can use input-mode subcommands when entering the text of a message. You must type mode subcommands at the beginning of an input line; you cannot type them in the middle of a line. By default, each input-mode subcommand begins with the tilde (**~**) character, called the *escape character.* You can use the *escape* environment variable to change the escape character, but in the documentation that follows, we always use tilde.

~.
> Marks the end of input in a mail message.

~?
> Displays a summary of the input-mode subcommands.

~A
> Inserts the *autograph string* at this point in the message. This autograph string is given by the *Sign* environment variable.

~a
> Is similar to **~A**, except that it uses the variable *sign*.

~b *name ...*
> Adds the specified names to the blind carbon copy list.

~c *name ...*
> Adds the specified names to the carbon copy list.

~d
> Reads in the **dead.letter** file; see the description of **DEAD** in "Environment Variables" on page 192.

~e
> Invokes an editor on the message that you have composed. The **EDITOR** variable determines the editor that is invoked.

^F [*refs*]
> "Forwards" the given messages. The text of the messages is inserted at this point in the message you are composing. The message headers are also inserted with all header fields regardless of the **discard**, **ignore**, and **retain** subcommands. This is valid only when you entered **mailx** in command mode and then went into input mode to compose a message.

~f [*refs*]
> Is similar to **~F** except that the header fields included are determined by the **discard**, **ignore**, and **retain** subcommands.

~h
> Prompts you to enter the following header lines:

```
Subject   Cc   Bcc   To
```

For some of these, **mailx** displays an initial value for the header. You can edit this initial value as if you had just typed it in yourself, using backspaces and line deletes.

~i *name*
　　Inserts the value of the named variable followed by a newline at this point in the message.

~M [*refs***]**
　　Inserts the text of the specified messages at this point in the message. If *refs* is not specified, the current message is used. Messages inserted in this way have each line prefixed with the value of the `indentprefix` variable. The message headers are also inserted with all header fields included regardless of the **discard**, **ignore**, and **retain** subcommands. This is valid only when you entered **mailx** in command mode and then went into input mode to reply to a message.

~m
　　Is similar to **~M, except that the header fields are determined by the discard, ignore, and retain subcommands.**

~p
　　Displays the message being composed.

~q
　　Quits input mode as if you had interrupted the message. If you have already composed part of a message, the partial message is saved in the **dead.letter** file; see the description of the **DEAD** environment variable for more information (see "Environment Variables" on page 192).

~r *filename*
　　Reads in the contents of the specified file and adds that text at this point in the message.

~s *text*
　　Sets the `Subject` line to the given *text*.

~t *address address ...*
　　Adds the given addresses to the `To:` list (people who will receive the message).

~v
　　Invokes a screen (visual) editor on the message that you have composed. The **VISUAL** variable determines the editor that is invoked.

~w *file*
　　Writes the current text of your message to the specified *file*. The header lines for the message are not written.

~x
　　Quits in the same way as **~q**, except that the message is not saved in the **dead.letter** file.

~< *filename*
　　Is the same as the **~r** command.

~< !*command*
　　Runs the given shell *command* and adds the standard output of that command at this point in the message. For example, your message might contain:

```
My program is giving me this odd output:
~< !prog
What do you think is causing it?
```

~: *mail_command*
　　Runs the given command-mode *mail_command*. This is valid only when you entered **mailx** in command mode and then went into input mode to compose a message.

~_ *mail_command*
　　Is the same as the **~: command.**

~! *command*

    Runs the given shell *command*. For example, you can use:

```
 ~! ls
to get a list of files in the working directory.
The shell that is invoked to run the command is given by the
SHELL environment variable; see
"Environment Variables" on page 192.
If the bang variable is set, mailx
replaces each unescaped exclamation mark
(!) in command with the command run
by the previous command or ~! command escape.
```

~.

    Marks the end of input in a mail message.

~| *command*

    Pipes the current message through the specified shell *command*. If the *command* ends with a successful exit status, the output of the command replaces the text of the current message. ~| uses the shell given by the **SHELL** environment variable to run *command*.

Startup Files

When you run **mailx** in command mode, **mailx** does the following:

- Sets all variables to their default values. **mailx** processes command-line options, using them to override any corresponding default values.

- Imports appropriate external environment variables, using them to override any corresponding default values.

- Reads commands from the *system startup file*, **/etc/mailx.rc**. This sets up variable values and definitions that should be common to all users. If you do not want **mailx** to read the system startup file, use the **–n** option on the **mailx** command line.

- After reading and processing the system startup file, **mailx** does the same with a "personal startup file." The default name of the personal startup file is **$HOME/.mailrc**. You can override the name of the personal startup file by setting the **MAILRC** environment variable to the path name of the personal startup file that you prefer **mailx** to use.

Startup files typically set up display options and define aliases. However, any command is valid in a startup file except for the following:

    Copy
    edit
    followup
    Followup
    hold
    mail
    preserve
    reply
    Reply
    respond
    Respond
    shell
    visual
    !

If a line in a startup file contains an error or an incorrect command, the rest of the startup file is ignored. **mailx** ignores blank lines in a startup file.

Examples

The following example composes and sends a message to several users. Items shown in italics are output by **mailx** itself.

```
mailx jean
Subject:
Greetings
This is just a short note to say hello.
 ~c juan john johann
 ~.
```

On the first line, the message is just addressed to jean. The **~c line adds more people who will receive copies of the message.**

Environment Variables

A large number of variables are used to control the behavior of **mailx**. These environment variables are divided into two classes: those that always come from the external environment, and those that may be set up in either the external environment or within a **mailx** session.

The following variables always come from the external environment; they can be changed inside a **mailx** session, except where marked.

HOME
> Gives the name of your home directory. This cannot be changed inside **mailx**.

LOGNAME
> Gives your login name.

MAIL
> Gives the path name of the user's mailbox file for purposes of incoming mail notification.

MAILDIR
> Gives the name of the directory where system mailboxes are stored. If this is not set, the default is **/usr/mail**. The actual name of a user's system mailbox is derived in a system-dependent way by combining **MAILDIR** and the user's login name. For **mailx** to work properly, the **MAILDIR** directory must exist.

MAILRC
> Gives the name of your startup file. This cannot be changed inside **mailx**. By default, **MAILRC** has the value **$HOME**/.mailrc. For more on startup files, see "Startup Files" on page 191.

The **HOME** and **LOGNAME** variables must be set before you enter **mailx**; otherwise, **mailx** does not work properly. These variables are set automatically for you if you enter the shell using the CMS OPENVM SHELL command. If you do not log in, you must set the variables in some other way, using the commands:

```
export LOGNAME=name
export HOME=directory
```

The remaining variables can be set in the external environment or in the course of a **mailx** session. You can set or change the value of a variable with the **set** subcommand; you can discard a variable with the **unset** subcommand. You may find it convenient to create a startup file that sets these variables according to your preferences; this eliminates the need to set variables each time you enter **mailx**.

Many of the following variables represent on-off options. If you set the variable itself (to any value), the option is turned on. To turn the option off, you can unset the variable, or set a variable consisting of *no* followed by the name of the original variable. For example, setting *autoprint* turns the autoprint option on, and setting *noautoprint* turns it off.

allnet
> Assumes that network addresses with the same login component refer to the same person. Network addresses typically consist of several components, giving information that lets a mail server identify a

machine on the network, a route to that machine, and the login name of a user on that machine. **mailx** assumes that the login name is the last component. For example:

```
print name
```

displays all messages that originated from the same login name, regardless of the rest of the network address. The default is *noallnet,* where different addresses are assumed to be different users, even if the login name components are the same.

append
Appends messages to the end of the *mbox* file (your personal mailbox) upon termination. The default is *noappend*; messages are placed at the beginning of the *mbox* file instead of the end.

ask
Prompts you for a Subject: line when composing a message (if you have not already specified one with the *–s* option). This option is on by default; to turn it off, set *noask*. *ask* is the same as *asksub*. *noask* is the same as *noasksub*.

askbcc
Prompts you for a Bcc: list when composing a message. The default is *noaskbcc*; you are not prompted.

askcc
Prompts you for a Cc: list when composing a message. The default is *noaskcc*; you are not prompted.

asksub
Prompts you for a Subject: line when composing a message (if you have not already specified one with the **–s** option). This option is turned on by default; to turn it off, set *noasksub*. *asksub* is the same as *ask*. *noasksub* is the same as *noask*.

autoprint
Automatically displays the last message deleted with the **delete** subcommand or the last message undeleted with **undelete**. The default is *noautoprint*; you are not shown messages that you delete or undelete.

bang
Records shell commands run inside the **mailx** session (for example, through the **~!** input-mode command). Then, if you issue a shell command and the shell command contains a ! character, **mailx** replaces that character with the command line for the previous shell command. The default is *nobang,* in which case a ! in a shell command line is not treated specially.

cmd
Should contain a command, possibly with options. This specifies a default command line to be used for the command-mode **pipe** subcommand. For example:

```
set cmd="cat"
```

pipes messages through **cat** when the **pipe** subcommand is invoked.

crt
Contains an integer number. If a message has more than this number of lines, the message is piped through the command given by the **PAGER** variable, whenever the message is displayed. *crt* is not set; the default is *nocrt*.

DEAD
Contains the name of a file that can be used as the **dead.letter** file. Partial messages are saved in this file if an interrupt or error occurs during creation of the message or delivery. By default, the name of this file is *$HOME/***dead.letter**.

dot
Accepts a line consisting only of a dot (.) to indicate the end of a message in input mode. Thus . is equivalent to **~.**. The default is *nodot*. If *ignoreeof* is set, **mailx** ignores a setting of *nodot*; the period is the only way to end input mode.

EDITOR
> Gives a command, possibly with options, that is run when using the command mode **edit** or the input mode **~e**. The default is ed (see **ed**).

escape
> Gives the character used to begin input-mode subcommands. The default is the tilde (**~**). If this variable is set to null, **mailx** disables command escaping.

flipr
> Reverses the meanings of the **R** and **r** subcommands. The default is *noflipr*. See also *Replyall*.

folder
> Contains the name of a directory. This lets you specify a standard directory for saving mail files. Whenever you specify a file name for a **mailx** command, putting a plus sign (+) in front of the name specifies that the file is to be accessed in the *folder* directory.
>
> If the value of *folder* begins with a slash, it is taken as an absolute path name; otherwise, **mailx** assumes that the directory is directly under your **HOME** directory. *folder* has no default value. If you want to use + in file names that appear on the **mailx** command line itself (as opposed to commands in a **mailx** session), you must make *folder* an exported shell environment variable.

header
> Displays a summary of message headers at the beginning of a **mailx** command-mode session. This is the default.

hold
> Keeps all messages in your system *mailbox* instead of saving them in your personal *mbox*. The default is *nohold*.

ignore
> Ignores interrupts received while composing a message. The default is *noignore*.

ignoreeof
> Ignores end-of-file markers found while entering a message. The message can be ended by "**.**" or **~.** on a line by itself. The default is *noignoreeof*.

indent
> Contains a string that **mailx** uses as a prefix to each line in messages that **~m** and **~M** insert. The default is one tab character.

indentprefix
> As with *indent*, contains a string that **mailx** uses as a prefix to each line in messages that **~m** and **~M** insert. The default is one tab character. If both *indent* and *indentprefix* are set, *indentprefix* takes precedence.

keep
> Does not remove your system *mailbox* if the mailbox contains no messages. The mailbox is truncated to zero length—that is, it is merely emptied, although it still exists. If the default *nokeep* is in effect, empty mailboxes are removed.

keepsave
> Keeps messages in your system mailbox even if they have been saved in other files. The default, *nokeepsave*, deletes messages from the system *mailbox* if they have been saved elsewhere.

LISTER
> Contains a command, possibly with options. **mailx** invokes this command when displaying the contents of the *folder* directory for the **folders** subcommand. If this variable is null or unset, **mailx** uses **ls**. By default, this variable is unset.

MAILRC
> Is the location of personal startup file. See "Startup Files" on page 191.

MAILSERV
> Identifies the mail server being used for remote mail.

MBOX
Gives the name of your *mbox* (personal mailbox) file. Messages that have been read but not saved elsewhere are saved here when you run **quit** (but not when you run **exit**). The default is **$HOME/ mbox**.

metoo
When replying to a message with your login name in the recipient list, sends a reply to all other recipients, the author, and you. If *nometoo* is set, you are not to be sent the reply. The default is *nometoo*.

onehop
Attempts to send replies directly to the recipients instead of going through the original author's machine. When you reply to a message, your reply is sent to the author and to all recipients of the message. On a network, **mailx** normally specifies the recipient addresses so that all the replies go to the original author's machine first, and then on to the other recipients.

outfolder
Causes files used to record outgoing messages (see the description of *record*) to be located in the directory given by folder unless folder contains an absolute path name.

The default is *nooutfolder*.

page
Tells the **pipe** subcommand to insert a form-feed character after each message that it sends through the pipe. The default is *nopage*.

PAGER
Contains a command, possibly including options. **mailx** sends display output through this command if the output is longer than the screen length given by *screen*. The default value is cat (see **cat**).

prompt
Contains a string that **mailx** displays to prompt for output in command mode. The default is a question mark followed by a space (?).

quiet
Does not display the opening message and version number when **mailx** begins a session. The default is *noquiet*.

record
Contains a file name where every message you send is to be recorded. If *record* is not an absolute path name and the *outfolder* variable has not been set, the file is located in the **HOME** directory. If the *outfolder* variable is set, the file is located in your *folder* directory. The default is *norecord*.

Replyall
Reverses the senses of the **reply** and **Reply** subcommands (so that **reply** replies only to the author of a message, and **Reply** replies to the author and all other recipients). See also *flipr*.

save
Saves messages in your **dead.letter** file if they are interrupted while being composed. The name of your **dead.letter** file is given by the **DEAD** variable. Setting *nosave* disables this automatic save feature. The default is *save*.

screen
Gives the number of headers that are to be displayed by the **headers** and **z** subcommands.

sendmail
Contains a command, possibly with options, that **mailx** invokes to send mail. The default is **mail**. It can be any command that takes addresses on the command line and message contents on standard input.

sendwait
When sending a message through a network, **mailx** waits for the mail server to finish before returning to your session. Normally, it just submits the message to the server and then returns immediately. The default is *nosendwait*.

SHELL

Contains a command, possibly with options. **mailx** assumes that this command is a command interpreter. **mailx** invokes this command interpreter whenever it is asked to run a system command (for example, through the `!` command-mode command). The default is **sh** (see **sh**).

showto

When displaying a header summary, displays the recipient's name instead of the author's for messages where you are the author. The default is *noshowto*.

sign

Contains a string that is inserted into a message when you use the input mode **~a** subcommand. **mailx** interprets *n* and *t* in this string as the newline and tab characters, respectively. The default is *nosign*.

Sign

Contains a string that is inserted into a message when you use the input mode **~A** subcommand. The default is *noSign*.

TERM

Contains the name of the terminal type. If *screen* is not set, **TERM** individually determines the number of lines in a screenful of headers.

toplines

Gives the number of header lines that the **top** subcommand is to display. The default is 5.

VISUAL

Contains a command, possibly with options, that **mailx** invokes when using the command-mode **visual** subcommand or the input mode **~v** subcommand.

Files

/etc/mailx.rc

Systemwide startup file.

$MAILRC

Personal startup file. By default, **MAILRC** has the value **$HOME/.mailrc**.

$HOME/mbox

Default location to save read messages. You can choose a different file by assigning the file name to the environment variable **MBOX.**

$MAILDIR

Directory containing system mailboxes. By default, this is **/usr/mail**. The system programmer must create the **MAILDIR** directory if it does not already exist.

$HOME/dead.letter

Default location to save partial letters.

Localization

mailx uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

Successful sending. (However, this does not guarantee that the mail was successfully received). 0 is also returned if **–e** is specified and there is no new mail. 0 is returned if there is new or unread mail. 1 means that there is no new or unread mail.

1

Returned if **–e** is specified and there is new mail. Also returned to indicate failure because of any of the following:

- There is no mail to read.
- Inability to create temporary file name or temporary file.
- Receipt of user interrupt while message was being composed.
- Inability to determine the user's identity.

2

Failure due to any of the following:

- Missing *number* after **–h**
- Missing *address* after **–r**
- Missing *subject* after **–s**
- Missing *user* after **–u**
- Incorrect command-line option
- Use of interactive options when not using command interactively

Portability

POSIX.2, X/Open Portability Guide, UNIX system V.

UNIX System V has a compatible **mailx** utility, whereas Berkeley Software Distribution (BSD) has a similar utility, known as **Mail**.

The **–F**, **–r**, and **–U** options; the **Copy**, **echo**, **followup**, **Followup**, **Save**, **Unread**, and **version** commands; and the *allnet*, *conv*, **MAILSERV**, *onehop*, *replyall*, *sendmail*, and *sendwait* variables are extensions of the POSIX standard.

Related Commands

echo, **ed**, **sh**

make — Maintain program-generated and interdependent files

```
make [−EeinpqrstuVvx] [−k|−S] [−c dir] [−f file] ...
    [macro definition ...] [−D macro definition ...] [target ...]
```

Purpose

make helps you manage projects containing a set of interdependent files, such as a program with many source and object files, or a document built from source files, macro files, and so on. **make** keeps all such files up to date with one another. If one file changes, **make** updates all the other files that depend on the changed file.

Options

−c *dir*
> Attempts to change into the specified directory when **make** starts up. If **make** can't change to the directory, an error message is printed. This is useful for recursive makefiles when building in a different directory.

−D *macro definition*
> Define *macro* on the command line before reading any *makefile.* Use the same form as a normal macro definition (*macro=string*). If you use this option, **make** assigns the value to the macro before reading the makefile; any definition of the same macro contained in the makefile supersedes this definition.
>
> **Note: make** uses any macros defined in this way before reading any makefile, including the startup file. This allows you to define a startup file by providing a value for **MAKESTARTUP** on the command line:
>
> ```
> make -D MAKESTARTUP=$HOME/project/startup.mk
> ```

−E
> Suppresses reading of the environment. If neither **−E** nor **−e** is specified, **make** reads the environment before reading the makefile.

−e
> Reads the environment after reading the makefile. If neither **−E** nor **−e** are specified, **make** reads the environment before reading the makefile, except for the **SHELL** environment variable, which you must explicitly export. This option does not affect the value of **MAKEFLAGS**.

−f *file*
> Uses *file* as the source for the makefile description. **make** ignores the makefiles specified as prerequisites to the `.MAKEFILES` target. You can use more than one **−f** option. If you specify *file* as a dash (—), **make** reads from standard input.

−i
> Tells **make** to ignore all errors and continue making other targets. This is equivalent to the `.IGNORE` attribute or macro.

−k
> Makes all independent targets, even if an error occurs. Specifying **−k** tells **make** to ignore the error and continue to make as much as possible. **make** does not attempt to update anything that depends on the target that was being made when the error occurred.

−n
> Prints out the commands that **make** would run to update the chosen targets, but does not actually run the commands. However, a command line (associated with the target), with a plus-sign prefix shall be executed. If **make** finds the string `$(MAKE)` in a recipe line, it expands it, adds **−n** to the MAKEFLAGS, and then runs the recipe line. This allows you to see what recursive calls to **make** do. This feature is

disabled inside group recipes. The output correctly shows line breaks in recipes that are divided into several lines of text using the \\<newline> sequence.

–p

Prints the digested makefile, including macro and target definitions. This display has a human-readable form that is useful for debugging, but cannot be used as input to **make**.

–q

Checks whether the target is up to date. If it is up to date, **make** exits with a status of 0; otherwise, it exits with a status of 1 (typically interpreted as an error by other software). When you specify **–q**, **make** does not run any commands unless they have a plus sign (+) prefix.

–r

Does not read the default rules from **/etc/startup.mk**.

–S

Ends **make** if an error occurs during operations to bring a target up to date (opposite of **–k**). This is the default.

–s

Tells **make** to do all its work silently. **make** displays neither the commands it runs nor warning messages. This is equivalent to the .SILENT attribute or macro.

–t

Touches the targets to mark them as up to date, but does not actually run commands to change the targets unless the target has a plus sign (+) prefix. **make** does not touch targets that are already up to date or targets that have prerequisites but do not have recipes. **make** displays a message for each target file, indicating the file name and the fact that it was touched.

–u

Forces an unconditional update: **make** behaves as if all the prerequisites of the given target are out of date.

–V

Prints the version number of **make** and a list of built-in rules.

–v

Causes **make** to display a detailed account of its progress. This includes what files it reads, the definition and redefinition of each macro, metarule and suffix rule searches, and other information.

–x

Exports all macro definitions to the environment. This happens just before **make** begins making targets (but after it has read the entire makefile).

Targets

A *target* is normally a file that you want to ensure is up to date with the files on which it is dependent (the prerequisites). **make** updates all targets that are specified on the command line. If you do not specify any target, **make** updates the targets in the first rule of the makefile. A target is out of date if it is older than any of its prerequisites (based on modification times) or if it does not exist.

To update a target, **make** first recursively ensures that all the target's prerequisites are up to date, processing them in the order in which they appear in the rule. If the target itself is out of date, **make** then runs the recipe associated with the target. If the target has no associated recipe, **make** considers it up to date.

make also supports another form of targets, known as *special targets,* described in .

Macros

Macro definitions can appear on the command line or in makefiles. The user must specify the **–D** option to override define macros used in command line prerequisites. Macro definitions on the command line may not have any white space between the macro name and the = character.

Macro definitions may take several forms.

```
macro = string
```

is the usual form. If *string* contains macro references, **make** does not expand them when the macro is defined, but when the macro is actually used.

```
macro := string
```

expands macros inside *string* before creating *macro*.

```
macro += string
```

adds *string* to the previous value of *macro*.

You can use any amount of white space on both sides of macro operators. **make** defines the name *macro* to have the value *string* and replaces it with that value whenever it is used as $(*macro*) or ${*macro*} within the makefile. It is possible to specify a $(*macro_name*) or ${*macro_name*} macro expansion, where *macro_name* contains more $(...) or ${...} macro expansions itself.

Normally, **make** does not include white space at the beginning and end of *string* in the definition of *macro*; however, it never strips white space from macros imported from the environment.

If you want to include white space in a macro definition specified on the **make** command line, you must enclose the definition in quotes.

make resolves macro definitions in the following order:

1. Macro definitions in the built-in inference rules
2. Contents of the environment
3. Macro definitions in the makefiles (in the order they appear)
4. Macro definitions on the command line

Definitions for macros in the prerequisite portion of a dependency line cannot be replaced by macro definitions from the command line. Prerequisite macros are expanded as they are read, but command line macro definitions are not applied to macros in the makefile until the entire file has been read. Therefore, with the exception of macros in the prerequisite of a dependency line, if a macro is already defined when **make** encounters a new definition for it, the new definition replaces the old one. For example, a macro definition for *name* on the command line overrides a definition for *name* in the makefile.

make supports macro expansions of the form:

```
$(macro_name:modifier_list:modifier_list:...)
```

Possible modifiers are:

^"string"
    Prefix tokens

+"*string*"
    Suffix tokens

b
    File portion of all path names, without suffix

d
    Directory portion of all path names

f
    File portion of all path names, including suffix

l
    All characters mapped to lowercase

s/*pat*/*string*/
    Simple pattern substitution

suffix=*string*
    Suffix replacement

t"*separator*"
> Tokenization

u
> All characters mapped to uppercase

For example, with:

```
test = D1/D2/d3/a.out f.out d1/k.out
```

we have:

```
$(test:d)           → D1/D2/d3 . d1
$(test:b)           → a f k
$(test:f)           → a.out f.out k.out
${test:db}          → D1/D2/d3/a f d1/k
${test:s/out/in}    → D1/D2/d3/a.in f.in d1/k.in
$(test:f:t"+")      → a.out+f.out+k.out
$(test:t"+")        → D1/D2/d3/a.out+f.out+d1/k.out
$(test:u)           → D1/D2/D3/A.OUT F.OUT D1/K.OUT
$(test:l)           → d1/d2/d3/a.out f.out d1/k.out
$(test:^"/rd/")     → /rd/D1/D2/d3/a.out /rd/f.out /rd/d1/k.out
$(test:+".Z")       → D1/D2/d3/a.out.Z f.out.Z d1/k.out.Z
```

Runtime macros can take on different values for each target.

$@
> The full target name. When building a normal target, this macro evaluates to the full name of the target. When building a library, it expands to the name of the archive library. For example, if the target is:
>
> ```
> mylib(member)
> ```
>
> $@ expands to:
>
> ```
> mylib.
> ```

$%
> The full target name. When building a normal target, this macro evaluates to the full name of the target. When building a library, it expands to the name of the archive member. For example, if the target is:
>
> ```
> mylib(member)
> ```
>
> $% expands to:
>
> ```
> member
> ```

$&
> The list of all prerequisites.

$?
> The list of all prerequisites that are newer than the target.

$^
> The list of all prerequisites taken from the list specified on the rule line of the recipe where the $^ appears.

$<
> Same as $^.

$>
> The name of the library if the current target is a library member.

$*
> The target name with no suffix ($(%:db)) or the value of the stem in a meta-rule.

The constructs $$@, $$%, $$>, and $$* can appear in a prerequisite list as dynamic prerequisites. $$@ stands for the target currently being made. For example:

```
fred : $$@.c
fred : fred.c
```

are equivalent. The construct can be modified, as in:

```
fred.o : $$(@:b).c
```

The runtime macros can be modified by the letters D and F to indicate only the directory portion of the target name or only the file portion of the target name. (The working directory is represented by a dot.) If **define.h** is the only prerequisite that is newer than the target, the macros $?D and $?F expand to dot (.) and to define.h.

If you are building a library, $$% stands for the name of the archive member being made. If you are building a normal target, $$% stands for the name of the target currently being made.

$$* stands for the name of the current target being made, but with no suffix.

If you are building a library, $$> stands for the name of the archive library being made. If you are not building a library, $$> is not valid.

Comments

Comments begin with the pound (#) character and extend to the end of the line. **make** discards all comment text.

Makefile Contents

Inside makefiles, you can split long lines over several lines of text. To do this, put a backslash (\) at the very end of the line. You can use this technique to extend comments as well as recipe lines and macro definitions, for example.

If a rule or macro definition must contain a # character, use \#; otherwise, **make** mistakes the # for the beginning of a comment. Also, $$ stands for $.

File names that contain a colon must always be enclosed in quotes, as in:

```
"a:target" : "a:prereq"
```

Rules

The general format of a rule is:

```
targets [attributes] ruleop [prerequisites] [;recipe]
{<tab> recipe}
```

where the items enclosed in square brackets are optional. (This is just a documentation convention; you do not actually enter the square brackets.) The parts of the rule are described as follows:

targets
    One or more target names.

attributes
    A list, possibly empty, of attributes to apply to the list of targets.

ruleop
    A separator string that separates the target names from the prerequisite names and may also affect the processing of the specified targets.

prerequisites
    A list of zero or more names on which the specified targets depend.

recipe

> May follow on the same line as the prerequisites, separated from them by a semicolon. A recipe is a group of commands following a target, which specifies how to make that target. If a recipe is present, **make** takes it as the first in the list of recipe lines defining how to make the named targets. Additional recipe lines can follow the first line of the rule. Each such recipe line must begin with a tab character.

The possible rule operators are listed as follows:

targets : prereqs

> Is a simple rule definition. You can specify only one set of rules for making a target, except within metarules. In metarules, you can specify more than one recipe for making the target. If a target has more than one associated metarule, **make** uses the first metarule that matches.

targets :! prereqs

> Executes the recipe for the associated targets once for each recently changed prerequisite. Ordinarily, **make** runs the recipe only once, for all out-of-date prerequisites at the same time.

targets :^ prereqs

> Inserts the specified prerequisites before any other prerequisites already associated with the specified targets.

targets :– prereqs

> Clears the previous list of prerequisites before adding the new prerequisites.

targets :: prereqs

> Is used for multiple rules applying to the same targets. Each rule can specify a different set of prerequisites with a different recipe for updating the target. If a target is out of date with respect to any of its prerequisites, **make** remakes the target using all the recipe lines associated with the rules that mention those prerequisites.

targets :| prereqs

> Is used in metarules. It tells **make** to treat each metadependency as an independent rule. For example:

```
%.o :| %.c rcs/%.c /srcarc/rcs/%.c
        recipe...
```

> is equivalent to:

```
%.o : rcs/%.c
        recipe...
%.o : /srcarc/rcs/%.c
        recipe...
```

You can follow the first line of a rule with any number of recipe lines. Each of these must begin with a tab character. The method of entering tab characters using XEDIT is discussed in *z/VM: OpenExtensions User's Guide*.

You can follow the tab with –, @, + or all three. – indicates that **make** is to ignore nonzero exit values when it runs this recipe line. @ indicates that **make** is *not* to display the recipe line before running it. + tells **make** to always run this line, even when **–n**, **–p**, or **–t** is specified. This is particularly useful when calling **make** recursively. If the recursive **make** line is preceded by a +:

```
make -n
```

runs the recursive **make** but puts the n in the **MAKEFLAGS** variable. This allows you to see what the subsidiary **make**s do. You can use a target that has prerequisites but no recipes to add the given prerequisites to that target's list of prerequisites.

Group recipes begin with [in the first non-white-space position of a line, and end with] in the first non-white-space position of a line. Recipe lines in a group recipe need not have a leading tab. **make** executes a group recipe by feeding it as a single unit to a shell. If you immediately follow the [at the beginning of a group recipe with one of –, @ or +, they apply to the entire group in the same way that they apply to single recipe lines.

Inference Rules

With inference rules you can specify general rules for building files rather than creating a specific rule for each target.

make provides two forms of inference rules: suffix rules and metarules. It provides suffix rules for compatibility with older makefiles. Metarules are a more general technique than suffix rules for specifying **make**'s default behavior. They provide a superset of the utility of suffix rules.

make uses the inference rules to infer how it can bring a target up to date. A list of inference rules defines the commands to be run. The default **startup.mk** file contains a set of inference rules for the most common targets. You can specify additional rules in the makefile.

When **make** finds no explicit target rule to update a target, it checks the inference rules. If **make** finds an applicable inference rule with an out-of-date prerequisite, it runs that rule's recipe. (See also "Special Targets" on page 207 which describes the .DEFAULT special target).

Suffix Rules

make treats targets that begin with a period and contain no slashes or percent signs as suffix rules. If there is only one period in the target, it is a single suffix inference rule. Targets with two periods are double-suffix inference rules. Suffix rules do not have prerequisites but do have commands associated with them.

When **make** finds no explicit rule to update a target, it checks the suffix of the target (*.s1*) to be built against the suffix rules. **make** examines a prerequisite based on the basename of the target with the second suffix (*.s2*) appended, and if the target is out of date with respect to this prerequisite, **make** runs the recipe for that inference rule.

Metarules take precedence over suffix rules.

If the target to be built does not contain a suffix and there is no rule for the target, **make** checks the single suffix inference rules. The single suffix inference rules define how to build a target if **make** finds a rule with one of the single suffixes appended. A rule with one suffix *.s2* defines how to build *target* from *target.s2*. **make** treats the other suffix (*.s1*) as null.

For a suffix rule to work, the component suffixes must appear in the prerequisite list of the .*SUFFIXES* special target. You can turn off suffix rules by placing the following in your makefile:

```
.SUFFIXES:
```

This clears the prerequisites of the .SUFFIXES target, which prevents suffix rules from being enacted. The order that the suffixes appear in the .SUFFIXES rule determines the order in which **make** checks the suffix rules.

The following steps describe the search algorithm for suffix rules:

1. Extract the suffix from the target.
2. Is it in the .SUFFIXES list? If not, quit the search.
3. If it is in the .SUFFIXES list, look for a double suffix rule that matches the target suffix.
4. If there is a match, extract the base name of the file, add on the second suffix, and determine if the resulting file exists. If the resulting file does not exist, keep searching the double suffix rules.

   If the resulting file does exist, use the recipe for this rule.
5. If a successful match is not made, the inference has failed.
6. If the target did not have a suffix, check the single suffix rules in the order that the suffixes are specified in the .SUFFIXES target.
7. For each single suffix rule, add the suffix to the target name and determine if the resulting file name exists.

8. If the file name exists, execute the recipe associated with that suffix rule. If the file name doesn't exist, continue trying the rest of the single suffix rules. If a successful match is not made, the inference has failed.

make also provides a special feature in the suffix rule mechanism for archive library handling. If you specify a suffix rule of the form:

```
:a.suff:
        recipe
```

the rule matches any target having the LIBRARYM attribute set, regardless of what the actual suffix was. For example, if your makefile contains the rules:

```
.SUFFIXES:  .a .o
:a.o  :
      echo adding $< to library $@
```

then if mem.o exists:

```
make "mylib(mem.o)"
```

causes:

```
adding mem.o to library mylib
```

to be printed.

Metarules

Metarules have one target with a single percent symbol that matches an arbitrary string called the *stem*; *A%B* matches any string that starts with prefix *A* and ends with suffix *B*. *A* or *B* or both may be null. The % in a dependency stands for the stem.

The inference rule to update a target matching pattern *p1%s1*, where *p1* and *s1* are prefix and suffix strings of the target, having a prerequisite *p2%s2*, where % is the stem from the target, is specified as a rule:

```
p1%s1 : p2%s2 ; recipe....
```

Either the prefix or suffix string may be empty.

With the internal macros you can specify general inference rules. If the target is out of date with respect to this prerequisite, **make** runs that inference rule's recipe.

Transitive Closure

Metarules provide a mechanism that allows several metarules to chain together to eventually create the target.

This is called *transitive closure*. For example, if you have metarules:

```
%.o : %.c
        ... rule body...
```

and:

```
%.c : %.y
        ... rule body ...
```

When you specify:

```
make file.o
```

make uses the first metarule to look for **file.c**. If it can't find an explicit rule to build **file.c**, it again looks through the metarules and finds the rule that tells it to look for **file.y**.

make allows each metarule to be applied only once when performing transitive closure to avoid a situation where it loops forever. (For example, if you have the rule:

```
% : %.c
        ... rule body ...
```

the command:

```
make file
```

causes **make** to look for **file.c**. If the metarules were not restricted and **file.c** did not exist, then **make** would look for **file.c.c**, and then **file.c.c.c**, and so on. Because each metarule is applied only once, this can't happen.)

Transitive closure is computed once for each metarule head the first time the pattern matches a target. When transitive closure is computed, all the computed rules are added to the rule set for that metarule head. For example, if you have the rules:

```
% : %.o
        recipe 1...
%.o : %c
        recipe 2...
```

and you are making *file*, this target matches successfully against % causing transitive closure to be computed for %. As a result of this computation, a new rule is created:

```
% : %.c
    recipe 2...
    recipe from .REMOVE target for %.o, if not .PRECIOUS
    recipe 1...
```

which is executed if **file.o** doesn't exist. When the computation for the rule head has been done, it is marked as *transitive closure computed*. Since all possible new rules have been added to the rule set the first time the computation is done, it is not necessary to do it again: Nothing new is added. The term *transitive closure* is adapted from the mathematical set theory.

Note: In set theory, if you have a set composed of pairs (*a,b*) and (*b,c*), then the set would be transitively closed if (*a,c*) is also in the set.

The best way to understand how this works is to experiment with little **make** files with the **–v** flag specified. This shows you in detail what rules are being searched, when transitive closure is calculated, and what rules are added.

Attributes

make defines several target attributes. Attributes can be assigned to a single target, a group of targets, or to all targets in the makefile. Attributes affect what **make** does when it needs to update a target. You can associate attributes with targets by specifying a rule of the form:

```
attribute_list : targets
```

This assigns the attributes in *attribute_list* to the given targets. If you do not specify any targets, the attributes apply to every target in the makefile. You can also put attributes inside a normal rule, as in:

```
targets attribute_list : prerequisites
```

The recognized attributes are:

.EPILOG
　　Insert shell epilogue code when running a group recipe associated with any target having this attribute set.

.IGNORE
　　Ignore an error when trying to make any target with this attribute set.

.LIBRARY

Target is a library.

.LIBRARYM

Target is a library member (cannot be set by the user).

.PRECIOUS

Do not remove this target under any circumstances. Any automatically inferred prerequisite inherits this attribute.

.PROLOG

Insert shell prolog code when running a group recipe associated with any target having this attribute set.

.SETDIR

Change the working directory to a specified directory when making associated targets. The syntax of this attribute is `.SETDIR=`*path*, where *path* is the path name of desired working directory. If *path* contains any `:` characters, the entire attribute string must be quoted, not just the path name.

.SILENT

Do not echo the recipe lines when making any target with this attribute set, and do not issue any warnings.

.SYMBOL

Target is an entry point into a module in a library (it cannot be set by the user). This attribute is used only when searching a library for a target. Targets of the form `lib((entry))` have this attribute set automatically.

You can specify any attribute except `.LIBRARYM` and `.SYMBOL`. You can use any attribute with any target, including special targets.

Special Targets

Special targets are called targets because they appear in the target position of rules; however, they are really keywords, not targets. The rules they appear in are really *directives* that control the behavior of **make**.

The special target must be the only target in a special rule; you cannot list other normal or special targets.

Some special targets are affected by some attributes. Any special target can be given any attribute, but often the combination is meaningless and the attribute has no effect.

.BRACEEXPAND

This target may have no prerequisites and no recipes associated with it. If set, the target enables the outdated brace expansion feature used in older versions of **make**. Older **make**s would expand a construct of the following form, beginning with each token in the token list:

```
string1{token_list}string2
```

Older **make**s would append *string1* to the front of each token in the list, and *string2* to the end of each token in the list. A more productive means for achieving the same result with modern versions of **make** relies on macro expansion with prefix and suffix modifiers:

```
$ (TOKEN_BASE:¬"prefix:+"suffix")
```

The double quotation marks are required. Brace expansion is an outdated feature available in past versions of **make**.

.DEFAULT

This target has no prerequisites, but it does have a recipe. If **make** can apply no other rule to produce a target, it uses this rule if it has been defined.

.ERROR

make runs the recipe associated with this target whenever it detects an error condition.

.EXPORT
  All prerequisites associated with this target that correspond to macro names are exported to the environment at the point in the makefile at which this target appears.

.GROUPEPILOG
  make adds the recipe associated with this target after any group recipe for a target that has the `.EPILOG` attribute.

.GROUPPROLOG
  make adds the recipe associated with this target after any group recipe for a target that has the `.PROLOG` attribute.

.IMPORT
  make searches in the environment for prerequisite names specified for this target and defines them as macros with their value taken from the environment. If the prerequisite `.EVERYTHING` is given, **make** reads in the entire environment (see **–e** and **–E** options).

.INCLUDE
  make parses another makefile just as if it had been located at the point of the `.INCLUDE` in the current makefile. The list of prerequisites gives the list of makefiles to read.

.INCLUDEDIRS
  The list of prerequisites specified for this target defines the set of directories to search when including a makefile.

.MAKEFILES
  The list of prerequisites is the set of files to try to read as the user makefile. These files are made in the order they are specified (from left to right) until one is found to be up to date. This is the file that is used.

.POSIX
  make processes the makefile as specified in the POSIX.2 draft standard. This target may have no prerequisite and no recipes associated with it. This special target must appear before the first non-comment line in the makefile. If this special target is present, the following facilities are disabled:

  • All recipe lines are run by the shell, one shell per line, regardless of the setting of *SHELLMETAS*.

  • Metarule inferencing is disabled.

  • Conditionals are disabled.

  • Dynamic prerequisites are disabled.

  • Group recipes are disabled.

  • Disables brace expansion (set with the *.BRACEEXPAND* special target).

  • **make** does not check for the string *$ (MAKE)* when run with the *–n* options specified.

.REMOVE
  make uses the recipe of this target to remove any intermediate files that it creates if an error is encountered before the final target is created. This does not remove files marked `.PRECIOUS` or files that existed before **make** began execution.

.SOURCE
  The prerequisite list of this target defines a set of directories to check when trying to locate a target file name.

.SOURCE.x
  Same as `.SOURCE`, except that **make** searches the `.SOURCE.x` list first when trying to locate a file matching a target with a name that ends in the suffix **.x**.

.SUFFIXES
  The prerequisite list of this target defines a set of suffixes to use when trying to infer a prerequisite for making a target.

A name of the form *library(member)* indicates a member of a library. The *library* portion is a target with the `.LIBRARY` attribute, and the *member* portion is a prerequisite of the library target.

A name of the form *library((entry))* indicates the library module that contains the given entry point. Once again, the library portion is a target with the `.LIBRARY` attribute. **make** regards the library member that contains the entry point *entry* as a prerequisite of the library target.

Control Macros

make defines a number of control macros that control **make**'s behavior. When there are several ways of doing the same thing, control macros are usually the best. A control macro that has the same function as a special target or attribute also has the same name.

Macros that are said to be *defined internally* are automatically created by **make** and can be used with the usual $(*name*) construct. For example, $(PWD) can be used to obtain the current directory name.

Recognized control macros are:

DIRSEPSTR
    Contains the characters used to separate parts in a path name and can be set by the user. **make** uses the first character in this string to build path names when necessary.

.EPILOG
    If assigned a non-null value, the `.EPILOG` attribute is given to every target.

GROUPFLAGS
    Specifies option flags to pass to GROUPSHELL when **make** invokes it to run a group recipe.

GROUPSHELL
    Gives the path name of the command interpreter (shell) that **make** calls to process group recipes.

GROUPSUFFIX
    Specifies a string for **make** to use as a suffix when creating group recipe files to be run by the command interpreter.

.IGNORE
    If this is assigned a non-null value, **make** assigns the `.IGNORE` attribute to every target.

INCDEPTH
    This is the current depth of makefile inclusion. It is set internally.

MAKE
    This is set by the startup file and can be changed by the user. The standard startup file defines it as:

```
$(MAKECMD) $(MFLAGS)
```

    The MAKE macro is not used by **make** itself, but the string $(MAKE) is recognized when using the **–n** option for single-line recipes.

MAKECMD
    This is the name with which **make** was invoked.

MAKEDIR
    This is the full path name of the initial directory in which **make** began execution.

MAKEFLAGS
    The MAKEFLAGS macro contains all the options (flags) and macros specified in the **MAKEFLAGS** environment variable plus all of the options and macros specified on the command line, with the following exceptions. Specifying **–c**, **–f**, or **–p** in the environment variable results in an error, and these same options specified on the command line do not appear in the MAKEFLAGS macro. Options in the **MAKEFLAGS** environment variable may have optional leading dashes and spaces separating the options. These are stripped out when the MAKEFLAGS macro is constructed.

    Note: make always reads the **MAKEFLAGS** environment variable before reading the makefile. The **–E** and **–e** options do not affect this.

MAKESTARTUP
    This has the default value:

    $(ROOTDIR)/etc/startup.mk

To change where **make** looks for its startup file, you can set the environment variable **MAKESTARTUP** before running **make**. Since **make** processes command-line macros after reading the startup file, setting this macro on the command line does not have the desired effect.

MFLAGS
This is the same as MAKEFLAGS, except that it includes the leading switch character.

NULL
This is permanently defined to be the NULL string.

.PRECIOUS
If this is assigned a non-null value, **make** assigns the .PRECIOUS attribute to every target.

.PROLOG
If this is assigned a non-null value, **make** assigns the .PROLOG attribute to every target.

PWD
This is the full path name of the working directory in which **make** is executing.

SHELL
Specifies the full path name of the command interpreter that **make** calls to process single-line recipes, when necessary. **make** passes recipe lines to this shell only if they contain one or more of the characters given in SHELLMETAS; otherwise, it runs them directly. By default, the value of the **SHELL** environment variable does not affect the value of this macro; however, you can use the *.IMPORT* special target to assign the environment variable's value to this macro. You can also use the *EXPORT* special target to assign this macro's value to the **SHELL** environment variable.

SHELLFLAGS
Specifies option flags to pass to the shell when invoking it to run a single-line recipe.

SHELLMETAS
Specifies a list of metacharacters that can appear in single recipe lines. If **make** finds any metacharacter, it invokes the recipe using the shell specified by SHELL; otherwise, it runs the recipe without the shell.

.SILENT
If this is assigned a non-null value, **make** assigns the .SILENT attribute to every target.

Making Libraries

A library is a file containing a collection of object files. To make a library, you specify it as a target with the .LIBRARY attribute and list its prerequisites. The prerequisites should be the object members that are to go into the library. When **make** makes the library target, it assigns the .LIBRARYM attribute to the prerequisites. This tells the file search mechanism to look for the member in the library if it cannot find an appropriate object file.

make tries to handle the old library construct format in a sensible way. When it finds *lib(member)*, it declares the *lib* portion as a target with the .LIBRARY attribute and the *member* portion as a prerequisite of the *lib* target. To make the library properly, old makefile scripts using this format must name the *lib* as a target and must try to bring it up to date. The same thing happens for any target of the form *lib((entry))*. These targets have an additional feature in that the *entry* target has the .SYMBOL attribute set automatically.

Conditionals

You specify the conditional expression as follows:

```
.IF expression
... if text ...
.ELSE
... else text ...
.END
```

or:

```
.IF expression
... if text ...
.ELSIF expression2
... elsif text ...
.ELSE
... else text ...
.END
```

The `.ELSE` or `.ELSIF` portion is optional, and you can nest the conditionals (that is, the text may contain another conditional). The `.IF`, `.ELSE`, `.ELSIF`, and `.END` conditionals must start in the first column of the line. *expression* or *expression2* can have one of three forms:

```
string
```

is true if the given string is non-NULL,

```
string == string
```

is true if the two strings are equal, and:

```
string != string
```

is true if the two strings are not equal. Typically, one or both strings contain macros, which **make** expands before making comparisons. **make** also discards white space at the start and end of the text portion before the comparison. This means that a macro that expands to nothing but white space is considered a NULL value for the purpose of the comparison. If a macro expression needs to be compared with a NULL string, compare it to the value of the macro `$(NULL)`.

The text enclosed in the conditional construct must have the same format that it would have outside the conditional. In particular, **make** assumes that anything that starts with a tab inside the conditional is a recipe line. This means that you cannot use tabs to indent text inside the conditional (except, of course, for recipe lines, which always begin with tabs).

Files

/etc/startup.mk
> The default startup file containing default rules.

Environment Variables

MAKEFLAGS
> Contains a series of **make** options that are used as the default options for any **make** command. You can specify the options with or without leading minus signs (-) and blanks between them. It can also include macro definitions of the form usually found on the command line.

MAKESTARTUP
> Contains the path name of the **make** stamp file. By default, **make** uses the file **/etc/startup.mk** as its startup file. To use a different file, set this environment variable before running **make**.

SHELL
> Contains a name of a command interpreter. To assign this value to the **SHELL** control macro, use the `.IMPORT` special target. You can also use the `.EXPORT` special target to assign the value of the **SHELL** macro to the environment variable.

Localization

make uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
Successful completion

1
Returned if you specified **–q** and file is not up to date

2
Failure due to any of the following:

- Unknown command-line option
- Missing argument to option, such as no file name for **–f**.

126
Recipe command was not executable.

127
Recipe command was not found.

255
Failure because of any of the following:

- Macro cannot be redefined
- Macro variables not assigned with `:=`
- Special target cannot be a prerequisite
- No file name for **–f**
- Too many makefiles specified
- Configuration file not found
- No makefile present
- Missing `.END` for `.IF`
- No target
- Inability to return to directory
- Too many open files
- Open failed
- File not found
- Closing file in slot
- Inability to change directory
- No more memory
- Line too long
- Circular macro detected
- Unterminated pattern string
- Unterminated replacement string
- Token separator string not quoted
- Unterminated separator string
- Expansion too long
- Suffix too long
- Unmatched quote
- `.IF .ELSE ... .END` nesting too deep

- `.ELSE` without `.IF`
- Unmatched `.END`
- Inference rules resulting in circular dependency
- No macro name
- Write error on temp file
- Target not found, and cannot be made
- Inability to make *NAME*
- <+ diversion unterminated
- <+ diversion cannot be nested
- <+ missing before +>
- Incomplete rule recipe group detected
- Inability to mix single and group recipe lines
- Unmatched] found
- Macro or rule definition expected but not found
- Name too long
- Inability to determine working directory
- Only one *NAME* attribute allowed in rule line
- Multiple targets not allowed in % rules
- Special target must appear alone
- Duplicate entry in target list
- Syntax error in % rule, missing % target
- Duplicate entry in prerequisite list
- Missing targets or attributes in rule
- Multiply defined recipe for target
- Empty recipe for special target
- Imported macro *NAME* not found in environment
- No `.INCLUDE` file(s) specified
- Include file *NAME,* not found
- *NAME* ignored on special target
- Attributes possibly ignored
- Inability to find member defining `SYMBOL((`*NAME*`))`
- Incorrect library format
- Inability to touch library member
- `SHELL` macro not defined
- Too many arguments
- Inability to export *NAME*
- Inability to open *file*
- Circular dependency detected
- Inability to stat `/`
- Inability to stat `.`
- Inability to open `..`
- Read error in `..`
- Metarule too long: "*rule*"

Limits

No single makefile script line can be longer than 8192 characters. In some environments the length of an argument string is restricted.

Usage Notes

When the `.SETDIR` special target is used, **make** checks the file attributes of targets and prerequisites on every pass through a rule. This can significantly increase the number of system accesses.

Portability

POSIX.2, UNIX systems.

The following features of **make** are enhancements to POSIX.2:

- The options: **–c***dir*, **–E**, **–u**, **–V**, **–v**, and **–x**.
- The **–n** option has enhanced functionality not covered by the standard; for more information, see the **–n** option and the `POSIX` special target for **make**.
- The runtime macros: **$&**, **$^**, **$>**.
- The dynamic prerequisites: **$$%**, **$$>**, **$$***, **$$@**.
- All macro expansions.
- Macro assignments of the following form:

  ```
  macroname  :=  stringassigned
  macroname  +=  stringassigned
  ```

- Brace expansion.
- Backslash continuation.
- The quoting mechanism, as in the following example:

  ```
              "a:target" :  "a:prerequisite"
  ```

- All rule operators except the colon (:).
- Conditionals.
- Metarules.
- All **make** attributes *except* `.IGNORE, .PRECIOUS, .SILENT` (referred to in POSIX.2 as special targets).
- All **make** special targets *except* `.DEFAULT, .POSIX, .SUFFIXES` (referred to in POSIX.2 as special targets).
- All **make** macros *except* **SHELL** (referred to in POSIX.2 as control macros).

For More Information

S. I. Feldman, "Make—Program for Maintaining Computer Programs," *Software—Practice and Experience* 9 (no. 4, April 1979):225–65 [Bell Labs, Murray Hill, NJ]

mkdir — Make a directory

---

mkdir [**−p**] [**−m** *mode*] *directory ...*

Purpose

The **mkdir** command creates a new directory for each named *directory* argument. The default mode for a directory created by the **mkdir** command is 755:

    owner = rwx
    group = r-x
    other = r-x

mkdir supports the following options:

−m *mode*
    Lets you specify permissions for the directories. The *mode* argument can have the same value as the *mode* for **chmod**; see **chmod** for more details.

−p
    Creates intermediate directory components that don't already exist. For example, if one of the *directory* arguments is **dir/subdir/subsub** and **subdir** doesn't already exist, **mkdir** creates it.

Localization

mkdir uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
    Successful completion

1
    Failure due to any of the following:

- Missing *mode* after **−m**
- Incorrect *mode*
- Incorrect command-line option
- Missing *directory* name
- Inability to create the directory

Messages and Return Codes

Possible error messages include:

Path not found
    The preceding structure (parent directory) of the named *directory* does not exist.

Access denied
> The requested *directory* already exists or is otherwise inaccessible.

Cannot create directory
> Some other error occurred during creation of the directory.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

rm, **rmdir**

mkfifo — Make a FIFO special file

mkfifo [**–p**] [**–m** *mode*] *file ...*

Purpose

mkfifo creates one or more FIFO special files with the given names.

Options

mkfifo recognizes the following options:

–m *mode*
> Lets you specify file permissions for the files. The *mode* argument can have the same value as the *mode* argument for **chmod**; see **chmod** for more details.

–p
> Creates intermediate directory components that don't already exist. For example, if one of the *file* arguments is **dir/subdir/file** and **subdir** doesn't exist already, this option creates it. Directories are created with the *mode* u+rwx, which means read, write, and search permissions to the owner.

Localization

mkfifo uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
> Successful completion

1
> Failure due to any of the following:
> - A missing *mode* after **–m**
> - An incorrect *mode*
> - An incorrect command-line option
> - A missing file name
> - Inability to create the desired *file*

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–p** option is an extension of the POSIX standard.

Related Commands

chmod, **create**, **mkdir**, **mknod**

mknod — Make a FIFO or character special file

```
mknod pathname c major minor
mknod pathname p
```

Purpose

mknod creates a special file with the given path name.

Options

c
Indicates character special files (for example, printers and other devices).

major minor
Gives the major and minor device types.

The high-order 16 bits of *device_identifier* hold the device major number. The device major number corresponds to a device driver supporting a class of devices—for example, interactive terminals. The low-order 16 bits of *device_identifier* hold the device minor number. The device minor number corresponds to a specific device within the class of devices referred to by the device major number.

The device major numbers currently defined for use by OpenExtensions services are:

> 3 **/dev/tty**
> 4 **/dev/null**

For device major numbers 3 and 4, the device minor number is ignored.

Device types can be either octal or decimal numbers. The shell differentiates between octal and decimal as follows:

- Any number that starts with 0 is octal.
- Any number that starts with 0x is hex.
- Any number that does not start with 0x or 0 is decimal.

p
Creates a FIFO special file (that is, a named pipe).

Note: mknod can be used only by a superuser.

Exit Values

Possible exit status values are:

0
Successful completion

1
Failure due to any of the following:

- Inability to create the desired file
- Incorrect *major* or *minor* number

2
Failure for any of the following:

- Too few command-line arguments
- A missing *major* or *minor* device number

mknod

Portability

UNIX systems. Within POSIX, **mknod** has been superseded by **mkfifo** for pipes. The POSIX family of standards have not yet designed an alternative to **mknod** for special files.

Related Commands

mkfifo

mount — See the OPENVM MOUNT command

The mount shell command is not available. Use the OPENVM MOUNT command in place of the mount command. See "OPENVM MOUNT" on page 407.

mv — Rename or move a file or directory

```
mv [–fi] file1 file2
mv [–fi] file ... directory
mv [–Rrfi] directory1 directory2
```

Purpose

mv renames files or moves them to a different directory. If you specify multiple *file*s, the target (that is, the last path name on the command line) must be a directory. **mv** moves the files into that directory and gives them names that match the final components of the source path names. When you specify a single source *file* and the target is not a directory, **mv** moves the source to the new name, by a simple rename if possible.

If a destination file exists for which you do not have write permission, **mv** prompts with the name of the existing file. If you answer y or yes, it deletes the destination and then moves the source.

Note: mv can be used only by the file owner or a superuser. Any users can move a file; those users must be a member of *group* (that *group* must have write permission). The permission for *other* is write.

Options

mv accepts the following options:

–f

Does not ask if you want to overwrite an existing destination without write permission; it automatically behaves as if you answered yes. If you specify both *–f* and *–i*, **mv** uses the option that appears last on the command line.

–i

Asks you if you want to overwrite an existing file whether or not the file is read-only. If you specify both *–f* and *–i*, **mv** uses the option that appears last on the command line.

–R

Moves a directory and all its contents (files, subdirectories, files in subdirectories, and so on). For example:

```
mv -R dir1 dir2
```

moves the entire contents of **dir1** to **dir2/dir1**. **mv** creates any directories that it needs.

–r

Is identical to **–R**.

Localization

mv uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

    Successful completion

1

    Failure due to any of the following:

- The argument had a trailing / but was not a directory
- Inability to find file
- Inability to open input file for reading
- Inability to create or open output file for output
- Read error on an input file
- Write error on an output file
- Input and output files identical
- Inability to unlink input file
- Inability to rename input file
- Irrecoverable error when using the **–r** option, such as:
  - Inability to access a file
  - Inability to read a directory
  - Inability to remove a directory
  - Inability to create a directory
  - A target that is not a directory
  - Source and destination directories identical

2

    Failure due to any of the following:

- Incorrect command-line option
- Too few arguments on the command line
- A target that should be a directory but isn't
- No space left on target device
- Out of memory to hold the data to be copied
- Inability to create a directory to hold a target file

Messages and Return Codes

Possible error messages include:

cannot allocate target string
    mv has no space to hold the name of the target file. Try to free some memory to give **mv** more space.

***filename*?**
    You are attempting to move a file, but there is already an existing file with that target name. If you really want to write over the existing file, type y and press <Enter>. If you do not want to write over the existing file, type n and press <Enter>.

    Note: This message is a prompt that appears only when the **–i** option is used.

source *name* and target *name* are identical
    The source and the target are actually the same file (for example, because of links). In this case, **mv** does nothing.

unreadable directory *name*
> **mv** cannot read the specified directory—for example, because you do not have appropriate permissions.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–R** and **–r** options are extensions of the POSIX standard.

Related Commands

cp, **cpio**, **rm**

newgrp — Change to a new group

```
newgrp [–l] [group]
newgrp [–] [group]
```

Purpose

newgrp lets you change to a new group. You stay logged in and your working directory does not change, but access permissions are calculated according to your new real and effective group IDs. If an error occurs, it may force you to exit the shell and start the shell again.

newgrp does not change the value of exported shell variables, and all others are either set to their default or are unset.

If you did not specify any arguments on the command line, **newgrp** changes to the default group specified for your user ID in the system user database. It also sets the list of supplementary groups to that set in the group database of the system.

If you specify a *group*, **newgrp** changes your real and effective group ID to that group. If a group has a password, and you are specified as a member of that group in the system group database, you do not require a password; otherwise, you are prompted for a password. If the group has no password, you are permitted to change to that group only if you are a member of that group, as specified in the system group database.

If the supplementary group list also contains the new effective group ID, **newgrp** changes the effective group ID. If the supplementary group list does not contain the new effective group ID, **newgrp** adds it to the list (if there is room).

Options

–l
　　Starts the new shell session as a login session. This implies that it can run any shell profile code.

–
　　Is the obsolescent version of **–l**.

Localization

newgrp uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

If **newgrp** succeeds, its exit status is that of the shell. Otherwise, the exit status is:

>0
　　Failure because **newgrp** was unable to obtain the proper user or group information or because it was unable to run the shell, and it will end the current shell.

Portability

POSIX.2, UNIX systems.

Related Commands

export, **fc**, **sh**

nm — Display symbol table of object, library, or executable files

```
nm [–AaefgnoPprsuv] [–t format] file ...
```

Purpose

nm displays the symbol table associated with an object, archive library of objects, or executable files.

Note: nm does not recognize the format of CMS created modules or execs.

By default, nm lists the symbols in the file in alphabetical order by name and provides the following information on each:

- File or object name (if you specified –A)
- Symbol name
- Symbol type. Not all of these symbol types are available on all systems. For instance, not all systems support the ability to determine different segment information.

  A

  Absolute symbol, global

  a

  Absolute symbol, local

  B

  Uninitialized data (bss), global

  b

  Uninitialized data (bss), local

  D

  Initialized data (bbs), global

  d

  Initialized data (bbs), local

  F

  Filename

  l

  Line number entry (see the –a option)

  N

  No defined type, global. This is an unspecified type, compared to the undefined type U.

  n

  No defined type, local. This is an unspecified type, compared to the undefined type U.

  S

  Section symbol, global

  s

  Section symbol, local

  T

  Text symbol, global

  t

  Text symbol, local (static)

  U

  Undefined symbol

- Symbol value
- Symbol size, if applicable

Options

The format shows the main functions of **nm**, which are defined as follows:

–A
Prefixes each line with the filename or archive member.

–a
Displays all symbols, including line number entries on systems that support them.

–e
Displays only global (external) and static symbols.

–f
Displays full output. This is the default because output is not suppressed.

–g
Displays only global symbols.

–n
Is equivalent to –v.

–o
Displays output in octal (same as –t o).

–p
Displays output in a portable POSIX-compliant format, with blanks separating the output fields.

- If you specified –A and *file* is not a library, the format is:

```
file: name type value size.
```

- If you specified –A and *file* is a library, the format is:

```
file [object_file] : name type value size
```

where *object_file* is the object file in the library that contains the symbol being described.

- If you did not specify –A, the format is:

```
name type value size
```

- If you did not specify the –t option, nm displays *value* and *size* in hexadecimal.

- If you did not specify –A and the command line contains more than one file, or *file* is a library, nm displays a line preceding the list of symbols for each specified file or each object file in a specified library. If *file* is a library, this line has the following format:

```
file[object_file]:
```

If *file* is not a library, the format is:

```
file:
```

–p
Does not sort output.

–r
Reverses sort order.

–s
Includes symbol size for each symbol.

–t *format*
Defines the numeric value formatting base. The format is one of d, o, or x, for decimal, octal, or hexadecimal, respectively. If this option is not used, numbers are displayed in decimal.

–u
Displays only undefined symbols.

−v

    Sorts output by value.

−x

    Displays information in hexadecimal (same as −t x).

Localization

nm uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLECT**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**

Exit Values

Possible exit status values are:

0

    Successful completion

1

    Failure due to any of the following:

- Invalid command-line option
- Missing filename
- Unknown symbol table type
- Invalid library file
- End-of-file found in library
- Bad record in the library
- Out of memory

If a file does not contain a symbol table, nm displays a warning and goes to the next file, but this is not considered an error.

Portability

The −a, −e, −f, −n, −o, −p, −r, −s, and −x options are not part of the POSIX standard.

Related Commands

ar, **strip**

nohup — Start a process that is immune to hang-ups

---

nohup *command-line*

Purpose

nohup invokes a utility program using the given *command-line*. The utility runs normally; however, it ignores the **SIGHUP** signal.

If the standard output is a terminal, **nohup** appends the utility's output to a file named nohup.out in the working directory. This file is created if it doesn't already exist; if it can't be created in the working directory, it is created in your home directory.

If the standard error stream is a terminal, **nohup** redirects the utility's error output to the same file as the standard output.

nohup simply runs a program from an executable file. *command-line* cannot contain such special shell constructs as compound commands or pipelines; however, you can use **nohup** to invoke a version of the shell to run such a command line, as in:

```
nohup sh -c 'command'
```

where *command* can contain such constructs.

Environment Variables

HOME
    Contains the user's home directory

PATH
    Determines the search path that **nohup** uses when locating the command specified in *command-line*.

Localization

nohup uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

1
    Incorrect argument to **nohup**.

126
    nohup found the utility program but could not invoke it.

127
    An error occurred, or **nohup** could not find the utility program.

Otherwise, the exit status is the exit status of the utility program that is invoked.

Portability

POSIX.2, UNIX systems.

Related Commands

exec, **sh**

od -- Dump a file in a specified format

od [–v] [–A *addr_fmt*] [–j *num* [bkm]] [–N *num*] [–t *type_string*] [*file ...*]
od [–bcDdhOoSsXx] [*file*] [[+]*offset*[.][b]]

Purpose

od (octal dump) dumps a file to the standard output in a format specified by command-line options. The default format is octal words. You can use combinations of options to generate multiple formats with the requested representation of each byte vertically aligned. The file seek address (in octal) precedes each line of new data.

od recognizes two syntaxes. The first one is the POSIX-conforming form. If you choose the first form, **od** displays files from the list *file* one at a time. If no *file* appears on the command line, **od** reads the standard input.

Options

The first form of **od** accepts the following options:

–A *addr_fmt*
    Specifies the format that **od** uses to display the address field. *addr_fmt* can be **d** (decimal), **o** (octal), **x** (hexadecimal), or **n** (do not display address). The default is **–A o**.

–j *num*
    Skips *num* bytes from the beginning of the file. If you precede *num* with 0X or 0x, **od** interprets it as hexadecimal. If you precede it with 0, **od** interprets it as octal; otherwise, **od** assumes it is decimal. You can also append **b**, **k**, or **m** to *num* to indicate 512-byte blocks, kilobytes, or megabytes instead of bytes.

–N *num*
    Processes a maximum of *num* bytes.

–t *type_string*
    Specifies the output format. *type_string* can contain the following format characters:

    a
        Named characters from the ISO 646 character set (similar to the **-c** option).

    c
        Characters. **od** displays nonprintable characters as backslash sequences.

    d
        Signed decimal. A one-digit number may follow **d** telling **od** how many bytes to use. This must correspond to the size of a *char*, a *short*, an *int*, or a *long*. The default size is the size of an *int*. A symbolic size character can follow **d**, rather than the number of bytes. These have the following meaning:

        C
            Corresponds to number of bytes in a *char*

        S
            Corresponds to number of bytes in a *short int*

        I
            Corresponds to the number of bytes in an *int*

        L
            Corresponds to the number of bytes in a *long int*

f

Floating point. A one-digit number can follow **f**, telling **od** how many bytes to use. This must correspond to the size of a *float*, *double*, or *long double*. The default size is the size of a *double*. A symbolic size character can follow **f**, rather than the number of bytes. These have the following meaning:

F

Corresponds to size of *float*

D

Corresponds to size of *double*

L

Corresponds to size of *long double*

o

Octal. A one-digit number can follow **o**, telling **od** how many bytes to use. This must correspond to the size of a *char*, a *short*, an *int*, or a *long*. The default size is the size of an *int*. A symbolic size character can follow **o**, rather than the number of bytes. These have the following meaning:

C

Corresponds to number of bytes in a *char*

S

Corresponds to number of bytes in a *short int*

I

Corresponds to the number of bytes in an *int*

L

Corresponds to the number of bytes in a *long int*

u

Unsigned decimal. A one-digit number can follow **u**, telling **od** how many bytes to use. This must correspond to the size of a *char*, a *short*, an *int*, or a *long*. The default size is the size of an *int*. A symbolic size character can follow **u**, rather than the number of bytes. These have the following meaning:

C

Corresponds to number of bytes in a *char*

S

Corresponds to number of bytes in a *short int*

I

Corresponds to the number of bytes in an *int*

L

Corresponds to the number of bytes in a *long int*

x

Hexadecimal. A one-digit number can follow **x**, telling **od** how many bytes to use. This must correspond to the size of a *char*, a *short*, an *int*, or a *long*. The default size is the size of an *int*. A symbolic size character can follow **x**, rather than the number of bytes. These have the following meaning:

C

Corresponds to number of bytes in a *char*

S

Corresponds to number of bytes in a *short int*

I

Corresponds to the number of bytes in an *int*

L

Corresponds to the number of bytes in a *long int*

Multiple format characters can appear in one *type_string* and multiple **−t** options can appear on the command line. If there is no **−t** option, the default is −t o2.

−v
> Displays all lines. Normally, **od** does not display multiple lines that differ only in the address. It displays the first line with a single * under it to show that any subsequent lines are the same.

The second form of the syntax is the historical (Berkeley Software Distribution) implementation of the command. If you use this form, you can specify only a single input *file*. If you do not give a *file* argument, **od** reads the standard input. You can supply an offset, but you must precede it with a plus sign (+) to distinguish it from a file name if no file is given. Giving an offset causes a seek to a position in the file where output begins. If the offset ends in a period (.), **od** considers it to be decimal; otherwise, **od** considers it octal. If you follow the offset with a b, **od** multiplies it by the block size of 512 bytes. The format of the offset determines the format of the address; that is, if it is interpreted as decimal, the addresses are displayed in decimal.

Note: The **od** command does not work on a file whose file name starts with either a digit or a plus (+) sign, unless the **−A**, **−N**, **−j**, or **−t** options are used.

The second form of **od** accepts the following options:

−b
> Bytes in octal

−c
> Bytes in ASCII

−D
> Unsigned decimal longs (4 bytes)

−d
> Unsigned decimal words (2 bytes)

−h
> Bytes in hexadecimal

−O
> Unsigned octal longs

−o
> Unsigned octal words

−S
> Signed decimal longs

−s
> Signed decimal words

−X
> Unsigned hexadecimal longs

−x
> Unsigned hexadecimal words

Localization

od uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_NUMERIC**

See for more information.

Exit Values

0

Successful completion

1

Failure due to any of the following:

- Inability to open the input file
- Badly formed offset
- Seek or read error on the input file

2

Failure due to any of the following:

- Incorrect command-line argument
- The wrong number of command-line arguments
- Incorrect format character
- Incorrect size modifier for format character

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The options to operate on longs (**–OSXD**) and the hex byte (**–h**) are extensions to the POSIX standard.

Related Commands

dd

paste — Merge corresponding or subsequent lines of a file

---

paste [–s] [–d *list***]** *file ...*

Purpose

paste concatenates lines of all the specified input files onto the standard output. If you specify – (dash) instead of a file, **paste** uses the standard input. Normally, an output line consists of the corresponding lines from all the input files. **paste** replaces the newline character at the end of each input line (except the one from the last file on the command line) with a tab character, or characters specified by the **–d** option.

Options

–d *list*
> Specifies a list of characters to be used one at a time instead of the tab character to replace the newline at the end of input lines. **paste** uses *list* circularly; when it exhausts the characters in *list*, it returns to the first character in the list. If you also specify the **–s** option, **paste** returns to the first character of *list* after processing each file. Otherwise, it returns to the first character after each line of output. *list* can contain any of the following standard C escapes such as \n, \t, \r, \b, \\, and \0, where \0 indicates that no separator is to be used.

–s
> Concatenates all lines from each input file together on the single output line. If the **–s** option is not specified and the end of the file is detected on any (but not all) of the input files, **paste** behaves as though empty lines have been read from those files.

Examples

The command:

```
ls | paste –s –d'\t\t\n' –
```

displays the output of **ls** in three tab separated columns.

If file A contains:

```
a
b
c
```

and file X contains:

```
x
y
z
```

then the command:

```
paste A X
```

produces:

```
a        x
b        y
c        z
```

and the command:

```
paste –s A X
```

produces:

```
a        b        c
x        y        z
```

Localization

paste uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
    Successful completion

1
    Failure due to any of the following:

    - Missing input files
    - Too many files specified
    - Inability to open a file

2
    Unknown command-line option

Messages and Return Codes

Possible error messages include:

Too many files at *name*
    You specified more files than **paste** can handle. The *name* given in the error message is the name of the first file that **paste** could not open. The number of files that **paste** can open depends on the number of files that other processes have open.

Portability

POSIX.2, X/Open Portability Guide, UNIX system V.

Related Commands

cut

pathchk — Check a path name

> **pathchk** [**-p**] *pathname...*

Purpose

pathchk checks one or more path names (specified by *pathname*) for validity and portability (based on the underlying file system). A path name is valid if you can use it to create or access a file without causing a syntax error. A path name is portable if the file system does not truncate the name when it tries to use it. **pathchk** writes an error message indicating the error detected and the erroneous path name if any path name:

- Is longer than PATH_MAX bytes
- Contains a component longer than NAME_MAX bytes
- Contains any component in a directory that is not searchable
- Contains any character in any component that is not valid

Options

−p

instead of using the previous criteria, writes an error message if *pathname*:

- Is longer than _POSIX_PATH_MAX bytes
- Contains any component longer than _POSIX_NAME_MAX bytes
- Contains any character in any component that is not in the portable file name character set

Localization

pathchk uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

All path names passed the check.

1

An error occurred.

2

Unknown command-line option.

Portability

POSIX.2.

pax -- Interchange portable archives

```
pax [-cdnqvz] [-f archive] [-s substitute] ...  [pattern ...]
pax -r [-cdiknuvz] [-f archive] [-o options ...] [-p string ...]
[-s substitute ...] [-V volpat] [pattern ...]
pax -w [-diLqtuvXz] [-b blocksize] [[-a] [-f archive]] [-o options ...]
[-s substitute ...] [-V volpat] [-x format] [pathname ...]
pax -r -w [-diklLnquvX] [-p string ...]
[-s substitute ...] [pathname ...] directory
```

Purpose

pax reads and writes archive files. An *archive file* concatenates the contents of files and directories, and can also record such information as file modification dates, owner names, and so on. You can therefore use a single archive file to transfer a directory structure from one machine to another, or to back up or restore groups of files and directories.

A file stored inside an archive is called a *component file*; similarly, a directory stored inside an archive is called a *component directory*. Together, component files and directories make up the *components* of the archive file.

You can specify the name of the archive file with the **–f** *archive* option. If you do not specify **–f** with either **–r** or **–w**, the **–r** option assumes the archive file is the standard input and the **–w** option assumes it is the standard output.

There are four possible formats for the **pax** command line:

- If you do not specify **–r** or **–w**, you are in *list mode*. In this mode, **pax** uses the standard output to display the table of contents of an existing archive file. **pax** displays information only on those component files whose names match one of the *pattern*s given on the command line; these are described in "Patterns" on page 240.

- If you specify **–r** but not **–w**, you are in *read mode*. In this mode, **pax** reads an archive file as input and extracts selected components from the archive. By default, **pax** selects the components using *pattern*s given on the command line. If the archive contains several components with the same name, **pax** extracts each of them with later components overwriting files created by earlier components with the same name.

  pax stores extracted components in the working directory. Extracted directories become subdirectories of the working directory. Ownership and permissions of the extracted files are discussed under the **–p** option.

- If you specify **–w** but not **–r**, you are in *write mode*. In this mode, **pax** writes out an archive file that contains the specified *pathname*s as components. If a *pathname* is a directory, the archive file contains all the files and subdirectories in that directory. If you do not specify any *pathname*, **pax** reads the standard input to get a list of path names to select; the input should give one path name per line.

- If you specify both **–r** and **–w**, you are in *copy mode*. In this mode, **pax** reads the specified *pathname*s and copies them to the specified *directory*. In this case, the given *directory* must already exist and you must be able to write to that directory. If a *pathname* is a directory, **pax** copies all the files and subdirectories in that directory as well as the directory itself. If you do not specify any *pathname*, **pax** reads the standard input to get a list of path names to copy; the input should give one path name per line.

pax can read input archives in **cpio** and **tar** format. It can also write these formats; see the **–x** option.

⚠️ **Attention:** On OpenExtensions, you need appropriate privileges to create character special files. If a non-superuser tries to restore character special files, **pax** cannot create them.

Patterns

Command-line patterns are similar to the wildcard constructs explained in **sh**. For example, the pattern * stands for any string of characters excluding slash characters. **pax** does not match the slash. A pattern such as ***.c** therefore selects all files with the suffix **.c** in the top-level directory of the archive. For example, it will not select any archive members with a path name containing a /.

If you do not specify any *pattern*s on a command line that accepts patterns, all archive members are selected. As a special case, the pattern * alone will select all archive members.

File Names

Although **pax** uses the locales defined by the various localization variables when doing substitutions and file name matching, file names are always written to the archive using the ISO/IEC 8859-1 character set.

Options

The following options can appear on **pax** command lines. Some of them are appropriate to only some forms of the command, as shown in the syntax list.

−a
Appends specified files or directories to the end of the contents of an existing archive. If the archive does not already exist, **pax** creates it.

−b *blocksize*
Specifies the *block size* in an output operation. Each output operation writes *blocksize* bytes, where *blocksize* is an integer appropriate to the output device. If b follows the *blocksize* number, the block size is the given number of 512-byte blocks. If k follows the *blocksize* number, the block size is the given number of 1024-byte blocks. The default *blocksize* is 10k for **tar** archives, 5k for **cpio** archives. The block size must be at least 512 bytes for reading.

−c
Selects all those files that do *not* match any of the *pattern*s given on the command line; this is the opposite of the usual behavior.

−d
Does not traverse directories. A pattern matching a directory extracts only the directory itself. When creating an archive, a directory name stores only the directory itself.

−f *archive*
Lets you specify the name of the archive file instead of using the standard input for list mode, read mode (**−r** operations), and the standard output for write mode (**−w**). *archive* can also be a device name.

−i
Lets you rename files as **pax** works. With extractions, **pax** displays the name of the component it is about to extract and gives you the chance to specify a name for the extracted file. With write operations, **pax** displays the name of the file or directory it is about to record in the archive, and lets you specify a different name to be assigned to the component. If you enter . as the name, **pax** processes the file or directory with no change to the name. If you just press <Enter>, **pax** skips the file (doesn't extract or archive it). **pax** ends if you enter end-of-file.

If you also specify **−s**, **pax** makes the given substitution before displaying the name of the component.

−k
Prevents the overwriting of existing files.

−L
Follows symbolic or external links. When you specify this option, **pax** copies the file to which a symbolic or external link points to the archive. Normally, only the symbolic link is copied.

−l
Is applicable only when you are in copy mode—that is, when you are using the **−rw** format to copy files to another directory. If you specify **−l**, **pax** creates links to the original files whenever possible, rather than copying them.

–n

Treats the *pattern* arguments as ordinary path names. You can use this option only when you specify **–r** but not **–w**. **pax** extracts only the first component with a given path name, even if the archive contains several components with the same name. **pax** checks the given path names against the archive before applying any renaming from the **–i**, or **–s** options. **pax** writes an error message for each specified file that cannot be found in the archive.

–o *options*

Provides information for modifying the algorithm for writing and extracting files that the file format specified with **–x** uses.

pax supports one option to **–o**. It converts data from one code set to another while reading or writing an archive. This option has the format:

```
–o keyword=value[,keyword=value]…
```

where *keyword* is either **to** or **from** and *value* is the name of a code set. The current valid values for code set names are:

ISO8859-1
ISO Latin-1

IBM-1047
Latin 1/Open System Interconnection code page 01047, used in the OpenExtensions shell.

Specifying an unknown *keyword* results in a warning message from **pax**.

You can omit either the **to** or **from** keyword. If you omit **to**, **pax** assumes that you want to write (or read) a portable archive tape and will convert the data to ISO/IEC 8859-1. If you omit **from**, **pax** assumes that you are converting from the system-specific local code set.

If your input contains a character that is not valid in the source code set, **pax** displays a warning and continues, leaving the character untranslated. If the source code set contains a character that is not in the destination code set, **pax** converts the character to an underscore (_).

Note: If you do not specify **–o**, no code set conversion is done. When making code set conversions, **pax** assumes that all files are text files, since only text files are portable.

–p *string*

Specifies file characteristic options.

The *string* can consist of any combination of the following specification characters:

a
Does not preserve file access times

e
Preserves the user ID, group ID, file mode, access time, and modification times

m
Does not preserve file modification times

o
Preserves the user ID and group ID

p
Preserves the file mode.

If a character in *string* duplicates or conflicts with another character in *string*, the one occurring last takes precedence. By default, **pax** restores modification time only.

–q

For input mode only, **pax** assumes that all created files are text files and extracts them to the local text file format. On systems with fixed length records, this might mean padding with blanks to the record length.

On UNIX and POSIX-compliant systems, **pax** removes all carriage return characters (\r) and retains only the newline characters (\n).

It might be desirable to have this option work when creating output to convert text to a system-independent format. However, due to the format of an archive file, this would (unacceptably) require all files to be read twice.

–r

Reads an archive file from standard input.

–s *substitute*

Modifies path names using a substitution command *substitute*. This is similar to the substitution command of the **ed** text editor. The full option has the form:

```
-s /bregexp/string/[gp]
```

where *bregexp* is a basic regular expression (see Appendix B, "Regular Expressions (regexp)," on page 471) and *string* is a string that **pax** is to insert in place of matches for the regular expression. *string* can contain an ampersand & (standing for the string matching *bregexp*), or \1, \2, and so on (with the meanings defined in **regexp**), for subexpression matching.

Normally, **–s** replaces only the first match for *bregexp*. A g following the *string* replaces all matches in the line.

A p following the *string* prints all successful substitutions on the standard error stream. **pax** displays a substitution in the format:

```
oldname >> newname
```

In this form of the command, the slash (/) is used as the character separating parts of *substitute*; you can use any non-null character instead.

There may be more than one **–s** option on the command line. In this case, **pax** tries the substitutions in the order given. **pax** stops trying to make these substitutions as soon as it makes its first successful substitution. If the null string replaces a file name, **pax** ignores that file name on both input and output.

–t

After reading files being archived, **pax** resets the access time to that prior to **pax**'s access.

–u

Compares component dates to dates of existing files with the same name. When extracting components with **–r** (read mode), **pax** extracts a file only if its modification date is more recent than the modification date on an existing file of the same name. In other words, it doesn't overwrite an existing file if the existing file is newer than the one in the archive.

Similarly, when copying files with **–rw** (copy mode), **pax** does not overwrite an existing file if the existing file is newer than the one being copied.

In a command that uses **–w** but not **–r** (write mode), **–u** checks to see if the file being added has the same name as a file already in the archive. If so, and if the file being added is newer than the one in the archive, **pax** leaves the old file in the archive and appends the new one at the end. In this case, **–u** automatically implies **–a**, which means that **pax** adds new files to the end of the archive.

–V *volpat*

Provides automatic multivolume support. **pax** writes output to files the names of which are formatted with *volpat*. It replaces any occurrence of # in *volpat* with the current volume number. When you invoke **pax** with this option, it asks for the first number in the archive set, and waits for you to type the number and a carriage return before proceeding with the operation. **pax** issues the same sort of message when a write error or read error occurs on the archive; the reasoning is that this kind of error means that **pax** has reached the end of the volume and is to go on to a new one. An interrupt at this point ends **pax**.

–v

Lists path names on the standard error stream just before beginning to process the files or directories, but after any **–i**, or **–s** options have had their effect. In list mode (neither **–r** nor **–w** is specified), **pax**

displays a "verbose" table of contents; this verbose format shows information about the components in the same format used by the **ls** command with the **–l** option.

–w

Writes files to the standard output in the specified archive format.

–X

Writes out only those files that are on the same device as their parent directory.

–x *format*

Specifies a format for an output archive. The *format* argument can be:

cpio

Standing for the ASCII format used by the **cpio** command; see "cpio -- Copy in/out file archives" on page 85.

cpiob

Standing for the binary format used by **cpio**.

tar

Standing for the old format of **tar** files; see "tar -- Manipulate the tar archive files to copy or back up a file" on page 318.

ustar

Standing for the (new) USTAR format used by the **tar** command.

The default *format* is ustar.

–z

Performs Lempel-Ziv compression. Output is always a 16-bit compression. On input, any compression up to 16-bit is acceptable.

Output

When the **–v** option is used in list mode, **pax** produces a verbose table of contents for the archive. The output has the format of the **ls** command with the **–l** option, with the addition of the notation:

```
pathname == linkname
```

which indicates that *linkname* is a hard link for *pathname*. See **ls –l** for an explanation of the format.

Examples

1. The following creates an archive file from all the files in the working directory:

   ```
   pax -w . >/dir/archive
   ```

2. The following extracts all the components of an archive file and puts them into the working directory:

   ```
   pax -r * </dir/archive
   ```

3. The following converts an archive file from one character set to another:

   ```
   pax -wf testpgm.pax -o from=CP1047,to=IS646 /tmp/posix/testpgm
   ```

   This command backs up the **/tmp/posix/testpgm** directory, which is in the character set CP1047, into an archive file that is targeted to an ASCII character set (IS646). CP1047 is the code page used in the OpenExtensions shell.

   The **–o** option is very useful for transferring text data between systems that use different code pages.

Localization

pax uses the following localization environment variables:

• **LANG**

- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
Successful completion

1
Failure due to any of the following:

- Incorrect option
- Incorrect command-line arguments
- Out of memory
- Compression error
- Failure on extraction
- Failure on creation

If **pax** cannot extract a particular file when reading, or find a particular file when writing, it generates an error message and continues to process other files but returns a status of 1. If any other sort of error occurs, **pax** ends immediately without attempting further processing.

If you see the following message after a write operation:

```
If you want to go on, type device/file name when ready
```

it indicates that your directory or device containing the archive file is full. To continue, enter the name of a new directory; to end **pax**, type <Ctrl-C>.

If you see that message after a read operation, it means that **pax** could not find the archive file you specified, or that it was damaged. In this case, type <Ctrl-C> to end the operation and then restart **pax** with the correct archive name.

Portability

POSIX.2.

The **–L**, **–q**, **–V**, and **–z** options are extensions of the POSIX standard.

Related Commands

cpio, **ls**, **tar**

pr — Format a file in paginated form and send it to standard output

```
pr [-adFfmprtW] [-c n] [-e[char][gap]]
    [-H header-fmt] [-h header] [-i[char][gap]] [-l n] [-n[char][n]]
    [-o n] [-s[char]] [-w n] [+n] [-n] [file ...]
```

Purpose

pr prints the specified *file*s on the standard output in a paginated form. If you do not specify any *file*s or if you specify a file name of −, **pr** reads the standard input. By default, **pr** formats the given files into single-column 66-line pages. Each page has a five-line header. The first line contains the file's path name, the date it was last modified, and the current page number; the other lines are blank (this is the default). A five-line trailer consists of blank lines.

If you specify multiple columns, **pr** places its output in columns of equal width separated by at least one space, truncating each line to fit in its column. Input lines can be ordered down the columns or across the page on output; or different columns can each represent different files.

Options

pr recognizes the following options:

+n

    Starts printing with the *n*th page of each file; that is, skips the first *n*−1 pages. The default for *n* is 1.

−n

    Prints *n* columns of output. When you specify this option, **pr** behaves as though you had also specified the **−e** and **−i** options. When you specify both this option and **−t**, **pr** uses the minimum number of lines possible to display the output. Do not specify this option with the −*m* option.

−a

    Orders input lines across the page on output, instead of down. You should use this option only with **−n**.

−c n

    Displays *n* columns of output. When you specify this option, **pr** behaves as though you had also specified the **-e** and **-i** options. When you specify both this option and **-t**, **pr** uses the minimum number of lines possible to display the output. Do not specify this option with **-m**.

−d

    Produces double-spaced output.

−e[char][gap]

    Expands each occurrence of the input tab character to a string of spaces so that the following character has the next column position which is a positive multiple of *gap*, plus 1. If you do not specify *gap*, or if it is zero, **pr** assumes that *gap* has the value of 8. If you specify the nondigit character *char*, **pr** treats it as the input tab character. Otherwise, **pr** uses the standard tab character.

−F

    Uses form feeds to separate pages. **pr** normally separates pages by sending a series of <newline> characters to fill the length of a page.

−f

    Uses form feeds to separate pages. When output is to a terminal, **pr** waits for you to press ENTER two times before displaying the text. **pr** normally separates pages by sending a series of <newline> characters to fill the length of a page.

−H *header_fmt*

    Lets you customize your header line by specifying a format with the string *header_fmt*. **pr** recognizes the following special formatting commands:

%c
Date and time

%F
Current file name, or *header* string given by **–h**

%P
Page number

%L
Line number

%D
Date

%T
Time

%u
Current user name

The default header format is equivalent to the option:

```
-H "%c   %F   Page %P"
```

–h *header*
Uses the *header* string instead of the file name on each succeeding page header.

–i[*char*][*gap*]
Replaces white space with tabs on output. *char*, if given, is the output tab character. The default is the tab character. **pr** sets tabs every *gap* positions; the default for *gap* is 8. If this tab character differs from the input tab character and the actual data contains this tab character, the result is liable to be quite a mess.

–l *n*
Sets the number of lines per page of output. The default is 66. The actual number of lines printed per page is this number less 5 for the header and 5 for the trailer. If *n* is less than 10 (the number of lines needed for the header and the trailer), **pr** displays neither the header nor the trailer.

–m
Prints each file in its own column down the page. This overrides the **–a** option, forcing the **–n** option to be the number of files given. When you also specify the **–n** option, it gives line numbers for the first column only.

–n[*char*][*n*]
Numbers the lines of each file. Each number takes up *n* positions; the default for *n* is 5. The character *char* separates the number from the line; this defaults to the tab character. If *char* is the same as the input tab character, **pr** follows the number with the spaces needed to get to the next tab stop. **pr** may in turn replace these spaces with the output tab character if you specified the **–i** option. For multicolumn output, **pr** adds line numbers to each column. The **–m** option gives the line number for the first column only.

–o *n*
Offsets each line of output by *n* character positions.

–p
Pauses before the beginning of each page if output is to a terminal device. **pr** waits for you to press ENTER two times.

–r
Suppresses error messages due to failures when opening files.

–s[*char*]
Prints each column at its correct length. The character *char* separates columns. The default value for *char* is the tab character. This character is never replaced by the output tab character. Normally **pr** pads each column with spaces or truncates it to the exact column width. Unless the **–w** option is also used, **–s** resets the page width to 512 column positions.

–t
> Does not print the headers and trailers, and quits after the last line of the file—it does not display any extra lines.

–W
> Folds lines at the column width when you do not specify the **–s** option; **pr** treats each separate part of the line as a separate line.

–w *n*
> Sets the width of the page to *n* column positions. If you do not specify this option, the default page width is 72 (if you did not specify **–s** option) or 512 (if you did specify **–s**). This page width does not normally apply to single-column output; however, single-column output with the **–W** option does use this width.

Files

/dev/tty
> For prompting.

Environment Variables

TZ
> Contains the local time zone. **pr** uses this value when displaying times in header lines.

Localization

pr uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
> Successful completion

1
> Failure due to any of the following:
>
> - Insufficient memory
> - Insufficient line width
> - Write error on the standard output

2
> Syntax error or unknown command-line option

Messages and Return Codes

Possible error messages include:

Missing header
> You specified **–h** or **–H** but did not supply a *header* or *header_fmt* string.

Width is insufficient
   The line is not wide enough to hold the given number of columns with the given column width; or a
   column is not wide enough to hold the minimum amount of data.

Portability

POSIX.2, X/Open Portability Guide.

The **–c**, **–H**, **–p**, and **–W** options are extensions of the POSIX standard.

Related Commands

cat, **fold**

print — Return arguments from the shell

---

print [**–npRrs**] [**–u**[*descriptor*]] [*argument …*]

Purpose

Calling **print** without options or with only the **–** option displays each *argument* to the standard output using the same escape conventions as **echo**. In this case, **print** and **echo** work the same way; see **echo**.

The options accepted by **print** increase its utility beyond that of **echo**.

–n
    Does not automatically add a new line to the end of the output.

–p
    Sends output to a coprocess.

–R
    Is similar to **–r**, except that **print** treats all subsequent options (except **–n**) as arguments rather than as options.

–r
    Ignores escape conventions.

–s
    Appends the output to the command history file rather than sending it to standard output.

–u[*descriptor*]
    Redirects the output to the file corresponding to the single digit file *descriptor*. The default file descriptor is 1.

Usage Notes

This command is built into the shell.

Exit Values

Possible exit status values are:

0
    Successful completion

1
    Failure due to any of the following:

    • Incorrect *descriptor* specified with **–u**

    • Nonexistent coprocess

2
    Failure due to an incorrect command-line option

Messages and Return Codes

Possible error messages include:

Cannot print on file descriptor …
    You tried to print on a file descriptor that was not opened for writing.

History not available
    You specified the **–s** option to write into a history file, but you are not now using a history file.

print

Portability

POSIX.2.

Related Commands

echo, **fc**, **read**, **sh**

printf — Write formatted output

---

printf *format* [*argument ...*]

Purpose

printf writes the *argument* operands to standard output, formatted according to the *format* operand.

format is a format string composed of conversion specifications that convert and add the next *argument* to the output. *format* can contain backslash-escape sequences. These conversions are similar to those used by the ANSI C standard. Conversion specifications have the form:

```
%[flag][width][precision][char]
```

where *flag* is one of the following:

-

Left-justifies the field; default is right justification.

+

Always prefixes a signed value with a sign (+ or -).

space
Reserves a character position at the start of the string for the minus sign (for negative numbers) or a space (for positive numbers). If both space and - appear as flags, the space flag is ignored.

#

Prefixes octal values with 0 and hexadecimal values with 0x or 0X. For floating-point values, this causes the decimal point always to be displayed even if no characters follow it.

0

Pads numeric values with leading zeros. If both 0 and - appear as flags, the 0 flag is ignored.

width is the minimum field width of the output field. If the converted value is shorter than the minimum width, **printf** pads it with spaces or zeros.

In a string, *precision* is the maximum number of bytes to be printed from the string; in a number, the precision is the number of digits to be printed to right of the decimal point in a floating-point value. *width* or *precision* can be specified as *, in which case the value is read from the next argument, which must be an integer. For example:

```
printf "%*.*d\n" 20 10 200
```

is equivalent to:

```
printf "%20.10d\n" 200
```

The conversion character *char* is one of the following:

d
Decimal integer.

i
Decimal integer.

o
Unsigned octal integer.

x,X
Unsigned hexadecimal integer.

u
Unsigned decimal integer.

f,F
Floating point.

e,E
Floating point (scientific notation).

g,G
The shorter of e and f (suppresses insignificant zeros).

c
Single character of an integer value; the first character of a string.

s
String.

b
A string that may contain a backslash-escape sequence. Valid escape sequences are those described in "echo — Write arguments to standard output" on page 107.

\0*ddd*
Where *ddd* is 0-to-3-digit octal number

\xdd
Where *dd* is a 0-to-2-digit hexadecimal number

\c
Indicates the first character of a string; number arguments are treated as strings.

When there are more arguments than positions in *format*, the *format* string is applied again to the remaining arguments. When there are fewer arguments than there are positions in the *format* string, **printf** fills the remaining positions with null strings (character fields) or zeros (numeric fields).

Localization

printf uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_NUMERIC**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
Successful completion

>0
The number of failures due to any of the following:

- Missing format specifications
- Arguments supplied for a *format* string that does not accept them (that is, that has no %s)
- Incorrect integer argument
- Incorrect floating-point argument

Portability

POSIX.2, UNIX system V.

The **%F** format and the handling of ⋆ as a width or precision argument are extensions of the POSIX standard.

Related Commands

echo, **print**

ps — Return the status of a process

> **ps** [**–Aacdefjln**] [**–G** *idlist*] [**–g** *grouplist*] [**–n** *name*]
> [**–o** *format*] ...
> [**–p** *proclist*] [**–s** *idlist*] [**–t** *termlist*] [**–U|u** *uidlist*]

Purpose

ps displays information about processes, provided that you have appropriate privileges to obtain information about the requested processes.

ps accepts several options. When a description says that **ps** lists "all processes", it means all the processes in your virtual machine, provided that you have appropriate privileges.

Options

–A

Displays information on all accessible processes. You cannot specify both **–a** and **–A**.

–a

Displays information on all processes associated with terminals. You cannot specify both **–a** and **–A**.

–c

Displays more detailed information about processes for the **–f** and **–l** options. **–c** is accepted but not currently implemented.

–d

Displays information for all processes except group leaders.

–e

Displays information on all accessible processes.

–f

Displays information as if the user specified:

```
–o ruser=UID,pid,ppid,stime,tty=TTY,atime,args
```

–G *idlist*

Displays information on processes with group ID numbers in *idlist*. Separate the numbers in *idlist* with either blanks or commas.

–g *grouplist*

Displays information on processes with real group ID numbers in *grouplist*. Separate numbers in *grouplist* with either blanks or commas.

–j

Displays information as if the user specified:

```
–o pid,sid,pgid=PGRP,tty=TTY,atime,args
```

–l

Displays information as if the user had specified:

```
–o state,ruid=UID,pid,ppid,nice,vsz=SZ,tty=TTY,atime,comm=COMD
```

–o *format*

Displays information according to the given *format* specifications. For further information, see "Format Specifications" on page 255.

–n *name*

Specifies the name of the executable file containing the kernel symbol table.

–p *proclist*
>   Displays information for processes with process ID numbers in *proclist*. Separate numbers in *proclist* with commas.

–s *idlist*
>   Displays information for processes with session ID numbers in *idlist*. Separate the numbers in *idlist* with commas.

–t *termlist*
>   Displays information for processes with terminals in *termlist*. You denote terminals in *termlist* with either the file name of the device (for example, `tty04`), or if the file name begins with `tty`. For example, `tty04` and `04` both denote the same terminal. Terminals in *termlist* are separated by either blanks or commas.

–U *userlist*
>   Displays information for processes with user IDs in *userlist*. Items in *userlist* can be user ID numbers or login names, and are separated by commas.
>
>   **Note:** A user can only view processes in their own virtual machine.

–u *userlist*
>   Displays information for processes with user IDs in *userlist*. Items in *userlist* can be user ID numbers or login names, and are separated by commas.

Format Specifications

The *format* specified with **–o** is a list of names separated with blanks or commas. At the beginning of the output display, **ps** displays column headings to tell you what you are seeing. For example, if you specify `ruser` (indicating that you want to see real user names), **ps** normally puts the heading RUSER at the top of the column that shows real user names.

If you do not specify the **–o** option, **ps** displays the information as though you had specified:

```
 -o pid,tty=TTY,time,comm
```

The following list shows the names that **ps** recognizes. At the end of each description, we put the default column heading inside square brackets.

`args`
>   Displays the command that is running, with all its arguments. `[COMMAND]`

`comm`
>   Displays the name of the command that is running. This string is padded on the right if necessary. `[COMMAND]`

`etime`
>   Displays the amount of real time that has elapsed since the process began running. **ps** shows the time in the form:

```
 [[dd-]hh:]mm:ss
```

>   where *dd* is the number of days, *hh* is the number of hours, *mm* is the number of minutes, and *ss* is the number of seconds. `[ELAPSED]`

`group`
>   Displays the effective group ID of the process, as a group name if possible and as a decimal group ID if not. `[GROUP]`

`nice`
>   Displays the nice value (urgency) of the process as a decimal value. `[NI]`

`pcpu`
>   Displays a percentage value giving the ratio of processor time used to processor time available. `[%CPU]`

`pgid`
>   Displays the process group ID as a decimal value. `[PGID]`

pid
Displays the process ID as a decimal value. Decimal *pid*s are reported with default actions. [XPID]

ppid
Displays the parent process ID as a decimal value. [PPID]

rgroup
Displays the real group ID of the process, as a group name if possible and as a decimal group ID if not. [RGROUP]

ruser
Displays the real user ID of the process, as a user name if possible and as a decimal user ID otherwise. [RUSER]

time
Displays the amount of processor time that the process has used since it began running. **ps** displays this time in form similar to that used by etime. [TIME]

tty
Displays the name of the controlling terminal (if any). [TT]

user
Displays the effective user ID of the process, as a user name if possible and as a decimal user ID otherwise. [USER]

vsz
Displays the amount of (virtual) memory that the process is using, as a decimal number of kilobytes. [VSZ]

xpgid
Displays the process group ID as a hexadecimal value. [XPGID]

xpid
Displays the process ID as a hexadecimal value. [XPID]

xppid
Displays the parent process ID as a hexadecimal value. [XPPID]

The following names are extensions to **ps**:

addr
Displays the address of the process. [ADDR]

atime
Displays the abbreviated processor time of the process. [TIME]

flags
Displays the process flags. [F]

gid
Displays the effective group ID of the process. [EGID]

pri
Displays the process priority. [PRI]

rgid
Displays the real group ID of the process. [GID]

ruid
Displays the real user ID of the process. [UID]

sid
Displays the session ID of the process. [SID]

state
Displays the process state. [STATE] Various values can be printed in this field:

K
Kernel wait (for example, pause or sigsuspend).

R
Running (not kernel wait).

Both of these values will be prefixed with M to denote the fact that the processes are potentially multithreaded.

stime
  Displays the start time of the process. [STIME]

uid
  Displays the effective user ID of the process. [EUID]

wchan
  Displays the channel upon which the process is waiting. [WCHAN]

If you want to specify your own column heading instead of using the defaults, put:

```
=heading
```

after the name in the *format* list. For example:

```
ps -o args,ruser=WHO
```

displays the command and the real user name. The heading for the command column is the default COMMAND, but the heading for the user name column is WHO. If you specify = with no heading, **ps** displays that column without a heading. If all columns have no heading, **ps** displays no heading line.

Environment Variables

ps uses the following environment variable:

COLUMNS
  Contains the maximum number of columns to display on one line.

Localization

ps uses the following localization environment variables:

• **LANG**
• **LC_ALL**
• **LC_CTYPE**
• **LC_MESSAGES**
• **LC_TIME**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
  Successful completion

1
  Failure due to the inability to open the process table

2
  Failure due to any of the following:

  • Unknown command-line option
  • Missing *format* string after **–o**
  • Missing lists after other options
  • Too many arguments on the command line

Portability

POSIX.2.

The **–c**, **–d**, **–e**, **–f**, **–g**, **–j**, **–l**, **–n**, **–s**, and **–u** options are extensions of the POSIX standard.

Related Commands

jobs, **kill**

pwd — Return the working directory name

<div style="border:1px solid">

pwd

</div>

Purpose

pwd displays the absolute path name of the working directory to standard output.

If the current working directory is a symbolic link to another directory, the path name displayed depends upon the setting of the shell's `logical` flag. See **set** for more information.

Usage Notes

pwd is a built-in utility.

Localization

pwd uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
    Successful completion
1
    Inability to determine the working directory

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

sh

read — Read a line from standard input

read [−prs] [−u[*d*]] [*variable*?*prompt*] [*variable ...*]

Purpose

When you call **read** without options, it reads one line from the standard input, breaks the line into fields, and assigns the fields to each *variable* in order.

To determine where to break the line into fields, **read** uses the built-in variable **IFS** (which stands for *internal field separator*). Encountering any of the characters in **IFS** means the end of one field and the beginning of the next. The default value of **IFS** is blank, tab, and newline.

In general, a single **IFS** character marks the end of one field and the beginning of the next. For example, if **IFS** is colon (:), **read** considers the input a::b to have three fields: a, an empty field, and b. However, if **IFS** contains blanks, tabs or escaped newlines, **read** considers a sequence of multiple blanks, tabs, or escaped newlines to be a single field separator. For example, "a   b" has two fields, even though there are several blanks between the a and b.

The *n*th *variable* in the command line is assigned the *n*th field. If there are more input fields than there are *variable*s, the last *variable* is assigned all the unassigned fields. If there are more variables than fields, the extra variables are assigned the null string (" ").

The environment variable **REPLY** is assigned the input when no variables are given. The exit status of **read** is 0, unless it encounters the end of the file.

Options

−p
    Receives input from a coprocess.

−r
    Treats input as raw data, ignoring escape conventions. For example, **read −r** does not interpret a final backslash (\) as a line continuation character, but as part of the input.

−s
    Adds input to the command history file as well as to the variables specified with *variable*.

−u[*d*]
    Reads input from the single-digit file descriptor *d,* rather than from the standard input. The default file descriptor is 0.

When the first variable parameter has the form:

```
variable?prompt
```

it defines a prompt for input. If the shell is interactive, **read** sends the *prompt* to the file descriptor *d* if it is open for write and is a terminal device. The default file descriptor for the *prompt* is 2.

Examples

```
IFS=':'
while read name junk junk1 junk2 junk3
do
   echo $name
done </etc/samples/comics.lst
```

provides a list of comic names from the sample **comics.lst** file.

Environment Variables

The following environment variables affect **read**:

IFS
Contains a string of characters to be used as internal field separators.

PS2
Contains the prompt string that an interactive shell uses when it reads a line ending with a backslash and you did not specify the **–r** option, or if a here-document is not terminated after you enter a newline.

REPLY
Contains the input (including separators) if you did not specify any variables. The ability of omitting the variable from the command and using the environment variable **REPLY** is an extension.

Usage Notes

This command is built into the shell.

Exit Values

Possible exit status values are:

0
Successful completion

1
Failure due to any of the following:

- End-of-file on input
- Incorrect *variable*
- Incorrect descriptor specified after **–u**
- Missing coprocess

2
Incorrect command-line argument

Messages and Return Codes

Cannot read on file descriptor ...
You tried to read a file descriptor that was not opened for reading.

Portability

POSIX.2, X/Open Portability Guide.

read is a built-in shell command.

The **–p**, **–s**, and **–u** options are extensions of the POSIX standard.

Related Commands

continue, **fc**, **print**, **sh**

readonly — Mark a variable as read-only

readonly [–p] [*name*[=*value*] ...]

Purpose

readonly prevents subsequent changes in the value of any of the *name* arguments. Parameters of the form:

```
name=value
```

assign *value* to *name* as well as marking *name* read-only. If **readonly** is called without arguments, it lists, with appropriate quoting, the names you have set as read-only in the following format:

```
Variable="value"
```

Options

–p
> Displays *export name=value* pairs that, when read by a shell, ensures the read-only status and values of variables. The shell formats the output so it is suitable for reentry to the shell as commands that achieve the same attribute-setting results.

Usage Notes

This is a special built-in command of the shell.

Exit Values

Possible exit status values include:

0
> Successful completion

2
> Failure due to incorrect command-line argument

Portability

POSIX.2, X/Open Portability Guide.

readonly is a special built-in shell command.

The behavior given for calling **readonly** with no arguments is an extension of the POSIX standard.

Related Commands

alias, **sh**, **typeset**

return — Return from a shell function or . (dot) script

return [*expression*]

Purpose

return returns from a shell function or **.** (dot) script. The exit status is the value of *expression*. The default value of *expression* is the exit status of the last command run.

Usage Notes

This command is built into the shell.

Exit Values

The current function or script returns the value of *expression*. If no *expression* is given, the exit status is the exit status of the last command run.

Portability

POSIX.2, X/Open Portability Guide.

return is a special built-in shell command.

Related Commands

exit, **sh**

rm — Remove a directory entry

> **rm** [**–fiRr**] *file ...*

Purpose

rm removes each specified *file* argument (provided that it is a valid path name). If you specify either **.** or **. .** as the final component of the path name for a *file*, **rm** displays an error message and moves onto the next file. If a file does not have write permission set, **rm** asks you if you are sure you want to delete the file; type the yes expression defined in **LC_MESSAGES** (the English expression is typically y or yes) if you really want it deleted.

Note: rm can be used only by the file owner or a superuser.

Options

–f

Deletes read-only files immediately without asking for confirmation. When you specify this option and a file does not exist, **rm** does not display an error message and does not modify the exit status. If you specify both **–f** and **–i**, **rm** uses the option that appears last on the command line.

–i

Prompts you for confirmation before deleting each file. If you specify both **–f** and **–i**, **rm** uses the option that appears last on the command line.

–R

Recursively removes the entire directory structure if *file* is a directory.

–r

Is equivalent to **–R**.

Localization

rm uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

Successful completion

1

Failure due to any of the following:

- Inability to remove a file
- Attempt to remove directory without specifying **–r** or **–R**
- Inability to find file information when using **–r** or **–R**
- Inability to read directory when using **–r** or **–R**

2

   Failure due to any of the following:

   - Incorrect command-line option
   - No file was specified

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

cp, **mv**, **rmdir**

rmdir — Remove a directory

---

rmdir [**−p**] *directory ...*

Purpose

The **rmdir** command removes each requested *directory*. Each directory must be empty for **rmdir** to be successful.

Options

−p

Removes all intermediate components. For example:

```
rmdir -p abc/def/ghi
```

is equivalent to:

```
rmdir abc/def/ghi
rmdir abc/def
rmdir abc
```

Localization

rmdir uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

Successful completion

1

Failure because *directory* is not a directory, or because it still contains files or subdirectories

2

Failure because of an incorrect command-line option, or no *directory* names specified

Messages and Return Codes

Possible error messages include:

Nonempty directory

Files or other directories are found under the directory to be removed. Use **rm −r** to remove the directory.

No such directory

The requested *directory* does not exist or is otherwise inaccessible.

Current directory illegal

You should use **cd** to change to another directory before removing the current directory.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

mkdir, **rm**

sed — Start the sed noninteractive stream editor

```
sed [–En] [script] [file ...]
sed [–En] [–e script] ... [–f scriptfile] ... [file ...]
```

Purpose

The **sed** command applies a set of editing subcommands contained in *script* to each argument input *file*. If you did not specify *file*, **sed** reads the standard input.

sed reads each input line into a special area known as the *pattern buffer*. Certain subcommands [**gGhHx**] use a second area called the *hold buffer*. By default, after each pass through the script, **sed** writes the final contents of the *pattern buffer* to the standard output.

Options

sed recognizes the following options:

–E
> Uses extended regular expressions. Normally, **sed** uses basic regular expressions. See Appendix B, "Regular Expressions (regexp)," on page 471 for more information.

–e *script*
> Adds the argument *script* to the end of the script.

–f *scriptfile*
> Adds the subcommands in the file *scriptfile* (one subcommand per line) to the script.

–n
> Suppresses all output except that generated by explicit subcommands in the **sed** script [**acilnpPr**]

If you need only one *script* argument, you can omit the **–e** and use the first form of the command.

sed subcommands are similar to those of the interactive text editor **ed**, except that **sed** subcommands necessarily view the input text as a stream rather than as a directly addressable file. Script subcommands can begin with zero, one, or two addresses, as in **ed**. Zero-address subcommands refer to every input line. One-address subcommands select only those lines matching that address. Two-address subcommands select those input line ranges starting with a match on the first address up to an input line matching the second address, inclusive. Permissible addressing constructions are:

n
> The number *n* matches only the *n*th input line.

$
> This address matches the last input line.

/regexp/
> This address selects an input line matching the specified regular expression *regexp*. If you do not want to use slash (/) characters around the regular expression, use a different character but put a backslash (\) before the first one. For example, if you want to use % to enclose the regular expression, write \%regexp%.

Subcommands

Each line of a script contains up to two addresses, a single-letter subcommand, possible subcommand modifiers, and an ending newline. The newline is optional in script strings entered on the command line.

The following **sed** subcommand summary shows the subcommands with the maximum number of legitimate addresses. A subcommand can be given fewer than the number of addresses specified, but not more.

aa

*A*ppends subsequent text lines from the script to the standard output. **sed** writes the text after completing all other script operations for that line and before reading the next record. Text lines are ended by the first line that does not end with a backslash (\\). **sed** does not treat the \\ characters on the end of lines as part of the text.

a,bb [*label*]

*B*ranches to `:`*label*. If you omit *label*, **sed** branches to the end of the script.

a,bc

*C*hanges the addressed lines by deleting the contents of the pattern buffer (input line) and sending subsequent text (similar to the *a* command) to the standard output. When you specify two addresses, **sed** delays text output until the final line in the range of addresses; otherwise, the behavior would surprise many users. The rest of the script is skipped for each addressed line except the last.

a,bd

*D*eletes the contents of the pattern buffer (input line) and restarts the script with the next input line.

a,bD

*D*eletes the pattern buffer only up to and including the first newline. Then it restarts the script from the beginning and applies it to the text left in the pattern buffer.

a,bg

*G*rabs a copy of the text in the hold buffer and places it in the pattern buffer, overwriting the original contents.

a,bG

*G*rabs a copy of the text in the hold buffer and appends it to the end of the pattern buffer after appending a newline.

a,bh

*H*olds a copy of the text in the pattern buffer by placing it in the hold buffer, overwriting its original contents.

a,bH

*H*olds a copy of the text in the pattern buffer by appending it to the end of the hold buffer after appending a newline.

ai

*I*nserts text. This subcommand is similar to the **a** subcommand, except that its text is output immediately.

a,bl

*L*ists the pattern buffer (input line) to the standard output so that nonprintable characters are visible. This subcommand works analogously to the **l** subcommand in **ed**. **sed** folds long lines to suit the output device, indicating the point of folding with a backslash (\\).

a,bn

Prints the pattern space on standard output if the default printing of the pattern space is not suppressed (because of the **-n** option). The *n*ext line of input is then read, and the processing of the line continues from the location of the **n** command in the script.

a,bN

Appends the *n*ext line of input to the end of the pattern buffer, using a new line to separate the appended material from the original. The current line number changes.

a,bp

*P*rints the text in the pattern buffer to the standard output. The **–n** option does not disable this form of output. If you do not use **–n**, the pattern buffer is printed twice.

a,bP

Operates like the **p** subcommand, except that it prints the text in the pattern buffer only up to and including the first newline character.

aq

*Q*uits **sed**, skipping the rest of the script and reading no more input lines.

ar *file*
Reads text from *file* and writes it to the standard output before reading the next input line. The timing of this operation is the same as for the **a** subcommand. If *file* does not exist or cannot be read, **sed** treats it as an empty file.

a,bs*/reg/sub/*[**g**p*n*][**w** *file*]
Substitutes the new text string *sub* for text matching the regular expression, *reg*. Normally, the **s** subcommand replaces only the first such matching string in each input line. You can use any single printable character other than space or newline instead of the slash (/) to delimit *reg* and *sub*. The delimiter itself may appear as a literal character in *reg* or *sub* if you precede it with a backslash (\). You can omit the trailing delimiter.

If an ampersand (&) appears in *sub*, **sed** replaces it with *reg*. A \n in *reg* matches an embedded newline in the pattern buffer (resulting,for example, from an **N** subcommand). The subcommand can be followed by a combination of the following:

n
Substitutes only the *n*th occurrence of *regexp.*

g
Forces all occurrences (rather than the default first occurrence) of *regexp* to be replaced.

p
Executes the print (**p**) subcommand only if a successful substitution occurs.

w *file*
Writes the contents of the pattern buffer to the end of *file,* if a substitution occurs.

a,bt [*label*]
Branches to the indicated *label* if a successful substitution has occurred since either reading the last input line or running the last **t** subcommand. If you do not specify *label*, **sed** branches to the end of the script.

a,bw *file*
Writes the text in the pattern buffer to the end of *file*.

a,bx
Exchanges the text in the hold buffer with that in the pattern buffer.

a,by*/set1/set2/*
Transliterates any input character occurring in *set1* to the corresponding element of *set2*. The sets must be the same length. You can use any character other than backslash or newline instead of the slash to delimit the strings.

a,b{
Groups all commands until the next matching } subcommand, so that **sed** runs the entire group only if the **{** subcommand is selected by its address(es).

: *label*
Designates a *label*, which can be the destination of a **b** or **t** subcommand.

a,b!*cmd*
Runs the specified *cmd* only if the addresses do *not* select the **!** subcommand.

#
Treats the script line as a comment unless it is the first line in the script. Including the first line in a script as #n is equivalent to specifying **–n** on the command line. An empty script line is also treated as a comment.

a=
Writes the decimal value of the current line number to the standard output.

Examples

Here is a filter to switch desserts in a menu:

```
sed 's/cake\(ic\)*/cookies/g'
```

Environment Variables

COLUMNS

Contains the width of the screen in columns. If set, **sed** uses this value to fold long lines on output. Otherwise, **sed** uses a default screen width of 80.

Localization

sed uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

Successful completion

1

Failure because of any of the following:

- Missing script
- Too many script arguments
- Too few arguments
- Unknown option
- Inability to open script file
- No noncomment subcommand
- Label not found in script
- Unknown subcommand
- Nesting **!** subcommand not permitted
- No \ at end of subcommand
- End-of-file in subcommand
- No label in subcommand
- Badly formed file name
- Inability to open file
- Insufficient memory to compile subcommand
- Bad regular expression delimiter
- No remembered regular expression
- Regular expression error
- Insufficient memory for buffers
- **y** subcommand not followed by a printable character as separator
- The strings not the same length
- Nonmatching { and } subcommands
- Garbage after command
- Too many addresses for command

- Newline or end-of-file found in pattern
- Input line too long
- Pattern space overflow during **G** subcommand
- Hold space overflow during **H** subcommand
- Inability to chain subcommand

Messages and Return Codes

The error messages are output only if **h** or **H** subcommands are used after **sed** outputs ?. Possible error messages include:

badly formed file name for *command* command
The given subcommand required a file name, but its operand did not have the syntax of a file name.

Cannot nest ! command
A **!** subcommand cannot contain a **!** subcommand of its own.

***subcommand* command needs a label**
The specified subcommand required a label, but you did not supply one.

must have at least one (noncomment) command
The input to **sed** must contain at least one active subcommand (that is, a subcommand that is not a comment).

no remembered regular expression
You issued a subcommand that tried to use a remembered regular expression—for example, s//abc. However, there is no remembered regular expression yet. To correct this, change the subcommand to use an explicit regular expression.

Limits

sed allows a limit of 1024 bytes per line and 28 000 lines per file. It does not allow the NUL character. The maximum length of a global command is 256 characters, including newlines.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–E** option is an extension of the POSIX standard and is unique to this version of **sed**.

Related Commands

awk, **diff**, **ed**, **grep**, **regexp** (see Appendix B, "Regular Expressions (regexp)," on page 471)

set — Set or unset command options and positional parameters

---

set [+|-abCefhiKkLmnpstuvx–] [+|-o[*flag*]] [+|-A *name*][*parameter ...*]

Purpose

Calling **set** without arguments displays the names and values of all environment variables, sorted by name, in the following format:

```
Variable="value"
```

The quoting allows the output to be reinput to the shell using the built-in command **eval**. Arguments of the form **–***option* set each shell flag specified as an option. Similarly, arguments of the form **+***option* turn off each of the shell flags specified as an option. (Contrary to what you might expect, **–** means *on,* and + means *off*.)

Note: You can set the positional *parameter*s, and all the shell flags except **–s**, on the shell command line at invocation.

Options

–a
> Sets all subsequently defined variables for export.

–b
> Notifies you when background jobs finish running.

–C
> Prevents the output redirection operator > from overwriting an existing file. Use the alternate operator >| to force an overwrite.

–e
> Tells a noninteractive shell to execute the ERR trap and then exit. This flag is disabled when reading profiles.

–f
> Disables file name generation.

–h
> Makes all commands use tracked aliases.

–i
> Makes the shell interactive.

–K
> Tells the shell to use KornShell-compatible behavior in any case where the POSIX.2 behavior is different from the behavior specified by the KornShell.

–k
> Allows assignment parameters anywhere on the command line and still includes them in the environment of the command.

–L
> Makes the shell a login shell. Setting this flag is effective only at shell invocation.

–m
> Runs each background job in a separate process group and reports on each as they complete.

–n
> Tells a noninteractive shell to read commands but not run them.

–o *flag*
> Sets a shell *flag*. If you do not specify *flag*, this option lists all shell flags that are currently set. *flag* can be one of the following:

allexport
> Is the same as the **–a** option.

errexit
> Is the same as the **–e** option.

bgnice
> Runs background jobs at a lower priority.

emacs
> Specifies **emacs** style in-line editor for command entry. This is accepted, but has no effect.

gmacs
> Specifies **gmacs** style in-line editor for command entry. This is accepted, but has no effect.

ignoreeof
> Tells the shell not to exit at the end of the file.

interactive
> Is the same as the **–i** option.

jdebug
> Starts tracing the internal shell operation for debugging the shell.

keyword
> Is the same as the **–k** option.

korn
> This is the same as the **–K** option.

logical
> Specifies that **cd, pwd** and the **PWD** variable use logical path names in directories with symbolic links. If this flag is not set, these built-ins and **PWD** use physical directory path names. For example, assume **/usr/spool** is a symbolic link to **/var/spool**, and that it is your current directory. If **logical** is not set, **PWD** has the value **/var/spool**, and **cd. .** changes the current directory to **/var**. If **logical** is set, **PWD** has the value **/usr/spool** and **cd. .** changes the current directory to **/usr**.

login
> Is the same as the **–L** option of **sh**.

markdirs
> Adds a trailing slash (/) to file name-generated directories.

monitor
> Is the same as the **–m** option.

noclobber
> Is the same as the **–C** option.

noexec
> Is the same as the **–n** option.

noglob
> Is the same as the **–f** option.

nolog
> Does not record function definitions in the history file.

notify
> Is the same as the **–b** option.

nounset
> Is the same as the **–u** option.

privileged
> Is the same as the **–p** option.

trackall

Is the same as the **–h** option.

verbose

Is the same as the **–v** option.

vi

Specifies **vi** style in-line editor for command entry. This is accepted, but has no effect.

xtrace

Is the same as the **–x** option.

–p

Resets the **PATH** variable to the default value, disables processing of **$HOME/.profile**, and ignores the value of the **ENV** variable.

–s

Sorts the positional parameters.

–t

Exits after reading and running one command.

–u

Tells the shell to issue an error message if an unset parameter is used in a substitution.

–v

Prints shell input lines as they are read.

–x

Prints commands and their arguments as they run.

Other options:

–

Turns off the **–v** and **–x** options. Also, parameters that follow this option do not set shell flags, but are assigned to positional parameters (see **sh**).

––

Specifies that parameters following this option do not set shell flags, but are assigned to positional parameters.

+A *name*

Assigns the parameter list to the elements of *name*, starting at *name*[0].

–A *name*

Unsets *name* and then assigns the parameter list to the elements of *name* starting at *name*[0].

Usage Notes

This command is built into the shell.

Environment Variables

PATH

Contains a list of directories that constitute the search path for executable utilities.

Exit Values

Possible exit status values are:

0

Successful completion

1

Failure due to an incorrect command-line argument

2

Failure resulting in a usage message, usually due to a missing argument

Portability

POSIX.2, X/Open Portability Guide. Several shell flags are extensions of the POSIX standard: **bgnice**, **ignoreeof**, **keyword**, **markdirs**, **monitor**, **noglob**, **nolog**, **privileged**, and **trackall** are extensions of the POSIX standard, along with the shell flags **±A**, **±h**, **±k**, **±p**, **±s**, and **±t**.

Related Commands

alias, **eval**, **export**, **sh**, **trap**, **typeset**

sh — Invoke a shell

[r]sh [-abCefhiKkLmnprtuvx] [-o *option*] [*cmd_file* [*argument...*]]
[r]sh -c *cmdstring* [-abCefhiKkLmnprtuvx] [-o *option*] [*cmd_name* [*argument...*]]
[r]sh -s [-abCefhikLmnprtuvx] [-o *option*] [*argument...*]

Purpose

sh contains the following subsections:

- Options and invocation
- Command syntax
- Command execution
- Quoting
- Directory substitution
- Parameter substitution
- Arithmetic substitution
- Command substitution
- File descriptors and redirection
- File name generation
- Variables
- Shell execution environments
- Built-in commands

Subsections dealing with substitution and interpretation of input appear in the order in which the shell performs those substitutions and interpretations.

Much of what the shell can do is provided through such built-in commands as **cd** and **alias**.

Invocation Options

The OpenExtensions shell, based on the KornShell, is upward-compatible with the Bourne shell.

Normally you invoke the shell with OPENVM SHELL. You can also invoke the shell by typing an explicit **sh** command. Some people find it useful to copy the **sh** file into a file named **rsh**. If you invoke the shell under the name **rsh**, the shell operates in *restricted* mode. This mode is described in connection with **–r**.

If you invoke the shell with a name that begins with the – character, it is a *login shell*. (You can also get a login shell if you invoke the shell with the **–L** option.) A login shell begins by running the file **/etc/profile**. It then runs **$HOME/.profile** using the **.** command (see **dot**). If **$HOME** is not set, the shell searches the working directory for:

```
.profile
```

and runs this file with the **.** command if it exists. You do not get an error message if any of these files cannot be found.

You can use these profile files to customize your session with **sh**. For example, your profile files can set options, create aliases, or define functions and variables.

If there is at least one argument on the **sh** command line, **sh** takes the first argument as the name of a shell script to run. (The exception to this is when **–s** is used.) Any additional arguments are assigned to the positional parameters; usually, these serve as arguments to the shell script. See "Parameter

Substitution" on page 283 for information about positional parameters, and see **set** for information about changing these parameters.

If **sh** finds the **ENV** environment variable set when it begins running (after profile processing), **sh** runs the file named by the expansion of the value of this variable (see "Variables" on page 290).

Command Options

The shell accepts the following options on the command line:

–c *cmdstring*
> Runs *cmdstring* as if it were an input line to the shell and then exits. This is used by programs (for example, editors) that call the shell for a single command. **sh** assigns arguments after *cmdstring* to the positional parameters. If you specify *cmd_name,* special parameter 0 is set to this string for use when running the commands in *cmdstring*.

–i
> Invokes an interactive shell, as opposed to running a script. With **–i**, the shell catches and ignores interrupts. Without **–i**, an interrupt ends the shell. For shells that read from the terminal, **–i** is the default.

–K
> Specifies KornShell-compatible behavior where the POSIX.2 behavior is different from the behavior specified by the KornShell. Without -K, the shell defaults to POSIX.2 behavior.

–L
> Makes the shell a *login shell,* as described earlier.

–r
> Invokes a restricted shell. (As noted earlier, you can also invoke a restricted shell by using the name **rsh**). In a restricted shell, you cannot do the following: use the **cd** command; change the values of the variables **ENV**, **PATH**, or **SHELL**; use > or >> to redirect output; or specify command names containing /. These restrictions do not apply during execution of your *profile* files.

–s
> Reads commands from standard input and assigns all *argument*s to the positional parameters. Normally, if there is at least one *argument* to the shell, the first such *argument* is the name of a file to run.

If you do not give either the **–c** or **–s** option, but you do specify *cmd_file,* the shell takes it as the name of a file that contains commands to be run. Special parameter 0 is set to this name.

In addition to these options, you can use any valid option to the **set** command (including **–o** *option*) as a command-line option to **sh**. See **set** for details.

Command Syntax

The shell implements a sophisticated programming language that gives you complete control over the execution and combination of individual commands. When the shell scans its input, it always treats the following characters specially:

```
*   ;   &   (   )   <   >   |   '   \   "
space   tab   newline
```

If you want to use any of these characters inside an actual argument, you must quote the argument (so that the shell doesn't use the special meanings of the characters). See "Quoting" on page 282 for more information.

A *simple command* is a list of *argument*s separated by characters in the **IFS** environment variable (the default value of **IFS** has blank, tabs, and newlines).

When a word is preceded by an unescaped pound sign (#), the remainder of the line is treated as a *comment,* and the shell discards input up to but not including the next newline. When a command starts with a defined alias, **sh** replaces the alias with its definition (see **alias**).

A *reserved-word command* starts with a *reserved word* (for example, **if**, **while**, or **for**). Reserved-word commands provide flow of control operations for the shell. These are described in "Reserved-Word Commands" on page 280.

A *command* can be any of the following:

command:

> *simple command*
> *reserved-word command*
> (*command*)
> *command* | *command*
> *command* && *command*
> *command* || *command*
> *command* & *command*
> *command* &
> *command* |&
> *command* ; *command*
> *command* ;
> *command*<newline>

The following is the order of precedence of the preceding operators. The highest priority operators are listed first, and operators on the same line have equal priority.

```
()
|
&&      ||
&       |&        ;        <newline>
```

The meaning of these operations is as follows:

(*command*)
Runs *command* in a subshell. The current shell invokes a second shell, and this second shell actually runs *command*. In this way, *command* runs in a completely separate execution environment; it can change working directories, change variables, open files, and so on without affecting the first shell. The subshell's environment begins as a copy of the current environment, so the value of the **ENV** environment variable is not run when a subshell starts.

|
Creates a pipe between the two *command*s that the **|** operator connects. The standard output of the first *command* becomes the standard input of the second *command*. A series of commands connected by pipes is called a *pipeline*. The exit status is that of the last command in the pipeline.

&&
Is the logical AND operator. The shell runs the second *command* if and only if the first *command* returns a true (zero) exit status.

||
This is the logical OR operator. The shell runs the second *command* if and only if the first *command* returns a false (nonzero) exit status.

&
Runs the *command* that precedes it asynchronously. The shell just starts the *command* running and then immediately goes on take new input, before the *command* finishes execution. On systems where asynchronous execution is not possible, this operation is effectively equivalent to ; .

|&
Runs the *command* that precedes it as a co-process. The *command* runs asynchronously, as with the & operator, but the *command*'s standard input and standard output are connected to the shell by pipes. The shell sends input to *command* 's standard input with the **print –p** command, and reads from *command*'s standard output with the **read –p** command. The *command* should not buffer its output. Because of this and other limitations, co-processes should be designed to be used as co-processes. On systems where asynchronous execution is not possible, co-processes are not supported.

;
> Is the sequential execution operator. The second *command* is run only after the first *command* has completed.

newline
> The unescaped newline is equivalent to the **;** operator.

Reserved-Word Commands

The shell contains a rich set of *reserved-word commands*, which provide flow of control and let you create compound commands. In the following list, a *command* can also be a sequence of *command*s separated by newlines. Square brackets (*[ ]*) indicate optional portions of commands, and are never part of the command syntax.

! *command*
> The exclamation point is the logical NOT operator. When *command* returns false (nonzero), **!** returns true (zero). When *command* returns true (zero), **!** returns false (nonzero).

{ *command*;**}**
> Enclosing a command in braces is similar to the (*command*) construct, except that the shell runs the *command* in the same environment rather than under a subshell. { and } are simply reserved words to the shell. To make it possible for the shell to recognize these symbols, you must put a blank or newline after the {, and a semicolon or newline before the }.

case *word* **in |** *[([pattern[|pattern] ...)command ;;] ... | [([pattern[| pattern] ...)command ;;] ...* **| esac**
> The **case** statement is similar to the **switch** statement of the C programming language or the **case** statement of Pascal. If the given *word* matches any one of the *pattern*s separated by "or" bar (**|**) characters, **sh** runs the corresponding *command*. The *pattern*s should follow the rules given in "File Name Generation" on page 289, except that the period (**.**) and slash (/) are not treated specially. Patterns are matched in the order they are given, so more inclusive patterns should be mentioned later. You must use the double semicolon (**;;**) to delimit *command* and introduce the next *pattern*.

for *variable* **[in** *word ...***] | do** *command* **| done**
> The **for** statement sets *variable* to each *word* argument in turn, and runs the set of *command*s once for each setting of *variable*. If you omit the **in** *word* part, **sh** sets *variable* to each positional parameter. You can divert the flow of control within the loop with the **break** or **continue** statements.

function *variable* **{ |** *command* **| } |** *variable*() **{ |** *command* **| }**
> Any one of these forms defines a **function** named *variable,* the body of which consists of the sequence of *command*s. You invoke a function just like any other command; when you actually call the function, **sh** saves the current positional parameters. The function's command-line arguments then replaces these parameters until the function finishes. **sh** also saves the current ERR and EXIT traps and any flags manipulated with the **set** command; these are restored when the function finishes. The function ends either by falling off the end of the code of the function body, or by reaching a **return** statement. If the function uses **typeset** to declare any variables in the function body, the variables are local to the function.

if *command* **| then** *command* **| [elif** *command* **| then** *command] ...* **| [else** *command] | **fi**
> In the **if** statement, if the first (leftmost) *command* succeeds (returns a zero exit status), **sh** runs the *command* following **then**. Otherwise, **sh** runs the *command* (if any) following the **elif** (which is short for "else if"); if that succeeds, **sh** runs the *command* following the next **then**. If neither case succeeds, **sh** runs the *command* following the **else** (if any).

select *variable* **[in** *word ...***] | do** *commands* **| done**
> The **select** statement can handle menulike interactions with the user. Its syntax is like the **for** statement. Each *word* is printed on the standard error file, one per line, with an accompanying number. If you omit the "**in** *word ...*" part, **sh** uses the positional parameters. **sh** then displays the value of the variable **PS3** to prompt the user to enter a numerical reply. If the reply is an empty line, **sh** displays the menu again; otherwise, **sh** assigns the input line to the variable **REPLY**, sets *variable* to the *word* selected, and then runs the *commands*. **sh** does this over and over until the loop is ended by an interrupt, an end-of-file, or an explicit **break** statement in the *commands*.

until *command1* **| do** *command2* **| done**
> The **until** statement runs *command1* and tests its exit status for success (zero) or failure (nonzero). If *command1* succeeds, the loop ends; otherwise, **sh** runs *command2* and then goes back to run and test *command1* again. **break** and **continue** commands in the *commands* can affect the operation of the loop.

while *command1* **| do** *command2* **| done**
> The **while** statement works similarly to the **until** statement. However, the loop ends whenever *command1* is unsuccessful (nonzero exit status).

Shell reserved words are recognized only when they are the unquoted first token of a command. This lets you pass these reserved words as arguments to commands run from the shell. The full list of reserved words is:

!	**elif**	**if**
{	**else**	**select**
}	**esac**	**then**
case	**fi**	**time**
do	**for**	**until**
done	**function**	**while**

Command Execution

Before running a *simple command,* the shell processes the command line, performing expansion, assignments, and redirection.

First, **sh** examines the command line and divides it into a series of *tokens,* which are either *operators* or *words*. An operator is either a control operator (described in "Command Syntax" on page 278) or a redirection operator (described in "File Descriptors and Redirection" on page 288). A word is any token that is not an operator.

Next, the shell expands words in the following order:

1. **sh** performs directory substitution (see "Directory Substitution" on page 283).
2. **sh** performs parameter substitution, command substitution, or arithmetic substitution, as appropriate, in the order that the words appear on the command line, expanding each word to a *field* (see the appropriate sections).
3. **sh** scans each field produced in step "2" on page 281 for unquoted characters from the **IFS** environment variable and further subdivides this field into one or more new fields.
4. **sh** expands any aliases to their definitions (see "alias — Display or create a command alias" on page 6).
5. **sh** performs path name expansion on each unquoted field from step "3" on page 281 (see "File Name Generation" on page 289).
6. **sh** removes all quote mechanisms (\, ', and ") that were present in the original word unless they have themselves been quoted (see "Quoting" on page 282).

The shell considers the first field of the expanded result to be a command.

The expanded simple command can contain variable assignments and redirections. Variable assignments affect the current execution environment. After expansion, the shell handles all redirection constructs, and the command, if one was found, it performs the redirection in a subshell environment (see "Shell Execution Environments" on page 290).

When a simple command contains a command name, variable assignments in the command affect only the execution of that command.

After the shell has expanded all appropriate arguments in a simple command, but before it performs file name generation, it examines the command name (if the command has one). **sh** checks the names against currently defined aliases (see **alias**) and functions (see **function** under "Command Syntax" on

page 278), and finally against the set of *built-in commands*: commands that the shell can run directly without searching for program files. Built-in commands are described in "Built-In Commands" on page 291.

If the command name is not a function or a built-in command, the shell looks for a program file or script file that contains an executable version of that command. The OpenExtensions shell uses the following procedure to locate the program file:

- If the command name typed to the shell has slash (/) characters in its name, the command is taken to be a full path name (absolute or relative). The shell tries to execute the contents of that file.
- Otherwise, the shell performs a *path search*. To do this, the shell obtains the value of the **PATH** environment variable. The value should be a list of directory names. **sh** searches under each directory for a file, the name of which matches the command name. **sh** runs the first matching file found.

Command names can be marked as tracked aliases. The first time you run a command with a tracked alias, the shell does a normal **PATH** search. If the search is successful, the shell remembers the file that it finds. The next time you run a command with the same name, **sh** immediately runs the file found on the last **PATH** search; there is no new search. This speeds up the time that it takes the shell to find the appropriate file.

The **set –h** command tells the shell that all commands should be treated as tracked aliases. See **alias** and **set** for more information.

Quoting

To let you override the special meaning of certain *words* or special characters, the shell provides several quoting mechanisms. In general, you can turn off the special meaning of any character by putting a backslash (\) in front of the character. This is called *escaping* the character.

For example, you can tell the shell to disregard the special meaning of the newline character by putting a backslash at the very end of a line. The shell ignores the escaped newline, and joins the next line of input to the end of the current line. In this way, you can enter long lines in a convenient and readable fashion.

Escaping characters by putting a backslash in front of them is the most direct way of telling the shell to disregard special meanings. However, it can be awkward and confusing if you have several characters to escape.

As an alternative, you can put arguments in various types of quotes. Different quotation mark characters have different "strengths." The single quotation mark characters are the strongest. When you enclose a command-line argument in single quotation mark characters, the shell disregards the special meanings of everything inside the single quotation marks. For example:

```
echo '*'
```

displays just the ∗ character.

Double quotation mark characters are weaker. Inside double quotation marks, the shell performs command substitutions of the form:

```
$(command)
```

or:

```
command
```

(See "Command Substitution" on page 287.) The shell does not perform such substitutions when they appear inside single quotation marks. In addition, the shell performs parameter substitutions of the form:

```
$parameter
```

when they are inside double quotation marks but not when they're inside single quotation marks (see "Parameter Substitution" on page 283). You can use the backslash to escape another character when

they appear inside double quotation marks, but inside single quotation marks the shell ignores this special meaning.

The shell treats internal field separator characters (that is, characters in the value of the **IFS** variable) literally inside quoted arguments, whether they're quoted with double quotation marks or single quotation marks. This means that a quoted argument is considered a single entity, even if it contains **IFS** characters.

Quoting can override the special meanings of reserved words and aliases. For example, in:

```
"time" program
```

the quotes around **time** tell the shell not to interpret **time** as a shell reserved word. Instead, **sh** does a normal command search for a command named **time**.

You must always quote the following characters if you want **sh** to interpret them literally:

```
|   &   ;   <   >   (   )   $   '   "   `   \
<space>  <tab>  <newline>
```

The following characters need to be quoted in certain contexts if they are to be interpreted literally:

```
*   ?   [   #   %   =   ~
```

Directory Substitution

When a word begins with an unquoted tilde (**~**), **sh** tries to perform *directory substitution* on the word. **sh** obtains all characters from the tilde (**~**) to the first slash (/) and uses this as a *user name*. **sh** looks for this *name* in the user profile, the file that contains information on all the system's users. If **sh** finds a matching name, it replaces *~name* with the name of the user's *home directory,* as given in the matching POSIX user database.

For example, if you specify a file name as:

```
 ~jsmith/file
sh would look up jsmith's home
directory and put that directory name in place
of the  ~jsmith
construct.
```

If you specify a **~** without an accompanying name, **sh** replaces the **~** with the current value of your **HOME** variable (see "Environment Variables" on page 292). For example:

```
echo  ~
displays the name of your home directory. Similarly, sh replaces
the construct  ~+ with the value of the PWD variable (the name of the
your working directory), and replaces the tilde hyphen ( ~-) with the
value of OLDPWD (the name of your previous working directory). In variable
assignments, tilde expansion is also performed after colons (:).
```

Parameter Substitution

The shell uses three types of parameters: positional parameters, special parameters, and variables. A positional parameter is represented with either a single digit (except 0) or one or more digits in curly braces. For example, 7 and {15} are both valid representations of positional parameters. Positional parameters are assigned values from the command line when you invoke **sh**.

A special parameter is represented with one of the following characters:

```
*   @   #   ?   !   -   $   0
```

The values to which special parameters expand are listed in the following paragraphs.

Variables are named parameters. For details on naming and declaring variables, see "Variables" on page 290.

The simplest way to use a parameter in a command line is to enter a dollar sign ($) followed by the name of the parameter. For example, if you enter the command:

```
echo $x
```

sh replaces $x with the value of the parameter *x* and then displays the results (because **echo** displays its arguments). Other ways to expand parameters are shown in the following paragraphs.

The following parameters are built in to the shell:

$1, $2, ... $9
Expands to the *d* positional parameter (where *d* is the single digit following the $). If there is no such parameter, $*d* expands to a null string.

$0
Expands to the name of the shell, the shell script, or a value assigned when you invoked the shell.

$#
Expands to the number of positional parameters.

$@
Expands to the complete list of positional parameters. If **$@** is quoted, the result is separate arguments, each quoted. This means that:

```
"$@"
```

is equivalent to:

```
"$1" "$2" ...
```

$*
Expands to the complete list of positional parameters. If **$*** is quoted, the result is concatenated into a single argument, with parameters separated by the first character of the value of **IFS** ("Variables" on page 290). For example, if the first character of **IFS** is a blank, then:

```
"$*"
```

is equivalent to:

```
"$1 $2 ..."
```

$−
Expands to all options that are in effect from previous calls to the **set** command and from options on the **sh** command line.

$?
Expands to the exit status of the last command run.

$$
Expands to the current process number of the original parent shell.

$!
Expands to the process number of the last asynchronous command.

These constructs are called *parameters* of the shell. They include the positional parameters, but are not restricted to the positional parameters.

We have already mentioned that you can expand a parameter by putting a $ in front of the parameter name. More sophisticated ways to expand parameters are:

${*parameter*}
Expands any parameter.

${*number*}

Expands to the positional parameter with the given number. (Remember that if you just enter $*d* to refer to the *d*th positional parameter, *d* can only be a single digit; with brace brackets, *number* can be greater than 9.) Since braces mark the beginning and end of the name, you can have a letter or digit immediately following the expression.

${*variable*[*arithmetic expression*]}

Expands to the value of an element in an array named *variable*. The *arithmetic expression* gives the subscript of the array. (See "Arithmetic Substitution" on page 286.)

${*variable*[*]}

Expands to all the elements in the array *variable*, separated by the first character of the value of **$IFS**

${*variable*[@]}

When unquoted, is the same as **${*variable*[*]}**. When quoted as "**${*variable*[@]}**," it expands to all the elements in the array *variable*, with each element quoted individually.

${#*parameter*}

Expands to the number of characters in the value of the given *parameter*.

${#*}, ${#@}

Expands to the number of positional parameters.

${#*variable*[*]}

Expands to the number of elements in the array named *variable*. Elements that do not have assigned values do not count. For example, if you only assign values to elements 0 and 4, the number of elements is 2. Elements 1 through 3 do not count.

${*parameter*:–*word*}

Expands to the value of *parameter* if it is defined and has a nonempty value; otherwise, it expands *word*. This means that you can use *word* as a default value if the parameter isn't defined.

${*parameter*–*word*}

Is similar to the preceding construct, except that the parameter is expanded if defined, even if the value is empty.

${*variable*:=*word*}

Expands *word* with parameter expansion and assigns the result to *variable*, provided that *variable* is not defined or has an empty value. The result is the expansion of *variable*, whether or not *word* was expanded.

${*variable*=*word*}

Is similar to the preceding construct, except that the *variable* must be undefined (it cannot just be null) for *word* to be expanded.

${*parameter*:?*word*}

Expands to the value of *parameter* provided that it is defined and non-empty. If *parameter* isn't defined or is null, **sh** expands and displays *word* as a message. If *word* is empty, **sh** displays a default message. After a noninteractive shell has displayed a message, it ends.

${*parameter*?*word*}

Is similar to the preceding construct, except that **sh** displays *word* only if *parameter* is undefined.

${*parameter*:+*word*}

Expands to *word*, provided that *parameter* is defined and nonempty.

${*parameter*+*word*}

Expands to *word*, provided that *parameter* is defined.

${*parameter*#*pattern*}

Attempts to match *pattern* against the value of the specified *parameter*. The *pattern* is the same as a case *pattern*. **sh** searches for the shortest prefix of the value of *parameter* that matches *pattern*. If **sh** finds no match, the previous construct expands to the value of *parameter*; otherwise, the portion of the value that matched *pattern* is deleted from the expansion.

${*parameter*##*pattern*}

Is similar to the preceding construct, except that **sh** deletes the longest part that matches *pattern* if it finds such a match.

${*parameter%pattern*}
> Searches for the shortest suffix of the value of *parameter* matching *pattern* and deletes the matching string from the expansion.

${*parameter%%pattern*}
> Is similar to the preceding construct, except that **sh** deletes the longest part that matches *pattern* if it finds such a match.

Arithmetic Substitution

Arithmetic substitution is available with the syntax:

```
$((arithmetic expression))
```

or:

```
$[arithmetic expression]
```

This sequence is replaced with the value of *arithmetic expression*. Arithmetic expressions consist of expanded variables, numeric constants, and operators. Numeric constants have the form:

```
[base#]number
```

where the optional *base* is a decimal integer between 2 and 36 inclusive, and *number* is any nonnegative number in the given base. The default base is 10. Undefined variables evaluate to zero.

The operators are listed in decreasing order of precedence in Table 9 on page 286. Operators sharing a heading have the same precedence. Evaluation within a precedence group is from left to right, except for the assignment operator, which evaluates from right to left.

Table 9. Shell Operators

Category	Function
Unary Operators	
-	Unary minus
!	Logical negation
+ ~	Identity, bitwise negation
Multiplicative Operators	
* / %	Multiplication, division, remainder
Additive Operators	
+ -	Addition, subtraction
Bitwise Shift Operators	
<< >>	Bitwise shift right, bitwise shift left
Relational Operators	
< >	Less than, greater than
<= >=	Less than or equal, greater than or equal
= = !=	Equal to, not equal to
Bitwise AND Operator	
&	AND
Bitwise Exclusive OR Operator	

Table 9. Shell Operators (continued)

Category	Function
^	Exclusive OR
Bitwise Inclusive OR Operator	
|	Inclusive OR
Logical AND Operator	
&&	Logical AND
Logical OR Operator	
||	Logical OR
Conditional Operator	
? :	If-else
Assignment Operator	
= *= /= %= += -= <<= >>= &= ^= |=	Assignment

Arithmetic expressions can be used without the enclosing $((and)) in assignment to an integer variable (see **typeset**) as an argument to the following built-in commands:

break	**exit**	**return**
continue	**let**	**shift**

and when used as arguments in **test** numeric comparisons (**–eq**, **–ge**, **–gt**, **–le**, **–lt**, and **-ne**) (see **test**).

Command Substitution

In *command substitution,* **sh** uses the expansion of the standard output of one command in the command line for a second command. There are two syntaxes.

The first syntax (called *backquoting*) surrounds a command with grave accents `, as in:

```
ls -l `cat list`
```

To process this command line, **sh** first runs the **cat** command and collects its standard output. The shell then breaks this output into arguments and puts the result into the command line of the **ls** command. The previous command therefore lists the attributes of all files, the names of which are contained in the file **list**.

This syntax is easy to type, but is not useful if you want to put one command substitution inside another (*nesting* command substitutions). A more useful syntax is:

```
$(command)
```

as in:

```
ed $(grep -f -l function $(find . -name '*.c'))
```

This command uses **find** to search the current directory and its subdirectories to find all files, the names of which end in **.c**. It then uses **grep –f** to search each such file for those that contain the string `function`. Finally, it calls **ed** to edit each such file.

There is a historical inconsistency in the backquoting syntax. A backslash (\) within a backquoted command is interpreted differently depending on its context. Backslashes are interpreted literally unless they precede a dollar sign ($), grave accent (` ), or another backslash (\). In these cases, the leading backslash becomes an escape character to force the literal interpretation of the $, `, or \. Consequently, the command:

```
echo '\$x'
```

issued at system level produces the output:

```
\$x
```

whereas the same command nested in a backquoted syntax:

```
echo `echo '\$x'`
```

produces the output:

```
$x
```

We recommend the $(*command*) syntax for command substitutions.

sh performs command substitutions as if a new copy of the shell is invoked to run the command. This affects the behavior of $- (standing for the list of options passed to the shell). If a command substitution contains $-, the expansion of $- does not include the **–i** option, since the command is being run by a noninteractive shell.

File Descriptors and Redirection

The shell sometimes refers to files using *file descriptors*. A file descriptor is a number in the range 0 to 9. It may have any number of digits. For example, the file descriptors 001 and 01 are identical to file descriptor 1. Various operations (for example, **exec**) can associate a file descriptor with a particular file.

Some file descriptors are set up at the time the shell starts up. These are the standard input/output streams:

- Standard input (file descriptor 0)
- Standard output (file descriptor 1)
- Standard error (file descriptor 2)

Commands running under the shell can use these descriptors and streams too. When a command runs under the shell, the streams are normally associated with your terminal. However, you can *redirect* these file descriptors to associate them with other files (so that I/O on the stream takes place on the associated file instead of your terminal). In fact, the shell lets you redirect the I/O streams associated with file descriptors 0 through 9, using the following command-line constructs.

number<file
> Uses *file* for input on the file descriptor, the number of which is *number*. If you omit *number*, as in <*file*, the default is 0; this redirects the standard input.

number>file
> Uses *file* for output on the file descriptor, the number of which is *number*. If you omit *number*, as in >*file*, the default is 1; this redirects the standard output. The shell creates the file if it doesn't already exist. The redirection fails if the file already exists and **noclobber** is set (see **set**).

number>|file
> Is similar to *number>file* but if *file* already exists, the output written to the file overwrites its current contents.

number< >file
> Uses *file* for input and output with the file descriptor, the number of which is *number*. This is most useful when the file is another terminal or modem line. If you omit *number*, as in < >*file*, the default

number is zero; this redirects the standard input. Output written to the file overwrites the current contents of the file (if any). The shell creates the file if it doesn't already exist.

number>>name
> Is similar to *number > file,* except that output is appended to the current contents of the file (if any).

number<<[-]name
> Lets you specify input to a command from your terminal (or from the body of a shell script). This notation is known as a *here-document.* The shell reads from the standard input and feeds that as input to file descriptor *number* until it finds a line that exactly matches the given *name.* If you omit *number,* the default is the standard input. For example, to process the command:

```
cat <<abc >out
```

> the shell reads input from the terminal until you enter a line that consists of the word abc. This input is passed as the standard input to the **cat** command, which then copies the text to the file **out**.

> If any character of *name* is quoted or escaped, **sh** does not perform substitutions on the input; instead, it performs variable and command substitutions, respecting the usual quoting and escape conventions. If you put - before *name,* **sh** deletes all leading tabs in the *here-document.*

number1<&number2
> Makes the input file descriptor *number1* a duplicate of file descriptor *number2*. If you omit *number1,* the default is the standard input (file descriptor 0). For example, <&4 makes the standard input a duplicate of file descriptor 4. In this case, entering input on 4 has the same effect as entering input on the standard input.

number1>&number2
> Makes the output file descriptor *number2* a duplicate of file descriptor *number2*. If you omit *number2,* the default is the standard output (file descriptor 1). For example, >&2 makes the standard output a duplicate of file descriptor 2 (the standard error). In this case, writing output on the standard output has the same effect as writing output on the standard error.

number<&-
> Closes input descriptor *number*. If you omit *number*, it closes the standard input.

number>&-
> Closes output descriptor *number*. If you omit *number*, it closes the standard output.

Normally, redirection applies only to the command where the redirection construct appears; however, see **exec**.

The order of *redirection* specifications is significant, since an earlier redirection can affect a later one. However, these specifications can be freely intermixed with other command arguments. Since the shell takes care of the redirection, the redirection constructs are not passed to the command itself.

Note: The shell performs the implicit redirections needed for pipelines before performing any explicit redirections.

File Name Generation

The characters * ? [are called *glob characters,* or *wildcard characters*. If an unquoted argument contains one or more glob characters, the shell processes the argument for file name generation. The glob characters are part of *glob patterns,* which represent file and directory names. These patterns are similar to regular expressions, but differ in syntax, since they are intended to match file names and words (not arbitrary strings). The special constructions that may appear in glob patterns are:

?
> Matches exactly one character of a file name, except for the separator character / and a . at the beginning of a file name. ? only matches an actual file name character and does not match nonexistent characters at the end of the file name. ? is analogous to the metacharacter . in regular expressions.

*
Matches zero or more characters in a file name, subject to the same restrictions as ?. ∗ is analogous to the regular expression .∗.

[*chars*]
Defines a *class* of characters; the glob pattern matches any single character in the class. A class can contain a range of characters by writing the first character in the range, a dash -, and the last character. For example, [A-Za-z], in the POSIX locale, stands for all the uppercase and lowercase letters. If you want a literal - character in the class, put it as the first or last character inside the brackets. If the first character inside the brackets is an exclamation mark (!), the pattern matches any single character that is *not* in the class.

Some sample patterns are:

[!a-f]*.c
Matches all **.c** files beginning with something other than the letters from a through f.

/???/?.?
Matches all files that are under the root directory in a directory with a three-letter name, and that have a basename containing one character followed by a . followed by another single character.

/.[chyl]
Matches all **.c**, **.h**, **.y**, and **.l** files in a subdirectory of the working directory.

~mks/*.ksh
Matches all shell scripts in the home directory of user mks (see "Directory Substitution" on page 283 for the use of ~).

If no files match the pattern, **sh** leaves the argument untouched. If the **set** option **–f** or **"–o noglob"** is in effect, the shell does not perform file name generation.

Variables

The shell maintains variables and can expand them where they are used in command lines; see "Parameter Substitution" on page 283 for details.

A variable name must begin with an uppercase or lowercase letter or an underscore (_). Subsequent characters in the name, if any, can be uppercase or lowercase letters, underscores, or digits 0 through 9. You can assign a value to a variable with:

```
variable=value
```

You can implicitly declare a variable as an array by using a subscript expression when assigning a value, as in:

```
variable[arithmetic expression]=value
```

You can use a subscripted array variable anywhere that the shell allows an ordinary variable. See "Arithmetic Substitution" on page 286 for the syntax of an *arithmetic expression*. Also see **typeset**, **export**, and **readonly** for details about the attributes of shell variables, and how shell variables can be exported to child processes.

For a list of variables that the shell either sets or understands, see "Environment Variables" on page 292.

Shell Execution Environments

A shell execution *environment* is the set of conditions affecting most commands run within the shell. It consists of:

- Open files
- The working directory (see **cd**)
- The file creation mask (see **umask**)

- The traps currently set (see **trap**)
- The shell parameters (see **set** and **export**)
- The shell functions currently defined (see "Command Execution" on page 281)
- Options (see **set**)

A *subshell environment* starts as a duplicate of the shell environment, except that traps caught by the shell are set to default values in the subshell. Since the subshell environment starts as a duplicate, the value of the **ENV** environment variable is not run. Changes made to a subshell environment do not affect the shell environment.

Command substitutions, commands within parentheses "(*command*)," and commands to be run asynchronously ("*command*&")—all run in subshell environments. Each command in a pipeline "*command***|***command*" runs in a subshell environment.

Shell utilities also run in a separate environment that does not affect the shell environment, except for certain built-in utilities (for example, **cd** and **umask**) that explicitly alter the shell environment. The environment of a shell utility is set up by the shell to include the following:

- Open files, subject to redirection.
- Working directory (see **cd**).
- File creation mask (see **umask**).
- Traps; traps caught by the shell are set to default values and traps ignored by the shell are ignored by the utility.
- Variables defined inside the shell and having the export attribute.

Built-In Commands

This section lists the commands that are built into the shell. Such commands are built into the shell to increase performance of shell scripts or to access the shell's internal data structures and variables. These internal commands are designed to have semantics indistinguishable from external commands.

:	exec	newgrp	times
.	exit	print	trap
alias	export	pwd	type
bg	false	read	typeset
break	fc	readonly	umask
cd	fg	return	unalias
command	getopts	set	unset
continue	jobs	shift	wait
echo	kill	test	whence
eval	let	time	

POSIX.2 recognizes a subset of these commands as *special* built-ins. Syntax errors in special built-in commands cause a noninteractive shell to exit with the exit status set by the command. The special built-in utilities are:

:	eval	readonly	trap
.	exec	return	typeset
break	exit	set	unset
continue	export	shift	

As well as built-in commands, the shell has a set of predefined aliases:

functions	**integer**	**stop**
hash	**nohup**	**suspend**
history	**r**	

See **alias** for details.

Examples

Software distributed over computer networks such as Usenet is often distributed in a form known as a *shell archive*. In essence, a shell archive is a shell script containing the data of one or more files, plus commands to reconstruct the data files and check that the data was sent correctly. The following shows a sample shell archive:

```
# This is a shell archive.
# It contains the one file "frag.ksh"
# To extract contents, type
# sh file
#
if      [ -f frag.ksh ]
then    echo frag.ksh exists: will not overwrite
else
    echo extracting frag.ksh
    sed 's/^X//' >frag.ksh <<_EOF_
X# This is frag.ksh
X# Not very interesting, really.
Xecho frag.ksh here!
_EOF_
    if [ "`sum frag.ksh|awk '{print $1}'`" !=  52575 ]
    then        echo frag.ksh damaged in transit
    fi
fi
```

The following is a simple script to produce as much of the Fibonacci sequence as can be calculated in integers:

```
# Print out Fibonacci sequence; start sequence
# with first two positional parameters:
# default 1 1
typeset -i x=${1:-1} y=${2:-1} z
while   [ x -gt 0 ]      # until overflow
do
    echo $x
    let z=y+x x=y y=z
done
```

The following implements the **basename** utility as a shell function:

```
# basename utility as shell function
function basename {
    case $# in
    1)      ;;
    2)      eval set \${1%$2} ;;
    *)      echo Usage: $0 pathname '[suffix]'
            return 1 ;;
    esac
    echo ${1##*/}
    return 0
}
```

Environment Variables

Table 10 on page 293 lists the environment variables and their purposes.

Table 10. Built-in Variables

Variable	Purpose
_	(Underscore) For every command that is run as a child of the shell, **sh** sets this variable to the full path name of the executable file and passes this value through the environment to that child process. When processing the **MAILPATH** variable, this variable holds the value of the corresponding mail file.
~	(Tilde) expands to value of the **HOME** directory.
CDPATH	Contains a list of directories for the **cd** command to search. Directory names are separated with colons. **CDPATH** works like the **PATH** variable.
COLUMNS	Used by several commands to define the width of the terminal output device.
EDITOR	Specifies the default editor (either **ed** or **sed**). This variable is usually set in your **.profile**.
ENV	**sh** performs parameter substitution on this value and uses the result as the name of an initialization file, or login script. This file is run with the **.** (**dot**) command; see the **dot** command. This variable is usually set in your **.profile**.
FCEDIT	Contains the name of the default editor for the **fc** command. If this variable is not set, the default is the **ed** command.
HISTFILE	Contains the path name of a file to be used as the history file. When the shell starts, the value of this variable overrides the default history file.
HISTSIZE	Contains the maximum number of commands that the shell keeps in the history file. If this variable contains a valid number when the shell starts, it overrides the default of 127.
HOME	Contains your home directory. This is also the default directory for the **cd** command. The **HOME** variable is set automatically from the Initial Working Directory field of the POSIX user database (CP directory or External Security Manager) when the user logs in.
IFS	Contains a series of characters to be used as *internal field separator* characters. Any of these characters may separate arguments in unquoted command substitutions such as `` `command` `` or `$(command)`, or in parameter substitutions. In addition, the shell uses these characters to separate values put into variables with the **read** command. Finally, the first character in the value of **IFS** separates the positional parameters in $* expansion. By default, **IFS** contains space, tab, and newline.
LANG	Contains the default locale value.
LC_ALL	Indicates the locale to be used to override any values for locale categories specified by **LANG** or any of the **LC_** variables, such as **LC_COLLATE**, **LC_CTYPE**, and **LC_MESSAGES**, which a user can set and interrogate.
LINENO	Contains the number of the line currently being run by a shell script.
LINES	Used by several commands to define the number of lines on the terminal output device.
LOGNAME	Contains the user login name. If a variable called LOGNAME exists in the CENV group of GLOBALV variables, LOGNAME is automatically set to this value. If LOGNAME does not exist in the CENV group, the LOGNAME environment variable is set to the user login name.

Table 10. Built-in Variables (continued)

Variable	Purpose
MAIL	Contains the path name of your system *mailbox*. If the **MAILPATH** variable is not set, the OpenExtensions shell tells you when new mail arrives in this file. The shell assumes that new mail has arrived if the file modification time changes.
MBOX	Contains the path name of your personal mailbox, usually **$HOME/mbox**, used to store messages that have been read from your system mailbox. This variable is usually set in your **.profile**.
MAILCHECK	Contains the number of seconds of elapsed time that must pass before the system checks for mail; the default value is 600 seconds. When using the **MAIL** or **MAILPATH** variables, the OpenExtensions shell checks for mail before issuing a prompt.
MAILPATH	Contains a list of mailbox files. This overrides the **MAIL** variable. The mailbox list is separated by colons. If any name is followed by **?**message or **%**message, **sh** displays the message if the corresponding file has changed. **sh** performs parameter and command substitution on *message*, and the variable _ (temporarily) expands to the name of the mailbox file. If no **?**message or **%** message is present, the default message is `you have mail in $_`.
OLDPWD	Contains the name of the directory you were previously working in. The **cd** command sets this variable.
PATH	Contains a list of directories that the system searches to find executable commands. Directories in this list are separated with colons. **sh** searches each directory in the order specified in the list until it finds a matching executable. If you want the shell to search the working directory, put a null string in the list of directories (for example, to tell the shell to search the working directory first, start the list with a colon or semicolon).
PID	Contains the decimal value of the process ID of the parent of the shell. See **ps**.
PS1	Contains the primary prompt string used when the shell is interactive. The default value is a dollar sign followed by a space ($). The shell expands parameters before the prompt is printed. A single exclamation mark (!) in the prompt string is replaced by the command number from the history list; see the **fc** command. For a real exclamation mark in the prompt, use !!. This variable is usually set in your **.profile**.
PS2	Contains the secondary prompt, or continuation prompt, used when completing the input of such things as reserved-word commands, quoted strings, and here documents. The default value of this variable is a greater than sign followed by a space (>).
PS3	Contains the prompt string used in connection with the **select** reserved word. The default value is a number sign followed by a question mark and a space (#?).
PS4	Contains the prefix for traced commands with **set -x**. The default value is a plus sign followed by a space (+).
PWD	Contains the name of the working directory. When the shell starts, the working directory name is assigned to **PWD** unless the variable already has a value.
RANDOM	Returns a random integer. Setting this variable sets a new seed for the random number generator.

Table 10. Built-in Variables (continued)

Variable	Purpose
SECONDS	Contains elapsed time. The value of this variable grows by 1 for each elapsed second of real time. Any value assigned to this variable sets the **SECONDS** counter to that value; initially the shell sets the value to 0.
SHELL	Contains the full path name of the current shell. It is not set by the shell, but is used by various other commands to invoke the shell. This is set automatically by the OPENVM SHELL command.
TMOUT	Contains the number of seconds before user input times out. If user input has not been received within this length of time, the shell ends.
TZ	Contains the system time zone value used for displaying date and time. This is set automatically from /etc/profile when the user logs in.

Files

/.sh_history
   The default history storage file.

.profile
   The user profile for login shell.

/etc/profile
   The systemwide profile for login shells.

/tmp/sh*
   Temporary files for here-documents, command substitution, history reexecution, and so on. The default directory **/tmp** can be overridden by setting the shell variable **TMPDIR** to the name of some other directory.

Localization

sh uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**

See for more information.

Exit Values

Possible exit status values are:

0
   Successful completion.

1
   Failure due to any of the following:

   - The shell was invoked with an incorrect option.
   - The shell was invoked to run a shell script and the command.
   - A command syntax error.
   - A redirection error.
   - A variable expansion error.

Otherwise, the exit status of the shell defaults to the exit status of the last command run by the shell. This default can be overridden by explicit use of the **exit** or **return** commands. The exit status of a pipeline is the exit status of the last command in the pipeline.

Messages and Return Codes

Ambiguous redirection
A redirection construct expanded to more than one path name.

Argument too long
Any single argument to a command is limited in length (see "Limits" on page 297). Command and parameter substitution may exceed this limit.

Cannot restore privileged state
This message occurs only when the implementation of POSIX does not support the *saved IDs* option (*_POSIX_SAVED_IDS*). The message is generated if you tried to use a *saved ID* feature to return to a privileged state.

File *file* already exists
You are attempting to redirect output into an existing file, but you have turned on the **noclobber** option ("set — Set or unset command options and positional parameters" on page 273). If you really want to redirect output into an existing file, use the construct >|*filename*, or turn off the option with:

```
set +o noclobber
```

File descriptor number already redirected
You attempted to redirect a file descriptor that was already being redirected in the same command. You can redirect a file descriptor only once.

Hangup
The shell received a *hangup* signal. This signal typically arises when a communication line is disconnected—for example, when a phone connection is cut off.

In base#number: base must be in [2,36]
In a number of the form base#number, the value of the base was larger than 36 or less than 2. The only valid range for bases is from 2 through 36.

Invalid subscript
A shell array was indexed with a subscript that was outside the defined bounds.

Illegal instruction
The shell received an *illegal instruction* signal. This signal typically occurs when a process tries to execute something that is not a valid machine instruction recognized by the hardware.

Misplaced subscript *array name*
The subscript for an array was missing or incorrect.

***name* is not an identifier**
You attempted to use a nonalphanumeric *name*.

***name*: readonly variable**
The given *name* is a read-only variable, and cannot be removed or changed (see **readonly**).

***name*: no expansion of unset variable**
The shell is operating with **set –u**, and you used an unset variable in a substitution. For more information, see "set — Set or unset command options and positional parameters" on page 273.

No file descriptor available for redirection
When a file descriptor is redirected, the old value is remembered by the shell by a duplication to yet another file descriptor. The total number of file descriptors is limited by the system; hence, the shell may run out, even though your command appears to be using far fewer than the maximum number of descriptors.

Nested aliases
You have more than nine levels of aliases. For example:

```
alias a1=a2 a2=a3 a3=a4 ... a10=command
```

causes this error.

Pipe for coprocess
The shell cannot create a pipe for a coprocess. This may mean that your session or the system as a whole has already set up its maximum number of pipes.

...: restricted
If the shell has been invoked as a restricted shell, certain things are disallowed—for example, the **cd** command, setting **PATH**, and output redirection.

Temporary file error using here document
sh tried to create a temporary file holding the contents of a <<*word* here-document. However, the temporary file could not be created. This may indicate a lack of space on the disk where temporary files are created.

Word after ... expanded to more than one argument
In a context where only one argument was expected, a construct expanded to more than one argument.

Limits

The size of the command argument and the exported variables passed between the shell and the utilities it runs is dependent on the operating system.

A single command line is restricted to 2024 bytes.

The maximum length of an executable file name, including subdirectories and extensions, is dependent on the operating system.

Portability

POSIX.2, X/Open Portability Guide.

The construct $[*arithmetic expression*] is an extension of the POSIX standard.

Related Commands

alias, **break**, **cd**, **continue**, **dot**, **echo**, **eval**, **exec**, **exit**, **export**, **fc**, **getopts**, **let**, **print**, **ps**, **pwd**, **read**, **readonly**, **return**, **set**, **shift**, **test**, **time**, **trap**, **true**, **typeset**, **unalias**, **unset**, **whence**

shift — Shift positional parameters

shift [*expression*]

Purpose

shift renames the positional parameters so that *i+n*th positional parameter becomes the *i*th positional parameter, where *n* is the value of the given arithmetic *expression*. If you omit *expression*, the default value is 1. The value of *expression* must be between zero and the number of positional parameters (**$#**), inclusive.

Usage Notes

This command is built into the shell.

Examples

The commands:

```
set a b c d
shift 2
echo $*
```

produce:

```
c d
```

Exit Values

Possible exit status values are:

0
  Successful completion

1
  Failure because the *expression* had a negative value or was greater than the number of positional parameters.

hledi messages. Possible error messages include:

bad shift count *expr*
  You specified an expression that did not evaluate to a number in the range from 0 to the number of remaining positional parameters.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Allowing an expression, rather than just a number, is an extension found in the OpenExtensions shell (a KornShell).

Related Commands

set, sh

showexp — See the OPENVM SHOWMMOUNT command

The showexp shell command is not available. Use the OPENVM SHOWMOUNT command in place of the showexp command. See "OPENVM SHOWMOUNT" on page 458.

sleep — Suspend execution of a process for an interval of time

---

sleep *seconds*

Purpose

The **sleep** command continues running until the specified number of *seconds* has elapsed. **sleep** can delay execution of a program or produce periodic execution in conjunction with shell commands.

The *seconds* argument can be either a number of seconds, or a more general time description of the form *n*h*n*m*n*s, with the *n*h, *n*m, and the *n*s being optional.

Examples

```
sleep 20h10m
```

sleeps for 20 hours and 10 minutes (or 72600 seconds).

Localization

sleep uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

Successful completion

2

Failure because you specified no *seconds* value or because *seconds* is an incorrect argument (for example, incorrect format).

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

date

sort — Start the sort-merge utility

> sort [-cmu] [-o *outfile*] [-t *char*] [–y[*n*]] [–z*n*] [–bdfiMnr]
>   [-k *startpos[,endpos]*] ... [*file* ...]
> sort [-cmu] [-o *outfile*] [–t *char*] [–y*n*] [–z*n*] [–bdfiMnr]
>   [+*startposition* [-*endposition*]] ... [*file* ...]

Purpose

The **sort** command implements a full sort-and-merge utility. **sort** operates on input files containing records that are separated by the newline character. The following options select particular operations:

–c

Checks input files to ensure that they are correctly ordered according to the key position and sort ordering options specified, but does not modify or output the files. This option affects only the exit code.

–m

Merges *file*s into one sorted output stream. This option assumes that each input file is correctly ordered according to the other options specified on the command line; you can check this with the **–c** option.

–u

Ensures that output records are unique. If two or more input records have equal sort keys, **sort** writes only the first record to the output. When you use **–u** with **–c**, **sort** prints a diagnostic message if the input records have any duplicates.

When you do not specify either the **–c** or the **–m** option, **sort** sorts the concatenation of all input files and produces the output on standard output.

Options

Options that control the operation of **sort** are:

–o *outfile*

Writes output to the file *outfile*. By default, **sort** writes output to the standard output. The output file can be one of the input files. In this case, **sort** makes a copy of the data to allow the (potential) overwriting of the input file.

–t *char*

Indicates that the character *char* separates input fields. When you do not specify the **–t** option, **sort** assumes that any number of white-space (blank or tab) characters separate fields.

–y[*n*]

Restricts the amount of memory available for sorting to *n* KB of memory (where a KB of memory is 1024 bytes). If *n* is missing, **sort** chooses a reasonable maximum amount of memory for sorting, dependent on the system configuration. **sort** needs at least enough memory to hold five records simultaneously. If you try to request less, **sort** automatically takes enough. When the input files overflow the amount of memory available, **sort** automatically does a polyphase merge (external sorting) algorithm, which is, of necessity, much slower than internal sorting. When you use **–u** with **–c**, **sort** prints a diagnostic message if the input records have any duplicates. Using the **–y** option may therefore improve sorting performance substantially for medium to large input files.

–z*n*

Indicates that the longest input record (including the newline character) is *n* bytes in length. By default, record length is limited to LINE_MAX.

The following options control the way in which **sort** does comparisons between records in order to determine the order in which the records are placed on the output. The ordering options apply globally

to all sorting keys except those keys for which you individually specify the ordering option. For more on sorting keys, see "Sorting Keys" on page 302.

–b

Skips, for comparison purposes, any leading white space (blank or tab) in any field (or key specification).

–d

Uses *dictionary* ordering. With this option, **sort** examines only blanks, uppercase and lowercase letters, and numbers when making comparisons.

–f

Converts lowercase letters to uppercase for comparison purposes.

–i

Ignores, for comparison purposes, nonprintable characters.

–k [*startpos* [*endpos*]].

Specifies a sorting key. For more information, see "Sorting Keys" on page 302.

–M

Assumes that the field contains a month name for comparison purposes. Any leading white space is ignored. If the field starts with the first three letters of a month name in uppercase or lowercase, the comparisons are in month-in-year order. Anything that is not a recognizable month name compares less than JAN.

–n

Assumes that the field contains an initial numeric value. **sort** sorts first by numeric value and then by the remaining text in the field according to options.

Numeric fields can contain leading optional blanks or optional minus (-) signs. **sort** does not recognize the plus (+) sign.

This option treats a field which contains no digits as if it had a value of zero.

–r

Reverses the order of all comparisons so that **sort** writes output from largest to smallest rather than smallest to largest.

Sorting Keys

By default, **sort** examines entire input records to determine ordering. By specifying *sorting keys* on the command line, you can tell **sort** to restrict its attention to one or more parts of each record.

You can indicate the start of a sorting key with:

```
-k m[.n][options]
```

where *m* and the optional *n* are positive integers. You can choose *options* from the set **bdfiMnr** (described previously) to specify the way in which **sort** does comparisons for that sorting key. Ordering options set for a key override global ordering options. If you do not specify any *options* for the key, the global ordering options are used.

The number *m* specifies which field in the input record contains the start of the sorting key. The character given with the **–t** option separates input fields; if this option is not specified, spaces or tabs separate the fields. The number *n* specifies which character in the *m*th field marks the start of the sorting key; if you do not specify *n*, the sorting key starts at the first character of the *m*th field.

You can also specify an ending position for a key, with:

```
-k m[.n][options],
   p[.q][options]
```

where *p* and *q* are positive integers, indicating that the sort key ends with the *q*th character of the *p*th field. If you do not specify *q* or if you specify a value of 0 for *q*, the sorting key ends at the last character of the *p*th field. For example:

```
-k 2.3,4.6
```

defines a sorting key that extends from the third character of the second field to the sixth character of the fourth field.

sort also supports a historical method of defining the sorting key. Using this method, you indicate the start of the sorting key with:

```
+m[.n][options]
```

which is equivalent to:

```
–k m+1[.n+1][options]
```

You can also indicate the end of a sorting key with:

```
–p[.q][options]
```

which when preceded with +*m*[.*n*] is equivalent to:

```
–k m+1[.n+1],p.0[options]
```

if q is specified and is zero, or

```
–k m+1[.n+1],p+1[.q+1][options]
```

Otherwise, for example:

```
+1.2 -3.5
```

defines a sorting key with a starting position that **sort** finds by skipping the first field and two characters of the next field, and an ending position that **sort** finds by skipping the first three fields and then the first five characters of the next field. In other words, the sorting key extends from the third character of the second field to the sixth character of the fourth field. This is the same key as defined under the **–k** option, described earlier.

With either syntax, if the end of a sorting key is unspecified or is not a valid position after the beginning key position, the sorting key extends to the end of the input record.

You can specify multiple sort key positions by using several **–k** options or several **+** and **–** options. In this case, **sort** uses the second sorting key only for records where the first sorting keys are equal, the third sorting key only when the first two are equal, and so on. If all key positions compare equal, **sort** determines ordering by using the entire record.

When you specify the **–u** option to determine the uniqueness of output records, **sort** looks only at the sorting keys, not the whole record. (Of course, if you specify no sorting keys, **sort** considers the whole record to be the sorting key.)

Examples

1. To sort an input file having lines consisting of the day of the month, white space, and the month, as in:

```
30 December
23    MAY
25 June
10    June
```

use the command:

```
sort -k 2M -k 1n
```

2. To merge two dictionaries, with one word per line:

```
sort –m –dfi dict1 dict2 >newdict
```

Environment Variables

TMPDIR
Contains the path name of the directory to be used for temporary files.

Files

/tmp/stm*
Temporary files used for merging and **–o** option. You can specify a different directory for temporary files using the **TMPDIR** environment variable.

Localization

sort uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_NUMERIC**
- **LC_TIME**

The **–M** option works only if **LC_TIME** identifies a locale that contains the same month names as the POSIX locale.

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
Successful completion. Also returned if **–c** is specified and the file is in correctly sorted order.

1
Returned if you specified **–c** and the file is not correctly sorted. Also returned to indicate a nonunique record if you specified **–cu**.

2
Failure due to any of the following:

- Missing key description after **–k**
- More than one **–o** option
- Missing *file* name after **–o**
- Missing character after **–t**
- More than one character after **–t**
- Missing *number* with **–y** or **–z**
- *endposition* given before a *startposition*
- Badly formed sort key
- Incorrect command-line option
- Too many key field positions specified

- Insufficient memory
- Inability to open the output file
- Inability to open the input file
- Error in writing to the output file
- Inability to create a temporary file or temporary file name

Messages and Return Codes

Possible error messages include:

Badly formed sort key position *x*
   The key position was not specified correctly. Check the format and try again.

File *filename* **is binary**
   sort has determined that *filename* is binary because it found a NULL (' ') character in a line.

Missing key definition after -k
   You specified **–k**, but did not specify a key definition after the **–k**.

Nonunique key in record ...
   With the **–c** and **–u** options, a nonunique record was found.

Not ordered properly at ...
   With the **–c** option, an incorrect ordering was discovered.

Line too long: limit *nn* **— truncated**
   Any input lines that are longer than the default number of bytes (LINE_MAX) or the number specified with the **–z** option are truncated.

No newline at end of file
   Any file not ending in a newline character has one added.

Insufficient memory for ...
   This error normally occurs when you specify very large numbers for **–y** or **–z** and there is not enough memory available for **sort** to satisfy the request.

Write error (no space) on output
   Some error occurred in writing the standard output. Barring write-protected media and the like, this normally occurs when there is insufficient disk space to hold all of the intermediate data.

Temporary file error (no space) for ...
   Insufficient space was available for a temporary file. Make sure that you have a directory named **/tmp**, and that this directory has space to create files. You can change the directory for temporary files using the **ROOTDIR** and **TMPDIR** environment variables.

Tempfile error on ...
   The named temporary (intermediate) file could not be created. Make sure that you have a directory named **/tmp**, and that this directory has space to create files. You can change the directory for temporary files using the **TMPDIR** environment variable.

Tempnam() error
   sort could not generate a name for a temporary working file. This should almost never happen.

Too many key field positions specified
   This implementation of **sort** has a limit of 64 key field positions.

Portability

POSIX.2.

Available on all UNIX systems, with only UNIX System V.2 or later having the full utility described here.

The **–M**, **–y**, and **–z** options are extensions of the POSIX standard.

Related Commands

awk, **comm**, **cut**, **join**, **uniq**

The **sortgen awk** script is a useful way to handle complex sorting tasks. It originally appeared in *The AWK Programming Language,* by Aho, Weinberger, and Kernighan. The POSIX standard regards the historical syntax for defining sorting keys as obsolete. Therefore, you should use only the **–k** option in the future.

strip — Remove unnecessary information from an executable file

strip *file ...*

Purpose

strip removes any data from an executable file with a view to conserving disk space for production (that is, already debugged) programs. This program does not modify the contents of any executable binary file. The **strip** command does not affect the contents of a file compiled under VM.

Exit Values

Possible exit status values are:

0

Successful completion

1

Failure due to any of the following:

- *file* could not be opened
- An error occurred while reading *file*
- *file* is not an executable file
- *file* is executable, but appears corrupted

2

No *file* was specified on the command line

Messages and Return Codes

Possible error messages include:

file *name*: system error
The named executable file does not exist or is unreadable.

Cannot create temporary file
A temporary file cannot be created.

Output error (no space) on file
There is insufficient disk space to hold a temporary copy of the executable file. For implementation reasons, **strip** makes a copy of each file being stripped.

File *name*: not in executable format
This is a warning that file *name* will not be modified.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

stty — Set or display terminal options

stty [–ag] [*operand*]

Purpose

stty sets or reports the terminal I/O characteristics for the standard input device. **stty**, entered without options or operands, reports only the terminal I/O characteristics that differ from the defaults. **stty**, entered with *operand*s enables, disables, or selects the full range of terminal I/O characteristics.

Options

This command recognizes the following options:

–a
> Displays all of the terminal I/O characteristics.

–g
> Displays all of the terminal I/O characteristics in a format that can be used as input to the **stty** command.

The **–a** option gives you a clear readable description, whereas the **–g** option provides the ability to save and restore the terminal I/O characteristics.

stty entered with *operand*s enables, disables, or selects the full range of terminal I/O characteristics.

Control Mode Operands

The valid operands for setting *control modes* are:

parenb
> Enable parity generation and detection.

–parenb
> Disable parity generation and detection.

parodd
> Select odd parity.

–parodd
> Select even parity.

cs5
> Select character size CS5.

cs6
> Select character size CS6.

cs7
> Select character size CS7.

cs8
> Select character size CS8.

number
> Set the input and output baud rates to *number*. A *number* of zero hangs up the modem line.

ispeed *number*
> Set the input baud rate to *number*. A *number* of zero sets the input baud rate to the same value as the output baud rate.

ospeed *number*
> Set the output baud rate to *number*. A *number* of zero hangs up the modem line.

hupcl
> Hang up the modem line on the last close.

–hupcl
> Do not hang up the modem line on the last close.

hup
> Hang up the modem line on the last close.

–hup
> Do not hang up the modem line on the last close.

cstopb
> Use two stop bits per character.

–cstopb
> Use one stop bit per character.

cread
> Enable the receiver.

–cread
> Disable the receiver.

clocal
> Assume a line without modem control.

–clocal
> Assume a line with modem control.

Input Mode Operands

The valid operands for setting *input modes* are:

ignbrk
> Ignore break on input.

–ignbrk
> Do not ignore break on input.

brkint
> Signal INTR on break.

–brkint
> Do not signal INTR on break.

ignpar
> Ignore parity errors.

–ignpar
> Do not ignore parity errors.

parmrk
> Mark parity errors.

–parmrk
> Do not mark parity errors.

inpck
> Enable input parity checking.

–inpck
> Disable input parity checking.

istrip
> Strip input characters to seven bits.

–istrip
> Do not strip input characters to seven bits.

inlcr
> Map newline to carriage return on input.

–inlcr
Do not map newline to carriage return on input.

igncr
Ignore carriage return on input.

–igncr
Do not ignore carriage return on input.

icrnl
Map carriage return to newline on input.

–icrnl
Do not map carriage return to newline on input.

ixon
Enable START/STOP output control.

–ixon
Disable START/STOP output control.

ixoff
Ask the system to send START/STOP characters to regulate the size of the input queue.

–ixoff
Ask the system not to send START/STOP characters to regulate the size of the input queue.

Output Mode Operands

The valid operands for setting *output modes* are:

onlcr
Converts newline characters to newline-carriage return sequences.

–onlcr
Newline characters are displayed as newlines only.

opost
Postprocess output.

–opost
Do not postprocess output. Ignore all other output modes.

Local Mode Operands

The valid operands for setting *local modes* are:

isig
Enable character checking against the special control characters INTR, QUIT and SUSP.

–isig
Disable character checking against the special control characters INTR, QUIT and SUSP.

icanon
Enable canonical input mode (ERASE and KILL processing).

–icanon
Disable canonical input mode (ERASE and KILL processing).

iexten
Enable any custom special control characters.

–iexten
Disable any custom special control characters.

echo
Echo every character typed.

–echo
Do not echo every character typed.

echoe
Enable the ERASE character to visibly erase the latest character.

–echoe
Do not enable the ERASE character to visibly erase the latest character.

echok
Echo newline after a KILL character.

–echok
Do not echo newline after a KILL character.

echonl
Echo newline (even when **echo** is disabled).

–echonl
Do not echo newline when **echo** is disabled.

noflsh
Disable flush after INTR, QUIT, and SUSP.

–noflsh
Enable flush after INTR, QUIT, and SUSP.

tostop
Send the **SIGTOU** signal for background output.

–tostop
Do not send the **SIGTOU** signal for background output.

Control Character Operands

The valid operands for assigning *special control characters* are:

min *number*
Set min to *number*.

time *number*
Set time to *number*.

eof *string*
Set end of file character to *char*.

eol *char*
Set end of line character to *char*.

erase *char*
Set ERASE character to *char*.

intr *char*
Set INTR character to *char*.

kill *char*
Set KILL character to *char*.

quit *char*
Set QUIT character to *char*.

susp *char*
Set SUSP character to *char*.

start *char*
Set START character to *char*.

stop *char*
Set STOP character to *char*.

pfx *char*
Set control sequence escape character to *char*.

rpfx
Return control sequence escape character to default (¢).

Combination Mode Operands

The valid operands for setting *combination modes* are:

saved-settings
Set the terminal I/O characteristics to the saved settings produced by the **–g** option.

evenp
Enable **parenb** and cs7; disable **parodd**.

parity
Enable **parenb** and cs7; disable **parodd**.

oddp
Enable **parenb**, cs7 and **parodd**.

–parity
Disable **parenb** and set **cs8**.

–evenp
Disable **parenb** and set **cs8**.

–oddp
Disable **parenb** and set **cs8**.

nl
Enable **icrnl**.

–nl
Disable **icrnl**; unset **inlcr** and **igncr**

ek
Reset ERASE and KILL characters to system defaults.

sane
Reset all modes to reasonable values.

Localization

stty uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
Successful completion

1
Failure because of any of the following:

Note: In the following exit status values, the word *termios* refers to the terminal's I/O settings as defined in termios.h.

- Error setting termios attributes
- Unknown mode
- Missing number after option
- Argument out of range
- Bad number after option

- Internal error
- Error reading termios attributes
- Missing character after option
- Badly formed argument option character
- Missing speed after **ispeed** or **ospeed**
- Bad speed argument

Portability

POSIX.2, UNIX system V.

su — Change the user ID associated with a session

> **su**

Purpose

The **su** command starts a new shell and lets you operate in it with the privileges of a superuser.

The **su** command changes your authorization to that of the superuser. The superuser environment is built and then a new session is initiated for the superuser. The new superuser session is run as a subshell of the shell issuing the **su** command. The session that is initiated will be started as a login shell.

The functions performed by **su** are as follows:

- **Changes the user ID to that of the superuser.** After verifying that the user is authorized, the user ID is changed to the superuser's user ID.
- **Sets up the shell environment for the superuser.** The superuser's environment is set up to be as similar as possible to the environment of the shell issuing the **su** command. Information is obtained from the user database. Values not found in the user database (the CP directory or External Security Manager) are defaulted. If the value for the initial program (shell) is not available, a default value of **/bin/sh** is used.
- **Executes the superuser shell.** Initialization of a login shell to run under the existing shell, as a subshell takes place. This subshell will be a child process of the shell issuing the **su** command. If the **su** command is run from a restricted shell (such as a shell that was started with the **–r** option), you will exit from the restricted shell and leave the protection of the trusted environment.

To restore the previous session, enter **exit** or press <EscChar-D>. This action ends the subshell initiated by the **su** command and returns you to the previous shell, user ID, and environment. See _z/VM: OpenExtensions User's Guide_ for more information on exiting the shell environment.

Usage Notes

To use this command, the BFS server must have CP authority to change POSIX user IDs. The CP directory entry for the BFS server must contain the line, `POSIXOPT SETIDS ALLOW`. If the server does not have this authority, then issuing this command will result in a system abend with CMS abend code ADE.

Exit Values

Possible exit status values are:

0

The command completed successfully

1

User is not authorized to obtain superuser authority

2

Failure due to any of the following:

- Unable to execute the shell
- No entry found for this user in the user database
- Unable to set up the superuser environment.

3

Failure due to any of the following:

- Incorrect command syntax
- Unable to open the message catalog

Messages and Return Codes

Possible error messages include:

User not authorized to obtain superuser authority
The user ID issuing the **su** command does not have the proper authorization to obtain superuser authority. Contact the system programmer.

Unable to set up the user environment. Processing terminates.
The environment variables required by the shell have not been set set up. Processing terminates. Contact the system programmer.

Unable to execute the shell.
The initial program (shell) was not run. Verify that the initial program (shell) exists on this system and that the user has permission to execute it.

Limits

By default, a user must be a superuser or a member of group ID 0 and have permission to execute set-id files to use the **su** command. See *z/VM: CP Planning and Administration* for more information about permission to execute set-id files.

Portability

None. This command is an extension that comes with OpenExtensions services.

Related Commands

sh, **OPENVM SHELL**

tail — Display the last part of a file

```
tail [−f] [−bcklmn [±]number] [file]
tail [−f] [±number[bcklmn]] [file]
```

Purpose

Calling **tail** without options displays the last ten lines of *file*. This is useful for seeing the most recent entries in log files and any file where new information is added on the end.

Note: The **tail** command is used with text files. To make a binary file input to the **tail** command, use the **−c** option. If a binary file is input without the **−c** option being specified, the entire file is sent to the screen.

Options

+|-number
    Is either of the following:

+number
    Skips to line *number* and then displays the rest of the file. For example, +100 prints from line 100 to the end of the file.

−number
    Prints *number* lines from the end of the file. For example, -20 prints the last 20 lines in the file.

You can precede or follow both **+**number and **−**number with one of the following letters to indicate the unit to be used:

b
    Blocks

c
    Bytes

k
    Kilobytes

l or n
    Lines

m
    Megabytes

The default unit is lines.

−f

Monitors a file as it grows. Every 2 seconds, **tail** wakes up and prints any new data at the end of the file. This option is ignored if **tail** read from the standard input, and standard input is a pipe.

Localization

tail uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

Successful completion

1

Failure due to any of the following:

- Insufficient memory
- Write error on the standard output
- Badly formed line or character count
- Missing number after an option
- Error reopening a file descriptor

2

Failure due to an unknown command-line option

Messages and Return Codes

Possible error messages include:

Badly formed line/character count *string*
In an option of the form **–n** *number* or **–***number*, the *number* was not a valid number.

Reopening file descriptor *number*
–f was used to follow a file as it grew. **tail** closed the file associated with the given file descriptor *number* and then tried to open it 2 seconds later. At this point, **tail** found it could not reopen the file for reading, and therefore could not follow the file any longer.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The POSIX standard does not include the use of **b**, **k**, or **m** as either options or suffixes. **-l** is an extension of the traditional implementation of **tail**.

Related Commands

cat, **head**

tar -- Manipulate the tar archive files to copy or back up a file

```
tar –c[#sbfvwlzU] [–V volpat] [tarfile] [blocksize] [–C pathname] [file …]
tar –r[#sbfvwlU] [–V volpat] [tarfile] [blocksize] [file …:]
tar –t[#sbfvzU] [–V  volpat] [tarfile] [blocksize] [–C pathname] [file …]
tar –u[#sbfvwlU] [–V volpat] [tarfile] [blocksize] [file …:]
tar –x[#sbfvwpmozU] [–V volpat] [tarfile] [blocksize] [file …]
```

Purpose

tar manipulates *archives*. An *archive* is a single file that contains the complete contents of a set of other files; an archive preserves the directory hierarchy that contained the original files, in a manner similar to **cpio**. The name **tar** was derived from *T*ape *AR*chiver; however, you can use archives with any medium.

This version of the **tar** utility writes and reads the original **tar** format from UNIX systems as well as the USTAR format defined by the POSIX (IEEE P1003.1) standards group.

Options

The five forms of the command shown in the syntax represent the main functions of **tar** as follows:

–c

Creates an archive. This command writes each named *file* into a newly created archive. Directories recursively include all components. Under the USTAR (**–U**) option, **tar** records directories and other special files in the tape archive; otherwise, it ignores such files. If **–** appears in place of any *file* name, **tar** reads the standard input for a list of files one per line. This allows other commands to generate lists of files for **tar** to archive.

–r

Writes the named *file*s to the end of the archive. It is possible to have more than one copy of a file in a tape archive using this method. To use this form of the command with a tape, it must be possible to backspace the tape.

Note: You cannot specify both the -u option and the -z option at the same time.

–t

Displays a table of contents. This displays the names of all the files in the archive, one per line. If you specify one or more *file*s on the command line, **tar** prints only those file names. Under the verbose (**–v**) option, more information about each tape archive member is printed, in a format similar to that produced by **ls –l**.

–u

Writes the named *file*s to the end of the archive only if it is not in the archive already or if it has been modified since being written to the archive. It is possible to have more than one copy of a file in a tape archive using this method. To use this form of the command with a tape, it must be possible to backspace the tape.

Note: You cannot specify both the -u option and the -z option at the same time.

–x

Extracts files from an archive. **tar** extracts each named *file* to a file of the same name. If you did not specify any files on the command line, all files in the archive are extracted. This extraction restores all file system attributes as controlled by other options.

You must specify one of the preceding basic options as the first character of an option string. You can add other characters to the option string. Unlike with other commands, you must give options as a single string; for example, you might specify **–tv**, but you cannot separate them, as in "**-t -v**." You can omit the leading dash **–** if you want. Other possible options in the option string are:

b

Sets the number of 512-byte blocks used for tape archive read/write operations to *blocksize*. The *blocksize* argument must be specified, and *blocksize* can be specified only when **b** is in the option string. When reading from the tape archive, **tar** automatically determines the blocking factor by trying to read the largest permitted blocking factor and using the actual number read to be the *blocksize*. For UNIX compatibility, the largest valid block size is 20 blocks; in USTAR mode, it is 60 blocks.

−C *pathname*

Is an unusual option because it is specified in the middle of your *file* list. When **tar** encounters a **−C** *pathname* option while archiving files, it changes the working directory (for **tar** only) to *pathname* and treats all following entries in your *file* list (including another **−C**) as being relative to *pathname*.

f

You must specify **f**. The **f** option uses the file *tapefile* for the tape archive rather than using the default. The *tapefile* argument must be specified, and *tapefile* can be specified only when **f** is in the option string. The *tapefile* argument must precede the *blocksize* argument if both are present. If *tapefile* is the character −, the standard input is used for reading archives, and the standard output is used for writing archives.

#s

#s is not supported on OpenExtensions. The default archive file name used by **tar** is **/dev/mt/0m**. This option is the least general way to override this default. For a more general method, see the **f** option. The file name generated by this option has the form **/dev/mt/#s**. The # can be any digit between 0 and 7, inclusive, to select the tape unit. The density selector *s* can be l (low), m (medium), or h (high).

l

Complains if all links are not resolved when adding files to the tape archive.

m

Does not restore a file's modification time stamp when extracting it from an archive. The default behavior is to restore the time stamp from information contained in the archive.

o

When writing files to an archive, does not record owner and modes of directories in the archive. If this is specified when extracting from an existing **tar** archive, **tar** does not restore any owner and group information in the archive. The default is to record this information when creating a **tar** archive, and to restore it when extracting from the archive.

p tar archive

When extracting, restores the three high-order file permission bits, exactly as in the archive. They indicate the set-user-ID, set-group-ID, and sticky bit. To use **p** on UNIX systems, you must have appropriate privileges; **tar** restores the modes restored exactly as in the archive and ignores the UMASK.

U

When creating a new tape archive with the **−c** option, forces **tar** to use the USTAR format. The default format used when creating a new archive is the original UNIX **tar** format. When you do not specify **−c**, **tar** can deduce whether the tape archive is in USTAR format by reading it, so you can use **U** to suppress a warning about USTAR format.

v

Displays each file name, along with the appropriate action key letter as it processes the archive. With the **−t** form of the command, this option gives more detail about each archive member being listed.

-V *volpat*

Provides automatic multivolume support. **tar** writes output to files—the names of which are formatted with *volpat*. Any occurrence of # in *volpat* is replaced by the current volume number. When you invoke **tar** with this option, it prompts for the first number in the archive set, and wait for you to type the number and a carriage return before proceeding with the operation. **tar** issues the same sort of message when a write error or read error occurs on the archive; this kind of error means that **tar** has reached the end of the volume and should go on to a new one.

w

Is used to confirm each operation, such as replacing or extracting. **tar** displays the operation and the file involved. You can then confirm whether you want the operation to take place. Typing in an answer that begins with "y" tells **tar** to do the operation; anything else tells **tar** to go on to the next operation.

z

Reads or writes, or both reads and writes, the tape archive by first passing through a compression algorithm compatible with that of **compress**.

Note: You cannot specify the **–r** or the **–u** option with the **–z** option at the same time.

Examples

1. The following command takes a directory and places it in an archive in compressed format:

```
tar -cvzf archive directory
```

2. To identify all files that have been changed in the last week (7 days), and to archive them to the **/tmp/posix/testpgm** file, enter:

```
find /tmp/posix/testpgm -type f -mtime -7 | tar -cvf testpgm.tar -
```

-type -f tells **find** to select only files. This avoids duplicate input to **tar**.

Exit Values

Possible exit status values:

0

Successful completion.

1

Failure due to any of the following:

- Incorrect option
- Incorrect command-line arguments
- Out of memory
- Compression error
- Failure on extraction
- Failure on creation

Limits

Path names in the tape archive are normally restricted to a maximum length of 100 bytes. However, in USTAR mode, path names can be up to 255 bytes long.

Portability

4.2BSD (Berkeley Software Distribution).

The **–U** option is an extension to provide POSIX USTAR format compatibility. The **–p** option is an common extension on BSD UNIX systems that is not available on UNIX system V systems.

Related Commands

cpio, **pax**

tee — Duplicate the output stream

---

tee [**–ai**] [*file ...*]

Purpose

The **tee** command clones an output stream. It copies the standard input to each output *file* as well as to the standard output.

Options

–a
> Appends to (rather than overwrites) each output *file*.

–i
> Ignores interrupt signals, making it suitable for use as a background process.

Examples

The following command runs the program **prog** and pipes the program's standard output into **tee**:

```
prog | tee file
```

As a result, **tee** writes the output to both the standard output and the specified *file*.

Localization

tee uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
> Successful completion

1
> Failure due to any of the following:
> - Out of memory when allocating I/O buffers
> - I/O error reading or writing to a file
> - Error creating an output file
> - Error opening an output file for appending

2
> Failure due to incorrect command-line option

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

tee

Related Commands

cat

test or [ ] — Test for a condition

```
test expression [ expression ]
```

Purpose

The **test** command checks for various properties of files, strings, and integers. It produces no output (except error messages) but returns the result of the test as the exit status.

The command line is a Boolean *expression*. The simplest expression is a *string* that is true if the string is nonempty (that is, has nonzero length). More complex expressions are composed of operators and operands, each of which is a separate argument (that is, surrounded by white space). The operators imply the number and type of their operands. The operators taking a *file* operand evaluate as false (without error) if the file does not exist.

Options

–b *file*
     True if the *file* is a block special file

–c *file*
     True if the *file* is a character special file

–d *file*
     True if the *file* is a directory

–e *file*
     True if the *file* exists

–f *file*
     True if the *file* is an ordinary file

–g *file*
     True if the set-group-ID attribute of the *file* is on

–h *file*
     True if the *file* is a hard link

–k *file*
     True if the "sticky" bit is on *file* is on

–L *file*
     True if *file* is a symbolic link

–n *string*
     True if the length of *string* is greater than zero

–p *file*
     True if the *file* is a FIFO (named pipe)

–r *file*
     True if the *file* is readable

–s *file*
     True if the size of the *file* is nonzero

–t *fd*
     True if the numeric file descriptor *fd* is open and associated with a terminal

–u *file*
     True if the set-user-ID attribute of the *file* is on

–w *file*
     True if the *file* is writable

–x *file*
> True if the *file* is executable

–z *string*
> True if the length of the *string* is zero

string
> True if *string* is not a null string

string1 **=** *string2*
> True if *string1* and *string2* are identical

string **!=** *string*
> True if *string1* and *string2* are not identical

number1 **–eq** *number2*
> True if *number1* and *number2* are equal

Within the OpenExtensions shell, either number can be an arbitrary *shell* arithmetic expression; the same applies for the other five numerical comparisons that follow. Both *number1* and *number2* must be integers.

number1 **–ge** *number2*
> True if *number1* is greater than or equal to *number2*

number1 **–gt** *number2*
> True if *number1* is greater than *number2*

number1 **–le** *number2*
> True if *number1* is less than or equal to *number2*

number1 **–lt** *number2*
> True if *number1* is less than *number2*

number1 **–ne** *number2*
> True if *number1* is not equal to *number2*

file1 **–nt** *file2*
> True if *file1* is newer than *file2*

file1 **–ot** *file2*
> True if *file1* is older than *file2*

file1 **–ef** *file2*
> True if *file1* has the same device and inode number as *file2*

expr1 **–a** *expr2*
> Logical AND; true if both *expr1* and *expr2* are true

expr1 **–o** *expr2*
> Logical OR; true if either *expr1* and *expr2* is true

! *expr*
> Logical negation; true if *expr* is false

(*expr* **)**
> Binding; true if *expr* is true

The precedence of the operators in descending order is: unary operators, comparison operators, logical AND, logical OR.

The second form of the test command:

```
[ expression ]
```

is synonymous with the first.

Usage Notes

test is built into the shell and is also implemented as a separate utility. **test** can compare variables; however, if the variable is null, the expression may be incorrect for **test**. For example:

```
NULL=
test $NULL = "so"
```

does *not* work, because the OpenExtensions shell expands this to:

```
test = "so"
```

which is not a valid expression for **test**. A way to get around this is to add some value to the beginning of both strings, as in:

```
test x$NULL = x"so"
```

Failure to quote variable expansions is a common mistake. For example:

```
test $NULL != string
```

If NULL is undefined or empty, this results in:

```
test != string
```

which is not a valid test expression. This problem can be fixed by enclosing $NULL in quotes.

Note: These two examples perform basically the same function; that is, they protect the command against a variable having a possible null value.

Examples

The following command reports on whether the first positional parameter contains a directory or a file:

```
if [ -f $1 ]
then
        echo $1 is a file
elif [ -d $1 ]
then
        echo $1 is a directory
else
        echo $1 neither file nor directory
fi
```

This example illustrates the use of **test**, and is not intended to be an efficient method.

Exit Values

Possible exit status values are:

0

The *expression* was true.

1

The *expression* was false.

2

The *expression* was badly formed.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–k**, **–L**, **–nt**, **–ot**, **–ef**, **–a**, and **–o** operators plus the use of parentheses to group operators together are all extensions of the POSIX standard.

test

Related Commands

expr, **find**, `let`, `ls`, **sh**

time — Display processor and elapsed times for a command

time [**–p**] *command-line*

Purpose

time runs the command given as its argument and produces a breakdown of total time to run (`real`), total time spent in the user program (`user`), and total time spent in system processor overhead (`sys`).

Times given are statistical, based on where execution is at a clock tick.

Options

–p
Guarantees that the historical format of the **time** command is output.

Usage Notes

time is a built-in shell command.

Environment Variables

time uses the following environment variable:

PATH
Determines the search path that **time** uses to locate the command specified in *command-line*.

Exit Values

Possible exit status values are:

0
Successful completion

1
An error occurred in the **time** utility.

2
Failure due to an invalid command-line option.

2
Invalid command-line argument.

126
time found *command* but was unable to invoke it.

127
time was unable to find *command*.

Localization

time uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_NUMERIC**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

time returns the exit status returned by *command-line*.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

sh

times — Get process and child process times

times [**−p**]

Purpose

times displays user and system times accumulated by the shell and commands run as children of the shell.

Options

times recognizes the following option:

−p

Formats the output in seconds without units. For example, 1 minute and 3.47 seconds is displayed as:

```
63.47
```

Times are displayed in minutes and seconds. User time is processor time spent in user programs. System time is processor time spent in the operating system on behalf of the user process. The output layout is:

```
shell user time        shell system time
child user time        child system time
```

Usage Notes

times is a built-in shell command.

Exit Values

Possible exit status values are:

0

Successful completion

2

Failure that resulted in a usage message, usually due to an incorrect command-line option

Portability

X/Open Portability Guide.

Related Commands

sh, **time**

touch — Change the file access and modification times

```
touch [–acm] [–f agefile] [–r agefile] [–t time] file ...
touch [–acm] time file ...
```

Purpose

The **touch** command changes certain dates for each *file* argument. By default, **touch** sets both the date of last file modification and the date of last file access to the current time. This is useful for maintaining correct release times for software and is particularly useful in conjunction with the **make** command.

Options

–a
> Sets only the access time.

–c
> Does not create any *file* that does not already exist. Normally, **touch** creates such files.

–m
> Sets only the modification time.

If you do not specify **–a** or **–m**, **touch** behaves as though you specified both.

To tell **touch** to use a time other than the current, use one of the following options:

–f *agefile*
> Is an obsolete version of the **–r** option.

–r *agefile*
> Sets the access and modification times (as indicated by the other options) to those kept for *agefile*.

–t *time*
> specifies a particular time using this format: `[[[[cc]yy]mm]dd]hhmm [.ss]`
>
> where:
>
> - *cc* is the first two digits of the year (optional)
> - *yy* is the last two digits of the year (optional)
> - *mm* is the number of the month (01—12) (optional)
> - *dd* is the day of the month (optional)
> - *hh* is the hour in 24-hour format (required)
> - *mm* is the minutes (required)
> - *ss* is the seconds (optional)
>
> An obsolete (but still supported) version of this command lets you omit the **–t**, but the format is:
>
> ```
> [[mm]dd]hhmm[.ss]
> ```
>
> or:
>
> ```
> mmddhhmmyy[.ss]
> ```

Examples

1. To set the modification time of **newfile** to the present, enter:

   ```
   touch newfile
   ```

2. To set the modification time of **oldfile** to 13:05 on July 3, 1994, enter:

```
touch -t 9407031305 oldfile
```

3. To set the modification time of **newfile** to that of **oldfile**, enter:

```
touch -r oldfile newfile
```

Environment Variables

TZ
  Contains the time zone that **touch** is to use when interpreting times.

Localization

touch uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
  Successful completion

1
  Failure due to any of the following:

- Inability to access the desired file
- Too early a date was specified
- Inability to create a file
- Inability to change a file's times

2
  Failure that resulted in a usage message, including:

- Unknown command-line option
- Only one of **–t**, **–f**, or **–r** is allowed
- **–r** was missing the *agefile*
- **–t** was missing its argument
- Incorrect date string

Messages and Return Codes

Possible error messages include:

Age file inaccessible
  Indicates that time could not be found for the file given with the **–f** or **–r** option either because that file does not exist or because the requesting user is not granted the appropriate permission for the file.

Missing age file argument
  You specified **–f** or **–r**, but did not give a file name after it.

Years earlier than year incorrect
Your system recognizes dates only back to the given *year*. **touch** does not accept dates before that time.

Bad date conversion

Only one –r, –f, or –t flag allowed

Missing the date or time argument

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

cp, **date**

tr — Translate characters

```
tr [–cs] string1 string2
tr –s [–c] string1
tr –d [–c] string1
tr –ds [–c] string1 string2
```

Purpose

tr copies data read from the standard input to the standard output, substituting or deleting characters as specified by the options and *string1* and *string2*. *string1* and *string2* are considered to be sets of characters. In its simplest form, **tr** translates each character in *string1* into the character at the corresponding position in *string2*.

Note: tr works on a character basis, not on a collation element basis. Thus, for example, a range that includes the multicharacter collation element ch in regular expressions, does not include it here.

Options

–c

Complements the set of characters specified by *string1*. This means that **tr** constructs a new set of characters, consisting of all the characters not found in *string1* and uses this new set in place of *string1*.

–d

Deletes input characters found in *string1* from the output. This string is in ascending order.

–s

tr checks for sequences of a *string1* character repeated several consecutive times. When this happens, **tr** replaces the sequence of repeated characters with one occurrence of the corresponding character from *string2*; if *string2* is not specified, the sequence is replaced with one occurrence of the repeated character itself. For example:,

```
tr -s abc xyz
```

translates the input string aaaabccccb into the output string of xyzy.

If you specify both the **–d** and **–s** options, you must specify both *string1* and *string2*. In this case, *string1* contains the characters to be deleted, whereas *string2* contains characters that are to have multiple consecutive appearances replaced with one appearance of the character itself. For example:

```
tr -ds a b
```

translates the input string abbbaaacbb into the output string bcb.

The actions of the **–s** option take place after all other deletions and translations.

You can use the following conventions to represent elements of *string1* and *string2*:

character

Any character not described by the conventions that follow represents itself.

ooo

An octal representation of a character with a specific coded value. It can consist of one, two, or three octal digits.

character

The \ (backslash) character is used as an escape to remove the special meaning of characters. It also introduces escape sequences for nonprinting characters, in the manner of C character constants: \b, \f, \n, \r, \t, and \v.

c1–c2

 This represents all characters between characters *c1* and *c2* (in the current locale's collating sequence) including the end values. For example, 'a–z' represents all the lowercase letters in the POSIX locale, whereas 'A–Z' represents all that locale's uppercase letters. One way to convert lowercase and uppercase is with the following filter:

```
tr 'a-z' 'A-Z'
```

 This is not, however, the recommended method; use the [:*class*:] construct instead.

c∗n

 This represents *n* repeated occurrences of character *c*. (If *n* has a leading zero, **tr** assumes it is octal; otherwise, it is assumed to be decimal.) You can omit the number for the last character in a subset. This representation is valid only in *string2*.

[:*class*:]

 This represents all characters that belong to the character class *class* in the locale indicated by **LC_CTYPE**. When the class [:upper] or [:lower:] appears in *string1* and the opposite class, [:lower:] or [:upper:] appears in *string2*, **tr** uses the **LC_CTYPE** tolower or toupper mappings in the same relative positions.

[=*c*=]

 This represents all characters that belong to the same equivalence class as the character *c* in the locale indicated by **LC_COLLATE**. Only international versions of the code support this format.

Examples

```
tr -cs "[:alpha:]" "\n*" <file1 >file2
```

creates a list of all words (strings of letters) found in file1 and puts it in file2.

Localization

tr uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**

See for more information.

Exit Values

Possible exit status values are:

0

 Successful completion

1

 Failure because of unknown command line option, or too few arguments

Portability

POSIX.2, X/Open Portability Guide

tr is downward-compatible with both the UNIX Version 7 and System V variants of this command, but with extensions (C escapes, handles ASCII NUL, internationalization).

trap — Intercept abnormal conditions and interrupts

trap [' *handler* '] [*event ...*]

Purpose

trap intercepts certain kinds of exception conditions. Any signal may be intercepted by specifying an event corresponding to the signal number.

If there are no arguments at all, **trap** prints a list of all the traps and their commands.

Operands

trap recognizes the following operands:

handler
> is a command list. It is usually more than one word, and so you must quote it to appear as a single argument. It is scanned when the trap function is initially invoked. When the trap condition is raised, the shell scans the command list again and runs the commands. A missing argument or an argument of - (dash) resets the default trap condition. A null argument ('') causes the trap condition to be ignored.

event
> is the condition to be intercepted.
>
> With an *event* of ERR, **trap** invokes the *handler* upon any command having a nonzero exit status. The exception to this is conditions in **if**, **while**, and **until** statements. This trap is not inherited within a function.
>
> With an *event* of 0 or EXIT, **trap** invokes the *handler* during exit from the shell. Within a function, it is invoked during exit from the function.
>
> Any other event corresponds to the name or number of a signal supported by OpenExtensions. These signal names and numbers are listed in Table 11 on page 335. When using a signal name, enter the name with uppercase characters and do not use the first three characters (SIG). For example, to use signal name SIGALRM, enter only ALRM.

Table 11. Signals Supported by OpenExtensions

Signal Name	Signal Number	Description
SIGABND	18	Abend
SIGABRT	3	Abnormal termination
SIGALRM	14	Timeout
SIGCHLD	20	Child process terminated or stopped
SIGCONT	19	Continue if stopped
SIGFPE	8	Erroneous arithmetic operation, such as division by zero or an operation resulting in overflow
SIGHUP	1	Hangup detected on controlling terminal
SIGILL	4	Detection of an incorrect hardware instruction
SIGINT	2	Interactive attention
SIGIO	23	Completion of input or output

Table 11. Signals Supported by OpenExtensions (continued)

Signal Name	Signal Number	Description
SIGKILL	9	Termination (cannot be caught or ignored)
SIGNULL	0	Null; no signal sent (cannot be caught or ignored)
SIGPIPE	13	Write on a pipe with no readers
SIGQUIT	24	Interactive termination
SIGSEGV	11	Detection of an incorrect memory reference
SIGSTOP	7	Stop (cannot be caught or ignored)
SIGTERM	15	Termination
SIGTSTP	25	Interactive stop
SIGTTIN	21	Read from a controlling terminal attempted by a member of a background process group
SIGTTOU	22	Write from a controlling terminal attempted by a member of a background process group
SIGUSR1	16	Reserved as application-defined signal 1
SIGUSR2	17	Reserved as application-defined signal 2

If a signal is being ignored when you enter the shell, the shell continues to ignore it without regard to any traps.

Usage Notes

trap is a built-in shell command.

Examples

On error or exit, this example deletes a temporary file created during command execution.

```
trap 'rm −f /tmp/xyz$$; exit' ERR EXIT
```

When an interrupt signal is received, the example prompts whether to abort, and exits if the answer is y.

```
trap 'read REPLY?"ABORT??"
      case $REPLY in
      y)      exit 1;;
      esac'   2
```

This example saves your shell history file (specified by the value you give the **HISTFILE** environment variable) before timing you out, so you can restore it when you log on again.

```
trap 'cp $HISTFILE $HOME/old_hist.bak; exit' ALRM
```

Exit Values

Possible exit status values are:

0

Successful completion

1

Failure due to any of the following:

- Incorrect signal name

- Incorrect signal number

2
Incorrect command-line argument

Messages and Return Codes

Possible error messages include:

name **not a valid trap name**
You specified an unrecognized trap name. The usual cause of this error is a typing mistake on the command line.

Portability

POSIX.2, X/Open Portability Guide.

Related Commands

sh

true — Return a value of 0

```
true [argument ...]
```

Purpose

The `true` command simply yields an exit status of zero (success). This can be surprisingly useful—for example, when you are evaluating shell expressions for their side effects.

Usage Notes

This command is provided as both an external utility and a shell built-in command.

Exit Values

Since this command always succeeds, the only possible exit status is:

0
    Successful completion

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

sh

tty — Return the user's terminal name

```
tty[−s]
```

Purpose

tty displays the file name of the terminal device associated with the standard input.

Options

−s
　　Does not display the name; the exit status of **tty** indicates whether the standard input is a terminal.

Localization

tty uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
　　Standard input is a terminal.

1
　　Standard input is not a terminal.

2
　　Failure because of an unknown command-line option, or too many arguments.

Messages and Return Codes

Possible error messages include:

Not a tty
　　The standard input is not associated with a terminal.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The POSIX standard considers the **−s** option to be obsolete.

type — Tell how the shell interprets a name

---

type *name ...*

Purpose

type identifies the nature of one or more names. Names can be shell reserved words, aliases, shell functions, built-in commands, or executable files. For executable files, the full path name is given.

Usage Notes

type is a built-in shell command.

Exit Values

Possible exit status values are:

0
    Successful completion

2
    Failure because of an incorrect command-line argument

Messages and Return Codes

Possible error messages include:

***name* is not found**
    type could not locate the specified name. Check that the *name* was specified properly and that you have the appropriate permissions.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

alias, **sh**, **whence**

typeset — Assign attributes and values to variables

---

typeset ±f[tux] *name name …*
typeset [±lprtuxH] [±iLRZ[*number*]] [*variable*[=*value*] …]

Purpose

Invoking **typeset** with no options displays a list of all variables and their attributes. This list is sorted by variable name and includes quoting so that it can be reinput to the shell with the built-in command **eval**. When only arguments of the form **+***option* are specified, **typeset** displays a list of the variables that have all specified attributes set. When only arguments of the form **–***option* are present, **typeset** displays a list of all the variables having all the specified attributes set, and also displays their values.

When the **f** option is used, **typeset** applies to functions; otherwise, it applies to variables. For functions, the only other applicable options are **–t**, **–u** and **–x**.

If the command line contains at least one *variable*, the attributes of each *variable* are changed. In this case, parameters of the form **–***option* turn on the associated attributes. Parameters of the form **+***option* turn off the associated attributes. (Notice that, contrary to what you might expect, - means *on,* and + means *off.*) Parameters of the form *variable=value* turn on the associated attributes and also assign *value* to *variable*.

When **typeset** is invoked inside a function, a new instance of each *variable* is created. After the function ends, each *variable* is restored to the value and attributes it had before the function was called.

Options

–H

Performs POSIX-to-host-name file mapping.

–i[*number*]

Marks each variable as having an integer value, thus making arithmetic faster. If *number* is given and is nonzero, the output base of each *variable* is *number*.

–l

Converts uppercase characters to lowercase in any value assigned to a *variable.* If the **–u** option is currently turned on, this option turns it off.

–p

Writes output to the coprocess. This option is a no-op.

–r

Makes each *variable* read-only. See **readonly**.

–t

Tags each *variable*. Tags are user-defined, and have no meaning to the shell. For functions with the **–f** option, this turns on the xtrace option. See **set** for a discussion of the xtrace option.

–u

Converts lowercase characters to uppercase in any value assigned to a *variable.* If the **–l** option is currently turned on, this option turns it off.

When used with **–f**, **–u** indicates that the functions named in the command line are not yet defined. The attributes specified by the **typeset** command are applied to the functions once they are defined.

–x

Sets each *variable* for automatic export. See **export**.

The last three options that follow justify, within a field, the values assigned to each *variable*. The width of the field is *number* if it is defined and is nonzero; otherwise, the width is that of the first assignment made to *variable*.

–L[*number*]
Left-justifies the values assigned to each *variable* by first removing any leading blanks. Leading zeros are also removed if the **–Z** option has been turned on. Then blanks are added on the end or the end of the value is truncated as necessary. If the **–R** flag is currently turned on, this option turns it off.

–R[*number*]
Right-justifies the values assigned to each *variable* by adding leading blanks or by truncating the start of the value as necessary. If the **–L** flag is currently turned on, this option turns it off.

–Z[*number*]
Right-justifies values assigned to each *variable*. If the first nonblank character of value is a digit, leading zeros are used. See also the **–L** option.

Usage Notes

This is a built-in command of the shell.

Exit Values

Possible exit status values are:

0
Successful completion

2
Failure due to an incorrect command-line argument

If the command is used to display the values of variables, the exit status value is the number of names that are incorrect.

Messages and Return Codes

Possible error messages include:

Base number not in [2,36]
You used the **–i** option to specify a base for an integer, but the base was not in the range 2 through 36. All bases must be in this range.

***name* not a function**
You tried to declare the given name as a function, but the name already referred to something that was not a function (for example, a variable).

Portability

POSIX.2.

Related Commands

export, **readonly**, **sh**

umask — Set or return the file mode creation mask

---

umask [**−S**] [*mode*]

Purpose

umask sets the file-creation permission-code mask of the invoking process to the given *mode*. You can specify the *mode* in any of the formats recognized by **chmod**; see "chmod — Change the mode of a file or directory" on page 61 for more information.

The file-creation permission-code mask (often called the **umask**) specifies the restrictions on the permissions for any file created by the process and plays a part in determining how permission bits are changed.

When a program creates a file, it requests that the file have certain permissions. The **umask** is applied to the requested permissions to determine the actual permissions that the file will have. The actual permissions will be as follows:

- If a permission bit in the **umask** is on, the corresponding bit in the actual permissions will be off.
- If a permission bit in the **umask** is off, the corresponding bit in the actual permissions will be as the program requested.

Therefore, the **umask** is used to "screen out" permissions that a program may request. The **umask** does not affect other mode values, such as set-user-ID, set-group-ID, and the sticky bit.

If the bit is turned off in the **umask**, a process can set it on when it creates a file. This can be done using a symbolic representation of the permissions as on the **chmod** command, or by specifying a numeric **umask** directly. With the symbolic method, you specify the permissions you want to allow. This is converted into the actual **umask** value, which conversely represents what will be screened out. If you specify:

```
umask a=rx
```

You have explicitly set it so that all users have read and execute access. If you were to look at the mask, it would be 0222. The write bit is set, because write is not allowed. If everyone were permitted rwx access, the **umask** would be 0000. For example, if a command attempts to create new files with permissions of w for all, and the **umask** was 0222 as above, the w permissions would not be set.

If you call **umask** without a *mode* argument, **umask** displays the current **umask**.

Options

−S

Displays the umask in a symbolic form:

```
u=perms,g=perms,o=perms
```

giving owner, group and other permissions. Permissions are specified as combinations of the letters **r** (read), **w** (write), and **x** (execute).

Localization

umask uses the following localization environment variables:

- **LANG**
- **LC_ALL**

- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

    Successful completion

1

    Failure due to an incorrect command-line argument, or incorrect *mode*

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

chmod

unalias — Remove alias definitions

```
unalias name...
unalias –a
```

Purpose

unalias removes each alias *name* from the current shell execution environment.

Options

–a
    Removes all aliases in the current shell execution environment.

Usage Notes

This command is built into the shell.

Localization

unalias uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
    Successful completion

>1
    There was an alias that could not be removed

2
    Failure due to an incorrect command-line option or there were two aliases that could not be removed

>2
    Tells the number of aliases that could not be removed

Portability

POSIX.2.

unalias is a built-in shell command.

Related Commands

alias, **sh**

uname — Display the name of the current operating system

---
uname [**−amnrsv**]

Purpose

The **uname** command lets shell scripts and other programs determine configuration information about the machine upon which the shell is running.

Options

The following options select the information to be displayed:

−a
All fields (equivalent to **−mnrsv**).

−m
The processor or machine type.

−n
The node name of this particular machine. The node name usually differentiates machines running at a single location.

−r
The level of CMS in use, expressed as a string **CMS_*l_s_f***, where:

l
The CMS level as returned by QUERY CMSLEVEL

s
The four-digit CMS service level as it appears in DMSLVLTB

f
The CMS level code returned by DMSQEFL in its output parameter *cms_level*.

−s
The name of the operating system. This is the default output, when no options are given.

−v
The level of CP in use, expressed as a string **CP_*l_s_f***, where:

l
The V.R.M number that identifies the CP in use (for example, **7.4.0** identifies Version 7 Release 4.0), taken from the output of QUERY CPLEVEL

s
The four-digit CP service level as it appears in the output of QUERY CPLEVEL

f
The CP level code returned by DMSQEFL in its output parameter *cp_level*.

uname displays the selected information in the following order:

```
<system name>   <nodename>   <release>      <version>      <machine>
```

Examples

The following shell command changes the prompt to identify the node name of the system:

```
export PS1=" `uname -n`$ "
```

Exit Values

Possible exit status values are:

0

    Successful completion

1

    Failure due to inability to find the desired information

2

    Failure due to a incorrect command-line option

Portability

POSIX.2, X/Open Portability Guide, UNIX system V.

Related Commands

sh

uncompress — Undo Lempel-Ziv compression

---

uncompress [−cDdfVv] [*file*]

Purpose

uncompress uses the Lempel-Ziv compression techniques to uncompress data in a file or from the standard input.

When the *file* argument is specified, **uncompress** searches for a file named *file*.Z. It replaces the input file with the uncompressed file named *file* (without the .Z suffix). If this file already exists, **uncompress** will not replace the file unless you specify the **−f** option.

If the *file* argument is not specified, the input data is read from the standard input and written to the standard output.

Because the number of bits of compression is encoded in the compressed data, **uncompress** automatically uses the correct number of bits to uncompress the data.

Options

uncompress accepts the following options:

−c

Writes uncompressed output to the standard output.

−D

Uses extra dictionary packing technique on uncompression. The file must have been compressed using **compress** with the **−D** option.

−f

Forces the file to be uncompressed. **uncompress** does not print an error message if this happens.

−V

Prints the version number of **uncompress**.

−v

Prints the name of each file as it is uncompressed.

Exit Values

Possible exit status values are:

0

Successful completion.

1

Failure due to any of the following:

- Unknown command line option
- Inability to obtain information about an argument file
- File has more than one link
- File is not a regular file
- File is not in compressed format
- File was compressed using more than 16 bits
- Insufficient memory for the decompression table
- Compressed file is corrupt

Messages and Return Codes

Possible error messages include:

uncompress: not in compressed format
> The input file was not compressed by the **compress** command.

Cannot allocate buffer
> There was not enough memory to allow **uncompress** to set up the decompression table or one of the internal work buffers.

cannot stat file
> **uncompress** could not obtain status information about the input or output file. Typically this happens because the file does not exist or you do not have appropriate permissions to obtain this information.

name **has** *n* **other links: unchanged**
> The file named cannot be replaced while it has links pointing to it.

name **not a regular file: unchanged**
> *name* does not refer to a byte file system file. It refers to a directory, socket, pipeline, device, or the standard I/O.

uncompress: file *name***: Incorrect format for –D option**
> The file was not originally compressed using the **–D** option, so you should not use the option on the **uncompress** command.

name **already exists; not overwritten**
> The **–f** option should be used to force overwriting of the output file.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

compress, **zcat**

uniq — Report or filter out repeated lines in a file

```
uniq [-c|-d|-u] [-f number1] [-s number2] [input_file [output_file]]
uniq [-cdu] [-number] [+number] [input_file [output_file]]
```

Purpose

uniq manipulates lines that occur more than once in a file. The file must be sorted, since **uniq** only compares adjacent lines. When you invoke this command with no options, it writes only one copy of each line in *input_file* to *output_file.* If you do not specify *input_file* or you specify –, **uniq** reads the standard input.

If you do not specify *output_file*, **uniq** uses the standard output.

Options

–c
  Precedes each output line with the number of times that line occurred in the input.

–d
  Displays only lines that are repeated (one copy of each line).

–f *number1*
  Ignores the first *number1* fields when comparing lines. Blanks separate fields in the input.

–s *number2*
  Ignores the first *number2* characters when comparing lines. If you specify both **–s** and **–f**, **uniq** ignores the first *number2* characters after the first *number1* fields.

–u
  Displays only those lines that are not repeated.

You can choose only one of the **–c**, **–d**, or **–u** options.

–number
  Equivalent to **–f** *number* (obsolescent).

+number
  Equivalent to **–s** *number* (obsolescent).

Examples

1. The command:

   ```
   uniq
   ```

   is a filter which prints one copy of each different line in its sorted input.

2. The command:

   ```
   uniq -f 2 -s 1
   ```

   compares lines starting with the second character of the third field.

3. The command:

   ```
   uniq -d
   ```

   prints one instance of each repeated line in the input (and omits all unique lines).

Localization

uniq uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
> Successful completion

1
> Failure due to any of the following:
> - Incorrect command-line option
> - Missing *number* after **–f**
> - Missing or incorrect *number* after **–s**
> - Inability to open the input or output file

Messages and Return Codes

Possible error messages include:

Missing character skip count
> You specified **–s** but did not supply a number after the **–s**.

Missing number of fields to skip
> You specified **–f** but did not supply a number after the **–f**.

Field skip not a number in *string*
> In a **-***number* or **+***number* construct, *number* was not a valid number. This could arise because of a typographical error in entering a **–** option.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

comm, **sort**

unset — Unset values and attributes of variables and functions

> **unset** *name...*
> **unset –fv** *name...*

Purpose

Calling **unset** with no options removes the value and attributes of each variable *name*.

Options

–f
> Removes the value and attributes of each function *name*.

–v
> Is equivalent to calling **unset** with no options.

unset cannot remove names that have been set read-only.

Usage Notes

unset is a built-in shell command.

Exit Values

Possible exit status values are:

0
> Successful completion

1
> Failure due to an incorrect command-line option

2
> Failure due to an incorrect command-line argument

Otherwise, **unset** returns the number of specified *names* which are incorrect, not currently set, or read-only.

Messages and Return Codes

Possible error messages include:

***name* readonly variable**
> The given *name* cannot be deleted because it has been marked read-only.

Portability

POSIX.2, X/Open Portability Guide.

Related Commands

sh, **readonly**

wait — Wait for a child process to end

> **wait** [*pid*|*job-id ...*]

Purpose

wait waits for one or more jobs or child processes to complete in the background. If you specify one or more *job-id* arguments, **wait** waits for all processes in each job to end. If you specify *pid*, **wait** waits for the child process with that process ID (PID) to end. If no child process has that process ID, **wait** returns immediately.

If you specify neither a *pid* nor a *job-id*, **wait** waits for the process IDs known to the invoking shell to complete.

Usage Notes

wait is a built-in shell command.

Localization

wait uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

If you specified a *job-id* that has terminated or is unknown by the invoking shell, an error message and a return code of 127 is returned. If you specified a *pid* that has terminated or is unknown to the shell, a return code of 127 is returned. If a signal ended the process abnormally, the exit status is a value greater than 128 unique to that signal; otherwise, possible exit statuses are:

0
> Successful completion.

1–126
> An error occurred.

127
> A specified *pid* or *job-id* has terminated or is unknown by the invoking shell.

Portability

POSIX.2, UNIX systems.

Related Commands

sleep

wc — Count newlines, words, and bytes

wc [−c|−m] [−w] [*file ...*]

Purpose

wc counts the number of <newline>s, words, characters, and bytes in text files. If you specify multiple files, **wc** produces counts for each file, plus totals for all files.

Options

−c
> Prints a byte count. You cannot specify this option with −m.

−l
> Prints a <newline> count

−m
> Prints a character count. You cannot specify this option with −c.

−w
> Prints a word count

The order of options can dictate the order in which **wc** displays counts. For example, **wc −cwl** displays the number of bytes, then the number of words, then the number of <newline>s. If you do not specify any options, the default is **wc −lwc** (<newline>s, then words, then bytes).

A word is considered to be a character or characters delimited by white space.

Note: wc counts bytes, not characters.

Localization

wc uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0
> Successful completion

1
> Failure because of an inability to open the input file

2
> Failure because of an incorrect command-line option

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The way the order of options **–c**, **–l** and **–w** affects the order of display is an extension to traditional implementations of **wc**.

Related Commands

awk, **ed**

whence — Tell how the shell interprets a command name

whence [−v] *name ...*

Purpose

whence tells how the shell would interpret each *name* if used as a command name. Shell keywords, aliases, functions, built-in commands, and executable files are distinguished. For executable files, the full path name is given.

Options

−v
> Gives a more verbose report.

Usage Notes

whence is a built-in shell command.

Exit Values

Possible exit status values are:

0
> Successful completion

1
> Command *name* could not be found

2
> Failure due to an incorrect command-line argument

Portability

POSIX.2.

Related Commands

command, **sh**

xargs — Construct an argument list and run a command

```
xargs [-I placeholder] [-i[placeholder]] [-L number] [-l number]
    [-n [number]] [-ptx] [-E [eofstr]] [-e [eofstr]] [-s size]
    [command [argument ...]]
```

Purpose

The **xargs** command line typically contains the skeleton, or *template,* of another command. This template looks like a normal command, except that it lacks some arguments. **xargs** adds arguments from the standard input to complete the command, then runs the resulting command. If more input remains, it repeats this process.

Options

xargs gets the needed arguments from the standard input. Different options tell how the standard input is to be interpreted to obtain these arguments.

–I *placeholder*
> With this option, **xargs** considers each full line in the standard input to be a single argument. *placeholder* is a string that can appear multiple times in the command template. **xargs** strips the input line of any leading white-space characters and inserts it in place of *placeholder*. For example, with:

> ```
> xargs -I {} mv dir1/{} dir2/{}
> ```

> the standard input should consist of lines giving names of files that you want moved from `dir1` to `dir2`. **xargs** substitutes these names for the `{}` placeholder in each place that it appears in the command template.

> When **xargs** creates arguments for the template command, no single argument can be longer than 255 characters after the input has replaced the placeholders. The **–x** option is automatically in effect if **–I** or **–i** is used.

–i[*placeholder***]**
> Behaves like **–I**, except that *placeholder* is optional. If you omit *placeholder*, it defaults to the string {}. Thus, the previous example could be written as either of:

> ```
> xargs -i mv dir1/{} dir2/{}
> xargs -i{} mv dir1/{} dir2/{}
> ```

–L *number*
> With this option, **xargs** reads *number* lines from the standard input and concatenates them into one long string (with a blank separating each of the original lines). **xargs** then appends this string to the command template and runs the resulting command. This process is repeated until **xargs** reaches the end of the standard input; if there are fewer than *number* lines left in the file the last time the command is run, **xargs** just uses what is there.

> With this option, a line must contain at least one nonblank character; blank lines are skipped and do not count toward the number of lines being added to the template. **xargs** considers a line to end at the first newline character, unless the last character of the line is a blank or a tab; in this case, the current line is considered to extend to the end of the next non-empty line.

> If you omit the **–L** or **–l** option, the default number of lines read from standard input is 1. The **–x** option is automatically in effect if **–l** is used.

–l *number*
> Acts like the **–L** option, but the *number* argument is optional. *number* defaults to 1.

–n *number*

In this case, **xargs** reads the given number of arguments from the standard input and puts them on the end of the command template. For example:

```
xargs -n 2 diff
```

obtains two arguments from the standard input, appends them to the **diff** command, and then runs the command. It repeats this process until the standard input runs out of arguments. When you use this option, **xargs** considers arguments to be strings of characters separated from each other by white-space characters (blanks, horizontal tabs, or newlines). Empty lines are always skipped (that is, they don't count as arguments). If you want an input argument to contain blanks or horizontal tabs, enclose it in double quotation marks or single quotation marks. If the argument contains a double quotation mark character ("), you must enclose the argument in single quotation marks. Conversely, if the argument contains a single quotation mark (') (or an apostrophe), you must enclose the argument in double quotation marks. You can also put a backslash (\) in front of a character to tell **xargs** to ignore any special meaning the character may have (for example, white-space characters, or quotation marks).

xargs reads fewer than *number* arguments if:

- The accumulated command line length exceeds the *size* specified by the **–s** option (or {LINE_MAX} if you did not specify **–s**)
- The last iteration has more than zero, but less than *number* arguments remaining

If you do not specify the **–n** option, the default number of arguments read from standard input is 1.

Typically, an **xargs** command uses exactly one of the options just described. If you specify more than one, **xargs** uses the one that appears last on the command line. If the command has none of these options, **xargs** keeps reading input until it fills up its internal buffer, concatenating arguments to the end of the command template. When the buffer is full, **xargs** runs the resulting command, and then starts constructing a new command. For example:

```
ls | xargs echo
```

prints the names of files in the working directory as one long line. When you invoke **xargs** this way, the total length of all arguments must be less than the size specified by the **–s** option (see "Other Options" on page 358).

If no command template appears on the command line, **xargs** uses **echo** by default. When **xargs** runs a command, it uses your search rules to find the command; this means that you can run shell scripts as well as normal programs.

The command you want to execute should be in your search **$PATH**.

xargs ends prematurely if it cannot run a constructed command or if an executed command returns a nonzero status.

If an executed command is a shell program, it should explicitly contain an **exit** command to avoid returning a nonzero by accident; see **sh** for details.

Other Options

You can use the following options with any of the three main options.

–E *eofstr*

Defines *eofstr* to represent end-of-file on the standard input. For example:

```
-E :::
```

tells **xargs** that ::: represents the end of the standard input, even if an input file continues afterward. If there is no **–E** or **–e** option, a single underscore (_) marks the end of the input.

−e *eofstr*

Acts like **−E** but the *eofstr* argument is optional. If you specify **−e** without *eofstr*, there is no end-of-file marker string, and _ is taken literally instead of as an end-of-file marker. **xargs** stops reading input when it reaches the specified end-of-file marker or the true end of the file.

−p

Prompts you before each command. This turns on the **−t** option so that you see each constructed command before it is run. Then **xargs** displays ? . . ., asking if you really want to run this command. If you type a string beginning with y, **xargs** runs the command as displayed; otherwise, the command is not run, and **xargs** constructs a new command.

−s *size*

Sets the maximum allowable size of an argument list to *size* characters (where *size* is an integer). The value of *size* must be less than or equal to the system variable LINE_MAX; if you omit the **−s** option, the default allowable size of an argument list is LINE_MAX. The length of the argument list is the length of the entire constructed command; this includes the length of the command name, the length of each argument, plus one blank for separating each item on the line.

−t

Writes each constructed command to the standard error just before running the command.

−x

Kills **xargs** if it creates a command that is longer than the size given by the **−s** option (or {LINE_MAX} is **−s** was not specified). This option comes into effect automatically if you specify **−i** or **−l**.

Examples

The following displays file names in three columns:

```
ls | xargs -n 3 echo
```

Environment Variables

PATH

Contains a list of directories that constitute your search path.

Localization

xargs uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

Successful completion of all commands.

1−125

Failure due to any of the following:

- **xargs** could not assemble a command line.
- One or more invocations of *command* returned a nonzero exit status.
- Some other error occurred.

> **126**
>> **xargs** found *command* but could not invoke it.
>
> **127**
>> **xargs** could not find *command*.

Limits

The maximum length of a constructed command is LINE_MAX bytes.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–e**, **–E**, **–i**, **–I**, **–l**, **–L**, and **–p** options are extensions of the POSIX standard.

Related Commands

echo, **find**, **sh**

yacc — Use the yacc compiler

yacc [–dhlmqstv] [–b *file.prefix*] [–o *file.c*] [–D *file.h*] [–p *prefix*]
   [–P *yyparse.c*] [–S *statesfile*] [–V *stats*] *gram.y*

Purpose

yacc converts a context-free LALR(1) grammar found in the input file *gram.y* into a set of tables that together with additional C code constitute a parser to recognize that grammar. If you specify an input file named -, **yacc** reads the grammar from the standard input. By default, **yacc** places the parsing tables and associated C code into the file **y.tab.c**.

You can find detailed information on writing parsers using **yacc** in *z/VM: OpenExtensions Advanced Application Programming Tools*.

Options

The following options modify the default operation of **yacc**:

–b *file_prefix*
> Uses *file_prefix* instead of y as the prefix for all output file names. For example, **yacc** would name the parsing table *file_prefix*.**tab.c** rather than **y.tab.c**.

–D *file.h*
> Generates the file *file.h*, which contains the constant definition statements for token names. This lets other modules of a multimodule program access these symbolic names. This is the same as **–d**, except that the user specifies the include file name.

–d
> Generates the file **y.tab.h**, which contains the constant definition statements for token names. This lets other modules of a multimodule program access these symbolic names. This is the same as **–D**, except that the user does not specify the header file name.

–h
> Displays a brief list of the options and quits.

–1
> Disables the generation of **#line** statements in the parser output file, which are used to produce correct line numbers in compiler error messages from *gram.y*.

–m
> Prints memory usage, timing, and table size statistics on the standard output.

–o *file.c*
> Places the generated parser tables into *file.c* instead of the default **y.tab.c**.

–P *yyparse.c*
> Indicates that the C parser template is found in the file **yyparse.c**. If you do not specify this option, this parser template is located in **/etc/yyparse.c**.

–p *prefix*
> By default, **yacc** prefixes all variables and defined parameters in the generated parser code with the two letters yy (or YY). In order to have more than one **yacc**-generated parser in a single program, each parser must have unique variable names. **–p** uses the string *prefix* to replace the yy prefix in variable names. *prefix* should be entirely in lowercase because **yacc** uses an uppercase version of the string to replace all YY variables. We recommend a short prefix (such as zz) because some C compilers have name length restrictions for identifiers. You can also set this identifier with a **%prefix** directive in the grammar file.

–q
> Disables the printing of warning messages.

−s

Writes a state description to the file **states.out**. This file is indexed by pointers in the table `yyStates`, so that any state can be quickly read and displayed.

−S

Is similar to **−s** except that the state description is written to *statesfile* file.

−t

Enables debugging code in the generated parser. **yacc** does not normally compile this code because it is under the control of the preprocessor symbol YYDEBUG.

This option is therefore equivalent to either setting YYDEBUG on the C compiler command line or specifying **#define YYDEBUG** statement in the first section of the grammar.

−V *stats*

Writes a verbose description of the parsing tables and any possible conflicts to the file *stats*.

This is the same as **-v** except the user specifies the file name.

−v

writes a verbose description of the parsing tables and any possible conflicts to the file **y.output**.

Files

y.output

Default statistics file when you specify **-v**.

y.tab.c

Default file for the generated parser.

y.tab.h

Default header file when you specify **−d**.

/etc/yyparse.c

Default parser template.

states.out

Default state description file when you specify **−s**.

Localization

yacc uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix C, "Localization," on page 477 for more information.

Exit Values

Possible exit status values are:

0

Successful completion

1

Failure because of any of the following:

- *number* rules never reduced
- Reduce-reduce conflict
- Shift-reduce conflict
- *NAME* should have been defined earlier

- \000 not permitted
- EOF encountered while processing *%union*
- EOF in string or character constant
- EOF inside comment
- Use of $*number* not permitted
- Nonterminal *number*, entry at *number*
- Action does not terminate
- Bad *%start* construction
- Bad syntax in *%type*
- Bad syntax on $<*ident*> clause
- Bad syntax on first rule
- Inability to find parser
- Inability to open input file
- Inability to open table file
- Inability to open temporary file
- Inability to open **y.output**
- Inability to place goto
- Inability to reopen action temporary file
- Default action causes potential type clash
- EOF before %}
- *%prec syntax* not permitted
- *nnn* construction not permitted
- Comment not permitted
- Option not permitted
- Incorrect or missing ' or "
- Incorrect rule: missing semicolon, or | ?
- Internal **yacc** error
- Incorrect escape, or incorrect reserved word
- Item too big
- More than *number* rules
- Must return a value, since *LHS* has a type
- Must specify type for *name*
- Must specify type of $*number*
- Newline in string.
- No space in action table
- Nonterminal *symbol* not permitted after *%prec*
- Nonterminal *symbol* never derives any token string
- Nonterminal *symbol* not defined
- Optimizer cannot open temporary file
- Out of space in optimizer
- Out of state space
- Redeclaration of precedence of *symbol*
- Redeclaration of type of *symbol*

- Syntax error
- Token incorrect on *LHS* of grammar rule
- Too many characters in ID's and literals
- Too many look-ahead sets
- Too many nonterminals
- Too many states
- Too many terminals
- Type redeclaration of nonterminal *symbol*
- Type redeclaration of token *symbol*
- Unexpected EOF before %
- Unterminated < `...` > clause
- Working set overflow
- **yacc** state or nolook error

Messages and Return Codes

Possible error messages include:

No input file
You did not specify a grammar file **gram.y** on the command line.

No parser produced
Analysis of the input grammar shows that it contains inaccessible or ungrounded nonterminal symbols. Check the preceding report and revise the grammar.

Out of memory at size bytes
The specified grammar is too complex to process within the memory resources of the current configuration.

Limits

yacc dynamically allocates all internal tables so that grammar size and complexity are limited only by available memory.

Portability

POSIX.2, UNIX systems.

The **–D**, **–h**, **–m**, **–p**, **–q**, **–S**, **–s**, and **–V** options are extensions of the POSIX standard.

Related Commands

For additional information, see *z/VM: OpenExtensions Advanced Application Programming Tools*.

zcat — Uncompress and display data

> **zcat** [**–DVv**] [*file…*]

Purpose

zcat takes one or more compressed data files as input and uncompresses them. The data files should be compressed with the **compress** command. If no data files are specified on the command line, **zcat** reads the standard input. You can also pass the standard input to **zcat** by specifying **\-** as one of the files on the command line.

zcat uncompresses the data in all the input files and writes the result on the standard output. **zcat** concatenates the data in the same way **cat** does.

zcat expects the names of all the compressed input files to end in .Z, even if a file name is specified as input without the suffix. For example, if the command is `zcat myfile.abc`, **zcat** looks for `myfile.abc.Z`.

zcat is equivalent to **uncompress -c**.

Options

zcat accepts the following options:

–D
> Uncompresses files that were compressed using the dictionary option of **compress**.

–V
> Prints the version number of **uncompress** that **zcat** calls.

–v
> Prints the name of each file as it is uncompressed.

Exit Values

Possible exit status values are:

0
> Successful completion.

1
> Failure due to any of the following:
>
> - Failure of **uncompress** command
> - Unknown command line option
> - File is not in compressed format
> - File was compressed with a number of bits **zcat** cannot handle
> - Insufficient memory for the decompression table
> - Compressed file is corrupt

Messages and Return Codes

Possible error messages include:

zcat: not in compressed format
> The input file was not compressed by the **compress** command.

***name* not a regular file: unchanged**
> *name* does not refer to a byte file system file. It refers to a directory, socket, pipeline, device, or the standard I/O.

zcat: file *name*: Incorrect format for -D option
> The file was not originally compressed with the **–D** option, so you should not use that option on the **zcat** command.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Commands

cat, **compress**, **uncompress**

Chapter 2. OPENVM CMS Commands

OPENVM commands may be used to manipulate data residing in the byte file system (BFS). These commands accept a BFS path name as input. OPENVM commands allow a user to perform such tasks as editing, erasing, renaming and changing permissions and ownerships of BFS files, directories, and other BFS object types.

Before using any of the OPENVM CMS commands, the CMS SET RELPAGE command must be set to ON, which is the default. If it has been set to OFF, you must issue the SET RELPAGE ON command to avoid virtual storage management problems while using OPENVM CMS commands.

Note: Considering strict POSIX terminology refers to all objects as files, for the sake of clarity, the term **file** will be used to refer to a BFS regular file.

The term **object** will be used in referring to all BFS data types (BFS regular files, directories, external links, symbolic links, named pipes, and so on). Note that a path name does not uniquely identify a file. There may be many links (or names) to a given file.

In addition to the BFS path name, a CMS short file name is associated with each unique file; this is a system generated value (unique to each file within a BFS).

Some Shared File System commands that accept *bfsid* (*filepoolid:filespaceid.*) as input may be used by an SFS administrator to operate on files within the BFS. These commands accept CMS short file name as the file name and file type of a file within the BFS. BFS subdirectories, and other BFS objects other than BFS regular files may not be operated on from these commands.

Understanding Byte File System (BFS) Path Name Syntax

All objects (files, directories, and so on) in the OpenExtensions byte file system (BFS) are identified through path names. A path name identifies the object within the BFS hierarchy by specifying the directories leading to the object.

A BFS path name can represent a file system accessed through the Network File System (NFS). The NFS file system can be on a remote or local system, which can be VM or non-VM. The OPENVM MOUNT command or the mount (BPX1MNT) callable service links an NFS file system to a BFS path name, enabling it to be used on most commands and interfaces that accept BFS path names.

For simplicity in command syntax, the BFS path name identifier is usually shown as the variable, *pathname.*

Format

Notes:

[1] A single quotation mark or a double quotation mark is an optional delimiter. If specified to signify the start of a path name, the identical delimiter also must be specified to signify the end of the path name. (See usage note "7" on page 371.)

[2] The minimum path name is a single slash (/).

[3] The ending slash is required only if one or more path name components are also specified.

Parameters

/../VMBFS:*filepoolid:filespaceid/*
> is a construct that identifies the byte file system. It is referred to as the fully qualified BFS root.

> **/../VMBFS:**
>> is a keyword string that indicates this is an OpenExtensions byte file system. This string is not case sensitive and must end with a colon (:).

> *filepoolid***:**
>> is the name of the file pool that contains the BFS data. The file pool name can be up to eight characters long and is not case sensitive. The first character must be alphabetic, but the remaining characters can be alphabetic or numeric. The name must be followed by a colon (:).

> *filespaceid/*
>> is the name of the file space where the BFS resides. The file space ID can be up to eight characters long and is not case sensitive. The name must be followed by a slash (/) if one or more path name components are also specified.

pathname_component
> is the name of an object in the BFS hierarchy. Each path name component can be 1-255 characters in length. The slash character (/) and the null character (X'00') are not valid within a path name component. Path name component names are case sensitive.

When multiple path name components are specified, they must be separated by slashes.

All path name components prior to the last one specified will be interpreted as directory names in the hierarchy. The last path name component, when not followed by a slash, can be a directory or another type of object. If the last path name component is followed by a slash, it will always be interpreted as a directory.

/

when specified as a single character path name, indicates the root (top) directory of the currently mounted byte file system. The root directory can be assigned by using the OPENVM MOUNT command, or by the POSIXINFO FSROOT statement in your user directory entry.

//

when a path name starts with exactly two slashes, it is not considered to be a BFS name. This type of path name is interpreted as a CMS record file system name by the functions that support redirection to the CMS record file system. When such a name is given as a parameter to a command or function that does not support redirection to the CMS record file system, the request will be rejected.

For example, the shell command:

```
$ c89 pgm.c -o //mymod.module.a
```

will create the file MYMOD MODULE A on your A-disk. The OPENVM command:

```
openvm get ./test/book/ch1.scr  //chapter1.script.a
```

will fail with an error message indicating the file name is not valid.

Note: A path name must not start with two slashes when in the XEDIT environment.

Usage Notes

1. A byte file system can be enrolled in the same file pool as other byte file systems and SFS users.

2. In the OpenExtensions environment, all byte file systems are uniquely identified with the `/../vmbfs:`*filepoolid*`:`*filespaceid* construct.

3. Path names can be specified in several ways:

   - When the first character of the path name is not a slash, the path name is known as a *relative path name*. The search for the BFS object starts at the working directory. To establish the working directory, use the OPENVM SET DIRECTORY command or the chdir (BPX1CHD) callable service. To find the value of the current working directory, use the OPENVM QUERY DIRECTORY command or the getcwd (BPX1GCW) callable service.

   - When `/../vmbfs:`*filepoolid*`:`*filespaceid*`/` is specified at the start of a path name, it is referred to as a *fully qualified path name*. The object is searched for in the byte file system, which is defined as file space *filespaceid* in file pool *filepoolid*. The byte file system does not need to be explicitly mounted.

   - If the path name starts with a slash (but not `/../vmbfs:`*filepoolid*`:`*filespaceid*`/`), the path name is known as an *absolute path name*. The search for the object starts from the root of the currently mounted byte file system. The root directory can be established by using the OPENVM MOUNT command or the mount (BPX1MNT) callable service, or by the POSIXINFO FSROOT statement in your user directory entry. To find the value of the root directory, use the OPENVM QUERY MOUNT command or the uname (BPX1UNA) callable service.

   For more information on OpenExtensions callable services, see *z/VM: OpenExtensions Callable Services Reference*. For more information on user directory statements, see *z/VM: CP Planning and Administration*.

4. The entire path name must be in the range of 1-1023 characters. Individual path name components cannot exceed 255 characters. All characters are valid within a path name, with the following restrictions:

   - The null character (X'00') is not permitted within a path name.

- A slash (/) is interpreted as the delineator of a path name component.

For an application to be portable to the broadest set of environments, POSIX standards suggest that the application restrict the maximum length of a BFS path name component to 14 characters and use only the following characters:

A-Z
 Uppercase alphabetic

a-z
 Lowercase alphabetic

0-9
 Numeric

.
 Period

_
 Underscore

-
 Dash

5. Path name components are case sensitive. For example, Abc, abC, and ABC are valid unique path name components. When a path name is entered on the CMS command line, it will not be uppercased. However, a path name entered on the XEDIT command line will be uppercased when SET CASE UPPER is in effect.

6. There are two BFS path name components that have special meaning during path name resolution. These are:

.
 The path name component consisting of a single dot character (.) refers to the directory specified by the preceding path name component.

 Some dot (.) examples:

 a. If you specified a path name of:

```
/joes/recipes/./pie
```

 It would be equivalent to:

```
/joes/recipes/pie
```

 b. If you specified a path name of:

```
./joes
```

 It would be equivalent to:

```
joes
```

..
 The path name component consisting of two dot characters (. .), known as dot-dot, refers to the parent directory of its predecessor. As a special case, in the root directory, dot-dot refers to the root directory itself. The construct `/.  ./vmbfs:`*filepoolid*`:`*filespaceid*`/` is the only exception.

 Some dot-dot (. .) examples:

 a. If you had previously set your working directory (using OPENVM SET DIRECTORY) to:

```
/joes/recipes/
```

And you specified a relative path name of `../tools`, this would be equivalent to specifying an absolute path name of:

```
/joes/tools
```

b. If you are working in `/bin/util/src`, and you want to go to `/bin/util`, you can enter:

```
openvm set directory ..
```

c. If you are working in `/u/rexx/prog/src`, and you want to refer to the file `test` in the directory `/u/rexx/appl/examples`, you could use the following path name to refer to that file:

```
../../appl/examples/test
```

7. Enclose a BFS path name within single quotation marks ('*pathname*') or double quotation marks ("*pathname*") if it contains any of the following characters. Results are unpredictable if a path name or path name component contains any of these characters and it is not enclosed within quotation marks.

> Blank space

(
> Left parenthesis

)
> Right parenthesis

'
> Single quotation mark

"
> Double quotation mark

*
> Asterisk

=
> Equal sign

Notes:

a. If a path name includes a single quotation mark, specify the path name in one of these ways:
   - Place double quotation marks around the path name.
   - Place single quotation marks around the path name, but be sure to use two additional single quotation marks to denote the single quotation mark that is part of the path name.

b. If a path name includes a double quotation mark, specify the path name in one of these ways:
   - Place single quotation marks around the path name.
   - Place double quotation marks around the path name, but be sure to use two additional double quotation marks to denote the double quotation mark that is part of the path name.

c. All characters are taken literally; no symbolic substitution is done.

Some CMS environment examples:

a. To list files in a directory called `my dir` that is directly under your root directory, you must specify:

```
openvm listfile '/my dir'
```

Note that:

```
openvm listfile  /a/b/c
```

is equivalent to:

```
openvm listfile  '/a/b/c'
```

b. To list the files in a directory called /a/b b'/c, you can enter the name in either of the following ways:

```
openvm listfile '/a/b b''/c'
openvm listfile "/a/b b'/c"
```

c. To list the files in a directory called /a/b b"/c, you can enter the name in either of the following ways:

```
openvm listfile "/a/b b""/c"
openvm listfile '/a/b b"/c'
```

Some XEDIT examples:

a. To XEDIT a file called my dir/my file that is directly under your root directory, you can specify:

```
xedit '/my dir/my file' (nametype bfs
```

The NAMETYPE BFS option was specified to distinguish the file being edited as a BFS file instead of a CMS file.

Note that:

```
xedit /a/b/c
```

is equivalent to:

```
xedit '/a/b/c'
```

b. To edit a file called /a/b b'/c, you can enter the name in either of the following ways:

```
xedit '/a/b b''/c'
xedit "/a/b b'/c"
```

c. To edit a file called /a/b b"/c, you can enter the name in either of the following ways:

```
xedit "/a/b b""/c"
xedit '/a/b b"/c'
```

8. In the CMS environment, the OPENVM commands can be entered on a single line or on multiple lines. To enter multiple lines, type OPENVM and press the Enter key. You will get a message prompting you to enter more input lines. You must enter a null line to indicate the end of your command input. This is particularly useful for entering long path names.

Leading and trailing blanks entered on an input line are preserved when multiple lines are put together. A blank is needed after a keyword and its following operand.

This is an example of entering multiple lines:

```
 openvm
```

(Press the Enter key)

```
DMSWOV2140R Enter operands: (enter a null line to
           indicate that you are finished)

 listfile
```

(where LISTFILE is followed by a blank, and you press the Enter key)

```
 '/A
 /B
```

(where /B is followed by a blank)

```
/c' (header
```

(and press the Enter key twice to enter a null line)

This is equivalent to the one-line command:

```
openvm listfile '/A/B /c' (header
```

Because the path name /A/B /c contains a blank, it must be enclosed in quotation marks on input.

9. Multiple adjacent slashes (//) in a path name (except the special case when a path name starts with exactly two slashes) are interpreted as a single slash by OPENVM commands. However, these multiple slashes are included in the maximum path name length check.

10. **Attention:**

   - You might need to change your terminal settings in order to specify a path name that contains certain special characters. For example, you want to use the # character in a name, but the default line end symbol is #. So you might have to change your logical line end symbol using the CP TERMINAL LINEND command.

     Use the QUERY LINEND and SET LINEND commands to find out and define your current line end character for full-screen CMS.

   - If you choose to enclose a path name containing blanks in double quotation marks ("), you might need to use the CP TERMINAL ESCAPE command to change your logical escape symbol, because the default value is a double quotation mark.

     Use the CP QUERY TERMINAL command to display the special characters that are in effect for your terminal.

Understanding Network File System (NFS) Path Name Syntax

The Network File System (NFS) path name identifies a file system exported by a remote NFS server. While NFS may be used to mount file systems on your local VM System, it is recommended that you use a BFS path name instead.

Format

Notes:

[1] A single quotation mark or a double quotation mark is an optional delimiter. If specified to signify the start of a path name, the identical delimiter must also be specified to signify the end of the path name. (See usage note "2" on page 374.)
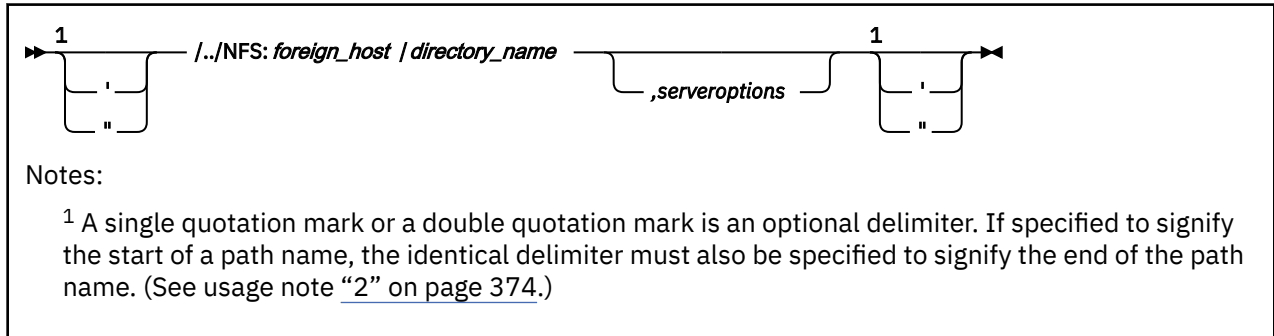
Parameters

/../NFS
is a keyword string that indicates the specified path name is a fully-qualified remote file system, accessed by way of a Network File System server. The NFS keyword is not case sensitive.

: (colon)
is a separator that must be specified following the NFS keyword.

foreign_host
identifies the name of the foreign host. Specify *foreign_host* using an internet host name or a dotted-decimal address. This name is not case sensitive.

/ (slash)
is a separator that must be specified following the *foreign_host*.

directory_name
identifies the file system or directory to be mounted. The format of *directory_name* is dependent upon the operating system running at the site identified by *foreign_host*. This name may be case sensitive.

serveroptions
are NFS server MOUNT options, which depend upon the NFS server at *foreign_host*.

The delimiter between *directory_name* and *serveroptions* is defined by the remote host. Typically a comma is used.

Unexpected results may occur if you provide any credentials (UID or GID) in *serveroptions* that differ from credentials used by the NFS client. See the NETRC, USERID, and ANONYMOUS parameters of "OPENVM MOUNT" on page 407 for information about how the NFS client determines which UNIX-style credentials are used on the request. If those credentials are not consistent with what the NFS server is using, you may have problems with operations such as file creation.

Usage Notes

1. The *directory_name* portion of the NFS path name is generally case sensitive. VM's minidisk file system and Shared File System are exceptions to this rule.

2. Enclose an NFS path name within single quotation marks ('*pathname*') or double quotation marks ("*pathname*") if it contains any of the following characters.

   Blank space

   (
   Left parenthesis

)

    Right parenthesis

'

    Single quotation mark

"

    Double quotation mark

*

    Asterisk

=

    Equal sign

Notes:

a. If a path name includes a single quotation mark, specify the path name in one of these ways:

  • Place double quotation marks around the path name.
  • Place single quotation marks around the path name, but be sure to use two additional single quotation marks to denote the single quotation mark that is part of the path name.

b. If a path name includes a double quotation mark, specify the path name in one of these ways:

  • Place single quotation marks around the path name.
  • Place double quotation marks around the path name, but be sure to use two double quotation marks to denote the double quotation mark that is part of the path name.

c. All characters are taken literally; no symbolic substitution is done.

OPENVM CREATE DIRECTORY

```
►►─ OPENVm ──── CREate ──── DIRectory ──── pathname ─►◄
```

Authorization

General User; Byte file system (BFS) permission checking applies to this command.

Purpose

The OPENVM CREATE DIRECTORY command will create a new, empty byte file system (BFS) directory.

Operands

pathname
> Specifies the name of the directory. See "Understanding Byte File System (BFS) Path Name Syntax" on page 368 for a description of the different forms of the BFS path name.

Usage Notes

1. Permissions assigned to the new directory are those in effect for the creation mask. For more information, see "OPENVM SET MASK" on page 453.

   Use OPENVM PERMIT to change permissions for an existing BFS file. For more information, see "OPENVM PERMIT" on page 424.

2. The owner IDs of the new directory are set to the effective UID of the issuer and the GID of the parent directory. Use OPENVM OWNER to change the settings after the directory is created. For more information, see "OPENVM OWNER" on page 416.

3. When *pathname* refers to an object in an NFS-mounted file system, you must meet the authorization requirements imposed by the remote NFS server.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example

```
HELP MSG DMS111E
```

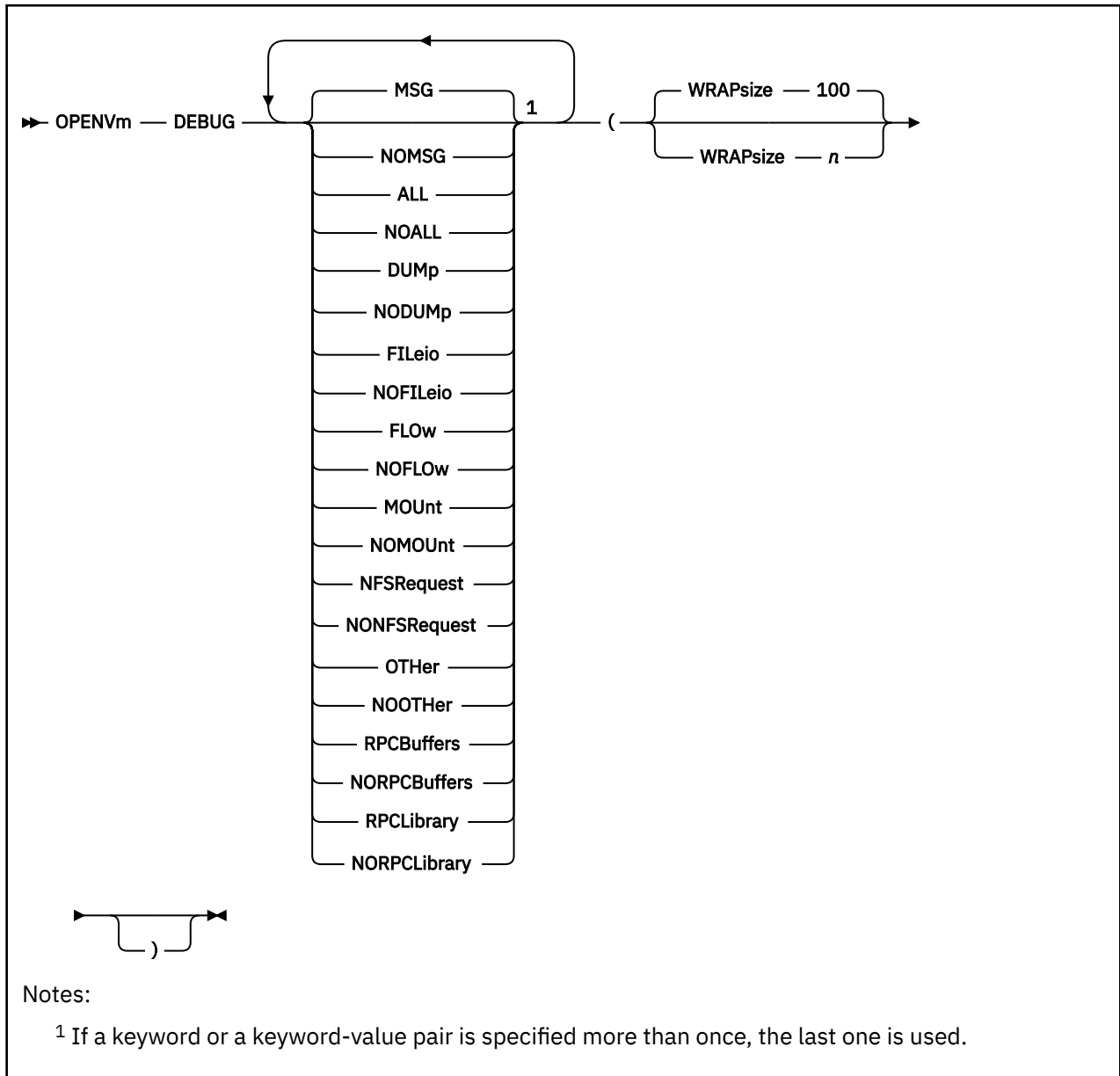| Number	Text	Return Code
DMS1311E	Object already exists: *pathname*	28

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM CREATE EXTLINK

```
►►── OPENVm ──── CREate ──── EXTlink ─── pathname ──►

       ┌──── CMSEXEC ─── fname ────────────────────────────┐
       │                     ┌─────────────────────┐  ¹    │
       │                     └─ ftype ─┬────────────┘       │
       │                               └─ fmode ─┘          │──►◄
       ├──── CMSDATA ─┤ string1 ├──────────────────┤
       ├──── MOUNT ─── string2 ──────────────────┤
       └──── CODE ─── ecode ─── string3 ─────────┘
```

string1

```
►►─┬─ // fname ─── ftype ─┬────────────┬──────┬─ ,access_mode_keyword ──┬──►
   │                      └─ fmode ─┘  ¹      └─ ,explicit_access_mode ─┘
   └─ DD: ─── ddname ──────────────────┘

              ┌──────────────◄──────────────┐
   ►─┬────────┴───────────────────────────┬─►◄
     ├─ ,BLKSIZE= ─┤
     ├─ ,LRECL= ───┤
     ├─ ,RECFM= ───┤
     └─ ,ASIS ─────┘
```

Notes:

¹ Only a space or a period is allowed as a separator between the parts of the CMS file ID.

Authorization

General User

Purpose

OPENVM CREATE EXTLINK creates a byte file system (BFS) object that is referred to as an external link. Depending on the parameters used to create it, an external link can be used to:

- Reference data outside of the BFS (data residing on a CMS minidisk or SFS directory)
- Create an implicit mount point
- Contain data in an application defined format.

Operands

pathname
 is the BFS path name of the external link being created. Refer to "Understanding Byte File System (BFS) Path Name Syntax" on page 368 for a description of the different forms of the BFS path name. *pathname* may not refer to a file in an NFS-mounted file system.

CMSEXEC
    indicates that the file to which the external link is being created is an executable file on a minidisk or
    accessed SFS directory. The referenced executable file must have a file type of MODULE. This CMS file
    will be executed when the external link is specified on the OPENVM RUN command.

fname
    is the file name of the CMS file to be run.

ftype
    is the file type of the CMS file to be run.

fmode
    is the file mode of the CMS file to be run.

CMSDATA
    indicates that the file to which the external link refers will be opened by the C Run Time Library (C
    RTL) ANSI-C *fopen()* routine when the external link is opened.

string1
    contains the parameter list to be associated with the ANSI-C *fopen()* C Run Time Library function.
    This parameter list consists of a CMS file ID or a data definition name (DDNAME) and an access mode
    variable. A CMS file ID must be preceded by exactly two slashes (//). A DDNAME must be preceded by
    the keyword DD:. The total length of *string1* must not exceed 1023 characters. For a more detailed
    description of *string1*, see "CMSDATA Usage Notes" on page 379 and "Examples" on page 380.

    Note: No verification of *string1* occurs when the external link is created.

MOUNT
    indicates the external link is a Mount External Link (MEL). When a MEL is encountered during path
    name resolution, it is treated like a directory with a file system mounted on it; path name resolution
    continues in the "mounted" BFS (that is, in the directory identified by *string2*).

    Note: No verification of *string2* occurs when the external link is created except that *string2* may not be
    a Network File System (NFS) path name.

string2
    is the BFS path name (generally fully qualified) identifying the target directory of the MEL. *string2* must
    be between 1 and 1023 characters.

CODE
    indicates the external link is in an application-defined format.

ecode
    is an integer in the range of 100-200. The *ecode* represents an application-defined format of external
    link.

string3
    is between 1 and 1023 characters long. The format and content of *string3* are defined by the
    application.

Usage Notes

1. The *string1*, *string2*, and *string3* parameters are specified after the keywords CMSDATA, MOUNT, and
   CODE, respectively. In the case of CMSDATA and MOUNT, everything following the keyword on the
   command line will be taken as part of the string parameter. In the case of CODE, everything following
   the *ecode* integer will be taken as part of the string parameter.

   In these string parameters, blanks do not have to be enclosed in quotes. Leading and trailing blanks
   are preserved and will not be suppressed, but one blank is skipped to separate the tokens. For
   example:

```
OPENVM CREATE EXTLINK /MYEXTLINK MOUNT   /../VMBFS:BFS:SANDY/N
                          (where /N is followed by two blanks)
```

   This will place the following, including the extra blanks before and after the path name, into an
   external link called /MYEXTLINK:

```
    /../VMBFS:BFS:SANDY/N
```

Everything after the keyword is included in the string, including quotes. For example:

```
OPENVM CREATE EXTLINK /MYEXTLINK MOUNT '/../VMBFS:BFS:SANDY/N'
```

This will place the following into an external link called /MYEXTLINK:

```
'/../VMBFS:BFS:SANDY/N'
```

Note: Quotes are not needed to delimit a string containing blanks. All string parameters have a maximum length of 1023.

2. Authorization to files from external links will be based on the permissions associated with the external link according to POSIX requirements.

   Additionally, traditional CP/CMS authorization rules are enforced for external links that refer to files residing on minidisks or SFS directories.

3. There is no syntax verification done on the content of external links when they are created. The syntax is verified by the individual functions that refer to the external link.

Mount External Link (MEL) Usage Notes

1. A MEL is not an actual mount point. Therefore, commands that act specifically on mount points (OPENVM QUERY MOUNT and OPENVM UNMOUNT) are not valid for MELs. Use OPENVM LISTFILE and OPENVM QUERY LINK to obtain information about existing MELs. Use OPENVM ERASE to delete a MEL.

2. You can create a MEL that in turn references other MELs. A maximum of eight levels of nesting is supported.

3. The target of a MEL must be a directory. However, the target directory does not have to exist when the MEL is created.

4. An attempt to reference multiple MELs with the same target directory will result in an error.

CMSDATA Usage Notes

1. *string1* must consist of either a CMS file ID or a DDNAME, followed by an access mode variable. The access mode variable must start with the access mode, which can be one of the following:

   ,*access_mode_keyword*
   which can be either of the following:

   ,&&&
   signifies that data to the file is read/written as text data. The read/write intent will be determined at the time the file is opened.

   ,&&b
   ,&&B
   signifies that data to the file is read/written as binary data. The read/write intent will be determined at the time the file is opened.

   ,*explicit_access_mode*
   as specified in .

   The access mode may be followed by one or more of the following keyword parameters:

   BLKSIZE= (blksize=)
   LRECL= (lrecl=)
   RECFM= (recfm=)
   ASIS (asis)

   For more details on access modes and keyword parameters listed, see the *"fopen()"* command in *XL C/C++ for z/VM: Runtime Library Reference*.

2. The access mode to be used for the internal ANSI-C *fopen()* will be coded in the external link as &&& or &&b (&&B is translated to &&b). The characters && will be replaced with the access mode specified on the open() request according to Table 12 on page 380 (see example "1" on page 380.):

Table 12. open() Request Access Modes and ANSI-C fopen() Access Modes

Access mode on *open()*	Access mode on the ANSI-C *fopen()*	
	for text data	for binary data
O_RDONLY	r	rb
O_WRONLY	r+	r+b
O_RDWR	r+	r+b
O_WRONLY + O_APPEND	a	ab
O_RDWRLY + O_APPEND	a+	a+b
O_WRONLY + O_TRUNC	w	wb
O_RDWRLY + O_TRUNC	w+	w+b
O_WRONLY + O_APPEND + O_TRUNC	-	-
O_RDWR + O_APPEND + O_TRUNC	-	-

Note: O_WRONLY is not strictly supported; it is mapped to O_RDWR. The O_CREAT, O_EXCL, O_NOCTTY and O_NONBLOCK flags are ignored.

3. If the external link contains an explicit access mode, rather than &&& or &&b (or &&B), the access mode specified on the open request will be overridden by the access mode coded in the external link.

4. External Link files that are opened in the parent will be marked as FD_CLOFRK. Such file descriptors cannot be inherited to a child process; an attempt to do so explicitly will cause a spawn() failure.

   The FD_CLOFRK flag cannot be reset or overridden by fcntl(); an attempt to do so will be ignored.

5. When referencing existing files in CMS format (as contrasted with OS format) with an external link, the specification of RECFM and TYPE parameters are generally sufficient to process the file.

   Note: If a RECFM parameter is not specified and the file is accessed exclusively in read mode, the existing file attributes will be in effect. However, if the referenced file is written to with the external link, *fopen()* defaults will apply. If you wish to retain existing file attributes on output, you must specify "RECFM=*".

6. If the *string1* parameter contains a DDNAME, a FILEDEF must be provided for that DDNAME before the external link path name may be opened successfully. For more information, see *z/VM: CMS Commands and Utilities Reference*.

7. Associating an external link with a DDNAME allows a BFS path name to reference files on any device supported by the FILEDEF command, including tapes, spooling devices, and CMS, OS and VSAM files.

8. If a DDNAME is used to associate a path name with a file, "attribute options" associated with the external link will override file attributes specified on the corresponding FILEDEF command. For more information, see *z/VM: CMS Commands and Utilities Reference*.

9. Coding a DISP MOD in a FILEDEF statement associated with an external link will force all output to the file that is to be appended to the existing file.

Examples

1. Let us assume you entered the command:

```
OPENVM CREATE EXTLINK /u/dpt37/payroll CMSDATA //PAYROLL.FILE.A,
&&&
```

If you later run a C program that has the following statements in it:

```
.....
fd = open("/u/dpt37/payroll",O_RDWR);
read(fd,buffA,n);
lseek(fd,offset,pos);
write(fd,buffB,m);
.....
```

The result would be as if you executed:

```
.....
FILE * stream;
stream = fopen("//PAYROLL.FILE.A","r+ ");
fread(buffA,1,n,stream);
fseek(stream,offset,origin);
fwrite(buffB,1,m,stream);
.....
```

2. If you wish to create a path name called 'abc' within your current working directory to an executable CMS module file named READING on any accessed minidisk or SFS directory, you could enter:

```
OPENVM CREATE EXTLINK  abc CMSEXEC  READING MODULE
```

3. If you wish to create a path name called 'input' within your current working directory to a file named TEST SCORES on a minidisk to be accessed as A that will be used in read only mode, you could enter:

```
OPENVM CREATE EXTLINK input CMSDATA  //TEST.SCORES.A, r
```

4. If you wish to create a path name called 'forms' in your root directory to a file that contains ASA print-control characters, you could enter:

```
OPENVM CREATE EXTLINK  /forms CMSDATA  //DENTAL FORMS A,
r,recfm=A
```

5. Suppose you had an existing file MYIN FILE A with attributes

```
recfm=F, lrecl=100, and blksize=100
```

and you created an external link using the following:

```
OPENVM CREATE EXTLINK  myextl CMSDATA  //MYIN FILE A, w,recfm=*
```

If you wrote to 'myextl', the resultant file would have the same attributes.

6. Suppose another external link was created as:

```
OPENVM CREATE EXTLINK myext2 CMSDATA //MYIN FILE A,
w,recfm=*,blksize=300,lrecl=300
```

If you wrote to 'myext2', the resultant file would have the attributes:

```
recfm=F, lrecl=300, and blksize=300
```

7. If you wish to create an external link called "ddlink" under your current working directory, such that the referenced file was established at execution time with a DDNAME of MYDATA, you could enter:

```
    OPENVM CREATE EXTLINK  ddlink CMSDATA  dd: MYDATA,r,recfm=FB
```

Prior to opening *ddlink*, you must provide a FILEDEF. For example, if you want to look at MY FILE A, you must enter the FILEDEF command:

```
FILEDEF MYDATA DISK MY FILE A
```

before opening *ddlink*.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS1026E	The operation is not supported for an object in an NFS-mounted file system.	28
DMS2112E	Contents of the external link must be between 1 and 1023 characters	40
DMS2143E	There is no external link data specified	24

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM CREATE LINK

```
►►── OPENVm ──── CREate ──── LINk ── pathname1 ──── pathname2 ──►◄
```

Authorization

General User

Purpose

Use OPENVM CREATE LINK to create a new byte file system (BFS) path name to be used to reference another file in the same BFS. The new name does not replace the old one, but provides an additional way to refer to the file.

You cannot create a link to a directory.

Operands

pathname1
> is the BFS path name of the file for which a link is to be created. For a description of the different forms of the BFS path name, see "Understanding Byte File System (BFS) Path Name Syntax" on page 368. *Pathname1* may not refer to a file in an NFS-mounted file system.
>
> **Note:** This request will fail with message DMS2115E when the object being linked has been mounted to your BFS but physically resides on another BFS or in an NFS-mounted file system.

pathname2
> is the BFS path name of the new link name being created to reference that file.

Usage Notes

1. OPENVM CREATE LINK allows you to create a link to another file in the same BFS.
2. The file being linked must exist.
3. Permissions and ownership of the link will be based on that of the file to which the link refers.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes* or enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS1311E	Object already exists: *pathname*	28
DMS2113E	Object does not exist: *pathname*	28
DMS2115E	Objects are on different file systems	88
DMS2126E	You may not link to a directory	88

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM CREATE SYMLINK

```
►►── OPENVm ──── CREate ──── SYMlink ──── pathname1 ──── pathname2 ──►◄
```
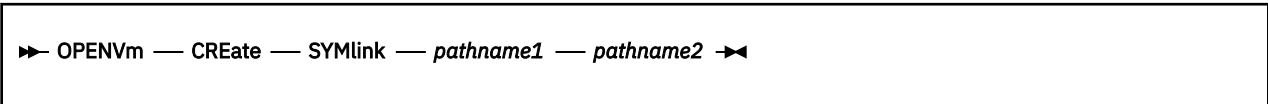
Authorization

General User

Purpose

Use OPENVM CREATE SYMLINK to create a byte file system (BFS) path name to be used to reference an object residing in a different BFS. This is known as a symbolic link. When creating a symbolic link, the object for which you are creating a link need not exist.

Operands

pathname1
is the BFS path name of the file to which the symbolic link is to be created. For a description of the different forms of the BFS path name, see "Understanding Byte File System (BFS) Path Name Syntax" on page 368.

pathname2
is the new BFS path name being created to refer to the file.

Usage Notes

1. The file for which the link is being created may be deleted without affecting the existence of the symbolic link.

2. File permissions, user IDs (UIDs), or group IDs (GIDs) associated with symbolic links entries are not used. Authorization is based on the permissions or file authorizations of the associated files. However, you need read permission to the directory containing the symbolic link to access a file through its symbolic link.

3. You may create a symbolic link that in turn references another symbolic link. However, a maximum of eight levels of nesting are allowed.

4. When path names refer to files in NFS-mounted file systems, you must meet the authorization requirements imposed by the remote NFS servers.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS1311E	Object already exists: *pathname*	28

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.

Reason	Location
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM DEBUG

```
                                    ┌──────────────────────┐
                                    │  ┌──── MSG ─────┐     │
>>── OPENVm ── DEBUG ──┬─┴─┬──────────────┬─┴── 1 ──┬─┤  ┌── WRAPsize ── 100 ──┐
                       │   ├── NOMSG ──────┤         (──┤                       ──>
                       │   ├── ALL ────────┤            └── WRAPsize ── n ──────┘
                       │   ├── NOALL ──────┤
                       │   ├── DUMp ───────┤
                       │   ├── NODUMp ─────┤
                       │   ├── FILeio ─────┤
                       │   ├── NOFILeio ───┤
                       │   ├── FLOw ───────┤
                       │   ├── NOFLOw ─────┤
                       │   ├── MOUnt ──────┤
                       │   ├── NOMOUnt ────┤
                       │   ├── NFSRequest ─┤
                       │   ├── NONFSRequest┤
                       │   ├── OTHer ──────┤
                       │   ├── NOOTHer ────┤
                       │   ├── RPCBuffers ─┤
                       │   ├── NORPCBuffers┤
                       │   ├── RPCLibrary ─┤
                       │   └── NORPCLibrary┘

        ┌─────┐
   ──┬──┴──)──┴──┬──><
```

Notes:

^{1} If a keyword or a keyword-value pair is specified more than once, the last one is used.

Authorization

General User

Purpose

Use the OPENVM DEBUG command for further problem determination of OPENVM command error messages and/or tracing for NFS and BFS Client events.

This command is to be used as directed by the systems administrator or support group that services your z/VM installation.

Operands

One or more of the following keyword parameters may be specified:

MSG

enables tracing for BFS errors. In the case of the NFS client, BFS errors are traced.

It allows the secondary error message DMS2134E to be displayed in addition to the primary error message from the OPENVM command that was entered within your virtual machine. Message DMS2134E will display the return and reason codes, and it will include the routine name that encountered the error. The MSG parameter also allows CMS Pipelines messages to be issued. It is the default for the OPENVM DEBUG command if no other parameters are specified. If other parameters are specified, MSG must also be specified to take effect.

NOMSG

causes the secondary error message DMS2134E and CMS Pipelines messages to be suppressed.

ALL

enables tracing for all NFS trace events; this indirectly turns on all tracing except DUMP, MSG, and RPCBUFFERS.

NOALL

disables tracing for all NFS trace events excluding DUMP, MSG, and RPCBUFFERS.

DUMp

enables dumping on certain error paths.

NODUMp

disables dump.

FILeio

enables tracing for file I/O processing.

NOFILeio

disables tracing for file I/O processing.

FLOw

enables tracing for entering/exiting NFS functions.

The process ids and thread ids are also shown.

NOFLOw

disables tracing for entering/exiting NFS functions.

MOUnt

enables tracing for mount requests.

NOMOUnt

disables tracing for mount requests.

NFSRequest

enables tracing for requests from the NFS client and shows the responses from the NFS server (local or remote host).

The NFS servers must support the Sun NFS V2 and/or V3 protocols. These NFS protocols are described in RFCs 1094 and 1813, respectively.

NONFSRequest

disables tracing for requests/responses between the NFS client and NFS server.

OTHer

enables tracing for initialization, termination, and anything that does not fit under the categories of dump, fileio, flow, mount, nfsrequest, rpcbuffers, or rpclibrary.

NOOTHer

disables tracing for other.

RPCBuffers

enables tracing for input and output buffers of the RPC requests.

NORPCBuffers

disables tracing for the RPC buffers.

RPCLibrary

enables tracing of the RPC Runtime library, VMRPC.

NORPCLibrary
> disables tracing for the RPC Runtime library, VMRPC.

Options

WRApsize n
> specifies how many trace events to retain in the trace table.*n* is a positive integer value greater than 0. When the wrapsize *n* is exceeded, the oldest trace event is discarded to make room for the newest arrival. The valid values are 1 through 99999999.
>
> If wrapsize is not specified, the default is 100.

Usage Notes

1. If you specify OPENVM DEBUG MSG, message DMS2134E can be displayed as a secondary message for the entire CMS session until you IPL CMS or turn the message off by entering OPENVM DEBUG NOMSG.

2. During initialization (IPL), the following defaults are in effect:
   - NOMSG
   - WRAPSIZE 100

   These values may be overridden when a user invokes the OPENVM DEBUG command with other specified keywords.

3. The OPENVM DEBUG keywords are processed in the order specified. For example, if the first keyword is MSG and is then followed by NOMSG, the first MSG is nullified. If a keyword from a keyword-pair is specified more than once, the last one specified takes effect.

4. Previous settings set during a session are respected. If OPENVM DEBUG MSG was issued, followed by OPENVM DEBUG FLOW, then both MSG and FLOW will be in effect. Note that IPLing will clear all settings and reset initial default settings.

5. Use OPENVM QUERY DEBUG to view information on the current trace settings. See "OPENVM QUERY DEBUG" on page 431 for more information.

6. With OPENVM DEBUG in effect, if an OPENVM command is issued from within a CMS Pipeline, the output from OPENVM DEBUG might be included in the output stream of the Pipeline instead of displayed on the console.

Example for OPENVM DEBUG MSG and OPENVM DEBUG NOMSG

If you enter:

```
openvm debug msg
```

Then you try to create a directory that already exists:

```
openvm create directory /test
```

You will receive these error messages:

```
DMSOVC1131E Directory '/test' already exists
DMSOVC2134E Return code 117 and reason code 56 (X'38')
           given on call to BPX1MKD
Ready(00028);
```

Secondary message DMS2134E is displayed, providing the return and reason codes from the BPX1MKD routine. For more information on these codes, see *z/VM: OpenExtensions Callable Services Reference*.

If you enter:

```
openvm debug nomsg
```

This will turn off the secondary error message. Therefore, when you try again to create a directory that already exists, only the primary error message and corresponding CMS return code is displayed:

```
openvm create directory /test
DMSOVC1131E Directory '/test' already exists
Ready(00028);
```

Example for NFS Tracing

If you enter:

```
openvm debug all
```

this will turn on tracing for FILEIO, FLOW, MOUNT, NFSREQUEST, OTHER, and RPCLIBRARY.

Here is a sample of the type of trace messages you will see if you now issue an OPENVM MOUNT command to mount an NFS file space:

```
14:15:17.852757 DTCCMAIN.24 MAIN_RTN: Hostname is GDLVM7

14:15:17.855736 DTCCMAIN.26 MAIN_RTN: Got socket 3

14:15:17.856867 DTCCMAIN.F0 LFSQmsg : ---- Entry

14:15:17.860201 DTCCMAIN.F4 LFSQmsg : LFS Q open OK

14:15:17.862589 DTCCMAIN.F8 LFSQmsg : Init Q send OK.

14:15:17.863675 DTCCMAIN.F9 LFSQmsg : gl_NFSCinit set.
```

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS3951E	Invalid integer *n* for WRAPSIZE option	24

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM ERASE

```
►►— OPENVm —— ERASE —— pathname —►◄
```

Authorization

General User; Byte file system (BFS) permission checking applies to this command.

Purpose

Use the OPENVM ERASE command to erase a byte file system (BFS) object.

Operands

pathname
> is the BFS path name of the link, directory, or other BFS object to be deleted. For a description of the different forms of the BFS path name, see "Understanding Byte File System (BFS) Path Name Syntax" on page 368.

Usage Notes

1. File data will be deleted when the last link to a file is erased. However, if another process has the file open when the last link is erased, the file is not deleted until the last process closes it.

2. When a symbolic or external link is deleted, the associated object is not modified.

3. Directories must be empty in order to be deleted.

4. When *pathname* refers to an object in an NFS-mounted file system, you must meet the authorization requirements imposed by the remote NFS server.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS1162E	Directory is not empty: *pathname*	40
DMS1199E	You cannot erase a top directory	88
DMS2121E	Operation may not be performed on {the file system root|. or ..}	88

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM FORMAT

```
►►─ OPENVm ── FORmat ─┬─────────────────────┬─ ( ─┬─ NOBuffer ─┬─────┬─ ►◄
                      └── number_entries ───┘     └── BUFfer ──┘  └─ ) ─┘
```
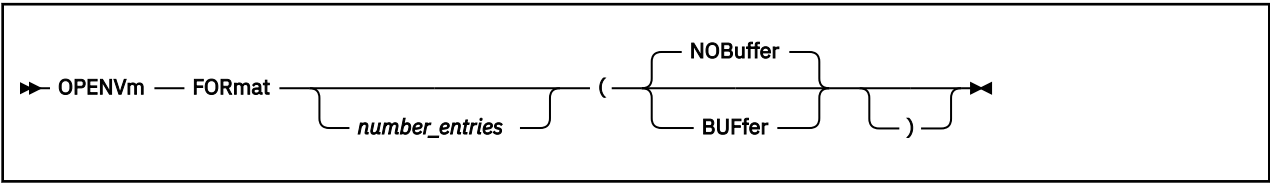
Authorization

General User

Purpose

Use the OPENVM FORMAT to display the trace table created by NFS Client events on the console.

The trace table contains information about the flow of requests between the NFS client and server.

This command is to be used as directed by the systems administrator or support group that services your z/VM installation.

Operands

number_entries
    indicates the number of entries to be displayed on the console.

    If the number of entries are not specified, all the entries in the trace table will be displayed.

    If you specify a number, the number specified will be displayed even if there are more entries in the table.

    The maximum number of entries that can be retained in the trace table are defined by the wrapsize. See the usage notes for more information.

Options

BUFfer
    writes buffer information to the console.

NOBuffer
    does not write buffer information to the console. This is the default.

Usage Notes

1. Use the OPENVM QUERY DEBUG command to view information on the current wrapsize setting. See the "OPENVM QUERY DEBUG" on page 431 for more information.
2. Trace table entries are written out from the oldest to the newest.

Messages and Return Codes

The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See the *z/VM: CMS Commands and Utilities Reference*.

OPENVM GETBFS

```
▶▶── OPENVm ── GETbfs ── pathname1 ──┤fileid├──────────────────────────────────▶◀
                                                ┌──────────────┐
                                          (──1──┤   Options    ├──
                                                └──────────────┘
                                                    └────)────┘
```

fileid

```
                              ┌───── A ─────┐
▶▶── fn ── ft ──┬─────────────┼─────────────┼──▶◀
               │             ├───── fm ─────┤
               │             └──── dirid ───┘
               └──── pathname2 ──────────────┘
```

Options

```
   ┌── NEWFile ──┐          ┌── BFSLine ── NL ──2──┐        ┌── NEWDate ──┐
▶▶─┼─────────────┼──────────┼──────────────────────┼────────┼─────────────┼──▶
   └── REPlace ──┘          └── BFSLine ──┬── lrecl ──┐     └── OLDDate ──┘
                                          ├── CRLF ───┤
                                          ├── CRNL ───┤
                                          ├── /string/ ┤
                                          ├── /hexstring/ ┤
                                          └── NONE ───┘
                            └──────── MODule ─────────┘

          ┌──────── NOTRAnslate ────────┐
▶─────────┼─────────────────────────────┼──▶◀
          └── TRAnslate ── codepage1 ── TO ── codepage2 ──┘
```

Notes:

  [1] You can enter Options in any order between the parentheses.
  [2] BFSLINE NL is the default unless the *ft* of fileid is MODULE, in which case MODULE is the default.
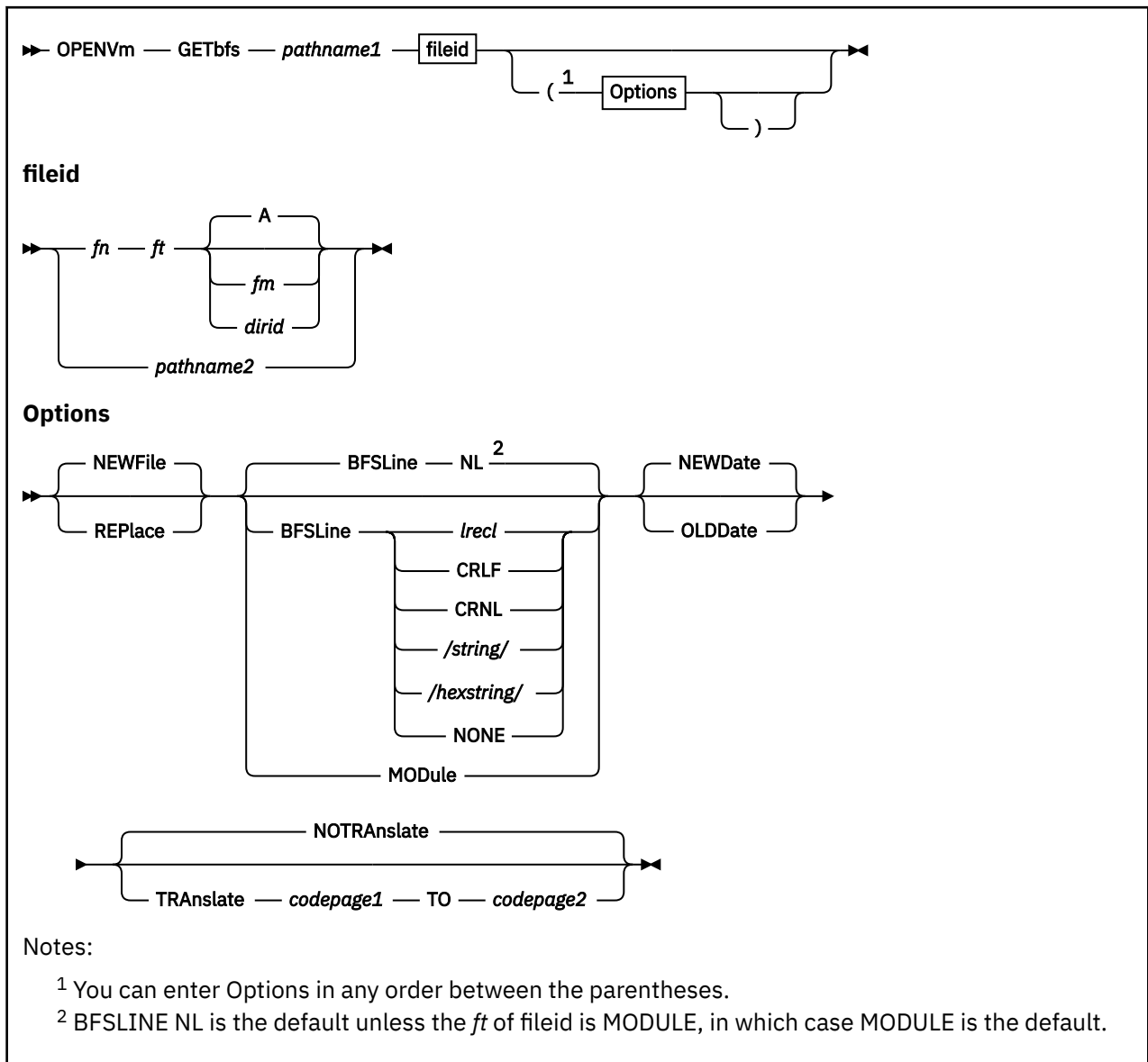
Authorization

General User;

Byte file system (BFS) permission checking applies to BFS objects used by this command.

Purpose

The OPENVM GETBFS command copies a byte file system (BFS) regular file into another BFS regular file, an SFS directory, or onto a CMS minidisk.

Operands

pathname1
>   is the BFS path name of the file to be copied. For a description of the different forms of the BFS path name, see "Understanding Byte File System (BFS) Path Name Syntax" on page 368.

fn ft **A**
fn ft fm
fn ft dirid
 is the CMS file to be created or replaced.

pathname2
 is the BFS path name of the file that is to be created or replaced.

Options

NEWFile
 checks that an object with the same file ID as the output file does not already exist. If the object does exist, an error message is displayed and the GETBFS command terminates. This option is the default so that an existing file is not inadvertently destroyed.

REPlace
 causes the output file to replace an existing file with the same file ID.

BFSLine
 Use the BFSLINE option to tell CMS how to translate a BFS byte stream into records. This option is ignored (has no effect) if the target is a BFS file. BFSLINE NL is the default unless the file type (*ft*) part of the **fileid** is MODULE.

 The BFSLINE option also determines the record format (RECFM) of the file if it is being copied to a SFS or a minidisk.

 If you specify anything other than BFSLINE *lrecl,* you can define an end-of-line character or characters for use in interpreting lines in a BFS file. When a file is read, everything up to the end-of-line character is interpreted as a line and presented as a 'record'. The end-of-line character is removed from all lines in the BFS file.

 lrecl
  indicates the file should be treated as a fixed file, with no interpretation of records based on end-of-line characters. When BFSLINE *lrecl* is in effect, the file is presented as a fixed record format (RECFM F) file with a logical record length (LRECL) equal to the *lrecl* value. No end-of-line characters are removed from the file when it is read.

  The last record will be padded with blanks if *lrecl* is greater than 1 and the last record does not completely fill the last logical record.

 NL
  indicates that the new line character (X'15') should be used to delineate lines when reading a BFS file.

 CRLF
  indicates that carriage return/line feed (X'0D25') should be used to delineate lines when reading a BFS file.

 CRNL
  indicates that carriage return/new line (X'0D15') should be used to delineate lines when reading a BFS file.

 /string/
  allows the user to specify a 1-2 character string that is used to delineate lines when reading a BFS file. Blanks may not be included in *string*. X' or x' are not valid character strings.

 /hexstring/
  specifies a hexadecimal string of 2 or 4 characters that defines the value to be used for BFSLINE. The *hexstring* must be in the format X'nnnn' or X'nn'. You must not specify any spaces in the string, and there must be 2 or 4 hexadecimal characters in the string.

 NONE
  indicates that the file should be treated as a variable file with no interpretation of records based on end-of-line characters. When BFSLINE NONE is in effect, the file is presented as a variable record format (RECFM V) file. Except for the last record in the file, the logical record length is 65535, the maximum record length for a CMS variable file.

MODule
Specifies that the BFS file is an executable file, such as a file created using c89, cxx, or OPENVM PUTBFS with the MODULE option. The MODULE option must be specified on OPENVM PUTBFS (or in effect by default) if the BFS file is executable in order for the resulting CMS file to be in the format of a file created by the GENMOD or BIND command.

NEWDate
uses the current date and time for the date and time of the new file. This is the default.

OLDDate
uses the time of last data modification of the source file as the:

- Update date and time of the target CMS file, or

- The time of last data modification of the target BFS file.

If the target file is a BFS file and you attempt to use the OLDDATE option, but you are not the owner of the target file or a superuser, a warning message will be issued and the current time will be used.

NOTRAnslate
Indicates that no code page translation should occur.

TRAnslate
Indicates that the characters in the file should be translated as part of the OPENVM GETBFS operation. This option is ignored if the MODULE option is specified.

  codepage1
  Specifies the code page for the source file

  TO *codepage2*
  Specifies the code page for the target file.

Any code page is allowed that is supported by the CMS Pipelines XLATE stage. See the *z/VM: CMS Pipelines User's Guide and Reference*.

If an end-of-line character is specified, it is not affected by code page translation. That is, code page translation is done after the byte stream is changed into records.

Usage Notes

1. When the target file is a new BFS file and the source file has at least one of its execute permissions on, the permissions for the new file are set to 'rwx r-x r-x'.

   When the target file is a new BFS file and the source file does not have any execute permissions on, the permissions are set to 'rw- r-- r--'.

   Setting the mask can turn off additional permissions. See "OPENVM SET MASK" on page 453 for more information. Use OPENVM PERMIT to change permissions after the file is created.

2. When a new BFS file is created, the owning UID established is the effective UID of the process that issued the request. The group name is the GID of the parent directory. Use OPENVM OWNER to change the defaults.

3. If the source or target of an OPENVM GETBFS is a BFS object, but it is not a BFS regular file, the command will fail.

4. Use the */string/* or */hexstring/* option when you want to specify a different end-of-line character than those specified above. For example, if your file uses X'0D' to indicate end-of-line, specify the BFSLINE /X'0D'/.

   When specifying a BFSLINE value for use on files containing DBCS characters, be careful to use a value that will not conflict with DBCS characters. The hexadecimal code for a DBCS character must be X'00', X'40', or in the range of X'41' to X'FE'.

5. If you are copying a BFS file to a CMS record file (SFS or minidisk), the records in your file may not exceed the maximum CMS record length. For fixed record format (recfm) files created when the **BFSLINE** *lrecl* option is used, the maximum record length is 2^{31}-1. For variable recfm files created when any other BFSLINE option is used, the maximum record length is 65535.

6. If you are copying a BFS file to a CMS record file (SFS or minidisk) and you are not using **BFSLINE** *lrecl*, an end-of-line character that is the first character in the file or two end-of-line characters in a row will result in a record of a single blank.

7. You can use the OPENVM GETBFS command to write an SFS file if you have the proper SFS authorization. When the file is to reside in a FILECONTROL directory, you can write the file using a file mode letter even if you have the directory accessed in read-only status. If you wish the OPENVM GETBFS command to respect the read-only status, use the SET RORESPECT ON command (see *z/VM: CMS Commands and Utilities Reference*). When the file is to reside in a DIRCONTROL directory, however, you must access the directory in read/write status.

8. OPENVM GETBFS to an existing BFS file will not change the permissions and ownership for the existing file.

9. The NL and CRLF mnemonics translate into values defined by code page IBM-1047.

10. If *fileid* specifies a CMS record file (sfs or minidisk file), the file ID will be converted to upper case during OPENVM GETBFS processing. However, if *fileid* specifies a BFS path name, a mixed case file ID will be respected and will not be converted to uppercase.

11. When path names refer to files in NFS-mounted file systems, you must meet the authorization requirements imposed by the remote NFS servers.

12. Use the TRANSLATE option carefully if the source or target files are in an NFS-mounted file system. The NFS mount allows you to specify whether file data is translated. Do not tell CMS to translate data a second time using the TRANSLATE option on OPENVM GETBFS.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS024E	File already exists; specify REPLACE option for: {*pathname2*|*fn ft fm*| *fn ft dirid*}	28
DMS037E	Filemode *fm* is accessed as read/only	36
DMS069E	Filemode *fm* not accessed	36
DMS173E	Empty output file *fn ft fm* not created	40
DMS618E	NUCEXT failed, return code *rc*	104
DMS639E	Error in {PIPE|DMSCCE} routine, return code was *rc*	28
DMS1137E	Object is locked; deadlock detected	70
DMS2041W	You are not permitted to use the OLDDATE option	4
DMS2109E	Object is a directory: *pathname*	40
DMS2125E	Path name ends with a slash: *pathname*	40
DMS2128E	Lines exceed the CMS maximum record length for: *pathname1*	40
DMS2538E	File is not in MODULE format	32

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in the Shared File System	See "SFS and CRR Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using a file	See "File Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM LISTFILE

Notes:

[1] You can enter Options in any order between the parentheses.

[2] OBJECT and NAMES are conflicting options if the path name is '/'.
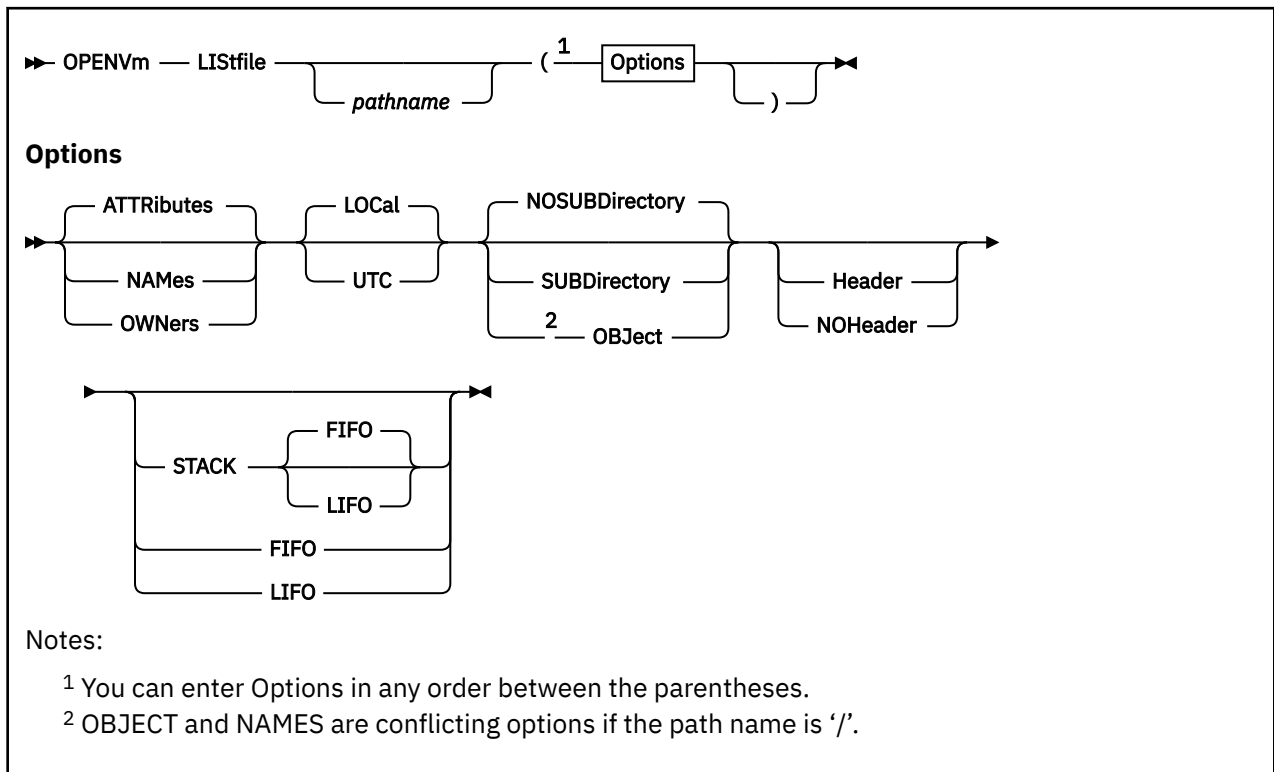
Authorization

General User

Purpose

Use the OPENVM LISTFILE command to display information about files and other objects residing in the byte file system (BFS) directory. If you do not specify any operands, OPENVM LISTFILE will list all objects that are within the current working directory.

Operands

pathname

specifies the BFS path name of the BFS directory with contents you wish to see displayed. For a description of the ways in which path name may be specified, see "Understanding Byte File System (BFS) Path Name Syntax" on page 368.

Options

ATTRibutes

lists the following information about the specified files:

- Object type:

  The object type will be displayed in OPENVM LISTFILE as a code. (See "Examples" on page 401 for a description of object type codes and their meanings.)

- Date and time that file data was last updated

- Number of links to this file

- File size in bytes

- BFS path name component.

NAMes

When the NAMes option is specified, the following information is displayed:

- CMS file name will be an 8 character system generated value.
- CMS file type will be an 8 character system generated value.
- BFS ID
- Object type
- BFS path name will be in the range of 1 to 255 bytes long. The BFS path name displayed here will not include the directory name used on invocation to OPENVM LISTFILE; it will represent a single BFS path name component.

The CMS file name and file type together represent a system generated value that is unique within a BFS to each file. Note that all links to a file will have the same CMS file name and file type. The CMS file name and file type may be used as input to some CMS administrative commands accessing BFS objects. The NAMes option is not valid when the OBJECT option is specified and the root directory (/) is the path name specified.

OWNers

This option is useful if you need to determine the user ID (UID), group ID (GID), and permissions associated with objects in the BFS. When the **OWNers** option is specified, the following information is displayed:

- User ID of file owner
- Group name of files owner
- Permissions (rwx rwx rwx)
- Object type
- BFS path name.

LOCal

indicates that the Update-Tm field contained within the ATTRIBUTES option screen will be displayed in Local Time. This is the default.

UTC

Universal Time, Coordinated (GMT - Greenwich mean time); indicates that the Update-Tm field contained within the ATTRIBUTES option screen will be displayed in UTC. The Update-Dt (date) field may be different from local time if UTC is specified.

NOSUBDirectory

indicates that file system objects contained within BFS subdirectories within the specified directory will not be displayed. This is the default when the path name specified on the LISTFILE command is a BFS directory.

SUBDirectory

indicates that the contents of subdirectories within the specified BFS directory will be displayed.

OBJect

indicates that the file system object specified in the path name parameter will be displayed. If the object specified is not a directory, OBJect is the default.

Output Format Options

Header

includes column headings in the listing. HEADER is the default unless you specify STACK, FIFO, or LIFO.

NOHeader

does not include column headings in the list. NOHEADER is the default if you specify STACK, FIFO, or LIFO.

Output Disposition Options

STACK

specifies that the information should be placed in the program stack (for use by an exec or other program) instead of being displayed at the terminal. The information is stacked either FIFO (first in first out) or LIFO (last in first out). The default is STACK FIFO.

Note: An entry will be truncated if the combined length of its fields are greater than 255 characters. This is true for all CMS stack related options: namely STACK, LIFO and FIFO.

You can use CMS PIPELINES to manipulate the output of OPENVM LISTFILE (without any of the stack options) to handle long path name components without potential truncation.

FIFO

specifies that the information should be placed in the program stack rather than displayed at the terminal. The information is stacked FIFO. The options STACK, STACK FIFO, and FIFO are all equivalent.

LIFO

specifies that the information should be placed in the program stack rather than displayed at the terminal. The information is stacked LIFO. This option is equivalent to STACK LIFO.

Usage Notes

1. If you enter the OPENVM LISTFILE command with no operands, a list of all objects in your current working directory is displayed at the terminal.

2. You may use the output of OPENVM LISTFILE as input to a CMS PIPELINE. For example:

```
PIPE CMS OPENVM LISTFILE '/mydir' (noheader | > MYBYTEFS FILES A
```

will create a list of the files in your BFS 'mydir' directory and place the information listed in the ATTRibutes screen into a file named MYBYTEFS FILES on your 'A' disk. The BFS path names will not be subject to truncation.

3. If you want to enter OPENVM LISTFILE from an exec program, you should precede it with the EXEC command; that is, specify:

```
exec OPENVM listfile
```

4. Path name components will be displayed with a single quotation mark around the name. If there are any quotation marks within the name, they will be translated into two quotation marks. For example, if you had two files named:

```
Aladdin
Aladdin's lamp
```

The display of the object name would look like this:

```
'Aladdin'
'Aladdin''s Lamp'
```

5. BFS path names will be displayed in mixed case.

6. File attributes for all links to the same file will be displayed as identical values.

7. When OPENVM LISTFILE is specified with the OWNERS option, data is returned about a user ID name. If the same user ID number is defined for more than one z/VM user in the directory, data is returned about one of the user ID names, but which one is unpredictable. If you need to have a unique user ID name returned, you need to have a unique UID number defined for each z/VM user.

8. You must have search permission on all components of the BFS path name and read permission to the directory specified in OPENVM LISTFILE. If the **SUBDirectory** option is specified, all subdirectories to which you have read permission will be displayed.

9. BFSs that are mounted in your hierarchy are displayed by OPENVM LISTFILE in their logical position.

10. The STACK, LIFO, and FIFO options cause the requested information to be placed in the program stack. If this information is to be stacked, the options relating to the display format (HEADER or NOHEADER) should not be specified.

11. If the object is a Mount External Link, specifying the name of the external link without a closing slash (/) gets information only about the link itself. To get information about the linked object, you must enter the name of the external link with a closing slash or specify its fully qualified path name.

12. When *pathname* refers to an object in an NFS-mounted file system, you must meet the authorization requirements imposed by the remote NFS server.

Examples

Unless the STACK, LIFO, or FIFO option is specified, the requested information is displayed at the terminal. Entering OPENVm LISTFILE with the ATTRibutes option displays the following information:

```
Directory ='bfsdname'
Update-Dt  Update-Tm  Type  Links  Bytes  Pathname Component
mm/dd/yyyy hh:mm:ss     t      l       b     pname
   .          .         .      .       .      .
   .          .         .      .       .      .
```

where:

bfsdname
is the name of the BFS directory name specified on the OPENVM LISTFILE invocation.

mm/dd/yyyy
is the date (month/day/year) the file's data was last modified.

hh:mm:ss
is the time the file's data was last modified.

b

is the number of bytes contained in the file. A dash will be in this field unless the object displayed is a BFS regular file.

t

refers to the object type. This will be one of the following:

B
Block special file

C
Character special file

D
Directory

E
External Link

F
BFS regular file

F*
BFS regular file that is in DFSMS/VM migrated status

L
Symbolic Link

P
FIFO

S
Socket.

When the object is an external link, there will be a secondary code displayed indicating the type of external link associated with the file. It may be one of the following:

1

CMSEXEC External Link

2

CMSDATA External Link

3

MOUNT External Link

nnn

CODE external link (where *nnn* is the user defined code ranging from 100 to 200).

When the object is a block or character special file, there will be a secondary code displayed indicating the device major number associated with the file. It may be one of the following:

3

/dev/tty

4

/dev/null

l

is the number of links that are associated with this file. A dash will be displayed in this field unless the object displayed is a BFS regular file.

pname

is the BFS path name component following the BFS directory name specified with OPENVM LISTFILE. This will be 1-1023 characters in length. If the displayed object resides in the directory specified with OPENVM LISTFILE, only the last object name component will be displayed. If the SUBDirectory option has been specified on OPENVM LISTFILE and the displayed object resides in a BFS subdirectory of the specified directory, all subdirectory names that follow under the specified directory name will be displayed.

For example, your environment is set up as shown in .

```
Mount point = /
Type   Stat   Mounted
BFS    R/W    '/../VMBFS:FP1:BFSNAME/geographic/data'
```
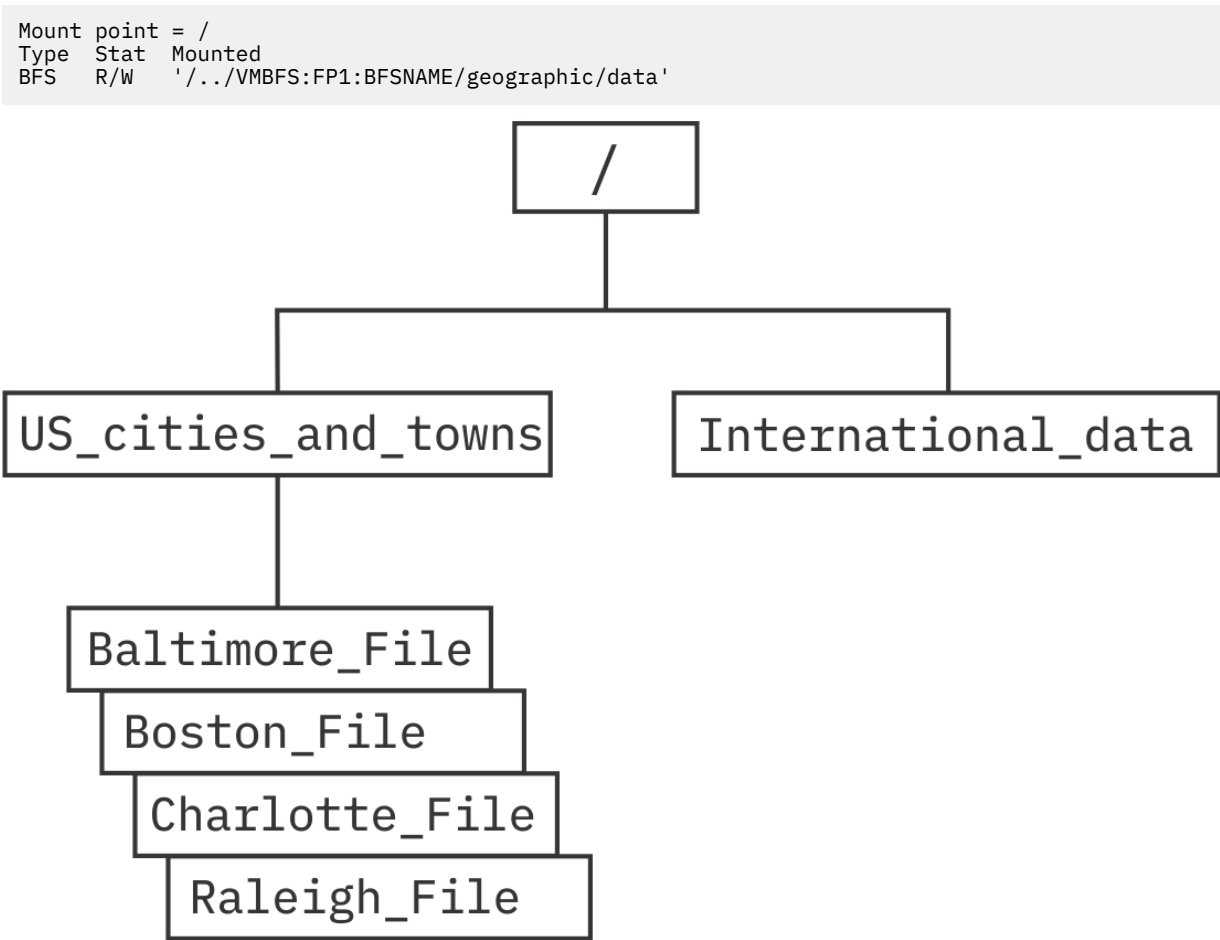

Figure 1. BFS environment with the OPENVM LISTFILE SUBDirectory option specified

If you entered 'OPENVM LISTFILE / (SUBDIR' you might see the following output:

```
Directory = '/'
Update-Dt   Update-Tm Type Links  Bytes  Path name component
02/02/1994  12:22:55   D    -       -     'US_cities_and_towns'
02/15/1994  14:32:35   F    1     20956   'US_cities_and_towns/Baltimore_File'
02/15/1994  14:32:35   F    1      2346   'US_cities_and_towns/Boston_File'
02/15/1994  14:32:35   F    2     10956   'US_cities_and_towns/Charlotte_File'
02/15/1994  14:32:35   F    1     34556   'US_cities_and_towns/Raleigh_File'
02/02/1994  12:23:55   D    -       -     'International_data'
```

Note that the BFS path name is enclosed in single quotation marks. If a quotation mark appears within the name, it will appear as two quotation marks.

If the NAMes option is specified, the information displayed is:

```
Directory ='bfsdname'
Filename Filetype Byte File System    Type   Path name component
 fn       ft      fsid                t      pname
 8        0       FP1:BFSNAME.        D      'US_cities_and_towns'
 .        .       .                   .      .
```

where:

bfsdname
  is the name of the BFS directory specified on the OPENVM LISTFILE invocation.

fn
  is an eight character system generated value that along with *ft* uniquely identifies a file within a BFS. This may be used as the file name input to some CMS administrative commands. This will be displayed as a dash when the object is a named pipe or socket. This file name and file type are only useful when the object is a BFS regular file.

ft
is an eight character system generated value that along with *fn* uniquely identifies a file within a BFS. This may be used as a file type input to CMS commands. This will be displayed as a dash when the object is a named pipe or socket.

fsid
is the name of the byte file system, or NFS.

If the file is in an NFS mounted file system, NFS is displayed. Use the OPENVM QUERY MOUNT command to display information about NFS mounted file systems.

If the file is in a byte file system, this is a character string in the form:

```
►►─────┬─────────────┬───┬──────────────┬──.─►◄
       └─ filepoolid ─:─┘   └─ filespaceid ─┘
```
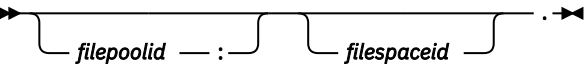
filepoolid
is the name of the file pool. If not specified, the default file pool that you set with the SET FILEPOOL command is used (the system does not supply a default). You (or the system administrator) can also set a default file pool in your user CP directory so you do not have to enter the SET FILEPOOL command each time you log on. The file pool name can be up to 8 characters long. The first character must be alphabetic, but the remaining characters can be alphabetic or numeric. Lowercase is converted to uppercase.

: (colon)
is a separator that must be specified following the *filepoolid* when it is part of a byte file system name.

filespaceid
is the name of the file space. It defaults first to the file space ID set with the SET FILESPACE command, and then to the user ID calling the routine. The file space ID can be up to 8 characters long.

. (period)
is a separator that must be specified following the *filespaceid*. If it is specified without the *filepoolid* and *filespaceid* parameters, it means the top directory in the default file space in the default file pool.

t
refers to the object type. This may be any of the types described with the ATTRibutes option.

pname
is the BFS path name of the displayed object.

One entry is displayed for each BFS object listed.

If the OWNers option is specified, the information displayed is:

```
Directory ='bfsdname'
User ID   Group Name   Permissions Type  Path name component
 uid         gid       rwx rwx rwx  t     pname
user1000  CMSUSRS      rwx r-x r-x  D     'new_directory'
 .           .           .     .     .
 .           .           .     .     .
```

where:

bfsdname
is the name of the BFS directory specified on the OPENVM LISTFILE invocation.

uid
is a 1-8 character name that represents the effective UID for the owner of the file. For NFS files, the UID will be displayed as a number. For BFS files, if the UID number cannot be mapped to a user ID name, then a dash is returned.

gid
>   is a 1-8 character name that represents the effective GID for the group of the file. For NFS files, the GID will be displayed as a number. For BFS files, if the GID number cannot be mapped to a group name, then a dash is returned.

rwx rwx rwx
>   contains information used to determine if a process has read, write, or execute/search permissions to a file. This is displayed in three groups: owner, group, and public. Each group consists of permissions for read, write, and execute access to that file. This is displayed in the following format:

```
   rwx rwx rwx
```

>   The first string represents file permissions belonging to owner. The second represents file permissions belonging to the group. The third represents file permissions belonging to all others (public). These are positional fields. The following characters are used:

>   **r**
>>   Indicates read permission

>   **w**
>>   Indicates write permission

>   **x**
>>   Indicates execute permission. If the displayed object is a directory, an 'x' indicates search permission.

>   **s or S**
>>   An executable file can have an additional attribute, which is displayed in the execute position (x). This permission setting is used to allow a program temporary access to files that are not normally accessible to other users. An s or S can appear in the execute position; this permission bit sets the effective user ID or group ID of the user process executing a program to that of the file whenever the file is run.

>>   **s**
>>>   In the owner permissions section, this indicates that both the set-user-ID bit and execute (search) permission are set.

>>>   In the group permissions section, this indicates that both the set-group-ID bit is set and execute (search) permission are set.

>>   **S**
>>>   In the owner permission section, this indicates that the set-user-ID bit is set, but the execute (search) permission is not.

>>>   In the group permissions section, this indicates that the set-group-ID bit is set, but the execute (search) permission is not.

>   **-**
>>   Indicates that the user class (owner, group, or public) did not have the permission signified by this position in the string to this particular file. A '-' may appear in any position of the string.

>   Symbolic links do not have permissions associated with them. Thus, permissions for symbolic links will appear as:

```
                --- --- ---
```

t
>   refers to the object type. This may be any of the types described with the ATTRibutes option.

pname
>   is the object name component of the BFS path name. This will be 1-1023 characters in length. This will be enclosed in single quotation marks. If a quotation mark appears within the name, it will appear as two quotation marks.

One entry is displayed for each BFS object listed.

Messages and Return Codes

For information on a specific error message, refer to *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:
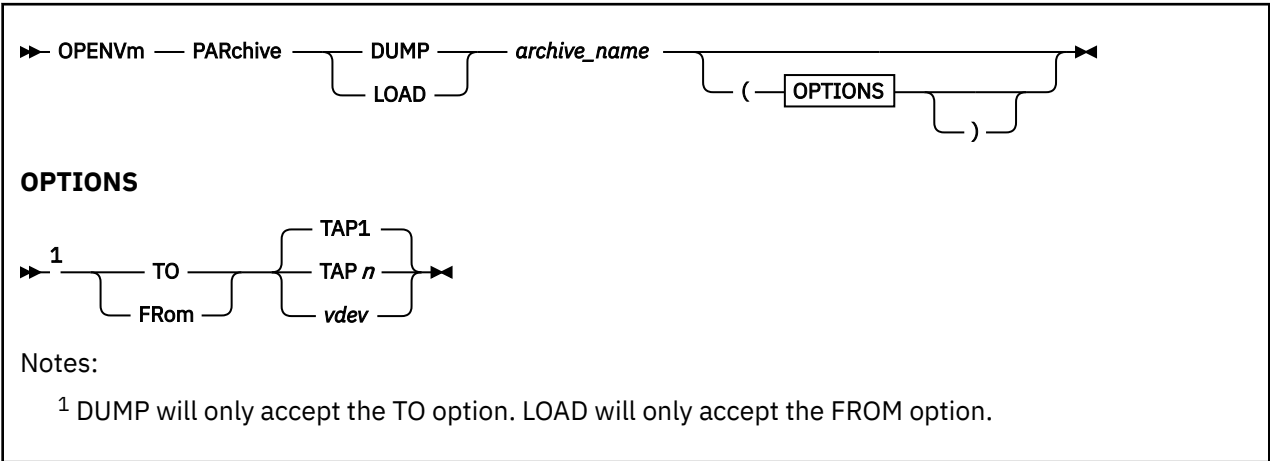
```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS003E	Invalid option: NAMES	24
DMS065E	*option* option specified twice	24
DMS066E	*option1* and *option2* are conflicting options	24
DMS1187E	Too many subdirectory levels in *pathname*	0
DMS1229E	*pathname* is empty	28

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM MOUNT

```
▶▶── OPENVm ── MOUnt ── pathname1 ─────────── / ───────────▶
                               └─ pathname2 ─┘

      ┌──────── WRIte ────┐ ┌── STArt ──┐
   ─┬─┤                   ├─┤           ├─ Local NFS Options ─┬─▶◀
    │ └──────── REAd ─────┘ └── NOStart ┘                     │
    └─( ──────────────────────────────────────────────── ) ──┘
```

Local NFS Options

```
      ┌── NETrc ──┐
▶▶──┬─┤           ├─┬──────────────────────────────────────────────▶
    │ └ NONETrc ──┘ │ ┌─ USErid ── username ─┐ ┌ PASsword ── password ┐
    └───────────────┴─┤                      ├─┤                      ├─
                      └─ ANOnymous ──────────┘ └──────────────────────┘

    ┌── PROmpt ──┐  ┌─ TRAnslate ── POSIX ──────────┐  ┌─ LIST ──┐
▶──┬┤            ├──┤                               ├──┤         ├──▶
   │└─ NOPROmpt ─┘  ├─ TRAnslate ── trans_tablename ┤  └─ NOLISt ┘
   │                └─ NOTRAnslate ─────────────────┘

    ┌─ ATTRCaching ── YES ─┐  ┌─ ATTRMax ── 60 ──────┐
▶──┬┤                      ├──┤                      ├──────────────▶
   │└─ ATTRCaching ── NO ──┘  └─ ATTRMax ── seconds ─┘

    ┌─ READAhead ── 1 ──────┐  ┌─ RETry ── 3 ──────┐  ┌─ TCP ─┐
▶──┬┤                       ├──┤                   ├──┤       ├──▶
   │└─ READAhead ── blocks ─┘  ├─ RETry ── retries ┤  └─ UDP ─┘
   │                           └─ RETry ── FORever ┘

    ┌─ TIMeout ── 7 ──────────┐  ┌─ VERsion ── 3 ─┐  ┌─ LOCalport ── ANY ─────┐
▶──┬┤                         ├──┤                ├──┤                        ├──▶◀
   │└─ TIMeout ── timeout_value┘  └─ VERsion ── 2 ─┘  └─ LOCalport ── portnum ─┘
```

Authorization

General User

Purpose

The OPENVM MOUNT command allows a byte file system (BFS) subdirectory tree, an entire BFS or a Network File System (NFS) to be logically included as a component of a BFS at any place in the hierarchy. In other words, it makes the files available for use in your virtual machine.

Use the OPENVM QUERY MOUNT command to display what is mounted in your hierarchy.

Operands

pathname1

is the fully qualified name of the BFS, BFS subdirectory name, or NFS file system to be mounted. Because you are identifying a new physical BFS to be included in your logical hierarchy, you must use the fully qualified form for the *pathname1* operand. That is, you would specify:

```
/../VMBFS:filepool:filespaceid/
```

or

```
/../VMBFS:filepool:filespaceid/subdirname
```

or

```
/../NFS:foreign_host/directory_name/
```

See "Understanding Byte File System (BFS) Path Name Syntax" on page 368 for a description of the different forms of the BFS path name, or see "Understanding Network File System (NFS) Path Name Syntax" on page 374 for a description of the NFS path name. Because you cannot mount a subdirectory in the file system where it resides, you must use the fully qualified form for the *pathname1* operand rather than using a shorter path name that picks up your root directory.

A mount of a NFS file system typically requires that the TCP/IP client minidisk TCPMAINT 592 be accessed so that system files, such as the NFS MODULE and TCP/IP translation tables, are available.

/

indicates that the BFS or subdirectory tree is to be mounted as the root directory. An NFS file system cannot be mounted as the root directory.

pathname2

is the location or the mount point where the new file system is to be mounted. The *pathname2* must identify a directory or a Mount External Link (MEL). Because you are identifying an object in the existing logical hierarchy, you must use the relative or absolute form of the path name; you cannot use the fully qualified form.

Options

WRIte

mounts the file system or directory in read/write mode. WRITE is the default.

REAd

mounts the file system or directory in read mode.

STArt

Byte file system is initialized and system resources are allocated. STArt is the default.

NOStart

The specified file system is mounted when the file system is initialized. This will happen implicitly when the file system is referenced at a later point. NOSTART has no effect when an NFS mount is being performed.

NETrc

Use information in the NETRC DATA file as an alternative to specifying *username* and *password*. For information on the NETRC DATA file, see *z/VM: TCP/IP User's Guide*.

NONETrc

Do not use information in the NETRC DATA file to supply *username* and *password*.

USErid *username*

provides a *username* for authentication at the remote host. The user ID and password should indicate how you want to be known at the remote host.

USERID has no effect unless the mount is being performed for an NFS file system.

Since USERID often represents a user ID on a non-VM system, it is not restricted to 8 characters like VM user IDs. Also, *username* is not uppercased.

The NETRC DATA file provides an alternative for specifying the *username* and *password* parameters. If these parameters are defined within this file for a specific foreign_host, you can omit them from an OPENVM MOUNT command issued for that host.

The NFS Client uses the Sun PC-NFS protocol to authenticate the user ID information at the remote host, and the remote host returns UID and GID values. Those values are passed to the NFS server on subsequent requests in the UNIX-style credentials.

If no username is provided and the ANONYMOUS option is not used, the UID and GID passed to the NFS server in the UNIX-style credentials are the values in effect for your VM user ID.

ANOnymous
indicates that you want the mount to be done anonymously. When you mount anonymously, CMS bypasses the use of the Sun PC-NFS protocol to authenticate the user ID information at the remote host. The UID and GID passed in the UNIX-style credentials to the NFS server are -2.

PASsword *password*
allows specification of a password on the command line.

Note: If *password* contains one or more blanks, the password must be specified using the NETRC DATA file

PROmpt
If *password* is not provided by the PASSWORD parameter or by NETRC DATA, prompt the user to enter the password on the user machine console. Display of the entered password is suppressed.

NOPROmpt
Do not issue a prompt for a missing password. Note that if a password is required but not provided by other means such as the PASSWORD parameter or the NETRC DATA file, your attempts to access data at the remote hose may fail, or may be done anonymously.

TRAnslate *trans_tablename*
Identifies the translation table to use when performing EBCDIC-ASCII data translation. CMS uses the first file found in your CMS search order named *trans_tablename TCPXLBIN*. *Trans_tablename* files are typically available on the the TCP/IP user disk, TCPMAINT 592.

If *pathname1* specifies a *foreign_host* that is an AIX®® or OS/2 system, for example, you may want to translate text file data so that it can be used from both platforms.

Some examples of *tablename* include "`Translate UK`" or "`Translate 10471252`".

TRANSLATE POSIX says that default VM BFS code page translations should be used when EBCDIC-ASCII translation takes place. EBCDIC (IBM-1047) is translated to and from ASCII (ISO 8859-1). The UNIX line terminator (lf - X'0A') is translated to the OpenExtensions VM line-end character (nl - X'15').

Trans_tablename may not be abbreviated.

NOTRAnslate
Do not translate file data.

LISt
LIST tells CMS that ASCII-EBCDIC translation should be done for data based on the value of the file extension. The file extension is defined to be the last component of a path name, that is, the characters following the last period in the path name. Up to eight characters are matched, and case is ignored.

The file extension list is defined by VMFILETYPE definitions in the first TCPIP DATA file found in the CMS search order. An error is returned if no TCPIP DATA file is found when the LIST keyword is used. If no VMFILETYPE or VMFILETYPEDEFAULT definitions are found in the TCPIP DATA file, translation is not done.

LIST is ignored when NOTRANSLATE is specified.

NOLISt
NOLIST tells CMS that ASCII-EBCDIC translation should be done for all file data.

NOLIST is ignored when NOTRANSLATE is specified.

ATTRCaching YES | NO
Specifies whether CMS should cache file and directory attributes. The default value is YES.

If ATTRCACHING NO is specified, READAHEAD 0 is set automatically.

ATTRMax *seconds*
Specifies the maximum lifetime of cached attributes in seconds. The valid range for *seconds* is 1–9999.

This option is ignored when ATTRCACHING NO is specified.

READAhead *blocks*
Specifies the maximum number of 8KB blocks to read ahead. The valid range is 0 to 16, and the default value is 1. The blocksize is 8192 (8KB).

Use the READAHEAD option with a value larger than 1 to improve performance when you are reading a large file sequentially.

This option is ignored when ATTRCACHING NO is specified.

RETry *retries*
Retries identifies the number of times to resend NFS requests to the remote host before returning an error. *Retries* may not exceed 9999.

If RETRY FOREVER is specified, CMS will continue attempts to resend a request until a response is received, or the CMS user terminates the attempt by entering HX. The CMS user may also terminate the attempt by ending the CMS session, logging off, IPLing, or issuing a SYSTEM RESET.

If not specified, RETRY 3 is the default.

TCP
Tells CMS that you want to use the TCP protocol to communicate with the remote host. If the NFS server does not support TCP, CMS will revert to use UDP.

UDP
Tells CMS that you want to use the UDP protocol to communicate with the remote host.

TIMeout *timeout_value*
Identifies how long CMS should wait for a response before resending a request to the remote host or returning an error.

Timeout_value value is specified in tenths of a second. The valid range for *timeout_value* is 1–9999.

If not specified, TIMEOUT 7 is the default.

VERsion *n*
Defines whether Version 3 (RFC 1813) or Version 2 (RFC 1094) of the NFS protocol should be used. If VERsion is not specified, CMS negotiates with the remote host to determine what version to use.

LOCalport *portnum*
Identifies the local reserved (well-known) port number CMS should use if reserved ports are required by the remote NFS server. For example, by default, some NFS server implementations on Linux® require that the client use reserved ports. The actual range of reserved ports varies between systems. Some NFS Servers will be satisfied with ports in the 1-1023 range, but others may require ports in the 512-1023 range. Check with the Server administrator to determine a system's specific range. Contact the TCP/IP administrator for your system to obtain reserved port numbers. The valid range for *portnum* is 1-1020.

Note that the z/VM TCP/IP stack restricts all "well-known" ports (ports 1-1023) from general use on a default basis. Thus, the default for the ASSORTEDPARMS statement is the RESTRICTLOWPORTS operand. Use the FREELOWPORTS operand, or specific port reservations via the PORT statement, to control authorization to use such ports.

CMS will attempt to obtain four ports (*portnum, portnum + 1, portnum + 2* and *portnum + 3*) to improve the performance of communications between NFS client and server. If the additional ports cannot be obtained, performance may be affected when reading or writing large files. Use the NETSTAT command to display the reserved ports in use.

LOCalport ANY
Ephemeral port numbers are chosen to establish connections with the remote NFS server. If not specified, LOCALPORT ANY is the default.

Usage Notes

1. The scope of the OPENVM MOUNT command is limited to the virtual machine in which the command is entered. In other words, the logical BFS hierarchy created by entering one or more OPENVM MOUNT commands is visible only to the processes that run in the virtual machine that issued the command. Other virtual machines may have different hierarchies.

2. You may also mount your root directory by including the POSIXINFO FSROOT statement in your CP directory entry.

3. You can use commands and programming interfaces that use path names without entering a MOUNT if you use the fully qualified BFS path name format for *pathname*. (That is, use the format including `'/../VMBFS:`*filepoolid*`:`*filespaceid*`/'`.)

4. After the OPENVM MOUNT command completes, the directory identified by *pathname1* is inserted into the logical hierarchy at the position identified by *pathname2*. From this point on, references to *pathname2* and its subdirectories refer to the subdirectories under *pathname1*, which is the mounted directory.

5. If a file system is already mounted on *pathname2*, you cannot mount a new file system on the same mount point without first unmounting the current file system.

6. OPENVM MOUNT (NOSTART can be used only for the ROOT. You may want to put this in your PROFILE EXEC.

7. For OPENVM MOUNT (NOSTART, it is not necessary that the BFS or subdirectory be available when the OPENVM MOUNT is entered. No permission checking or checking for existence is done until objects are used.

8. An OPENVM MOUNT affects all processes in the virtual machine.

9. The establishment of a new root affects the resolution of current working directories that are set with the OPENVM SET DIRECTORY command.

   For example, if you entered these commands:

   ```
   OPENVM MOUNT /../VMBFS:VMSYS:ROOT/ /
   OPENVM SET DIRECTORY /My_department/Reports
   ```

   When you referred to 'file-a', you would really be referring to

   ```
   /../VMBFS:VMSYS:ROOT/My_department/Reports/file-a
   ```

   If you then entered:

   ```
   OPENVM UNMOUNT /
   OPENVM MOUNT /../VMBFS:VMSYS:ROOT2/ /
   ```

   When you referred to 'file a', you would be referring to

   ```
   /../VMBFS:VMSYS:ROOT2/My_department/Reports/file-a
   ```

10. You can have up to ten concurrent mount points. They can be directories in different byte file spaces or network file systems. You cannot have multiple subdirectories from the same byte file space mounted concurrently.

11. The READ mode settings are enforced only within the virtual machine that issued the OPENVM MOUNT command.

12. Your access to files and directories in the mounted file system is controlled by the NFS server at the *foreign_host* specified as part of *pathname1*. The remote host typically authenticates the user ID and password provided and allows access to data based on the UID and GID associated with that user ID on the remote system. Alternatively, if no user ID is provided, the NFS server typically treats requests as anonymous.

    Some NFS servers may have different methods of protecting data. For example, VM's NFS server may use a minidisk link password provided in the *serveroptions* portion of the mount string to make sure that access to a CMS minidisk is allowed.

13. When a *username* and *password* are provided for an NFS mount, either through NETRC data or the USERID and PASSWORD options, CMS sends the user ID information to the remote host using the Sun PC-NFS protocol.

14. Use care when deciding whether or not to specify the TRANSLATE parameter. Specify it if the remote host is a system that stores text data in ASCII format and you need the data in EBCDIC format. Data is typically stored by S/390®® systems in EBCDIC format. The exception is Linux, which stores text data in ASCII.

15. OPENVM MOUNT for an NFS file system sets the value of the _EDC_KEEP_EMSG variable in the CENV group of GLOBALV to Y so that C will not set EMSG OFF.

Examples

First you need to set the root. The root can be set by the OPENVM MOUNT command. Enter:

```
OPENVM MOUNT /../VMBFS:VMSYS:ROOT/ /
```

The hierarchy might look something like .

```
/../VMBFS:VMSYS:ROOT/  /
```
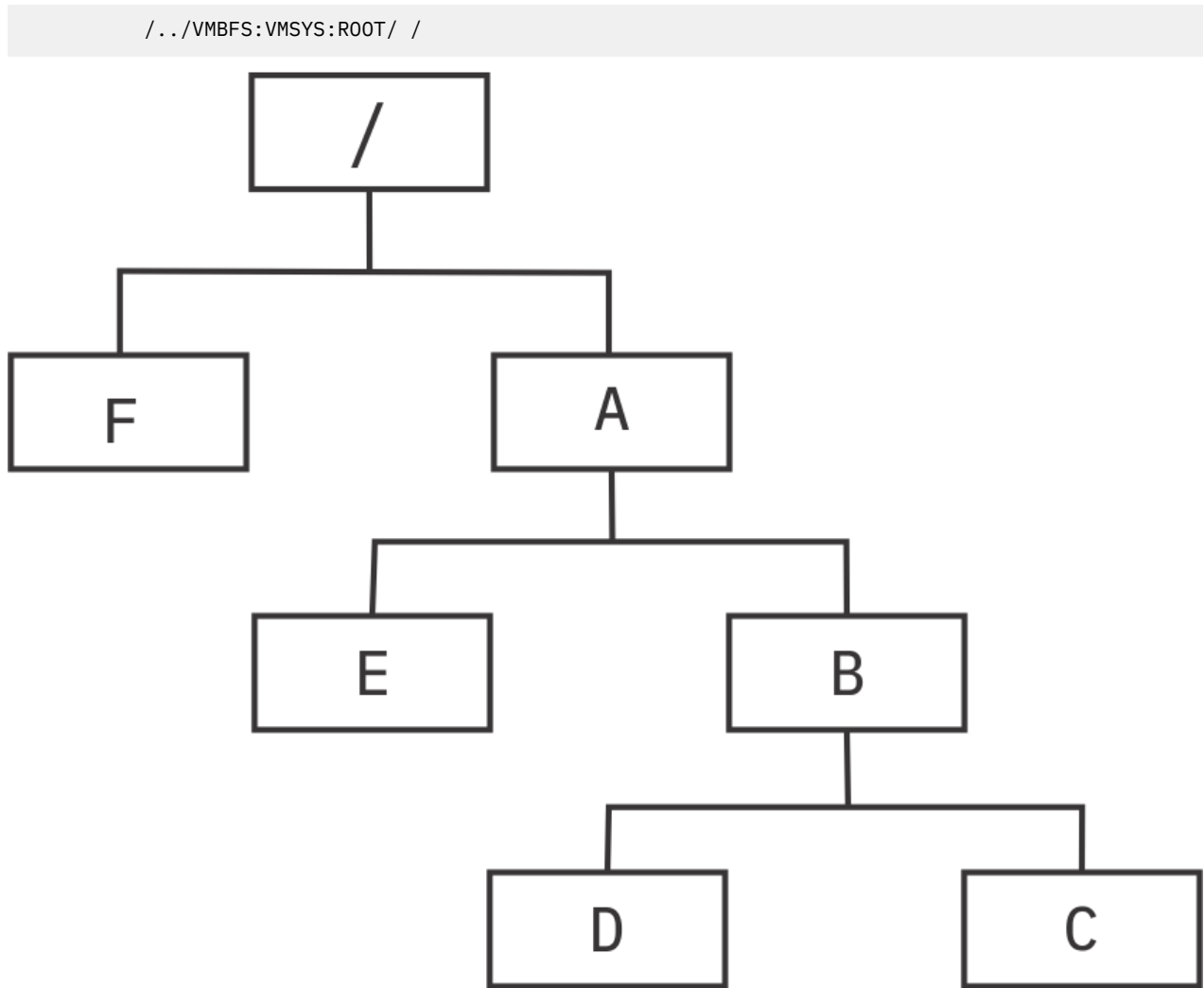

Figure 2. Setting the BFS path name root using OPENVM MOUNT

If you want to mount another byte file space on subdirectory /A/B/C, enter:

```
OPENVM MOUNT /../VMBFS:VMSYS:ROOT2/  /A/B/C
```

If ROOT2 contained objects x, y and z, the hierarchy would look something like .

```
/../VMBFS:VMSYS:ROOT/ /
```

```
                          ┌───────────┐
                          │     /     │
                          └───────────┘
            ┌────────────────────┴────────────────────┐
      ┌───────────┐                            ┌───────────┐
      │     F     │                            │     A     │
      └───────────┘                            └───────────┘
                                 ┌────────────────────┴────────────────────┐
                           ┌───────────┐                            ┌───────────┐
                           │     E     │                            │     B     │
                           └───────────┘                            └───────────┘
                                                        ┌────────────────────┴────────────────────┐
                                                  ┌───────────┐                            ┌───────────┐
                                                  │     D     │                            │     C     │
                                                  └───────────┘                            └───────────┘
                                                                                      ┌───────────┐
                                                                                      │     X     │
                                                                                      └───────────┘
                                                                          ┌────────────────────┴────────────────────┐
                                                                    ┌───────────┐                            ┌───────────┐
                                                                    │     Y     │                            │     Z     │
                                                                    └───────────┘                            └───────────┘
```
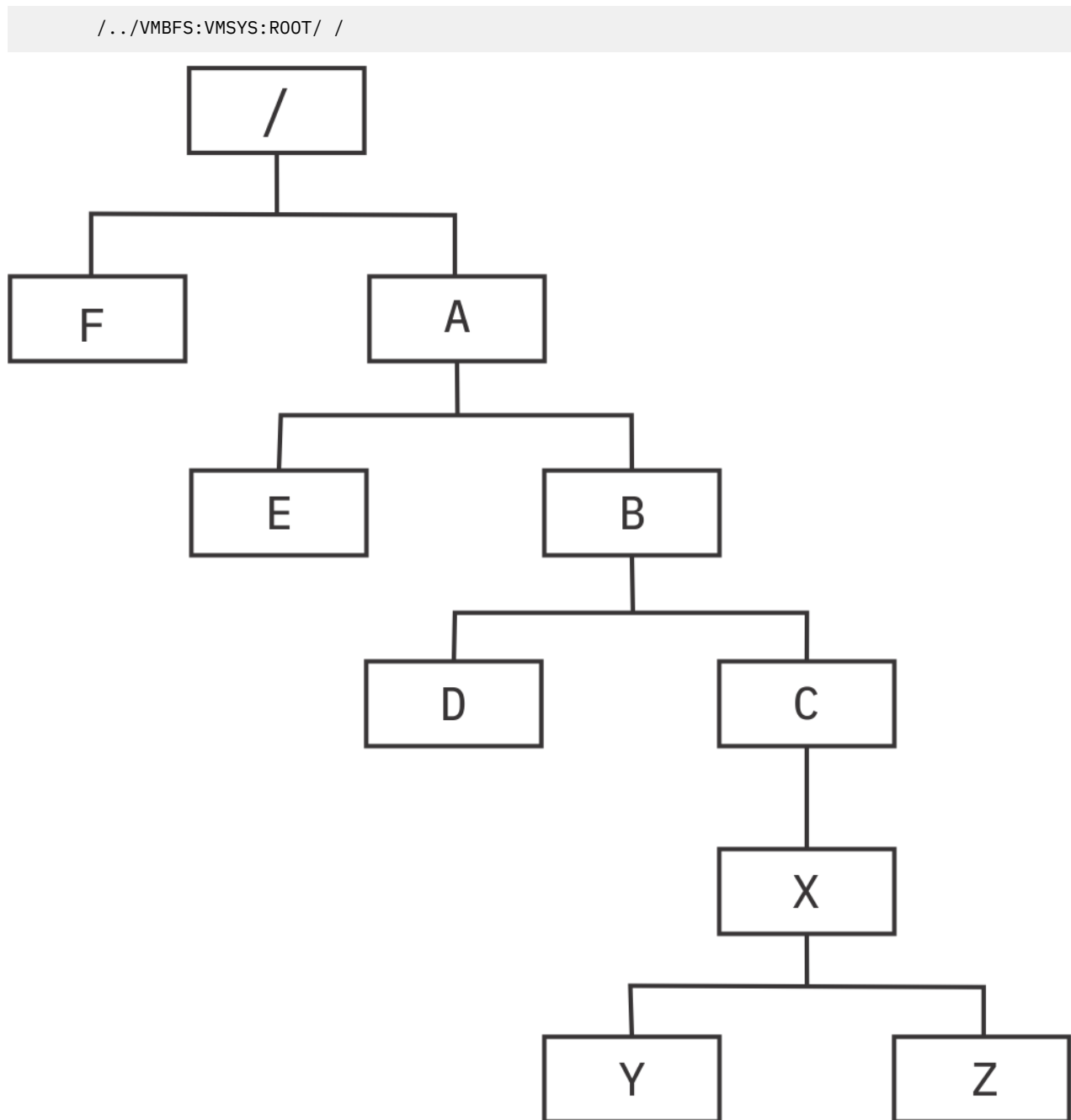
Figure 3. Mounting another BFS file space

You can then refer to file Z, which is in a different byte file space, but is logically included in your BFS hierarchy, as: /A/B/C/X/Z.

- Mounting an HFS directory on a z/OS® system

```
OPENVM MOUNT /../nfs:mvs/hfs/u/user,binary /u/mvsdir (notranslate
```

- Mounting an AIX home directory

```
OPENVM MOUNT /../nfs:aix6000/home /u/aixdir
```

- Mounting a BFS directory on a remote z/VM system

```
OPENVM MOUNT /../nfs:vmsys/../VMBFS:FP:FS/u/userid,trans=no /u/newdir
```

- Mounting an SFS directory on a remote z/VM system

```
OPENVM MOUNT /../nfs:vmsys/fp:fs.sub1,lines=nl,trans=no,nlvalue=15 /u/newdir
```

- Mounting a minidisk on a remote z/VM system

```
OPENVM MOUNT /../nfs:vmsys/userid.vdev,lines=nl,trans=no,nlvalue=15 /u/newdir
```

- Mounting an OS/2 directory.

```
OPENVM MOUNT   /../nfs:os2_serv/d:  /os2drive (translate posix nolist anonymous
```

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS002E	File *fn ft fm* not found	28
DMS029E	Invalid parameter *pathname*	24
DMS1018E	Your username and password could not be authenticated. The PC-NFS program at *foreign_host* returned *rc*	99
DMS1028E	The address is already in use	55
DMS1029E	Too many file systems mounted	55
DMS1153E	File pool is unavailable or unknown	99
DMS1153E	File space is unavailable or unknown	99
DMS2110E	Object is not a directory: *pathname*	40
DMS2113E	File system is not valid or not available	28
DMS2119E	Path name is not fully qualified: *pathname*	28
DMS2123E	File system {is already mounted|cannot be mounted at that mount point because something is already mounted there}	40

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM OWNER
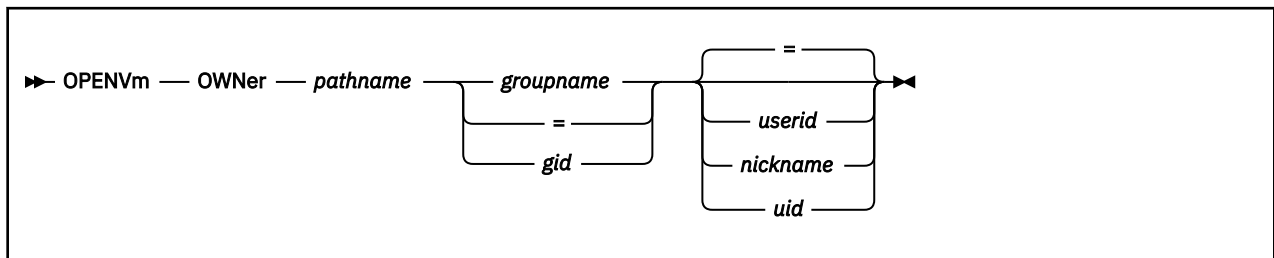
```
►►─ OPENVm ── OWNer ── pathname ──┬── groupname ──┬──┬─────── = ───────┬──►◄
                                  │               │  │                 │
                                  └─── = ─────────┘  ├───── userid ────┤
                                  └─── gid ──────────┤  ├──── nickname ─┤
                                                     └───── uid ────────┘
```

Authorization

Byte File System (BFS) permission checking applies to this command.

Purpose

The OPENVM OWNER command changes the group or owner (or both) of a byte file system (BFS) object.

Operands

pathname
Specifies the name of the object. See "Understanding Byte File System (BFS) Path Name Syntax" on page 368 for a description of the different forms of the BFS path name.

groupname
specifies the group name to whom you are changing ownership. You can specify the current group name (or =) if there is to be no change. This will be translated to the corresponding group ID (GID) as defined in the CP directory.

=
indicates that you do not wish to change the owning GID.

gid
specifies the GID to which you are changing ownership. You can specify the current GID (or =) if there is to be no change.

=
indicates that you do not wish to change the owning UID. This is the default.

userid or **nickname**
specifies the user ID (UID) to whom you are changing ownership. This will be translated to the corresponding UID as defined in the CP directory. If a nickname is used, it may not represent a list of users (use the NAMES command to define nicknames).

uid
specifies the UID to which you are changing ownership. You can specify the current UID (or =) if there is to be no change.

Usage Notes

1. OPENVM OWNER changes the owner GID, owner UID, or both of a BFS object. The owning GID can be changed by the current owner or a superuser. The specified group name's GID value must be the effective GID or a supplementary GID of the invoker. Only a superuser can change the owning UID of an object.

2. The group name must evaluate to a valid entry in the group data base. The user ID must evaluate to a valid CP directory entry.

3. If this command is entered specifying a symbolic link or an External Link of type MOUNT, the link name is resolved to a file and the ownership of the file is changed.

4. If the path name refers to a BFS regular file, the set-user-ID-on-execution and set-group-ID-on-execution permissions are automatically turned off when ownership is modified.

5. Use the OPENVM LISTFILE command with the option OWNERS to query (or determine) the group name, user ID, and permissions associated with objects in the BFS.

6. When *pathname* refers to an object in an NFS-mounted file system, you must meet the authorization requirements imposed by the remote NFS server.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS149E	{Userid *userid*|Groupname *groupname*} not valid	32
DMS637E	Missing *nodeid* for the AT operand	24
DMS647E	Userid not specified for *nickname* in *userid* NAMES file	32
DMS1209E	Nickname *nickname* resolved to more than one user ID; the owner can be set for only one user at a time	88
DMS2129E	{UID|GID} not found for {Userid|Groupname}	32

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM PARCHIVE

```
►►─ OPENVm ── PARchive ─┬─ DUMP ─┬─ archive_name ─┬──────────────────┬─►◄
                        └─ LOAD ─┘                 └─( ─┤ OPTIONS ├─┬──┘
                                                                   └─)─┘

OPTIONS

►►─┬─ 1 ─ TO ───┬─┬─ TAP1 ──┬─►◄
   └─── FRom ───┘ ├─ TAP n ─┤
                  └─ vdev ──┘
```

Notes:

   [1] DUMP will only accept the TO option. LOAD will only accept the FROM option.

Authorization

General User

Purpose

Use the OPENVM PARCHIVE command to copy a portable archive to or from tape.

Operands

DUMP
   copies archive to tape.

LOAD
   copies archive from tape.

archive_name
   the name of the existing archive.

Options

TAP*n*
vdev
   specifies the device name (TAP*n*), or alternatively the virtual device number (*vdev*) of the tape device on which the command is to operate. The following names and corresponding virtual device numbers are valid. (See *z/VM: CMS User's Guide* for details on device names and virtual device numbers for tape devices.) The default is TAP1.

Device Name	Device Number
TAP0	180
TAP1	181
TAP2	182
TAP3	183
TAP4	184
TAP5	185
TAP6	186

Device Name	Device Number
TAP7	187
TAP8	288
TAP9	289
TAPA	28A
TAPB	28B
TAPC	28C
TAPD	28D
TAPE	28E
TAPF	28F

Usage Notes

1. An *archive_file* contains information that completely represents a set of other files, including their names, contents, attributes, permission bits, positions in the directory hierarchy, and so on. You create an archive file by using the **tar**, **pax** or **cpio** shell command. See "tar -- Manipulate the tar archive files to copy or back up a file" on page 318, "cpio -- Copy in/out file archives" on page 85, and "paste — Merge corresponding or subsequent lines of a file" on page 236.

2. For tape operations the OPENVM PARCHIVE command uses Phase Encoding (PE) format with a data density of 1600 BPI, which is the most standard interchange format.

3. If OPENVM PARCHIVE cannot extract a particular file when reading or find a particular file when writing, it generates an error message. If an I/O error occurs during tape operations, OPENVM PARCHIVE ends immediately without further processing.

4. For error messages DMS110S and DMS111S, the problem is reflected in the return code:

    RC=3
    I/O error

    RC=4
    Device is not valid.

    RC=5
    Device does not exist.

    RC=6
    Volume is write protected.

    RC=7
    Manual rewind/unload of tape.

5. To end the OPENVM PARCHIVE LOAD command, you should enter 'END' or 'end' in response to 'DMSPAI441R Enter VOLID information:' message. Entering CANCEL in response to the 'DMSPAI441R Enter VOLID information:' message will terminate the OPENVM PARCHIVE command abnormally.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS027E	Invalid device *vdev*	24
DMS043E	TAP*n* (*vdev*) is file protected	36

Number	Text	Return Code
DMS110S	Error reading TAPn(*vdev*)	100
DMS111S	Error writing TAPn(*vdev*)	100
DMS113S	TAP*n* (*vdev*) not attached	100
DMS115S	Device *name* cannot write the *format* recording format	88
DMS431E	TAP*n* (*vdev*) has been rewound and unloaded by operator. Requested tape function may not have been executed.	4

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM PATHDEF CREATE

```
►►── OPENVm ──── PATHdef ──── CREate ──── ddname ──── pathname ──►◄
```

Authorization

General User

Purpose

The OPENVM PATHDEF CREATE command establishes path definitions for OS ddnames (data definition names) that are to be opened by **fopen()** system calls.

Operands

ddname
> specifies a 1- to 8-character simulated MVS ddname (data definition name) that represents the path definition.

pathname
> specifies a byte file system (BFS) path name. See "Understanding Byte File System (BFS) Path Name Syntax" on page 368 for a description of the different forms of the BFS path name.

Usage Notes

When *pathname* refers to an object in an NFS-mounted file system, you must meet the authorization requirements imposed by the remote NFS server.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.

OPENVM PATHDEF DELETE

```
►►─ OPENVm ── PATHdef ── DELete ─┬─ ddname ─┬─►◄
                                 └─── * ────┘
```

Authorization

General User

Purpose

The OPENVM PATHDEF DELETE command deletes path definitions for OS ddnames (data definition names).

Operands

ddname
    specifies a 1- to 8-character simulated MVS ddname that represents the path definition to be deleted.

*
    causes paths for all active path and ddname associations to be deleted.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.

OPENVM PATHDEF QUERY

```
►►─ OPENVm ── PATHdef ── Query ──┬─ ddname ─┬──────────┬──►◄
                                 └─── * ────┘  └─(STACK─┘
```

Authorization

General User

Purpose

The OPENVM PATHDEF QUERY command displays path definitions for OS ddnames (data definition names).

Operands

ddname
> specifies a 1- to 8-character simulated MVS™ ddname. When specified, the OPENVM PATHDEF QUERY command returns the POSIX path associated with the ddname.

*
> causes paths for all active path and ddname associations to be returned.

Options

STACK
> causes the results of the QUERY command to be placed in the program stack instead of being displayed at the terminal. These results are stacked LIFO (last in first out).

Usage Notes

1. The output of OPENVM PATHDEF QUERY is truncated to 255 characters if the STACK option is used. Use CMS Pipelines for responses greater than 255 characters. For more information on CMS Pipelines, see *z/VM: CMS Pipelines User's Guide and Reference*.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS2145I	No user defined PATHDEFs in effect	4
DMS2146E	No user-defined PATHDEFs in effect for ddname: *ddname*	4

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.

OPENVM PERMIT
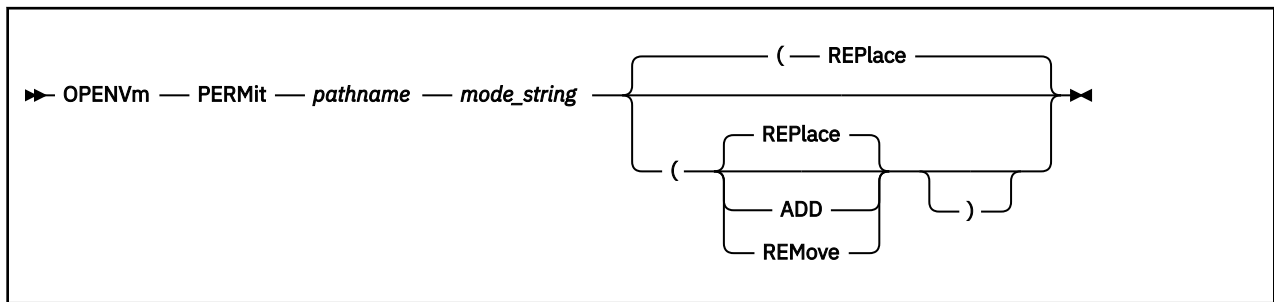
Authorization

Byte file system (BFS) permission checking applies to this command.

Purpose

The OPENVM PERMIT command changes the permission bits used to control the owner access, group access, and general access to a byte file system (BFS) object. It can also be used to set flags that allow the user ID (UID) and group ID (GID) of an executable file to be set during execution.

Operands

pathname
Specifies the name of the file or directory. See "Understanding Byte File System (BFS) Path Name Syntax" on page 368 for a description of the different forms of the BFS path name.

mode_string
mode_string is entered in the following format:

```
rwx rwx rwx
```

and contains the information used to determine the permissions the user wants to change for the owner, group and public. This is entered in three groups: owner, group and public. Each group consists of permissions for read, write, and execute access to BFS objects. The owner and group strings also contain permissions for the set-UID and set-GID. The first three characters apply to the owner, the second three to the group, and the third three to public. Each group of characters is mapped as follows:

r
in the first position indicates read permission is to be changed for the BFS object specified.

-
in the first position indicates read permission is to be

- turned off if the REPlace option was used for the BFS object specified.
- left as is if the ADD or REMove options were used for the BFS object specified.

w
in the second position indicates write permission is to be changed for the BFS object specified.

-
in the second position indicates write permission is to be

- turned off if the REPlace option was used for the BFS object specified.
- left as is if the ADD or REMove options were used for the BFS object specified.

x
in the third position indicates execute permission is to be changed if the BFS object specified is a BFS file; search permission will be changed for a BFS directory.

s or S

An executable file can have an additional attribute, which is indicated in the execute (third) position. This permission setting is used to allow a program temporary access to files that are not normally accessible to other users. This permission bit sets the effective user ID or group ID of the user process executing a program to that of the file whenever the file is run. This permission is valid only in the third position of the owner and group.

s

In the owner permissions section, indicates that the set-user-ID on execution bit and execute (search) permission are to be set for the BFS file.

In the group permissions section, indicates that the set-group-ID on execution bit and execute (search) permission are to be set for the BFS file.

S

In the owner permissions section, indicates that the set-user-ID on execution bit is set, but the execute (search) permission is not.

In the group permissions section, indicates that the set-group-ID on execution bit is set, but the execute (search) permission is not.

-

in the third position indicates execute/search permission is to be

- turned off if the REPlace option was used for the BFS object specified.
- left as is if the ADD or REMove options were used for the BFS object specified.

REPlace

The specified permissions will replace the existing permission. REPlace is the default.

ADD

The specified permissions will be added to the existing permission.

REMove

The specified permissions will be removed from the existing permission.

Usage Notes

1. The effective UID must match the owner UID of the object, or the issuer must have the appropriate privileges.

2. By setting the set-UID-on-execution permission, when this file is run, the effective UID of the process is set to the file owner's UID. The process then seems to be running under the UID of the file owner instead of the UID of the actual invoker.

3. By setting the set-GID-on-execution permission, when this file is run, the effective GID of the caller is set to the file owner's GID. The caller then seems to be running under the GID of the file instead of the GID of the actual invoker.

   Note: The set-GID-on-execution bit is cleared when the caller does not have the appropriate privileges and the GID of the file owner does not match the effective GID (or one of the supplementary GIDs) of the caller.

4. If this command is entered specifying a symbolic link or an External Link of type MOUNT, the link name is resolved to a file and the permissions of the file are changed.

5. When *pathname* refers to an object in an NFS-mounted file system, you must meet the authorization requirements imposed by the remote NFS server.

Examples

1. The object **names** was created previously, and you want to add write permission to the group associated with the **names** file and add read permission to public.

```
openvm list (own
Directory = '/'
User ID   Group Name  Permissions Type  Path name component
```

```
user1000  CMSUSRS     rwx r-x --x  F     'names'
Ready;
openvm permit names --- -w- r-- (ADD
Ready;
```

OPENVM LISTFILE will now show:

```
openvm list (own
Directory = '/'
User ID   Group Name  Permissions Type  Path name component
user1000  CMSUSRS     rwx rwx r-x  F     'names'
Ready;
```

2. If you enter:

```
openvm permit /names --- --- r-- (remove
```

Public read permission is removed from the /names file.

3. If you enter:

```
openvm permit /names r-x r-x r-x
```

The permission bits for the /names file are replaced with 'r-x r-x r-x'

OPENVM LISTFILE will now show:

```
openvm list (own
Directory = '/'
User ID   Group Name  Permissions Type  Path name component
user1000  CMSUSRS     r-x r-x r-x  F     'names'
Ready;
```

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM PUTBFS

```
►►── OPENVm ── PUTbfs ──┤ Fileid ├── pathname2 ──────────────────────►◄
                                           └─( ─┤ Options ├─┐
                                                        └─ ) ─┘
```

Fileid

```
►►── fn ── ft ──┬── * ──────┬──►◄
                ├── fm ─────┤
                └── dirid ──┘
         └──── pathname1 ────┘
```

Options

```
   ┌── NEWFile ──┐    ┌── BFSLine ── NL ──────────┐¹   ┌── NEWDate ──┐
►►─┴── REPlace ──┴────┤                            ├────┴── OLDDate ──┴──►
                      ├── BFSLine ──┬── NONE ──────┤
                      │             ├── CRLF ──────┤
                      │             ├── CRNL ──────┤
                      │             ├── /string/ ──┤
                      │             └── /hexstring/┤
                      └──────── MODule ────────────┘

   ┌──────── NOTRAnslate ─────────────┐
►──┴── TRAnslate ── codepage1 ── TO ── codepage2 ──┴──►◄
```

Notes:

¹ BFSLINE NL is the default unless the *ft* of Fileid is MODULE, in which case MODULE is the default.

Authorization

Byte file system (BFS) permission checking applies to this command.

Purpose

Use the OPENVM PUTBFS command to copy data into a byte file system (BFS) regular file from another BFS regular file, a file in a SFS directory, or a file on a CMS minidisk.

Operands

pathname1
>   is the BFS path name of the file to be copied. See "Understanding Byte File System (BFS) Path Name Syntax" on page 368 for a description of the different forms of the BFS path name.

fn ft *
fn ft fm
fn ft dirid
>   is the SFS or minidisk file to be copied.

pathname2
> is the BFS path name of the file to be created or replaced.

Options

NEWFile
> checks that an object with the same file ID as the output file does not already exist. If it does exist, an error message is displayed and the OPENVM PUTBFS command terminates. This option is the default so that an existing file is not inadvertently destroyed.

REPlace
> causes the output file to replace an existing BFS regular file with the same path name.

BFSLine
> Use the BFSLINE option to tell CMS how to translate records into a BFS byte stream. This option is ignored (has no effect) if the source file is a BFS file.
>
> BFSLINE NONE does not do any insertion of an end-of-line character.
>
> If you specify anything other than BFSLINE NONE, you can define an end-of-line character or characters to insert into the file.
>
> **NL**
> > says that the new line character (X'15') should be inserted at the end of each record to delineate lines. This is the default.
>
> **NONE**
> > says that all records in the file should be concatenated into a byte stream with no end-of-line characters inserted.
>
> **CRLF**
> > says that carriage return/line feed (X'0D25') should be inserted at the end of each record to delineate lines.
>
> **CRNL**
> > says that carriage return/new line (X'0D15') should be inserted at the end of each record to delineate lines.
>
> **/string/**
> > allows the user to specify a 1-2 character string that is inserted at the end of each record to delineate lines. Blanks may not be included in *string*, and it may not be X' or x'.
>
> **/hexstring/**
> > specifies a hexadecimal string of 2 or 4 characters that defines the value to be used for BFSLINE. The *hexstring* must be in the format X'nnnn' or X'nn'. You must not specify any spaces in the string, and there must be 2 or 4 hexadecimal characters in the string.
>
> BFSLINE NL is the default unless the *ft* for **Fileid** is MODULE.

MODule
> specifies that the CMS file is in the format created by the GENMOD or BIND command. MODULE must be specified (or in effect by default) if the CMS file is in this format in order for the resulting BFS file to be executable. This option is ignored if the source file is a BFS file.
>
> MODULE is the default if the file type of the CMS file is MODULE.

NEWDate
> uses the current date and time for the date and time of the new file. This is the default.

OLDDate
> uses:
>
> • Time of last data modification (if the source file is a BFS file), or
> • Date of last update (if the source file is a CMS file) as the time of last data modification of the target file.

If you attempt to use the OLDDATE option and you are not a superuser or the owner of the target file, a warning message will be issued and the current time will be used. If the source file is a CMS file that is older than January 1, 1970, the date of last update used for the target BFS file will be January 1, 1970 with a time of last data modification of 00:00:00.

NOTRAnslate
Indicates that no code page translation should occur.

TRAnslate
Indicates that the characters in the file should be translated as part of the OPENVM PUTBFS operation. This option is ignored if the MODULE option is specified.

> ***codepage1***
> Specifies the code page for the source file.

> **TO** ***codepage2***
> Specifies the code page for the target file.

Any code page is allowed that is supported by the CMS Pipelines XLATE stage. See *z/VM: CMS Pipelines User's Guide and Reference*.

If an end-of-line character is specified, it is not affected by code page translation. That is, code page translation takes place prior to the insertion of end-of-line characters that turn records into a byte stream.

Usage Notes

1. Permission for a new BFS file are set to 'rwx r-x r-x' under either of these conditions:

   - The source file is a BFS file and at least one of its execute permission bits is on.
   - The source file is a CMS file and the MODULE option is used or is the default because the file type of the source file is MODULE.

   If neither of these conditions is true, the permissions are set to 'rw- r-- r--'.

   Setting the mask can turn off additional permissions. See "OPENVM SET MASK" on page 453. Use OPENVM PERMIT to change permissions after the file is created.

2. An OPENVM PUTBFS (REPLACE to an existing BFS file will not change the permissions of the file.

3. When a new BFS file is created, the owning UID established is the effective UID of the process that issued the request. The group name is the GID of the parent directory. Use OPENVM OWNER to change either the owning user ID or group name.

4. If the source or target of an OPENVM PUTBFS is a BFS object, but it is not a BFS regular file, the command will fail.

5. Use the */string/* or */hexstring/* option when you want to specify a different end-of-line character than those listed above. For example, if you want to use X'0D' to indicate end-of-line, specify the BFSLINE /X'0D'/ option.

   When specifying a BFSLINE value for use on files containing DBCS characters, be careful to use a value that will not conflict with DBCS characters. The hexadecimal code for a DBCS character must be X'00', X'40', or in the range of X'41' to X'FE'.

6. The NL and CRLF mnemonics translate into values defined by code page IBM-1047.

7. Avoid copying modules generated with the MAP option of the GENMOD command, or non-relocatable modules, into the byte file file system. The multitasking nature of POSIX could interfere with running modules that have either of these characteristics. These types of modules rely on fixed resources:

   - The module generated by the GENMOD MAP option relies on the loader tables as its fixed resource.
   - The non-relocatable module relies on its fixed location in storage where the module must be loaded.

   Either of these fixed resources could be overwritten by another program.

8. If **Fileid** specifies a CMS record file (sfs or minidisk file), the file ID will be converted to upper case during OPENVM PUTBFS processing. However, if **Fileid** specifies a BFS path name, a mixed case file ID will be respected and will not be converted to uppercase.

9. When path names refer to files in NFS-mounted file systems, you must meet the authorization requirements imposed by the remote NFS servers.

10. Use the TRANSLATE option carefully if the source or target files are in an NFS-mounted file system. The NFS mount allows you to specify whether file data is translated. Do not tell CMS to translate data a second time using the TRANSLATE option on OPENVM PUTBFS.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS024E	File already exists; specify REPLACE option for: *pathname2*	28
DMS069E	Filemode *fm* not accessed	36
DMS132E	File too large: *pathname*	88
DMS618E	NUCEXT failed, return code *rc*	104
DMS639E	Error in {PIPE|DMSCCE} routine, return code was *rc*	104
DMS1137E	Object is locked; deadlock detected	70
DMS1184E	File *fn ft fm* not found or you are not authorized for it	28
DMS2041W	You are not permitted to use the OLDDATE option	4
DMS2109E	Object is a directory: *pathname*	40
DMS2125E	Path name ends with a slash: *pathname*	40
DMS2538E	File is not in MODULE format	32

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in the Shared File System	See "SFS and CRR Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using a file	See "File Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM QUERY DEBUG

```
►►── OPENVm ──── Query ──── DEBUG ──►◄
```

Authorization

General User

Purpose

The OPENVM QUERY DEBUG command displays information about settings set by the OPENVM DEBUG command. This command is useful for problem diagnosis in the BFS and NFS client environments.

Responses

The response shows the settings for all debug tracing options. It also displays the WRAPSIZE value, which shows the number of NFS Request trace events retained in the trace table.

Usage Notes

1. Use the OPENVM QUERY DEBUG command to display information about settings set with the OPENVM DEBUG command. See "OPENVM DEBUG" on page 387 for more information.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM QUERY DIRECTORY

```
►►─ OPENVm ── Query ── DIRectory ─┬──────────────────────────────┬─►◄
                                  │  ┌──────── FIFO ────────┐     │
                                  └─(─┬─ STACK ─┼─────────────┼──┬─┬─ ) ─┘
                                      │          └─ LIFO ─┘    │  │
                                      ├────────── FIFO ────────┤  │
                                      └────────── LIFO ────────┘  │
```

Authorization

General User

Purpose

Use the OPENVM QUERY DIRECTORY command to display your current working directory.

Options

STACK
> causes the results of the QUERY command to be placed in the program stack instead of being displayed at the terminal. The information is stacked either FIFO (first in first out) or LIFO (last in first out). The default order is FIFO.

FIFO
> (first-in first-out) is the default option for STACK. FIFO causes the results of the QUERY command to be placed in the program stack instead of being displayed at the terminal. The information is stacked FIFO. The options STACK, STACK FIFO, and FIFO are all equivalent.

LIFO
> (last-in first-out) causes the results of the QUERY command to be placed in the program stack rather than being displayed at the terminal. The information is stacked LIFO. This option is equivalent to STACK LIFO.

Responses

If no current working directory has been specifically established, you would see:

```
Directory = '/'
```

After entering OPENVM QUERY DIRECTORY, you might see:

```
Directory = '/childrens/animal facts/ '
```

Usage Notes

1. The display is bounded by quotes because it can contain blanks. If a quote is actually part of the path name, it will be displayed as two quotes in a row.

2. The fully qualified byte file system (BFS) path name can be up to 1023 characters in length.

3. You can establish a current working directory using the OPENVM SET DIRECTORY command or by specifying the POSIXINFO IWDIR statement in your CP directory entry.

4. The output of OPENVM QUERY DIRECTORY will be truncated to 255 characters if any of the STACK options are used and the response data exceeds 255 characters in length. Use CMS Pipelines for responses greater than 255 characters.

5. Use the OPENVM QUERY MOUNT command to display your root.

6. If the working directory is an absolute path name, it must reside in the mounted logical hierarchy so it can be resolved by OPENVM QUERY DIRECTORY. (This will also allow it to be displayed.)

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM QUERY FORK

```
►►─ OPENVm ── Query ── FORk ─►◄
```

Authorization

General User

Purpose

Use the OPENVM QUERY FORK command to display the current setting for fork (BPX1FRK) processing. Use "OPENVM SET FORK" on page 452 to set fork (BPX1FRK) processing. For information about the fork (BPX1FRK) service, see *z/VM: OpenExtensions Callable Services Reference*.

Responses

```
FORK  =  ON
```

or

```
FORK  =  OFF
```

Where:

ON
   indicates that fork (BPX1FRK) calls will be processed.

OFF
   indicates that fork (BPX1FRK) calls will not be processed.

Messages and Return Codes

There are no error messages issued by this command. However, system messages may be issued. The reasons for these messages and their location are:

| Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

OPENVM QUERY LINK

```
►►── OPENVm ── Query ── LINk ── pathname ──►

    ┌─────────────────────────────────────────┐
    ▼                                         │
 ───┴──┬─ ( ──────────────────────────┬────┬──►◄
       │                              │    │
       │         ┌─ FIFO ─┐           │    │
       └─ STACK ─┴────────┴─ LIFO ────┤ ) ─┘
       │                              │
       ├──────────── FIFO ────────────┤
       │                              │
       └──────────── LIFO ────────────┘
```
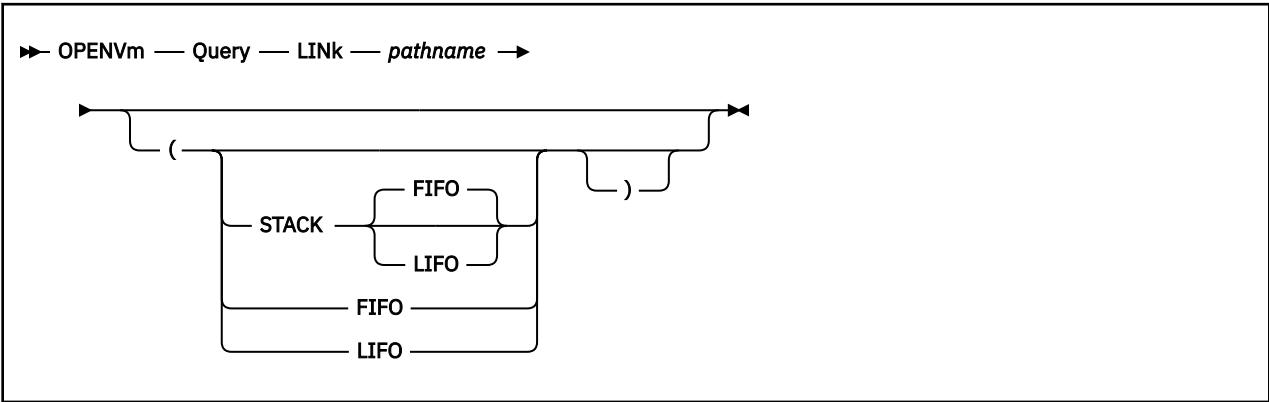
Authorization

General User

Purpose

The OPENVM QUERY LINK command displays information associated with symbolic or external links in a byte file system (BFS).

Operands

pathname
Specifies the name of the file or directory. See "Understanding Byte File System (BFS) Path Name Syntax" on page 368 for a description of the different forms of the BFS path name.

Options

STACK
causes the results of the QUERY command to be placed in the program stack instead of being displayed at the terminal. The information is stacked either FIFO (first in first out) or LIFO (last in first out). The default order is FIFO.

FIFO
(first-in first-out) is the default option for STACK. FIFO causes the results of the QUERY command to be placed in the program stack instead of being displayed at the terminal. The information is stacked FIFO. The options STACK, STACK FIFO, and FIFO are all equivalent.

LIFO
(last-in first-out) causes the results of the QUERY command to be placed in the program stack rather than being displayed at the terminal. The information is stacked LIFO. This option is equivalent to STACK LIFO.

Responses

Results will be in the format:

```
Type       ExtType     Data
linktype   elinktype   linkdata
```

Where:

linktype
indicates what type of link is represented by *pathname*. It will have one of the following values:

> **SYMBOLIC**
> Indicates *pathname* represents a link to a BFS path name. When *linktype* is **SYMBOLIC**, *linkdata* will be displayed as a BFS path name, delineated by single quotation marks.
>
> **EXTERNAL**
> Indicates the link references data that may reside outside a BFS. It can be used to reference a CMS executable file, or a file residing in the record file system or in an application-defined format.

elinktype
Indicates the type of data stored as part of external link. It will be displayed as one of the following values:

> **-**
> A dash will be displayed if *linktype* is **SYMBOLIC**.
>
> **CMSEXEC**
> Indicates the external link is to a CMS executable file. When the type is **CMSEXEC**, the accompanying data is to be interpreted as a CMS file ID (in other words, file name, file type, and file mode).
>
> **Note:** The data will consist of whatever was entered when the external link was created.
>
> If file type and file mode were not explicitly entered, they will not appear as part of the output.
>
> **CMSDATA**
> Indicates the external link refers to a file that will be opened by the C run time library ANSI-*fopen()* routine when the external link is opened. When an *elinktype* of **CMSDATA** is displayed, the accompanying *linkdata* will be the *string1* information specified when the external link was created. This is expected to be in the format of an *fopen()* parameter list.
>
> **MOUNT**
> Indicates the external link is an MEL (Mount External Link). When an *elinktype* of MOUNT is displayed, the *linkdata* is the *string2* specified in the OPENVM CREATE EXTLINK syntax when the external link was created.
>
> **nnn**
> This may be a numeric value in the range of 100-200. This will be displayed if this is an application defined format of an external link if the **CODE** keyword is used to create the external link. When *nnn* is displayed, the format and contents of the accompanying *linkdata* will be as it was entered in the *string2* parameter when the external link was created. Its format and meaning is defined by the application.

linkdata
Consists of the data stored with the external or symbolic link. The format and expected content of *linkdata* is a function of *linktype* and *elinktype* as described above.

Usage Notes

1. The output of OPENVM QUERY LINK will be truncated to 255 characters if any of the STACK options are used and the response data exceeds 255 characters in length.
2. Use the OPENVM QUERY LINK command to display information associated with symbolic or external links. For information about links to BFS regular files, use OPENVM LISTFILE with the NAMES option.
3. When *pathname* refers to an object in an NFS-mounted file system, you must meet the authorization requirements imposed by the remote NFS server.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS2117E	Object is not a symbolic or external link: *pathname*	88

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM QUERY MASK
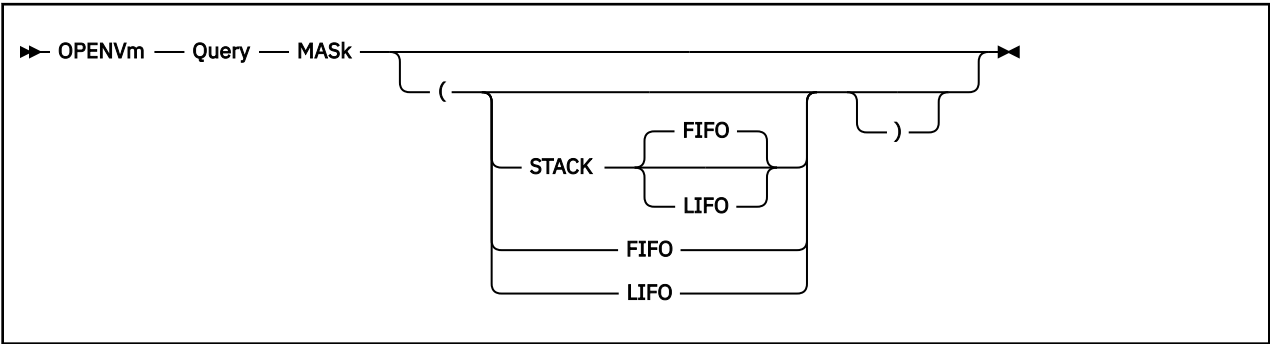
Authorization

General User

Purpose

Use the OPENVM QUERY MASK command to display the file creation mask permission values in effect. Use "OPENVM SET MASK" on page 453 to set the file creation mask.

Options

STACK
> causes the results of the QUERY command to be placed in the program stack instead of being displayed at the terminal. The information is stacked either FIFO (first in first out) or LIFO (last in first out). The default order is FIFO.

FIFO
> (first-in first-out) is the default option for STACK. FIFO causes the results of the QUERY command to be placed in the program stack instead of being displayed at the terminal. The information is stacked FIFO. The options STACK, STACK FIFO, and FIFO are all equivalent.

LIFO
> (last-in first-out) causes the results of the QUERY command to be placed in the program stack rather than being displayed at the terminal. The information is stacked LIFO. This option is equivalent to STACK LIFO.

Responses

OPENVM QUERY MASK displays the value in the file creation mask for the owner, group, and public in terms of read, write, and execute (search) access. Permissions are specified as combinations of the letters r (read), w (write), and x(execute).

```
OWNER    GROUP    PUBLIC
 bbb      bbb       bbb
```

where *b* may be one of the following:

r
> Indicates permission for the specified user class (owner, group, or public) and READ access mode have been allowed by the OPENVM SET MASK command.

w
> Indicates permission for the specified user class (owner, group, or public) and WRITE access mode have been allowed by the OPENVM SET MASK command.

x

Indicates permission for the specified user class (owner, group, or public) and EXECUTE access mode have been allowed by the SET MASK command.

-

Indicates permission for the specified user class (owner, group or public) and that access mode (read, write, or execute) are denied. For example, if all of the owner fields are marked with a dash (-), then the owner will not have read, write, or execute permission to any newly created file.

Usage Notes

1. The file creation mask is used whenever a file or directory is created in the BFS by the current process.

2. Permissions marked as '**-**' in the file creation mask will not be granted even if the program that creates the file attempts to grant them. For example, if the output of OPENVM QUERY MASK is:

```
OWNER     GROUP      PUBLIC
 r-x       r--        r--
```

this indicates that all permissions will be denied when a new file is created except read for group and public, and read and execute for the owner. For more information, see "Handling Security for Your Files" in *z/VM: OpenExtensions User's Guide*.

3. You can change the permissions of a particular file or directory by using the OPENVM PERMIT command.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.

OPENVM QUERY MOUNT
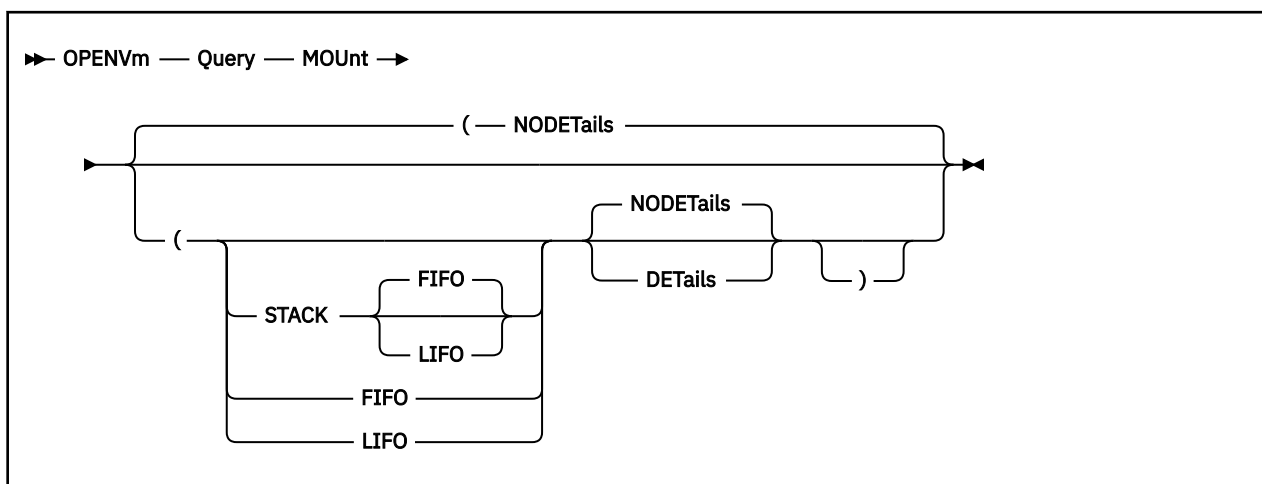
Authorization

General User

Purpose

Use the OPENVM QUERY MOUNT command to display what is mounted in your hierarchy.

Options

STACK
causes the results of the QUERY command to be placed in the program stack instead of being displayed at the terminal. The information is stacked either FIFO (first in first out) or LIFO (last in first out). The default order is FIFO.

FIFO
(first-in first-out) is the default option for STACK. FIFO causes the results of the QUERY command to be placed in the program stack instead of being displayed at the terminal. The information is stacked FIFO. The options STACK, STACK FIFO, and FIFO are all equivalent.

LIFO
(last-in first-out) causes the results of the QUERY command to be placed in the program stack rather than being displayed at the terminal. The information is stacked LIFO. This option is equivalent to STACK LIFO.

DETails
includes detailed information for the mount point. The format and content of this information depends upon the type of file system mounted.

DETAILS displays additional information only for Network File System (NFS) mounts. DETAILS displays values for local NFS options and shows RPC and NFS request statistics.

NODETails
does not include detailed information for the mount point. This is the default.

Responses

The following is an example of the output if you have only your root mounted:

```
Mount point = /
Type  Stat  Mounted
BFS   R/W   '/../VMBFS:VMSYS:ROOT/ /'
```

where:

Mount point

Is the location where data is mounted If the file system containing this location is not available, the **Mount point** will be "UNKNOWN".

Type

Identifies what is mounted **BFS** indicates that a byte file system (BFS) or BFS subdirectory is mounted.

Stat

Identifies how the file system or directory is mounted.

R/W -

Indicates that it is mounted in write mode.

R/O -

Indicates that it is mounted in read only mode.

***

Indicates that the file system or directory mounted is not available.

Mounted

Is the name of the BFS or BFS subdirectory, or NFS file system that is mounted.

If the **Stat** field is ***, indicating that the file system or directory is not available, then as much information as possible is displayed.

You may have other things mounted in addition to your root. For example:

```
Mount point = /
Type  Stat  Mounted
BFS   R/W   '/../VMBFS:VMSYS:ROOT/ /'

Mount point = '/another directory'
Type  Stat  Mounted
BFS   R/W   '/../VMBFS:VMSYS:ROOT2/activity reports/1993'
```

If nothing is mounted, the following response is returned:

```
 Nothing is mounted
```

```
openvm q mount (details
Mount point = '/u/mvsdir'
Type  Stat  Mounted
NFS   R/W  '/../NFS:MVS/hfs/u/user'
UID      9999     GID        8888      Userid   mvsuserid
Translate          NOTRANSLATE          NOLIST
Attrcach  YES      Attrmax   60        Protocol   UDP       Version    3
Readahd   1        Rsize     8192      Retry      3         Timeout    7
Wsize     8192
RPC       138      Null      0         Getattr    16        Setattr    0
Lookup    22       Read      37        Write      0         Create     0
Remove    0        Rename    0         Link       0         Readdir    8
Statfs    1        Mkdir     0         Rmdir      0         Symlink    0
Readlink  0        Access    38        Mknod      0         Readdir+   8
Fsstat    0        Fsinfo    0         Pathconf   0         Commit     0

Mount point = '/u/aixdir'
Type  Stat  Mounted
NFS   R/W  '/../NFS:AIX6000/home'
UID      9999     GID        8888      Userid   aixuserid
Translate          POSIX                LIST
Attrcach  YES      Attrmax   60        Protocol   TCP       Version    3
Readahd   1        Rsize     8192      Retry      3         Timeout    7
Wsize     8192
RPC       158      Null      0         Getattr    16        Setattr    0
Lookup    42       Read      37        Write      0         Create     0
Remove    0        Rename    0         Link       0         Readdir    8
Statfs    1        Mkdir     0         Rmdir      0         Symlink    0
```

```
Readlink   0          Access    38       Mknod      0         Readdir+   8
Fsstat     0          Fsinfo    0        Pathconf   0         Commit     0


Mount point = '/os2drive'
Type  Stat  Mounted
NFS   R/O  '/../NFS:OS2_SERV/d:'
UID       -2        GID       -2        Userid   ANONYMOUS
Translate           POSIX               NOLIST
Attrcach  NO        Attrmax   0         Protocol  UDP        Version    2
Readahd   0         Rsize     8192      Retry     3          Timeout    7
Wsize     8192
RPC       7         Null      0         Getattr   0          Setattr    0
Lookup    0         Read      0         Write     0          Create     0
Remove    0         Rename    0         Link      0          Readdir    0
Statfs    1         Mkdir     0         Rmdir     0          Symlink    0
Readlink  0         Access    4         Mknod     0          Readdir+   2
Fsstat    0         Fsinfo    0         Pathconf  0          Commit     0


Mount point = '/'
Type  Stat  Mounted
BFS   R/W  '/../VMBFS:SERVBFS:MARYELLN/'
Ready;
```

Usage Notes

1. If nothing is mounted and the STACK, LIFO, or FIFO option was specified, the return code is set to 6, indicating that no data was stacked.

2. The path names in the display are bounded by quotes because they can contain blanks. If a quote is actually part of the path name, it will be displayed as two quotes in a row.

3. The output of OPENVM QUERY MOUNT will be truncated to 255 characters if any of the STACK options are used and the response data exceeds 255 characters in length. Use CMS Pipelines for responses greater than 255 characters.

4. When the DETAILS option is specified, the display includes information about how the NFS client is known at the NFS server. This includes a *username* and UID and GID values.

   - When *username* is specified on a Mount or picked up from the NETRC DATA file, the UID and GID displayed are those returned by the NFS server in response to a Sun PC-NFS request. That UID and GID are returned to the NFS server in the UNIX-style credentials on subsequent requests.

     Up to nine characters of the *username* are displayed.

   - ANONYMOUS indicates that the NFS Mount request was done anonymously from the NFS client's point of view. That is, no Sun PC-NFS request was sent by the NFS client.

     Note that the *serveroptions* provided with the NFS path name on the Mount request may include user ID information, so that in the NFS server's view, the mount request is not anonymous.

     In this case, UID −2 and GID−2 are passed to the NFS server in the UNIX-style credentials.

   - When ✶✶✶✶✶✶✶✶✶ is displayed, it indicates that neither *username* nor ANONYMOUS were used on the NFS Mount request. The UID and GID passed to the NFS server in the UNIX-style credentials are the values in effect for your VM user ID. The current effective UID and GID are displayed in the command output.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM RENAME

```
►►── OPENVm ──── REName ──── old_pathname ──── new_pathname ──►◄
```

Authorization

General user; Byte file system (BFS) permission checking applies to this command.

Purpose

The OPENVM RENAME command renames or relocates a byte file system (BFS) object.

Operands

old_pathname
   specifies the name of the object to be renamed. See "Understanding Byte File System (BFS) Path Name Syntax" on page 368 for a description of the different forms of the BFS path name.

new_pathname
   specifies the new name of the object.

Usage Notes

1. When renaming a file, if the new name specified points to an existing file, the old file will be deleted (unlinked) and the file specified as old will be given the new path name.

2. When renaming a directory, if the new name refers to an existing directory, the existing directory must be empty.

3. If the *old_pathname* points to a file, the *new_pathname* cannot point to a directory and vice versa. In other words, you cannot replace a file with a directory or a directory with a file.

4. You must have write permission to the directory containing the old name and write permission to the directory containing the new name. If they both are directories, the caller does not need write permission to the object being renamed.

5. If *old_pathname* and *new_pathname* are links referring to the same file, no action is taken and the command completes successfully.

6. You cannot specify a new path name that physically resides in a different byte file system than the old path name. The path names (new and old) must be on the same byte file system.

7. For symbolic and external links, only the name of the link itself is changed; the contents are not changed.

8. The OPENVM commands can be entered on a single line or on multiple lines. To enter multiple lines, type OPENVM and press the enter key. You will get a message prompting you to enter more input lines. You must enter a null line to indicate the end of your command input. This is particularly useful for entering long path names.

9. When path names refer to files in NFS-mounted file systems, you must meet the authorization requirements imposed by the remote NFS servers.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS1162E	Directory is not empty: *pathname*	40
DMS2115E	Objects are on different file systems	88
DMS2121E	Operation may not be performed on {the file system root|. or ..}	88
DMS2124E	Path name is part of the new name for *pathname*	40

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM RUN
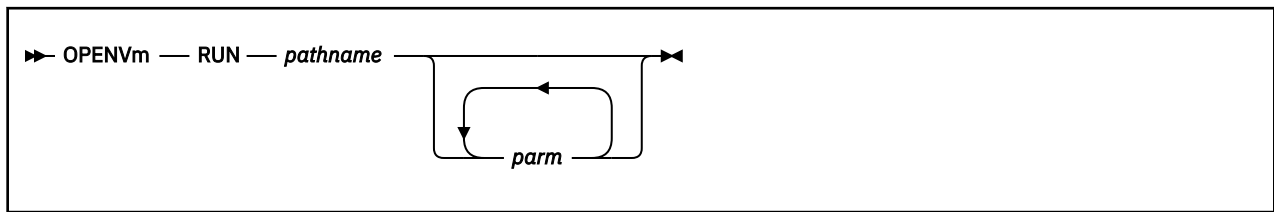
```
►►── OPENVm ── RUN ── pathname ─┬─────────────┬─►◄
                                │  ◄─────────┐ │
                                └──┬─ parm ──┴─┘
```

Authorization

General user; byte file system (BFS) permission checking applies to this command. The user must have execute permission for the file that is being invoked.

Purpose

The OPENVM RUN command starts an application that is an executable CMS module.

Operands

pathname
 is the path name of the CMS module to be executed.

parm
 are the parameters that the user wants to pass to the program. User parameters are tokenized (delimited by blanks) unless they are enclosed in quotes. Each parameter is passed to the program as a separate argument. Any number of parameters can be passed.

 A user parameter may contain special characters, such as single quotation marks ('), double quotation marks ("), blank spaces, and so on. However, you must follow the same rules that apply to these special characters when used in a BFS path name. Specifying **"** or **""** passes a NULL string as a parameter. See "Understanding Byte File System (BFS) Path Name Syntax" on page 368 for more information.

Usage Notes

1. The user must have execute permission to the file to run the program.
2. OPENVM RUN attempts to locate an executable file in the BFS. If one is not found, OPENVM RUN attempts to parse the path name into a CMS file ID and search for that file in the CMS record file system (SFS or minidisk). Case sensitivity of the path name is respected, even when it is interpreted as a CMS file ID.

   For more information about how CMS searches for the path name, see the BPX1EXC routine in *z/VM: OpenExtensions Callable Services Reference*.

   To specify a CMS record file system file ID for the path name parameter, the entire file ID must be enclosed in quotes if you specify a CMS file type and file mode because they are separated by blanks.
3. The OPENVM RUN command calls the BPX1SPN function, which creates a child process for running the specified application. BPX1SPN then calls BPX1EXC to run the specified application. OPENVM RUN waits for the child process to end. See *z/VM: OpenExtensions Callable Services Reference* for more information on BPX1SPN and BPX1EXC, including restrictions and entry conditions.
4. The OPENVM RUN command queries the CENV group of GLOBALV to obtain a list of variables. These variables are used to initialize the POSIX environment variable for the user program. If no LOGNAME variable is found in the CENV group of GLOBALV, it is set to the user's POSIX login name. If no PATH variable is found in the CENV group of GLOBALV, it is set to /bin:/usr/bin. The HOME environment variable is set to the initial working directory field in the POSIX user database entry. If HOME is

found in the CENV group of GLOBALV, it is overridden. The SHELL environment variable is set to the initial user program field in the POSIX user database entry. If SHELL is found in the CENV group of GLOBALV, it is overridden. Each variable string that is passed to the user program is terminated by a NULL character (X'00'). The NULL character is included in the string length that is passed to the user program.

5. The file name, which is the last path name component in the path specified for the command, is passed to the user program as the first argument. Any user arguments that are specified on the command are passed in the order entered, after the file name. All arguments have a NULL character appended, and this NULL character is included in the argument string length that is passed to the user program.

6. The OPENVM RUN command cannot be invoked while in DOS mode or subset mode.

7. The OPENVM RUN command opens the terminal as file descriptors 0, 1, and 2 if these are not already in use. If the user program completes and returns with a status of 0, but an error is encountered closing one of the files that were opened, the return code will reflect the close error.

8. The application that OPENVM RUN starts must be expecting BPX1EXC (exec()) style entry conditions. For example, if you create an external link to a module that expects a tokenized or extended parameter list, it cannot successfully use any of the parameters passed to it by OPENVM RUN, and it may even generate a program check while trying to look at the parameter list.

9. A null string can be passed as a parameter to the invoked program by specifying two quotes in a row. For example:

```
OPENVM RUN /bin/aprog parml '' parm3
```

10. When *pathname* refers to an object in an NFS-mounted file system, you must meet the authorization requirements imposed by the remote NFS server.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; For example:

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS132S	File *pathname* too large	104
DMS2105E	Permission is denied	28
DMS2113E	Object does not exist: *pathname*	28
DMS2117E	Object is not {a BFS regular file|in the proper format to be an executable file}: *pathname*	28
DMS2134E	Return code *retcode* and reason code *reascode* [X'*hexreascode*'] given on call to routine *routinename*	104

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

For additional messages that may be generated, see the LOADMOD command in *z/VM: CMS Commands and Utilities Reference*.

OPENVM RUN Return Codes

If the application is successfully started, the return code will be 0 if the exit status of the child process is 0 and OPENVM RUN did not encounter any other errors. However, if the exit status of the child process is **not** 0, the OPENVM RUN return code will be 1000 plus the return code field of the exit status. This field contains a value from 0 to 255. It is often the return code of the exiting process. If the return code is 1000, it is most likely because the application was terminated by a signal.

OPENVM SET DIRECTORY

```
▶▶─ OPENVm ─── SET ─── DIRectory ─── pathname ─▶◀
```

Authorization

General User

Purpose

Use the OPENVM SET DIRECTORY command to set or change your working directory from the current one to a new one. The current working directory is the starting point for path searches of path names not beginning with a '/'.

Operands

pathname
> is the byte file system (BFS) path name that is to be used as your new current working directory.

> **Note:** This *pathname* may use the current working directory that is in effect prior to successful completion of this command.

> See "Understanding Byte File System (BFS) Path Name Syntax" on page 368 for a description of the different forms of the BFS path name.

Usage Notes

1. Use the POSIXINFO FSROOT statement in your CP directory entry, or use the OPENVM MOUNT command to establish your root.

   You may use the POSIXINFO IWDIR statement in your CP directory entry to establish an initial working directory.

   For information about the z/VM CP directory entry, see *z/VM: CP Planning and Administration*.

2. OPENVM SET DIRECTORY sets the default working directory for the current process and for any new process that is created until your virtual machine is IPL'ed or until another OPENVM SET DIRECTORY is entered.

3. You may use any form of relative BFS path name when changing your working directory. For the following examples, assume you have a BFS that included the directories displayed in Figure 4 on page 450 and you had entered:

   ```
   OPENVM MOUNT /../VMBFS:VMSYS:ROOT/TRAVEL   /
   ```

   to establish your root directory.
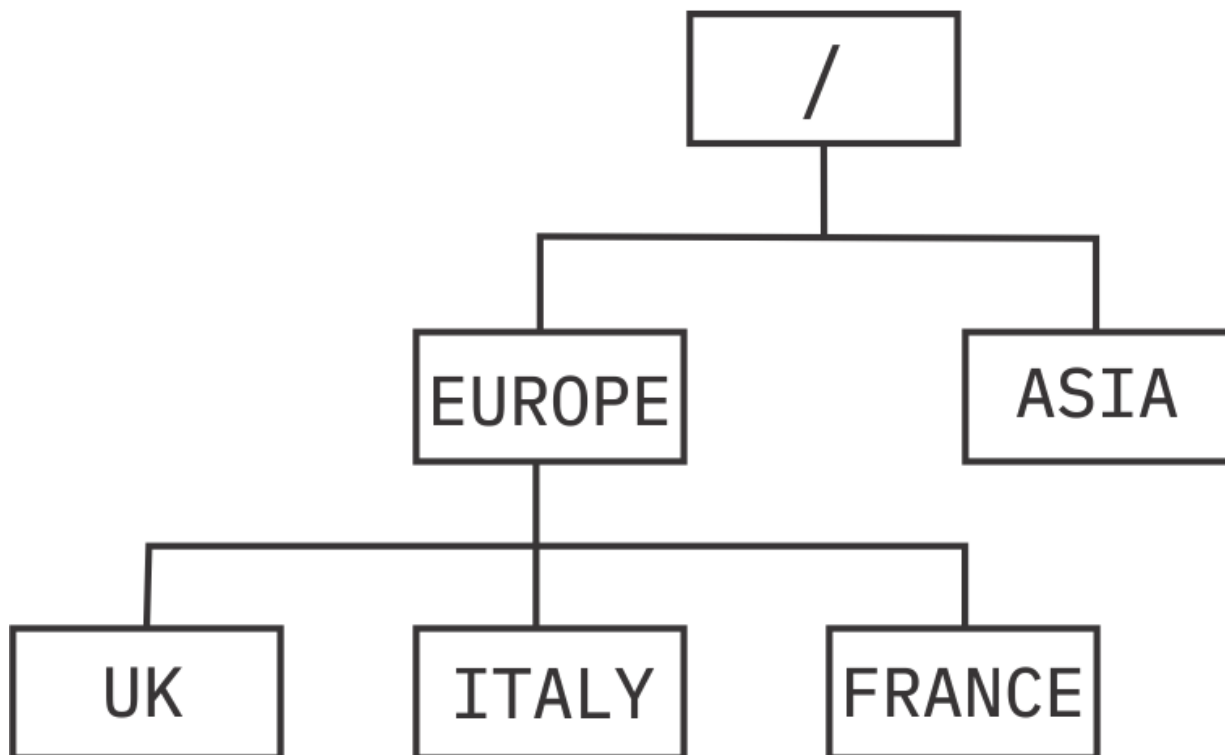
Figure 4. Sample BFS directory hierarchy

If you did not specifically designate a working directory, your current working directory would be the same as your root directory (in this case, the `'/../VMBFS:VMSYS:ROOT/TRAVEL'` directory).

  a. If you wished to set your working directory to the UK directory, you could do so by entering:

```
OPENVM SET DIRECTORY   EUROPE/UK
```

  b. Suppose you then wanted to set set your working directory to EUROPE; you could do so by entering:

```
OPENVM SET DIRECTORY   '..'
```

  c. From here, if you wanted to set your working directory to ITALY, you could do so by entering:

```
OPENVM SET DIRECTORY   ITALY
```

  d. You could go from your ITALY directory to your FRANCE directory by entering:

```
OPENVM SET DIRECTORY   '../FRANCE'
```

4. The establishment of a new root (using the OPENVM MOUNT command) affects the resolution of the current working directory.

For example, if you entered these commands:

```
OPENVM MOUNT /../VMBFS:FILEPL8:BYTEFS/ /
OPENVM SET DIRECTORY '/My_department/Reports'
```

When you referred to 'file-a', you would really be referring to:

```
/../VMBFS:FILEPL8:BYTEFS/My_department/Reports/file-a
```

If you then entered:

```
OPENVM MOUNT /../VMBFS:FILEPL8:DIFFBFS/ /
```

When you referred to 'file-a', you would be referring to:

```
/../VMBFS:FILEPL8:DIFFBFS/My_department/Reports/file-a
```

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM SET FORK

```
►►── OPENVm ── SET ── FORk ──┬── ON ──┬──►◄
                            └── OFF ──┘
```
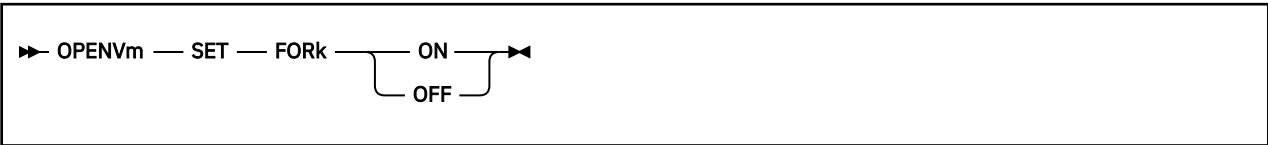
Authorization

General User

Purpose

Use the OPENVM SET FORK command to indicate whether calls to the fork (BPX1FRK) callable service should be processed. z/VM invokes fork (BPX1FRK) to handle fork() function calls in a C or C++ program running on z/VM. Calls to fork (BPX1FRK) may also be coded directly in an assembler or REXX program.

The OpenExtensions implementation of the fork (BPX1FRK) service has some limitations not found in other implementations. In certain situations, you may need to modify your application to accommodate these limitations. To understand the OpenExtensions implementation of fork (BPX1FRK), see *z/VM: OpenExtensions Callable Services Reference*. To avoid the limitations of fork (BPX1FRK), you should consider modifying your application to use spawn (BPX1SPN). If you determine that the processing provided by fork (BPX1FRK) is sufficient for your needs, you must use the OPENVM SET FORK command to explicitly turn that processing ON before running your program.

Operands

ON
specifies that calls to the fork (BPX1FRK) service are to be processed.

OFF
specifies that calls to the fork (BPX1FRK) service are not to be processed.

Usage Notes

1. The initial (default) setting for fork (BPX1FRK) processing in a CMS session is OFF.
2. If fork (BPX1FRK) processing is set OFF, an indirect or direct call to the fork (BPX1FRK) service is not processed, except to return a return value of -1 and a return code and reason code indicating that the call is not supported.

Messages and Return Codes

There are no error messages issued by this command. However, system messages may be issued. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

OPENVM SET MASK

```
►►── OPENVm ── SET ── MASk ─┬─── CLEAR ───┬─►◄
                            │             │
                            ├─ mode_string ─┤
                            │             │
                            └─── RESET ───┘
```
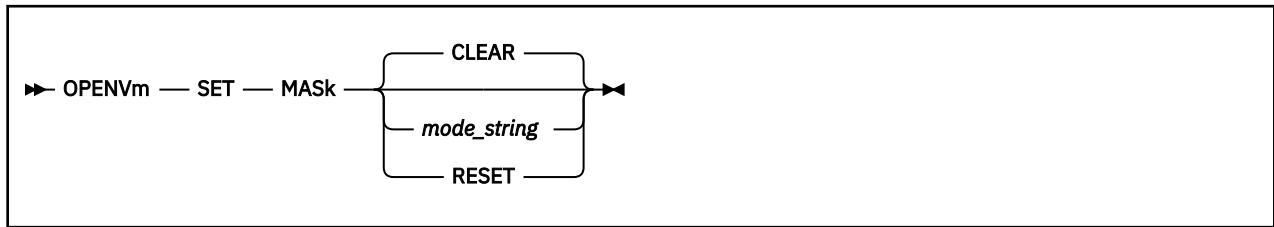
Authorization

General User

Purpose

Use the OPENVM SET MASK command to change the file creation mask (often called the **umask**). The file creation mask is used, along with the mode specified by an application, to determine the access permissions (read, write, and execute) to any new file or directory created by an OPENVM or shell command.

By specifying permissions with the OPENVM SET MASK command, you can control who will be denied access to a file you create. You cannot guarantee that an application will allow a permission that you have set with the OPENVM SET MASK command, but an application cannot grant a permission that you have denied.

Operands

CLEAR
> turns off all permissions (read, write, and execute) for all users (owner, group, and public) for any new file being created. This is the default.

mode_string
> is the set of permissions that you are granting for any new file being created. These permissions are specified in the following format:

```
rwx rwx rwx
```

> From left to right, the first three characters apply to the owner, the second three characters to the group, and the third three characters to public. Each group of characters is mapped as follows:

> **r**
>> Indicates read permission is to be allowed if requested by an application creating a new file.

> **w**
>> Indicates write permission is to be allowed if requested by an application creating a new file.

> **x**
>> Indicates execute (search) permission is to be allowed if requested by an application creating a new file.

> **-**
>> Indicates that permission is to be denied. When in the first position (instead of **r**), this indicates read permission is to be denied. When in the second position (instead of **w**), this indicates write permission is to be denied. When in the third position (instead of **x**), this indicates execute (search) permission is to be denied.

RESET

resets the file creation mask to the initial system default. In OpenExtensions this is:

```
OWNER        GROUP        PUBLIC
 rwx          r-x          r-x
```

This setting denies write access to group and public.

Usage Notes

1. The mask set by OPENVM SET MASK controls permission to newly-created BFS objects. A permission value specified for this command is granted for a file being created only if the corresponding mode is specified by the application creating the file.

   For example, if you wanted to deny execute access to users other than the file owner as a default when creating new files in the BFS, you would specify:

   ```
   OPENVM SET MASK rwx r-- r--
   ```

   If an application subsequently issues **mkdir()** or **open()** with permissions specified as `rwx rwx --x`, the actual permissions will be `rwx r-- r--`.

   For more information, see "Handling Security for Your Files" in *z/VM: OpenExtensions User's Guide*.

2. If permissions are specified on the OPENVM SET MASK command, the corresponding bit will be turned ON in the **umask**. All other bits in the **umask** are turned OFF.

3. The OPENVM SET MASK command sets the default file mode creation mask for the current process, and for any new process that is created, until you IPL your virtual machine or another OPENVM SET MASK command is entered.

4. If the OPENVM SET MASK command has not been entered, the system default permissions will be given to newly created BFS objects unless a mode has been supplied that denies these permissions:

   ```
   OWNER        GROUP        PUBLIC
    rwx          r-x          r-x
   ```

5. You can clear the file creation mask (no permission will be granted by default) by entering:

   ```
   OPENVM SET MASK
   ```

   with no operands, or

   ```
   OPENVM SET MASK CLEAR
   ```

6. You can reset the file creation mask to the system default by entering:

   ```
   OPENVM SET MASK RESET
   ```

7. Use OPENVM QUERY MASK to determine the current value of the file creation mask.

8. Another example showing how the OPENVM SET MASK command can be used follows:

   ```
   openvm set mask rwx r-x r-x
   Ready;
   openvm query mask
   Owner     Group      Public
    rwx        r-x         r-x
   Ready;
   openvm create dir Z
   Ready;
   openvm list (own
   Directory = '/'
   User ID   Group Name  Permissions Type  Path name component
   user1000  CMSUSRS      rwx r-x r-x  D     'Z'
   Ready;
   ```

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM SHELL

```
►►── OPENvm ── SHell ──────────────────────────────────►◄
                    └─ parm1 ─┐
                       └─ parm2 ──────────┐
                          └─ ... ─┘
```

Authorization

General user; byte file system (BFS) permission checking applies to this command. The user must have execute permission for the file that is invoked as the shell.

Purpose

Use the OPENVM SHELL command to start an OpenExtensions shell and enter the shell command environment.

Operands

parm1, parm2, ...
    are the parameters that the user wants to pass to the program. User parameters are tokenized (delimited by blanks). Each parameter is passed to the program as a separate argument. Any number of parameters can be passed.

    A user parameter may contain special characters, such as single quotation marks ('), double quotation marks ("), blank spaces, and so on. However, you must follow the same rules that apply to these special characters when used in a BFS path name. Specifying **"** or **""** passes a NULL string as a parameter. See "Understanding Byte File System (BFS) Path Name Syntax" on page 368 for more information.

Usage Notes

1. The OPENVM SHELL command invokes the initial user program, as defined in the POSIX user database entry as a login shell in POSIX compliant mode. If no initial user program is defined in the POSIX user database entry, the default OpenExtensions shell, `/bin/sh`, will be invoked.

   For information about the POSIX user database, see *z/VM: CP Planning and Administration*.

2. When the OPENVM SHELL command starts an OpenExtensions shell, it starts the shell with OPENVM RUN. For more information, see "OPENVM RUN" on page 446.

3. OPENVM SHELL attempts to GLOBAL the LOADLIBs that are needed to run the shell. This list of LOADLIBs is defined in the file `/etc/openvmdefaults` on lines that begin with the keyword CLIBNAMES. The keyword CLIBNAMES must be in upper case, and lines in the file are delimited by the newline character (X'15'). There may be multiple lines with the CLIBNAMES keyword, and multiple LOADLIB names can be listed on a single line after the CLIBNAMES keyword. If OPENVM SHELL cannot read the `/etc/openvmdefaults` file, or if no CLIBNAMES are defined, it uses SCEERUN as the default loadlib.

   Before trying to GLOBAL the LOADLIBs, OPENVM SHELL looks for each one on the currently accessed file modes. If any of the LOADLIBs are not found, OPENVM SHELL looks in the `/etc/openvmdefaults` file for lines that begin with the keyword CLINKNAME. The CLINKNAME keyword must be in upper case, and the data that follows the keyword must be one of the following:

   • A nickname defined in a CMS NAMES file by a :NICK tag

- A *userid* and *owner_vdev* pair that identifies the user ID of the owner of a minidisk and the virtual device number as defined in the owner's user directory entry
- .DIR *dirname*.

OPENVM SHELL will issue VMLINK for each CLINKNAME line in the file.

OPENVM SHELL will issue the GLOBAL command only if any of the specified LOADLIBs are not already GLOBALed. It appends the list of missing LOADLIBs to the list of LOADLIBs currently GLOBALed. Before OPENVM SHELL completes, it restores the list of LOADLIBs that have been GLOBALed.

4. OPENVM SHELL sets the value of the _EDC_KEEP_EMSG variable in the CENV group of GLOBALV to Y so that C will not set EMSG OFF.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

For additional messages that may be generated, see the LOADMOD command in *z/VM: CMS Commands and Utilities Reference*.

For additional information on return codes, see "OPENVM RUN Return Codes" on page 448.

OPENVM SHOWMOUNT

```
▶▶── OPENVm ── SHOwmount ──┬──────────────┬──┬──────────────────┬──▶◀
                           └─ foreign_host ─┘  │  ┌── Export ──┐  │
                                               └─(─┼── Active ──┼─┬─)─┘
                                                   └─ Directory ─┘
```

Authorization

General user

Purpose

Use the OPENVM SHOWMOUNT command to display information about mountable or mounted file systems at *foreign_host*.

Operands

foreign_host
> Identifies the host for which mount information is to be displayed. If not specified, the local host is the default.

Options

Export
> Displays *foreign_host's* export list and the list of clients allowed to mount each one. This is typically the list of file systems that can be mounted. VM's NFS server allows mounting of file systems that are not in the export list.

Active
> Shows the list of file systems mounted at *foreign_host* and the foreign host name or IP address of the client that requested the mount.

Directory
> Shows the list of file systems mounted at *foreign_host*.

Responses

Following is sample output for the EXPORT option where the remote host is on a z/VM system. For a z/VM NFS server, the display will always show `everyone` as the list of clients allowed to mount each exported file system. (The NFS server for z/VM determines whether a mount is allowed based on whether the user ID associated with the mount is authorized or permitted to use the file system. In the case of a password-protected minidisk, the password provided on the mount determines whether the mount is allowed.)

```
openvm showmount vmthere (export
/PC/Your/191/Disk
   (everyone)
/PC/Your/SFS/in/VMSYS1
   (everyone)
/PC/Your/SFS/in/VMSYS2
   (everyone)
/UNIX/Your/191/Disk
   (everyone)
/UNIX/Your/SFS/in/VMSYS1
   (everyone)
```

```
/UNIX/Your/SFS/in/VMSYS2
  (everyone)
/BFS/FSROOT/IWDIR/in/VMSYS1
  (everyone)
/BFS/FSROOT/IWDIR/in/VMSYS2
  (everyone)
Ready;
```

The following is sample output for the EXPORT option where the remote host is on an AIX system. In addition to the exported file system, the display includes a list of which clients are allowed to mount each one. The names in the group list are implementation-specific.

```
openvm showmount aixthere (export
/home/books
  user1
  user2
/cd1
  (everyone)
/home/user1
  user1
Ready;
```

If the foreign_host does not have any file systems exported, the following response is returned:

```
No exported file systems for host foreign_host
```

```
openvm showmount vmthere (active
5.55.12.121/MARYSMIT.191
5.55.12.12//../VMBFS:VMSYS2:ROOT/
5.55.12.12/VMSYS1:JOHNSMIT.
5.55.12.12/VMSYS2:MARYSMIT.
Ready;
```

If the foreign_host does not have any file systems mounted, the following response is returned:

```
Nothing is mounted
```

An "*" will be displayed in the output if the foreign host returns a null hostname or file system name.

```
openvm showmount vmthere (directory
MARYSMIT.191
/../VMBFS:VMSYS2:ROOT/
VMSYS1:JOHNSMIT.
VMSYS2:MARYSMIT.
Ready;
```

If the foreign_host does not have an file systems mounted, the following response is returned:

```
Nothing is mounted
```

An "*" will be displayed in the output if the foreign_host returns a null file system name.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS002E	File STANDARD TCXPXLBIN * not found	28
DMS065E	*option* specified twice	24
DMS1060E	MOUNT [DUMP | EXPORT] program is not available at *foreign_host*	99

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

OPENVM UNMOUNT

```
▶▶─ OPENVm ── UNMount ──┬─────── / ───────┬─▶◀
                        └─ pathname ─┘
```

Authorization

General User

Purpose

OPENVM UNMOUNT removes a file system from your hierarchy.

The file system to be removed may be a byte file system (BFS), BFS subdirectory, or Network File System (NFS) previously mounted with OPENVM MOUNT. Use the OPENVM QUERY MOUNT command to display what is mounted in your hierarchy.

Operands

pathname
> is the BFS path name from which the BFS or BFS subdirectory tree is to be removed. See "Understanding Byte File System (BFS) Path Name Syntax" on page 368 for a description of the different forms of the BFS path name.
>
> *pathname* may also be an NFS file system name that was used on a mount request. See "Understanding Network File System (NFS) Path Name Syntax" on page 374 for a description of the NFS path name.

/
> indicates that the root directory is to be unmounted.

Usage Notes

1. OPENVM UNMOUNT will unmount the file system and all file systems mounted on it. It will unmount everything, even if there are open files.

2. An OPENVM UNMOUNT affects all processes in the virtual machine.

3. After an OPENVM UNMOUNT *pathname* operation, you will again be able to see the BFS subdirectory tree that was overlaid by the original OPENVM MOUNT.

4. If you had entered an OPENVM SET DIRECTORY command to set your current working directory, it will **not** be changed by OPENVM UNMOUNT. It will remain in effect until you IPL your virtual machine or enter another OPENVM SET DIRECTORY command.

5. A mount is no longer in effect for your CMS virtual machine following a system reset, such as LOGOFF or IPL. However, the foreign server may still consider the mount active. Use the NFS format of the path name on OPENVM UNMOUNT to clean up mounts at a foreign server following a LOGOFF or IPL.

Messages and Return Codes

For information on a specific error message, see *z/VM: CMS and REXX/VM Messages and Codes*. You can also enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS111E
```

Number	Text	Return Code
DMS2110E	Object is not a directory: *pathname*	40
DMS2113E	File system is not mounted or not available	28
DMS2113E	Object does not exist: *pathname*	28
DMS2118E	Path name is not the root of a file system or path name is the root of a file system but it is not mounted	40
DMS2127W	Nothing is mounted	4

Additional system messages may be issued by this command. The reasons for these messages and their location are:

Reason	Location
Errors in command syntax	See "Command Syntax Error Messages" in *z/VM: CMS Commands and Utilities Reference*.
Errors in using the BFS	See Appendix E, "Common Error Messages When Using BFS Files," on page 545

Appendix A. OpenExtensions Command Summary

The following list presents OpenExtensions shell commands and utilities grouped by the task a user might want to perform. Similar tasks are organized together. Commands that are OpenExtensions extensions to POSIX.2 are indicated with an "OE".

Shell Command Summary

General Use

cms
Allow any CP or CMS command to be run from the shell.

command
Run a simple command

date
Display the date and time

echo
Write arguments to standard output

print
Return arguments from the shell

printf
Write formatted output

sh
Invoke a shell (command interpreter)

Note: Use OPENVM SHELL to invoke the OpenExtensions shell initially.

su
Start a new shell and change your authorization to superuser

time
Display processor and elapsed times for a command

whence
Tell how the shell interprets a command name

Controlling Your Environment

alias
Display or create a command alias

env
Display environments, or set an environment for a process

export
Set the export attributes for variables, or show currently exported variables

fc
Process a command history list

id
Return the user identity

locale
Get locale-specific information

logger
Log messages

logname
Return a user's login name

newgrp
Change to a new group

readonly
Mark a variable as read-only

return
Return from a shell function or . (dot) script

set
Set or unset command options and positional parameters

shift
Shift positional parameters

stty
Set or display terminal options

touch
Change the file access and modification times

tty
Return the user's terminal name

unalias
Remove alias definitions

uname
Display the name of the current operating system

unset
Unset values and attributes of variables and functions

Managing Directories

basename
Return the nondirectory components of a path name

cd
Change the working directory

dirname
Return the directory components of a path name

ls
List file and directory names and attributes

mkdir
Make a directory

mv
Rename or move a file or directory

pathchk
Check a path name

pwd
Return the working directory name

rm
Remove a directory entry

rmdir
Remove a directory

Managing Files

cat
Concatenate or display a text file

chgrp
Change the group owner of a file or directory

chmod
Change the mode of a group or directory

chown
Change the owner or group of a file or directory

cksum
Calculate and write checksums and byte counts

cmp
Compare two files

comm
Show and select or reject lines common to two files

compress
Compress data in a file or from the standard input

cp
Copy a file

cut
Cut out selected fields of each line of a file

dd
Convert and copy a file

diff
Compare two text files and show the differences

ed
Use the **ed** line-oriented text editor

find
Find a file meeting specified criteria

fold
Break lines into shorter lines

head
Display the first part of a file

iconv
Convert characters from one code set to another

join
Join two sorted, textual relational databases

ln
Create a link to a file

mkfifo
Make a FIFO special file

mknod OE
Make a FIFO or character special file

od
Dump a file in a specified format

paste
Merge corresponding or subsequent lines of a file

sed
Start the **sed** noninteractive stream editor

sort
    Start the sort-merge utility

tail
    Display the last part of a file

tee
    Duplicate the output stream

tr
    Translate characters

umask
    Set or return the file mode creation mask

uncompress
    Uncompress data in a file or from the standard input

uniq
    Report or filter out repeated lines in a file

wc
    Count newlines, words, and bytes

zcat
    Uncompress data in one or more files or from the standard input

Printing Files

lp
    Send a file to a printer

pr
    Format a file in paginated form and send it to standard output

Computing and Managing Logic

bc
    Use the arbitrary-precision arithmetic calculation language

break
    Exit from a for, while, or until loop in a shell script

colon or :
    Do nothing, successfully

continue
    Skip to the next iteration of a loop in a shell script

dot or .
    Run a shell file in the current environment

eval
    Construct a command by concatenating arguments

exec
    Run a command and open, close, or copy the file descriptors

exit
    Return to the parent process from which the shell was called or to CMS

expr
    Evaluate arguments as an expression

false
    Return a nonzero exit code

grep
    Search a file for a specified pattern

let
  Evaluate an arithmetic expression

test
  Test for a condition

trap
  Intercept abnormal conditions and interrupts

true
  Return a value of 0

Controlling Processes

bg
  Move a job to the background

fg
  Bring a job into the foreground

jobs
  Return the status of jobs in the current session

kill
  End a process or job, or send it a signal

nohup
  Start a process that is immune to hangups

ps
  Return the status of a process

sleep
  Suspend execution of a process for an interval of time

time
  Display processor and elapsed times for a command

wait
  Wait for a child process to end

Writing Shell Scripts

getconf
  Get configuration values

getopts
  Parse utility options

read
  Read a line from standard input

type
  Tell how the shell interprets a name

typeset
  Assign attributes and values to variables

xargs
  Construct an argument list and run a command

Developing or Porting Application Programs

ar
  Create or maintain library archives

awk
  Process programs written in the **awk** language

c89/cxx
    Compile C or C++ source code and create an executable file

lex
    Generate a program for lexical tasks

make
    Maintain program-generated and interdependent files

strip
    Remove unnecessary information from an executable file

yacc
    Use the **yacc** compiler

Communicating with the System or Other Users

mailx
    Send or receive electronic mail

Working with Archives

cpio
    Copy in/out file archives

pax
    Interchange portable archives

tar
    Manipulate the **tar** archive files to copy or back up a file

Shell and CMS Commands that Work with Directories and Files

You can use OPENVM commands to do certain tasks with the Byte File System. Some of these are tasks that UNIX users traditionally perform while in the shell. Because these are CMS commands, you can perform these Byte File System tasks whether or not you have the OpenExtensions Shell and Utilities installed.

Table 13 on page 468 describes the OPENVM command and the equivalent shell command.

Table 13. CMS and Shell Command Equivalents

CMS	Shell	Function
OPENVM CREATE DIRECTORY	**mkdir**	Create a directory. The mkdir command has an option for creating intermediate directories in a pathname.
OPENVM CREATE EXTLINK	-	Create a BFS pathname to be used to reference a file or other data that resides outside of the BFS.
OPENVM CREATE LINK	**ln**	Link another name to a file (in addition to its original name).
OPENVM CREATE SYMLINK	**ln**	Create a BFS pathname to be used to reference an object residing in one BFS using a pathname in the same or another BFS.
OPENVM ERASE	**rm**	Remove (erase, or delete) a file from a directory.
	rmdir	Remove (erase, or delete) a directory that is empty of files.
OPENVM GETBFS	**cp**	Copy a file.
OPEMVM LISTFILE	**ls**	List the files in a directory.

Table 13. CMS and Shell Command Equivalents (continued)

CMS	Shell	Function
OPENVM MOUNT	-	Add a mountable BFS, or BFS sub-directory tree, or Network File System (NFS), to the file hierarchy.
OPENVM OWNER	**chgrp**	Change the group owner of a file or directory. To use this command, you must be a superuser or the owner of the file or directory.
	chown	Change the owner or group of a file or directory. To use this command, you must be a superuser.
OPENVM PARCHIVE	-	Process archive tapes.
OPENVM PERMIT	**chmod**	Change access permission to a directory or file. To use this command, you must have appropriate privileges—you must have write authority, or be the file owner, or be a superuser.
OPENVM PUTBFS	**cp**	Copy a file.
OPENVM QUERY LINK	-	Display information associated with symbolic or external links.
OPENVM QUERY MASK	**umask**	Display the file creation mask values in effect.
OPENVM QUERY MOUNT	-	Display what is mounted in your hierarchy.
OPENVM RENAME	**mv**	Move a file from one directory to another directory, or rename a file or directory.
OPENVM QUERY DIRECTORY	**pwd**	Display your working directory.
OPENVM RUN	-	Execute an application that is an executable CMS module.
OPENVM SET DIRECTORY	**cd**	Change a working directory.
OPENVM SET MASK	**umask**	Define the file creation mask to be used when creating a BFS object.
OPENVM UNMOUNT	-	Remove a previously mounted BFS or BFS subdirectory tree from your hierarchy.
XEDIT	**ed**	Create or edit text in a file.

Appendix B. Regular Expressions (regexp)

Many OpenExtensions shell commands use a type of pattern known as a *regular expression* to select lines from a file for processing. A regular expression is a formula for evaluating whether a given line of a file should be selected. If some string within the line satisfies the formula given by the regular expression, then the line is selected and processed, otherwise the line is skipped.

A regular expression is written in terms of literals that must be present -- such as a, b, or `fish` -- and certain functions that can be performed on these literals, such as repeating them one or more times. The functions are expressed by special characters, called *metacharacters*, that appear in the regular expression. This appendix gives the rules for composing a regular expression and defines the sets of recognized metacharacters and their meanings.

A regular expression is categorized as *basic* or *extended* according to the set of metacharacters it uses. The shell commands **expr** and **ed** accept only basic regular expressions. The shell commands **grep** and **sed** usually accept basic regular expressions, but will accept extended regular expressions if the **-E** option is used. All other shell commands accept extended regular expressions.

The following variables are used in the definitions of the forms of regular expressions:

metachar

Any element of the set of metacharacters for a regular expression. There are two sets of metacharacters, one set for basic regular expressions and one set for extended regular expressions.

The basic regular expression metacharacters are:

```
.
 ^
 $    [
 \    *   \{   \}   \(   \)
 .    [   $    [   \    *   \{   \}   \(   \)
```

The extended regular expression metacharacters are:

```
.
 ^
 $  [
 \  *  +
 ?  {  }  |  (  )
```

The utilities that use regular expressions interpret them according to the Latin 1/Open System Interconnection Code Page 1047. See the appendix of *z/VM: OpenExtensions User's Guide* to ensure your terminal or emulator is generating the correct code points for these characters.

char

Any character which is not an element of the metacharacter set for the type of regular expression under consideration. For example, when we are discussing basic regular expressions, **|** is a *char*; when we are discussing extended regular expressions, **|** is a *metachar*.

digit

Any of the characters **1, 2, 3, 4, 5, 6, 7, 8,** or **9**.

anychar

Any character.

Given these definitions, we can begin to build up a list of the acceptable forms of a regular expression. We can say that a regular expression *regexp* takes these forms:

char

Matches one occurrence of *char*. For example, regular expression **a** specifies that any line containing the string a should be selected.

.

Matches one occurrence of any character. For example, regular expression **.** specifies that any line containing at least one character should be selected.

anychar

Matches one occurrence of *anychar*. In other words, **** is an "escape" character that permits searching for metacharacters. For example, regular expression **\\.** specifies that any line containing . should be selected, and regular expression **** specifies that any line containing \ should be selected.

Note: *****digit* is a special case and is described later.

[*bracket_expression*]

A string enclosed in square brackets matches any one character in the string. [1] For example, regular expression **[abc]** matches a, b, or c.

Within *bracket_expression*, certain characters have special meanings, as follows: [2]

- If the first character of *bracket_expression* is either a dash (**-**) or a closing square bracket (**]**), then it is interpreted literally rather than being given special meaning. For example, regular expression **[-abc]** matches any line containing -, a, b, or c.

- Within *bracket_expression* you can specify a *collation sequence* by enclosing the name of the collation sequence within square brackets and periods. For example, regular expression **[[.ch.]]** matches the multicharacter collation sequence ch (if the current language supports that collation sequence). Any single character is itself. Do not give a collation sequence that is not part of the current locale.

- Within *bracket_expression* you can specify an *equivalence class* by enclosing a character or collation sequence within square brackets and equal signs. For example, regular expression **[[=a=]]** matches any character in the same equivalence class as a. This normally expands to all the variants of a in the current locale; for example, a, \(a:, \(a', and so on. On some locales it might include both the uppercase and lowercase of a given character. In the POSIX locale, this always expands to only the character given.

- Within *bracket_expression* you can specify a *character class expression* by enclosing its name within square brackets and colons. These constructs are used for internationalization and handle the different collation sequences as required by POSIX. The following character class expressions are supported:

  [:alpha:]
    Any alphabetic character.

  [:lower:]
    Any lowercase alphabetic character.

  [:upper:]
    Any uppercase alphabetic character.

  [:digit:]
    Any digit character.

  [:alnum:]
    Any alphanumeric character (alphabetic or digit).

  [:space:]
    Any white-space character (blank, horizontal tab, vertical tab).

  [:graph:]
    Any printable character, except the blank character.

  [:print:]
    Any printable character, including the blank character.

---

[1] Bracket expressions are used not only in regular expressions, but also in pattern matching as performed by the **fnmatch()** function (used in file name expansion).

[2] **** does not serve as an escape character inside a bracket expression.

[:punct:]
Any printable character that is not white space or alphanumeric.

[:cntrl:]
Any nonprintable character.

For example, regular expression **[[:lower:]]** matches any lower case alphabetic character.

- Character ranges are specified by a dash (**–**) between two characters or collation sequences. This indicates all character or collation sequences that collate between two characters or collation sequences. It does not refer to the native character set. For example, in the POSIX locale, regular expression **[a-z]** means all the lowercase alphabetics, even if they don't agree with the binary machine ordering. However, since many other locales do not collate in this manner, use of ranges is not recommended, and they are not used in strictly conforming POSIX.2 applications.

  An end-point of a range may explicitly be a collation sequence; for example, regular expression **[[.ch.]-[.ll.]]** is valid. However, equivalence classes or character classes are not: regular expression **[[=a=]-z]** is not permitted.

- Once you have composed *bracket_expression*, you can invert its meaning by prefixing it with a circumflex (**^**). [3] For example, regular expression **[^abc]** matches any line containing neither a, b, nor c.

Practical, useful regular expressions are built by combining several smaller, simpler regular expressions along with certain special operators. Recognizing the recursion inherent in this situation, we can add to our list the following permitted forms for *regexp*:

concatenation of *regexp_1* and *regexp_2* with no intervening blank
When regular expressions *regexp_1* and *regexp_2* are concatenated, the line must match *regexp_1* and *regexp_2* in succession in order to be selected. For example, we can concatenate regular expressions **a** and **b** to form regular expression **ab**; regular expression **ab** matches any line containing the string ab.

^*regexp*
Specifies that the string matching *regexp* must appear at the beginning of the line. For example, regular expression **^A** matches the letter A at the beginning of a line. The **^** character is special only at the beginning of a regular expression or after **(** or **|**.

***regexp*$**
Specifies that the string matching *regexp* must appear at the end of the line. For example, regular expression **c$** matches line abc but does not match line cba.

***regexp*\{*n*\} (basic) or *regexp*{*n*} (extended)**
A number enclosed in braces represents a number of repetitions of *regexp*. For example, regular expression **X{3}** is equivalent to regular expression **XXX**, and both of these match string XXX.

***regexp*\{*min*,\} (basic) or *regexp*{*min*,} (extended)**
When followed by a comma, a number enclosed in braces represents a minimum number of repetitions of a regular expression. For example, regular expression **X{3,}** represents at least three repetitions of regular expression **X**.

***regexp*\{*min*,*max*\} (basic) or *regexp*{*min*,*max*} (extended)**
When a regular expression is followed by a pair of numbers in braces, the numbers represent a minimum and maximum number of repetitions respectively. For example, regular expression **X{3,7}** stands for three to seven repetitions of regular expression **X**.

regexp*
A regular expression *regexp* followed by ***** matches a string of zero or more strings that would match *regexp*. For example, regular expression **A*** matches A, AA, AAA and so on. It also matches the null string (zero occurrences of A).

***regexp*+ (extended only)**
A regular expression *regexp* followed by **+** matches a string of one or more strings that would match *regexp*. For example, regular expression **a+** matches a, aa, and so on.

---

[3] In **fnmatch()**, the complement character is the exclamation mark (**!**) rather than the circumflex.

*regexp***? (extended only)**
> A regular expression *regexp* followed by **?** matches a string of zero or one occurrences of strings that would match *regexp*.

*regexp_1***|***regexp_2* **(extended only)**
> This regular expression matches a string that would match either regular expression *regexp_1* or *regexp_2*.

\(*regexp***\) (basic) or (***regexp***) (extended)**
> Parentheses let you group parts of regular expressions. This is useful for influencing the order of evaluation of the regular expression, just as parentheses are used to influence the order of evaluation of terms in a mathematical equation. For example, regular expression **(ab){2}** matches string *abab*, but regular expression **ab{2}** matches string *abb*.

digit
> This pattern is equivalent to the string matching the *digit*th expression enclosed within parentheses found at an *earlier point* in the regular expression. Parenthesized expressions are numbered by counting **(** characters from the left.

> Constructs of this form can be used in the replacement strings of substitution commands (for example, the sub function of **awk**) to stand for constructs matched by parts of the regular expression.

Because a regular expression is a kind of mathematical formula for expressing a matching pattern, it's important to keep in mind that the operators used in regular expressions -- that is, the metacharacters -- do have an order of precedence associated with them. The order of precedence for operators in regular expressions is:

1. parenthetical groupings
2. bracket expressions
3. *
4. ?
5. +
6. brace expressions
7. concatenation
8. |

For example,

- **ab+ = (a)(b+)**, *not* **(ab)+**
- **abc* = (ab)(c*)**, *not* **(abc)***
- **abc|def+ = (abc)|((de)(f+))**

Finally, the newline character at the end of each input line is never explicitly matched by any regular expression or part thereof. In other words, you can't match a string that extends over multiple lines.

Summary: The commands that use basic and extended regular expressions are as follows:

Basic
> **ed**, **expr**, **grep**, **sed**

Extended
> **awk**, **grep** with **-E** option, **sed** with the **-E** option.

Table 14 on page 474 summarizes how regular expression features apply to shell commands.

Table 14. Regular Expression Features

Notation	awk	ed	grep -E	expr	sed
.	X	X	X	X	X
^	X	X	X		X

Table 14. Regular Expression Features (continued)

Notation	awk	ed	grep -E	expr	sed
$	X	X	X	X	X
[...]	X	X	X	X	X
[::]	X	X	X	X	X
re*	X	X	X	X	X
re+	X		X		
re?	X		X		
re|re	X		X		
\d	X	X	X	X	X
(...)	X		X		
\(...\)		X		X	X
\<					
\>					
\{ \}	X		X		X

Examples: The following regular expressions are given as illustrations, along with descriptions of what they match:

abc
Matches any line of text containing the three letters abc in that order.

a.c
Matches any line of text containing the letter a, followed by *any* character, followed by the letter c.

^.$
Matches any line containing exactly one character (the newline is not counted).

a(b*|c*)d
Matches any line of text containing a letter a, followed by either zero or more of the letter b or zero or more of the letter c, followed by the letter d.

.* [a–z]+ .*
Matches any line containing a *word*, where a *word* is a sequence of lowercase alphabetic characters delimited by at least one space on each side.

morty.*morty
Matches a line containing at least two occurrences of the string `morty`.

(morty).*\1
This regular expression is functionally equivalent to regular expression **morty.*morty**.

[[:space:][:alnum:]]
Matches any character that is either a white-space character or alphanumeric.

regexp

Appendix C. Localization

Internationalization enables you to work in a cultural context that is comfortable for you through locales, character sets, and a number of special environment variables. The process of adapting an internationalized application or program, particular to a language or cultural milieu, is termed *localization*.

A *locale* is the subset of your environment that deals with language and cultural conventions. It is made up of a number of categories, each of which is associated with an environment variable and controls a specific aspect of the environment. The following list shows the categories and their spheres of influence:

LC_COLLATE
    Collating (sorting) order.

LC_CTYPE
    Character classification and case conversion.

LC_MESSAGES
    Formats of informative and diagnostic messages and interactive responses.

LC_MONETARY
    Monetary formatting.

LC_NUMERIC
    Numeric, nonmonetary formatting.

LC_TIME
    Date and time formats.

To give a locale control over a category, set the corresponding variable to the name of the locale. In addition to the environment variables associated with the categories, there are two other variables which are used in conjunction with localization, **LANG** and **LC_ALL**. All of these variables affect the performance of the shell commands. The general effects apply to most commands, but certain commands such as **sort**, with its dependence on **LC_COLLATE**, require special attention to be paid to one or more of the variables; this manual discusses such cases in the *Localization* section of the command. The effects of each environment variable is as follows:

LANG
    Determines the international language value. Utilities and applications can use the information from the given locale to provide error messages and instructions in that locale's language. If **LC_ALL** variable is not defined, any undefined variable is treated as though it contained the value of **LANG**.

LC_ALL
    Overrides the value of **LANG** and the values of any of the other variables starting with **LC_**.

LC_COLLATE
    Identifies the locale that controls the collating (sorting) order of characters and determines the behavior of ranges, equivalence classes, and multicharacter collating elements.

LC_CTYPE
    Identifies the locale that defines character classes (for example, *alpha, digit, blank*) and their behavior (for example, the mapping of lowercase letters to uppercase letters). This locale also determines the interpretation of sequences of bytes as characters (such as singlebyte versus doublebyte characters).

LC_MESSAGES
    Identifies the locale that controls the processing of affirmative and negative responses. This locale also defines the language and cultural conventions used when writing messages.

LC_MONETARY
    Determines the locale that controls monetary-related numeric formatting (for example, currency symbol, decimal point character, and thousands separator).

LC_NUMERIC
    Determines the locale that controls numeric formatting (for example, decimal point character and thousands separator).

LC_TIME
　　Identifies the locale that determines the format of time and date strings.

Appendix D. OpenExtensions Shell and Utilities Messages

GSU6001   **Unknown option** *option*

Explanation:
You specified an option that is not valid for this command.

User response:
Check the description in this book for the command you were using to find the valid list of options for that command.

GSU6003   **input file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6004   **write error on standard output**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6007   **insufficient memory for string storage**

Explanation:
There were not enough free system resources to use for string storage.

User response:
Free up more system resources, or modify your program to require less string storage.

GSU6013   **Missing script file**

Explanation:
You specified the **–f** option without providing the name of a script file.

GSU6014   **not enough available file descriptors**

Explanation:
There were less than four available file descriptors.

User response:
Free up more file descriptors.

GSU6016   **invalid character** *char* **(hex** *hexnum*

Explanation:
The invalid character *char* was encountered while processing the input file.

User response:
Check the input file for invalid characters.

GSU6017   **Newline in regular expression**

Explanation:

A newline was encountered while reading a regular expression.

User response:
Check for a missing **/** delimiter.

GSU6018   **Newline in string**

Explanation:
A newline was encountered while reading a string constant.

User response:
Check for a missing **"** delimiter.

GSU6019   **EOF in regular expression**

Explanation:
The end-of-file character was encountered while reading a regular expression.

User response:
Check for a missing **/** delimiter.

GSU6020   **EOF in string**

Explanation:
The end-of-file character was encountered while reading a string constant

User response:
Check for a missing **"** delimiter.

GSU6022   **inadmissible use of reserved keyword**

Explanation:
You tried to use a reserved keyword in an unacceptable way (for example, as a function or variable name).

User response:
Choose a different name for your function or variable.

GSU6023   **attempt to redefine builtin function**

Explanation:
You tried to redefine one of the built-in **awk** functions.

User response:
Choose a name for your function that is not the name of any built-in function. Refer to the description of the **awk** command, "awk — Process programs written in the awk language" on page 13, for lists of built-in arithmetic and string functions.

GSU6039   *string***: not found**

Explanation:

You specified a *command-name* that the shell was unable to find.

User response:
Make sure that *command-name* is spelled properly and that you have the appropriate permissions.

GSU6047 **unredirected getline in END action**

Explanation:
The default output stream has already been closed when the **end** action is performed, so a **getline** function must be redirected or it fails.

User response:
Redirect the **getline** function to read from a named file.

GSU6048 **too many open streams to *funcname* onto *filename***

Explanation:
awk can only have a limited number of files open at one time. There were too many open files.

User response:
Make sure that unused files are being closed properly, or restructure your program to have fewer files open at the same time.

GSU6049 **insufficient arguments to printf or sprintf**

Explanation:
You did not specify enough arguments to match the number required by the specified format string.

User response:
Check your format string and number of arguments.

GSU6052 **Too many fields (LIMIT: *number***

Explanation:
awk read a record with more fields than it was able to handle.

User response:
Edit the input file to decrease the number of fields in the record.

GSU6053 **Record too long (LIMIT: *number* bytes)**

Explanation:
awk read a record that was longer than the maximum record size it can handle. On UNIX and POSIX-compliant systems, the maximum record length is 20000 characters.

User response:
Edit the record so that it does not exceed the limit.

GSU6054 **division (/ or %) by zero**

Explanation:
An arithmetic operation using **/** or **%** resulted in an attempt to divide by zero.

User response:
Modify your program so that division by zero does not occur.

GSU6055 **too deeply nested for in loop (LIMIT: *number***

Explanation:
"For" loops can be nested only *number* levels deep.

User response:
Rewrite the program to use fewer levels of nesting.

GSU6058 **lvalue required in assignment**

Explanation:
You did not specify a variable or array element as the left-hand side of an assignment expression.

User response:
Specify a valid variable or array element on the left-hand side of the assignment operator.

GSU6059 **return outside of a function**

Explanation:
A **return** statement was encountered that was not part of a function.

User response:
Use the **return** statement only inside a function definition.

GSU6060 **may delete only array element or array**

Explanation:
You tried to use the **delete** statement to delete a scalar variable.

User response:
Use **delete** only to delete arrays and array element.

GSU6063 **SYMTAB must have exactly one index**

Explanation:
You tried to reference the SYMTAB array using more than one index.

User response:
Always reference SYMTAB with exactly one index.

GSU6065 **impossible function call**

GSU6069 **regular expression error**

Explanation:
An error occurred while processing a regular expression.

User response:
Check the regular expression.

GSU6070 **second parameter to "split" must be an array**

Explanation:

You invoked the **split** function but the second parameter was not an array.

User response:
Ensure that **split** is invoked with an array as the second parameter.

GSU6074	Unknown FP error

Explanation:
An unknown error occurred during a floating-point operation.

User response:
Contact your system programmer.

GSU6075	Domain

Explanation:
A domain error occurred when executing a floating-point operation. For example, taking the square root of a negative number would cause this error.

User response:
Make sure that you are performing a valid mathematical operation.

GSU6076	Singularity

Explanation:
The program executed a floating-point division that resulted in an infinite value.

User response:
Make sure that you are performing mathematical operations that produces finite results.

GSU6077	Overflow

Explanation:
The program executed an operation that resulted in a number that is larger than **awk** can represent on this platform.

User response:
Correct the program to use values that are supported on this system or use **bc**.

GSU6078	Underflow

Explanation:
The program executed an operation that resulted in a number that is smaller than **awk** can represent on this platform.

User response:
Correct the program to use values that are supported on this system or use **bc**.

GSU6079	Total loss of precision

Explanation:
The program executed a floating-point operation that used an intermediate result that cannot be properly generated on this platform.

User response:

Correct the program or use **bc**.

GSU6080	Partial loss of precision

Explanation:
The program executed a floating-point operation that used in an intermediate result that cannot be properly generated on this platform.

GSU6081	error in function *funcname* at *arg*

Explanation:
A math error occurred while performing the function *funcname* on argument *arg*.

User response:
Make sure that you are passing a proper argument to the function *funcname*.

GSU6084	Missing width after -w

Explanation:
You specified the **–w** option without providing the width argument.

User response:
Provide the missing width.

GSU6088	Usage: basename filename [suffix]

Explanation:
The **basename** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6089	break statement found outside of loop

Explanation:
bc encountered a **break** statement when it was not performing a "for" or "while" loop.

User response:
Make sure that all "break" statements occur within "for" or "while" loops.

GSU6090	warning: body of if/else statement is empty

Explanation:
You did not supply any statements for the body of an "if" or "if/else" construct. **bc** only generates this message when you have specified **–i** option.

User response:
Make sure that this is what you intended.

GSU6092	empty stack (too few arguments ?)

Explanation:

An error occurred while executing a function, probably because the function was called with fewer arguments than required.

User response:
Make sure that you call functions with the correct number of arguments.

GSU6095   valid array index is 0 through *num*

Explanation:
You specified an array index that was not in the range 0 to **BC_DIM_MAX**-1, where **BC_DIM_MAX** is a configuration variable indicating the maximum number of elements that a **bc** array may have.

User response:
Specify an array index in the indicated range.

GSU6099   shell command failed to execute

Explanation:
You specified the **sh** statement with *command* as its argument and **bc** failed to run *command*.

User response:
Check the syntax of the specified command.

GSU6101   end of file in comment starting on line *num* of *filename*

Explanation:
bc encountered the end-of-file character when reading a comment which begins on line *num* of the file *filename*.

User response:
Make sure that the file *filename* properly closes all comments.

GSU6102   end of file in string starting on line *num* of *filename*

Explanation:
bc encountered the end-of-file character when reading a string that begins on line *num* of the file *filename*.

User response:
Make sure that the file *filename* contains a double quotation mark (") at the end of the string.

GSU6103   warning: '=-' operator assumed

Explanation:
This version of **bc** permits the use of the old style assignment operators like **=-** rather than **-=**. This can be ambiguous since **a=-2** can mean **a =- 2** or **a = -2**. **bc** has assumed that you meant to use the **=-** operator.

User response:
Use spaces to clarify the syntax of the expression.

GSU6104   numerical constant is too long

Explanation:

You specified a numerical constant that was longer than the maximum permitted length, as defined by the value of the configuration variable **BC_STRING_MAX**.

User response:
Specify a shorter numerical constant.

GSU6105   string is too long

Explanation:
You specified a string that was longer than the maximum permitted length, as defined by the value of the configuration variable **BC_STRING_MAX**.

User response:
Specify a shorter string.

GSU6107   Unknown option *option*

Explanation:
You specified an option that is not valid for this command.

User response:
Check the description in this book for the command you were using to find the valid list of options for that command.

GSU6110   usage: bc [-i] [-l] [file ...]

Explanation:
The **bc** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6112   command too long

Explanation:
You specified a command line to pass to the system with the **!** operator that was longer than 1000 bytes.

User response:
Use a shorter command line.

GSU6114   save: args

Explanation:
You tried to use the **s** or **S** operator when there was no value on the stack.

User response:
Make sure that there is at least one value on the stack before trying to use the **s** and **S** commands.

GSU6118   negative argument to of the stack was negative. Q cannot take a negative argument.

Explanation:
You tried to use the **Q** operator but the value on the top of the stack was negative. **Q** cannot take a negative argument.

User response:
Make sure that the stack has a positive number on top when using the **Q** operator.

GSU6119　　　　**readstk?**

Explanation:
You tried to pop too many values off the stack with the **Q** operator.

User response:
Make sure that the top value on the stack is not greater than the number of currently executing strings.

GSU6120　　　　**L?**

Explanation:
You tried to pop a value off an empty stack variable using the **L** operator.

User response:
Correct your program.

GSU6121　　　　**Q?**

Explanation:
You specified a string argument to the **Q** command. This is invalid. The **Q** command requires a numeric argument.

User response:
Correct your program.

GSU6122　　　　**negative index**

Explanation:
You tried to use a negative number as an array index.

User response:
Use a positive number as an array index.

GSU6123　　　　**index too big**

Explanation:
You tried to use an array index that was greater than 2047.

User response:
Use an array index that is less than or equal to 2047.

GSU6124　　　　**cannot execute number**

Explanation:
You tried to use the **x** operator to execute a string, but the value on the top of the stack was a number.

User response:
Only use the **x** operator when there is a string on top of the stack.

GSU6125　　　　**divide by 0**

Explanation:
You tried to divide a number by 0.

User response:
Do not divide numbers by 0.

GSU6126　　　　**exponent must be an integer from 0 to *max***

Explanation:
You specified an exponent that was not an integer in the range 0 **SHRT_MAX**-1.

User response:
Specify an exponent in the valid range.

GSU6130　　　　**sqrt of negative number**

Explanation:
You tried to take the square root of a negative number. The **sqrt** function must be used with positive numbers.

User response:
Use the **sqrt** function only with positive numbers.

GSU6131　　　　**stack too deep**

Explanation:
You tried to put more values on the stack than it was able to hold. The maximum size of the stack is limited by the size of the maximum integer your system can represent.

User response:
Check for uncontrolled recursion.

GSU6132　　　　**empty stack**

Explanation:
You attempted an operation that required popping a value from the stack, but the stack was empty.

User response:
Push a value onto the stack and try the operation again.

GSU6133　　　　**out of memory**

Explanation:
There were not enough free system resources to allocate the required space.

User response:
Free up more resources and try again.

GSU6134　　　　**out of memory (fatal)**

Explanation:
bc ran out of system resources but was unable to recover sufficient storage to continue.

User response:
Free up more resources and try again. Pay particular attention to large arrays.

GSU6140　　　　***string*: *command:* external links not supported by OS**

Explanation:
You tried to extract a file specified as an external link in the archive. External links are not supported on all operating systems.

User response:
Do not use external links on this system.

GSU6141   *string* **external link to** *name1*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6142   **External link name too long: Not extracted**

Explanation:
Couldn't allocate enough memory to hold the external link's name.

User response:
Archive contains external name which is too large; no action possible.

GSU6143   **Symbolic or external link name too long: Not extracted**

Explanation:
Couldn't allocate enough memory to hold the symbolic or external link's name.

User response:
Archive contains symbolic or external name which is too large; no action possible.

GSU6144   **external link** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6145   *string*: **external link** *command:* **tar format does not permit external links to pathnames longer than** *filename*

GSU6146   **external link to** *filename*

GSU6147   *string*: *command:* **is a socket file-- not dumped**

Explanation:
You tried to dump a socket file *filename* when writing a non-USTAR **tar** file.

User response:
Do not specify socket files to be included in **tar** archives. If you want to archive socket files, use a USTAR format archive.

GSU6155   **insufficient memory**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6178   **cat: input file** *filename* **is identical with output**

Explanation:

You specified *filename* as both an input and output file. It is also possible that the output file was linked to *filename*.

User response:
Use a file other than *filename* as the output file.

GSU6179   **Usage: cat [-usvte] [file ...]**

Explanation:
The **cat** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6180   **file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6181   *string*: **fatal error during "-R" option**

Explanation:
You specified the **–R** option but some file or directory in the directory structure was inaccessible.

User response:
Make sure that you have access to all files in the directory structure.

GSU6184   *string*: **file** *command:* **You are not a member of the** *filename* **group**

Explanation:
You tried to change the group ownership of *filename* to *group*, but you are not a member of the specified group.

User response:
Specify a group to which you belong.

GSU6185   *string*: **group** *command:* **is unknown**

Explanation:
You specified a group name that could not be found in the group database.

User response:
Specify a valid group name or use a valid numeric group ID.

GSU6186   **Usage: chgrp [-Rf] group file ...**

Explanation:
The **chgrp** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6187 **Missing mode argument.**

Explanation:
You did not specify an argument representing the new access permissions.

User response:
Provide the missing argument.

GSU6188 **stat file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6189 **read directory** *pathname*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6190 **Usage: chmod [-fR] mode file …**

Explanation:
The **chmod** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6191 *string***: user** *command:* **is unknown**

Explanation:
You specified a user name that could not be found in the user database.

User response:
Specify a valid user name or use a valid numeric user ID.

GSU6192 **Usage: chown [-Rf] user[:group] file …**

Explanation:
The **chown** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6196 *string***: not executable**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6199 **[read error]**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6200 **Usage:** *command:* **[-ciprt] [file …]**

Explanation:
The **mv** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6205 *string file1* **differ: char** *file2* **line** *char_num*

GSU6210 **cannot determine PATH_MAX**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6213 **cannot determine NAME_MAX**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6214 **cannot allocate buffer**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6219 **output file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6236 **source** *name1* **and target** *name2* **are identical**

GSU6237 **no space on device for file** *filename*

Explanation:
You tried to copy (or move) a file to *filename* on a device that has no space for it.

User response:
Free up space on the target device or copy (or move) the file to another device.

GSU6238 **cannot unlink source file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6239 **cannot unlink target file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6241 **Unknown option** *option*

Explanation:
You specified an option that is not valid for this command.

User response:

Check the description in this book for the command you were using to find the valid list of options for that command.

GSU6242   **target** *pathname* **must be a directory**

Explanation:
You tried to copy (or move) two or more files but the target indicated by *name* was not a directory.

User response:
When copying (or moving) two or more files, ensure that the final *name* on the command line is a directory.

GSU6243   **cannot allocate target string**

Explanation:
There are not enough free system resources to hold the name of the target file.

User response:
Free up more system resources.

GSU6244   **cannot rename** *file1* **to** *file1*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6245   **link to target** *filename* **failed**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6246   **cannot rmdir** *pathname*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6247   **stat error for** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6248   **unreadable directory** *pathname*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6249   **recursive copy to directory** *pathname*

Explanation:
You tried to copy a directory to itself.

User response:
Choose a different *pathname*.

GSU6250   **target** *pathname* **is not a directory**

Explanation:
When recursively copying (or moving) multiple files using the **–r** or **–R** option, the target must be a

directory. You specified a target *pathname* that is not a directory.

User response:
Check spelling of target *pathname*.

GSU6251   **cannot mkdir** *pathname*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6254   **"***string***" is a directory (not copied)**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6255   **fifo** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6256   **special file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6257   **cannot allocate I/O buffer**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6258   **cannot open file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6259   **target file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6260   **write error on file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6261   **read error on file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6262   **Usage:** *command: options* **file1 [file2 ...] target**

Explanation:
The **command** command entered was not syntactically correct.

User response:

The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6263   *string*: must run as setuid root

Explanation:
You must be logged in with the user ID of **root** to run the specified command.

User response:
Log in as **root** or contact your system manager to run this command.

GSU6319   temporary file

Explanation:
A system error indicating the cause is displayed with this message.

GSU6342   write error

Explanation:
A system error indicating the cause is displayed with this message.

GSU6349   cannot create temporary file

Explanation:
A system error indicating the cause is displayed with this message.

GSU6367   Usage: cut -b list [file ...] cut -c list [file ...] cut -f list [-d char] [-s] [file ...]

Explanation:
The **cut** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6368   cut: bad list for -f, -b, or -c option *list*

Explanation:
You specified a list for the **–f**, **–b**, or **–c** option that contained non-numeric entries.

User response:
Specify a list that contains only numeric entries.

GSU6369   cut: badly formed range in list *list*

Explanation:
You specified a list that contained a range that was not in the form: *num1-num2*

User response:
Reenter the command line using the proper syntax for a range.

GSU6371   Missing character after -d

Explanation:

You specified the **–d** option, but did not provide a field separator character as its argument.

User response:
Provide the missing field separator character.

GSU6372   Unknown option *option*

Explanation:
You specified an option that is not valid for this command.

User response:
Check the description in this book for the command you were using to find the valid list of options for that command.

GSU6373   Must specify "-f", "-b" or "-c" option

Explanation:
You did not specify any of the **–f**, **–b**, or **–c** options.

User response:
Specify one of the three options.

GSU6374   out of memory

Explanation:
There were not enough free system resources to allocate as internal buffers.

User response:
Free up more system resources and try again.

GSU6375   cut: no fields specified in list *list*

Explanation:
cut did not recognize anything in *list* as indicating a field.

User response:
Check the syntax of the list and reenter the command.

GSU6376   Bad range *num1* in list

Explanation:
You specified a list containing the range *num1-num2* where *num2* was less than *num1*. Ranges must be specified with the lower value first.

User response:
Reenter the command line, making sure to list the lower value first when specifying the range.

GSU6378   date: no permission to set date

Explanation:
You do not have proper permissions for changing the system date.

User response:
If you need the system date changed, talk to your system programmer.

GSU6380   The option "*string*" does not contain a "="

Explanation:
You specified *option* without providing the required equals sign (**=**).

User response:
Provide the missing equals sign.

GSU6381   **dd: cbs=***number* **given without ascii/ebcdic/ibm/block/unblock conversion**

Explanation:
You specified the **cbs=size** option but did not specify a conversion option that uses it.

User response:
Provide the missing conversion option.

GSU6382   **dd: out of memory for buffers**

Explanation:
dd was unable to allocate the system resources that it needed for conversion buffers.

User response:
Free up more system resources.

GSU6383   *string=string* **is an unknown option**

Explanation:
You specified an option that is not valid for **dd**.

User response:
Check "dd — Convert and copy a file" on page 95 for a list of options.

GSU6384   *number+number* **records in** *number+number* **records out**

GSU6388   **dd: unknown conversion "***string***"**

Explanation:
You specified a conversion value following **conv=** that **dd** did not recognize.

User response:
Check "dd — Convert and copy a file" on page 95 for a list of options.

GSU6389   **dd: badly formed number "***string***"**

Explanation:
Make sure that *num* is a valid number. If it is also followed by a letter to indicate the block size unit. Then check the **bs=** option, "dd — Convert and copy a file" on page 95, for a list of valid letters.

GSU6390   **dd: absolute I/O must be in** *number* **byte units**

Explanation:
You tried to read from, or write to, a device that requires block sizes to be in multiples of its sector size (in this case, *num* bytes).

User response:

Specify a block size that is a multiple of the device's sector size.

GSU6391   **seek output**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6392   **seek input**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6393   **read error**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6394   **Usage: dd [option=value] ...**

Explanation:
The **dd** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6404   **directory** *pathname*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6405   **insufficient memory (try diff -h)**

Explanation:
diff ran out of system resources when generating the data structures used in the differencing algorithm (see the *LIMITS* section of **diff**, "diff — Compare two text files and show the differences" on page 99). **diff -h** requires fewer system resources than **diff**.

GSU6406   **cannot allocate name buffer**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6407   **Missing number after** *option* **option**

Explanation:
You specified *option* but did not specify a number following it.

User response:
Specify a number following the *option* option.

GSU6408   **Missing #ifdef symbol after -D**

Explanation:
You did not specify a conditional label on the command line after **–D** option.

GSU6409   **only one file may be "-"**

Explanation:
Only one of the two files being compared may be the standard input.

User response:
Specify – (standard input) as, at most, one of the two files to be compared.

GSU6411   **internal error--cannot create temporary file**

Explanation:
diff was unable to create a working file that it needed.

User response:
Ensure that you either have a **/tmp** directory or that the environment contains a variable **TMPDIR** which names a directory where **diff** can store temporary files. Also, ensure that you have sufficient permissions on this directory to create a temporary file.

GSU6412   **couldn't stat file system for** *filesystem*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6417   **too many lines in file** *filename*

Explanation:
The file *filename* contained more than the value of the **INT_MAX**. **diff** cannot handle a file that large.

GSU6429   **Usage: dirname pathname**

Explanation:
The **dirname** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6432   **Addressed line out of range**

Explanation:
You specified an address for a command that referenced a line that does not exist.

User response:
Modify the address given to correctly reference the desired lines.

GSU6434   **Only one file name is allowed.**

Explanation:
You specified more than one file name on the command line when you invoked **ed**.

GSU6437   **File** *filename system_error*

Explanation:

A system error indicating the cause is displayed with this message.

GSU6438   **Usage: ed [-p prompt] [-bsx] [file] red [-p prompt] [-bsx] [file]**

Explanation:
The **ed** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6439   **Temporary file error**

Explanation:
An error occurred when accessing the paging file. Check the description of **ed**, "ed — Use the ed line-oriented text editor" on page 109.

User response:
See your system programmer.

GSU6440   **Badly constructed regular expression**

Explanation:
You made an error in the syntax of a regular expression.

User response:
Refer to Appendix B, "Regular Expressions (regexp)," on page 471 and correct the error.

GSU6441   **No remembered regular expression**

Explanation:
You tried to use **&;** to refer to a remembered regular expression when there was no remembered regular expression.

User response:
Issue the command again, but specify a regular expression this time.

GSU6442   **Missing trailing delimiter after pattern.**

Explanation:
You specified a pattern as part of a **ed** command but did not delimit it.

User response:
Provide a trailing delimiter for the pattern.

GSU6446   **Out of memory for lines**

Explanation:
ed was unable to allocate system resources while trying to insert or append lines to the buffer.

User response:
Split the file into small pieces.

GSU6447   **Unknown command**

Explanation:
You entered a command that does not exist in **ed**.

User response:
Check the description of **ed**, "ed — Use the ed line-oriented text editor" on page 109, for a list of valid commands.

GSU6448   **Illegal command suffix**

Explanation:
You specified a command suffix for a command that does not accept suffixes.

User response:
Check the description of **ed**, "ed — Use the ed line-oriented text editor" on page 109, for a list of valid commands and their syntaxes.

GSU6449   **Warning: file not saved**

GSU6450   **No match found for regular expression**

Explanation:
The **/** command failed to find any matching lines.

User response:
Try a different regular expression.

GSU6451   **Wrong number of addresses for command**

Explanation:
You specified the wrong number of addresses for the command that you entered.

User response:
Check the description of **ed**, "ed — Use the ed line-oriented text editor" on page 109, for a list of valid commands and the number of addresses that you can specify with each.

GSU6452   **Need space after command**

Explanation:
You did not separate a command from its file name argument with a space.

User response:
Reenter the command with the required space.

GSU6453   **Name too long**

Explanation:
The file name specified on the **ed** command line was too long.

User response:
Use a shorter file name.

GSU6454   **Badly formed name**

Explanation:

You specified an improperly formed or missing file name with a command which requires a file name as an argument (for example, **e** or **f**).

User response:
Correct or provide the file name.

GSU6455   **Illegal command redirection**

Explanation:
You tried to use the **!** command redirection with the **f** command.

User response:
Do not use the **!** command redirection with the **f** command.

GSU6456   **Restricted shell**

Explanation:
You invoked the restricted form of **ed** (**red**), but then tried to use a command that is not allowed in the restricted editor (the **!** command).

User response:
See the **ed** command, "ed — Use the ed line-oriented text editor" on page 109, for a discussion of the differences between **ed** and **red**.

GSU6457   **No remembered file name**

Explanation:
You tried to execute a command that uses a remembered file name when there was no remembered file name.

User response:
Issue the command again, but specify a file name this time.

GSU6458   **Mark name must be lower case**

Explanation:
You tried to use the **k** command to mark an addressed line with a character other than a lowercase letter.

User response:
Use **k** to mark the line with a lowercase letter.

GSU6459   **Undefined mark name**

Explanation:
You tried to reference a mark name that you have not assigned.

User response:
Use the **k** command to assign the mark name to a line, or specify a previously assigned mark name.

GSU6460   **'m' and 't' require destination address**

Explanation:
You issued an **m** or **t** command but did not provide a destination address.

User response:

Provide a destination address with the **m** or **t** command.

GSU6461	**Destination cannot straddle source in 'm' and 't'**

Explanation:
You specified a range of lines to be moved or copied by **m** or **t** that included the destination address.

User response:
Ensure that the specified range of lines for **m** or **t** does not include the destination address.

GSU6462	**command not allowed inside g, v, G, or V**

Explanation:
You specified a command that cannot be used with the issued global command (**g**, **v**, **G**, or **V**).

User response:
Check the description of **ed**, "ed — Use the ed line-oriented text editor" on page 109, for a list of commands that cannot be used with the various global commands

GSU6463	**Incomplete regular expression.**

Explanation:
You issued a **g** or **G** command but did not provide a regular expression as an argument.

User response:
Provide a regular expression as an argument to the command.

GSU6465	**Global command too long**

Explanation:
You specified a global instruction (**g** or **v**) that was longer than 256 characters, including newlines.

User response:
Specify a global instruction that is less than 256 characters in length.

GSU6466	*string*: **too many environment variables**

Explanation:
You specified more than 512 environment variables in a single **env** command.

User response:
Do not specify more than 512 environment variables in a single **env** command.

GSU6467	**Usage: env [-i] [name=value ...] [command argument ...] env [-] [name=value ...] [command argument ...]**

Explanation:
The **env** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6473	**expr: internal tree error**

Explanation:
You specified an expression that **expr** was unable to evaluate, due to either syntax errors or unusual complexity.

User response:
Correct the syntax errors, or simplify the expression (perhaps by breaking it into parts).

GSU6476	**Usage: expr expression**

Explanation:
The **expr** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6478	**Only one *f* option allowed**

Explanation:
You specified the given option more than once.

User response:
Specify the given option once only.

GSU6503	**find: unable to allocate memory for expression tree**

Explanation:
find requires system resources to build an expression tree. There were not enough free resources to do so.

User response:
Free up more system resources or specify a less complex expression.

GSU6504	**find: bad number specification in *string***

Explanation:
You specified an option that takes a numeric value (for example, **–atime**, **–ctime**), but you did not specify a valid number after the option.

User response:
Ensure that options that take a numeric value are followed by a valid number (only decimal digits, preceded by an optional plus or minus sign).

GSU6505	**find: -type *character* is invalid**

Explanation:
You specified the **–type** primary but did not follow with a valid character to represent the file type.

User response:

Check the description of **find**, "find — Find a file meeting specified criteria" on page 131, for a list of valid characters for use with the **–type** primary.

GSU6506	find: non-terminated *primary* argument list

Explanation:
You specified the **–exec** or **–ok** primary and did not terminate the argument list following it with a semicolon (**;**).

User response:
Terminate the argument list following **–exec** or **–ok** with a semicolon.

GSU6507	find: must specify option after *primary*

Explanation:
You specified –*primary*, but did not provide the argument that it requires.

User response:
Specify a valid argument after –*primary*.

GSU6508	cannot stat file *filename* for -newer

Explanation:
A system error indicating the cause is displayed with this message.

GSU6509	find: -cpio option not yet supported

Explanation:
You specified the **–cpio** primary. At this time, **find** does not support this primary.

User response:
Do not use the **–cpio** primary.

GSU6510	find: user name *user* is unknown

Explanation:
You specified the **–user** primary, but did not provide a valid user name.

User response:
Provide a valid user name after the **–user** primary.

GSU6511	find: group name *name* is unknown

Explanation:
You specified the **–group** primary but did not specify a valid group name.

User response:
Specify a valid group name after the **–group** primary.

GSU6512	unable to access *pathname*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6513	error reading directory *pathname*

Explanation:
You tried to read the directory *pathname*. You do not have read permissions on this directory.

User response:
If you need to access the directory *pathname*, see your system manager about acquiring read permissions for that directory. If you do not need to access it, no corrective action is required.

GSU6515	cannot execute *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6516	Usage: find directory ... expression

Explanation:
The **find** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6521	Usage: fold [-#] [-bs] [-w width] [file ...]

Explanation:
The **fold** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6527	no room for buffers

Explanation:
There were not enough free system resources for **grep** to allocate the buffers that it requires.

User response:
Free up more system resources.

GSU6528	(standard input)

GSU6529	out of space for pattern "*string*"

Explanation:
grep did not have enough system resources available to store the code needed to work with the given pattern (regular expression). The usual cause is that the pattern is very complex.

User response:
Make the pattern simpler, or free more system resources.

GSU6535	Badly formed line/character count *num*

Explanation:
The value *num*, following a **–b**, **–c**, –k, **–l**, **–m**, or **–n** option was not a valid number.

User response:
Ensure that *num* is a valid number. For more information on the find command, refer to "find — Find a file meeting specified criteria" on page 131.

GSU6541	*string*: invalid user name: *command:*

Explanation:
You specified a user name that was not found in the user database.

User response:
Check that you spelled the user name correctly.

GSU6542	Usage: id [user] id -G [-n] [user] id -g [-nr] [user] id -u [-nr] [user]

Explanation:
The **id** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6543	Usage: join [-a n | -v n] [-e s] [-1 m] [-2 m] [-o list] [-t c] file1 file2 join [-a n] [-e s] [-jn m] [-o list] [-t c] file1 file2

Explanation:
The **join** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6544	Bad file number specification in *string*

Explanation:
You specified a file number that was not **1** or **2** with the **–j** option.

User response:
Specify a file number of **1** or **2** when using the **–j** option.

GSU6545	Badly constructed output list at *string*

Explanation:
You specified an improperly constructed list of output fields with the **–o** option.

User response:
Check the description of **join**, "join — Join two sorted, textual relational databases" on page 155, for details on constructing a list of output fields for the **–o** option.

GSU6546	Missing -e string

Explanation:
You specified the **–e** option without a string argument.

User response:
Provide the missing string.

GSU6547	Missing join field number

Explanation:
You specified the **–j**, **–1**, or **–2** option without specifying which field to use as the join field.

User response:
Provide the missing join field number.

GSU6548	Bad join field number

Explanation:
You specified a value to indicate the join field that was not a valid number.

User response:
Make sure to use a valid number to indicate join the field.

GSU6549	Missing character after -t

Explanation:
You specified the **–t** option without specifying a field separator as an argument.

User response:
Provide the missing field separator.

GSU6550	Must specify -o with -e

Explanation:
You specified the **–e** option without also specifying the **–o** option.

User response:
Always specify the **–o** option when using the **–e** option.

GSU6551	join: too many -o list elements

Explanation:
You specified more than 512 fields in the list of output fields given as the argument to the **–o** option.

User response:
Specify no more than 512 output fields.

GSU6553	Out of dfa move space: increase *num* from *number*

Explanation:
There were not enough move entries for **lex** to process your input.

User response:
Increase move table size with the *hexnum* directive.

GSU6555	dfabuild: stack overflow
GSU6556	eclosure: list overflow
GSU6563	Error writing temp file *filename*

Explanation:
An error occurred while trying to write the temporary file *filename*.

User response:
Check the directory indicated by **TMPDIR**, or **/tmp** and ensure that the directory is writable and has sufficient space.

GSU6564   **No lex rules**

Explanation:
You specified **lex** input that did not contain any translation rules, possibly due to empty or badly formatted input.

User response:
Make sure that your input file is specified properly, and that the contents are properly formatted.

GSU6565   **Write error on *filename***

Explanation:
An error occurred while **lex** was writing the output file.

User response:
Check that space exists on the output device and that you have appropriate permissions to write the file.

GSU6569   **Out of NFA state space: increase *num* from *number***

Explanation:
You did not reserve enough space for the NFA tables.

User response:
Use the *number* directive to increase the space for the NFA tables.

GSU6570   **Out of DFA state space: increase *num* from *number***

Explanation:
You did not reserve enough space for the DFA tables.

User response:
Use the *integer* directive to increase the space for the DFA tables.

GSU6571   **Too many character classes (more than *num*)**

Explanation:
lex ran out of space for character classes.

User response:
Simplify your scanner input.

GSU6572   **Too many translations (more than *num*)**

Explanation:
lex ran out of space for translation rules.

User response:
Simplify your scanner input.

GSU6573   **Table for *item* too large for machine (*num* bytes)**

Explanation:
You tried to use the **lex Malloc** function to allocate a block of memory that is larger than the hardware segment size. This error occurs only on systems with segment architecture.

User response:
Use **Malloc** to allocate a block of memory that is smaller than the hardware segment size.

GSU6574   **No more memory for *item***

Explanation:
There were not enough free system resources to allocate to *item*. Your scanner input was too large or too complicated, or you requested too much space for a table.

User response:
Simplify your input expressions, or request less space for tables.

GSU6579   **Too many move *num* entries: *number***

Explanation:
You did not reserve enough space for move tables.

User response:
Use the *hexnum* directive to increase the space for move tables.

GSU6582   **premature eof in prototype**

Explanation:
lex encountered an end-of-file character in the prototype file when it was not expecting it, probably due to a badly formatted prototype file.

User response:
Ensure that the prototype file is not corrupted. If using a private prototype file, ensure that it has the same layout as the distributed version.

GSU6600   **Cannot use character class or equivalence class in range**

Explanation:
You tried to use a character class or an equivalence class (that is, **[: :]** or **[= =]**) in a character range within a regular expression.

User response:
Rewrite the regular expression.

GSU6601   **Poorly formed *char* sequence *string***

Explanation:
You specified a **[. .]**, **[= =]**, or **[: :]** sequence improperly.

User response:
Specify the sequence correctly.

GSU6602	Unknown class *class*

Explanation:
You specified a regular expression containing a character class **[***class***:]** that is not supported in the POSIX locale.

User response:
Rewrite the regular expression.

GSU6603	Unknown collating element *col_element*

Explanation:
You specified a regular expression containing a collating element that is not supported by the POSIX locale.

User response:
Rewrite the regular expression.

GSU6604	Multi-character collating element *col_element* not supported

Explanation:
You specified a regular expression containing a multicharacter collating element that is not supported by the POSIX locale.

User response:
Rewrite the regular expression.

GSU6605	Collation in [= =] not supported (yet)

Explanation:
You tried to use an equivalence class [=[.*collation-symbol*.]=] within a regular expression. **lex** does not support this construct.

User response:
Rewrite the regular expression.

GSU6606	badly formed equivalence class *equiv_class*

Explanation:
You tried to use a multicharacter equivalence class in a regular expression. lex does not support non-POSIX locales.

User response:
Rewrite the regular expression.

GSU6614	*string*: Option *command:* argument missing

Explanation:
You did not provide an argument for **-option**

User response:
Provide the missing argument.

GSU6630	compress not initialized

GSU6632	no space for compression tables

Explanation:
There were not enough free system resources to allocate to compression tables.

User response:
Free up more resources.

GSU6633	compression not closed

GSU6634	compress: unknown error

Explanation:
An unknown compression error occurred.

User response:
Contact your system manager.

GSU6635	not initialized

GSU6636	not in compressed format

Explanation:
You specified a file to be uncompressed that was not in compressed format.

User response:
Specify a compressed file.

GSU6637	compressed with *num1* bits, can only handle *num2* bits

Explanation:
You specified a file to be uncompressed that was compressed with *num1* bits, but only a maximum of *num2* bits are supported.

User response:
Request a copy of the file compressed using *num2* bit compression.

GSU6638	no space for decompress tables

Explanation:
There were not enough free system resources to allocate to the decompress tables.

User response:
Free up more resources.

GSU6639	compressed file is corrupt

Explanation:
You specified a compressed file that was damaged.

User response:
Get a new copy of file and try again.

GSU6640	not closed

GSU6641	unknown error

Explanation:
An unknown decompression error occurred.

User response:
Contact your system manager.

GSU6642	Insufficient memory

Explanation:

There were not enough free system resources to perform the specified operation.

User response:
Free up more resources.

GSU6643	getgroups failed

Explanation:
A system error indicating the cause is displayed with this message.

GSU6644	Unknown or missing operator in symbolic mode *modestring*

Explanation:
When using the symbolic mode to indicate new access permissions, you specified a string *modestring* which was either missing an operator or contained an unrecognized operator.

User response:
Make sure that all *mode* values in symbolic mode contain one of the following operators: **+**, **-**, or **=**.

GSU6645	Octal mode may contain only digits [0-7] in *numstring*

Explanation:
When using the octal mode to indicate new access permissions, you specified a string *numstring* which contained a character other than the digits 0 to 7.

User response:
Make sure that all *mode* values in octal mode are valid octal numbers, containing only the digits 0 through 7.

GSU6647	failed to match

Explanation:
A match was found for the specified regular expression.

User response:
No action is required.

GSU6648	invalid collation element

Explanation:
You specified a regular expression that contains an invalid collating element.

User response:
Make sure that all collating elements in the regular expression are valid in the locale indicated by **LC_COLLATE**.

GSU6649	trailing \ in pattern

Explanation:
You specified a regular expression with a trailing ****.

User response:
Remove the trailing **** or complete the escape sequence.

GSU6650	newline found before end of pattern

Explanation:
You specified a regular expression that contained a newline before the end of the pattern.

User response:
Check the regular expression for a missing **/**.

GSU6652	number in \[0-9] invalid

Explanation:
You specified a number that was greater than the number of matching subexpressions.

User response:
Specify a number that is less than or equal to the number of matching subexpressions.

GSU6653	[ ] imbalance or syntax error

Explanation:
You specified a regular expression that contained a **[ ]** imbalance.

User response:
Make sure that all **[** and **]** characters appear in matched pairs in the regular expression.

GSU6654	( ) or \(\) imbalance

Explanation:
You specified a regular expression that contained a **()** or **\(\)** imbalance.

User response:
Make sure that all **(** and **)** characters and all **\(** and **\)** characters appear in matched pairs in the regular expression.

GSU6655	{ } or \{ \} imbalance

Explanation:
You specified a regular expression that contained a **{}** or **\{\}** imbalance.

User response:
Make sure that all **{** and **}** characters and all **\{** and **\}** characters appear in matched pairs in the regular expression.

GSU6656	invalid endpoint in range

Explanation:
You specified a regular expression that contained a range expression with an invalid endpoint.

User response:
Specify a valid endpoint.

GSU6657	?, *, or + not preceded by valid regular expression + which was not preceded by a valid regular expression. the regular expression

is preceded by a valid regular expression.

GSU6658   **invalid character class type**

Explanation:
You specified a regular expression that contained a reference to an invalid character class.

User response:
Make sure that all character classes referenced in the regular expression are valid in the locale indicated by **LC_CTYPE**.

GSU6659   **syntax error**

Explanation:
You specified an invalid regular expression.

User response:
Correct the syntax of the regular expression.

GSU6660   **contents of { } or \{ \} invalid**

Explanation:
The contents of **\{\}** or **{}** in the specified regular expression were invalid: not a number, too large a number, more than two numbers, first number larger than second.

User response:
Make sure that the contents of **\{\}** or **{}** are valid.

GSU6661   **internal error**

GSU6662   **unknown regex error**

Explanation:
The error code that was passed to **regerror** is not a known error.

User response:
Check your program to verify that *errcode* was retrieved from **regexec** or **regcomp**.

GSU6684   **unknown command**

GSU6700   **Charmap information not available.**

Explanation:
For some reason, **locale** was unable to list the set of available charmap files.

User response:
Contact your system programmer.

GSU6701   **Unknown keyword name *name***

Explanation:
You specified a *name* that is not a keyword.

User response:
Specify a valid keyword name.

GSU6745   **Insufficient memory**

Explanation:

A system error indicating the cause is displayed with this message.

GSU6751   **Directory *pathname***

Explanation:
A system error indicating the cause is displayed with this message.

GSU6762   **console device *dev***

Explanation:
A system error indicating the cause is displayed with this message.

GSU6763   **writing to console device *dev***

Explanation:
A system error indicating the cause is displayed with this message.

GSU6764   **logname: cannot get login name**

Explanation:
logname was unable to access the system **.utmp** file, or the process was not a currently logged in user.

User response:
Check that the system **utmp** file is accessible.

GSU6768   **!opening archive *arch_name***

Explanation:
A system error indicating the cause is displayed with this message.

GSU6769   ***string* is not a valid archive**

Explanation:
arch_name is not a valid archive. The recognized formats are system specific.

User response:
Ensure that you specified the correct file.

GSU6770   **!insufficient memory**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6771   **!opening temporary archive *arch_name***

Explanation:
A system error indicating the cause is displayed with this message.

GSU6773   **!rename *arch_name1* to *arch_name2***

Explanation:
A system error indicating the cause is displayed with this message.

GSU6780   **executable file *filename***

Explanation:

A system error indicating the cause is displayed with this message.

GSU6781	*string*: file *command:* **Not an executable file**

Explanation:
You specified a file on the command line that is not an executable file.

User response:
Specify an executable file.

GSU6782	**Usage: strip exec_file ...**

Explanation:
The **strip** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6784	**cannot allocate memory for sorting**

Explanation:
There were not enough system resources available for `ls` to sort its output.

User response:
Free up more system resources or use option and path names on the command that will produce less output.

GSU6785	**File or directory *name* is not found**

Explanation:
You specified a *pathname* that does not exist.

User response:
Check to make sure that you did not omit or misspell any components of *pathname*.

GSU6786	**too many directory entries in *dir***

Explanation:
`ls` ran out of dynamically allocated system resources.

User response:
Free up more system resources.

GSU6788	**Usage: ls [-RaAd1CxmlnogrtucpFbqisfLDW] [file ...]**

Explanation:
The **ls** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6807	**expression syntax error**

Explanation:
You specified an expression argument that was not a well-formed expression.

User response:
Check for unbalanced parentheses, missing quotes, and undefined variables.

GSU6816	**creating temporary file name**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6817	**temporary file *filename***

Explanation:
A system error indicating the cause is displayed with this message.

GSU6818	**Missing number of hops after "-h"**

Explanation:
You specified the **–h** option without an argument.

User response:
Provide the missing argument.

GSU6819	**Missing address after "-r"**

Explanation:
You specified the **–r** option without an address argument.

User response:
Provide the missing address argument.

GSU6820	**Missing subject after "-s"**

Explanation:
You specified the **–s** option without providing a subject string as an argument.

User response:
Provide the missing subject string.

GSU6821	**Missing user after "-u"**

Explanation:
You specified the **–u** option without a user name argument.

User response:
Provide the missing user name.

GSU6822	**Options applying only to interactive use were given.**

Explanation:
You specified the **–e**, **–f**, **–H**, **–N**, or **–u** options when attempting to send mail. These options are only for use when reading mail.

User response:
Check the description of **mailx**, "mailx — Send or receive electronic mail" on page 180, for usable options when sending mail.

GSU6826	**variable storage**

Explanation:

A system error indicating the cause is displayed with this message.

GSU6827   *string*: **read-only variable**

Explanation:
You cannot change the values of some environment variables, such as **HOME** and **MAILRC**, from within **mailx**. You tried to change the value of such a variable.

User response:
Do not try to change the value of read-only variables.

GSU6828   *string*: **no such variable**

Explanation:
You tried to make use of a variable that does not exist.

User response:
Check to make sure that you have spelled the variable name correctly or define the variable with a **set** command.

GSU6829   **building pathname** *pathname*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6830   **Misplaced shell meta-character**

Explanation:
You provided an invalid file name pattern.

User response:
Ensure that the pattern given is correct.

GSU6831   **Expansion memory allocation failure**

Explanation:
The system could not allocate sufficient system resources to perform the requested operation.

User response:
Free up more resources.

GSU6832   **Shell syntax error**

Explanation:
You provided an invalid file name pattern.

User response:
Ensure that the pattern given is correct.

GSU6833   **Ambiguous**

Explanation:
You provided a file name pattern that expanded into more than one file name.

User response:
Be more specific in naming the file you want.

GSU6834   **alias storage**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6835   *string*: **no such alias**

Explanation:
You tried to unalias *alias*; however, no alias with this name exists.

User response:
Make sure that you spelled *alias* correctly, or specify an alias that does exist.

GSU6859   **pipe through command failed**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6860   **Missing file name**

Explanation:
You issued a command which requires a file name without providing one.

User response:
Specify a file name.

GSU6861   **Missing pipe command**

Explanation:
You specified the **~|** command without providing a shell command.

User response:
Provide the missing shell command.

GSU6865   **mail to command** *command_name*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6866   **mail to file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6868   **remote from** *remote_term*

GSU6869   **No recipients specified**

Explanation:
You tried to send a mail message without specifying any recipients.

User response:
When sending mail, please specify recipients either on the command line or on the carbon copy (or blind carbon copy) list.

GSU6879   *string*: **no matching "if" statement**

Explanation:
You issued an **else** or **endif** command without a corresponding **if** command.

User response:
Ensure that all **else** and **endif** commands are preceded by an **if** command.

GSU6880   **EOF inside "if" statement**

Explanation:
While processing an **if** command, **mailx** encountered an end-of-file condition.

User response:
If the **if** command is in your start-up file, ensure that you have included a corresponding **endif** command. If you are entering the **if** in command mode, do not enter the EOF character before issuing the **endif** command.

GSU6881   **cannot lock file** *filename*

Explanation:
mailx was unable to acquire exclusive access to a mail folder.

User response:
Wait for a little while and try again.

GSU6882   **rewriting** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6883   **allocating message header**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6884   **allocating message address**

Explanation:
A system error indicating the cause is displayed with this message.

GSU6900   **Invalid message number**

Explanation:
You used a message number of 0 or one that is greater than the number of messages in the mailbox.

User response:
Use a message number in the range from 1 to the number of messages in the mailbox.

GSU6901   **Inappropriate message**

Explanation:
You tried to perform a command on an inappropriate message. For example, you tried to undelete a message that was not deleted or you tried to respond to a deleted message.

User response:
Check the description of the command you are using to ensure that you are using it correctly.

GSU6903   **Referencing before first message**

Explanation:
You used the **-** notation to try to reference the message before the first one in the mailbox.

User response:
Do not use **-** when the current message is the first message in the mailbox.

GSU6904   **Referencing beyond last message**

Explanation:
You used the **+** notation to try to reference the next message when the current message was the last one in the mailbox.

User response:
Do not use **+** when the current message is the last message in the mailbox.

GSU6905   **Non-numeric second argument**

Explanation:
The second argument in a message list was not numeric.

User response:
Ensure that, when specifying a range of messages as arguments for a command, you indicate the first and last message in the range with integers in the range 1 to the number of messages in the current mailbox.

GSU6906   **No args expected**

Explanation:
You specified arguments for a command that does not take arguments.

User response:
Do not specify arguments for this command.

GSU6907   **Only one arg allowed**

Explanation:
You tried to use a command that takes only one argument, but you specified either more or less than one argument.

User response:
Specify only one argument for this command.

GSU6908   **Variable "cmd" not set.**

Explanation:
You tried to use the **pipe** command without specifying a shell command to pipe the messages through and the variable **cmd** was not set.

User response:
Either specify a shell command with **pipe** or set the **cmd** to a default shell command to use with **pipe** when no shell command is explicitly specified.

GSU6909   **command** *command*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6911   **No value set for "folder" variable**

Explanation:

You have not provided a value for the **mailx** variable **folder**.

User response:
Provide a value for the variable **folder** either in the start-up file or in command mode.

GSU6912	No previous file.

Explanation:
You used **#** to represent the file name of the previous file when there was no previous file.

User response:
Use a different file name indicator.

GSU6934	Cannot nest "if"s

Explanation:
You tried to nest one **if** command within another.

User response:
Do not nest **if** commands.

GSU6935	if: "s" or "r" are permissible arguments

Explanation:
You used an argument other than **r** or **s** with the **if** command.

User response:
Use only **r** or **w** as the argument for an **if** command.

GSU6939	Missing file after source command

Explanation:
You issued a **source** without specifying a file name.

User response:
Specify a file name with the **source** command.

GSU6940	command file *cmdfile_name*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6941	Usage: unalias name ...

Explanation:
The **unalias** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6942	Usage: unset variable ...

Explanation:
The **unset** command entered was not syntactically correct.

User response:

The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6954	*string*: cannot find out who you are

Explanation:
The **mailx** command was unable to find your user ID.

User response:
Check with your system programmer.

GSU6956	copy buffer

Explanation:
A system error indicating the cause is displayed with this message.

GSU6974	Usage: mkdir [-m mode] [-p] directory ...

Explanation:
The **mkdir** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU6975	fifo file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6977	Missing major/minor device

Explanation:
You failed to specify the major or minor device type argument for a character or block special file.

User response:
Provide the missing argument.

GSU6978	character special file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6979	block special file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU6980	Usage: mknod name p mknod name [bc] major minor

Explanation:
The **command** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7001   **su: The user database does not contain an entry for this user.**

User response:
Contact the system programmer.

GSU7002   **su: Unable to execute the shell.**

Explanation:
The initial program (shell) was not run. Verify that the initial program (shell) exists on this system and that the user has permission to execute it.

GSU7003   **su: User not authorized to obtain superuser authority.**

Explanation:
The user ID issuing the su command does not have the proper authorization to obtain superuser authority.

User response:
Contact the system programmer.

GSU7004   **su: Unable to set up the user environment. Processing terminates**

Explanation:
The environment variables required by the shell have not been set up.

System action:
Processing terminates.

User response:
Contact the system programmer.

GSU7005   **Usage: su**

Explanation:
The **su** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7033   *string*: **unable to find your user name**

Explanation:
newgrp was unable to find your user name in the system user database.

User response:
Contact your system programmer.

GSU7034   *string*: **unknown group** *command:*

Explanation:
You specified a *groupname* that was not in the system group database.

User response:
Use the **id** command to get a list of all groups you may access.

GSU7035   **setgroups call failed**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7036   **set group ID to** *groupname*

Explanation:
A system error indicating the cause is displayed with this message.

GSU7037   **setuid**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7038   **exec default shell** *shell*

Explanation:
A system error indicating the cause is displayed with this message.

GSU7041   **Usage: newgrp [-[l]] [group]**

Explanation:
The **newgrp** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7056   **Usage: nohup command [argument ...]**

Explanation:
The **nohup** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7057   **Unknown format character "*character*"**

Explanation:
You specified an unrecognized format character as an argument to the **–t** option.

User response:
Check the description of **od**, "od -- Dump a file in a specified format" on page 232, for a list of valid format characters.

GSU7058   **Invalid size modifier for "*character*" format**

Explanation:
You specified an invalid size modifier for the *char* format character.

User response:

Check the description of **od**, "od -- Dump a file in a specified format" on page 232, for the valid size modifiers for each format character.

GSU7059   **Missing argument for "-*character*" option**

Explanation:
You specified the *–opt* option but did not follow it with the expected argument.

User response:
Provide the expected argument. Check the description of **od**, "od -- Dump a file in a specified format" on page 232, for a list of valid options and their arguments.

GSU7060   **seek error on input**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7061   **od: badly formed offset "*string*"**

Explanation:
You specified an *offset* that was not a decimal or octal value.

User response:
Specify a valid *offset*.

GSU7062   **od: offset must be multiple of 512**

Explanation:
You specified an *offset* value that was not a multiple of 512.

User response:
Specify an *offset* value that is a multiple of 512.

GSU7063   **too many output formats, (maximum *number*)**

Explanation:
You specified too many output formats on the **od** command line. The maximum number of output formats is *num*.

User response:
Do not specify more than *num* output formats on the **od** command line.

GSU7064   **Usage: od [-v] [-A doxn] [-N #] [-j #[bkm]] [-t acdfoux[#]] [file ...] od [-bcdhosvxDOSX] [file] [[+]offset[.] [b]]**

Explanation:
The **od** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7079   **truncate file "*string*" to length *number* failed**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7081   **tempnam() error**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7082   **tempfile error on "*string*"**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7105   **Usage: paste [-d list] [-s] file ...**

Explanation:
The **paste** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7106   **cannot determine OPEN_MAX**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7107   **paste: must specify input files**

Explanation:
You did not specify any input files.

User response:
Specify at least one input file.

GSU7108   **paste: too many files at *filename***

Explanation:
You specified more files than **paste** can handle. *filename* is the first file that **paste** was unable to open. The number of files that **paste** can open depends on the number of files that other processes have open.

User response:
Close files that other processes have open to increase the number of files that **paste** can open.

GSU7128   **Usage: pathchk [-p] pathname ...**

Explanation:
The **pathchk** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7139   *string*: compress: *command:*

Explanation:
A problem occurred in the compression of the archive.

User response:
Check whether a file is not a regular file, has other links, or if there is not enough space for compression tables.

GSU7140   *string*: decompress: *command:*

Explanation:
Normally implies that the archive is corrupted.

User response:
Obtain a new copy of the archive file.

GSU7141   **Unable to open terminal** *term*

Explanation:
A system error indicating the cause is displayed with this message.

GSU7142   **Existing file** *filename* **is newer**

Explanation:
filename was not extracted from the archive because an existing file with the same name was newer.

User response:
If you really want to extract *filename*, use the **–u** option.

GSU7143   **cannot create parent directory to** *pathname*

Explanation:
A system error indicating the cause is displayed with this message.

GSU7144   **cannot link** *name1* **to** *name2*

Explanation:
A system error indicating the cause is displayed with this message.

GSU7145   *string*: *command:* **links not supported by OS**

Explanation:
The **pax**, **tar**, or **cpio** file being extracted contained hard links, which are not supported by the operating system. These files are not extracted.

User response:
Since this message appears only on systems with no hard link support, there is no way to extract the file as a hard link. One can manually make a copy of the file referenced by the link.

GSU7146   *string* **symbolic link to** *name1*

Explanation:
A system error indicating the cause is displayed with this message.

GSU7147   *string* **link to** *name1*

Explanation:
A system error indicating the cause is displayed with this message.

GSU7148   **cannot create file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU7149   *string*: **Invalid file name** *command:*

GSU7150   **; converted to** *newname*

GSU7151   **I/O buffer allocation**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7152   *string*: **-6 not supported**

Explanation:
You specified the **–6** option, which is not currently implemented.

User response:
Do not use the **–6** option.

GSU7153   *string*: **Unknown option** *command:*

Explanation:
You specified an option that is not valid for this command.

User response:
Check the description in this book for the command you were using to find the valid list of options for that command.

GSU7154   **Must specify one of -i, -o, or -p**

Explanation:
When using **cpio**, you must specify one and only one of the **–i**, **–o**, or **–p** options.

User response:
Specify one of the required options.

GSU7155   *string*: **-r option disabled with -p**

Explanation:
cpio cannot use the **–r** option (rename files) with **–p** (pass, which copies files from one location to another directory).

User response:
When using the **–p** option, do not also specify the **–r** option.

GSU7156   **Usage: cpio -o [aBcvz] [-yV volmask] [-C blocksize] [-O file] cpio -i [bBcdfmrsStuv6qz] [-yV volmask] [-C blocksize] [-I file]**

[patterns] cpio -p [Badlmruv] directory

Explanation:
The **cpio** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

| **GSU7158** | *string*: **bad magic number in archive** |

Explanation:
Either the wrong file was passed, or the file has been corrupted.

User response:
Check your archive file.

| **GSU7159** | *string*: **unsupported file mode** *filename* |

Explanation:
When creating a **cpio** archive, the mode (file type) of the specified file on the file system is not valid for including in a **cpio** archive.

User response:
Check the file type of the named file, and correct if possible. (The file type may be an extension to POSIX, which is valid on the host operating System, but shouldn't be included in a portable cpio archive.)

| **GSU7160** | *string*: **Unknown mode field** *filename* |

Explanation:
The type of the file specified in the *mode* field of the **cpio** archive is not supported by the operating system.

User response:
No action possible, as the file cannot exist on the host system.

| **GSU7161** | *string*: *command:* **not found.** |

Explanation:
You specified the name of an archive member, but it was not found in the archive.

User response:
Get a full table of contents of the archive to see if you are using the correct name.

| **GSU7162** | *string*: **file** *filename* **Unable to represent filename in ISO/IEC 8859 -- not saved** |

Explanation:
Characters in *filename* cannot be represented in the character set used in **tar** archives.

User response:

Rename the specified file to contain only characters in ISO/IEC 8859.

| **GSU7163** | **symbolic link** *filename* |

Explanation:
A system error indicating the cause is displayed with this message.

| **GSU7164** | *string*: **bad format in header** |

Explanation:
Either the wrong file was passed, or the file has been corrupted.

User response:
Check your archive file.

| **GSU7165** | *string*: **archive file name** *command:* **characters** |

Explanation:
The archive contained a path name that was longer than that permitted on the local system.

User response:
Re-create the archive using a shorter relative path.

| **GSU7167** | *string*: **interactive EOF** |

Explanation:
When using the interactive rename option, an end-of-file was encountered.

User response:
None.

| **GSU7169** | *string*: **cannot set access/modify time on** *command:* |

| **GSU7170** | *string*: **cannot set mode** |

Explanation:
A system error indicating the cause is displayed with this message.

| **GSU7171** | *string*: **cannot set uid/gid** |

Explanation:
A system error indicating the cause is displayed with this message.

| **GSU7172** | **Warning: file** *filename* **character** *hexnum* |

| **GSU7173** | *string*: **out of memory for link tables** |

Explanation:
There were not enough free system resources to create the needed link tables. When archiving files with multiple links, each link must be remembered.

User response:
Archive in smaller pieces.

| **GSU7174** | *string*: **missing** *command: num* **to** *s* |

Explanation:

Not all links to a given file were archived.

User response:
None.

GSU7175	Missing file characteristics after "p"

Explanation:
You specified the **–p** option without providing an argument that indicated the file characteristics to be preserved.

User response:
Provide the missing argument.

GSU7176	Missing blocking factor after "b"

Explanation:
You specified the **–b** option without providing an argument that indicated the size of an output block.

User response:
Provide the missing block size.

GSU7177	Missing filename after "f"

Explanation:
You specified the **–f** option without providing a file name as an argument.

User response:
Provide the missing file name.

GSU7178	Missing substitution after "s"

Explanation:
You specified the **–s** option without providing a substitution command as an argument.

User response:
Provide the missing substitution command.

GSU7179	Missing format after "x"

Explanation:
You specified the **–x** option without providing an archive format as its argument.

User response:
Provide the missing archive format.

GSU7180	Missing volume pattern after "V"

Explanation:
You specified the **–V** option without providing a volume pattern as an argument.

User response:
Provide the missing volume pattern.

GSU7181	Missing keyword list after "o"

Explanation:
You specified the **–o** option without providing a keyword list as an argument.

User response:
Provide the missing keyword list.

GSU7182	Unable to convert from codeset *codeset1* to codeset *codeset2*

Explanation:
The host file name character set (*codeset1*) was unable to map to and from the archive character set (*codeset2*).

User response:
Correctly specify **to** and **from** keywords with the **–o** option of **pax**.

GSU7183	*string*: cannot read archive from terminal

Explanation:
You tried to extract or list an archive from a tty device.

User response:
Specify a non-tty file name to **pax**.

GSU7184	*string*: can't use -a or -u with stdout

Explanation:
You specified the **–a** or **–u** option when sending output to the standard output. These options can only be used with archive files.

User response:
Make sure that you specify an archive file when using the **–a** or **–u** options.

GSU7185	*string*: blocking factor must be at least 512 for read operations

Explanation:
You specified a blocking factor of less than 512 bytes and attempted to perform read operations.

User response:
Use the **–b** option to specify a block size of at least 512 bytes.

GSU7187	Usage: pax [-cdnv] [-f archive] [-s replstr] [-Lz] [-V volpattern] [pattern ...]

Explanation:
The **pax** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7188	pax -r [-cdiknuv] [-f archive] [-s replstr] [-p [rk.aemop]] [-zq]
GSU7189	[-o [from=codeset,][to=codeset]] [-V volpattern] [pattern ...]
GSU7190	pax -w [-dituvX] [-b blocking] [-a] [-f archive]] [-s replstr]

GSU7191	**[-o [from=codeset,][to=codeset]] [-x format] [-V volpattern]**

GSU7192	**[-Lzq] [pathname ...]**

GSU7194	*string*: **badly formed number:** *command:*

Explanation:
You specified an invalid number as the argument of a **–b** option.

User response:
Specify a valid number.

GSU7195	*string*: **overflow in blocking factor:** *command:*

Explanation:
You specified a *blocksize* argument to the **–b** option that was too large.

User response:
Use a smaller value for *blocksize*

GSU7196	*string*: **blocking factor of 0 not allowed**

Explanation:
You specified the **–b** option with an argument that evaluated to zero.

User response:
Specify a nonzero value as the argument to the **–b** option.

GSU7197	*string*: **invalid archive format selected:** *command:*

Explanation:
You specified an argument to the **–x** option that is not a supported format.

User response:
Check the description of **pax**, "pax -- Interchange portable archives" on page 239, for a list of supported archive formats.

GSU7199	*string*: **medium not seekable**

Explanation:
You tried to append to an archive that was not seekable.

User response:
Create a new archive instead.

GSU7200	*string*: *command:* **Not a directory**

Explanation:
You specified pass mode with either the **–p** option for **cpio** or the **–r** and **–w** options for **pax**, but the destination given was not a directory.

User response:
Make sure that the destination *pathname* is a directory.

GSU7203	*string*: **error in string replacement:** *command:*

GSU7204	**Warning: blocking factor** *blocksize* **not portable to UNIX**

Explanation:
You specified a blocking factor (*blocksize*) that was larger than 20. This may create an archive that does not work on a UNIX system.

User response:
To guarantee portability to a UNIX system, use a *blocksize* of 20 or less. A larger value may work but is not guaranteed.

GSU7205	**TAR file already set**

Explanation:
You specified the **f** option more than once on the command line.

User response:
Specify the **f** option only once.

GSU7206	**Must specify one of 'c', 'r', 't', 'u', or 'x'**

Explanation:
tar requires that you specify one of the **c**, **r**, **t**, **u**, or **x** options as the first character of its option string. You did not do this.

User response:
Specify one of the required options at the beginning of the option string.

GSU7207	*string*: **"z" (compress) option unavailable with** *command:*

GSU7208	**opening archive** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU7209	*string*: **tape archive medium not seekable**

Explanation:
You tried to use the replace (**r**) option on an archive file that was not seekable.

User response:
Only use the **r** option with archive files that are seekable.

GSU7210	*string*: **'u' function not implemented--using 'r'**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7211	**chdir to** *pathname*

Explanation:

A system error indicating the cause is displayed with this message.

GSU7216	symbolic link to *filename*

GSU7218	*string*: *command:* **name too long**

Explanation:
The path name *filename* was too long to be included in a tar archive.

User response:
Rename or move *filename*, such that its path name is shorter.

GSU7219	*string*: *command:* **Unknown mode** *filename*

Explanation:
The type of the file specified in the *mode* field of the **tar** archive entry is not supported by the operating system.

User response:
No action possible, as the file cannot exist on the host system.

GSU7220	*string*: *command:* **is a special file-- not dumped**

Explanation:
You tried to dump a special file *filename* when writing a non-USTAR **tar** file.

User response:
Do not specify special files to be included in **tar** archives. If you want to archive special files, use a USTAR format archive.

GSU7221	*string*: **file** *command:* **hard link to** *name1* **ignored: tar format does not permit links to pathnames longer than** *name2*

GSU7222	*string*: **symbolic link** *command:* **tar format does not permit symbolic links to pathnames longer than** *filename*

GSU7223	**tape read**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7224	*string*: **incomplete tape block**

Explanation:
When reading a tape archive header, a block that was not the same size as the archive block size was read.

User response:
Check to see if the archive was corrupted.

GSU7226	*string*: *command:* **name too long... switching to USTAR format**

GSU7227	*string*: *command:* **grew in size**

GSU7228	*string*: *command:* **shrank in size**

GSU7229	**scratch file**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7230	**sorting**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7231	**updating archive**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7263	**insufficient memory**

Explanation:
There were not enough free system resources to perform the requested operation.

User response:
Free up more system resources.

GSU7266	**Width is insufficient**

Explanation:
The line was not wide enough to hold the given number of columns with the given column width; or a column was not wide enough to hold the minimum amount of data.

User response:
Use the **–w** option to increase the width of the page.

GSU7267	**Too many files for merge(-m) option; limit** *num*

Explanation:
You specified too many files for the **–m** option to handle. The limit was *num* files.

User response:
Specify fewer files.

GSU7273	*string*: **can't access your terminal**

Explanation:
The terminal associated with the standard input was not accessible.

User response:
When running **ps** without a controlling terminal, you must specify one of the **–G**, **–g**, **–p**, **–t –U**, or **–u** options to identify the processes on which **ps** is to report.

GSU7274	**process table**

Explanation:

A system error indicating the cause is displayed with this message.

GSU7275   *string*: **no matching processes found.**

Explanation:
ps did not find any find any processes which matched the specified search criteria.

User response:
Confirm the command options for **ps**.

GSU7276   *string*: **badly constructed format string** *format*

Explanation:
The output format string was not correct.

User response:
Check the description of **ps**, "ps — Return the status of a process" on page 254, for a list of possible format specifications.

GSU7277   *string*: **more than** *command:* **items in** *num* **list at** *option*

Explanation:
You specified more than *num* items in the list given as an argument to –*option*.

User response:
Specify no more than *num* items in the –*option* argument list.

GSU7280   **rm: not allowed to remove** *filename*

Explanation:
You specified either **.** (current directory) or **..** (parent directory) as a *pathname*. **rm** will not remove these directories.

User response:
Do not specify **.** or **..** as a *pathname*.

GSU7282   **rm: use "-r" to remove directory** *pathname*

Explanation:
You tried to use **rm** to remove a directory without specifying the **–r** option.

User response:
Specify the **–r** option when you want to use **rm** to remove a directory.

GSU7283   **rm: fatal error during "-r" option**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7286   **cannot stat entry** *filename*

Explanation:

A system error indicating the cause is displayed with this message.

GSU7287   **cannot open directory** *pathname*

Explanation:
A system error indicating the cause is displayed with this message.

GSU7288   **Usage: rm [-firR] file ...**

Explanation:
The **rm** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7290   **insufficient memory for buffers**

Explanation:
There were not enough free system resources to allocate as buffers.

User response:
Free up more resources.

GSU7291   **"y" command may not be followed by a newline**

Explanation:
You followed the **y** command with a newline rather than its required arguments.

User response:
Provide the missing arguments.

GSU7292   **in y command, the strings** *set1* **and** *set2* **must be the same length**

Explanation:
You specified two strings *set1* and *set2* as arguments to the **y** command that were not the same length.

User response:
Make sure that the *set1* and *set2* strings are the same length.

GSU7293   **non-matching "{" and "}" commands**

Explanation:
You specified a **{** command without the matching **}**.

User response:
Provide the missing **}**.

GSU7294   **garbage after command**

Explanation:
You specified invalid characters after a script command.

User response:
Remove the surplus characters.

GSU7295   *number* **addresses given for command expecting at most** *num1*

Explanation:
You specified a command with *num1* addresses that uses a maximum of *num2* addresses.

User response:
Use the correct number of addresses.

GSU7296   **newline or end of file found in pattern**

Explanation:
sed encountered a newline or end-of-file character when reading a pattern from the script or script file.

User response:
Check the pattern for a missing delimiter.

GSU7297   **Missing script**

Explanation:
You specified the **–e** option but did not provide a script as its argument.

User response:
Provide the missing script.

GSU7299   **label** *label* **not found in script**

Explanation:
You specified *label* as an argument to the **b** or **t** command, but *label* does not exist in the script.

User response:
Make sure that **b** and **t** commands refer to labels that exist in the script.

GSU7300   **cannot nest "!" command**

Explanation:
You tried to execute one **!** command from within another.

User response:
Remove any nested **!** commands.

GSU7301   **"\" must terminate the "***character***" command**

Explanation:
You specified the *cmd* command, but you did not provide the backslash (****) required to terminate its input.

User response:
Provide the missing ****.

GSU7302   **End of file in** *cmd* **command**

Explanation:
sed encountered an end of file while parsing the command *cmd*.

User response:

Check the script file for missing closing quotes, missing regular expression delimiters, and other syntactical errors.

GSU7303   **"***character***" command needs a label**

Explanation:
You specified a command that requires a label as an argument, but you did not provide the label name.

User response:
Provide the missing label name.

GSU7304   **sysconf(_SC_OPEN_MAX) failed**

GSU7305   **no memory file file table**

Explanation:
There were not enough free system resources to perform the requested operation.

User response:
Free up more resources.

GSU7306   **badly formed file name for** *cmd* **command**

Explanation:
You specified the *cmd* which requires a file name as an argument, but the given argument does not have the syntax of a file name.

User response:
Specify a valid file name.

GSU7308   **insufficient memory to compile command**

Explanation:
There were not enough free system resources for **sed** to compile a given command.

User response:
Free up more resources.

GSU7309   **bad regular expression delimiter after **

Explanation:
You used a backslash (****) to indicate an alternate regular expression delimiter, but you did not follow it with a valid delimiter.

User response:
Provide a valid delimiter following the **** (that is, any character other than newline, space, tab, or EOF).

GSU7310   **no remembered regular expression**

Explanation:
You issued a command that tried to use a remembered regular expression, but there was no remembered regular expression.

User response:

Specify the regular expression explicitly.

GSU7311   **script file** *filename*

Explanation:
A system error indicating the cause is displayed with
this message.

GSU7312   **Usage: sed [-n] script [file...] sed [-
                       nE] [-e script] [-f scriptfile] [file...]**

Explanation:
The **sed** command entered was not syntactically
correct.

User response:
The usage message displays the correct syntax for
this command. Reenter the command with the correct
syntax.

GSU7313   **sed: regular expression error:**
                       regular_expression_error

Explanation:
You have entered a regular expression incorrectly. See
the regular expressions (regexp) section, Appendix B,
"Regular Expressions (regexp)," on page 471.

GSU7315   *string*: **restricted**

Explanation:
You were using the restricted version of the shell
(for example, by specifying the **–r** option for **sh**). The
restricted shell does not allow the use of the specified
command.

User response:
To use the specified command, you must be using a
nonrestricted shell.

GSU7316   *string*: **readonly variable**

Explanation:
You tried to change or remove the variable *name* which
was marked as read-only.

User response:
Do not attempt to change or remove a read-only
variable.

GSU7317   **temporary file** *filename* **error using
                       here document**

Explanation:
A system error indicating the cause is displayed with
this message.

GSU7318   **!cannot open script** *filename*

Explanation:
A system error indicating the cause is displayed with
this message.

GSU7320   **missing command after -c**

Explanation:

You specified the **–c** option but did not provide a
command as an argument.

User response:
Provide the missing command.

GSU7321   **Unknown option** *option*

Explanation:
You specified an option that is not valid for this
command.

User response:
Check the description in this book for the command
you were using to find the valid list of options for that
command.

GSU7325   **return: not executing function**

Explanation:
You specified a **return** command when you were not
executing a function.

User response:
Only use **return** to return from a function.

GSU7326   **!reading script**

Explanation:
A system error indicating the cause is displayed with
this message.

GSU7327   **signal number** *num* **not
                       conventional**

Explanation:
You specified a conventional signal number *num*, but
this system does not use conventional signal numbers.

User response:
Use the corresponding signal name.

GSU7328   **too many outstanding signals**

GSU7329   **<(command) and >(command) not
                       implemented**

Explanation:
You tried to use <(command) or >(command), which
are not implemented in this version of the shell.

User response:
Do not use these constructs.

GSU7330   **<<***string* **unclosed**

Explanation:
The shell encountered an end-of-file character while
reading a here-document before it encountered *name*.

User response:
Make sure that *name* appears in the text of the input
file.

GSU7331   **too many << in line**

Explanation:
You specified more than 10 here documents using **<<**.

User response:
Simplify your command line to use fewer here documents.

GSU7332   **syntax error: got *string1* expecting *string2***

Explanation:
When processing your input, the shell encountered *string1* when it was expecting *string2*.

User response:
Check the description of **sh**, "sh — Invoke a shell" on page 277, for the correct syntax for various shell commands. Reenter your input with the correct syntax.

GSU7334   **not an identifier**

Explanation:
You specified a **for**, **function**, or **select** statement, but did not follow it with a valid identifier.

User response:
Provide a valid identifier after the statement.

GSU7338   **execute: internal error *num***

GSU7339   **ambiguous redirection**

Explanation:
You specified a file name in a redirection construct that expands to other than a single word.

User response:
Ensure that the file name in a redirection construct expands to a single word.

GSU7340   **file descriptor *fd* already redirected**

Explanation:
You tried to redirect the file descriptor *fd*, which was already being redirected in the same command.

User response:
Only redirect a file descriptor once.

GSU7341   **bad file descriptor *fd***

Explanation:
You tried to read from, or write to, the file descriptor *fd*, which was not open for that operation.

User response:
Open the file descriptor *fd* for the appropriate operation.

GSU7342   **file *filename* already exists**

Explanation:
You tried to redirect output into an existing file, but you have turned on the **noclobber** option (see **set 1**).

User response:
Use the construct **>|***filename* to redirect the output into an existing file or turn the **noclobber** option off with **set +o noclobber**.

GSU7343   **!cannot open *filename* for *input/ output***

Explanation:
A system error indicating the cause is displayed with this message.

GSU7345   **no file descriptor available**

Explanation:
You tried to redirect a file descriptor but none were available. When a file descriptor is redirected, the old value is remembered by the shell by duplicating it to yet another file descriptor. The total number of file descriptors is limited by the system and hence the shell may run out while it looks like your command is using far fewer than the maximum number of descriptors.

User response:
Free up a file descriptor.

GSU7346   **!no pipes available**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7347   **!cannot open *filename***

Explanation:
A system error indicating the cause is displayed with this message.

GSU7349   **only one co-process allowed**

Explanation:
You tried to create more than one coprocess.

User response:
Do not attempt to create more than one coprocess.

GSU7350   **e_cmd: negative result?**

GSU7351   **not found**

Explanation:
You tried to execute a command that could not be found.

User response:
Ensure that the command exists and that the **PATH** environment variable is valid.

GSU7352   **recursion too deep**

Explanation:
You defined a function that has too many levels of recursion.

User response:
Simplify the function to use fewer levels of recursion.

GSU7353   **Usage: *command:***

Explanation:

The **mv** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7354   **"***string***" is not an identifier**

Explanation:
You tried to use an nonalphanumeric *name* as an identifier.

User response:
Use only alphanumeric names for identifiers.

GSU7355   **bad file descriptor *fd***

Explanation:
You tried to read from, or write to, the file descriptor *fd* which was not open for that operation.

User response:
Open the file descriptor *fd* for the appropriate operation.

GSU7356   **history not available**

Explanation:
The shell was unable to open a history file when you logged in.

User response:
Make sure that the environment variable **HISTFILE** is set to a file which is named properly and for which you have appropriate permissions. You may have to log in again.

GSU7357   **no active co-process**

Explanation:
You tried to receive input from or send output to a coprocess when there was no active coprocess.

User response:
Do not use the **-p** option when there is no active coprocess.

GSU7358   **no HOME directory**

Explanation:
You tried to use **cd** to return to your home directory; however, the environment variable **HOME** was not defined.

User response:
Set the environment variable **HOME** to the path name of your home directory.

GSU7359   **no previous directory**

Explanation:
You tried to use the command **cd –** to return to your previous working directory; however, there was no record of what your previous directory was.

User response:
Specify the desired directory explicitly.

GSU7360   **pattern *old* not found in *dir***

Explanation:
You tried a command of the form **cd** *old new*. However, the name of the current directory *dir* does not contain any string matching the regular expression *old*.

User response:
Ensure that the name of the current directory contains the regular expression *old*.

GSU7362   **!writing**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7363   **!reading**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7364   **bad shift count *expr***

Explanation:
You specified an expression that did not evaluate to a number in the range from 0 to the number of remaining positional parameters.

User response:
Specify an expression that evaluates to a number in the range from 0 to the number of remaining positional parameters.

GSU7366   **"***string***" not a valid trap name**

Explanation:
You specified an unrecognized trap name.

User response:
Check that you spelled the trap name correctly.

GSU7367   **base *number* not in [2,36]**

Explanation:
You used the **–i** option to specify a base for an integer, but the base was not in the range from 2 to 36.

User response:
Specify a base in the range from 2 to 36.

GSU7369   **Cannot restore privileged state**

Explanation:
You specified **+o privileged** or **–p**, but for some reason, the shell cannot restore the affected values.

User response:
Exit the current shell and start a new one.

GSU7370   **. file [arg ...]**

GSU7371   **alias [-tx] [name[=value] ...]**

GSU7372   **break [count]**

GSU7373	**cd [dir] cd old new**
GSU7374	**command [-pvV] command ...**
GSU7376	**continue [count]**
GSU7378	**eval [arg ...]**
GSU7379	**exec [command ...]**
GSU7380	**exit [value]**
GSU7381	**export [-p] [name[=value] ...]**
GSU7383	**fc [-e editor] [-lnr] [first [last]] fc -s [old=new] [num]**
GSU7384	**getopts opts name [arg ...]**
GSU7385	**jobs [-lnp] [job ...]**
GSU7386	**kill -l [status] kill [-s SIGNAL] job ...**
GSU7387	**let expression ...**
GSU7389	**print [-nprRsu[n]] [arg ...]**
GSU7390	**read [-prsu[n]] [name?prompt] [name ...]**
GSU7391	**readonly [-p] [name[=value] ...]**
GSU7392	**return [value]**
GSU7394	**shift [count]**
GSU7396	**time [-p] command ...**
GSU7397	**times [-p]**
GSU7398	**trap [action] [name ...]**
GSU7400	**type command ...**
GSU7401	**typeset [-f[tux]] [+-lrtuxH] [+-LRZi[n]] [name[=value] ...]**
GSU7403	**unalias [-a] [name ...]**
GSU7404	**unset [-f] [name ...]**
GSU7405	**umask [-S] [mode]**
GSU7406	**wait [job ...]**
GSU7407	**whence [-v] command ...**
GSU7409	**fg [job]**
GSU7410	**insufficient memory available**
GSU7411	**cannot create temporary file**

Explanation:
A temporary file was required to perform the requested operations. The shell was unable to create this file, for example, because the disk was full.

User response:
Free up more disk space.

GSU7412	**no command matches *command***

Explanation:
You asked to edit a command beginning with a particular *string*, but there was no such command in the history file.

User response:
Use the **fc** command to browse through the history file to ensure that *string* is entered correctly.

GSU7421	***string* is corrupt**
GSU7422	***string* is not found**

Explanation:
You specified a *name* that **type** was unable to find.

User response:
Check that the *name* exists, was spelled properly, and that you have the appropriate permissions.

GSU7423	**stack overflow**

Explanation:
You specified an expression that was too complicated for the stack to handle.

User response:
Try simplifying the expression.

GSU7424	**misplaced subscript**

Explanation:
The subscript for an array was missing or invalid.

User response:
Make sure that you provide a valid subscript for the array.

GSU7425	**unknown operator**

Explanation:
You specified an unknown operator.

User response:
Check the *Arithmetic Substitution* subsection of the description of **sh**, "sh — Invoke a shell" on page 277, for a table showing the valid arithmetic operators.

GSU7426	**base must be in [2,36]**

Explanation:
You specified a base that was not in the range 2 to 36.

User response:
Specify a base in the 2 to 36 range.

GSU7427	**unmatched ? :**

Explanation:
You specified the **?** operator without the **:**.

User response:
Specify the missing **:**.

GSU7428	**expression: internal error**
GSU7429	**assign only to variable**

Explanation:

You specified an assignment where the left hand side that was not a variable.

User response:
Only use the assignment operators to assign values to variables.

GSU7430	*string* in arithmetic expression "*string*"

Explanation:
An *error* occurred in the arithmetic expression *expr*.

User response:
Look up *error* in the error listing for more details.

GSU7431	*string* in arithmetic expression "*string*" near *character*

Explanation:
An error occurred in the arithmetic expression *expr* near the substring *substr*.

User response:
Look up *error* in the error listing for more details.

GSU7433	*string*: No such job

Explanation:
You specified a nonexistent job identifier.

User response:
Use the **jobs** command to get a list of jobs that are currently job controlled.

GSU7438	job control disabled

Explanation:
You tried to use the **fg** or **bg** command when **set –o monitor** (or **set –m**) was not set.

User response:
Turn on monitor mode with **set –o monitor** or **set –m**. This mode is not supported on all systems.

GSU7439	job *job-id* not job controlled

Explanation:
You specified the job identifier of a job which was not being job controlled.

User response:
Use the **jobs** command to get a list of jobs currently being job controlled.

GSU7440	!cannot continue job

Explanation:
A system error indicating the cause is displayed with this message.

GSU7441	"*string*" not a valid signal

Explanation:
You specified a non-integer signal for **kill** that was not a valid signal name, or you specified a signal that is outside the range of valid signal numbers.

User response:
Make sure that you specify a valid signal number or name for *signal*.

GSU7442	"*string*" is not a job

Explanation:
You specified a job-identifier that is not valid.

User response:
Specify a valid job-identifier.

GSU7443	j_freejob(NULL)!

GSU7449	*string*: Unknown file type field value *pathname*

Explanation:
A file with an invalid file type was encountered when extracting or listing an archive's contents.

User response:
This archive is invalid. No action is possible.

GSU7453	Argument to -n must be numeric

Explanation:
You specified an argument to the **–n** option that was not a number.

User response:
Specify a numeric argument for **–n**.

GSU7455	*string*: warning--file size error in *command:* truncated

GSU7456	*string*: warning--file size error in *command:* padded with spaces

GSU7457	*string*: checksum error on tape (got *command:* expected *hexnun1*

Explanation:
A bad checksum was found in a **tar** header.

User response:
Check to see if the archive was corrupted.

GSU7458	*string*: non-USTAR header in USTAR archive at *command:*

Explanation:
When reading a USTAR format archive, **pax** encountered a header without the USTAR magic number

User response:
Check to see if your archive has been corrupted.

GSU7460	*string*: try "c" option for ASCII archive

Explanation:
The magic number in the **cpio** archive header appeared to be in ASCII.

User response:
Try using the **–x cpio** option instead of **–x cpiob**.

GSU7461   *string*: *command:* **final component of name too long**

Explanation:
The USTAR format extends the old **tar** file name limit from 100 to 256 bytes; however, this requires breaking up the file name into one piece of 156 bytes or less and another piece of 100 bytes or less. The break occurs between directory components (that is, at a slash). In the case of *pathname,* the second component could not be made to fit into 100 bytes.

User response:
Move or rename *pathname* to have shorter path components.

GSU7462   *string*: *command:* **Unable to split name to fit in tar header**

Explanation:
The USTAR format extends the old **tar** file name limit from 100 to 256 bytes; however, this requires breaking up the file name into one piece of 156 bytes or less and another piece of 100 bytes or less. The break occurs between directory components (that is, at a slash). In this case, the characteristics of *pathname* would not allow it to be broken up in such a manner.

User response:
Move or rename *pathname* to have shorter path components.

GSU7463   **Usage: lp [-cmsw] [-d dest] [-n number] [-o printer-option] [-t title] [file...]**

Explanation:
The **lp** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7473   **Missing variable assignment**

Explanation:
You specified the **–v** option but did not follow it with a variable assignment.

User response:
Provide a variable assignment following the **–v** option.

GSU7475   *string*: *command:* **is not a member of group *username***

Explanation:
The user *username* is not included in the list of users who are members of the group *groupname,* and the group did not have a password.

User response:
See your system programmer about adding *username* to the members of *groupname*.

GSU7476   **-f filename invalid if message given**

Explanation:
On the **logger** command line, you used the **–f** to indicate a file from which **logger** is to read log messages; however, you also provided the argument *string* to be used as a log message. You cannot use both methods on the same command line.

User response:
Either specify a file to be read with the **–f** option or provide a log message on the command line, but not both.

GSU7477   **Formatted log message too long -- limit LINE_MAX *num***

Explanation:
The formatted log message was longer than *num* characters. *num* is the value of the configuration variable **LINE_MAX**.

User response:
Shorten your log message.

GSU7478   **Warning: newgrp utility probably not setuid to root.**

GSU7488   **files too large, trying "-h" option ...**

Explanation:
You specified the **–H** option, but there were not enough free system resources to handle the files. **diff** will now try to compare the files using the **–h** option.

User response:
If you are comparing these two files again, specify the **–h** option on the command line for faster operation.

GSU7490   **Missing field separator**

Explanation:
You specified the **–F** option but did not follow it with a field separator.

User response:
Provide a field separator following the **–F** option.

GSU7499   *string*: *command:* **symbolic links not supported by OS**

Explanation:
You tried to extract a file specified as a symbolic link in the archive. Symbolic links are not supported on all operating systems.

User response:
Do not use symbolic links on this system.

GSU7501   *string*: **bad substitution expression: *sub_pattern***

Explanation:

You invoked with a **–s** option, but the *sub_pattern* argument was empty, or did not contain a leading delimiter.

User response:
Specify a valid *sub_pattern* argument, such as **–s** *old/new*.

GSU7502	fifo special file *filename* fifo not supported by local o/s

Explanation:
You tried to extract an archive file containing a FIFO file, and the host operating system does not support FIFOs.

User response:
Since the operating system does not support FIFOs, no action is possible.

GSU7506	Unknown tape density *num*

Explanation:
You specified a tape drive number followed by an argument that should be a tape density, but the argument was not **l**, **m**, or **h**.

User response:
Specify a valid tape density (**l**, **m**, or **h**).

GSU7507	Missing file name after "f"

Explanation:
You specified the **f** option but you did not specify the name of an archive file as its argument.

User response:
Provide the missing file name.

GSU7509	Missing blocking factor after "b"

Explanation:
You specified the **–b** option without providing an argument that indicated the size of an output block.

User response:
Provide the missing block size.

GSU7513	Blocking factor *blocksize* is non-numeric

Explanation:
You specified the **–b** option, but the *blocksize* argument was not a valid number.

User response:
Specify a valid number as the value of *blocksize*.

GSU7514	Blocking factor larger than *number*

Explanation:
You specified the **–b** option, but the *blocksize* argument that you provided is too large.

User response:
Use a smaller *blocksize*.

GSU7517	*string*: -O: Must specify -o option

Explanation:
You specified the **–O** option, but did not specify the **–o** option.

User response:
To use the **–O** option, you must specify the **–o** option.

GSU7518	*string*: bad numeric ID at *command: option*

Explanation:
You specified a numeric identifier for a process, group, or session that was not a valid number.

User response:
Make sure that all numeric IDs are valid numbers.

GSU7519	*string*: unknown user ID at "-u *command:"*

Explanation:
You specified an unknown login name or a bad user ID as an argument to the **–u** option.

User response:
Check the arguments to the **–u** option carefully.

GSU7520	*string*: -I: Must specify -i option

Explanation:
You specified the **–I** option, but did not specify the **–i** option.

User response:
To use the **–I** option, you must specify the **–i** option.

GSU7522	*string*: missing directory after -C

Explanation:
You specified the **–C** but did not provide a directory name as an argument.

User response:
Provide the missing directory name.

GSU7524	Only one character allowed after -t

Explanation:
You specified a field separator that was longer than one character as an argument to the **–t** option.

User response:
Use a one-character field separator.

GSU7673	Profiling timer expiry
GSU7674	CPU time limit
GSU7675	File size limit
GSU7676	I/O possible
GSU7723	-- core dumped

Explanation:
A program called by the shell terminated with a core dump.

User response:

Ensure that the program was called correctly.

GSU7725   **!history**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7726   **!cannot spawn**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7727   **shell metacharacter in wordexp()**

Explanation:
The **wordexp** function was called by the shell to expand a string that contained special shell characters.

User response:
Check the program which calls **wordexp** and ensure that the string to be expanded does not contain any special shell characters.

GSU7728   **bad ${} modifier**

Explanation:
You specified an invalid modifier in a **${}** construct.

User response:
Check the description of **sh**, "sh — Invoke a shell" on page 277, for a list of valid modifiers in **${}** constructs.

GSU7729   **missing closing** *char*

Explanation:
You specified a **{, =, \', `, ", (, ((,** or **[** and did not provide the corresponding closing character.

User response:
Provide the missing closing character.

GSU7730   *string*: **no expansion of unset variables**

Explanation:
You attempt to expand an unset variable when **set –o nounset** was on.

User response:
Use **set +o nounset** to turn **nounset** off and retry the expansion.

GSU7731   *string*: **cannot assign part of the undefined variable** *name* **in the construct ${***name=word*} **or as part of the empty or undefined variable** *name* **in the construct $ {***name:=word*}. *name* **when using these two constructs.**

GSU7732   *string*: **parameter null or not set**

Explanation:

You specified a **${***name***?}** construct where *name* was not set and no message followed the **?**.

User response:
Set *name* and specify a message after the **?**.

GSU7733   **no command substitution permitted**

Explanation:
The **wordexp** function was called by the shell to expand a string that contained command substitution, such as **$(***cmd***)** or f(CW'*cmd*'.

User response:
Check the program that called **wordexp** and ensure that the string to be expanded does not contain any command substitutions.

GSU7734   **substitute: internal error**

GSU7735   **Null signal**

GSU7736   **Hangup**

GSU7738   **Quit**

GSU7739   **Illegal instruction**

Explanation:
The shell received an illegal instruction signal. This signal typically occurs when a process tries to execute something that is not a valid machine instruction recognized by the hardware.

User response:
Contact your system programmer.

GSU7740   **Trap**

GSU7741   **Abort**

GSU7742   **Emulator trap**

GSU7743   **Floating point exception**

GSU7744   **Killed**

GSU7745   **Bus error**

GSU7746   **Segmentation violation**

GSU7747   **Bad system call**

GSU7748   **Pipe broken**

GSU7749   **Alarm clock**

GSU7750   **Terminated**

GSU7751   **User-defined signal 1**

GSU7752   **User-defined signal 2**

GSU7753   **Death of child**

GSU7754   **Power failure**

GSU7755   **Continued**

GSU7756   **Stopped (user)**

GSU7757	**Stopped (tty input)**
GSU7758	**Stopped (tty output)**
GSU7759	**Window size change**
GSU7760	**Urgent I/O**
GSU7761	**Pollable event**
GSU7762	**Virtual timer expiry**
GSU7766	**unbalanced [ ]**

Explanation:
You specified a **[** without providing the matching **]**.

User response:
Provide the missing **]**.

GSU7767	**missing closing)**

Explanation:
You specified a **(** as part of the test expression but did not provide the closing **)**.

User response:
Provide the missing **)**.

GSU7768	**Not a 'Shell Regular Built-in Utility'**

Explanation:
You tried to execute a command that is not a regular built-in utility.

User response:
Specify only shell regular built-in utilities.

GSU7769	**Not enough memory**

Explanation:
There were not enough free system resources to perform the requested operation.

User response:
Free up more resources.

GSU7770	**Too many arguments**

Explanation:
The system limit for the size of an argument list was exceeded.

User response:
Specify fewer arguments.

GSU7771	**execl failed**

Explanation:
The shell could not be executed to run a built-in shell utility.

User response:
Have your system programmer ensure that the shell is both accessible and executable.

GSU7773	**Usage: sleep [#h#m]#[s]**

Explanation:

The **sleep** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7774	**Usage: sort [-cmu] [-o outfile] [-y[kmem]] [-zmaxrec] [-dfiMnrb] [-tx] [-k keydef] [file...]**

Explanation:
The **sort** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7778	**Missing -o file**

Explanation:
You specified the **–o** option without providing a file name as an argument.

User response:
Provide the missing file name.

GSU7780	**-position "*string*" must follow +position**

Explanation:
You specified a **–***endpos* option either before or without a **+***startpos* option.

User response:
Reverse the order of the **–** and **+** options on the command line, or provide the missing **+***startpos* option.

GSU7813	**error setting termios attributes**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7814	***string*: unknown mode *command:***

Explanation:
You specified an unknown **stty** operand.

User response:
Check the description of **stty**, "stty — Set or display terminal options" on page 308, for a list of valid **stty** operands.

GSU7815	***string*: missing number after *command:***

Explanation:
You did not specify a numeric value as the argument of *operand*.

User response:
Supply an appropriate number as the argument of *operand*.

GSU7818	*string*: internal error 1

GSU7823	**reading termios attributes**

Explanation:
A call to **tcgetattr()** failed to return the necessary information.

User response:
Contact your system programmer.

GSU7824	*string*: **missing character after** *command:*

Explanation:
You did not specify a control character as the argument of *operand*.

User response:
Supply an appropriate argument for *operand*.

GSU7825	*string*: **badly formed** *command:* **character** *operand*

Explanation:
You specified *char* as the argument to *operand* but *char* is not a valid control character.

User response:
Specify a valid control character.

GSU7826	*string*: **missing speed after** *command:*

Explanation:
You did not specify a baud rate with the **ispeed** or **ospeed** operand.

User response:
Supply a valid baud rate.

GSU7827	*string*: **argument** *command:* **not valid after** *arg*

Explanation:
You specified an invalid baud rate, *arg*, after the **ispeed** or **ospeed** operand.

User response:
Contact your system programmer or check your reference manuals for a list of baud rates supported by your machine.

GSU7835	**not enough memory for buffering**

Explanation:
A system error indicating the cause is displayed with this message.

GSU7836	**re-opening file descriptor** *fd*

Explanation:
A system error indicating the cause is displayed with this message.

GSU7853	**creating file** *filename*

Explanation:

A system error indicating the cause is displayed with this message.

GSU7854	**opening file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU7859	**Bad date conversion**

Explanation:
You specified an invalid date string on the command line.

User response:
Specify a valid date string.

GSU7860	**Only one -r (-f) or -t flag allowed**

Explanation:
You specified a **–f**, **–r**, or **–t** option on the same command line with one or more other **–f**, **–r**, or **–t** options.

User response:
Remove the excess options, leaving only one **–f**, **–r**, or **–t** option.

GSU7861	**stat: age file** *filename* **inaccessible**

Explanation:
You specified a *filename* that either does not exist, or one for which you do not have appropriate permissions.

User response:
Check that *filename* exists, was named properly, and that you have appropriate permissions.

GSU7862	**Missing argument to option** *option*

Explanation:
You specified *option* without providing the required argument.

User response:
Provide the missing argument.

GSU7865	**Usage: tr [-cs] string1 string2 tr -s [-c] string1 tr -d [-c] string1 tr -ds [-c] string1 string2**

Explanation:
The **tr** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7871	**not a tty**

Explanation:
The standard input was not associated with a terminal.

User response:
Check if **tty** was meant to be used on a redirected file.

GSU7872   **Usage: tty [-s]**

Explanation:
The **tty** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7873   **Usage: uname [-snrvma]**

Explanation:
The **uname** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7876   **Missing number of fields to skip**

Explanation:
You specified the **–f** option without providing the number of fields to skip as its argument.

User response:
Provide the missing number of fields.

GSU7877   **Missing character skip count**

Explanation:
You specified the **–s** option without providing the number of characters to skip as its argument.

User response:
Provide the missing number of characters.

GSU7878   **skip not a number in** *string*

Explanation:
You specified an argument to the **+**, **–**, **–f**, or **–s** option that was not a valid number.

User response:
Specify a valid number.

GSU7879   **Usage: uniq [-udc] [-f #fields] [-s #characters] [input [output]] uniq [-udc] [-#fields] [+#characters] [input [output]]**

Explanation:
The **uniq** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU7945   **Line too long**

Explanation:

You tried to add text to a line that would cause its length to exceed the maximum indicated by the configuration variable **LINE_MAX**.

User response:
Make shorter lines.

GSU8028   *string***: more than** *command:* **bytes of arguments**

Explanation:
The resulting command line that was constructed was larger than the maximum allowed.

User response:
Specify arguments and options that produce a shorter command line.

GSU8029   *string***: more than** *command:* **arguments**

Explanation:
You specified more than 1023 arguments on the command line.

User response:
Specify less than 1024 arguments.

GSU8030   **size for** *option* **must be** *num*

Explanation:
You specified a numeric argument for *–option* that was greater than or equal to *num*.

User response:
Specify a numeric argument that is less than *num*.

GSU8031   *string***: -l number too large**

Explanation:
You specified the **–l** option with a numeric argument that was greater than 1023.

User response:
Specify an argument to the **–l** option that is less than 1024.

GSU8032   *string***: -n number too large**

Explanation:
You specified the **–n** option with a numeric argument that was greater than 1023.

User response:
Specify an argument to the **–n** option that is less than 1024.

GSU8049   **gspace: should be** *num1* **is** *num2*

GSU8053   **accept: j =** *num1* **!= nrule =** *num2*

GSU8054   **Code started at line** *num* **never ends**

Explanation:
Your grammar contained a **yacc** action that was not terminated with a **}**.

User response:
Provide the missing **}**.

GSU8055   **union declaration started at line *num* never ends**

Explanation:
Your grammar contained a **union {** declaration that lacked an ending **}**.

User response:
Provide the missing **}**.

GSU8058   **Variables aren't allowed here**

Explanation:
Your grammar attempted to set precedence/ association of a variable (nonterminal).

User response:
You can only set precedence or association of tokens.

GSU8059   **Sorry, value *num* is reserved for EOF/error**

Explanation:
Your grammar attempted to use a **yacc** internal token number.

User response:
Use a different token number.

GSU8060   **Warning: *name* redefined**

GSU8061   **Start symbol must be a variable**

Explanation:
Your grammar used a token as a start symbol.

User response:
You must use a variable (nonterminal) as a start symbol.

GSU8062   **Warning: start symbol redefined; was *value***

GSU8063   **%.hexnum prefix is already set to *prefix***

Explanation:
Your grammar used more than one *hexnum* prefix setting, or attempted to combine *hexnum* prefix with the **–p** *prefix*.

User response:
Remove extra *hexnum* prefix, or avoid using both the *hexnum* prefix directive and the **–p** option.

GSU8068   **ITEM TOO BIG!**

Explanation:
yacc was unable to create a human-readable state list due to a lack of available system resources. This only occurs when **yacc** reports on conflicts in the grammar.

User response:
Fix conflicts in the **yacc** grammar.

GSU8069   **ITEM and lookahead TOO BIG**

Explanation:
yacc was unable to create a human-readable state list due to a lack of available system resources. This only occurs when **yacc** reports on conflicts in the grammar.

User response:
Fix conflicts in the **yacc** grammar.

GSU8070   **ispace: should be *num1* is *num2***

GSU8071   **Unknown reserved word: *word***

Explanation:
Your grammar contained a **%** keyword that **yacc** did not recognize, most likely due to a misspelling in *word*.

User response:
Correct the spelling of *word*.

GSU8072   **Comment started at line *num* never ends**

GSU8073   **End of file in character constant**

Explanation:
Your grammar contained a character constant that was missing the closing quote.

User response:
Provide the missing quote.

GSU8074   **Empty character string**

Explanation:
Your grammar contained a quoted character string with no characters.

User response:
Make sure that all quoted strings contain characters.

GSU8075   **Sorry, value 0 is reserved for EOF**

GSU8076   **Mangled character constant**

Explanation:
Your grammar contained an illegal character constant.

User response:
Check and correct grammar.

GSU8103   **Out of memory at *num* bytes**

Explanation:
yacc has run out of system resources for this input grammar.

User response:
Simplify your grammar, or free more system resources.

GSU8105   **pspace: should be *num1* is *num2***

GSU8131   **mangle bad action**

GSU8143   **topt: state *num1* len *num2***

GSU8144   **Bad goto mark in temp file**

GSU8145   **Undefined nonterminal *'name'***

Explanation:
A grammar rule referenced a rule or token which is not defined.

User response:
Add the appropriate grammar rule or token.

GSU8155	Missing keydefinition after -k

Explanation:
You specified the **–k** option without providing a key definition as an argument.

User response:
Provide the missing key definition.

GSU8157	Usage: tail [-f] [-clnbkm [+-]number] [+-number[clbkm]] [file]

Explanation:
The **tail** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU8181	String started at line *num* never ends

Explanation:
Your grammar contained a string in a **yacc** action that was not terminated.

User response:
Make sure the string is terminated.

GSU8183	Code segment started at line *num* never ends

Explanation:
Your grammar contained a code segment that lacked an ending **%}**.

User response:
Provide the missing **%}**.

GSU8189	Warning: precedence of *name* redefined

GSU8211	Null to Expand

GSU8212	Expand recursing on Expand_buf

GSU8213	Invalid hop count: *num*

Explanation:
You specified the **–h** option, but the argument that you provided with it is not a valid number.

User response:
Provide a valid number as the argument to the **–h** option

GSU8214	Badly specified macro

Explanation:
The syntax of the macro is incorrect.

User response:
Use the correct syntax to specify the macro.

GSU8215	Usage: *mailx* [-defHiNn] [-u user] [filename] *mailx:* [-dFinU] [-h number] [-r address] [-s subject] user ...

Explanation:
The **mailx** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU8216	Characters after substitute modifier ignored

GSU8219	tsmail: writing mailbox *mailbox*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8223	Expansion too long

Explanation:
After expansion, the macro is too long for the supplied buffer.

User response:
Modify the macro so that it expands to less than **STRING_SIZE** (8192) bytes.

GSU8226	*string*: Error code *command:*

Explanation:
The indicated command in the recipe line returned with a nonzero return code.

User response:
Make treats this as an error unless the **.IGNORE** attribute has been used, or if the recipe line was preceded by a **-** character. If a nonzero return code is acceptable, modify the recipe line in the makefile so that the return code from this command line is ignored.

GSU8228	Freeing NIL pointer

GSU8229	Incomplete rule recipe group detected

Explanation:
You specified a group recipe but omitted the closing **]**.

User response:
Add the closing square bracket.

GSU8230	Cannot mix single and group recipe lines

Explanation:
You tried to mix recipe lines with group recipes for the same rule.

User response:
Either make the entire recipe a group, or remove the group.

GSU8231   **Found unmatched ']'**

Explanation:
You specified a **]** in your makefile for a group recipe without providing the matching **[**.

User response:
Provide the missing **[**.

GSU8232   **Expecting macro or rule defn, found neither**

Explanation:
Make expected this line in the makefile to contain a macro or rule definition, but it didn't. This probably indicates a syntax error in the makefile, or a comment which is missing the # symbol.

User response:
Correct this line in the makefile so it follows Make syntax rules. If the line is a comment, ensure that it starts with the # symbol.

GSU8233   **Illegal parser state** *num*

GSU8236   **Only a single % allowed in a target pattern**

Explanation:
A metarule target contained more than one '%'. It may only contain one.

User response:
Remove the additional percent signs.

GSU8241   **Unable to determine current directory**

Explanation:
make was unable to find out what its current directory was.

User response:
Verify that you have all necessary permissions to determine your current directory.

GSU8242   **Operator after special target treated as ':'**

Explanation:
You specified a modifier, such as !, with a rule defining a special target. **make** ignores any such modifiers.

User response:
Remove the extraneous modifier.

GSU8244   **Multiple targets are not allowed in % rules**

Explanation:
You specified a metarule with more than one target. A metarule can have only one target specified.

User response:
See the section in *z/VM: OpenExtensions Advanced Application Programming Tools* on inference rules and correct the makefile.

GSU8245   **Special target must appear alone**

Explanation:
You specified a special target which cannot appear with any other target in a rule. For example, a rule with **.ERROR** as a special target cannot mention any other target.

User response:
Correct the line.

GSU8247   **Syntax error in % rule, missing % target**

Explanation:
You specified your meta-rule incorrectly. The target must contain a %.

User response:
Correct the syntax of the rule.

GSU8249   **Missing targets or attributes in rule**

Explanation:
When reading input, **make** encountered a rule that had no targets or attributes specified.

User response:
Correct the syntax of your makefile.

GSU8253   **No .INCLUDE file(s) specified**

Explanation:
You specified a **.INCLUDE** special target without providing the names of the files to be included.

User response:
Refer to the description of the **.INCLUDE** target in *z/VM: OpenExtensions Advanced Application Programming Tools* and add the missing file names.

GSU8257   **Attributes possibly ignored**

Explanation:
A special target may inherit attributes, but only certain attributes take effect on specific special targets.

User response:
Refer to the description of **make**, "make — Maintain program-generated and interdependent files" on page 198, for more information about which attributes may be applied to which special targets.

GSU8260   **Nonglobal attributes ignored**

Explanation:

You specified attributes that are nonglobal. **make** will ignore them.

User response:
Remove the attributes.

GSU8263 Invalid library format

Explanation:
make attempted to access a library that was not in the correct format.

User response:
Verify that your library is correct and rebuild it if necessary.

GSU8267 Too many arguments -- limit *num*

Explanation:
Too many arguments were produced when **make** tried to execute a line in a recipe.

User response:
Simplify the recipe line.

GSU8279 Unknown or missing operator in symbolic audit mode *operator*

Explanation:
There is a missing or invalid operator in the specified symbolic **–audit** or **–aaudit**.

User response:
Refer to the description of the **find** command, "find — Find a file meeting specified criteria" on page 131 for the correct values and reenter the command.

GSU8280 Octal audit mode may contain only digits [0-7] in *option*

Explanation:
When you specify attributes in octal audit mode, the possible values are expressed by some combination of the digits 0 through 7 (for example, 777). You specified a number outside that range or you have specified characters along with or instead of digits.

User response:
Check the description of the **find** command, "find — Find a file meeting specified criteria" on page 131, for the correct values and reenter the command.

GSU8281 getgroupsbyname failed

Explanation:
This message indicates a system error.

User response:
Record any other messages and return codes that appear with this one and consult your system programmer or follow local procedures for reporting a problem to IBM.

GSU8282 Invalid printer format: *forms*

Explanation:

You specified too many arguments for **–d** (*dest*) on the **lp** command. "Destination_name", "class" and "forms" are the only permissible arguments on **–d**. They must be specified in that order.

User response:
Reissue the command with valid arguments on **–d**.

GSU8283 Invalid class: *class*

Explanation:
You specified the *class* operand of the **lp** command incorrectly. *class* cannot be longer than one character. Valid values are A-Z and 0-9, but your installation may not have all valid values defined.

User response:
Reissue the command with an appropriate value for *class*.

GSU8284 Unable to access printer.

Explanation:
The z/VM system did not recognize one or more of the operands you specified on the **lp** command.

User response:
Check what you specified for "destination_name", "class", and "forms". You may need assistance from your local help desk or a system programmer.

GSU8285 Unable to open printer ddn (*ddn*)

Explanation:
The system could not OPEN the SYSOUT data set.

User response:
This message probably indicates a system error. Consult your system programmer or follow local procedures for reporting a problem to IBM.

GSU8288 logger: wto failed, rc=*rc*

Explanation:
The **logger** command could not write your message to the operator console.

User response:
Record the return code and any associated messages that appear with this one and consult your system programmer.

GSU8565 tsmail: temporary file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8566 tsmail: writing temporary file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8697 Usage: tsmail user ...

Explanation:
The **tsmail** command entered was not syntactically correct.

User response:
Contact your sage displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU8703   No 'makefile' present

Explanation:
make was unable to find **Makefile** or **makefile**, and did not have any default rules.

User response:
Create the missing makefile, or add default rules to **startup.mk**.

GSU8704   Missing .END for .IF

Explanation:
You specified a **.IF** statement without the corresponding **.END** statement.

User response:
Provide the missing **.END** statement, or remove the extra **.IF** statement.

GSU8705   No target

Explanation:
make had a makefile to process, but did not find a rule defining a target to be made.

User response:
Add a target rule to your makefile, or specify a target on the command line.

GSU8707   Openfile: bad name

Explanation:
make attempted to open a file with an invalid or NULL name.

User response:
Edit the makefile and correct the file name.

GSU8714   No more memory

Explanation:
make was unable to allocate storage space.

User response:
Free up some resources and try again.

GSU8723   Unmatched "quote

Explanation:
You specified an opening " on a line that did not contain a closing ".

User response:
Correct the line.

GSU8724   .ELSE without .IF

Explanation:

You specified a **.ELSE** statement without a corresponding **.IF** statement.

User response:
Provide the corresponding **.IF** and **.END** statements (if necessary), or remove the **.ELSE** statement.

GSU8725   Unmatched .END

Explanation:
You specified a **.END** statement without the corresponding **.IF** statement.

User response:
Provide the missing **.IF** statement, or remove the extra **.ELSE** statement.

GSU8726   No macro name

Explanation:
A macro assignment **=** appears without a macro name.

User response:
Correct the line.

GSU8728   Write error on temp file

Explanation:
An error occurred while trying to write on a diversion or group recipe temporary file.

User response:
Ensure that there is space on the file system containing the temporary file.

GSU8730   <+ diversion unterminated

Explanation:
You specified a **<+** to begin a diversion, but did not specify the corresponding **+>** to end it.

User response:
Provide the closing **+>**.

GSU8731   Directory stack empty in pop

GSU8732   <+ missing before +>

Explanation:
You specified a **+>** to end a diversion before specifying the corresponding **<+** to begin it.

User response:
Ensure that corresponding **<+** and **+>** symbols appear in the correct order.

GSU8734   cannot access file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8735   Too many mail folders specified on command line.

Explanation:
The **–f** flag was specified, and more than one mail folder was named on the command line.

User response:
List only one file name on the command line.

GSU8736	tsmail: invalid user *user*

Explanation:
The name *user*, which was specified as a recipient of the message, is not a valid user on the system.

User response:
Check the spelling of the recipient's name, and try to send your message again.

GSU8737	tsmail: cannot lock file *filename*

Explanation:
The mailbox *filename* could not be locked, so the message could not be delivered.

User response:
Wait a little while and try to send the message again.

GSU8738	tsmail: re-opening temporary file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8739	tsmail: chowning mailbox *mailbox*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8772	Internal Error:

Explanation:
An internal error occurred.

User response:
Contact your system programmer. Follow local procedures for reporting a problem to IBM.

GSU8773	Warning: missing ; at end of rule
GSU8774	% prec needs a token; *string* isn't
GSU8775	Mangled $n construction
GSU8776	Warning: $n value *number* too big
GSU8778	Newline in string started at line *number*
GSU8779	Warning: redeclaration of type of *name*
GSU8780	Default action does not apply to null rules
GSU8781	Default action causes type clash
GSU8782	Need a type for $$
GSU8783	"*string*" is not a token
GSU8784	Mangled %type construction
GSU8785	grammar file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8786	header file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8787	listing file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8788	parser file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8789	file I/O error

Explanation:
A system error indicating the cause is displayed with this message.

GSU8813	Warning: newline in character constant
GSU8819	unlink temp file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8820	write error on temporary file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8821	file *filename* is binary

Explanation:
You specified the binary file *filename* as a **diff** input file. **diff** only works on text files.

User response:
Only specify text files as **diff** input files.

GSU8822	file *filename* line too long: limit *LINE_MAX*

Explanation:
The input line is too long.

User response:
Try again with a shorter input line.

GSU8824	yacc bug:

Explanation:
An internal error occurred.

User response:

Contact your system programmer. Follow local procedures for reporting a problem to IBM.

GSU8958　　　　read error on *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8977　　　　*string*: source *name1* and target *name2* are identical

Explanation:
You specified source and target files that are actually the same file (for example, because of links).

User response:
No further action is required.

GSU8978　　　　*string*: target directory *command:* on different file system than source *pathname*

GSU8979　　　　target *filename* must exist

Explanation:
The destination directory must exist for this utility to work.

User response:
Check the command line arguments. You may need to create the target directory.

GSU8980　　　　cannot create parent directory for target *filename*

Explanation:
An error occurred while trying to create the parent directory of the specified target file.

User response:
Make sure you have permissions to create the directory.

GSU8981　　　　Error copying file *file1* to *file1*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8982　　　　*string*: internal error: unknown return code from m_cp: *command:*

Explanation:
An internal error occurred.

User response:
Contact your system manager.

GSU8983　　　　Cannot reset times on file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8984　　　　Cannot reset permissions on file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU8985　　　　Cannot reset uid or gid on file *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU9007　　　　Field delimiter specified by -d must be one character

Explanation:
You specified a field delimiter (as an argument to the **–d** option) that was more than one character long.

User response:
Specify a single character field delimiter.

GSU9008　　　　file "[standard input]"

Explanation:
A system error indicating the cause is displayed with this message.

GSU9010　　　　date: bad format or date output longer than *number* bytes

Explanation:
The format string supplied to **date** is invalid, or the output is longer than the size of the date buffer.

User response:
Confirm that the date format string on the command line is valid.

GSU9011　　　　*number* truncated records

GSU9012　　　　*number* truncated record

GSU9013　　　　*number* read errors

GSU9014　　　　*number* read error

GSU9015　　　　*number* write errors

GSU9016　　　　*number* write error

GSU9086　　　　no space for line table

Explanation:
There were not enough free system resources to allocate initial resources for **ed**.

User response:
Free up more system resources and restart program.

GSU9087　　　　Input line too long

Explanation:
You entered an ed command which was too long.

User response:
Simplify the command and try again.

GSU9088　　　　no memory for pages

Explanation:

There were not enough free system resources to allocate initial resources for **ed**.

User response:
Free up more system resources and restart program.

GSU9090   **no memory for line number tables**

Explanation:
There were not enough free system resources to allocate initial resources for **ed**.

User response:
Free up more system resources and restart program.

GSU9091   **Result of substitution would produce a line too long**

Explanation:
You specified a replacement string in a substitution command that would produce a line that is too long for **ed** to handle.

User response:
Specify a shorter replacement string or split the original line into shorter lines before performing the substitution.

GSU9092   **Result line of join too long**

Explanation:
You attempt to use the **j** command to join a range lines into one line; however, the resulting line would be too long for **ed** to handle.

User response:
Specify a smaller range of lines to be joined.

GSU9093   **no space for expression or string**

Explanation:
There were not enough free system resources for **expr** to allocate for a string or expression.

User response:
Simplify the expression.

GSU9094   **find: must specify a command after -exec/-ok**

Explanation:
You specified either the **–exec** or the **–ok** primary without specifying a command to be performed.

User response:
Provide the missing command.

GSU9096   **Usage: head [-c|l|n|b|k|m number] [file ...] head [-number] [file ...]**

Explanation:
The **head** command entered was not syntactically correct.

User response:

The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9098   *string*: *command:* **illegal character sequence(s) for codeset in input file**

GSU9164   **Internal error: 10 too small in Get_token()**

Explanation:
An internal error occurred.

User response:
Contact your system programmer. Follow local procedures for reporting a problem to IBM.

GSU9165   **.IF .ELSE ... .END nesting too deep**

Explanation:
The nesting of **.IF .ELSE** ... **.END** structures is too deep.

User response:
Modify your makefile so that these structures are not nested as deep.

GSU9169   **Internal, buildList buffer too small**

GSU9170   **<+ diversion cannot be nested**

Explanation:
You tried to put one **<+** diversion inside another **<+** diversion. **make** does not permit this.

User response:
Remove the nested **<+** diversion.

GSU9174   **!reading file**

Explanation:
A system error indicating the cause is displayed with this message.

GSU9176   **Detected circular dependency using** *target*

Explanation:
After expansion, a target depends upon itself. Make does not permit this.

User response:
Modify the makefile to eliminate the circular dependency.

GSU9179   **seek past EOF on input**

Explanation:
The seek offset specified on the command line was greater than the size of the input file.

User response:
Check the offset and try again.

GSU9181   *string*: **component too long.**

Explanation:

One of the components of the path name provided is longer than is allowed by the filesystem (or by POSIX, if **–p** was specified).

User response:
Try to shorten the component or components of the path name.

___

GSU9182   *string*: **pathname too long.**

Explanation:
The length of the path name provided is longer than that allowed by the filesystem (or by POSIX, if **–p** was specified).

User response:
Try to shorten some of the components of the path name, in order to reduce the overall length of the path name.

___

GSU9183   *string*: **Not searchable.**

Explanation:
You specified a path name *pathname* that was not searchable.

User response:
Specify a different path name.

___

GSU9184   *string*: **requested format differs from the existing archive format**

Explanation:
You used the **–a** option with **–x** *format*, where the archive already existed with a different format.

User response:
Do not specify the format when appending to an existing archive, or specify the correct format.

___

GSU9185   **Symbolic link name too long: Not extracted**

Explanation:
Couldn't allocate enough memory to hold the symbolic link's name.

User response:
Archive contains symbolic name which is too large; no action possible.

___

GSU9186   **Missing format specification**

Explanation:
You did not specify a format specification on the command line.

User response:
Provide the missing format specification.

___

GSU9187   **Usage: printf format [argument ...]**

Explanation:
The **printf** command entered was not syntactically correct.

User response:

The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

___

GSU9188   **unused argument at** *arg*

Explanation:
You specified a format string without any conversion specifications.

User response:
Add at least one conversion specification to your format string.

___

GSU9190   **not a valid real argument** *string*

Explanation:
You specified a format specification that was expecting a real (that is,floating-point) number, but you provided the argument *string* which was not a valid real number.

User response:
Provide a valid real number in place of *string*.

___

GSU9191   **cannot allocate buffer for pathname**

Explanation:
A system error indicating the cause is displayed with this message.

___

GSU9192   **cannot determine working directory**

Explanation:
A system error indicating the cause is displayed with this message.

___

GSU9195   **cannot unlink entry** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

___

GSU9196   **cannot remove directory** *pathname*

Explanation:
A system error indicating the cause is displayed with this message.

___

GSU9199   **input line too long**

Explanation:
A line in the input file was longer than 10240 bytes.

User response:
Make sure that the input file is a text file.

___

GSU9200   **reading from file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

___

GSU9201   **input file** *filename* **is binary**

Explanation:

You specified the binary file *filename* as a **sed** input file. **sed** only works on text files.

User response:
Only specify text files as **sed** input files.

GSU9202	Unexpected return value from m_fgets(): *value*

GSU9203	unmatched {} commands

Explanation:
There is a '{' command in your script which does not have a corresponding '}' to terminate it.

User response:
Make sure that there are as many '}' as there are '{'.

GSU9204	pattern space overflow during G command

Explanation:
The content of the hold buffer was too long to be appended to the pattern buffer.

User response:
Place a smaller amount of text in the hold buffer.

GSU9205	hold space overflow during H command

Explanation:
You tried to "hold" more data than would fit in the **sed** hold buffer.

User response:
Reorganize your script to require less data in the hold buffer.

GSU9206	Can't chain *cmd* command

Explanation:
A system error indicating the cause is displayed with this message.

GSU9207	improper word after <<

GSU9208	!cannot redirect (dup2)

Explanation:
A system error indicating the cause is displayed with this message.

GSU9209	!cannot execute

Explanation:
A system error indicating the cause is displayed with this message.

GSU9210	Traced functions not effective unless -o korn is set

Explanation:
"typeset -ft function" (turn on tracing for the named function) was specified, but KornShell mode wasn't enabled.

User response:

Don't specify "typeset –ft", or enable KornShell mode with "set –o korn".

GSU9211	set [+-abCefhikKmnptuvx] [+-o option] [-s] [+-A name] [arg ...]

GSU9212	Undefined functions not implemented

Explanation:
typeset –fu specifies attributes for a function that will be defined later. This is currently not implemented.

User response:
Specify the function's attributes when defining it, instead of using **typeset –fu**.

GSU9221	!get limit failed

Explanation:
A system error indicating the cause is displayed with this message.

GSU9222	!set limit failed

Explanation:
A system error indicating the cause is displayed with this message.

GSU9224	bad number *num*

Explanation:
Invalid string given where a number was expected. (**MAILCHECK**, **TMOUT**, **OPTIND**, **HISTSIZE**, and **COLUMNS** environment variables, or array subscript.)

User response:
Specify a decimal number (containing only the digits 0 through 9) to the appropriate environment variables or subscripts.

GSU9225	no memory: *system_error*

Explanation:
A system error indicating the cause is displayed with this message.

GSU9228	undefined variable

GSU9229	Reset tty pgrp from *curr_pgrp* back to our pgrp *old_pgrp*

GSU9230	Internal error: j_close: no processes

Explanation:
An internal error occurred.

User response:
Contact your system programmer. Follow local procedures for reporting a problem to IBM.

GSU9244	Usage: sh *string* [-*abCefhikKLmnprtuvx*] [-o option] file [arg ...] sh [--*abCefhikKLmnprtuvx*] [-o option] [-s [arg ...] sh [--

abCefhikKLmnprtuvx] [-o option] -c command [name [arg ...]]

Explanation:
The **sh** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9253	Usage: wc [-lw] [-c|-m] [file ...]

Explanation:
The **wc** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9256	Symbol on left side of rule must be a variable; *name* isn't

GSU9257	Mangled $<...> construction

GSU9258	It's too late to start using types

GSU9259	Need an explicit type for $n when n <= 0

Explanation:
In a grammar with a **union** declaration, an action references a Yacc symbol value **$n**, where *n* <= 0, but no type is specified for **$n**.

User response:
Add an explicit type, of the form **$**type**n**.

GSU9261	Need a type for *name*

Explanation:
In a grammar with a **union** declaration, an action is referencing a Yacc symbol value that does not have a type associated with it.

User response:
Use **%type** *type* rule to assign a type to a rule, or **%token** *type tokenname* to assign a type to a token. Alternatively, you can use explicit types within the action, in the form **$**\fItype**n**.

GSU9268	Too many makefiles specified.

Explanation:
You specified too many files using the **–f** option.

User response:
Combine one or more files into a single file.

GSU9270	Too many open files. Max nesting level is *num*

Explanation:
You have exceeded the maximum limit of **.INCLUDES**.

User response:
Check to see if you have recursively included a **make** file, or simplify your makefile.

GSU9271	Could not create string *string1*

Explanation:
A system error indicating the cause is displayed with this message.

GSU9275	Usage: awk [-f scriptfile] [-Fc] [-v var=val] [script] [var=val ...] [file ...]

Explanation:
The **awk** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9276	can't pass scalar to *var*

Explanation:
You tried to pass a scalar value to a function expecting an array parameter.

User response:
Correct your program.

GSU9277	can't pass array to *var*

Explanation:
You tried to pass an array into a function expecting a scalar parameter.

User response:
Correct your program.

GSU9278	built-in *var* can't be used as a parameter or auto variable

Explanation:
You tried to use the name of a built-in function or variable as a parameter or local variable in a function.

User response:
Correct your program.

GSU9279	*string*() is not a function

Explanation:
You tried to use *name* as a function when it was not defined as such.

User response:
Correct your program, or make sure that the spelling of *name* is the same as was used when defining the function.

GSU9280	'*string*' can only have values from *num1* through *num2*

Explanation:

You tried to assign a value to a built-in variable that is outside the permitted range.

User response:
Check the description for the **bc** command, "bc — Use the arbitrary-precision arithmetic calculation language" on page 29, and correct your program to use a value that is within the acceptable range for that variable.

GSU9281   **while executing function *funcname***

Explanation:
An error occurred while executing the named function.

User response:
Determined by remainder of message.

GSU9283   **internal error: Converting wide character back to MB**

Explanation:
An internal error occurred.

User response:
Contact your system manager.

GSU9285   **Number *string* not in range *num1 num2***

Explanation:
An invalid user ID was specified.

User response:
Ensure that the command line arguments are correct.

GSU9286   **Usage: comm [-123] file1 file2**

Explanation:
The **comm** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9299   **Usage: date [-cu] [+format] date [-cut] <date_time>**

Explanation:
The **date** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9310   **Badly formed line/byte count *num***

Explanation:
You gave an invalid number for the **–n** option.

User response:
Correct the command line.

GSU9311   **Missing number after *option* option**

Explanation:
You specified the *–option* option without providing a number as its argument.

User response:
Provide the missing number.

GSU9312   **byte count not in range *num1***

Explanation:
You gave an invalid byte count.

User response:
On the command line, correct the byte count to a number that can be expressed by the machine architecture.

GSU9356   ***string*: Internal error: nextrecord: Unexpected status return from m_fgetws: *command:***

Explanation:
An internal error occurred.

User response:
Contact your system manager.

GSU9357   **Writing to standard output**

Explanation:
A system error indicating the cause is displayed with this message.

GSU9358   **Writing unpaired records**

Explanation:
A system error indicating the cause is displayed with this message.

GSU9363   **unknown**

GSU9364   **Bad date conversion: *string***

Explanation:
The string passed to **m_readdate** was not in a format that the function recognized.

User response:
Check the format of the date and try again.

GSU9367   **Usage: logger [-t tag] [-siITu] [-p priority] [-d destination] [-f filename]/ logger [-t tag] [-siITu] [-p priority] [-d destination] log-message**

Explanation:
The **logger** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9368   ***string*: Only printable characters are permitted in log messages.**

Explanation:
You specified a nonprintable character in a log message.

User response:
Replace the nonprintable character with one or more printable characters.

GSU9374   *string*: **Failed to strip file** *command:*

Explanation:
An error occurred while trying to strip an executable file.

User response:
No action possible.

GSU9376   **Macro** *macroname* **cannot be redefined**

GSU9377   **WARNING: Macro** *macroname* **redefined after use**

GSU9378   **Special target** *target* **cannot be a prerequisite**

Explanation:
You tried to use a special target as a prerequisite.

User response:
Edit the makefile, and remove the special target from the prerequisite list.

GSU9379   **Option -c failed to change directory to** *pathname* *system_error*

Explanation:
A system error indicating the cause is displayed with this message.

GSU9380   *string*: **Unknown option** *command:*

Explanation:
You specified an option that is not valid for this command.

User response:
Check the description in this book for the command you were using to find the valid list of options for that command.

GSU9381   *string*: **Option** *command:* **argument missing**

Explanation:
You specified the *–option* option without providing its required argument.

User response:
Provide the missing argument.

GSU9383   **Configuration file** *filename* **not found**

Explanation:

Could not open the **MAKESTARTUP** configuration file.

User response:
The **MAKESTARTUP** file may be either misnamed or missing. Ensure that the **MAKESTARTUP** file exists, and that it is accessible. This may require setting the *[MACRO MAKESTARTUP] macro or **MAKESTARTUP** environment variable.

GSU9384   **Unable to return to directory** *pathname*

Explanation:
Make could not set the directory back to the original directory. The original directory may have been deleted, renamed, or had its permissions changed since Make was started.

User response:
Ensure that the directory exists and has the correct permissions. Attempt the make operation again.

GSU9385   **!file** *filename*

Explanation:
A system error indicating the cause is displayed with this message.

GSU9387   **Unable to change directory to** *pathname*

Explanation:
Make could not set the directory back to the specified directory. The specified directory may have been deleted, renamed, or had its permissions changed since Make was started.

User response:
Ensure that the directory exists and has the correct permissions. Attempt the make operation again.

GSU9388   **Left temp file** *filename*

GSU9392   **file is binary**

Explanation:
A file that was supposed to contain rules contained binary data.

User response:
Ensure that the correct file name is specified and that the contents of the specified file are correct.

GSU9393   **line too long: limit** *num*

Explanation:
The makefile contains a line that exceeds the **LINE_MAX** limit.

User response:
Shorten the line. You can use the continuation character (backslash) to spread long rules over several lines in the makefile.

GSU9394   **!error reading file**

Explanation:

A system error indicating the cause is displayed with this message.

GSU9401	Inference rules result in circular dependency for *target*

Explanation:
The inference rules result in a target that depends upon itself.

User response:
Verify that the recipe lines are correct. Ensure that the meta rules or suffix rules are correctly specified. In some cases, you may need to use an explicit rule to override the action of the inference rules.

GSU9407	Macro name *macroname* truncated to *shortname*

GSU9412	[*string*] not remade because of errors for *target* if *time1* > *time2*

GSU9415	Don't know how to make *target*

Explanation:
Make does not know how to make the given target.

User response:
Ensure that the target is defined in the makefile. The target may be declared directly by an explicit rule or indirectly by an inference rule.

GSU9419	Mismatched braces in token *token*

Explanation:
The number of open braces (**{**) does not match the number of close braces (**}**) in this token.

User response:
Edit the token so that each open brace has a matching close brace.

GSU9420	Detected circular macro *macroname*

Explanation:
The macro refers to itself, either directly or through a chain of several macros. Macros cannot refer to themselves.

User response:
Modify the macro so that it does not refer to itself after expansion.

GSU9421	Unterminated pattern string: *string*

Explanation:
The pattern string specified in the pattern substitution is not terminated with a separator character (normally **/**).

User response:
Modify the pattern substitution so the separator characters are correctly placed. The first character after the **:s** is used as the separation character.

GSU9422	Unterminated replacement string: *string*

Explanation:
The substitution string specified in the pattern substitution is not terminated with a separator character (normally **/**).

User response:
Modify the pattern substitution so the separator characters are correctly placed. The first character after the **:s** is used as the separator character.

GSU9423	Modifier *modifier* can't be used in a modifier list - ignored

GSU9424	Argument string to *modifier* must be quoted with "

Explanation:
The argument to this macro expansion must be quoted using double quotation marks.

User response:
Modify the macro expansion so that the argument is quoted using double quotation marks.

GSU9425	Unterminated argument string *string* for modifier *modifier*

Explanation:
The argument to this macro expansion must be quoted using double quotation marks. The terminating double quotation mark character was missing.

User response:
Modify the macro expansion so the argument is properly quoted in double quotation marks.

GSU9426	Illegal modifier *modifier* in macro *macroname*

Explanation:
You specified a character that is not a legal macro modifier.

User response:
Modify the macro expansion expression to use only legal modifiers.

GSU9427	Name too long *pathname*

Explanation:
You specified a path name that exceeds the maximum length allowed for directory names.

User response:
Use a shorter path name for this directory. Move the files higher up in the directory tree so the path name does note exceed the maximum path length.

GSU9428	Ambiguity in *target_list* targets *target* chose *string*

GSU9429	meta-rule too long: *rule*

Explanation:
The meta-rule contained its maximum number of characters before Make reached the end of the meta-rule.

User response:
Shorten the meta-rule so it fits in **DONE_STATE** characters.

GSU9430	Internal, bad current dfa state *num* in *node_name*

Explanation:
An internal error occurred.

User response:
Contact your system administrator.

GSU9431	Only one *.SETDIR* attribute allowed in rule line

Explanation:
You have a rule with more than one **.SETDIR** attribute.

User response:
If you want Make to search for a file in a number of different directories, use the **.SOURCE** special target.

GSU9432	Duplicate entry *target* in target list

GSU9433	Duplicate entry *prereq* in prerequisite list

GSU9434	Multiply defined recipe for target *target*

Explanation:
You specified more than one recipe for *target* in different rules, and the rules use the **:** operator.

User response:
Either use the **:** operator to handle independent recipes, or correct your makefile.

GSU9435	Empty recipe for special target *target*

Explanation:
The special target specified requires that a recipe also be specified for it.

User response:
Refer to the documentation for the target and add an appropriate recipe.

GSU9436	*string* ignored on suffix rule *.SETDIR*

Explanation:
The attribute is ignored, so it cannot be applied to this suffix rule.

User response:
Remove the attribute from the suffix rule.

GSU9437	Imported macro *macroname* not found in environment

Explanation:
make attempted to import a macro that was not present in the program environment.

User response:
Define the appropriate environment variable, remove the import rule, or add the **.IGNORE** attribute to the import rule.

GSU9439	Include file *filename* not found

Explanation:
make couldn't find the file *filename*.

User response:
Check that the file exists, was named properly and that you have the appropriate permissions. Also check the prerequisites of the **.INCLUDIRS** target to make sure that it specifies the correct path.

GSU9440	*string* ignored on special target *.SETDIR*

GSU9441	Target *target* cannot mix ':' and '::' rules

Explanation:
You defined a rule for *target* using the **::** operator, and then followed this with another rule for target using the **::** operator.

User response:
Either modify the second rule to use **::** or remove it.

GSU9442	WARNING: duplicate meta-rule *rule1* : *rule2* new rule replaces old

GSU9443	Multiple *.SETDIR* for *target* ignored

GSU9444	Cannot find member defining *archive*

GSU9450	Warning -- empty .SUFFIXES target - suffix rules ignored.

GSU9452	(*string*): Can't extract library member timestamp; using EPOCH

GSU9453	*string*(*string*): Can't touch library member

GSU9454	*string* macro not defined

Explanation:
You tried to execute a recipe that required the shell and either the *[MACRO GROUPSHELL] macro or the **SHELL** macro was not defined.

User response:
Make sure that the macro is defined properly in your makefile or **startup.mk** file.

GSU9455	Could not export *env_string*

GSU9456	Cannot open *pathname*

Explanation:

make was unable to open a temporary file for a diversion or group recipe. You may not be able to write to your **TMPDIR** directory.

User response:
Make sure that the **TMPDIR** environment variable is set up properly, that you have the appropriate permissions in that directory and that there is space on the file system.

GSU9458   **Usage: mkfifo [-m mode] [-p] file ...**

Explanation:
The **mkfifo** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9464   **allocating buffer for backslash interpretation**

Explanation:
A system error indicating the cause is displayed with this message.

GSU9465   **internal error: Converting "%%b" format argument from wide to M B**

Explanation:
A system error indicating the cause is displayed with this message.

GSU9466   **internal error: unexpected return value from bs()**

Explanation:
A system error indicating the cause is displayed with this message.

GSU9467   **argument** *arg*

Explanation:
A system error indicating the cause is displayed with this message.

GSU9501   **parsing format string**

Explanation:
A system error indicating the cause is displayed with this message.

GSU9502   **Usage: rmdir [-p] directory ...**

Explanation:
The **rmdir** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9503   **cannot open file** *filename* **in** *cmd* **command**

Explanation:
The file named in the *cmd* command could not be opened, either because the maximum number of files was already open, or because you were not permitted to write to *file*.

User response:
Either simplify your script, so that it requires fewer open files, or check to ensure that you do have permission to write to the file.

GSU9505   **Warning: unknown process** *process_id* **terminated**

GSU9508   **Usage: tee [-ia] [file] ...**

Explanation:
The **tee** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9510   **Usage: touch [-amc] [-r|-f file] [-t time] [date_time] file ...**

Explanation:
The **touch** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9528   **file** *filename* **is binary**

Explanation:
A system error indicating the cause is displayed with this message.

GSU9529   **file** *filename* **line too long: limit** *LINE_MAX*

Explanation:
A system error indicating the cause is displayed with this message.

GSU9530   **Usage: xargs [-l#][-L #] [-irepl][-I repl] [-n#] [-tpx] [-s#] [-eeof][-E eof] [cmd [args ...]]**

Explanation:
The **xargs** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9532   **can't open parser resource file** *filename*

GSU9533   **disk error: cannot write temp file**

Explanation:
A system error indicating the cause is displayed with this message.

GSU9535   **Usage: yacc [-dhlmqstv] [-D hdr] [-o out] [-p pfx] [-P proto] [-L[C|P]] [-S states] [-V *stats*] [-b fpfx] gram.y**

Explanation:
The **yacc** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9536   **no input file**

Explanation:
No input file name was specified on the **yacc** command.

GSU9538   **yacc -- parser generator language Usage: yacc [-dhlmqstvw] [-D hdr] [-o out][-p pfx] [-P proto] [-L[C|P]] [-S states] [-V stats] [-b fpfx] [-W rcfile] gram.y -d generate header file 'ytab.h' -h print this message and stop -l disable generation of #line directives in output parser -m show memory usage and timing statistics for each pass -q quiet mode--no warnings -s write state information to states.out -t define YYDEBUG symbol for debugging code and tables -v generate verbose statistics file 'y.out' -w create a Windows® compatible resource file for tables -D hdr generate header file into 'hdr' -o out put parser into 'out' rather than 'ytab.c' -p pfx all variables will use prefix 'pfx' rather than 'yy' and 'YY' -P proto parser prototype is in 'proto' rather than '/etc/yyparse.c' -LP generate Turbo Pascal output into 'ytab.pas', and header into 'ytab.hp'. -LC generate C++ output into 'ytab.cpp', and header into 'ytab.hpp' -S states write state information into file 'states' rather than states.out -V stats generate verbose statistics like -v into file 'stats' -b fpfx specify output files to begin with 'fpfx' rather than 'y' -W rcfile specify name of Windows compatible resource file**

Explanation:
Specifying -h attribute on the **yacc** command displays this help message.

GSU9564   **Insufficient disk space on device or Bad temporary file (read)**

Explanation:
Yacc encountered a problem while reading a temporary file.

User response:
Ensure that the disk is not full or defective.

GSU9566   **expanded length of string too long; limit *char_set_size***

Explanation:
You specified a string that expanded to a length greater than the number of characters in the character set. Since a given character may appear only once in the first string, you specified a character more than once in that string.

User response:
Remove any repeated characters in the first string.

GSU9567   **Starting endpoint *hexnum1* does not precede the second endpoint *hexnum2***

Explanation:
The starting point of a range of characters is after the end point you have indicated.

User response:
Reverse the start and end points of the range.

GSU9569   **Invalid character class *class***

Explanation:
You specified a character class *class* that is not defined in the locale indicated by **LC_CTYPE**.

User response:
Specify a character class that is defined in the locale indicated by **LC_CTYPE**.

GSU9570   **Collation string is not supported in [=equiv=].**

Explanation:
You specified a string for *equiv* in a **[=*equiv*=]** expression that contained more that one character. **tr** accepts only a single character for the equivalence class.

User response:
Specify a one-character equivalence class.

GSU9571   **syntax error in [x*n] expression expression.**

User response:
Provide the missing **]**.

GSU9572 **[x*0] construct may only occur once**

Explanation:
You tried to fill the string using the **[x*0]** construct more than once.

User response:
Remove the second fill request.

GSU9581 **!memory allocation failure**

Explanation:
A system error indicating the cause is displayed with this message.

GSU9583 **reading windowsize attributes**

Explanation:
A system error indicating the cause is displayed with this message.

GSU9585 **Only classes [:upper:] and [:lower:] are valid as a translate result, and then only if the corresponding character class is specified**

Explanation:
You specified a *class* in a **[:*class***:] construct in *string2* that was not **lower** or **upper**, or you specified **[:lower:]** or **[:upper:]** in *string2* without specifying the other one at the corresponding spot in *string1*.

User response:
Specify *string2* in a form that gives an equivalent result without using the **[:class:]** construct, or specify **[:upper:]** or **[:lower:]** (as appropriate) at the correct point in *string1*.

GSU9586 **input file *filename***

Explanation:
A system error indicating the cause is displayed with this message.

GSU9587 ***string*: input line to long in *command:***

Explanation:
A line in the input file *filename* was longer than **LINE_MAX** bytes.

User response:
Use **cmp** to compare non-text files.

GSU9588 ***string*: input file *command:* is a binary file**

Explanation:
You specified *filename* as the input file; however, *filename* is a binary file. **uniq** only works with text files.

User response:
Specify a text file as the input file.

GSU9591 **string file *filename***

GSU9592 **regular expression error in *regular_expression_error string***

GSU9593 **dd: only one of conv=ucase and conv=lcase may be specified**

GSU9594 **dd: only one of conv=block and conv=unblock may be specified**

GSU9595 **dd: only one character set translation option may be specified**

GSU9596 **non-numeric argument *string***

GSU9597 **Options were given that can only be used when sending mail.**

GSU9598 **History command number *commandnumber* is not valid**

GSU9599 **too few or too many args**

GSU9600 **Usage: cmp [-blsx] file1 file2 [seek1 [seek2]]**

Explanation:
The **cmp** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9607 **cut: list [*string*]: numbers must be nonzero**

GSU9608 **Usage: df [-kPt] [file_system ...] [file ...]**

Explanation:
The **df** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9610 **cannot specify -m with either -e or -f**

GSU9611 **Usage: diff [-befhHimnNrsw] [-c[n]] [-C n] [-Dname] file1 file2**

Explanation:
The **diff** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9612	Usage: bdiff [-befimnNrsw] [-c n]] [-C n] [-Dname] file1 file2 n

Explanation:
The **bdiff** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9613	Usage: diffh [-befimnNrsw] [-c[n]] [-C n] [-Dname] file1 file2

Explanation:
The **diffh** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9614	Usage: dircmp [-ds] dir1 dir2

Explanation:
The **dircmp** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9618	Internal error: wcspagein
GSU9619	Internal error: wcspageout
GSU9620	!File read error
GSU9621	*number* line(s) too long -- truncated
GSU9625	*string*: Invalid variable name
GSU9626	Usage: getconf system_var getconf path_var pathname getconf -a

Explanation:
The **getconf** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9627	*stringstring*: input lines truncated - result questionable
GSU9628	*string*: input lines truncated - result questionable
GSU9629	out of space for pattern
GSU9631	file *file* line *lineno error*

GSU9632	line *lineno error*
GSU9633	Usage: egrep *command:* [-bI] [-e pattern] [-f patternfile] [pattern] [file ...]

Explanation:
The **egrep** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9634	Usage: iconv -f from-codeset -t to-codeset [-sc] [file ...]

Explanation:
The **iconv** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9635	iconv -l [-v]
GSU9636	System does not support querying the set of character sets
GSU9637	Warning: multibyte locale not supported
GSU9638	Number *numstring* not in range *num1*
GSU9639	Number *numstring* not in range *num1*
GSU9640	Badly formed number in *numstring*
GSU9641	backtrack stack overflow: expression generates too many alternatives
GSU9643	Usage: locale [-a|-m]

Explanation:
The **locale** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9644	locale [-ck] name ...
GSU9645	No archive operation specified
GSU9646	More than one archive operation specified
GSU9647	Missing position name
GSU9648	Missing archive name

GSU9649	**Missing member name**
GSU9651	**Filename** *filename* **too long**
GSU9652	**!creating extracted file** *filename*
GSU9653	**Usage: ar -d[Ilv] [-F format] archive member …**

Explanation:
The **ar** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9654	**ar -m[abiIlv] [-F format] [posname] archive file …**
GSU9655	**ar [-pt[Isv]] [-F format] archive file …**
GSU9656	**ar -q[.clv] [-F format] archive file …**
GSU9657	**ar [-ru[abciIluv]] [-F format] [posname] archive file …**
GSU9658	**ar -x[CIsTv] [-F format] archive file …**
GSU9659	*string* **not found**
GSU9660	**Unknown command ″ Type** *command* **for help.**
GSU9661	**pipe buffer**
GSU9662	**Command** *command* **is illegal in command file.**
GSU9663	**Command** *command* **is illegal in input mode.**
GSU9664	**Unknown command** *command*
GSU9665	**Unknown colon modifier** *colon_modifier*
GSU9669	**Unrecognized scrolling command** *command*
GSU9670	**Usage: make [-eEinpqrstuvVx] [-k|-S] [-c dir] [-D macro=value] [-f file] [macro=value …] [target …]**

Explanation:
The **cmp** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9673	*string*: **Nonportable character** *pathname char* **found.**
GSU9674	*string*: **Nonportable byte** *pathname* **found.**
GSU9679	**Existing file** *filename* **exists; it will not be overwritten**
GSU9681	*number* **illegal character sequence(s) for codeset extracting file** *num*
GSU9682	**Cannot append to compressed archive**
GSU9683	**Warning: -o** *keyword* **Unknown keyword value**
GSU9685	**invalid option on substitution** *letter*
GSU9686	**Usage: tar -[crtux][lpmovwzU[0-7 lmh]] [[-b blocks] [-f tarfile] [-V volpattern] [file [-C pathname] …]**

Explanation:
The **tar** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9687	**Usage: pr [+#] [-#	-c #	-m] [-adFfprtW] [-ec#] [-h header] [-ic#] [-l #] [-nc#] [-H header-format] [-o #] [-sc] [-w #] [file …]**

Explanation:
The **pr** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9688	**Options -c and -m are mutually exclusive**
GSU9692	**Usage: ps [-Aacdefjl] [-G idlist] [-g grouplist] [-n name] [-o format] … [-p proclist] [-s idlist] [-t termlist] [-u|-U uidlist]**

Explanation:
The **ar** command entered was not syntactically correct.

User response:
The usage message displays the correct syntax for this command. Reenter the command with the correct syntax.

GSU9693	**Address not in range** *numstring*

GSU9694	Improper sleep interval specification *seconds*
GSU9695	"*string*" exceeds range *seconds* seconds
GSU9696	!write error on file "*string*"
GSU9697	Badly formed sort key position *pos*
GSU9698	Must specify number in *option*
GSU9699	Mutually exclusive options: -o and -c
GSU9700	too many key field positions specified
GSU9701	key value in *key* out of bounds
GSU9702	invalid key specification *key*
GSU9703	Option *option* unknown in field *field*
GSU9704	file *file* no newline at end of file
GSU9705	file *filename* line too long: limit *maxrec* -- truncated
GSU9706	!read error on file *file*
GSU9707	file *file* line *line* non-unique key in *record*
GSU9708	file *file* line *line* not ordered properly *record*
GSU9710	!temporary file error *filename*
GSU9711	*string* must be in range from 0 to 127
GSU9712	Upper/lower case conversion must be specified in the same relative positions
GSU9713	Upper/lower case conversion mutually exclusive with complement (-c) option
GSU9891	Warning: unknown type: *type*
GSU9892	Warning: unknown type: *type*
GSU9893	Type is *type1* should be *type2*
GSU9894	Unknown type: *type*
GSU9895	Warning: encountered literal character token(s) with value greater than 255
GSU9922	internal execution tree error at *string*
GSU9923	unbalanced *char*
GSU9924	Unknown option *option*
GSU9925	invalid character *char* (hex *hexnum*

GSU9926	error reading file
GSU9927	error splitting record: *regular_express_error*
GSU9928	invalid wide character %x
GSU9929	scalar *name* cannot be used as array
GSU9930	array *name* cannot be used as a scalar
GSU9931	variable *name* cannot be used as a function
GSU9932	syntax error *regular_express_error* in *line*
GSU9933	cannot assign to function *funcname*
GSU9934	function *funcname* nesting level > *number*
GSU9935	function "*string*" nesting level too deep
GSU9936	wrong number of arguments to function *funcname*
GSU9937	*string* function requires an array
GSU9938	*string*: *asort* is not a function
GSU9939	function *funcname* redefined
GSU9950	input lines truncated - result questionable

Explanation:
The file being grep'ed is a binary file. grep assumes that the file it is searching has records terminated by new line characters. When it finds no new line characters then it assumes that the input line has been truncated, and quits.

User response:
Do not use grep to search binary files

GSU9952	!history file $HISTFILE *pathname*

Explanation
The system could not open the history file specified for one of the following reasons;

- One or more of the directories in the path do not exist.
- You are not authorized to write to the file.
- You are not authorized to write to one or more directories in the path.

User response:
Ensure that the path and file name to be stored in HISTFILE are correct. Ensure that you have the

appropriate permissions to access the file and each
directory in the path.

Appendix E. Common Error Messages When Using BFS Files

If you enter a command that interacts with a Byte File System (BFS) or Network File System (NFS), you might receive error messages such as those listed below. For a detailed description of a message and the suggested action to resolve the error, see *z/VM: CMS and REXX/VM Messages and Codes*.

Table 15. Common Error Messages while using BFS Files

Number	Text	Return Code
DMS029E	Invalid parameter *pathname*	24
DMS062E	Invalid character in pathname *pathname*	20
DMS099E	This is not allowed in the CMS/DOS environment	40
DMS109E	Virtual storage capacity exceeded	104
DMS132E	File too large: *pathname*	104
DMS250E	I/O error	100
DMS389E	Invalid character: X'00'	24
DMS389E	Invalid operand: *operand*	24
DMS389E	Invalid positive integer: *number*	24
DMS512E	This is not allowed in CMS subset mode	40
DMS1016E	{PC-NFS	MOUNT [DUMP	EXPORT]} program is not available at foreign host	99
DMS1017E	Too many levels of remote file systems	88
DMS1018E	Your username and password could not be authenticated. The PC-NFS program returned an error	76
DMS1019E	Network File System name is not allowed	28
DMS1020E	Foreign host cannot be reached. The request returned *rc* indicating the network is down	55
DMS1020E	Foreign host cannot be reached. The request returned *rc* indicating the network is unreachable	55
DMS1020E	Foreign host cannot be reached. The request returned *rc* indicating the connection was terminated	55
DMS1020E	Foreign host cannot be reached. The request returned *rc* indicating the connection was reset	55
DMS1020E	Foreign host cannot be reached. The request returned *rc* indicating the connection timed out	55
DMS1020E	Foreign host cannot be reached. The request returned *rc* indicating the connection was refused	55
DMS1020E	Foreign host cannot be reached. The request returned *rc* indicating the host is unreachable	55

Table 15. Common Error Messages while using BFS Files (continued)

Number	Text	Return Code
DMS1020E	Foreign host cannot be reached. The request returned *rc* indicating the host is down	55
DMS1021E	Foreign host responded that {the file handle is stale|the cookie is bad}	28, 55
DMS1022E	Not enough buffer space is available	104
DMS1023E	An error was returned on a call to identify the host on which the program is running	55
DMS1026E	The operation is not supported for an object in an NFS-mounted file system	28
DMS1073E	No sockets are available for the request	55
DMS1131E	Directory already exists: *pathname*	28
DMS1137E	Object is locked; deadlock detected	70
DMS1139E	You are not permitted to issue this command	76
DMS1147E	Storage management error trying to free storage	104
DMS1151E	File pool is unavailable	99
DMS1153E	File pool is unavailable or unknown	99
DMS1153E	File space is unavailable or unknown	99
DMS1155E	CSL routine *cslname* {is not loaded|has been dropped}	40
DMS1162E	Directory is not empty: *pathname*	40
DMS1174E	The MAXCONN limit has been reached. You have tried to establish more APPC/VM connections than is allowed for your user ID. There are no inactive communication paths available for reuse for the current request.	55
DMS1174E	Your attempt exceeds the number of APPC/VM connections allowed for file pool	55
DMS1176E	Virtual storage capacity exceeded for file pool	99
DMS1184E	A directory is not found, or your are not permitted to use a directory in *pathname*	28
DMS1184E	File not found or you are not authorized for it	28
DMS1259E	File pool has run out of physical space in the storage group	99
DMS1311E	{Object|File} already exists: *pathname*	28
DMS2105E	Permission is denied	28
DMS2106E	No space is available in the file system	40
DMS2107E	Object is temporarily unavailable: *pathname*	70
DMS2108E	Object is busy: *pathname*	70
DMS2108E	Operation is interrupted on *pathname*	70
DMS2109E	Object is a directory: *pathname*	40
DMS2110E	Object is not a directory: *pathname*	40

Table 15. Common Error Messages while using BFS Files (continued)

Number	Text	Return Code
DMS2110E	A node in path name is not a directory for *pathname*	40
DMS2111E	OPENVM limit exceeded	88
DMS2112E	Path name or a component of path name is too long	40
DMS2112E	Contents of the external link must be between 1 and 1023 characters	40
DMS2113E	Object does not exist: *pathname*	28
DMS2113E	File system is not mounted or not available	28
DMS2114E	The file system is read only	36
DMS2115E	Objects are on different file systems	88
DMS2116E	A loop is encountered in symbolic links	40
DMS2117E	Object is not {a BFS regular file | a regular file | a symbolic link or external link | in the proper format to be an executable file | a BFS file space} [:*pathname*]	28
DMS2119E	Path name is not fully qualified: *pathname*	28
DMS2120E	Unable to resolve current working directory for path name *pathname*	28
DMS2121E	Operation may not be performed on {the file system root|. or ..}	88
DMS2122E	Invalid symbolic link {content|length} *pathname*	40
DMS2123E	File system is already mounted	40
DMS2124E	Path name is part of the new name for *pathname*	40
DMS2125E	Path name ends with a slash: *pathname*	40
DMS2126E	You may not link to a directory	88
DMS2132E	Error obtaining UID or GID. User not authorized	104
DMS2132E	Error obtaining UID or GID. User not found	104
DMS2132E	Error obtaining UID or GID. Database not available	104
DMS2132E	Error obtaining UID or GID. Command not allowed in CMS/DOS environment, in CMS subset mode, or on this level of CP	104
DMS2132E	Error obtaining UID or GID. Group not found	104
DMS2132E	Error obtaining UID or GID. User or group not found	104
DMS2134E	Return code *retcode* and reason code *reascode* [X'*hexreascode*'] given on call to routine *rtnname* [for pathname *pathname*.]	104
DMS2140R	Enter operands: (enter a null line to indicate that you are finished)	0
DMS2140E	No operands were entered for the OPENVM command	24
DMS2141E	Missing quote or invalid quote specification	24
DMS2142E	There are no characters in a quoted string or an extraneous quoted string was specified	24
DMS2143E	There is no external link data specified	24
DMS2153E	File is migrated and DFSMS/VM is not available	51

Table 15. Common Error Messages while using BFS Files (continued)

Number	Text	Return Code
DMS2154E	File is migrated and implicit RECALL is set to OFF	50
DMS2510E	Requested function is not supported for specified file object	99
DMS3085E	You do not have permission to mount this directory or the remote NFS server requires the use of low port numbers	99
DMS3995E	You are not authorized to mount in read/write mode	76

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information may contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Programming Interface Information

This publication documents information NOT intended to be used as Programming Interfaces of z/VM.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., in the United States and/or other countries. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on IBM Copyright and trademark information (https://www.ibm.com/legal/copytrade).

Adobe is either a registered trademark or a trademark of Adobe Systems Incorporated in the United States, and/or other countries.

The registered trademark Linux is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Terms and Conditions for Product Documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal Use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see:

• The section entitled **IBM Websites** at IBM Privacy Statement (https://www.ibm.com/privacy)
• Cookies and Similar Technologies (https://www.ibm.com/privacy#Cookies_and_Similar_Technologies)

Acknowledgments

InterOpen/POSIX Shell and Utilities is a source code product providing POSIX.2 (Shell and Utilities) functions to the OpenExtensions services offered with VM. InterOpen/POSIX Shell and Utilities is developed and licensed by Mortice Kern Systems (MKS) Inc. of Waterloo, Ontario, Canada.

Information in this document has been adapted from the *InterOpen/POSIX Shell and Utilities User Manual*, supplied by Mortice Kern Systems (MKS) Inc. for use by licensees of their InterOpen/POSIX Shell and Utilities source code product.

Bibliography

This topic lists the publications in the z/VM library. For abstracts of the z/VM publications, see *z/VM: General Information*.

Where to Get z/VM Information

The current z/VM product documentation is available in IBM Documentation - z/VM (https://www.ibm.com/docs/en/zvm).

z/VM Base Library

Overview

- *z/VM: License Information*, GI13-4377
- *z/VM: General Information*, GC24-6286

Installation, Migration, and Service

- *z/VM: Installation Guide*, GC24-6292
- *z/VM: Migration Guide*, GC24-6294
- *z/VM: Service Guide*, GC24-6325
- *z/VM: VMSES/E Introduction and Reference*, GC24-6336

Planning and Administration

- *z/VM: CMS File Pool Planning, Administration, and Operation*, SC24-6261
- *z/VM: CMS Planning and Administration*, SC24-6264
- *z/VM: Connectivity*, SC24-6267
- *z/VM: CP Planning and Administration*, SC24-6271
- *z/VM: Getting Started with Linux on IBM Z*, SC24-6287
- *z/VM: Group Control System*, SC24-6289
- *z/VM: I/O Configuration*, SC24-6291
- *z/VM: Running Guest Operating Systems*, SC24-6321
- *z/VM: Saved Segments Planning and Administration*, SC24-6322
- *z/VM: Secure Configuration Guide*, SC24-6323

Customization and Tuning

- *z/VM: CP Exit Customization*, SC24-6269
- *z/VM: Performance*, SC24-6301

Operation and Use

- *z/VM: CMS Commands and Utilities Reference*, SC24-6260
- *z/VM: CMS Primer*, SC24-6265
- *z/VM: CMS User's Guide*, SC24-6266
- *z/VM: CP Commands and Utilities Reference*, SC24-6268

- *z/VM: System Operation*, SC24-6326
- *z/VM: Virtual Machine Operation*, SC24-6334
- *z/VM: XEDIT Commands and Macros Reference*, SC24-6337
- *z/VM: XEDIT User's Guide*, SC24-6338

Application Programming

- *z/VM: CMS Application Development Guide*, SC24-6256
- *z/VM: CMS Application Development Guide for Assembler*, SC24-6257
- *z/VM: CMS Application Multitasking*, SC24-6258
- *z/VM: CMS Callable Services Reference*, SC24-6259
- *z/VM: CMS Macros and Functions Reference*, SC24-6262
- *z/VM: CMS Pipelines User's Guide and Reference*, SC24-6252
- *z/VM: CP Programming Services*, SC24-6272
- *z/VM: CPI Communications User's Guide*, SC24-6273
- *z/VM: ESA/XC Principles of Operation*, SC24-6285
- *z/VM: Language Environment User's Guide*, SC24-6293
- *z/VM: OpenExtensions Advanced Application Programming Tools*, SC24-6295
- *z/VM: OpenExtensions Callable Services Reference*, SC24-6296
- *z/VM: OpenExtensions Commands Reference*, SC24-6297
- *z/VM: OpenExtensions POSIX Conformance Document*, GC24-6298
- *z/VM: OpenExtensions User's Guide*, SC24-6299
- *z/VM: Program Management Binder for CMS*, SC24-6304
- *z/VM: Reusable Server Kernel Programmer's Guide and Reference*, SC24-6313
- *z/VM: REXX/VM Reference*, SC24-6314
- *z/VM: REXX/VM User's Guide*, SC24-6315
- *z/VM: Systems Management Application Programming*, SC24-6327
- *z/VM: z/Architecture Extended Configuration (z/XC) Principles of Operation*, SC27-4940

Diagnosis

- *z/VM: CMS and REXX/VM Messages and Codes*, GC24-6255
- *z/VM: CP Messages and Codes*, GC24-6270
- *z/VM: Diagnosis Guide*, GC24-6280
- *z/VM: Dump Viewing Facility*, GC24-6284
- *z/VM: Other Components Messages and Codes*, GC24-6300
- *z/VM: VM Dump Tool*, GC24-6335

z/VM Facilities and Features

Data Facility Storage Management Subsystem for z/VM

- *z/VM: DFSMS/VM Customization*, SC24-6274
- *z/VM: DFSMS/VM Diagnosis Guide*, GC24-6275
- *z/VM: DFSMS/VM Messages and Codes*, GC24-6276
- *z/VM: DFSMS/VM Planning Guide*, SC24-6277

- *z/VM: DFSMS/VM Removable Media Services*, SC24-6278
- *z/VM: DFSMS/VM Storage Administration*, SC24-6279

Directory Maintenance Facility for z/VM

- *z/VM: Directory Maintenance Facility Commands Reference*, SC24-6281
- *z/VM: Directory Maintenance Facility Messages*, GC24-6282
- *z/VM: Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6283

Open Systems Adapter

- Open Systems Adapter/Support Facility on the Hardware Management Console (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/SC14-7580-02.pdf), SC14-7580
- Open Systems Adapter-Express ICC 3215 Support (https://www.ibm.com/docs/en/zos/2.3.0?topic=osa-icc-3215-support), SA23-2247
- Open Systems Adapter Integrated Console Controller User's Guide (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/SC27-9003-02.pdf), SC27-9003
- Open Systems Adapter-Express Customer's Guide and Reference (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/ioa2z1f0.pdf), SA22-7935

Performance Toolkit for z/VM

- *z/VM: Performance Toolkit Guide*, SC24-6302
- *z/VM: Performance Toolkit Reference*, SC24-6303

The following publications contain sections that provide information about z/VM Performance Data Pump, which is licensed with Performance Toolkit for z/VM.

- *z/VM: Performance*, SC24-6301. See z/VM Performance Data Pump.
- *z/VM: Other Components Messages and Codes*, GC24-6300. See Data Pump Messages.

RACF® Security Server for z/VM

- *z/VM: RACF Security Server Auditor's Guide*, SC24-6305
- *z/VM: RACF Security Server Command Language Reference*, SC24-6306
- *z/VM: RACF Security Server Diagnosis Guide*, GC24-6307
- *z/VM: RACF Security Server General User's Guide*, SC24-6308
- *z/VM: RACF Security Server Macros and Interfaces*, SC24-6309
- *z/VM: RACF Security Server Messages and Codes*, GC24-6310
- *z/VM: RACF Security Server Security Administrator's Guide*, SC24-6311
- *z/VM: RACF Security Server System Programmer's Guide*, SC24-6312
- *z/VM: Security Server RACROUTE Macro Reference*, SC24-6324

Remote Spooling Communications Subsystem Networking for z/VM

- *z/VM: RSCS Networking Diagnosis*, GC24-6316
- *z/VM: RSCS Networking Exit Customization*, SC24-6317
- *z/VM: RSCS Networking Messages and Codes*, GC24-6318
- *z/VM: RSCS Networking Operation and Use*, SC24-6319
- *z/VM: RSCS Networking Planning and Configuration*, SC24-6320

TCP/IP for z/VM

- *z/VM: TCP/IP Diagnosis Guide*, GC24-6328
- *z/VM: TCP/IP LDAP Administration Guide*, SC24-6329
- *z/VM: TCP/IP Messages and Codes*, GC24-6330
- *z/VM: TCP/IP Planning and Customization*, SC24-6331
- *z/VM: TCP/IP Programmer's Reference*, SC24-6332
- *z/VM: TCP/IP User's Guide*, SC24-6333

Prerequisite Products

Device Support Facilities

- Device Support Facilities (ICKDSF): User's Guide and Reference (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/ickug00_v2r5.pdf), GC35-0033

Related Products

XL C++ for z/VM

- *XL C/C++ for z/VM: Runtime Library Reference*, SC09-7624
- *XL C/C++ for z/VM: User's Guide*, SC09-7625

z/OS

IBM Documentation - z/OS (https://www.ibm.com/docs/en/zos)

Additional Publications

XL C/C++ for z/VM: Runtime Library Reference, SC09-7624
XL C/C++ for z/VM: User's Guide, SC09-7625

Index

Special Characters

copying *(continued)*
    file *(continued)*
        to target named by the last argument on command
        line 82
        with data conversion 95
    in/out file archive 85
    portable archive to and from tape 418
    sections of files 89
    standard input to each output file 321
counting bytes, characters, lines, and words 354
cpio archive 85
cpio command 85
creating
    archives 85
    BFS path name reference 383
    BFSpath name reference 385
    command aliases 6
    directories for each named directory argument 215
    directory (OPENVM) 376
    executable file 46
    external link 377
    FIFO special files 217
    libraries 210
    library archives 9
    link to files 165
    path definition for OS ddnames 421
    symbolic link 385
crt environment variable 193
current mail message 183
current working directory
    changing
        to new 449
        to previous working directory 56
    displaying
        information about files in 398
        path name of the 259
    setting to value of the HOME environment variable 56
cut command 89

D

dash in shell command syntax, explanation of 1
data
    manipulating 13
    reading 95
    removing from executable files 307
    writing 95
database, joining two 155
date command 91
date, displaying the 91
dd command 95
DEAD environment variable 193
dead letter 193
debug, OPENVM command error messages 387
defining BFS file creation mask 453
delaying program execution 300
deleting
    alias definitions 345
    arguments 264
    attributes of shell variables and functions 352
    directory entries or directories 264, 266
    information from executable files 307
    path definition for OS ddnames 422
    trailing part of file names 104

deleting *(continued)*
    values of shell variables and functions 352
description section, shell command descriptions 3
descriptor file, opening, closing, and copying 120
destination file, setting destination or modification time 82
detecting
    aliases 345
    end of child processes or jobs 353
diff command 99
directory
    changing modes, owners, or permissions 59, 64
    copying files 85
    creating for each named directory argument 215
    moving files to a different 222
    removing directories or directory entries 264, 266
    setting owners and groups 64
    substitution 283
dirname command 104
displaying
    checksum for each input file 66
    command aliases 6
    current operating systems, names of 346
    current working directory 432
    currently exported variables 122
    dates and times 91
    differences between two files 99
    elapsed time for a command 327
    environment variable names and values 273
    environments 117
    external link, BFS 435
    file creation mask values, BFS 438
    file names 339
    files 54
    first part of files 147
    information about BFS files 398
    information about locales 168
    last part of files 316
    lines common to two files 74
    names of current operating systems 346
    names of environment variables 273
    number of bytes in each input file 66
    path name of working directories 259
    process status 254
    processor time 256
    processors 327
    symbolic link, BFS 435
    system time accumulated by commands 329
    terminal names 339
    terminal options 308
    unprintable characters 54
    user ID (UID) of person who entered commands 172
    user time accumulated by the shell 329
    values of environment variables 273
    what is mounted in your BFS hierarchy 440
dot (.) command 106
dot (.) script, returning from 263
dot environment variable 193
dot path name component 370
dot-dot path name component 370
double-spacing output 245
dumping files to standard output or in octal 232
duplicating output stream 321
dynamic scoping 37

E

EBCDIC to ASCII conversion 95
echo command 107
ed command 109
editing
    ed command 109
    red command 109
    restricted 109
    subcommands, starting 268
EDITOR environment variable 194, 293
electronic mail, sending and receiving 180
ellipsis in shell command syntax, explanation of 2
end of file 316
ending
    jobs or processes 157
    shell 121
entering a CMS command from the shell 70
env command 117
ENV environment variable 293
environment variables
    _ 292
    ~ 293
    allnet 192
    ask 193
    askbcc 193
    askcc 193
    asksub 193
    autoprint 193
    bang 193
    CDPATH 293
    cmd 193
    COLUMNS 293
    crt 193
    DEAD 193
    displaying names and values of 273
    dot 193
    EDITOR 194, 293
    ENV 293
    escape 194
    FCEDIT 293
    flipr 194
    folder 194
    header 194
    HISTFILE 293
    HISTSIZE 293
    hold 194
    HOME 192, 293
    IFS 293
    ignoreeof 194
    indent 194
    indentprefix 194
    keep 194
    keepsave 194
    LANG 293
    LC_ALL 293
    LC_COLLATE 293
    LC_CTYPE 293
    LC_MESSAGES 293
    LINENO 293
    LINES 293
    LISTER 194
    LOGNAME 293
    MAIL 192, 294

environment variables *(continued)*
    MAILCHECK 294
    MAILDIR 192
    MAILPATH 294
    MAILRC 192, 194
    MAILSERV 194
    MBOX 195, 294
    metoo 195
    names, displaying 273
    OLDPWD 294
    onehop 195
    outfolder 195
    page 195
    PAGER 195
    PATH 294
    PID 294
    prompt 195
    PS1 294
    PS2 294
    PS3 294
    PS4 294
    PWD 294
    quiet 195
    RANDOM 294
    record 195
    Replyall 195
    save 195
    screen 195
    SECONDS 295
    sendmail 195
    sendwait 195
    SHELL 295
    showto 196
    sign 196
    Sign 196
    TMOUT 295
    toplines 196
    TZ 295
    values, displaying 273
    VISUAL 196
environment variables section, shell command descriptions 3
environment, displaying or setting for a process 117
equivalence class (regexp) 472
erasing a BFS object 391
escape character
    cent sign xiii
    using xiii
escape environment variable 194
escape sequences 282
escaping characters 282
eval command 119
evaluating
    arguments
        as an expression 123
        in the current shell environment 119
    arithmetic expressions 160
    shell expressions 73
examples section, shell command descriptions 3
exception condition, trapping 335
exec command 120
executable file
    displaying symbol table 227
executable file, creating 46

OPENVM RENAME command 444
OPENVM RUN command 446
OPENVM SET DIRECTORY command 449
OPENVM SET FORK command 452
OPENVM SET MASK command 453
OPENVM SHELL command 456
OPENVM SHOWMOUNT command 458
OPENVM UNMOUNT command 461
operating system, displaying name of the current 346
operators, shell 281
options
    explanation of shell command 1
    obtaining from a list of parameters 142
    order of shell command 1
options section, shell command descriptions 3
order
    of shell command options 1
    of shell items on command line 2
OS ddname path definitions
    creating 421
    deleting 422
    querying 423
outfolder environment variable 195
output file, copying standard input to each 321
output stream, cloning 321
overlaying commands 120

P

page environment variable 195
PAGER environment variable 195
paginated file, formatting and printing 245
parameter
    positional, in shell 283
    positional, setting and unsetting 273
    positional, shifting 298
    special, in shell 283
    substitution 283
parent environment, displaying 117
parsing utility options 142
pass small amounts of input to filter or file 107
paste command 236
path definitions for OS ddnames
    creating 421
    deleting 422
    querying 423
PATH environment variable 294
path name component, BFS 368
path name, BFS
    absolute 369
    checking for validity and portability 238
    component 368
    creating
        external link 377
        reference to a 383, 385
    description 368
    directory components of 104
    displaying 259
    fully qualified 368
    quotes, using 368
    reference to a 383, 385
    relative 369
    returning 28, 104
    syntax 368

path name, BFS *(continued)*
    understanding 368
    valid characters 370
path name, NFS
    description 374
    syntax 374
    understanding 374
path search 282
pathchk command 238
pattern
    buffer 268
    finding, using regular expressions 471
    rules for 280
    searching 144
pax command 239
permission-code mask 343
permissions
    access 225
    applied with umask 343
    archive file 419
    associated with objects in the BFS 399
    bits 405, 424
    changing 61, 424
    character definitions 405
    checking 411
    controlling 454
    default file 5
    denying 453
    determining 453
    displaying 404, 438
    execute 62, 424
    external link 379, 425
    file creation mask 438
    file, default 5
    file, description of 177
    granting 86
    group 424
    group file 405
    in effect 376
    incorrect 134
    link 383
    lists 132
    mask, file creation 453
    matching 132
    owner 62, 424
    owner file 405
    public 424
    public file 405
    querying 417, 438
    read 61, 424
    removing 425
    renaming 444
    replacing 425
    resetting 454
    restoring when extracting 319
    screening out 343
    search 62, 424
    set-GID-on-execution 417, 425
    set-UID-on-execution 417, 425
    setting 61, 62, 424
    symbolic link 385, 425
    temporary access 425
    temporary access to files 405
    turning off 429

shell *(continued)*
    variables, removing attributes of 352
SHELL environment variable 295
shift command 298
shift positional parameters 298
short circuit evaluation 15
showexp 458
showing
    current operating systems, names of 346
    current working directory 432
    currently exported variables 122
    differences between two files 99
    elapsed time for a command 327
    environment variable names and values 273
    environments 117
    external link, BFS 435
    file creation mask values, BFS 438
    first part of files 147
    information about locales 168
    last part of files 316
    lines common to two files 74
    names of current operating systems 346
    names of environment variables 273
    path name of working directories 259
    process status 254
    processors 327
    symbolic link, BFS 435
    system time accumulated by commands 329
    terminal names 339
    user time accumulated by the shell 329
    values of environment variables 273
    what is mounted in your BFS hierarchy 440
showto environment variable 196
SID (session ID), displaying 255
SIGHUP signal, ignoring the 230
sign environment variable 196
Sign environment variable 196
signal
    intercepting 335
    sending to processes 157
simple shell command, running a 76, 281
skip to the next iteration of a loop in a shell script 81
sleep command 300
socket file type 177
sort command 301
sort-merge utility 301
sorted files, locating 74
sorting keys 302
source file, managing 198
special built-in commands 291
special parameter 283
special target 207
specifying command lines for another command 120
split output stream 321
starting
    application to a CMS module in the record file system 446
    BFS application 446
    shell 277
    sort-merge utility 301
status, displaying 254
STDIN (standard input)
    closing 289
    copying data read from 333

STDIN (standard input) *(continued)*
    copying to each output file 321
    explanation of 3
    reading lexical syntax description from 162
    reading lines from 260
STDOUT (standard output)
    closing 289
    copying standard input to each 321
    displaying arguments to the 249
    dumping file to 232
    explanation of 3
    sending paginated files to 245
    writing arguments to 107, 251
    writing configuration values to 138
stop shell 121
string
    finding, in text files 471
    searching for 144
strip command 307
stty command 308
su command 314
subscript-in-array condition 15
subshell environment 291
substitute commands 287
substitute directories 283
superuser
    change owning GID or UID 416
    changing authorization to 314, 463
    command authority 314
    environment 314
    privileges 239, 314
    process authority 158
    session 314
    shell, executing 314
suspending program execution 300
swapping bytes 86
symbol table
    displaying the 227
symbolic link
    creating a 385
    displaying information about 435
SYMTAB symbol table 15
syntax
    explanation of shell command 1
    lexical, reading description of 162
    shell command 1
syntax diagrams, how to read xiv
system files 323

T

tail command 316
tape archive 318
tar archive files 318
tar command 318
target 199
tee command 321
template, shell command 357
temporary files 115
terminal name, displaying 339
terminal options, displaying and setting 308
terminal settings, changing 373
test command 323
test condition 323

IBM ®

Product Number: 5741-A09

Printed in USA