[an error occurred while processing this directive]

A UNIX Reference

UNIX Command Syntax

UNIX commands are usually given by typing the name of the command, followed by options (if needed), followed by command-line arguments (if needed), all separated by spaces. The command is executed when you hit the return key.

Options are usually one or two characters preceeded by a minus sign. Command-line arguments are often names of files or directories. For example, the following command prints the file named `lab1a.C` to the printer named `mwah177` (`-Pmwah177` option) in a 2-column rotated format (`-2r` option).

```
enscript -2r -Pmwah177 lab1a.C
```

UNIX Input and Output Redirection

Normally, when a UNIX command is executed its input comes from the keyboard and its output goes to the screen. Redirection allows a command to be executed with input coming from a file or output going to a file.

```
command < file1
```

executes *command* with input from *file1*,

```
command > file2
```

executes *command* with out to *file2*, and

```
command < file1 > file2
```

executes *command* with input from *file1* and output to *file2*.

UNIX Pipes

Piping is used to connect the output from one program to the input of another program. For example,

```
command1 | command2
```

executes both *command1* and *command2*, but *command2* will get its input from the output produced by *command1*, rather than reading input from the keyboard.

Environmental Variables

In UNIX, you can define environmental variables, which are named string values. The name, preceded by a dollar sign, can be used in UNIX commands. One use for environmental variables is defining shortcut names for files or directories with long pathnames.

The `printenv` command is used to display environment variable definitions. If it is given with no arguments then all environmental variables wil be displayed. You can also give the command with a single argument, which is the name of an environmental variable without a dollar sign. Then the definition of that variable is displayed.

The `setenv` command is used to define an environmental variable. The first argument names the variable. The second argument is the string value for the variable. It is a good practice to put the string value in quotes. This is essential if the string value contains any blanks.

For example, suppose you want to define a variable named `INTRO` to have the string value "/home/usra/COURSES/cs1623". To do this you give the command

```
setenv INTRO "/home/usra/COURSES/cs1623"
```

Then to check the variable give the command

```
printenv INTRO
```

This should display the string value without quotes.

If you want to read the file `/home/usra/COURSES/cs1623/UNIX` using vi, you would give the command

```
vi $INTRO/UNIX
```

The UNIX Command History Mechanism

Most UNIX command interpreters can record earlier commands and have a special syntax for repeating earlier commands. The following commands are understood by the UNIX C shell command interpreter:

```
!!
```

reexecute the previous command

```
!pattern
```

reexecute the most recent command that matches *pattern*.

For example, giving the command `!v` results in reexecuting the most recent command that begins with the letter v. For `vi` users this is a way of

resuming an edit session on a file after an interuption.

UNIX Directories

UNIX files are located in directories, which are special files that contain a table of their contents. A directory can contain other directories, resulting in a heirarchical directory structure, similar to a family tree. When a file or directory A is is contained in directory B then we say that B is the parent of A and that A is a child of B. A *path* is a sequence of directories or files in which each is the parent or child of the one that follows it. A *downward path* is a sequence of directories or files in which each is the parent of the one that follows it. If there is a downward path from A to B then we say that A is an *ancestor* of B and that B is a *descendant* of A. In a UNIX system there is a directory called the *root directory* that is an ancestor of every file or directory.

While you are logged on a UNIX system, there will always be one directory whose contents are readily accessible. This directory is called your *current working directory*. When you first log on to a UNIX system, your current working directory is a directory that is called your *home directory*. Each system user has their own home directory.

At first, you will probably keep all of your files in your home directory, which will not have any subdirectories. After you have created a lot of files you will probably want to use subdirectories to organize the files. When you get to this point, you should read about naming UNIX files.

UNIX has a command, mkdir, for creating new directories. If you need to change the structure of your directories, you can use the rmdir command to remove directories and the mv command to move or rename directories.

You can change your current working directory using the cd command. This gives you the capability of navigating through the UNIX file system. You can see what your current working directory is by using the pwd command and list the contents of the current working directory with the ls command. Finally, you can set access permissions on directories with the chmod command.

Naming UNIX Files and Directories

When you first start working with UNIX, file names are simple. You keep all of your files in your home directory and if you need to work with them you use the name that you specified when you created the file. After a while, you accumulate too many files to keep track of or to list using the `ls` command. Then you need to start using subdirectories to organize your

files.

In UNIX, the simple names that you used at first only apply to files that are in your current working directory. For other files, you need to specify the directory that contains the file in addition to its simple name. There are two ways of doing this: absolute path names, and relative path names.

Absolute Path names

An absolute path name always begins with a forward slash (`/`). It specifies a downward path from the root directory to a file or directory. The absolute pathname for a file or directory is constructed by listing, in order, the names of all of the directories in the path from the root directory to the named file or directory. The names are separated by forward slashes with no spaces.

For example, the root directory contains a subdirectory named `usr`, which contains a subdirectory named `bin`, which contains an executable file named `vi`. This is the `vi` editor program. Its absolute path name is `/usr/bin/vi`. The leading forward slash says to begin at the root directory (it is an absolute pathname). The `usr` says go into the `usr` subdirectory. The `bin` says go into the `bin` subdirectory. The `vi` says take the file named `vi`.

Relative Path names

A relative path name is constructed like an absolute path name in that it is formed with a sequence of directory or file names separated by forward slashes. However, a relative path name does not begin with a forward slash, and it describes a path starting from the current working directory, which is not included in the name. If your current working directory is `/user` then the relative path name of the `vi` program is `bin/vi`. The relative path name of a file in your current working directory is just the simple name of the file.

Relative path names do not have to use downward paths. They can use upward steps with the `..` filename abbreviation. For example, the directory `/usr` also contains a subdirectory named `etc`. If this is your current working directory then the relative pathname of the `vi` file is `../bin/vi`.

UNIX Filename Abbreviations

Here are some abbreviations that can be used to simplify long path names for files and directories:

- `.` - the current working directory
- `..` - the directory above the current working directory

~ - your home directory

* - matches any string of characters that does not contain a /.

For example, the command

```
cp /home/usra/COURSES/cs1623/lab3/* .
```

copies all files from the directory named `/home/usra/COURSES/cs1623/lab3` to your current working directory. When the `*` appears in a command-line argument, the UNIX command interpreter generates a list of file names with one name for each file whose name matches the argument. UNIX environmental variables can also be used for customized pathname abbreviations.

Navigating Through the UNIX File System

The `cd` command changes your current working directory. If no command-line argument is given, it makes your home directory the new working directory. Otherwise, there should be one command-line argument, which is the name of the the directory that becomes the new working directory.

Displaying the Current Working Directory

The `pwd` command displays the absolute pathname of the current working directory. It has no command-line arguments or options.

Listing Contents of Directories

The ls program lists files. Without command-line arguments, it lists the files and subdirectories of the current working directory. Arguments can be given to specify the name of files or directories to list. Some command options are:

-l - list with more complete file information, including access permissions and the latest modification time

-F - show subdirectories with a trailing slash (/) and executable programs with a trailing asterisk (*)

-a - show hidden files and directories. Normally, files whose name begins with a period are not shown by ls.

When the -l option is used, files are listed as shown below:

```
-rw-r--r--   1 gshute   users       13875 Mar 10 13:17 UNIX
|\-/\-/\-/   | \----/   \---/       \---/ \----------/ \--/
| |  |  |    |   |        |           |        |        |
t u  g  o    l  user    group        size  time stamp  name
```

The meanings of these fields are:

t - file type:
    - for ordinary files
    d for directories.
u, g, o - owner, group, and others permissions:
    r for read
    w for write
    x for execute
l - number of links to the file
user - user owner of the file
group - group owner of the file
size - size of the file in bytes (characters)
timestamp - last modification time of the file
name - simple name of the file

UNIX File and Directory Permissions

The `chmod` command is used to change access permissions on files and directories. The first argument specifies the desired permissions and the remaining arguments name the files and directories. The first argument can be formed as a sequence of characters containing either a plus sign for adding permission or a minus sign for removing permissions. Letters preceding the sign indicate who is affected by the change. Letters after the sign indicate what kind of permission is affected.

For the letters preceding the sign use one or more of the following:

- `u` - you
- `g` - your group
- `o` - others
- `a` - all (you, your group, and others)

You will rarely need to change permissions for yourself or your group.

For the letters after the plus or minus sign use one or more of the following:

- `r` - read permission
- `w` - write permission
- `x` - execute permission

Read permission is required to read a file or to list the contents of a directory. Write permission is required to modify a file or to add files to or remove files from a directory. Execute permission is required to execute a program file or to navigate into a directory (make it into your current working directory).

For example, if you want to deny all types of access to files named filea and fileb to others, you would give the following command:

```
chmod o-rwx filea fileb
```

You can check the permissions on files and directories using the ls command with the -l option.

Making New Directories

The `mkdir` command creates a new directory for each of its arguments. The arguments are the names of the new directories.

Moving and Renaming Files and Directories

The `mv` command moves or renames files or directories. It has two forms. To rename a file or directory use the following form:

```
mv oldName newName
```

where *oldName* is the name of an existing file or directory and *newName* is the new name for the file or directory.

To move files or directories to a new location use the following form:

```
mv fileOrDirectory ... directory
```

All arguments except the last name existing files or directories, which are are moved into the directory named by *directory*. *directory* must be the name of an existing directory.

Removing Files and Directories

The `rm` command removes files and the `rmdir` command removes directories. For either command you can specify any number of arguments, which are the names of files or directories that you want to remove. In order to remove a directory, you must first remove all of its files and subdirectories. On most UNIX systems, the `rm` command is set up to request verification prior to removing each file.

Browsing Files

UNIX has two commands specifically designed for browsing files: `more` and `less`. These commands are used for reading files a page (screenful) at a time. For both of these commands, the name of the files should be given as command-line arguments. For example, to read a file named `lab1a.C`, give one of the following commands:

```
more lab1a.C
```

or

```
less lab1a.C
```

After the program starts up, the file can be read a page at a time by hitting the space bar. Type `q` to terminate the program. While the program is running, you can get information on the commands that the program understands by typing a question mark.

In addition, any UNIX editor can be use for browsing files.

Compilers

Large UNIX operating systems usually come with the following compilers:

javac - compiler for the Java language
cc - compiler for the C language
CC - compiler for the C++ language
f77 - compiler for the Fortran language (1977 standard)
pc - compiler for the Pascal language

In addition, two compilers written by the GNU free software foundation are often installed:

gcc - compiler for the C language
g++ - compiler for the C++ language

The compilers will normally read source code from files whose names have a suffix that indicates the language:

.java	- Java
.c	- C
.C, .cc, .cpp, or .cxx	- C++
.f	- Fortran
.p or .pas	- Pascal

Most C++ compilers can also compile a C program, but the source code file should use a C++ suffix.

NOTE: The following does not apply to Java programming, which uses a different model of computation and execution.

If a program in any language is contained in a single source code file, then a UNIX compiler can directly produce an executable file using the `-o` option to specify the name of the executable file. For example, if you have a C or C++ source code file named `myprog.C` and you want to produce an

executable file named `myprog`, then you give one of the following commands:

```
CC -o myprog myprog.C
```

or

```
g++ -o myprog myprog.C
```

UNIX compilers are also designed for separate compilation, where a program is broken up into several files. To do this, first each file is compiled separately to produce a file called an object file, then the object files are linked together to form an executable file. Object files produced by UNIX compilers use the same name as the source code file except the suffix is changed to `.o`. For example, if a C or C++ language program is split into two files `main.C` and `other.C` then it can be compiled into an executable file named `myprog` with the following commands:

```
g++ -c main.C
g++ -c other.C
g++ -o myprog main.o other.o
```

The `-c` option in the first two commands direct the compilers to produce object files. The first command produces an object file named `main.o`. The second produces an object file named `other.o`. The third links these object files together to produce an executable file named `myprog`.

Copying Files

The cp command copies files. It has two forms. To make a copy of a file with a new name use the following form:

```
cp oldName newName
```

oldName is the name of the existing file and *newName* is the name for the new copy.

To copy files to a new location use the following form:

```
cp file ... directory
```

directory must be the name of an existing directory. The remaining arguments name existing files. These files are copied into the directory named by *directory*, keeping their original names.

Displaying Files

The `cat` command displays files. The arguments to the `cat` command should be names of files. All of the files are displayed on the screen without pausing.

Editing Files

Most UNIX systems have two editors: `vi` and `emacs`. Often, there is a third editor: `pico`. Use of these programs is not described here.

Online Documentation

You can get online documentation on all UNIX commands by using the UNIX `man` command. For example, to get documentation on the `ls` command give the command:

```
man ls
```

The quality of documentation given by the `man` command varies considerably.

Electronic Mail

Most UNIX systems have two mail programs: `mail` and `elm`. Often, there is a third mail program: `pine`. Use of these programs is not described here.

Printing Files

Many UNIX systems use the `lpr` command for printing, and some also use the `enscript` command. For all printing commands, the name of the file to be printed is specified as a command-line argument. The printing commands also accept a `-P` option to specify which printer to print on. For example, to print a file named `lab1a.C` using enscript to a printer named `mwah177` you give the following command:

```
enscript -Pmwah177 lab1a.C
```

Recording a UNIX Session

The UNIX `script` command can be used to record a sequence of UNIX commands and the output that they generate. It takes one optional command-line argument. If the argument is given it specifies the name of the file in which the record will be placed. If the argument is missing then the record is placed in a file named `typescript`.

As soon as you give the `script` command, the program begins recording characters that you type along with output generated by programs that you run. You can terminate the `script` program by typing **ctrl-D**.

You should not run an editor or a text browsing program while the `script` program is running. Editors and text browsing programs put out a lot of screen control characters that make the typescript difficult to read. If you

want to display a file and have it captured in the typescript, give the UNIX command

```
cat filename
```

Other Sources of Information

- The C Shell tutorial
- An Introduction to the C shell
- Archive for "Using csh & tcsh"

[an error occurred while processing this directive]