

Contents	Page
What is UNIX?	1
UNIX Users.....	1
The UNIX File System.....	1
The Shell	1
The Tool Concept	1
Standard I/O	1
Pipelines - Building Blocks.....	1
Getting Help - The Man Pages and InfoExplorer	1
Logging On	2
UNIX Commands - A Sampling.....	2
cat	2
more	2
ls	2
chmod.....	2
cd	3
mkdir	3
pwd.....	3
rm	3
rmdir.....	3
cp	3
mv.....	4
lpr	4
Editors - A Summary	4
Pico	4
Miscellaneous Other Commands	5
History - A Poor Typist's Helper	5
Compiling and Running Programs	6

What is UNIX?

UNIX is a high-function, interactive, multi-tasking, multi-user operating system. While UNIX is very powerful, it is not user friendly, which will become evident as we discuss UNIX commands.

UNIX originated at ATT Bell Laboratories in the late 1960s. It is based on an *open-architecture* in which all interfaces and interprocess communication is standard. This facilitates expansion of the operating system and development of new system and application features.

UNIX Users

UNIX users are assigned a user name which is the login id, such as ralph. The user is also assigned a userid, which is a unique numerical identifier, such as 2398. In addition, each user is assigned a group id which can be used to gather a collection of users into common affiliation; this is useful in managing projects. The user name, user id, and group id are assigned by the system administrator and cannot be changed by the user.

The UNIX File System

The file system is the basis of UNIX, or more correctly is UNIX. All elements of the system are files. There are conventional text and data files, binary files, and special files called directories, which point to collections of files. Each file is related to other files via the file system tree structure, and all directories descend from the top-most directory which is known as */*. Thus,

```
/u/sam/project1/data/feb24
```

points to a specific file, **feb24**, which exists in a tree structure that descends from */* (the top-most directory) to the *u* directory (the user files) to the *sam* directory (a specific user) to the *project1* directory (a specific project belonging to sam) to the *data* directory which contains the specific file. Files names can be alphabetic or numeric and can contain special characters.

Each file is associated with an owner. Each file also has a set of access flags associated with it, known as the file mode, which indicate what access privileges are assigned to the file's owner, the owner's group, and other users. Basic privileges are read (r), write (w), and execute (x). More will be presented about this below.

The Shell

All UNIX commands, processes, etc., operate under control of a shell, the user interface with the UNIX system, which interprets commands and passes them to the operating system, and then returns operating system information to the user. There are three principal shells: **sh** - The Bourne Shell, **csh** - The C Shell, and **ksh** - The Korn Shell. Each shell provides a set of commands it recognizes. While most commands are common to all shells, there are special features unique to each shell. Specifics of the

different shells will not be covered here. The command forms presented assume the Shell is being used.

The Tool Concept

A major underlying principle of UNIX is the concept of the tool. Rather than include elaborate applications systems, UNIX provides a set of basic tools each of which is designed to perform one function well. Each tool has a single input called Standard Input (**SI**) and two outputs called Standard Output (**SO**) and Standard Error (**SE**). Therefore, each tool presents the same user interface. Optional input in the form of files or command modifiers may be specified or additional output may be produced; the principal function, however, is controlled by **SI** and **SO**. With this standard, the output of one tool can readily serve as input to another. Using this, commands can be connected allowing basic tools to build more complex ones. When some previously unrecognized requirement is discovered, either a new tool is developed or existing tools are combined to perform the new function.

Standard I/O

In the traditional UNIX environment, Standard Input comes from the keyboard while Standard Output and Standard Error go to the video display. There are times, however, when this is not desirable, such as when input is complex or lengthy, or output is to be saved. Consequently, UNIX allows Redirection of both input and output. Redirection of Standard Input is indicated by **<** while redirection of Standard Output is indicated by **>**. For example, suppose the command **foo** entered without any redirection mirrors typed input. Using redirection, the command can be used to copy a file as follows:

```
foo < input.file > output.file
```

Pipelines - Building Blocks

In order to combine individual tools, some means is needed to pass the output of one tool as the input to the next. This can be done using intermediate files.

```
fee < somefile > feeout
```

```
fifo <feeout > fifo-out
```

```
fum < fifo-out > anew.and.betterfile
```

However, this is expensive in terms of disk space and an inefficient use of processor resources. Instead, UNIX provides pipes (**|**) which allow the connection of two or more processes, passing the output from the leading process as the input to the trailing process. Thus tools can be connected to develop new tools.

```
fee < somefile | fifo | fum > anew.and.betterfile
```

Getting Help - The Man Pages and InfoExplorer

Documentation for UNIX is available on-line via the **man** command, which can be used to display information on a specific command. For example, to get information on the man command, enter:

```
man man
```

Or, the **man** command with the **-k** parameter can be used to obtain a list of commands related to a particular keyword. To get a list of commands dealing with the keyword **exit**, enter:

```
man -k exit or apropos exit
```

The output from the man command is displayed using the **more** filter, one screen at a time.

On Grinder, under AIX, InfoExplorer is available. InfoExplorer is an on-line information system consisting of articles connected by hypertext links, a one-way path from one piece of information to another. Type **info** to access the InfoExplorer.

Logging On

Users logon to UNIX by entering their user name, which was assigned to them by the system administrator, in response to the system's login prompt. In most systems, the user will then be prompted for a password. Initially, this is assigned by the system administrator also. All users should change their passwords at first logon and periodically thereafter for security reasons.

Some systems implement mandatory periodic password changes. In this case, after the password has been entered, the system will ask the user to enter a new password, then to enter the new password a second time as verification. On subsequent logons, the user should enter the new password that he or she chose.

To change a password any time after logon, enter the **passwd** command with no operands. The system will prompt for the user's current password. After this is entered and verified by the system, the user will be asked to enter a new password and then asked to enter it a second time as verification. From then on, the user should use the new password until he or she changes it again. In the event of a forgotten password, contact the system administrator.

To terminate a UNIX session, enter:

```
logout or exit
```

UNIX Commands - A Sampling

Below is an overview of common UNIX commands. It is by no means an exhaustive list of commands nor of options applicable to those commands. A couple words about UNIX command syntax and structure: first, UNIX is **case sensitive**. Generally, commands and parameters (usually single letters) are entered in lower case. If a command parameter is shown in upper case, it must be entered in upper case because a different parameter may

exist that has another meaning and is designated with the same letter but in lower case. Less common but also a possibility are parameters which are **position sensitive**. Therefore, it may be a good idea to try out new commands or old commands with new parameters on a non-critical application.

UNIX commands can be interrupted by pressing **Control C**.

cat - concatenate files

The **cat** command reads multiple files as input and produces a single file as output with the contents of the input files stacked one after another. Output is to Standard Output unless redirected. If no input files are specified, input comes from Standard Input. For example:

```
cat small.file bigger.file > verybigfile
```

more - list file(s)

The **more** command reads one or more files and displays them on Standard Output, one screen at a time. To see the next screen, press the space bar. To see the next line, press the return/enter key. When displaying multiple files, **more** will stop at the end of each file.

```
more small.file
```

To display the previous screen, type **b**. Output from other commands is often piped to **more** so the results will be displayed one screen at a time.

ls - list directory contents or file statistics

When specified without operands, the **ls** command lists the contents of the current directory. Optionally, an alternative directory or even a filename may be specified, in which case the information on the specified directory or file is listed.

```
ls /somedir
```

The **-l** option may be specified on the **ls** command and a long listing will be produced identifying the mode (access flags), the owner, the file size, and the last modification date and time.

```
ls -l somedir or ls -l somefile
```

chmod - change the mode (access flags) of a file

The **chmod** command can be used by the owner of a directory or file to set the values for the file mode defining the access privileges.

The access flags consist of three octal digits. The first digit represents the access privilege of the owner, the second of the group, and the third of other users. The bits of each octal digit represent read, write, and execute privileges, respectively. For

example, the octal digit 6, bit pattern 110, would indicate read and write privileges.

The **chmod** command can be used to specify the value of the three octal digits to be assigned as the mode of the file. For example:

```
chmod 751 somefile
```

This will assign read, write, and execute privileges for the file owner (first octal digit is 7, which is bit pattern 111), will assign read and execute privileges for the group (second octal digit is 5, which is bit pattern 101), and will assign execute-only privilege for others (third octal digit is 1, which is bit pattern 001).

However, some users prefer a more mnemonic approach, especially if only changing one setting. In this case, the character designation u (for user/owner), g (for group) and o (for others) can be associated with the privilege designation r (for read), w (for write), and x (for execute) using either a + to indicate that the privilege is to be added or a - to indicate it is to be removed. For example:

```
chmod uo+x somefile
```

This will assign execute privilege (x) to user and other.

cd - change directory

When users first log onto a UNIX system, they are placed in the home directory. If the user wishes to move to another directory, the **cd** command is used to effect this change. At its most basic, the **cd** command moves to a next lower directory when specifying:

```
cd nextdir or cd ./nextdir
```

where . represents the current directory. Optionally, an entirely separate path can be entered by starting with the top-level directory /. For example,

```
cd /usr/special/files
```

The home directory can be referenced as ~ so it is possible to descend another branch directly. For example, if the user's home directory is /u/mine and the user is at /u/mine/some/deeper wanting to go to /u/mine/somewhere/else, they can enter:

```
cd ~/somewhere/else
```

To pop up one level in the directory structure enter:

```
cd ..
```

This construct can also be used to change to another directory with an attachment point one level above the current directory. For example:

```
cd ../another
```

mkdir - make a new directory

The **mkdir** command is used to create a new directory. The most common use is to create the new directory attached to the current directory. For example, if the current directory is /u/myaccount, the command **mkdir newdir** will create the new directory attached as /u/myaccount/newdir. However, new directories can be created at any level as long as the user has write permission for the parent directory. For example,

```
mkdir existdir/newdi
```

creates newdir under existdir, which is attached to the current directory.

pwd - print working directory

Sometimes moving about in directories can cause users to lose track of where they are. To determine the name and path of the current directory, enter:

```
pwd
```

and the system will print the full pathname of the current directory.

rm - remove files or directories

The **rm** command in combination with the wildcard character * can be used to delete files or entire directories, including all files. For example,

```
rm *data
```

will delete all files in the current directory ending with the characters "data". A specific file can be deleted with:

```
rm myone.and.only
```

Normally, **rm** works silently. The parameter -i can be specified to request that the system prompt for confirmation before deleting each file, although that can be a lengthy process. The command:

```
rm -i *data
```

will request permission before deleting each file ending with "data".

Optionally, the -r parameter can be specified to indicate that the filename is a directory and it, along with all the files in it, is to be deleted. Care should be exercised in using this option. For example,

```
rm -r projdir
```

will delete the directory projdir along with all the files in that directory.

rmdir - remove an empty directory

The **rmdir** command is used to delete a directory that contains no files. If the directory is not empty, the command will be rejected.

The **rm -r** command can be used instead. For example,

```
rmdir projdir
```

will delete the directory projdir if it does not contain any files.

cp - copy files or a directory

The **cp** command is used to make a copy of a file, to copy several files to a directory, and to copy the contents of one directory to another. Wildcards may be used. To copy a single file, enter:

```
cp original itsacopy
```

To copy several files to a directory, enter:

```
cp file1 file2 mydirectory
```

Finally, supplying the **-r** parameter with the **cp** command allows an existing directory to be copied in its entirety:

```
cp -r olddir newdir
```

If the target directory does not exist, it will be created as long as the user has write privileges in the current directory.

mv - move files or a directory

The **mv** command is used to move a single file, move several files to a directory, and to move the contents of one directory to another. Wildcarding is permitted. In essence, **mv** accomplishes a rename of a single file or directory in that the original file or directory no longer exists. To move a single file, enter:

```
mv original itsacopy
```

To move several files to a directory, enter:

```
mv filea fileb mydirectory
```

Note that **filea** and **fileb** will no longer exist in the source directory.

Finally, supplying the **-r** parameter with the **mv** command allows an existing directory to be moved in its entirety, as in:

```
mv -r olddir newdir
```

lpr - print a file

The **lpr** command is used to print a file. By default, the **lpr** command prints to the printer defined by the system administrator as the local printer. However, there may be other printers attached to the UNIX system that can be accessed via **lpr**. In those cases, the **-P** parameter is used to identify which printer is to be used. For example,

```
lpr myfile
```

will print **myfile** on the local/default system printer, whereas:

```
lpr -P main myfile
```

will print **myfile** on printer main. Ask your system administrator which printers are available. Please note that there is

no default "filtering"; that is, there is no facility for changing the format of the file being sent to the printer. If the printer supports PostScript, the file sent to it must be PostScript. Some systems provide filter programs; check with your system administrator.

Editors - A Summary

Basic UNIX includes the **ex** line editor, the **sed** stream editor, and the **vi** full-screen editor.

The line editor is not widely used in interactive processing.

The stream editor is useful in shell scripts (canned routines) for modifying files under control of a file containing edit commands.

The **vi** full-screen editor provides most needed functions, though it is not a widely used editor.

Numerous other editors have been imported into UNIX, which are too complex to explain here; use the **man** command to review the documentation.

One of the easier ones is Pico, which is available on Grinder, and which is briefly explained below.

Pico

Pico is a simple, easy-to-use text editor. To use it on Grinder, at either ksh or csh prompt type:

```
pico
```

to edit a new file, or:

```
pico file_name
```

to edit an existing file.

The status line at the top of the display shows pico's version, the current file being edited, and whether or not there are outstanding modifications that have not been saved. The third line from the bottom is used to report informational messages and for additional command input. The bottom two lines list the available editing commands.

Each character typed is automatically inserted into the buffer at the current cursor position. Editing commands and cursor movement (besides arrow keys) are given by typing special control-key sequences. A caret, **^**, is used to denote the control key, sometimes marked CTRL. The CTRL-q key combination, therefore, is written as **^q**.

The functions below are available in Pico; where applicable, corresponding function key commands are in parentheses.

- ^ g** Display this help text
- ^ f** Move forward a character
- ^ b** Move backward a character
- ^ p** Move to the previous line
- ^ n** Move to the next line
- ^ a** Move to the beginning of the current line
- ^ e** Move to the end of the current line
- ^ v** Move forward a page of text
- ^ y** Move backward a page of text
- ^ w** Search for (where is) text, neglecting case
- ^ l** Refresh the display
- ^ d** Delete the character at the cursor position
- ^ ^** CTRL-^, mark cursor position as beginning of selected text
Note: Setting mark when already set unselects text
- ^ k** Cut selected text (displayed in inverse characters)
Note: The selected text's boundary on the cursor side ends at the left edge of the cursor. So, with selected text to the left of the cursor, the character under the cursor is not selected
- ^ u** Uncut (paste) last cut text inserting it at the current cursor position
- ^ i** Insert a tab at the current cursor position
- ^ j** Format (justify) the current paragraph
Note: paragraphs delimited by blank lines or indentation
- ^ t** To invoke the spelling checker, press space to continue spelling check
- ^ c** Report current cursor position
- ^ r** Insert an external file at the current cursor position
- ^ o** Output the current buffer to a file, saving it
- ^ x** Exit Pico, saving buffer

Pine and Pico are trademarks of the University of Washington. No commercial use of these trademarks may be made without prior written permission of the University of Washington.

Miscellaneous Other Commands

There are a number of other commands available to the UNIX user. Below is a brief summary of some of these.

who - find out who is logged on

To see which users are currently logged onto the system, enter:

who

Optionally, if you forget who you, are enter: **whoami**

finger - gain information about users

Entered without parameters, the finger command provides more detailed information on users logged on, including their "real name"; for example:

finger

When specified with a user name, it will provide detailed information to the extent provided by the user and/or system administrator; for example, **finger ralph**.

If you wish to publish additional information about yourself, create a **.plan** and **.project** file in your home directory. By convention, **.project** identifies the group with which you are affiliated, and **.plan** identifies your short-term or long-term goals. As always, included information should conform to good networking etiquette standards.

mail - send and receive mail messages

Grinder provides full mail services with many available options to users within the Grinder system only.

Briefly, entering:

mail

will display a list of mail messages received. To view a message, type,

t#

where # is the message number. To reply to a message, type:

r#

To save a message, type,

s# filename

where **filename** is the file to contain the message. To delete a message, type:

d#

To quit, type:

q

To send a message, enter the mail command as follows:

mail -s subject recipient

where **subject** is the topic of the message and **recipient** is the person to whom the message is being sent. A simple name such as **joe**, implies that the recipient is on the same system. Otherwise, an Internet address should be supplied, such as **joe@other.place.edu**.

The mail program is then ready for input. You may type your message directly but note that there is no automatic line wrap; you must insert carriage returns at the ends of lines to make them readable. Optionally, you may type

~r filename

to cause the file identified as **filename** to be inserted as part of the mail message. This allows composition of the message with an editor. In either case, when you are ready to send the message, hold down the Control Key while typing d (**CTRL-D**). You will then be prompted to enter recipient addresses for the carbon copy (cc:) list. Sometimes it is convenient to specify your own address

so that you can receive the message, as it was sent, to verify content and format. Typing **Enter** will send the message.

History - A Poor Typist's Helper

In the Shell, a feature exists called history which keeps a record of the *n* most recently entered commands.

To review this list of recently executed commands, type:

history

The list will be printed with each command preceded by a number, the most recent having the highest number.

history may be used as a shortcut for re-entering commands, particularly long commands with many parameters. Using history, you may re-execute any command in the list simply by typing

rn

where **n** is the number of the command (from the history list) to be executed.

Compiling and Running Programs

While any number of compilers are available for UNIX, we will discuss two: C and FORTRAN.

Each compiler may have specific parameters, which can be supplied during compilation. Check the **man** pages or other system documentation for the system you are using.

The invocation of the compiler also invokes the loader. As a standard, each compiler in UNIX produces an executable module from the compilation and load, and this module is named **a.out**. However, each compiler also recognizes a parameter (-o), which permits the naming of the load module.

For FORTRAN:

f77 -o mymod myprog.f
or
xlF -o mymod myprog.f (on Grinder)

For C:

cc -o mymod myprog.c

Note that C source programs end in **.c** and FORTRAN source programs end in **.f**.

Multiple files can be compiled by specifying a group of names:

cc -o mymod firstprog.c secprog.c

Optionally, the -c parameter can be supplied to the compiler and an object module, instead of an executable module, will be produced. An object module is designated by the suffix **.o**. At a later time, these can be reprocessed, maybe with the compilation of new source files, to produce a executable module. For example:

cc someprog.c

which will produce an object module **someprog.o** and later this can be processed by:

cc -o mynewmod other.c someprog.o

There are other parameters that can be supplied to the compilers. Check the **man** pages and with your system administrator.