Chapter 2

UNIX Command Summary

2.1 Overview

Just a few things that we all need to know before we start.

- Unix is case sensitive... `PROG1.C` is *not* the same as `prog1.c`
- `.` is the name of the current directory
- `..` is the name of the current parent directory
- `$HOME` is a variable that contains the full name of your home directory. It contains something like `/usr2/z912345` which includes your login-ID (your Z number)
- files that begin with a '.' are hidden files (see ls(1))
- the default command-line prompt on `mp` is `%`

2.1.1 Additional Documentation

Some other sources that you'll want to check are:

- *Unix in a Nutshell*, Daniel Gilly an O'Reily book.

- *Unix System V A Practical Guide*, 3rd ed.
- The online Unix reference manual (see below)

2.2 Online Unix Reference Manual

The online Unix reference manual is read by using the man(1) command. Most texts that reference Unix commands include the Unix manual page section as a reference. For example, the reference to man(1) in this sentence indicated that `man` is a command that is documented in section 1 of the online Unix reference manual. This online manual system displays what are commonly known as ”man pages”.

Most of the man-pages can be overwhelming the first time you experience them. Skim them when help is needed in order to try to figure out what is needed. If they do not help, look elsewhere, but return to the man-page afterward in order to try to figure out how that information was presented in the online manual and how you could have determined the answer without the outside help. After a while, you will understand the philosophy of the man(1) system and how to use it for fast answers to questions.

Useful options to the man(1) command are -s *section* and -k *keyword*

-s *section* Used to specify the section number of the desired man-page. The section number is optional, and often not necessary. However if a command-line program has the same name as a C function, or something else that may be in the on-line manual, you may get a man-page for the term you specify, but from the wrong section and, thus, the wrong thing. For example, there is a write(1) and a write(2) in the online manual. If you leave out the section number, you will get the page for write (1) which is a command used to send instant-messages to other users on the system (write(2) is the doc for a C function that can be used to write data into a file).

In order to prevent confusion about these things, the man-pages include a set of notes at the end that include references to other man-pages that may be more suitable to what you are looking for. Additionally, the online manual is sectioned based on the types of documents of interest to programmers. Commonly accessed sections are:

section 1	User Commands
section 2	system calls
section 3c	C library functions
section 3f	Fortran library functions
section 4	devices and network interfaces
section 5	file formats

The sections of interest to C programmers are section 2, 3 and 3c. Examples of how to use the online manual include:

```
man ls
man -s1 ls
man -s3c strstr
man -s2 write
man -s3c printf
```

-k *keyword* This option is used to request a keyword search. It will result in giving you a list of pages including the specified *keyword* in their name or in their synopsis. For example:

```
man -k printf
man -k strncmp
```

2.3 Command Line Editing

There are some special key-strokes that will help you when typing on the command line.

- `^U` will backspace and delete the entire line
- `^W` will backspace and delete a word at a time
- `^H` will backspace and delete the previous character
- `^C` will interrupt the running program
- `^Z` will suspend the running program (you may type `fg` to start it back up again)

2.4 Changing Your Unix Password

You may change your Unix password at any time by logging into `mp.cs.niu.edu` and using passwd(1). An example password change scenario is shown below:

```
% passwd                              - type the command
passwd:  Changing password for winans - the program comments
Enter login(NIS+) password:           - enter my new password
New password:                         - and again to verify
```

2.5 ps(1)

To see what is happening on a Unix machine, you may use ps(1) to get a listing of the programs (called processes) that are currently executing. To get a complete listing, consider running `ps -ef`

2.6 quota(1M)

At NIU, each student may use only a limited amount of disk space. In order to check your current usage and limit, use quota(1M). Most of the time you will want to use: `quota -v`

2.7 File Management Commands

2.7.1 ls(1)

To see what files are in a given directory, use ls(1). You can used several options to give you different types of information. Some common options include:

`-a` will list all the hidden files as well as the normal ones

`-l` gives a longer listing with date, time and permissions

`-R` will recursively list all subdirectories and their files

`-d` will list only the directories and their files

`-F` will put a special character behind each filename to indicate the type of the file. These are

`/`	for a dir
`*`	for executable files
`@`	for symbolic links

2.7.2 mkdir(1)

To create a new directory, use mkdir(1). A common option is -p which will create intervening parent directories if they don't already exist.

Some examples:

```
mkdir Assn1
mkdir progs/assn1
mkdir -p this/is/a/test
```

2.7.3 cd(1)

Changes the current working directory. Some examples:

- `cd` Changes the current working directory to the user's home directory
- `cd ..` Changes the current working directory to the parent directory
- `cd 241` Changes the current working directory to a subdirectory named 241; note that the 241 subdirectory must already exist for this to work.

2.7.4 rm(1)

You delete unwanted (and if you're not careful... wanted) files with rm(1). *Warning*: once you have deleted a file it is *gone* for good! There is no "undelete" like in DOS.

A common option is:

- `-I` will ask you to answer 'y' or 'n' to make sure you want to delete the file

An example:

```
% rm -I Primes.ps
rm: remove Primes.ps (yes/no)?
```

The `-I` option is most useful when you do things like`rm -I *c` which would delete all files ending with the letter 'c'.

2.7.5 rmdir(1)

Without special options, the rm(1) command can only delete files. To remove directories use rmdir(1). You can only use rmdir(1) on empty directories. An example:

```
rmdir -p $HOME/241/assignments/a4/prog1.c
```

2.7.6 mv(1)

In Unix, you do not "rename" a file or directory, you "move" it. Use mv(1) to move file. You can move a file or directory to a new name in the same directory or into a different directory.

Some examples:

- To rename a file in the same directory

  ```
  mv prog1.c program1.c
  ```

- To move a file into another directory

  ```
  mv prog1.c ../anotherdir/
  mv prog1.c ../diffdir/diffname.c
  ```

- To move a directory into another directory

  ```
  mv subdir ..
  mv subdir ../diffdir/
  ```

2.7.7 cp(1)

To make a copy of a file, use cp(1).

Useful options:

- `-r` will recursively copy a directory, its files, and its subdirectories to a destination directory, duplicating the tree structure.
- `-I` will prompt for confirmation before overwriting an existing file

Some examples:

- To copy two files to their parent directory

  ```
  cp prog1.c prog1.h ..
  ```

- To copy a file to the testprogs directory in your home directory with a confirmation

  ```
  cp -I prog4.c $HOME/testprogs
  ```

2.7.8 cat (1)

To list a file's contents on the screen (normally called standard-out or stdout) use cat(1)–the concatenate command. This may be used to list one or more files.

Useful options:

- `-n` will print the file with line numbers
- `-v` displays any control characters using carat (^) notation

- `-e` prints a $ at the end of a line (useful to detect unwanted spaces at the ends of lines)
- `-t` will print `^I` for each tab (useful to see what the instructor wants when you must generate *exact* output)

Some examples:

- To display a file on the screen with all the above options turned on

  `cat -nvet prog4.key`

- To combine two (or more) files into a big one

  `cat part1file part2file > bigfile`

- To append `prog10.c` to the end of an existing file named `programs`

  `cat prog10.c >> programs`

2.7.9 less(1) & more(1)

less(1) and more(1) do the same thing. They list a file on standard-out while pausing at each screenful of text. less(1) is a nicer lister than more(1). These commands are often used with a pipe `"|"`. When the command runs, you may:

- hit the enter key to advance one line
- hit the space bar to advance a page
- hit `q` to exit
- when using less(1), press the `j` or `k` keys to scroll up and down one line at a time and `b` to scroll up a page at a time

Some examples:

- `cat -nvet prog7.c | more`
- `ls -alFR | less`
- `ps -e | more`
- `more prog3.c`

2.7.10 pwd(l)

To print the name of the current working directory use pwd(1).

2.7.11 chmod(1)

To change the access mode of the specified file(s) use chmod(1). This is a very important topic!! If you're not careful, anyone can look at or even modify your work or E-mail. If a classmate takes advantage of your lack of action in this area, and it is caught with a copy of your work, you will *both* be in a lot of trouble since it is impossible to tell who cheated on whom. So pay special attention to this section. If you were to `ls -l` in your assignment 1 directory you will get the following output:

```
% cd CSCI241/a1
% ls -l
total 14}
-rw-r--r--1 z912345 csci      154 Jan  6 14:55 Makefile
-rw-------1 z912345 csci     5116 Dec 18 13:56 Primes.tex
-rw-------1 z912345 csci      921 Dec 18 13:56 primes.c
```

Look at the first ten characters.

Position(s)	Meaning
1	Either a regular file (-) or a directory (d)
2-4	The permissions for the owner (r) read (w) write (x) execute
5-7	The permissions for the group rwx is the same
8-10	The permissions for other[or world] rwx is the same

Binary values:

000 = 0	001 = 1	010 = 2	011 = 3
100 = 4	101 = 5	110 = 6	111 = 7

Now think of the 1's as on and the 0's as off so if you break up the characters 2-10 up into three groups of three. The pattern of the permissions is always rwx. If you want to have read, write, and execute permissions for yourself and only execute permissions for group and others you would want the following:

```
-rwx--x--x
111001001
```

If we break the bits into groups of three, that gives us 111, 001, and 001 (in binary) and 7, 1, and 1 in octal.

Examples:

- `chmod 711 prog1.c` will perform the above example
- `chmod 700 prog2.c` will only allow access to you, the owner
- `chmod 644 public.letter` will give you read and write and the group and others will get only read access

Note that you need not worry about any of this if you do a chmod 700 on your home directory. This is because setting the permissions on your home directory such that only you can access and manipulate it will prevent anyone else from looking at it, listing the files in it or anything else to it or its contents.

2.7.12 ftp(l)

Use ftp(1) to move files between machines. There are many ftp programs that you can download from the internet that have all sorts of GUI displays and so on. The only thing that matters is that it be capable of `get`-ting from and `put`-ting files to remote computer systems. The most commonly available versions of ftp(1) are command-line versions.

To use command-line ftp(1), you run the program and supply the name of the remote machine to connect to and answer the questions to log into the remote machine. Then use the get and put commands to transfer files back and forth.

FTP INSTRUCTIONS on the NIU lab systems:

1. Select FTP which is found under Communications.
2. When you get the FTP prompt type:

   ```
   open mp
   ```

3. You can use the ls instruction to view the file names in the present directory.
4. If you want to see what directory on your unix account you are presently in type:

   `pwd`
5. To change a directory type:

   `cwd newdirectoryname`
6. To ftp a file from your unix account to a disk in drive a: type:

   `get mpfilename a:\newfilename.extension`
7. To ftp a file from you're a: drive to the current directory in your unix account type:

   `put a:\filename.extension`

   This will copy the file from the a: drive to your unix account and retain the file name.
8. To log off ftp type:

   `bye` or `quit`

Here is a copy of an ftp(1) session getting a file from `mp.cs.niu.edu` and putting it on a windows machine at home. As you can see, the ls(1) command works so you can see what is there and so on.

```
C:\niu>ftp mp.cs.niu.edu
Connected to mp.cs.niu.edu.
220 mp FTP server (UNIX(r) System V Release 4.0) ready.
User (mp.cs.niu.edu:(none)): winans
331 Password required for winans.
Password:
230 User winans logged in.
ftp> ls
200 PORT command successful.
150 ASCII data connection for /bin/ls (207.152.77.4,4618) (0 bytes).
CS
CVS
CSCI241
foo
icap
local
```

```
msql
notes.tex
pub
public-html
tex
tmp
226 ASCII Transfer complete.
79 bytes received in 0.31 seconds (0.25 Kbytes/sec)
ftp> cd CS/241/Assignments/a2
250 CWD command successful.
ftp> ls
200 PORT command successful.
150 ASCII data connection for /bin/ls (207.152.77.4,4619) (0 bytes).
Hanoi.tex
Makefile
hanoi.c
hanoi-iterative.c
test-hanoi.c
226 ASCII Transfer complete.
63 bytes received in 0.04 seconds (1.58 Kbytes/sec)
ftp> get hanoi.c
200 PORT command successful.
150 ASCII data connection for hanoi.c (207.152.77.4,4620) (575 bytes).
226 ASCII Transfer complete.
610 bytes received in 0.10 seconds (6.10 Kbytes/sec)
ftp>
```

2.7.13 diff(1)

Use diff(1) to see if two files are identical and to summarize any differences.

You might be asked to have output that exactly matches the instructor's output. They might refer to this as a "clean diff" because, if the two files are exactly alike, there will be no differences listed and prompt will immediately reappear. If they are not identical (even characters you can't see such as spaces and tabs count) then you will get a message saying as such (see cat(1) to help you tell). The lines that are different will appear with a < with the text from the first file and a > with text that differs in the second file.

For example, if you had the following data saved in file `fn1`:

```
xxx
yyy
zzz
The cat  and the dog slept.
a
 b
  c
```

and the following saved in file `fn2`:

```
xxx
yyy
zzz
the Cat and the dog slept.
a
 b
  zdfgsdfgsdfg
```

The command `diff fn1 fn2` would result in the following output:

```
4c4
< The cat  and the dog slept.
---
>    c
---
>    zdfgsdfgsdfg
```

As you can see, there were two lines that differed in the files and they are on lines 4 and 7.

2.7.14 I/O Redirection

You will be doing some redirection for some of your programs. Redirection can be from standard input, standard output and standard error. The redirection of I/O is documented in csh(1) because it is performed by the Unix command line shell.

Sometimes you will be given a data file to test your program. You use it by running your program (with its command line options) and then using the less-than symbol '<' followed by a filename that contains the data to send to the running program.

To redirect the data in the file named `data1` into prog1

```
prog1 < data1
```

Sometimes, rather than printing on the screen, you will want to save your output in a file for later use. To do this, you redirect standard output to a file using the greater-than symbol '>'.

The operating system will automatically create the file for you if it does not already exist. *Caution:* if the specified output file already exists doing this command will erase the contents of the file and fill it with the new output.

To save the output of `prog2` into a file called `out1`

```
prog2 > out1
```

To append data to the end of a file you would do the following:

```
prog2 >> $HOME/241/out
```

You may also redirect the standard output and the standard error (which includes compile error messages) to a file. (You may be asking what is standard error? It's a separate output stream that is used, by convention, by programs when they are printing error messages). One very handy use is to redirect the errors you get from the c compiler to a file. Then you can print it out..

```
g++ prog1.c >& prog1.err
```

If you want to *append* to the end of the output file, you would again use a double-greater-than symbol like this:

```
g++ prog1.c >>& prog1.err
```

Rather than redirecting the output of a command into a file, you might want to redirect into the input of another program. To do this, you use the pipe symbol. For example, if you want to sort the output from the command xyzzy you would pipe it to sort(1) like this:

```
xyzzy -l | sort
```

Note that the standard input for the xyzzy -l is not altered and will be the keyboard. The standard output of the sort is also not altered and will print to the screen.

You can also mix redirection. For example, if you want your prog to take data from a file and put the output into another file you do the following:

```
prog6 < datafile > output
```

Or you might even want to do something like this:

```
prog6 < datafile | sort > output
```

2.8 Termination and Output

2.8.1 Job Control

Job control is used to run more than one command at a time. In order to run more than one command at a time, you place an ampersand '&' at the end of your Unix command.

To find out what jobs/processes you have going, type `jobs -l` at the prompt and it will list the jobs and their statuses:

```
    [1] + 12345 Running loop
    [2] - 54321 stopped loop2
    [3] 32145   stopped pico prog1.c
```

The number inside the `[ ]` is the job number, the next number is the process identification number of the job, the next column tells about the state of the job, and the last column is the name of the process. The + stands for the current job and the - stands for the previous job.

As you can see 2 and 3 are in the background and 1, the loop, is running in the foreground.

1. To bring job 3 into the foreground type: `fg %3`
2. To put job 1 into the background type: `bg %1`
3. To kill job 1 just type: `kill -KILL %1`
4. (or) To kill job 1 just type: `kill -KILL 12345`
5. To kill the current job just type: `kill -KILL %`
6. Of course you can also use `^C` to kill foreground jobs.

2.8.2 kill(1)

Use kill(1) to send a signal to a process and let it know that it should be stopped. To make sure that the process will die use `kill -KILL (Process id or job number).` This is useful to kill an unwanted endless loop. Depending on what you have gotten yourself into, you may have to login on another terminal to do this.

2.8.3 Printing Files With lpr(1)

To print in the labs at NIU, use lpr(1).

Use the command `lpr -P lpXXX prog1.c` where the XXX is replaced by `csl` if you want your output printed in the Psych/Math building, `ucl` for Graham Hall, `scl` for the Stevens Annex building, and `frl` for Faraday lab.

Examples:

- `lpr -P lpcsl hanoi.c`
- `lpr -P lpucl a1/primes.c a2/hanoi.c`
- `lpr -P lpucl */*.c */*.h`

Chapter 3

Advanced Unix Command Summary

3.1 The Unix Shell: csh(1)

The csh(1) shell is a command interpreter that handles your I/O redirection and job control. If you have questions about those issues, see csh(1).

When csh(1) starts running, it reads and executes a startup file named `.cshrc` from your home directory. You may add any commands to that file that you wish.

3.1.1 Command Aliases

An alias is a nice way to shorten long and/or often typed commands.

To see what all the aliases are, type `alias` at the command/shell prompt.

To make an alias called h that is short for history, you would use:

```
alias h history
```

To get rid of an alias that you don't want, use the `unalias` command. For example:

```
unalias h
```

The `source` command will execute shell commands in from a file. It might come in handy to have a set of aliases for each assignment you are working on. If you have to run your program against a lot of data files it will save you from having to type long commands all the time. You could start by creating a file holding all your aliases: `pico assn1alias` and then doing a `source assn1alias` at the prompt.

An example:

```
alias t 'cc -o prog1 prog1.c'
alias g 'fname/prog1 < input2.file > output.file1
alias r 'cat output.file | more'
alias d 'diff output.file answer.key'
```

Instead of having to type all of this in every time you want check to see if you have the correct output, all you have to do is type `t` then `g` then `r` then `d` and you are off and going. You will simply have to type `source assn1alias` every time you login.

3.1.2 History

The `history` function is very useful. It allows you to quickly enter a command again. To see the last 20 commands you've typed in type history at the prompt.

In general the exclamation '!' followed by the number of the command will reexecute that command.

If you see the following after typing `history`:

```
12 cat prog1.c
13 pico prog1.c
14 man -s3c getopt
15 pico /usr2/z912345/assign1/prog4.c
16 gcc -o prog1 prog1.c
17 pico prog1.c.save
```

- `!!` will redo the last command typed
- `!12` will redo the twelfth command you typed
- `!p` will re-execute the last command that started with a p

3.1.3 Example `.cshrc` and `alias` Files

The hidden file `.cshrc` is executed each time a new csh(1) is started. Since you now know about the `source` command and a little about `history` and `alias`, here is a file that you might want to add to your home directory. This was named `.myalias` so it would be hidden. To get your aliases automatically set when you log in, edit your `.cshrc` file and insert `source .myalias` at the end of the file on its own line.

Here is an example of a file named `.myalias`.

```
# Comments start with a # in column 1. . . like this line
alias h history
# An alias to show the processes of the user
alias psu ps -U \$USER
alias cat cat -nvet
alias ls ls -alF
# An alias to change the prompt so that it will show where you are and the
# number of the command you are typing
alias cd 'cd \!*;set prompt = "$cwd \#\! >"'
alias quota quota -v
alias jobs jobs -l
# Some alias to force the user to acknowledge when deleting or copying
alias rm rm -I
alias cp cp -I
# most of you will probably be printing in the csl lab. ex) Type pcsl prog1.c
alias pcsl 'lpr -P lpcsl \!:1'
# Some alias for DOS users}
alias del rm
alias dir ls
alias cls clear
alias cd.. cd ..
```

Other helpful things to put into your `.cshrc` file

- `set filec` when typing a file name if you push the `Esc` key it will complete the rest of the name for you.
- `set ignoreeof` prevents accidental logout if you type `^D` at the command line

- `set noclobber` prevents the accidental destruction of files by I/O redirection