The Basics of UNIX/Linux

Michael Crocker

This tutorial was made for CSE321 students who are not familiar with Unix/Linux OS. If you have any questions, please contact Prof. Michael Niemier or the teaching assistants.

1 Command: ls

ls	list standard files in the current directory
ls -a	list **all** files in the current directory
ls -l	list files and file attributes
ls *.txt	list files ending in .txt
ls -R	list files and files in all subdirectories

For the command "ls -l" the output might look something like the following.

```
-rwxr-xr-x   1 mcrocker campus    200220 Jun 22 13:02 spec-parser*
-rw-r--r--   1 mcrocker campus     36417 Jun 22 13:03 test.in
-rw-r--r--   1 mcrocker campus      6351 Jun 22 13:01 utilities.c
-rw-r--r--   1 mcrocker campus      4648 Jun 22 13:02 utilities.o
drwxrwxr-x   2 mcrocker campus      2048 Jun 22 13:06 words/
-rw-rw-r--   1 mcrocker campus      1088 Jun 22 13:31 Makefile
```

The first column contains the unix permissions, the third shows the file owner, the fifth shows the file size in bytes, the sixth through eighth show the last modified date, and the last shows the filename.

If you do a combination of the options "-a" and "-l" you will see two directories called "." and ".." which correspond to the current directory and the parent directory. Also, note that in UNIX the forward slash "/" is the separator for directories and subdirectories.

```
drwxrwxr-x   2 mcrocker campus      2048 Jun 22 14:38 ./
drwxrwxr-x   5 mcrocker campus      2048 Jun 22 12:36 ../
```

2 Unix Permissions

When doing a long listing of the files in a unix directory, there will be 10 characters on the far left for each file indicating permissions.

The 10 characters should be broken into sets of 1, 3, 3, and 3. The first tells the file type, either directory or normal. The second set tells the file permissions for the owner. The third set tells the permissions for the owning group. The fourth set tells the permissions for everyone. The three characters either show a hyphen "-" for no permission, or read "r", write "w", and execute "x" permission.

3 Command: chmod

The following example shows how the “chmod” command could be used to change the permissions of a text file so that everyone can read only, except the owner who can read and write.

```
chmod 644 lyrics.txt

-rw-r--r--   1 mcrocker campus     14648 Apr 09 01:23 lyrics.txt
```

The three digit code corresponds to the three types of unix permissions discussed in the last section. The following table shows what each digit means. You can think of it as a truth table.

0	- - -	none	4	r - -	read
1	- - x	execute	5	r - x	read, execute
2	- w -	write	6	r w -	read, write
3	- w x	write, execute	7	r w x	read, write, execute

The most common codes I use are 644 and 755. As I said before, 644 means I can read and write, and everyone else can only read. While only a bit different, 755 means that I can read, write, and execute, while others can read and execute. You can use any combination containing three digits of 0-7.

4 AFS Permissions

At Notre Dame, there are separate permissions for the Andrew File System (AFS). You can list and set these permissions with the “fs” command. I am not going to go into detail here, as the permissions are more complicated and AFS specific. The AFS permissions apply to specific AFS users, the system:anyuser group, the system:administrators group, etc.

fs la .	list AFS permissions for current directory
fs sa . system:anyuser lrw	set permissions for current directory so that everyone can List, Read, and Write

5 Commands: cd, pwd

The command “cd” allows you to change directory. You can always find out what directory you are in by using the “pwd” command (think Print Working Directory).

cd bin	change to the bin subdirectory
cd	change to my home directory
cd ˜/	change to my home directory
cd ˜mcrocker/	change to the home directory for user mcrocker
cd ˜/doc	change to the doc subdirectory below my home directory
cd ..	go up one directory level
cd /usr/local/bin	change to a directory using its full path
pwd	display the current working directory

The tilda and forward-slash characters “˜/” represent the current user’s home directory.

6 Commands: cp, mv, rm, mkdir, rmdir

The “cp” command is for copying files and directories. The “mv” command is for moving files and directories. It can also be used to rename files. Both commands take two arguments, source files and the destination.

cp file1 ../	copy file1 to the parent directory
mv file1 ../	move file1 to the parent directory
cp file1 ../file2	copy file1 to the parent directory renamed to file2
mv file1 file2	rename file1 to file2
mv ./* temp/	move all files in the current directory to the temp directory
mv temp/ ../temp/	move the temp directory to the parent directory

The “mkdir” command creates a new directory, taking one argument, the name of the directory to be created. The “rm” command deletes files, while “rmdir” command deletes directories (only if they are empty). These commands also take one argument.

mkdir old/	make a new directory called old
rmdir old/	delete the directory named old
mkdir ../../user/bin/new	make a new directory called new at that relative path location
rm lyrics.txt	delete the file named lyrics.txt
rm *.txt	delete all text files in the current directory

7 Piping

The “cat” command outputs the contents of a file as ASCII to the standard output, and thus to the terminal. Be careful not to do this with non-text files, as the strange characters can mess up the terminal.

cat lyrics.txt	display the contents of the file as ASCII
cat *.txt	display the contents of all text files as ASCII

This is a good time to bring up the concept of pipes. For example, if a command (or program) outputs text, you can “pipe it” to a file using the greater-than symbol “>”. You can also feed the contents of a file as input to a command using the less-than symbol “<”. If you want the output of one command to be input to another command, use the vertical-line pipe symbol “\|”. If the output of a file is very long, you can use the commands “more” or “less” to view one screen at a time. While these two commands are similar, less is more! Also, if you are only looking for certain lines in a text file, you can use the command “grep” to output only those lines.

cat lyrics1.txt > lyrics2.txt	pipe the contents of lyrics1.txt to new file lyrics2.txt
cat lyrics1.txt >> lyrics2.txt	pipe the contents of lyrics1.txt to the file lyrics2.txt if it exists
bc < sum.txt	pass the contents of sum.txt as input to the big calculator
cat long.txt | more	display the contents of long.txt one screen at a time
cat long.txt | less	display the contents with the ability to go back and forth
cat long.txt | grep “friday”	find all lines that contain the string “friday”

The “more” and “less” commands will not exit until the end of the file. Type ‘q’ to exit early.

8 Background Tasks

All commands discussed so far run in the terminal. However, there are programs which will run in their own window. If they are called from the command line, they will run in a new window, but the terminal that called them will be frozen until they are done. You can run them in the background using ampersand "&" at the end of the command. Useful programs that you might use are "emacs", "xemacs", "xfig", "ghostview", "acroread", "gimp", "matlab", etc.

acroread unixtut.pdf &	view the PDF file without locking up the terminal
xemacs &	run the text editor xemacs in the background

9 Running Tasks

If you want to see what processes are currently running on your machine, use either the command "ps" or "top" to do so. While "ps" works like most other commands, "top" is an interactive program. There are many options for both. To exit from "top" press the 'q' key. Once you know the process ID number, you can try to terminate the process by using the "kill" command. If the process cannot be stopped cleanly, you can force it to die with the dash-9 option.

top	run top interactively
ps	show running tasks
ps -a	show more running tasks
ps -aA	show all running tasks
kill 786	attempt to terminate process #786
kill -9 786	die!

10 Storage Space

Files take up space, and you can check that using the "du" command. Think "Disk Usage" for a directory when using this command. However it will check all subdirectories recursively. The numbers listed are in blocks. If you use the dash-k option, the numbers are in kilobytes. There are also storage limits enforced by AFS. To check your quota, use the "quota" command.

du -k	show disk usage for the current directory and all subdirectories
du -k ~/music/mp3/LedZeppelin	how much space are my Led Zeppelin mp3s taking up?
du -k ~/music/midi/QFG	how much space are my Quest for Glory midis taking up?
quota	show me my AFS quota status

11 Command: man

If you want to know more about some of these commands, you can check their manual pages using the "man" command.

man grep	show me more about the "grep" command

12 Paths, Aliasing, and Startup/Login Files

When ever you attempt to run something from the command line, the system checks your PATH for where to find the executable. Just because a command is not found does not mean it doesn't exist. It must be in your PATH or you must run it by listing the full path. Say that I have put an interesting program into my Public space called "simulator" and you want to run it. If you type "simulator" it will not run. If you type "˜mcrocker/Public/simulator" and you have permissions to execute, it will run. Also, you can add my Public directory to your PATH.

You can also give alternate names to commands and paths using the "alias" command. If the command you are aliasing contains spaces because of arguments, be sure to put single quotes around the whole thing.

echo $PATH	show me my PATH listing
setenv PATH ˜/bin:$PATH	add my home bin directory to my path listing
alias	show all registered aliases
alias z 'cat lyrics.txt | grep "love" '	the next time you type z you will search the lyrics for all lines that contain the word love

Things like your PATH, aliases, and other settings are many times set by your .login and .cshrc files when you login or open a terminal. These files can be found in your home and/or Public directories. Since these files are checked upon startup, they will not take effect until you open a new terminal. If you have made changes and want to use these new settings without opening a new terminal, use the "source" command.

source .cshrc	reload the settings in your .cshrc file
source .logic	reload the settings in your .login file

13 Command: time

The "time" command gives you a way to monitor the runtime of a program. Simply pass the program as an argument to time, and it will track the user, system, and real elapsed time for the execution of that program. You can even time programs that require arguments. Also, use the "-o" option to output the timing results to file, and the "-a" option to append it to the file, otherwise it will be overwritten.

time ls	perform a directory listing and show how long it takes
time gcc -c tree.c	compile tree.c and show how long it takes
time -o output.txt -a du	check disk usage, and write how long it takes to the file "output.txt"

The output might look something like this,

```
1.32u 0.33s 0:01.94 85.0%
```

or like this:

```
20.68user 0.37system 0:42.52elapsed 49%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (471major+66829minor)pagefaults 0swaps
```

The first column shows the user CPU time, the second shows the system CPU time, the third shows the real elapsed time, and the fourth shows the CPU usage. In most cases in which you are recording the time it takes for a program to run, the user time is what you want.

14 C/C++ Executable Generation

For most Linux systems, “gcc” is the basic C compiler. While you will probably use Makefile to help you build C programs, a basic understanding of the compiler command line syntax is important, especially when you create Makefile. The syntax for the GNU C++ compiler, “g++”, is very similar.

gcc -c tree.c	compile tree.c into the object file tree.o
gcc -o tree tree.o	link tree.o into the executable called tree

Note that compiling code can require compiler and linker flags and arguments, which can be inserted at the end of the compile and link commands. Also, when linking, multiple object files can be listed. If C code is contained in many files, compiling can become tedious. The command “make” can be used in conjunction with a “Makefile” to compile large projects more easily. If you are in a working directory that has a “Makefile” in it, simply run the “make” command, and the code will compile. The key is that the “Makefile” must be formatted correctly. Here is a simple one for the creation of the tree program.

```
CC = gcc -O2

OBJS = tree.c

CFLAGS = ‘‘
LFLAGS = ‘‘

tree : $(OBJS)
        $(CC) -o tree $(OBJS) $(LFLAGS)

tree.o : tree.c
        $(CC) -c tree.c $(CFLAGS)
```

In make files, you can define variables using the equal sign “=” to assign a value on the right to a variable name on the left. To use that variable later, use the dollar sign brackets “$()” placing the variable name inside. It is useful to assign the compiler to a variable so that you can change the compiler easily. After defining variables, you can set up rules with the colon “:” separating the rule name on the left and the rule dependencies on the right. On the next line after a single tab, the rule command is given. The first rule defined is the default rule, and is the rule checked when running the “make” command. If you want to run a different rule, you can give it as an argument to make. Also, if you want to add compiler optimizations, the best place to add it is in the CC variable definition. This example shows a “-O2” compiler optimization.

make	check the default rule in the Makefile
make tree	check the rule named “tree”
make tree.o	check the rule named “tree.o”

The key to all this is how a rule is checked. The dependencies listed for a rule must all be checked first before the rule’s command can be run. In the case of our tree program example, if the default rule is checked, then the “tree.o” rule and file must be checked, and thus the “tree.c” rule and file must be checked. Since there is no “tree.c” rule then the “tree.c” file is checked to see if it has been changed since the last time the “make” command was issued. If it has, then the “tree.o” file is recompiled, and then the “tree” executable is relinked. If the “tree.c” file has not been changed, then “make” reports that tree is up to date.

15 Teasers

There are many more things you can do from the command line using various scripting languages, regular expressions, and other commands that I did not discuss in this tutorial. However, I will list them here in categories in case you want to look them up yourself or ask me about them later.

ln	make hard and soft file links
bc	a big calculator
sftp	secure file transfer
ssh	secure remote login
awk, sed, python, perl	scripting languages
lp, lpq, lprm	print commands
vi	the most best editor ever
find	search for files in a directory hierarchy
wc	counts characters, words, and lines in files
dis	disassembler