GGG201A Computer Skills

Location: VM3B 1105 F 4:10-5:00

- Unix/Linux
- Text Editor
- Markdown

Unix/Linux

There is a difference but we don't care.

Text Editor

There are many. Choose one and be done with it.

Markdown

It's really easy. We are going to take all our notes in Markdown.

- Ordinary text, **bold**, *italic*
- Headers - underline or #
- Horizontal line ----
- Lists - various kinds
- Code blocks
- Code inline uses `backticks`
- Blockquotes uses email style > symbols

Your Book

During your course of study, you will be writing your very own how-to guide for introductory informatics for biologists. Like any book, you need a title, authors, chapters, etc. Eventually, you will compile this into an ebook that can be read using a Kindle. That's right, your very own book on the Kindle store (if you want).

Each of your books you create will be written by you. They say nobody learns more than the teacher, so you will be teaching others to optimize your learning.

Let's look at a book I'm in the middle of writing. Things to note:

- I've organized files into directories
- The directories have names that make sense
- The chapter files are organized by number

I do the ebook build with a single command line

```
cat Part*/* | pandoc -o dirtbag.epub --epub-cover-image=Figures/cover.png
```

Another command builds the Kindle form

```
~/Code/KindleGen_Mac_i386_v2_9/kindlegen dirtbag.epub
```

Getting Organized

Before we begin on this informatics journey, we need to get organized. If you currently spray-paint your computer desktop with files, it's time to stop. You need a place where you're building your book and another place you're going to put your code. If you don't have your own plan, create directories called *Code* and *Book* in your home directory.

First Commands

Open a Terminal application in your Unix/Linux operating system and use the `date` command.

```
date
```

Now format the date how you like it. For example, to print the date as year-month-day with a time stamp, do the following.

```
date "+%Y-%m-%d %H:%M:%S"
```

The arcane syntax after the `date` command is a **command line argument**. Many programs have command line arguments and optional parameters (more on that below).

Getting Help

One of the best ways to get help is to Google whatever Unix term you're interested in. But another way is to use the Unix manual pages. This is built-in documentation about all the programs. To get information about the `date` command, type the following:

```
man date
```

Press space bar to load the next page and the `Q` key to quit.

Injury Prevention

Typing is bad for your health. Seriously, if you type all day, you will end up with a repetitive stress injury. Don't type for hours at a time. Make sure you schedule breaks. Unix has several ways to save your fingers. Let's go back run the `date` program again. Instead of typing it in, use the **up arrow** on your keyboard to go backwards through your command history. If you scroll back too far, you can use the **down arrow** to move forward through your history (but not into the future, Unix isn't that smart).

```
date "+%Y-%m-%d %H:%M:%S"
```

You can use the **left arrow** and **right arrow** to edit the text on the command line.

To see your entire history of commands in this session, use the `history` command.

```
history
```

If you want to repeat a particular command line, you can select a specific line of your history. For example, to re-run command line #7, you type the following:

```
!7
```

Tab Completion

Probably the most important finger saver in Unix is **tab completion**. When you hit the tab key, it completes the rest of the word for you. For example, instead of typing out `history` you can instead type `his` followed by the tab key, the rest of the word will be completed. If you use something less specific, hit the tab key a second time and Unix will show you the various legal words. Try typing `h` and then the tab key twice. Those are all the commands that begin with the letter h. We will use tab completion constantly. Not only does it save you key presses and time, it also ensures that your spelling is correct. Try misspelling the `history` command to observe the error it reports.

```
historie
```

Creating and Viewing Files

(For the this exercise and others that follow, it may be useful to have your graphical desktop displaying the contents of your home directory. That way you can see that typing and clicking ar related.)

There are a number of ways to create a file. The `touch` command will create a file or change its modification time (Unix records when the last time a file was edited) if it already exists.

```
touch foo
```

The file `foo` doesn't contain anything at all. It is completely empty. Now lets create a file with some content in it.

```
cat > bar
```

After you hit return, you will notice that you do not return to a command line prompt. Keep typing. Write a bunch of nonsense lines. Keep going. Write poetry if you must. Everything you're entering at the keyboard is now going into the file called `bar`. To end the file, you need to send the end-of-file character, which is **control-D**. This character is at the end of every text file. Hit the control key and then the letter d. Another way of saying this is hit the `^D` character. Note that you don't type the ^ or the capital D. It's just the way we communicate in writing that one should type the control key and then the d key.

To see the contents of the file, you can also use the `cat` command. This dumps the entirety of the file into your terminal.

```
cat bar
```

This isn't very useful for viewing unless you're a speed reader. You can inspect the first 10 lines of a file with the `head` command.

```
head bar
```

Similarly, you can inspect the last 10 lines with `tail`.

```
tail bar
```

Both of these commands have command line options so that you can see a different number of lines. For example, to see just the first line you would do the following:

```
head -1 bar
```

A more useful way to look at files is with a **pager**. The `more` and `less` commands let you see a file one terminal page at a time. This is what you used before when viewing the manual page for the `date` command. Use the spacebar to advance the page and q to quit. `more` and `less` do more or less the same thing, but oddly enough `less` does more than `more`.

```
less bar
```

Editing Files

You can edit files with nano or gedit. Let's stay in the terminal and use nano for now.

```
nano bar
```

This brings up a terminal-based editor. Now you can change the random text you just wrote. Use the arrow keys to move the cursor around. Add some text by typing. Remove some text with the delete key. At the bottom, you can see a menu that uses control keys. To save the file you hit the ^O key (control and then the letter o). You will then be prompted for the file name, at which point you can overwrite the current file (bar) or make a new file with a different name. To exit nano, use ^X. Note, you don't need to give nano an argument when you start it up.

Open up a new terminal. Create a new file called `unix_notes.txt` (either by touching it first or saving an empty nano file). Use this to record the various things you've learned today. You might want to go back to this later.

Unix file names often have the following properties:

- all lowercase letters
- no spaces in the name (use underscores or dashes instead)
- an extension such as .txt

Navigating Directories

Whenever you are using a terminal, your focus is a particular directory. The files you just created above were in a specific directory. To determine what directory you are currently in, use the `pwd` command (print working directory).

```
pwd
```

This will tell you your location is `/home/bios180student` (or maybe not if you're reading this outside the context of the class or if the documentation is out of date - it doesn't matter, this is your home directory). The current directory is also known by the single letter `.` (that is not the end of the sentence, it's the period character). If you want to know what files are in your current working directory, use the following command:

```
ls .
```

`ls` will list your current directory if you don't give it an argument, so an equivalent statement is simply:

```
ls
```

Notice that `ls` reports the files and directories in your current working directory. It's a little difficult to figure out which ones are directories right now. So let's add a command line option to the `ls` command.

```
ls -F
```

This command adds a character to the end of the file names to indicate what kind of files they are. A forward slash character indicates that the file is a directory. These directories inside your working directory are sub-directories. They are **below** your current location. There are also directories **above** you. There is at most one directory immediately above you. We call this your parent directory, which in Unix is called `..` (again, not the end of this sentence, but rather two period characters together). You can list your parent directory as follows:

```
ls ..
```

The `ls` command has a lot of options. Try reading the `man` pages and trying some of them out. Now is a good time to experiment with a few command line options. Note that you can specify them in any order and collapse them if they don't take arguments (some options have arguments). Don't forget to log your learning in your other terminal.

```
man ls
ls -a
ls -l
ls -l -a
ls -a -l
ls -la
ls -al
```

There are two ways to specify a directory: **relative** path and **absolute** path. Everything so far has been the relative path. The command `ls ..` listed the directory above the current directory. The command `nano bar` edited the file `bar` in the current directory. What if you want to list some directory somewhere else or edit a file somewhere else? To specify the absolute path, you precede the path with a forward slash. For example, to list the absolute root of the Unix file system, you would type the following:

```
ls /
```

To list the contents of /usr/bin you would do the following

```
ls /usr/bin
```

This works exactly the same from whatever your current working directory is. But this is not true of the relative path.

```
ls ..
```

To change your working directory, you use the `cd` command. Try changing to the root directory

```
cd /
pwd
ls
```

Now return to your home directory by executing `cd` without any arguments.

```
cd
pwd
ls
```

Now go back to the root directory and create a file in your home directory. There are a couple short-cut ways to do this. The `HOME` variable contains the location of your home directory. But ~ (tilde) is another shortcut for the same thing and requires less typing. Note that to get the contents of the `HOME` variable, we have to precede it with a `$`.

```
cd /
touch $HOME/file1
touch ~/file2
```

Moving and Renaming Files

Your home directory is starting to fill up with a bunch of crap. Let's organize that stuff. First off, let's create a new directory for 'Stuff' using the `mkdir` command.

```
cd
mkdir Stuff
```

Notice that the directory starts with a capital letter. This isn't required, but it's a good practice. Now let's move some files into that new directory with the `mv` command.

```
mv foo Stuff
```

If there are 2 arguments and the 2nd argument is not a directory, the `mv` command will rename the file. Yes, it's weird that `mv` both moves and renames files. That's Unix.

```
mv bar barf
mv barf Stuff
ls
```

You can move more than one file at a time. The last argument is where you want to move it to.

```
mv file1 file2 file3 file4 Stuff
ls
ls Stuff
```

Copying and Aliasing

The `cp` command copies files. Let's say you wanted a copy of barf in your home directory.

```
cp Stuff/barf .
ls
ls Stuff
```

Now you have 2 copies of barf. If you edit one file, it will be different from the other. Try that with nano.

```
nano barf
# do some editing and save it
cat barf
cat Stuff/barf
```

When you do the copying, you don't have to keep the same name.

```
cp Stuff/barf ./bark
ls
```

Things are getting a little messy, so let's put these files back into the Stuff directory. But let's do it using the * wildcard, which is somewhat magical because it fills in missing letters of arguments. Every file that starts with `ba` will now be moved to Stuff.

```
mv ba* Stuff
ls
```

That was just two files, but I think you can imagine the power of moving 100 files with a single command.

An alias is another name for a file. These are often useful to organize your files and directories. Let's try it using the `ln -s` command (we always use the `-s` option with `ln`).

```
ln -s Stuff/foo ./foof
nano foof
# do some editing
cat foof
cat Stuff/foo
```

Deleting Files

You delete files with the `rm` command. Be careful, because once gone, they are gone forever.

```
ls Stuff
rm Stuff/file1
ls Stuff
rm foof
```

You didn't type those file names completely, right? You used tab completion, right?

You can delete multiple files at once too and even whole directories. But you're better off using the graphical interface to delete files. Deleting files is not something we wish you to automate.

Binary Files and Compression

Most of the files you use will be in text format. What's the alternative to text? Binary. If you want to see what a binary file looks like, try using `less` on the `less` program itself. Use the `which` command to find out where `less` is.

```
which less
```

On my computer, it's at `/usr/bin/less`.

```
less /usr/bin/less
```

Bleh. Those are machine instructions not meant for human consumption. Every once in a while, you will need to use non-text files and they will look something like this. Why? Either for efficiency in time or space.

One of the more common binary files you will see are compressed files. These are typically created with the `gzip` command and decompressed with the corresponding `gunzip` command. To begin, let's download a text file of sequence. Let's not make it a huge file. Use your web browser to go to the NCBI and download E. coli K12 or something. Alternatively, just type random letters into an editor and save it. Let's say you save the file as `sequences.fasta`. To compress that with `gzip` is simple.

```
gzip sequences.fasta
```

Note that this replaces the original file with a new file containing a `.gz` extension. What was once a text file is now a compressed binary file. Sequences typically compress about 3-4 fold. So the human genome, which is normally about 3G can be stored in less than 1G. That's a good thing. To restore the binary file to text, you `gunzip` it.

```
gunzip sequences.fasta.gz
```

What if you have multiple files you want to compress? The best thing to do is to put them all into a *tarball* and then compress it. A *tarball* is a file created by the unix `tar` command. Let's create a directory, put some files in it, and then create the archive.

```
mkdir Stuff
touch Stuff/file1
touch Stuff/file2
tar -c -f stuff.tar Stuff
```

That last command line told the `tar` program to create `-c` an archive with filename `-f` stuff.tar. We can collapse the single letters if we want.

```
tar -cf stuff.tar Stuff
```

Now we can compress this with `gzip`.

```
gzip stuff.tar
```

There's an even more convenient way to do this because `tar` knows how to compress files on its own.

```
tar -czf stuff.tar.gz Stuff
```

To decompress the archive, we do the following

```
tar -xzf stuff.tar.gz
```

This creates the Stuff directory and everything inside it. Try moving the `stuff.tar.gz` file to somewhere else in your home directory and inflate it. You'll find you now have another copy of the files.

Using File Permissions

A file can have 3 kinds of permissions: read, write, and execute. These are abbreviated as `rwx` when you do a `ls -l`. Read and write are obvious, but execute is a little weird. Programs and directories need executable permission to access them. Important data files, like the genome of C. elegans, should not be edited. To ensure that, they should have read only permission. Every file has permissions for the user, group, and public. So a file can be readable by you and inaccessible to the public. Let's look at some files.

```
ls -l
```

The first character is a hyphen. This means the file is an ordinary file. If the file was a directory, the letter would be a d. If it was an alias, the letter would be an l. The 3 rwx symbols that follow are the read, write, and execute permissions for user, group, and public. So everyone has permission to read, write, and execute this file. If you copy files from a USB flash drive, it will typically have all these permissions because the file system on most flash drives is not Unix-based. Having all permissions on is generally not a good idea. Most files fit into one of a few categories.

Category	**Code**	**Oct**	**Meaning**
raw data	`-r--r--r--`	444	anyone can read, nobody can write
private	`-rw-------`	600	I can read/write, others nothing
shared	`-rw-r--r--`	644	I can read/write, group/public can read
shared	`-rw-rw-r--`	664	We can read/write, public can read

You can change the permissions of a file with the `chmod` command. There are two different syntaxes. The more human readable one looks like this.

```
chmod u+x chromosomes.txt
```

The octal format is convenient because it changes all of the permissions at once, but can be confusing for people who don't think in octal. 4 is the read permission. 2 is the write permission. 1 is the execute permission. Each rwx corresponds to one octal number from 0 to 7. So `chmod 777` turns on all permissions for all types of people and `chmod 000` turns them all off. I generally use only two permissions: 444 and 644. That is, everything is readable, but somethings only I can write.

Unix Reference

Token	Function
.	your current directory (see pwd)
..	your parent directory
~	your home directory (also $HOME)
^c	send interrupt signal to current process
^d	send end-of-file character
^z	send sleep signal current process: see bg and fg
tab	tab-complete names
*	wildcard - matches everything
&	send command to background
|	pipe output from one command to another
>	redirect output to file

Command	Example	Intent
`bg`	`bg`	send a sleeping process to background
`cat`	`cat > f`	create file f and wait for keyboard (see ^d)
	`cat f`	stream contents of file f to STDOUT
	`cat a b > c`	concatenate files a and b into c
`cd`	`cd d`	change to relative directory d
	`cd ..`	go up one directory
	`cd /d`	change to absolute directory d
`chmod`	`chmod 644 f`	change file permissions
	`chmod u+x f`	change file permissions
`cp`	`cp f1 f2`	make a copy of file f1 called f2

`cut`	`cut -f 5 ff`	print column 5 from file ff
`date`	`date`	print the current date
`df`	`df -h .`	display free space on file system
`du`	`du -h ~`	display the sizes of your files
`fg`	`fg`	send sleeping process to foreground
`grep`	`grep p f`	print lines with the letter p in file f
`gzip`	`gzip f`	compress file f
`gunzip`	`gunzip f.gz`	uncompress file f.gz
`head`	`head f`	display the first 10 lines of file f
	`head -2 f`	display the first 2 lines of file f
`history`	`history`	display the recent commands you typed
`kill`	`kill 1023`	kill process with id 1023
`less`	`less f`	page through a file
`ln`	`ln -s f1 f2`	make f2 an alias of f1
`ls`	`ls`	list current directory
	`ls -l`	list with file details
	`ls -la`	also show invisible files
	`ls -lta`	sort by time instead of name
	`ls -ltaF`	also show file type symbols
`man`	`man ls`	read the manual page on ls command
`mkdir`	`mkdir d`	make a directory named d
`more`	`more f`	page through file f (see less)
`mv`	`mv foo bar`	rename file foo as bar
	`mv foo ..`	move file foo to parent directory
`nano`	`nano`	use the nano text file editor
`passwd`	`passwd`	change your password

`pwd`	`pwd`	print working directory
`ps`	`ps`	show current processes
	`ps -u ian`	show processes user ian is running
`rm`	`rm f1 f2`	remove files f1 and f2
	`rm -r d`	remove directory d and all files beneath
	`rm -rf /`	destroy your computer
`rmdir`	`rmdir d`	remove directory d
`sort`	`sort f`	sort file f alphabetically by first column
	`sort -n f`	sort file f numerically by first column
	`sort -k 2 f`	sort file f alphabetically by column 2
`ssh`	`ssh hal`	remotely log into machine hal
`tail`	`tail f`	display the last 10 lines of file f
	`tail -f f`	as above and keep displaying if file is open
`tar`	`tar -cf ...`	create a compressed tar-ball (-z to compress)
	`tar -xf ...`	decompress a tar-ball (-z if compressed)
`time`	`time ...`	determine how much time a process takes
`top`	`top`	display processes running on your system
`touch`	`touch f`	update file f modification time (create if needed)
`wc`	`wc f`	count the lines, words, and characters in file f