Unix Guide

Preface: For the purposes of this document, it will be handy to have examples of nested directories and files to use as examples. We will such define them here, with a schematic, so you can follow along as you read the examples.

We will be navigating around this particular tree

Home/your_user_name/documents/homework/week1

Note: In this tutorial “>>” will denote your prompt in terminal.

<tab> . . . .auto complete filename or command. (only works on unique files, etc, so if you have 3 files starting with abc you need to at least type one unique letter beyond that to let tab complete it).

& . . . Use this after a command if you want it to open in background; i.e., open a new window and allow you to still use terminal at the same time. For example, typing the command >>>emacs file.py will open emacs, but as a foreground process, until you close emacs, you will not be able to use the terminal you opened it from.

pwd . . . print working directory. This command allows you to see exactly which folder you are in. It is the equivalent to looking at the top of a windows or mac browser window and seeing you are in /user/documents/homework/etc. It will tell you the full path name of your current location.

cd . . . change directory. This command is how you navigate through the UNIX system of directories (folders). There are various nuances to how this command is used.

1. Simply typing cd with a space after it will change your directory to whatever your home directory is on a given network. For you it is likely /home/username/
2. cd will always work with a full path destination. I.e., regardless of where you are in a tree of directories, typing cd /home/user/folder1/folder2/folder3 will take you to the given folder.
3. If you are in a given folder, and want to change directories into a folder within that directory (for example, you are in your documents folder and want to move into a contained folder called homework) then you simply need to type >>cd homework (no slash symbols necessary). This syntax indicates to the computer that the directory you are attempting to cd into is within the

current directory you are in. (again, to check which directory that is, use pwd).

4. If you are in a directory and want to cd to the directory outside of it (for example, you are in homework, and want to get out to documents), you can type[>>cd ..] This notation tells the computer you want to cd to the folder that contains your current one. The reason is that a single period [.] has the meaning “here”. So if you typed >>cd . you would not change directories (or you would change to be exactly where you were). As stated, cd .. takes you one directory out. To move two directories out, use >>cd ../.. and to move three out, use cd ../../.. , and so on.
5. You can also move down multiple directories. For example, if you were in “home”, you could move to homework by typing >>cd documents/homework. Note the lack of slash before documents, because documents is within /home/. Typing cd /home/documents/homework would achieve the same effect.

---

clear . . . clear screen. Not super useful, but can be when trying to explain things to someone and wanting a blank canvas

---

mv . . . move. Moves a file. Syntax:

>>mv filename newlocation

for example, to move a file from homework to documents, type

>>mv filename ..

(remembering the syntax for one folder out). Again, it is always correct to use the full path if needed; ie,

>>mv filename /home/documents

You can also rename a file in the process of moving it. The syntax is

>> mv oldfilename newfilename newlocation

Interestingly, this makes mv the command for renaming files as well. To rename a file, “move it,” (giving it a new name) to the same location (by using a period as the destination)

The syntax for mv above assumes you are in the directory of the file to be moved. That form of syntax, with the filename following the command, will ONLY work in this case. If you were in /home/ and wanted to move a file in homework to documents, you would need to type

>>mv /home/user/documents/homework/filename /home/user/documents

(usually it is just easier to cd into the directory with the file, then move it).

---

rm . . . remove. Removes a file. BE CAREFUL. This is not windows or mac, there is no recycle bin, when something is deleted, it is gone forever. Luckily, the astro server computers all have a setting enabled that will prompt you "are you sure? Y/N" before actually deleting anything. (Which for now you should leave on). In any case, the syntax is just

>>rm filename

(assuming you are in the directory with the file, see discussion in mv about this). You can remove multiple files using the * wildcard, for example,

>>rm *.py

would delete all your python files (probably not a good idea by the way),

>>rm *

would just wipe out everything, and

>>rm *end*

would wipe out any file that had "end" anywhere in the name [see discussion of wild card right below for more info].

---

rmdir . . . remove directory. Same as above, but for directories.

---

cp . . . copy. This is how you copy and paste a file. The syntax is

>>cp filename newlocation

(of course assuming you are in the directory of the file). You also have the option of renaming it along the way:

>>cp oldfilename newfilename newlocation

(you can also copy a file to the same folder you are in, with a new name, by specifying the new location as ".")

Asterisk* . . . the wildcard. This one can be a bit tricky to understand conceptually at first, but it becomes very useful in day to day work. The * is basically a holder that can stand for any character, and any number of characters. It just means “anything”. So if you wanted to delete every file in a folder that started with a capital E, you would type

```
>>rm E*
```

and it would delete all files that had E[anything], regardless of filetype, etc. To delete all files of a certain filetype, use

```
>>rm *.py
```

and it will remove any file that ends in .py. (You can use any command with wild cards though, like mv, cp, etc, I simply use rm to be consistent). You can use multiple wild cards in a statement, for example, *fgw* would equate to any file which had the string fgw anywhere in its name, including at the very beginning (the * can also stand for nothing).

---

grep . . . “search”. Grep will search a file for a certain phrase or word. The syntax is >>grep “phrase” filename. There is a useful flag (or option) for grep so it will also display the line the phrase was found on.

```
>>grep –n “phrase” .
```

to search say, all your python files in a folder, you can

```
>>grep –n “phrase” *.py
```

(which will search all .py files in the directory)

---

ls . . . list files. This is a very important command which will allow you to see all files in a given directory. Generally it is used without arguments (the things that come after the command), for example, if you were in your homework folder and typed

```
`    >>ls
```

it would show you all files in that folder, similar to the way you would see them in a gui interface. However, as always, full path names work, so if you were in /home/, you could type

```
>>ls /home/user/documents/homework
```

and get the same results. There are many useful flags for ls.

>>ls -a will list all files in a directory, including "hidden" ones that don't normally show up with ls (these are never normal files, they usually begin with a period like ".alias" and are set up and permissions files you normally don't have to worry about).

>>ls -ltc will sort by most recently edited/modified.

>>ls -s will include file sizes, and

>>ls -Ss will sort by size

>>ls -r will reverse the sort

>>ls -X will sort by extensions

>>man<command> . . . Manual. Every UNIX command has a manual description of how to use it, and which flags and options can be applied to it. So anytime you remember a command (say, cp) but don't remember how to structure the arguments, or a specific flag for an option, you can type

>>man(cp)

and it will come up. You can scroll down through the description to find the flag you are looking for. To escape the manual, just hit "q".

---

mkdir . . . make directory. This command will make a new directory in the directory you are in. For example, if you are in homework, typing

>>mkdir week1

will make a new directory within homework called week1.

---

ssh . . . secure shell. This is how you login to astro computers from elsewhere, or switch which astro computer you are on. In terminal, say you are on aquarius.

>>ssh nemesis

Will prompt for a password, then put you onto nemesis (which for example has ipython while aquarius does not). Keep in mind that these are "networked" or server-based computers. I.e., your home/user/ folder, and everything within (along with anyone else's for that matter), are accessible from any computer in the network. The reason to switch is mainly related to which packages are installed on which, and whether some are running

slow due to high traffic. To learn more about sshing into the servers from your own computers, check out the Link in the resources page.

lpr . . . print. That's basically it. To print a file like pdf from terminal type

```
>>lpr filename
```

Assuming you are in the right directory, and this should print to the astro lab printer.

cat . . . catalogue. Syntax is

```
>>cat filename
```

And the output is allowing you to see the contents of a file.

Similarly, typing

```
>>more filename
```

Will pop open the beginning of a file, and show you what percentage of it is on screen. Hit enter to advance through the file until it ends, and you will automatically be returned to terminal.

---

tar . . . archive. The tar command is used to create .tar files, which are compressed file storage units similar to .zip, or .rar as you would have on a pc or mac (and programs like winrar can open tar files). This is a convenient way of packaging up multiple files to scp or email over to another location. Syntax:

```
>>tar –cf name.tar file1 file2 file3
```

Will create a .tar archive named "name" containing those three files. You can also use * to specify all files in a given directory, etc. to extract the files out of a tar, use

```
>> tar –xf name.tar
```

And the archive will unpack and the individual files/directories within will be accessible.

ALIASING

One thing you will learn extremely quickly as a programmer is that typing things sucks. That's why we have tab completion. That's also why there is a trick called aliasing that will allow

you to customize the commands you use commonly in order to allow you to move around with ease.

To start, cd to your user directory, /home/user/. Inside, if you were to run “>>ls –a” you would see a file called .aliasfile

You can open this file any way you like (vim, emacs, etc), and you will see that there are actually a crap ton of aliases already there, that are given by default. Some of these may be useful (for example, this is where they have set “rm” to actually mean “rm –i”, which means interrogate before deleting. However, I wouldn’t recommend taking the time to memorize all the aliases, since that defeats the purpose of having them! You can leave them alone or delete the ones you really don’t want (or want to change).

Anyways, now is your opportunity to set some aliases of your own. The syntax is

alias yourcommand “what the full real command should be”

For example, if you were too lazy to write ls –a everytime you wanted to list all files, and wanted to just type “la” instead, you would write

alias la “ls –a”

One of the most useful ways of using aliases are for cd’s. For example, you could do:

alias hmrk “cd /home/user/documents/homework”

alias docs “cd /home/user/documents”

and then, no matter where you were in any directory tree, typing

>>hmwrk

Would immediately whiz you to your homework folder. Keep in mind though that the alias is simply a shortcut for the long command you give it. So you if you delete directories, or move directories around, then the alias needs to be updated to still work. Usually I don’t recommend aliasing every folder then, but rather only doing the main ones you may use very often.