UNC P&A Linux Tutorial

Sheila Kannappan

Actually do everything in *blue italics*!

Log in using your onyen and password. Once in, right click on the background. You'll see various options, such as leaving (never shut down, just log out) or locking the screen. *Try locking and unlocking the screen.*

For most commands under linux, we want to use the "command line" in a terminal window rather than menus. Unfortunately, by default the terminal window is not easy to find, so you have to use the search feature in the start menu. *Search for "xterm" and start an xterm.*

Type *pwd* ("print working directory") on the command line. The information shown is the "path" for the home directory you start in by default. It is not local to the machine you are on! The UNC Physics & Astronomy Dept. uses the "afs file system," which shares files/folders between many computers over the network. You automatically land in your personal home directory when you first log in to any machine.

PROS of afs	CONS of afs
files look the same from any computer	file input/output (I/O) is over the ethernet (slow)
easy to share/move files	credentials expire – can't stay logged in indefinitely
well backed up	if afs server goes down, you can't work anywhere

Type *df* ("disk free") to see a summary of file system top level directories. Repeat with *df –h* (where -*h* means "human readable") for a more intuitive listing of the information. [Aside: flags (aka "options") like –h are summarized in "man" (manual) pages for every linux built-in program/command. Type *man df* to see a sample, and hit the *space bar to page through it*, or *q* (for "quit") to get out.] The output of *df -h* shows how the locally mounted hard disk space for the workstation you happen to be sitting at is divided into partitions. For example /dev/sda1 (a.k.a. /boot) is a small partition that holds the system software needed to boot up the machine. You should also see a partition called /afs, which is *not* locally mounted but goes across all machines. Except for server-class machines, you won't normally put files in local disk space (as this space is not backed up and is inaccessible from other machines), but will instead use afs space. To determine the afs space available to you in your home directory, type *fs listquota.*

Now type *ls* to "list" all files and subdirectories in your home directory – they're distinguished by the fact that directories end in a "/". You'll see "public" and "private" folders – if you create/copy a file in these directories, the file will automatically be readable or inaccessible, respectively, by other users. Notice that although afs is available when you log onto a Windows machine, Windows can't necessarily read all the files. Linux machines can read Word/Excel/Powerpoint documents using *soffice* (also called *ooffice* or *libreoffice*).

Use *cd public* to "change directory" to the "public" subdirectory. Type *ls* to see what's in it. The simplest way to make a file is *touch filename* – you can substitute anything for the filename, e.g., *touch foo.txt* , but you should include the ".txt" extension as by convention linux users always include a file extension. The "touch" command really changes the last-modified date on a file to be the current time, but if a file doesn't exist, then it has to create it (as an empty file).

Type *date* and then *ls –l* ("list long-format") to get the current time and see the file timestamp. *Use the up arrow key* to go back to previous commands and *repeat touch, date, and ls –l* until you manage to get the current time and file timestamp to match.

You can also make a subdirectory – type *mkdir test* and then use *cd, pwd,* and *ls* to go inside "test" and verify that it is empty.

Now *cp* ("copy") a file from your partner's public directory to your own. The syntax is *cp sourcepath/filename destinationpath/filename* where the only difference between the sourcepath and destination path is in the last part (…/m/a/maria). Since paths are long (/afs/cas.unc.edu/….) you may not want to type all that. Here are some shortcuts you'll find useful:

1) Mouse cut and paste – *left button selects/copies* and *middle button pastes.*

   Experiment with using "pwd" and mouse cut and paste to copy your partner's file.

2) Tab auto-completion – if you type *cd /afs/ca* and then *hit the tab key*, linux shows all possible completions, but if you get as far as *cd /afs/cas.un* then the completion is unique, and hitting the tab key gives *cd /afs/cas.unc.edu/.* *Hit tab again* and you'll see all the next-level subdirectories – start typing "u", hit tab again, etc. You can keep going until you get the whole path.

   Experiment with tab auto-completion to copy your partner's file. Note that <u>cp does not protect you from overwriting a file</u> when you copy to the same filename over again.

3) The destinationpath can be omitted if you want to put the file in the directory you're already in. Even better, the filename can be given as "." if you want to keep it the same.

   Try *cp sourcepath/filename newfilename* and *cp sourcepath/filename .* then compare the results with ls. You can also compare two files with the "difference" command *diff newfilename filename*. Note diff returns nothing if there's no difference. Since these two files' contents do not differ, it makes sense to *delete one using rm* ("remove").

4) The ".." syntax which indicates a relative path starting one level up from the current directory. In your public directory, type type *cp filename ../private/filename2*. One way to compare the files is to type *ls –l filename* and *ls –l ../private/filename2*. Another way is to cd between the directories, e.g., type *ls –l filename* then *cd ../private* then *ls –l filename2.*

5) Some other shortcuts to move around are *cd –* ("cd to last directory") and *cd* alone with no argument ("jump back to home directory").

Now try to copy your partner's filename2 from their private directory to your own space. Of course, you can't do it, because you don't have the right permissions. (Aside: In case you have heard of “chmod” bits you may be confused by the fact that the bits you see in the *ls –l* output don’t reflect the actual permissions – here afs controls are overriding the chmod bits.[1])

However, one can always copy a protected file with the owner’s help, using “secure copy”: *scp maria@machinename.astro.unc.edu:/afs/cas.unc.edu/users/m/a/maria/private/filename2 filename3*, where filename3 could be just a dot “.” or could have a path in front. Since afs space is visible to all physics machines, the “machinename.astro” could have been any physics or Astro machine. Now the owner (“maria”– substitute your partner’s onyen) will have to type their password when prompted.

If you want to copy everything from your partner’s public directory without knowing the filenames, you can type *cp –r sourcepath .* with no trailing slash (but with a dot at the end). Here the –r “recursive” flag is necessary because sourcepath specifies a directory not a file. The exact same *–r* syntax works for *scp* as well.

An alternative to copy is *mv* (“move”), which does not create a new file, even if you change the name. In fact *mv* can be used to rename files without actually moving them! To see this, use *ls –l* to compare before and after typing *mv filename filename3*.

Okay, we’ve copied and moved a lot of files around without putting anything inside one! A somewhat silly way to put text in a file is using “echo”: *echo “hello everyone!” > anotherfilename* . The “>” is called a “redirect” and it sends the output of echo to the file. If you just type *echo “hello everyone!”* by itself you’ll see it normally sends output to the terminal.

Type *more anotherfilename* to see the contents of your file. To add a whole bunch more text, *copy the file /afs/cas.unc.edu/users/s/h/sheila/public/reu/studenttravel.txt into your directory*, then type *cat studenttravel.txt >> anotherfilename* to add the contents of this new file to your file (note that >> appends whereas > overwrites). You can use “cat” (short for catalogue) or “more” to view anotherfilename.txt at this point, but “more” is better because it gives you a page’s worth at a time (use the spacebar to page through and type “q” to get out). Actually many people consider *less* an even better version of *more*. ☺ Try it!

If we want to pick out just a short bit of a long file, we can use another type of redirect that goes not from command to file, but from command to command. The pipe symbol | takes the output of one command to be the input of the next. For example if you want to find the line of the file containing your name, type *cat anotherfilename | grep “yourname”*. Alternatively, maybe you want everyone except yourself: *cat anotherfilename.txt | grep -v “yourname”* where the –v indicates exclusion. The “grep” command can also work directly on the command line, for example *grep “yourname” *.txt* would pull out everything with your name in it in every .txt file in the whole directory – the * is what’s known as a “wild card”. Other useful commands on files are *wc* (“wordcount”) and *sort* (which can sort either by numbers or letters, see the man page).

[1] If you would like a custom permissions arrangement, such as a directory structure viewable by a certain group of people, you may wish to learn more about how afs space is controlled by the *pts, aklog, fs,* and *fsr* commands; see the the man pages. For questions, visit https://help.unc.edu/ and submit a ticket with a note at the top “Please assign to Stephen Joyce/OASIS-Linux”.

But enough playing around, what if you really want to do something? First of all, you're likely to want to start programs that run in their own separate windows. When you type a command to start one of these, say *firefox,* notice that it opens firefox, but the original window you were using can't take input anymore. We need to use the character "&" to run firefox "in the background," meaning it won't tie up the command line. Quit firefox by clicking on the X in the corner and try again, this time typing *firefox &* .Notice the difference! There are many other processes with separate windows you may wish to start this way – for example try *xterm &* or *konsole &* to get another terminal window.

Here are a few more linux-related things you will probably want to learn about in the near future:

(1) What if you want to write a program? It's time to *learn emacs or vi*, the two most powerful linux text editors, or perhaps *nano*, which is less powerful but a lot easier to use. All are ideal for programming in python because they show equal spaced characters and spaces (which have syntactical meaning for python). Complete the *built-in tutorial in the emacs help menu* or use the *nano instructions at the bottom of the nano window* or type *vimtutor* at the command line (or do the *shorter vi tutorial* at *https://github.com/galastrostats/general/blob/master/vi.md*). By default emacs opens in a separate window, so start it with "emacs &" or use it in no-window mode by typing "emacs –nw". By default vi and nano open in the terminal, although you can start them separately with "gvim &" or "nano &". Note that all three options can also be used under Mac and Windows and are more powerful than the Anaconda Spyder editor (the other common laptop option), albeit installing them under Windows can be tricky – the path of least resistance is to install Git Bash, which includes vi.

(2) You'll need to add the command `unset autologout` to your .mycshrc and .mylogin files in your home directory, or you'll be automatically logged out when idle, which can be intensely frustrating. Use vi/emacs/nano to edit these files (or if they don't exist, create them). Note these "dot files" (files that start with a dot) are invisible by default unless you say *ls .** and they are automatically sourced when you log in. You might also want to create your own *.alias* file, which allows you to create personal aliases for long commands you use a lot, common typos, etc.

(3) If you run a program that crashes, or you get autologged out, or your machine just seems weirdly slow, you may want to monitor currently running processes (as you would with the Windows Task Manager with Ctrl-Alt-Del). Three useful commands are:
`top` shows all processes using CPU time on a given machine
`ps aux | grep username` finds processes associated with "username"
`kill -9 #` kills your process with ID number #

(4) You may wish to log into another server for your most serious computing (with more processors and perhaps high-capacity local disks with fast I/O in addition to afs access). To use such a machine remotely, assuming it runs under linux, we use "secure shell" like so: *ssh –X youronyen@machinename.astro.unc.edu* (where astro might become "physics"). The capital X is necessary to enable X-forwarding (tunneling) from remote machines, which allows windows to pop up as if you were sitting in front of the machine. In fact, many servers are "headless" so all connections to them are via ssh.