Introduction to Linux

by Sheila Kannappan

(as implemented in the UNC Physics & Astronomy Dept., assuming primary usage via remote link from a Windows laptop with XWin32 + either PuTTY or Git Bash)

Mac users: This tutorial was written by a Windows user but the **Appendix** offers suggested Mac equivalents. Everything done with XWin32 & putty should be possible if you install and run XQuartz. Also if using Mac OS X, start the X11.app terminal instead of the Terminal.app that you may be used to – otherwise X forwarding may not work per the linux IT gurus.

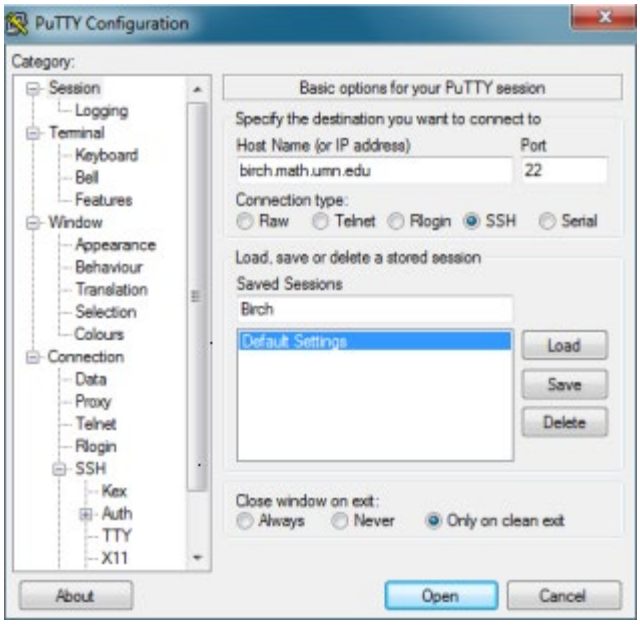
Actually do everything in *blue italics*!

Startup

Start XWin32 and let it run in the background. Follow the PuTTY or Git Bash instructions below.

PuTTY: *Start PuTTY and create a saved session* for a department linux machine as follows:

1. Add full hostname (e.g., machinename.astro.unc.edu)
2. Switch Protocol to SSH
3. Type name of session in saved sessions (e.g. blarney)
4. Click 'Save'



5. Expand the 'SSH' tab from the 'Category' list
6. Choose 'X11' from 'SSH' list
7. Check 'Enable X11 Forwarding'
8. Choose 'Session' from 'Category' list
9. Click 'Save'

10. Now click ‘Colours’ on the left, and under ‘Select a default colour to adjust’ choose ANSI Blue and then click ‘Modify’. Select a medium light blue/purple.
11. Go back to ‘Session’ and click ‘Save’ again.

Load the session and log in using your onyen and password [it will not echo]. The terminal window that comes up is an “xterm” and it accepts text commands at the prompt (“command line”). For most commands under linux, we want to use the command line in a terminal window rather than menus.

Git Bash: *Start Git Bash* and type the following commands:
export DISPLAY=localhost:0.0
ssh –X onyen@machinename.astro.unc.edu [enter your password; it will not echo]
(for example – any departmental linux server should work)

Main Tutorial

Before starting, if you don’t like the font size in your PuTTY/Git Bash terminal, start a larger remote terminal by typing *xterm -fa 'Monospace' -fs 12 &* (with any #). In a second or two it should appear – it may take a short while because it is generated by the remove server and “tunnels” back to your display. Now you can get started.

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 just for comparison. Repeat with *df –h* (where the flag means “human readable”) for a more intuitive listing of the file system usage information. You should see that the largest space available is not in /afs, but in /spare, which is a partition of a locally mounted hard disk. However, you probably won’t want to use this space, which is not backed up and is inaccessible from other machines.

Note that flags (aka “options”) like the –h above 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. If you ever really need to know how much space you’re using, try “disk usage” *du –h –s* and perhaps also *fs listquota* (specific to afs space).

Now type *ls* to “list” all files and subdirectories in your home directory – they’re distinguished by the fact that directories end in a “/”(and PuTTY/Git Bash will also color them). 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 very last part (e.g., /m/a/maria). Since the paths are really 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* in all three of PuTTY, Git Bash, and a real linux terminal, but pasting is different in each: *right button pastes in PuTTY,* while *shift-insert pastes in Git Bash* and *middle button pastes at a real linux terminal.* Right-clicking on a Git Bash window also brings up a copy-paste menu.

   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 cp does not protect you from overwriting a file 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 astro or physics 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

[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".

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).

Finally, it often happens that you want another window. If so, you can just type *xterm &* (or *xterm -fa 'Monospace' -fs 12 &* as mentioned previously) in the existing PuTTY/Git Bash terminal connected to the remote server. (Under PuTTY, this secondary xterm may not allow right-button pasting the way the PuTTY one does, so another option is just to start a second PuTTY session with the remote server.) The character "&" runs the xterm "in the background," meaning it won't tie up the command line. To see the difference, first type just "xterm" on the command line and note how you can't type anything else in your first xterm anymore. Now close the new xterm by clicking the X in the corner, and try again, this time typing "xterm &". There are many other processes with separate windows you may wish to start this way, including the souped-up "konsole" alternative to "xterm". The purpose of XWin32 is to allow such windows to "tunnel" through to your screen remotely. Be forewarned that graphics intensive processes will be slow – you probably don't want to type *firefox &* on the remote server.

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. Both are especially 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 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 (which is faster when using it from a remote server) by typing "emacs –nw". By default vi opens in the terminal, although you can start it in a separate window with "gvim &". Note that vi and emacs can also be installed/used directly on your laptop, and are more powerful than the Anaconda Spyder editor (the other common laptop option), but 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 or emacs to edit these files (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) As you already saw if you're using Git Bash, you can log into remote servers from a linux command line using "secure shell": *ssh –X youronyen@machinename.astro.unc.edu* (where astro might become "physics"). The capital X is necessary to enable X-forwarding (tunneling) from the remote machine, 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.

Appendix: Notes for Mac Users

(from Ryan Beauchemin and Maria Arias de Saavedra Benitez)

getting set up

From the Mac's built in linux terminal (which might be in the application window, or you can just pull up by typing terminal in the spotlight when you click the magnifying glass in the top right corner of the screen), you can start with your first command being

ssh -X youronyen@machinename.astro.unc.edu (the -X enables X11 forwarding)

Typing the line above opens X11 after you type your onyen password (note that the cursor will not move while you type the password, nor will anything appear on the screen. Also, bear in mind that if you mistype your password three times you will not be allowed access, and will need to check with IT about resetting). After doing this you can run programs as in the tutorial.

Another way of doing this is typing "xterm" into Terminal when you first open it, which opens X11, and an X11-terminal window. Typing ssh -X youronyen@machinename.astro.unc.edu here does the same thing, just in a different interface. However, the Mac xterm doesn't allow copying and pasting, whereas the regular Mac terminal allows you to do so using command+C and command+V.

emacs

Since Mac has emacs you can do the tutorial from your own computer instead of ssh-ing.

Also, by default if you open text files in emacs remotely to edit them it opens an emacs native to the computer being ssh'd to. The remote display can be pretty slow. One way to bypass this and open files on your local emacs is to go to a separate Mac terminal window and type:

emacs /onyen@machinename.astro.unc.edu:/path/file.txt

emacs will ask you for your password, to that site, so just enter your onyen pw and ta-da! Your own emacs likely doesn't take 5 minutes to start up every time, like the remote one does.