Assorted additional topics in Linux computing

Khaldoun Makhoul
khaldoun@nmr.mgh.harvard.edu

October 6th 2011
Outline

- Remote access
- The shell environment
- File permissions
- Using the Martinos computing cluster
Remote access

- Whether you use your laptop as your main computer or just when you’re away from your desk, it’s convenient to be able to do remote computing

- Depending on your needs, network connection, and preferences, you will have the following main options:
  - **ssh**: secure shell – the most basic, but necessary
  - **VNC**: Virtual Network Computing – connect to a graphical desktop on another computer
  - **screen**: lets you keep shell sessions going across network connection lapses (and does much more!)
  - **NX**: great optimization of X (e.g. GUI) display over a network connection [not covered in the talk for time considerations]
Remote access: preliminaries

- You will need a terminal application on your own computer to access remote computers.
- On Windows, you can use PUTTY (ssh & a terminal emulator) for basic needs. You’ll be limited to text-only (can’t open a separate window UI).

![Putty Configuration](en.wikipedia.org/wiki/PuTTY)

![SSH Terminal](en.wikipedia.org/wiki/PuTTY)
Remote access: preliminaries

- On Mac OS, you can use the built-in Terminal application (search in Spotlight if you don’t know where to find it).

![Image of Terminal application on Mac OS](en.wikipedia.org/wiki/Apple_Terminal)
Remote access: \texttt{ssh}

- Great when all you need is to log in, run a few commands, maybe a few text-only tasks.
- Even when you have the ability to open GUI programs, doing this over an ssh connection is likely to be painfully slow if you’re away from the lab (WiFi Aaaah!).
- \textbf{If you lose your connection}, you lose all unsaved work, all interactively running jobs are interrupted, terminal & GUI windows tend to hang, and \textit{general misery ensues}. 
Remote access: ssh

The -Y option will be needed for opening “windowed” programs later

Syntax: username@computer

```
khaldoun@dhcp-203-229:$ ssh -Y khaldoun@gate.nmr.mgh.harvard.edu
khaldoun@gate.nmr.mgh.harvard.edu's password:
```
Remote access: `ssh`

I open some program on gate (here, `gedit`). Note the use of `&` to run in background and get my command line prompt back.

You can see this is running on gate as opposed to my laptop, from gedit’s file open dialog showing my home directory on nmr
Now let's simulate what would happen if I started to do some work and lost my network connection. First I'm going to type some random stuff in gedit.
Remote access: ssh

Next I’m going to kill my ssh connection. You would not have any reason to do this, but this will simulate losing your connection (closing your laptop, losing WiFi, etc)
Remote access: ssh

As soon as I press ENTER on the kill command, the gedit window disappears.
Remote access: ssh

Next time I log into gate, I run ps to see if gedit is running... Nope :(. All that work is now lost.
Remote access: VNC

• VNC allows you to open up a login session that is essentially identical to a login session you would experience if you sat down at a particular computer.

• Only images of your remote session are being sent. The session itself resides on the remote computer, which means that...

• ...if you lose your network connection, no problem! Just reconnect to the VNC session.

• VNC is also platform-independent. You can use VNC from and to any given Operating System.
Remote access: VNC

Martinos computer

• Start a VNC server. This is the session running on the remote computer (i.e. NOT your laptop) that will be sitting there waiting for you to connect to it. (As long as you don’t kill the session, and as long as the computer isn’t shutdown or restarted, you will not need to repeat this part.)

Home computer

• Start a ssh tunnel to the correct VNC server.
• Start a VNC client*.

*Martinos-recommended clients: UltraVNC for Windows, Chicken of the VNC for Mac

https://www.martinos.org/martinos/userInfo/computer/remoteAccess.php#vnc
Remote access: starting a VNC server

Martinos computer

- Start a VNC server.

Start a server on a computer other than gate/entry/launchpad etc. This should be YOUR OWN Martinos computer

Create a directory where VNC will store needed files

Start the server with this command (you don’t have to specify geometry, but I like to control the size of the desktop that will be started)

Enter a VNC password so that no one else can connect to your VNC session

Note this desktop number!
Remote access: starting a VNC server

Martinos computer

- Start a VNC server.

You can now close this connection, and even close the Terminal window if you like. This part is now done :)
Remote access: ssh tunnel for VNC

Home computer

• Start a ssh tunnel to the correct VNC server.

The command here includes:

- the name of the computer where you started the VNC server: ptx2
- the desktop number you got from the VNC server command output: the 1 in 5901

You’ll have to keep this connection open while you use VNC!
Remote access: start the VNC client

Home computer

- Start a VNC Client

Host is always localhost

The Display is the desktop number you got from the output of the VNC server command

The password is what you used as a VNC password
Remote access: start the VNC client

The VNC client is now showing me my desktop on ptx2, and I'll be able to use it as if I was sitting at the computer there.
Remote access: losing connection

I’m now going to do the same demonstration: opening gedit, typing a few things in there, and then killing the ssh connection to simulate losing network access.
Remote access: losing connection

Killing ssh...
Remote access: losing connection

My VNC client complains, rightly so!
Remote access: reconnecting

I restart the ssh tunnel...

I restart the VNC client....

And my session is just as I left it! No work lost :)

This machine, entry, is NOT for data analysis. It is to be used as a gateway to other NMR Center systems or for accessing e-mail. Data analysis programs found running on this system will be terminated. Thank you for your cooperation.
Remote access: `screen`

- `screen` lets you do something similar to VNC, but with just a shell environment. This is awesome.
- You can now use `ssh` to log into your machine, start a `screen` session (or many if you like), and disconnect from/reattach to those sessions remotely without interrupting your work.
- As long as the remote computer (i.e. the Martinos computer) isn’t shut down or rebooted, you’ll keep your `screen` sessions.
Remote access: screen

Let's start with a standard log in using ssh (I log into ptx2 in this example, as often) Once logged in, type “screen” and press ENTER. You should now see something like the example below (I use bash as my default shell; you probably don’t. No worries...)
Remote access: `screen`

Next, I just run an endless loop so that we can get some output in the Terminal. This will run until I stop it with Control-C. Now let’s see what happens if I lose my network connection.
Remote access: screen

My standard method of simulating losing network access: I kill the ssh connection. Note that the upper Terminal now shows me logged in to my laptop (dhcp blah blah), and I have lost my connection to ptx2.
Remote access: `screen`

But let’s see if we can reconnect! (Spoiler: we can.) First I log back into ptx2.

The `screen` command with the option `-ls` will list any current `screen` sessions, including “detached” sessions, such as the one we left behind when we killed our connection.

The `screen` command with the `-r` option will then reattach the specified session (you can get the name from the listing).

When I press ENTER, I’m greeted with the same output that is *still* going :). No work lost!
The shell environment

• Whenever you type anything into a command line at a terminal, you are “inside” a shell. The shell interpreter is the program that takes the sequence of characters you give it and interprets it as meaningful commands.

• You can change shells if you prefer another one (csh is the default at Martinos, bash is a common alternative).

• Even if you never need to use another shell, you need to know how the one you’re using is set-up
The shell environment: startup files

(For C shell)

• `~/.cshrc`: these commands are executed every time you start a shell (new terminal window, running a script, etc).

• `~/.login`: these commands are only executed if the shell in question is being run as you log in (say, if you `ssh` into a computer).

• Putting various configuration commands in these files is very useful, but use caution: if you’re running jobs on the computing cluster, any problems caused by these files can be hard to spot.

• If you’re making changes, remember to check the default startup files that are read before your own: `/etc/csh.cshrc` and `/etc/csh.login`
Environment variables are string (i.e. text) variables defined in your shell session. They’re always referenced with the usage `$VARIABLENAME`.

Several of these are defined by default (`$HOME` for example), and several are needed to run certain software or some scripts (freesurfer for example), but you can also create your own if you like.

At any time, you can look at all the currently defined environment variables, set new ones, clear previous variables, etc (`printenv`, `setenv`, `unsetenv`, etc).
File permissions

• File permissions in most Linux distributions consist of **read**, **write**, and **execute** permissions, and can be assigned separately to the **user**, **group** and **other**

• You can check the permissions on a given file by doing `ls -l`

• You can change the permissions by using `chmod`. Also useful to know: `chown` and `chgrp`. 
File permissions

Here’s a sample listing of directories and files in long format (ls -l). Let’s break down the permissions columns (first 10 characters).
File permissions

- The first character, d, simply means that it is listing a directory
- The next three characters refer to the user permissions: rwx means I have read, write and execute permissions (For a directory, execute permission just means that I can list the contents of the directory)
- The next three characters refer to the group permissions: rwx again means that anyone in the group specified has read, write and execute permissions
- The last three characters refer to the permissions for all other users: r-x means that they have read and execute permissions but cannot write (i.e. cannot change the name or directory or delete it)

- The permissions for files are very similar in this case, with the exception of the execute permissions. Since these are not executable files (e.g. compiled programs or shell scripts), the execute permissions are set to off for user, group and other
• Sharing files with other users here is best done if all the users involved are in the same “group”.
• You can ask the help desk to create a group for you if you have a specific need, and you can be part of more than one group.
• You can also make an entire directory and files created within that directory automatically group-shared... but you’ll need to do some regular maintenance.
• Let’s demonstrate.
File permissions: sharing

Using the `groups` command, I see that Ellen and I are both members of the `whynhow` group, which I will now use to share files with her.

We can change the group that these files belong to using the `chgrp` command.

(Find out the precise syntax using `man chgrp`) ^_^

Just for the sake of demonstration, I use the `chmod` command to change permissions on a file. Here, I remove the group write permission from the file `/log`.
By default, files are created as members of the user group (named after the user, and only containing the user as a member).

This command (Martinos only) will make the directory I specify as the last argument a group directory for the whynhow group: all files are now in the whynhow group, and all have group read/write permission.

Furthermore, files created inside that folder will be in the whynhow group as default.... but be wary of commands like cp and mv, which will not respect this convention.
File permissions: sharing

Files created inside that folder will be in the whynhow group as default.... but be wary of commands like `cp` and `mv`, which will not respect this convention.
Using the Martinos computing cluster

• The first step to sending a job to the computing cluster queue is to make sure that what you’re sending actually works in your local environment, and to do as much debugging as possible there.

• Once that’s done, sending a job to the queue is quite straightforward... if all goes right.

• But we also have several tools at our disposal to debug problems

• Let’s walk through an example together.
I’ve created a simple shell script and made it executable with:

```
chmod u+x ./quickscript.csh
```

The script does the following:
- print the current working directory
- echo an empty string into a separate log file
- iterate 3 times, each time echoing the iteration number and then the date/time into the log file
- wait 10 seconds between each iteration

```
#!/bin/csh

pwd
echo "" > mylogfile.txt

foreach i (1 2 3)
  echo $i | tee -a mylogfile.txt
date | tee -a mylogfile.txt

sleep 10
end
```

---

Using the Martinos computing cluster
Using the Martinos computing cluster

Now I need to log into launchpad, and run the pbsubmit command.

I specify the command to run with the -c option, and give the full path to the script.

You can see the output of your job in the /pbs/username folder:
- a file with the command line output (*.oJOBID)
- a file with the error stream output (*.eJOBID)
- a file with the environment variables that were set during the job (*.env)
- a status file that tells you where and when the job ran, and the command submitted, etc.
If you have questions on more advanced stuff, shoot me an email!

khaladoun@nmr.mgh.harvard.edu