CS107 Unix Session Walkthrough

(based off of Sumi's notes, rewritten into a walkthrough format by Mindy, revised and expanded by Schuyler, edited by Travis)

Link to this page: http://bit.ly/cs107_unix_walkthrough

Getting started with Unix

Everything in this lab involves running commands on Unix. Developed in the '70s at Bell Labs, Unix is an operating system, like Windows or Android. Over the years Unix and its derivatives have become ubiquitous. In fact, OS X is Unix-certified! Linux and Unix are "POSIX-compliant," meaning they share a set of similar commands and behaviors. Technically, Linux is not official Unix, but merely "Unix-like". However, for our purposes, Linux and Unix are basically synonymous.

In CS107 we'll be working on **myth**, a cluster of computers in the basement of Gates running Ubuntu Linux. The myth computers are desktops located in Gates B08, and you can use them remotely or in-person. You'll mostly use them remotely to do assignments, because the room is not always available to everyone (for example, during scheduled labs).

Let's get started!

- ➔ Sign into a Myth computer in person - Your SUNet ID is your username, and your password is your SUNet password.
  - To sign in remotely, use SSH. See the section below for details.
- ➔ The computer is running Linux, which is an operating system like Windows or Mac OS X
- ➔ All the files you see on this desktop are stored on AFS, Stanford's distributed file system.
  - Hence, your files will be the same no matter which myth machine you log into.
- ➔ Every student has server space on AFS, and your AFS desktop persists across most Stanford computers. It lasts until your SUNet ID is deactivated.
- ➔ There are many ways to access your files on AFS. You can:
  - Use a Myth computer
  - Log in remotely with **ssh** or **sftp**
  - Use WebAFS, an online AFS browser.
- ➔ See 'Other resources' on the CS107 webpage for helpful Unix, text editor resources

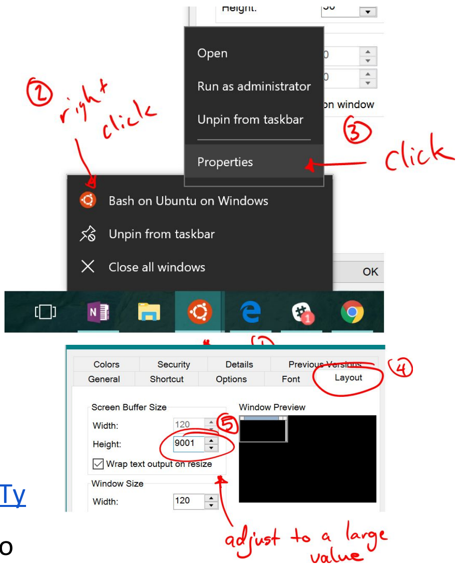
SSH

➔ You can sign on to Myth computers and do your assignments remotely using **ssh**, which stands for "secure shell."

- ssh shows you a terminal running on another computer. Anything you type will be received by the other computer, and anything you see is the output of the other computer.
- *If you use Mac OSX*, you can ssh through your terminal, which is called "Terminal.app" (find it in Applications or via Spotlight search)
    - `Type the command "ssh <suid>@myth.stanford.edu", eg. "ssh mindyh@myth.stanford.edu"` and press return.
    - You can logout by typing "`exit`", or just by quitting the terminal
- *If you have Windows 10,* you should be able to use *Bash on Ubuntu on Windows*. After installing it, just use **ssh** like any Unix computer.

Adjusting the screen buffer size for the bash prompt (so that you can scroll back and see your command history):

Right click the Bash on Ubuntu on Windows logo,
Right click on the "Bash on Ubuntu on Windows text,
Click on properties
Click on the layout tab
Adjust Screen Buffer Height to a large value (eg 9000)
(you'll see results in any new windows)



- *Otherwise, if you use Windows older than Windows 10*, you must download a Unix-like terminal emulator. Popular options are PuTTy and MobaXterm. These programs often connect to SSH for you, so you won't have to type the above commands. Instead, you should set the "host" to myth.stanford.edu and use your SUNet ID credientials as the username and password.

Command-line interface (CLI)

The **Command-line interface**, a term given to text-based methods of interacting with software, is prevalent today particularly among developers, even with the rise of **GUIs (Graphical User Interfaces)** in most consumer applications. CLIs are especially useful for remotely accessing servers and embedded devices, where network limitations and lack of GUI software mean that there is no other way to interact with the computer. CLI commands also offer convenient scripting possibilities, useful for automating sequences of commands.

Let's meet the CLI

- ➔ Open the terminal on the Myth machine: click app in the top-left corner (or hit the Windows key) and search for "terminal".
    - ◆ If you're using SSH, you already have a terminal open
- ➔ type "`date`" and hit enter
    - ◆ this is an example of a **command** with no arguments
    - ◆ It gives the date on the myth machines, the timestamp by which deadlines are measured in CS107.
    - ◆ What goes on behind the scenes: the shell searches through the filesystem, looking for the program `'date'` located in some common or specific location, and runs it. That's why '`date`' can be run from any directory.
- ➔ Try the command "`echo Hello world`"
    - ◆ This is a command with arguments - "`echo`" is a simple C program, and "`Hello world`" is passed into echo's main() function as the arguments.
- ➔ You may also add options/flags in commands. Options are used for running the program in different ways:
    - ◆ "`echo -n Hello world`" echoes without the newline at the end.
    - ◆ You can see the list of all options using "`man echo`"
- ➔ If you ever get confused about the syntax of a command, use "`man`" (short for manual)
    - ◆ Run "`man ls`" to learn about "`ls`," the command to list files.
    - ◆ Man also works for C function, eg. "`man strlen`"
    - ◆ If there are both a C function and a command with the same name, you must indicate which section of the manual to search. Type the number as the first argument to "man," followed by the term to search for. For example, "`man 3 printf`" will find the Linux function "`printf`," while "`man printf`" will find the user command.
        - ● Unix commands, like "`ls`," are in section 1.
        - ● System calls, like "`sigaction`," are in section 2.

        - C functions, like “`printf`,” are in section 3.
- ➔ Try the command “`clear`”
    - Useful to clear a cluttered screen.
    - Also achieved by pressing CTRL-L.
- ➔ Unix file structure is very similar to what you’re used to in Mac or PC
- ➔ **Directory** is just another word for folder
- ➔ **/** is the **root directory.** It’s like C:/ in Windows.
- ➔ ~ is your **home directory.** It’s the folder that holds the Documents, Desktop, Downloads folders, just like in Windows/Mac.
- ➔ “`ls`” command (short for “list”, because the authors thought typing four letters was too much)
    - Without arguments, shows what you have in your current working directory
    - “`ls -a`” will also show hidden files and folders
    - “`ls -l`” will show more details like permissions, file sizes, and last modified dates.
- ➔ “`cd`” will change your **working directory**
    - “`cd /afs/ir/class/cs107`”
    - “`ls`” to see the files in the directory
- ➔ See the samples directory? `cd` into it. This is where Julie puts the code from lectures
    - `cd samples`
- ➔ You only need to say “`samples`” in the cd command above, instead of “`/afs/ir/class/cs107/samples`”, because the folder is one level down *relative* to the cs107 directory. This is called a **relative path.** “`/afs/ir/class/cs107/samples`” is the **absolute path**, unambiguously locating the directory no matter which directory you’re in.
- ➔ “`cd ~`”
    - to get back to our home directory.
- ➔ “`cd`” into any subdirectory you want
- ➔ “`cd ..`”
    - goes up to a **parent directory.** You should be back in ~ now
    - .. represents the parent directory like ~ represents the home directory
- ➔ “`cd .`”
    - stays in same dir
    - . represents the **current directory**
- ➔ “`mkdir myDir`”
    - to create a directory called “myDir” within the current directory.

- "`ls`" should show you that there is now a directory that wasn't there before.
- You should also be able to see this folder in the file explorer with the GUI when using Myth in-person.

➔ "`mv myDir myNewDir`"

- Moves myDir to myNewDir, which is equivalent to renaming it.
- You can move individual files into other directories. For example, to move the file "test" into "myDir":
  - "`mv test myDir`"
  - Directories are just files, so there's no special syntax for them in most cases.

➔ "`touch test`"

- Creates a new blank file called "`test`", or does nothing if "`test`" already exists.

➔ "`cp test test_copy`"

- copies the file "`test`" and names the copy "`test_copy`"
- Use "`cp -r`" to copy directories (the r is short for *recursive*, which also copies files, etc. inside the directory).

➔ "`rm test`"

- Deletes the file.

➔ "`rm -rf myNewDir`" **(be careful)**

- Deletes the entire directory
- As for cp, `-r` means *recursive*, which walks into directories. `-f` means *force*, which doesn't ask for confirmation for each file. Use both with caution! **There's no "undo" for rm, and the files are gone forever!**

➔ **Just to reiterate:** *There is no undo feature.* Sometimes you don't get warnings when you move or copy and files just get overwritten, so be careful!

➔ "`grep <pattern> <file(s) to search>`"

- Ninja powers: search for the given pattern in "files to search"
- The pattern can be as simple as a word, or can use regular expression notation
- Use the "`-r`" flag to make grep recursive and search through all subdirectories of the current directory
- Use the "`-i`" flag to ignore case (match upper- and lowercase versions of the pattern)
- Example: find the word "Cool" in the current directory or any subdirectory:
  - `grep -r Cool ./`
- Example: find lines beginning with "Verily" in files in the current directory

        - `grep “^Verily” ./`
            - The hat (^) indicates the beginning of the line
            - Without “`-r`”, we only look in the current directory (no subdirectories)
    - Example: find the word “hi” in any file with the extension “.txt” in the current directory
        - `grep hi *.txt`
            - The “`*.txt`” is a “blob” that the shell expands to any file with the “`.txt`” extension
    - More grep examples

- *some other UNIX tips:*
    - the **up arrow** goes through your old commands
    - **ctrl+c will force quit** any application that’s running**.**
    - !! (two exclamation points, or “bangs”) will repeat the previous command.
        - Example: You can use “`time`” to measure the execution time of a program. So you can use “`time !!`” to re-run the previous command and time it.
    - **“Tab” will autocomplete** the filename as you’re typing it
        - ex. in your home directory, “`cd Doc`” and then pressing tab will complete it to “`cd Documents`”
        - If multiple files match the prefix you’ve typed, autocompletion stops as soon as there’s an ambiguity. Press tab again to see a list of completion options.
    - you can **copy-paste** with **Ctrl+Shift+C, Ctrl+Shift+V**
        - *On Mac*: In Terminal.app, use Cmd+C and Cmd+V (just as you would in other programs)
    - “`cat filename`” will print out the contents of the file
    - If you know what a command starts with, typing ! (bang) followed by the prefix and then TAB will complete the command. For example,
        - Run the command “ls -a”
        - Type “!l” (exclamation followed by the lowercase L) and then TAB
        - The shell will complete to “ls,” the most recent command with the given prefix. Useful for recalling long commands.
    - To learn more, go to the CS107 website!

Choosing a Text Editor

- ➔ https://xkcd.com/378/
- ➔ One of the first important decisions you'll make in CS107 is which editor you'll use to write your assignments. Unlike other CS classes you may have taken before, we won't be using an IDE in CS107, and we don't mandate you use any particular editor. You have a few options.
- ➔ Command line editor (recommended)
    - Using a command line editor is the most flexible option, and learning one is a good life skill. However, it will likely take some getting used to.
    - Wars have been waged over which editor is best. The two most popular options are **vim** and **emacs**. Both are very old and very flexible. You may want to try both and see which you prefer. Also see https://en.wikipedia.org/wiki/Editor_war
- ➔ GUI editor
    - Another option that may be more familiar is an editor with a GUI. There are many popular options in this category as well.
    - **Emacs** has a graphical mode that you can try on myth (or over ssh using ssh -Y, see below). **Gedit** is another popular GUI editor that comes preinstalled on myth. Finally, you you can use Vim in graphical mode using "gvim."
    - Alternately, you can edit your code locally on your own computer using any editor you wish. *This is more complex to set up, so do so with caution.*

VIM

- ➔ Unlike emacs, Vim is also the name of a household cleaning product. This makes it superior.
- ➔ "`vim test`"
    - opens a file called "`test`". If `test` doesn't already exist, vim will create it.
    - **Note:** vim opens all kinds of text files, even ones without extensions. Extensions are used so the OS knows what program to open the file with. But *a file doesn't necessarily need a .txt extension to hold text.*
- ➔ Vim has 2 modes - **command mode** and **insert mode**.
    - Command mode lets you execute commands (like undo, redo, find and replace, etc.)
    - insert mode lets you insert text.

    - Vim opens in command mode by default
- Type `i` to go to insert mode. You'll see "`--INSERT--`" appear at the bottom of the screen. Type your name.
- Hit escape to go back to command mode.
- *Some useful commands for command mode:*
    - dd - delete a whole line
    - u - to undo what you just did
    - `Ctrl + r` to redo
    - `:w` - save
    - `:wq` - save and quit
    - `:q` - quit (it'll give an error message if you try to quit without saving)
    - `:q!` - quit without saving
    - /(word) - search for your word in the document
- You can find info on Vim on the CS107 website in our CS107 Guide to Vim. Or, go through the Vim tutorial on myth with the command `vimtutor`
- Exit the document. Make sure you're in command mode!
- Set up your `.vimrc`. This is a file that configures your vim environment to let you do things like see line numbers and click with your mouse. (Clicking does not seem to work with Mac, sorry!)
    - Create the file with `vim ~/.vimrc`
    - Copy-paste the contents from here: http://web.stanford.edu/class/cs107/sample_vimrc
    - Save and quit
    - When you next launch vim, you should see the changes.

Emacs

- Emacs was created by Richard Stallman, whose beard is far more impressive than those of the creators of vim. This makes Emacs the superior text editor. Stallman is also a saint in the Church of Emacs.
- "`emacs test`"
    - opens a file called `test`. If `test` doesn't already exist, emacs will create it.
- Commands in emacs are usually combinations of Ctrl and other keys.
    - `Ctrl+X Ctrl+S` to save
    - `Ctrl+X Ctrl+C` to quit

- ➔ See the CS107 Guide to Emacs for much more info.
    - You may also want to read Emacs: The Bare Essentials, and this Emacs Refcard

gedit

- ➔ gedit is a popular graphical editor for Linux (though you can also install it on other operating systems). While we encourage you to become comfortable with vim and/or emacs, gedit will probably look more familiar.
- ➔ On a Myth machine, launch gedit as you would any other application.
- ➔ You can also run Gedit and other graphical programs remotely using **X11 forwarding**.
    - On **Windows**: Use MobaXterm.
    - On **Mac**:
        - Install XQuartz.
        - When you ssh into myth, add the **`-Y`** argument.
            - ex. ssh -Y skysmith@myth.stanford.edu
        - You can also enable compression to save some bandwidth with the -C flag.
    - On **both**:
        - Launch gedit from the command line by running the `gedit` program.
            - Like emacs/vim, you can also supply a file to open as an argument.
            - Launch gedit in the **background** by appending an & (ex. "`gedit &`"). This lets you keep using the command line.

Local Editors

- ➔ **Warning: Do this at your own risk. This is more complex to set up than the above, and can be finicky (as in, if it crashes you could lose data).**
- ➔ Rather than using ssh, you can also mount your AFS directory directly from your laptop. See here for more details: https://uit.stanford.edu/service/afs
- ➔ Then, you can use your favorite local editor (ex. Sublime Text), while still compiling and testing your code via ssh.

Setting up Mercurial

- Mercurial is a version control system. You'll use it to organize your work and submit assignments.
  - Version control software lets you backup your work, so if you screw up really badly you can go back to an old version. It also lets multiple people work on the same project together without overwriting each other's work, but we won't use this feature in 107 until the final assignment.
  - Mercurial is very similar to Git. Today Git is much more popular, but several years ago when CS107 was created that wasn't the case. Learning Mercurial will help you understand Git, if you ever use it later (which you probably will!).
- You need to set up Mercurial before you can begin Assignment 0.
- We need to make configuration file called .hgrc. Go to home directory (`cd ~`) and create a file called .hgrc (`vim .hgrc`) (note the dot; it's part of the required filename)
- Write this in the .hgrc file (replace my info with yours)

  -
    ```
    [ui]
    editor = vim
    username = Mindy Huang <mindyh@stanford.edu>

    [extensions]

    color =

    hgk =
    ```
- Save and quit - `:wq`
- `hg showconfig ui`
  - to see if the .hgrc file worked. If you don't see anything you may have an error/typo in your hgrc file.
- To learn more, go to the CS107 Guide to Mercurial. You can find the .hgrc starter file there, too.

Starting Assign0 - the bare minimum

- Open up Assign0 description on the cs107 website. *Most of the below is covered in more depth in the assignment description.*

- ➔ First thing to do is to clone the assignment files. Cloning copies the files from the server into your current directory.
- ➔ “`hg clone /afs/ir/class/cs107/repos/assign0/$USER assign0`”
  - ◆ This will create an assign0 folder with the starter code inside.
  - ◆ If you see an abort, you may not be enrolled in the class, so email us and we'll get you started with a repo.
- ➔ In the assign0 folder, open readme.txt. Just fill out your name for now.
- ➔ Save and quit
- ➔ “`hg status`”
  - ◆ shows you what was modified since your last commit. If you see nothing, no changes have been made, and your name wasn’t saved.
- ➔ Now commit your changes to Mercurial, so if you make a mistake you can revert to an earlier version.
  - ◆ `hg commit -m "added name in readme.txt"`
  - ◆ if you created any files (you shouldn’t have yet) you’ll need to `hg add filename`. This tells Mercurial it should start tracking the new file, so the next time you commit, this file will be saved as well
  - ◆ **Remember that you need to commit before submitting**. The submit script just uploads your latest commit.
- ➔ `hg log`
  - ◆ to see a list of all commits in order from most recent to oldest.
- ➔ Ready to submit? Use the cs107 submit script: `/afs/ir/class/cs107/tools/submit`
  - ◆ The submit script usually runs a simple check to see if you've filled out the text files you're supposed to, and sometimes runs a simple sanity check to see if your code (from assign1 onwards) satisfies some basic test cases.
  - ◆ You have an infinite number of submits, so use them! Only the last one will count. It’s always better to submit a rough draft first than to hold off on submitting until everything is perfect, especially if you’re running up against the clock. There have been multiple people who missed the hard deadline by a few seconds and got zeros; don’t be one of them!

Summary

- Any command is easily googleable - search ‘Unix command to do [thing]’
- `man` command: like Google, but lives in UNIX, will get you all the info you need on UNIX commands and C functions
  - ex. `man ls`, `man malloc`
- Things you will need to how to use know for Assign0
  - ssh
  - Vim or Emacs
  - Mercurial
  - Valgrind (we didn’t cover this, more info is on the website)
- All the info (on Unix, Mercurial, ssh, vim, emacs, etc.) to help you get started is on the CS107 site!

Quiz of some important things

- What’s the difference between relative paths and absolute paths? When do you use each one?
- What are the symbols for current, parent, root, and home directories?
- What is a man page? How do you read one?
- What does the submit script submit?
- How do you save and quit in Vim?
- How do you force quit a running program?