Basic Use of the Linux/UNIX Shell

Henri Casanova (henric@hawaii.edu)

What is This “Lecture” About?

- Some people in the class may need some type of “UNIX refresher”
- So here we go...
  - We could talk about all this for days, but I’ll let you discover things on your own
  - I won’t talk about text editors
    - vim, emacs, etc.
  - And there is obviously TONS of content on-line about all this
- If you’re familiar with UNIX/Linux, this is going to be pretty boring :)

Basics

- You'll be using the Shell: the command-line interface to the system (in a "terminal")
  - Either by SSh-ing into a server
  - Or by logging in to your own Linux (Virtual) box
- There are many kinds of Shell
- The most standard one is: /bin/bash
  - We're going to assume bash from now on
- To find out which shell your default is:
  - **`echo $SHELL`**
  - I use **`this font`** to denote commands you type
- SHELL is an environment variables
  - More on this later

Commands

- Every command, system program, or API call has a “man page”
  - **`man xxxx`**
- Commands take arguments, and/or input from stdin, and produce output on stdout
- Commands you know, but that may have tons of cool options you don’t know about
  - **`ls, cp, mv, rm, mkdir`**, …
- Reading man pages is a very worthwhile activity
  - Common thing heard in the work place “go read the man page”
- Some man pages are very instructive
  - **`man`** is a command, and you can do **`man man`**
- Let’s briefly go over few key “things”:
  - wildcards, gcc, make, pwd, cat, grep, |, less, wc, jobs (&,^Z, kill, fg)
- I am just going to go through a bunch of “random” examples

Wildcards

- `pwd`
  - Prints out the current directory
- `ls *.c`
  - Shows the list of all files named xxxx.c in the current directory
- `ls -l *.h *.c */*.g`
  - Shows a detailed list of all xxx.h and xxx.c files in the current directory and all xxx.g files in any q-level-deep subdirectory
- `ls -l dc??d*.c`
  - Shows all files in the current directory whose names start with “dc”, then 2 arbitrary characters, then “d”, and then an arbitrary number (possibly 0) of arbitrary variables

cat and grep

- The **`cat`** command takes as argument a file name and prints its content it to stdout (i.e., you will see it in the Shell terminal)
  - **`cat file.c`**
- **`grep`** finds a string in a file or in a set of files and prints the corresponding lines to stdout
  - **`grep main file.c`**
  - **`grep hello *.c */*.c`**
  - **`grep -v hello somefile`**
    - Will fine all lines that SO NOT contain “hello”

|, less, wc

- | is used to “pipe” commands together
  - The standard output of the command on the left of the ‘|’ goes to the standard input of the command on the right of the ‘|’
- **`less`**: sends a file to stdout but wait for user input to display more than the window size
  - e.g., **`cat file.c | less`**
- **`wc`**: counts lines, words, and characters in a file (-l for counting lines)
  - e.g., **`cat *.c | grep pthread_create | wc -l`**
    - counts the number of lines of code that contain “pthread_create”
  - e.g., **`ls | grep "a\.c" | wc -l`**
    - counts the number of files that contain a.c
    - The ‘\’ is used to “escape” the ‘.’ character, which is special (grep uses it as a wildcard)

Job management

- You can always start a command “in the background” with the & symbol
  - `ls -R | wc -l &`
- You get control right away and the running command is then called a “job”
- **`jobs`** is used to look at running jobs
- jobs can be accessed as %1, %2, ...
- **`fg %2`** brings job #2 to the foreground
- If a job is already running, hitting ^Z suspends the job and gives it a job id
- **`bg %4`** resumes suspended job in the background
- **`kill %7`** kills jobs #7
  - **`kill -9 %7`** is more violent

Environment Variables

- There are many environment variables:
  - **`printenv`**
  - **`echo $SHELL`**
  - **`echo $HOME`**
  - **`echo $USER`**
- Sometimes you'll have to set/modify environment variables
- Setting a new environment variable (or overwriting another one):
  - **`export NEWTHING= "a:b/c"`**
- Adding to a new environment variable:
  - **`export NEWTHING= "$NEWTHING hello"`**

Changing the Shell

- If you log in to a machine, and the Shell isn't the one you like, you can always just type, e.g., **`bash`**
  - The **`chsh /bin/bash`** command will change your default Shell to bash forever
  - Note that it needs the full path to the bash executable
  - If you don't pass it a valid path for bash, you're in trouble
- Finding the path to a command:
  - **`which`** ...
  - e.g., **`which ls`**
  - e.g., **`which gcc`**
- What's in your path?
  - **`echo $PATH`**
  - An important environment variable
- Adding to your path?
  - **`export PATH=$PATH:/some/new/directory/for/binaries`**

Customizing your Shell

- Default Shell behavior is stored in a file at the root of your directory called .bashrc
- In that file you can:
  - Create aliases
  - Set environment variables
  - And do a bunch of other things we won't talk about
- There is an art to .bashrc files
  - Changing the prompt is always amusing
  - The Web is full of sample .bashrc files, some simple, some less simple
- Let's look at the basic two things above

Aliases and Env Variables

- In your .bashrc file, anywhere, you can have a line like:
  `alias foo='blah'`
  - From now on, each time you type the foo command, the Shell will replace it by the blah command
- Highly recommended aliases
  - **`alias rm='rm -i'`**
  - **`alias mv='mv -i'`**
  - **`alias cp='cp -i'`**
- In your .bashrc file you can also set environment variables:
  `export FOO=BLAH`
  - Very useful for the PATH variables
  - **`export PATH=$PATH:/home/casanova/bin`**
    - Don't forget to just *add* to the old path, which comes with good default
    - Doing **`export PATH=foo`** will not be good as your Shell won't be able to run any commands

Customizing the Shell

- Once you've modified your .bashrc file, you need to "reset" the Shell
  - you can log out and back in, or
  - you can do **`source $HOME/.bashrc`**

TAB-completion, up arrow

- Tab-completion
  - While typing Shell commands, the TAB key is used to complete file names
  - One of the most useful features as nobody wants to type long file names
  - If there are multiple possible completions, hitting TAB again shows them all
- Up arrow
  - Hitting the up arrow recalls the last commands
  - Very useful to not re-type things over and over
- See the “history” command as well

That's it for Now

- The Shell is much more powerful than many people think and can do a lot for you
- Obviously we've only scratched the surface
- Bash scripts are real programs
- Being a Shell expert will impress your co-workers
- Knowing a scripting language (Perl, Python, etc.) is a good idea for your future
  - Could be useful for programming assignment #1 to avoid a bunch of by-hand work
  - Most people these days don't really learn much Shell programming and do everything in better scripting languages for rapid development
  - Take our Scripting Languages course (ICS215)

Conclusion

- Do not waste time on Shell/Linux issues if you get stumped
  - Google is your friend for resolving Shell issues!
  - Just ask questions, come to office hours
  - Most students end the semester having learned a lot, and some being “converts” :)