CSE 303
Lecture 3

bash shell continued:
processes; multi-user systems; combining commands

read *Linux Pocket Guide* pp. 25-33, 104-107,
111-113, 118, 122, 128-131, 138

slides created by Marty Stepp
http://www.cs.washington.edu/303/

Lecture summary

- processes and basic process management
- connecting to remote servers (`attu`)
  - multi-user environments
- combining commands
  - input/output redirection
  - pipes

Processes

Process commands

command	description
`ps or jobs`	list processes being run by a user; each process has a unique integer id (PID)
`top`	show which processes are using CPU/memory; also shows stats about the computer
`kill`	terminate a process by PID
`killall`	terminate several processes by name

- **process**: a program that is running (essentially)
  - when you run commands in a shell, it launches processes for each

- use `kill` or `killall` to stop a runaway process (infinite loop)
  - similar to ^C hotkey, but doesn't require keyboard intervention

Background processes

command	description
`&`	(special character) when placed at the end of a command, runs that command in the background
`^Z`	(hotkey) suspends the currently running process
`fg , bg`	resumes the currently suspended process in either the foreground or background

- If you run a graphical program like `gedit` from the shell, the shell will lock up waiting for the graphical program to finish
  - instead, run the program in the background, so the shell won't wait:

    ```
    $ gedit resume.txt &
    ```

  - if you forget to use `&` , suspend `gedit` with `^Z` , then run `bg`

Connecting with ssh

command	description
ssh	open a shell on a remote server

- Linux/Unix are built to be used in multi-user environments where several users are logged in to the same machine at the same time
  - users can be logged in either locally or via the network
- You can connect to other Linux/Unix servers with ssh
  - once connected, you can run commands on the remote server
  - other users might also be connected; you can interact with them

The attu server

- attu : The UW CSE department's shared Linux server

- connect to attu by typing:

```
ssh attu.cs.washington.edu
```

  (or `ssh username@attu.cs.washington.edu` if your Linux system's user name is different than your CSE user name)

- attu uses a shell other than bash as its default, so the first time you ever log in to it, you must run chsh to change shell to bash
  - when prompted for a new shell, type:

```
/bin/bash
```

- Note: There are several computers that respond as attu (to spread load), so if you want to be on the same machine as your friend, you may need to connect to attu2, attu3, etc.

Multi-user environments

command	description
`whoami`	outputs your username
`passwd`	changes your password
`hostname`	outputs this computer's name/address
`w` or `finger`	see info about people logged in to this server
`write`	send a message to another logged in user

- Linux/Unix are built to be used in a multi-user environment where several users are logged in to the same machine at the same time
  - users can be logged in either locally or via the network

- *Exercise* : Connect to `attu`, and send somebody else a message.

Network commands

command	description
`links or lynx`	text-only web browsers (really!)
`ssh`	connect to a remote server
`sftp or scp`	transfer files to/from a remote server
`wget`	download from a URL to a file
`curl`	download from a URL and output to console
`pine, mail`	text-only email programs

Text editors

command	description
`pico or nano`	simple but crappy text editors (recommended)
`emacs`	complicated text editor
`vi or vim`	complicated text editor

- you cannot run graphical programs when connected to `attu` (yet)
  - so if you want to edit documents, you need to use a text-only editor
- most advanced Unix/Linux users learn `emacs` or `vi`
  - these editors are powerful but complicated and hard to learn
  - we recommend the simpler `pico` (hotkeys are shown on screen)

Output redirection

command `>` ***filename***

- run ***command*** and write its output to ***filename*** instead of to console;
  - think of it like an arrow going from the command to the file...
  - if the file already exists, it will be overwritten (be careful)
  - `>>` appends rather than overwriting, if the file already exists
  - ***command*** `> /dev/null` suppresses the output of the command

- Example: `ls -l > myfiles.txt`
- Example: `java Foo >> Foo_output.txt`
- Example: `cat > somefile.txt`
  (writes console input to the file until you press ^D)

Input redirection

command* < *filename

- run ***command*** and read its input from ***filename*** instead of console
  - whenever the program prompts the user to enter input (such as reading from a `Scanner` in Java), it will instead read the input from a file
  - some commands don't use this; they accept a file name as an argument

- Example: `java Guess < input.txt`
- Example: `write stepp < insult.txt`
- Example: `java Guess < input.txt > output.txt`

- note that this affects *user input*, not *parameters*
- useful with commands that can process standard input or files:
  - e.g. `grep`, `more`, `head`, `tail`, `wc`, `sort`, `uniq`, `write`

Combining commands

command1 | ***command2***

- run ***command1*** and send its console output as input to ***command2***
- very similar to the following sequence:

```
command1 > filename
command2 < filename
rm filename
```

- Examples:

```
grep secret *.txt | uniq
ls -l *.txt | more
```

- *Exercise* : `names.txt` contains CSE student names, one per line, in "LASTNAME, FIRSTNAME" format. We are interested in students whose first names begin with "J", such as "Sherry, Justine".
  - Find out of how many such students are in the file.
  - Then figure out how many total letters (including comma and spaces) are in the full name of the last student in alphabetical order from this group.

Misusing pipes and cat

- Why doesn't this work to compile all Java programs?

```
ls *.java | javac
```

- Misuse of `cat`
  - bad: `cat filename | command`
  - good: `command < filename`
  - bad: `cat filename | more`
  - good: `more filename`
  - bad: `command | cat`
  - good: `command`

Commands in sequence

command1 `;` ***command2***

- run ***command1*** and then ***command2*** afterward (they are not linked)

command1 `&&` ***command2***

- run ***command1***, and if it succeeds, runs ***command2*** afterward
- will not run ***command2*** if any error occurs during the running of 1

- Example: Make directory `songs` and move my files into it.

```
mkdir songs && mv *.mp3 songs
```

Exercise

- The file `byebye.txt` contains a list of students who got < 2.0 in CSE 143, one per line, in sorted order by name.
  - Some names occur more than once, because they retook the course.

- Run the Java `Reject` program located in `Reject.class`, which emails rejection letters to these students.
  - The program reads the student names from the console; give the program the student names from `byebye.txt` as its input.
  - The `Reject` program should process each student only once, since it would be rude to send one person two rejection letters.
  - Once you get this to work, make it save the list of people sent rejection letters to the file rejected.txt.