98-172 : Great Practical Ideas for Computer Scientists

Now I Know My ABCs

Switching to Bash

The first order of business is switching your default shell to `bash`. To do this, use the following steps:

- `ssh` to `unix.andrew.cmu.edu` (Remember you can use the shortcut `ssh andrew`, if you set it up last time.)
- Type the command `chsh` at the prompt. You will be asked to type in a new shell; enter `/bin/bash`
- Even though it says it will take 24 hours, it should be nearly instantaneous. Log out of the session, and log back in.
- At this point, your default shell should be bash! Congratulations!!

Getting a `.bashrc`

Adam has written a `.bashrc` that should provide a bunch of neat features, as well as make your future life at CMU easier. You can get it (and the other files it needs) by typing the following commands. (**Hint:** Don't re-type the long file path multiple times; make use of the bash history by using the up arrow!)

- `cd ~`
- `cp /afs/andrew.cmu.edu/course/98/172/config/.bash_login .`
- `cp /afs/andrew.cmu.edu/course/98/172/config/.bashrc .`
- `cp /afs/andrew.cmu.edu/course/98/172/config/.bashrc_gpi .`

Then, **log out** again, and log back in. When your terminal starts up, type `gpi`, and hit enter. If you see a message with some help, then everything went correctly!

Basic Terminal Usage From Last Time

(a) **cd** ("change directory") is a program that will let you move between directories.

(b) **ls** ("list") is a program that will list the files in the current directory.

(c) **pwd** ("print working directory") is a program that will tell you the name of the current directory.

(d) **echo** <**what you want to echo**> is a program that will write whatever you want to your terminal.

(e) **cat** <**filename**> is a program that you should use to list the contents of a file

(f) **cp** <**source file**> <**destination file**>is a program that will "copy" files.

(g) **mv** <**source**> <**destination**> is a program that you can use for two purposes. The first is to "move" a file from one place to another. The second is to rename a file.

(h) **rm** <**file**> ("remove file") is a program that will **PERMANENTLY** delete a file or files. The `-r` switch will delete files recursively.

Bash Variables

Bash Variables are a good way of keeping track of information in Bash.

- To set a variable `VARIBLE` to the value `VALUE`, you should use the command `VARIABLE=VALUE`
- To use the variable `VARIABLE`, you should reference `$VARIABLE`. For instance, if you want to echo it, you would write `echo $VARIABLE`
- If you want to put a variable near something else, it might be necessary to do `${VARIABLE}` instead. For instance, if you wanted to do `echo ThisIsASentenceWithA${VARIABLE}AndNoSpaces`, the braces would be necessary.

Creating Links to Handin Directories

Many classes ask you to hand in your homework on AFS, and we have found it useful to create symbolic links to these directories for easy access. The first thing to do here is to create a directory to store all of them in one place. We suggest you name it `links`, but you can name it anything you want by changing the name below. Here are the steps to create the directory:

- `cd ~` (Go to the directory that you want to create it in; here, we suggest your home directory)
- `mkdir links` (Make a directory in the current directory called `links`)

Now that we have a directory, we should create the actual link:

- `cd links` (Move to where we want to create the symbolic link)
- `ln -s PATH_OF_HANDIN_DIRECTORY NAME_OF_SYMBOLIC_LINK` (Create the actual symbolic link)

The handin directories are as follows for selected classes:

15-112. You hand in your homework using Autolab

15-122. You hand in your homework using the `handin` script.

15-150. `PATH_OF_HANDIN_DIRECTORY` should be `/afs/andrew.cmu.edu/course/15/150/handin/${USER}` and `NAME_OF_SYMBOLIC_LINK` could be `150`.

At the very least, you should create a symbolic link with `PATH_OF_HANDIN_DIRECTORY` as `/afs/andrew.cmu.edu/course/98/172/handin/${USER}` and `NAME_OF_SYMBOLIC_LINK` could be `gpi` for this class, as an example.

Hints on Using a Terminal

Up Arrow. Bash keeps a history of the commands you type. You can navigate through them using the up and down arrow keys.

Tab. Bash can autocomplete commands, file names, etc. You can invoke auto-completion by hitting the tab key a couple of times.

Ctrl-R. `Ctrl-R` does a reverse search of your bash history. If you start typing in a piece of a command, it will look for commands you recently typed with that string in them. If the one it displays isn't what you want, you can cycle through them by typing `Ctrl-R`.

Ctrl-D. `Ctrl-D` sends an end of file character to whatever program you are in. This can be useful to get out of programs that are expecting input (like `python` and `cat`).

Ctrl-C. `Ctrl-C` makes an attempt to kill whatever program you are currently running.

Globs ("The star"). If you type `cat a*`, Bash will `cat` all the files that start with `a`. (The `*` tells bash to match any positive number of any types of characters in place of the `*`.)

Question Mark. If you type `cat a?`, Bash will `cat` all the files that are two characters and start with `a`. (The `?` tells bash to match any one character in place of the `?`.)

Current and Parent Directories. In a directory path, "." means the same directory, and ".." means the parent directory.

Other's Home Directories. To refer to other user's home directories, use `~andrewid`. For example, to go to Adam Blank's home directory, you would use `cd ~adamblan`.

Background and Foreground

The process that is currently controlling your terminal is called the **foreground** process. Any other process that you are running is a **background** process.

Listing the Current Jobs. To list the currently running **jobs** (processes in your current terminal), you should use the `jobs` command:

```
adamblan@unix34:~$ jobs
[1]-  Running                 sleep 1000 &
[2]+  Stopped                 vim
```

Starting a Process in the Background. To start a process in the background (if you want it to run but not take up control of your terminal), you should put an `&` at the end of the command:

```
adamblan@unix34:~$ sleep 1000 &
[1] 15025
```

Suspending the Current Foreground Process. To suspend the current program and put it in the "Stopped" state, you should press `Ctrl-Z`.

```
adamblan@unix34:~$ vim
(Ctrl-Z was pressed)
[2]+  Stopped                 vim
```

Foregrounding a Process. If a process is "Stopped," and you would like it to run in the foreground, type `fg <job #>`:

```
adamblan@unix34:~$ jobs
[1]+  Stopped                 sleep 1000
[2]-  Stopped                 vim
adamblan@unix34:~$ fg 1
sleep 1000
```

Backgrounding a Process. If a process is "Stopped," and you would like to run it in the background, type `bg <job #>`:

```
adamblan@unix34:~$ jobs
[1]+  Stopped                 sleep 1000
[2]-  Stopped                 vim
adamblan@unix34:~$ bg 1
[1]+ sleep 1000 &
adamblan@unix34:~$
```

Redirecting Input/Output

Choosing Standard In.

```
adamblan@unix34:~$ cat < file_with_text
This is a file with some text
adamblan@unix34:~$ wc -l < file_with_text
1
```

Redirecting Standard Out.

```
adamblan@unix34:~$ cat file_with_text > out_goes_here
adamblan@unix34:~$ cat out_goes_here
This is a file with some text
```

Appending Standard Out.

```
adamblan@unix34:~$ cat file_with_text
This is a file with some text
adamblan@unix34:~$ cat out_appends_here
This text is already in this file
adamblan@unix34:~$ cat file_with_text >> out_appends_here
```

```
adamblan@unix34:~$ cat out_appends_here
This text is already in this file
This is a file with some text
```

Redirecting Standard Error. Redirecting standard error looks nearly identical to redirecting standard out, except that instead of >, you use 2>

It is important to realize that standard out **will not be directed** if you just do this.

Redirecting BOTH Standard Out and Error.

- To redirect standard error **and** standard out, both to **standard out**, use `2>&1`
- To redirect standard error **and** standard out, both to a file, use `2>&1>`

More Useful Commands

(a) `alias NAME_OF_ALIAS='THE COMMAND'`
makes it so that typing `NAME_OF_ALIAS` runs `THE COMMAND`

(b) `date` is a command that tells you the current date and time.

(c) `watch 'THE COMMAND'`
will show you successive values of running `THE COMMAND` every few seconds.

(d) `time <program>`
will execute `program` tell you how long `program` took to run, when it exists.

(e) `kill -9 <process_id>` will terminate the program with process id `process_id`

(f) `killall -9 <program_name>` will terminate all processes which refer to programs with the name `program_name`

(g) `diff <file1> <file2>` will tell you the lines that differ between `file1` and `file2`.

(h) `wc <file>` will tell you how many lines, words, and characters are in `file`. If you do `wc -l`, it will only say how many lines.

(i) `fs`

fs la `<directory>`
This will list the permissions for the directory `directory`.

fs sa `<directory> <andrewid> X`, where `X` is either `none` or some of the following letters: `rwlidka`
This will set the permissions of `directory` for the user `andrewid` to `X`.

(j) `finger <andrewid>` is a command that will tell you information about that user. If you only know the person's name, you can do `finger "First Last"`, where the quotes are necessary, and `First` and `Last` are the person's first and last names, respectively.

(k) `whoami` is a command that tells you what your username is.

(l) `locate <filename>` is a command that tells you everywhere it can find that file on your `$PATH`.

(m) `ps` is a command that lists the processes currently running in your terminal. If you run `ps aux` instead, it will tell you all the processes running on that machine.

(n) `sort <filename>` is a command that will sort the lines of the given file and print them out.