CS2043 - Unix Tools & Scripting
Lecture 3
Making Bash Work For You

Spring 2015 [1]

Instructor: Nicolas Savva

January 26, 2015

[1] based on slides by Hussam Abu-Libdeh, Bruno Abrahao and David Slater over the years

Course Logistics

- Enroll by Wednesday (01/26 add deadline)
- Assignment 1 is due this coming Wednesday
- You can find the OH(s) time/locations through piazza
- We will, hopefully, have several TAs joining the staff (sometime this week)

Assignment 1: Questions/Concerns?

Using Bash Efficiently

In this lecture:

- More on file permissions
- File compression
- Customizing the prompt
- Shell shortcut keys
- Reusing history
- Aliasing
- Special character expansion

Revisiting File Permission

-rwxrwxrwx

- User's Permissions
- Group's Permissions
- Other's permissions

R = Read, W = Write, X = Execute

Directory Permissions begin with a `d` instead of a `-`

File Permissions - Another Convenient Way

We can think of r, w, x as binary variables:

- 0 = OFF
- 1 = ON

$$r * 2^2 + w * 2^1 + x * 2^0$$

Examples

- `chmod 755:` `rwxr-xr-x`
- `chmod 777:` `rwxrwxrwx`
- `chmod 600:` `rw-------`

Default Permissions - umask

This command will set the default permission of all the files you create (during the particular session)
Umask acts as the last stage filter that strips away permissions as a file or directory is created:

umask

`umask mode`

- Remove mode from the file's permissions

Examples

- `umask 077:` full access to the user, no access to everybody else
- `umask g+w:` enables the group write permission (alternative notation)
- `umask -S:` display the current mask

File Compression

- Compress and archive files into a single file
- A new file is created (.zip) and the original files stay intact

zip

```
zip <zip_file_name> <files_to_compress>
```

unzip

```
unzip <zip_file_name>
```

Read the man page for more options

- add files to existing zip
- encrypt files and use password
- ...

More File Compression

gzip

```
gzip <files_to_compress>
```

gunzip

```
gunzip <compressed_file_name>
```

- compress file using Lempel-Ziv coding
- Does not bundle files
- replaces original files

More File Compression - tar

Archive multiple files together

tar - Tape Archive

```
tar -cf <tar_file_name> <files_to_compress>
 ⟹   create tar archive
tar -xf <tar_file_name>
 ⟹   extract all files from tar archive
```

- `tar` bundles multiple files together into a single file
- Does not compress
- Does not replace the files

A Backup Script

Here is something a little more practical - a simple script to back up all the files in your documents directory:

Example: backup.sh

```
#!  /bin/bash
tar -czf ~/backups/cs2043.backup.tar.gz \
~/Documents/cs2043/
```

This script makes use of the `tar` archiving command:

Making Tarballs:

```
tar -c(z/j)f <dest_archive> <source>
tar -x(z/j)f <archive>
```

- `-c` version creates a new archive from a source file/dir
- `-x` extracts an existing archive to the current dir
- pick either `-z` or `-j` options (-z ⇒ .tar.gz , -j ⇒ .tar.bz2)

wget and curl

wget

```
wget [OPTIONS] [URL...]
```

Download a file from a remote location over HTTP. Popular options:

- `-r` : recursive
- `-c` : continue a partial download

curl

```
curl [OPTIONS] [URL...]
```

Transfer data from/to web servers.

For more info on these commands, consult the `man` pages.

Shells Again

Many shells for UNIX-like systems:

- `sh`: The Bourne Shell -
  a popular shell made by Stephen Bourne

- `bash`: The Bourne Again Shell -
  default shell for the GNU OS, most Linux distros, and OSX

- `csh`: The C Shell -
  interactive and close to C
  default shell for BSD-based systems

- `zsh`: The Z Shell -
  possibly the most fully-featured shell inspired by sh, bash, ksh, and tcsh

Shells Again

- Since `bash` is the gold standard of shells and has more than enough features for this course, we'll stick with it.
- For more info, use Wikipedia as a starting point: http://en.wikipedia.org/wiki/Comparison_of_command_shells

My machine/CSUGlab does not default to bash

If the machine automatically put you into a shell that is not bash:

- If you are already logged in, just type `bash`
- More importantly we would like the csuglab machine to automatically put us into bash when we login. One way to do this is by editing the file ~/`.login` which gets executed each time you log into the server.

Start bash automatically

Add the following line to the end of ~/`.login`

```
if ( -f /bin/bash) exec /bin/bash --login
```

If you had root privileges you could just edit /etc/passwd and find the line corresponding to the current user.

Modifying your Prompt

The environment variable $PS1 stores your default prompt. You can modify this variable to spruce up your prompt if you like:

Example

First `echo $PS1` to see its current value
`\s-\v\$` (default)

It consists mostly of backslash-escaped special characters, like `\s` (name of shell) and `\v` (version of bash). There are a whole bunch of options, which can be found at

http://www.gnu.org/software/bash/manual/bashref.html#Controlling-the-Prompt

Modifying Your Prompt

Once you have a prompt you like, set your $PS1 variable

Define your prompt

```
nsavva@x200t:~$ export PS1="New Prompt String"
```

- Type this line at the command prompt to temporarily change your prompt (good for testing)
- Add this line to `~/.bashrc` or `~/.bash_profiles` to make the change permanent.

Note: Parentheses must be used to invoke the characters.

Examples

`PS1="\u@\h:\w\>"` ⇒ `nsavva@x200t:~>`

`PS1="\h \w(\@-\d]"` ⇒ `x200t ~(12:05 PM-Mon Jan 26]`

Shell Shortcuts

Make entering commands easier:

- Tab completion
- Up-down arrow: browse through command history
  - so you do not have to retype everything
- Ctrl + e: jump cursor to end of line
- Ctrl + a: jump cursor to beginning of line
- Ctrl + u : delete everything from cursor to beginning of line
- Ctrl + k : delete everything from cursor to end of line
- Ctrl + w : delete everything from cursor to beginning of word
- Ctrl + l : clear the screen

More shortcuts at:
http://linuxhelp.blogspot.com/2005/08/bash-shell-shortcuts.html

Reusing History: Bang (!)

Use the *bang* operator (`!`) to repeat a command from your history that begins with the characters following it.

Example

```
nsavva@x200t:~$ pdflatex lecture3.tex
nsavva@x200t:~$ !p
nsavva@x200t:~$ !pdf
```

`!p` and `!pdf` will recall `pdflatex lecture3.tex`

Using `!` can save you many keystrokes when repeating tasks.

Reusing History: Search

You can search through your command history using the shortcut Ctrl + R:

- Press Ctrl + R and type a search string. Matching history entries will be shown.
- Press Ctrl + R again to see other matches.
- If you like an entry, press ENTER to re-execute it.
- Press ESC to copy the entry to the prompt without executing.
- Press Ctrl + G to exit search and go back to an empty prompt.

Example

```
(reverse-i-search)‘sc’:  scp -r n@x1.cornell.edu: /Downloads/ ./
```

Note that you will not be able to type if there are no matches.

Aliases

The more you use bash the more you see what options you use all the time. For instance `ls -l` to see permissions, or `rm -i` to insure you don't accidentally delete a file. Wouldn't it be nice to be able to make shortcuts for these things?

Alias:

`alias name=command`

- The alias allows you to rename or type something simple instead of typing a long command
- You can set an alias for your current session at the command prompt
- To set an alias more permanently add it to your `.bashrc` or `.bash_profile` file in your home directory.

Alias Examples

Examples

```
alias ls=‘ls --color=auto’
alias dc=cd
alias ll="ls -l"
```

- Quotes are necessary if the string being aliased is more than one word
- To see what aliases are active simply type `alias`
- Note: If you are poking around in .bashrc you should know that any line that starts with # is commented out.

Shell Expansion

In a bash shell, if we type:

```
$ echo This is a test
This is a test
```

But if we type

```
$ echo *
Lec1.pdf Lec1.dvi Lec1.tex Lec1.aux
```

What happened?
The shell expanded * to all files in the current directory. This is an example of path expansion, one type of shell expansion.

Shell Expansion

`*` `^` `?` `{` `}` `[` `]` Are all "wildcard" characters that the shell uses to match:

- Any string
- A single character
- A phrase
- A restricted set of characters

The shell's ability to interpret and expand commands is one of the powers of shell scripting.

Shell Expansion

- `*` matches any string, including the null string (i.e. 0 or more characters).

Examples:

Input	Matched	Not Matched
`Lec*`	`Lecture1.pdf Lec.avi`	`ALecBaldwin/`
`L*ure*`	`Lecture2.pdf Lectures/`	`sure.txt`
`*.tex`	`Lecture1.tex Presentation.tex`	`tex/`

Shell Expansion

- ? matches a single character

Examples:

Input	Matched	Not Matched
`Lecture?.pdf`	`Lecture1.pdf Lecture2.pdf`	`Lecture11.pdf`
`ca?`	`cat can cap`	`ca cake`

Shell Expansion

- `[...]` matches any character inside the square brackets
  - Use a dash to indicate a range of characters
  - Can put commas between characters/ranges

Examples:

Input	Matched	Not Matched
`[SL]ec*`	`Lecture Section`	`Vector.tex`
`Day[1-4].pdf`	`Day1.pdf Day2.pdf`	`Day5.pdf`
`[A-Z,a-z][0-9].mp3`	`A9.mp3 z4.mp3`	`Bz2.mp3 9a.mp3`

Shell Expansion

- `[^...]` matches any character **not** inside the square brackets

Examples:

Input	Matched	Not Matched
`[^A-P]ec*`	`Section.pdf`	`Lecture.pdf`
`[^A-Za-z]*`	`9Days.avi .bash_profile`	`vacation.jpg`

Shell Expansion

- **Brace Expansion:** `{...,...}` matches any phrase inside the comma-separated brackets

Examples:

Input	Matched
`{Hello,Goodbye}\ World`	`Hello World Goodbye World`

NOTE

Brace expansion must have a list of patterns to choose from. (i.e. at least two options)

Shell Expansion

And of course, we can use them together:

Input	Matched	Not Matched
`*i[a-z]e*`	`gift_ideas profile.doc`	`DriVer.exe`
`[bf][ao][ro].mp?`	`bar.mp3 foo.mpg`	`foo.mpeg`

Interpreting Special Characters

The following are special characters:

```
$ * < > & ?  { } [ ]
```

- The shell interprets them in a special way unless we escape (\$) or place them in quotes "$".
- When we first invoke a command, the shell first translates it from a string of characters to a UNIX command that it understands.
- A shell's ability to interpret and expand commands is one of the powers of shell scripting.

We will cover all those special characters later in the course.

Later in the course...

More to come:

- Piping, input/output redirection
- More useful unix commands
- text editing: `nano`, `vi/vim`
- `ssh/sftp/scp`, `screen/tmux`
- And much more!