Lecture 5 – Shell programming

Learning Objectives:

2. Become familiar with the use of Bash, shell programming, and console editors

   2.7 Understand the use of environment-setting shell files.

   2.8 Understand the syntax of flow control elements.

   2.9 Learn the variable naming conventions of Bash.

   2.10 Produce shell scripts.

4. Produce code that is reproducible and produces results that are replicable.

   4.5 Use scripts to make command-line functions reproducible.

Announcement: Pick an *R* package to present during class 9/23 & 9/28!

Your Bash Shell

Two types of Shells: login and non-login

- Login shell: for interactive instances (mostly)
  - logs in with `/bin/login` and `/etc/profile.d/`
  - will read in `~/.bash_profile` instead of `~/.bashrc` by default
  - most instances of Terminal, other prompt-type interfaces
  - When tested with `echo $0` should return `-bash`
- Non-login shell: started by a program without a login, by just passing the name of the shell
  - will call `~/.bashrc` but not `~/.bash_profile`
  - `~/.bashrc` (if it exists) will call `/etc/profile.d/`
  - mostly called by executed scripts

Your Bash Shell

Setting your `.bash_profile`

- Purpose: configure your personal shell environment
- Location: in home directory, hidden file
- put this in `.bash_profile` to ensure that you have the same working environment in your login and nonlogin shells:

```
if [ -f ~/.bashrc ]; then . ~/.bashrc; fi
```

- setting aliases: this is really helpful for creating shortcuts to common locations and programs

```
alias matlab="/Applications/MATLAB_R2019a.app/bin/matlab
-nodisplay -nosplash -nodesktop"

alias gobox="cd '/Users/waldrop/Dropbox (Chapman)/'"
```

- after making changes, don’t forget to source it to load those new commands: `source ~/.bash_profile`

Your First Bash Script

Create a script file:

```
$ touch helloworld.sh
```

In console editor:

```
#!/bin/bash

#Prints ‘Hello world’
echo Hello world
```

Run the script:

```
$ sh helloworld.sh
```

Make the script executable, then run:

```
$ chmod +x helloworld.sh

$ ./helloworld.sh
```

Why use scripts?

- You can look at it without running it
- Can be run by you or anyone else
- Makes your analysis reproducible!!

Your First Bash Script

Create a script file:

```
$ touch helloworld.sh
```

In console editor:

```
#!/bin/bash

#Prints 'Hello world'
echo Hello world
```

Parts of a shell script:

1. **Hashbang** — tells shell how to interpret commands
2. **Contents** – Content of script (from Bash shell)
3. **Comments** — begin with pound.

Run the script:

```
$ sh helloworld.sh
```

Make the script executable, then run:

```
$ chmod +x helloworld.sh

$ ./helloworld.sh
```

Your Second Bash Script

Group work: Make a script of the NW1 exercise from last class.

Variables in Bash

Set a variable:

```
$ X=lobster
```

Call a variable:

```
$ echo $X
$ echo “$X”
$ echo ${X}
```

NOT:

```
$ echo X
$ echo ‘$X’
```

Curly brackets and backslashes help define names:

```
$ echo ${X}_file
$ echo $X\_file
```

NOT:

```
$ echo $X_file
```

Concept check: Add a variable to your helloworld.sh script and have it print the variable out!

Variables in Bash

Use externally define variable: positional parameters

```
#!/bin/bash

#Prints ‘Hello world’
echo Hello world

File=$1
cat $File
```

```
$ sh helloworld.sh examplefile.txt
```

or

$$\texttt{\$ } \underbrace{\texttt{./helloworld.sh}}_{\text{script}}\ \underbrace{\underbrace{\texttt{examplefile1.txt}}_{1}\ \underbrace{\texttt{examplefile2.txt}}_{2}}_{\text{positional parameters}}$$

Concept check: Try a few other text files as positional parameters.

Variables in Bash

Use externally define variable: positional parameters

```
#!/bin/bash

#Prints 'Hello world'
echo Hello world

File=$1
cat $File
```

Optionally set a file!

```
D=default
File=${1:-$D}
cat $File
```

```
$ sh helloworld.sh
```

See link "parameter expansion"!

```
$ sh helloworld.sh examplefile.txt
```

or

```
$ ./helloworld.sh examplefile1.txt examplefile2.txt
```

$\underbrace{\texttt{./helloworld.sh}}_{\text{script}}\ \underbrace{\texttt{examplefile1.txt}}_{1}\ \underbrace{\texttt{examplefile2.txt}}_{2}$

positional parameters

Concept check: How could you return an error message if the script is given no argument?

Variables in Bash

Setting patterns within bash variable names:

`${var%pattern}` removes pattern from end of variable

```
$ F=file.100
$ echo ${F%.100} ⟶ file
```

`${var#pattern}` removes pattern from beginning of variable

```
$ F=file.100
$ echo ${F#file.} ⟶ 100

$ F="git-practice-repo/practicefiles/file.100"
$ echo ${F##*/} ⟶ file.100
```

Concept check: For the last example, write an echo command to return the directory name *only*: git-practice-repo, practicefiles.

Your Second Bash Script

Group work: Make a script of the NW1 exercise from last class.

Group work: For your NW1 script, now make it so you add the file to be split as a positional parameter and it automatically splits the two files. Note that none of the line numbers can be hard coded (or coded directly as numbers in the script), they should be calculated within the program!

More Information

Difference between login and nonlogin shells:
http://howtolamp.com/articles/difference-between-login-and-non-login-shell/

Bash Programming Tutorial:
https://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html

More on parameter expansion:
http://wiki.bash-hackers.org/syntax/pe