Lecture 18 – 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.

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 setup/analysis/ whatever 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. (Just put lines of code in a script that's executable. Don't worry now about hard-coding numbers.)

Variables in Bash

Note: unlike R, no whitespace is allowed!!

Set a variable:

```
$ X=lobster
```

Call a variable:

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

NOT:

```
$ echo X
$ echo ‘$X’
$ echo `$X`
```

Curly brackets and backslashes help call names:

Add “file” to lobster:

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

NOT:

```
$ echo $X_file
$ echo ${X_file}
```

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

Variables in Bash

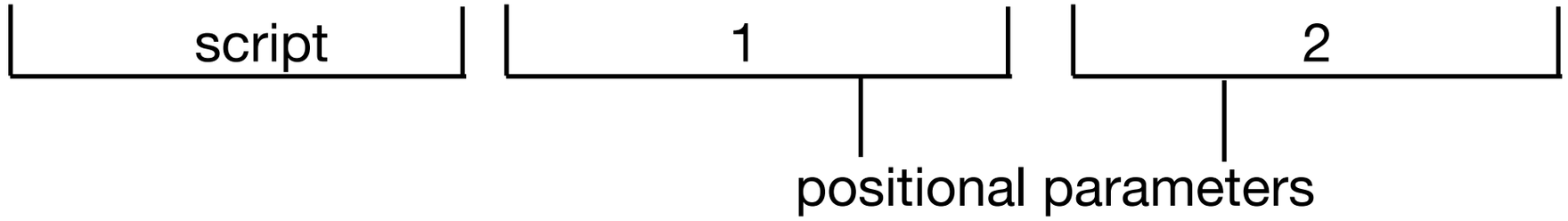
Use externally define variable: positional parameters

```
#!/bin/bash

#Prints ‘Hello world’
echo Hello world

File=$1
cat $File
```

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

or

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



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
```

script | 1 | 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: 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