Bash Scripting

Research Computing
UNIVERSITY OF COLORADO **BOULDER**

Be Boulder.

Bash Scripting

- Daniel Trahan
- *Email: Daniel.Trahan@Colorado.edu*
- *RC Homepage: https://www.colorado.edu/rc*

Sign in! http://tinyurl.com/curc-names

- Slides available for download at: https://github.com/ResearchComputing/CU_DENVER_HPC_2019

Adapted from presentations by RC members Andrew Monaghan, Aaron Holt and John Blaas: 1, 2, 3, 4.

Outline

- Bash and Bash Scripts
- Variables
- Conditionals and Tests
- Iteration
- Syntax
- Command line Arguments

Bash?

- A shell is the environment in which commands are interpreted in Linux
- GNU/Linux provides various shells; bash most popular
  - sh Bourne-again Shell (Bash)
  - csh C-Shell
  - tcsh Tc-Shell
  - ksh Korn-shell
- Shell scripts are files containing collections of commands for Linux systems that can be executed as programs

Bash Script

- To create a bash shell script file, the first line must be:

```
#!/bin/bash
```

- Program loader recognizes the `#!` as an interpreter directive. This is followed by `/bin/bash` which tells the OS which shell should be used.
- A `#` denotes a comment in the bash script.
- Example:

```
#!/bin/bash

cd /home/user

hostname
echo “Hello!” > file.out
```

Permissions

- Before you can run a script you need to make sure the script has the appropriate permissions
- At the command line, type, `ls -l`
- Column 1: Permissions
  - d, r, w, x
  - Owner, group, global
- Chmod changes permissions
  - `chmod a+x filename.sh`
  - Makes the file executable for everyone
- Run it, using `./filename.sh`
- Could also have done bash filename.sh and avoided permissions

Bash Variables

- Local variables are usable within the current shell
- Environment variables are usable to any child process of the shell
- Several pre-defined environment variables in your container
  - Type `env` in the terminal window
- Variables can be stored in arrays can be used too!
- Examples (Try these out!)

Local Variables

```
NAME=CUBOULDER
echo $NAME

ARRAY=(Hello World)
echo ${ARRAY[0]}
```

Environment Variables

```
export ENVVAR=Persists!
echo $ENVVAR

/bin/bash
echo $ENVVAR
```

Bash Conditionals

- 3 varieties of conditionals exist: if then, elif then, else
- Many comparative operators! (See next slides…)
- Syntax is a bit odd:

`if then, else` loop

```
if [ $x -gt 10 ]; then
    echo $x
else
    hostname
fi
```

`if then, elif then, else` loop

```
if [ $x -gt 10 ]; then
    echo $x
elif [ $x -lt 4 ]; then
    let x++
else
    hostname
fi
```

Tests I

- Conditions are evaluated between `[ ]` or `[[ ]]` after the test word.
- File comparisons
  - Exists `[ -f file ]`
  - Executable `[ -x file ]`
  - Newer than `[ file1 -nt file2 ]`
  - Older than `[ file1 -ot file2 ]`
- Integer comparisons
  - Equal `[ num1 -eq num2 ]`
  - Not Equal `[ num1 -ne num2 ]`
  - Less than `[ num1 -lt num2 ]`
  - Less or equal `[ num1 -le num2 ]`
  - Greater than `[ num1 -ge num2 ]`

Tests II

- String comparisons
  - Equal `[ string1 = string2 ]`
  - Not equal `[ string1 != string2 ]`
  - Contains `[ string1 =~ string2 ]`
  - Non zero `[ -n string1 ]`
  - Zero `[ -z string1 ]`
- Combining tests
  - And `[ exp1 -a exp2 ]`
  - Or `[ exp1 -o exp2 ]`

Example: Testing a Variable

- Lets use our newly found knowledge to test a variable we set:
- Create a script that sets the variable $DOGS and checks to see if the variable $DOGS is equal to 10. If so, use the echo command to print:

```
There are 10 DOGS
```

- Tips:
  - Remember #!/bin/bash
  - Make sure to use numerical tests!

Example: Testing a Variable

- Answer

```
#!/bin/bash

DOGS=10

if [ $DOGS -eq 10 ]; then
    echo “There are 10 DOGS”;
fi
```

Bash Iteration

- Two standard loops: `for` and `while`:
  - For loops iterate over a set count
  - While loops iterate until a certain condition is met
- `for` syntax:

```
for i in {0..10}; do
    echo $i;
done;
```

```
abc = (a b c)
for i in ${abc[@]}; do
    echo $i;
done;
```

`while` syntax:

```
i=0
while [ $i -gt 10 ]; do
    echo $i;
    let i++;
done;
```

Example: File Creation

- Create a shell script that generate 6 files named 0, 1, 2, 3, 4, 5
- Utilize the touch command! Creates an empty file. Syntax:
- Tips:
  - Remember the #!/bin/bash
  - Try touching a file without the loop

```
touch <file-name>
```

Example: File Creation

- Answer:

```
#!/bin/bash

for i in {0..5}; do
    touch $i
done
```

Command and Variable Substitution

- Sometimes you may need to use a variable or the output of a command in another command.
- Easily done with command and variable substitution!
- Command Substitution
  - Syntax `$(COMMAND)`

```
ls $(pwd)/another/directory
```

- Variable Substitution
  - Syntax: `${VARIABLE}`

```
NAME=Dan
USERFULL=${NAME}iel
```

The Many Braces of Bash

- As you may have noticed, syntax for Bash can be a *bit* overwhelming
- A lot of the complexity comes from strict syntax and specific uses for characters...
- Braces, Brackets, and Parenthesis

• Single Braces:	`[ ]`	Conditional Expressions
• Double Braces:	`[[ ]]`	Conditional Expressions
• Single Parenthesis	`( )`	Array declaration and Subshell
• With leading	`$( )`	Command Substitution
• Double Parenthesis	`(( ))`	Arithmetic Instruction
• With leading	`$(( ))`	Arithmetic Expansion
• Single Curly Brace	`{ }`	Group commands in current shell and ranges
• With leading	`${ }`	Parameter Expansion

Quoting

- Quoting is used to remove the special meaning of certain characters or words to the shell.
  - Lots of characters have special meanings in Bash
- Types of Quoting
  - Escaping: `\$` Remove the special meaning of the next character.
  - Single quotes: `'string'` Convert the string to whatever is literally written
  - Double quotes: `"$var"` Return as string but interpret variables.

Command Line Arguments

- Arguments are a great way to pass your own variables into a shell script.
- Special Variables are filled with arguments you provide when invoking the script:
  - $0 denotes the script name.
  - $1 denotes the first argument, $2 the second, up to ${99}.
  - $# the total number of arguments.
  - $* all arguments as a single word
  - $@ all arguments as individual words.

```
./example_script.sh arg1 arg2
```

```
#!/bin/bash

echo $0
echo $1
echo $2
```

Example: User Specified Files

- Lets expand on our previous file creation example by allowing a user to provide their own filename.
- Use Command Line arguments
- Filenames should be named <user-argument>1.txt, <user-argument>2.txt, etc...

Example: User Specified Files

- Answer:

```
#!/bin/bash


for i in {0..5}; do
touch ${1}${i}.txt;
done
```

Thanks!

- Please fill out the survey: http://tinyurl.com/curc-survey18
- Sign in! http://tinyurl.com/curc-names
- Contact information:

  rc-help@Colorado.edu

  Daniel.Trahan@Colorado.edu

  Andrew.Monaghan@Colorado.edu
- Slides and Examples from this course: https://github.com/ResearchComputing/CHANGE_2019
- Slurm Commands: https://slurm.schedmd.com/quickstart.html