Introduction to Bash Shell Scripting

Andrew Monaghan
andrew.monaghan@Colorado.edu
https://www.rc.colorado.edu

Slides available for download at
_https://github.com/rctraining/HPC_Short_Course_Spring_2018_

Adapted from a __presentation__ by Tim Brown, RC , 12 Feb 2015

Overview

- ▸ Introduction
- ▸ Variables
- ▸ Quoting
- ▸ Command Substitution
- ▸ Arithmetic Expansion
- ▸ Tests
- ▸ Decisions (`if`)
- ▸ Loops (`for, while`)
- ▸ Arguments
- ▸ Functions
- ▸ Alternatives

Research Computing
UNIVERSITY OF COLORADO **BOULDER** 2/28/18 Introduction to Bash Scripting **Be Boulder.**

Introduction

A *shell* is the environment in which commands are interpreted in Linux.

GNU/Linux provides various numerous shells; the most common one is the Bourne Again shell (`bash`).

Other common shells available on Linux systems include:
1) sh ; 2) csh ; 3) tcsh ; 4) ksh

Shell scripts are files containing collections of commands for Linux systems that can be executed as programs.

Shell scripts are powerful tools for performing many types of tasks.

Research Computing   UNIVERSITY OF COLORADO **BOULDER**   2/28/18   Introduction to Bash Scripting   **Be Boulder.**

Introduction

- ▸ Can be programmed interactively, directly on the terminal.
- ▸ It can also be programmed by script files. The first line of the file must contain `#!/bin/bash`.
- ▸ The program loader reconizes the `#!` and will interpret the rest of the line (`/bin/bash`) as the interpreter program.
- ▸ If a line starts with `#`, it is a comment and is not run.

An example shell script:

```
1   #!/bin/bash
2   # Shell script to list
3   # the files in /tmp.
4
5   cd /tmp
6   ls
```

← ———————— Shell to run

← ———————— Comments

← ———————— Change directories

← ———————— Directory listing

Research Computing
UNIVERSITY OF COLORADO **BOULDER**   2/28/18   Introduction to Bash Scripting

Be Boulder.

Variables

- There are no data types.
- A variable can contain a number, a character, a string of characters.
- Shell variables are local.
- Environment variables are global.

```
$ PI=3.14159
$ name=(Andy Monaghan)
$ echo ${name[0]}
$ Andy
$ echo $USER
$ monaghaa
```

Quoting

Quoting is used to remove the special meaning of certain characters or words to the shell.

Quotation	Description
`'string'`	Literally treat as string
`"$var"`	Treat as string but interpret variables
`{ }`	Disambiguation

Creating a file with my username in it's name.

```
$ touch "output_${USER}.txt"
```

Research Computing
UNIVERSITY OF COLORADO **BOULDER**

2/28/18   Introduction to Bash Scripting

Be Boulder.

Command Substitution

Command substitution allows the output of a command to be substituted in place of the command name itself.

- ► By enclosing the command with `$()`.
- ► Legacy syntax is using backticks `` `` ``.

```
$ NOW=$(date +%Y-%m-%d)
$ echo $NOW
2015-02-11
```

Be Boulder.

Arithmetic Expansion

Arithmetic expansion provides a mechanism for evaluating an arithmetic expression and substituting its value.

- By enclosing the command with `$(())`.

```
$ sqr_two=$(( 2 * 2 ))
$ echo ${sqr_two}
$ 4
```

Note that Bash only does integer math by default, however it is easy to do floating point math with 'bc'….

```
$ echo "5.6/9.4" | bc -l
$ .59574468085106382978
```

Research Computing   UNIVERSITY OF COLORADO **BOULDER**   2/28/18   Introduction to Bash Scripting   **Be Boulder.**

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 ]`

A full list is in the `test` manual page (`man test`).

Decisions I

The if command executes a compound-list.

  ▸ Consisting of if, elif, else and fi.

```
x=$(date +%M)
if [ $x -gt 30 ] ; then
      echo "last half of the hour"
elif [ $x -lt 15 ] ; then
      echo "first quarter of the hour"
else
      echo "we're at ${x}"
fi
```

Be Boulder.

Decisions II

The `case` command executes a compound-list too.

▸ Consisting of `case` and `esac`.

```
x=10
case ${x} in
  1)  echo "one"       ;;
  5)  echo "five"      ;;
  10) echo "ten"       ;;
  *)  echo "unknown"   ;;
esac
```

Loops

There are two types of loops:

```
x=0
while [ $x -lt 10 ] ; do
  echo $x
  x=$(( $x + 1 ))
done
```

```
list=(a b c)
for v in ${list} ; do
  echo $v
done
```

▸ `continue` will start the next iteration.
▸ `break` will exit the loop.

Be Boulder.

Arguments I

It is often useful to pass arguments to a shell 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.

Arguments II

```bash
#!/bin/bash
# Calculate the sine of the argument.

if [ $# -eq 1 ] ; then
  sine=$(echo "s($1)" | bc -l )
  echo "The sine of $1 is ${sine}"
else
  echo "Usage: $0 <number in radians>" 2>&1
  exit 1
fi
```

Research Computing
UNIVERSITY OF COLORADO **BOULDER** 2/28/18 Introduction to Bash Scripting **Be Boulder.**

Functions I

A function is a user-defined name that is used as a simple command to call a compound command with new positional parameters.

```
function_name () {
  commands
}
```

It is good practice to check the exit status of commands. I do this repeatedly in scripts, so a function is best to define it.

Research Computing
UNIVERSITY OF COLORADO **BOULDER** 2/28/18 Introduction to Bash Scripting **Be Boulder.**

Functions II

```
#/bin/bash

e () {
  echo $1
}

#now test
e Hello
e World
```

Alternatives for Scripting

- ► `csh/tcsh`   C-shell (tcsh: updated version of csh).
- ► `ksh`   Korn shell; related to sh/bash
- ► `perl`   exceptional text manipulation and parsing.
- ► `python`   excellent for scientific and numerical work.
- ► `ruby`   general scripting.
- ► `make`   building executables from source code

Thank you!

Please fill out the survey:
http://tinyurl.com/curc-survey16

Additional Bash learning resources:
http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html (general)
https://www.shell-tips.com/2010/06/14/performing-math-calculation-in-bash/ (math)

Bash kernel for jupyter notebooks *(install anaconda first)*:
https://github.com/takluyver/bash_kernel

Be Boulder.