Supercomputing Spinup: Bash

Research Computing
UNIVERSITY OF COLORADO **BOULDER**

Be Boulder.

Introduction to Bash Shell Scripting

- Gerardo Hidalgo-Cuellar

- *RC Homepage:* https://www.colorado.edu/rc/
- *RC Docs:* https://curc.readthedocs.io/en/latest/
- *RC Helpdesk:* rc-help@colorado.edu

- Course Materials: https://github.com/ResearchComputing/Supercomputing_Spinup

- Survey: http://tinyurl.com/curc-survey18

Intro Demonstration

- A quick intro to demonstrate how you might use shell scripts
- Demo: Making a tool to manage the output from `squeue`
  - Often 1000 lines on Summit
  - Hard to manage
  - Harder to make meaningful

Let's log in to RC

- (If you are not using your local system)
- To connect to a remote system, use Secure Shell (SSH)
  - From Windows – GUI SSH app such as PuTTY
  - From Linux or Mac OS X terminal, or Windows GUI such as PuTTY or Gitbash, ssh on the command line:

```
ssh <username>@login.rc.colorado.edu
```

Access the slides and examples

- How to get there: github.com/ResearchComputing/Supercomputing_Spinup
- Navigate to your workspace (e.g. /projects/$USER), go into the scripts directory for our demos:

```
[user@shas0137 ~]$ git clone
https://github.com/ResearchComputing/Supercomputing_Spinup.git
[user@shas0137 ~]$ cd Supercomputing_Spinup/bash_spinup/scripts
```

Overview

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

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:

- sh, csh, tcsh, ksh, zsh

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.

- 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 recognizes 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.

```
#!/bin/bash
# the files in /tmp.
cd /tmp
ls
```

Shell to run
Comments
Change directories
List everything in /tmp

File editing

- **nano** – simple and intuitive to get started with; not very feature-ful; keyboard driven
- **vi/vim** – universal; keyboard driven; powerful but some learning curve required
- **emacs** – keyboard or GUI versions; helpful extensions for programmers; well-documented
- **LibreOffice** – for WYSIWYG
- Use a local editor via an SFTP program to remotely edit files.

Example 1

`test.sh`

Note: you can use “nano” to edit files in this tutorial

- type `nano <filename>` at the prompt.
- You can edit text as you would in, e.g. MS Word.
- When you are finished, type ctrl-o to write, ctrl-x to exit. See commands at the bottom of the screen.
- How can we run the script?

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=(Gerardo Hidalgo)
$ echo ${name[0]}
Gerardo
$ echo $USER
gehi0941
```

Example 2

`local_vs_global.sh`

- Try to run it first (make sure it's executable)
- Take a look at the file
- Why does the last "echo" command print correctly?

- **Task: What do we need to set to make this script run as intended?**

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

E.g. creating a file with username in its name:

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

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
2018-10-09
```

Example 3

`hello_world.txt & hello.txt`

- Can we execute `hello_world.txt`?
- What is a command we could use to see the contents of `hello_world.txt`?

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 the Bash calculator tool (called `bc`)

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

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
```

Example 4

`test_for_file.sh`

- Task: Write a script (you can name it whatever you'd like) that tests if "calcsine.sh" is executable.

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
```

Example

`case_example.sh`

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.

Example

`while_example.sh`

`for_example.sh` & `for_example_2.sh`

- task: fix the depreciated syntax in `for_example.sh`

`dateloop_allbash.sh`

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.*

*behave differently with double quotes “”

Arguments II

```
#!/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>"
  exit 1
fi
```

Example

`calcsin.sh`

- Task: Create a script that prints out the first (non-file name) argument.
- Extra: Check that one (and only one) argument was passed; otherwise let the user know.

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.

Functions II

```
#/bin/bash

# function e
e () {
  echo $1;
}

#now test e
e Hello
e World
```

Example

`function.sh`

- Task: Modify “function.sh” to echo two arguments passed into the script.

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!

Gerardo Hidalgo-Cuellar
https://www.colorado.edu/rc

Survey:
http://tinyurl.com/curc-survey18

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