

# Introduction to Linux Scripting

Brett Milash and Wim Cardoen  
CHPC User Services

# Getting the exercise files

- For today's exercises open a session to `linuxclass.chpc.utah.edu` and run the following commands:

```
wget https://home.chpc.utah.edu/~u0424091/LinuxScripting2.tar.gz
```

```
tar xvfz LinuxScripting2.tar.gz
```

```
cd LinuxScripting2
```

# What is a shell script?

- A script is a series of shell commands stored in a file
- A script can be executed in several ways:
  - `bash scriptname.sh`
  - `./scriptname.sh` (if the script file executable)
  - `scriptname.sh` (if the script is on your PATH environment variable)
- commands are separated by:
  - new line
  - semi colon “;”
- Commands executed sequentially until
  - the end of the file has been reached
  - an error happens
  - the “exit” command is executed

# Scenarios for scripting

- Using the batch system at CHPC (discussed in the talk on Slurm Basics)
- Automating pre- and post- processing of datasets
- Performing lots of menial, soul draining tasks efficiently and quickly

# Exercise 1: Write a first script

Create a file named `ex1.sh` using `nano`.

First line always contains `#!/` followed by the language interpreter.

Put the following content in a file:

```
#!/bin/bash
echo " My first script:"
echo " My userid is:"
whoami
echo " I am in the directory:"
pwd
echo "Today's date:"
date
echo " End of my first script."
```

Make the script executable. Run this command:

```
chmod u+x ./ex1.sh
```

Now run your script:

```
./ex1.sh
```

# Script Arguments

Command line arguments to a script are available in the script as \$1, \$2, and so on.

For example, if a script is named “myscript.sh” and the script is executed with “./myscript.sh value1 value2 value3”:

- the variable \$1 has the value “value1”
- the variable \$2 has the value “value2”
- the variable \$3 has the value “value3”
- \$0 contains the name of the script
- \$# contains the # arguments

# Saving results of a command

- The output of a command can be put directly into a variable with the backtick: `
- The backtick is not the same as a single quote:

Backtick: `    Single quote: ‘

- For example:

```
VAR=`wc -l $FILENAME`
```

- You can also do this:

```
VAR=$(wc -l $FILENAME)
```

# String replacement

A neat trick for changing the name of your output file is to use string replacement to mangle the filename.

```
#!/bin/bash
IN="myfile.in"
#changes myfile.in to myfile.out
OUT=${IN/.in/.out}
./program < $IN > $OUT
```

- In bash, `${VAR/search/replace}` is all that is needed.
- You can use the `sed`, `awk`, or `tr` commands for more powerful manipulations.



# Exercise 2.0

Write a script that takes a file name as an argument, searches that file for exclamation points with `grep`, puts all the lines with exclamation points into a new file named "outfile", and then counts the number of lines in outfile. Use "histan-qe.out" as your test file.

Don't forget `#!/bin/bash`

Variables - Bash style: `VAR="string"` (no spaces!)

Arguments - `$1 $2 $3 ...`

Grep - `grep 'string' filename`

Counting Lines - `wc -l filename`

# Solution to Exercise 2.0

```
#!/bin/bash
INPUT=$1
grep '!' $INPUT > outfile
cat outfile | wc -l
```

The output from your script should have been “34”.

# Dates and Times

- Date strings are easy to generate in Linux
  - “date” command gives the date, but not nicely formatted for filenames
  - “date –help” will give format options (use +)
- `date +'Today is: %D'`      “Today is 05/31/18”
- `date +%r`      ”10:51:17 AM”
- `date +%Y-%m-%d_%k-%M-%S_%N`  
    “2014-09-15\_17-27-32\_864468693”

# Exercise 2.1

Modify your previous script so that instead of writing to an output file with a fixed name, the output filename is derived from the input file (e.g., ‘file.out’ becomes ‘file.todays\_date’). Don’t forget to copy your script in case you make a mistake!

Command execution to string - **VAR=`command`** (use the backtick)

Bash replacement – **\${VAR/search/replace}**

Dates - **date +%Y-%m-%d\_%k-%M-%S\_%N** (or pick your own format)

# Solution to Exercise 2.1

```
#!/bin/bash
INPUT=$1
DATE=`date +%Y-%m-%d_%k-%M-%S_%N`
OUT=${INPUT/out/}$DATE
grep '!' $INPUT > $OUT
wc -l $OUT
```

Every time you run the script, a new unique output file should have been generated.

# Conditionals (If statements)

```
#!/bin/bash
VAR1="name"
VAR2="notname"
if [ "$VAR1" == "$VAR2" ]
then
    echo "VAR1 and VAR2 have the same value."
else
    echo "VAR1 and VAR2 have different values."
fi
if [ -d $VAR1 ]
then
    echo "$VAR1 is a directory!"
else
    echo "$VAR1 is not a directory!"
fi
```

- The operators =, !=, &&, ||, <, > and a few others work.
- The “else” clause is optional.
- You can test variable values and file properties.
- See the manual page with “man test” for all the options.

# Conditionals (File properties)

Test	bash
Is a directory	-d
If file exists	-a, -e
Is a regular file (like .txt)	-f
Readable	-r
Writable	-w
Executable	-x
Is owned by user	-O
Is owned by group	-G
Is a symbolic link	-h, -L
If the string given is zero length	-z
If the string is length is non-zero	-n

- The last two flags are useful for determining if an environment variable exists.
- The rwx flags only apply to the user who is running the test.

# Loops (for statements)

```
#!/bin/bash
for i in 1 2 3 4 5
do
    echo $i
done
for i in *.in
do
    touch ${i/.in/.out}
done
for i in `cat files`
do
    grep "string" $i >> list
done
```

- Loops can be executed in a script --or-- on the command line.
- All loops respond to the wildcard operators \*, ?, [a-z], and {1,2}
- The output of a command can be used as a for loop input.
- There are also while and until loops.



# Exercise 2.2

Run the script called ex2.sh. This will generate a directory "ex2" with 100 directories and folders with different permissions. Write a script that examines all the directories and files in "ex2" using conditionals and for loops. For each iteration of the loop:

1. Test if the item is a directory. If it is, delete it.
2. If the file is not a directory, check to see if it is executable.
  - A. If it is, then change the permissions so the file is not executable.
  - B. If the file is not executable, change it so that it is executable and rename it so that it has a ".script" extension.
3. After all the files have been modified, execute all the scripts in the directory.

For loops : **for VAR in \*; do ... done**

If statements : **if [ condition ]; then ... elif ... else ... fi**

Useful property flags - **-x** for executable, **-d** for directory

-You can reset the directory by re-running the script ex2.sh

**-Make sure that you do not write your script in the ex2 directory, or it will be deleted!**

# Solution to Exercise 2.2

```
#!/bin/bash
for i in ex2/*; do
  if [[ -d $i ]]; then
    rm -rf $i
  else
    if [[ -x $i ]]; then
      chmod u-x $i
    else
      chmod u+x $i
      mv $i $i.script
    fi
  fi
done
for i in ex2/*.script; do
  ./$i
done
```

# Basic Arithmetic

```
#!/bin/bash
#initialization
i=1
#increment
i=$(( i++ ))
#addition, subtraction
i=$(( i + 2 - 1 ))
#multiplication, division
i=$(( i * 10 / 3 ))
#modulus
i=$(( i % 10 ))
#not math, echo returns "i+1"
i=i+1
```

- Bash uses `=$(( ))` for arithmetic operations.
- Important! This only works for integer math. If you need more, use python, R, etc.

# Bash “Strict” Mode

- Some bash settings simplify debugging:

```
set -e          # Exit immediately on any error
set -u          # Error if referencing undefined variable
set -o pipefail # Error on any pipe command
```

```
# Example: this code should fail:
```

```
pattern="somestring $some_undefined_variable"
grep $pattern non_existent_file | wc -l
```

- You can do this all at once:

```
set -euo pipefail
```

- See Aaron Maxwell’s blog:

- <http://redsymbol.net/articles/unofficial-bash-strict-mode/>

- Also helpful is “bash -x yoursript.sh” or “set -x”: prints each line before execution

# More advanced scripting techniques

- Locating files with “find”
- Create functions with “function”
- Redirect the standard error with “2>”
- Run commands in background with “&”
- Persistent background tasks using “nohup”
- Persistent login sessions using “screen”
- Run commands in parallel with “xargs”
- Catch signals with “trap”

# Questions?

Email: [brett.mail@utah.edu](mailto:brett.mail@utah.edu)

[wim.cardoen@utah.edu](mailto:wim.cardoen@utah.edu)

[helpdesk@chpc.utah.edu](mailto:helpdesk@chpc.utah.edu)