Introduction to Linux – Part 3

Anita Orendt and Martin Cuma

Center for High Performance Computing

What is a script?

- A script is a collection of linux commands that:
  - are stored in a file
  - the file **MUST** be executable
  - commands are separated by:
    - either being a carriage return (new line)
    - or separated by the semi colon (“;”)
  - executed sequentially until
    - the end of the file has been reached
    - or an error is met

Why scripting?

Scripting is a timesaver

The real question: When should you script?

Scenarios for scripting

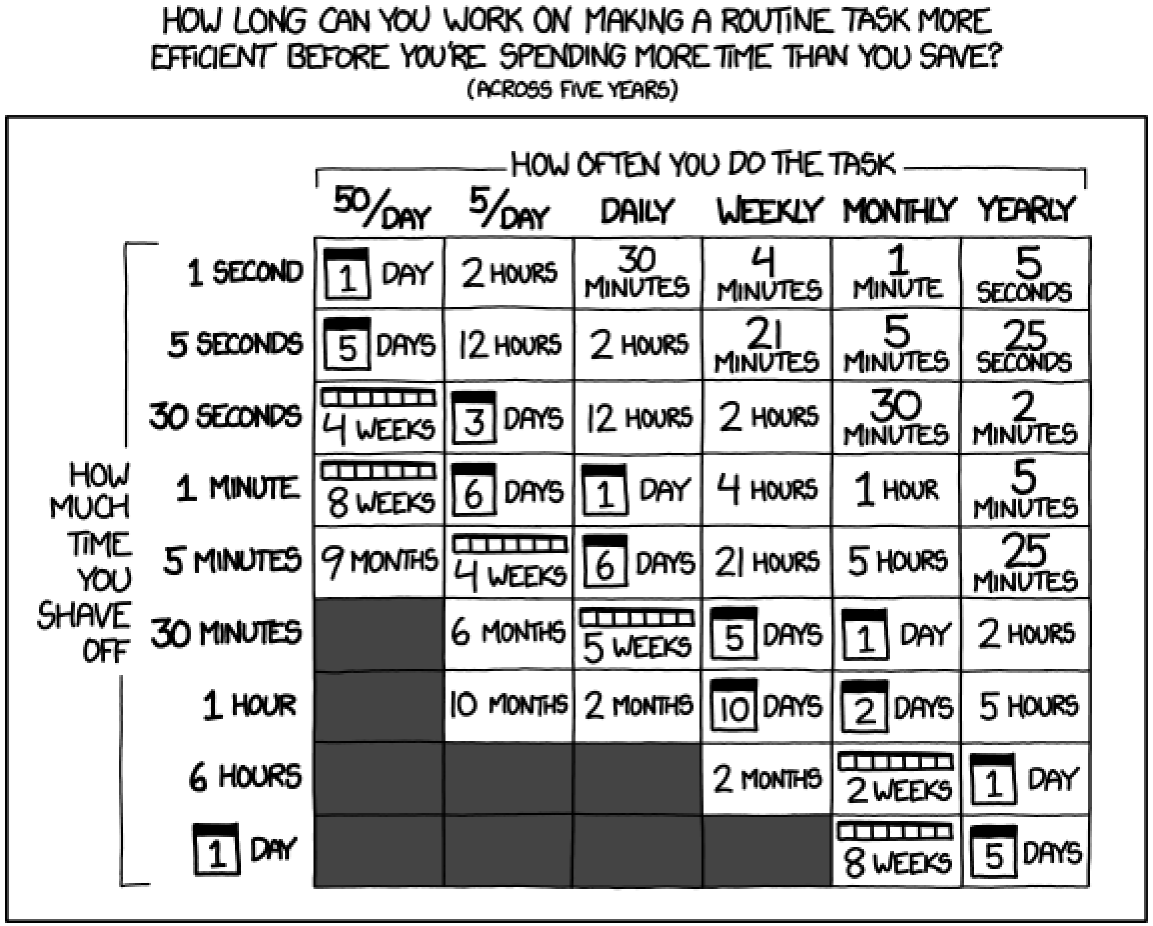
- Using the batch system at CHPC (discussed

in the talk on Slurm Basics)

- Automating pre- and post- processing of datasets

- Performing lots of menial, soul draining tasks efficiently and quickly (like building input files)

How long should you script?



HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

		HOW OFTEN YOU DO THE TASK					
		50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
HOW MUCH TIME YOU SHAVE OFF	1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
	5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
	30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
	1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
	5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
	30 MINUTES	■	6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS
	1 HOUR	■	10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS
	6 HOURS	■	■	■	2 MONTHS	2 WEEKS	1 DAY
	1 DAY	■	■	■	■	8 WEEKS	5 DAYS

http://xkcd.com/1205/

Task time saver calculator: http://c.albert-thompson.com/xkcd/

What to script in?

- Basic scripting needs can be done in the bash shell or the tcsh/csh shell.
- If you have more complicated tasks to perform, then you should consider something more advanced (like python* or matlab).
- If your workload is computationally heavy, you should be consider to write your application in a compiled language (e.g. C/C++, Fortran, …).

*CHPC also holds a three part workshop focusing on Python

bash vs tcsh/csh

- A Shell is:
  - a. user interface to the OS’s services
  - b. a layer (=> shell) around the kernel
  - c. programming env.
- CHPC currently supports 2 types of “shell-languages”/shells:
  - a. B(ourne) Again Shell (bash)
  - b. Csh/Tcsh shelll
- Syntactic differences are significant (and quirky) => **NO MIXING ALLOWED**
- Some programs do not support different shells (rather rare)
- Very easy to switch between shells
- What shell do I currently use? ***echo $SHELL***

WHILE LEARNING TO SCRIPT,

PICK ONE AND STICK WITH IT

For this training we will be using bash

Getting the exercise files

- For today’s exercises, open a session and when in your home directory run:

```
cp  ~u0028729/IntroLinux3.tar
tar  -xvf  IntroLinux3.tar
cd IntroLinux3/
```

A comment about running scripts/programs

- Last time we ran script by
  - bash scriptname (bash goostats.sh)
  - This works if the script is in the current directory and is written in bash syntax
  - Can also use first line of script to tell OS what language/interpreter to use on script
- When you execute a command, the shell must first find the program you want to run
- Either:
  - if you are in directory can use ./ to tell shell (e.g. "./ex1.sh")
- Otherwise can provide full path when executing script
- OR put directory where shell is located in the PATH environment variable
  - **echo $PATH**
- "**`which`**" command: shows where a command is found

Write a first script (ex1)

- Open a file named ex1.sh using nano
- Note -- ‘#’ character at start of line – indicates line is a comment
- Top line always contains the ‘she-bang’ followed by the language interpreter:

  ‘#!/bin/bash’ (if script is in bash syntax)
- Put the following content in a file:

```
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 + execute:

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

Script Arguments (refresher)

- If the script is named “myscript.sh” the script
  - is executed with “myscript.sh myarg1 myarg2   ...   myargN”

$0 returns the name of the script

$1 returns the first argument

$2 returns the second argument

.

.

$N returns the Nth argument

Using grep and wc (refresher)

- grep searches files for test strings and outputs lines that contain the string
  - VERY fast, very easy way to parse output
  - can use regex and file patterns

```
grep "string" filename
```

use quotes if any special characters (spaces, @, !)

- wc can count the number of lines in a file

```
wc -l filename
```

Command line redirection (refresher)

- You can output to a file using the “>” operator.

```
cat filename > outputfile
```

- You can append to the end of a file using “>>”

```
cat filename >> outputfile
```

- You can redirect to another program with “|”

```
cat filename | wc –l
```

Exercise 2

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

Don’t forget **`#!/bin/bash`**

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

Grep - **`grep 'string' filename`**

Counting Lines - **`wc –l filename`**

Solution to Exercise 2

```
#!/bin/bash
grep '!' $1 > outfile
wc -l outfile
```

Run as ./ex2.sh The output from your script should have been “34 outfile”.

Setting and Using Variables

```
#!/bin/bash
#set a local variable (no spaces around =)
VAR="hello bash!"
#print the variable
echo $VAR

#make it permanent  “global”
export VAR2="string"
#print the variable
echo $VAR2

#remove VAR2
unset VAR2
```

Be careful what you export! Don’t overwrite something important!

Commands to string

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

` '

- Bash form: `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}
echo $IN
echo $OUT
#run program that takes the
./program < $IN > $OUT
```

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

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 +)
- A nice formatted string format (ns resolution) – do “man date” to get explanation of:

```
date +%Y-%m-%d_%k-%M-%S_%N
"2014-09-15_17-27-32_864468693"
```

- For a really unique string, you can use the following command to get a more or less unique string (not recommended for cryptographic purposes)

```
$(cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 32 | head -n 1)
```

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.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
DATE=`date +%Y-%m-%d_%k-%M-%S_%N`
OUT=${1/out/}$DATE
echo $OUT
grep ‘!’ $1 > $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 "True"
else
  echo "False"
fi
if [[ -d $VAR1 ]]; then
  echo "Directory!
fi
```

- The operators ==, !=, &&, ||, <, > and a few others work.
- You can use if statements to test two strings, or test file properties.

Conditionals (File properties)

Test	bash
Is a directory	-d
If file exists	-a,-e
Is a regular file (like .txt)	-f
Readable	-r
Writeable	-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/do/done statements) - refresh

```
#!/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.

Exercise 2.2

Run the script called ex2prep.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 - Bash : **for VAR in *; do ... done**

If statements - Bash : **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
```