Introduction to Linux Scripting

Albert Lund

CHPC User Services

Overview

- What is scripting?

- Compiling mini-excercise

- Basic bash/tcsh scripting exercises

Slides: home.chpc.utah.edu/~u0403692/IntroScripting.pdf

vi Refresher/Exercise

- A few commands will get you started:
  - Press 'i' for insert! (Insert mode, Replace mode)
  - Press 'Esc' to get back to command mode!

  - :w - 'write'
  - :wq! - 'write and quit'
  - :q! - 'quit without saving (good for mistakes)
  - Press 'u' to undo in command mode
- Exercise: write something in vi and save it!
  - Try it with 'vim' too

Why script?

Scripting is a timesaver

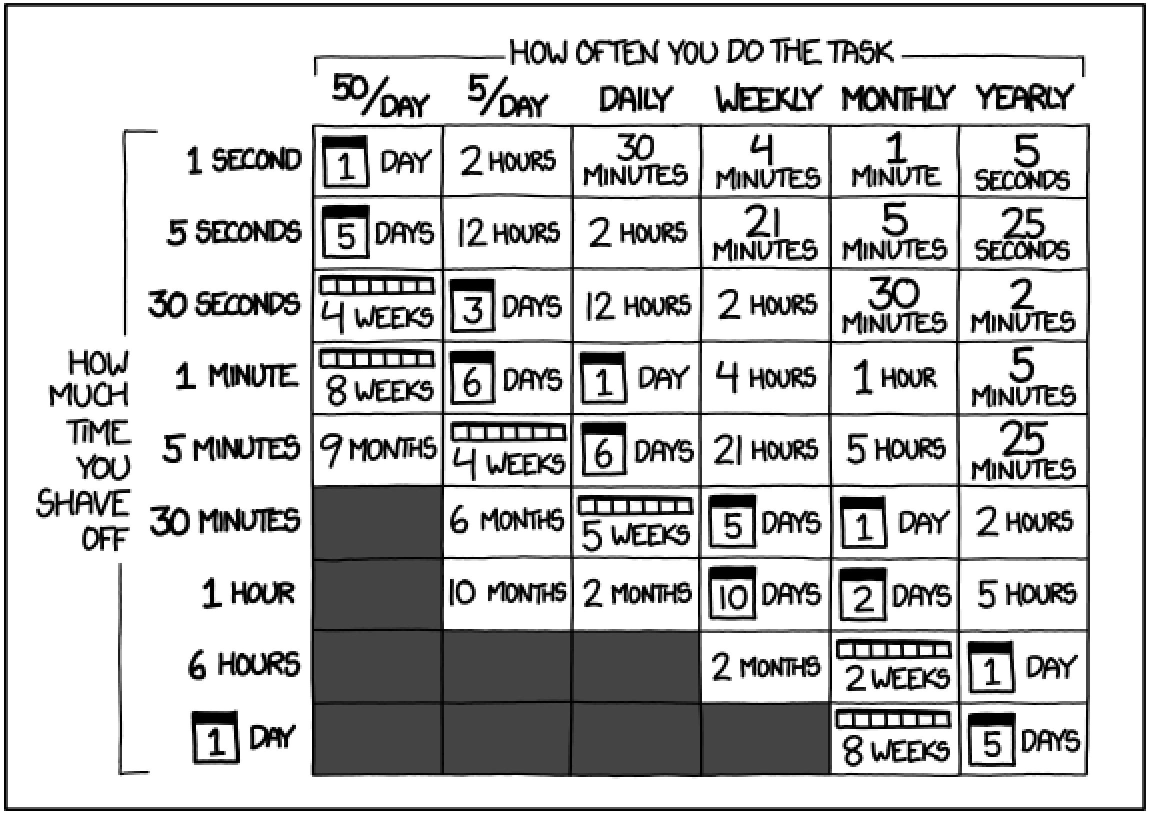
The real question: When should you script?

Scenarios for scripting

- Using the batch system at CHPC
- Automating pre- and post- processing of datasets
- Performing lots of repeated, menial, soul draining tasks efficiently and quickly

How long should you spend writing a 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					
HOW MUCH TIME YOU SHAVE OFF	50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
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/

Don't script when it doesn't save you time!

What to script in?

- Most scripting needs can be covered by bash or tcsh.
- If you have more complicated analyses to perform, then you should consider something more advanced (like python* or matlab).
- If your workload is very computation heavy, you should be considering an application written in C/C++ or Fortran (not scripting).

*CHPC will hold a workshop in the fall on Python

bash vs tcsh

- Syntactic differences are significant (and quirky)
- Some programs do not support different shells
- Very easy to switch between shells
- You can write shell scripts in any language regardless of your default shell.

WHILE LEARNING TO SCRIPT, PICK ONE AND STICK WITH IT.

How to change your default shell on CHPC systems

- You can see what your default shell is using “echo $SHELL” when logged into CHPC systems.
- To change your default shell: go to chpc.utah.edu, click “Sign In” in the upper right, and login with your U of U credentials. You will be presented with your profile, which will have a link “Edit Account Settings”. A new dialogue will show, and you will see an option to change shell. Change it to whatever you want, and save it. Changes will go through in about 15 minutes.
- (Also can be used to change your email on record, please do this if you change email addresses.)

Mini-Exercise: Compiling

- Download and compile numbertools:

```
wget  chpc.utah.edu/~u0403692/numbertools.tar.gz
tar  -xzf  numbertools.tar.gz
cd numbertools/
make all
```

- Try running each of the programs:

square 4.0 - area of a square with sides 4.0

circle 4.0 - area of a circle with radius 4.0

prime <n> - determines if an integer <n> is prime

randgen <n> - generates <n> random integers (up to 10^6)

What is a script?

- A script is a set of linux commands condensed into a single text file.
- When a script is executed, the commands in the script are executed sequentially, as if they were being typed into the command line.
- Commands are separated by a carriage return (enter key) or a semicolon (;).

Scripting Basics - # and #!

- # is the character that starts a comment in many, many languages (many).
  - Comments can still do stuff (#!, #SLURM)
- #!/bin/bash --or-- #!/bin/tcsh can be used to indicate what program should run the script
  - you can put any program (/path/program), but the script language should match the program, otherwise weird things will happen
  - use "chmod u+x script" to enable the execute bit on a script

Setting and Using Variables

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

#make it permanent
export VAR2="string"
echo $VAR2

#remove VAR2
unset VAR2
```

```tcsh
#!/bin/tcsh
#set a variable
set VAR = "hello tcsh!"
#print the variable
echo $VAR

#make it permanent (no =)
setenv VAR2 "string"
echo $VAR2

#remove VAR2
unset VAR2
```

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

Mini Exercise: Echo command

- The echo command prints a string or variable to the command line:
  - echo "Hello World" writes Hello World to standard output
  - bash> HELLO="hello world"; echo $HELLO
  - tcsh> set HELLO="hello world"; echo $HELLO
  - beware the difference between double and single quotes! (variables do not expand in single quotes)

Exercise 1

- Write a script from scratch where you pick a number, assign it to a variable, and then run square, circle, and prime on it.
- Run the script from a different directory than the numbertools directory. Set a variable to the path of the numbertools directory and use that to run each program (e.g., $BINDIR/square)
- Use the echo command to the script output (so that you know what output came from which program)

Don't forget **`#!/bin/bash`** or **`#!/bin/tcsh`**

`Make sure to run "chmod u+x" on your script!`

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

Tcsh style: **`set VAR = “string”`**

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

Solution to Exercise 1

```
#!/bin/bash
NUMBER="4"
BINDIR="/path/numbertools/"

echo "Running programs..."
echo "Number:"$NUMBER
echo "Square area"
$BINDIR/square $NUMBER
echo "Circle area"
$BINDIR/circle $NUMBER
echo "Is it prime?"
$BINDIR/prime $NUMBER
```

```
#!/bin/tcsh
set NUMBER = 4
set BINDIR = /path/numbertools

echo "Running programs..."
echo "Number:"$NUMBER
echo "Square area"
$BINDIR/square $NUMBER
echo "Circle area"
$BINDIR/circle $NUMBER
echo "Is it prime?"
$BINDIR/prime $NUMBER
```

Script Arguments

```
#!/bin/bash
ARG1=$1
ARG2=$2
#ARG3=$3, and so on
echo $ARG1
echo $ARG2
```

```
#!/bin/tcsh
set ARG1 = $1
set ARG2 = $2
#set ARG3 = $3, so on
echo $ARG1
echo $ARG2
```

If the script is named “myscript.sh” (or “myscript.csh”), the script is executed with “myscript.sh myarg1 myarg2   …   myargN”

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`"``
- Tcsh form: ``set VAR="`wc -l $FILENAME`"``

Dates and Times

- Date strings are easy to generate in Linux
  - The “date” command gives the date, but not nicely formatted for filenames
  - "date --help" will give format options (use +)
- A nicely formatted string format:

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

- 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

Modify the script you wrote in Exercise 1 so that the number is assigned from a script argument, and the output is written to a file that is dated. Use the date command in combination with backticks to create a filename.

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

Dates - **date +%Y-%m-%d_%k-%M-%S** (or pick your own format)

Command 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
```

Solution to Exercise 2

```bash
#!/bin/bash
NUMBER=$1
DATE=`date +%Y-%m-%d_%k-%M-%S`
FILENAME="myfile-$DATE"

BINDIR="/path/numbertools/"

echo "Running programs..."
echo "Number:"$NUMBER >> $FILENAME
echo "Square area" >> $FILENAME
$BINDIR/square $NUMBER >> $FILENAME
echo "Circle area" >> $FILENAME
$BINDIR/circle $NUMBER >> $FILENAME
echo "Is it prime?" >> $FILENAME
$BINDIR/prime $NUMBER >> $FILENAME
```

```tcsh
#!/bin/tcsh
set NUMBER = $1
set DATE = "`date +%Y-%m-%d_%k-%M-%S`"
set FILENAME="myfile-$DATE"

set BINDIR="/path/numbertools/"

echo "Running programs..."
echo "Number:"$NUMBER >> $FILENAME
echo "Square area" >> $FILENAME
$BINDIR/square $NUMBER >> $FILENAME
echo "Circle area" >> $FILENAME
$BINDIR/circle $NUMBER >> $FILENAME
echo "Is it prime?" >> $FILENAME
$BINDIR/prime $NUMBER >> $FILENAME
```

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 $VAR ]]; then
  echo "Directory!
fi
```

```
#!/bin/tcsh
set VAR1="name"
set VAR2="notname"
if ($VAR1 == $VAR2) then
  echo "True"
else
  echo "False"
endif
if ( -d $VAR ) then
  echo "Directory!"
endif
```

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

-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/foreach 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
```

```
#!/bin/tcsh
foreach i (1 2 3 4 5)
  echo $i
end
foreach i ( *.in )
  touch "$i:gas/.in/.out/"
end
foreach i ( `cat files` )
  grep "string" $i >> list
end
```

- 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 3

- Write a new script that uses randgen and prime to determine if a random list of integers is prime or not. Use a combination of a for loop and an if statement.
- Write all of the prime numbers into one file, and non-prime numbers into the other. Do this for a list of at least 300 integers.
- Prime will always output "IsPrime" if the number is prime

For loops - Bash : **for VAR in `command`; do ... done**
Tcsh : **foreach VAR (`command`) ... end**

If statements - Bash : **if [[condition]]; then ... else ... elif ... fi**
Tcsh : **if (condition) then ... else ... else if ... endif**

Solution to Exercise 3

```
#!/bin/bash
COUNT=300
BINDIR=/path/numbertools

for i in `$BINDIR/randgen $COUNT`; do
 RESULT=`$BINDIR/prime $i`
 if [[ $RESULT == "IsPrime" ]]; then
   echo $i >> primes
 else
   echo $i >> notprimes
 fi
done
```

```
#!/bin/tcsh
set COUNT=300
set BINDIR=/path/numbertools

foreach i (`$BINDIR/randgen $COUNT`)
 set RESULT="`$BINDIR/prime $i`"
 if ( $RESULT == "IsPrime" ) then
   echo $i >> primes
 else
   echo $i >> notprimes
 endif
end
```

End of day 3!

Questions?

Email issues@chpc.utah.edu

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
```

```
#!/bin/tcsh
set IN = “myfile.in”
#changes myfile.in to myfile.out
set OUT=“$IN:gas/.in/.out/”
./program < $IN > $OUT
```

- In tcsh the ‘gas’ in `“$VAR:gas/search/replace/”` means to search and replace all instances (“global all substrings”); there are other options (use “man tcsh”).
- In bash, `${VAR/search/replace}` is all that is needed.
- You can use 'sed' or 'awk' for more powerful manipulations.