Introduction to Linux Part 2: Scripting and Compiling

Martin Čuma, Wim Cardoen
CHPC User Services

Overview

- Basic bash/tcsh scripting exercises
- Cluster jobs and submission on CHPC
- Compiling and linking software

Why script?

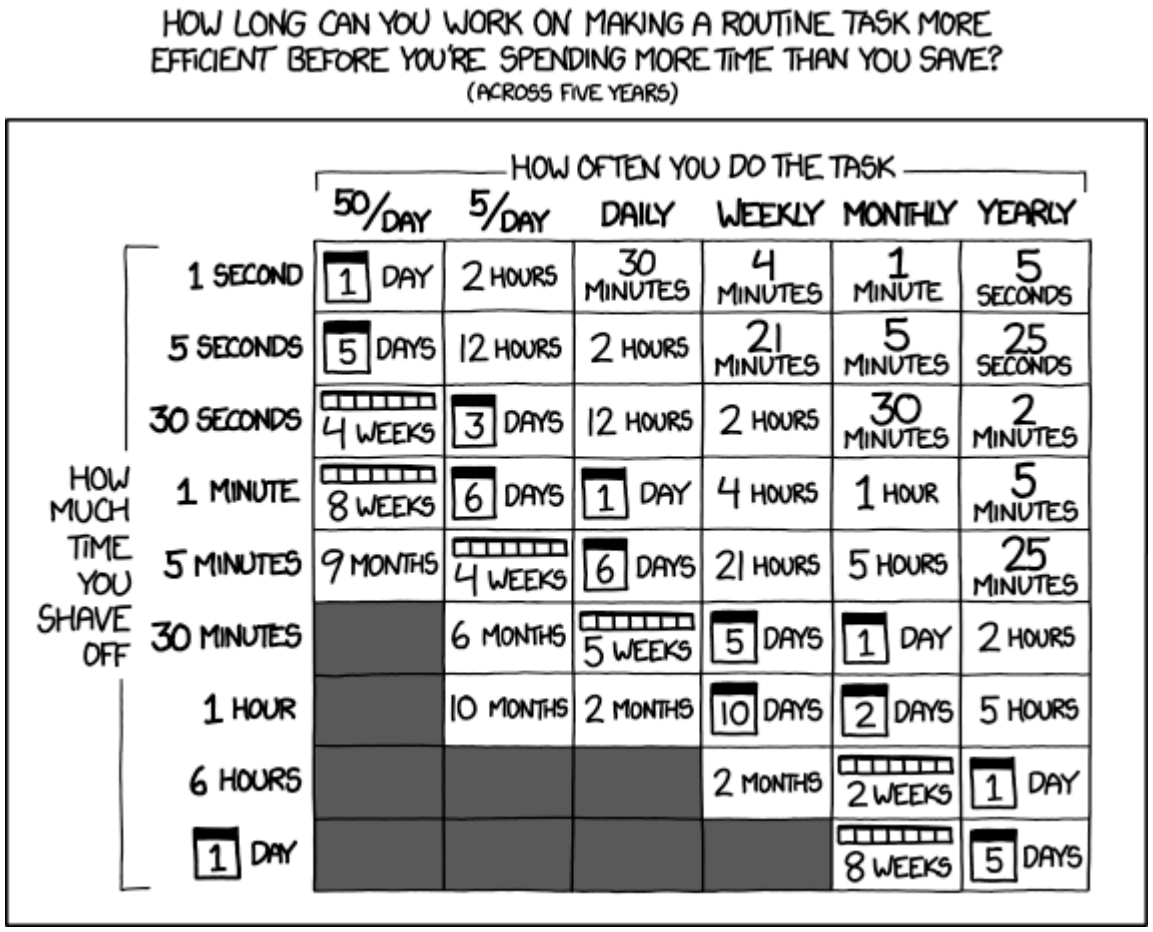
Scripting is a timesaver

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 menial, soul draining tasks efficiently and quickly (like building input files)

How long should you script?



http://xkcd.com/1205/

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

What to script in?

- Basic 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 a application written in C/C++ or Fortran (not scripting).

*CHPC will hold a three part workshop in mid February focusing on Python

bash vs tcsh

- Syntactic differences are significant (and quirky)
- Some programs do not support different shells
- Very easy to switch between shells

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

How to change your default shell

- 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 and login with your U of U credentials. You will be presented with your profile, which will have a link “Edit Profile”. 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.)

Getting the exercise files

- For the exercises today, open a session to one of the cluster interactives and run the following commands:

wget /uufs/chpc.utah.edu/~u0101881/talks/LinuxScripting.tar.gz
tar -xzf LinuxScripting.tar.gz
cd LinuxScripting/

- The slides are at:

https://www.chpc.utah.edu/presentations/images-and-pdfs/IntroScriptingJun2016.pdf

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 (;).

The Basic Script

1. Set up the #SLURM directives for the scheduler
2. Set up the working environment by loading appropriate packages in order
3. Add any additional libraries or programs to $PATH and $LD_LIBRARY_PATH
4. Set up temporary directories if needed
5. Switch to the working directory
6. Run the program with your input
7. Clean up any temporary files or directories

Scripting Basics - # and #!

- # is the character that starts a comment in many, many languages (many).
  - Comments can still do stuff (#!, #PBS)
- #!/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 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

```
#!/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
```

```
#!/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!

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”

Using grep and wc

- 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
  - use backslash (\) to search for special characters (e.g. to search for "!" use "\!")

```
grep "string" filename
```

- 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 1

Write a 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`** or **`#!/bin/tcsh`**

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

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

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

Counting Lines - **`wc –l filename`**

Solution to Exercise 1

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

```
#!/bin/tcsh
set INPUT = $1
grep '\!' $INPUT > outfile
wc -l outfile
```

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

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`"`

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.

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):

```
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

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}**

Tcsh replacement – **“$VAR:gas/search/replace/”**

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

Solution to Exercise 2

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

```
#!/bin/tcsh
set INPUT = $1
set DATE = "`date +%Y-%m-%d_%k-%M-%S_%N`"
set OUT = “$INPUT:gas/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 "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

Run the script called exercise4.sh. This will generate a directory "exercise4" with 100 directories and folders with different permissions. Write a script that examines all the directories and files in "exercise4" 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**
Tcsh : **foreach VAR (*) ... end**

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

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

-You can reset the directory by re-running the script exercise4.sh
-Make sure that you do not write your script in the exercise4 directory, or it will be deleted!

Solution to Exercise 3

```
#!/bin/bash
for i in exercise4/*; do
  if [[ -d $i ]]; then
    rm -rf $i
  else
    if [[ -x $i ]]; then
      chmod u-x $i
    else
      chmod u+x $i
      mv $i $i.script
    fi
  fi
done
for i in exercise4/*.script; do
  ./$i
done
```

```
#!/bin/tcsh
foreach i ( exercise4/* )
  if ( -d $i ) then
    rm -rf $i
  else
    if ( -x $i ) then
      chmod u-x $i
    else
      chmod u+x $i
      mv $i $i.script
    endif
  endif
end
foreach i ( exercise4/*.script )
  ./$i
end
```

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
```

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

- Bash uses **$(( ))**, whereas tcsh uses **@**
- Important! This only works for integer math. If you need more, use python.

The Basic SLURM Script – batchscript.sh

```
#! /bin/bash
#SBATCH -A account
#SBATCH –p partition
#SBATCH –N 2
#SBATCH –n 24
#SBATCH –t 1:00:00

#Set up whatever program we need to run with
module load myprogram

#set up the temporary directory
TMPDIR=/scratch/local/u0123456/data
mkdir -P $TMPDIR

#Set up the path to the working directory
WORKDIR=/uufs/chpc.utah.edu/common/home/u0123456/data
cd $WORKDIR

#Run the program with our input
myprogram < $WORKDIR/input > $WORKDIR/output

rm -rf $TMPDIR
```

The Basic SLURM Script – batchscript.csh

```
#! /bin/tcsh
#SBATCH -A account
#SBATCH -p partition
#SBATCH -N 2
#SBATCH -n 24
#SBATCH -t 1:00:00

#Set up whatever program we need to run with
module load myprogram

#set up the temporary directory
set TMPDIR="/scratch/local/u0123456/data"
mkdir -P $TMPDIR

#Set up the path to the working directory
set WORKDIR="/uufs/chpc.utah.edu/common/home/u0123456/data"
cd $WORKDIR

#Run the program with our input
myprogram < $WORKDIR/input > $WORKDIR/output

rm -rf $TMPDIR
```

Exercise 4

Modify the script exercise1.sh or exercise1.csh and launch it from the command line using qsub:

1. Change the account name in #SLURM -A to owner-guest.
2. Change the partition in #SLURM -p to kingspeak-guest
3. Add the jobname (#SLURM –J <name>)
4. Change the walltime to ten minutes
5. Change the number of tasks to correct number on kingspeak (16, 20 or 24 core nodes on kingspeak)
6. Replace all instance of u0123456 with YOUR unid.

-Try launching both the .csh and .sh version of the script

-Use **squeue -u u0123456** to see your jobs in the queue. Make sure to put in your own UNID! (Also, u is for user.)

-You can use **scancel <jobnumber>** to delete your jobs if you make a mistake. You can find the job numbers by using squeue.

Solution to Exercise 4

- You should have an output file with a name like "slurm-123456.out".
- If you inspect the contents of the file, you should see the path of the temporary directory, a bunch of random strings, and a report of how many time the letter 'a' appears in the random string
- You may also see errors.

Compiling

- Many programs come as source code.
- To run such programs, the source code must be compiled and linked into an executable
- Source – most often a C or Fortran program in text file format
- Executable – binary form of program that can be executed

Compilers

- Compilers are system-specific, but, there are quite a few vendors (CHPC has all three):
- GNU: `gcc`, `g++`, `g77`, `gfortran` – open source, free
- Intel: `icc`, `icpc`, `ifort` – commercial but free for academia
- PGI: `pgcc`, `pgCC`, `pgf77`, `pgf90` – commercial

How to compile

- C program compilation (GNU)
  ```
  gcc hello.c -o helloC.x
  ```
- Fortran compilation
  ```
  g77 hello.f -o helloF.x
  ```
- Compiling to object
  ```
  g77 -c hello.f
  gcc -c hello.c
  ```
- Linking objects
  ```
  g77 hello.o -o helloF.x
  ```
- Compiling without -o generates "a.out"

Optimization and debugging

- The compiler can perform optimizations that improve performance.
  - common flags `-O3` (GNU), `-fast` (Intel), `-fastsse` (PGI)
  - Beware! -O3,etc can sometimes cause problems (solutions do not calculate properly)
- In order to debug program in debugger, symbolic information must be included
  - flag `-g`
  - The easiest debugging is to just add `printf` or `write` statements (like using echo)

Exercise 5

Run Switch to directory "exercise5". There are a few source files in this directory (which you saw on Tuesday). Compile these programs using the following steps:

1. Compile cpi_ser.c using gcc. Use -o to name the program something different from "a.out". Run the program when compiled and note how long it runs (the wall clock time).
2. Try to compile pi3_ser.f using g77. See what happens.
3. Compile ctimer.c with gcc as an object (.o) using the -c flag. Then hybrid compile ctimer.o and pi3_ser.f using g77.
4. Try compiling cpi_ser.c like you did in Step 1, but using the -O3 flag. Compare how long the program runs with and without -O3.

Compiling: **gcc hello.c -o hello.x**
g77 hello.f -f hello.x

Object compilation: **gcc -c hello.c**
Linking: **gcc hello.o -o hello.x**

Using optimization: **gcc -O3 hello.c -x helloFast.x**

Solutions to Exercise 5

1. Compiling a C program:
   gcc cpi_ser.c -o cpi_ser.x (Time: ~1.625 s)
2. Compiling a Fortran program:
   g77 pi3_ser.f -o pi3_ser.x -- Gives errors! (dependencies)
3. Compiling an object for linking, then linking:
   gcc -c ctimer.c (Creates ctimer.o)
   g77 ctimer.o pi3_ser.f -o pi3_ser.x (Time: ~2.449 s)
4. Compiling with -O3
   gcc -O3 cpi_ser.c -o cpi_ser.x (Time: ~0.709 s)
   g77 -O3 ctime.o pi3_ser.f -o pi3_ser.x (Time: ~0.724 s)

Compiling serious packages

- Some packages are far more complicated than one or two source files.
  - Many packages use gnu config/make
  - Others use cmake (useful for cross-platform)
  - Others of less repute
- You will almost certainly encounter a package like this if you continue in scientific computing
  - CHPC can help compile programs (can be hard) but knowing how to do it yourself is useful.

GNU config and make

- Configure: A scripting utility that checks for certain libraries and applications, as well as compiler capablities, and building makefiles.
  - Executed by the **./configure** script in the package directory.
  - You can use **./configure --prefix=<PATH>** to decide where to install the package, otherwise it will install in the same location as the package source.
- Make: Takes instructions from a makefile (a special script) to compile source in order to make a program.
  - As simple as executing **make** in a folder with a Makefile (or specifying the makefile with **-f**)
  - Sometime you need to use **make install** to finish the compilation process.

Exercise 6

You will download and compile the zlib library in this exercise. zlib is used for file compression by many programs.

1. Make a directory called "zlib" and cd to it.
2. Download and untar the zlib library with the following :
   - **wget http://zlib.net/zlib-1.2.8.tar.gz**
   - **tar -xzf zlib-1.2.8.tar.gz**
3. Configure zlib so that it installs in the current directory (zlib) and not the source directory (zlib-1.2.8).
   - **./configure --prefix=<PATH>** (change the path appropriately)
4. Compile using **make** and then **make install**.
5. Check to see if the library was installed properly in zlib/lib (the files libz.so, libz.a should exist).

Solutions for Exercise 6

1. ~~Make a directory called "zlib" and cd to it.~~
2. ~~Download and untar the zlib library with the following :~~
   - ~~**wget http://zlib.net/zlib-1.2.8.tar.gz**~~
   - ~~**tar -xzf zlib-1.2.8.tar.gz**~~
3. Configure zlib so that it installs in the current directory (zlib) and not the source directory (zlib-1.2.8).
   - **./configure --prefix=$HOME/zlib** (as an example)
4. Compile using **make** and then **make install**.
5. Check to see if the library was installed properly in zlib/lib (the files libz.so, libz.a should exist).

Questions?

Email issues@chpc.utah.edu