Express Introduction to Linux Scripting (Beginner Level)

Ritu Arora

February 11, 2016

Email: rauta@tacc.utexas.edu

Prerequisites & Objectives

- Prerequisites
  - Basic knowledge of Linux commands
  - "Express Linux Tutorial" from January 28, 2016

- Objectives
  - Writing a basic shell script
  - Using variables in shell scripts
  - Passing command-line arguments to shell scripts
  - Decision-Making in shell scripts
  - Loops in shell-scripts
  - Overview of writing a SLURM job-script for running applications in batch-mode on Stampede or Lonestar5

Linux Scripting

- Instead of typing commands directly in the shell, you can place the commands in a file, grant execute permissions to the file, and then run the file from the command prompt
  - Such a file that contains the Linux commands is known as a shell script

- Three commonly used types of scripts: Bourne shell, C shell, Korn shell

- We will work with Bourne shell script – the simplest one

Creating and Running a Simple Shell Script

- Create a file named `myScript.sh` and put the text below in it
  - You may use `vi` or `nano` or any other editor of your choice

```
#!/bin/sh
echo "hello world"
```

- Change the permissions on the file to 700 , or just use, “+x”

```
chmod 700 myScript.sh
chmod +x myScript.sh
```

- Execute the file by typing `./myScript.sh`

Comments in the Script

Except the first line in the script (indicating the shell to be used), any line beginning with a **#** character in the first column is taken to be a comment and is ignored

```
#!/bin/sh
# purpose: print current directory path and its contents
pwd
ls
```

Shell Variables (1)

- Shell script can contain variables – just like in C/C++/Fortran - and these variables are treated as strings
- Variables can be system-defined (*e.g.*, `$HOME`, and `$PATH`) or user-defined (`$name` see below)
- The script variables can be assigned values, manipulated, and used
  - Note that there should be no space before and after the assignment operator
  - Note how the variable is used in the example below by prefixing "`$`" sign

```
#!/bin/sh
#Below is the declaration of a variable
name="Ritu"
echo "The name is $name"
```

Shell Variables (2)

```
#!/bin/sh
clear
echo "Hello $USER"
echo "Today is  "; date
echo "Number of files in directory: " ; ls | wc -l
echo "Calendar is presented below"
cal
```

`clear` command will clear the terminal screen

`$USER` is system-defined variable

`date` command prints the current date along with the time – notice "`;`" for separating the echo command and data

`cal` command prints the calendar

Passing Arguments to Scripts

- The special variables **$1-$9** correspond to the arguments passed to the script when it is invoked
- Although the Bourne shell can have any number of parameters, the positional parameters (or variables) are limited to numbers 1 through 9

```
#!/bin/sh
echo "The name entered is
$1 $2"
```

```
login2$ ./myScript4.sh Ritu Arora
The name entered is Ritu Arora
```

Doing Arithmetic With Shell Scripts

```
login2$cat myScript5.sh

#!/bin/sh
sum=`expr $1 + $2`
echo "Sum is $sum"


login2$ ./myScript5.sh 3 5
Sum is 8
```

Type these lines in a file, save the file with the name myScript5.sh and run it as shown below

Note the usage of the **expr** command with back-quotes. It is used to evaluate an expression. Back-quotes are used to execute a command from inside another command

Conditionals in Shell Scripts

- The if-statement begins with the keyword **`if`**, and ends with the keyword **`fi`**
- The **`if`** keyword is followed by a condition, which is enclosed in square brackets, *e.g.*, `if [$1 > 5]`
- The line after the **`if`** keyword contains the keyword **`then`**
- You may include an **`else`** part if needed:

```
#!/bin/sh
if [ -d $1 ]
then
    ls $1
else
    cat $1
fi
```

The **`if`** condition checks if **`$1`** is a directory or not

Loops

```
#!/bin/sh
for i in `ls`; do
 echo $i
done
```

Loops are used to repeat steps.
Two kinds: `for` and `while`

```
#!/bin/sh
for (( i = 0; i <= 2; i++))
do
    for (( j = 0 ; j <= 4; j++ ))
    do
          echo  "hello test, i is $i and j is $j"
    done
  echo ""
done
```

Exercise-1

- `md5` is the command that can be used to find the checksums of a file, the usage syntax is as follows:

  `md5 <file-name>`

- Write a script that finds the checksum of the file whose name is provided as input

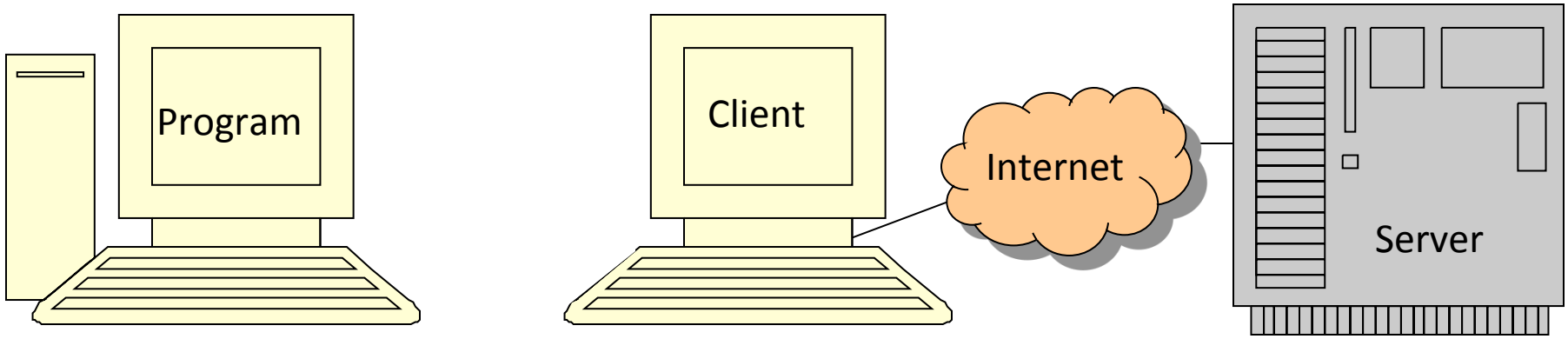
Exercise-2

- Find the checksum of all files in a directory
- The directory path should be passed as an argument to the script.

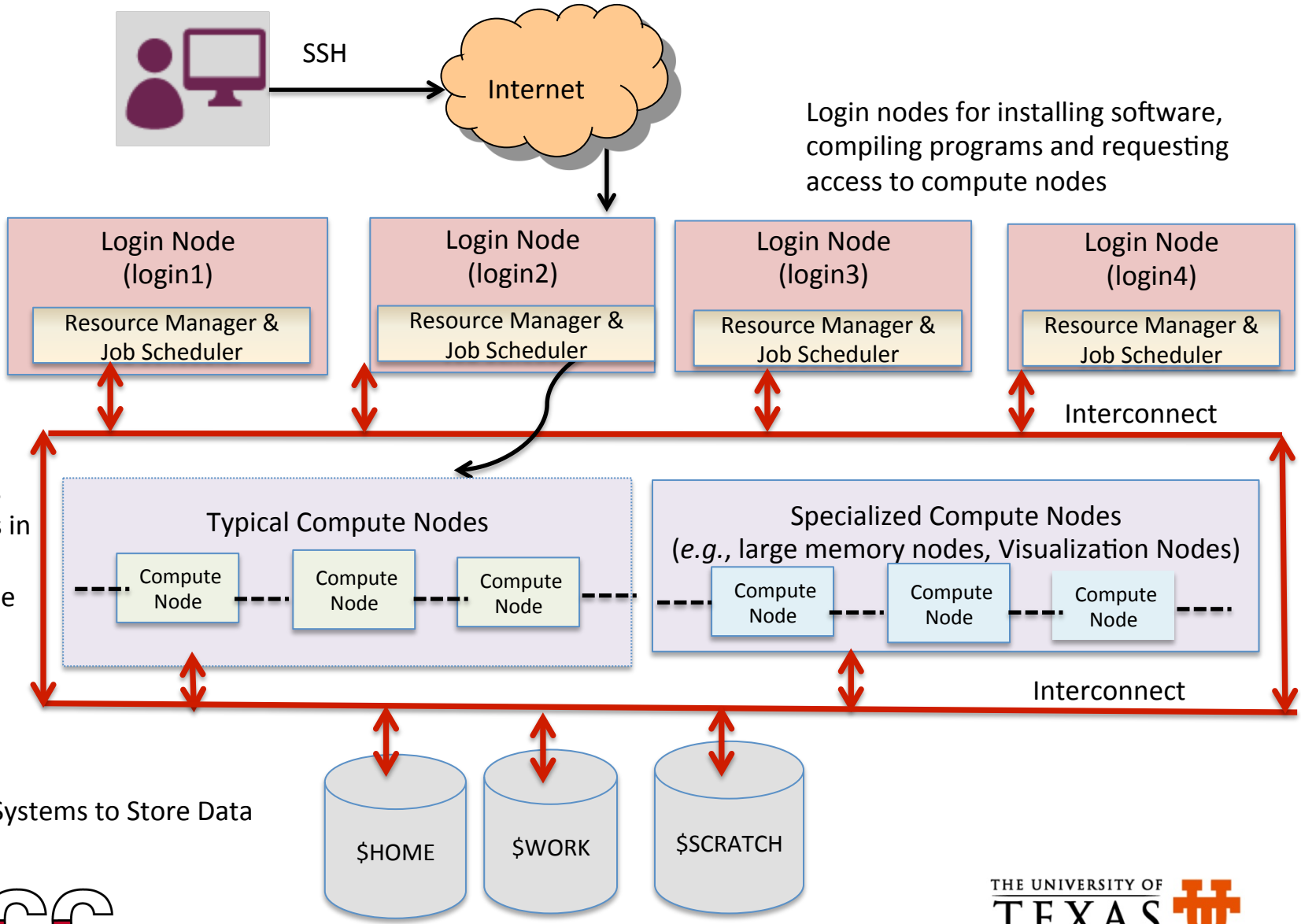
The following content is meant for providing basic information on working on a remote Linux system or supercomputing resources.

Local Access vs. Remote Access

- Local (Desktop/Laptop)
- Remote (Servers)



Working on a Remote Computational Resource like Stampede or Gordon (Oversimplified Diagram)



For Connecting to Remote Servers

- For secure (encrypted) communication, including data transfer across networks, you need an SSH client
  - **SSH Secure Shell Client, Putty, GSI-OpenSSH, OpenSSH**

- If you are an XSEDE user, you can also use the XSEDE Single Sign-On (SSO) login hub – this is through the SSH client

- Next few slides show how to use SSH client and SSO from a Windows or Mac computer

How to access Linux systems remotely from a Windows machine?

- Using client programs on Windows machines
  - SSH Secure Shell Client
    https://shareware.unc.edu/
  - PuTTY http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
- Other options:
  - Install Linux on a USB stick: http://www.pendrivelinux.com/
  - Use Cygwin/VM Ware (runs as a windows process)
- Video showing the usage of SSH secure shell client
  https://www.youtube.com/watch?v=cigMNqXIkRE

For Mac Users

- You can have remote access to servers through your "Terminal" application

- After opening the terminal type the SSH command below after replacing `username` with the one provided to you – you will be prompted for password after that

```
staff$ ssh username@stampede.tacc.utexas.edu
```

User Environment

- An important component of a user's environment is the **login shell** as it interprets text on command-line and statements in shell scripts - `echo $SHELL` command tells the shell you are using
- There are **environment variables** for defining values used by the shell (*e.g.*, bash, tcsh) and programs executed on command line
  - *e.g.*, the `PATH` environment variable defines a list of directories that the shell should search to find an executable program that you have referred to on the command line - this allows you to execute that program without having to type the entire directory path to the executable file
- An **environment management package** provides a command-line interface to manage the collection of environment variables associated with various software packages, and to automatically modify environment variables as needed – *e.g.*, **modules**
- User environment can be customized via **startup scripts**

Modules - how to use them?

- Environment variables make many tasks including the following easy:
  - Creating scripts, porting code, running compilers and software
  - Viewing and managing applications, tools and libraries TACC/XSEDE systems is **Modules**
- Some of the module commands are
  - To see what modules have been loaded: **`module list`**
  - To see what modules are available: **`module avail`**
  - To swap one module for MPI library with another:
    `module swap mvapich2 impi`
  - To get help on a module (named foo) : **`module help foo`**

Module Commands Demo

```
staff$ module list
Currently Loaded Modules:
  1) TACC-paths   2) Linux   3) cluster-paths   4) intel/13.0.2.146   5)
   mvapich2/1.9a2   6) cluster   7) TACC   8) xalt/0.5.2

staff$ module spider gcc
  gcc:
  Versions:
        gcc/4.4.6
        gcc/4.6.3
        gcc/4.7.1

staff$ module load gcc/4.7.1
Lmod is automatically replacing "intel/13.0.2.146" with "gcc/4.7.1"
Due to MODULEPATH changes the following have been reloaded:
  1) mvapich2/1.9a2

staff$ module list
Currently Loaded Modules:
  1) TACC-paths   2) Linux   3) cluster-paths   4) cluster   5) TACC   6)
   xalt/0.5.2   7) gcc/4.7.1   8) mvapich2/1.9a2
```

Batch Mode and Interactive Mode

- A sequence of commands to be executed on the compute nodes is listed in a file (often called a batch file, command file, or shell script) and submitted for execution as a single unit – this is batch mode of job submission
  - Various resource managers and job schedulers used across different TACC/ XSEDE resources
  - Example, Gordon uses TORQUE resource manager and PBS job scheduler whereas Stampede uses SLURM as resource manager and job scheduler

- Interactive mode is opposite of batch mode - commands to be run are typed individually on the command-prompt
  - Interactive access to compute nodes is allowed on some XSEDE resources like Stampede – see user-guide for more information
  - Do not run your programs on login nodes – only do installation and compiling of code here

Job Scheduling

- All TACC/XSEDE compute resources use a job scheduler for running jobs
- All jobs are placed in a queue after they are submitted

Job Schedulers

- Attempt to balance queue wait times of competing jobs with efficient system utilization
  - Job prioritization influenced by number of cores and wall clock time requested
  - FIFO queues with fair use mechanisms to keep a single user from dominating the queue
  - Backfilling unused nodes with smaller jobs
- Will not start jobs if they will not finish before scheduled system maintenance

Script for Submitting a Batch Job

- Refer the user-guide of the TACC/XSEDE resource that you are using to find a sample batch script. A **sample SLURM job script**, named **`myJob.sh`**, that can be used for Stampede is shown below:

```
#!/bin/bash
#SBATCH -J myMPI # Job Name
#SBATCH -o myMPI.o%j # Name of the output file
#SBATCH -e myMPI.e%j # Name of the output file
#SBATCH -n 32 # Requests 16 tasks/node, 32 cores total
#SBATCH -p normal # Queue name normal
#SBATCH -t 00:10:00 # Run time (hh:mm:ss) - 1.5 hours
#SBATCH -A A-ccsc # Mention your account name (xxxxx)
set -x # Echo commands
ibrun ./example1
```

Submitting, Monitoring and Cancelling a Batch (SLURM) Job

```
staff$ sbatch myJob.sh
------------------------------------------------------------
             Welcome to the Stampede Supercomputer
------------------------------------------------------------

--> Verifying valid submit host (staff)...OK
--> Verifying valid jobname...OK
--> Enforcing max jobs per user...OK
--> Verifying availability of your home dir (/home1/01698/rauta)...OK
--> Verifying availability of your work dir (/work/01698/rauta)...OK
--> Verifying availability of your scratch dir (/scratch/01698/rauta)...OK
--> Verifying valid ssh keys...OK
--> Verifying access to desired queue (development)...OK
--> Verifying job request is within current queue limits...OK
--> Checking available allocation (A-ccsc)...OK
Submitted batch job 4439141
```

```
staff$ squeue -u rauta
             JOBID    PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           4439141 development    myMPI    rauta  R       0:04      1 c559-702
```

```
staff$ scancel 4439141
```

SLURM Command for Interactive Access to a Compute Node on Stampede

- SLURM's `srun` command will interactively request a batch job, returning a compute-node name as a prompt

- Issue the `srun` command only from a login node

- `srun` command syntax is:

```
srun --pty -A projectnumber -p queue -t hh:mm:ss -n tasks -N
nodes /bin/bash -l
```

```
login1$ srun --pty -p development -t 01:00:00 -n16 /bin/bash -l
  ...
c423-001$
c423-001$ ibrun ./a.out
```

References

- http://www.freeos.com/guides/lsst/ch02sec01.html
- http://www.cs.jhu.edu/~joanne/unixRC.pdf
- http://www.tacc.utexas.edu/documents/13601/118360/LinuxIntro_HPC_09+11+2011_hliu.pdf

Solutions to Exercises

Exercise-1

```
#!/bin/sh
md5 $1
```

Exercise-2

```
#!/bin/sh
for f in $1/*
do
 md5 $f
done
```