XSEDE

Extreme Science and Engineering Discovery Environment

Introduction to Linux

Presented by:

- Je'aime Powell, Texas Advanced Computing Center (jpowell@tacc.utexas.edu)
- Ritu Arora, Texas Advanced Computing Center (rauta@tacc.utexas.edu)

XSEDE Code of Conduct

XSEDE
Extreme Science and Engineering Discovery Environment

Reference: **`https://www.xsede.org/codeofconduct`**

This external code of conduct for XSEDE sponsored events represents XSEDE's commitment to providing an inclusive and harassment-free environment in all interactions regardless of gender, sexual orientation, disability, physical appearance, race, or religion. The code of conduct below extends all XSEDE-sponsored events, services, and interactions.

Anyone who experiences, observes, or has knowledge of threatening behavior are expected to immediately report the incident to a member of the event organizing committee, XSEDE staff, or one of the XSEDE Ombudspersons listed below. XSEDE reserves the right to take appropriate action.

- Linda Akli, SURA (akli@sura.org)
- Lizanne Destefano, Georgia Tech (lizanne.destefano@ceismc.gatech.edu)

Learning Objectives

- After attending this training, the participants would have
  - Developed familiarity with Linux and working in the command-line mode
  - Learnt to use the basic Linux commands
  - Learnt to create and edit files on a Linux OS
  - Learnt the basics of bash scripting
  - Learnt to connect to remote XSEDE resources such as Stampede2
  - Familiarized themselves with the user environment of the Stampede2 supercomputer (which has the Linux OS)
  - Learnt to submit batch jobs on the Stampede2 supercomputer
  - Learnt to transfer files to and from a remote Linux system

Please make sure your DUO is working, Globus Connect is installed, and if you are on Windows then PuTTY is installed too

To check DUO:

Portal.xsded.org → “Sign In” → “MY XSEDE” → “Profile” → “Manage DUO”

To check Gobus Connect:

Globus.org → “Log In” → “Endpoints”

Chef's Menu

- **Section 1:** Linux Primer, Basic Commands, and Connecting to the Stampede2 supercomputer at TACC

- **Section 2:** Linux Commands Continued, SSH, Editing Text From the Command Line, Bash Scripting

- **Section 3:** Using XSEDE Resources, Job Submissions, GSISSH, and Globus

Accessing the content for this class

https://tinyurl.com/yckewsmy

Section 1: History of Linux, Basic Commands, and Connecting to the Stampede2 Supercomputer

Linux is an Operating System (OS)

- What is an OS?
  - Software interface between the user and the computer hardware
  - Controls the execution of other programs
  - Responsible for managing multiple computer resources (CPU, memory, disk, display, keyboard, etc.)
  - Examples of OS: Windows, Unix/**Linux**, OS X

In the beginning there was UNIX and it was EXPENSIVE!!

PDP-7 (1969)

Grace Hopper

XSEDE

Where did Linux come from??

- Between 1960-1980 UNIX ruled the world!
- In 1984 GNU and in 1991 Linux were both designed as UNIX alternatives
- The two were later combined into a single kernel named GNU/Linux.

Torvald

29 Million Users
or
.7% Market
Share

Stallman

How much does Linux cost?

- Personal Use = Free!
  - Download from the web.
  - Borrow from a friend.
  - Purchase a Disk / Flash drive and install Linux on it.
- Enterprise = Up to $10,000 US (Service)
  - Direct purchase from a Linux distribution company.

Are there choices in Linux?

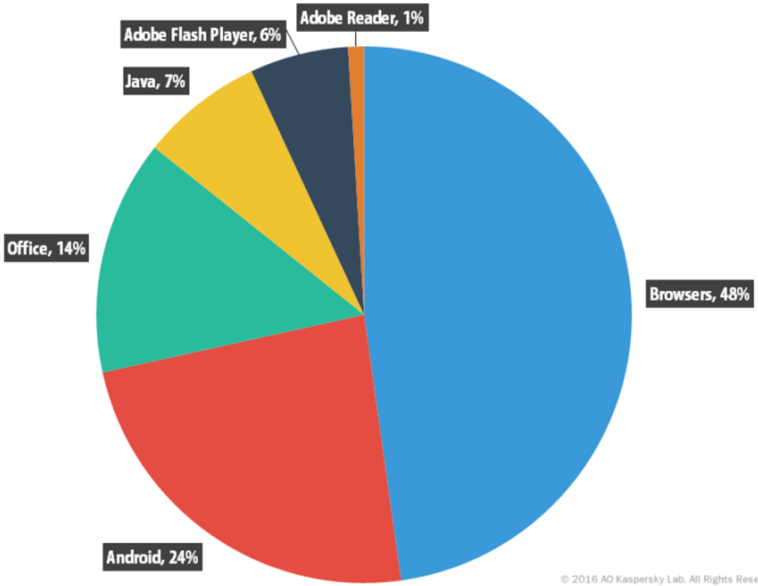
How can I use Linux when Microsoft owns my soul?

How can I use Linux when Microsoft owns my soul?

	WINDOWS	LINUX
Shell	Microsoft Exploder	Gnome, KDE, BASH, Borne, C, Z, T ….
Desktop Publishing	Microshaft Office 2019/365 $249 or $69 - $99/year	OpenOffice (Free)
Web Browser	Edge (*AKA the thing I use to download Firefox/Chrome*)	Firefox/Chrome/Chromium
Email	Windows Mail App	Evolution Mail/Thunderbird
Software Compiler	Visual Studio (*Free as of 2014 prior $1200*)	GCC, G++, Make, GFortran, OpenMPI ….

Oh one more thing ...

Vulnerable applications exploited by cybercriminals



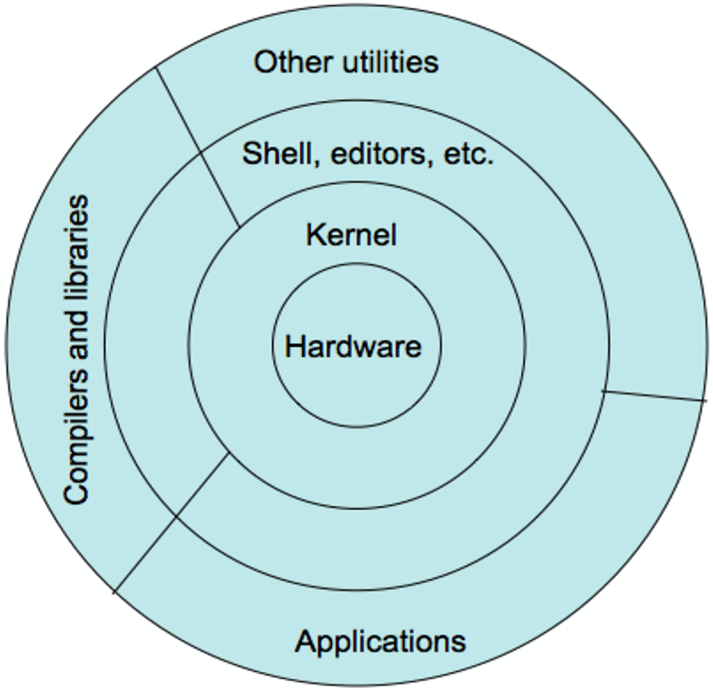
Sample of ~16M Ref: *https://securelist.com/it-threat-evolution-in-q2-2016-statistics/75640/*

XSEDE

Power of “and” and the tyranny of “or”

- Giving credit where it is due
  - Microsoft office brings productivity
  - Linux brings scalability
- Instead of picking between “Linux” or “Windows”, you can very conveniently choose to work with both from your computer should you wish to
  - You can install a Linux distribution on a pen-drive and boot your computer from it
    - https://www.pendrivelinux.com
  - You can install Linux in a VirtualBox and run it on a Windows system
  - We hear that the latest versions of Windows support WSL - that is, a Windows Subsystem for Linux
- In fact, you can work with Linux from your browser window - see slide # 19

What is a shell?



- Linux has a kernel and one or more shells
- The shell is the command line interface through which the user interacts with the OS. Most commonly used shell is "bash"
- The kernel sits on top of the hardware and is the core of the OS; it receives tasks from the shell and performs them

4

Files in Linux and the Naming Convention

- A file is a basic unit of storage and should have a name associated with it – please note – Linux is case-sensitive

- Files are organized into directories and sub-directories
  - A directory in Linux is a special file

- A directory is similar to a “Folder” in Windows OS

- In Linux, paths begin at the root directory which is the top-level of the file system and is represented as a forward slash (/)
  - Relative path versus absolute path

- Forward slash is used to separate directory and file names

Some top-level directories that you will see after installing Linux

/ - the root or base directory

/bin - non-essential binaries (applications)

/sbin - binaries essential to the system

/dev - contains all folders for devices

/etc - contains all configuration files

/home - user specific folder

/opt - holds software add-on packages

/var - holds spooling data

Note for OS/X users: You will see additional directories under the root directory if you are in your Terminal.

General trivia: You can create additional files and directories under the root directory and if you do so, you will see a different tree of sub-directories under the root directory.

Accessing a Linux terminal/shell from your browser

https://tinyurl.com/y8sp8aam

Interacting with the shell

- Type a command (**`ls`**) at the prompt (**`$ `**) and press ENTER Example: **`$ ls`**
- Shell starts a new process for executing the requested command, the new process executes the command and the shell displays any output generated by the command
- When the process completes, the shell displays the prompt and is ready to take the next command
- Specific information is passed to the command via more arguments
- The shell is killed by "**`exit`**" or **`CTRL-D`**

```
$ exit
logout
```

The exit command will not work in the browser terminal – try this later on Stampede2

Let's try some Linux commands

- To print the name of the current/working directory, use the `pwd` command

```
$ pwd
/home1/01698/rauta
```

- To make a new directory, use the `mkdir` command

```
$ mkdir csula222
```

- To change your working directory, use the `cd` command

```
$ cd csula222
```

Create, display, and edit a file using “cat”

- You can add content to a file as follows

```
$ cat > test.txt
This is what I am entering from the console
CTRL-D
```

- You display the contents of the file using the `cat` (short for concatenation) command

```
$ cat test.txt
This is what I am entering from the console
```

- You can append content to a file as follows

```
$ cat >> test.txt
Appending more lines
CTRL-D
```

Let us check the currently available files/directories

- To list the contents of a directory, use the `ls` command

```
$ ls
```

- To list the contents of a directory along with the time-stamp and in reverse chronological order, use the `ls` command

```
$ ls -tr
```

- To see all files and directories, including hidden ones use the `-a` flag with the `ls` command. Hidden files have a "." infront of them

```
$ ls -a
```

Note: your current working directory can be checked by using the `pwd` command. Also, you can combine multiple flags in a single command.

Copying files and changing directories

- To copy contents of one file to another, use the `cp` command

```
$ cp test.txt copytest.txt
$ cp test.txt test3.txt
```

One more example:

Relative path example

```
$ mkdir junk
$ cp test.txt ./junk/test2.txt
```

(The command above copies a file to the sub-directory `junk` that exists at the current path)

```
$ cd junk
$ ls
```

- To go a level up from the current working directory

```
$ cd ..
```

Exercise-1 (Part A)

- Run the following commands to make a directory:

```
$ mkdir csula29
$ cd csula29
```

- Create a file using **cat** command in `csula29` (see slide # 22)

```
$ cat > test.txt
```

- Run the following commands in the `csula29` directory

```
$ cp test.txt test2.txt
$ mkdir junk
$ mkdir junk2
$ cp test2.txt ./junk/test2.txt
$ cp test2.txt ./junk2/test2.txt
$ ls
```

Exercise-1 (Part B)

- Run the following commands starting from the `csula29` directory that you created in Part A of Exercise-1

```
$ ls
$ cd junk
$ ls
$ cd ..
$ cd junk2
$ ls
$ cd ..
$ ls
$ cp test.txt test3.txt
```

Deleting files and directories – there is no “undo”

- To remove a file, use the `rm` command

```
$ rm test2.txt
```

- To remove a directory, use the “ `-r` ” option with the `rm` command

```
$ rm -r junk2
```

- You can also use the `rmdir` command to remove an empty directory

```
$ rmdir junk2
```

Note: `rmdir` command does not have `-r` option

Renaming files and directories, getting help, checking differences between files

- A file can be renamed by moving it. The same can be achieved by using the `mv` command

```
$ mv test3.txt newtest3.txt
```

- Use the `man` command to get more information about a command – it is like using help in Windows

```
$ man rmdir
```

- Use the `diff` command to see the differences in two files

```
$ diff test.txt newtest3.txt
```

Checking the previously run commands

- Previously executed commands in a shell can be viewed by using the `history` command. For example:

```
$ history
  1  man ls
  2  ls -tr
  3  ls -t -r
  4  ls -tr
  5  history
```

Viewing large files more or less

- If the contents to display are more than one page, you could use the `more/less` command for paging through text a screenful at a time

```
$ more test.txt
$ less test.txt
```

Creating a tar file

- TAR (Tape Archive) command bundles files and sub-directories together and creates an archive (known as tar file or tarball)

- To create a tarball of all the files and sub-directories in the directory `csula29` that you created in Exercise 1, use **`c`** flag:

```
$ tar -cvf mytar.tar *
```

- To extract the contents of a tar file use **`x`** flag:

```
$ tar -xvf mytar.tar
```

Creating a compressed tar file

- To compress the tar file as it is being created use the **z** flag with the **c** flag :

```
$ tar -cvzf mytar.tar.gz *
```

- To extract the contents of a compressed tar file use **x** flag:

```
$ tar -xvf mytar.tar.gz
```

Note: the **c**, **v**, and **f** flags mean create a new archive, be verbose so that the files being archived are listed, and write the archive to a file.

Connecting to remote computers using a SSH client

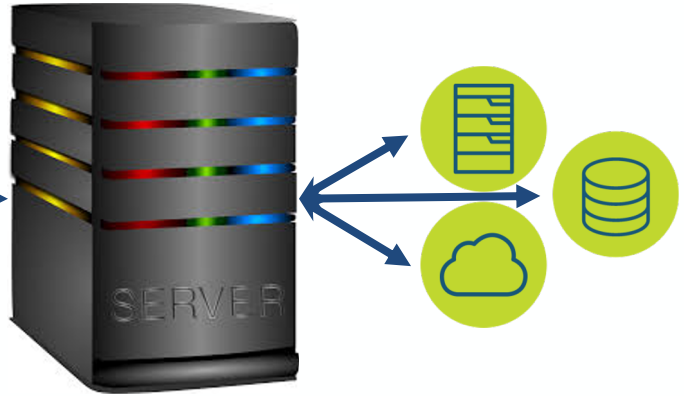
What is a Secure Shell (SSH) Client?

- The “Secure” part of SSH means the connection utilizes some form of cryptographic (shuffled characters) method to make the plain text traffic, coded to outside viewers.
- The “SHell” part of SSH is the interface (e.g., blinking cursor) the user transmits commands or network data through.
- The client is an application used to create the SSH connection to a remote computer over a given network.

What happens during a SSH connection

Local

Remote



$ssh jhpowell@stampede2.tacc.utexas.edu

```
OpenSSH_7.9p1, LibreSSL
2.7.3
debug1: Reading
configuration data
/Users/jpowell/.ssh/con
fig
….
```

login1(597)$

XSEDE

Some Sample SSH Clients

Note: We don't care what SSH client you use as long as you can log into the server.

Windows

-PuTTY

-cmd

Linux

-Terminal

iOS

- SSH Term

Chromium

-Secure Shell

MacOS

-Terminal

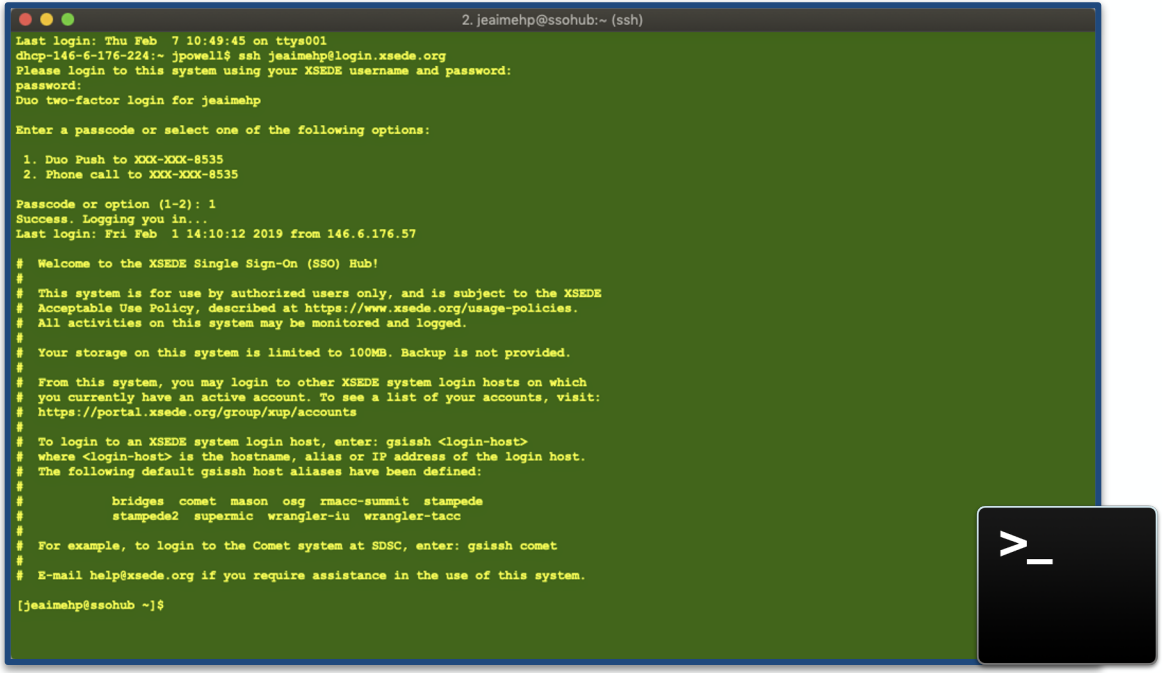
-iTerm

Android

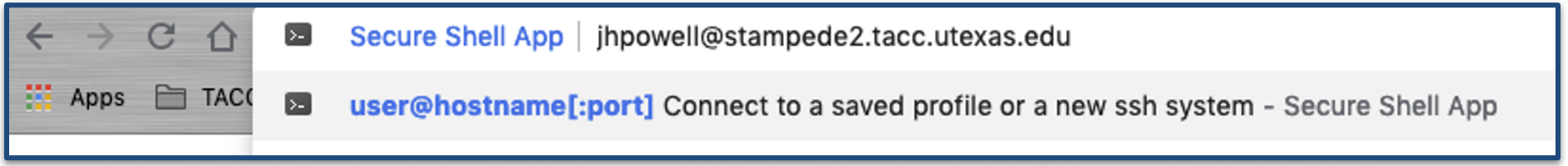
-JuiceSSH

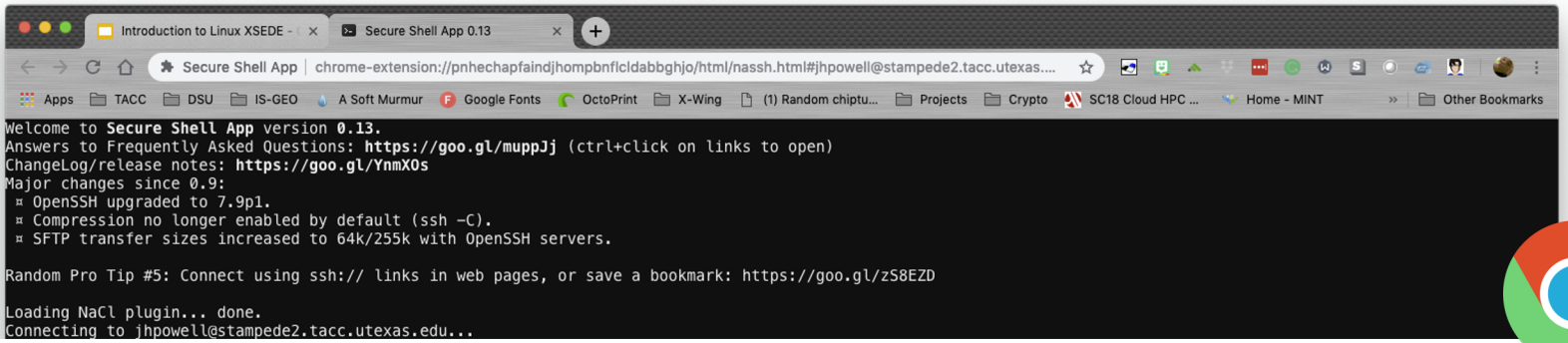
Example from MacOS Terminal App

MacintoshHD → “Applications” → “Utilities” → “Terminal”

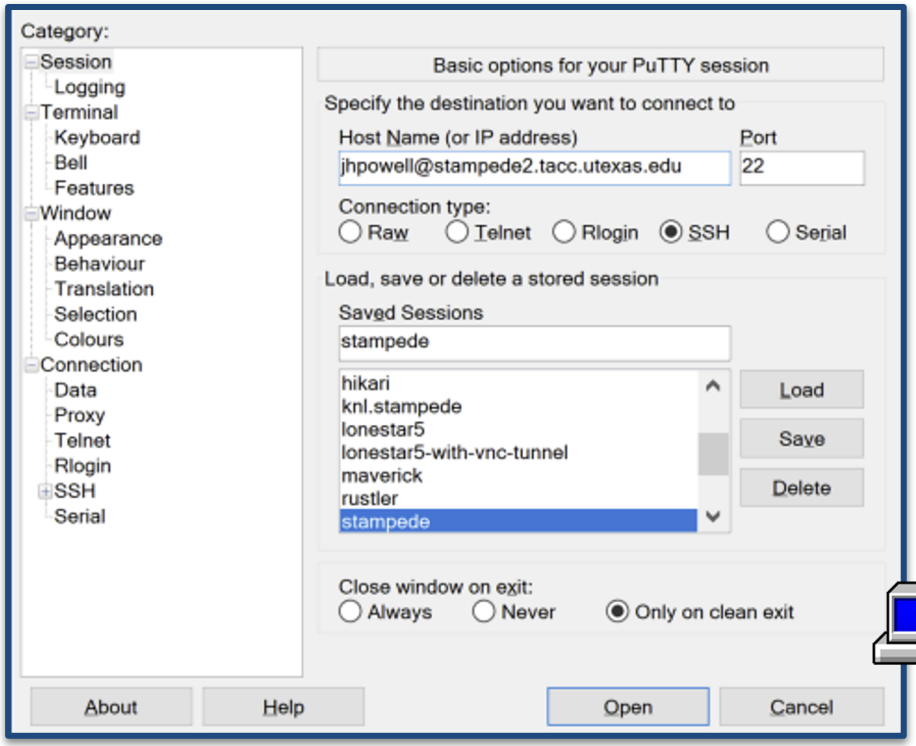


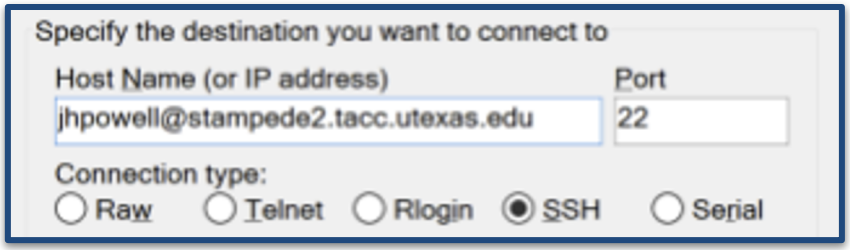
Example from Chrome “Secure Shell App” Extension





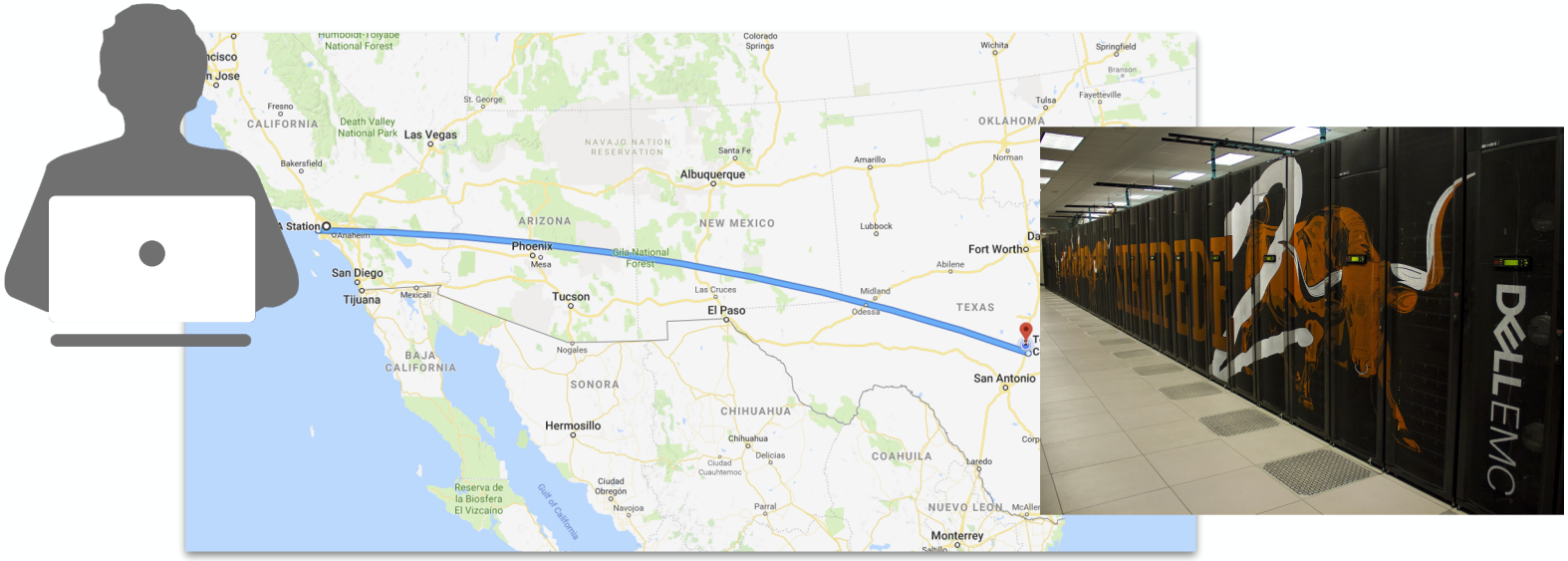
Example from Windows PuTTY





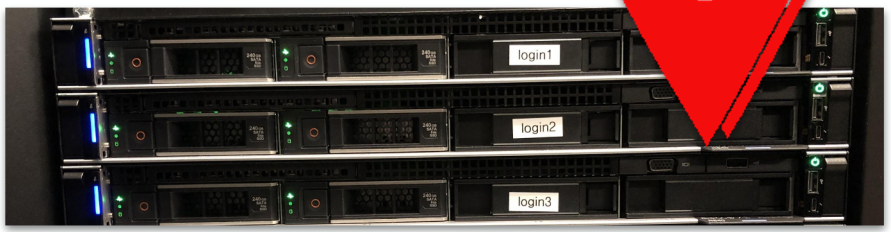
XSEDE

Where you are ...



XSEDE

Where you are on Stampede 2 ...



Login Nodes

Home Directory “~”
Storage

XSEDE

Let's connect!

[**Note**: The **`username`** is the training account provided]

SSH to this server:

***`username`*`@login.xsede.org`**

`Then "gsissh stampede2"`

`or`

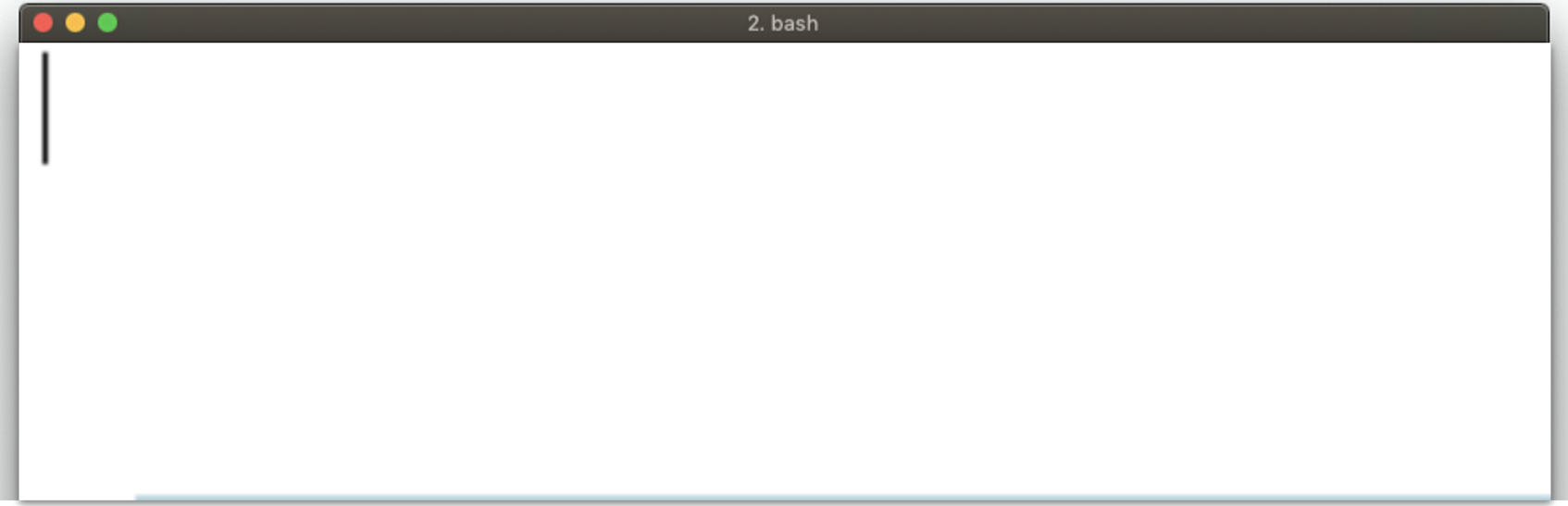
`username@stampede2.tacc.utexas.edu`

What we have done so far...

- Introduced to Linux history and the Opensource movement/indoctrination
- Used some basic Linux Commands
- Connected to **`login.xsede.org`**

NOTE: MAKE SURE YOU ARE LOGGED INTO login.xsede.org

Section 2: Linux Commands Continued, Editing Text From the Command Line, Bash Scripting



XSEDE

Overview

- Further practice of Linux commands
- Learning to gather Linux system information
- Filesystem navigation
- File manipulation
  - *Text Editors*

NOTE: MAKE SURE YOU ARE LOGGED INTO login.xsede.org

Let us run all the commands that you have learnt so far

- Repeat Exercise 1

- Try the following commands:

```
$ ls -ltr
$ ls -ltra
```

You will notice a hidden directory named `.ssh`

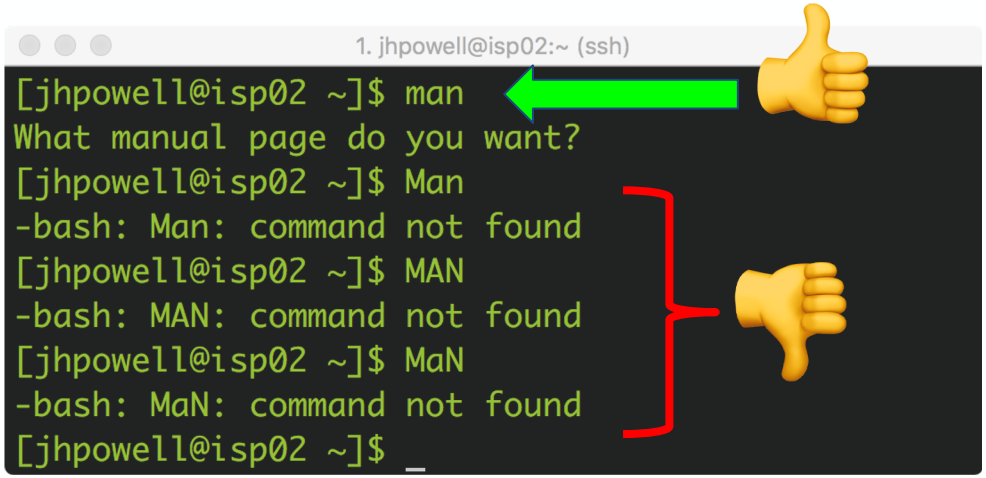
Some Observations and Generalizations

General Linux Command Line Syntax

- The command line (and Linux in general) is case sensitive

Example:
The "**man**" command

man ≠ Man ≠ MAN ≠ MaN



General Linux Command Line Syntax

- A Linux command generally has the following syntax…

`command -options file1 file2 …`

Example of how to use the **`man`** command to see what the **`ls`** command does:

Syntax:

`man ls`

Ok I logged in, now where am I?

- `hostname` ➔ Shows the name of the server
- `hostname -i` ➔ Show the IP address of the server
- `pwd` ➔ Prints the working path or where you are
- `ls` ➔ Lists the files and directories
  - `ls -a` ◆ Lists all files
  - `ls -l` ◆ Lists files with a long listing

What can this machine do?

- **`df -h`**
  - ➔ Displays the sizes of the mounted storage
    - ◆ the **`-h`** is an option for "*human-readable*"
- **`who`**
  - ➔ Displays currently logged in users
- **`uname -r`**
  - ➔ Displays the release of Linux on the machine
- **`ping`**
  - ➔ Allows you to check if you can communicate with other servers
    - ◆ e.g. **`ping google.com`**

How can I move around? Using the commands you have already learnt…

- **cd** *<dir>*
- **cd ~**
- **mkdir** *<dir>*
- **cp** *</dir/file> </dir/file>*

➔ Change to the directory *<dir>*

➔ Change to your $HOME directory
  - $HOME is a variable associated with your user account

➔ Make (create) a directory named *<dir>*

➔ Copy a file from one location to another

Check username and group

- Three types of users: owner or user, group, all others
- To check the login name use the command **`whoami`** or **`echo $USER`**
- To check the groups you are a member of use the command **`groups`**
  - Members of a Linux group can share files with each other
  - Each account is assigned a primary group, and an account can be a member of multiple groups
- To check your user id, or group id use the command **`id`**

File Permissions (1)

- Users typically perform the following operations on files:
  - Read files (using more, cat, *etc.*)
  - Write files (using >, editors such as vi, *etc.*)
  - Execute commands in a file (executables, *etc.*)

- Each file has three permissions – read, write and execute (`rwx`)

- Person creating the file is the owner or user and can modify permissions as desired
  - Owner can modify permissions on files to grant or revoke access to other users

File Permissions (2)

- To check the file permissions use the `-l` flag with the `ls` command

```
$ ls -l
total 24
drwx------  2 rauta G-25072 4096 Jan 17 14:07 junk
drwx------  2 rauta G-25072 4096 Jan 17 14:15 junk2
-rw-------  1 rauta G-25072   65 Jan 17 13:59 test.txt
```

File Permissions (3)

- `chmod` command is used to change permissions on a file
- To add specific permission use `chmod +`
  - To add write permission to all users use:

    `chmod a+w filename`
  - To add read permission to only the users in your group use:

    `chmod g+r filename`
  - To make a file executable and runnable by any user

    `chmod a+x myfile`
- To remove specific permission use `chmod -`
- Add and remove permissions can be combined in a single step
  - **`chmod u+x,g+r,o-rwx filename`**

File Permissions (4)

- Instead of using alphabets u, g, o for user, group, and others we can use numbers to specify file permissions

```
rwx = 111 = 7
rw- = 110 = 6
r-x = 101 = 5
r-- = 100 = 4
-wx = 011 = 3
-w- = 010 = 2
--x = 001 = 1
--- = 000 = 0
```

- `chmod go+rx filename` = `chmod 755 filename`

(assuming the user already has the `r`, `w`, and `x` permissions.)

Directory Permissions

- To check the contents of a file with `ls` command, you would need read permission
- To add or remove files in a directory, you would need write and execute permission
- To change to a directory or to go through its contents, you would need execute permission
- To list files in a directory using `ls -l` command you would need read and execute permissions

Redirecting Output

- By default, the output is displayed on the screen

- " > " symbol can be used to redirect the output to a file or a utility (*e.g.,* `ls`). Example:

```
ls -ltr > myContent
```

- The " | " symbol is used to connect the output of one process to the input of another process

```
ls -l | wc -l
```

`wc` counts the number of lines

Other Directives

- " < " symbol is used for input redirection

```
$ mail -s "CSULA class" rauta@tacc.utexas.edu < test.txt
```

- " >> " symbol is used for appending output to a file

```
$ cat test3.txt >> test.txt
```

- " ; " is used to execute multiple commands in one step

```
$ clear;date
```

Process and process control

- `ps` – display process information on the system
- `kill` *`pid`* – terminates the process id
- `^c` (CTRL+c) terminates the running program

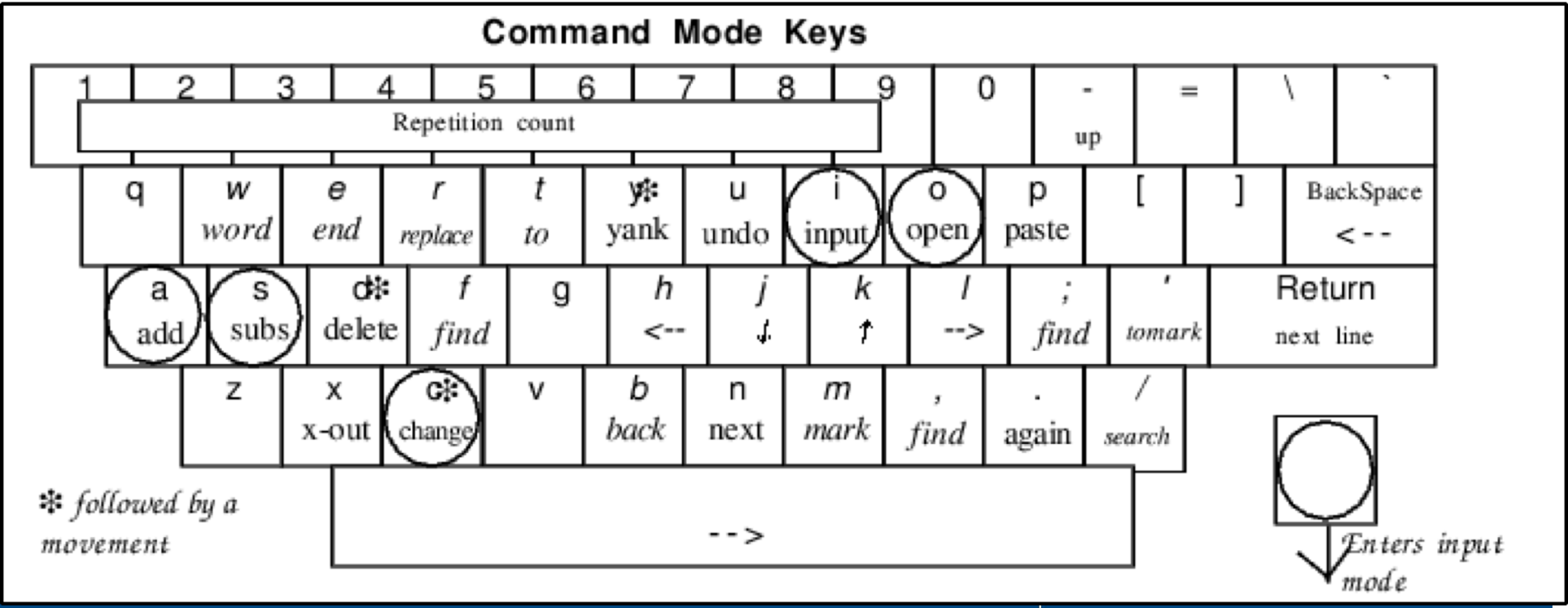
```
$ ps
  PID TTY          TIME CMD
20482 pts/32   00:00:00 bash
21035 pts/32   00:00:00 ps
```

Basic commands Recap

- hostname
- pwd
- ls
  - -ltrah
- cd
  - ~
- mkdir
- cp
- history
- cat
- rm
- mv
- exit

Command Line Text Editors

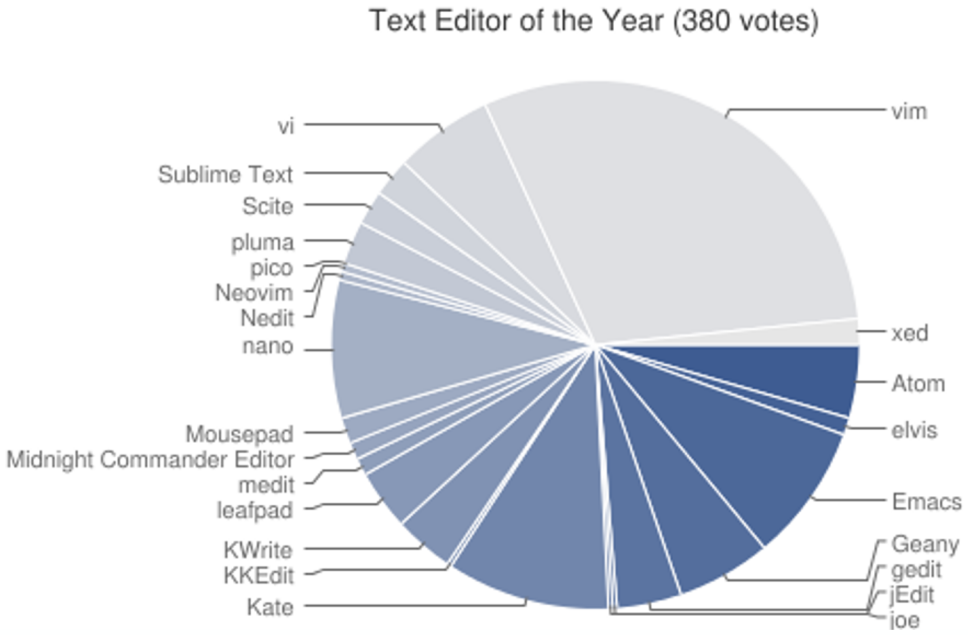
Imagine if you, will a time when ...



XSEDE

It is up to you to choose your editor

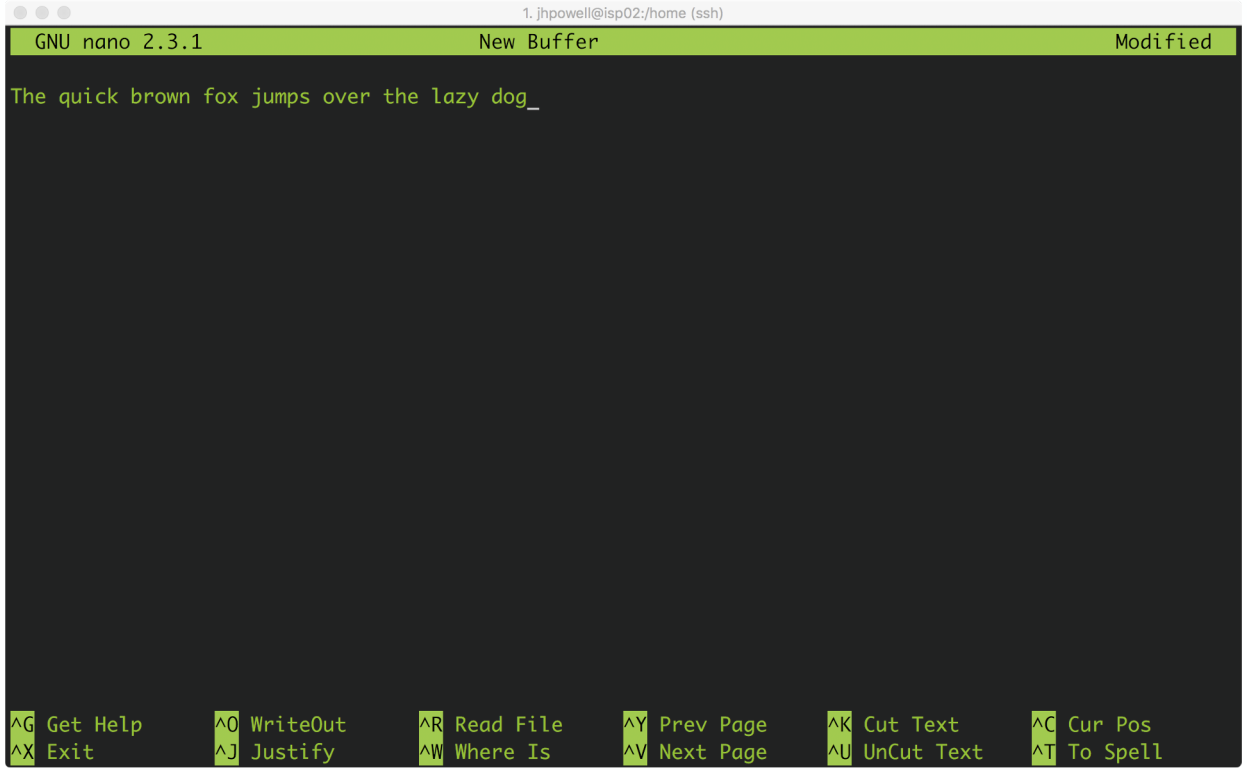
- **nano**
- **vi**
- **emacs**



Ref:
https://www.reddit.com/r/vim/comments/5trtbj/and_the_text_editor_of_the_year_is/

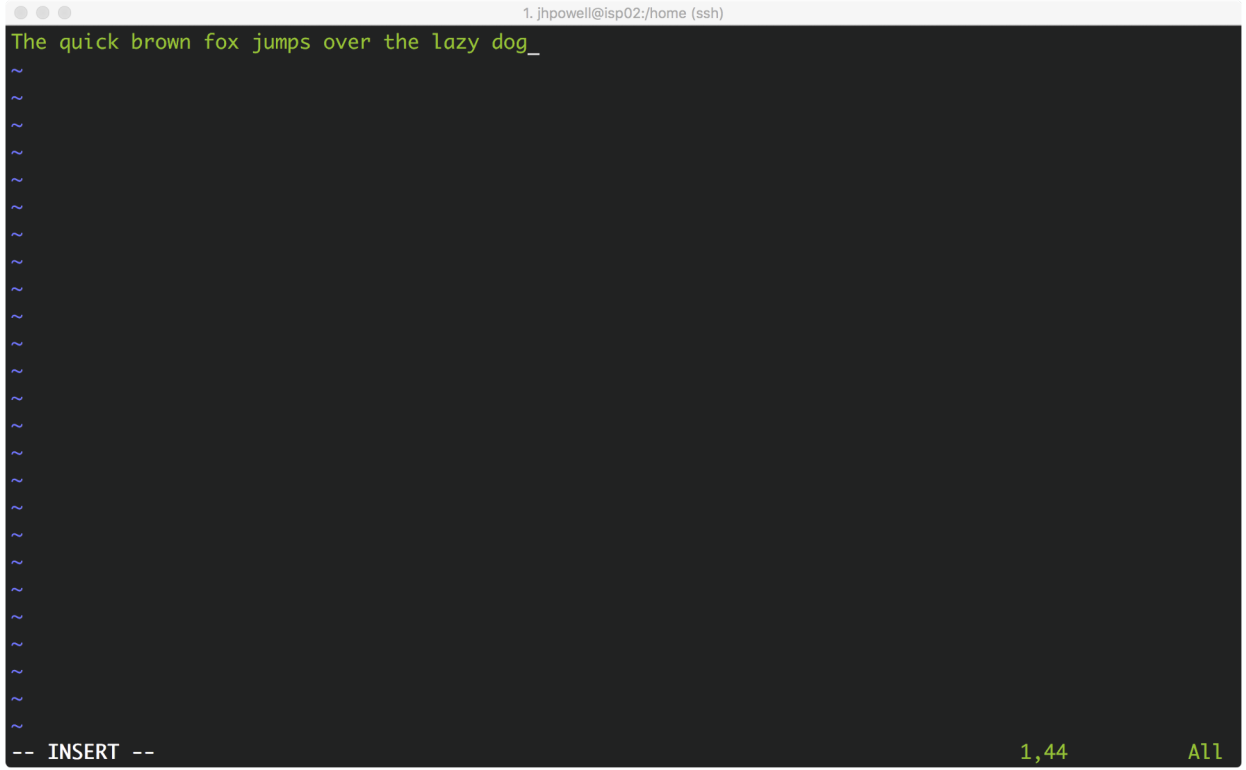
nano

To exit: press [Ctrl] [X] and then **N** and [Enter] to not save



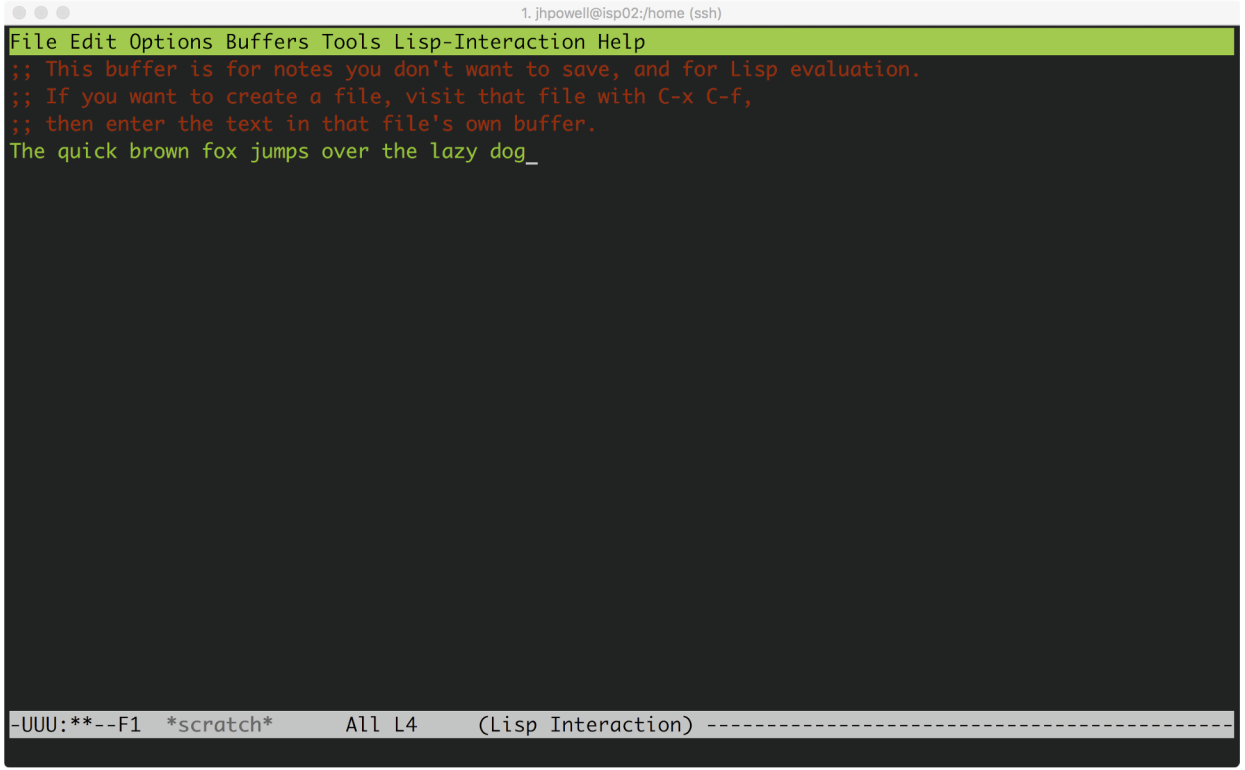
vi/vim

To exit:
press [Esc]
and then
type `:q!`
and [Enter]
to not save



emacs

To exit: press [Ctrl][x] then [Ctrl][c] to not save



Quick vim exercise

Goal: Type:

"The quick brown fox jumps over the lazy dog."

into a text file in your home directory.

Hints:

- Do not touch your mouse/trackpad!!!
- **`vi filename.txt`** to start
- Use **`i`** to enter insert mode
- Use [Esc] to leave insert mode

Command	Effect
vim file.txt	open "file.txt" and edit with vim
i	toggle to insert mode
<Esc>	toggle to normal mode
<arrow keys>	navigate the file
:q	quit editing the file
:q!	quit editing the file without saving
:w	save the file, continue editing
:wq	save and quit

One more time - editing files using the vi editor

- To create a new file use the `vi` command (see cheat-sheet)

```
$ vi test2.txt
```

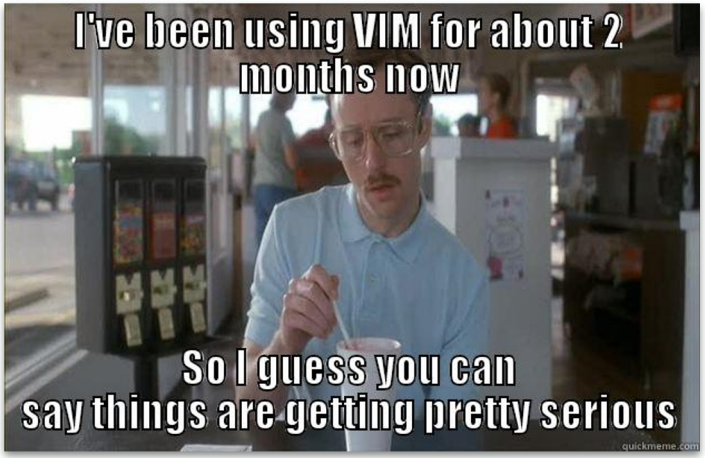
  - Press i to start inserting text
  - Type some text: `Hello Class!`
  - To save and quit, press " Esc " key, and enter `:wq!`

    (press the enter key after typing `:wq!`)
  - To quit without saving, press " Esc " key if in insert mode, and enter " `:q!` "

- To display the contents of the file, use the `cat` short for concatenation) command

```
$ cat test2.txt
```

Try the three (or more) editors and choose one!



Scripting Time!

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 scripts

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 date

- cal command prints the calendar

Passing arguments to shell 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"
```

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

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

Section 3: Job Submissions, Using XSEDE Resources, GSISSH, and Globus

Loops in shell scripts

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

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

Exercise-2

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

  `md5sum` <file-name>

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

Solution to exercise-2

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

Objectives

Student will…

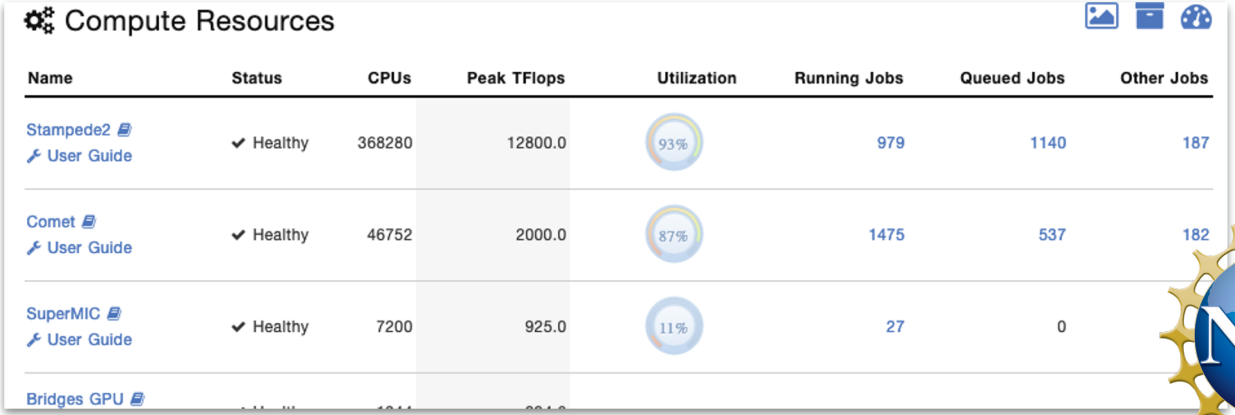
- Connect to the XSEDE SSO Login Node
- Use gsissh to access Stampede2
- Submit a job to Stampede2
- Use Globus to transfer files

NOTE: MAKE SURE YOU ARE SSH'D INTO
stampede2.tacc.utexas.edu

What is XSEDE?

"*XSEDE is an NSF-funded virtual organization that integrates and coordinates the sharing of advanced digital services - including supercomputers and high-end visualization and data analysis resources - with researchers nationally to support science.*"

In short, they connect scientists, researchers, and students (like you!) to large scale computing resources.



XSEDE Prerequisites

- An active xsede.org account
- An active allocation on a XSEDE resource
  - https://portal.xsede.org/allocations/startup

Getting on a XSEDE resource the SSO and `gsissh`

1. First we are going to SSH first to: login.xsede.org

2. Second we are going to "**`gsissh`**" to Stampede 2

But why two SSH steps??

- This allows you user credentials (username and pasword) to travel with you based on your "Allocations"!

Batch mode versus 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
  - Multiple resource managers and job schedulers used across different XSEDE resources
  - Examples include: TORQUE resource manager and PBS job scheduler, 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
  - See user-guide for more information
  - Do not run your programs on login nodes – only do installation and code compiling

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`**, 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 48 # Requests 48 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

```
$ sbatch myJob.sh

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
```

Let's download our sample program

Go to the following site and download the zip file to your computer.

[**Note**: Remember where you save it, you will need it again]

`https://github.com/jeaimehp/xsedeexample`

Transferring data using Globus.org

1. Install “Globus Connect Personal”
    a. Make sure to add your computer/endpoint to Globus during setup
2. Go to globus.org and “Log In”
3. Go to “Your Collections” and choose your “Endpoint”
4. Select the “Panel” with two(2) columns
5. Add the Collection “XSEDE TACC Stampede 2”
6. Select the file(s) you want to transfer and click “Start”
    a. xsedeexample-master.zip
7. Click “refresh list” to see the new files when completed

Submitting a job script (SLURM scheduler)

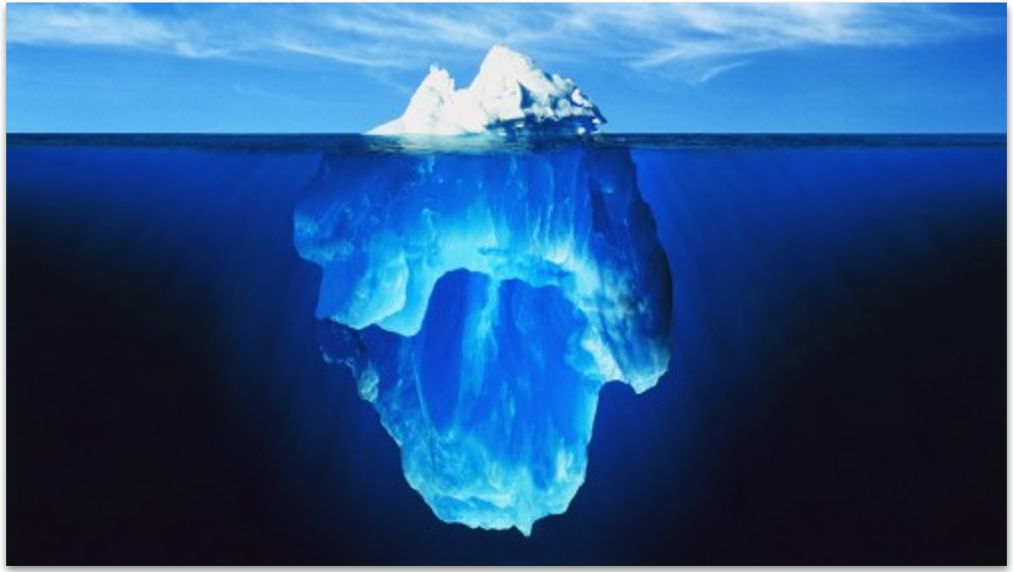
1. **ssh** to `login.xsede.org` then **gsissh** to **stampede2**
2. **unzip xsedeexample-master.zip**
3. **cd** to **xsedeexample-master**
4. Edit (use either `nano` or `vi`) example_submission.bash to modify the email address to yours

   Ex.

   ```
   #SBATCH --mail-user=jpowell@tacc.utexas.edu
   ```

1. **sbatch example_submission.bash**

Whew!!! That was a lot ….



Session Survey

XSEDE
Extreme Science and Engineering Discovery Environment

- Please complete a short on-line survey about this module at http://bit.ly/xsedesurvey. We value your feedback, and will use your feedback to help improve our training offerings.