LINUX

ERIC REES, PHD

CS 4352

TEXAS TECH UNIVERSITY – SPRING 2023

LINUX

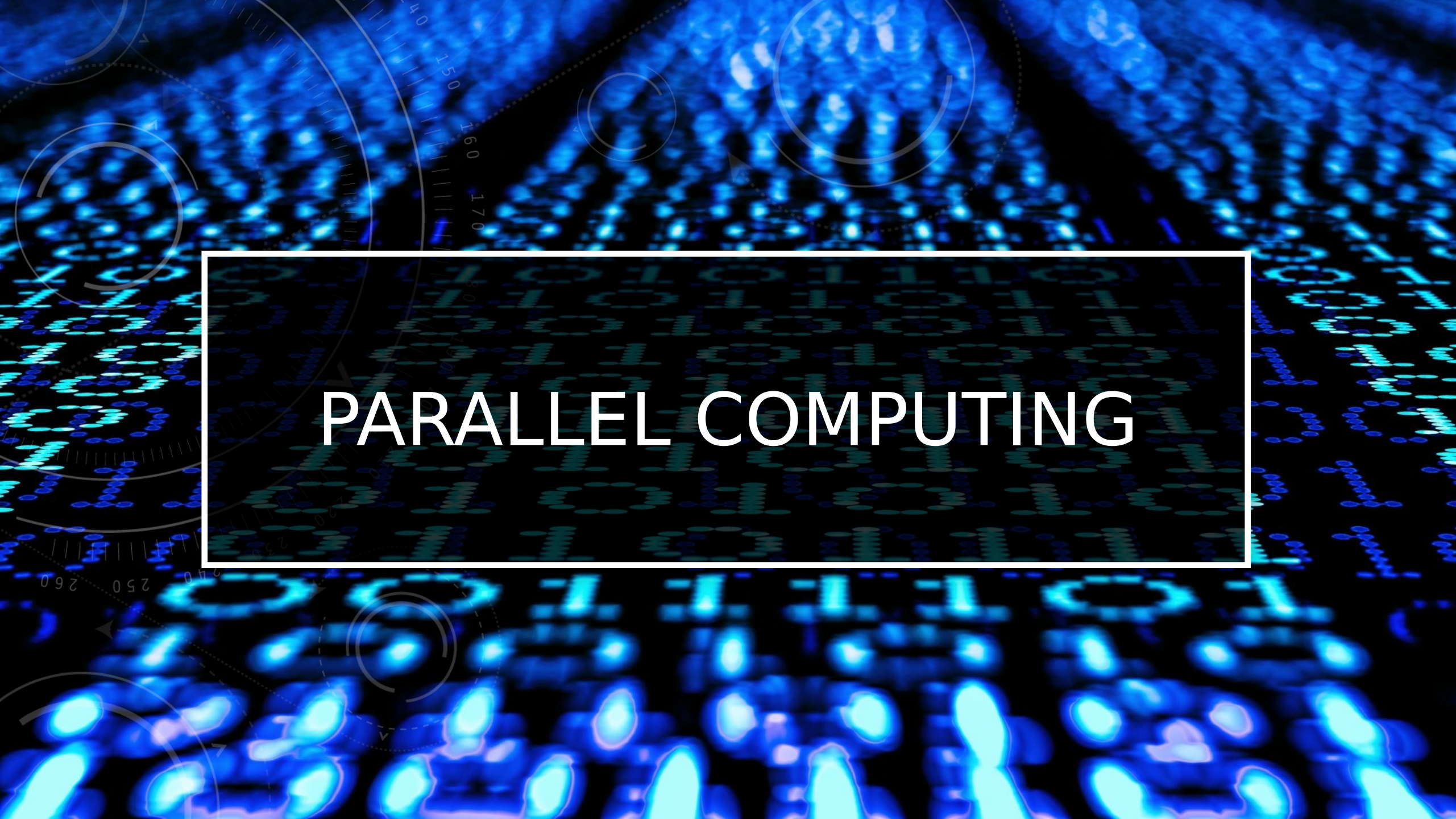
- **GNU Project**
  - Richard Stallman started the GNU project in 1983
    - After AT&T imposed commercial licensing on Unix Operating System
  - Reprogramming some of the Unix Tools. running on a small kernel (TRIX)
  - Required more advanced and reliable OS Kernel

- **Linux**
  - Linus Torvalds created the Linux kernel in early 1990's
    - Linus released first kernel under the GPL (General Public License)
    - Requires that the source code remains freely available to everyone

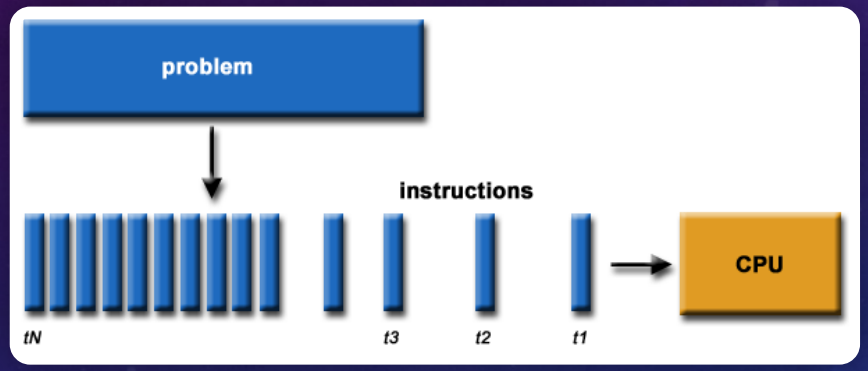
LINUX DISTRIBUTIONS

- Linux has various distributions (Distros):
  - Why so many distributions?
    - Server vs. Desktop
    - Commercial user support
    - Special hardware support

  - Debian / RedHat / SUSE / …
    - The major difference is the software package management on these distros.

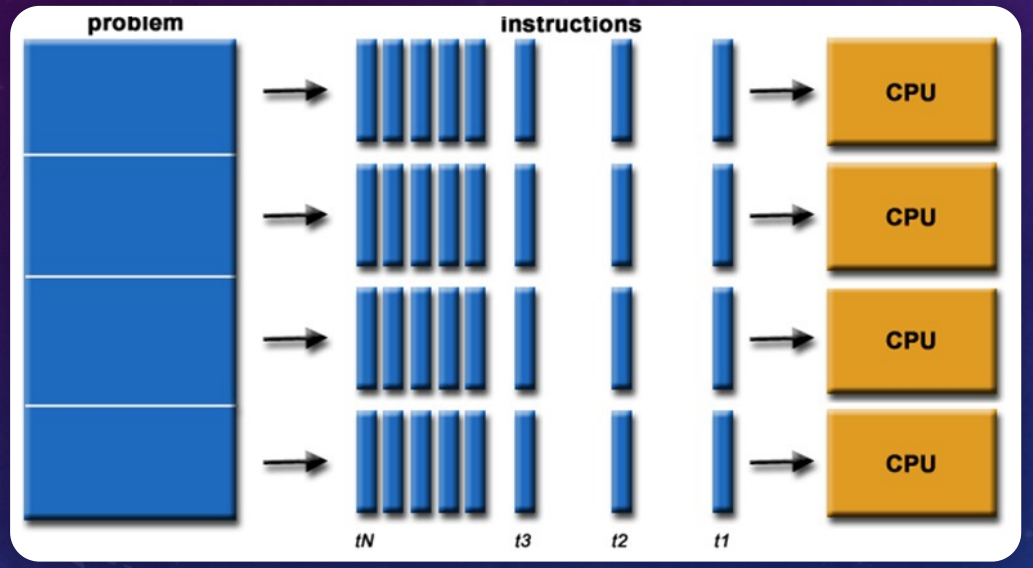

PARALLEL COMPUTING

WHAT IS PARALLEL COMPUTING?



- Software is often written and perceived to run serially.
  - Execution occurs on a single computer using a single CPU core.
  - A problem is broken into a discrete series of instructions.
  - Instructions are executed one after another.
  - Only one instruction may execute at any given time.

WHAT IS PARALLEL COMPUTING?



- **Parallel Computing** is the simultaneous use of multiple compute resources to solve a computational problem.
  - Execution occurs across multiple CPU cores.
  - A problem is broken into discrete parts that can be solved concurrently.
  - Each part is further broken down into a series of instruction, executed one after another.
  - Instructions from each part execute simultaneously on different CPU cores.

COMMON TERMINOLOGY

- **Node**
  - A standalone server.
- **CPU**
  - One of the most misused terms in computing – often because the term's meaning has shifted over time.
  - For our purposes, the CPU will refer to a physical processor within a node, with each CPU likely containing multiple "cores"
- **Core**
  - CPUs can be subdivided into multiple "cores", each being a unique execution unit .

COMMON TERMINOLOGY

- **Wall Clock Time**
  - The actual amount of time a process takes to run.
  - Parallelism should cause this value to decrease.

- ***CPU Time***
  - The amount of time a process ran in terms of parallelism.
  - In a perfect world this value would be

COMMON TERMINOLOGY

- **Observed Speedup**
  - A simple and widely used indicator of a parallel program's performance.
  - Defined as

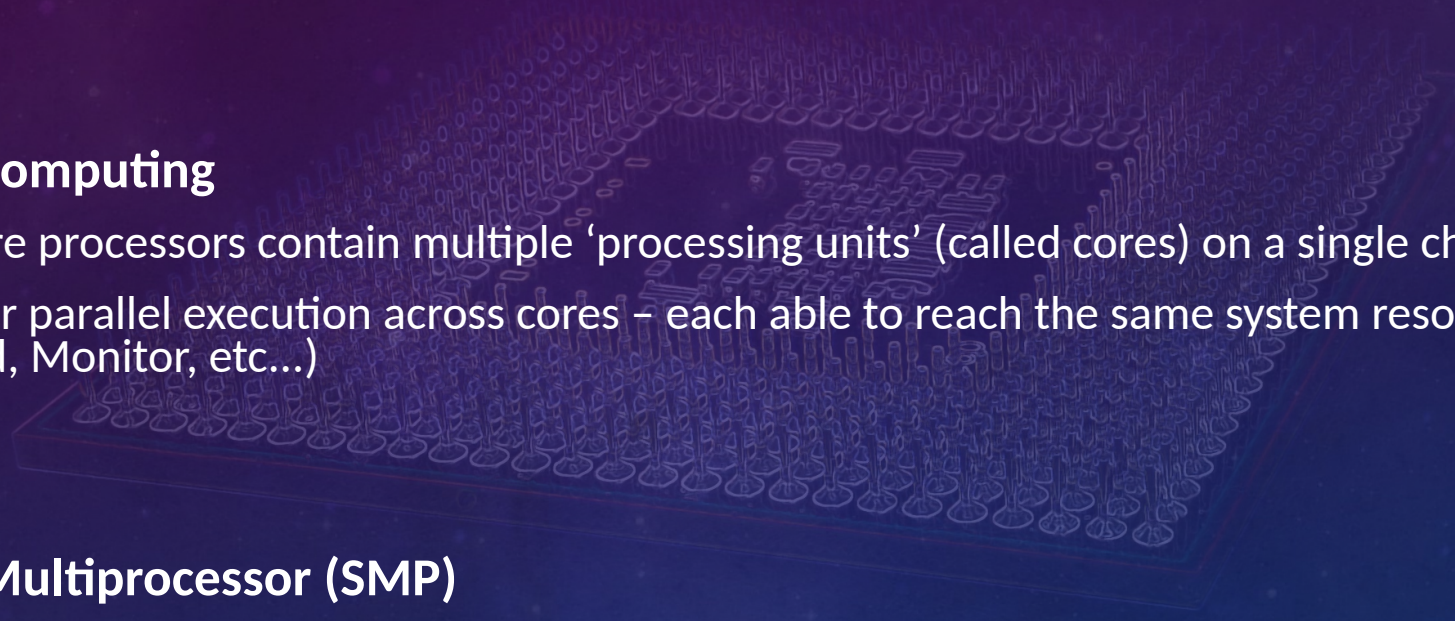
- **Parallel Overhead**
  - The amount of time required to coordinate parallel tasks, as opposed to doing actual computation.
    - Task startup time
    - Synchronization time
    - Data communications

COMMON TERMINOLOGY

- **Embarrassingly Parallel**
  - Each task of a problem can be solved independently of all other tasks.
  - Example: Matrix Addition

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} \\ a_{31}+b_{31} & a_{32}+b_{32} & a_{33}+b_{33} \end{bmatrix}$$

CLASSES OF PARALLEL COMPUTERS



- **Multi-core Computing**
  - Multi-core processors contain multiple ‘processing units’ (called cores) on a single chip.
  - Allows for parallel execution across cores – each able to reach the same system resources (RAM, Keyboard, Monitor, etc…)

- **Symmetric Multiprocessor (SMP)**
  - A symmetric multiprocessor is a computer system with multiple identical processors.
  - Each processor likely has multiple cores.
  - Allows for parallel execution across cores – each able to reach the same system resources (RAM, Keyboard, Monitor, etc…)

CLASSES OF PARALLEL COMPUTERS

- **Clusters**
  - A group of loosely coupled computers working together closely.
  - Processes can be spread across multiple nodes but processes are unable to reach the same system resources (RAM, Keyboard, Monitor, etc…)

- **Massively Parallel Processors (MPP)**
  - A group of tightly coupled computers working together closely across a specialized high-speed interconnect.
  - Processes can be spread across multiple nodes but processes are unable to reach the same system resources (RAM, Keyboard, Monitor, etc…)

CLASSES OF PARALLEL COMPUTERS

- **Grid Computing**
  - Highly distributed form of parallel computing.
  - Clusters or single resources are spread across multiple sites using the Internet for connectivity.
  - Plagued by high latency issues.

- **Cloud Computing**
  - Same as Grids but often run by for-profit entities.
  - Sites are often spread over large geographic areas but most processing occurs within a single site – to help avoid latency issues.

WHY DOES IT MATTER?

- Exposes 3 kinds of programming models:
  - **Serial programming**
    - Executes serially using a single core / thread
    - Single core machines
  - **Multi-core / Multi-threaded Programming**
    - Executes in parallel using multiple cores / threads
    - All threads are running on the same machine and access the same RAM
    - Multicore & Symmetric Multiprocessing
  - **Massively Parallel Programming**
    - Executes in parallel using multiple machines
    - Clusters, Massive Parallel Processors, & Grid/Cloud


HPCC RESOURCES

HPCC RESOURCES

- High Performance Computing Center (HPCC)
  - Operates and maintains three data centers
    - HPCC Main Data Center - TTU ESB
    - Chemistry Data Center
    - Reese Center - TTU TIEHH / Reese Technology Center
  - Operates the RedRaider Computing Cluster
    - Cluster is broken into 5 distinct partitions

QUANAH PARTITION

- Commissioned in 2017
- Running CentOS 7.4
- Currently consists of:
  - 467 Nodes
  - 16,812 Cores (36 cores/node)
  - 87.56 TB Total RAM (192 GB/node)
  - Xeon E5-2695v4 Broadwell Processors
  - Omni-Path (100 Gbps) Fabric
- Benchmarked at 485 Teraflops/sec

XL QUANAH PARTITION

- Commissioned in 2021
- Running CentOS 8.1
- Currently consists of:
  - 16 Nodes
  - 576 Cores (36 cores/node)
  - 4.00 TB Total RAM (256 GB/node)
  - Xeon E5-2695v4 Broadwell Processors
  - Mellanox (56 Gbps) Fabric
- Not benchmarked

NOCONA PARTITION

- Commissioned in 2020
- Running CentOS 8.1
- Currently consists of:
  - 240 CPU Nodes
  - 30,720 Cores (128 cores/node)
  - 122.88 TB Total RAM (512 GB/node)
  - AMD Rome Processors
  - HDR InfiniBand (200 Gbps) Fabric
- Benchmarked at 804 Teraflops/sec

MATADOR PARTITION

- Commissioned in 2020
- Running CentOS 8.1
- Currently consists of:
  - 20 GPU Nodes
  - 40 NVIDIA Tesla V100 (2/node)
  - 25,600 tensor cores (1280/node)
  - 204,800 CUDA cores (10,240/node)
  - HDR InfiniBand (100 Gbps) Fabric
- Benchmarked at 226 Teraflops/sec

TOREADOR PARTITION

- Commissioned in 2021
- Running CentOS 8.1
- Currently consists of:
  - 11 GPU Nodes
  - 33 NVIDIA Ampere A100 (3/node)
  - 14,256 tensor cores (1,296/node)
  - 228,096 CUDA cores (20,736/node)
  - HDR InfiniBand (100 Gbps) Fabric
- Not benchmarked

LUSTRE STORAGE SYSTEM

- High speed parallel file system
- 6.9 PB of storage space
  - /home/<eraider>
  - /lustre/work/<eraider>
  - /lustre/scratch/<eraider>
- 1.0 PB backup system
- **Quota / Backup / Purge per Lustre Area**

Area	Quota	Backup	Purged
/home/<eraider>	300 GB	Yes	No
/lustre/work/<eraider>	700 GB	No	No
/lustre/scratch/<eraider>	None	No	Yes

USING THE BASH SHELL

This section is based upon the TTU HPCC "*Introduction to Linux*" training slides - https://www.depts.ttu.edu/hpcc/about/training.php#intro_linux

WHAT IS THE SHELL?

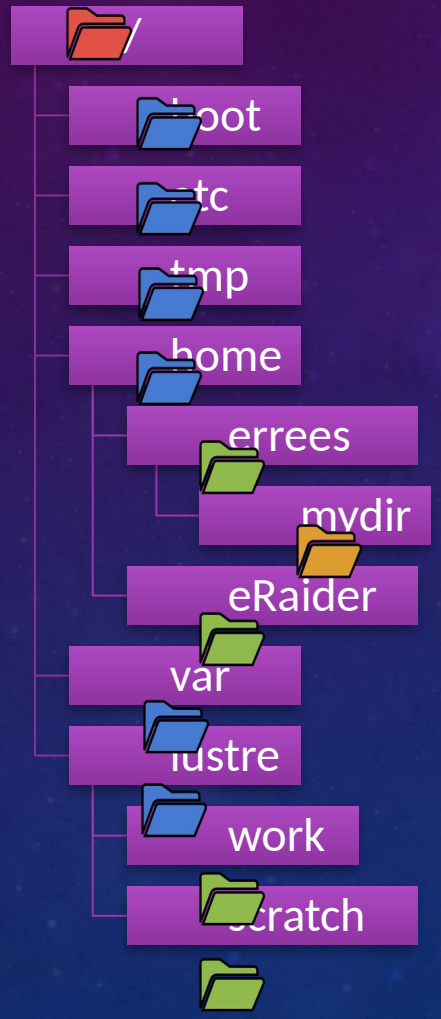
- The shell is a program that takes commands from user's keyboard and passes them to the operating system to execute.
  - The shell runs, by default, in Interactive Mode which is an infinite "read-evaluate-print" loop (REPL)
- There are many shell programs available for Linux:
  - bash, sh, csh, tcsh, ksh, zsh, …
- For this course, we will use bash, but other shells are conceptually similar
- Example:

```
quanah:$ command
```

USING THE BASH SHELL

LINUX DIRECTORIES

UNDERSTANDING LINUX DIRECTORY STRUCTURE



- /
- /boot
- /etc
- /tmp
- /home
- /home/errees
- /home/errees/mydir
- /home/eRaider
- /var
- /lustre
- /lustre/work
- /lustre/scratch

BASIC DIRECTORY OPERATIONS

Where am I?

- **`pwd`** command to print the current working directory

```
quanah:$ pwd
/home/username
```

- **`ls`** command to list contents of the current working directory

```
quanah:$ ls
test1.txt
Mydir1
mydir2
```

BASIC DIRECTORY OPERATIONS

Make/Remove/Go into a directory?

- **`mkdir`** command to make a new directory

```
quanah:$ mkdir my_new_dir
quanah:$ ls
my_new_dir
```

- **`cd`** command to change into a directory

```
quanah:$ cd my_new_dir
```

- **`rmdir`** command to remove an empty directory

```
quanah:$ rmdir my_new_dir
```

BASIC DIRECTORY OPERATIONS

More about **`ls`**. (Commands with options)

- **`ls -a (--all)`** list all files including hidden files

```
quanah:$ ls -a
```

- **`ls -l`** show file details

```
quanah:$ ls -l
```

- Combine multiple options for a command

```
quanah:$ ls -a -l
quanah:$ ls -al
```

SPECIAL DIRECTORY NAMES (CHARACTERS)

- Return to home directory (~)

```
quanah:$ cd
quanah:$ cd ~
quanah:$ cd ~/
```

- Current directory (.)

```
quanah:$ ls .
quanah:$ ls ./
```

- Parent directory (..)

```
quanah:$ cd ..
quanah:$ cd ../
quanah:$ ls ../../
quanah:$ cd ../../mydir/
```

ABSOLUTE VS. RELATIVE PATH

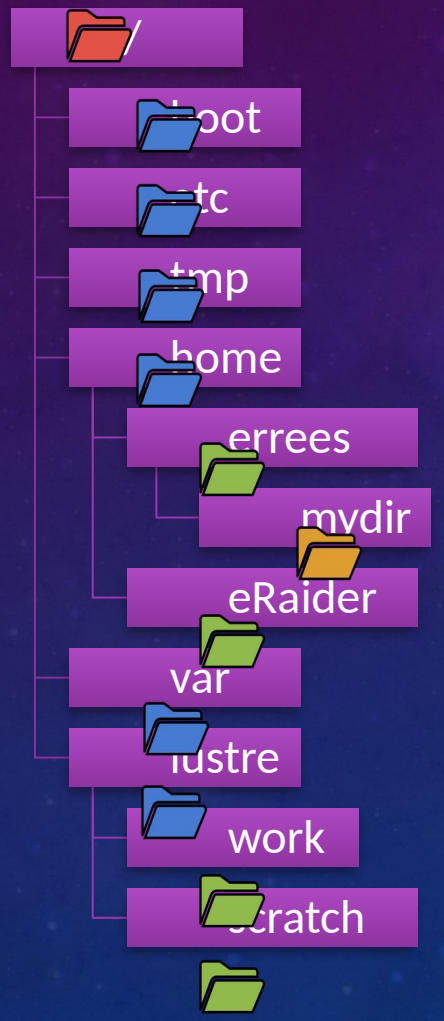
- Absolute Path of a file/directory
  - Defines the path to a file/directory starting from the root of the file system (/)
  - Example:

```
quanah:$ cat /home/errees/mydir/file1.txt
```

- Relative Path of a file/directory
  - Defines the path to a file/directory starting from the current or parent directory.
  - Example

```
quanah:$ cat ../mydir/file1.txt
```

ABSOLUTE VS. RELATIVE PATH (EXAMPLE)



CWD

```
quanah:$ pwd
/home/errees
quanah:$ cd /lustre/work
quanah:$ pwd
/lustre/work
quanah:$ cd ..
quanah:$ pwd
/lustre
quanah:$ cd ../tmp
quanah:$ pwd
/tmp
quanah:$ cd ~/mydir
quanah:$ pwd
/home/errees/mydir
```

USING THE BASH SHELL

LINUX FILES

FILE AND DIRECTORY NAMES

- What characters should be used?
  - Letters and numbers (`[A-Z], [a-z], [0-9]`)
    - File and directory names are case sensitive (e.g. '`M`' and '`m`' are different letters).
  - Periods (**.**) are not required but can be used anywhere in the name
    - Using (.) at the beginning of the file/directory name makes it hidden
    - Using (.) alone will be considered as current working directory

- What characters should be avoided?
  - Avoid spaces in file or directory names
  - Avoid these characters:
    - **( ) " ' ? $ * \ / :**

- A - Z
- a - z
- 0 - 9
- . dot
- - dash
- _ underscore

- **()** parenthesis
- **"** double quotes
- **'** single quote
- **?** Question mark
- **$** dollar sign
- ***** asterisk
- **** back slash
- **/** slash
- **:** colon

BASIC FILE OPERATIONS

- **`touch`** creates an empty file (Not the primary job!)

```
quanah:$ touch my_file
quanah:$ ls
my_file
```

- **`echo`** prints its arguments to standard output (`stdout`)
  - **`>`** redirects the standard output to a file (Creates/Rewrite the file)
  - **`>>`** redirects the standard output to a file (Append to the file)

```
quanah:$ echo "hello world!" > hello.txt
quanah:$ echo "second line" >> hello.txt
```

BASIC FILE OPERATIONS

- **`cat`** shows the contents of a file(s)

```
quanah:$ cat hello.txt
hello world!
second line
```

- **`wc`** counts the number of lines, words, and characters in a text file
  - **`-l`** Only count the number of lines.
  - **`-w`** Only count the number of words.

```
quanah:$ wc hello.txt
 2  4 25 hello.txt
quanah:$ wc -l hello.txt
2 hello.txt
quanah:$ wc -c hello.txt
25 hello.txt
```

BASIC FILE OPERATIONS

- **`cp`** command to make a copy of a file or directory
  - **`-r`** Recursively copy the directories and subdirectories as well

```
quanah:$ ls
hello.txt
quanah:$ cp hello.txt hello2.txt
quanah:$ ls
hello.txt hello2.txt
quanah:$ cp -r ../mydir ./
quanah:$ cp ../mydir2/*  /home/bob/
quanah:$ cp ./*.txt  ~/text_dir/
```

BASIC FILE OPERATIONS

- **mv** command to move a file or directory to a new location
  - Can also be used to rename a file or directory.

```
quanah:$ ls
hello.txt hello2.txt
quanah:$ mv hello2.txt hello_again.txt
quanah:$ ls
hello.txt hello_again.txt
quanah:$ mv mydir/ ../temp/
quanah:$ mv ../docs/*.pdf  ./
```

BASIC FILE OPERATIONS

- **`rm`** command to remove a file or directory permanently
  - **`-r`** Recursively delete the directories and subdirectories as well
  - This command is **extremely dangerous**. Removing files and directories with this command is permanent.
    - There is no recycle bin to undo this action.

```
quanah:$ ls
hello.txt hello_again.txt
quanah:$ rm hello_again.txt
quanah:$ ls
hello.txt
quanah:$ rm ../mydir/*
quanah:$ rm -r ../mydir/
```

OUTPUT REDIRECTION

- So far, all the commands that we used sent their output to the screen.
- We can control this in Linux using redirection symbols
  - `<` redirects to the standard input
  - `>` redirects the standard output
  - `>>` appends the standard output
  - `2>` redirects the standard error
  - `&>` redirects the standard output and standard error
  - `2>&1` redirects standard error to where the standard output is set
  - `1>&2` redirects the standard output to where the standard error is set
  - `|` pass the output of one command to another

USING THE BASH SHELL

ESSENTIAL LINUX COMMANDS

LINUX ESSENTIAL COMMANDS

- **man** command to search and open the manual page for a Linux command

```
quanah:$ man ls
quanah:$ man cp
```

- Once inside the man program use the following shortcuts:
  - **/word** Search the man page for the text 'word'
    - **n** search for the next found match
    - **N** search for the previous found match
  - **g** Go to the beginning of the manual page
  - **G** Go to the end of the manual page
  - **q** Exit (quit)

LINUX ESSENTIAL COMMANDS

- **`less`** command for paging through text one screenful at a time

```
quanah:$ less README
quanah:$ cat README | less
```

- **`grep`** command for selecting text within a file based on a given pattern.
  - **`-i`** Ignore case distinctions in both the PATTERN and the input file(s)
  - **`-v`** Invert the sense of matching, to select non-matching lines

```
quanah:$ grep "is" ../exercise1/test1.txt
Exercise:
This is a test file
here is the last line
```

LINUX ESSENTIAL COMMANDS

- **head** command to see the first lines of a file (10 lines by default)
  - **-n <num>** Display the first <num> lines of a file/output.

```
quanah:$ head -n 15 README
quanah:$ cat README | head -n 3
```

- **tail** command to see the last lines of a file (10 lines by default)
  - **-n <num>** Display the last <num> lines of a file/output

```
quanah:$ tail -n 15 README
quanah:$ cat README | tail -n 3
```

LINUX ESSENTIAL COMMANDS

- **`history`** command to view the commands you have run in the past
  - The `history` command shows the previous 1000 commands run along with all commands run in the current shell.
  - Your command history is also saved in the `.bash_history` file in your home directory.
    - This only has the past 1000 commands run in previous shells.

```
quanah:$ history
quanah:$ history | less
quanah:$ history | tail -n 15
quanah:$ history | grep cp
```

LINUX ESSENTIAL COMMANDS - SED

- **`sed`** command to perform pattern matching and replacement.
  - Example uses:
    - Text substitution,
    - Selective printing of text files,
    - In-place editing of text files,
    - Non-interactive editing of text files.

```
quanah:$ cat text.txt
This is the first line.
I am the second line.
I am one after the second line.
This is the fourth line.
Look out, I'm the 5th line.
Pity, I am the final line.
```

LINUX ESSENTIAL COMMANDS - SED

- **`sed`** command can be passed multiple commands.
  - **`-e <command string>`** Perform specific commands.
    - **`d`** Delete line(s)
    - **`s`** Perform a REGEX replacement

- To instruct SED to remove the first, second, and fifth line of the file, we will issue three separate commands.
  - SED begins counting lines at #1 – so the first line is 1, not 0.
  - This only deletes the lines from output. The lines are not deleted from the source file.

```
quanah:$ sed -e '1d' -e '2d' -e '5d' text.txt
I am one after the second line.
This is the fourth line.
Pity, I am the final line.
```

LINUX ESSENTIAL COMMANDS - SED

- To instruct SED to remove lines 2 – 4, we can issue the following command.

```
quanah:$ sed -e '2,4d' text.txt
This is the first line.
Look out, I'm the 5th line.
Pity, I am the final line.
```

- We can also specify an address range using text patterns. The following pattern will delete all lines between the patterns “second” and “out”.

```
quanah:$ sed -e '/second/,/out/d' text.txt
This is the first line.
Pity, I am the final line.
```

LINUX ESSENTIAL COMMANDS - SED

- **`sed`** command can be passed multiple commands.
  - **`-e <command string>`** Perform specific commands.
    - **`d`** Delete line(s)
    - **`s`** Perform a REGEX replacement

- The **`s`** pattern replacement is of the form:
  - `[address1[,address2]]s/pattern/replacement/[flags]`
  - The 'g' flag is used to make the replacement global

```
quanah:$ echo "Hello World"
Hello World
quanah:$ echo "Hello World" | sed 's/World/Again/g'
Hello Again
```

LINUX ESSENTIAL COMMANDS - SED

- To instruct SED to replace “line” with “test” on all lines of the file *text.txt*.

```
quanah:$ sed 's/line/test/g' text.txt
This is the first test.
I am the second test.
I am one after the second test.
This is the fourth test.
Look out, I'm the 5th test.
Pity, I am the final test.
```

LINUX ESSENTIAL COMMANDS - SED

- To instruct SED to replace “line” with “test” on all lines with the word “second” in them of the file *text.txt*.

```
quanah:$ sed '/second/ s/line/test/g' text.txt
This is the first line.
I am the second test.
I am one after the second test.
This is the fourth line.
Look out, I'm the 5th line.
Pity, I am the final line.
```

LINUX ESSENTIAL COMMANDS - AWK

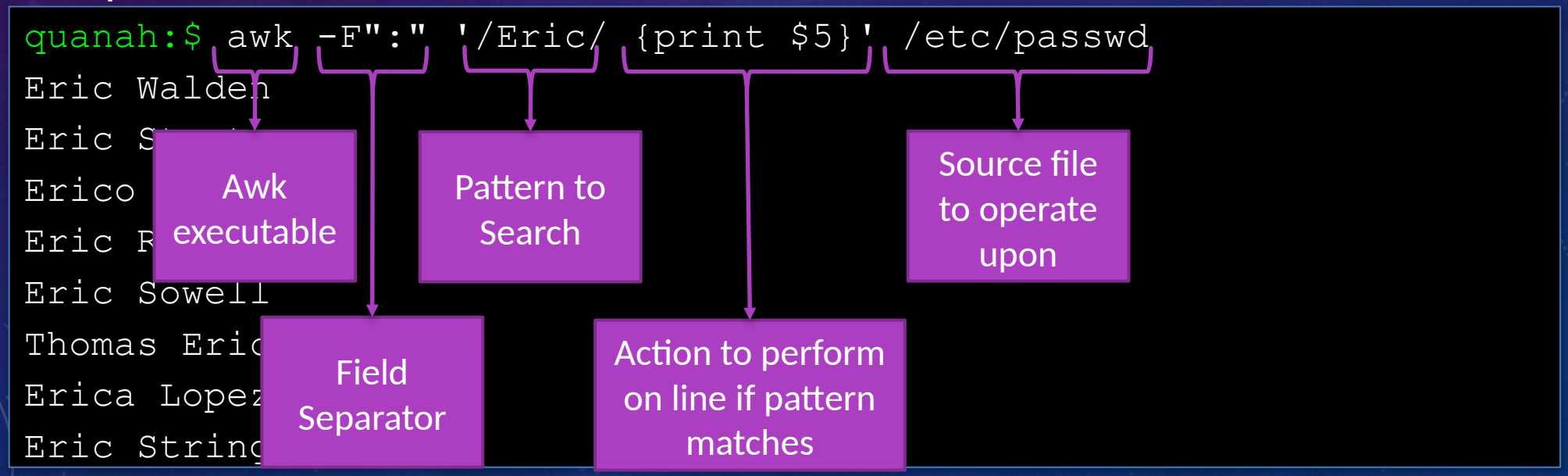
- What is awk?
  - Simply put, awk is a programming language designed to search for, match patterns, and perform actions on files.

- How it works
  - awk reads from a file or from standard input and outputs to standard output.
  - awk recognizes the concepts of files, records, and fields.
  - A file consists of records, with each record being lines. One line = one record.
  - A record consists of fields, which by default are separated by whitespace.
  - Field number 1 is accessed using $1, field 2 with $2, etc…
  - $0 refers to the entire record.

LINUX ESSENTIAL COMMANDS - AWK

- **`awk`** command can be passed multiple commands.
  - **`-F <string>`** Defines the field separator to use between fields.

- Example:



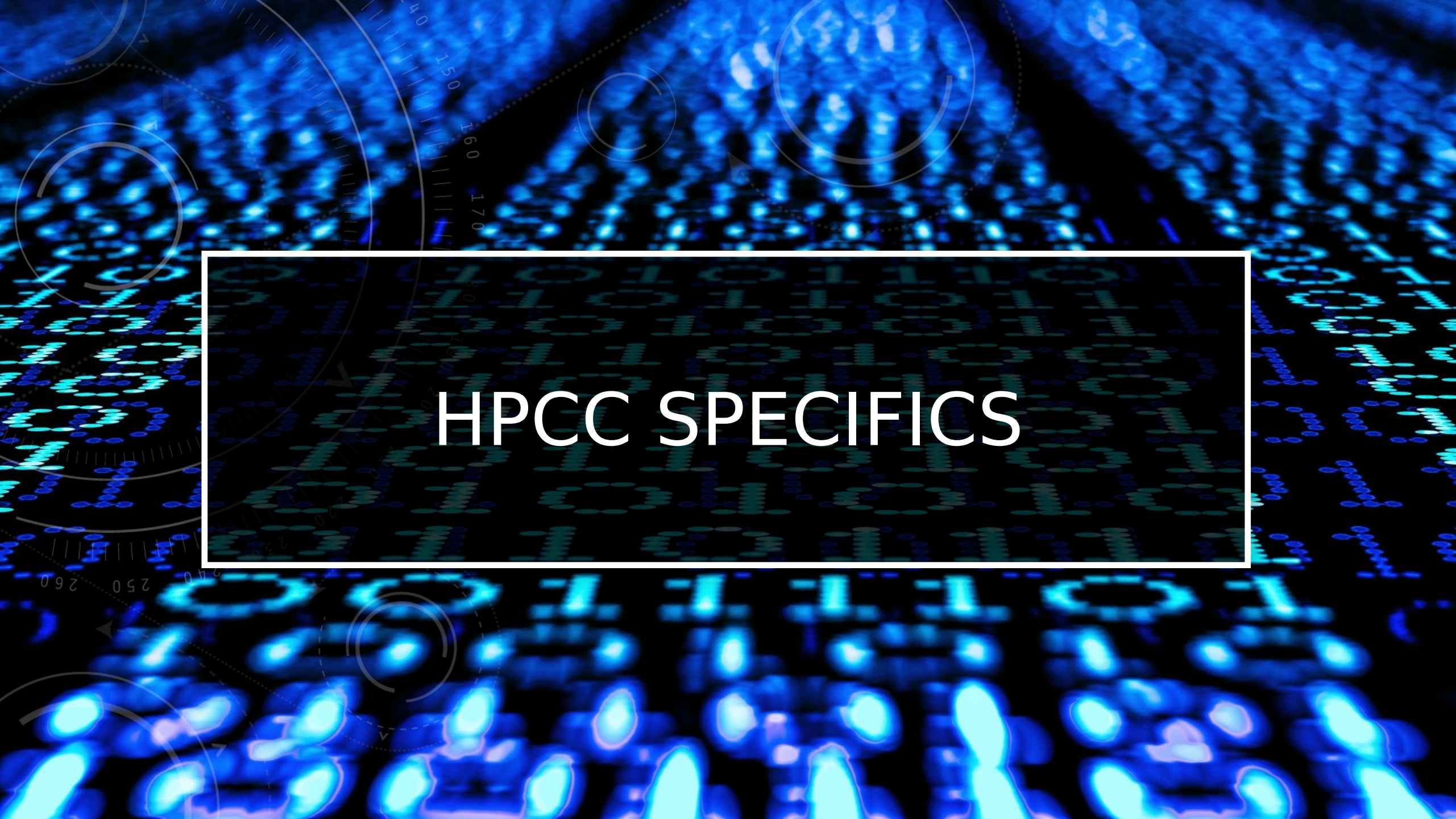
LINUX ESSENTIAL COMMANDS - AWK

- Another example:

```
quanah:$ cat text.txt
This is the first line.
I am the second line.
I am one after the second line.
This is the fourth line.
Look out, I'm the 5th line.
Pity, I am the final line.
quanah:$ grep "is" text.txt
This is the first line.
This is the fourth line.
quanah:$ grep "is" text.txt | awk '{print $4}'
first
fourth
```

LINUX ESSENTIAL COMMANDS

- Required sed tutorial.
  - https://www.tutorialspoint.com/sed/index.htm
- Required awk tutorial
  - https://www.tutorialspoint.com/awk/index.htm
- Work through these tutorials and be sure you fully understand how sed and awk work, with a special emphasis on the specific commands we have used in these slides.


HPCC SPECIFICS

GETTING STARTED

- User Guides
  - The best place to get detailed information regarding using the clusters
  - http://www.depts.ttu.edu/hpcc/userguides/index.php
- Queue Status
  - Current view of the pending and running jobs on each cluster
  - https://www.depts.ttu.edu/hpcc/status/slurm_web.php

GETTING STARTED

- Logging into HPCC Resources
  - User Guide: http://tinyurl.com/ttu-hpcc-login
- On or Off Campus?
  - On Campus: Wired TTU network & TTUnet wireless network
  - Off Campus: Any other network connection
- Logging in from Off Campus
  - Log in via the SSH gateway
  - Establish a VPN - https://goo.gl/4LbuWG
  - Neither system is owned or maintained by HPCC

LOGGING IN

Upcoming or Current Downtimes

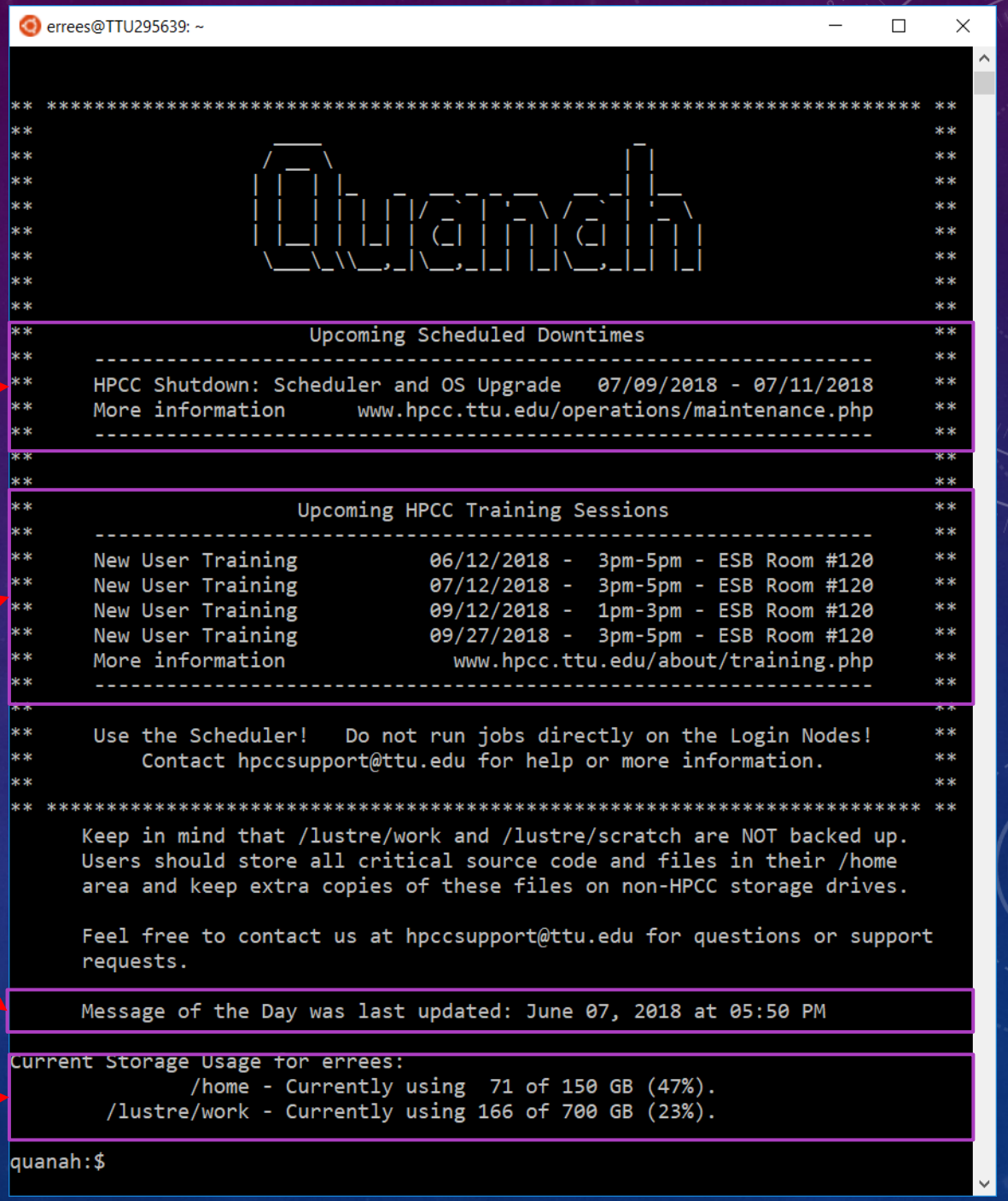
Always good to check this every time you log in.

Upcoming HPCC Training Sessions

Not always just “New User Training”

MoTD Last Updated Date/Time

Current Storage Usage



USER ENVIRONMENTS

- .bashrc
  - Bash script that runs whenever you start an interactive login.
  - Often used to set up user environments.
  - While you may add to your .bashrc file, **do not remove any default settings**!
- Modules
  - The primary way to change your user environment.

User Guide: http://www.depts.ttu.edu/hpcc/userguides/general_guides/software_environment.php

MODULES

- Modules commands:
  - module avail
  - module list
  - module load <module_name>
  - module unload <module_name>
  - module spider <keyword>
  - module purge

```
quanah:$ module avail

------------------------------------------ /opt/apps/nfs/module/modulefiles ------------------------------------------
   gnu/5.4.0    gurobi/v751    intel/17.3.191    java/1.8.121    matlab/R2017b    perl/5.16.3

Use "module spider" to find all possible modules.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".

quanah:$ module spider python

----------------------------------------------------------------------------------------------------------------
  python:
----------------------------------------------------------------------------------------------------------------
    Description:
      Python-2.7.9 compiled with Intel

     Versions:
        python/2.7.9-gnu
        python/2.7.9-intel

----------------------------------------------------------------------------------------------------------------
  For detailed information about a specific "python" module (including how to load the modules) use the
name.
  For example:

     $ module spider python/2.7.9-intel
----------------------------------------------------------------------------------------------------------------
```

TRANSFERRING DATA

- To transfer data between the HPCC and your personal devices you must make use of Globus Connect.

- User Guide
  - https://www.depts.ttu.edu/hpcc/userguides/general_guides/file_transfer.php#transfer_pc_to_HPCC

HOW TO SCHEDULE JOBS

- Watch the HPCC Tutorial on how to write submission scripts for use with the HPCC systems.
- Slide Decks
  - Part 1 (Optional) – LINK
  - Part 2 (Required) – LINK
- Videos:
  - Part 1 (Optional) – LINK
  - Part 2 (Required) – LINK

PROTECTING YOUR WORK

LOCK DOWN PERMISSIONS TO YOUR DIRECTORIES

- Setting your Home/Work/Scratch Areas to viewable only to you:
  - Run the following commands:
    - `chmod 700 /home/$USER`
    - `chmod 700 /lustre/work/$USER`
    - `chmod 700 /lustre/scratch/$USER`

  - You can run the three commands above without any changes (copy and paste).
  - If typing the command, keep in mind the entire command is case sensitive.
  - Keep in mind, you are responsible for protecting your work within the HPCC.


BASH SCRIPTING

QUOTING AND COMMENTS

- Single Quotes
  - ‘something’ – Preserve the literal string.
- Double Quotes
  - “something” – Allow for expansion of $ variables.
- $ Escaped
  - $’Hello\nWorld’ – Allows for the use of escape strings such as \n and \t.
- # Comments
  - #Comment – Allows for comments to be inserted into code.

COMBINING LISTS OF COMMANDS

- **`cmd1 ; cmd2 ; … ; cmdN`**
  - Execute commands sequentially

- **`cmd1 &`**
  - Execute command asynchronously (spawn a new thread)

- **`cmd1 && cmd2`**
  - Execute cmd2 iff cmd1 has an exit code of 0.

- **`cmd1 || cmd2`**
  - Execute cmd2 iff cmd1 has an exit code that is non-zero.

COMMANDS AND PIPELINES

- A simple command is a sequence of words
  - First word defines the command
  - Can be combined with &&, ||, ;, etc…
  - `echo "Hello" && echo "World"`

- A pipeline is a sequence of commands
  - Each command reads the previous commands' output.
  - `echo "hello world" | wc -c`
    - Prints "hello world" and passes that string into the wc command.

VARIABLES AND EXPRESSIONS

- Variables are placeholders for a value.
  - The shell does variable substitution during interpretation.

- **`$var`** or **`${var}`** is value of variable

- Assignment is performed using **`var=value`**
  - no space before or after the equal sign!
  - Also, **`let "x = 17"`** or **`let "b = b + 10"`**

- Variables are untyped, their "data type" is interpreted based on context
  - Uninitialized variables have no value

ENVIRONMENT VARIABLES

- Shell variables are generally not visible to programs

- Environment
  - List of name/value pairs passed to sub-processes
  - All environment variables are also shell variables, but not vice versa
  - **`env`** command used to print the environment to standard output.

- Make variables visible to processes with export, as in
  - **`export foo`**
  - **`export foo=17`**

SHELL VARIABLES

- **`${N}`**
  - Retrieve the current value of the variable ‘N’.
- **`$$`**
  - Special symbol, retrieves the process ID assigned to the currently running application.
- **`$?`**
  - Special symbol, retrieves the previous command’s exit status

- Standard environment variables include:
  - **`$HOME`** User’s home directory
  - **`$PATH`** List of directories to search
  - **`$USER`** User’s username
  - **`$TZ`** User’s timezone (e.g., US/Eastern)

LOOPING CONSTRUCTS

- Looping constructs are similar to C/Java/Python constructs, but with commands:
  - **`until` *`test-commands`*`;` `do` *`consequent-commands`*`;` `done`**
  - **`while` *`test-commands`*`;` `do` *`consequent-commands`*`;` `done`**
  - **`for` *`name`* `[in` *`words`* `...];` `do` *`commands`*`;` `done`**

- Can be written on single a single line (shown above) or across separate lines (similar to C).

- The **`break`** and **`continue`** keywords can be used to control the loops
  - They behave as you would expect.

BRANCH CONSTRUCTS

- Branch constructs are similar to C/Java/Python constructs, but with commands:
  - **if *test-commands*; then**
    consequent-commands;
    [elif more-test-commands; then
    more-consequents;]
    [else *alternate-consequents*;]
    fi

BRANCH CONSTRUCTS – TEST CONDITIONS

- **`-eq`**
  - Tests “is equal to”
  - Example: `if [ "$a" -eq "$b" ]`
- **`-ne`**
  - Tests “is not equal to”
  - Example: `if [ "$a" -ne "$b" ]`
- **`-gt`**
  - Tests “is greater than”
  - Example: `if [ "$a" -gt "$b" ]`
- **`-ge`**
  - Tests “is greater than or equal to”
  - Example: `if [ "$a" -ge "$b" ]`
- **`-lt`**
  - Tests “is less than”
  - Example: `if [ "$a" -lt "$b" ]`
- **`-le`**
  - Tests “is less than or equal to”
  - Example: `if [ "$a" -le "$b" ]`

ARITHMETIC EXPRESSIONS

- Arithmetic Expansion
  - The preferable way to do math in Bash is to use shell arithmetic expansion.
  - The built-in capability evaluates math expressions and returns the result.
  - The syntax for arithmetic expansions is: `$((expression))`

- Example:
  - `count=$((count+1))`
  - `$((count++))`

RESOURCES

- Example BASH script
  - https://replit.com/@errees/BashExamples#main.sh
- Advanced Bash-Scripting Guide
  - https://tldp.org/LDP/abs/html/