PACE - Linux 101

Marian Zvada, MSc, MBA
Partnership for an Advanced Computing Environment at GT
September 14, 2022

Slides and content by:
Aaron Jezghani, Eric Coulter, and Marian Zvada

Outline

- High performance computing at Georgia Tech
- What is Linux and why use it?
- What is bash?
- Using the command line – basic commands, help, and editing files
- Environment variables
- Scripting and command automation
- Session customization
- Useful material
- **Movement/Screen break after ~1h**

Georgia Tech

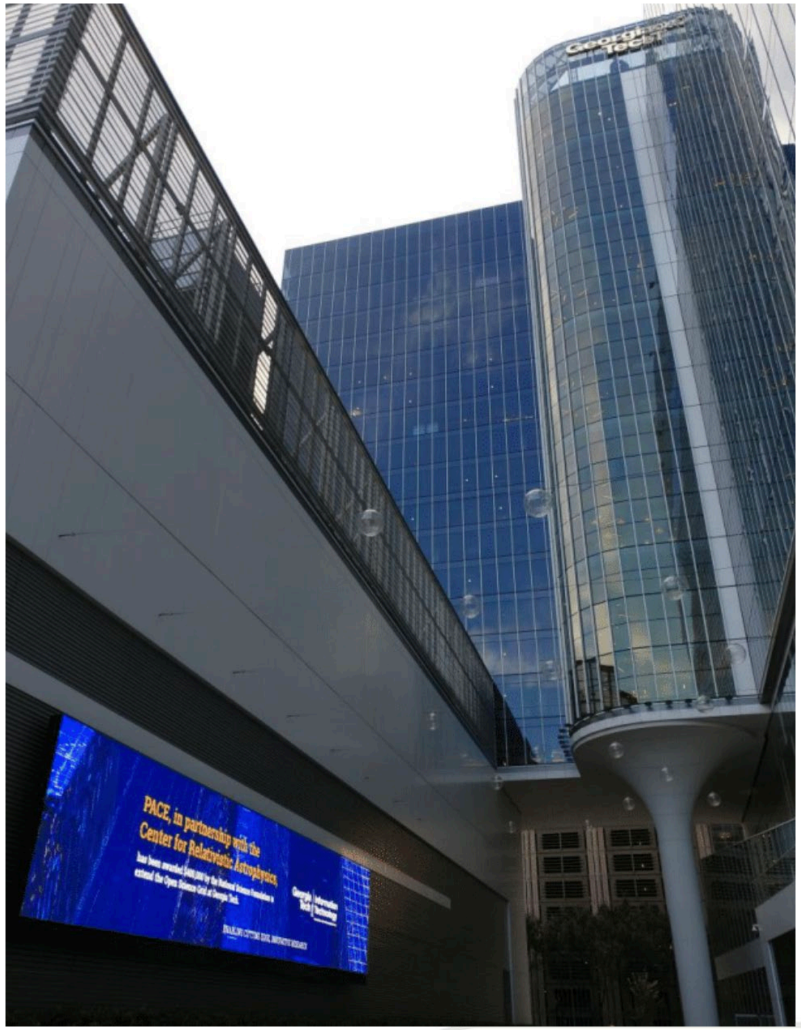
Partnership for an Advanced Computing Environment

- PACE provides faculty participants sustainable leading-edge advanced research computing resources with technical support services, infrastructure, software, and more.
- **Free tier provides any GT academic or research faculty the equivalent of 10,000 CPU-hours per month on a 192GB compute node and 1 TB of project storage at no cost on Phoenix HPC Cluster**
- Virtual tour of Coda datacenter hosting PACE resources: https://pace.gatech.edu/coda-datacenter-360-virtual-tour
- PACE is a service which operates as a Cost Center.
- Participation in PACE and the participation calculator: https://pace.gatech.edu/participation
  - Learn more about paid options for compute & storage

Sign up for our hands-on workshops taught by PACE Research Scientists: **PACE Clusters Orientation**, **PACE OSG Orientation**, **Linux 102**, **Python 101: Intro to Data Analysis with NumPy**, **Git 101**, **Optimization 101**, and **Applications of Machine Learning**

For more detailed questions about your work, attend our weekly PACE Consulting Session

https://pace.gatech.edu/training


Georgia Tech
PACE, in partnership with the Center for Relativistic Astrophysics

Georgia Tech

PACE Team

Partnership for an **A**dvanced **C**omputing **E**nvironment provides faculty participants sustainable leading-edge advanced research computing resources with technical support services, infrastructure, software, and more.

Please attend the PACE Clusters Orientation for more information: https://pace.gatech.edu/content/orientation

Ken Suda, Sr. Applications developer

Craig Moseley, Team lead Customer Exp

Michael Brandon, Management Support

Pam Buffington, Director

Allan Metts, Digital Learning Principal Architect

Matt Guidry, Sr. Systems Support Eng

Greg Beyer, Sr. Systems Support Eng

Paul Manno, RBW-IT Professional

Ruben Lara, Sr. Systems Support Mgr

Dee Womack, Systems support Eng

Peter Wan, Sr. Systems Support Eng

Dan (Ann) Zhou, Storage Architect

Aaron Jezghani, PhD, Scheduler Architect

Semir Sarajlic, Team Lead, ASE

Trever Nightingale, Cloud Architect

Ronald Rahaman, Research Software Eng

Eric Coulter, PhD, CI Integration Architect

Deepa Phanish, PhD Research Computing Facilitator

Fang (Cherry) Liu, PhD, Lead Research Software Eng

Marian Zvada, Research Computing Facilitator

Jeffrey Valdez, Research Computing Facilitator

Michael Weiner, PhD, Research Computing Facilitator

IO500

Open Science Grid

NSF

XSEDE
Extreme Science and Engineering Discovery Environment

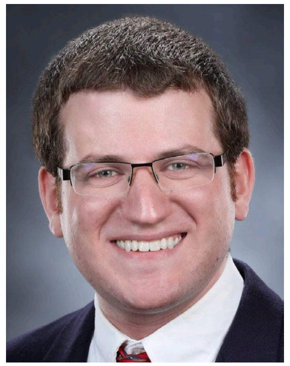
Georgia Tech

PACE-RCF

Meet the **R**esearch **C**omputing **F**acilitation team! We interact with the advanced computing research community at Georgia Tech, respond to a wide range of requests submitted by faculty & student researchers.

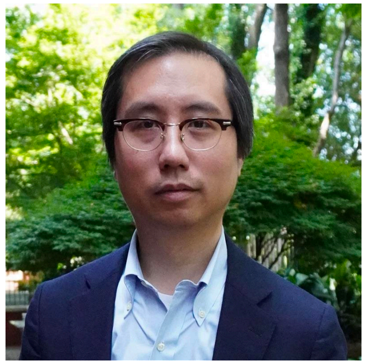
Fang (Cherry) Liu, PhD
RCF Team Lead

"HPC brings you to an amazing parallel universe!"



Michael D. Weiner, PhD
Research Computing Facilitator

"First used HPC in my research, still in awe of its power after over a decade!"



Jeffrey Valdez, MS, MBE
Research Computing Facilitator

"Always happy to help the research community at GT!"

Deepa Phanish, PhD
Research Computing Facilitator

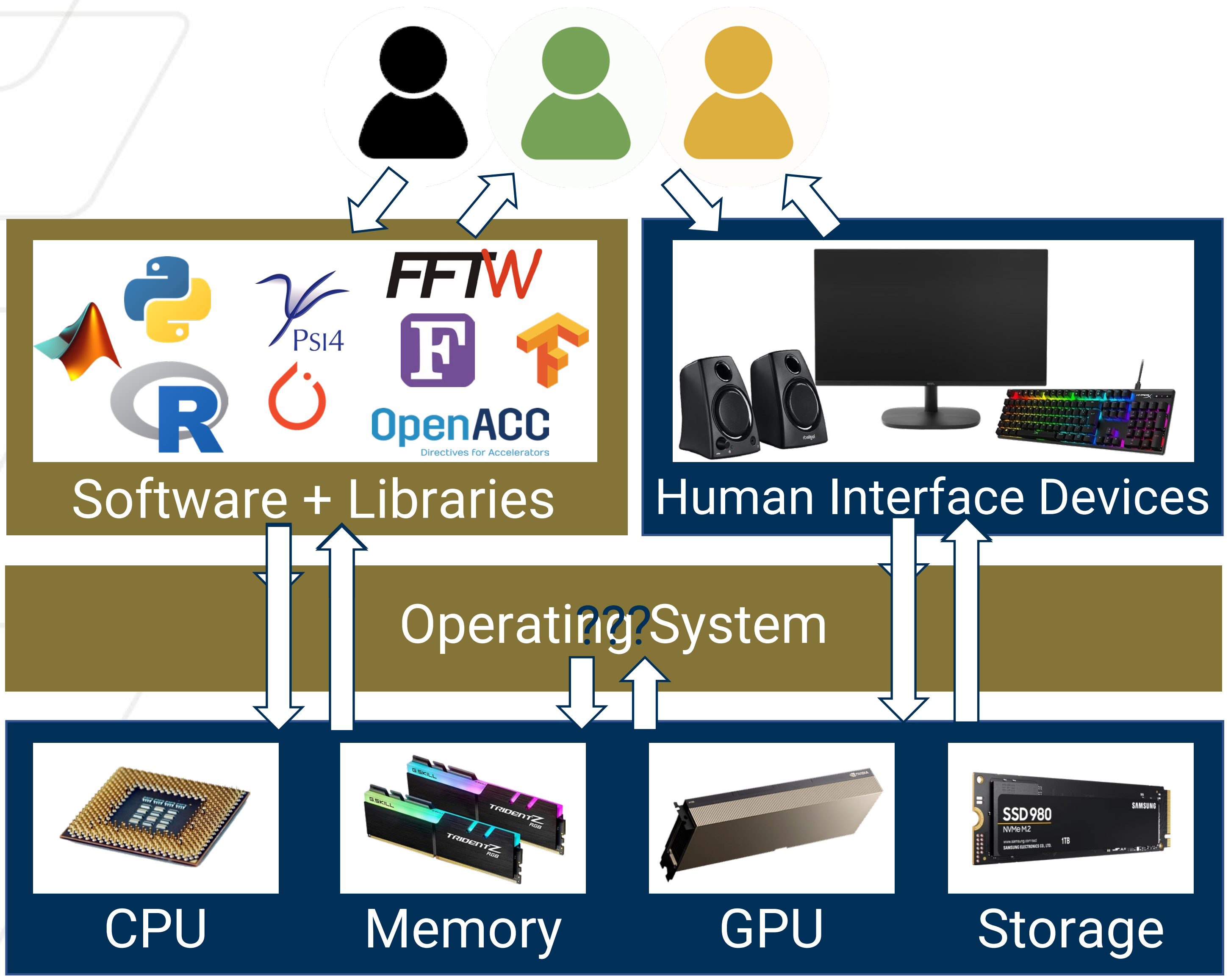
"I love GT! Just jumped over the PACE fence!"

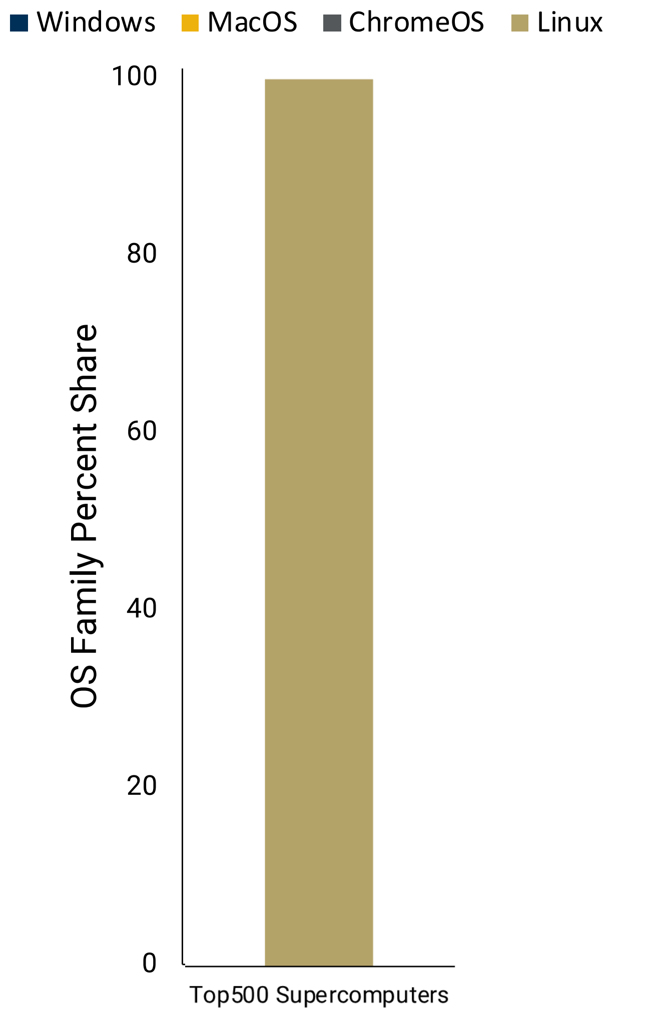
Marian Zvada, MSc, MBA
Research Computing Facilitator

"I like to refine business and enhance PACE customer experience. Always ready to play chess, too!"

Georgia Tech

What is an Operating System?

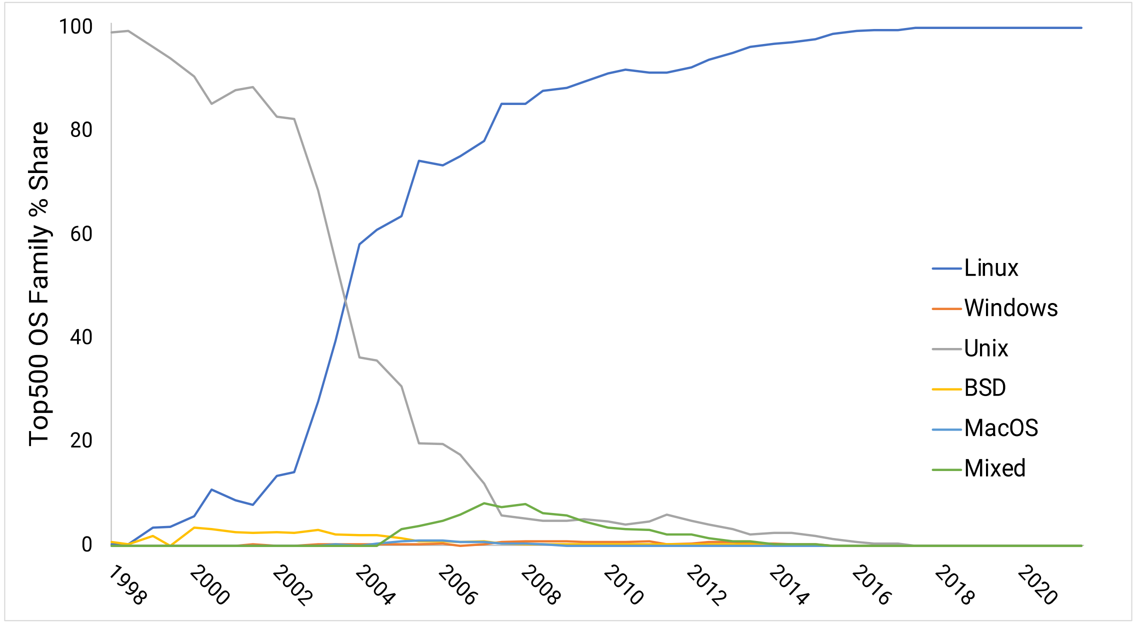




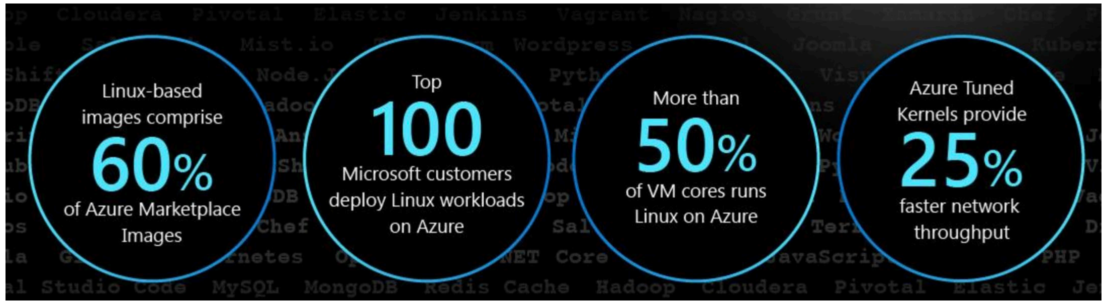
https://gs.statcounter.com/os-market-share/desktop/worldwide https://top500.org/statistics/

Why use Linux?

- It is free and open source
  - Exceptions for enterprise systems, but that's a choice
  - With a little know-how, you can customize everything!
- It is efficient
  - Better background process management
  - Less "nannying"
  - Faster filesystem (although you probably won't notice)
- It is reliable
  - Reboots not required!
  - Easy access to system services



https://top500.org/statistics/overtime



Linux-based images comprise 60% of Azure Marketplace Images
Top 100 Microsoft customers deploy Linux workloads on Azure
More than 50% of VM cores runs Linux on Azure
Azure Tuned Kernels provide 25% faster network throughput

https://www.linkedin.com/posts/adirron_linuxonazure-progress-activity-6665502370795003905-DY1G

How do I “use Linux”?

- Install it!
  - Distros like Ubuntu, Fedora, and openSUSE are free
  - On Windows, try out the Windows Subsystem for Linux (WSL)
- Try a browser-based emulator!
  - https://bellard.org/jslinux/
  - https://cocalc.com/features/terminal
  - https://distrotest.net/index.php
- Access a remote cluster (like those managed by PACE)!
  - OpenSSH – available by default on most OSs (including Windows 10 > 18.03)
  - 3rd Party apps like PuTTY, MobaXterm, XSHELL, etc.

Using OpenSSH

- OpenSSH is a suite of secure networking tools built on the SSH protocol, and includes:
  - **ssh**: the remote login client
  - **scp**: a utility to securely copy files between a local and a remote host, or between two remote hosts
  - **sftp**: another secure file transfer utility based on FTP, so it can also do things like list files
  - **ssh-keygen**: a utility to inspect/generate keys for user and host authentication
    - On clusters like those managed by PACE, keys are generated that allow your jobs to run across multiple servers with tools like MPI
  - **ssh-copy-id**: gathers available keys and uses ssh to make them available for authentication on the remote host

Using SSH to Access PACE-ICE

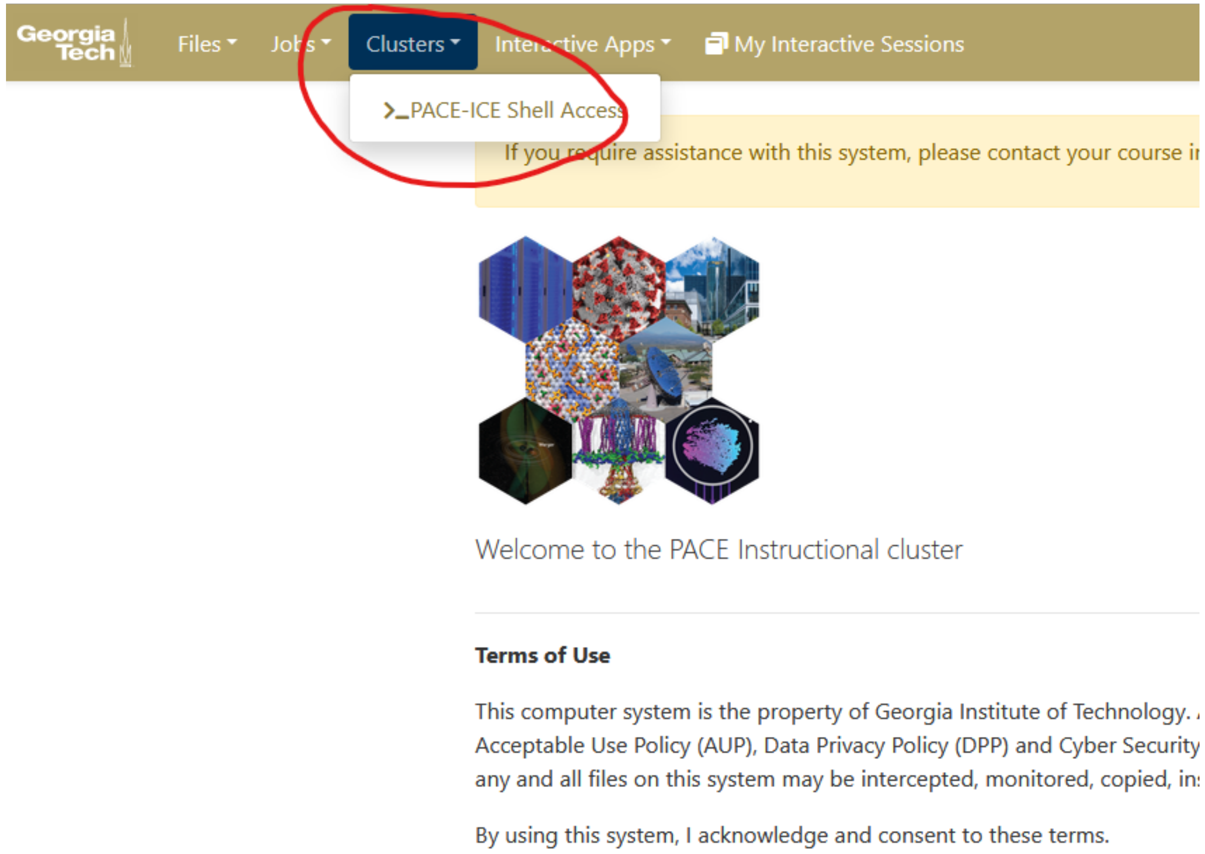
- To login to a remote cluster like PACE-ICE, use ssh
  - Command looks like `ssh <username>@<remote_host>`
  - For PACE systems, use your GT username
  - PACE-ICE login hostname is ‘pace-ice.pace.gatech.edu’
  - When prompted, enter your GT password (you won’t see anything as you type)
  - Congratulations! You’re logged in to the cluster
  - You will need to be on VPN
- If you plan on using graphical applications, you’ll want to enable X11 forwarding
  - You’ll need an X11 client (Xming or VcXsrv on Windows, Xquartz on MacOS)
  - Modify the above by including ‘-X’: `ssh -X <username>@<remote_host>`

Using Open OnDemand to Access PACE -ICE

- You need to be connected to Georgia Tech's VPN to access the resources
  - Browser-based VPN, see https://vpn.gatech.edu/global-protect/login.esp
  - GlobalProtect client (newer option, recommended) For information about VPN access, see http://docs.pace.gatech.edu/gettingStarted/vpn/
- Navigate to:
  https://ondemand-pace-ice.pace.gatech.edu
  and login with your GT credentials

Using Open OnDemand to Access PACE -ICE

- Start an interactive shell session by clicking 'Clusters' and selecting '>_PACE-ICE Shell Access'



If you require assistance with this system, please contact your course i

Welcome to the PACE Instructional cluster

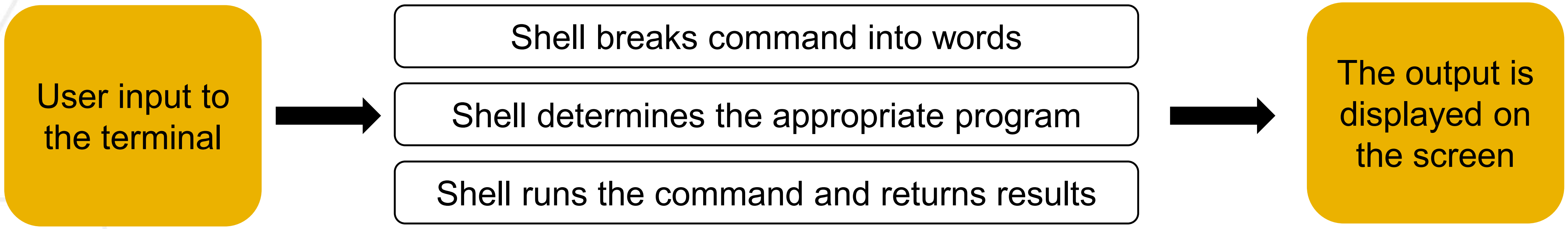
Terms of Use

This computer system is the property of Georgia Institute of Technology.
Acceptable Use Policy (AUP), Data Privacy Policy (DPP) and Cyber Security
any and all files on this system may be intercepted, monitored, copied, in

By using this system, I acknowledge and consent to these terms.

What is a Shell…and What is Bash?

- The **B**ourne-**A**gain **SH**ell is the command language interpreter for most Linux distributions
  - Reads input from the user
  - Interprets the input into instructions, separating words according to the internal field separator , `$IFS` (defaults to `<space><tab><newline>`)
  - Executes commands, interactively or non-interactively and synchronously or asynchronously
  - Displays the results for the user in a readable format


User input to the terminal
Shell breaks command into words
Shell determines the appropriate program
Shell runs the command and returns results
The output is displayed on the screen

- Contains a large number of built-in commands
  - These would be impossible or inconvenient to obtain separately
  - Generally these should not (or cannot!) be implemented in other applications
- Can be customized with functions and aliases for ease of use

Command Line Keyboard Shortcuts

- `Tab`: Attempts to complete a command
- `Ctrl+A`: move cursor to the beginning of the command line
- `Ctrl+E`: move cursor to the end of the command line
- `Ctrl+K`: delete all characters after the cursor
- `Ctrl+U`: delete all characters before the cursor
- `Ctrl+W`: delete the word before the cursor
- `Ctrl+L`: clear this terminal (equivalent to typing `clear`)
- `Ctrl+C`: end a running program and return to the prompt
- `Ctrl+D`: log out of the current shell session (equivalent to typing `exit` or `logout`)
- `Ctrl+Z`: suspend a program

Bash Commands for Exploring Directories

- `pwd` shows you the path of the current working directory

```
[gburdell3@login-pace ~]$ pwd
/storage/home/hpace1/gburdell3
```

- `ls` shows the contents of the current directory

```
[gburdell3@login-pace ~]$ ls
dir1  file1  file2  p-gburdell3-0  scratch
```

- The color option is enabled with `--color=auto` (automatic on PACE):
  - White = file, blue = directory, green = executable, red* = archive file, cyan = symbolic link (references another file/directory)
    - Red with black background is broken link (black terminal = background always black!)
- Use the option `-a` to show all files (including hidden)

Bash Commands for Navigation

- `cd` changes the working directory to the target directory

```
[gburdell3@login-pace ~]$ pwd
/storage/home/hpace1/gburdell3
[gburdell3@login-pace ~]$ cd p-gburdell3-0/
[gburdell3@login-pace p-gburdell3-0]$ pwd
/storage/home/hpace1/gburdell3/p-gburdell3-0
```

  - If no target directory is provided to `cd`, it will return you to your home (~/)
- Each directory contains two special paths, `.` and `..`
  - `.` points to the current directory
  - `..` points to the current directory's parent (one level up)

```
[gburdell3@login-pace p-gburdell3-0]$ cd ..
[gburdell3@login-pace ~]$ pwd
/storage/home/hpace1/gburdell3
```

Georgia Tech

Bash Commands for Exploring Directories (cont'd)

- Use the `-l` option to show the long listing format

```
[gburdell3@login-pace ~]$ ls -l
lrwxrwxrwx 1 root       gtperson 35 May 21 15:58 p-gburdell3-0 ->
/storage/project/0/gburdell3/
drwxr-xr-x 3 gburdell3  gtperson  3 May 21 14:20 dir1
lrwxrwxrwx 1 root       gtperson 28 May 21 15:54 scratch ->
/storage/scratch/9/gburdell3/
-rw-r--r-- 1 gburdell3  gtperson 12 Jan 22 02:17 file1
-rw-r--r-- 1 gburdell3  gtperson 30 Jan 22 14:56 file2
```

- Use `chmod` to change the permissions for files and directories
  - User, Group, Other: the order in which permissions are applied
    - 644 = user can read+write, group+other can read = -rw-r—r--
    - 755 = user can read+write+execute, group+other can read+execute = -rwxr-xr-x
  - 4 (read), 2 (write), 1 (execute); multiple permissions granted by summing
    - 6 = read+write, 5 = read+execute, 7 = read+write+execute

Bash Commands for Files

- touch can be used to create new, empty files
- mkdir can be used to create new, empty directories

```
[gburdell3@login-pace ~]$ ls
dir1  file1  file2  p-gburdell3-0  scratch
[gburdell3@login-pace ~]$ touch new_file && mkdir new_dir
[gburdell3@login-pace ~]$ ls
dir1  file1  file2  new_dir  new_file  p-gburdell3-0  scratch
```

- file can be used to see what a file "is"

```
[gburdell3@login-pace ~]$ file p-gburdell3-0
p-gburdell3-0: symbolic link to `/storage/coda1/project/0/gburdell3/'
[gburdell3@login-pace ~]$ file file1
file1: ASCII text
```

How Do I Edit Files from the Terminal?

- **N**ano's **ANO**ther editor, "An enhanced free Pico clone"
  - Small, free, and friendly editor
  - Simple in features = easier to use
  - Commands are displayed on screen for easy reference
- **Vi IM**proved, "A Programmer's Text Editor"
  - Operates in two modes – "insert" and "command line"
  - In "insert" mode, `-- INSERT --` is displayed at the bottom of the terminal
  - A thorough tutorial can be accessed via `vimtutor`
- **E**ditor **MAC**ros, "The extensible, customizable, self-documenting real-time display editor"
  - Can write in many languages (human and programming), as well as compile, run, and test programs
  - Commands involve the CONTROL (`Ctrl`) and META (`Esc`) keys
  - A thorough tutorial can accessed by starting emacs and typing `C-h t`

Bash Commands for Files (cont’d)

- `mv` is used to modify a file’s path (including renaming)

```
[gburdell3@login-pace ~]$ ls
dir1  file1  file2  new_dir  new_file  p-gburdell3-0  scratch
[gburdell3@login-pace ~]$ mv new_file renamed_file
[gburdell3@login-pace ~]$ ls
dir1  file1  file2  new_dir  renamed_file  p-gburdell3-0  scratch
[gburdell3@login-pace ~]$ mv renamed_file new_dir/
[gburdell3@login-pace ~]$ mv file1 new_dir/moved_and_renamed
[gburdell3@login-pace ~]$ ls
dir1  file2  new_dir  p-gburdell3-0  scratch
[gburdell3@login-pace ~]$ ls new_dir
moved_and_renamed  renamed_file
```

Bash Commands for Files (cont’d)

- `cp` is used to copy files (to copy directories, you’ll need to use `-r`)

```
[gburdell3@login-pace ~]$ ls
dir1  file2  new_dir  p-gburdell3-0  scratch
[gburdell3@login-pace ~]$ cp file2 new_dir
[gburdell3@login-pace ~]$ cp new_dir/renamed_file .
[gburdell3@login-pace ~]$ cp renamed_file new_dir/copied_and_renamed
[gburdell3@login-pace ~]$ cp -r new_dir copied_dir
[gburdell3@login-pace ~]$ cp -r new_dir copied_dir/copied_again
[gburdell3@login-pace ~]$ ls
copied_dir  dir1  file2  new_dir  p-gburdell3-0  renamed_file  scratch
[gburdell3@login-pace ~]$ ls new_dir
copied_again        file2              renamed_file
copied_and_renamed  moved_and_renamed
```

Bash Commands for Files (cont'd)

- `rm` is used to delete files (to copy directories, you'll need to use `-r`)

```
[gburdell3@login-pace ~]$ ls
copied_dir  dir1  file2  new_dir  p-gburdell3-0  renamed_file  scratch
[gburdell3@login-pace ~]$ rm file2
[gburdell3@login-pace ~]$ rm new_dir/renamed_file
[gburdell3@login-pace ~]$ rm copied_dir
rm: copied_dir/: is a directory
[gburdell3@login-pace ~]$ rm -r copied_dir
[gburdell3@login-pace ~]$ ls
dir1  new_dir  p-gburdell3-0  renamed_file  scratch
[gburdell3@login-pace ~]$ ls new_dir
copied_again  copied_and_renamed  file2  moved_and_renamed
```

- The `-f` option can be used to force remove (and ignore errors)

Bash Commands for Files (cont'd)

- `cat` can be used to view unformatted file content

```
[gburdell3@login-pace ~]$ cat somefile
THE STORY OF THE THREE LITTLE PIGS.

   Once upon a time there was an old sow
with three little pigs, and as she had not
enough to keep them, she sent them out to
seek their fortune. The first that went off
met a man with a bundle of straw, and said
to him, "Please, man, give me that straw
to build me a house;" which the man did,
and the little pig built a house with it.
Presently came along a wolf, and knocked
at the door, and said, -
```

Bash Commands for Files (cont'd)

- For longer files, cat can be problematic as it prints out the whole file
- head shows content from the beginning of the file
  - Default is first 10 lines, but can use −n<int> to specify another number

```
[gburdell3@login-pace ~]$ head -n3 somefile
THE STORY OF THE THREE LITTLE PIGS.


   Once upon a time there was an old sow
```

- Similarly, tail shows lines from the end of the file
  - Default is also 10, and −n<int> is still allowed

```
[gburdell3@login-pace ~]$ tail -n2 somefile
him for supper, and lived happy ever after-
wards.
```

Bash Commands for Files (cont’d)

- Another option is `less`, which displays one page at a time
  - You can scroll with the up/down arrows, use `f`/`b` to advance/backtrack a whole page, search forward/backward with `/`/`?`, and exit with `q`
  - The `:` means you can scroll further until it changes to `(END)`

```
[gburdell3@login-pace ~]$ less somefile
THE STORY OF THE THREE LITTLE PIGS.

   Once upon a time there was an old sow
with three little pigs, and as she had not
enough to keep them, she sent them out to
seek their fortune. The first that went off
met a man with a bundle of straw, and said
:
```

Bash Commands for Files (cont'd)

- `grep` is used to search for patterns (regex) within files

```
[gburdell3@login-pace ~]$ grep pigs somefile
with three little pigs, and as she had not
he did to the other little pigs, and said, -
```

- The `–i` option is used to ignore case

```
[gburdell3@login-pace ~]$ grep –i pigs somefile
THE STORY OF THE THREE LITTLE PIGS.
with three little pigs, and as she had not
he did to the other little pigs, and said, -
```

Bash Filename Expansions (Globbing Patterns)

- Sometimes, you may want to operate on multiple files simultaneously
  - If there are many files, it can be convenient to describe the filenames using a pattern
  - `?` – can be used to represent any single character
  - `*` – can be used to represent any number of character (including 0)
  - `[ ]` – can be used to specify a list of allowed values or a range (in arguments)
  - `[^]` – logical NOT to exclude characters
  - `{ }` – can be used to specify patterns using parameter expansion
  - `\` – escape character, used to protect a subsequent special character

```
[gburdell3@login-pace ~]$ ls * #lists all files in directory
[gburdell3@login-pace ~]$ rm file[124-6] #deletes file1, file2,
file4, file5, and file6
[gburdell3@login-pace ~]$ cat file1? #reads file10-file19
```

Learning More about Command -line Tools

- With so many commands, it can be helpful to get help on the terminal
  - Three main tools: `man`, `info`, and `-h`/`--help` options

```
[gburdell3@login-pace ~]$ man rm
```

- **Manual page:** description, available options/arguments, associated shell variables, related `man` pages, author/copyright information)

Forward (backward) one line: `e`, ↵, ↓ (`y`, ↑) *Forward (backward) one window*: `f`, `z` (`b`, `w`)
Search forward (backward): `/<pattern>` (`?<pattern>`) *End of manual*: `Shift+g` *Exit*: `q`

```
[gburdell3@login-pace ~]$ info gcc
```

- **GNU project documentation:** hypertext with links, very detailed (like a digital book with chapters; may open `man` page if no info manual exists)

Forward (backward) one line: ↓ (↑) *Forward (backward) one page*: `n`, `]` (`p`, `[`) *Search*: `Ctrl+s`
Cycle through links: `Tab` *Go to Main home*: `d` *Go to Topic Home*: `t` *Follow link*: `Enter` *Exit*: `q`

```
[gburdell3@login-pace ~]$ tar --help
```

- **CLI tool help:** documents some of the built-in commands and keywords of the shell (think of it as a cheat sheet – neat and succinct)

Environment Variables

- Environment variables are named objects that contain data used by one or more applications
  - Each environment variable has a name and a value
  - The name is case sensitive, and can include any character except = and `NUL`
  - The value can provide several key pieces of information
    - The location of all executable files in the file system
    - The default text editor or compiler that should be used
    - The system locale settings (language, country, character encoding, etc.)
- `printenv` lists the environment variables defined for the current session
- Some important environment variables:
  - `HOME`
  - `HOSTNAME`
  - `USER`
  - `SHELL`
  - `PATH`
  - `LD_LIBRARY_PATH/LIBRARY_PATH`
  - `PBS_O_WORKDIR`
  - `PBS_NODEFILE`
  - `PBS_GPUFILE`
  - `PBS_JOBID`
  - `PBS_ARRAYID`
  - `PBS_JOBNAME`

Working with Environment Variables

- The shell uses `$` to de-reference environment variables, e.g. `echo $PATH`
- Environment variable values can be changed a few ways:
  - Assignment (`VARIABLE="NEWVALUE"`) – *good for local variables*
  - Prepend (`VARIABLE="NEWVALUE:$VARIABLE"`) – *good if you need to point to a custom application instead of the default*
  - Append (`VARIABLE="$VARIABLE:NEWVALUE"`) – *for when you just need to point to a new, unique location*
- By default, all user defined variables are local – they won't be visible to child processes (processes started by the shell)
  - Use `export` to make the variable available to all processes run from that shell
  - Amendments to an already global variable will already be exported
  - If you want the variable to persist from session to session, define and export it in your `~/.bash_profile` file (discussed shortly)

…and How Software Modules Work

- An abundance of software is available, pre-compiled and configured for use on PACE clusters
  - Software environment, version all set – no need to edit `PATH`, `LD_LIBRARY_PATH`, etc.
- Modules are used to set everything up

`module avail`	lists all available modules
`module list`	displays all currently loaded modules
`module load <target>`	loads target module to the environment

TIP! `ml` can be used in place of `module list` AND `module load <target>`

`module rm <target>`	removes target module from the environment
`module purge`	removes all loaded
`module show`	displays the variables set by the module

- Default modules can be loaded per job in the PBS script or loaded manually on the command line

Automation with Shell Scripts

- A shell script is a text file containing shell commands that can be run in the shell non-interactively
  - Any command from the command line can be used in a bash script
- There are many benefits to implementing commands in a script
  - Task automation: write it once, use it many times
  - Consolidation: combine multiple steps into one command
  - Portable: the shell commands are the same in Bash, no matter the OS
  - Repeatability: you can look back over the file to see what you did

NOTE: By default, files will be created with 644 permissions (-rw-r--r--); to execute a shell script by typing `./myscript.sh`, change permissions to 755 to allow for execution (-rwxr-xr-x)

Developing Shell Scripts

- The first line of a script begins with `#!`, followed by the designated interpreter
  - Bash shell scripts start with `#!/usr/bin/bash`
- Any normal commands can be included as lines in the script
  - Commands can be separated by `&&`, `;`, or a new line
- Conditional execution:

```
if <condition>; then <command>; else <command>; fi
```

- For loops:

```
for <variable> in <array>; do <command>; done
```

OR

```
for ((expr1;expr2;expr3)); do <command>; done
```

- While loops:

```
while <condition>; do <command>; done
```

Developing Shell Scripts (cont’d)

- Arguments can be passed to a Bash script:
  - If the number of arguments is fixed, use the positional variables `$1`, `$2`, `$3`, …

    ```
    echo $1 $2 $3 #prints out the 1st, 2nd, and 3rd arguments passed via CLI
    ```
  - If the number of arguments is variable, use `$@` (which represents an array of all arguments passed to the script) and `$#` (which represents the number of arguments passed to the script)

    ```
    for ARGS in $@; do echo $ARGS; done
    ```
- Functions can be defined to minimize repetition (declaration is optional):

  ```
  function my_function() {
     #insert code here
  }
  ```
  or
  ```
  my_other_function() {
     #insert code here
  }
  ```
- Variables can be defined (`myvar=“value”`) and read (`echo $myvar`)

Passing Notes with I/O Redirects

- There are three standard streams in Linux:
  - **stdin (0)**: carries data from a user to a program
  - **stdout (1)**: writes data generated by a program, default to the terminal
  - **stderr (2)**: writes errors generated by a program, defaults to the terminal
- Streams can be redirected to/from existing files, or to a new file
  - `1>filename`: stdout data overwrites filename
  - `1>>filename`: stdout data appends filename
  - `2>filename`: stderr data overwrites filename
  - `2>>filename`: stderr data appends filename
  - `0<filename`: stdin data is read from filename
- Streams can also be redirected to chain commands
  - `command1 | command2`: stdout data from command1 is fed to command2

Shell Customization: `~/.bash_profile`

- The personal initialization file, executed for login shells
- Two main purposes:
  - Gets any aliases and functions from `~/.bashrc`, if it exists
  - Sets user specific environment and startup programs
- If you install Tier3 software and need to set your `PATH`, `LD_LIBRARY_PATH`, etc., this is the place to do it
- If your custom application needs to reference a shell script before running, this is the place to point to it
- To implement changes, log out and then log in, or type `source ~/.bash_profile`

Shell Customization: `~/.bashrc`

- The personal initialization file, executed for non-interactive login shells
- Two main purposes:
  - Get any globally defined aliases and functions
  - Define any user specific aliases and functions
- Example 1: Define a function that creates and moves into a new directory

```
mcd () {
      mkdir -p $1
      cd $1
}
```

- Example 2: Define an alias that will run `qstat` to detail current jobs for only the current user

```
alias qme='qstat -tn1u $USER'
```

Shell Customization: `~/.bash_logout`

- The personal initialization file, executed when login shells exit
- One main purposes:
  - Keep the Bash environment tidy for each time you login
- Clean up temporary files here:
  - You can simply tell it to `rm` any unwanted files
  - Or you can automate file compression for archiving with `tar`
  - Or you can copy them to remote storage for backup
- If you want to terminate all processes that you are currently running
- You can also add a line to give yourself a friendly message upon exit:

```
echo Have a wonderful day!
```

One Last Thing…

https://b.gatech.edu/2TceuOq

Georgia Tech

PACE Team

Partnership for an **A**dvanced **C**omputing **E**nvironment provides faculty participants sustainable leading-edge advanced research computing resources with technical support services, infrastructure, software, and more.

Please attend the PACE Clusters Orientation for more information: https://pace.gatech.edu/content/orientation

Ken Suda, Sr. Applications developer

Craig Moseley, Team lead Customer Exp

Michael Brandon, Management Support

Pam Buffington, Director

Allan Metts, Digital Learning Principal Architect

Matt Guidry, Sr. Systems Support Eng

Greg Beyer, Sr. Systems Support Eng

Paul Manno, RBW-IT Professional

Ruben Lara, Sr. Systems Support Mgr

Dee Womack, Systems support Eng

Peter Wan, Sr. Systems Support Eng

Dan (Ann) Zhou, Storage Architect

Aaron Jezghani, PhD, Scheduler Architect

Semir Sarajlic, Team Lead, ASE

Trever Nightingale, Cloud Architect

Ronald Rahaman, Research Software Eng

Eric Coulter, PhD, CI Integration Architect

Deepa Phanish, PhD Research Computing Facilitator

Fang (Cherry) Liu, PhD, Lead Research Software Eng

Marian Zvada, Research Computing Facilitator

Jeffrey Valdez, Research Computing Facilitator

Michael Weiner, PhD, Research Computing Facilitator

IO500

Open Science Grid

NSF

XSEDE Extreme Science and Engineering Discovery Environment

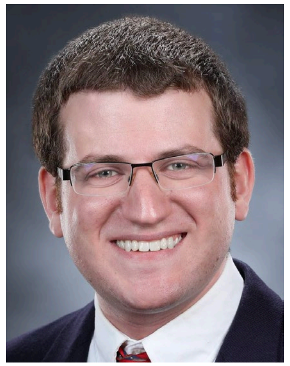
Georgia Tech

PACE-RCF

Meet the **R**esearch **C**omputing **F**acilitation team! We interact with the advanced computing research community at Georgia Tech, respond to a wide range of requests submitted by faculty & student researchers.

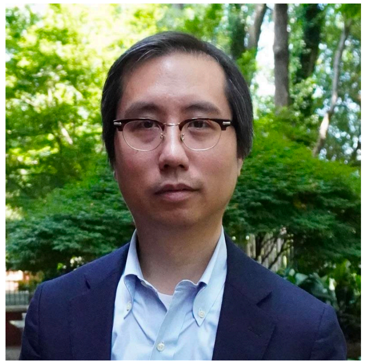
Fang (Cherry) Liu, PhD
RCF Team Lead

"HPC brings you to an amazing parallel universe!"



Michael D. Weiner, PhD
Research Computing Facilitator

"First used HPC in my research, still in awe of its power after over a decade!"



Jeffrey Valdez, MS, MBE
Research Computing Facilitator

"Always happy to help the research community at GT!"

Deepa Phanish, PhD
Research Computing Facilitator

"I love GT! Just jumped over the PACE fence!"

Marian Zvada, MSc, MBA
Research Computing Facilitator

"I like to refine business and enhance PACE customer experience. Always ready to play chess, too!"

Georgia Tech

External References

- General Linux
  - Wikibooks Guide to Linux: https://en.wikibooks.org/wiki/Linux_Guide
  - Siever, Ellen. *Linux in a Nutshell*. O’Reilly Media, Inc. (2009)
  - Welsh, Matt and Dalheimer, Matthias. *Running Linux*. O’Reilly Media, Inc. (2006)
- Shells
  - Linux Command: http://www.linuxcommand.org
  - Explain shell: https://explainshell.com
  - Kochan, Stephen G. and Wood, Patrick. *Unix Shell Programming*. O’Reilly Media, Inc. (2003)
  - Newham, Cameron. *Learning the Bash Shell*. O’Reilly Media, Inc. (2005)
  - Quigley, Ellie and Hawkins, Scott. *The Complete Linux Shell Programming Training Course*. Prentice Hall PTR (2000)
- Text Editors
  - Nano Cheat Sheet: https://www.nano-editor.org/dist/latest/cheatsheet.html
  - Vim Cheat Sheet: https://vim.rtorr.com
  - Emacs Reference Card: https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf

Georgia Tech

Command Cheatsheet

- `pwd` – show current working direcory
- `cd` – change directory
- `ls` – list contents of $pwd
- `chmod <permissions> <filename>` – change permissions (755 or u+x format) of filename
- `mkdir <dirname>` – make new empty directory
- `touch <filename>` - make new empty file
- `cat <filename>` - print out file contents
- `head –n X <filename>` – print first X lines
- `tail –n X <filename>` – print last X lines
- `less <filename>` - read file in ‘less’ reader
- `grep <pattern> <filename>` - search for <pattern> inside <filename>
- Help commands:
  - `man <command>`
  - `info <command>`
  - `<command> --help [or <command> –h]`
- File Editors:
  - `nano <filename>`
  - `vim <filename>`
  - `emacs <filename>`