

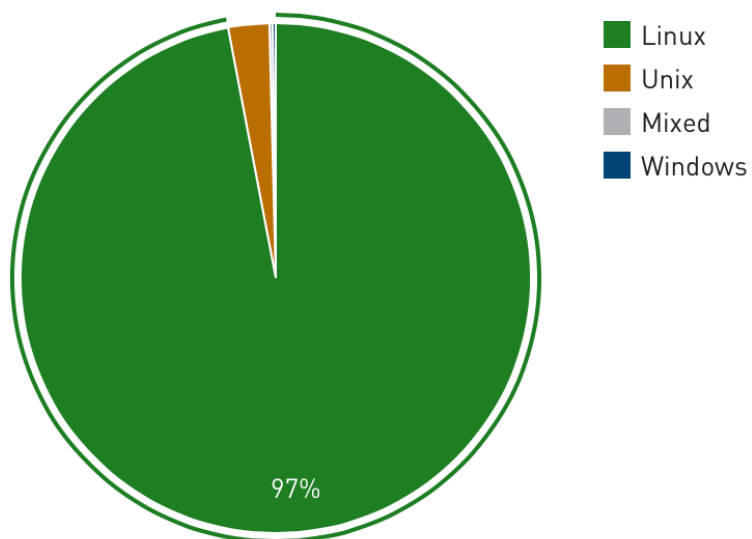
# Introduction to Linux

**Wei Feinstein**  
**HPC User Services**

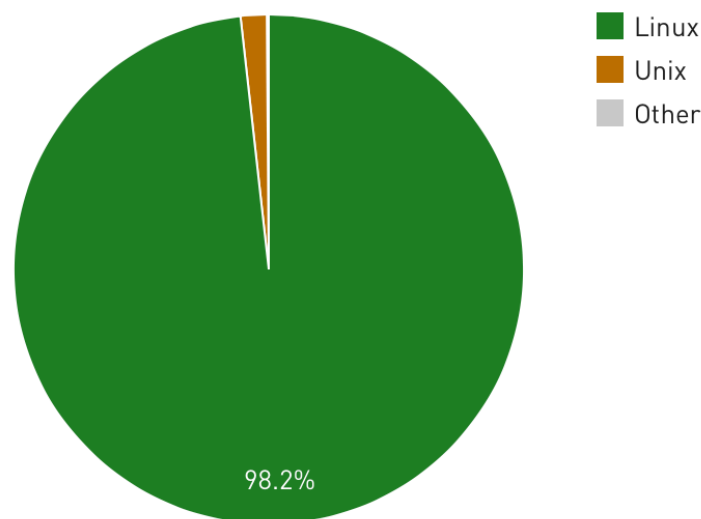
**LSU HPC & LON**  
**[sys-help@loni.org](mailto:sys-help@loni.org)**

# Why Linux for HPC - 2014

OS family system share

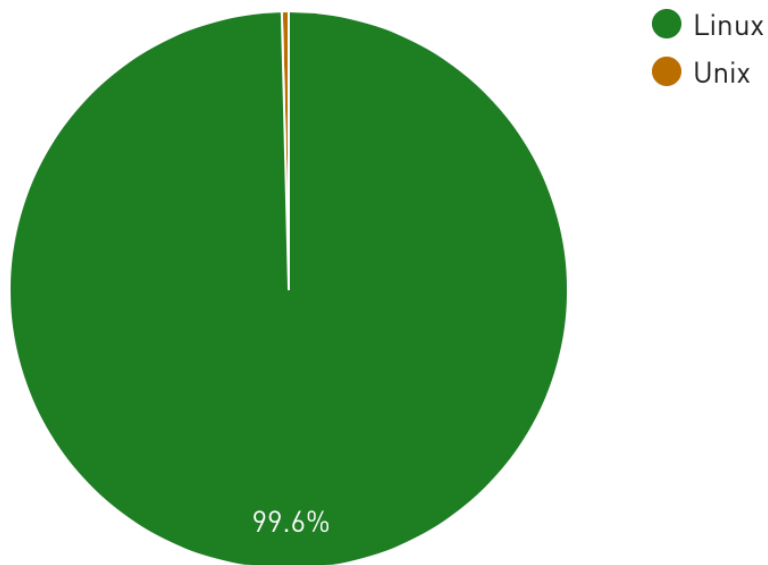


OS family performance share

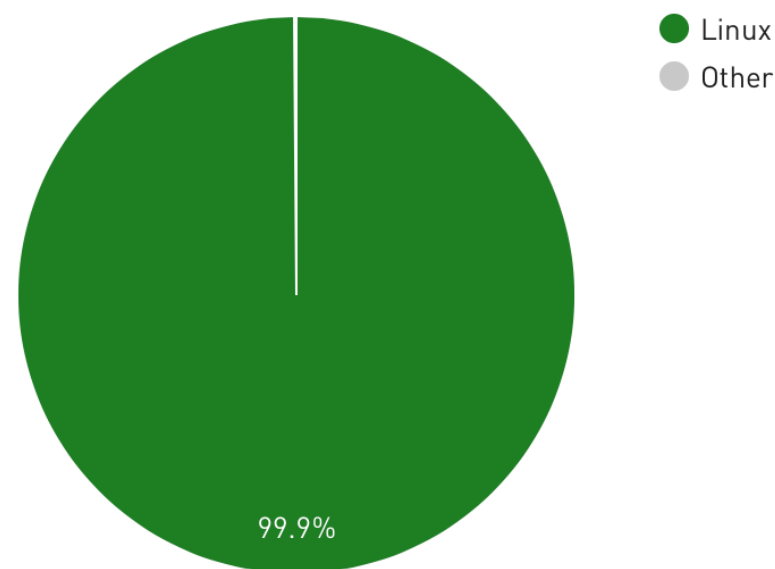


# Why Linux for HPC - 2017

OS family system share



OS family performance share



Linux is the most popular OS used in supercomputers

# Roadmap

- What is Linux
- Linux file system
- Basic commands
- File permissions
- Variables
- Use HPC clusters
- Processes and jobs
- File editing

# History of Linux (1)

- Unix was initially designed and implemented at AT&T Bell Labs 1969 by Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna
- First released in 1971 and was written in assembly language
- Re-written in C by Dennis Ritchie in 1973 for better portability (with exceptions to the kernel and I/O)

# History of Linux (2)

- Linus Torvalds, a student at University of Helsinki began working on his own operating system, which became the "Linux Kernel", 1991
- Linus released his kernel for free download and helped further development



# History of Linux (3)

- Linux as the kernel only, applications on top of the kernel were still missing
- The GNU Project by Richard Stallman started in 1983  
-Creating a “complete Unix-compatible software system” with entirely free software
- "GNU/Linux"(Linux): Linux kernel + free software from the GNU project
- GNU/Linux (Linux) released under the GNU Public License (GPL): Free to use, modify and re-distribute **iff** later distributions are also under GPL

# What is Linux

- Essential components: Linux kernel + GNU system utilities + installation scripts + management utilities etc.
- Many software vendors release their own packages, known as distributions
  - Debian, Ubuntu, Mint
  - Red Hat, Fedora, CentOS, Scientific Linux
  - Slackware, OpenSUSE, SLES, SLED
  - Gentoo
- Linux distributions offer a variety of desktop environment

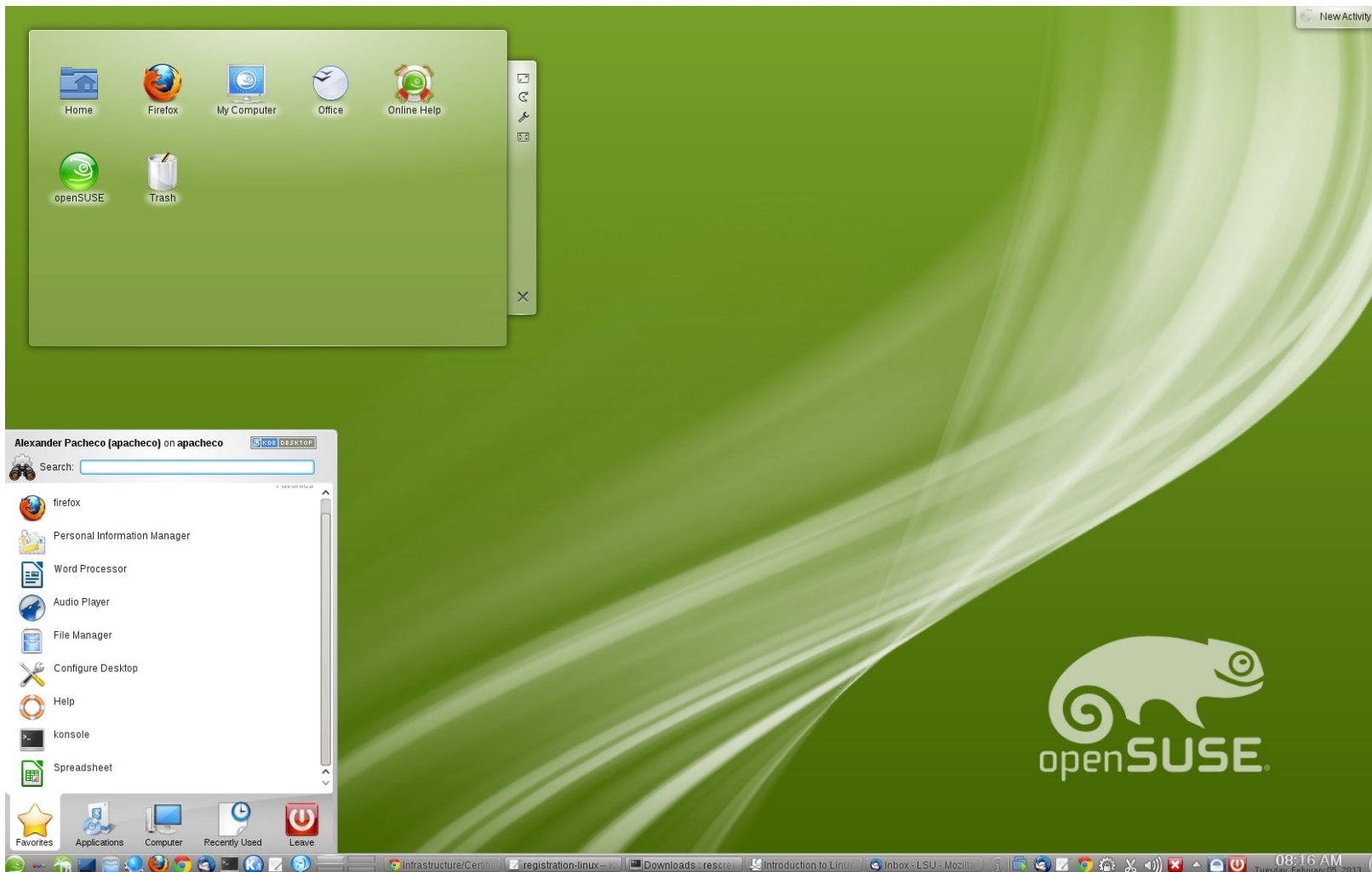
Redhat, KDE, GNOME, XFCE, LXDE, Cinnamon, MATE



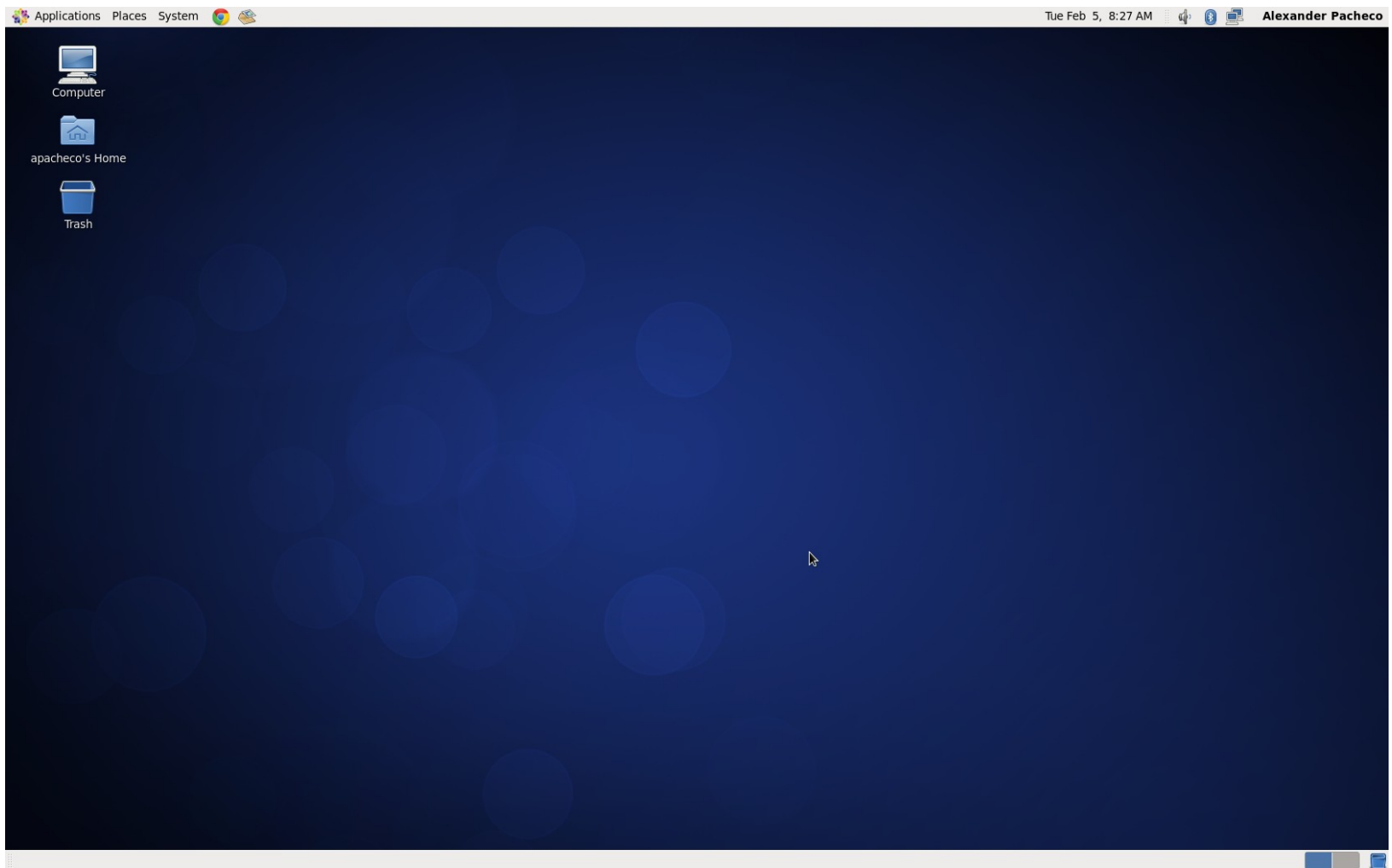
# Redhat Desktop



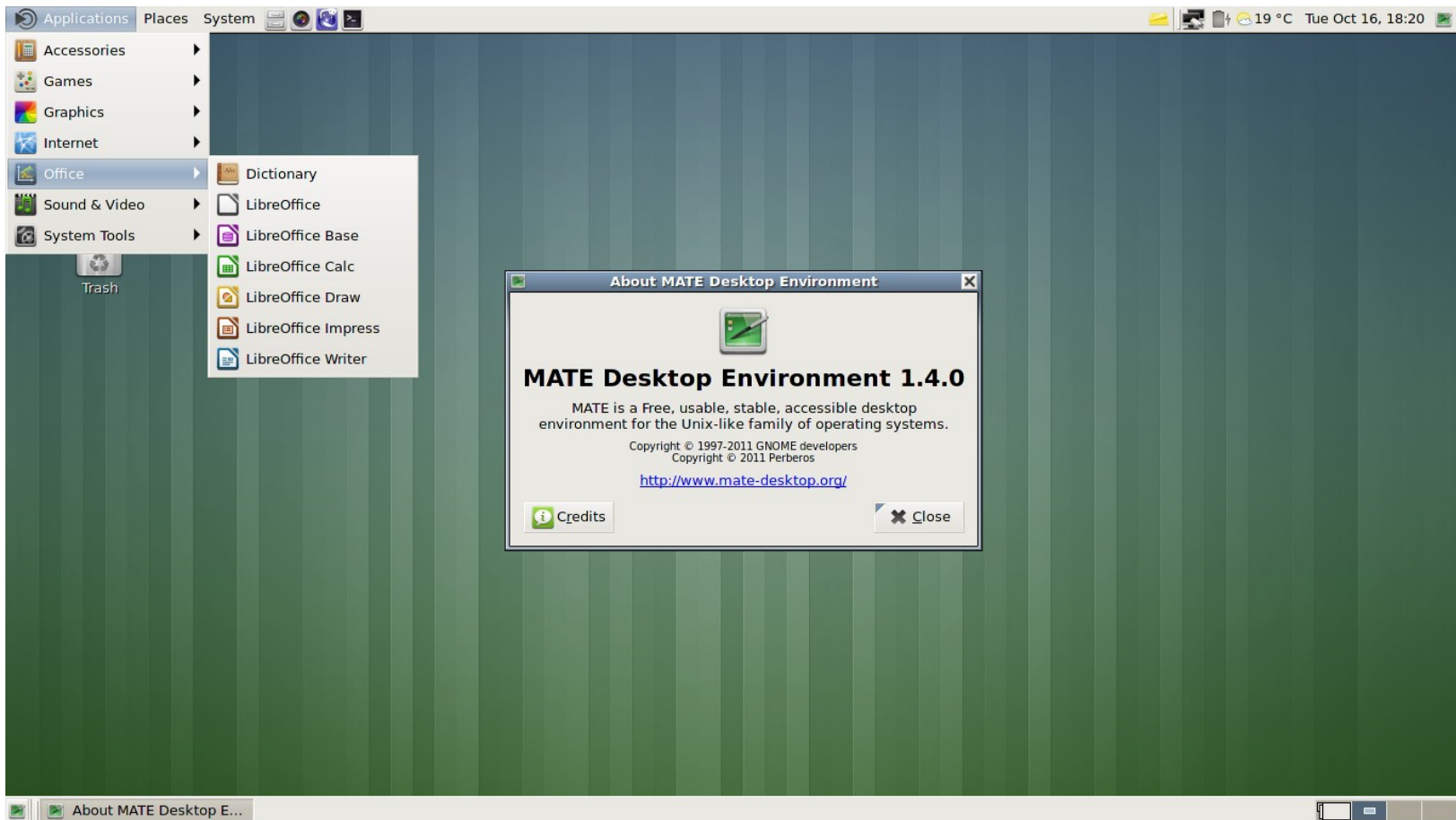
# openSUSE KDE Desktop



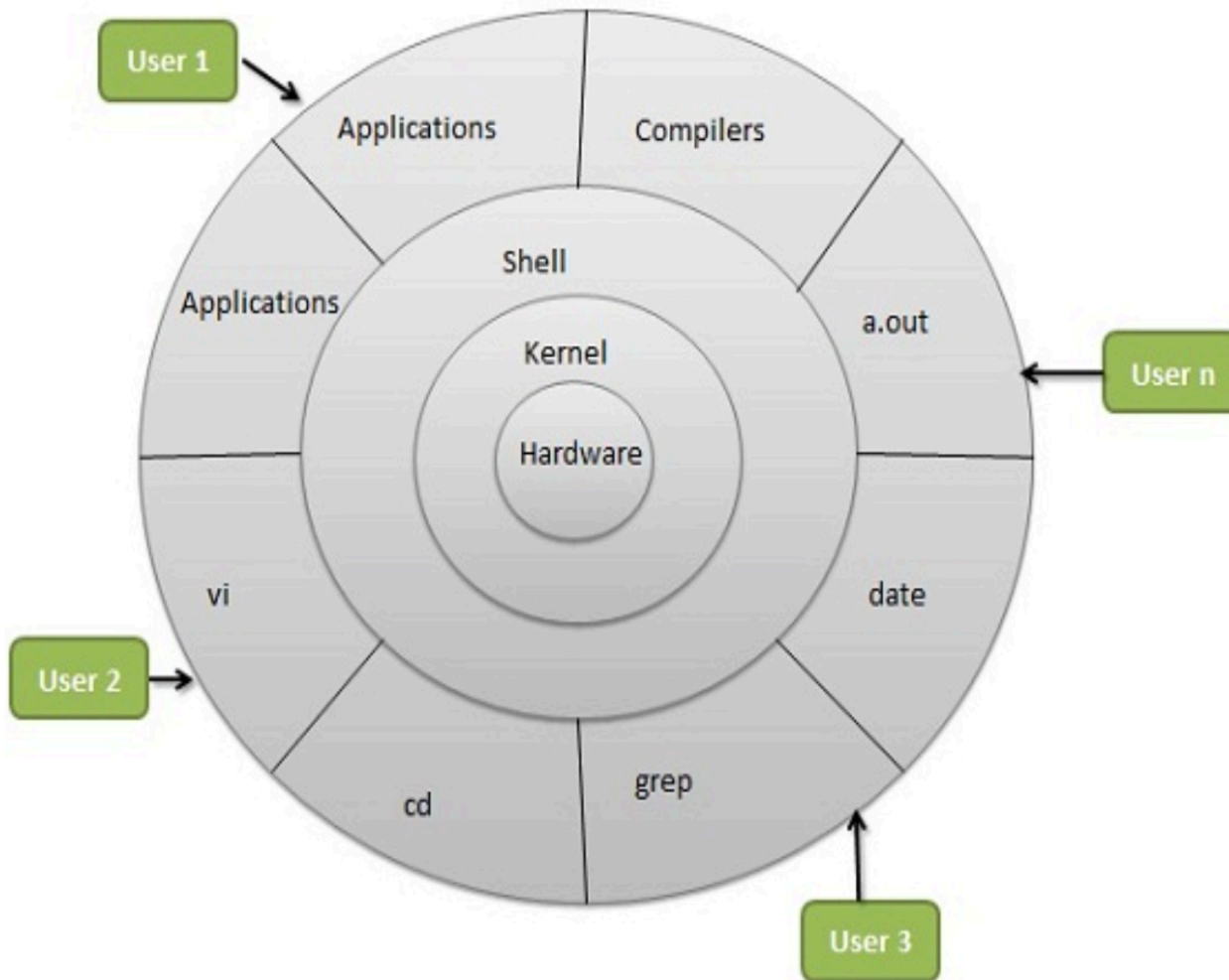
# CentOS GNOME Desktop



# Debian MATE Desktop



# Linux System Architecture



# Linux Kernel

## What is a Kernel

- The core component of an OS
- Manage the system's resources, memory, file systems...
- Provide the lowest level abstraction layer to upper layer components
- Inter-process communications and system calls are used to make services available

# Linux Shell

## What is a Shell

- An application running on top of the kernel and provides a powerful interface to the system
- Process user's commands, gather input from user and execute programs
- Types of shell with varied features
  - sh
  - csh
  - ksh
  - bash
  - tcsh

# Shell Comparison

Software	sh	csch	ksh	bash	tcsh
Programming language	y	y	y	y	y
Shell variables	y	y	y	y	y
Command alias	n	y	y	y	y
Command history	n	y	y	y	y
Filename autocompletion	n	y*	y*	y	y
Command line editing	n	n	y*	y	y
Job control	n	y	y	y	y

\*: not by default

<http://www.cis.rit.edu/class/simg211/unixintro/Shell.html>



# Shell Comparison

Software	sh	csch	ksh	bash	tcsh
Programming language	y	y	y	y	y
Shell variables	y	y	y	y	y
Command alias	n	y	y	y	y
Command history	n	y	y	y	y
Filename autocompletion	n	y*	y*	y	y
Command line editing	n	n	y*	y	y
Job control	n	y	y	y	y

\*: not by default

<http://www.cis.rit.edu/class/simg211/unixintro/Shell.html>

# Linux Applications

- GNU compilers, e.g., gcc, gfortran
- OpenOffice
- Editors, e.g., vim, emacs
- parallel
- wget
- cat, ls, cp
- ....

<https://directory.fsf.org/wiki/GNU>

# What can you do with a shell?

- Check the current shell
  - `echo $SHELL`
- List available shells on the system
  - `cat /etc/shells`
- Change to another shell
  - `exec sh`
- Date and time
  - `date`
- `wget`: get online files
  - `wget https://ftp.gnu.org/gnu/gcc/gcc-7.1.0/gcc-7.1.0.tar.gz`
- Compile and run applications
  - `gcc hello.c -o hello`
  - `./hello`

# Roadmap

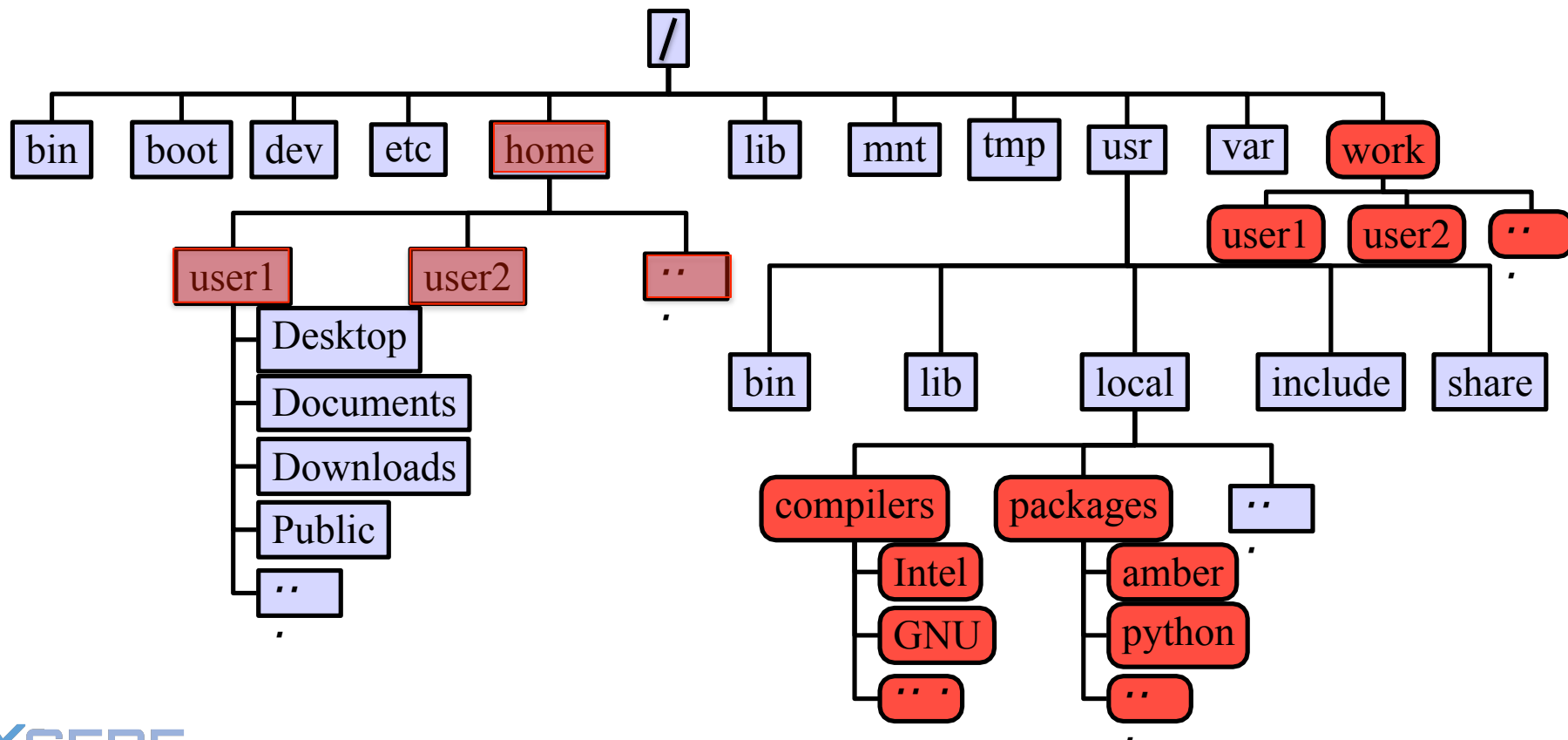
- What is Linux
- **Linux file system**
- Basic commands
- File permissions
- Variables
- Use HPC clusters
- Processes and jobs
- File editing

# Files

- All data in Linux/UNIX are organized in a file format
- A file is a collection of data created by users, system admins...
  - Documents composed of ascii text
  - Program written in high level programming languages
  - Executables that you can run
  - Directory containing information about its content

# File Directory Structure

- ❖ All files are arranged in directories.
- ❖ These directories are organized into the file system



# Important Directories

/bin	contains files that are essential for system operation, available for use by all users.
/lib,/lib64	contains libraries that are essential for system operation, available for use by all users.
/var	used to store files which change frequently (system level not user level)
/etc	contains various system configurations
/dev	contains various devices such as hard disk, CD-ROM drive etc
/sbin	same as bin but only accessible by <b>root</b>
/tmp	temporary file storage
/boot	contains bootable kernel and bootloader
/usr	contains user documentations, binaries, libraries etc
/home	contains home directories of all users. This is the directory where you are at when you login to a Linux/UNIX system.

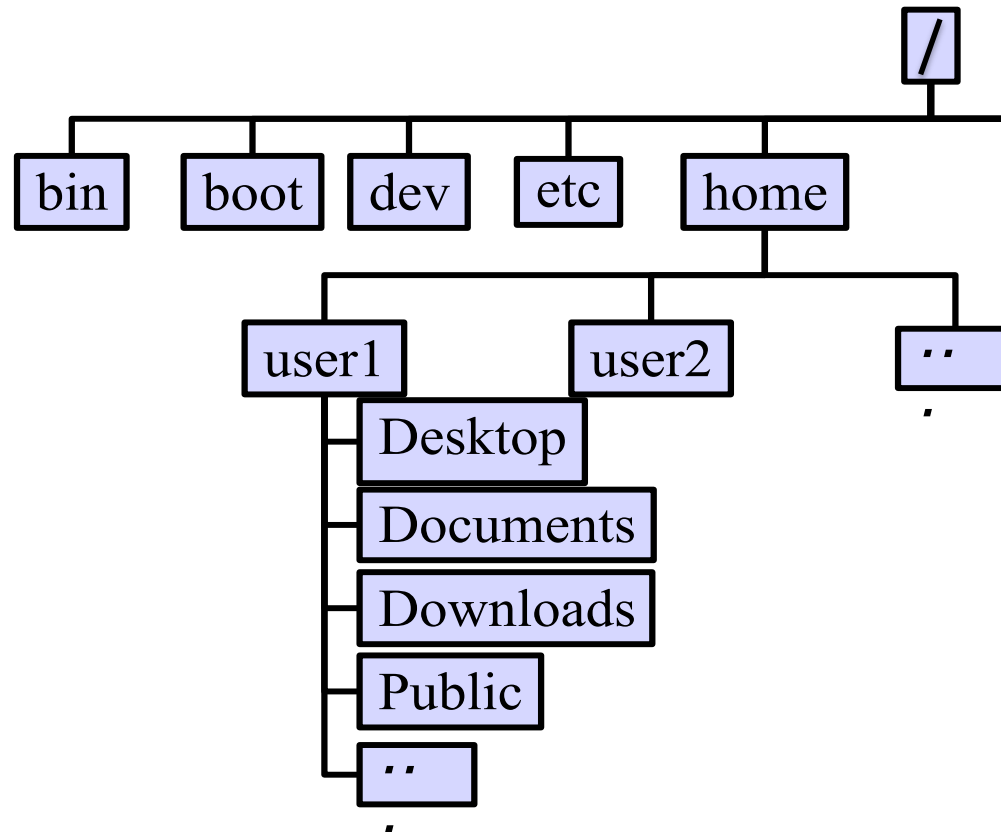
# File Path

Definition: position/address in the directory tree

- Absolute path
  - Uniquely defined and does NOT depend on the current path
  - /tmp is unique
- Relative path
  - Depend on the current location in the directory tree
  - . is the current working directory
  - .. is one directory up
  - : ../tmp is not unique



# File Path Examples



# Linux is Case Sensitive

- All names are case sensitive
  - Commands, variables, files etc.
- Example: MyFile.txt, myfile.txt, MYFILE.TXT are three different files in Linux

# Roadmap

- What is Linux
- Linux file system
- **Basic commands**
- File permissions
- Variables
- Use HPC clusters
- Processes and jobs
- File editing

# Basic Commands

- Command format: `command_name [options] arguments`  
`ls -l /home/$USER`
- Command prompt: a sequence of characters used in a command line interface to indicate the readiness to accept commands
  - Prompt user to take action
  - A prompt usually ends with one of the characters `$,%#,:,>` and often includes information such as user name and the current working directory
  - The format can be changed via `$PS1`

# Get More Information

- **Man**: show the manual for a command or program
  - The manual shows how to use the command and list the different options and arguments
  - **Usage:** `man <command name>`
  - **Example:** `man ls`
- **Apropos**: show all of the man pages that may be relevant to a certain command or topic
  - **Usage:** `apropos <string>`
  - **Example:** `apropos editor`
- **Info**: More information of a command
  - `info ls`

# Commands: ls

- **ls** command list the contents of a directory
  - Usage: **ls <options> <path>**
  - Example: **ls**
  - The default path will be current working directory if `path` is omitted.
- Options
  - **-l**: show long listing format
  - **-a**: (**--all**) show hidden files(name starts with an "." is hidden)
  - **-r**: reverse order when sorting
  - **-t**: show modification times
  - **-h**: (**--human-readable**) use file sized in SI units (bytes, kbytes, megabytes etc.)
  - **-d**: (**--directory**) list directory entries instead of contents, and do not dereference symbolic links

# Commands: cat, more/less, head/tail

- Display the content of a file to screen
  - **cat**: show content of a file
  - **more**: display contents one page at a time
  - **less**: display contents one page at a time, and allow forward/backward scrolling
- Usage: `cat/more/less <options> <filename>`
- **head**: output the first part of files
- **tail**: output the last part of files
- Usage: `head/tail <options> <filename>`
- Be careful when using those commands on binary files
  - **file**: reveal what type of file the target is

# Auto-completion

- Allows automatic completion of typing file, directory or command name via the TAB key
  - Convenient, also error-proof
  - If there is no unique name, all matching names will show
- The default feature in `bash` and `tcsh`
- Example: your home directory contains directories `Desktop`, `Documents` and `Downloads`
  - Enter command `ls D`, then press tab
  - Enter command `ls Do`, then press tab
  - Enter command `ls Dow`, then press tab



# Wildcards

Linux allows the use of wildcards for strings

- \*: any number of characters
  - Example: `ls *.gz` will list all the file ending with .gz
- ?: any *single* character
- [: specify a range
 

e.g.: `ls test[1-9]` list the file test1,test2 ...

# Commands: pwd and cd

- **pwd**: print the current working directory
  - Usage: `pwd`
  - Example: `pwd`
- **cd**: allow one to change the current working directory
  - Usage: `cd [destination]`
  - Example: `cd /tmp`
  - The default destination is the home directory if `[destination]` is omitted
  - `~` stands for home directory (bash)

# Commands: mkdir

- **mkdir**: create a directory
- Usage: **mkdir** <options> <path>
- Example: `mkdir ~/testdir`
- By default, the directory is created in the current directory
- Options
  - p: create the target directory as well as any directories that appear in the path but does not exist

# Commands: cp

- **cp**: copy a file or directory
- Usage: **cp** **<options>** **<sources>** **<destination>**
- Example: `cp $HOME/.bashrc ~/testdir`
- Options
  - **-r**: copy recursively, required when copying directories.
  - **-i**: prompt if file exists on destination and can be copied over.
  - **-p**: preserve file access times, ownership etc.
- By default **cp** will overwrite files with identical names (!!)
- If there are more than one source files, then the destination must be a directory

# Commands: rm

- **rm**: removes files and directories
- Usage: **rm** <options> <list of files/dirs>
- Examples: `rm testdir/.bashrc ~/testfile`
- Options
  - `-r`: remove recursively, required when deleting directories
  - `-i`: prompt if the file really needs to be deleted
  - `-f`: force remove (override the `-i` option)
- BE CAREFUL: DELETED FILES **\*CANNOT\*** BE RECOVERED!!!

# Commands: mv

- **mv** : moves or renames a file or directory
- **Usage:** `mv <options> <sources> <dest>`
- **Example:** `mv test test1`
- Use the `-i` option to prompt if a file or directory will be overwritten.
- If there are more than one source files, the destination must be a directory

# Commands: alias

- **alias**: create a shortcut to another command or name to execute a long string
- Usage
  - bash/sh/ksh: `alias <name>="<actual command>"`
  - csh/tcsh: `alias <name> "<actual command>"`
- Example
  - bash/sh/ksh: `alias lla="ls -altr"`
  - csh/tcsh: `alias lls "ls -altr"`
- **alias** can be used to prevent files from being deleted accidentally
  - Example: `alias rm "rm -i"`
- **alias**: list all aliases currently defined (without arguments)
- **unalias**: remove an alias

# Roadmap

- What is Linux
- Linux file system
- Basic commands
- **File permissions**
- Variables
- Use HPC clusters
- Processes and jobs
- File editing



# File Permission

```
weis-MacBook-Pro:Laplace2D wei$ ll -al
total 56
```

drwxr--r--	9	wei	staff	306	Jun	13	16:13	.
drwxr-xr-x	12	wei	staff	408	May	4	17:09	..
-rwxr--r--	1	wei	staff	1614	May	4	17:09	README
-rwxr--r--	1	wei	staff	2083	May	4	17:09	answer.c
-rwxr--r--	1	wei	staff	1945	May	4	17:09	answer.f90
-rwxr--r--	1	wei	staff	2004	May	4	17:09	exercise.c
-rwxr--r--	1	wei	staff	1833	May	4	17:09	exercise.f90
lrwxr-xr-x	1	wei	staff	10	Jun	13	16:13	linkMe -> exercise.c
-rwxr--r--	1	wei	staff	1572	May	4	17:09	timer.h

# Linux File Permission

- Designed as the multi user environment, the access restriction of files to other users on the system is embedded.
- Three types of file permission
  - Read (r)
  - Write (w)
  - Execute (x)
- Three types of user
  - User (u) (owner)
  - Group (g) (group members)
  - World (o) (everyone else on the system)

# Linux File Permission

Each file in Linux has the following attributes:

- **Owner permissions:** determine what actions the owner of the file can perform on a file
- **Group permissions:** determine what actions a user, who is a member of the group that a file belongs to, can perform on a file
- **Other (world) permissions:** indicate what action all other users can perform on a file

# File Permission

```
weis-MacBook-Pro:Laplace2D wei$ ll -al
total 56
drwxr--r--  9 wei  staff   306 Jun 13 16:13 .
drwxr-xr-x 12 wei  staff   408 May  4 17:09 ..
-rwxr--r--  1 wei  staff  1614 May  4 17:09 README
-rwxr--r--  1 wei  staff  2083 May  4 17:09 answer.c
-rwxr--r--  1 wei  staff  1945 May  4 17:09 answer.f90
-rwxr--r--  1 wei  staff  2004 May  4 17:09 exercise.c
-rwxr--r--  1 wei  staff  1833 May  4 17:09 exercise.f90
lrwxr-xr-x  1 wei  staff    10 Jun 13 16:13 linkMe -> exercise.c
-rwxr--r--  1 wei  staff  1572 May  4 17:09 timer.h
```

The first column indicates the type of a file/dir/link

- d: for directory
- l: for symbolic link
- - for normal file

# File Permission

```
weis-MacBook-Pro:Laplace2D wei$ ll -a
total 56
drwxr--r--  9 wei  staff   306 Jun  1
drwxr-xr-x 12 wei  staff   408 May
-rwxr--r--  1 wei  staff  1614 May
-rwxr--r--  1 wei  staff  2083 May
-rwxr--r--  1 wei  staff  1945 May
-rwxr--r--  1 wei  staff  2004 May
-rwxr--r--  1 wei  staff  1833 May
lrwxr-xr-x  1 wei  staff    10 Jun  1
-rwxr--r--  1 wei  staff  1572 May
```

Owner  
(u)

Group  
(g)

Others  
(o)

# Changing File Permission

- **chmod** is a \*NIX command to change permissions on a file
- Usage: `chmod <option> <permissions> <file or directory name>`
- `-R`: change permission recursively in a directory
- **chmod in Symbolic Mode:**

Chmod operator	Description
+	Adds the designated permission(s) to a file or directory.
-	Removes the designated permission(s) from a file or directory.
=	Sets the designated permission(s).

e.g. `chmod u+rwx filename`

`chmod o-w filename`

# Changing File Permission

## ■ Chmod in Absolute Mode:

Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--X
2	Write permission	-W-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-WX
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-X
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwX

e.g. chmod 755 test.txt  
           ↑  ↑  ↑  
       (u) (g) (o)

# Permission Effect on File vs Directory

Permission	File	Directory
r	read the file content	Ls files under the directory
w	write to the file	create new files and directories, delete existing files and directories, rename and move the existing files and directories
x	execute the file (if executable)	cd into the directory



# User Groups at HPC/LONI

- Users are organized into groups
  - **groups** command to find your group membership
- Group membership makes sharing files with members of a group easy
- Each user is in at least one group and can be in multiple groups
  - Groups in LONI systems:
    - `lsuusers, latechusers, unousers,`
    - `ullusers, sususers, tulaneusers,`
    - `loniusers, xavierusers`
  - Groups in LSU HPC system
    - `Users, Admins, xsede...`
  - You are only in one of the above groups due to software licensing

# Roadmap

- What is Linux
- Linux file system
- Basic commands
- File permissions
- **Variables**
- Use HPC clusters
- Processes and jobs
- File editing

# Variables

- Linux allows the use of variables
  - Similar to programming languages
  - Number, character or string
- Rules for variable names
  - Must start with a letter or underscore
  - Number can be used anywhere else
  - Do not use special characters such as @, #, %, \$
  - Case sensitive
  - Allowed: VARIABLE, VAR1234able, var\_name, \_VAR
  - Not allowed: 1var, %name, \$myvar, var@NAME
- Two types of variables:
  - Global variables (ENVIRONMENT variables)
  - Local variables (user defined variables)

# Global Variables

- Environment variables provide a simple way to share configuration settings between multiple applications and processes in Linux
  - Using all uppercase letters
  - Example: `PATH`, `LD_LIBRARY_PATH`, `DISPLAY` etc.
- To reference a variable, prepend \$ to the name of the variable
- Example: `$PATH`, `$LD_LIBRARY_PATH`, `$DISPLAY` etc.
- `printenv`/`env` list the current environmental variables in your system.

# List of Some Environment Variables

PATH	A list of directory paths which will be searched when a command is issued
LD_LIBRARY_PATH	colon-separated set of directories where libraries should be searched for first
HOME	indicate where a user's home directory is located in the file system.
PWD	contains path to current working directory.
OLDPWD	contains path to previous working directory.
TERM	specifies the type of computer terminal or terminal emulator being used
SHELL	contains name of the running, interactive shell.
PS1	default command prompt
PS2	Secondary command prompt
HOSTNAME	The systems host name
USER	Current logged in user's name
DISPLAY	Network name of the X11 display to connect to, if available.

# Editing Variables

- Assign values to variables

Type	sh/ksh/bash	csch/tcsch
Shell (local)	name=value	set name=value
Environment (global)	export name=value	setenv name value

- Shell variables is only valid within the current shell, while environment variables are valid for all subsequently opened shells.
- Example: useful when running a script, where exported variables (global) at the terminal can be inherited within the script.
  - \$ export v1=one
    - \$ bash
      - \$ echo \$v1 → one

# Editing Variables at the curent login

Example: to add a directory to the PATH variable

```
sh/ksh/bash: export PATH=/path/to/executable:${PATH}  
csh/tcsh: setenv PATH /path/to/executable:${PATH}
```

- Path order matters, first in line takes a higher priority

# Persistent variables for each login

- No change at each login
- Make setting changes available in both login and non-login shells
- Define these variables in the `~/.bashrc` file.
  - `vim ~/.bashrc`
  - `source ~/.bashrc`



# Input & Output Commands

The basis I/O statements:

- **echo**: display info to screen
  - The `echo arguments` command will print arguments to screen or standard output, where `arguments` can be a single or multiple variables, string or numbers
- **read**: reading input from screen/keyboard/prompt
  - The `read` statement takes all characters typed until the Enter key is pressed
  - Usage: `read <variable name>`
  - Example: `read name`

# Input & Output Commands

- Examples
  - `echo "hello !"`
  - `hello`
- By default, `echo` eliminates redundant whitespaces (multiple spaces and tabs) and replaces it with a single whitespace between arguments.
- To include redundant whitespace, enclose the arguments within double quotes

# Other Useful Commands

<b>passwd</b>	Change password ( <b>does not work on LSU HPC and LONI systems</b> )
<b>chsh</b>	Change default shell ( <b>does not work on LSU HPC and LONI systems</b> )
<b>df</b>	Report disk space usage by filesystem
<b>du</b>	Estimate file space usage - space used under a particular directory or files on a file system.
<b>sudo</b>	Run command as root ( <b>only if you have access</b> )
<b>mount</b>	Mount file system ( <b>root only</b> )
<b>umount</b>	Unmount file system ( <b>root only</b> )
<b>shutdown</b>	Reboot or turn off machine ( <b>root only</b> )
<b>top</b>	Produces an ordered list of running processes
<b>free</b>	Display amount of free and used memory in the system
<b>find</b>	Find a file
<b>alias</b>	enables replacement of a word by another string

# Other Useful Commands

<b>vi</b>	Edit a file using VI/VIM
<b>emacs</b>	Edit a file using Emacs
<b>file</b>	Determine file type
<b>wc</b>	Count words, lines and characters in a file <code>wc -l file</code>
<b>grep</b>	Find patterns in a file <code>grep alias file</code>
<b>awk</b>	File processing and report generating <code>awk '{print \$1}' file</code>
<b>sed</b>	Stream Editor <code>sed 's/home/HOME/g' file</code>
<b>set</b>	manipulate environment variables <code>set -o emacs</code>
<b>touch</b>	change file timestamps or create file if not present
<b>date</b>	display or set date and time
<b>which</b>	Location of a command

# Other Useful Commands

<b>ln</b>	Link a file to another file <code>ln -s file1 file2</code>
<b>wait</b>	Wait for each specified process and return its termination status.
<b>which</b>	Shows the full path of (shell) commands
<b>who</b>	Show who is logged on
<b>whoami</b>	Print effective userid
<b>finger</b>	User information lookup program
<b>whatis</b>	Display manual page descriptions
<b>history</b>	Display the command history list with line numbers. An argument of n lists only the last n lines.

❏ `man command`: learn more about these commands

# Roadmap

- What is Linux
- Linux file system
- Basic commands
- File permissions
- Variables
- **Use HPC clusters**
- Processes and jobs
- File editing

# Login Remote Systems

- Most Linux systems allocate secure shell connections from other systems
- Log in using the **ssh** command to the LSU HPC and LONI clusters
- Usage: `ssh <username>@<remote host name>`
  - Example: `ssh user@mike.hpc.lsu.edu`
- **-X** option: forward the display of an application
- The default port is 22 for `ssh`
  - `ssh -p <port number> <username>@<hostname>`

# File Transfer between Two Systems

- **scp** : copy files between two hosts over the ssh protocol
- Usage:
  - `scp <options> <user>@<host>:/path/to/source`  
`<user>@<host>:/path/to/destination`
- If the user name is the same on both systems, omit `<user>`
- If transferring files from or to localhost, `<user>@<host>:` option can be omitted
- Options are `-r` and `-p`, same meaning with `cp`
- Examples
  - `scp user@mike.hpc.lsu.edu:/home/user/file1 .`
  - `scp -r file user@mike.loni.org:/home/user`



# File Transfer between Two Systems

- **rsync** is another utility for file transferring
- **Usage:** `rsync <options> <source> <destination>`
- Delta-transfer algorithm
  - Only transfer the bits that are different between source and destination
- Widely used for backups and mirroring as an improved copy command for everyday use
- Command options
  - -a: archive mode
  - -r: recursive mode
  - -v: increase verbosity
  - -z: compress files during transfer
  - -u: skip files that are newer on the receiver
  - -t: preserve modification times

# Compressing and Archiving a single file

Reduce storage usage or bandwidth while transferring files.

- **Compress:** `gzip`, `zip`, `bzip2`
- **Decompress:** `gunzip`, `unzip`, `bunzip2`
- Options
  - Recursively, use the `-r` option
  - Overwrite files while compressing/uncompressing, use the `-f` option
- By convention
  - gzipped files: `.gz`, `.z` or `.Z`
  - zipped files: `.Zip` or `.zip`
  - bzipped files: `.bz2` or `.bz`

# Compressing and Archiving Files

- **tar**: create and manipulate streaming archived files.
- Usage: **tar** **<options>** **<file>** **<path>**
  - **<file>** tar archived file, usually with extension .tar
  - **<path>** files/directories being archived
- Common options
  - -c: create/compress an archive file
  - -x: extract/decompress an archive file
  - -t: list contents of archive (for testing)
  - -z: filter the archive through gzip
  - -j: filter the archive through bzip2
  - -f: archive
  - -v: verbosely list files processed

# Compressing and Archiving Files

**tar**: create and manipulate streaming archived files.

Examples:

- File compressing

- `tar czvf file.tgz ${HOME}/*`
- `tar cjvf file.tgz2 ${HOME}/*`

- File decompressing

- `tar xzvf file.tgz -C [dest]`
- `tar xjvf file.tgz2 -C [dest]`

- File listing for testing

- `tar tvf file.tgz`

# Pipes

- Pipe commands: connect two or more commands together using “|”
- **grep**: searches certain patterns from a file(s)  
`cat file | grep [option] pattern`

Option	Description
-v	Print all lines not match pattern
-n	Print the matched line and line number
-l	Print only the names of files with matching pattern
-i	Match either upper- or lowercase.
-c	Print the count of matching lines

# Pipes: sort, wc, more, less

- sort: arranges lines of text alphabetically or numerically
- `ls | sort -k2`

Option	Description
-n	Sort numerically
-r	Reverse the order of sort
-k	Sort by a certain column
-t	Field separator

- `ls | wc`
- `cat file | more`
- `Cat file | less`

# I/O Redirection

- Three file descriptors for I/O streams (everything is a file in Linux)



- I/O redirection allows users to connect applications
  - <: a file to STDIN as input
  - >: save STDOUT to a file
  - >>: append STDOUT to a file

# I/O Redirection Examples

- Write STDOUT to file:
  - `ls -l > ls.out`
- Write STDERR to file:
  - `ls -l 2 > ls.err`
- Write STDERR and STDOUT to file:
  - `ls -l >output 2>&1`
- Discard STDOUT and STDERR:  
`command > /dev/null 2>&1`



# Roadmap

- What is Linux
- Linux file system
- Basic commands
- File permissions
- Variables
- Use HPC clusters
- **Processes and jobs**
- File editing

# Processes

- Process: an instance of a running program
- Linux create and start a new process (PID) for each command
  - `ps` or `top`
- A process can be run in :
  - **Foreground:** the command prompt is not returned until the current process has finished executing.
  - **Background:** the command prompt back to do some other useful work e.g. `ls -l &`

# Processes and Jobs

- Two ways to send a job into the background:
  1. `command &`
  2. suspend a running job using `Ctrl-z` and `bg`.
- When a process is running in background or suspended, it will be entered on to a list along with a job number (not PID) `jobs -l`
- Restart a suspended job to foreground  
`fg %<job number>`

# Managing Processes and Jobs

- To kill or terminate a process:
  - Job running in foreground: `Ctrl-c`
  - Job whose job ID you know: `kill %<job number>`
  - Job whose PID you know: `kill <PID>`

# Roadmap

- What is Linux
- Linux file system
- Basic commands
- File permissions
- Variables
- Use HPC clusters
- Processes and jobs
- **File editing**

# File Editing

- Most commonly used editors on Linux/Unix systems
  - vi or vim (vi improved)
  - Emacs
- vi/vim is installed by default on Linux/Unix systems and has only a command line interface (CLI).
- Emacs has both a command line interface (CLI) and a graphical user interface (GUI).
  - use `emacs -nw` to open file in console
- Other editors: nano, pico, kate, gedit, gvim, kwrite, nedit

# File Editing (vi)

vi works in two modes:

- Command mode
  - This is the mode when entering vi
  - Commands can be issued at the bottom of the screen, e.g. copy, paste, search, replace etc.
  - Press “i” to enter editing mode
- Editing mode
  - Text can be entered in this mode
  - Press “esc” key to go back to the command mode

# Most used commands (vi)

Description	Command
Insert at cursor	i
Insert at the beginning of line	I
Delete a line	dd
Copy a line	yy
Paste	p
Search forward	/pattern
Search backward	?pattern
Search again	n
Go to line #n	n
Replace text	%s/new/old/g
Save and exit	wq



# Editor cheatsheet (1)

## Cursor Movement

- move left
- move down
- move up
- move right
- jump to beginning of line
- jump to end of line
- goto line *n*
- goto top of file
- goto end of file
- move one page up
- move one page down

## vi

- h
- j
- k
- l
- 0
- \$
- nG
- 1G
- G
- C-u
- C-d

## emacs

- C-b
- C-n
- C-p
- C-f
- C-a
- C-e
- M-x goto-line ← *n*
- M-<
- M->
- M-v
- C-v

# Editor cheatsheet (2)

## File Manipulation

- save file
- save file and exit
- quit
- quit without saving
- delete a line
- delete *n* lines
- paste deleted line after cursor
- paste before cursor
- undo edit
- delete from cursor to end of line
- search forward for *patt*
- search backward for *patt*
- search again forward (backward)

## vi

- :w
- :wq, ZZ
- :q
- :q!
- dd
- ndd
- p
- P
- u
- D
- \patt
- ?patt
- n

## emacs

- C-x C-s
- 
- C-x C-c
- 
- C-a C-k
- C-a M-n C-k
- C-y
- 
- C-\_
- C-k
- C-s *patt*
- C-r *patt*
- C-s (r)

# Editor cheatsheet (3)

## File Manipulation (contd)

- replace a character
- join next line to current
- change a line
- change a word
- change to end of line
- delete a character
- delete a word
- edit/open file *file*
- insert file *file*
- split window horizontally
- split window vertically
- switch windows

## vi

- r
- J
- cc
- cw
- c\$
- x
- dw
- :e *file*
- :r *file*
- :split or C-ws
- :vsplit or C-wv
- C-ww

## emacs

- 
- 
- 
- 
- 
- C-d
- M-d
- C-x C-f *file*
- C-x i *file*
- C-x 2
- C-x 3
- C-x o

# Shell Scripts

- Script: a program written for a software environment to automate execution of tasks
  - A series of shell commands put together in a file
  - When the script is executed, those commands will be executed one line at a time automatically
  
- The majority of script programs are “quick and dirty”, where the main goal is to get the program written quickly
  - May not be as efficient as programs written in C and Fortran

# Script Example (~/.bashrc)

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# User specific aliases and functions
export PATH=$HOME/packages/eFindsite/bin:$PATH
export LD_LIBRARY_PATH=$HOME/packages/eFindsite/lib:$LD_LIBRARY_PATH
alias qsubI="qsub -I -X -l nodes=1:ppn=20 -l walltime=01:00:00 -A
my_allocation"
alias lh="ls -altrh"
```

# Getting Help

- User Guides
  - LSU HPC: <http://www.hpc.lsu.edu/docs/guides.php#hpc>
  - LONI: <http://www.hpc.lsu.edu/docs/guides.php#loni>
- Documentation: <http://www.hpc.lsu.edu/docs>
- Archived HPC training:  
<http://www.hpc.lsu.edu/training/archive/tutorials.php>
- Contact us
  - Email ticket system: [sys-help@loni.org](mailto:sys-help@loni.org)
  - Telephone Help Desk: 225-578-0900

# Upcoming Trainings

September 5,2018: HPC User Environment 1

September 12,2018: HPC User Environment 2

September 19,2018: Basic Shell Scripting

September 26,2018: Version Control using Git

October 10,2018: Introduction to R

October 17,2018: Data Analysis in R

October 24,2018: Introduction to Deep Learning