

# Introduction to Linux

Le Yan/Alex Pacheco  
HPC User Services

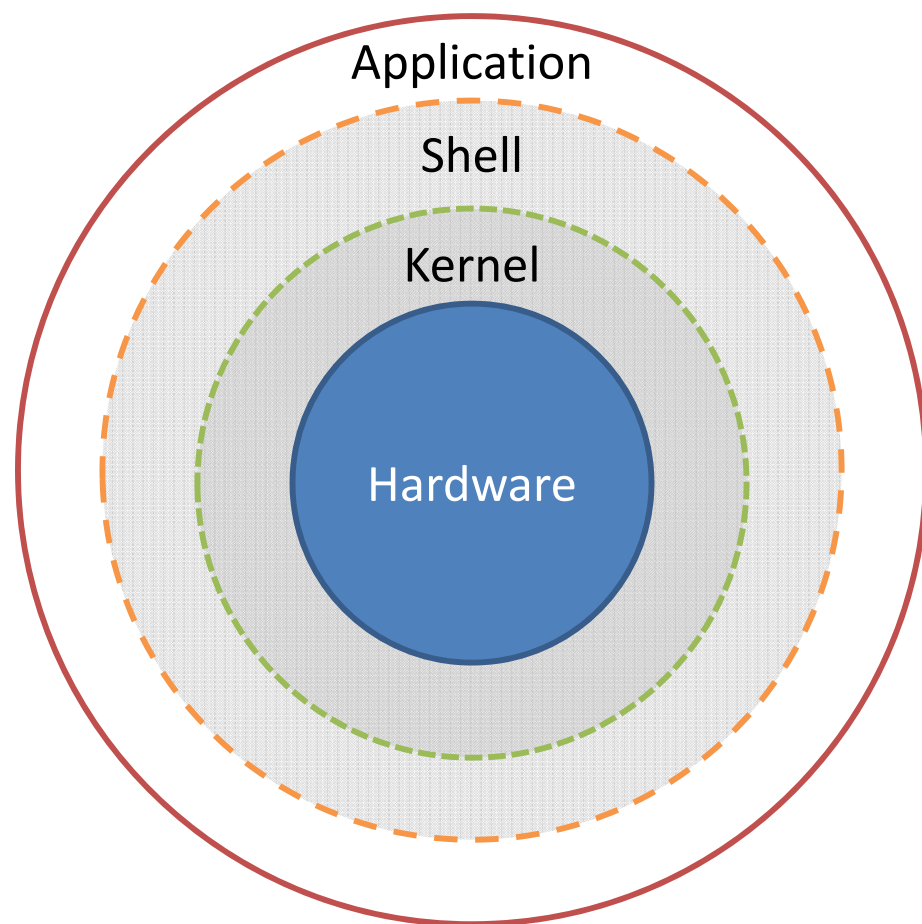
# Outline

- What is Linux
- Variables
- Basic commands
- File permissions
- Processes and jobs
- File editing

# Outline

- What is Linux
- Variables
- Basic commands
- File permissions
- Processes and jobs
- File editing

# What Do Operating Systems Do?



- Operating systems work as a bridge between hardware and applications
  - Kernel: hardware drivers etc.
  - Shell: user interface to kernel
  - Some applications (system utilities)

# History of Linux (1)

- Unix was conceived and implemented in 1969 at AT&T Bell labs by Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna.
- First released in 1971 and was written in assembler.
- In 1973, Unix was re-written in the programming language C by Dennis Ritchie (with exceptions to the kernel and I/O).
- The availability of an operating system written in a high-level language allowed easier portability to different computer platforms.
- The GNU Project, started in 1983 by Richard Stallman, had the goal of creating a "complete Unix-compatible software system" composed entirely of free software.
- 386BSD released in 1992 and written by Berkeley alumni Lynne Jolitz and William Jolitz. FreeBSD, NetBSD, OpenBSD and NextStep (Mac OSX) descended from this.
- Andrew S. Tanenbaum wrote and released MINIX, an inexpensive minimal Unix-like operating system, designed for education in computer science.

# History of Linux (2)

- Frustrated with licensing issues with MINIX, Linus Torvalds, a student at University of Helsinki began working on his own operating system which eventually became the "Linux Kernel"
- Linus released his kernel for anyone to download and help further development.

## Linus's message to comp.os.minix on Aug 26, 1991

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (email address)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-).

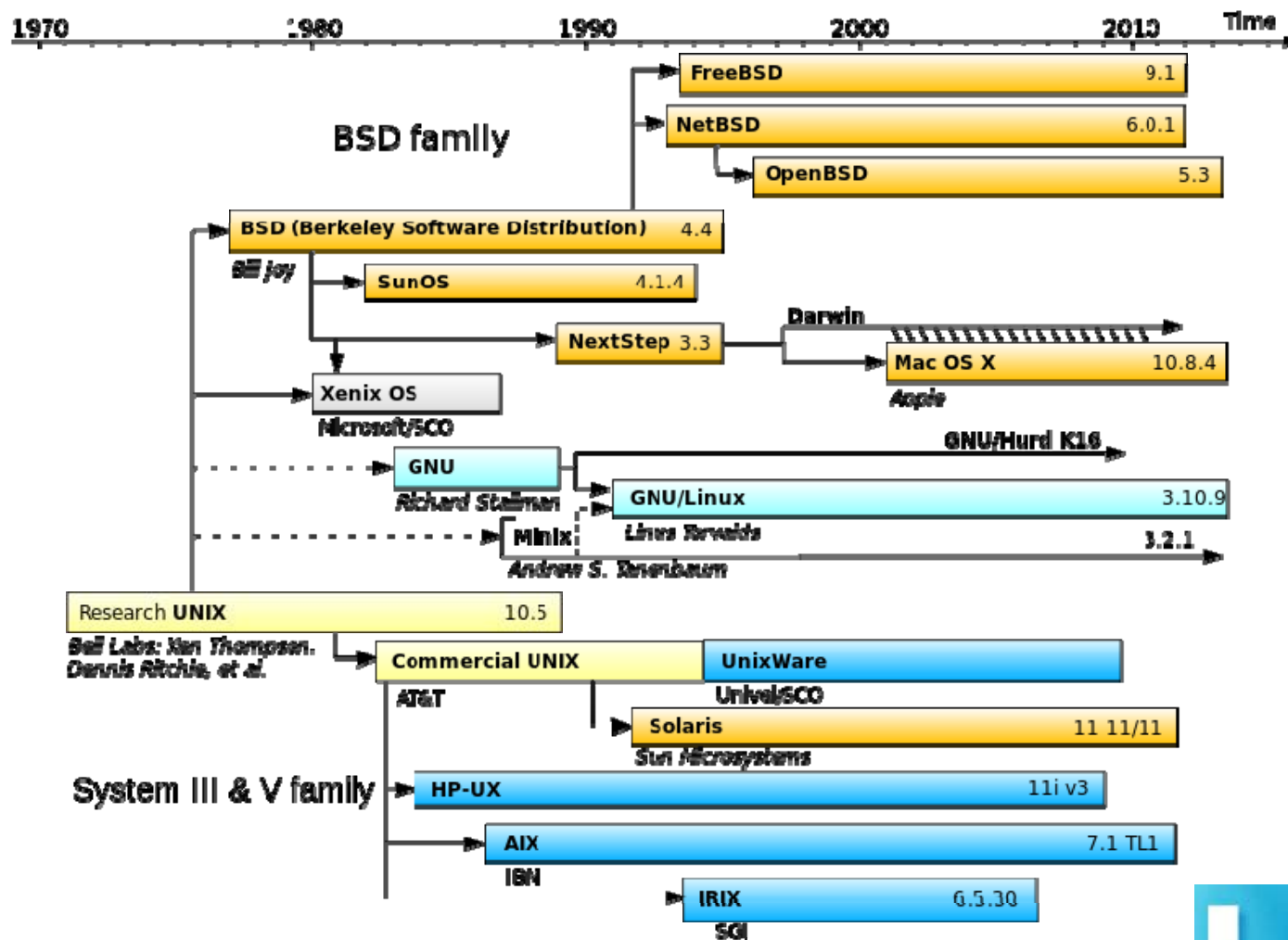
<https://groups.google.com/forum/?fromgroups=#!msg/comp.os.minix/dlNtH7RRrGA/SwRavCzVE7qJ>

## History of Linux (3)

- Linux is only the kernel, while an Operating System also requires applications that users can use.
- Combined with free software available from the GNU project gave birth to a new Operating System known as "GNU/Linux"
- GNU/Linux or simply Linux is released under the GNU Public License: Free to use, modify and distribute provided that you distribute under the GNU Public License.

*<http://en.wikipedia.org/wiki/Linux>*

# History of Linux (4)





# What is Linux (1)

- Linux is an operating system that evolved from a kernel created by Linus Torvalds
- Linux is the most popular OS used in a supercomputer.

OS family	Count	Share %
Linux	485	97
Unix	12	2.4
Windows	2	0.4
Mixed	1	0.2

- If you are using a supercomputer for your research, the chances are that it will be running an \*nix OS.

# What is Linux (2)

- Many software vendors release their own packages Linux OS, known as a distribution
  - Linux kernel + GNU system utilities + installation scripts + management utilities etc.
  - Debian, Ubuntu, Mint
  - Red Hat, Fedora, CentOS
  - Slackware, openSUSE, SLES, SLED
  - Gentoo
- Application packages on Linux can be installed from source or from customized packages
  - deb: Debian based distros, e.g. Debian, Ubuntu, Mint
  - rpm: Red Hat based distros, Slackware based distros
- Linux distributions offer a variety of desktop environment: KDE, GNOME, XFCE, LXDE, Cinnamon, MATE

# What is Linux (3)

- Linux distributions are tailored to different requirements such as
  - Server
  - Desktop
  - Workstation
  - Router
  - Embedded devices
  - Mobile devices (Android is a Linux-based OS)
- Almost any software that you use on Windows has a (roughly) equivalent software on Linux
  - Often multiple, and open source
- For a complete list, visit
  - [http://wiki.linuxquestions.org/wiki/Linux\\_software\\_equivalent\\_to\\_Windows\\_software](http://wiki.linuxquestions.org/wiki/Linux_software_equivalent_to_Windows_software)

# Linux Components

- Similar to all operating systems, Linux is made up of three parts
  - Kernel
  - Shell
  - Applications

# Linux Components (1)

- Kernel
  - The kernel is the core component of most operating systems
  - Kernel's responsibilities include managing the system's resources
  - It provides the lowest level abstraction layer for the resources (especially processors and I/O devices) that application software must control to perform its functions
  - It typically makes these facilities available to application processes through inter-process communication mechanisms and system calls

# Linux Components (2)

- Shell
  - The command line interface is the primary user interface to Linux/Unix operating systems.
  - Each shell has varying capabilities and features and the users should choose the shell that best suits their needs
  - The shell can be deemed as an application running on top of the kernel and provides a powerful interface to the system.

# Type of Shell

- sh (Bourne Shell)
  - Developed by Stephen Bourne at AT\&T Bell Labs
- csh (C Shell)
  - Developed by Bill Joy at University of California, Berkeley
- ksh (Korn Shell)
  - Developed by David Korn at AT&T Bell Labs
  - Backward-compatible with the Bourne shell and includes many features of the C shell
- bash (Bourne Again Shell)
  - Developed by Brian Fox for the GNU Project as a free software replacement for the Bourne shell
  - Default Shell on Linux and Mac OSX
  - The name is also descriptive of what it did, bashing together the features of sh, csh and ksh
- tcsh (TENEX C Shell)
  - Developed by Ken Greer at Carnegie Mellon University
  - It is essentially the C shell with programmable command line completion, command-line editing, and a few other features.

# Shell Comparison

Software	Sh	Csh	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>

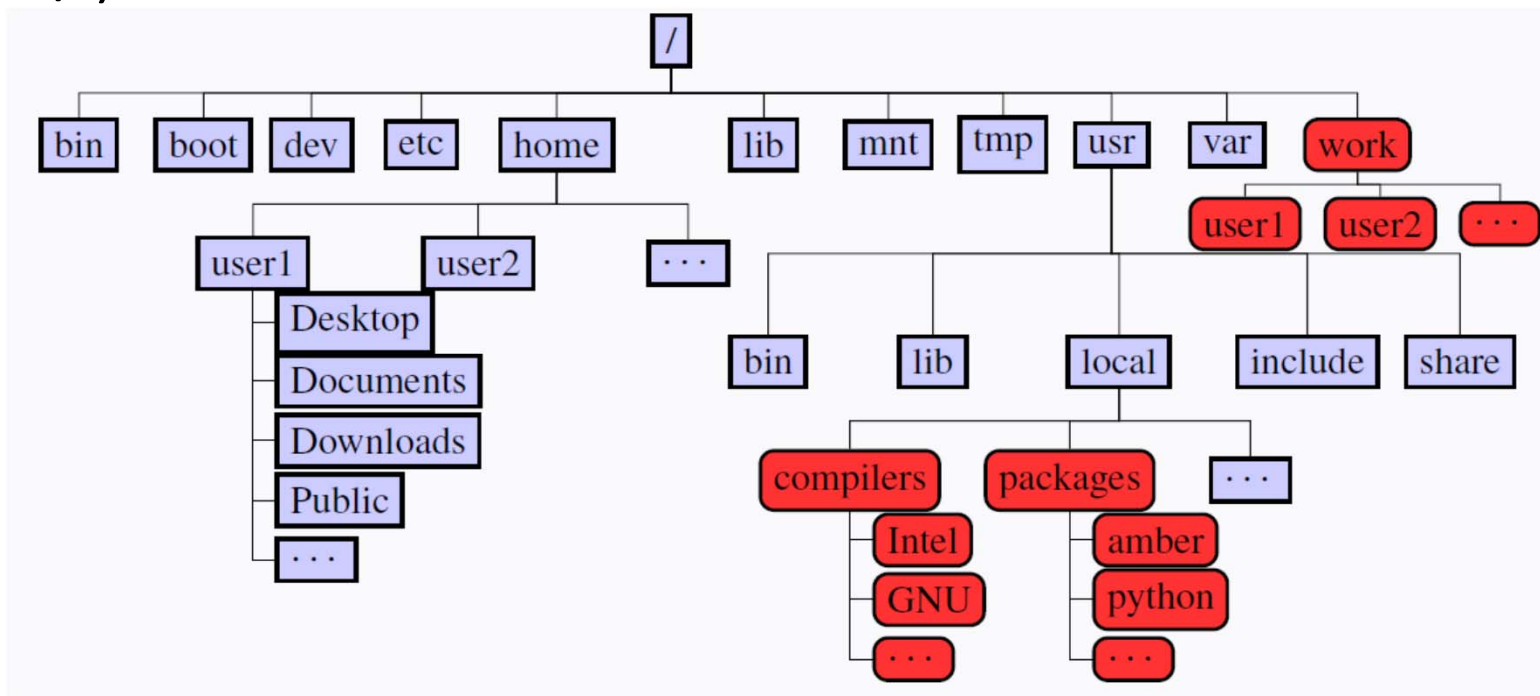


# Files and Processes

- Everything in Linux/UNIX is either a file or a process
- A file is a collection of data
- Example of files
  - Documents composed of ascii text
  - Program written in high level programming languages
  - Executables that you can run
  - Directory containing information about its content
- A process is an executing program identified by a unique process identifier, aka PID.

# Directory Structure

- All files are arranged in a hierarchial structure, like an inverted tree.
- The top of the hierarchy is traditionally called **root** (written as a slash /)



# 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.

# Path

- Path means a position in the directory tree
- You can either use relative or absolute path
- In absolute path expression
  - The path is defined uniquely and does NOT depend on the current location (path)
  - Example: /tmp is unique
- In relative path expression
  - The meaning of a relative path depends on the current location
  - . Is the current working directory
  - .. Is one directory up
  - You can combine . And .. To navigate the file system hierarchy
  - The relative path is not defined uniquely
  - Example: ../tmp is not unique since it depends on the current working directory

# User Groups

- Linux/UNIX OS's are designed for multi user environment, i.e. multiple users can exist on the system
- Special user call **root** is the administrator and has access to all files in the system
- Users are organized into groups
  - Use the groups command to find your group membership
- Each user is in at least one group and can be in multiple groups
  - On LONI systems, users are in one of the following groups: lsuusers, latechusers, unousers, ullusers, sususers, tulaneusers, loniusers or xavierusers
  - Due to software licensing, you cannot be in more than one of the above groups.
  - On LSU HPC system, users are in the Users group.
- Group membership makes it easier to share files with members of your group

# 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

# Outline

- What is Linux
- **Variables**
- Basic commands
- File permissions
- Processes and jobs
- File editing

# Variables

- Linux allows the use of variables
  - Similar to programming languages
- A variable is a named object that contains data
  - Number, character or string
- There are two types of variables: ENVIRONMENT and user defined
- Environment variables provide a simple way to share configuration settings between multiple applications and processes in Linux
  - Environment variables are often named using all uppercase letters
  - Example: PATH, LD\_LIBRARY\_PATH, DISPLAY etc.
- To reference a variable, prepend \$ to the name of the variable, e.g. \$PATH, \$LD\_LIBRARY\_PATH
  - Example: \$PATH, \$LD\_LIBRARY\_PATH, \$DISPLAY etc.



# Variables Names

- Rules for variable names
  - Must start with a letter or underscore
  - Number can be used anywhere else
  - Do not use special characters such as @, #, %, \$
  - (again) They are case sensitive
  - Example
    - Allowed: VARIABLE, VAR1234able, var\_name, \_VAR
    - Not allowed: 1var, %name, \$myvar, var@NAME

# Querying Environment Variables

- The command `printenv` list the current environmental variables.
- The command `env` is used to either print a list of environment variables or run another utility in an altered environment without having to modify the current existing environment.

# List of Some Environment Variables

PATH	A list of directory paths which will be searched when a command is issued
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
LD_LIBRARY_PATH	colon-separated set of directories where libraries should be searched for first
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 (1)

- How to assign values to variables depends on the shell

Type	sh/ksh/bash	csh/tcsh
Shell	<code>name=value</code>	<code>set name=value</code>
Environment	<code>export name=value</code>	<code>setenv name=value</code>

- Shell variables is only valid within the current shell, while environment variables are valid for all subsequently opened shells.

# Editing Variables (2)

- 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}
```

- sh/ksh/bash: no spaces except between export and PATH
- csh/tcsh: no “=” sign
- Use colon to separate different paths
- The order matters: If you have a customized version of a software say perl in your home directory, if you append the perl path to PATH at the end, your program will use the system wide perl not your locally installed version

# Exercise

## Exercise

- Create two shell variables containing
  - 1 your name  
e.g. MYNAME=Alex
  - 2 a standard greeting  
e.g. Greet=Hello
- We'll make use of these variables in a few slides when we learn some basic commands.

# Outline

- What is Linux
- Variables
- **Basic commands**
- File permissions
- Processes and jobs
- File editing

# Basic Commands

- Command is a directive to a computer program acting as an interpreter of some kind, in order to perform a specific task
- Command prompt is a sequence of characters used in a command line interface to indicate readiness to accept commands
  - Its intent is literally to prompt the 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
- Each command consists of three parts: name, options and arguments



# Get More Information On Commands

- **man** shows the manual for a command or program
  - The manual is a file that shows how to use the command and list the different options and arguments
  - Usage: `man <command name>`
  - Example: `man ls`
- **apropos** shows all of the man pages that may shed some light on a certain command or topic
  - Usage: `apropos <string>`
  - Example: `apropos editor`

# Commands: pwd and cd

- **pwd** command prints the current working directory
  - Usage: `pwd`
  - Example: `pwd`
- **cd** command allows one to change the current working directory
  - Usage: `cd [destination]`
  - Example: `cd /tmp`
  - The default destination is the home directory if `[destination]` is omitted
  - In bash `~` stands for home directory
  - In bash `-` stands for previous working directory

# 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`: show hidden files
    - Files whose name starts with an “.” is hidden
  - `-r`: reverse order when sorting
  - `-t`: show modification times
  - `-h`: use file sized in SI units (bytes, kbytes, megabytes etc.)

# Auto-completion

- Auto-completion of file name or command is a default feature in `bash` and `tcsh`
- It allows you to automatically complete the file, directory or command name that you are typing up to the next unique characters using the TAB key
  - Convenient, also error-proof
  - TAB will try to complete the command or filename; if there is no unique name, it will show all matching names
- 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
    - Example: `ls *[1-9]*` will list the file `test1a`, but not `testa`

# Commands: alias

- **alias** is a command to 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"`
- The `alias` command can be used to prevent files from being deleted accidentally
  - Example: `alias rm "rm -I"`
- Use the `alias` command without argument to list all aliases currently defined
- Use the `unalias` command to remove an alias

# Commands: mkdir

- **mkdir** is a command to 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 directory that appears in the path but does not exist

# Commands: cp

- **cp** is a command to 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 without giving a warning (!!!)
- If there are more than one source files, then the destination must be a directory



# Commands: rm

- **rm** commands removes files and directories
- Usage: `rm <options> <list of files and/or directories>`
- 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!!!**
  - Use alias if you are paranoia about the safety of your files

# Commands: mv

- **mv** command 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: cat, more and less

- To display a file to screen, Linux provides three commands
- **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>`
- Be careful when using those commands on binary files
  - The **file** command reveal what type of file the target is

# Input & Output Commands (1)

- The basis I/O statement are `echo` for displaying to screen and `read` for reading input from screen/keyboard/prompt
- **echo**
  - 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**
  - The `read` statement takes all characters typed until the Enter key is pressed
  - Usage: `read <variable name>`
  - Example: `read name`

# Input & Output Commands (2)

- Examples

```
[lyan1@mike2 ~]$ echo $SHELL
/bin/bash
[lyan1@mike2 ~]$ echo Welcome to HPC      training
Welcome to HPC training
[lyan1@mike2 ~]$ echo "Welcome to HPC      training"
Welcome to HPC      training
```

- 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

# Quotation

- Double quotation
  - Enclosed string is expanded
- Single quotation
  - Enclosed string is read literally
- Back quotation
  - Enclose string is executed as a command

# Quotation

- Examples

```
[lyan1@mike2 ~]$ str1="I am $USER"  
[lyan1@mike2 ~]$ echo $str1  
I am lyan1  
[lyan1@mike2 ~]$ str2='I am $USER'  
[lyan1@mike2 ~]$ echo $str2  
I am $USER  
[lyan1@mike2 ~]$ str3=`echo $str2`  
[lyan1@mike2 ~]$ echo $str3  
I am $USER
```

# Exercise

## Exercise

- Print out the variable you created a few slides back

```
echo $MYNAME 
```

```
echo $Greet 
```

- Read a variable for greeting message

```
read message 
```

```
Welcome to HPC 
```

- Combine and print your name, the greeting and the message

```
echo $Greet $MYNAME $message 
```

- What is the output of the following command?

```
echo $Greet $MYNAME, $message Training 
```



# Other Useful Commands (1)

- passwd: change password (**does not work on LSU HPC and LONI systems**)
- chsh: change default shell (**does not work on LSU HPC and LONI systems**)
- df: report disk space usage by filesystem
- du: estimate file space usage - space used under a particular directory or files on a file system.
- sudo: run command as root (**only if you have access**)
- mount: mount file system (**root only**)
- umount: unmount file system (**root only**)
- shutdown: reboot or turn off machine (**root only**)
  - top: Produces an ordered list of running processes
  - free: Display amount of free and used memory in the system
  - file: Determine file type
- touch: change file timestamps or create file if not present
- date: display or set date and time

# Other Useful Commands (2)

find : Find a file

```
find /dir/to/search -name file-to-search
```

wc: Count words, lines and characters in a file

```
wc -l .bashrc
```

grep: Find patterns in a file

```
grep alias .bashrc
```

awk: File processing and report generating

```
awk '{print $1}' file1
```

sed: Stream Editor

```
sed 's/home/HOME/g' .bashrc
```

set: manipulate environment variables

```
set -o emacs
```

ln: Link a file to another file

```
ln -s file1 file2
```

# Other Useful Commands (3)

**wait:** wait until all backgrounded jobs have completed

**which:** shows the full path of (shell) commands

**whatis:** display manual page descriptions


**!name:** rerun previously executed command with the same arguments as before,  
**name <args>.**

Note that you do not always have to type the full command **name**, just the minimum unique characters (no spaces) of **name** need to be entered.

If you had entered two commands **name <args>** and **nbme <args>**, then to rerun **name**, use the command **!na**  .

**history:** display a list of last executed commands. Optional argument **m** will list the last **m** commands.

All previously executed commands will be listed with a number **n**.

To rerun a command from history which has number **n**, run the command  
**!n** 

To learn more about these commands, type **man command** on the command prompt

# How to Log into Remote Systems

- Most Linux systems allocate secure shell connections from other systems
- You need to log in using the `ssh` command to the LSU HPC and LONI clusters
- Usage: `ssh <username>@<remote host name>`
- Example: `ssh lyan1@eric.loni.org`
- If you need to forward the display of an application, add `-X` option
- The default port is 22 for `ssh`
  - If the remote machine is listening to a non-default port (i.e. different from 22), you need to specify the port number: `ssh -p <port number> <username>@<hostname>`

# File Transfer between Two Systems (1)

- **scp** is a command to copy files between two \*nix 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, you can omit `<user>@>`
- If transferring files from or to localhost, the `<user>@<host>:` option can be omitted
- Options are `-r` and `-p`, same meaning with `cp`
- Examples

```
scp lyan1@mike.hpc.lsu.edu:/work/lyan1/somefile .  
scp -r code lyan1@eric.loni.org:/home/lyan1
```



# File Transfer between Two Systems (2)

- **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
- `rsync` is 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 Files (1)

- Quite often you need to compress and uncompress files to reduce storage usage or bandwidth while transferring files.
- \*nix systems have built-in utilities to compress and uncompress files
  - To compress, the commands are: **gzip, zip, bzip2**
  - To uncompress, the commands are: **gunzip, unzip, bunzip2**
- Options
  - To compress/uncompress files recursively, use the **-r** option
  - To overwrite files while compressing/uncompressing, use the **-f** option
- By convention
  - Gzipped files have extension **.gz, .z or .Z**
  - Zipped files have extension **.Zip or .zip**
  - Bzipped files have extension **.bz2 or .bz**

# Compressing and Archiving Files

- \*nix provides the **tar** package to create and manipulate streaming archive of files.
- Usage: `tar <options> <file> <patterns>`
  - `<file>` is name of the tar archive file, usually with extension `.tar`
  - `<patterns>` are pathnames for files/directories being archived
- Common options
  - `-c`: create an archive file
  - `-x`: extract an archive file
  - `-z`: filter the archive through `gzip`
  - `-j`: filter the archive through `bzip2`
  - `-t`: list contents of archive
  - `-v`: verbosely list files processed
- Example: `tar -cvfz myhome.tar.gz ${HOME} / *`
- This is useful for creating a backup of your files and directories that you can store at some storage facility e.g. external disk.



# I/O Redirection

- There are three file descriptors for I/O streams (remember everything is a file in Linux)
  - STDIN: Standard input
  - STDOUT: standard output
  - STDERR: standard error
- 1 represents STDOUT and 2 represents STDERR
- I/O redirection allows users to connect applications
  - <: connects a file to STDIN of an application
  - >: connects STDOUT of an application to a file
  - >>: connects STDOUT of an application by appending to a file
  - |: connects the STDOUT of an application to STDIN of another application.

# I/O Redirection Examples

- Write STDOUT to file: `ls -l > ls-l.out`
- Write STDERR to file: `ls -l &2 > ls-l.err`
- Write STDERR to STDOUT: `ls -l 2>&1`
- Send STDOUT as STDIN for another application: `ls -l | less`

# Outline

- What is Linux
- Variables
- Basic commands
- **File permissions**
- Processes and jobs
- File editing

# File Permission (1)

- Since \*NIX OS's are designed for multi user environment, it is necessary to restrict access of files to other users on the system.
- In \*NIX OS's, you have three types of file permissions
  - Read (r)
  - Write (w)
  - Execute (x)
- for three types of users
  - User (u) (owner of the file)
  - Group (g) (group owner of the file)
  - World (o) (everyone else who is on the system)

# File Permission (2)

```
[lyan1@mike2 ~]$ ls -al
```

```
total 4056
```

```
drwxr-xr-x  45 lyan1 Admins      4096 Sep  2 13:30 .
drwxr-xr-x 509 root  root      16384 Aug 29 13:31 ..
drwxr-xr-x   3 lyan1 root        4096 Apr  7 13:07 adminscript
drwxr-xr-x   3 lyan1 Admins      4096 Jun  4 2013 allinea
-rw-r--r--   1 lyan1 Admins         12 Aug 12 13:53 a.m
drwxr-xr-x   5 lyan1 Admins      4096 May 28 10:13 .ansys
-rwxr-xr-x   1 lyan1 Admins    627911 Aug 28 10:13 a.out
```

- The first column indicates the type of the file
  - d for directory
  - l for symbolic link
  - - for normal file

# File Permission (2)

```
[lyan1@mike2 ~]$ ls -al
```

```
total 4056
```

drwxr-xr-x	45	lyan1	Admins	4096	Sep	2	13:30	.
drwxr-xr-x	509	root	root	16384	Aug	29	13:31	..
drwxr-xr-x	3	lyan1	root	4096	Apr	7	13:07	adminscript
drwxr-xr-x	3	lyan1	Admins	4096	Jun	4	2013	allinea
-rw-r--r--	1	lyan1	Admins	12	Aug	12	13:53	a.m
drwxr-xr-x	5	lyan1	Admins	4096	May	28	10:13	.ansys
-rwxr-xr-x	1	lyan1	Admins	627911	Aug	28	10:13	a.out

- The next nine columns can be grouped into three triads, which indicates what the owner, the group member and everyone else can do

# File Permission (2)

```
[lyan1@mike2 ~]$ ls -al
```

```
total 4056
```

```
drwxr-xr-x 45 lyan1 Admins 4096 Sep 2 13:30 .
drwxr-xr-x 509 root root 16384 Aug 29 13:31 ..
drwxr-xr-x 3 lyan1 root 4096 Apr 7 13:07 adminscript
drwxr-xr-x 3 lyan1 Admins 4096 Jun 4 2013 allinea
-rw-r--r-- 1 lyan1 Admins 12 Aug 12 13:53 a.m
drwxr-xr-x 5 lyan1 Admins 4096 May 28 10:13 .ansys
-rwxr-xr-x 1 lyan1 Admins 627911 Aug 28 10:13 a.out
```

- We can also use weights to indicate file permission
  - $r=4, w=2, x=1$
  - Example:  $rwX = 4+2+1 = 7, r-x = 4+1 = 5, r-- = 4$
  - This allows us to use three numbers to represent the permission
  - Example:  $rwXr-xr-w = 755$

# File Permission (3)

- Difference between files and directories

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



# Changing File Permission

- **chmod** is a \*NIX command to change permissions on a file
- Usage: `chmod <option> <permissions> <file or directory name>`
- To change permission recursively in a directory, use the `-R` option
- Example:
  - To give user `rw`, group `rx` and world `x` permission, the command is:  
`chmod 751 <file name>`
- The symbolic representation of permission works with `chmod` too
  - Use `[u|g|o|a][+|-][rwx]` in place of `<permissions>`
  - Example:
    - Add read and execution permission to the owner and group members:  
`chmod ug+rx hello.sh`

# Changing Group Membership

- The **chgrp** command is used to change the group ownership between two groups that you are a member of.
- Usage: `chgrp <options> <new group> <file name>`
- The `-R` option works with `chgrp` as well

# Outline

- What is Linux
- Variables
- Basic commands
- File permissions
- **Processes and jobs**
- File editing

# Processes and Jobs

- A process is an executing program identified by a unique PID
  - To see information about your running processes and their PID and status, use the **ps** or **top** command
- A process may be in foreground, background or be suspended.
  - Processes running in foreground, the command prompt is not returned until the current process has finished executing.
  - If a job takes a long time to run, put the job in background in order to obtain the command prompt back to do some other useful work
  - There are two ways to send a job into the background:
    - Add an ampersand & to the end of your command to send it into background directly.
    - First suspend the job using **Ctrl-z** and then type **bg** at the command prompt.
      - If you type **fg** then the job will run in foreground again and you will lose the command prompt.

# Output from top

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
26927	ommolina	20	0	376m	223m	18m	R	56.9	0.3	0:07.74	PreEngine.exe
26923	ommolina	20	0	536m	175m	28m	S	29.8	0.3	0:08.89	PreGui_ogl.exe
26666	ommolina	20	0	108m	2792	1188	S	8.6	0.0	0:01.71	sshd
20937	tliyan1	20	0	108m	2368	1052	S	1.7	0.0	0:00.79	sshd
20938	tliyan1	20	0	58096	2560	1592	S	0.7	0.0	0:00.42	sftp-server
28980	lyan1	20	0	17776	1960	972	R	0.7	0.0	0:00.06	top
3427	ajbarley	20	0	111m	5060	1112	S	0.3	0.0	0:28.35	sshd
4775	root	20	0	0	0	0	S	0.3	0.0	28:35.68	kiblnsd_sd_15
4777	root	20	0	0	0	0	S	0.3	0.0	332:01.31	ptlrpcd-brw
19407	tloeffl	20	0	111m	5864	1500	S	0.3	0.0	0:00.37	bash
27122	ommolina	20	0	108m	2260	1188	S	0.3	0.0	0:00.10	sshd
27296	ommolina	20	0	266m	26m	15m	S	0.3	0.0	0:00.44	SolverManager.e
1	root	20	0	21416	1376	1132	S	0.0	0.0	0:30.41	init

# Managing Processes and Jobs

- When a process is running, background or suspended, it will be entered onto a list along with a job number (not PID), which can be listed by the **jobs** command
- To restart a suspended job in foreground or background, type
  - fg %<job number>**
  - bg %<job number>**
- 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>**
- **pstree**: display a tree of processes
- **pkill**: kill process by its name, user name, group name, terminal, UID, EUID, and GID.

# Outline

- What is Linux
- Variables
- Basic commands
- File permissions
- Processes and jobs
- File editing

# File Editing

- The two most commonly used editors on Linux/Unix systems are:
  - vim 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 CLI and a graphical user interface (GUI).
  - If emacs GUI is installed then use `emacs -nw` to open file in console
- Other editors you may come across: kate, gedit, gvim, pico, nano, kwrite
- To use vi or emacs is your choice, but you need to know one of them



# File Editing

- Emacs has only one mode
- vi has 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” to go back to the command mode

# Editor cheatsheet

## Insert/Appending Text

- insert at cursor
- insert at beginning of line
- append after cursor
- append at end of line
- newline after cursor in insert mode
- newline before cursor in insert mode
- append at end of line
- exit insert mode

## vi

- i
- I
- a
- A
- o
- O
- ea
- ESC

# Editor cheatsheet

## 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
- ^
- \$
- nG
- 1G
- G
- C-u
- C-d

## emacs

- C-b
- C-n
- C-p
- C-f
- C-a
- C-e
- M-x goto-line  $\leftarrow$  n
- M-<
- M->
- M-v
- C-v

# Editor cheatsheet

## 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
- ^
- \$
- nG
- 1G
- G
- C-u
- C-d

## emacs

- C-b
- C-n
- C-p
- C-f
- C-a
- C-e
- M-x goto-line  $\leftarrow$  n
- M-<
- M->
- M-v
- C-v

# Editor cheatsheet

## 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

- A script is a program written for a software environment that automate the execution of tasks which could alternatively be executed one-by-one by a human operator.
  - Shell scripts are a series of shell commands put together in a file
  - When the script is executed, it is as if someone type those commands on the command line
- The majority of script programs are “quick and dirty”, where the main goal is to get the program written quickly.
  - Might not be as efficient as programs written in C and Fortran, with which source files need to be compiled to get the executable
- We do not have time to cover how to write a script today
  - Check out the “shell scripting” tutorials in our archives

# Startup Scripts

- When you login to a \*NIX computer, shell scripts are automatically loaded depending on your default shell
- sh/ksh (in the specified order)
  - /etc/profile
  - \$HOME/.profile
- bash (in the specified order)
  - /etc/profile (for login shell)
  - /etc/bashrc or /etc/bash/bashrc
  - \$HOME/.bash\_profile (for login shell)
  - \$HOME/.bashrc
- csh/tcsh (in the specified order)
  - /etc/csh.cshrc
  - \$HOME/.tcshrc
  - \$HOME/.cshrc (if .tcshrc is not present)
- .bashrc, .tcshrc, .cshrc, .bash\\_profile} are script files where users can define their own aliases, environment variables, modify paths etc.



# An Example

```
# .bashrc

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

# User specific aliases and functions
alias c="clear"
alias rm="/bin/rm -i"
alias psu="ps -u apache"
alias em="emacs -nw"
alias ll="ls -lF"
alias la="ls -al"
export PATH=/home/apache/bin:${PATH}
export g09root=/home/apache/Software/Gaussian09
export GAUSS_SCRDIR=/home/apache/Software/scratch
source $g09root/g09/bsd/g09.profile

export TEXINPUTS=./usr/share/texmf//:/home/apache/LaTeX//:${TEXINPUTS}
export BIBINPUTS=./home/apache/TeX//:${BIBINPUTS}
```



# 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>
- Online courses: <http://moodle.hpc.lsu.edu>
- Contact us
  - Email ticket system: [sys-help@loni.org](mailto:sys-help@loni.org)
  - Telephone Help Desk: 225-578-0900
  - Instant Messenger (AIM, Yahoo Messenger, Google Talk)
    - Add “lsuhpchelp”

# Questions?

## Exercise (1)

- Login to a Linux machine and open a terminal
- Enter the following commands or carry out operations asked for.
- Understand what you are doing and ask for help if unsure. Some commands are incorrect or will fail; if this is the case, enter the correct ones

## Exercise (2)

- `echo hello world`
- `pwd`
- `whoami`
- `cd /tmp`
- `cd -`
- `mkdir test/testagain`
- `cd test/testagain`
- `touch file`
- Go back to your home directory
- Figure out which shell you are using

## Exercise (3)

- Create an alias for removing files which prompt for confirmation and delete the file that you created.
- From your home directory get a list of files and directory in long format in reverse order with file sizes listed in human readable format.
- (On HPC or LONI clusters) Find out the location of `vi`, `emacs`, `perl` and `ifort`.
- Change the permission of the `testagain` directory to be world writable.
- Open a few applications of choice in foreground one by one and then suspend them,
- Get a list of suspended jobs,
- Foreground job 1 and close it,
- Background job 2,
- Kill job 3,
- Put job 2 in foreground and close it,
- Check if you still have any jobs running.