Introduction to Linux

Le Yan

High Performance Computing @ LSU

Based on the tutorial developed by Alex Pacheco

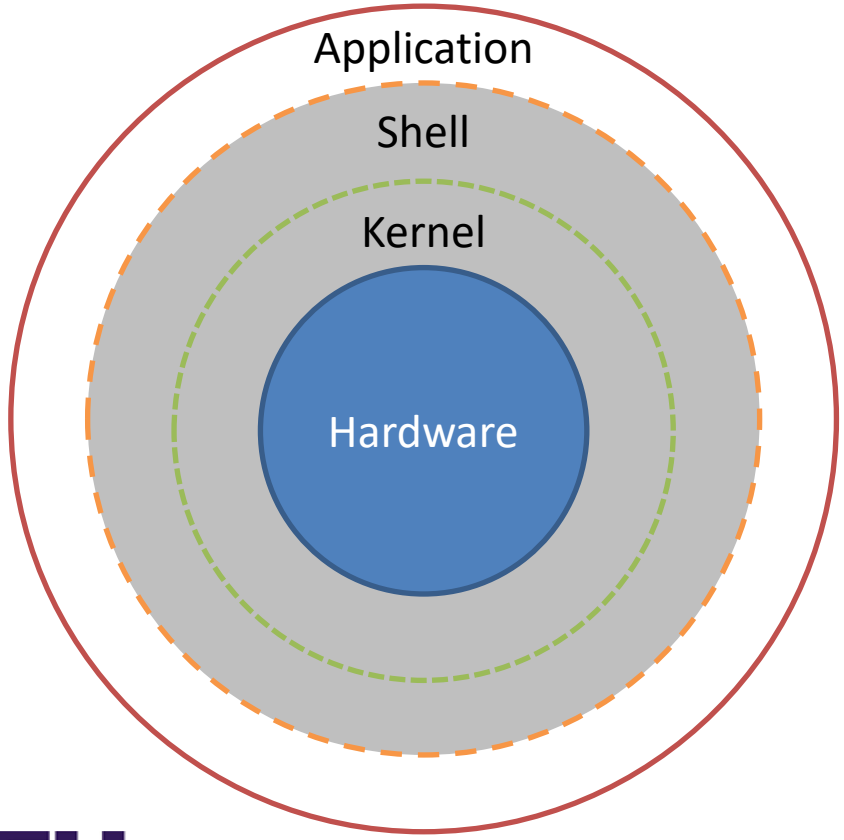
Outline

- What is Linux – basic concepts
- Basic commands
- File permissions
- Variables
- File editing

Outline

- **What is Linux – basic concepts**
- Basic commands
- File permissions
- Variables
- File editing

What Do Operating Systems Do?



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

: Operating System

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 assembly language.
- In 1973, Unix was re-written in the C programming language 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 everyone 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)

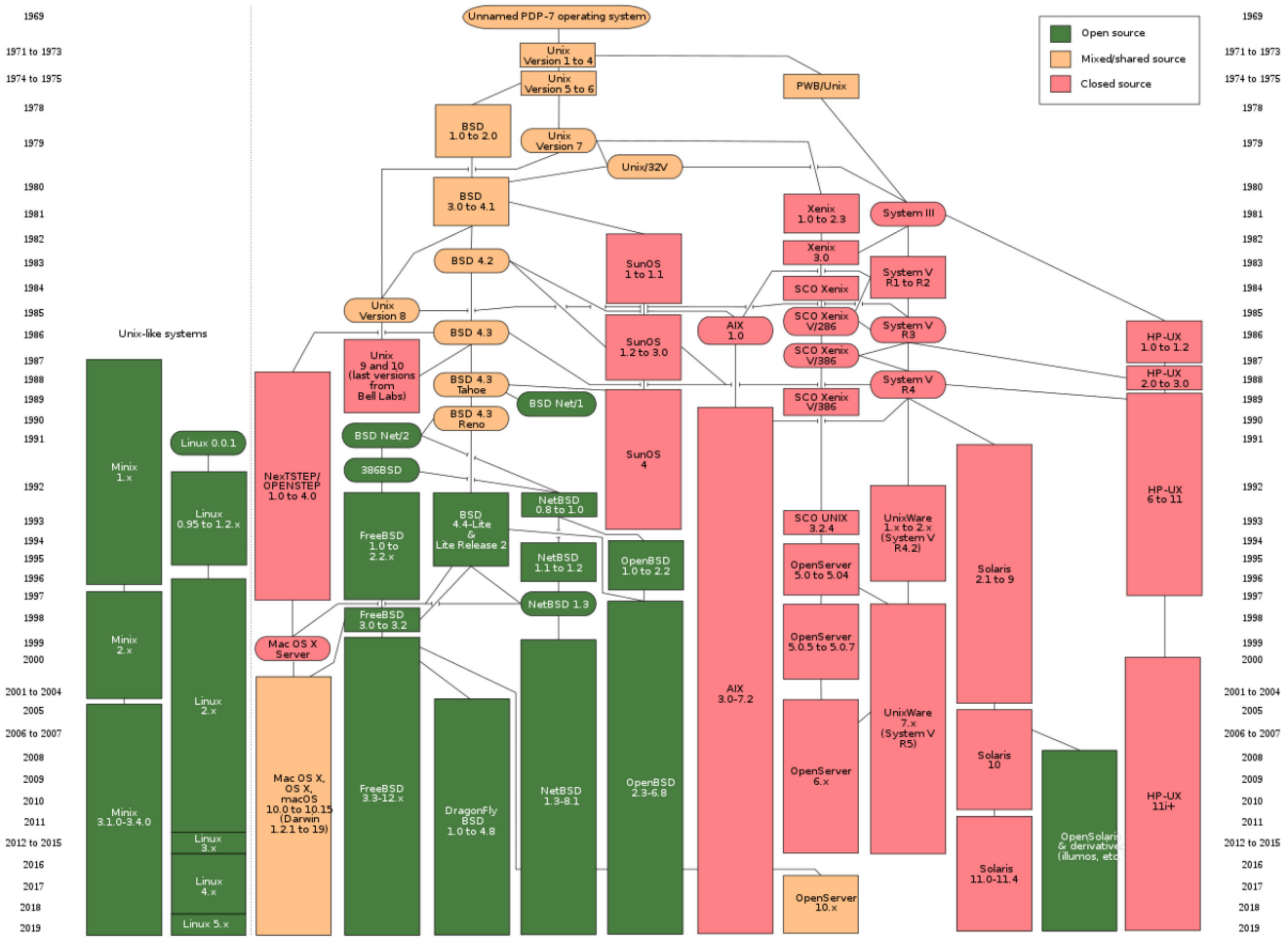
PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

https://groups.google.com/forum/?fromgroups=#!msg/comp.os.minix/dlNtH7RRrGA/SwRavCzVE7gJ

History of Linux (3)

- Linux is actually only the kernel, while a full 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



https://en.wikipedia.org/wiki/Unix-like

What is Linux (1)

- Linux is an operating system that evolved around the kernel created by Linus Torvalds
- Linux is the most popular OS used in supercomputers/clusters
  - If you are using a HPC cluster for your research, chances are that it will be running some Linux (or Unix-like/*nix) OS.

http://www.top500.org/statistics/list/

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

Linux Components (1)

- Kernel
  - The kernel is the core component of most operating systems
  - The kernel's responsibility is to manage/control the system's hardware 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-like 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

Shell	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 auto completion	n	y*	y*	y	y
Command line editing	n	n	y*	y	y
Job control	n	y	y	y	y

*: not by default

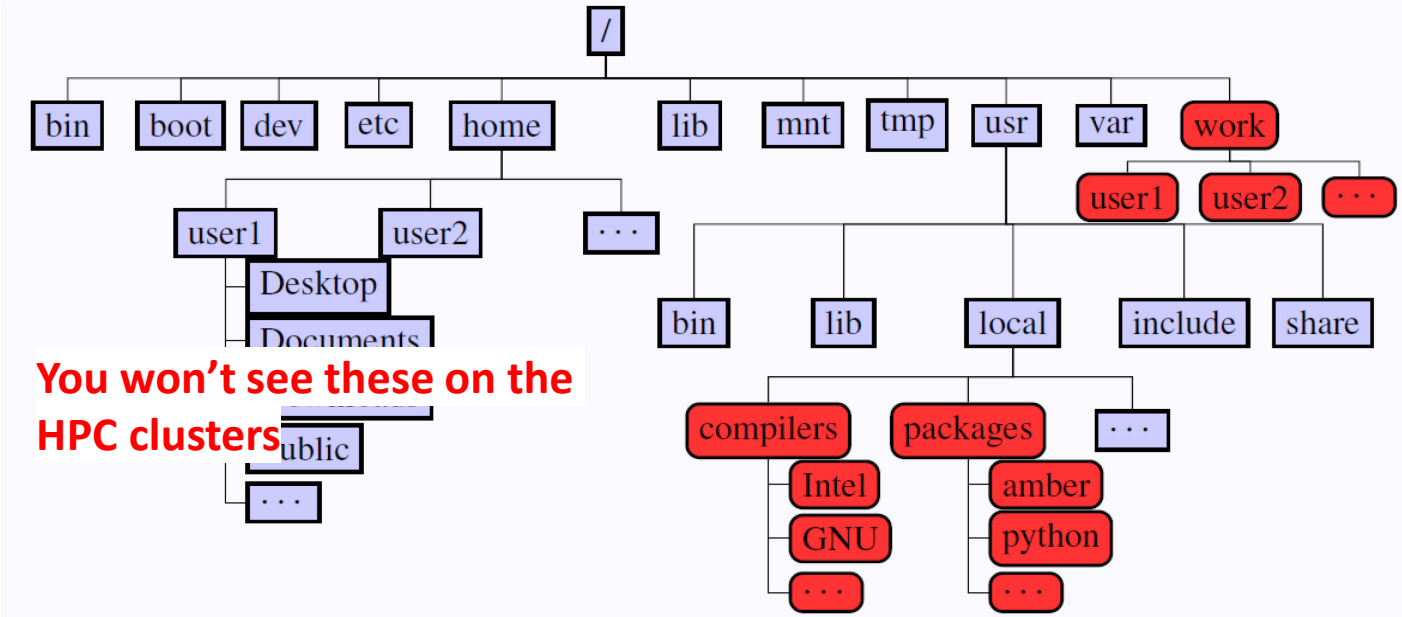
http://www.cis.rit.edu/class/simg211/unixintro/Shell.html

Files and Processes

- Nearly everything in Linux/UNIX is abstracted as either a file or a process
- A file (in most cases) is a collection of data
  - Example of files
    - Documents composed of ASCII text
    - Executables or binaries
    - Directories
    - Devices
- A process is an executing program identified by a unique process identifier, aka PID.

Directory Structure

- All files are arranged in a hierarchical 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 **root**
`/tmp`	temporary file storage
`/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.
`/work` OR `/scratch`	Is used for job input/output on HPC clusters. Not likely to be present on other computing systems.

Path (1)

- Path is used to indicate a position in the directory tree
  - Example: `/usr/local/packages/some_application/some_file`
- A path can be either **absolute** or **relative**
- **Absolute paths**
  - Starts with "/"
  - Is defined uniquely
  - Does **NOT** depend on the current location
  - Example: `/tmp/some_file` is unique

Path (2)

- **Relative paths**
  - The meaning of a relative path depends on the current location
  - `.` is the shorthand for the current location
  - `..` is the shorthand for 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 location

User Groups

- Linux/UNIX operating systems are designed for a multi-user environment, i.e. multiple users exist on the system at the same time
- Special user called **root** has access to all files in the system as well as many privileges
- Users are organized into groups
  - Use the `groups` command to find out 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`, `susers`, `tulaneusers`, or `loniusers`
  - 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

Outline

- What is Linux – basic concepts
- **Basic commands**
- File permissions
- Variables
- File editing

Linux Is Case Sensitive

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

Basic Commands

- Linux command syntax

```
Command <options> <arguments>
```

- Command prompt is a sequence of characters used in a command line interface to indicate readiness to accept commands
  - A prompt usually ends with one of the characters $,%#,:,> and often includes information such as the user name and the current working directory

```
[2022-01-18 11:49:15 lyan1@qbc2 ~]$ ls
```

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`
- **info** shows more information about a command
- **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`
- **`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 (`$OLDPWD`)

Commands: `ls`

- **`ls`** command list the contents of a directory
  - Usage: `ls <options> <path>`
  - The default path will be current working directory if `path` is omitted.
- Options
  - `-l`: show long listing format
  - `-a`: show hidden files
    - By default, files whose name starts with an "." is hidden from "`ls`"
  - `-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: `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> <source1> <source2> … <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!!!**
  - Unless a backup exists, which is not the case for most files on the HPC clusters
  - Use `alias` if you are paranoia about the safety of your files

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: `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 the content of a file on 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
  - "less is more"
- If the file is large
  - **`head`**: display the first few lines
  - **`tail`**: display the last few lines
- Usage: `cat/more/less/head/tail <options> <filename>`
- Be careful when using those commands on binary files
  - The **`file`** command reveal what type of file the target is

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** Enter .

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** Enter

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@qb.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 Linux 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.
- Linux 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 (2)

- Linux 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`

Processes

- 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

```
  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 kiblnd_sd_15
 4777 root      20   0      0    0    0 S   0.3   0.0 332:01.31 ptlrpcd-brw
19407 tloeff1   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
```

Sending Processes to Background

- A process may be in foreground, background or be suspended.
  - When a processes is running in foreground, the command prompt is not returned until the current process has finished executing.
  - If a process takes a long time to run, put it in background so that you can get the command prompt back to do some other useful work
  - There are two ways to send a process into the background:
    - Add an ampersand `&` to the end of your command to send it into background directly.
    - First suspend the process 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.

Managing Processes

- To kill or terminate a process:
  - Job running in foreground: `Ctrl-c`
  - 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 – basic concepts
- Basic commands
- **File permissions**
- Variables
- File editing

File Permission (1)

- Since Linux is designed for a multi-user environment, it is necessary to restrict access of files to other users on the system.
- In Linux, you have three types of file permissions
  - Read (r)
  - Write (w)
  - Execute (x)
- …for each of the 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 Linux 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 rwx, 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 – basic concepts
- Basic commands
- File permissions
- **Variables**
- File editing

Variables (1)

- 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**

Variables (2)

- **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: the content of the `PATH`variable is a list of directories where Linux search for commands when they are issued by the user
- To reference a variable, prepend `$` to the name of the variable
  - Example: `$PATH`, `$LD_LIBRARY_PATH`, `$DISPLAY` etc.

```
$ echo $PATH
/usr/local/packages/mvapich2/2.3.3/nc23zj7q/bin:/usr/local/package
s/Modules/4.5.0/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/s
bin:/usr/local/bin
```

Command: `echo`

- **`echo`**
  - Print arguments to screen or standard output, where `arguments` can be a single or multiple variables, string or numbers
- 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

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`

Editing Variables (1)

- How to assign values to variables depends on the shell

Type	sh/ksh/bash	csh/tcsh
User defined	`name=value`	`set name=value`
Environment	`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.

```
$ myvar=hello

$ echo $myvar
hello

$ cat runme.sh
echo $myvar

$ bash runme.sh


$ export myvar=hello

$ bash runme.sh
hello
```

Editing Variables (3)

- 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 there are executables with the same name but located in different directories, the first encountered in `PATH` will be executed when you type its name at the command line

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 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.

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
```

Outline

- What is Linux – basic concepts
- Basic commands
- File permissions
- Variables
- **File editing**

File Editing

- The two most commonly used editors on Linux/Unix systems are:
  - `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 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	vi
insert at cursor	`i`
insert at beginning of line	`I`
append after cursor	`a`
append at end of line	`A`
newline after cursor in insert mode	`o`
newline before cursor in insert mode	`O`
append at end of line	`ea`
exit insert mode	`ESC`

Editor cheatsheet

Cursor Movement	vi	emacs
move left	h	C-b
move down	j	C-n
move up	k	C-p
move right	l	C-f
jump to beginning of line	^	C-a
jump to end of line	$	C-e
goto line n	nG	M-x goto-line ⟵ n
goto top of file	1G	M-<
goto end of file	G	M->
move one page up	C-u	M-v
move one page down	C-d	C-v

Editor cheatsheet

Cursor Movement	vi	emacs
move left	`h`	`C-b`
move down	`j`	`C-n`
move up	`k`	`C-p`
move right	`l`	`C-f`
jump to beginning of line	`^`	`C-a`
jump to end of line	`$`	`C-e`
goto line `n`	`nG`	`M-x goto-line ⟵ n`
goto top of file	`1G`	`M-<`
goto end of file	`G`	`M->`
move one page up	`C-u`	`M-v`
move one page down	`C-d`	`C-v`

Editor cheatsheet

File Manipulation (contd)	vi	emacs
replace a character	`r`	
join next line to current	`J`	
change a line	`cc`	
change a word	`cw`	
change to end of line	`c$`	
delete a character	`x`	`C-d`
delete a word	`dw`	`M-d`
edit/open file *file*	`:e file`	`C-x C-f file`
insert file *file*	`:r file`	`C-x i file`
split window horizontally	`:split or C-ws`	`C-x 2`
split window vertically	`:vsplit or C-wv`	`C-x 3`
switch windows	`C-ww`	`C-x o`

Shell Scripts

- A script is a program written to automate the execution of tasks
  - A shell scripts is 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, but no compilation is needed
- We will have a separate “shell scripting” tutorial in 3 weeks

Startup Scripts

- When you login to a Linux computer, some shell scripts are automatically executed depending on your default shell
  - Users can use them to define/configure the environment
- `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)

An Example

```
# .bashrc

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

# User specific aliases and functions

export PATH=$HOME/packages/eFindsite/bin:$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
  - LONI: http://hpc.loni.org/docs/guides.php
- Contact us
  - Email ticket system: sys-help@loni.org
  - Telephone Help Desk: 225-578-0900

Questions?