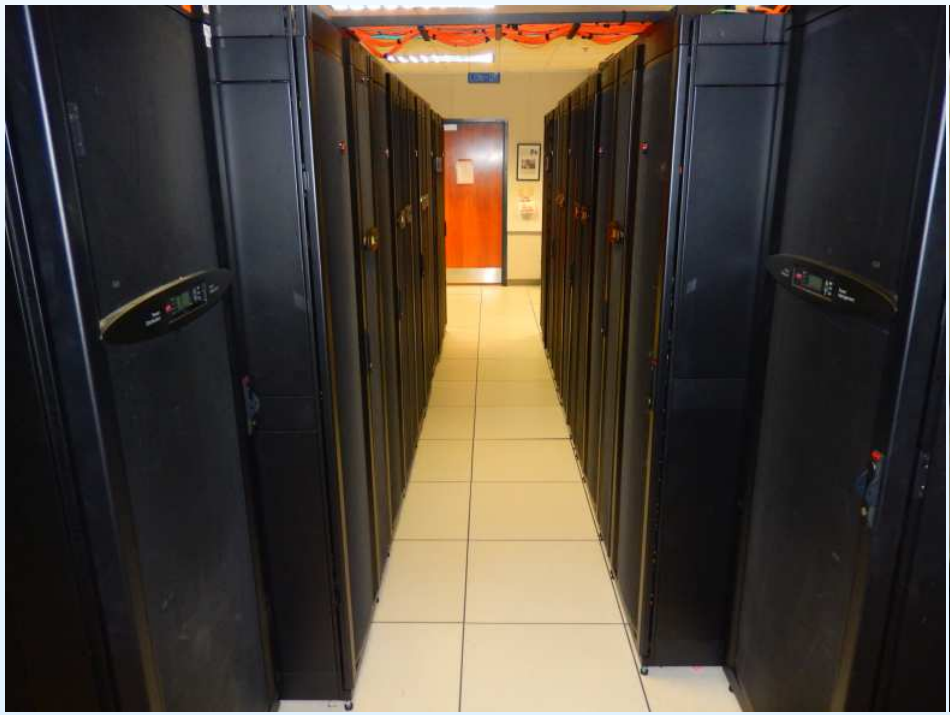
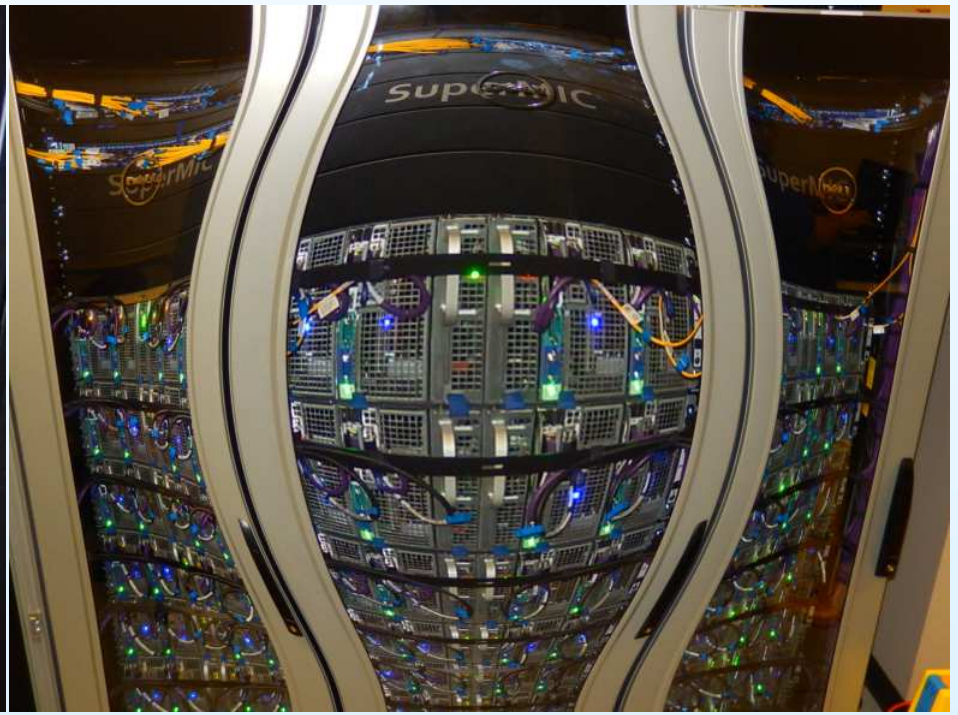
Introduction to Linux

Xiaoxu Guan

High Performance Computing, LSU

January 31, 2018





Outline

- What is an OS or **Linux** OS?
- Basic commands for **files/directories**
- Basic commands for **text** processing
- Linux **globbing** wildcards
- The **process** basics
- File **permission** and management
- Basic **vi/vim** editing

What is an Operating System (OS)?

- An **OS** is a software suite that controls and manages a computer's hardware along with providing a platform or environment we can run applications and programs on;
- Essential features on most systems:
  - Manage and maintain **memory** system (load applications, allocate and free up memory space);
  - Provide **low-level** support to interact with hardware (system calls);
  - Support **Input/Output** (I/O);
  - Multi-task OS even on a single-core system;
  - A modern OS is **multi-tasking**, **multi-user**, and **multi-processing** system;
  - **Environment** (CLI/GUI) supporting a user interaction with OS;

Linux and Unix

- **Unix** was created by Ken Thompson, Dennis Richie, and others (Bell Labs) in `C` in the late 1960s and early 1970s;
- A variety of UNIX systems available: `AIX` (IBM), `HP-UX` (HP), `IRIX` (SGI), `Solaris` (SUN), `BSD`, `MINIX`, ...;
- Most Unix systems are commercial;
- **Linux kernel** was created by Linus Torvalds in 1991 in `C` and it was greatly influenced by `MINIX`;
- "*Linux is a Unix clone written from scratch* ..."
- Linux has a number of distributions: `Red Hat`, `CentOS`, `Debian`, `SUSE`, `Ubuntu`, ...;
- Almost 97% of HPC clusters (`TOP 500`) are based on **Linux**;
- Linux is **free**[**doom**] (nothing to do with price)!
- Richard Stallman's **GNU** Project;

Basic commands for files/directories

(`ls`, `pwd`, `cd`, `cp`, `mv`, `rm`, `mkdir`, and `rmdir`)

Basic commands for files/directories

- How to login to a cluster (without X11-Forwarding):
- `$ ssh username@qb.loni.org`
- `$ ssh -l username qb.loni.org`

Command	Explanation
`ls`	list information for files and directories
`pwd`	show the work directory
`cd`	change to a directory
`cp`	copy files or directories
`mv`	rename files or directories
`rm`	remove files or directories
`mkdir`	make a new directory
`rmdir`	remove a directory

The `ls` command

- The `ls` command is used to list files and directories;

Option	Explanation
`-a`	list all files including hidden files (beginning with a dot)
`-F`	mark directories, exes, and links separately
`-r`	sort the list in reverse order
`-l`	long format with 7 attributes
`-d`	list only directories
`-t`	sort the list by last modification time

- Without any options, the default list is ordered in sequence of numbers, uppercase and lowercase;
- Our alias for `ls` is `ls -color=auto` to mark files, dirs and exes in different colors;

`cd`, `pwd`, and relative pathnames

- The `cd` command: **c**hange **d**irectroy;
- The `pwd` command: **p**rint **w**ork **d**irectroy;
- **Relative** pathnames:
  - A single dot (`.`) is for the **current** directory;
  - A double dot (`..`) is for the **parent** directory;
  - All pathnames that begin with the above symbols are called **relative** pathnames;

```
$ pwd
/home/xiaoxu/dir1/dir2/source_code
$ cd ..  ; pwd
/home/xiaoxu/dir1/dir2
$ cd source_code
$ cd ../../ ; pwd
/home/xiaoxu/dir1
```

Rename and remove commands

- The `mv` command: **m**o**v**e file and directories (rename);

```
$ mv my.dat ../.
$ mv my.dat my.dat.bak
$ mv mydir mydir.orgi
```

- The `rm` command: re**m**v files and directories (**unlink** filename and data);

```
$ rm my.dat
$ rm -i my.dat
$ rm ../source_code/parameters.h
$ rm -r mydir
$ rm -rf mydir
```

- `rm` supports many flags: `-i` (prompt before removing), `-f` (never prompt before removing);
- The above two commands work on both files and directories;

Making and removing directories

- The `mkdir` (**mk**e **dir**ectory) command creates new directories;

```
$ mkdir test
$ mkdir bin examples doc
```

- The `rmdir` (**rm**ove **dir**ectory) command removes existing directories;
- Directories can also be removed with the `rm` command;

```
$ rmdir doc
$ rmdir doc bin
```

- Some common errors:

```
$ mkdir: cannot create directory ‘mydir’: File exists
$ rmdir: failed to remove ‘emtest’: Dir. not empty
```

Basic commands for text processing

(`head`, `tail`, `cat`, `less`, and `more`)

Basic commands for text processing

- Basic commands for **text processing**:

Command	Explanation
`head`	display the beginning part of text file
`tail`	display the bottom part of text file
`cat`	display, combine, or create text files
`tac`	inverse of `cat`
`less`	view text files
`more`	view text files

The `head` and `tail` commands

- The `head` command shows the **beginning** part of a text file;

```
$ head my.dat       # default is top 10 lines.
$ head -20 my.dat   # show top 20 lines.
$ head -n 20 my.dat # show top 20 lines.
$ head 20 my.dat    # what happens to this one?
```

- The `tail` command shows the **bottom** part of a text file;

```
$ tail my.dat       # default is last 10 lines.
$ tail -20 my.dat   # show last 20 lines.
$ tail -n 20 my.dat # same last 20 lines.
```

- What happens if I don't how many lines to `head` or `tail`?

```
$ head -n -2 my.dat # all lines except last 2 lines -n -.
```

```
# all lines except the top 3 lines -n +.
$ tail -n +4 my.dat
```

The `cat` and `tac` commands

- The `cat` and `tac` commands can **view**, **create**, **copy**, and **combine** text files;

```
$ cat my.dat
$ cat my.dat > mynew.dat  # redirect to a new file.
# combine them into a new file.
$ cat a.out aa.out b.out > allout.dat
# combine them into a new file.
$ cat my[0-9].dat > mynew.dat
$ cat > my.dat  # create an empty file.
```

- What is `tac`?

```
$ tac my.dat  # inverse of cat.
```

The `less` command

- **View** text files without directly opening it (page based):

```
$ less [options] filename
# start with 10th line from the beginning:
$ less +10 myroutine.f90
```

- Page navigation: `Spacebar` or `f` to next page (`b` for backward);
- Forward search mode: `/pattern` (`n` for next match, `N` for previous match). Backward search mode: `?pattern` (`n` for next match, `N` for previous match);
- Jump to the **start** of file: `g`, Jump to the **end** of file: `G`
- File information: `=`
- Quit: `q`
- Useful to `pipe` output to the `less` command;

Linux globbing wildcards

(`?`, `*`, `[]`, and `[!]`)

Globbing wildcards

- **Two** types of wildcards:
  (1) **globbing** wildcards (2) **regular expressions** (regex)
- **Globbing** wildcards are used by commands on **multiple** `files names`;
- **Regex** handles `content` of text files (searching, parsing, or manipulation);
- **Six** globbing wildcards: `?`, `*`, `[]`, `[!]`, `{}`, and `\`
  - `?` (question mark): any single character;
  - `*` (asterisk): any number (including zero) of characters (except a leading dot in hidden file names);
  - `[]` (square brackets): one character in a list or range;
  - `[!]`: negate of `[]`;
  - ...

Globbing wildcards

```
$ ls ?.dat
$ ls -ltr abc??
```

```
$ ls *
$ ls *.????
$ rm out*.dat
```

```
$ cat o[a,t,i]d
$ ls -ltr o[a-c]d
$ ls -ltr *[a-c]*
$ rm -f a*[0-9][0-9][0-9]
```

```
$ mv ../[!a-d]*.out .
$ ls -ltr a*[!0-9][!0-9][!0-9]
```

The process basics

(`ps`, `kill`, `bg`, `fg`, and `jobs`)

The process basics

- A **program** is an executable file held in storage; It can be a binary file or a type of script file;
- A **process** is an **executing** or running instance of a program (also known as task); It has a lifetime;
- A process can create other **new** processes (child processes);
- A process has many **attributes** including security attributes and state (`pid`, `ppid`, `uname`, `rss`, `etime`, `tty`, `cmd`, etc);
- A process can run in the background or in the foreground;
- Linux kernel handles processes including scheduling, maintaining state and security attributes, allocating memory space, and other resources;

Basic commands for process management

Command	Explanation
`ps`	provide info for current processes
`jobs`	display current status of jobs
`bg`	resume stopped jobs and run them in background
`fg`	resume stopped jobs and rum them in foreground
`kill`	kill processes by sending signals

The ps command

- The ps (**p**rocess **s**tatus) command provides a lot of info about current processes;
- A process can have many attributes pid, ppid, uname, rss, etime, tty, cmd, etc;

```
$ ps [options]
```

- ps supports both GNU syntax (with -), BSD syntax (w/o -), and long option syntax (with --);

```
$ ps ax  # list all processes (a).
$ ps -e  # list every process (-e).
$ ps aux  # list all process with details (username u).
$ ps -ef  # list every process with details (-f).
```

The `ps` command

```
# list all processes of a user (-u) with details.
$ ps -u xiaoxu -f
$ ps -C bash  # search the process by command name.
# show the details for a given process (-p).
$ ps -f -p pid  # full info for a given process.
$ ps -o uname,pid,cmd  # specify output entries.
```

- Process **attributes**:
  - `uname`: username of the process owner;
  - `pid`: process id;
  - `ppid`: pid of parent process;
  - `rss`: resident memory allocated;
  - `cmd`: command path and name;
  - `etime`: elapsed time;

The `ps` and `kill` command

```
# list all processes of a user (-u) with details.
$ ps -u xiaoxu -f
$ ps -C bash  # search the process by command name.
# show the details for a given process (-p).
$ ps -f -p pid  # full info for a given process.
$ ps -o uname,pid,cmd  # specify output entries
```

- How to **terminate** processes:

```
$ kill pid  # send SIGTERM (default) to the process;
$ kill -9 pid  # send SIGKILL to the process;
```

- Depending on the situation, the process or kernel may decide to ignore or block signals;

Linux command-line options

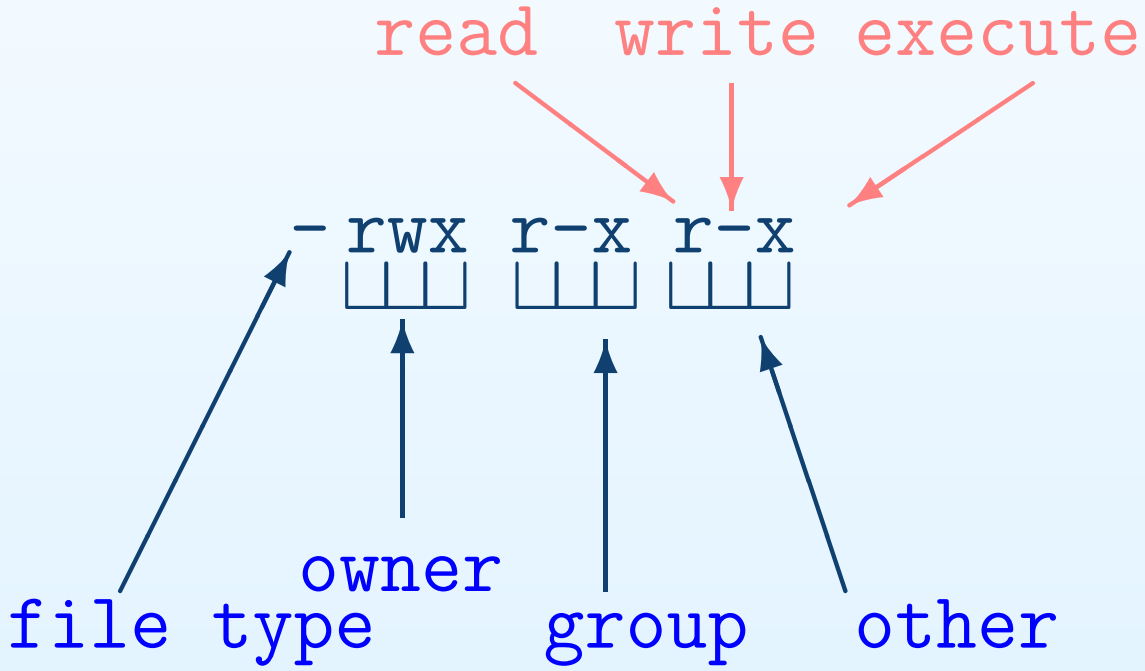
- We have many command-line options;
- Linux command-line options: `Unix` style (single dash) and `GNU` standard (double dash);

Command	Explanation
`-h --help`	print help and usage message
`-v --version`	show version info
`-a --all`	operate on all arguments or append
`-l`	long format or login
`-o`	output
`-f`	file, force, or full
`-e`	execute, exclude, or end
`-q`	quiet or queue

File permission and management

File permission

- Linux files and directories have **three** permission groups: `owner`, `group`, and `other`;
- **Three** basic permission types: `read`, `write`, and `execute`;
- Security controls what actions are granted for a user;



File permission

```
$ ls -ltr test.dat
  -rw-r-r- 1 xiaoxu Admins 26 Oct 4 2016 test.dat
```

- The 1st entry is file type: `-` (ordinary file), `d` (directory), and `l` (symbolic link);
- The rest 9 entries divides into three groups: owner, group, and other's permissions;
- Each group has 3 permission bits: it is a `dash` if not set;
  - `r (read)`: `ls`, `cp`, `cat`, `less`, view its contents, ...;
  - `w (write)`: edit file, `rm`, `mv`, ...;
  - `x (execute)`: executable and run it;

Directory permission

- A directory also has 3 permission bits;
- They are **different** from file permission;

```
$ ls -ld test
  drwxr-xr-x 2 xiaoxu Admins 4096 Jul 18 2016 test
```

  - `r (read)`: `ls`, `cp`, read files name stored in the directory;
  - `w (write)`: work with `x` bit, make new files, rename files, remove files stored in the directory, if `x` bit is set;
  - `x (execute)`: enter. work with `w` bit and if `w` bit is set, rename or delete the directory;

- **Octal** notation (base-8) for file and directory permissions: Assign `4` to `r`, `2` to `w`, `1` to `x`. It is `0` if not set;
- Sum them over for each of three groups;

Octal notation for file and directory permission

`--x--x--x`	111	execute	`rwxr-----`	740
`-w--w--w-`	222	write	`rwxrwx---`	770
`-wx-wx-wx`	333	read	`rwxr-xr-x`	755
`r--r--r--`	444	read	`rwxrwxrwx`	777

- How to **change** permissions: `chmod` (**ch**ange **mod**e)
- `chmod` supports relative (+/−) and absolute (=) operations;

`u`	(User/Owner)
`g`	(Group)
`o`	(Others/World)
`a`	(All above, `ugo`)

`+`	Assign permission
`-`	Remove permission
`=`	Assign absolute permission

The `chmod` command

- **Relative** operation:

```
$ chmod u-r my.dat  # remove r bit for u.
$ chmod u+x myscript  # assign x bit for u.
$ chmod go+x myscript  # assign x bit for g and o.
$ chmod a+r my.dat  # assign r bit for all users (ugo).
```

- **Absolute** operation:
  - Absolute operation only assigns specified permission and remove others;

```
# only set r for u and unset all permission for all others (go).
$ chmod u=r my.dat
$ chmod go=x myscript # set x bit only to g and o.
$ chmod a=x myscript  # set x bit to u, g, and o.
```

Create symbolic (soft) links

- A **symbolic** (or **soft**) link is a special file that contains a reference to another file or directory;
- A link can point to a **file** or **directory**;

```
$ ln -s /path/to/source_file /path/to/target_link
```

```
# my.dat is a link to my2018.dat
$ ln -s my2018.dat my.dat
$ ls -ltr my*.dat
-rw-r--r-- 1 ... my2018.dat
lrwxrwxrwx 1 ... my.dat -> my2018.dat
```

```
$ ln -s /home/xiaoxu/patha/pathb/project   \
        /zone/subzonea subzonea
$ cd subzonea  # cd to the above directory.
```

Locating files: the `find` command

- The `find` command searches directories **recursively** to match criteria;

```
$ find path_where_to_search what_to_search
$ find /home/xiaoxu -name *.dat
$ find . -name *.txt # search current directory.
$ find / -name lib*.so # Don't run it on our clusters!!
# search files that were modified less than 5 days ago.
$ find . -mtime -5 *.dat
# search files that were accessed more than 30 days.
$ find . -atime +30 [a-c]*.dat
# search files that were modified more than 3 days but less
 than 10 days ago.
$ find . -type f -mtime +3 -mtime -10
```

Basic `vi`/`vim` editing

The `vi`/`vim` command

- The command-line editor `vi`/`vim` is the default editor on most Linux systems; It is also available on Windows and Mac OS - **universal** availability;
- `vim` stands for **vi im**proved;
- The command `vi` is an **alias** of `vim` on most Linux systems;
- Cryptic commands to most beginners;
- In addition to editing multiple files, `vi` can move or copy a file to other files;
- `vi` operates in three modes: **Input** mode, **Command** mode, and **Ex** mode;
- `ESC` key plays an essential role;

The `vi`/`vim` command

- **Command** mode: The default mode. Everything you typed are interpreted as `vi` commands;
- **Input** mode: It allows us to insert something in text. `i`, `a`, `A`, and `o` are four common commands to enter the **Input** mode; Pressing `ESC` key to quit **Input** mode and back to **Command** mode;
  - `i` Insert text at the current position of cursor.
  - `a` Insert text to right of current position of cursor.
  - `A` Insert text at the end of line.
  - `o` Start a new line below and insert text.
- **Save** and/or **quit**: In **Command** mode
  - `:w` Save and continue editing.
  - `:wq` Save and quit editing (same as `:x`).
  - `:q!` Quit without saving.

The vi/vim command

- **Delete** characters or lines:

`x`	Delete character at current position of cursor
`5x`	Delete 5 characters from current position of cursor
`dd`	Delete current line
`3dd`	Delete 3 lines from current line and below
`dG`	Delete all lines from current line to end of file
`d1G`	Delete all lines from current line to beginning of file
`d^`	Delete from current cursor to beginning of line
`dw`	Delete current word
`d4w`	Delete current and next 4 words

The `vi`/`vim` command

- **Yank** (copy) and **paste** lines:

`yy`	yank current line
`3yy`	yank 3 lines below
`p`	paste below cursor
`P`	paste above cursor
`J`	combine current and next lines

- **Undo** changes: `u` to undo the last change;

The vi/vim command

- Cursor **movement** commands:

`0`	move to beginning of current line
`$`	move to end of current line
`4w`	move to next 4 words on the same line
`gg`	move to beginning of file
`:10`	jump to `10th` line
`10G`	jump to `10th` line
`G`	move to end of file
`Ctrl-f`	page down
`Ctrl-b`	page up

Summary

We have covered basic Linux commands for files/directories, text processing, globbing wildcards, file permissions, basic processes, and vi/vim editing.

The best way to master **Linux**

is to run Linux commands yourself!

Questions?

sys-help@loni.org