PHY 607 - Computational Physics

Linux Command Line

August 28, 2025

The Linux Command Line

- The Linux command line interface is provided through the "shell", which interprets and executes commands entered in a terminal emulator
- Several shells have been developed and used over the years

  sh The original shell developed in 1977, also called the "Bourne shell"; still guaranteed to be present in all Unix/Linux systems

  bash "Bourne shell again", replacement for "sh"; default shell in current Linux distributions.

  zsh A derivation of bash which is the default on MaxOSX

We will use "bash" in this class

The Linux File System

- The Linux file system is made up of many nested directories.
  - The location of each file is given by a "path"
- Paths can be "absolute" or "relative"
- Absolute paths start with "/"
- Example, "/usr/bin", the directory where most system tools reside
- Relative paths are relative to the "current working directory"
  - The command "pwd" (print working directory) will display the full absolute path of the current working directory

The Linux File System

- Paths relative to the current working directory
  - One dot "." denotes the current working directory
  - Two dots ".." denotes the parent of the current working directory

If current working directory is "/usr/local/bin, then "./lyx" designates the same file as "/usr/local/bin/lyx".

If current working directory is "/usr/local/lib", then "../bin/lyx" designates the same file as "/usr/local/bin/lyx".

- The "home" directory
  - Tilde "~" denotes the user's home directory

If the user's home directory is "/home/jsmith", then "~/Documents/syllabus.pdf" designates the same file as "/home/jsmith/Documents/syllabus.pdf".

The Linux File System

Navigating the file system

- "pwd" displays the current working directory
- "cd" changes the current working directory
  - Without an argument it changes to the default "home" directory
  - With an argument it changes to the specified directory
- "ls" lists the files in the current working directory

Example

```
[phy607@ahn-x200 ~]$ pwd
/home/phy607
[phy607@ahn-x200 ~]$ ls
bin Documents Music Pictures Templates
Desktop Downloads phy607 Public Videos
[phy607@ahn-x200 ~]$ cd phy607
[phy607@ahn-x200 phy607]$ ls
lecture1 lecture3 resources syllabus.lyx
lecture2 lecture4
```

Example

```
[phy607@ahn-x200 phy607]$ cd ../bin
[phy607@ahn-x200 bin]$ pwd
/home/phy607/bin
[phy607@ahn-x200 bin]$ cd ~/phy607
[phy607@ahn-x200 phy607]$ pwd
/home/phy607/phy607
```

Examining Files

Text files, that is

tail *file_name* displays last few lines of a file. Good for looking at log files

head *file_name* displays first few lines of a file. Good for quick check of file contents

more *file_name* allows paging through a file

less *file_name* allows scrolling through a file in both directions

Spaces are not allowed in Linux file names!

"file_name" can also be a complete path.

Creating Files

cat *>file_name* Copies from terminal to file. Finish with ^d.

touch *file_name* Changes date/time of file to current; creates if doesn't exist.

nano *file_name* Simple text editor; creates file if doesn't exist.

vim *file_name* More powerful text editor; creates file if doesn't exist.

emacs *file_name* Extremely powerful and extensible text editor; creates file if doesn't exist.

Moving, Copying, and Deleting Files

cp *file1 file2*	Copy file1 to file2.
cp *file1 dir/*	Copy file1 to file1 in directory "dir". The "/" is not necessary if directory already exists (but is recommended to avoid unintended consequences if the directory does not exist).
cp *file* dir/*	Copy all files matching pattern "file*" to directory "dir".
mv *file1 file2*	Move file1 to file2.
mv *file1 dir/*	Move file1 to file1 in directory "dir".
rm *file*	Remove file. Be careful, there is no undoing this (there is no trash can or recycle bin!)
rm *-rf **	Remove directory, all files in it, and all its subdirectories. You probably really don't want to do this.
rmdir *dir*	Remove directory "dir". By default this will not remove a non-empty directory.

Wildcards

The command line will accept "wildcards", which makes operations on multiple files much easier. Wildcards are a special case of the more general "regular expressions", which we will cover in more detail later.

The wildcard "*" means match zero or more of any character.

`ls *` List all standard files and the contents of all standard directories in the current working directory.

`ls .*` List all "hidden" files in the current working directory, and all non-hidden files in hidden directories. By default, files whose names start with a "." are hidden and don't show up in directory listings.

`mv foo*.dat newdir/` Move all files whose names start with "foo" and end with ".dat" to the directory "newdir". If "newdir" does not exist and error message will be printed.

Switches

Most Linux commands accept a number of switches which modify the behavior of the command. We've already seen one example for the "rm" command.

- For some commands the switches are simple letters or keywords appropriately placed.
- For others they are letters or keywords following a single dash"-"
- And for yet others they are keywords following a double dash "–". Newer commands use this format, and some commands also keep the older format for compatibility.

How can you know? Check the documentation!

Documentation

Where does one find Linux Documentation?

1. Check the “man” pages. Most Linux commands will have a “man” page which can be accessed via “man”.
   - Example: “man ls”
2. Try the “info” command. Many Linux commands will have an “info” section, which provides more detailed information than “man”. The info can be accessed via “info”.
   - Example: “info ls”
3. Try inline help. Many commands will display a short help when the switch “–help” is supplied.
   - Example: “ls –help”.
4. Do a “Google” search.

Redirection and Pipes

Combining simple utilities to make more complex operations

- Many programs send their normal output to "standard output (stdout)" and error output to "standard error (stderr)" by default.
  - For example, this is the case for ls, head, and tail.
  - Some programs also accept input from "standard input (stdin)".
- These can be redirected through use of the following symbols:

`> file_name` Redirects stdout to a file named file_name. If the file does not exist, it is created. If it exists, it is overwritten.

`>> file_name` Redirects stdout to a file in append mode. If the file does not exist, it is created. If it exists, the new information is appended to the file.

`|` Pipes the stdout of one program to the stdin of another program.

`tee file_name` Read from stdin and write to stdout and file. Useful with pipes.

Redirection and Pipes

```
ls -l /usr/bin |tee ls-output.txt | less
```

```
sed -e "s/\‘define/\#define/" <sde_trigger_options.vh | \
sed -e "s/\‘ifndef/\#ifndef/" | \
sed -e "s/\‘endif/\#endif/" | \
sed -e "s/’h/0x/" \
> ../drivers/sde_trigger_v1_2/src/sde_trigger_options.h
```

```
ls -l /usr/bin >ls-output.txt 2>&1
```

```
ls -l /usr/bin >/dev/null 2>ls-errors.txt
```

File Permissions

```
[ahnitz@ahn-nuc hdl]$ ls -l /bin | tail --lines=4
-rwxr-xr-x  1 root root  770248 Apr  5  2012 vi
lrwxrwxrwx. 1 root root       2 Aug 28  2014 view -> vi
lrwxrwxrwx. 1 root root       8 Aug 28  2014 ypdomainname -> hostname
-rwxr-xr-x  1 root root      62 Mar 17  2014 zcat
```

The fields displayed are:

1. Permissions (11 characters)
2. # of hard links (usually not important – use soft links instead)
3. Owner of the file
4. Owner group of the file
5. Size of the file
6. Date of last update
7. Name of the file

File Permissions

```
[ahnitz@ahn-nuc hdl]$ ls -l /bin | tail --lines=4
-rwxr-xr-x  1 root root  770248 Apr  5  2012 vi
lrwxrwxrwx. 1 root root       2 Aug 28  2014 view -> vi
lrwxrwxrwx. 1 root root       8 Aug 28  2014 ypdomainname -> hostname
-rwxr-xr-x  1 root root      62 Mar 17  2014 zcat
```

The permissions are further broken down as

1st character: "-"=normal file, "d"=directory, "l"=symbolic link, + less seen
Characters 2-4: Owner permissions for read, write, and execute, respectively
Characters 5-7: Group permissions for read, write, and execute, respectively
Characters 8-10: World permissions for read, write, and execute, respectively
Character 11: Indicates alternate access method (eg. SELinux)

- In the 3 character permissions
  - `-` indicates permission is not granted
  - `r` means read permission is granted
  - `w` means write permission is granted
  - `x` means execute permission is granted

Letters & field positions are redundant; rwxr-xr-x = 0755 octal

File Permissions & Links

```
[ahnitz@ahn-hnuc ~]$ ls -l /etc | tail --lines=2
-rw-r--r--.  1 root          root        813 Oct 14  2014 yum.conf
drwxr-xr-x.  2 root          root       4096 Jan  8
2015 yum.repos.d

[ahnitz@fedora .git]$ ls -l config
-rw-r--r--. 1 ahnitz ahnitz 255 Aug 28 15:32 config
```

- Top "ls" shows example of difference between file permissions and directory permissions
- Bottom "ls" shows example of symbolic link to create a "shortcut" for a long path

ln -s *Target Link_name* Create a symbolic link in the current directory with name *Link_name* to file or directory *Target*

- Symbolic links can cross file systems, even link files between local machine disk and network file system disks
- Hard links are limited to local file system

Shell Scripts

- Linux provides a natural way to combine existing commands into customized commands using a "shell script".
- If a shell script is given the "x" permission, and is placed in the user's search path, then it can be invoked just like any other Linux shell command.
  - For example, "chmod 755 my_command"
- Aside: How does one include custom commands in your search path?
  - One way is to add the directory that contains them to your PATH variable in your .bash_profile (or .bashrc or .profile) file

Environment Variables

- ‘export‘ sets environment variables
- ‘source‘ executes a script and adds any environment variables as if run by hand
- Key variables
  - path: controls what commands are accessible

Metacharacters on the Command Line

- Metacharacters have a special meaning aside from their literal meaning
- Literal characters are those that are taken at face value
- We've already discussed the shell metacharacters *, |, >, <, and >>. Some other shell metacharacters are

?	Match any one character
[abc...] or [a-z,0-9]	Match any one of the enclosed characters
[!abc...]	Match any character not enclosed
\	Interpret the next character literally (eg. *)
‘cmd‘	Replace with output of cmd (eg. gcc ‘ls m*.c‘) using the backtick character
;	Execute a sequence of commands entered in one line (eg. cd other; ls)
||	Execute the second command only if the 1st command fails (eg. gcc myprog.c || echo compilation failed)
&&	Execute the second command only if the 1st command succeeds (eg. gcc myprog.c || echo compilation succeeded)

Regular Expressions

- Regular expressions are made up of metacharacters and literal characters
  - Can be used in arguments to some programs (find, grep, sed, ...)
  - Some metacharacters behave differently in regular expressions and the command line.

Alphanumeric characters (eg. A-Z, a-z, 0-9) match themselves

.	matches any single character
?	matches 0 or 1 of the preceding element
+	matches 1 or more of the preceding element
*	matches 0 or more of the preceding element
[ ]	matches any single character within the backets
[^]	matches any single character not within the brackets
^	matches the starting position within a string or line
$	matches the ending position within a string or line

Regular Expressions

a.?b	matches ab, abb, a1b, but not aaab.
a?b	matches ab, b, but not abb.
a.+b	matches abb but not ab.
[a-z]?1	matches 1, a1, z1, but not aa1.
.at	matches any three-character string ending with "at", including "hat", "cat", and "bat" (this & following examples are from Wikipedia).
[hc]at	matches "hat" and "cat".
[^b]at	matches all strings matched by .at except "bat".
[^hc]at	matches all strings matched by .at other than "hat" and "cat".
^[hc]at	matches "hat" and "cat", but only at the beginning of the string or line.
[hc]at$	matches "hat" and "cat", but only at the end of the string or line.
\[.\]	matches any single character surrounded by "[" and "]" since the brackets are escaped, for example: "[a]" and "[b]".
s.*	matches any positive number of characters preceded by s, for example: "saw" and "seed".

Some Other Important Commands

chmod *mode file_name*	Change permissions of *file_name*
diff *file1 file2*	Compare *file1* and *file2*
echo *string*	Echo a string to the terminal; useful in shell scripts
file *file_name*	Display the type of a file
find	Search for files in the file system
git	Version control system
grep *regexp file_name*	Search file(s) for regular expressions
lpr *file_name*	Print file with minimal formatting
make	Build programs following a set of rules
rsync	Synchronize files between directories or computers
scp	Copy files to or from a remote computer
sed	Stream editor; useful in shell scripts
sort *file_name*	Sort a file
ssh *user@computer*	Open a terminal session on a remote computer
tar	Insert or extract files in an archive
wc *file_name*	Display number of words, bytes, or lines in a file

Keeping Files Synchronized with "rsync"

- Many of you probably have access to multiple computers. For example, you may have
  - a desktop in your apartment
  - a laptop that you travel to campus with
  - file space on a Syracuse or remote computing cluster
- "rsync" is one option
  - Included by default in most Linux distributions

Example "rsync" scripts

Syncing files to server

```
#!/bin/sh
#
rsync -av ~/phy607/ ahnitz@example.syracuse.edu:ph607
```

Syncing files from server

```
#!/bin/sh
#
# rsync4390_from
rsync -av ahnitz@example.syracuse.edu:phy607/ ~/phy607
```