> Linux command-line for HPC

10/3/2024
presented by Jason Buechler

Slides > https://rcdata.nau.edu/hpcpub/workshops/linux.pdf

NORTHERN ARIZONA UNIVERSITY

Outline

- What is Linux? (3 slides)
- *Let's get started*: The command-line (3 slides)
- Intro to the shell (4 slides)
- Navigating the file-system (10 slides + 1 demo + 1 exercise)
- Managing files (3 slides + 1 demo + 1 exercise)
- Dealing with text (4 slides + 1 exercise)
- Dealing with processes (2 slides + 1 demo + 1 exercise)
- Advanced stuff (6 slides)

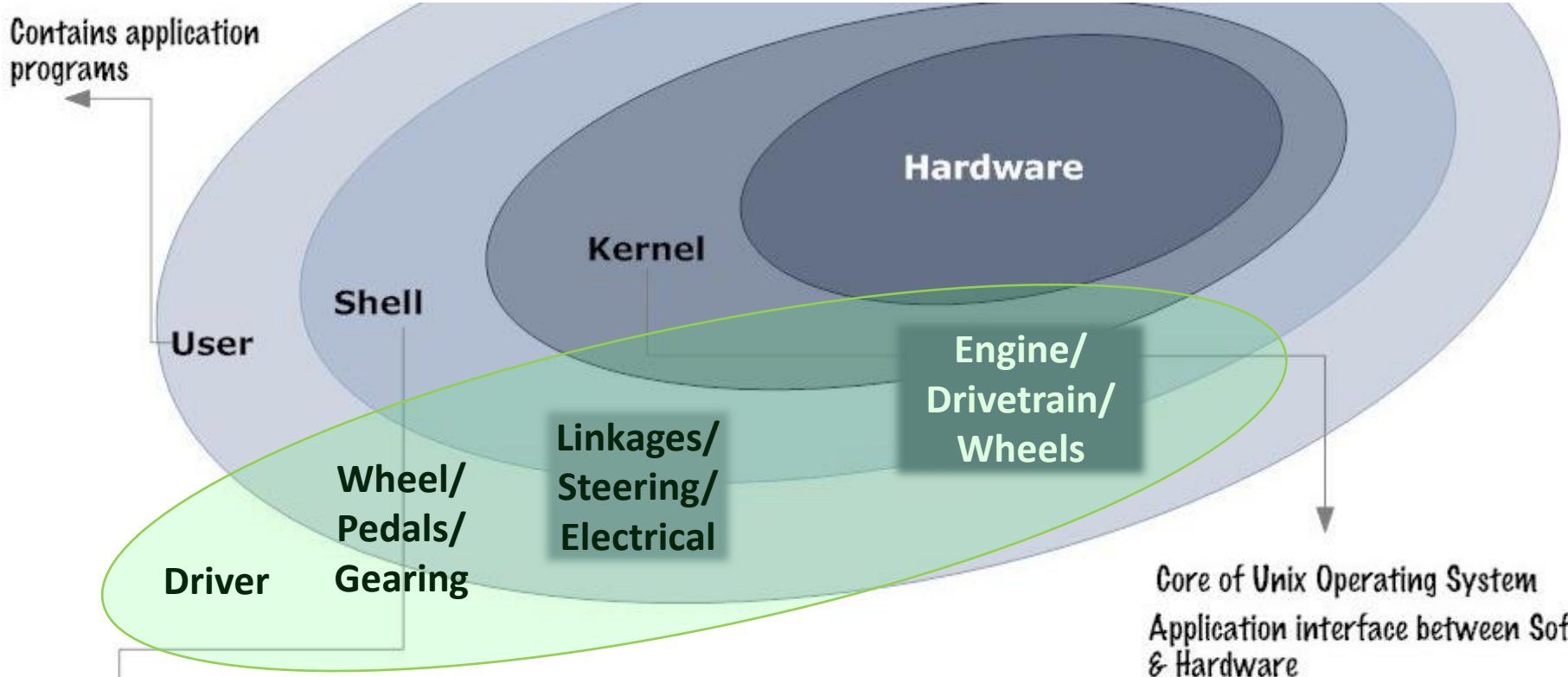
Introductions

- Introduce yourself
  - Name
  - Department / Group
  - Linux or Unix experience

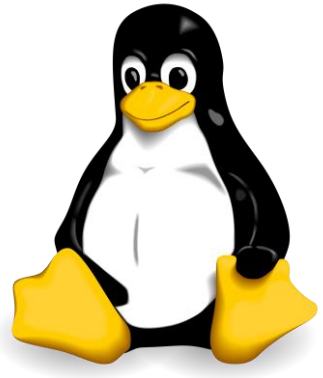
What is Linux?

- Linux is a computer operating system like Mac OS or Windows
  - Keep in mind: an operating system is (much) more than the user-interface
- It is an open-source operating system where the defining piece is the Linux **kernel** which was developed by Linus Torvalds in 1991
  - *Linus + UNIX = Linux*
- The Linux operating system is:
  - Linux kernel, and
  - Open-source, and
  - More open-source software

Unix architecture layers



Linux continued



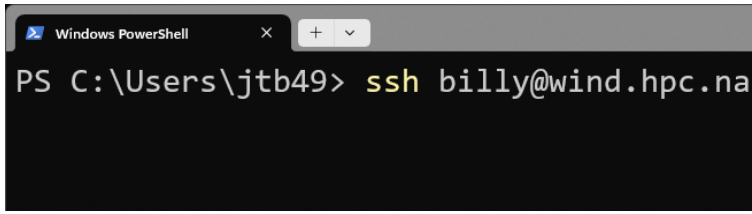
- Linux powers businesses, universities, the internet, and HPC clusters.
- Linux powers 100% of the top 500 HPC clusters in the world
- http://www.top500.org/statistics/details/osfam/1
- HPC is the future of computing
  - A hint of what will be on your desktop, or your hand, in 10 years
- HPC is built on linux, so futureproof your skills by learning linux skills early!

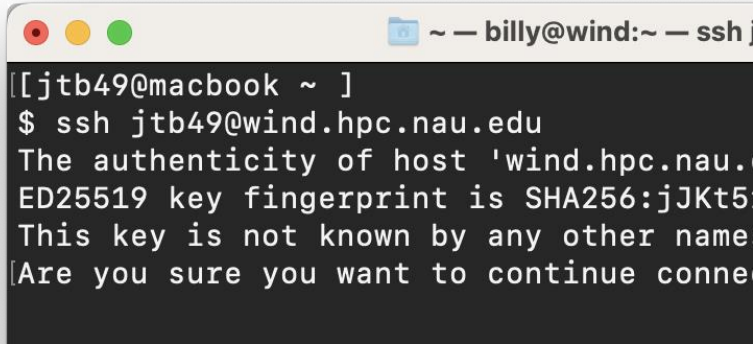
Let's get started: *The command-line*

- Most user applications use a GUI, but this workshop uses a text-based "shell" to interface with the operating system
  - `Shell commands` (i.e. what you type) will be colored like this
  - **<pointy brackets>** indicate values that will vary by person/choice
- Your computer's front-end interface "app"
  - Mac OS – Use **Terminal** (Applications > Utilities > Terminal)
  - Windows – Use **Powershell** (or Putty if you prefer)
  - More info: https://in.nau.edu/arc/overview/connecting-to-monsoon

Logging in

- You must first be on the NAU network, **or NAU VPN**
- Open Terminal (on Mac) or Powershell (on Windows)
- Use the `ssh` command to connect to Monsoon
  - `ssh <NAUID>@monsoon.hpc.nau.edu`
  - Classroom students: replace **`monsoon`** with **`rain`**
- You'll be prompted to accept a SSH key, type `Y`.
- Type your LOUIE password & hit Enter
- NOTE: no *'s or characters are printed!
  (no visual feedback for passwords)



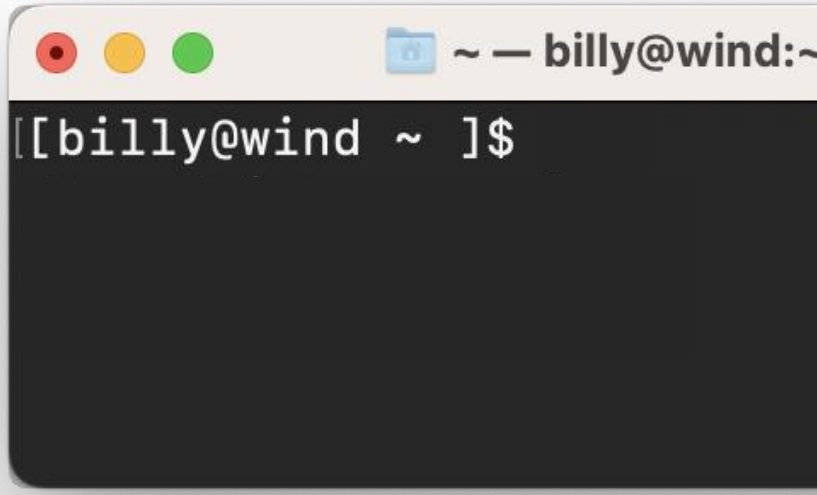


I’m logged in, now what?

- You should see a “prompt” like this:

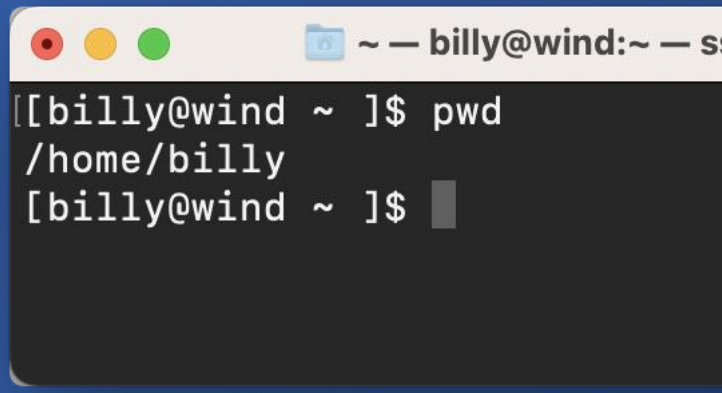
  `[<NAUID>@wind ~ ]$`

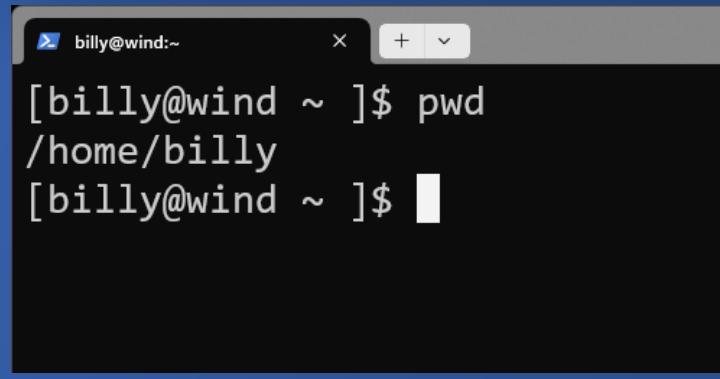
- You are now “on” one of Monsoon’s login nodes, in this case “`wind`”
  - or `rain`, for classroom students
- Note that the login nodes are not meant for heavy processing, they are solely for:
  - Editing / submitting job scripts
  - Moving data to/from monsoon
  - Trivial debug work (short tests of <=30 min)



Intro to the shell

- The shell is how you interact with Linux
  - **It's just a program** (analogous to Finder/explorer.exe)
  - A user types commands "into" the shell
  - The shell sends them to the kernel, where the work is done
  - Result: text is printed to the screen and/or to a file
- The Linux shell used in this course is called "bash"
- One can see what shell they're in/using
  - `echo $SHELL`





Interacting with the shell

There is no mouse, so we must use the keyboard keys!

- Arrow keys
  - left/right: moves cursor across text when entering commands
  - up/down: iterate through previous shell commands
- TAB to *complete* typing (one) matching filename, directory, or cmd
- TAB-TAB to show multiple matching expansions
- control-c to interrupt any program

Try some commands out

- `pwd` - print working directory (where you currently sit)
- `id` - this is your user id, and the groups you belong to
- `ls` - list files in the current directory, (try “ls -l” too)
- `w` - who’s logged in today, system load, and uptime
- `getquotas` - report your quota
- `date` - get the current date and time
- `echo` - print a message, e.g. `echo “hello world”`

- Try a few (now!), then use the up arrow to check out your history!

About options (flags)

- Most linux commands accept extra options or “flags”
  - `ls -a` (list all files, including hidden ones)
  - `ls -l` (list files, as a table with lots of details)
  - `ls -t` (list files, and sort by modification time)
- Combine options one after another
  - Order *usually* does not matter
  - `ls -a -l` == `ls -l -a` (list all files, with details)
- Flags are usually specified after a dash “-”, or double-dash “--”
  - Single-dash flags can often be combined
  - `ls -alt` (list all files, long listing, sort by time)

Navigating the file-system

In Windows and Mac OS, the GUI has informative windows

- displays what folder you're looking in
- offers controls to change how the contents are displayed
- double-clicking a folder views that different folder

To navigate the system *via our shell*, we will use:

- `pwd` – print working directory (compare with the prompt!)
- `ls` – list files in current/a directory
- `cd` – change directory

Demonstration: Navigating the filesystem

To navigate the system we will use:

- `pwd` – print working directory (compare with the prompt!)
- `ls` – list files
  - `ls` , `ls ..` , `ls ~` , `ls <dir>` , `ls <dir/subdir>`
- `cd` – change directory
  - `cd` , `cd ..` , `cd ~` , `cd <dir>` , `cd <dir/subdir>`

List files using extra options

```
[nauid@wind ~/linux_workshop ]$ ls
hello.txt
[nauid@wind ~/linux_workshop ]$ ls -l
total 4
-rw-r--r-- 1 nauid cluster 13 Sep 17 13:55 hello.txt
[nauid@wind ~/linux_workshop ]$ ls -la
total 16
drwxr-xr-x  2 nauid cluster   31 Sep 17 13:55 .
drwx------ 41 nauid cluster 8192 Sep 17 13:56 ..
-rw-r--r--  1 nauid cluster   13 Sep 17 13:55 hello.txt
```

Listing files: dot-files

Note the funny looking filenames there:

```
[NAUID@wind ~ ]$ ls -la
drwx------   2 NAUID cluster        4096 Sep 22 13:56 .
drwxr-xr-x 234 root  root           8192 Sep 16 13:06 ..
-rw-r--r--   1 NAUID cluster         213 Jul 11  2014 .bashrc
```

File names that begin with a "." are (usually) hidden.

- "." is the current directory
- ".." is the parent directory
- .bashrc is a hidden bash configuration file

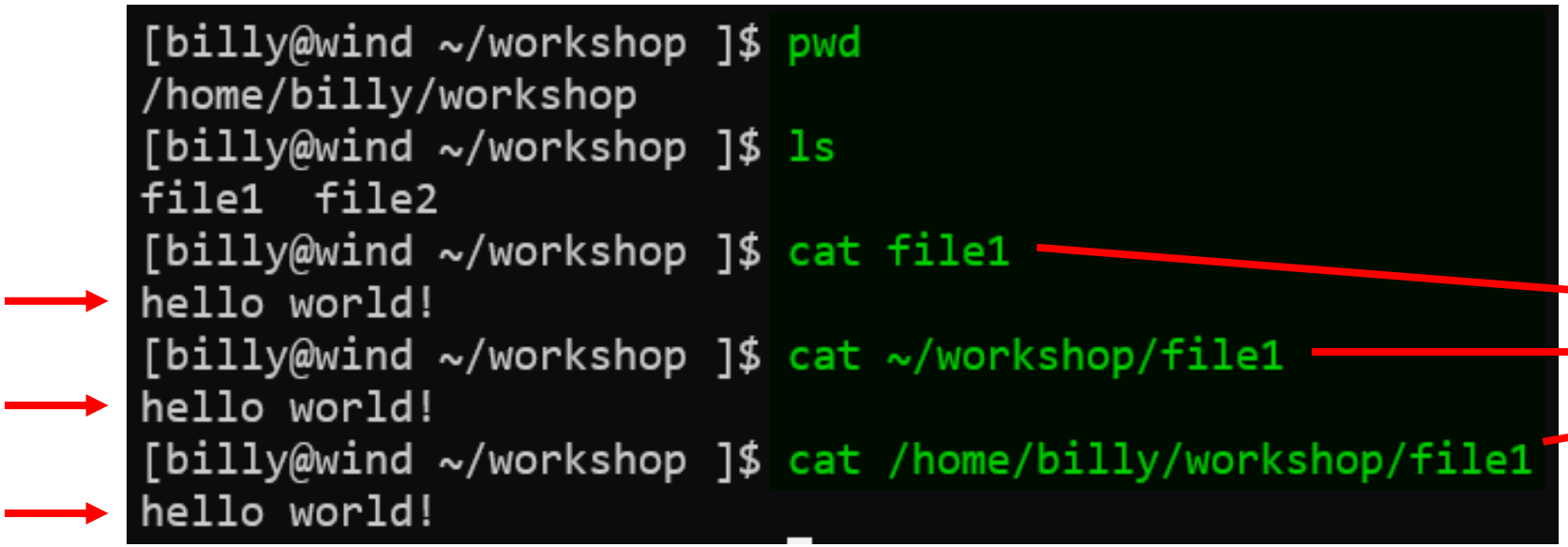
Listing files: relative hierarchy

```
$ tree -F /home/billy/jobs
/home/billy/jobs
├── 2023/
│   ├── q1/
│   ├── q2/
│   │   └── jobs_april.txt
│   ├── q3/
│   └── q4/
├── 2024/
│   ├── q1/
│   ├── q2/
│   └── q3/
│       └── jobs_july.txt
├── file1
└── file2
```

```
[billy@rain ~ ]$ cd ~/jobs
[billy@rain ~/jobs ]$ ls
2023  2024  file1  file2
[billy@rain ~/jobs ]$ ls 2023
q1  q2  q3  q4
[billy@rain ~/jobs ]$ ls ./2024
q1  q2  q3
[billy@rain ~/jobs ]$ cd 2024
[billy@rain ~/jobs/2024 ]$ ls ../
2023  2024  file1  file2
[billy@rain ~/jobs/2024 ]$ ls ../2023/q2
jobs_april.txt
```

Absolute vs relative paths

- File/dir locations can be absolute or relative
- Absolute paths start with "/" ("~" = "/home/nauid")
- Relative paths are just filenames, or start with "." or ".." or a directory



Same result now…

…buuuuut after cd'ing elsewhere?

** Note that Linux filesystems are CASE SENSITIVE with regard to almost everything!!*

** Also: not recommended to have spaces in filenames and directory names! It can be a pain.*

More commands

- `cat <file>` - print contents of a file to the screen
- `file <file>` - print the type of a file: ascii, dir, symlink,…
- `mkdir <dirname>` - create a directory name ”dirname”
- `rmdir <dirname>` - remove a directory named “dirname”
- `rm <filename>` - remove a file
- `cd <dirname>` - “open” directory “dirname”
- `touch <file>` - create an empty file, or update modified timestamp
- `less <file>` - view a file with a useful interactive viewer
- `man <command>` - view the manual for a command (“q” to exit)

Lab 1 – directory structure

1. Print your working directory, where you are currently (`pwd`)
2. List the contents of the directory you are in (`ls`)
3. Create a directory in your home directory named “linux” (`mkdir`)
4. Change directory to the new directory “linux” (`cd`)
5. Create a directory named “is” inside of the linux directory (`mkdir`)
6. Change directory to the “is” directory (`cd`)
7. Create a file in the ”is” directory named “awesome” (`touch`)
8. Change directory back to your home (`cd`)
9. Do a recursive listing on the “linux” directory: `ls -lR linux`
10. Try this and note changes: `ls -lR linux | grep -v total`

Lab 1 - Solution

```
[NAUID@wind ~ ]$ ls -lR linux
linux:
total 0
drwxr-xr-x 2 NAUID cluster 28 Sep 21 14:20 is

linux/is:
total 0
-rw-r--r-- 1 NAUID cluster 0 Sep 21 14:20 awesome
```

Wildcards

- While in the shell, you can select files/directories based on wildcards
  - ? - matches any 1 character
  - * - matches 0, or 1, or more characters
- Note that this may not work within interactive programs
  - Programs like Matlab or R (etc…) have shells with their own rules

Wildcard Examples

- While in the shell, you can specify files/directories based on wildcards
  - Multiple wildcards can be specified at once
- `ls *.txt` - lists all files/folders that end in “.txt”
- `ls lin*` - lists all files/folders that start with “lin”
- `ls *2024*` - lists all files/folders with “2024” in their name
- `ls 20?4-fall*` - list 2014-fall.pdf, 2014-fall.txt, 2024-sum.txt, etc

Demonstration: Bash basics & wildcards

- “cd” to the `/common/contrib/tutorials/linux` directory
- List all the filenames that end with “.pdf”
- List all the files that have the exact string “ADIOS” in their name

- List all the files in your home (~) directory *from here*
- “cd” to your home directory
- Show the sizes of all files in the first directory with ‘Tol’ in their name

Review: Navigating the file-system

To navigate the system we can use use commands like:

- `cd` move into/open a directory
- `pwd` print current directory (that you're in)
- `ls` print contents of a/current directory
- `rm` remove (delete) a file

To get more/varied output from your commands:

- Some commands accept/require "input" args (e.g. cd ***some_dir***)
- Most commands offer "options" (e.g. ls ***-a***)

File/dir locations can be absolute or relative

- Absolute paths start with "/"
- Relative paths are *just* filenames, or start with "." or ".." or "~" or a directory

Managing Files:

- Interpreting `ls -l` details
- Permissions and ownership
- Moving, copying, deleting files (and directories)

Managing Files: File permissions

```
drwxr-xr-x  2     NAUID   cluster  28     Sep 21 14:20   linux
1           2     3       4        5      6              7
```

1. The mode and type of the file, in this case a “d” (directory), mode 755
   - **From left to right: Type,User,Group,Other**
   - *Type* is directory (d) (could also be “-” (file), “l” (link), others)
   - *User* has read (r), write (w), and execute (x)
   - *Group* has read (r) and execute (x)
   - *Other* has read (r) and execute (x)
2. Number of hardlinks (you can kinda forget about this)
3 & 4. The owning-user and owning-group
5. Size of the file in bytes
6. The date, of last modified
7. The name of the file or directory

	($4=2^2$) read +	($2=2^1$) write +	($1=2^0$) exec =	bits
user	4 +	2 +	1 =	7
group	4 +		1 =	5
other	4 +		1 =	5
= “mode” 755				

Changing file permissions

- Default permissions for files and directories:
  - File: rw- for owning-**u**ser, r-- for **g**roup, and r-- for **o**thers
  - Directory: rwx for owning-**u**ser, r-x for **g**roup, and r-x for **o**thers
- Change owner/owner-group
  - ~~`chown billy some_file`~~
  - `chown :SICCS-Beekman-lab some_file`
- Change mode (permissions)
  - `chmod g+rw some_file` – add read and write for **g**roup
  - `chmod +x some_file` – add execute to a file, for **u**ser,**g**roup,**o**ther

Managing files: commands

- `cp file target` - make a copy ("target") OR copy INTO directory "target"
  - If "target" is an existing directory, "cp" assumes you want a same-name copy there
- `mv file target` - move "file" to directory "target" OR rename to "target"
  - If "target" is an existing directory, "mv" assumes you want to move "file" there
- `touch file` - create empty file, or update time stamp
- `rmdir dir` - remove (empty, only!) directory
- `mkdir dir` - make directory
- `rm file` - remove file
- `rm -f file` - force removal of file/directory (no verify prompt)
- `rm -r dir` - recursively remove a directory
- `rm –rf dir` - force remove recursively (CAUTION!!!!)

Demonstration: Operating on multiple files

- `cp * ` works like `ls *`
  - (Not) Including “hidden” dot-files (!!)
  - Multiple sources -> single target
- Recursive copy for directories
- Forcing deletions
- Deleting non-empty directories

Lab 2 – Editing/moving files

1. Change directory to the linux/is directory (`cd`)
2. Rename the file “awesome” to be “best” (`mv`)
3. Make a directory in the “is” directory named “the” (`mkdir`)
4. Move the “best” file to the “the” directory (`mv`)
5. Edit the “best” file, with contents “of course!” (`nano, emacs, vi`)
6. Copy the “best” file to the “is” directory, naming it “fun” (`cp`)
7. Find out what type of file ”fun” is (`file`)
8. Print the contents of the “fun” file (`cat`)
9. *BONUS* Make the file hidden

Lab 2 – Editing/moving files Solutions

```
[NAUID@wind ~ ]$ cd linux/is
[NAUID@wind ~/linux/is ]$ mv awesome best
[NAUID@wind ~/linux/is ]$ mkdir the
[NAUID@wind ~/linux/is ]$ mv best the
[NAUID@wind ~/linux/is ]$ cd the
[NAUID@wind ~/linux/is/the ]$ nano best
[NAUID@wind ~/linux/is/the ]$ cd ..
[NAUID@wind ~/linux/is ]$ cp best fun
[NAUID@wind ~/linux/is ]$ file fun
fun: ASCII text
[NAUID@wind ~/linux/is ]$ cat fun
of course
```

Dealing with text (and text-data)

It's all text! Everywhere!

- Text editors
- Pagers for viewing large files (most notably: "man" manuals)
- Not all screen text is equal
  - **Intended** command output =/= **error** output
- Redirecting command output
  - …into new files, or appending to existing
  - …directly into another command (no intermediate file!)
- Finding and isolating specific file-contents

Editors (vs text-pagers)

Lots of editors:

- nano
  - Simple to use
  - Onscreen “menu”
- vi, vim, emacs
  - more featureful
  - have learning curves

Start out using nano:

- ctrl-o: save (“O” as in write-Out)
- ctrl-x: exit (AND prompt to save)

Pagers (text-pagers):

- Fill a different role than editors
- How you read “manual pages”
- less
  - arrow keys navigate (PgUp/PgDn also)
  - h enter help screen
  - q exit
  - / start a search
    - n: next result
    - N: prev result

Redirecting Input and Output

- Default system streams
  - stdin/stdout/stderr = File Descriptors 0/1/2

- `>` Redirects output to another file, overwriting if it exists
- `>>` Appends to a file
- `2>&1` Redirects error messages to standard output
- `&>` Redirects stdout, and stderr to a file
- `|` (vertical bar) Redirects ("pipes") output from one program to another's input (*more on this later*)

Redirection Examples

- `ls > out.txt` - sends output from ls to “out.txt” file
- `ls >> out.txt` - appends output from ls to “out.txt”
- `ls foo 2> error.txt` - sends only errors to ”error.txt”
- `ls foo &> out.txt` - writes output and errors to out.txt
- `ls | wc -l` - send output from ls to the wc (wordcount) program and counts lines

```
[billy@radar ~/abc ]$ ls
hello.txt
[billy@radar ~/abc ]$ ls hello.txt foo
ls: cannot access 'foo': No such file or directory
hello.txt
[billy@radar ~/abc ]$ ls hello.txt foo 2>err.txt >out.txt
[billy@radar ~/abc ]$ grep ^ err.txt out.txt
err.txt:ls: cannot access 'foo': No such file or directory
out.txt:hello.txt
```

Lab 3 [guided] – Editing files

- `cd ~/linux`
- `nano grepfile.txt` (enter this text, then ctrl-x)

  Hello world!

  The world is a big place.
- Try: `grep place grepfile.txt`
- Try: `grep -v world grepfile.txt` ("-v" will invert results)
- Use grep recursively to find a term in any files nested within directories

```
[NAUID@wind ~/linux ]$ grep -r course *
is/the/best:of course!
is/fun:of course!
```

Lab 3 [guided] – continued

- Remember the | symbol (pipe)?
  - We can redirect the output of one command to the input of another
- Let's add a few lines to our grepfile.txt so it looks like this:

```
Hello world!
The world is a big place
test 1
test 2
testing 3
```

- We can grep for test, and pipe the output to grep for the character "2"

```
[NAUID@wind ~/linux ]$ grep test grepfile.txt
test 1
test 2
testing 3
[NAUID@wind ~/linux ]$ grep test grepfile.txt | grep 2
test 2
```

Dealing with processes

It's all text! Everywhere! …so how do I close/cancel something?

- `ps` and `top` list running processes
- `kill` ends a running process (of yours)
- Ctrl-c to "force quit" an active process (usually)

Processes

- `top` – Real-time view all running processes on this login-node
  - akin to task manager in windows
  - Hotkey “u” – show only one user’s processes
  - Hotkey “k” – kill a process (use ESC key to cancel)
  - Hotkey “q” immediately exits
- `ps` – Shows current processes
  - The “ps -u” option has a more useful format, including cpu %
- `kill <process id>` – Terminates a running process (if you are the owner of the process)

Demonstration: Processes

- Run the `top` program to view all processes currently running. Alternatively, you can run `ps` for a one-time snapshot, and `top -u <userid>`
- Look for your sleep process in the list. Specifically, look at the first column labelled “PID”. This means “process id”. Take note of your sleep process’s PID
- Press `q` to quit top and get back to the terminal
- Type `kill <PID>` where PID is your sleep process PID. This will end the sleep process

Lab 4 – Pipes and Processes

1. Start a new process by running sleep for 999 seconds (`sleep`)
2. Open another shell and `cd` to ~/linux again
3. Find the PID of your sleep process using `ps` and `grep`
4. Kill your sleep process (`kill`)
5. Verify your process is gone by running your previous `ps` and `grep` command
6. Do a long recursive listing of your linux directory, filter the results so only filenames with the word "best" are returned, and send the output to a file called results.txt (`ls, grep, >`)
7. List the contents of "results.txt" to verify the results (`cat`)
8. Remove the file "results.txt" (`rm`)

Lab 4 – Pipes and Processes Solutions

```
NAUID@wind:~/linux$ sleep 999
```

```
NAUID@wind:~/linux$ ps | grep sleep
  46396 pts/0    00:00:00 sleep
NAUID@wind:~/linux$ kill 49396
[1]+  Terminated: 15          sleep 999
NAUID@wind:~/linux$ ps | grep sleep
NAUID@wind:~/linux$ ls -alR | grep best > results.txt
NAUID@wind:~/linux$ cat results.txt
-rw-r--r--  1 NAUID  cluster    11B Oct 22 11:45 best
NAUID@wind:~/linux$ rm results.txt
```

Some more-advanced stuff!

- Variables
- Command substitution:
  - getting output into a variable
  - nesting one command within another!
- Loops
- (Soft-) Links

Variables

- Set your own variables: `MYVAR=1234`
- Un-set a variable: `unset MYVAR`
- Use existing variables by prepending them with `$`
- `echo "$MYVAR"` to display variable contents
  - Best practice is to use double-quotes with echo
  - But note that single-quotes prevent ***variable expansion***

Variables - Example

```
$ MYVAR="this is my variable"
$ echo $MYVAR
this is my variable
$ echo '$MYVAR'
$MYVAR
$ unset MYVAR
$ echo $MYVAR

$
```

Command substitution

```
$ which bash
/usr/bin/bash
$ x=`which bash`
$ y=$(which bash)
$ ls -l -h $x $y
-rwxr-xr-x 1 root root 1.1M Feb 10  2024 /usr/bin/bash
-rwxr-xr-x 1 root root 1.1M Feb 10  2024 /usr/bin/bash
$ ls -l -h $(which bash)
-rwxr-xr-x 1 root root 1.1M Feb 10  2024 /usr/bin/bash
```

Loops: simple one-liners

- One-line loop over “words”

```
$ for i in red blue green; do echo “$i is a color”; done
red is a color
blue is a color
green is a color
```

- One-line loop over consecutive numbers

```
$ for c in `seq 1 10 21`; do echo “count is $c”; done
count is 1
count is 11
count is 21
```

- Use a custom variable, e.g. `i` and reference it with the `$` sign, just like any other variable

Loops: multiline & nested

```
$ touch fileA fileB fileC
$ ls file*
fileA  fileB  fileC
$ for name in `ls file*`
> do
$name-copy
> done
$ ls file*
fileA-copy  fileB-copy  fileC-copy
```

```
$ for N in 2023 2024
> do
> for L in a b c
> do
> echo "$N$L"
> done
> done
2023a
2023b
2023c
2024a
2024b
2024c
```

Soft links

- We will focus on soft links for now; just know there are hard links too
- Soft links (AKA: symbolic links) are the most common type of links that you will use/encounter
  - `ln -s <existing_file> <symlink_name>`
- Basically same as ‘shortcuts’ in Windows, or ‘aliases’ in Mac OS

```
$ ln -s /scratch/NAUID /home/NAUID/scratch_link
$ cd ~
$ ls -l scr*
lrwxrwxrwx 1 NAUID cluster  4 Sep 22 10:15 scratch_link -> /scratch/NAUID
```

Questions?

- Lots more to Linux
- Try this book out for more:
  - http://linuxcommand.org/tlcl.php
- Refer to the advanced workshop!
  - link / calendar /etc