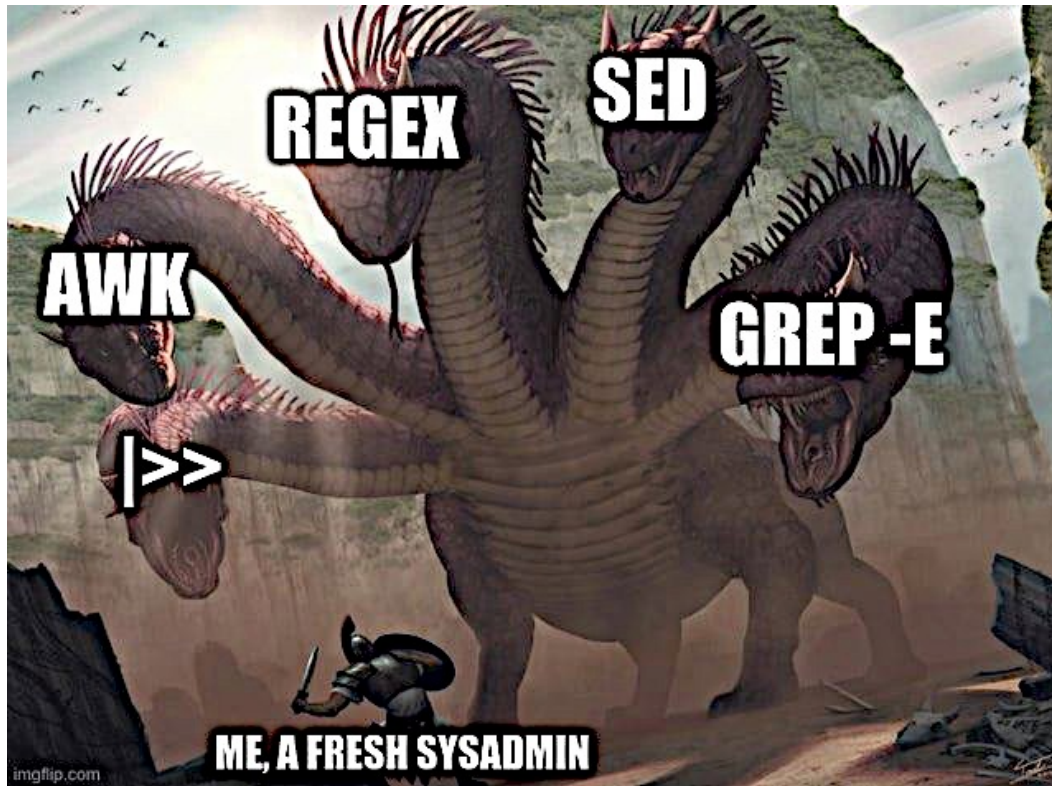


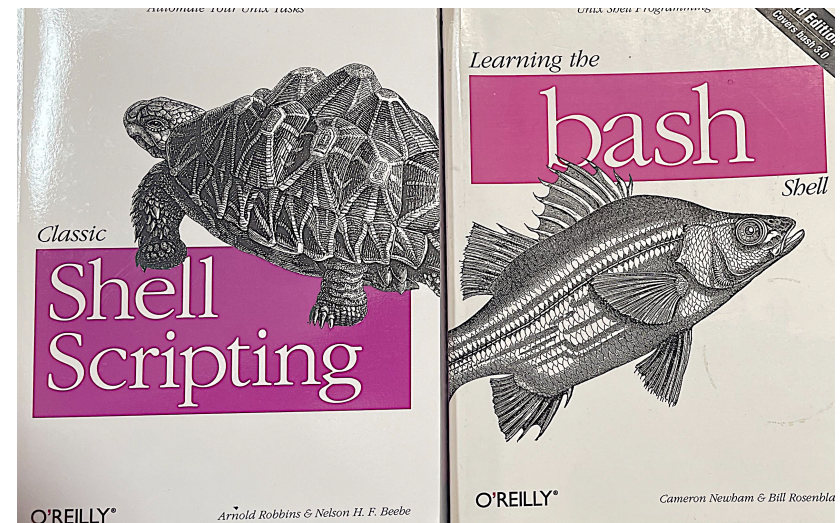
# Lecture 2: Bash Language Part II

Photo of a new  
Linux user learning  
the command line



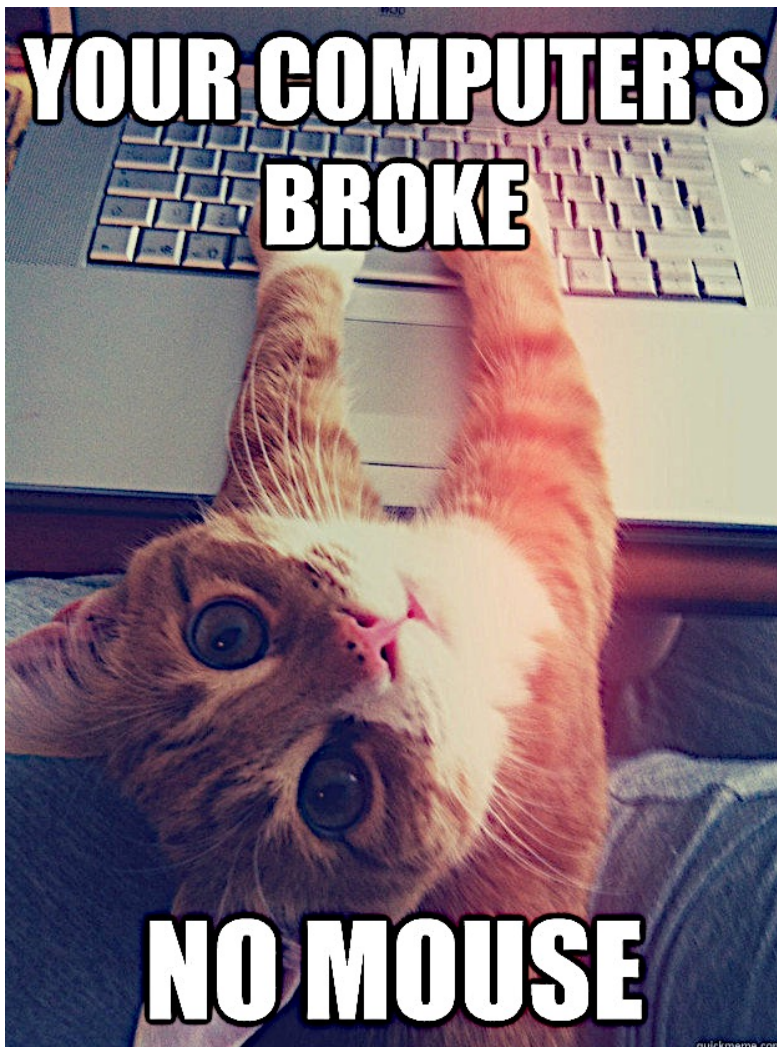
## Suggested Literature

Both books are available at  
<https://learning.oreilly.com>



# RECAP: GUI vs Terminal

You won't need to use `bash` all the time, but when you need it, understanding the following lectures can save you so much time.

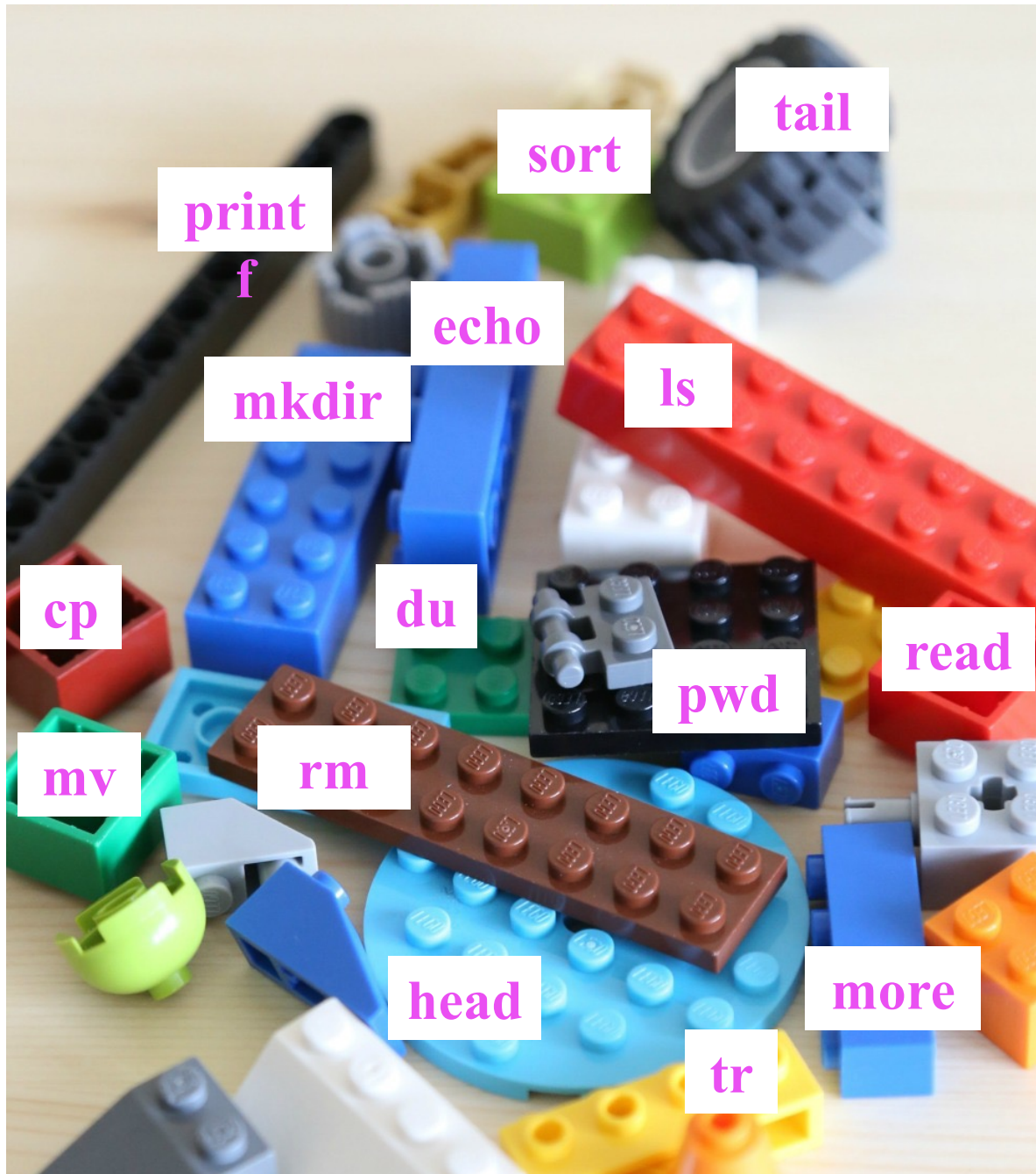


Most servers don't have VNC that allows remote GUI

Even with VNC, GUI requires a lot of bandwidth.

**We need to be comfortable with terminals**

# RECAP: bash scripts ~ playing Lego



We have many small pieces (programs).

From “gluing” programs, we build pipelines.



# RECAP: the location of shell programs

```
whovian@phy504:/bin$ type -a echo
echo is a shell builtin
echo is /usr/bin/echo
echo is /bin/echo
whovian@phy504:/bin$ type -a more
more is /usr/bin/more
more is /bin/more
whovian@phy504:/bin$ type -a cp
cp is /usr/bin/cp
cp is /bin/cp
whovian@phy504:/bin$ type -a mv
mv is /usr/bin/mv
mv is /bin/mv
```



- **type -a name** displays all locations containing the executable name
- Two locations may hold copies of the same program (here, **/bin** is just a **symbolic link** of **/usr/bin**).
- How does the system know where to look them up (and what is the priority of execution)?

```
whovian@phy504:/bin$ echo "$PATH"
/opt/conda/bin:/usr/local/nvidia/bin:/usr/local/cuda/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

# RECAP: **man** & **--help** commands

## MacOS vs Docker and the **--help**

```
Last login: Mon Jan 27 10:19:55 on ttys001
RiverSong:~ vivianmiranda$ ls --help
ls: unrecognized option '--help'
usage: ls [-@ABCFGHILOPRSTUWabcdefghiklmnopqrstuvwxy1%,] [--color=when] [-D format] [file ...]
RiverSong:~ vivianmiranda$
```



```
whovian@phy504:~$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILES (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
-a, --all                do not ignore entries starting with .
-A, --almost-all        do not list implied . and ..
--author                 with -l, print the author of each file
-b, --escape             print C-style escapes for nongraphic characters
--block-size=SIZE        with -l, scale sizes by SIZE when printing them;
                           a zero block size M' see SIZE format below

-B, --ignore-back       ng with ~
-c                        me (time of last
                           n);
                           name;
                           first

-C
--color[=WHEN]
-d, --directory
-D, --dired
-f
-F, --classify[=W
--file-type
--format=WORD

below
their contents
cs' dired mode
der
to entries WHEN
*'
-x, long -l,
rtical -C
```



Different systems may have different rules. MacOS is based on the BSD (Berkeley Software Distribution) Unix, while Linux adopts the GNU Core Utils as the bash distribution. They are not 100% compatible.

# Basic File Manipulation

# Hidden Files

Some files/folders are hidden (not shown by default **ls** and GUI)

```
whovian@phy504:~$ ls
bash.bashrc  host  stablesort.txt
whovian@phy504:~$ ls -d .*
.  ..  .cache  .cmake  .conda  .jupyter  .local  .ssh
whovian@phy504:~$ more .conda/

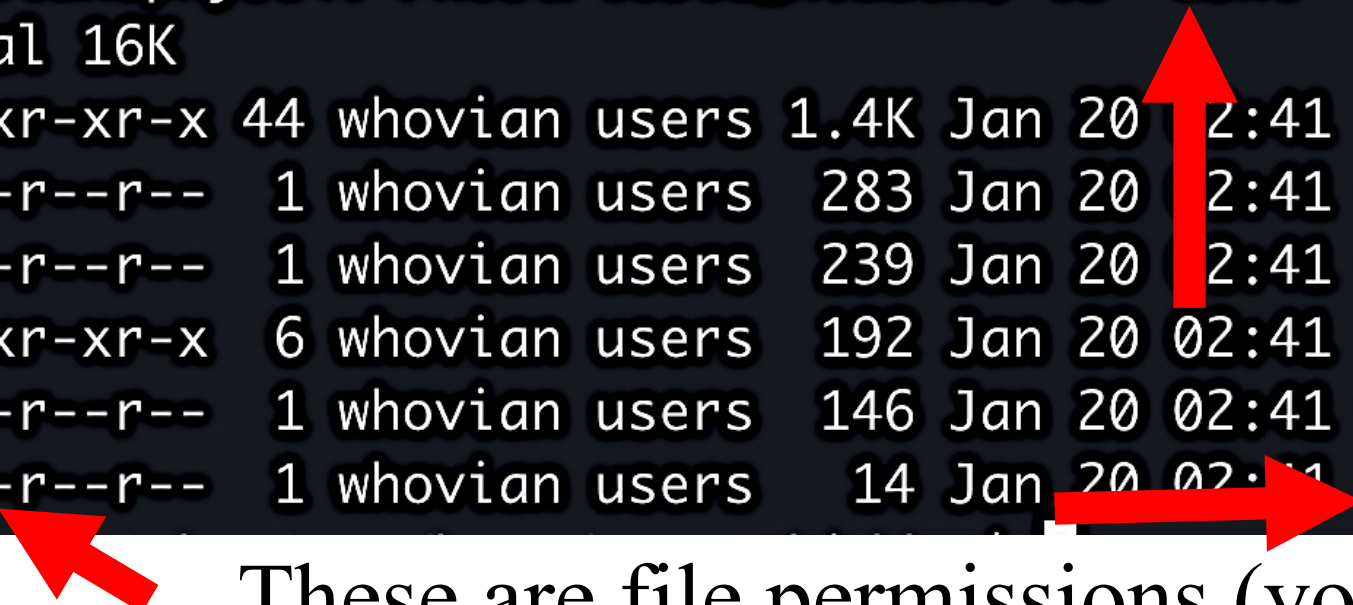
*** .conda/: directory ***
```

They are usually cache and system files. What are those? These are files that will crash the OS if deleted. Hidden files on Unix start with **.** (dot)

# Hidden Files

Some files/folders are hidden (not shown by default **ls**)

```
whovian@phy504:~/host/teste_hidden$ ls -Shl
total 12K
-rw-r--r-- 1 whovian users 283 Jan 20 02:41 whileloop3.sh
-rw-r--r-- 1 whovian users 239 Jan 20 02:41 whileloop2.sh
-rw-r--r-- 1 whovian users 146 Jan 20 02:41 whileloop.sh
whovian@phy504:~/host/teste_hidden$ ls -aShl
total 16K
drwxr-xr-x 44 whovian users 1.4K Jan 20 02:41 ..
-rw-r--r-- 1 whovian users 283 Jan 20 02:41 whileloop3.sh
-rw-r--r-- 1 whovian users 239 Jan 20 02:41 whileloop2.sh
drwxr-xr-x 6 whovian users 192 Jan 20 02:41 .
-rw-r--r-- 1 whovian users 146 Jan 20 02:41 whileloop.sh
-rw-r--r-- 1 whovian users 14 Jan 20 02:41 .myhiddenfile.txt
```

Three red arrows are present: one points from the bottom left to the permissions '-rw-r--r--' of the last line; another points from the bottom right to the permissions '-rw-r--r--' of the second-to-last line; and a third points from the middle right to the directory name '..' in the first line of the second command's output.

These are file permissions (you may not be allowed to write on (or execute) all files.)

# **.bashrc** once more. Customization

- I hope you understand by now the large impact options have on the output of all bash programs, even simpler ones like **ls** (another example below)

```
whovian@phy504:~$ ls
bash.bashrc  host
whovian@phy504:~$ ls --color=auto -ACF
bash.bashrc*  .bash_logout  .cmake/      host/        .local/      .ssh/
.bash_history .bashrc       .emacs.d/    .jupyter/    .profile     .viminfo
whovian@phy504:~$ ls --color=auto -alF
total 56
drwsrws--- 1 whovian users 4096 Jan 27 02:35 ./
drwxr-xr-x 1 root   root  4096 Jan  6 19:31 ../
-rwxrwxrwx 1 root   root  1990 Jan  6 18:35 bash.bashrc*
-rw----- 1 whovian users  492 Jan 28 01:20 .bash_history
-rw-rw-r-- 1 whovian users  220 Aug 20 08:58 .bash_logout
-rw-rw-r-- 1 whovian users 3770 Jan  6 19:31 .bashrc
drwsrwsr-x 1 whovian users 4096 Jan  6 19:42 .cmake/
drwx--S--- 3 whovian users 4096 Jan 27 02:35 .emacs.d/
drwxr-xr-x 10 root   root   320 Jan 28 02:17 host/
drwsrwsr-x 1 whovian users 4096 Jan  6 20:05 .jupyter/
drwxr-sr-x 3 whovian users 4096 Jan 27 02:34 .local/
-rw-rw-r-- 1 whovian users  807 Aug 20 08:58 .profile
drwxr-xr-x 6 root   root   192 Jan 26 23:12 .ssh/
-rw----- 1 whovian users  556 Jan 27 02:34 .viminfo
-rw-rw-r-- 1 whovian users  212 Jan  6 19:42 .wget-hsts
```

- Use **.bashrc** to change the default behavior of a bash program. How?

**alias ls='ls**

**--color=auto -alF'**

# ACF? ALF? How to find help?

- The amateur way (not good for Planet Earth)

Is difference between alF and ACF options

Thought for 22s >

They're just different bundles of `ls` flags:

```
ls -alF
```

- `-a` = **all** entries, including **hidden** files *and* the special `.` and `..`
- `-l` = **long** format (permissions, owner, size, time, etc.)
- `-F` = **classify**: append a marker to names
  - `/` dir, `*` executable, `@` symlink (varies slightly by system)

So `-alF` = "show literally everything in long format, with type markers."

```
ls -ACF
```

- `-A` = **almost all**: includes hidden files **but excludes** `.` and `..`
- `-C` = **columns** (multi-column listing; usually the default when output is a terminal)
- `-F` = **classify** (same as above)

So `-ACF` = "show entries in columns, include hidden files except `.` and `..`, with type markers."

Quick practical differences you'll notice

- `-alF` prints a **detailed one-per-line** list and includes `.` and `..`.
- `-ACF` prints a **compact multi-column** list and **does not** include `.` and `..`.

If you want, paste a directory listing goal (e.g., "I want hidden files but not dot directories, and I want sizes") and I'll suggest the cleanest `ls` flags for that.

But, if you use AI,  
do it like that. A  
simple question, try  
to learn from the  
answer (spend time  
digesting the  
answer).

Avoid **cognitive  
offloading**

# ACF? ALF? How to find help?

- The pro way (using **man** command)

```
ls(1)                                                    General Commands Manual                               ls(1)

NAME
  ls - List directory contents.  Ignore files and directories starting with a '.' by default

SYNOPSIS
  ls [--help] [--format] [-C ] [-ll--long] [-x ] [-Tl--tabsize] [-m ] [--zero] [-Dl--dired] [--hyperlink] [-1 ] [-o ] [-g ] [-nl--nu-
  meric-uid-gid] [--quoting-style] [-Nl--literal] [-bl--escape] [-Ql--quote-name] [-ql--hide-control-chars] [--show-control-chars] [--time]
  [-c ] [-u ] [--hide] [-Il--ignore] [-Bl--ignore-backups] [--sort] [-S ] [-t ] [-v ] [-X ] [-U ] [-Ll--dereference] [--dereference-com-
  mand-line-symlink-to-dir] [-Hl--dereference-command-line] [-Gl--no-group] [--author] [-al--all] [-Al--almost-all] [-f ] [-dl--directory]
  [-hl--human-readable] [-kl--kibibytes] [--si] [--block-size] [-il--inode] [-rl--reverse] [-Rl--recursive] [-wl--width] [-sl--size]
  [--color] [--indicator-style] [-Fl--classify] [--file-type] [-p ] [--time-style] [--full-time] [-Zl--context] [--group-directories-first]
  [-Vl--version] [paths]

DESCRIPTION
  List directory contents.  Ignore files and directories starting with a '.' by default

OPTIONS
  --help  Print help information.

  --format
          Set the display format.

  -C      Display the files in columns.

  -l, --long
          Display detailed information.

  -x      List entries in rows instead of in columns.

Manual page ls(1) line 1 (press h for help or q to quit)
```

# ACF? ALF? How to find help?

- The pro way (using **man** command)

`-F, --classify[=<when>]`

Append a character to each file name indicating the file type. Also, for regular files that are executable, append '\*'. The file type indicators are '/' for directories, '@' for symbolic links, 'l' for FIFOs, '=' for sockets, '>' for doors, and nothing for regular files. when may be omitted, or one of:

none - Do not classify. This is the default.

auto - Only classify if standard output is a terminal.

always - Always classify. Specifying --classify and no when is equivalent to --classify=always. This will not follow symbolic links listed on the command line unless the --dereference-command-line (-H), --dereference (-L), or --dereference-command-line-symlink-to-dir options are specified.

[possible values: always, auto, never]

# **.bashrc** once more. Customization

- But now that you have changed the default **ls** behavior, with **alias ls='ls --color=auto -alF'** what if you want to change the options again? Command line arguments have precedence

```
whovian@phy504:~$ ls
total 56
drwsrws--- 1 whovian users 4096 Jan 27 02:35 ./
drwxr-xr-x 1 root    root  4096 Jan  6 19:31 ../
-rwxrwxrwx 1 root    root  1990 Jan  6 18:35 bash.bashrc*
-rw----- 1 whovian users  492 Jan 28 01:20 .bash_history
-rw-rw-r-- 1 whovian users  220 Aug 20 08:58 .bash_logout
-rw-rw-r-- 1 whovian users 3770 Jan  6 19:31 .bashrc
drwsrwsr-x 1 whovian users 4096 Jan  6 19:42 .cmake/
drwx--S--- 3 whovian users 4096 Jan 27 02:35 .emacs.d/
drwxr-xr-x 10 whovian users  320 Jan 28 02:17 host/
drwsrwsr-x 1 whovian users 4096 Jan  6 20:05 .jupyter/
drwxr-sr-x 3 whovian users 4096 Jan 27 02:34 .local/
-rw-rw-r-- 1 whovian users  807 Aug 20 08:58 .profile
drwxr-xr-x 6 whovian users  192 Jan 26 23:12 .ssh/
-rw----- 1 whovian users  556 Jan 27 02:34 .viminfo
-rw-rw-r-- 1 whovian users  212 Jan  6 19:42 .wget-hsts
whovian@phy504:~$ ls --color=auto -ACF
bash.bashrc*  .bash_logout  .cmake/  host/  .local/  .ssh/  .wget-hsts
.bash_history  .bashrc      .emacs.d/  .jupyter/  .profile  .viminfo
```

# The **cp** (copy) and **mv** (move) commands

**cp** or **mv** commands accept new filenames or directories (*dir*) as their second args. If *dir*, then type either relative or absolute paths

```
whovian@phy504:~/host$ ls
es.txt
whovian@phy504:~/host$ cp es.txt es2.txt
whovian@phy504:~/host$ mkdir test
whovian@phy504:~/host$ cp es.txt ./test/
whovian@phy504:~/host$ ls
es2.txt es.txt test
whovian@phy504:~/host$ ls ./test/
es.txt
whovian@phy504:~/host$ ls
es2.txt es.txt test
whovian@phy504:~/host$ mkdir test2
whovian@phy504:~/host$ mv es2.txt test2/
whovian@phy504:~/host$ ls
es.txt test test2
whovian@phy504:~/host$ ls ./test2/
es2.txt
whovian@phy504:~/host$
```

**cp** or **mv** commands have many options. Try

**cp --help**

and

**mv --help**

**Copy directories**

**cp -r**

one dash '-' and **r**

HW: research one vs two dashes on options

# Parameter expansion

If you need to create many directories, use parameter expansion

```
whovian@phy504:~/host/test$ mkdir data{0..12}
whovian@phy504:~/host/test$ ls
data0 data1 data10 data11 data12 data2 data3 data4 data5 data6 data7 data8 data9
```

If you need to move/copy many files, use parameter expansion

```
whovian@phy504:~/host/test$ ls
data0 data1 data10 data11 data12 data2 data3 data4 data5 data6 data7 data8 data9 tmp2
whovian@phy504:~/host/test$ cp -r data{0..12} tmp2/
whovian@phy504:~/host/test$ ls ./tmp2/
data0 data1 data10 data11 data12 data2 data3 data4 data5 data6 data7 data8 data9
```



Parameter expansion also works with different file types

```
whovian@phy504:~/host/test$ touch file.txt file.png
whovian@phy504:~/host/test$ cp file{.txt,.png} ./tmp2/
whovian@phy504:~/host/test$ ls tmp2/
data0 data1 data10 data11 data12 data2 data3 data4 data5 data6 data7 data8 data9 file.png file.txt
whovian@phy504:~/host/test$
```

# Absolute Pathnames

- **Absolute paths** can be used to change working directories. Try this:

**cd /**

This command changes your working directory to the root dir

Verify this by executing the **pwd** command!

Return to your home directory? **cd** command with **no** arguments

```
whovian@phy504:~$ pwd
```

```
/home/whovian
```

```
whovian@phy504:~$ cd /
```

```
whovian@phy504:/$ pwd
```

```
/
```

```
whovian@phy504:/$ ls
```

```
bin    dev    home  lib32  libx32  mnt          opt    root  sbin  sys  usr
```

```
boot  etc    lib   lib64  media   NGC-DL-CONTAINER-LICENSE  proc  run   srv   tmp  var
```

```
whovian@phy504:/$ cd
```

```
whovian@phy504:~$ pwd
```

```
/home/whovian
```

**pwd** provides the absolute path of  
the current directory

**ls** has many interesting options

```
whovian@phy504:~$ ls --help
```

```
Usage: ls [OPTION]... [FILE]...
```

```
List information about the FILES (the current directory by default).
```

```
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.
```

check **ls --help**

# Relative Pathnames

Sometimes, absolute paths are the quickest way to refer to a file/dir

The absolute path to a file or directory may be long!

```
RiverSong:Vivian vivianmiranda$ ls
Docker          PHY504.pptx
RiverSong:Vivian vivianmiranda$ pwd
/Users/vivianmiranda/Library/CloudStorage/OneDrive-StonyBrookUniversity/teaching/Phy504/Vivian
```

**My laptop**

This could make commands unwieldy! Is there an easier way?

A **relative path** or **relative pathname** specifies the position of a file/dir relative to some other entity the shell knows about

```
RiverSong:Vivian vivianmiranda$ ls ../Docker/
RiverSong:Vivian vivianmiranda$ ls ~/
Applications
Applications (Parallels)
```

**from current dir**      OneDrive - Stony Brook University

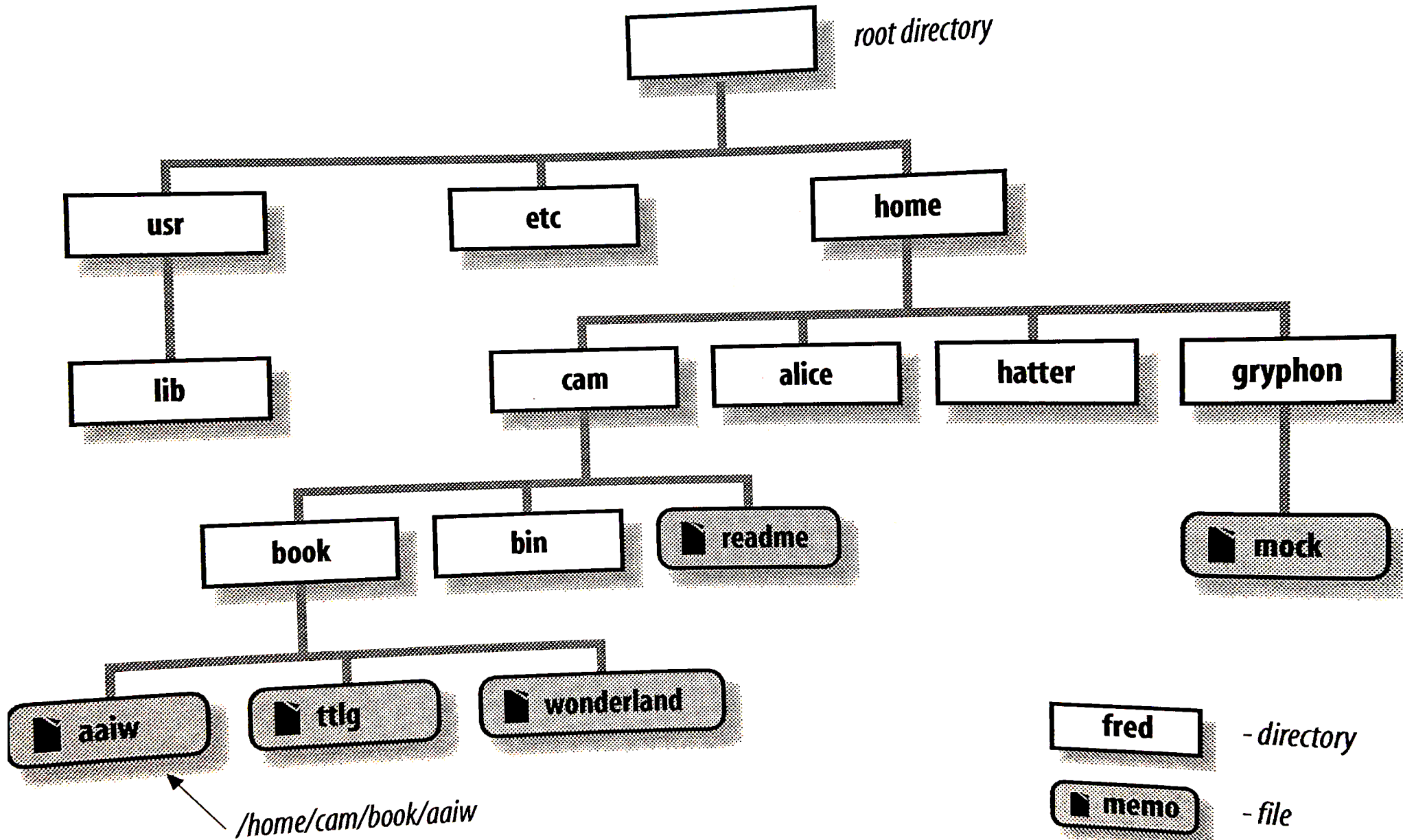
```
RiverSong:Vivian vivianmiranda$ ls ../
PHY277      PHYS305_2022  Shared
RiverSong:Vivian vivianmiranda$ ls ../../
Chicago DL  Phy504  SM.pdf  books
RiverSong:Vivian vivianmiranda$
```

**from parent directory**      Vivian      bash

**from grandparent directory**

# ? Parent / Grandparent directories ?

Directories in Unix system form a directory tree



# Deleting Directories

Directories can be removed (deleted) with the **rmdir** (remove dir)

**THERE IS NO TRASH (file recovery) – BE CAREFUL**

**Syntax:**

**rmdir** *<name\_of\_new\_directory>*

**Example:**

**rmdir codes**

Deletes the directory named **codes**

```
whovian@phy504:~$ rmdir --help
Usage: rmdir [OPTION]... DIRECTORY...
Remove the DIRECTORY(ies), if they are empty.

  --ignore-fail-on-non-empty
                        ignore each failure that is solely because a directory
                        is non-empty
  -p, --parents         remove DIRECTORY and its ancestors; e.g., 'rmdir -p a/b
                        ' removes 'a/b' and 'a'
  -v, --verbose         output a diagnostic for every directory processed
  --help               display this help and exit
  --version             output version information and exit
```

# Deleting Files/Dir

Directories can be removed (deleted) with the **rm** (Remove file)

BE EXTREMELY CAREFUL WITH **rm** COMMAND

Syntax:

**rm** *<name\_of\_new\_file>*

Example:

**rm codes.txt**

Deletes the file named **codes**

**BE CAREFUL WITH **rm****

This command deletes

**ALL** your **files/dir**

**-r** (dir), **-f** (force)



# Power comes with responsibility

You can erase all files in your system with wrong command

## Terminal with docker

Don't give the VM access to all your files. Create an empty folder

Example: I created `/Users/vivianmiranda/data/teaching/Phy504Docker`

```
RiverSong:Phy504Docker vivianmiranda$ pwd
/Users/vivianmiranda/data/teaching/Phy504Docker
RiverSong:Phy504Docker vivianmiranda$
RiverSong:Phy504Docker vivianmiranda$ docker run --platform linux/amd6
4 --hostname phy504 -it -p 8080:8888 -v $(pwd):/home/whovian/host/ -v
~/.ssh:/home/whovian/.ssh:ro vivianmiranda/whovian-phy504
```

```
*****
*
* Welcome to Docker Whovian-PHY504 docker image. I hope this VM
* will help you learn shell, C/C++/Fortran, and Python
*
```

# Power comes with responsibility

You can erase all files in your system with wrong command

## Terminal with docker

Your files will be seen inside the VM at **/home/whovian/host/**

Don't give the VM access to all your files. Create an empty folder

```
RiverSong:Phy504Docker vivianmiranda$ ls
2          example.sh      io.txt          singers2.txt
es.txt     hello.txt               singers.txt
```

```
whovian@phy504:~$ cd ~/host/
whovian@phy504:~/host$ ls
2 es.txt example.sh hello.txt io.txt singers2.txt singers.txt
```

# The **more** command

The **more** command displays a text file (quite powerful features)

```
RiverSong:lects vivianmiranda$ more -s 121_equations.tex
\documentclass[12pt]{article}
\usepackage[left=1.0in,right=1.0in]{geometry}
%\geometry{letterpaper,landscape,margin=0.75in}
```

```
\begin{document}
```

**-s squeeze multiple  
blank lines into one**

```
Slide 3 equation 1:
```

The **-s** option squeezes multiple blank lines into one single blank line

```
RiverSong:lects vivianmiranda$ more +35 121_equations.tex
\[
f(x) = x^2
\]
with  $x_L=1$  and  $x_H=2$ .
```

**press 'q' to quit**

**displaying the file**

The **+num** skips the first num lines

Powerful feature **+pattern** find a pattern (word or **regular expressions**)

# The **head** command

The **head** command displays the beginning of a text file

```
whovian@phy504:~/host$ head -n 2 es.txt
Aadil
Aali
whovian@phy504:~/host$ head -n 4 es.txt
Aadil
Aali
Aaliyah
Aaltje
whovian@phy504:~/host$ head -n 2 -v es.txt
==> es.txt <==
Aadil
Aali
```

default  
behavior:  
show first 10  
lines

- The **-v** option = verbose (show file name)
- The **-n** option = number of lines to show
- [Interesting fact about -n option](#)

# The **head** command

Understand language syntax (not your preconception notions). Example: the minus sign



The info page and [the online manual](#) for GNU `head` contain this part:

7

For compatibility `head` also supports an obsolete option syntax `-[NUM][bkm][cqv]`, which is recognized only if it is specified first.



The idea that `head -1` is the same as `head -n 1` is that the dash is not a minus sign, but a marker for a command line option. That's the usual custom: things that start with dashes are options controlling how to do processing, other stuff in the command line are file names or other actual targets to process. In this case, it's not a single-character option, but a shorthand for `-n`, but it's still basically an option, and not a filename. In `head +1` or `head 1`, the `+1` or `1` would be taken as file names, however.

A double dash `--` or `--something` also has a distinct meaning, by itself (`--`) it stops option processing, and when followed by something else, it marks a GNU style long option. So having `head --1` for `head -n -1` wouldn't match the custom.

If I were to guess, I'd assume the quaint shortcut for `-n i` exists for positive `i` but not for negative `i` since the former case is useful more often and easier to implement. (Besides, the standard `head` is only defined for a positive value of lines.)

# The **head** command

- The **-n** option also accepts negative integers!

```
whovian@phy504:~/host$ more singers.txt
4 Avril_Lavigne.
10 Rihanna.
2 Lady_Gaga.
5 Usher
14 Shakira
1 Justin_Timberlake.
20 Billie_Joe_Armstrong
3 Taylor_Swift.
whovian@phy504:~/host$ head -n -2 singers.txt
4 Avril_Lavigne.
10 Rihanna.
2 Lady_Gaga.
5 Usher
14 Shakira
1 Justin_Timberlake.
```

Negative number (example -2) = print ALL but the last two

# The **tail** command

Finding in real time who is trying to **hack** my server

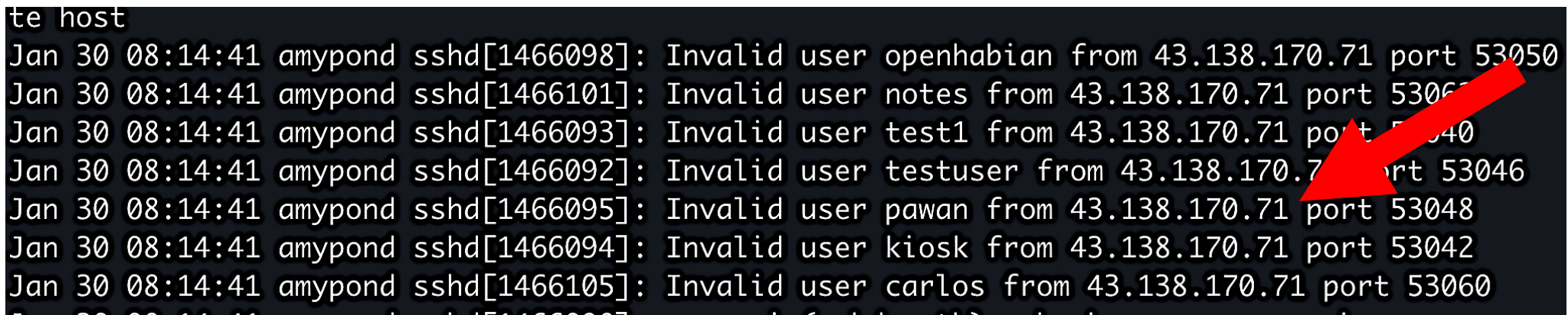
```
vivianmiranda@amypond:~$  
vivianmiranda@amypond:~$ tail -f -n 0 /var/log/auth.log  
Jan 26 14:32:50 amypond sshd[360949]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh  
ruser= rhost=202.51.74.123 user=root  
Jan 26 14:32:52 amypond sshd[360949]: Failed password for root from 202.51.74.123 port 50690 ssh2  
Jan 26 14:32:53 amypond sshd[360949]: Received disconnect from 202.51.74.123 port 50690:11: Bye Bye [preauth]  
Jan 26 14:32:53 amypond sshd[360949]: Disconnected from authenticating user root 202.51.74.123 port 50690 [preauth]  
Jan 26 14:32:55 amypond sshd[360977]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh  
ruser= rhost=65.21.2.173 user=root  
Jan 26 14:32:57 amypond sshd[360977]: Failed password for root from 65.21.2.173 port 59946 ssh2  
Jan 26 14:32:58 amypond sshd[360977]: Received disconnect from 65.21.2.173 port 59946:11: Bye Bye [preauth]  
Jan 26 14:32:58 amypond sshd[360977]: Disconnected from authenticating user root 65.21.2.173 port 59946 [preauth]  
Jan 26 14:33:12 amypond sshd[361035]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh  
ruser= rhost=43.135.163.98 user=root  
Jan 26 14:33:12 amypond sshd[361033]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh  
ruser= rhost=81.182.248.193 user=root  
Jan 26 14:33:15 amypond sshd[361035]: Failed password for root from 43.135.163.98 port 56836 ssh2  
Jan 26 14:33:15 amypond sshd[361033]: Failed password for root from 81.182.248.193 port 47134 ssh2  
Jan 26 14:33:15 amypond sshd[361035]: Received disconnect from 43.135.163.98 port 56836:11: Bye Bye [preauth]  
Jan 26 14:33:15 amypond sshd[361035]: Disconnected from authenticating user root 43.135.163.98 port 56836 [preauth]  
Jan 26 14:33:15 amypond sshd[361033]: Received disconnect from 81.182.248.193 port 47134:11: Bye Bye [preauth]  
Jan 26 14:33:15 amypond sshd[361033]: Disconnected from authenticating user root 81.182.248.193 port 47134 [preauth]
```

That is one reason why you can't have a weak password. If your laptop connects to the internet, it is vulnerable to hacking.

# The **tail** command

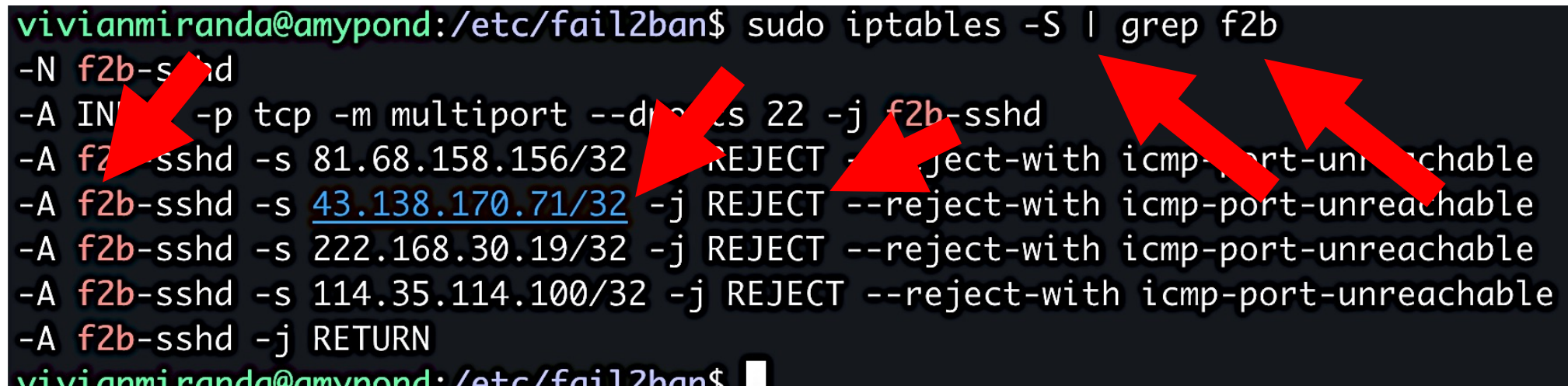
Finding in real time who is trying to **hack** my server

```
te host
Jan 30 08:14:41 amypond sshd[1466098]: Invalid user openhabian from 43.138.170.71 port 53050
Jan 30 08:14:41 amypond sshd[1466101]: Invalid user notes from 43.138.170.71 port 53067
Jan 30 08:14:41 amypond sshd[1466093]: Invalid user test1 from 43.138.170.71 port 53040
Jan 30 08:14:41 amypond sshd[1466092]: Invalid user testuser from 43.138.170.71 port 53046
Jan 30 08:14:41 amypond sshd[1466095]: Invalid user pawan from 43.138.170.71 port 53048
Jan 30 08:14:41 amypond sshd[1466094]: Invalid user kiosk from 43.138.170.71 port 53042
Jan 30 08:14:41 amypond sshd[1466105]: Invalid user carlos from 43.138.170.71 port 53060
```



I use a software named **fail2ban** that locks for 120 minutes anyone who tries to connect to my server more than 5 times.

```
vivianmiranda@amypond:/etc/fail2ban$ sudo iptables -S | grep f2b
-N f2b-sshd
-A INPUT -p tcp -m multiport --dports 22 -j f2b-sshd
-A f2b-sshd -s 81.68.158.156/32 -j REJECT --reject-with icmp-port-unreachable
-A f2b-sshd -s 43.138.170.71/32 -j REJECT --reject-with icmp-port-unreachable
-A f2b-sshd -s 222.168.30.19/32 -j REJECT --reject-with icmp-port-unreachable
-A f2b-sshd -s 114.35.114.100/32 -j REJECT --reject-with icmp-port-unreachable
-A f2b-sshd -j RETURN
vivianmiranda@amypond:/etc/fail2ban$
```



If you want to create a server, **iptables** and **grep** commands are useful.

# The **tail** command

Finding in real time who is trying to **hack** my server via VNC

```
vivianmiranda@amypond:/var/log$ tail -f -n 10 /var/log/syslog
Jan 26 14:37:02 amypond vncserver-virtuald[1610]: Connections: connected: 94.232.45.214::61000 (TCP)
Jan 26 14:37:34 amypond vncserver-virtuald[1610]: Connections: disconnected: 94.232.45.214::61000 (TCP) ([System-10
4] read: Connection reset by peer (104))
Jan 26 14:38:00 amypond mattermost[6826]: {"timestamp":"2023-01-26 14:38:00.782 -05:00", "level":"info", "msg":"Simpl
eWorker: Job is complete", "caller":"jobs/base_workers.go:88", "worker":"ExpiryNotify", "job_id":"z6t6iqwp3ybdcnx687n
3puffh"}
Jan 26 14:39:01 amypond CRON[362266]: (root) CMD ( [ -x /usr/lib/php/sessionclean ] && if [ ! -d /run/systemd/syst
em ]; then /usr/lib/php/sessionclean; fi)
Jan 26 14:39:06 amypond systemd[1]: Starting Clean php session files...
Jan 26 14:39:06 amypond systemd[1]: phpsessionclean.service: Succeeded.
Jan 26 14:39:06 amypond systemd[1]: Finished Clean php session files.
Jan 26 14:42:07 amypond systemd[1]: Started Session 309 of user vivianmiranda.
Jan 26 14:45:01 amypond CRON[363709]: (root) CMD (command -v debian-sa1 > /dev/null && debian-sa1 1 1)
Jan 26 14:45:18 amypond vncserver-x11[1664,root]: Connections: disconnected: 92.255.85.183::64623 (TCP) ([IdleTimeo
ut] The authentication period has timed out.)
```

IP address details

**94.232.45.214**

 Sofia, Sofia-Capital, Bulgaria

IP address details

**92.255.85.183**

 Moscow, Moscow, Russia

This problem was mitigated last year.

How? Firewall + geoblocking

# The **tail** command

Mitigated not equal completely solved (from this morning)

```
vivianmiranda@amypond:~$ tail -f -n 0 /var/log/auth.log
```

```
Feb  3 10:15:01 amypond CRON[3842227]: pam_unix(cron:session): session opened for user root by (uid=0)
Feb  3 10:15:01 amypond CRON[3842227]: pam_unix(cron:session): session closed for user root
Feb  3 10:17:01 amypond CRON[3843895]: pam_unix(cron:session): session opened for user root by (uid=0)
Feb  3 10:17:01 amypond CRON[3843895]: pam_unix(cron:session): session closed for user root
Feb  3 10:25:01 amypond CRON[3850476]: pam_unix(cron:session): session opened for user root by (uid=0)
Feb  3 10:25:01 amypond CRON[3850476]: pam_unix(cron:session): session closed for user root
Feb  3 10:28:23 amypond sshd[3853165]: Invalid user  from 64.62.197.174 port 57947
Feb  3 10:28:27 amypond sshd[3853165]: Connection closed by invalid user 64.62.197.174 port 57947 [preauth]
Feb  3 10:30:01 amypond CRON[3854489]: pam_unix(cron:session): session opened for user root by (uid=0)
Feb  3 10:30:01 amypond CRON[3854489]: pam_unix(cron:session): session closed for user root
Feb  3 10:35:01 amypond CRON[3858577]: pam_unix(cron:session): session opened for user root by (uid=0)
Feb  3 10:35:01 amypond CRON[3858577]: pam_unix(cron:session): session closed for user root
Feb  3 10:36:23 amypond sshd[3859766]: Invalid user rpc from 103.82.26.182 port 48622
Feb  3 10:36:23 amypond sshd[3859766]: pam_unix(sshd:auth): check pass; user unknown
Feb  3 10:36:23 amypond sshd[3859766]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=103.82.26.182
Feb  3 10:36:26 amypond sshd[3859766]: Failed password for invalid user rpc from 103.82.26.182 port 48622 ssh2
Feb  3 10:36:27 amypond sshd[3859766]: Connection closed by invalid user rpc 103.82.26.182 port 48622 [preauth]
```

## Geolocation data from



**IP ADDRESS:** 103.82.26.182



**COUNTRY:** Vietnam 🇻🇳



**REGION:** Ha Noi



**CITY:** Hanoi

# The **cut** command

The **cut** command allows you to remove sections of a specified file and print the result to standard output.

```
whovian@phy504:~/host$ more employees.txt
Adeline Bird      clerk      $654
Weston Bradley    clerk      $789
Anaya Rice        director   $1334
Andre Wilkins     clerk      $446
Ian Norris        hygiene    $655
Omar Landry       accountant $465
Harley Edwards    secretary  $4563
Marie Snow        manager    $4566
Haylee Sweeney    CEO        $12334
Isis Ware         manager    $4566
Jaelyn Khan       researcher $789
```

```
whovian@phy504:~/host$ cut -c 1-5 employees.txt
Adeli
Westo
Anaya
Andre
Ian N
Omar
Harle
Marie
```

To extract the first char (**-c**) from each input line, run:

```
whovian@phy504:~/host$ cut -c 1 employees.txt
A
W
A
A
I
O
H
M
H
I
J
```

You can also extract char ranges with **-c X-Y**

You can also extract multiple ranges with **-c X-Y, Z-M**

# The **cut** command x2 (delimiters)

The **cut** command allows you to remove sections of a specified file and print the result to standard output.

To extract based on delimiters, e.g., spaces, use (**-d**)

```
whovian@phy504:~/host$ echo "Miranda is an IT provider" | cut -d ' ' -f 1
Miranda
whovian@phy504:~/host$ echo "Miranda is an IT provider" | cut -d ' ' -f 2
is
whovian@phy504:~/host$ echo "Miranda is an IT provider" | cut -d ' ' -f 3
an
whovian@phy504:~/host$ echo "Miranda is an IT provider" | cut -d ' ' -f 1-
Miranda is an IT provider
whovian@phy504:~/host$ echo "Miranda is an IT provider" | cut -d ' ' -f 1-3
Miranda is an
```

PS: the default delimiter is the **TAB** character, not **spaces**

# The **cut** command x3 (delimiters)

**cut** allows changing the delimiter on output w/ **--output-delimiter**

```
whovian@phy504:~/host/C$ grep whovian /etc/passwd
whovian:x:1000:1000:~/home/whovian:/bin/bash
whovian@phy504:~/host/C$ grep whovian /etc/passwd | cut -d ':' -f 1,7
whovian:/bin/bash
whovian@phy504:~/host/C$ grep whovian /etc/passwd | cut -d ':' -f 1,7 --output-delimiter=$'\n'
whovian
/bin/bash
```

Another option: **-s**; **cut** only removes chars when it finds a delimiter

```
whovian@phy504:~/host/C$ echo "HelloWorld" | cut -d " " -f 1 -s
whovian@phy504:~/host/C$ echo "Hello World" | cut -d " " -f 1 -s
Hello
```

Finally, there is **--complement** (self-explanatory)

```
whovian@phy504:~/host/C$ echo "Hello to the whole World" | cut -d " " -f 1
Hello
whovian@phy504:~/host/C$ echo "Hello to the whole World" | cut -d " " -f 1 --complement
to the whole World
```

# HW Part I: create examples that...

- Create a directory called **bpl** inside \$HOME. Populate this directory with two subdirectories, bin and scripts
- Move a file called **oldfile.txt** to a new location called **newfolder**.
- Rename a file called **oldfile.txt** to **newfile.txt**.
- Move multiple files at once.
- Move all files with a specific extension (e.g., **.txt**) to a new folder
- Move a folder and its contents to a new location
- Move a file and overwrite the destination file if it already exists.
- Rename a file.
- Move a file and give it a new name at the same time.
- Move a file to a new location, but only if the destination file does not already exist.
- Move a file to a new location, but only if the destination file is older than the source file.
- Move a file and give it a new name, but only if the destination file does not exist.

# HW Part II: create examples that...

- copy a file called **file.txt** from the current directory to a directory called **backup**
- copy a directory called **folder** and all its contents to a new location
- copy a file and give it a new name at the same time
- copy a file and preserve its original permissions
- copy a file and prompt before overwriting an existing file
- copy multiple files to a directory
- copy with force overwrite of an existing file
- copy a directory and its contents recursively
- copy all files with the **.jpg** extension from the current directory to a directory called **photos** and move the original files to a directory called **old\_photos**
- Using the command **find** (with command line option **-exec**), find all **.txt** files in a given directory and move them to a specific location

# HW Part III: **ssh** command line options

- In my **.bashrc**, I included the option **-o ServerAliveInterval=60** in all my **ssh** commands. Can you guess why? What **-o** stands for?

```
alias ocelote='ssh -o ServerAliveInterval=60 vivianmiranda@hpc.arizona.edu'  
alias arizona='ssh -o ServerAliveInterval=60 vivianmiranda@amypond.cosmoatuofa.com'  
alias arizona2='ssh -o ServerAliveInterval=60 vivianmiranda@128.196.208.65'  
  
alias amypond0='ssh -X -o ServerAliveInterval=60 vivianmiranda@192.168.1.5'  
alias amypond='ssh -XY -o ServerAliveInterval=60 vivianmiranda@129.49.63.5'  
alias amypond1='ssh -o ServerAliveInterval=60 grads@129.49.63.5'  
alias amypond2='ssh -o ServerAliveInterval=60 undergrads@129.49.63.5'  
alias amypond3='ssh -o ServerAliveInterval=60 postdocs@129.49.63.5'
```

The option **-o key=value** is a complex command because the number of configurations that can be adjusted to alter the behavior of either the **SSH client** or **server** is large. Do I need to know all possible key values? **Of course not!!**

# HW Part IV: `ssh` command line options

- `-o ServerAliveInterval=60` implies that the **SSH client** will periodically send a signal to the **SSH server** to prevent the SSH connection from being lost.
  - What are the two things that can happen after you send a signal (hint: sending a signal is like trying to say hello to someone)?
  - If the **SSH client** is “ghosted,” the connection will drop after a finite number of attempts. We can control that if we intend to keep trying the connections at all costs. Check all possible keys associated with `-o` option using `man` and try to guess which key will control that. This is not that hard, **SSH** options have reasonable names.

# HW V: create examples that...

- **Research about the `tail` shell command**
- display the last 20 lines of a file (hint: this uses the command `tail`)
- display the first 15 lines of a text file called `file.txt`
- display the first 10 lines of multiple files at once
- open two terminals. Using `cd`, make sure they are located in the same folder. In the first terminal execute,

```
for i in {0..100}; do sleep 2; echo $i >> ./output_hw3.txt ; done
```

Now, in the second terminal, write a command that shows the last 10 lines of `output_hw3.txt`, with automatic refresh while the file is being updated by the command on the first terminal.

# Additional Reading

- This article on medium <https://medium.com/@codingmaths/bin-bash-what-exactly-is-this-95fc8db817bf> explains why we always write **#!/bin/bash** in the first line of bash script. Additional reading on this topic here (StackOverflow answer): <https://unix.stackexchange.com/a/733163/464161>
- This article on Medium <https://betterprogramming.pub/how-to-do-twice-as-much-with-half-the-keystrokes-using-bashrc-e68622825c2e> explains some nice tricks that can be saved on **.bashrc** file
- This article on Medium <https://betterprogramming.pub/3-uncommon-bash-tricks-that-you-should-know-c0fc988065c7> shows some nice tricks on bash.
- Nice introduction to **ssh** with a discussion about public-private keys: <https://blog.robertelder.org/what-is-ssh/>
- If you don't like any of the shells in the market (**bash**, **zsh** ...), you can create **(with years of work)** your own shell using C. Here is a nice tutorial on how to do that, which can provide insights into how a shell works: <https://brennan.io/2015/01/16/write-a-shell-in-c/>