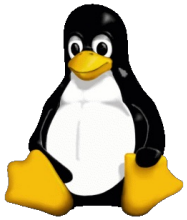
Introduction to the Command Line:

The Shell and Basic Commands

Ricky Patterson
Research Data Services
https://library.virginia.edu/data
University of Virginia Library
ricky@virginia.edu

What We Will Learn

- The fundamental commands of the Unix operating system.
- Everything here is also applicable to the Linux operating system. I will refer to both of these as *nix systems.



Outline

- Introduction to the Shell
  - Redirections/Pipes
- Files and Directories
  - Listing Files, File Types
  - Directory Structure
  - File Permissions
  - Wildcards
- Getting Help
- Advanced Commands, Editing

What Is *nix?

- *nix is a computer operating system, a control program that works with users to
  - run programs,
  - manage resources, and
  - communicate with other computer systems.
- *nix systems are
  - Multiuser: Several people can use a *nix computer at the same time
  - Multitasking: Any of these users can also run multiple programs at the same time
  - Plain Text Data Storage
  - Hierarchical File System

How do you get *nix?

- Linux and macOS are built on Unix, so it is available natively on these operating systems. On a Mac, you will use Terminal to interact with the shell.
- Windows 10 and 11 (64-bit) now allow you to activate the Windows Subsystem for Linux (WSL). You need to be an administrator of your computer in order to do this.
  - https://docs.microsoft.com/en-us/windows/wsl/install
  - Alternatively, you can install a Linux Virtual Machine under Windows, but WSL is much more straightforward.
- For this class, in a pinch, you can make use of a browser terminal testbed: https://bit.ly/fedora-term

Command Line of *nix

•When you first log into a ***nix (Unix or Linux)** system, you are presented with something that looks like this, or this:

```
(base) UL-RJP0I-MBP13:~ rjp0i$
```

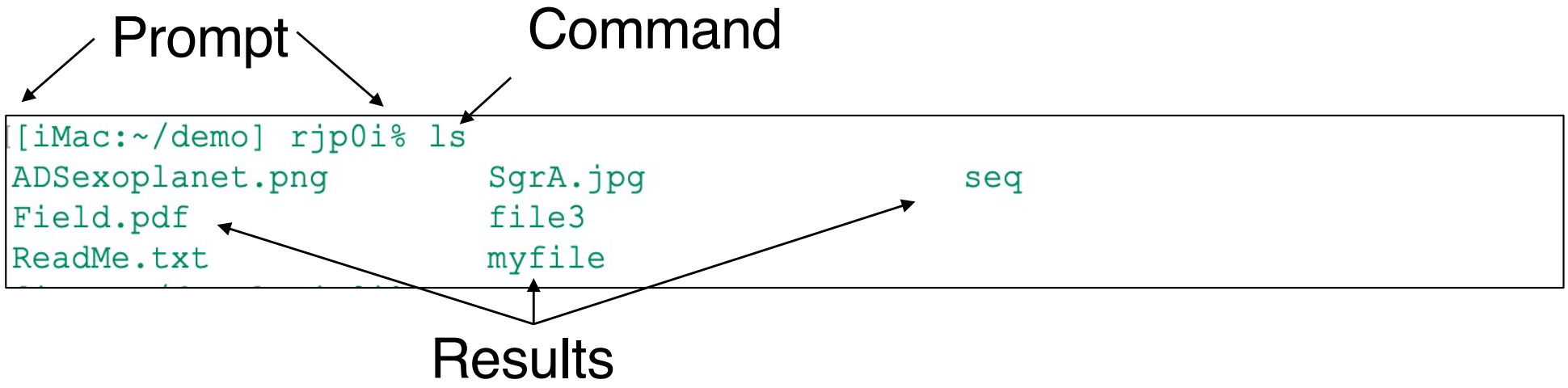
```
bash-3.2$
```

•That "something" is called a **prompt.** It is prompting you to enter a **command**.

•Every *nix **command** is a sequence of **letters** and **numbers.** (There are no spaces in a command name itself).

•*nix is also **case-sensitive**. This means that ***cat*** and ***Cat*** and ***CAT*** are all different commands.

Command Line



Why use the command line, instead of a GUI interface?

- While a graphical tool is usually available for most tasks, you can **do more** from the command line
- Command line is always available, and it works the same from platform to platform. GUI tools can vary or not even be available (e.g., through remote access).
- Command line commands are easily documented/recorded and **easily reproduced**.

The Shell

- The prompt is displayed by a special program called the **shell**. It is the interface between the operating system and the user.

- **Shells** **accept** commands from user and uses the operating system to **run** those commands. It then returns the results to the user.

- **Shells** can also be programmed in their own language. These programs are called "**shell scripts**". Shell scripts are powerful, but beyond the scope of this introduction.

The Shell

- The shell parses commands before executing them.
- It expands **wildcard** expressions ("ls *.dat" to show all files ending in .dat)
- The shell stores a **history** of previous commands
- The shell provides a way for the user to define commands (**aliases**)
- The shell maintains a set of user-defined variables (environment and system variables)

The Shell

- The shell has the ability to **auto-complete** commands or file names.
- The shell provides control structures (while/do, if/then/else) that allow you to write programs (**shell scripts**)
- The shell provides some plumbing which allows you to connect commands together with **pipes**, as well as to redirect input and output

The Shell

- When you first login, the **prompt** is displayed by **the default shell**, and you are running your first *nix program. (This might be bash, or zsh, or…)

- If you remain logged in, the *shell* will constantly be running (unless you choose to change to another shell).

- Other shells available include csh, tcsh, ksh, zsh and fish. They all behave very similarly but have differences/quirks that appeal to different users.

Command Syntax: Options

```
[iMac:~/demo] rjp0i% ls -l
total 1632
-rw-r--r--@ 1 rjp0i  staff   71412 Jun  6 11:33 ADSexoplanet.png
-rw-r--r--@ 1 rjp0i  staff  126316 Jun  6 11:32 Field.pdf
-rw-r--r--  1 rjp0i  staff       0 Jun  6 11:39 ReadMe.txt
-rw-r--r--@ 1 rjp0i  staff  628504 Jun  6 11:32 SgrA.jpg
-rw-r--r--  1 rjp0i  staff       0 Jun  6 11:39 file3
-rw-r--r--  1 rjp0i  staff       0 Jun  6 11:33 myfile
-rwxr-xr-x  1 rjp0i  staff     328 Jun  6 11:34 seq
```

- Commands can usually be modified with options (switches). An option modifies how the program **runs.**
- For **ls**, **-l** is an **option** that lets you see more information about the contents of the current directory (type **ls -l** in your terminal)
- A few other switches for **ls**
  -S (combined with **–l**) sorts files in order of descending size
  -T (combined with **–l**) sorts files in reverse time order
  -a Lists all files, including hidden files
- There are many, many more options for ls.

Command Syntax: Options

- Many *nix commands are like **ls**.
  - They have **options**, which are generally one character after a dash, and they have **arguments**.

- Unlike **ls**, some commands *require* certain arguments and/or options.

- Options can be combined: **ls –als**

- Some options can be given as words, rather than letters. If this is the case, it is used with a double dash: `lpr --help`

Autocompletion

- If you want to see commands beginning with c you can write **c** then press **Tab** key

```
c++             chage           codepage        continue
c++decl         charset         col             control-panel
c++filt         chattr          colcrt          convert_smbpasswd
c2ph            checkalias      collateindex.pl cp
c_rehash        chfn            colrm           cpio
cal             chgrp           column          cpp
calibrate_ppa   chmod           comm            cproto
cancel          chown           command         crontab
captoinfo       chsh            comp            csh
card            chvt            comp_err        csplit
case            ci              compgen         ctags
cat             cjpeg           compile_et      cut
catchsegv       cksum           complete        cvs
cc              clear           composeglyphs   cvsbug
cd              cmp             compress        cxpm
cdecl           cmuwmtopbm      consolechars    cytune
chacl           co              consolehelper
```

…and 104 more commands…

Now, what happens if you type “h” after the “c” and press Tab again?

Command-Line History:

```
~/demo> history
   349  16:14   wget http://download.adobe.com/pub/adobe/magic/svgviewer....
   350  16:15   tar tzvf adobesvg-3.01x88-linux-i386.tar.gz
   351  16:15   tar xzvf adobesvg-3.01x88-linux-i386.tar.gz
   352  16:15   cd adobesvg-3.01
   353  16:15   dir
   354  16:15   cd ..
   355  16:15   rm adobesvg-3.01*
   356  16:15   rm -rf adobesvg-3.01*
   357  16:19   git clone git://people.freedesktop.org/~cworth/svg2pdf
   358  16:19   cd svg2pdf
   359  16:19   dif
   360  16:19   dir
   361  16:19   git pull
   362  16:19   make
   363  16:19   dir
   364  16:20   ./svg2pdf ../drawing.svg
   365  16:20   ./svg2pdf ../drawing.svg junk.pdf
   366  16:20   acroread junk.pdf
```

The “history” command shows you commands you've recently entered.

You can use the up and down arrow keys to recall previously-typed commands and re-use them. If you know the beginning of a previously-entered command, you can re-run it by entering a “!” followed by the beginning of the command.

Aliases

```
~/demo> which echo
echo: shell built-in command.

~/demo> which rm
rm:       aliased to rm -i
```

Creating aliases:

In the bash shell:

```
~/demo> alias blarg=ls
```

In the tcsh shell:

```
~/demo> alias blarg ls
```

```
~/demo> blarg
clus.pdf      data-for-everybody.1.dat   phase2
cluster.pdf   ForYourEyesOnly.dat        readme.txt
cpuinfo.dat   phase1                     ReadMe.txt
```

(For aliased commands in bash, use **command –v** instead of **which** to check definition)

Stringing Commands Together

*nix commands can be strung together to carry out complex actions. The output of one command can be sent to the input of another, and so on. This is done commonly through “pipes” and “backticks”

In a directory with a lot of files, typing **ls –l** will quickly fill the screen. We can try typing **ls –l | less**
This shows the output of ls one page at a time, by redirecting the output of the **ls –l** command into the **less** command.

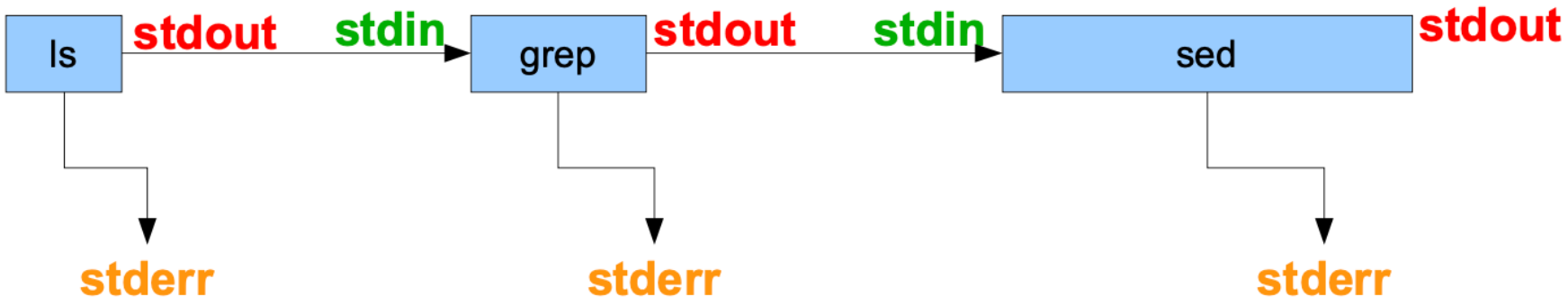
Much more complicated command strings are possible:
ls –l | grep stars | sed -e ‘s/star/STAR/’ | awk ‘{print $3,$NF}’

Stringing Commands Together

Keeping in mind where input/output and error messages go is important when stringing commands together.

- **stdout** is the channel the program uses to print output
- **stdin** is the channel the program uses to obtain input
- **stderr** is the channel the program uses to report errors

```
ls -l | grep stars | sed -e 's/st/ST/'
```



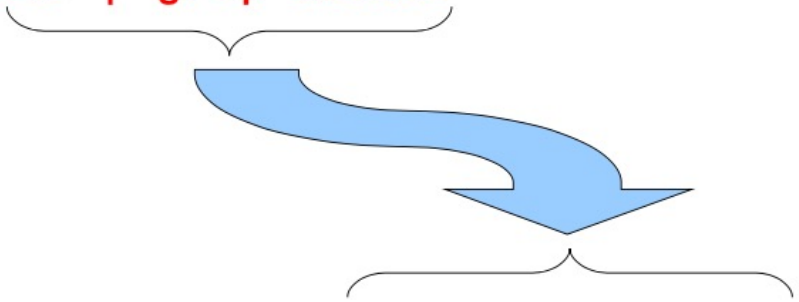
Redirecting Output to a file

- ls –l | grep stars > filename
  - Create a new file called **filename**, and fill it with the output (stdout) of the command. If filename already exists, it will be overwritten (if permitted)
- ls –l | grep stars >> filename
  - *Append* the output (stdout) from the commands to an existing file called **filename**

- To redirect **both** the stdout and the stderr to a file:
- In tcsh: **ls –l | grep stars >& filename**
- In bash: **ls –l | grep stars > filename 2>&1**

Nesting Commands with backticks (`)

```
~/demo/phase2> ls

bad_users.txt   laundry_list.txt  recipes.txt
good_users.txt  random_junk.txt   ugly_users.txt
...et cetera.

~/demo/phase2> ls | grep users

bad_users.txt
good_users.txt
ugly_users.txt


~/demo/phase2> grep Bryan `ls | grep users`

ugly_users.txt:Bryan
```



Commands between backticks ` ` are evaluated and then the output is inserted into the command line. It is just as though you had typed the output of the backticked command directly into the command line. The other commands patiently wait for the backticked commands to be evaluated.

Files and Directories

Files and Directories

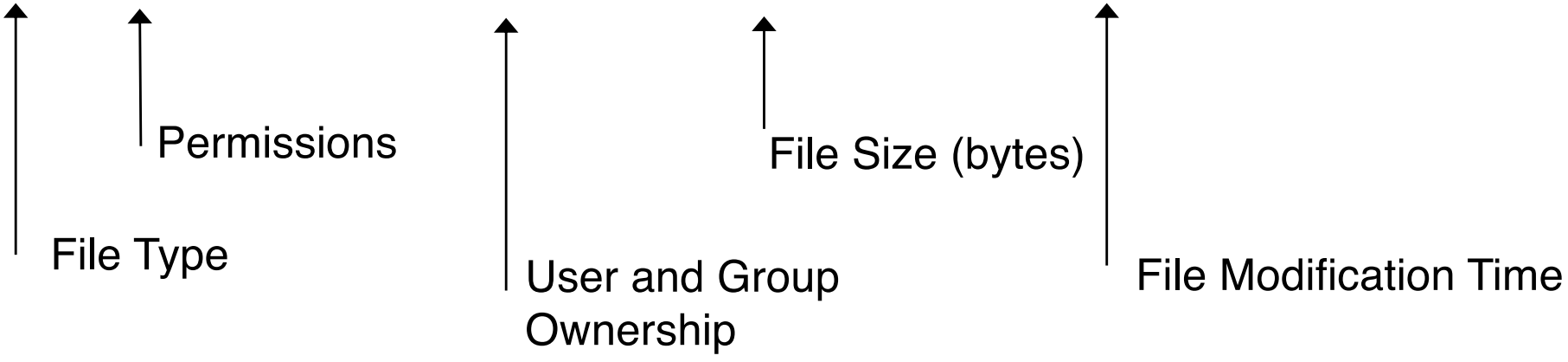
Listing Files in Current Directory

```
bash-4.2$ ls
ADSexoplanet.png        SgrA.jpg                myfile
Field.pdf               deeper                  seq
ReadMe.txt              file3                   test_run
bash-4.2$ ls -l
total 1640
-rw-r--r--@ 1 rjp0i  staff   71412 Jun  6 11:33 ADSexoplanet.png
-rw-r--r--@ 1 rjp0i  staff  126316 Jun  6 11:32 Field.pdf
-rw-r--r--  1 rjp0i  staff    1706 Jun  6 16:14 ReadMe.txt
-rw-r--r--@ 1 rjp0i  staff  628504 Jun  6 11:32 SgrA.jpg
drwxr-xr-x  2 rjp0i  staff      64 Jun  6 16:17 deeper
-rw-r--r--  1 rjp0i  staff       0 Jun  6 11:39 file3
-rw-r--r--  1 rjp0i  staff       0 Jun  6 11:33 myfile
-rwxr-xr-x  1 rjp0i  staff     328 Jun  6 11:34 seq
drwxr-xr-x_ 2 rjp0i  staff      64 Jun  6 16:17 test_run
```

- Note that file names are case sensitive. **Caution: This is true in Linux, but not quite true in OS X. OS X preserves the case of the filename, but ignores it.**

Listing Files in Current Directory

```
bash-4.2$ ls
ADSexoplanet.png        SgrA.jpg                myfile
Field.pdf               deeper                  seq
ReadMe.txt              file3                   test_run
bash-4.2$ ls -l
total 1640
-rw-r--r--@ 1 rjp0i  staff   71412 Jun  6 11:33 ADSexoplanet.png
-rw-r--r--@ 1 rjp0i  staff  126316 Jun  6 11:32 Field.pdf
-rw-r--r--  1 rjp0i  staff    1706 Jun  6 16:14 ReadMe.txt
-rw-r--r--@ 1 rjp0i  staff  628504 Jun  6 11:32 SgrA.jpg
drwxr-xr-x  2 rjp0i  staff      64 Jun  6 16:17 deeper
-rw-r--r--  1 rjp0i  staff       0 Jun  6 11:39 file3
-rw-r--r--  1 rjp0i  staff       0 Jun  6 11:33 myfile
-rwxr-xr-x  1 rjp0i  staff     328 Jun  6 11:34 seq
drwxr-xr-x  2 rjp0i  staff      64 Jun  6 16:17 test_run
```



Symbolic Links

```
bash-4.2$ ls -l
total 1640
-rw-r--r--@ 1 rjp0i  staff   71412 Jun  6 11:33 ADSexoplanet.png
-rw-r--r--@ 1 rjp0i  staff  126316 Jun  6 11:32 Field.pdf
-rw-r--r--  1 rjp0i  staff    1706 Jun  6 16:14 ReadMe.txt
-rw-r--r--@ 1 rjp0i  staff  628504 Jun  6 11:32 SgrA.jpg
drwxr-xr-x  2 rjp0i  staff      64 Jun  6 16:17 deeper
-rw-r--r--  1 rjp0i  staff       0 Jun  6 11:39 file3
-rw-r--r--  1 rjp0i  staff       0 Jun  6 11:33 myfile
lrwxr-xr-x  1 rjp0i  staff      16 Jun  6 16:32 opt_link -> /Users/rjp0i/opt
-rwxr-xr-x  1 rjp0i  staff     328 Jun  6 11:34 seq
drwxr-xr-x  2 rjp0i  staff      64 Jun  6 16:17 test_run
```

Symbolic links are alternative names for a file or directory. Here "opt_link" is a symbolic link pointing to a real file (a directory, in this case) called "opt" (which is located in a different directory).

To create a symbolic link, use the ln –s command:

```
ln -s RealFile SymlinkedFile
```

Order is important!

Directory Files

```
bash-4.2$ ls -l
total 1640
-rw-r--r--@ 1 rjp0i  staff   71412 Jun  6 11:33 ADSexoplanet.png
-rw-r--r--@ 1 rjp0i  staff  126316 Jun  6 11:32 Field.pdf
-rw-r--r--  1 rjp0i  staff    1706 Jun  6 16:14 ReadMe.txt
-rw-r--r--@ 1 rjp0i  staff  628504 Jun  6 11:32 SgrA.jpg
drwxr-xr-x  2 rjp0i  staff      64 Jun  6 16:17 deeper
-rw-r--r--  1 rjp0i  staff       0 Jun  6 11:39 file3
-rw-r--r--  1 rjp0i  staff       0 Jun  6 11:33 myfile
lrwxr-xr-x  1 rjp0i  staff      16 Jun  6 16:32 opt_link -> /Users/rjp0i/opt
-rwxr-xr-x  1 rjp0i  staff     328 Jun  6 11:34 seq
drwxr-xr-x  2 rjp0i  staff      64 Jun  6 16:17 test_run
```

Directories can contain other directories (sub-directories of the first directory). The sub-directories appear in the listing above. They can in turn contain other files and other directories (and symbolic links).

Storing Information - Directories

- In *nix, the collection of directories and files is called the file system. The file system consists of at least one directory, called the "**root**" directory
- Within the "**root**" directory, there are more directories, and inside those directories are files and yet more directories.
- In *nix there is a single directory tree (unlike Windows, with the separately lettered C: drive, D: drive, etc.)
- Note that *nix uses "/" for the directory separator, while Windows uses "\"

Special Directories: The Current Directory

- You can see what directory you are currently in using **pwd**

```
bash-4.2$ pwd
/Users/rjp0i/demo
```

- The path to a file or directory is given as a list of parent directories, separated by slashes.
- You can change your current directory using **cd**

```
bash-4.2$ cd deeper
bash-4.2$ pwd
/Users/rjp0i/demo/deeper
```

```
bash-4.2$ cd /Users/rjp0i/Desktop
bash-4.2$ pwd
/Users/rjp0i/Desktop
```

Special Directories: The Home Directory

- Each user has a home directory, which is the current directory when you first log in

```
bash-4.2$ echo $HOME
/Users/rjp0i
bash-4.2$ ls $HOME/demo
```

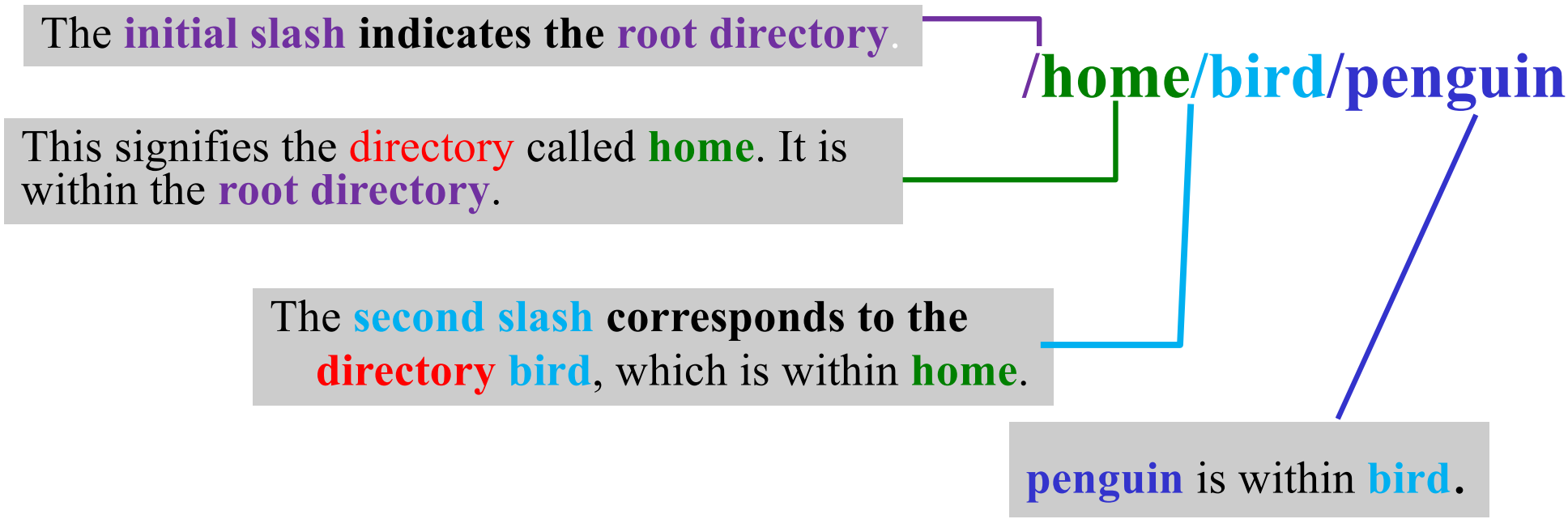
- You can always change to your home directory using **cd** without argument

```
bash-4.2$ pwd
/Users/rjp0i/demo/deeper
bash-4.2$ cd
bash-4.2$ pwd
/Users/rjp0i
```

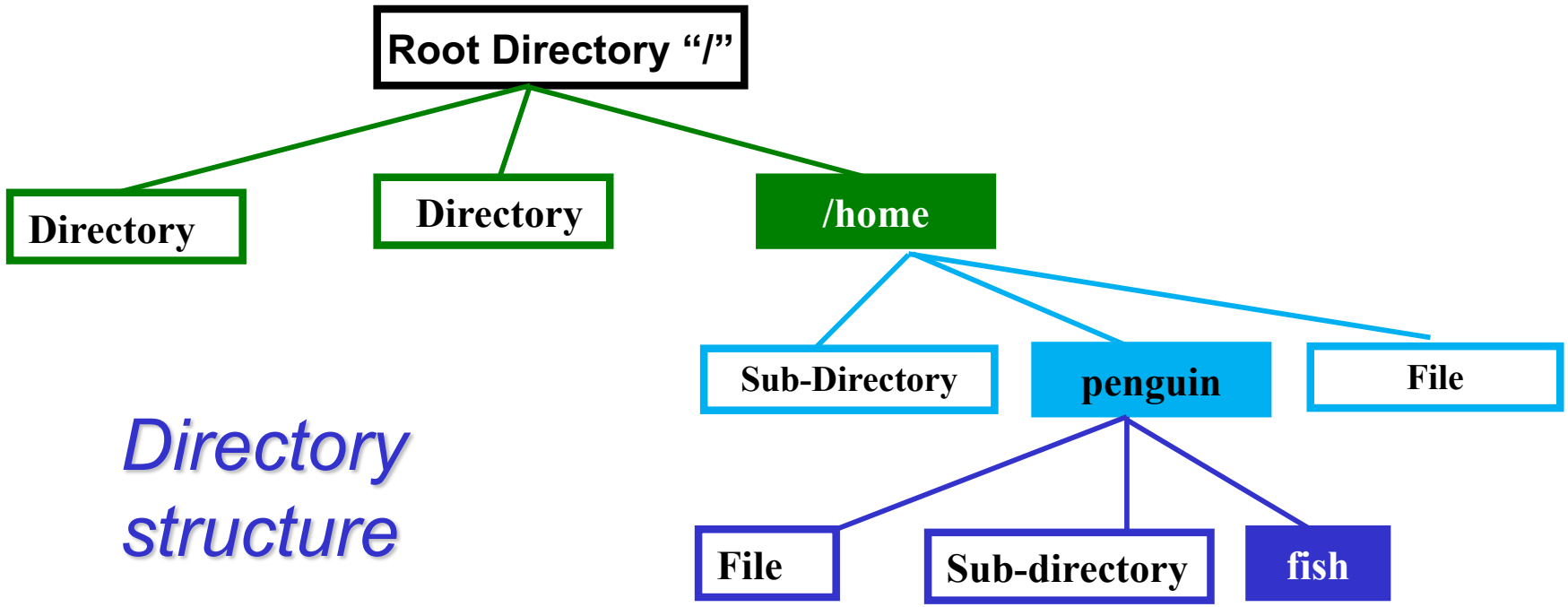
- The tilda is a shorthand way to refer to the home directory

```
bash-4.2$ ls ~/demo
```

- Each file and each directory have a name. (Keep in mind that directories are actually files)
- A short name for a file could be **penguin**,
- while it's "full name" could be **/home/bird/penguin**
  The full name is usually called the **path**.
- The path can be divided into a sequence of directories.
- For example, here is how **/home/bird/penguin** is read:



- A path may refer to either a directory or a filename, so **fish** could be either.
- All the items before the short name must be directories.



Directory structure

Navigating Directories

```
bash-3.2$ cd /home

bash-3.2$ cd
```

• Reminder: If you **omit the optional argument** *directory*, you're **returned to your home**, or original directory (the same as typing **cd ~**). Otherwise, **cd** will change you to the specified directory.

• There are two directory names used only for relative pathnames:

  • The directory "." refers to the current directory

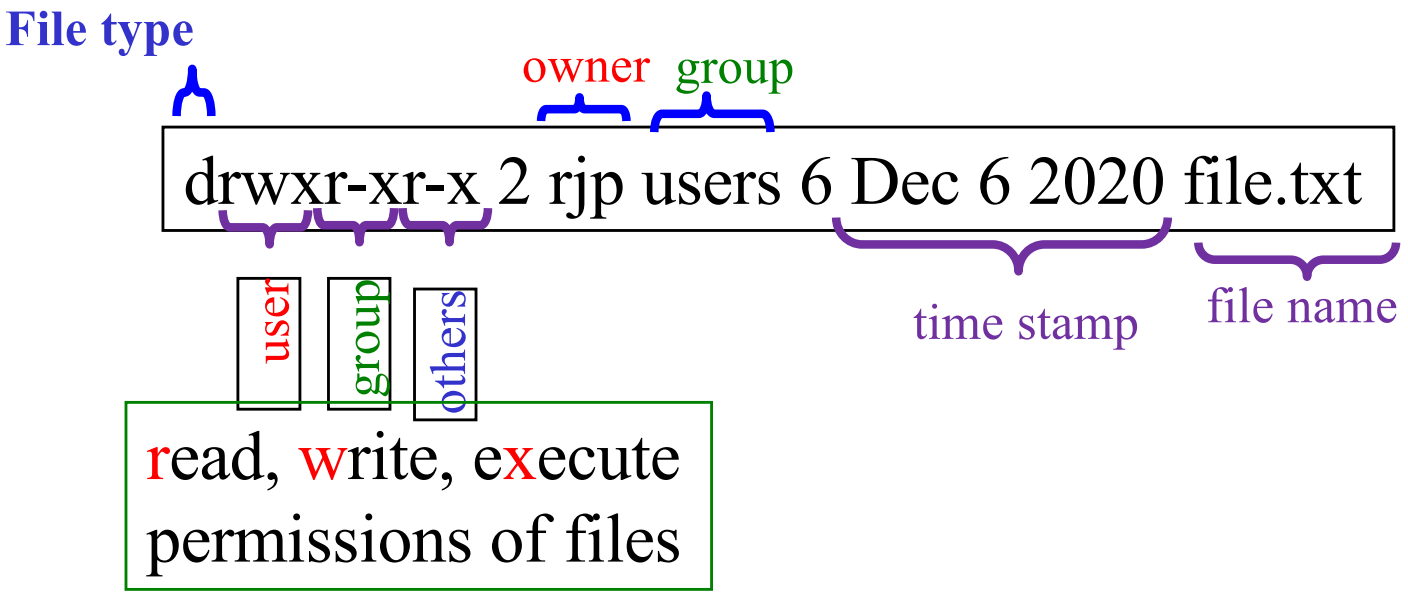
  • The directory ".." refers to the parent directory of the current directory

•The directory "**..**" is most useful moving back up a directory:

cd ..

•The command "**cd –**" will return you to the most recent directory visited.

Permissions



For each file, three sets of permissions bits (read, write, and execute) can be set. They apply to the three types of users. The user who owns the file, the group that owns the file, and everybody else (other) who has an account on the file system.

Permissions: chmod

- **chmod** (**ch**ange **mod**e) is used to change the permissions on a file.

(owner) (group) (others)

chmod [number][number][number] file1

Number = (read)4 + (write)2 + (execute)1

- Example: chmod 754 file1

for owner: *read*, *write* and *execute* permissions (4+2+1)

for group: *read* and *execute* permissions (4+0+1)

for others: only *read* permission (4+0+0)

Moving around on the command line

- ***Ctrl+A*** **Move to *beginning* of line**
- ***Ctrl+E*** **Move to *end* of line**
- ***Ctrl+L*** **Clear the screen**
- ***Ctrl+U*** **Clear the line *before* the cursor position**
- ***Ctrl+K*** **Clear the line *after* the cursor position**
- ***Ctrl+C*** **Kill the command that is currently running**
- ***Ctrl+D*** **Exit the current shell**
- ***Alt+F (or Esc+F)*** **Move cursor *forward* one word**
- ***Alt+B (or Esc+B)*** **Move cursor *backward* one word**

Wildcards – Globbing

- *nix recognizes wildcards that allow it to match patterns to look for files with similar names. **ls *.txt** will return a list of all files in the current directory that have a filename ending in ".txt"
- The shell automatically expands the wildcard, and passes the resulting matches to the command (in this case, **ls**)

*	Match any string of characters.
?	Match any single character.
[abc]	Match a single a, b or c.
[a-zA-Z]	Match any character in the range a-z or A-Z.

Some Examples of Wild-card Matching:

```
~/demo> ls -l
total 2
-rw-rw-r--    1 bkw1a  bkw1a          0 Aug  6 18:42 a.1
-rw-rw-r--    1 bkw1a  bkw1a          0 Aug  6 18:42 b.1
-rw-rw-r--    1 bkw1a  bkw1a          0 Aug  6 18:42 c.1
-rw-rw-r--    1 bkw1a  bkw1a        466 Aug  6 17:48 t2.sh
-rw-rw-r--    1 bkw1a  bkw1a        758 Jul 30 09:02 test1.txt

~/demo> ls -l t?.sh
-rw-rw-r--    1 bkw1a  bkw1a        466 Aug  6 17:48 t2.sh

~/demo> ls -l [ab]*
-rw-rw-r--    1 bkw1a  bkw1a          0 Aug  6 18:42 a.1
-rw-rw-r--    1 bkw1a  bkw1a          0 Aug  6 18:42 b.1

~/demo> ls -l [a-c]*
-rw-rw-r--    1 bkw1a  bkw1a          0 Aug  6 18:42 a.1
-rw-rw-r--    1 bkw1a  bkw1a          0 Aug  6 18:42 b.1
-rw-rw-r--    1 bkw1a  bkw1a          0 Aug  6 18:42 c.1
```

Getting Help

man

- The **man** command displays reference pages for the command you specify.
- The *nix **man** pages (**man** is short for *manual*) cover every command available.
- To search for a **man** page, enter **man** followed by the name of the command.
- For example:

```
bash-4.2$ man ls
```

```
LS(1)                    BSD General Commands Manual                    LS(1)

NAME
     ls -- list directory contents

SYNOPSIS
     ls [-ABCFGHLOPRSTUW@abcdefghiklmnopqrstuwx1%] [file ...]

DESCRIPTION
     For each operand that names a file of a type other than directory, ls
     displays its name as well as any requested, associated information.  For
     each operand that names a file of type directory, ls displays the names
     of files contained within that directory, as well as any requested, asso-
     ciated information.

     If no operands are given, the contents of the current directory are dis-
     played.  If more than one operand is given, non-directory operands are
     displayed first; directory and non-directory operands are sorted sepa-
     rately and in lexicographical order.

     The following options are available:

:
```

To view more, press spacebar
To exit, press “q”

Getting Help: man

- There is also a **keyword function** in **man**.
- For example;
  - If you are interested in any commands that deal with Postscript, the printer control language for Adobe
  - Type **man -k ps** or **man -k Postscript**,

    you'll get a listing of all commands, system calls, and other documented parts of *nix that have the word "ps" (or "Postscript") in their name or short description.
- This can be very useful when you're looking for a tool to do something, but you don't know its name - or if it even exists!

Navigating Directories

pwd

- **pwd** (**p**resent **w**orking **d**irectory) tells you your current directory.
  - *Note: Most commands act, by default, on the current directory. For instance, **ls** without any arguments displays the contents of the current directory.*

cd

- **cd** is used to **c**hange **d**irectories.
- The format of this command :

  cd new-directory (where new-directory is the name of the new directory you want).

mkdir

mkdir (**m**a**k**e **dir**ectory) is used to create a new directory,

- It can take more than one argument, interpreting each argument as another directory to create.
- By default, it will create the new directory as a subdirectory of the current directory

rmdir

rmdir (**r**e**m**ove **dir**ectory) is used to remove a directory,

- **rmdir** will refuse to remove a non-existent directory, as well as a **directory that has anything in it**.

Moving Files/Directories

- The primary commands for manipulating files under *nix are **cp**, **mv**, and **rm**. They stand for **c**o**p**y, **m**o**v**e, and **r**e**m**ove, respectively.

cp

- **cp** is used to copy contents of file1 to file2

  cp file1 file2 (*contents of file1 is copied to file2 in the same directory*)

  cp folder1/file1 folder2 (*contents of file1 is copied to file1 in the inside of folder2 directory*)

rm

- **rm** is used to remove a file.
  - **rm *filename* --->** removes a file named *filename*

mv

- mv is used to move a file.
  - **mv *filename /path/newname* --->** moves a file named *filename* to a new location, with a new name
- looks like **cp**, except that it deletes the original file after copying it.
- **mv** will rename a file if the second argument is a file. If the second argument is a directory, **mv** will move the file to the new directory, keeping it's shortname the same.

Operating on Files

- In addition to the commands like **cd**, **mv**, and **rm**, we learned in the directories discussion, there are other commands that just operate on files, but not the data in them.
- These include **touch**, **chmod**.
- None of these commands care what is within the file.

Examining Files

- There are two major commands used in *nix for listing files, **cat**, and **more** (and **less**). **head** and **tail** are also useful.

cat

- **cat** shows the contents of the file.

  cat [-nA] [file1 file2 . . . fileN]

- **cat** is not a user-friendly command-it doesn't wait for you to read the file and is mostly used in conjunction with pipes.
- However, **cat** does have some useful command-line options. For instance, **n** will number all the lines in the file, and **A** will show control characters.

Examining Files: more and less

more

- **more** is much more useful, and is the command that you'll want to use when browsing ASCII text files

more [-l] [+*linenumber*}] [*file1 file2 ... fileN*]

- A useful option is **l**, which will tell **more** that you aren't interested in treating the character Ctrl-L} as a ``new page'' character. **more** can also start on a specified line number.

less

less is similar to **more** but also allows you to scroll backwards or forward through a file. **less** also quickly loads a file and can operate on a file that is still being written to.

less > **more** or "**less** is **more**, more or less"

Examining Files: head and tail

head

head will display the first ten lines in the listed files.

head [- *lines*}] [l *file1 file2 ... fileN*]

- Any numeric option will be taken as the number of lines to print, so **head -15 frog** will print the first fifteen lines of the file *frog*

tail

- Like **head**, **tail** display only a fraction of the file.
- **tail** also accepts an option specifying the number of lines.

tail [-*lines*] [l *file1 file2 ... fileN*]

More info about the file

These commands will search a file, perform certain operations on the file, or display statistics about the file.

grep

- **grep** is the **g**eneralized **r**egular **e**xpression **p**arser.
- This is a fancy name for a powerful utility which can only search a text file.

grep [-nvwx] [-number] { *expression*} [*file1 file2 ... fileN*]

More info about the file

diff

- The GNU version of **diff** has over twenty command line options. It shows you what the differences are between two files
- **diff** ***file1 file2***

Some Other *nix Commands

The Power of *nix

- The power of *nix is hidden in small commands that don't seem too useful when used alone, but when combined with other commands produce a system that's much more powerful, and flexible than most other operating systems.
- The commands include **sort**, **grep**, **more**, **cat**, **wc**, **spell**, **diff**, **head**, and **tail**.

Common Text Editors

nano

- **nano** is a simple text editor. There are many text editors available. But nano is simple:
  - **nano file**

vi (vim)

- **vi** is the original Unix "visual editor" and is available on all *nix computers (but can be non-intuitive). Like emacs, it is context aware (autocompletion, indentation and color coding of text). A self-professed great Vim cheat sheet: http://vimsheet.com/

emacs

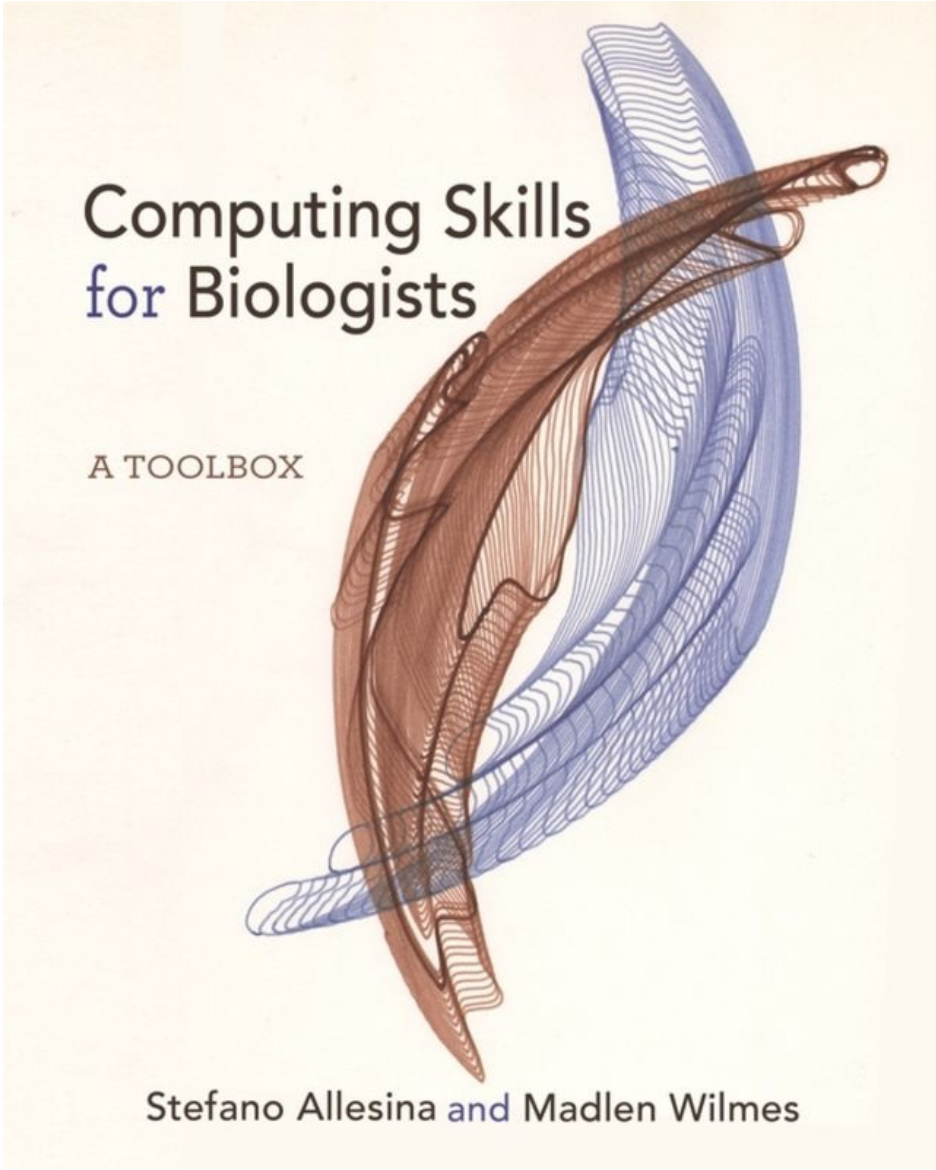
- **emacs** is a very powerful, extensible editor. Emacs and vi are among the most commonly used editors.
  Emacs cheat sheet:
  https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf

More help

- Chapter 1 of

Computing Skills for Biologists
(Allesina & Wilmes)

- See Virgo for free access to ebook



- See website for downloads
  https://computingskillsforbiologists.com

- Clone their git repository for examples:
  `git clone https://github.com/CSB-book/CSB.git`

This book is not just for biologists but is valuable for graduate students in any datacentric field. It covers R, Python, Regular Expressions, version control, LaTeX, SQL, and the command line.

More help

- **LinkedIn Learning** free courses, including:
  - "Learning Linux Command Line"
  - "Unix for Mac OS X Users"
    - Be sure to access LL materials for free by using this Library link to take advantage of the UVA subscription: https://library.virginia.edu/linkedin-learning
- Lots of online resources (unix.stackexchange…)
- http://www.doc.ic.ac.uk/~wjk/UnixIntro/
- Me – Please contact me (ricky@virginia.edu)