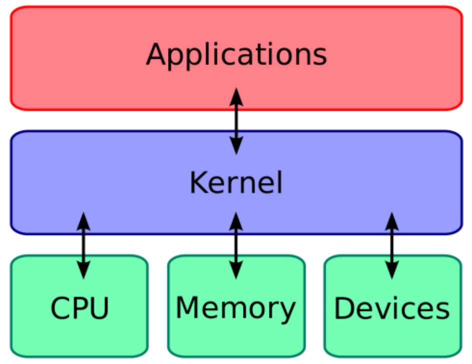
Day 1, AM Session

Richard Flamio Jr.

Kristina M. Ramstad

University of South Carolina Aiken

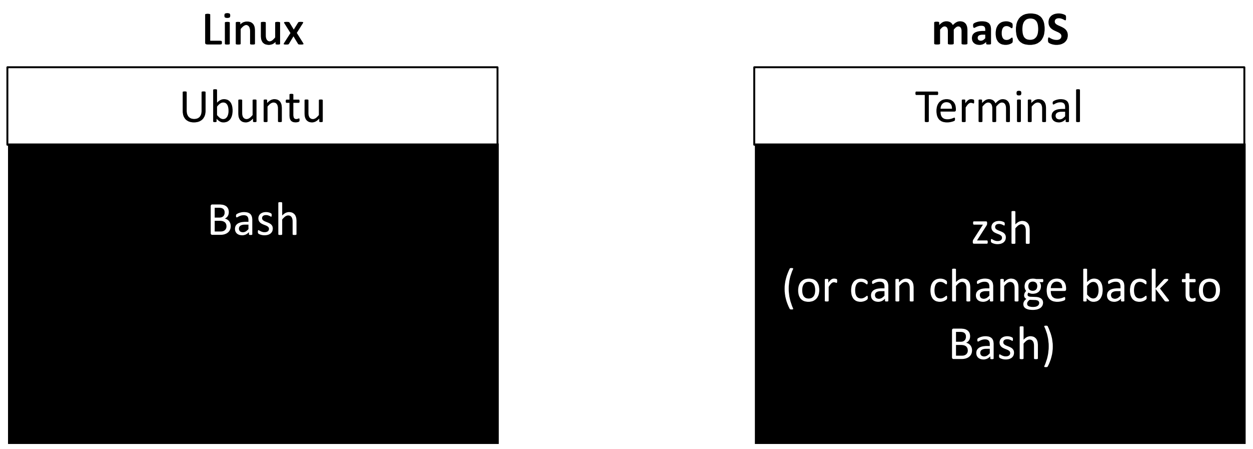


- Open-source kernel (computer program at core of operating system)
- Originally based on Unix operating system
- Multiple distributions (Ubuntu, Debian, SUSE, Fedora, etc.)

- Side note: macOS is Unix-like but neither Linux nor Unix

Shell

- Interactive command interpreter environment
- Keyboard command → shell (interpretation and passing commands) → operating system
- Ubuntu's default shell is the BASH (Bourne Again Shell) shell
- Apple macOS's default shell was BASH but is now zsh
  - still very similar, basic commands nearly identical



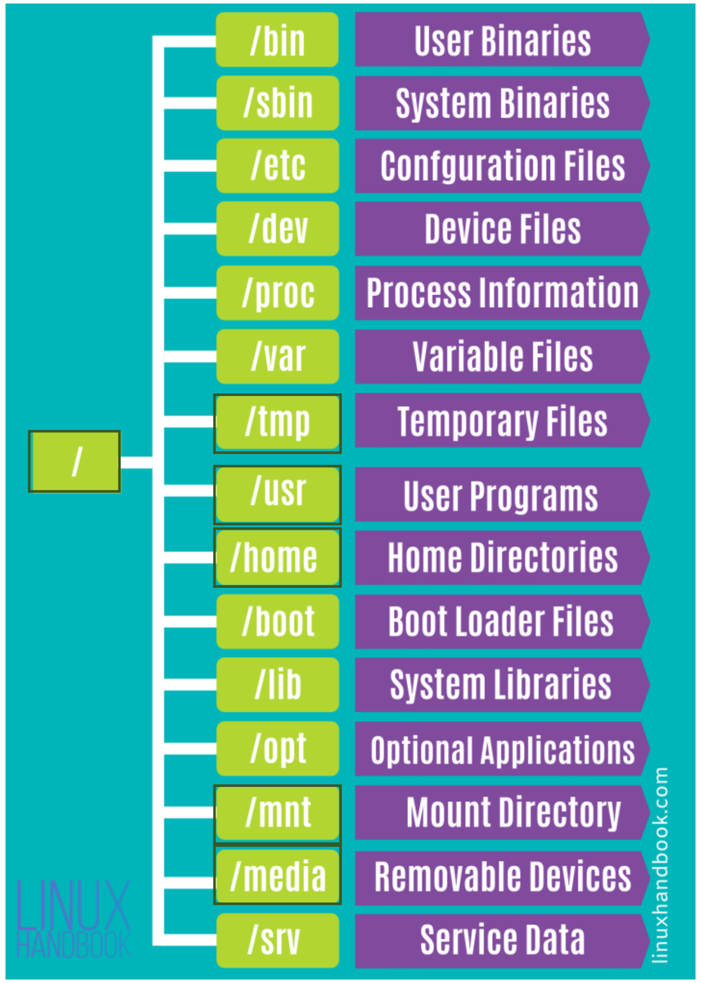
Attention Mac Users

`chsh -s /bin/bash`	to set as bash shell
`chsh -s /bin/zsh`	to set as zsh shell

Command+	make text in shell bigger
Command-	make text in shell smaller

Directory Structure

root directory



Temporary files

Installed software (/usr/local/bin)

Personal directories (user data and files)

Access USB drives and external storage devices

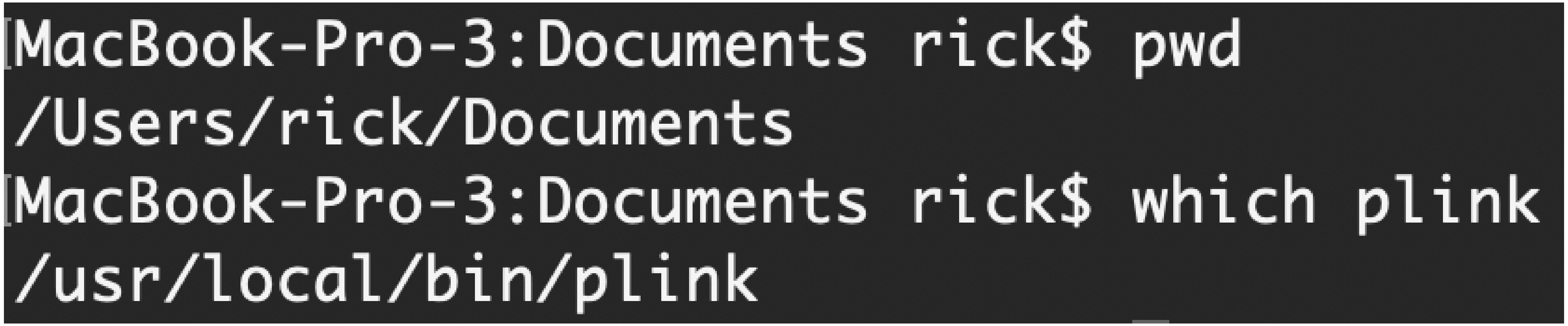
PATH

- **PATH:**
  - Environmental variable dictating file or folder location
  - In which directory should I look for a file/executable?
  - Example: /home/richard.flamio@usca.edu/DIR

- Absolute PATH = complete location information from the root directory
  - Example: /home/richard.flamio@usca.edu/DIR
- Relative PATH = location information relative to the current working directory
  - Example: /richard.flamio@usca.edu/DIR

PATH continued

- pwd = print current working directory
- Want to know where a software is installed? Use “which”

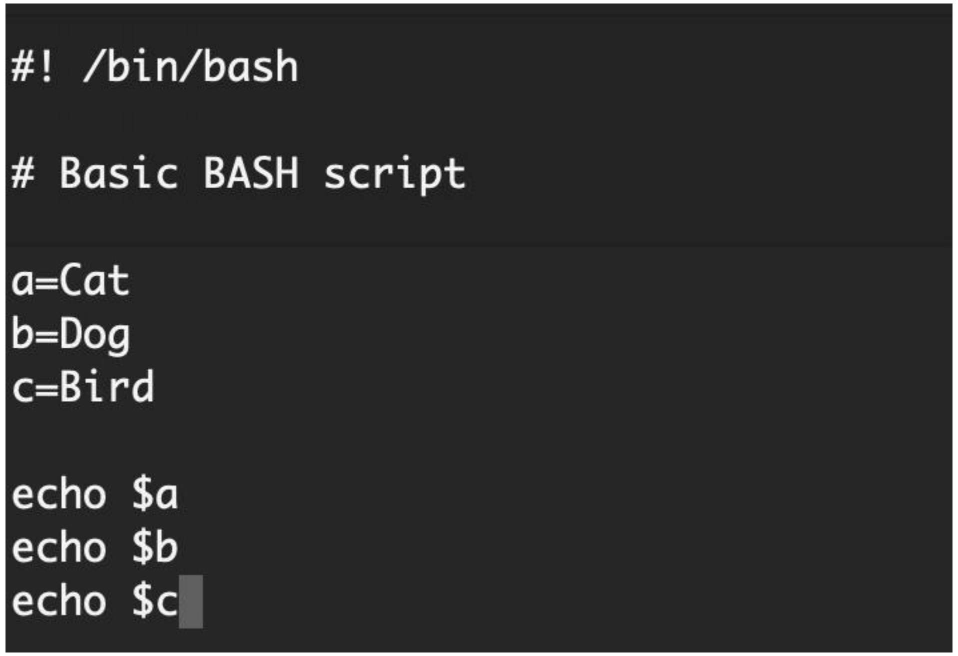


```
[MacBook-Pro-3:Documents rick$ pwd
/Users/rick/Documents
[MacBook-Pro-3:Documents rick$ which plink
/usr/local/bin/plink
```

Executables

Executable:

- File that contains a program or script to run
- In Linux, some common extensions
  - .sh = bash shell script
  - .pl = Perl-program script
  - .py = python script



```
#! /bin/bash

# Basic BASH script

a=Cat
b=Dog
c=Bird

echo $a
echo $b
echo $c
```

Directory Operations

Command	**Description**
pwd	Print current working directory
mkdir *name*	Make new directory
cd *directory*	Change directory (make sure it’s in your path)
ls	List files in a directory
man *command*	Check manual for command information including options

Different ways to use cd

Command	Description
cd ..	Change path to one directory up
cd ../..	Change path to two directories up
cd *data/RNAseq*	Change directory using relative path
cd */home/data/RNAseq*	Change directory using absolute path

Tab key
1x autocomplete path (if unique)
2x show all path options

ls command options

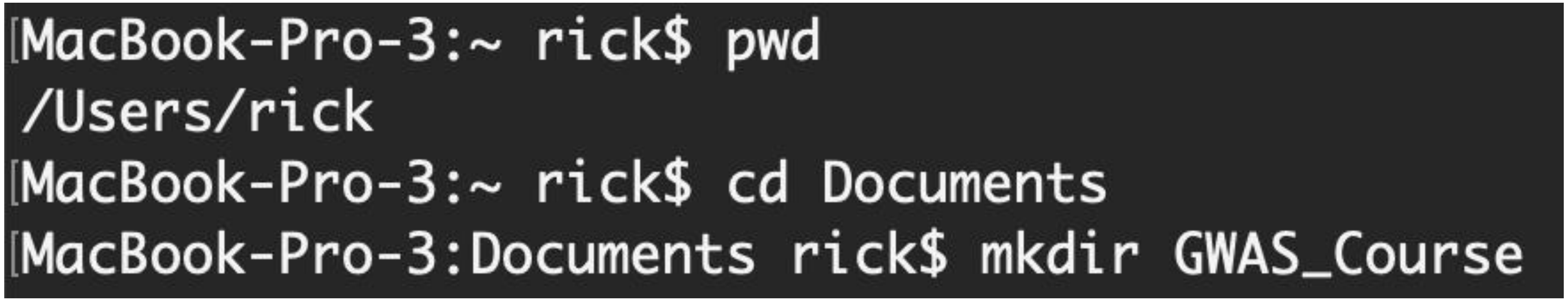
Command	Description
ls	List files in a directory
ls -a	Show all files including hidden files (e.g., .bashrc)
ls -l	Show files in long format with more details
ls -lh	Shows files with file sizes in human readable format
ls -lr	Show files in long format and in reverse order
ls -lt	Show files in long format sorted by time

Directory Operations Exercise

Exercise:

1. Open your command prompt.
2. Print your current directory.
3. Move to a directory where you want to add files (for example, within Documents).
4. Make a new directory named ‘GWAS_Course’.

Exercise Answer



```
[MacBook-Pro-3:~ rick$ pwd
/Users/rick
[MacBook-Pro-3:~ rick$ cd Documents
[MacBook-Pro-3:Documents rick$ mkdir GWAS_Course
```

File Operations

Command	**Description**
cat *file1*	View file in command line
touch *file2*	Create a new file
cp *file1 file2*	Copy file
cp *file1 ../*	Copy file to one directory above
head *file1*	Print first 10 lines of a file
mv *file1 file2*	Move/rename file
mv *file1 ../file2*	Move file up one directory
nano *file1*	Read and edit file
rm *file1*	Remove file
rm –r *directory*	Remove directory (and all contents)
tail *file1*	Print last 10 lines of a file

Multiple ways to create a new file

Some examples:

Method 1: touch *file1.txt*

Method 2: cat > *file1.txt* → add text → press Control D to return Command prompt

mv command options

Command	**Description**
mv *file1 file2*	Move/rename file
mv *file1 ../file2*	Move file up one directory
mv -i *file1 file2*	Move file and ask if file should be overwritten
mv -n *file1 file2*	Move file and do not overwrite file
mv -f *file1 file2*	Move file and overwrite file

cp command options

Command	**Description**
cp *file1 file2*	Copy file
cp *file1* ../	Copy file to one directory above
cp -i *file1 file2*	Copy file and ask if file should be overwritten
cp -n *file1 file2*	Copy file and do not overwrite file
cp -f *file1 file2*	Copy file and overwrite file

Check files without opening

Command	Description
more *file1*	Look at file (scroll with Enter/Space, quit with q)
head –n *file1*	Show the first n lines
tail –n *file1*	Show the last n lines
tail -n +2 *file1*	Show last n lines excluding the last line

Word Count

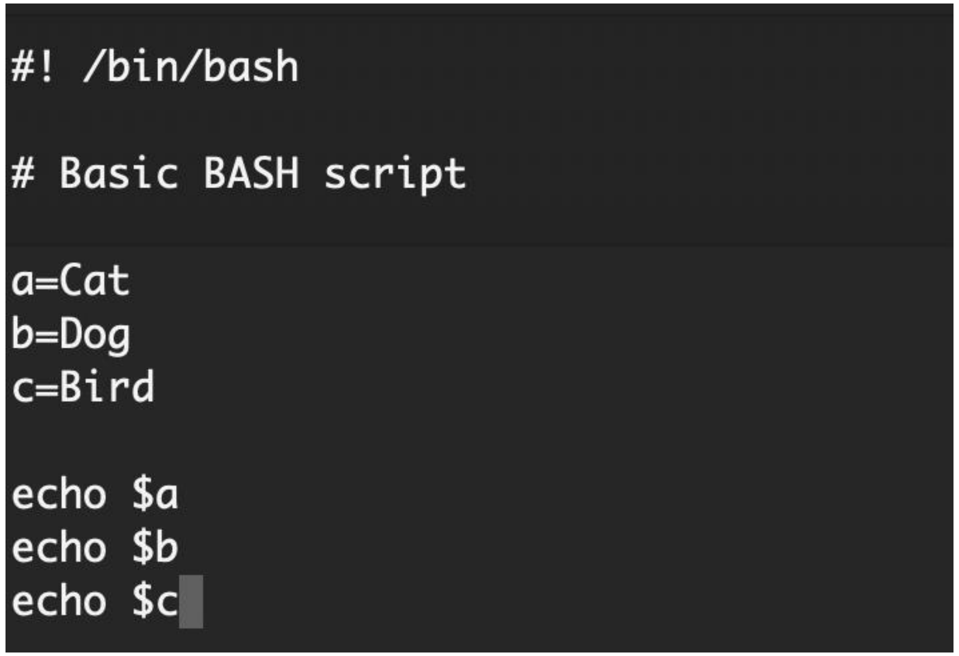
Command	**Description**
wc *file1*	Tells the lines, words, and characters (in order) in a file
wc –l *file1*	How many lines?
wc –w *file1*	How many words?
wc –m *file1*	How many characters?

File Operations Exercise

1. Move to the ‘GWAS_Course’ directory.
2. Create the file ‘polkadots.txt’ using `cat > polkadots.txt`.
3. Type ‘hello’. This will be appended to the file’s first line.
4. Press Control D to return to the command prompt.
5. List the files in the directory.
6. Read and edit ‘polkadots.txt’ and type ‘goodbye’ on the second line.
7. Save and exit. (Control X → Y → Enter).
8. Move up one directory and create the directory ‘Practice’.
9. Copy ‘polkadots.txt’ to this directory.
10. Within ‘Practice’, move ‘polkadots.txt’ to ‘stripes.txt’.
11. Print the last 10 lines of ‘stripes.txt’.
12. Move up to the parent directory that contains both ‘GWAS_Course’ and ‘Practice’.
13. Remove ‘Practice’ directory.

Making and executing a basic Bash Script

Basic Bash Script (.sh)



```
#! /bin/bash

# Basic BASH script

a=Cat
b=Dog
c=Bird

echo $a
echo $b
echo $c
```

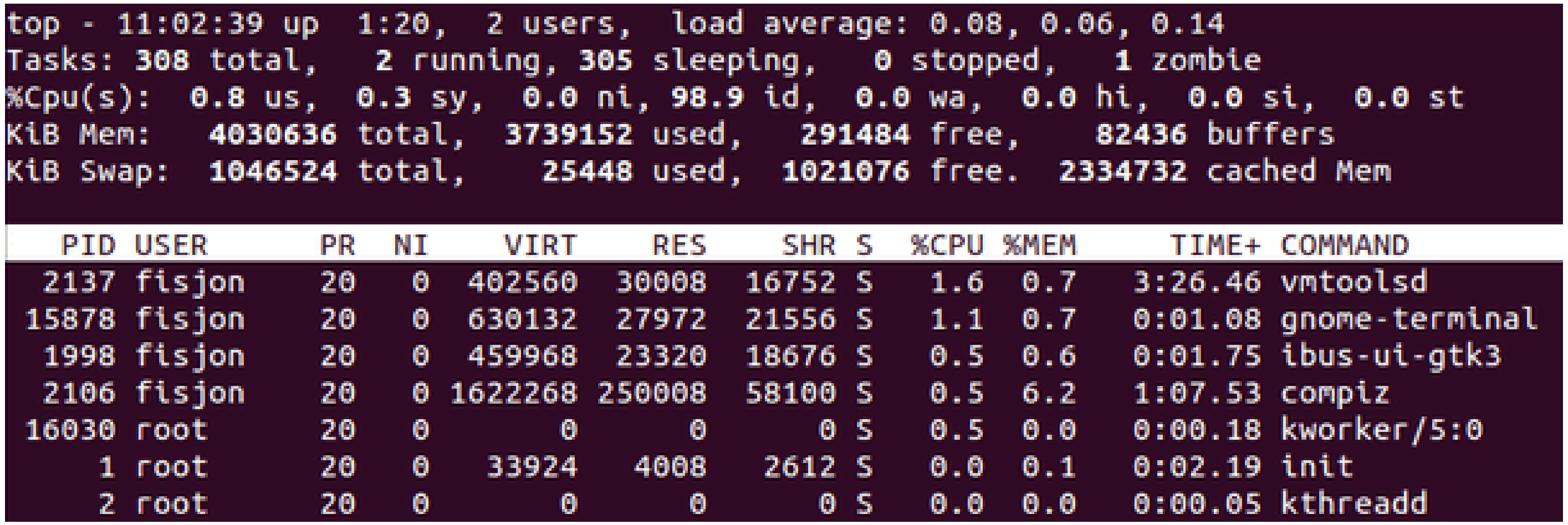
Command	**Description**
chmod u+x *name.sh*	Make file executable
./script.sh	Run script

Bash Script Exercise

1. Move to the ‘GWAS_Course’ directory.
2. Create the file ‘patterns.sh’.
3. Determine the path to bash.
4. Within the file, add the Shebang and tell the script to use the bash shell in the first line.
5. On line three, type ‘cp polkadots.txt paisley.txt’.
6. Save and exit file.
7. Make the script executable.
8. Run the script.
9. List the files to make sure it worked. You should have two files in the directory: ‘polkadots.txt’ and ‘paisley.txt’.

Top Command

Command	Description
top	Show processes
q	Exit top
kill *pid*	Kill process using PID



```
top - 11:02:39 up  1:20,  2 users,  load average: 0.08, 0.06, 0.14
Tasks: 308 total,   2 running, 305 sleeping,   0 stopped,   1 zombie
%Cpu(s):  0.8 us,  0.3 sy,  0.0 ni, 98.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   4030636 total,  3739152 used,   291484 free,    82436 buffers
KiB Swap:  1046524 total,    25448 used,  1021076 free.  2334732 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 2137 fisjon    20   0  402560  30008  16752 S   1.6  0.7   3:26.46 vmtoolsd
15878 fisjon    20   0  630132  27972  21556 S   1.1  0.7   0:01.08 gnome-terminal
 1998 fisjon    20   0  459968  23320  18676 S   0.5  0.6   0:01.75 ibus-ui-gtk3
 2106 fisjon    20   0 1622268 250008  58100 S   0.5  6.2   1:07.53 compiz
16030 root      20   0       0      0      0 S   0.5  0.0   0:00.18 kworker/5:0
    1 root      20   0   33924   4008   2612 S   0.0  0.1   0:02.19 init
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.05 kthreadd
```

Screen

use when you want to run a long process in the background and are afraid of accidently terminating

Command	**Description**
screen -S *name*	Create a new screen with a given name
Control A + D	Detach from screen
screen -ls	List screens
screen -r *name*	Reattach a screen
kill *pid*	Kill screen using PID

Detaching = running in background
Reattaching = bring to foreground

Compressing and uncompressing files

use to reduce space needed to store files

Command	Description
gzip *file1*	Compress a file, adds .gz extension
gunzip *file1*	Uncompresses a file with a .gz extension

Connecting to a remote server and transferring files

use when you want to connect to a remote server and transfer files between your local computer and the server

Command	**Description**
ssh *user_name@host*	Connect to remote server
scp ./*file user_name@host*:./	Copy local file to server
scp *user_name@host:/home/user_name/file.txt ./*	Copy file on server to local directory

10 minute break

Useful command-line utilites

- grep = global regular expression print
  - Searches input files for a search string and prints matching lines
  - grep “*string*” *filename*

- sed = stream editor
  - Efficiently performs text transformations such as substitutions (search and replace)
  - sed –e ‘s/*originaltext*/*newtext*/’ *filename*
  - Change every occurrence using a greedy search
    - sed –e ‘s/*originaltext*/*newtext*/g’ *filename*
  - Overwrite the original file
    - sed –i –e ‘s/ *originaltext*/*newtext*/’ *filename*

- Use ‘>’ to direct the output to a new file
  - Follow grep or sed commands with this symbol and the name of the new file
    - grep “*string*” *filename* > *newfilename*

grep Exercise

1. Produce a file ‘an.txt’ with the following text on separate lines: ant, anteater, dinosaur, and, andover, cranium, antebellum, argyle, andes.
2. Count the number of words in the text file.
3. Count the number of characters in the text file.
4. Use grep to print every line that contains the string ‘an’ in the file.

Other grep options

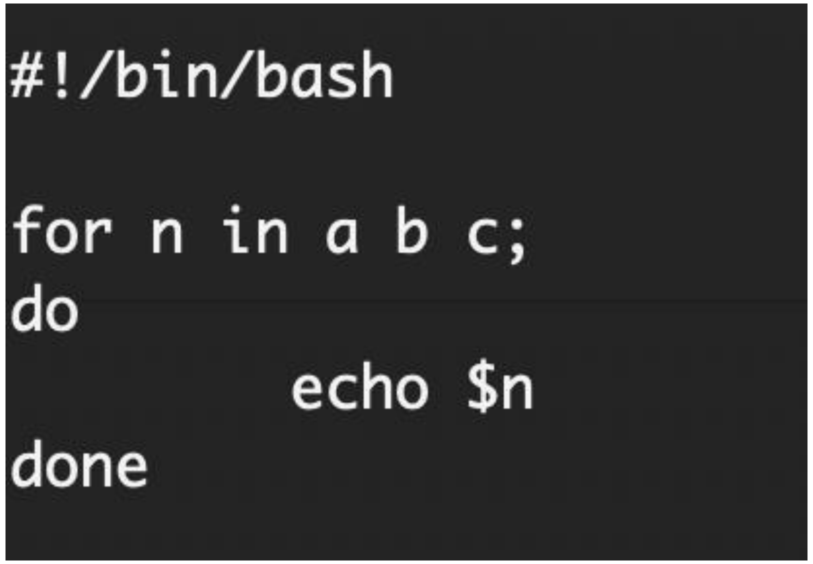
Options	Description
grep –n “*string*” *filename*	Which lines matched the string
grep –vn “*string*” *filename*	Which lines do not match the string
grep –c “*string*” *filename*	How many lines match the string
grep –l “*string*” *	Which files contain the string
grep –i “*string*” *filename*	Print lines ignoring case
grep –x “*string*” *filename*	Print exact matches

Loops

- Code that allows you to replicate a command multiple times until a condition is met
- For loops (simple, range-based, array iteration, c-style, infinite)
  - Use when you need to perform the same function on a list of items
- While loop
  - Use when you need to check a condition at the start of each loop to continue running the program
    - In other words, runs until condition is false

Simple For loop

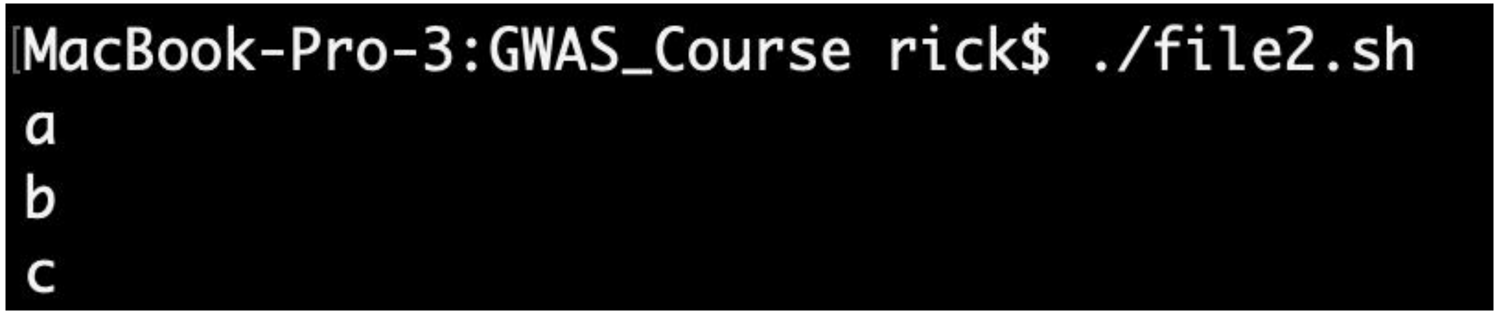
Input:



```
#!/bin/bash

for n in a b c;
do
        echo $n
done
```

Output:

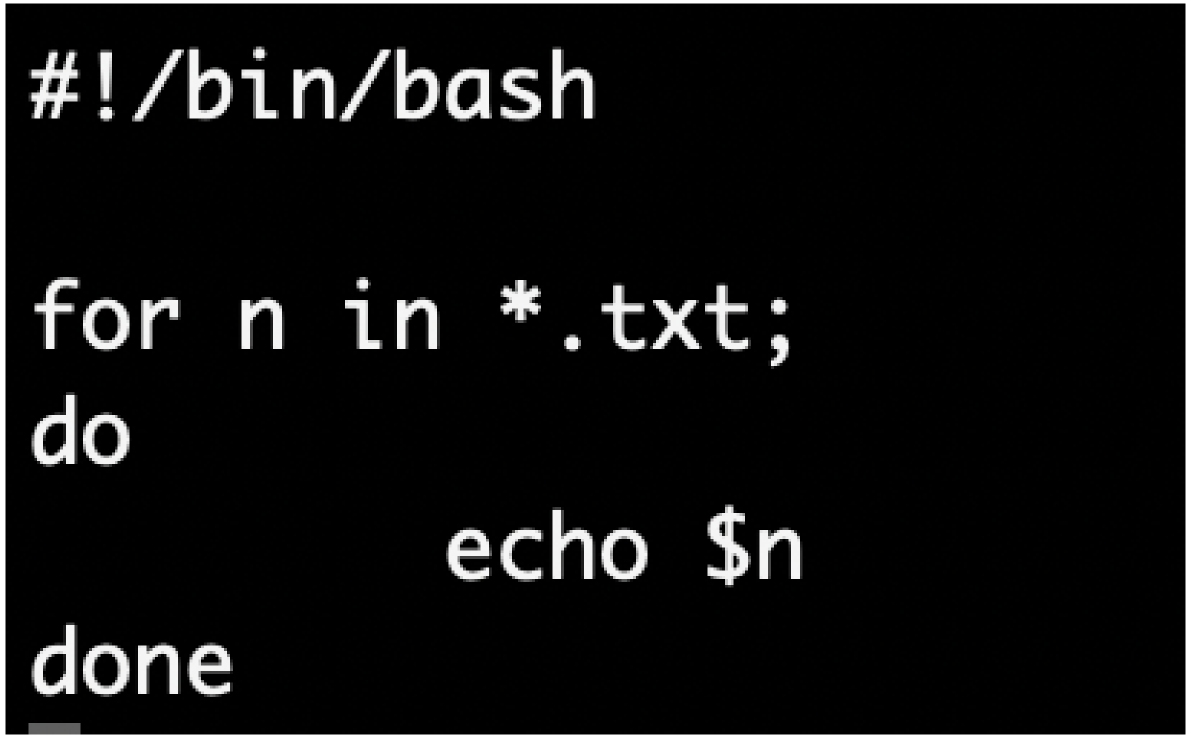


```
[MacBook-Pro-3:GWAS_Course rick$ ./file2.sh
a
b
c
```

For Loop Exercise

1. Navigate to the ‘GWAS_Course’ directory
2. Make a file called ‘stripes.txt’
3. Produce a script that runs through all the text files in the directory and prints their names as output.

Possible Answer

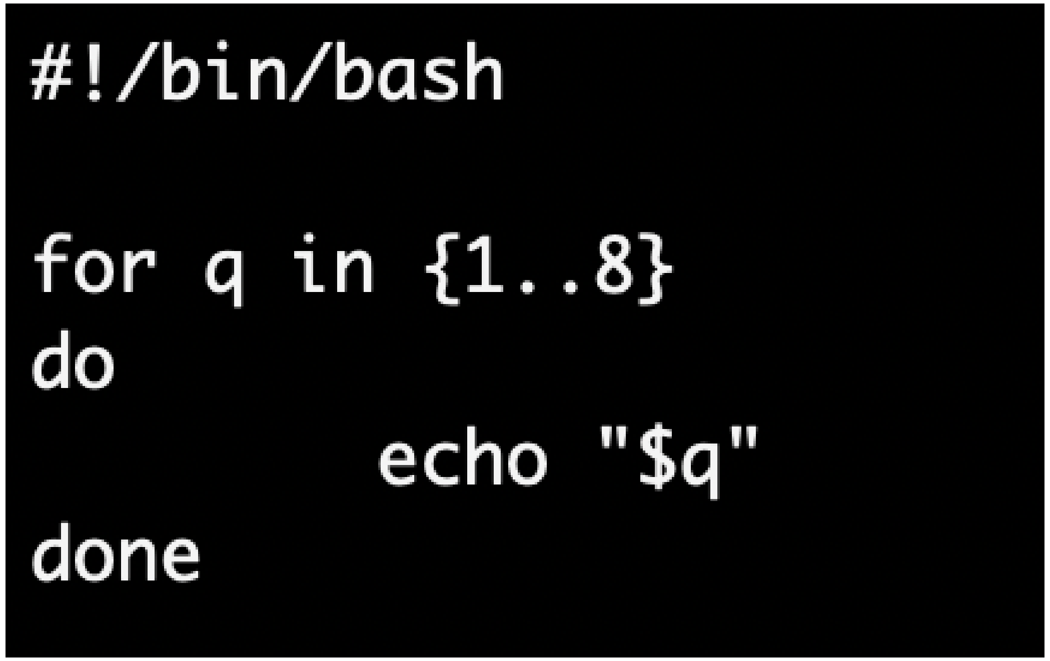


```
#!/bin/bash

for n in *.txt;
do
        echo $n
done
```

Range-based for loops

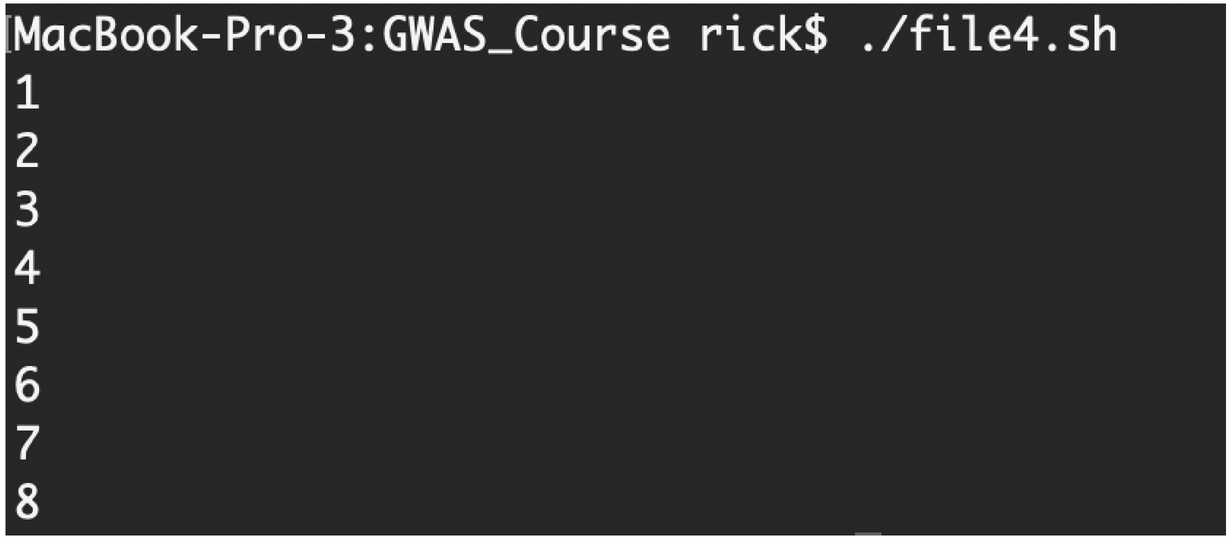
Input:



```
#!/bin/bash

for q in {1..8}
do
        echo "$q"
done
```

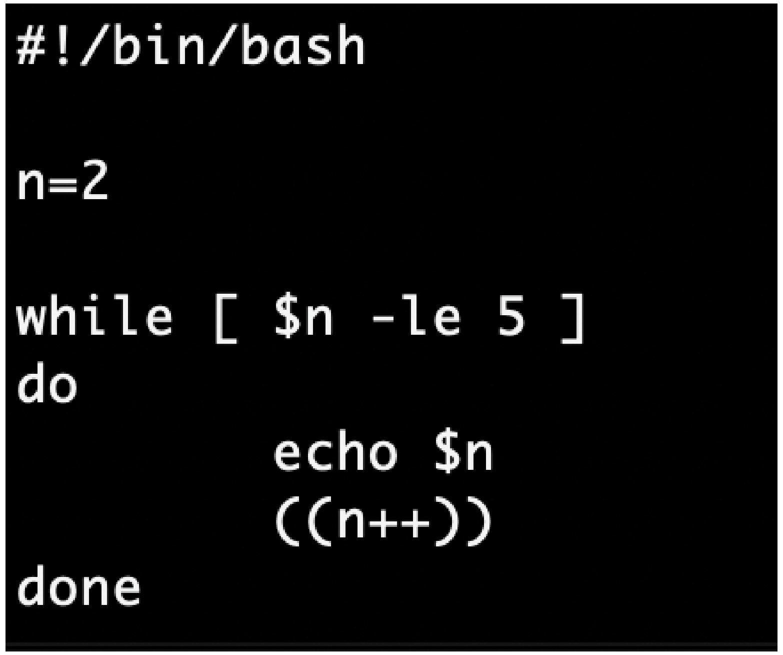
Output:



```
MacBook-Pro-3:GWAS_Course rick$ ./file4.sh
1
2
3
4
5
6
7
8
```

While Loop

Input:

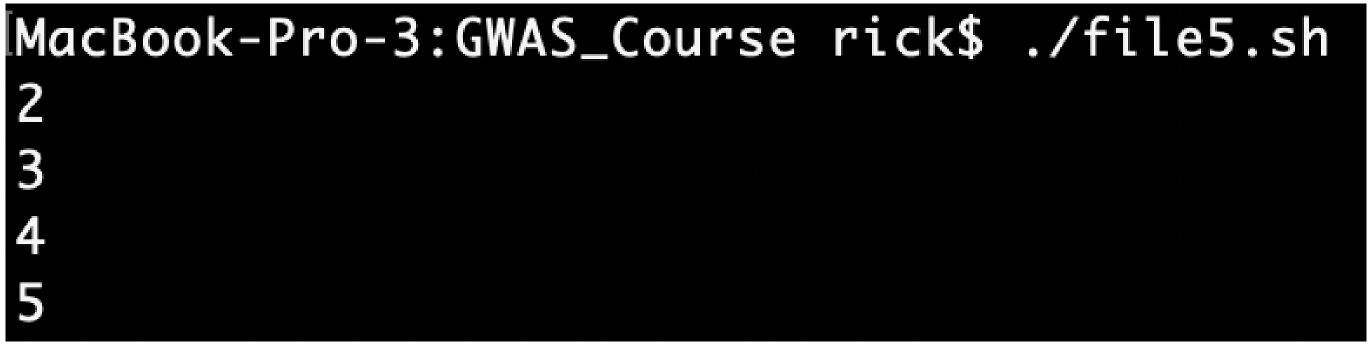


```
#!/bin/bash

n=2

while [ $n -le 5 ]
do
        echo $n
        ((n++))
done
```

Output:



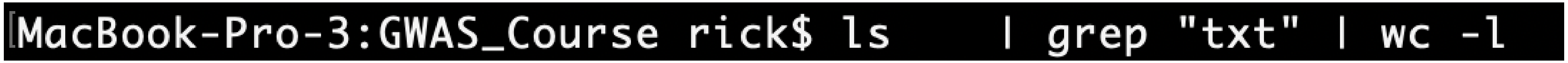
```
MacBook-Pro-3:GWAS_Course rick$ ./file5.sh
2
3
4
5
```

Piping

- Pipes allow you to process multiple functions in the same command.
- Example:
  - ls | grep “a”

Exercise: Create a command that contains two pipe operators and counts the number of text files in the directory ‘GWAS_Course’.

Piping Answer



```
[MacBook-Pro-3:GWAS_Course rick$ ls    | grep "txt" | wc -l
```

References

- https://www.geeksforgeeks.org
- https://www-users.york.ac.uk/~mijp1/teaching/2nd_year_Comp_Lab/guides/grep_awk_sed.pdf
- https://www.howtogeek.com/438882/how-to-use-pipes-on-linux/