An Introduction to the UNIX Shell

Logan Benninghoff

What is UNIX?

- UNIX is an operating system, like Windows, or MacOS (based on UNIX), and LINUX (open-source UNIX).
- How do we interact with the operating system?
- We can use a “shell”, which is a “Command Line Interpreter” (CLI).
- We use a command line because many tasks are easier to automate via text commands. Computing servers also often use some form of LINUX.

The BASH Shell

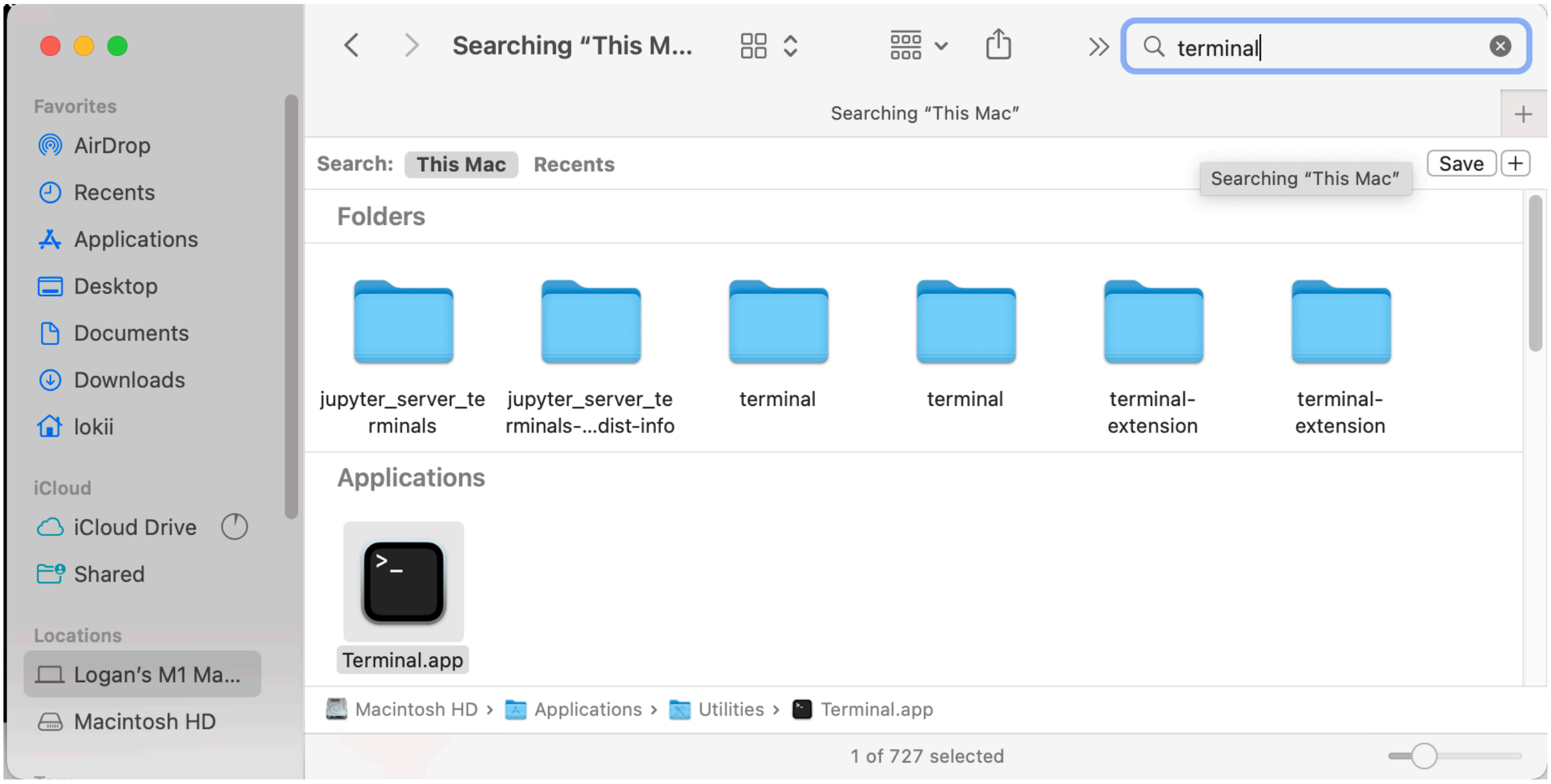
- The most common UNIX shell is “bash”, which is also a programming language.

- On a Mac, bash commands are run on the Terminal (search for Terminal.app in Spotlight/Finder).

- Windows does not natively support bash commands, we have to install a program that can interpret these. (“wsl --install”, then “wsl ~”)

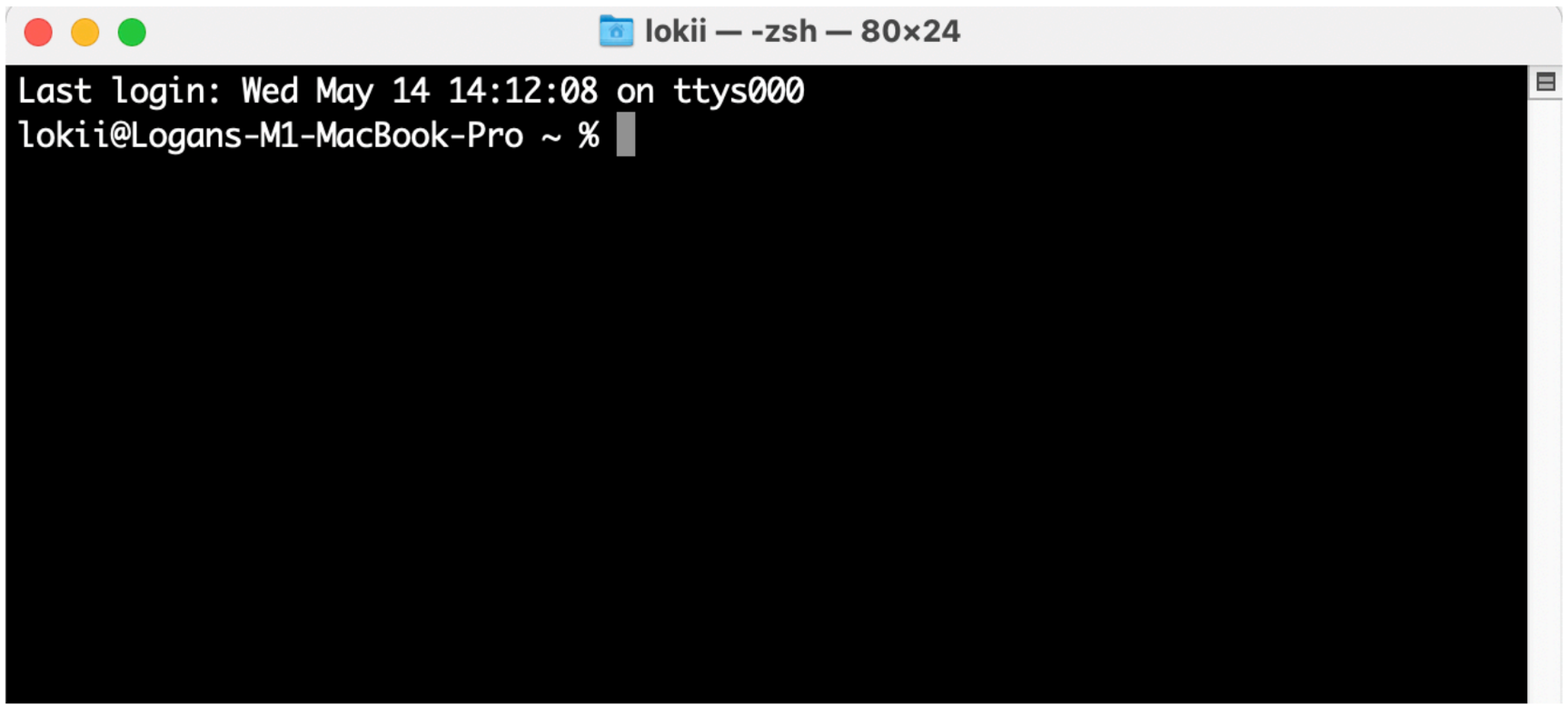
- For chromebooks, there is a pre-installed terminal app, or a terminal can be opened with Jupyter.

Open Your Command Line



Open Your Command Line

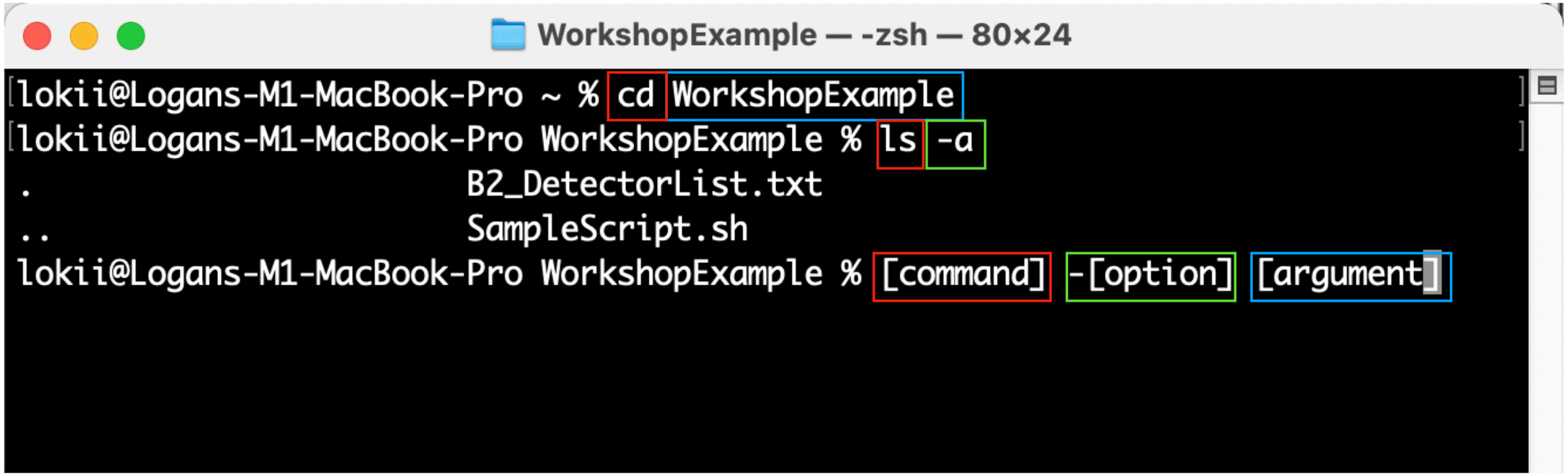
Once the command line is open, your computer indicates that it’s waiting for an input with a prompt character. For me, this is “%”; a dollar sign “$” is also commonly used.



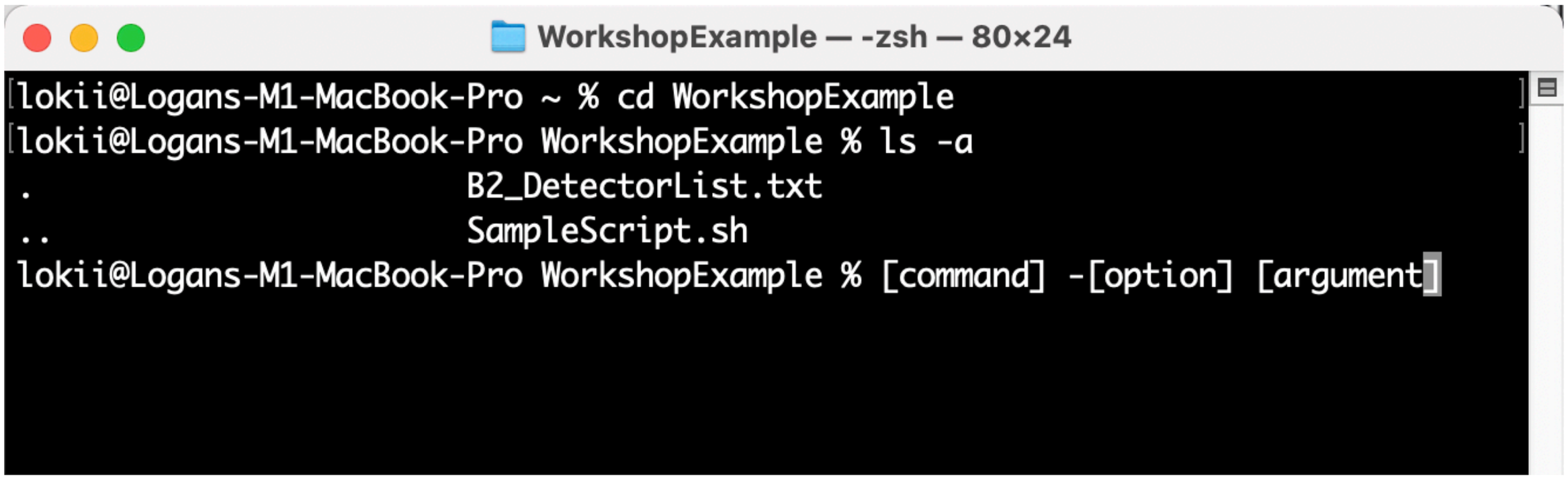
BASH Commands

- Many bash commands are abbreviations of what the command does (helps with remembering them).
- For example “cd” is the command to change directory.
- Commands have an “argument”, given after a space. For example, if we want to change directory, we need to say where we are changing to.
- There are also command line options, which alter how a command line runs, indicated by a hyphen “-”, then a letter i.e. ~ “-a”

BASH Command Example



BASH Command Example

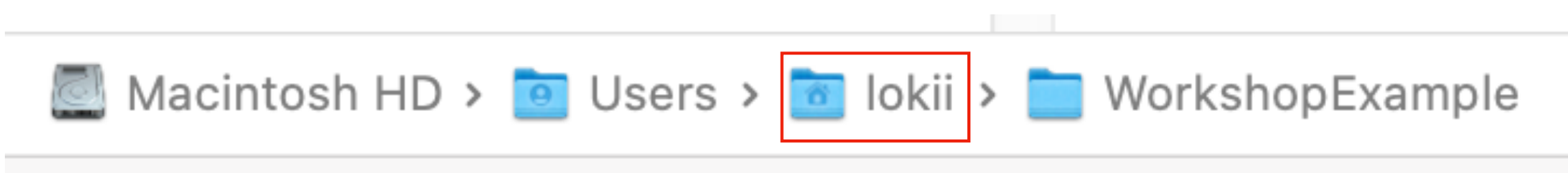


File System Navigation

- Information is stored within files on a computer, and are organized into directories (folders) which contain files/directories.

- The “home” directory is usually your starting point when opening the command line, and the highest directory accessed by a user.

- For example, my home directory,



File System Example

Directories can be thought of as a vertical stack. Here the “highest” directory is “WorkshopExample”.

Files are indicated by a file extension (“.txt” or “.sh” for example), while directories have no extension.

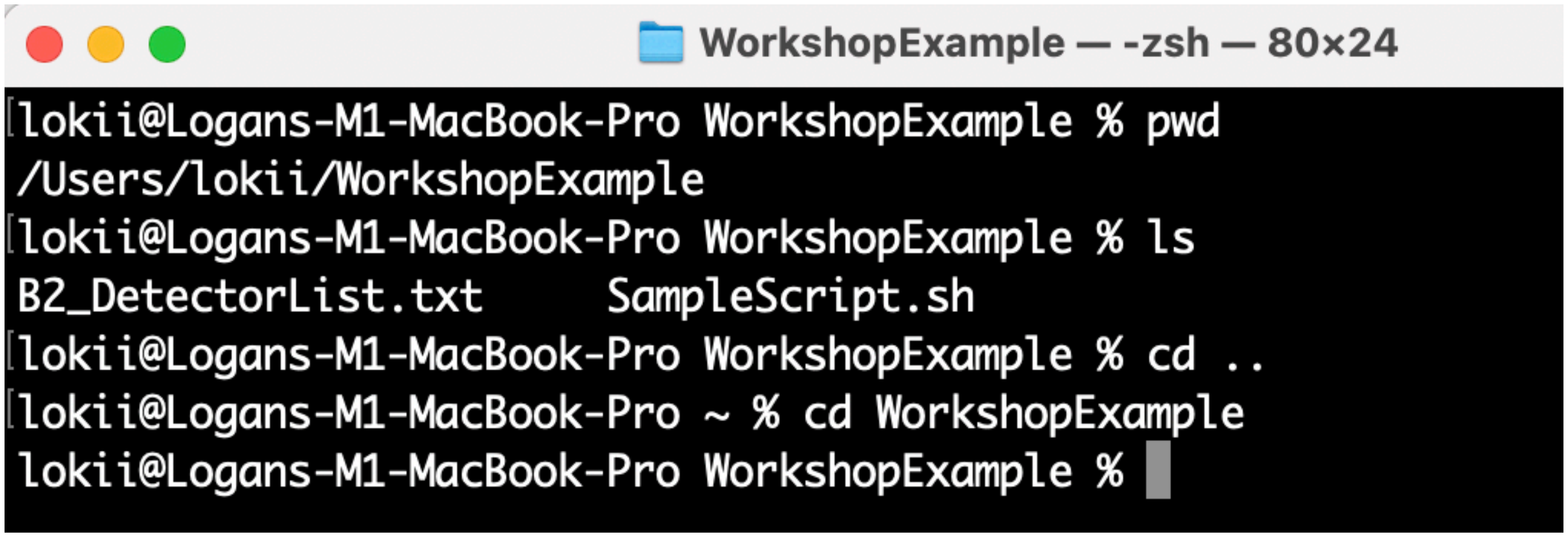
```
WorkshopExample
├── B2_DetectorList.txt
├── Notes
│   ├── CommandList.txt
│   └── LiveNotes.txt
└── SampleScript.sh

2 directories, 4 files
```

Navigation Commands

- pwd ~ “print working directory” ~ Outputs the directory you are currently in to the command line.
- ls ~ “list show”/“listing” ~ Shows all files and directories that are inside the current directory.
- cd [...] ~ “change directory” ~ Moves you to a directory that you list. If you don’t give an argument, it takes you to the home directory.
- Note ~ bash has shorthand for the directory above your current directory, “..” and the current directory, “.” You can use these with cd to navigate quickly.

Navigation Example



- Line by line, what am I doing here?

Task 1: Navigating

1) Print your current working directory.

2) Look at what files/directories are inside your working directory.

3) Pick a directory in that list, then move into that directory.

4) Return to the directory you started in.

- Remember commands pwd, ls, cd. If you finish quickly, you can use the option “--help” to see more info about command options. ~ i.e. “ls --help”

Creating Directories

- We now want to produce new directories to work with.

- mkdir [...] ~ “make directory” ~ Creates a directory with the name you give as the argument.

- Don’t use spaces or start the name with “-”, since these affect the command output.

- If you need to refer to a file with spaces in it, use single quotes ‘…’

Creating Directories

- rmdir […] ~ “remove directory” ~ Deletes the empty directory given as an argument.

- ! Warning ! ~ rmdir will not ask you for confirmation. Once you enter the command, that directory will be deleted, so be sure beforehand.

- There is a separate command for deleting files, which we use to delete a directory and its contents.

- rm -r […] ~ “remove” -“recursive” ~ Deletes all files and directories within the directory, step by step.

Detour: Paths

- So far, we've only used commands that operate inside our working directory, but we don't have to.
- If we want to reference a file outside our working directory, we have to tell the command where it is.
- Generally the "path" starts at your user, then includes each directory below it, separated by forward slashes, "/"
- /Users/lokii/WorkshopExample/Notes

Detour: Paths

- Your home directory can be abbreviated as “~/” so for me this would be “/Users/lokii”

- Remember that “.” indicates your current directory, so “./” refers to the path to your current directory.

- To create the directory “Remote” inside WorkshopExample (a directory that we are not in),

- mkdir ~/WorkshopExample/Remote

```
lokii@Logans-M1-MacBook-Pro ~ % mkdir ~/WorkshopExample/Remote
lokii@Logans-M1-MacBook-Pro ~ % ls ~/WorkshopExample
B2_DetectorList.txt     Remote
Notes                   SampleScript.sh
lokii@Logans-M1-MacBook-Pro ~ %
```

Task 2: Creating Directories

1) Start in your home directory, and make a new directory named “Workshop”.

2) Move into the Workshop directory and make a new directory named “Local”.

3) Return to the home directory, and from there, create a directory named “Remote” inside of Workshop.

4) Delete Remote from outside the Workshop directory, then move into Workshop and delete Local.

Commands
Navigate: pwd, ls, cd
Directories: mkdir, rmdir

Task 2: Completed

```
[lokii@Logans-M1-MacBook-Pro ~ % mkdir Workshop
[lokii@Logans-M1-MacBook-Pro ~ % cd Workshop
[lokii@Logans-M1-MacBook-Pro Workshop % mkdir Local
[lokii@Logans-M1-MacBook-Pro Workshop % ls
 Local
[lokii@Logans-M1-MacBook-Pro Workshop % cd
[lokii@Logans-M1-MacBook-Pro ~ % mkdir ~/Workshop/Remote
[lokii@Logans-M1-MacBook-Pro ~ % ls ~/Workshop
 Local    Remote
[lokii@Logans-M1-MacBook-Pro ~ % rmdir ~/Workshop/Remote
[lokii@Logans-M1-MacBook-Pro ~ % ls ~/Workshop
 Local
[lokii@Logans-M1-MacBook-Pro ~ % cd Workshop
[lokii@Logans-M1-MacBook-Pro Workshop % rmdir Local
[lokii@Logans-M1-MacBook-Pro Workshop % ls
 lokii@Logans-M1-MacBook-Pro Workshop %
```

Commands

Navigate:
pwd, ls, cd

Directories:
mkdir,
rmdir

Creating Directories II

- mv [old] [new] ~ “move” ~ Moves a directory/file from the old file location to the new file location.

- Note ~ We can use mv to rename files, we put the old name in [old], and the new name in [new].

- cp [old] [new] ~ “copy” ~ Makes a copy of the directory/file from the old location in the new location.

- Note ~ When we’re transferring files between different machines we use scp ~ “secure copy”

Creating (Empty) Files

- touch […] ~ Creates an empty file with the argument name given (make sure you have a file extension, such as .txt).

- We do not always have to start with an empty file, we can insert some text in at creation.

- cat […] ~ con-“cat”-tenate ~ Prints the text inside the file or files (separated by spaces) given as the argument.

Creating Files

- A command output can be stored in a new file when it is made using “>”, which means redirect.

- cat > […] ~ Will bring up a line for you to enter text that will then be put into the new file name you gave as the argument.

- Once you have typed your text, you can hit return to go to a new line, then “ctrl-d” to finish your input.

- Note ~ “>” will overwrite an existing file if you reuse a file name. Using “>>” appends text to the end.

Editing Files

- A text editor is needed to make changes to files. The easiest to start with is NANO, but more powerful options exist, like VIM or EMACS.

- nano [file.txt] ~ Opens the nano text editor where you can make changes to your file.

- Nano has a bottom bar that lists what many commands do. Keep in mind that “^” stands for “ctrl” in these.

Handling Many Files

- We can use commands on multiple files at once by using wildcards, “*” and “?”.
- When placed in the name of a file, * stands for one or more characters, and ? stands for only one character. What do I mean by this?
- If I have two files, RedApple.txt and GreenApple.txt, if I want to see the text in both I type,
- cat *Apple.txt ~ shows text inside both files.

Task 3: Writing Files

1) In “Workshop”, create directories “Notes” and “B2Info”.

2) In Notes, create “Detectors.txt” and fill it with names/abbreviations you can remember for detectors that we’ve learned so far.

3) Move this file from Notes to B2Info, then make a copy of Detectors.txt from B2Info back over to Notes, and call it “DetectorList.txt”.

Commands
Navigate: pwd, ls, cd
Directories: mkdir, rmdir
Files: touch, cat, nano
Moving: mv, cp
Misc: *, ?, >, ~, /, ..

Task 3: Completed

```
lokii@Logans-M1-MacBook-Pro WorkshopExample % mkdir Notes B2Info
mkdir: Notes: File exists
mkdir: B2Info: File exists
lokii@Logans-M1-MacBook-Pro WorkshopExample % cd Notes
lokii@Logans-M1-MacBook-Pro Notes % cat > Detectors.txt
PXD
SVD
CDC
ECL
TOP
ARICH
KLM
lokii@Logans-M1-MacBook-Pro Notes % mv Detectors.txt ~/WorkshopExample/B2Info
lokii@Logans-M1-MacBook-Pro Notes % ls
CommandList.txt LiveNotes.txt
lokii@Logans-M1-MacBook-Pro Notes % ls ~/WorkshopExample/B2Info
Detectors.txt
lokii@Logans-M1-MacBook-Pro Notes % cp ~/WorkshopExample/B2Info/Detectors.txt ~/WorkshopExample/Notes/DetectorList.txt
lokii@Logans-M1-MacBook-Pro Notes % ls
CommandList.txt         DetectorList.txt        LiveNotes.txt
lokii@Logans-M1-MacBook-Pro Notes % cat DetectorList.txt
PXD
SVD
CDC
ECL
TOP
ARICH
KLM
lokii@Logans-M1-MacBook-Pro Notes %
```

Task 3.5: Writing Files

1) In Notes, make 6 empty text files with the letters used to represent the quarks (u,d,s,c,b,t) ~ for example, “u.txt”

2) Make a directory named “Quarks” inside of Workshop.

3) We want to move the 6 quark files over to the Quarks directory all at once, but we don’t want to move Detectors.txt with them. What’s the correct wildcard to do this?

Commands
Navigate: pwd, ls, cd
Directories: mkdir, rmdir
Files: touch, cat, nano
Moving: mv, cp
Misc: *, ?, >, ~, /, ..

Task 3.5: Completed

```
lokii@Logans-M1-MacBook-Pro Notes % touch u.txt d.txt s.txt c.txt b.txt t.txt
lokii@Logans-M1-MacBook-Pro Notes % ls
CommandList.txt         b.txt                   s.txt
DetectorList.txt        c.txt                   t.txt
LiveNotes.txt           d.txt                   u.txt
lokii@Logans-M1-MacBook-Pro Notes % mkdir ~/WorkshopExample/Quarks
lokii@Logans-M1-MacBook-Pro Notes % mv ?.txt ~/WorkshopExample/Quarks
lokii@Logans-M1-MacBook-Pro Notes % ls
CommandList.txt         DetectorList.txt        LiveNotes.txt
lokii@Logans-M1-MacBook-Pro Notes % ls ~/WorkshopExample/Quarks
b.txt   c.txt   d.txt   s.txt   t.txt   u.txt
lokii@Logans-M1-MacBook-Pro Notes %
```

Connecting to Computers

- ssh username@remotehost ~ “Secure Shell” ~ To connect to external databases or computing centers, ssh creates a UNIX shell where we can send commands to be run on that external device.

- ssh requires some setup, generally you generate a key pair, and give one of the keys to the external computer.

- Your key (and a corresponding password) identifies your computer when you connect to the external computer.

Wrap-Up

- This will serve as our introduction to UNIX, some of your basic needs are covered here.
- If you want to see all this in more depth, you can look up “The UNIX Shell - Software Carpentry” and follow along with those tutorials.
- There are also extra slides to this presentation, that list more of these commands on the right.

Commands
Navigate: pwd, ls, cd
Directories: mkdir, rmdir
Files: touch, cat, nano
Moving: mv, cp
Sort: sort, head, tail, cut
Loops: for [.] in [.], do … done, echo
Misc: *, ?, >, >>, ~, /, .., |, ${}

End

Extra Exercises ->

Sorting

- sort […] ~ Sorts the content of a file alphabetically.
- We can send the output of this into a new file with > or append it with >>.
- Caution: Do not send the output back into the original file given.
- We can use -n to tell sort to organize numerically.
- We can use head […] to output the first line of a file, or -n 4 to output the first 4 for example. (tail gives last lines).

Removing Entries

- cut -d , -f 2 [...] ~ “cut” -“delimiter” , -“field” 2 ~ Deletes segments from a file.

- In this case, we are saying the entries are separated by a comma, the delimiter, and that you should delete the second field of that type.

- I.e. ~ This would delete the second column of a file that is organized with commas (a .csv file).

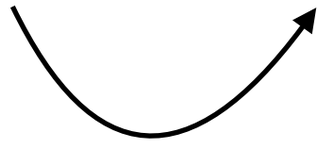
Pipes

- We can pass the output of a command to the input of another command using a “pipe”, “|”.

- sort […] | head ~ will give you the first line of the now sorted file […]. (Note that it does not rearrange the original file, since we didn’t save this output.)

- Think of this as,

- sort [file.txt] | head […]



Task 4: Pipes and Output

1) In Notes, sort Detectors.txt alphabetically, then save the output to a file called “DetectorsSorted.txt”.

2) Take the first two lines of DetectorsSorted.txt and save them to a file named “subset.txt”.

3) Sort Detectors.txt alphabetically, take the first two lines, and append them to subset.txt, all in one line of commands.

4) Delete “subset.txt”.

Commands
Navigate: pwd, ls, cd
Directories: mkdir, rmdir
Files: touch, cat, nano
Moving: mv, cp
Sort: sort, head, tail, cut
Misc: *, ?, >, >>, ~, /, .., |

Task 4: Completed

```
lokii@Logans-M1-MacBook-Pro Notes % sort DetectorList.txt > DetectorsSorted.txt
lokii@Logans-M1-MacBook-Pro Notes % cat DetectorsSorted.txt
ARICH
CDC
ECL
KLM
PXD
SVD
TOP
lokii@Logans-M1-MacBook-Pro Notes % head -n 2 DetectorsSorted.txt > subset.txt
lokii@Logans-M1-MacBook-Pro Notes % cat subset.txt
ARICH
CDC
lokii@Logans-M1-MacBook-Pro Notes % sort DetectorList.txt | head -n 2 >> subset.txt
lokii@Logans-M1-MacBook-Pro Notes % cat subset.txt
ARICH
CDC
ARICH
CDC
lokii@Logans-M1-MacBook-Pro Notes % rm subset.txt
lokii@Logans-M1-MacBook-Pro Notes % ls
CommandList.txt         DetectorsSorted.txt
DetectorList.txt        LiveNotes.txt
lokii@Logans-M1-MacBook-Pro Notes %
```

Loops

- We want to run a command multiple times without retyping it.

- for [X] in [list of X's] ~ The loop will run one time for each item in [list of X's], and $[X] will have that "value" for that cycle of the loop.

- This command will open a prompt where you will type do on the first line, then write the commands you want to run, then type done on the last line.

- Note ~ $ indicates a variable.

Loops

```
lokii@Logans-M1-MacBook-Pro Quarks % ls
b.txt   c.txt   d.txt   s.txt   t.txt   u.txt
lokii@Logans-M1-MacBook-Pro Quarks % for quark in d u s c b t
for> do
for>     rm ${quark}.txt
for> done
lokii@Logans-M1-MacBook-Pro Quarks % ls
lokii@Logans-M1-MacBook-Pro Quarks % for quark in d u s c b t
for> do
for>     touch ${quark}.txt
for> done
lokii@Logans-M1-MacBook-Pro Quarks % ls
b.txt   c.txt   d.txt   s.txt   t.txt   u.txt
lokii@Logans-M1-MacBook-Pro Quarks %
```

- Note ~ I had to use ${…} here to show bash where my variable started and ended.

Useful Loop Components

- echo [text] ~ Outputs whatever was typed in [text]. For variables, instead of saying the variable name, it gives the value of the variable.

- When looping over integers, we can shorten the listing of numbers 1 2 3 4 5 6 7 to {1..7}

- Note ~ Here the first and last number given in {1..7} both run a cycle of the loop.

- Note ~ Indexing differs in programming languages. Python indexes start at 0 when not specified.

Task 5: Loops

1) In Notes, create an empty file named “numbers.txt” and a directory named “Experiment”.

2) Use a loop to fill numbers.txt text file with the numbers 1 to 50.

3) Use a loop to create 10 files inside of Experiment named “trialX.txt”, where X is a number 1-10, and fill each file with the first X lines of “numbers.txt”.

- For (3), this is a harder task, and keep in mind we are working in two directories.

Commands
Navigate: pwd, ls, cd
Directories: mkdir, rmdir
Files: touch, cat, nano
Moving: mv, cp
Sort: sort, head, tail, cut
Loops: for [.] in [.], do … done, echo
Misc: *, ?, >, >>, ~, /, .., |, ${}

Task 5: Completed

```
lokii@Logans-M1-MacBook-Pro Notes % mkdir Experiment
lokii@Logans-M1-MacBook-Pro Notes % touch numbers.txt
lokii@Logans-M1-MacBook-Pro Notes % for n in {1..50}
for> do
for>     echo ${n} >> numbers.txt
for> done
lokii@Logans-M1-MacBook-Pro Notes % cat numbers.txt
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```

Task 5: Completed

```
49
50
lokii@Logans-M1-MacBook-Pro Notes % cd Experiment
lokii@Logans-M1-MacBook-Pro Experiment % for n in {1..10}
for> do
for>     head -n ${n} ~/WorkshopExample/Notes/numbers.txt > Trial${n}.txt
for> done
lokii@Logans-M1-MacBook-Pro Experiment % ls
Trial1.txt      Trial2.txt      Trial4.txt      Trial6.txt      Trial8.txt
Trial10.txt     Trial3.txt      Trial5.txt      Trial7.txt      Trial9.txt
lokii@Logans-M1-MacBook-Pro Experiment % cat Trial1.txt
1
lokii@Logans-M1-MacBook-Pro Experiment % cat Trial4.txt
1
2
3
4
lokii@Logans-M1-MacBook-Pro Experiment %
```

Scripts

- We've been typing all these commands into the command line manually, but what if we want to save a set of commands to run later?

- We can create a script, a ".sh" file.

- Scripts are run by typing bash script.sh

- This will run the commands that we've written inside of script.sh, which we can do using a text editor like nano.

Scripts

- We can tell bash that there are additional inputs when we run the script by putting $1, $2, etc. into our .sh file. Then when we run,

- bash script.sh [1] [2]

- Any time we wrote $1 or $2 in our script.sh file, it will be replaced with whatever we put as [1], [2].

- (!) Note ~ If we put a for loop inside our script, we have to write it as for [X] in [list]; then we put our do […] done section after the semicolon ";".

Scripts

- We can tell bash that there are additional inputs when we run the script by putting $1, $2, etc. into our .sh file. Then when we run,

- bash script.sh [1] [2]

- Any time we wrote $1 or $2 in our script.sh file, it will be replaced with whatever we put as [1], [2].

- (!) Note ~ If we put a for loop inside our script, we have to write it as for [X] in [list]; then we put our do […] done section after the semicolon ";".

Useful for Scripts

- When we need to do mathematical calculations, we can enclose the expression with $((…)).
- When we want a loop where the maximum number of cycles we want is controlled by a variable $VAR, we need to use this in our syntax.
- for ((i = 1; i <= $VAR; i++));
- Note ~ This is close to the syntax for loops in C.
- We can also create comments; by putting a “#” at the start of the line, bash will ignore following text.

Useful for Scripts

- We can create variables as needed to store info. Conventionally, variables are in all-caps. When referenced later, we put a $ sign in front of the name.
- i.e. ~ VAR = 3 -> Called later as $VAR or ${VAR}
- When we put a $ sign in front and enclosing parenthesis (…) around a command, bash runs that command in a subshell, then gives the output.
- VAR = “$(…)” ~ Double quotes are good practice, they are a precaution for multiline commands.

Task 6: Script

1) In Notes, create a script called "numbering.sh" that takes an input text file [input], numbers the lines of that text file, and outputs that to a new file.

2) Use this script on Detectors.txt

- We need wc -l […] for this, which gives the "word count", but -l means it outputs the number of lines in the file.
- We also need to select just one line from a file at a time. I will give you these pieces, go to next slide.

Commands
Navigate: pwd, ls, cd
Directories: mkdir, rmdir
Files: touch, cat, nano
Moving: mv, cp
Sort: sort, head, tail, cut
Loops: for [.] in [.], do … done, echo
Misc: *, ?, >, >>, ~, /, .., |, ${}

Task 6: Script

1) In Notes, create a script called "numbering.sh" that takes an input text file [input], numbers the lines of that text file, and outputs that to a new file.

2) Use this script on Detectors.txt

- Assorted useful pieces,
- for ((i = 1; i <= $VAR; i++));
- "$(wc -l ${1} | cut -d " " -f 8)"
- $(head -n ${i} ${1} | tail -n 1)

Commands
Navigate: pwd, ls, cd
Directories: mkdir, rmdir
Files: touch, cat, nano
Moving: mv, cp
Sort: sort, head, tail, cut
Loops: for [.] in [.], do … done, echo
Misc: *, ?, >, >>, ~, /, .., |, ${}

Task 6: Completed

```
  GNU nano 2.0.6                File: numbering.sh

# Numbers the text lines inside a file
# Two arguments, 1 ~ input file, 2 ~ Name of new file without giving file extension

touch ${2}.txt

MAXLINES="$(wc -l ${1} | cut -d " " -f 8)"
echo $MAXLINES

# The echo command above is for troubleshooting the output of $MAXLINES

for (( x = 1; x <= $MAXLINES; x++ ));
do
        echo "${x}. $(head -n ${x} ${1} | tail -n 1)" >> ${2}.txt
done
```

```
lokii@Logans-M1-MacBook-Pro Notes % nano numbering.sh
lokii@Logans-M1-MacBook-Pro Notes % bash numbering.sh DetectorList.txt DetectorNumbered
7
lokii@Logans-M1-MacBook-Pro Notes % cat DetectorNumbered.txt
1. PXD
2. SVD
3. CDC
4. ECL
5. TOP
6. ARICH
7. KLM
lokii@Logans-M1-MacBook-Pro Notes %
```

Searching

- grep [arg1] [arg2] ~ “get regular expression” ~ The command searches the files or group of files in a directory given as arg2 for the words or characters given in arg1.

- find [arg1] [arg2] ~ This searches the directory given as arg2 for the words given as arg1.

- Wildcards are often used with grep and find in searches.

Covered earlier (copied)

- ssh username@remotehost ~ “Secure Shell” ~ To connect to external databases or computing centers, ssh creates a UNIX shell where we can send commands to be run on that external device.

- ssh requires some setup, generally you generate a key pair, and give one of the keys to the external computer.

- Your key (and a corresponding password) identifies your computer when you connect to the external computer.

Wrap-Up (Repeated)

- This will serve as our introduction to UNIX, most of your basic needs are covered here.
- If you want to see all this in more depth, you can look up “The UNIX Shell - Software Carpentry” and follow along with those tutorials.

Commands
Navigate: pwd, ls, cd
Directories: mkdir, rmdir
Files: touch, cat, nano
Moving: mv, cp
Sort: sort, head, tail, cut
Loops: for [.] in [.], do … done, echo
Misc: *, ?, >, >>, ~, /, .., |, ${}