Unix Essential Training:

Kevin Skoglund, Linked In Learning

CHAPTER 1

INTRO:

→ Every time you open a new terminal window, it is starting a new session. You can have multiple sessions at once by opening multiple terminal windows
→ You can be logged in to multiple systems with different usernames, etc. Each window open is separate.

- Command ⇨ ***whoami*** → will tell you what user you are logged in as
- Command ⇨ ***who*** → reports list of all currently logged in user (all the windows open including GUI)
- Command ⇨ ***echo*** 'message' → outputs the quoted text

→ use up and down arrows to cycle through ***previous command history***
→ left and right arrows move along the input line
→ ***ctl+a*** moves cursor to start of line
→ ***ctl+e*** moves cursor to end of line
→ ***tab*** key will try to complete a command for you or a filename
→ ***clear*** (or ***cmd+k***) will clear screen and scroll back
→ ***exit*** to quit a session (or close window, or closer terminal through menu)

COMMANDS:

→ Command Structure:

- Three parts: command + options + arguments
- Command itself is always a single word
- Options are optional. They modify the default behavior of the command in some way.
    - EX: ***echo 'Hello World!"*** ⇦ echo is the command, 'Hello World!" is the argument. Quotes tell it that it is a single line of text.
    - ***echo -n 'Hello World!'*** ⇦ -n option will now suppress the new line return after the output
- Commands, options and arguments are separated by spaces.
- There are also options that do not take any arguments
    - Ex: ***make -v*** ⇦ this will report the version of the command that we have installed.
- Options are either a single dash followed by a single letter or double dash followed by keyword
    - Ex: ***make -v*** and ***make --version*** are the same commands with the same options
- You can also have multiple options at the same time
    - Ex: ***ls -l -a -h /usr*** ⇦ This will point to a directory
    - Order of options input does not matter
    - You can also smash them together, Ex: ***ls -lah /usr*** will do the same as the above
- Options that have their own arguments cannot be combined in the same way, however
    - Ex: ***banner -w 50 'Hello World!"*** ⇦ 50 is an argument for -w (how wide the banner should be)
    - These can also be smashed together if it makes it more clear:
        - Ex: ***banner -w50 'Hello World!"*** ⇦ creates a vertical banner, for continuous printing
- A command may take multiple arguments:
    - Ex: ***cat -n file1.txt file2.txt*** ⇦ the cat command is going to concatenate the two files, and the -n command tells it to number the lines in the output (option could have been left out)

KERNELS and SHELLS:

→ **kernel** ⇨ the core of the operating system in Unix, responsible for allocating time and memory to each program, does the fundamental management of how commands and programs conduct their work

→ **shell** ⇨ the outer layer of the operating system. It's what we see when we open up a terminal window.
→ We're working in the shell. It interacts with the user and we can think of it as our working environment.
→ the shell will send requests to the kernel. The kernel will then do its job and launch programs and then the results will be returned back to the shell so that we can interact with them again.
→ We have the ability to choose different shells. They're all similar but have some slightly different features.
→ The very first shell was called sh, known as the Thompson shell because it was created by Ken Thompson.
→ The bash shell has been one of the most popular shells for the last 30 years, and it's often the default shell.
→ It's also possible to switch shells while you're working and for different users on a multi-user system to each pick the shell that they like best
→ A lot of the difference between them is going to be the features that are available to high-end users for tasks like advanced shell scripting.
→ command: ***echo $SHELL*** will tell you the default shell you have set up, ***echo $0*** will tell the current shell
→ you can switch shells, for example, by command: ***bash***, which will automatically switch over to bash
→ There is also ***ksh,*** (korn shell), ***tcsh, csh*** ⇦ every time you "switch" shells, you are actually in the new shell inside the old shell, nested. The ***exit*** command will incrementally exit each nested shell.

UNIX Text Editors:

→ **vim** ⇨ still popular. Fingers can stay centered on keyboard and navigation is through various modes such as input mode, etc.
→ pico and nano are better for beginners. We will use nano

NANO:

- Command: ***nano*** ⇦ drops you into the nano text editor
- Commands with the ^ at the bottom indicate the ctl key
- Command: ***nano some_filename.txt*** ⇦ will open a specified file to edit.

→ Any text editor that stores information just as text can be used through Unix, but nothing that stores style information. Code editors will work.

CHAPTER 2

File System Basics:

→ current directory is called the **working directory**. Commands often work based on this directory unless otherwise specified.
→ command: ***ls*** ⇦ returns a list of the contents of the working directory. When you first log in, you will be in your user home directory by default.
→ command: ***pwd*** ⇦ returns path to your present working directory
→ command: ***ls -l*** ⇦ gives the same list of the contents of the directory, but now with additional info

```
torage/GoogleDrive-evancarrhomeschool@gmail.com
drwxr-xr-x  30 evancarr  staff         960 Oct  5 20:42 Homeschool Things
drwx------@ 88 evancarr  staff        2816 Aug 11 11:06 Library
drwx------   6 evancarr  staff         192 Jul 29 12:12 Movies
drwx------+  4 evancarr  staff         128 Jul 28 04:49 Music
drwx------+  4 evancarr  staff         128 Jul 14 16:26 Pictures
drwxr-xr-x+  4 evancarr  staff         128 Jul 14 16:25 Public
-rw-r--r--@  1 evancarr  staff      817484 Sep 22 12:29 insurance_cards.png
-rw-------   1 evancarr  staff  1414329825 Aug 16 18:05 java_error_in_pycharm.hprof
```

→ to get the file list of a directory other than the one you are in, ***ls /directory_to_list***
→ command ***ls -l -h*** ⇦ gives the same but with the file sizes in human readable format
→ including ***-a*** (or just adding ***a*** to options) will return ALL files in the director including . files (config or system files, usually not included)
→ The information in the first column tells about each file. ***d*** means directory. means file

```
(base) evancarr@EMC ~ % ls -lah
total 2764168
drwxr-x---+ 50 evancarr  staff   1.6K Oct  8 10:37 .
drwxr-xr-x   5 root      admin   160B Aug 24 04:59 ..
-r--------   1 evancarr  staff     7B Jul 14 16:25 .CFUserTextEncoding
-rw-r--r--@  1 evancarr  staff    14K Oct  7 18:28 .DS_Store
drwx------+  8 evancarr  staff   256B Oct  8 10:34 .Trash
-rw-r--r--   1 evancarr  staff   507B Jul 14 17:12 .bash_profile
drwxr-xr-x   3 evancarr  staff    96B Oct  7 14:34 .cache
```

→ the **dot . and ..** in the top two entries are not actually in this directory.
→ The first is a representation of the **current directory**, a way to see information about the current directory or address it in commands. .. is the **parent directory**
→ command: ***cd*** ⇦ change directory, navigate through file system
→ you can go into more than one directory by using / . Ex: ***cd Library/Preferences***
→ Use ***tab*** to autocomplete after the beginning of a command or directory
→ to go back more than one directory at a time, command: ***cd ../..***
→ These paths are relative to the current directory, but you can also use paths relative to the root directory of the hard drive, called ***absolute path*** by using a ***/*** at the beginning of the argument given to ***cd***

- Ex: ***cd /Users/evancarr/Library/Preferences***
- Command: ***cd ~*** will take you back to your home directory
- Command: ***cd -*** will take you back to the last directory you were in

Unix Filesystem

Directory	Description
/bin	Commands/programs
/etc	Configurations
/home	User home directories
/lib	System libraries
/tmp	Temporary files
/usr	Unix system resources
/var	Variable system data files

Creating Files:

→ filenames have a 255 character max, are case-sensitive and only accept period, hyphen and underscore
→ use underscores instead of spaces. Must use \ to escape spaces or quotes around names with spaces
→ file endings are not required but are still helpful
→ with nano, you can start a new file at the command line with command: ***nano new_filename***

Reading Files:

→ ***cat*** ⇦ short for concatenate. Given several arguments, it will read the file and concatenate them.

- If given only one file name, it will display the contents of that file in the terminal window and not perform any concatenation.
- It is the normal way in Unix to get the contents of a file.
- Works well with very short files, but not as much with longer ones.

→ ***more*** ⇦ original Unix command that returns paginated (in pages) output, rather than just lines and lines of output. It has recently been replaced with ***less***
→ ***less*** ⇦ includes backwards scrolling and better memory use

- Takes you to a viewer and shows you the contents of the file. Space bar will move down pages.
- ***F*** for forward, ***B*** for back, ***H*** for help, ***Q*** to exit

→ These commands can be used from any directory to access files in home directory by ***less ~/my_file.txt***

Creating Directories:

→ ***mkdir new_directory*** ⇦ make directory in the current working folder
→ ***mkdir /existing_directory/new_directory*** ⇦ creates a new directory inside of an existing folder in the current directory
→ ***mkdir -p existing_directory/new_directory/new_subdirectory*** ⇦ ***-p*** (creating a parent directory for the subdirectory you are creating) allows you to create multiple new and nested directories at one time.

Moving and Renaming Files and Directories:

→ ***mv file_to_move destination_directory*** ⇦ give the ***mv*** command to move, then filename to be moved, then directory to which it should go (You can also add the name of the file being moved after the directory it is being moved to, for clarity and to make sure you do not rename, since the same command is used to rename)

- Ex. ***mv file_to_move destination_directory/file_to_move***

→ RENAMING: use the same command as moving files, but give it a new file name inside of the directory to which it is being moved, Ex: ***mv other_file experimental/this_file***

- To rename a file in place: ***mv old_filename new_filename***

→ moving and renaming directories works the same way

mv Options

Option	Description
-f	Force overwriting (default)
-n	No overwriting
-i	Interactive overwriting, "ask me"

Copying Files and Directories:

→ **COPYING FILES**: ***cp lorem lorem2*** ⇦ made a copy of lorem, called lorem2, into the same / current directory, command cp + target + path to where to copy and / or filename

→ **COPYING DIRECTORIES**: requires one extra option, -R.

- Ex: ***cp -R new_directory new_directory2*** ⇦ made a copy of the directory into the same location
- ***-R*** is for recursive, meaning it will copy the directory, all its files, subdirectories, and their files as well

cp Options

Option	Description
-R	Recursive copy directories
-f	Force overwriting (default if empty)
-n	No overwriting
-i	Interactive overwriting, "ask me"

Deleting Files and Directories:

→ Not the same as putting things in the trash. When you delete a file or folder, it is gone forever.

→ ***rm delete_me*** ⇦ this deletes the file called "delete_me" forever. No recycling.

→ just as you use -R to copy a directory, you have to use it to delete a directory

- Ex: ***rm -R delete_directory*** ⇦

Creating Symbolic Links:

→ aka "sym link", similar to shortcuts as in a GUI, creating an object that refers to an object somewhere else in the file system, but the **difference** is that it **references a file path**, NOT a file itself.

- Consequently, the symlink is broken once a file is moved or deleted.

→ ***ln -s file_name symlink_name*** ⇦ you have to make sure you use the ***-s***. Otherwise you create a "hard link", which works very differently and is rarely used.

Searching for Files and Directories:

Wildcard Characters

Wildcard	Description
*	Zero or more characters (glob)
?	Any one character
[ ]	Any character in the brackets

→ find + path + expression, Ex:

→ will search all files and directories in that path

→ most common and basic expression is ***-name***

- Ex: ***find . -name "exact_filename"*** ⇦ will find a file within the current directory (. means within current directory) any file that matches the name in quotes.
- It returns a path to the file you have searched.
- Ex: ***find . -name "*rem"*** ⇦ returns any files with "rem" in the filename
- ***find . -name "f*"*** ⇦ returns anything with a filename starting with f.
- ***find . -name '*[fmt]ile'*** ⇦ returns anything with file, mile, or tile in the name of the file.

→ you can also search using regular expressions, by file size, by owner, as well as many other examples of meta data

CHAPTER 3

Users and Groups:

→ Unix is designed to be a multi-user environment, and ownership and permissions are important concepts in a multi-user environment. Each user on the system needs the ability to control access to their files by others

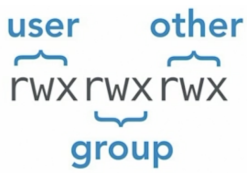
→ on a system, each user gets their own home directory, which is stored as a variable, visible with the command: ***echo $HOME***
→ Groups allow permissions to apply to a set or group of users. It allows semi-private group access to files and directories
→ to see the groups a user belongs to, use the command: ***groups***

File and Directory Ownership:

→ when you look at a directory with ***ls -l***, the third and fourth columns with the username and the user group tells to whom the each file belongs, who has ownership of the files
→ using ***ls -la*** will show the hidden files and their owners
→ to change ownership of a file, command: ***chown user:group some_file_path*** ⇦ change owner command
→ ***chown evancarr:staff lorem*** ⇦ whatever username is put in here becomes the owner of the file that is the last argument of this command
→ You can also change ownership with either just the username or just the group name
→ for directories, if you supply just the directory, it will ONLY change ownership of the directory itself, not the files within it. If we want to apply the change to those as well, again, use ***-R***

- Ex: ***chown -R evancarr experimental/new_directory2*** ⇦ This will change ownership of that directory and all files and subdirectories within
- If the operation is not permitted, probably need to use the command ***sudo*** at the front of the ***chown*** command, which will ask for your system password to perform the operation

File and Directory Permissions:



→ the first column when you look at your current directory with ***ls -la*** tells you all about your files

- The first letter tells you ***d*** for directory, ***-*** for file, and ***s*** for symlink
- The next 9 characters are the permissions on that file or directory
    - Look at the 9 characters in sets of 3, the 3 categories of permissions
    - Each category has 3 permissions to it: Read, Write, Execute
    - Read, whether you can read a file or directory, write, and execute whether you can run a file like a program or search inside a directory
    - User refers to the owner for the file
- Octal Notation: for this, every r = 4, w = 2, and x = 0

rwxrw-r-- = 764 ⇨

File Permissions: Alpha Notation

	user	group	other
Read	r	r	r
Write	w	w	-
Execute	x	-	-
	rwx	rw-	r--

File Permissions: Octal Notation

	user	group	other
Read	4	4	4
Write	2	2	0
Execute	1	0	0
	7	6	4

Setting Permissions:
→ use the command: ***chmod ugo=rwx filename*** ⇦ Here we are setting the groups u (user), g (group), and o (other) equal to r, w, x, so that all three groups have access to read, write, and execute
→ Another example: ***chmod u=rwx, g=rw, o=r filename*** ⇦ Here are more restrictive settings
→ you can use + or - to edit: ***chmod ug+w filename*** or ***chmod o-w filename***
→ Doing the same thing but using octal notation ⇨ all permissions for all, all for user read and write for group and read for other, all of user read and execute for group and other, only read for user and nothing for anyone else
→ just like with changing ownerships, when you change permissions of a directory, you must use -R to change the files within a directory and the subdirectories

```
chmod 777 filename
chmod 764 filename
chmod 755 filename
chmod 400 filename
```

Root, Sudo, and Sudoers:
→ The **root user** is the most privileged user account. It can do absolutely anything on the Unix system. It can open any file, it can run any program. It can change any permissions. It can create new users and delete the accounts of existing users.
→ **Sudo** grants the privileges of the root user. It stands for "**substitute user and do**"
→ the default is to give access to a user as if they were the root user, but there are other settings as well.
→ it is used as a prefix to other commands: ***sudo cat file.txt*** ⇦ Tells the computer that we want root user authority to see the contents of this file.
→ It asks for your password to make sure you have permission to run the command you are requesting.
→ The password authentication stays valid for about 5 min
→ use ***sudo -k*** if you want it to expire right away or ***sudo -u*** if you want to use another username
→ if you have admin privileges, you are usually a sudoer. The sudoers are in the sudoer file, located at ***/etc/sudoers***.
→ see the list of sudoers by using: ***sudo cat /etc/sudoers*** ⇦ the list of admin and root users are the sudoers
→ When you get permissions denied, run ***sudo !!***, and that will run the last command, but this time using sudo

CHAPTER 4

Commands and Processes:

→ commands are programs, which are just files in the system that we have to run/execute
→ many of these program files are in the ***/bin*** or ***usr/bin***
→ you can find out where a command is stored with the command: which ***command_name***
→ for each command, there is a **manual page**, retrieved by command: ***man command_name***
→ Unix keeps up with the different commands by way of the $PATH variable / value, which you can see with the command: ***echo $PATH*** ⇦ This returns a list of directories where Unix will search, in order, for a command
→ This can all be **customized**
→ When you want to add to your commands or customize, you can change the path by command: ***PATH=***
→ If you change the path this way, it reverts back once that shell window is closed. Later we will learn about customizing this.

System Information Commands:

→ ***date*** ⇨ returns the current system date
→ ***uptime*** ⇨ will tell us how long the computer has been online, i.e. since the last time it was rebooted.
→ ***users*** ⇨ will give a list of all users logged into the system
→ ***who*** ⇨ gives us all users who are logged in
→ ***uname*** ⇨ returns the name of the type of Unix we are using.
- With ***-a***, you get a lot more info, ***-p*** gives the processor name

→ ***df*** (disk freespace) ⇨ tells the amount of free space on the disk (with ***-h***, human readable)
→ ***du file_path*** (disk usage) ⇨ gives a list of disk usage for the directory and all inside
- (***-h***, human readable, ***-a*** to get all files, ***-d integer*** to specify the depth)

Monitoring Processes:

→ ***ps*** (process status) ⇨ returns a list of the processes that are you, the logged in user's, processes
- With ***-a***, it will give processes from all users
- ***-x*** gives all processes, or ***ps aux*** for full info

→ ***top*** ⇨ gives a summary of the processes, memory, and usage

Stopping Processes:

→ most graceful is to just let a program finish
→ exiting a program is usually: ***q***, ***ctl+q***, ***:q***, ***x***, ***ctl+x***, or ***esc***
→ ***ctl+c*** is the universal way to stop a program if all else fails or if you start a process that you cannot stop any other way.
→ Closing a window may not close all processes
→ How to kill a process running in the background:

- Use command: ***kill -9 process_number***
- Get process id with ***ps***
- Do not overuse kill command

Using Command History:

→ use the command: ***history***, where it gets commands when using the up or down arrow
→ use ***ls -la*** from the home directory to see history files
→ using ! before a command number in history will run that command.
→ you can use ! and a negative command number, and it will run the command that was that many ago
→ ! and a keyword will pull up the history command and run it
→ !$ will bring up the most recent argument, even if it was used with a different command

Command History

Command	Bash	Zsh
history	All entries	15 most recent entries
history 100	100 most recent entries	Entries starting with #100
history 1	1 most recent entry	All entries
history -100	Invalid option	100 most recent entries

Command History Shortcuts

Command	Description
!3	References history command #3
!-2	References command which was two commands back
!cat	References most recent command beginning with "cat"
!!	References previous command
!$	References previous command's arguments

CHAPTER 5

Directing Output to a File:

→ stdin / standard input = keyboard
→ stdout / standard output = text terminal / screen
→ ***command > file*** will send the output of the command to a file, usually creating a new file

- Directed output this way will overwrite files

→ appending to a file, ***command >> file*** ⇦ this will just append the data to the end of the file

Directing Input from a File:

→ ***command < file*** ⇦ will bring in information from the file. File must already exist.
→ ***cat < file*** ⇦ would take the contents of the file as input, but it would output the same as if normally called
→ ***sort < names*** ⇦ this takes the file that is a list of names and outputs them in the terminal window sorted
→ The output of an input can be used this way: ***sort < names > sorted_names***

- This creates a new txt file with the output, the names sorted, as the content
- Make sure to use input first and output second
- More often you would see this as ***sort names > sorted_names***

→ Can also be used to send email***: mail -s 'Subject'*** ***someone@email.com*** ***< msg_file***

- If you already have a file saved, it could be sent as input to an email message like this.

→ Or to input something into a database: ***mysql -u user -p database < db_backup.sql***

- A file that contains mysql commands can be used as input to execute the commands and add the output to a database

Piping Output to Input:

→ | is the piping operator. The command on the left's output becomes the input to the command on the right
→ always from one command to another

- ***cat my_file | sort*** ⇦ the output from ***cat my_file*** is fed into the ***sort*** command which outputs to the terminal
- ***cat my_file | sort | uniq*** ⇦ takes the contents of my_file by using cat, sorts it using sort, and passes it to uniq which gets rid of any duplicates. ***cat names | sort | uniq*** ⇦ stdout removed duplicate names
- ***ps aux | less*** ⇦ creates a paginated version
- ***ps aux | grep keyword*** ⇦ will print out anything from the processes with that keyword
- ***history 1 | grep nano*** ⇦ will print out any command in the history with the keyword nano

Suppressing Output:

→ dev/null = "null device", "bit bucket", "black hole" ⇨ unix discards any data directed there
→ ***ls -l > /dev/null*** ⇨ all the output gets flushed away
→ useful when you are running commands that also print output and we want the action but not the output

CHAPTER 6

Configuring Your Environment:

FOR BASH SHELL:
→ in bash, the files to edit are ***~/.bash_profile*** and ***~/.bash_login*** (profile is the main one.)
→ ***~/.bashrc*** ⇨ runs every time you start a subshell
→ ***~/.bash_logout*** ⇨ runs every time you log out
→ to have configurations apply in every shell and subshell, they would need to be in both ***profile*** and ***rc*** files
→ You can tell the bash ***profile*** to always load what is in the ***rc*** file so that it always behaves consistently:

- ***if [-f ~/.bashrc]; then***
  source ~/.bashrc
  fi

FOR Z SHELL:
→ login loads: ***~/.zshenv, ~/.zprofile, ~/.zshrc, ~/.zlogin***
→ starting a sub shell loads: ***~/.zshenv, ~/.zshrc***
→ logging out loads ***~/.zlogout***
→ beginners use ***~/.zshrc*** for everything

INSIDE ***~/.zhrc:*** (It will run these lines every time I log in.)

```
echo ""                              # creates new line
echo -n "Welcome, ""; whoami        # ; lets you put more than one command on a line, says welcome to user
echo ""                              # another new line
echo -n 'Today is ' ; date           # tells the date and time
echo ""                              # another new line
```

→ ***source filename*** ⇦ tells it to read that file right now and run the commands

Setting Up Command Aliases:

→ command: ***alias*** ⇦ will give a list of all the currently defined aliases

→ to create a new alias, command: ***alias new_command="old_command"***
→ if you just input the command this way, it is only active during this session, but if we add it to one of the files for the z shell, it will be permanent

MY ALIASES SO FAR:

alias ll="ls -lah"
alias h="history -25"
alias mv='mv -i'
alias cp='cp -i'
alias rm='rm -i'
alias mkdir='mkdir -p'
alias df='df -h'
alias du='du -h'
alias current="pwd"
alias home="cd ~"
alias go="cd"
alias c="clear"
alias del="rm -i"
alias mkdir='mkdir -p'
alias df='df -h'
alias du='du -h'
alias current="pwd"
alias home="cd ~"
alias go="cd"
alias c="clear"
alias del="rm -i"
alias aliases="nano ~/.zshrc"
alias see="cat"
alias todo="nano ~/evans/todo"
alias add=">>"
alias create='mkdir -p'
alias mine='cd ~/evans'

Environment Variables:

→ dynamic, named values used by commands and applications, and they function a bit like settings
→ when you make them, they are only useful if a program or the environment makes use of them
→ to set, command: ***export FULL_NAME="Evan Marie Carr"*** ⇦ sets this as an environment variable
→ if you want them to stick around, they must be in a configuration file rather than just typing into command prompt
→ to edit ***PATH***, which is the path where Unix looks for any command you give it, and add to it, go to ***.zshrc*** and make ***export PATH = "$PATH: new_directories"*** ⇦ This will both retain the old path where all commands are and allow you to add new directories for Unix to search for commands.

Customizing the Command Prompt:

→ ***echo $PS1*** ⇦ shows you the current value set to be your command prompt
→ You can change this with ***PS1 = "desired prompt characters"***
→ if you set ***PS1 = "%n "*** ⇦ This sets it to be your username

Command Prompt Format Codes

Bash	Zsh	Output Description
\u	%n	Username
\s	%N	Current shell
\w	%d	Current working directory
\W	%1d	Basename of current working
\H	%M	Hostname
\h	%m	Hostname up to first period
\!	%!	History number of this command
\d	%w	Date in weekday-date format
\A	%T	Time in 24-hour HH:MM format
\t	%*	Time in 24-hour HH:MM:SS format
\@	%t	Time in 12-hour HH:MM am/pm format
\D{format}	%D{format}	Use strftime format (%Y-%m-%d)