Checklist01

Introduction

- What is **Unix**?
  - a family of operating systems
- What is a **shell**?
  - the unix shell is an interpreter: a program that translates commands into actions on-the-fly
- How would you use **git** to:
  - create a copy of a repo?
    - we **fork** the upstream repo to create a remote copy of the project called the origin in our own account
    - we then **clone** the repo to get the copy locally on our machine
  - Create a new branch?
    - `git switch -c myNewBranch`
  - View the changes you've made?
    - to see line by line changes: `git diff`
    - to see log of commits: `git log`
  - Save a change you've made?
    - put the file in the staging area: `git add myChangedFiled`
    - save the changes to the file: `git commit -m "Making Changes"`
  - Share the changes you've made?
    - `git push -u origin myBranchName`
  - Retrieve the changes from a remote location?
    - `git pull origin main`
      - this does a `git fetch` : which downloads changes but does **not** merge
      - this does a `git merge` which merges to the current branch I'm on
- What are **aliases**
  - How would you create one?
    - `alias lh='ls -lh'`
  - How would you make it persistent?
    - add the line above to the `~/.bashrc` file
- What are environmental variables?
  - they are key-value pairs that are maintained by the shell and passed to processes to configure their behavior

- How do you set and modify them?
  - set via `export myVar="new_value"`
  - creates myVar so that child processes inherit
  - set via `MYVAR="lol"
  - creates a shell variable
- Be familiar with:
  - HOME : stores the path to the users home directory
  - PATH : contains which directories to search for programs (it is colon sepreated)
  - SHELL: stores the path to your default shell program

Commands:

1. `git`
   - version control system for managing files
2. `pwd`
   - print current working directory
3. `whoami`
   - print the username of the current user
4. `id`
   - print user and group information

Files

Concepts

- What is the general layout or hierarchy of a typical Unix **filesystem** :
  - is is typically a tree containing these top level directories:
    - `/bin` : system applications and programs (basic commands : `ls` , `cp` , `cat`)
    - `/etc` : configuration files
    - `/lib` : essential libraries for `/bin`
    - `/usr` : user applications,
      - contains `/usr/bin` : most user commands
      - contains `/usr/lib` : libraries for `/usr/bin` programs
      - contains `/usr/local` : locally installed software
    - `/tmp` : temporary files (everyone can read or write)
    - `/var` : application data
- What is the difference between **absolute** and **relative** path:
  - absolute path: starts at the root directory

        - reference via: `/`
    - relative path: relative from current directory
        - reference via `./` for current directory or `../` for parent directory
- How do Unix file **permissions** work?
    - every file and directory has permissions for three classes of users:
        - owner (u)
        - group (g)
        - others (o)
    - How would you set them to restrict certain operations and certain classes?
        - set octals values using `chmod` for the specified restrictions
        - EX: `chmod 600 file.txt`
    - You should be able to translate from octal to **rwx** triplets and back.

Octal	rwx Equivalent
0	---
1	--x
2	-w-
3	-wx
4	r--
5	r-x
6	rw-
7	rwx

- Remember that `-rwxr-xr--`
    - the first dash means the thing is a file
        - if it were a `d` it would mean directory
    - then in this EX we have read, write, and executable permission for owner
    - only read and executable permission for group
    - only read permission for everyone else
    - the octal is 754
- What is difference between **hard link** and **soft link**
    - the soft link contains the **path** of the target file, not the inode
        - to create you do `ln -s original.txt softlink.txt`
        - can break if the original file is deleted
    - the hard link points directly to the **inode** of the original file
        - to create you do `ln orignial.txt hardlink.txt`

- will not break if original file is deleted

Commands:

```
ls #list the contects of a directory (long, human-readable, sorted)

chmod #set permissions using OCTAL and symbolic arguments

stat # view the inode information for a file

du #view the total disk usage for a file or directory

ln #create hardlinks and softlinks

find #search directory for files by name and type
```

Process

Concepts

- What is a **process**?
  - It is a running instance of a program
- What are typical **attributes** of a process:
  - **Process ID (PID)** : unique identifier
  - **Parent Process ID (PPID)** : PID of process that created this process
  - **Priority**: Determines CPU scheduling priority
  - **Nice Number**: User set adjustment to process prior(-20 to 19) (higher means lower priority)
  - **Terminal/TTY**: The terminal associated with the process
  - **UID/GUID**: User ID and Group ID of the process owner
- What does it mean to **signal** a process?
  - a signal is a software interrupt sent to a process to notify it of an event
  - in Unix, signals are used to communicate with processes
- What do the different signals do (TERM, INT, KILL)?
  - `SIGTERM (TERM)` -> graceful termination, allows process to clean up and exit
  - `SIGINT (INT)` -> interrupt from terminal (CTRL+C), can be caught and handles by process
  - `SIGKILL (KILL)` -> forceful termination, **cannot be caught or ignored**, immediate stop

- How do we **suspend**, **foreground**, and **background** a process? (know appropriate keybindings and commands to perform these actions)
  - Suspend (stopping temporarily) -> Ctrl + Z
  - Terminate -> Ctrl + C
  - Foreground -> `fg %job_number`
  - Background -> `bg %job_number`

Commands:

```
ps #list the processes for the current user and for the whole system

top # Interactively view and manage processes.

kill # signal a process by PID

time # measure how long it takes a process to execute

pkill #signal a process by name

## ALSO USE "-9" when using kill and pkill to terminate forcefully
```

I/O Redirection

- What three **files** does every process start with?
  - stdin (0)
  - stdout (1)
  - stderr (2)
- How do you redirect the output of a command into a new file?
  - `ls > myFile.txt`
- How do you redirect the output of a command and append it to a file?
  - `ls >> myFile.txt`
- How do you redirect the output of one command into the input of another?
  - `cat myFile.txt | grep welcome`
- How do you direct the contents of a file into the input of a command?
  - `wc < myFile.txt`
- What is `/dev/null`? How would you use it to output `stderr` but ignore `stdout` of a program?
  - `/dev/null` is like a black hole, everything is thrown away

    - `ls -l 1>/dev/null`
- How would you ignore both `stdout` and `stderr`
    - `command &>/dev/null`
- What is a **here document** and how would you use it?
    - allows you to provide multi-line input directly to a command from within the shell script or a command line. Usage:
    -
```
cat <<EOF
line1
line2
...
EOF

# THE EOF is the delimeter
```

Commands:

```
tee #redirects stdin to both stdout and a file
#Example would be tee myfile.txt

write #write to another usrs terminal
```

Networking

Concepts

- What is **bandwidth** and **latency**? (You should know what each measures and what programs we use to perform these measurements)
    - bandwidth is capacity (how much data can be transferred per unit of time)
        - you use `wget` or `curl` to measure it
    - latency is delay (how long it takes for a single packet of data to travel from source to destination)
        - you use `ping` to see the delay
        - you use `traceroute` to trace the networking route
- What is an **IP address**? How is a **domain name** related to it? (know commands to lookup this info)
    - the IP address is a unique numerical identifier assigned to a device on a network
    - the domain name is a human readable address that maps to an IP address

    - you can use `host nd.edu` or `dig nd.edu` or `nslookup nd.edu` to look this information up
- What is a network **port**?
    - basically like a door on a computer that lets network data go to the right program or service
        - each program listening on the network uses a different port number
    - so IP = the computer; and port = the specific service on that computer
- How to scan for ports
    - `nmap example.com`
    - `nmap -p 1-1000 example.com` scans only the specified range
- How to connect to a machine at a specific domain (or ip address) and port
    - `ssh user@example.com`
    - `ssh -p 2222 user@example.com` this only hits up port 2222
    - `nc example.com 80` opens a raw connection to that port

Commands:

```
ip / ifconfig #list the IP addresses on the current machine

nmap #scan remote machine for open ports

nc #general purpose tools to communicate with TCP-based network services

curl #communicate using HTTP (i.e download files off the web)

ping #measure the latency from host to remote machine
```

Shell Scripting

Concepts

- What is a **shell script**? What is a **she-bang** and why do we need it (or what does it do)? (You should interpret how a shell script is interpreted and the impact the she-bang has on this interpretation. You should also know how to execute a shell script)
    - A **shell script** is a text file containing a sequence of shell commands
        - its purpose is to automate tasks you would normally type manually in the shell
    - A **she-bang** is the very first line in a script starting with `#!`
        - EX: `#!/bin/sh`

        - is specifies the path to the interpreter so the systems knows which interpreter should run the script
    - How shell script is interpreted:
        - the OS reads the she-bang first
            - it runs with the interpreter specified
            - if there is no she-bang, it runs on the current shell
    - making shell script executable:
        - `chmod +x myscript.sh`
    - running it:
        - `./myscript.sh`
- Basics of shell programming language
    -

```
#VARIABLES

myVar="Hello World" #no spaces around =
echo $myVar #accessing value

---

read name #reading input from user
echo "hello $name"

---------------------------

#COMMAND SUBSTITUTION
today=$(date) #or you could do today=`date`
echo "today is $today"

---------------------------

#SHORT-CIRCUIT EVALUATION

command1 && command2 #command 2 runs only if command 1 succeeds (exit
code 0)
command1 || command2 #command2 runs only if command1 fails(non-zero
exit code)
---------------------------

#IF STATEMENT

if [ condition ]; then
  commands
elif [ other_condition ]; then
```

```
  commands
else
  commands
fi

---------------------------

#CASE STATEMENT

case "$var" in
  pattern1)
       commands ;;
  pattern2)
       commands ;;
  *)
       commands ;;
esac

---------------------------

# FOR LOOP

for arg in "$@"; do
     echo "Argument: $arg"
done

---------------------------

# WHILE LOOP
count=1
while [ $count -lt 5 ]; do
  echo "COUNT $count"
  count=$((count+1))
done

---------------------------

# FUNCTION
#definition
myFunc() {
echo "Hello from function"
}
#call
myFunc

---------------------------
```

```
#parsing command line arguments in shell scritps

while [ $# -gt 0 ]; do
  echo "Processing argument: $1"
  shift
done
```

Commands:

```
sh #bourne shell interperet

test #perform checks on file types and values (EX [ -e file.txt ] or test -e file.txt)

tar -xzvf mytar.tar -C /path/to/destination #extrating mytar.tar

seq START END
seq START STEP END
SEQ END #defaults to starting at 1
```

Filters

Concepts

- What are the **three tenets** of **Unix Philosophy**?
  - write programs that do **one thing** and **do it well**
  - write programs that work **together**
  - write programs that handle **text streams**, because that is a universal interface
- What are regular expressions?
  - A pattern that describes a set of strings
  - used to search, match, and manipulate text

Know these metacharacters:

Metacharacter	Meaning / Use
`.`	Matches any single character except newline
`*`	Matches **zero or more** of the preceding character or pattern
`[]`	Matches any **one** character inside the brackets
`[^]`	Matches any **one** character **not** inside the brackets
`^`	Matches the **start** of a line
`$`	Matches the **end** of a line
`()`	Groups expressions or captures matched text
|	Acts as **OR** between patterns
`\n`	Represents a **newline** character
`{m,n}`	Matches preceding element at least **m** times, at most **n** times

Know these character classes:

POSIX Class	Character Set Equivalent
`[:alpha:]`	`[a-zA-Z]`
`[:lower:]`	`[a-z]`
`[:upper:]`	`[A-Z]`
`[:space:]`	`[ \t\r\n\v\f]`
`[:digit:]`	`[0-9]`

Commands:

```
cut -d ',' -f 5
#cuts the output into fiels with delimerator of comma and only gives you the
fifth field; can use '-c 1-10' to only select specific charcters

sort #orders text

uniq # removes duplicates, only use after sort

tr #translates
tr '[[:lower:]]' '[[:upper:]]' #lowecase to uppercase

grep #search based on patterns
#-v flag means show lines WITHOUT this
```

```
sed #search and repalce or extract text
sed 's/apple/orange/g' #changes all "apple" to "orange"

head #displays first few lines of input

tail #displays last few lines of input
```