Communication in Linux:

———————————————————

With Network; other linux systems; remote users

```
cd /Users/seaqueue/Documents/CS/CY\ 2550\ Cybersecurity/cy2550/
cd /Users/seaqueue/Documents/CS/CS\ 3500\ \ OOD/Projects/G
```

Linux commands:

———————————

user@computer:	prompt
ping <IP or hostname>	checking the connection
ftp <IP or hostname>	(file transfer protocol) Logging in and establish a connection with a remote host; upload to and download files from remote computer; Navigating through directories of the remote computer; Browsing contents of the directories of the remote computer.
telnet <IP or hostname>	Connect to a remote Linux computer and work on it: Run programs remotely and conduct administration. (When run command on the terminal, it will executed on the remote computer.)
ssh <IP or hostname>	Replacement of telnet Securely connect to a remote computer. It is used by system administrator to control [ssh login.ccs.neu.edu]
scp <file> path	copy <file> from this machine to other machine. scp <file> login.ccs.neu.edu:/home/seaqueue
write <uid> cat <file> | write <uid>	write message to someone on Khoury server
exit	disconnect to the remote computer

chmod 777 <file>	assign/change linux file permission r = 4, w = 2, x =1
command + &	run the command at background
CTRL + z	suspend the process
bg	run the suspended process at background
jobs -l	listing the process in the background
fg %<job id>	bring background running job to foreground
chmod ugo-w <file>	remove write permission from user/group/other
chmod -r <file>	remove read permission from user/group/other
ls -all	check the file details Meaning of 10 bits: first bit: **d** means directory, otherwise it means a file. Next 3 bits of permission (r, w, x) is for owner of the file, and next 3 bits are for the group user the owner created, and final 3 bits are for other users.
sudo apt-get install <software name> sudo apt intall <software name>	package manager
cd cd ~ cd / cd <path>	change directory to home change directory to root change directory to <path>
. ..	current directory previous directory
ls ls -a ls -l	list directory contents display all files including hidden file display files with extra info
pwd	print working directory
cat <file>	display human readable text
head <file>	show first 5 or 10 line of contents
touch <file>	creating a new file

echo “message” <file>	write the message to the file
vi <file> emacs <file>	open/create <file> with vim / emacs text editors
file <file>	check the file type
mkdir <directory name>	creating directory
cp <file> <path>	copy <file> to <path>
cp -r <directory> <path>	copy <directory> to <path>
rm <file>	remove file
rm -r -f <directory>	remove non-empty <directory>
mv <file> <path>	move <file> to <path>; rename
clear	clear the terminal
man <git>	check the manual page, such as git
control + command	get the cursor out of the VM
ifconfig	find the ip-address
uname	find basic system information
javac -version java -version	checks the JDK version checks the JRE version

vi commands:

– – – – – – – – – – – –

vi-tutorial

i	Insert at cursor
a	write after cursor
A	write at the end of line
shift + zz	save the file and quit
esc	terminate insert mode
:q	quit without saving
:wq	save the file and quit
:w	save the file but keep it open
dd	delete line
3dd	delete 3 lines
k / j / h / L	move up / down / left / right

git - -help:

———————————

Tip 1	• If you are working on a certain feature (e.g. a new method, tests for a class, etc.) stage and commit only when you have it working. Usually you want commits to represent incrementally working versions that you may want to revert to, not rough work.
Tip 2	• Before stopping your work for the day, or at suitable times, push your changes to the remote repo (using git push origin master).
sudo apt install git	
brew install git	
git clone <url>	copy remote repository to local

git remote set-url origin <url>	HTTPS or SSH
git config - -list	check the setting
git init	Take a local directory that is not under version control, and turn it into a git repository. To push this repo to remote repo, create a new repo in Github, follow the instruction under the quick set up page.
git add <file>	
git commit -m “message”	
git commit -a -m “message”	don’t need to add the MODIFIED file before commit, -a does it.
git add <file> git commit - - amend	adding a forgotten file to last commit, reduce commit times
git diff	to see the changes of staged and not staged
git status	
git status -s	show status concisely
git log	check previous commits
git log - -pretty=oneline	display previous commits, other options to replace oneline: short, full, fuller
git checkout <commit id> <file>	overwrite the current file version with the version of <commit id>
git rm - -cached <file>	Remove the file from staged area, but keep it in the local hard drive.
git push git pull	
git pull - -rebase	pull the version on remote, shows the conflicts locally, manually fix the conflicts, git add <changed_file> git rebase - -continue
Branch:	
git branch	list branches I have

git branch <branch name>	This creates a new pointer to the same commit you're currently on.
git branch -v	show the last commit of all the branches
git branch -d <branch name>	delete the given branch
git branch -D <branch name>	If you want to delete a branch with the work you have not merged yet, you can force to delete it, and lose the work.
git branch - -merged	shows which branches are already merged into the branch you're on
git branch - -no-merged	To see all the branches that contain work you haven't yet merged in
git branch -f remote_branch_name origin/ remote_branch_name	pull the remote_branch down to local machine
git log - -oneline - -decorate	shows which branch the HEAD is pointing to, and all the commits in a concise way
git log - -oneline - -decorate - -all	shows all branches
git checkout <branch_name>	switch to the given branch name
git log - -oneline - -decorate - -graph - -all	print out the history of your commits, showing where your branch pointers are and how your history has diverged.
git checkout -b <new branch name>	create and go to that new branch.
git switch <branch name>	Switch to an existing branch
git switch -c <new branch>	create a new branch and switch to it, -c means create
git switch -	return to previous branch
git merge <branch name>	while one master branch, call this command would merge the given branch with the master branch.
git merge master	while you are on your branch, and master was changed, and you want to have that change in your branch

git branch - -move bad-name correct-name	Rename the branch locally
git push - -set-upstream origin correct-name	To let others see the corrected branch on the remote
git branch - -all	list current naming condition
git push origin - -delete bad-name	delete the old name; also can delete a no-longer needed remote branch
Tag:	
git tag	list the existing tags
Advanced commands:	
git log - -pretty=format: “%h - %an, %ar : %s”	
git diff - - cached git diff - - staged	

gpg -h:

What is gpg:	GUN Privacy Guard(gpg) support the Pretty Good Privacy (PGP), it’s a free, open-source, command-line implementation of PGP.
gpg - -full-gen-key	

gpg - -edit-key <UID>	allows you to change aspects of the key with the given UID. This command drops you into an interactive mode with many commands, which can be listed by typing "help". Commands that might be useful for this project include "addphoto" and "sign". When you're done editing a key, type "quit".
gpg - -list-keys gpg - -list-sigs	
gpg - -import <file>	import a public key
gpg - -armor - -export <uid> > <file>	export the public key to <file>
gpg - -armor - -export-secret-keys > <file>	esport the secret key to <file>
gpg - -armor - -sign <file>	armor: ASCII file plain text sign the <file>
gpg - -armor - -recipient (or -r) <uid> - -sign - -encrypt <file>	sign and encrypt the <file>
gpg - -sign-key <UID>	sign the key <UID>
gpg - -armor - -export <uid> > <file>	export the signed key to <file>
gpg - -fingerprint <email address>	show the fingerprint of a public key
curl <url> > <file>	copy the pub key to <file> find the url from Github, rawView

Special Symbol:

———————————

Symbols	control command spacebar
Edit Equation	option command e
	option shift k
≠	option =
≈	option x
±	option shift =
÷	option /
∞	option 5
π	option p
ø	option o
√	option v
¬	option l
∫	option b
∑	option w
… (ellipses)	option ;
• (bullet point)	option 8
º (degree)	option 0
© (copyright)	option g
™ (trademark)	option 2
® (registered trademark)	option r
¿	option shift /
† (dagger) for footnote	option t
¶ (Pilcrow) starting paragraphs	option 7
Entire Screen	command shift 3
Selected area	command shift 4
Specific window	command shift 4 spacebar
Empty trash	command shift delete
Adding notes	command shift k

Latex:

$A\tilde{B}C$ *vs* $\widetilde{ABC}$	\tilde{ABC}\ vs\ \widetilde{ABC}
$a\bar{b}c$ *vs* $\overline{abc}$	\bar{abc}\ vs\ \overline{abc}
$\begin{pmatrix} a \\ \sqcup \\ \varnothing \end{pmatrix}$	\begin{equation} \begin{pmatrix} a \\ ⊔\\ ∅ \end{pmatrix} \end{equation}
$p[i,j] = \min \begin{pmatrix} p[i-1,j-1] + 2 \times d(i,j) \\ p[i,j-1] + d(i,j) \\ p[i-1,j] + d(i,j) \end{pmatrix}$	\begin{equation} p[i,j] = \min \begin{pmatrix} p[i-1,j-1] + 2\times d(i,j) \\ p[i,j-1] + d(i,j) \\ p[i-1,j] + d(i,j) \end{pmatrix} \end{equation}
$D_{it} = \begin{cases} 2 & \text{if you like me} \\ 1 & \text{if you hate me} \\ 0 & \text{other waise} \end{cases}$	\begin{equation} D_{it} = \begin{cases} 2 & \text{if you like me} \\ 1 & \text{if you hate me} \\ 0 & \text{other waise} \end{cases} \end{equation}

Command line Interface VS Graphical User Interface:

There are 2 types of ways to interact with the computers:

1. Graphical User Interface (GUI)
2. Command Line Interface (CLI)

Interface:

Interface is the middleman between you and the machine. For example, Human —> TV remote (interface) —> TV (machine).

Graphical User Interface:

It's where we use mouse, touch screen to click through various icons and perform the desired action on our computers. This interface was made to make the operations on computers easier for regular users who might not be as tech-savy as programmers or software engineers.

Command Line Interface:

Before GUI, we use a computer via the CLI which is a text-only interface. Three main advantages of CLI are:

1. Since the CLI came first, this means GUI is still playing "catch-up" and this means you can do more things through the CLI than the GUI.
2. CLI is very resource-efficient (most of what you pay for in terms of hardware like CPU, GPU, mouse, touch-pad, touch-screen are all made for running the GUI. Because of the resource-efficiency, almost all of the web-servers and cloud infrastructure are run via CLI without a GUI.
3. Once you become a power user of Linux on the CLI, you can do a lot of fun projects which you never thought possible! This can include things like hosting your own website straight from your laptop, or tweak your old laptop to run as smoothly as a freshly bought one!

What is a command-line:

A command-line is a line of text (specific for a given purpose not some random text) which tells our computer to perform a specific action.
e.g., understands **sudo apt install** but not **please install software.**

Unfortunately, our computers have not reached a state where we can just order the computer around with natural language processing AI. We are getting there, but there is and awful lot of processing involved, which is inefficient. Due to this inefficiency, voice input is never going to replace the good old command-line interface when it comes to cloud computing and data centers. This is because lower power consumption and lower running costs means more profit for the company!

Command-line Format:

4 parts:
command sub-command options arguments
Command is mandatory, the other three are optional.

“Sudo apt-get install getdit”:
Command: apt-get
Sub-command: install
Options: none
Arguments: gedit

The tern sudo stands for “super-user do”. This term is used to get administrative privileges so that we can install and remove software from our computer. Usually, sudo commands are followed by a password request, this is a security measure to prevent malware from getting automatically installed on our computers.

The apt-get command is responsible for installation, removal, and updating of software in our system. A good analogy is to think of your computer as a **factory** and the apt-get command as the **manager** in that factory who is responsible for the installation of new equipment, removal of equipment that is no longer needed, and update desired equipment to their latest versions, while maintaining records of the equipment names and versions which are currently present in the factory.

```
root@ei:~# sudo apt-get install gedit
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  enchant-2 gedit-common gir1.2-gtksource-4 gir1.2-peas-1.0 gstreamer1.0-gl libamtk-5-0
  libamtk-5-common libenchant-2-2 libgraphene-1.0-0 libgspell-1-2 libgspell-1-common
  libgstreamer-gl1.0-0 libgtksourceview-4-0 libgtksourceview-4-common libjavascriptcoregtk-4.0-18
  libpeas-1.0-0 libpeas-common libtepl-4-0 libwebkit2gtk-4.0-37 libyelp0 python3-gi-cairo
  xdg-dbus-proxy yelp yelp-xsl zenity zenity-common
Suggested packages:
  gedit-plugins libenchant-2-voikko gstreamer1.0-libav
The following NEW packages will be installed:
  enchant-2 gedit gedit-common gir1.2-gtksource-4 gir1.2-peas-1.0 gstreamer1.0-gl libamtk-5-0
  libamtk-5-common libenchant-2-2 libgraphene-1.0-0 libgspell-1-2 libgspell-1-common
  libgstreamer-gl1.0-0 libgtksourceview-4-0 libgtksourceview-4-common libjavascriptcoregtk-4.0-18
  libpeas-1.0-0 libpeas-common libtepl-4-0 libwebkit2gtk-4.0-37 libyelp0 python3-gi-cairo
  xdg-dbus-proxy yelp yelp-xsl zenity zenity-common
0 upgraded, 27 newly installed, 0 to remove and 31 not upgraded.
Need to get 21,9 MB of archives.
After this operation, 98,7 MB of additional disk space will be used.
Do you want to continue? [Y/n] 
```

The manager is smart enough to understand that you need an entire system, and you expect him to install everything for you.

Let's assume the manager has a list of files by all the equipment part manufacturers. So naturally, the first thing the manager will do is to have a look at the documents of the parts needed to build the system. As you can see the first line in the screenshot says "**Reading package lists …Done**" which means apt-get went ahead and read all the available software in the repos.

Next, the manager makes a shopping list of all the parts: "**Building dependency tree**" which means the apt-get is trying to see which other

packages are required to make our desired package, gedit to tun. Dependency tree means certain parts are build on top of the other parts.

Once apt-get knows what software to download and where to download, it tells us what additional packages will be downloaded and installed and how much extra space will be needed. Once we confirm this list by typing the letter “Y” and press enter, apt-get downloads and installs everything for us.

In our factory analogy, the manager needs the company credit card to make the purchases, so by using “sudo” in our command and typing in the password, we are giving the manager the permission to use the company credit card!

What is apt-get:

——————————

APT stands for **Advanced Packaging Tool** which is a tool that is used to manage software in our system. “manage software” refers to 5 basic tasks:

- install software;
- remove software;
- upgrade software;
- upgrade system;
- Maintain the list of software that is installed in our system

What are packages in a Linux System:

——————————————————————

Linux OS is basically made up of 3 parts:

- The hardware
- The linux Kernel
- Software packages that work with the kernel to give us a complete Operating System

In simple words:
Hardware includes CPU, HDD, SSD, RAM, and all other components that makes a computer.

Kernel is a special software which is part of the OS that virtualizes and manages all the available hardware resources.

The software packages are typically applications that we run which can make use of these hardware resources by requesting access to them through the kernel. Examples of software packages can include application software like text editors, word processors, etc. or they can be the GUN utilities like bash, cron, dd, etc. or they can be device drivers to talk to the hardware. These packages are managed by a special class of software known as “Package managers”.

Contents of packages:

4 main components:

- Binaries or the executable programs
- Metadata files containing the version, dependencies, signatures, and other relevant information.
- Documentation and manuals
- Configuration files.

These packages are stored online in servers (software repositories).

What are software repositories:

This is the next question we must address. **Repositories (or repos for short) are basically a place where verified packages are stored for easy retrieval and installation.** They can be online like the APT repository or they can be on a local folder or a DVD where you have a special collection of software that you need.

This is one of the strong areas of Linux against other operating systems like Windows and Mac OSX, because on the 2-layer OS we usually need to go to the software vendor’s website to download a software package and

manually install them, whereas in Linux you can just type one lien of code to get your desired software.

Also because we need to go-to some untrusted vendor website to download software, which are not curated by the OS, there can be security issues where a malware could be come along with the download, whereas in Linux as long as you stick to the official software repositories, you don't need to worry about security issues as these software are already verified before being stored in the repo!

Windows and Mac OSX are improving this situation through their app-stores but Linux is miles ahead of other OS in this race!

How software packages are managed in Linux:

introduction-to-linux-package-managers

Needs for Package managers like apt and yum:

Manually downloading is time-consuming:

- Dependency: downloading a field requires downloading many other files it depends on.
- Package Verification: Checking package's integrity.
- Uninstall: hundreds files for one app.
- Updating: removing and reinstalling.
- Getting info: version, records…
- Architecture: 64-bit system will not work on a 32-bit system…

Thankfully, lots of Linux developers chipped in to make this process of installing and managing software smoother than ever by writing these wonderful tools called Package manager!

Functions of a Package manager:

Package managers serve the following functions to fulfill the needs mentioned above:

- Automatically resolves dependencies by keeping track of what software is needed to make a package work
- Verifies the integrity of the package before installing it
- Uninstall and update with ease
- Verifies the architecture compatibility
- Keeps track of all the packages installed in the system so that the system administrator can easily get information about what packages are present, when was it installed, what version are they running in, etc.

Next, let's see how these "Packages" are organized to make the job of package managers easier:

Architecture of Package managers:

In the beginning, only one level— two biggest distro around that time, Debian's **dpkg** (Debian package manager) and RedHat's **RPM** (RedHat Package Manager). These managers don't do everything for you, they leave a lot of work to end-user. Hence, second level was developed which does more automation. **YUM** (Yellowdog Updater Modified) for **RPM; apt** for **dpkg.**

By now, we know what are packages, why we need package managers, what functions they accomplish and the architecture of package managers.

RadHat based:
RPM and YUM/DNF : RHEL, CentOS, Fedora…

Debian based:
dpkg and APT: Debian, Ubuntu, Mint ….

Can I use yum on Debian based systems or vise versa?

Yes, but you need to do the work that the repository does. In other words, you need to compile your own packages which is not recommended.

Apt vs apt-get:

Both these commands are just different methods of invoking our package manager dpkg. The “apt-get” command is the older of these two and “apt” is just a refreshed version with more focus on making things simpler for the end-user. You can use both to do the same thing in most cases.

Apt has progress bar and apt-get doesn’t.
Apt has limited functionality and apt-get has more.
Apt-get is used with bash-scripts more than apt because of guaranteed backward compatibility.

When to use which:
It is recommended to use apt commands on the terminal and apt-get on the bash scripts.

This is because “apt” being the newer version is constantly evolving and can sometimes break backward compatibility. If we were to use apt on our scripts then we need to go into the script files and edit them again version after version till its finalized. For this reason, the “apt-get” family of commands is preferred over “apt” as they have guaranteed backward compatibility.
Also apt-get family has more functionality for usage in scripts that will give the programmer more control over the required outcome.

What does “Debian based system” mean?

There are two types of Linux distros
1.Original distributions
2.Derived distributions

Linux Distro:
Linux isn’t like Windows or Mac OSX. Microsoft combines all the bits of Windows internally to produce each new release of Windows and

distributes it as a single package. If you want Windows, you'll need to choose one of the versions Microsoft is offering.

Linux works differently. The linux operating system isn't produced by a single organization. Different organizations and people work on different parts. There's the Linux kernel, the X server (produces a graphical desktop), and more. System services, graphical programs, terminal commands— many are developed independently from another. They are all open-source software distributed in source code form.

If you want to, you can grab the source code for the Linux kernel, GNU shell utilities, and every other program on a Linux system, assembling it all yourself. However, compiling the software would take a lot of time — not to mention the work involved with making all the different programs work properly together.

Linux distributions do the hard work for you, taking all the code from the open-source projects and compiling it for you, combing it into a single operating system you can boot up and install. They also make choices for you, such as choosing the default desktop environment, browser, and other software. Most distributions add their own finishing touches, such as themes and custom software — the Unity desktop environment Ubuntu provides, for example.

NOW:

Original distributions are made from zero that take the kernel, GNU utilities, application software, etc and combine them into an installable operating system and distribute them to the end-users usually over the internet. Popular original distros include Debian, RedHat, Slackware, etc.

Derived distros are ones that take one of these original distributions, make some changes to it, so that it's more suitable for a specific purpose and then distribute them as an installable operating system.

Debian is the original distro here that used "apt-get" and "apt" and all the distros derived from it also use the same package management system as

their defaults. Some examples of Debian derived distros include the Ubuntu family (Ubuntu, Lubuntu, Kubuntu, etc), Linux Mint, Kali Linux and PureOS.

ssh (Secure Shell):

———————————

SSH is a network protocol that is widely used to access and manage a device remotely. It helps you to log into another computer over a network and allows you to execute commands in a remote machine. You can move files from one machine to another. SSH protocol encrypts traffic in both directions, which helps you to prevent trafficking, sniffing, and password theft.

telnet (Terminal Network):

———————————

Telnet protocol is mostly used by network admin to access and manage network devices remotely. It helps them access the device by telnetting to the IP address or hostname of a remote device. It allows users to access any application on a remote computer. This helps them to establish a connection to a remote system.

KEY DIFFERENCES:

- Telnet is the standard TCP/IP protocol for virtual terminal service, while SSH or Secure Shell is a program to log into another computer over a network to execute commands in a remote machine.
- Telnet is vulnerable to security attacks while SSH helps you to overcome many security issues of Telnet .
- Telnet uses port 23, which was designed specifically for local area networks, whereas SSH runs on port 22 by default.
- Telnet transfers the data in plain text while in SSH data is sent in encrypted format via a secure channel.
- Telnet is suitable for private networks. On the other hand, SSH is suitable for public networks.