Linux

**LINUX** stands for **Lovable Intellect Not Using XP.**

🡪A complete Linux system package called a distribution.

🡪**Linux**is an operating system that was developed by Linus Torvalds in 1991.

**cp**: - This command is used to copy files

Syntax: - cp src\_file dest\_file

* If the destination is a [directory](https://www.geeksforgeeks.org/linux-directory-structure/), the file is copied into that directory.
* If the `**cp`** command contains two file names, it copies the contents of the first file to the second file. If the second file doesn’t exist, it is created, and the content is copied into it. However, if the second file already exists, it is overwritten without warning.
* cp file\_name directory/ - To copy file into directory.
* cp -R directory1/ directory2/ - To copy from one directory to another.
* cp -i 🡪 here**-i(interactive): i** stands for Interactive copying. With this option the system first warns the user before overwriting the destination file. **cp** prompts for a response, if you press **y** then it overwrites the file and with any other option leaves it uncopied.
* cp -f - If the system is unable to open destination file for writing operation because the user doesn’t have writing permission for this file then by using **-f** option with **cp** command, destination file is deleted first and then copying of content is done from source to destination file.
* cp -p - With **-p**option **cp**preserves the following characteristics of each source file in the corresponding destination file: the time of the last data modification and the time of the last access, the ownership (only if it has permissions to do this), and the file permission-bits.
* cp \*.txt folder1 - The star wildcard represents anything i.e., all files and directories. Suppose we have many texts documents in a directory and want to copy it to another directory, it takes lots of time if we copy files 1 by 1 or command becomes too long if specify all these file names as the argument, but by using \* wildcard it becomes simple.

**ls**: - **ls** is a Linux shell command that lists directory contents of files and directories.

* ls -1 – lists one file/directory per line.
* ls -t – Sort files and directories by their last modification time, displaying the most recently modified ones first. ls -t | head -1 🡪 It gives the latest modified file/directories; here “**head -1**” picks up the latest file.
* ls -l – To show long listing information about files & directories

**-rw-r--r-- 1 root root 0 Sep 24 01:41 e**

Here starting character ‘–‘ for file, ‘d’ then it is a directories, ‘s’ for socket file, ‘l’ for link file.

**-rw-r--r-- 🡪** this field represents read, write and execute access for root, user group and other respectively.

**Here 1 🡪** specifies no.of links for that file.

**Here 1st root 🡪** Specifies the owner of the file.

**Here 2nd root 🡪** Specifies the group of the file.

**Here 0 🡪** Specifies the size of file in bytes.

**Here Sep 24 01:41 🡪** Specifies the last modified date & time.

**Here e 🡪** Specifies the file name.

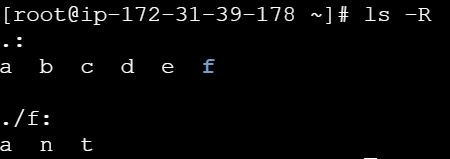
* ls -l directories/ - Used to see the files in that directory in long listing.

ls -ld directory\_name – Used to see that specific directory list details.

* ls -t – Used to sort the file names displayed in the order of last modification time.
* ls -ltr – To sort the file names in the last modification time in reverse order.
* ls -a – To show the hidden files in the directory, it will also show the files including (‘.’ Current directory and ‘..’ parent directory).

ls -A - To show the hidden files but not the ‘.’ & ‘..’

* ls -n – Lists the output like -l, but shows the uid and gid in numeric format instead of names.
* ls -S – Sort files and directories by their sizes, listing the largest ones first.
* ls -R – List files and directories recursively, including subdirectories.

Here’.:’ represents files & ‘./f:’ represents directories

* ls -g – Displays the group ownership of files and directories instead of the owner.
* ls -lhr – This is used to print files/directories in longlisting, reverse order of files/directories , and human readable.

**touch**: - The **touch** command is a standard command used in the UNIX/Linux operating system which is used to create, change and modify the timestamps of a file.

* touch file\_name – To create an empty file, you can also create multiple files at a time by using same touch command like touch file1 file2 file3 ..etc.

**pwd**: - it stands for ‘print working directory’.

* **pwd -L:** Prints the symbolic path.
* **pwd -P:** Prints the actual path.

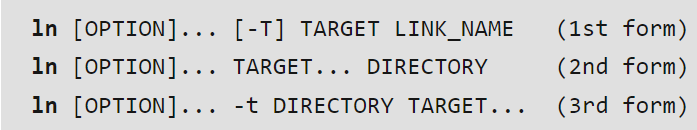
**mv: - It serves** two primary purposes:moving files and directories from one location to another and renaming them.

* mv source-file destination-file - This command is renamed ‘source-file’ to ‘destination-file’. If ‘destination-file’ already exists, in that case, it will be overwritten without prompting for confirmation.
* mv source-file directory/ - This is used to move the source-file to the directory. You can also move multiple files to a directory like mv srcfile1 srcfile2 directory/.
* mv -b file1 file2 - This will create a backup file if both files already exist with the tilde character (~) appended to it.
* mv -n file1 file 2 – here **-n** option prevents an **existing file** from being overwritten.
* mv -f file1 file2 - **mv**prompts for confirmation overwriting the destination file if a file is**write-protected.**The **-f** option overrides this minor protection and overwrites the destination file forcefully and deletes the source file.

**ln**: - The ‘**ln’** command in Linux is a powerful utility that allows you to create links between files.

Hard link :

**The ln command supports three forms, each with specific use cases:**



* ln source target\_link – This cmd is used to create hard link, here source represent the file/directory

**grep**: - The grep command in Unix/Linux is a powerful tool used for searching and manipulating text patterns within files. “g/re/p (globally search for a regular expression and print matching lines”

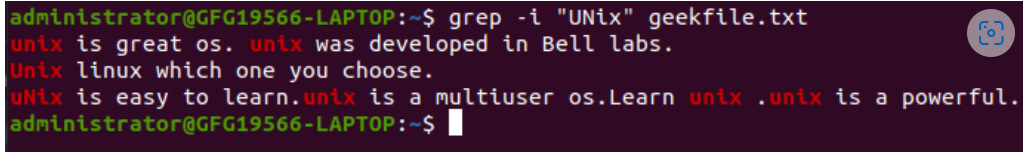
Syntax: - **grep [options] pattern/string [files]**

[options]: These are command-line flags that modify the behavior of grep.

[pattern]: This is the regular expression you want to search for.

[file]: This is the name of the file(s) you want to search within. You can specify multiple files for simultaneous searching.

* grep -i “pattern/string” filename – The -i option enables to search for a string case **insensitively** in the given file.



* grep -c “pattern/string” filename – We can find the number of lines that matches the given string/pattern. It prints the count
* grep -l “pattern/string” \* - It just display the **files** that contain the given string/pattern. Only if the starting substring matches.
* grep -w “pattern/string” filename - By default, grep matches the given string/pattern even if it is found as a substring in a file. The -w option to grep makes it match only the **whole words**.
* grep -o “patter/string” filename - By default, grep displays the entire line which has the matched string. We can make the grep to display only the matched string by using the **-o** option.
* grep -n “patter/string” filename - To show the **line number** of file with the line matched.
* grep -v “patter/string” filename - You can display the lines that are not matched with the specified search string pattern using the -v option.
* grep “^pattern/string” filename - The ^ regular expression pattern specifies the **starting of a line**. This can be used in grep to match the lines which start with the given pattern/string.
* grep -e “pattern/string” -e “pattern/string” -e “pattern/string” …. filename – We can search multiple pattern/string by -e flag.
* grep -f file1 file2 – Here **‘-f’** Takes patterns from file, one per line. Here **file1** has the pattern/string and **file2** contains the file we need to search on.
* grep -A [Number of Lines(n)] searchfile – Replace the -A flag: -A prints the searched line and n lines after the result, -B prints the searched line and n lines before the result, and -C prints the searched line and n lines after and before the result.
* grep -R pattern/string directory – ‘**-R’**prints the searched pattern in the given directory recursively in all the files.

**wc**: wc stands for **word count**. As the name implies, it is mainly used for counting purposes.

* wc filename - By default, it displays **four-columnar output.**
* The first column shows the number of lines present in a file specified, second column shows number of words present in the file, third column shows number of characters present in file and fourth column itself is the file name which are given as argument.
* wc -l filename - **-l:** This option prints the **number of lines** present in a file.
* wc -w filename - **-w:** This option prints the **number of words** present in a file. It displays number and file name. To display without filename, we can wc -w < filename
* wc -c/-m filename - **-c:** This option displays **count of bytes/Character** present in a file. With this option it displays two-columnar output, 1st column shows number of bytes/character present in a file and 2nd is the file name. Using **-m** option ‘wc’ command displays **count of characters** from a file.
* wc -L filename - **-L:** The ‘wc’ command allow an argument **-L**, it can be used to print out the length of longest (number of characters) line in a file.
* ls directory | wc -l - As we all know [**ls**](https://www.geeksforgeeks.org/practical-applications-ls-command-linux/) command in unix is used to display all the files and folders present in the directory, when it is piped with **wc** command with **-l** option it display count of all files and folders present in current directory.

**mkdir** : It is used to create directories mkdir(make directory)

* mkdir --help – It displays help related information and exits.
* mkdir ‘-v’ or ‘-verbose’ – It displays a message for every directory created. Syntax: – mkdir – v dir1 dir2 . . .
* mkdir -p - A flag which enables the command to create parent directories as necessary. If the directories exist, no error is specified.
* mkdir -m a=rwx dir - This option is used to set the file modes, i.e. permissions, etc. for the created directories.  (dir – non exist directory).
* mkdir /path/to/directory – This command creates a directory named “directory” at the specified absolute path.
* mkdir dir/sub\_dir – This command creates a directory structure with “my\_folder” as the parent directory and “sub\_folder” as its subdirectory.
* mkdir “My Documents” (or) mkdir My\ Documents – You can create directory name with space.

**rm**: **rm** command is used to remove objects such as files, directories, symbolic links and so on from the file system like UNIX.

* rm filename – To remove the file and to remove multiple files use rm file1 file2 file3 etc
* rm -i filename – To delete file interactively. rm -f filename – To remove file forcefully. rm -r file – **-r** It is Recursive deletion. rm \* - To remove all files at a time. rm -- -file – To remove the file starts with ‘–‘ this extra dash is necessary so that rm does not misinterpret the file name as an option.

**cat**: It can be used to read, concatenate, and write file contents to the standard output.

* cat filename – To view the content of the single file. cat file1 file2 file3 etc – To view the content of multiple files. cat -n filename – To view the content of the file preceding with line number.
* cat > filename - If you want to **create a new file** or **overwrite an existing file** with new content.
* cat file1 file2 file3 . . > file - To merge the contents of multiple files into a single file.
* cat >> filename – You can append the content in existing file by using this cmd and If you want to add the content of one file to another, ‘cat’ can be used along with the append (>>) operator cat file1 >> file2.
* cat -s filename – The -s option comes in handy when dealing with files containing repeated empty lines. It suppresses these repetitions, providing a cleaner output.

A screenshot of a computer

Description automatically generated

* tac filename – This command will print the content in reverse order like displaying the last line first followed by the second-to-last line and so on.
* cat -- “-filename” – Used to read the filename starts with ‘–‘.
* cat “filename” | more – Cat command if the file has a lot of content and can’t fit in the terminal.
* cat \*.txt – To display the content of all text files in a folder.

**echo**: The **echo** command in Linux is a built-in command that allows users to display lines of text or strings that are passed as arguments.

* echo “displayable\_content” – To display a text or string on the terminal.
* echo -e “geeks \bfor \bgeeks” - **-e** here enables the interpretation of backslash escapes **1. \b:** it removes all the spaces in between the previous text.
* echo -e “geeks \cfor geeks” - **\c:** suppress trailing new line with backspace interpreter ‘-e‘ to continue without emitting new line.
* echo -e “geeks \nfor \ngeeks” - **\n:** this option creates a new line from where it is used.
* echo -e “geeks \tfor \tgeeks” - **\t:** this option is used to create horizontal tab spaces.
* echo -e “geeks \vtfor \vgeeks” - **\v:** this option is used to create vertical tab spaces.
* echo \* - prints the existing files & directories. echo .\* - prints the existing hidden files & directories. echo .\* \* - prints the existing hidden files & directories.
* echo -n “geeks for geeks” - **-n:** this option is used to omit echoing trailing newline.



* echo “geeks for geeks” > filename - The output of the `echo` can be redirected to a file instead of displaying it on the terminal. We can achieve this by using the `>` or `>>` operators for output redirection.

**df**: - The ‘df’ command stands for “disk free” and is used to display information about the file systems on your Linux system. It provides a snapshot of disk usage, showing the total space, used space, available space, and the percentage of use for each **mounted file** system.

* df -a - If there is a need to display all file systems along with those which has zero block sizes then use **-a**option with **df**.
* df -h - This is used to make **df** command display the output in human-readable format.
* df -k - This displays the file system information and usage in 1K blocks.
* df -B - Scales sizes by the specified size Eg: - **df -BK for KB, df -BM for MB etc.**
* df --total - This option is used to produce total for a size, used and available columns in the output.
* df -T - With the help of this option, you will be able to see the corresponding type of the file system.
* df -t filetype- This is used when you wish the disk usage information of the file systems having a particular type only.
* df -x filetype - Now, you can also tell df to display the disk usage info of all the file systems except those having a particular type with the help ofthe  **‘-x’** option.
* df -i filename - This option is used to display **inode** information in the output.
* df -- sync - By default, the df command produces output with **– -no-sync** option which will not perform the sync system call prior to reporting usage information. Now we can use – -sync option which will force a sync resulting in the output being fully up to date.
* df -l - When we run df command then by default it shows any externally mounted file systems which include those from external NFS or Samba servers. We can hide the info of these external file systems from output with -l option syntax of which is shown below.

**cd** – The command allows you to change the current working directory within the file system.

**pwd** –Shows the present working directory.

Syntax: - cd directory

* cd / - By using `/` as an argument in `cd` we can change the directory to the root directory. The root directory is the first directory in your filesystem hierarchy.
* cd directory/sub\_directory/sub\_directory – To change the directory to the sub-directory.
* cd ~ - To change the directory to the home directory from any location in the Linux System.
* cd directory\ name - This command is used to navigate to a directory with white spaces.

**uname**: - Let’s start with the basics. The term “**uname**” stands for “**Unix Name**,” and the command itself is designed to provide you with key details about your Linux systems.

* uname -a - It prints all the system information in the following order: [Kernel](https://www.geeksforgeeks.org/kernel-in-operating-system/) name, network node hostname, kernel release date, kernel version, machine hardware name, hardware platform, operating system.
* uname -s – It prints the kernel name. uname -n -> Prints host name. uname -r – It prints kernel release date. uname -v -> It prints the version of the current version. uname -m -> It prints the machine hardware name of the machine, such as ‘**x86\_64′** for a 64-bit system. uname -p -> It prints the type of processor like 64bit/32bit. uname -i -> It prints the platform of the hardware or “unknown” if it can’t be determined. uname -o -> It prints the name of the operating system running on the machine.

**Wget**: - Wget is the non-interactive network downloader which is used to download files from the server even when the user has not logged on to the system and it can work in the background without hindering the current process.

* wget <http://example.com/sample.php> - To download a single file from a URL, simply specify the URL.
* wget -b <http://example.com/sample.php> - Run the download process in the background using the -b or --background option:
* wget [http://example.com/filenam.txt -o /path/filename.txt](http://example.com/filenam.txt%20-o%20/path/filename.txt) - Redirect the output messages to a log file using the -o option.
* wget -c <http://example.com/sample.php> - If a download is interrupted, resume it using the -c option.
* wget --tries=10 <http://example.com/samplefile.tar.gz> - Specify the number of retry attempts using the --tries option.
* wget -w 10 <http://example.com/samplefile.tar.gz> - Set the wait time between retrievals using the -w option (in seconds, minutes, hours, or days)
* wget -r <http://example.com/> - Enable recursive retrieval to download an entire website using the -r option.
* wget -i filename - Read URLs from a file and download them using the **-i** option.

**whereis**: - The ‘**whereis**’ command is used to find the location of the source, binary, and manual sections of a command in the Linux system.

**rmdir**: - The ‘**rmdir'** command in [Linux](https://www.geeksforgeeks.org/introduction-linux-shell-shell-scripting/) is specifically designed for removing empty directories.

* rmdir dir1 dir2 dir3 . . – To remove multiple empty directories.
* rmdir -p dir/subdir - You can use the **-p** option with the rmdir command to delete an empty directory, including all the subdirectories. Here -p represents parent directories.
* rmdir -v dir1 dir2 dir3 . . – If you want to display the message after removing the directory, you can use the -v option with the rmdir command. You can also delete multiple directories with the same expression, suppose you have files L1, L2, L3 … for that the cmd to remove directories is rmdik -v L\*.
* rmdir --ignore-fail-on-non-empty dir - to ignore the occurrences due to the non-empty directories. For instance, let’s remove the LINUX directory that contains sub-directories. We can also use rm -r directory to remove non-empty directories with displaying message and rm -rf directory to remove non-empty directories without displaying message.

**locate**: - locate command in Linux is used to find the files by name.

* locate filename – It will search for filename in a particular directory.
* locate file1 -n 10 - It will show 10 results for the searching of file ending with ‘file1’.
* locate -c [.txt] \*- It will count files ending with “*.txt*.”
* locate -i \*SAMPLE.txt\* - This command is configured to process queries in a case sensitive manner. It means SAMPLE.TXT will show a different result than *sample.txt*.
* locate -i -0 \*sample.txt\* - Default separator for locate command is the newline (\\n) character. But if someone wants to use a different separator like the ASCII NULL, then he/she can do so using the *-0*-command line option.

**Process Status(ps)**: - In Linux, a process is a running instance of a program. When you execute a program, it becomes a process, an independent, executing entity with its own memory space.We can use multiple commands to list the running processes in Linux like **ps**,**top**,**htop**,and **atop** commands in Linux.

* ps - Shows the processes for the current shell.

A black and white screen with numbers and numbers

Description automatically generated

Here **PID** – The unique Process ID

**TTY** – Terminal type that the user is logged into

**TIME** – Amount of CPU in min & sec that the process has been running.

**CMD** – Name of the cmd that launches the process.

* ps -A/-e - To view all running processes, use either of the following options with the ‘**ps’**command.
* ps -a - View all processes except both session leaders and processes not associated with a terminal.

**whoami**: - The ‘**whoami**’ command is a simple yet powerful utility designed to reveal the current username associated with the active user session.

* whoami – To display the current username associated with the user account.
* whoami --help – It gives help msg and exits.
* whoami --version – It gives the version info and exits.
* whoami -u – This option displays UID instead of username.
* whoami -e – This option shows the effective user ID, providing information about the user’s privileges.

**SORT**: - SORT command sorts the contents of a text file, line by line. The sort command is a command-line utility for sorting lines of text files.

It supports sorting alphabetically, in reverse order, by number, by month, and can also remove duplicates.

The sort command can also sort by items not at the beginning of the line, **ignore case sensitivity**, and return whether a file is sorted or not.

**The sort command follows these features as stated below:**

1. Lines starting with a number will appear before lines starting with a letter.
2. Lines starting with a letter that appears earlier in the alphabet will appear before lines starting with a letter that appears later in the alphabet.
3. Lines starting with a uppercase letter will appear before lines starting with the same letter in lowercase.

Note: - This command does not actually change the input file.

* sort filename - To sort the lines alphabetically.
* When we have a mix file with both uppercase and lowercase letters then first the upper-case letters would be sorted following with the lower-case letters.
* sort -n filename - By default, ‘sort’ treats numbers **as text and sorts them accordingly**. If you want to perform numeric sorting, use the ‘-n’ option.
* sort -r filename – It is used to Sort lines in reverse order.

A screen shot of a computer

Description automatically generated

* sort -k3 filename – It indicates that the sorting should be done based on the 3rd Column. Here K reps’ column.

**Git**

Git is a popular **version control system**. It was created by Linus Torvalds in 2005 and has been maintained by Junio Hamano since then.

It is used for:

* Tracking code changes
* Tracking who made changes
* Coding collaboration

What does Git do?

* Manage projects with **Repositories**
* **Clone** a project to work on a local copy
* Control and track changes with **Staging** and **Committing**
* **Branch** and **Merge** to allow for work on different parts and versions of a project
* **Pull** the latest version of the project to a local copy
* **Push** local updates to the main project

What is GitHub?

* Git is not the same as GitHub.
* GitHub makes tools that use Git.
* GitHub is the largest host of source code in the world and has been owned by Microsoft since 2018.

**Git Configuration:** When working with Git, it's essential to set up your username and email address as they are used to identify the author of commits. This configuration is a one-time setup per machine and will be associated with all your commits. However, you can override these settings on a per-repository basis if needed.

* git config --global user.name “name” – To change the username.
* git config --global user.email “e-mail” – To change the user email.

**Note:** Use global to set the username and e-mail for **every repository** on your computer. If you want to set the username/e-mail for just the current repo, you can remove global

**Creating Git folder**: Firstly, we can create a folder/directory that we need to work on our project. If you have already a folder to work change to that path in CLI (If it is a windows open that folder and right click Git bash there).

* mkdir directory\_name 🡪 cd directory\_name

Initialize the Git: After creating the directory or changing the path to existing directory we need to initialize the git; **Git now knows that it should watch the folder you initiated it on. Git creates a hidden folder to keep track of changes.**

**Git Staging Environment**: One of the core functions of Git is the concepts of the Staging Environment, and the Commit. **Staged** files are files that are ready to be **committed** to the repository you are working on.

* git add filename – To Stage the files.
* git add --all/-A – To Stage all/multiple files at a time.

**Git commit**: Adding commits keep track of our progress and changes as we work. Git considers each commit change point or "save point". It is a point in the project you can go back to if you find a bug or want to make a change.

* When we commit, we should **always** include a **message**.
* git commit -m “message” filename – Used to commit the staged file.
* git commit -a -m “message” filename – Used to automatically stage all tracked file and then made commit.
* git commit -m “message” – Commits all staging area files in a single commit.

**Git commit without Stage:** Sometimes, when you make small changes, using the staging environment seems like a waste of time. It is possible to commit changes directly, skipping the staging environment. The -a option will automatically stage every changed, already tracked file. Now to check the status use git status --short.

**Warning:** Skipping the Staging Environment is not generally recommended. Skipping the stage step can sometimes make you include unwanted changes.

**Note: Short status flags are:**

* **?? - Untracked files**
* **A - Files added to stage**
* **M - Modified files**
* **D - Deleted files**

git log - To view the history of commits for a repository.

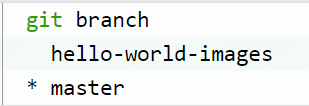
**Git help**: - If you are having trouble remembering commands or options for commands, you can use Git help.

* git command -help - See all the available options for the specific command.
* git help --all – See all possible commands.

**Warning:** This will display a very long list of commands

**Git Branch:** - In Git, a branch is a new/separate version of the main repository. We are working in our local repository, and we do not want to disturb or possibly wreck the main project.

* git branch “branch\_name” – To create a new branch also you can check whether the branch created or not by using “git branch”.



the **\*** beside master specifies that we are currently on that branch.

* git checkout “branch name” - **checkout** is the command used to check out a branch. Moving us **from** the current branch **to** the one specified at the end of the command. **This command is also used to switch between branches.**
* git checkout -b “branch name” – Emergency branch using **-b** option on checkout will create a new branch, and move to it, if it does not exist.

**Git branch merge: -** It is used to integrate changes from different branches.

* git merge “mergable-branch” – This command is used to merge the branches. For example, git merge mb here let’s assume before using this command we are in master branch now we need to merge master with **mb** then we use this cmd.
* git checkout branch\_name – This command is used to switch between branches.

**Git branches delete**: -

* git branch -d branch\_name – Used to delete the branch which is already merged. If the branch is not merged it will give you the error msg.
* git branch -D branch\_name – Used to delete the branch forcefully even it is not merged with any other branch.

**Git-Hub**

Git-hub is more than just a code repository and a cloud-based platform that allows software developers to store, track and collaborate on code. It’s a robust platform that has become essential tool for developers worldwide.

🡪Firstly, we need to create a GitHub account we need not worry it’s free of cost.

🡪After creating GitHub account, we can create our own repository.

🡪Push Local repository to GitHub.

* git remote add origin URL – To add our local repo to GitHub [URL – we need to copy from our GitHub repo that we created goto code and then https copy the URL].
* git push --set-upstream origin branch\_name – To push our local repo to GitHub it will ask for user\_name and password, give the username and for password [GitHub🡪settings🡪developer settings🡪tokens🡪generate token] then it will generate token save that token for further use. git push -u origin branch\_name here we used **-u** for forceful push.

**Note: - Here origin is the default name for the primary remote repository**

**Code edit in GitHub**: - In addition to being a host for Git content, GitHub has a very good code editor.

* Select file to edit and make your changes after making the changes you will see the “Commit Changes” button click on it and add message for that commit boom! It’s done you have successfully edited and committed.

**Pull** is a combination of 2 different commands fetch and merge.

* **Git fetch: -** Gets all the change history of a tracked branch/repo git fetch origin after this we can see the recent changes, we can track our status git status If you made any changes in GitHub and committed that changes we get this msg in git status “We are behind the origin/branch by 1 commit” you can also double check with logs git log origin/branch and in output you can see that and we can also verify by showing difference between our local branch and origin/branch git diff origin/branch after confirmation now we can safely merge.
* **Git merge:** - merge combine the current branch with a specified branch git merge origin/branch after merging you can check the status git status but if you just want o update your local repo without going through all this steps you can use git pull.
* **Git Pull:** - **pull** is a combination of fetch and merge. It is used to pull all changes from a remote repository into the branch you are working on. git pull origin
* **That is how you keep your local Git up to date from a remote repository**

**GitHub branch: -** You can also create branch on GitHub

* On GitHub access your repository and click on branch in search bar it will say “Find or create a branch…” there you can **search/create** a branch. After creating the branch should now the branch should be created and active.
* Here you need to select the branch that you need to take reference from.
* After creating new branch, you can make some changes in it and in git you can use pull to get it up to date. In git check for new branch git branch -a – To see all the branched including remote git branch -r – To see branches in remote.

🡪 If you click the "Compare & pull request", you can go through the changes made and new files added.

🡪There you can compare between branches, and you can make changes after making changes click on “create pull request” and then click on “Merge pull request” 🡪”Confirm merge” now the pull request is successfully merged. To keep the repo from getting overly complicated, you can delete the now unused branch by clicking "Delete branch".

🡪You could compare the branches if any changes made then you can merge them.

**GitHub flow**: - The GitHub flow is a workflow designed to work well with Git and GitHub. It focuses on branching and makes it possible for teams to experiment freely, and make deployments regularly.

The GitHub flow works like this:

* Create a new Branch
* Make changes and add Commits
* Open a Pull Request
* Review
* Deploy
* Merge

**Git GitHub Fork**: - A fork is a copy of a repository. This is useful when you want to contribute to someone else's project or start your own project based on theirs.

🡪 **fork** is not a command in Git, but something offered in GitHub and other repository hosts. Let's start by logging in to GitHub, and fork our repository:  
<https://github.com/w3schools-test/w3schools-test.github.io>

🡪At the heart of Git is collaboration. However, Git does not allow you to add code to someone else's repository without access rights.

**Git Clone from GitHub: -** Now we have our own fork, but only on GitHub. We also want a clone on our local Git to keep working on it.

🡪 A clone is a full copy of a repository, including all logging and versions of files.

* git clone URL – To clone the repository from where we forked.
* git clone URL directory – To only clone a particular directory/folder.

🡪You can check the repo whether it is cloned or not by giving ls after giving you can see the directory/repo that you have cloned.

🡪Now go to that directory/repo by cd then check the status. Also, you can see the log, now we have a full copy of the original repository.

**Configuring remotes: -**

**🡪**Basically, we have a full copy of a repository, whose origin we are not allowed to make changes to.

🡪We see that **origin** is set up to the original "w3schools-test" repository, we also want to add our own fork.

First, we rename the original origin remote:

* git remote rename origin upstream – To change the origin name here upstream can be renamed to any relevant name.

Then fetch the URL of our own fork:

* In GitHub go to forked repo and coy the https URL

And add that as origin:

* git remote add origin URL, paste the copied url here. Then check the remote version git remote -v.

**Note**: - According to Git naming conventions, it is recommended to name your own repository **origin**, and the one you forked for **upstream**.

Now we have 2 remotes:

* **origin** - our own fork, where we have read and write access
* **upstream** - the original, where we have read-only access

**Git GitHub send pull request**: -

🡪Now open any file just make some minor change and git add and git commit then push that origin to our GitHub git push origin.

🡪After pushing Go to GitHub, and we see that the repository has a new commit. And we can send a Pull Request to the original repository.

🡪Click on “Contribute”-> In Contribute you can see “open pull request”->Give the title and description then “create pull request” then pull request is sent.

🡪Approving pull request: -

* Now any member with **access** can see the Pull Request when they see the original repository.
* And they can see the **proposed changes (you can see in pull request)**🡪They will comment on that changes and **merge (you can see the “Merge pull request” button click on it )** those after merging they will **confirm(By clicking “confirm merge”)**🡪And they can see changes have been merged with master.

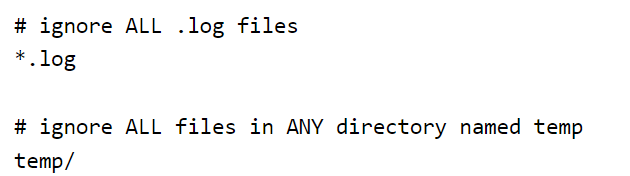
**Git Ignore**: - When sharing your code with others, there are often files or parts of your project, you do not want to share.

Examples: - log files, temporary files, hidden files, personal files etc..

🡪Git can specify which files or parts of your project should be ignored by Git using a .gitignore file.

🡪Git will not track files and folders specified in .gitignore. However, the .gitignore file itself **IS** tracked by Git.

Create **.gitignore** file in root directory [touch .gitignore]and open the .gitignore file by using a text editor add the files and directories that you need to ignore.



Here \*.log represents any files with .log extension; temp/ represents all files in that directory.

You can change a remote origin from HTTPS to SSH with the command:

git remote set-url *remote-name* git@github.com:*username*/*repository*.git

**Eg: -** git remote set-url origin [git@github.com:w3schools-test/hello-world.git](mailto:git@github.com:w3schools-test/hello-world.git)

**Git revert**: - revert is the command we use when we want to take a previous commit and add it as a new commit, keeping the log intact.

Synax:- git revert commit\_id

* git revert HEAD --no-edit - We revert the latest commit using git revert HEAD (revert the latest change, and then commit), adding the option --no-edit to skip the commit message editor (getting the default revert message).

Note: - To revert to the earlier commits, use git revert HEAD~x (x being a number.)

**Git reset**: - Revert is the command we use when we want to move the repository back to a previous commit, discarding any changes made after that commit.

Syntax: -git reset commit\_id

**Git amend**: -commit **--amend** is used to modify the most recent commit.

🡪It combines changes in the staging environment with the latest commit and creates a new commit. This new commit replaces the latest commit entirely.

Compact version of the status repository – git status –short

Show all git possible commands in command line – git help -all