**Git Version Control**

 Git is a distributed version control system (DVCS) that is widely used for tracking changes in source code during software development. It allows multiple developers to collaborate on a project and keep track of their changes. Here are some key concepts and commands associated with Git:  
  
### Key Concepts:  
  
1. \*\*Repository (Repo):\*\*  
- A Git repository is a collection of files and version history.  
- Repositories can be local (on your machine) or remote (on a server).  
  
2. \*\*Commit:\*\*  
- A commit is a snapshot of the changes to the repository at a particular point in time.  
- It includes a commit message that describes the changes.  
  
3. \*\*Branch:\*\*  
- A branch is a separate line of development.  
- Branches allow you to work on features or bug fixes without affecting the main codebase.  
  
4. \*\*Merge:\*\*  
- Merging combines changes from different branches.  
  
5. \*\*Pull Request (PR):\*\*  
- In a collaborative environment, a pull request is a way to propose changes and discuss them before merging.  
  
6. \*\*Clone:\*\*  
- Cloning is the process of copying a repository from a remote server to your local machine.  
  
7. \*\*Push and Pull:\*\*  
- Pushing is sending your local changes to a remote repository.  
- Pulling is fetching changes from a remote repository to update your local copy.  
  
8. \*\*Remote:\*\*  
- A remote is a version of your repository that is hosted on a server.  
  
### Basic Commands:  
  
1. \*\*Initialize a Repository:\*\*  
```bash  
git init  
```  
  
2. \*\*Clone a Repository:\*\*  
```bash  
git clone <repository\_url>  
```  
  
3. \*\*Check Status:\*\*  
```bash  
git status  
```  
  
4. \*\*Add Changes:\*\*  
```bash  
git add <file(s)>  
```  
  
5. \*\*Commit Changes:\*\*  
```bash  
git commit -m "Commit message"  
```  
  
6. \*\*Create a Branch:\*\*  
```bash  
git branch <branch\_name>  
```  
  
7. \*\*Switch Branches:\*\*  
```bash  
git checkout <branch\_name>  
```  
  
8. \*\*Merge Branches:\*\*  
```bash  
git merge <branch\_name>  
```  
  
9. \*\*Push Changes to Remote:\*\*  
```bash  
git push origin <branch\_name>  
```  
  
10. \*\*Pull Changes from Remote:\*\*  
```bash  
git pull origin <branch\_name>  
```  
  
11. \*\*Check Remote Repositories:\*\*  
```bash  
git remote -v  
```  
  
These are just basic commands, and Git has a rich set of features for managing version control in various scenarios. Understanding these concepts and commands will help you get started with Git.

**What is Git?**

 Git is a distributed version control system (DVCS) designed to handle everything from small to very large projects with speed and efficiency. It was created by Linus Torvalds in 2005, the same person who developed the Linux operating system. Git is free and open-source software, and it has become the most widely used version control system in the world.  
  
Here are some key features and aspects of Git:  
  
1. Distributed Version Control:  
- Git is distributed, meaning that every developer has a full copy of the entire repository, including its history. This allows for offline work and provides redundancy.  
  
2. Branching and Merging:  
- Git makes branching and merging straightforward. Developers can work on separate branches for features or bug fixes and then merge their changes back into the main branch.  
  
3. Snapshot-Based:  
- Git doesn't store changes as a list of file-based differences. Instead, it captures a snapshot of the project at each commit. This approach provides speed and efficiency in managing and retrieving project history.  
  
4. Commit History:  
- Each commit in Git has a unique identifier (hash) and contains information about the changes made, the author, and a timestamp. This makes it easy to track the history of the project.  
  
5. Collaboration:  
- Git facilitates collaboration among developers by allowing them to work on different aspects of a project simultaneously. Changes can be shared through a central repository, and developers can pull and push changes to stay in sync.  
  
6. Open Source and Community Support:  
- Git is open-source, meaning that its source code is freely available for anyone to inspect, modify, and enhance. It has a large and active community that contributes to its development and provides support.  
  
7. GitHub and GitLab:  
- GitHub and GitLab are web-based platforms that provide hosting for Git repositories. They offer additional collaboration features, issue tracking, and pull requests. Many open-source and private projects use these platforms to manage their source code.  
  
8. Command Line and GUI:  
- Git can be used through the command line, which provides powerful and fine-grained control. Additionally, there are several graphical user interfaces (GUIs) available for those who prefer a visual approach.  
  
Git has become an essential tool for software development, enabling version control, collaboration, and project management across a wide range of industries and applications.

**Daily Notes**

 Git is widely regarded as one of the most popular and robust version control systems, primarily due to its numerous features that enhance collaboration, flexibility, and reliability. Here are nine prominent features of Git:  
  
1. \*\*Distributed Version Control:\*\*  
- Git is a distributed version control system, meaning that each developer has a complete copy of the entire repository, including the full history of changes. This decentralization enables developers to work independently and makes the system more resilient to network failures.  
  
2. \*\*Branching and Merging:\*\*  
- Git provides robust support for branching and merging. Branching allows developers to work on separate features or bug fixes without affecting the main codebase. Merging allows these branches to be combined seamlessly, facilitating collaboration and parallel development.  
  
3. \*\*Fast and Efficient:\*\*  
- Git is designed to be fast and efficient, even with large repositories. The use of advanced algorithms, such as the Delta Compression algorithm, ensures that only the changes (deltas) between versions are stored. This minimizes the amount of data that needs to be transferred and stored.  
  
4. \*\*Commit Integrity:\*\*  
- Each commit in Git is identified by a unique hash based on the content of the commit. This ensures the integrity of the version history. If any part of the commit changes, the hash is different, making it easy to detect and correct errors.  
  
5. \*\*Staging Area (Index):\*\*  
- Git has a staging area (also known as the index) that allows developers to selectively choose which changes to include in the next commit. This adds a level of granularity to the version control process, enabling developers to commit specific changes while leaving others uncommitted.  
  
6. \*\*Open Source and Community Support:\*\*  
- Git is open source, meaning that its source code is freely available for anyone to inspect, modify, and contribute to. This open nature has led to a vibrant community that actively supports and extends Git, contributing to its continuous improvement.  
  
7. \*\*Compatibility:\*\*  
- Git is highly compatible with different operating systems and platforms. It works seamlessly across Windows, macOS, and Linux, as well as integrates well with various IDEs (Integrated Development Environments) and other tools. This broad compatibility makes Git accessible to a wide range of developers.  
  
8. \*\*Security:\*\*  
- Git provides strong data integrity and security features. All data transferred between repositories is encrypted, and the content of the files is secured using hashing algorithms. Access controls can be implemented to restrict user permissions, and cryptographic signatures ensure the authenticity of commits.  
  
9. \*\*Flexible Workflow:\*\*  
- Git supports a flexible and adaptable workflow, accommodating various development strategies. Whether it's centralized, feature branching, Gitflow, or others, developers can choose a workflow that best suits the requirements of their project. This adaptability contributes to Git's popularity across different types of development projects.

**Benefits of using Git**

 Benefits of using Git  
A version control application allows us to keep track of all the changes that we make in the files of our project(s). Every time we make changes in files of an existing project, we can push those changes to a repository.  
  
Other developers are allowed to pull your changes from the repository and continue to work with the updates that you added to the project files.  
  
  
Major benefits of Git are:  
  
Saves Time  
  
Git is lightning fast technology. Each command takes only a few seconds to execute so we can save a lot of time as compared to login to a GitHub account and find out its features.  
  
Offline Working  
  
One of the most important benefits of Git is that it supports offline working. If we are facing internet connectivity issues, it will not affect our work. In Git, we can do almost everything locally. Comparatively, other CVS like SVN is limited and prefer the connection with the central repository.  
  
Undo Mistakes  
  
One additional benefit of Git is we can Undo mistakes. Sometimes the undo can be a savior option for us. Git provides the undo option for almost everything.  
  
Track the Changes  
  
Git facilitates with some exciting features such as Diff, Log, and Status, which allows us to track changes so we can check the status, compare our files or branches.

**Instructions for configuration**

 Instructions for configuration  
Configuring Git  
  
Now that we have install git lets config our environment. To configure we use the git config command  
  
Git supports a command called git config that lets you get and set configuration variables that control all facets of how Git looks and operates. It is used to set Git configuration values on a global or local project level.  
  
Setting user.name and user.email are the necessary configuration options as your name and email will show up in your commit messages.  
  
Setting username  
  
The username is used by the Git for each commit.  
  
$ git config --global user.name "Firstname Lastname"  
  
Setting email id  
  
The Git uses this email id for each commit.  
  
$ git config --global user.email "test@mydomain.com"  
  
Other configuration options include:  
  
Setting editor  
  
You can set the default text editor when Git needs you to type in a message. If you have not selected any of the editors, Git will use your default system's editor.  
  
To select a different text editor, such as Vim,  
  
$ git config --global core.editor Vim  
  
Checking Your Settings  
  
You can check your configuration settings; you can use the git config --list command to list all the settings that Git can find at that point.  
  
$ git config -list.  
  
Colored output  
  
You can customize your Git output to view a personalized color theme. The git config can be used to set these color themes.  
  
Color.ui  
  
$ Git config -global color.ui true  
  
The default value of color.ui is set as auto, which will apply colors to the immediate terminal output stream. You can set the colour value as true, false, auto, and always.  
  
Git configuration levels  
  
The git config command can accept arguments to specify the configuration level. The following configuration levels are available in the Git config.  
  
local  
global  
system  
--local  
  
It is the default level in Git. Git config will write to a local level if no configuration option is given. Local configuration values are stored in .git/config directory as a file.  
  
--global  
  
The global level configuration is user-specific configuration. User-specific means, it is applied to an individual operating system user. Global configuration values are stored in a user's home directory.  
  
--system  
  
The system-level configuration is applied across an entire system. The entire system means all users on an operating system and all repositories. The system-level configuration file stores in a gitconfig file off the system directory.  
  
The order of priority of the Git config is local, global, and system, respectively. It means when looking for a configuration value, Git will start at the local level and bubble up to the system level.

**Git Tools**

 Git Tools  
To explore the robust functionality of Git, we should learn about git tools. Git comes with some tools like Git Bash, Git GUI to provide the interface between machine and user. It supports inbuilt as well as third-party tools.  
  
Git comes with built-in GUI tools like git bash, git-gui, and gitk for committing and browsing. It also supports several third-party tools for users looking for platform-specific experience.  
  
Git Package Tools  
Git provides powerful functionality tools such as commands, command line, Git GUI. Let's have a glimpse of some essential package tools.  
  
GitBash  
Git Bash is an application for the Windows environment. It is used as Git command line for windows. Git Bash provides an emulation layer for a Git command-line experience. Bash is an abbreviation of Bourne Again Shell. Git package installer contains Bash, bash utilities, and Git on a Windows operating system.  
  
Bash is a standard default shell on Linux and macOS. A shell is a terminal application which is used to create an interface with an operating system through commands.  
  
By default, Git Windows package contains the Git Bash tool. We can access it by right-click on a folder in Windows Explorer.  
  
Git Bash Commands  
Git Bash comes with some additional commands that are stored in the /usr/bin directory of the Git Bash emulation. Git Bash can provide a robust shell experience on Windows. Git Bash comes with some essential shell commands like Ssh, scp, cat, find.  
  
Git Bash also includes the full set of Git core commands like git clone, git commit, git checkout, git push, and more.  
  
Git GUI  
Git GUI is a powerful alternative to Git BASH. It offers a graphical version of the Git command line function, as well as comprehensive visual diff tools. We can access it by simply right click on a folder or location in windows explorer. Also, we can access it through the command line by typing below command.  
  
$ git gui  
Gitk  
gitk is a graphical history viewer tool. It's a robust GUI shell over git log and git grep. This tool is used to find something that happened in the past or visualize your project's history.  
  
Gitk can invoke from the command-line. Just change directory into a Git repository, and type:  
  
$ gitk [git log options]

**Git Terminology**

 Git has vast terminology and jargon, which can often be difficult for new users, or those who know Git basics but want to become Git masters. Let’s expound and explore the terminology behind the tools. Let's have a glimpse at the commonly used terms.  
  
Branch  
A branch is a version of the repository that diverges from the main working project. It is an essential feature available in most modern version control systems. A Git project can have more than one branch. We can perform many operations on Git branch-like rename, list, delete, etc.  
  
Checkout  
In Git, the term checkout is used for the act of switching between different versions of a target entity. The git checkout command is used to switch between branches in a repository.  
  
Cherry-Picking  
Cherry-picking in Git is meant to apply some commit from one branch into another branch. In case you made a mistake and committed a change into the wrong branch, but do not want to merge the whole branch. You can revert the commit and cherry-pick it on another branch.  
  
Clone  
The git clone is a Git command-line utility. It is used to make a copy of the target repository or clone it. If I want a local copy of my repository from GitHub, this tool allows creating a local copy of that repository on your local directory from the repository URL.  
  
Fetch  
It is used to fetch branches and tags from one or more other repositories, along with the objects necessary to complete their histories. It updates the remote-tracking branches.  
  
HEAD  
HEAD is the representation of the last commit in the current checkout branch. We can think of the head like a current branch. When you switch branches with git checkout, the HEAD revision changes, and points to the new branch.  
  
Index  
The Git index is a staging area between the working directory and repository. It is used as the index to build up a set of changes that you want to commit together.  
  
Master  
Master is a naming convention for Git branch. It's a default branch of Git. After cloning a project from a remote server, the resulting local repository contains only a single local branch. This branch is called a "master" branch. It means that "master" is a repository's "default" branch.  
  
Merge  
Merging is a process to put a forked history back together. The git merge command facilitates you to take the data created by git branch and integrate them into a single branch.  
  
Origin  
In Git, "origin" is a reference to the remote repository from a project was initially cloned. More precisely, it is used instead of that original repository URL to make referencing much easier.  
  
Pull/Pull Request  
The term Pull is used to receive data from GitHub. It fetches and merges changes on the remote server to your working directory. The git pull command is used to make a Git pull.  
  
Pull requests are a process for a developer to notify team members that they have completed a feature. Once their feature branch is ready, the developer files a pull request via their remote server account. Pull request announces all the team members that they need to review the code and merge it into the master branch.  
  
Push  
The push term refers to upload local repository content to a remote repository. Pushing is an act of transfer commits from your local repository to a remote repository. Pushing is capable of overwriting changes; caution should be taken when pushing.  
  
Rebase  
In Git, the term rebase is referred to as the process of moving or combining a sequence of commits to a new base commit. Rebasing is very beneficial and visualized the process in the environment of a feature branching workflow.  
  
From a content perception, rebasing is a technique of changing the base of your branch from one commit to another.  
  
Remote  
In Git, the term remote is concerned with the remote repository. It is a shared repository that all team members use to exchange their changes. A remote repository is stored on a code hosting service like an internal server, GitHub, Subversion and more.  
  
In case of a local repository, a remote typically does not provide a file tree of the project's current state, as an alternative it only consists of the .git versioning data.  
  
Repository  
In Git, Repository is like a data structure used by VCS to store metadata for a set of files and directories. It contains the collection of the file as well as the history of changes made to those files. Repositories in Git is considered as your project folder. A repository has all the project-related data. Distinct projects have distinct repositories.  
  
Stashing  
Sometimes you want to switch the branches, but you are working on an incomplete part of your current project. You don't want to make a commit of half-done work. Git stashing allows you to do so. The git stash command enables you to switch branch without committing the current branch.  
  
Tag  
Tags make a point as a specific point in Git history. It is used to mark a commit stage as important. We can tag a commit for future reference. Primarily, it is used to mark a projects initial point like v1.1. There are two types of tags.  
  
Light-weighted tag  
Annotated tag  
Upstream and Downstream  
The term upstream and downstream is a reference of the repository. Generally, upstream is where you cloned the repository from (the origin) and downstream is any project that integrates your work with other works. However, these terms are not restricted to Git repositories.  
  
Git Revert  
In Git, the term revert is used to revert some commit. To revert a commit, git revert command is used. It is an undo type command. However, it is not a traditional undo alternative.  
  
Git Reset  
In Git, the term reset stands for undoing changes. The git reset command is used to reset the changes. The git reset command has three core forms of invocation. These forms are as follows.  
  
Soft  
Mixed  
Hard  
Git Ignore  
In Git, the term ignore used to specify intentionally untracked files that Git should ignore. It doesn't affect the Files that are already tracked by Git.  
  
Git Diff  
Git diff is a command-line utility. It's a multi-use Git command. When it is executed, it runs a diff function on Git data sources. These data sources can be files, branches, commits, and more. It is used to show changes between commits, commit, and working tree, etc.  
  
Git Cheat Sheet  
A Git cheat sheet is a summary of Git quick references. It contains basic Git commands with quick installation. A cheat sheet or crib sheet is a brief set of notes used for quick reference. Cheat sheets are so named because the people may use it without no prior knowledge.  
  
Git Flow  
GitFlow is a branching model for Git, developed by Vincent Driessen. It is very well organized to collaborate and scale the development team. Git flow is a collection of Git commands. It accomplishes many repository operations with just single commands.  
  
Git Squash  
In Git, the term squash is used to squash previous commits into one. Git squash is an excellent technique to group-specific changes before forwarding them to others. You can merge several commits into a single commit with the powerful interactive rebase command.  
  
Git Rm  
In Git, the term rm stands for remove. It is used to remove individual files or a collection of files. The key function of git rm is to remove tracked files from the Git index. Additionally, it can be used to remove files from both the working directory and staging index.  
  
Git Fork  
A fork is a rough copy of a repository. Forking a repository allows you to freely test and debug with changes without affecting the original project.  
  
Great use of using forks to propose changes for bug fixes. To resolve an issue for a bug that you found, you can:  
  
Fork the repository.  
Make the fix.  
Forward a pull request to the project owner.

**Daily Notes**

 What is the difference between git reset and git revert?  
Git Reset: This command is used to move the HEAD and the current branch pointer to a specific commit. It can be used to undo changes and discard commits. It has options like --soft, --mixed, and --hard to control whether to keep changes in the working directory or not.  
Git Revert: This command is used to create a new commit that undoes the changes made by a previous commit. It does not remove the commit but adds a new one that negates the changes introduced.  
  
When would you use one over the other?  
Use git reset when you want to discard commits and move the branch pointer to a previous commit.  
Use git revert when you want to undo changes in a way that maintains a clear history and does not alter existing commits.  
2. What is the difference between git merge and git rebase?  
Git Merge: This command integrates changes from one branch into another. It creates a new commit that has two parent commits, showing a merge point.  
Git Rebase: This command moves or combines a sequence of commits to a new base commit. It provides a linear project history and avoids unnecessary merge commits.  
  
When would you use one over the other?  
Use git merge when you want to combine changes from different branches and preserve the branch history.  
Use git rebase when you want a cleaner, more linear history without unnecessary merge commits. It's useful for feature branches before merging into a main branch.  
3. What is the difference between git stash pop and git stash apply?  
Git Stash Pop: This command is a combination of git stash apply and git stash drop. It restores the changes from the most recent stash and removes it from the stash list.  
Git Stash Apply: This command is used to apply the changes from a stash onto the current working directory. Unlike pop, it does not remove the stash from the stash list.  
When would you use one over the other?  
Use git stash pop if you're confident that you won't need the stashed changes again and want to remove the stash immediately.  
Use git stash apply if you want to keep the stash around after applying the changes, perhaps to apply the same changes to multiple branches.  
4. What kinds of things can you do in interactive mode when rebasing?  
When rebasing interactively (git rebase -i), you can:  
Rearrange Commits: Change the order of commits.  
Edit Commits: Combine, split, or modify commit messages.  
Delete Commits: Remove unnecessary commits.  
Squash Commits: Combine multiple commits into one.  
Reword Commits: Change commit messages.

**Basic Git Command**

 Basic Git Command  
Here is a list of git commands that commonly:  
  
Git Config  
Git init  
Git clone  
Git add  
Git commit  
Git status  
Git push  
Git pull  
Git Branch  
Git Merge  
Git log  
Git remote  
Git clone command  
This command is used to make a copy of a repository from an existing URL. If I want a local copy of my repository from GitHub, this command allows creating a local copy of that repository on your local directory from the repository URL.  
  
Syntax  
  
$ git clone URL  
  
Git add command  
This command is used to add one or more files to staging (Index) area.  
  
Syntax  
  
To add one file  
  
$ git add Filename  
  
To add more than one file  
  
$ git add\*  
  
Git commit command  
Commit command is used in two scenarios. They are as follows.  
  
Git commit -m  
  
This command changes the head. It records or snapshots the file permanently in the version history with a message.  
  
Syntax  
  
$ git commit -m " Commit Message"  
  
Git commit -a  
  
This command commits any files added in the repository with git add and also commits any files you've changed since then.  
  
Syntax  
  
$ git commit -a  
  
Git status command  
The status command is used to display the state of the working directory and the staging area. It allows you to see which changes have been staged, which haven't, and which files aren?t being tracked by Git. It does not show you any information about the committed project history. For this, you need to use the git log. It also lists the files that you've changed and those you still need to add or commit.  
  
Syntax  
  
$ git status  
  
Git push Command  
It is used to upload local repository content to a remote repository. Pushing is an act of transfer commits from your local repository to a remote repo. It's the complement to git fetch, but whereas fetching imports commits to local branches on comparatively pushing exports commits to remote branches. Remote branches are configured by using the git remote command. Pushing is capable of overwriting changes, and caution should be taken when pushing.  
  
Git push command can be used as follows.  
  
Git push origin master  
  
This command sends the changes made on the master branch, to your remote repository.  
  
Syntax  
  
$ git push [variable name] master  
  
Git push -all  
This command pushes all the branches to the server repository.  
  
Syntax  
  
$ git push --all  
  
Git pull command  
Pull command is used to receive data from GitHub. It fetches and merges changes on the remote server to your working directory.  
  
Syntax  
  
$ git pull URL  
  
Git Branch Command  
This command lists all the branches available in the repository.  
  
Syntax  
  
$ git branch  
  
Git Merge Command  
This command is used to merge the specified branch?s history into the current branch.  
  
Syntax  
  
$ git merge BranchName  
  
Git log Command  
This command is used to check the commit history.  
  
Syntax  
  
$ git log  
  
By default, if no argument passed, Git log shows the most recent commits first. We can limit the number of log entries displayed by passing a number as an option, such as -3 to show only the last three entries.  
  
$ git log -3

**Inspect and Undo changes**

 Git log  
The advantage of a version control system is that it records changes. These records allow us to retrieve the data like commits, figuring out bugs, updates. But, all of this history will be useless if we cannot navigate it. At this point, we need the git log command.  
  
Git log is a utility tool to review and read a history of everything that happens to a repository. Multiple options can be used with a git log to make history more specific.  
  
Generally, the git log is a record of commits. A git log contains the following data:  
  
A commit hash, which is a 40-character checksum data generated by SHA (Secure Hash Algorithm) algorithm. It is a unique number.  
Commit Author metadata: The information of authors such as author name and email.  
Commit Date metadata: It's a date timestamp for the time of the commit.  
Commit title/message: It is the overview of the commit given in the commit message.  
Basic Git log  
Git log command is one of the most usual commands of git. It is the most useful command for Git. Every time you need to check the history, you have to use the git log command. The basic git log command will display the most recent commits and the status of the head. It will use as:  
  
$ git log  
  
The above command is listing all the recent commits. Each commit contains some unique sha-id, which is generated by the SHA algorithm. It also includes the date, time, author, and some additional details.  
  
Git Log Stat  
The log command displays the files that have been modified. It also shows the number of lines and a summary line of the total records that have been updated.  
  
Generally, we can say that the stat option is used to display  
  
the modified files,  
The number of lines that have been added or removed  
A summary line of the total number of records changed  
The lines that have been added or removed.  
It will be used as follows:  
  
$ git log --stat  
  
Git log P or Patch  
  
The git log patch command displays the files that have been modified. It also shows the location of the added, removed, and updated lines.  
  
It will be used as:  
  
$ git log --patch Or $ git log -p  
  
Generally, we can say that the --patch flag is used to display:  
  
Modified files  
The location of the lines that you added or removed  
Specific changes that have been made.  
Git Checkout  
In Git, the term checkout is used for the act of switching between different versions of a target entity. The git checkout command is used to switch between branches in a repository. Be careful with your staged files and commits when switching between branches.  
The git checkout command operates upon three different entities which are files, commits, and branches. Sometimes this command can be dangerous because there is no undo option available on this command.  
  
It checks the branches and updates the files in the working directory to match the version already available in that branch, and it forwards the updates to Git to save all new commit in that branch.  
  
Operations on Git Checkout  
We can perform many operations by git checkout command like the switch to a specific branch, create a new branch, checkout a remote branch, and more. The git branch and git checkout commands can be integrated.  
  
Checkout Branch  
To switch between branches, use the below command.  
  
$ git checkout <branchname>  
  
Create and Switch Branch  
  
The git checkout commands let you create and switch to a new branch. You can not only create a new branch but also switch it simultaneously by a single command. The git checkout -b option is a convenience flag that performs run git branch <new-branch>operation before running git checkout <new-branch>.  
  
$ git checkout -b <branchname>  
  
Checkout Remote Branch  
  
Git allows you to check out a remote branch by git checkout command. It is a way for a programmer to access the work of a colleague or collaborator for review and collaboration. Each remote repository contains its own set of branches. So, to check out a remote branch, you have first to fetch the contents of the branch.  
  
$ git checkout <remotebranch>

**Collaborating**

 Git Fetch  
Git "fetch" Downloads commits, objects and refs from another repository. It fetches branches and tags from one or more repositories. It holds repositories along with the objects that are necessary to complete their histories to keep updated remote-tracking branches.  
  
The "git fetch" command is used to pull the updates from remote-tracking branches. Additionally, we can get the updates that have been pushed to our remote branches to our local machines. As we know, a branch is a variation of our repositories main code, so the remote-tracking branches are branches that have been set up to pull and push from remote repository.  
We can fetch the complete repository with the help of fetch command from a repository URL like a pull command does. See the below output:  
  
$ git fetch< repository Url>  
  
The git command that allows to fetch a specific branch from a repository. It will only access the element from a specific branch.  
  
$ git fetch <branch URL><branch name>  
  
The git fetch command allows to fetch all branches simultaneously from a remote repository.  
  
$ git fetch -all  
  
Git Pull / Pull Request  
The term pull is used to receive data from GitHub. It fetches and merges changes from the remote server to your working directory. The git pull command is used to pull a repository.  
The "git pull" command  
The pull command is used to access the changes (commits)from a remote repository to the local repository. It updates the local branches with the remote-tracking branches. Remote tracking branches are branches that have been set up to push and pull from the remote repository. Generally, it is a collection of the fetch and merges command. First, it fetches the changes from remote and combined them with the local repository.  
  
The git pull command is syntax is:  
  
$ git pull <option> [<repository URL><refspec>...]  
  
In which:  
  
<option>: Options are the commands; these commands are used as an additional option in a particular command. Options can be -q (quiet), -v (verbose), -e(edit) and more.  
  
<repository URL>: Repository URL is your remote repository's URL where you have stored your original repositories like GitHub or any other git service.  
  
Git Pull Remote Branch  
Git allows fetching a particular branch. Fetching a remote branch is a similar process, as mentioned above, in git pull command.  
  
$ git pull <remote branch URL>  
  
Git Pull Origin Master  
There is another way to pull the repository. We can pull the repository by using the git pull command. The syntax is given below:  
  
$ git pull origin master  
  
Git Push  
The push term refers to upload local repository content to a remote repository. Pushing is an act of transfer commits from your local repository to a remote repository. Pushing is capable of overwriting changes; caution should be taken when pushing.  
Moreover, we can say the push updates the remote refs with local refs. Every time you push into the repository, it is updated with some interesting changes that you made. If we do not specify the location of a repository, then it will push to default location at origin master.  
  
The "git push" command is used to push into the repository. The push command can be considered as a tool to transfer commits between local and remote repositories. The basic syntax is given below:  
  
$ git push <option> [<Remote URL><branch name><refspec>...]  
  
Push command supports many additional options. Some options are as follows under push tags.  
  
Git Push Tags  
  
<repository>: The repository is the destination of a push operation. It can be either a URL or the name of a remote repository.  
  
<refspec>: It specifies the destination ref to update source object.  
  
--all: The word "all" stands for all branches. It pushes all branches.  
  
--prune: It removes the remote branches that do not have a local counterpart. Means, if you have a remote branch say demo, if this branch does not exist locally, then it will be removed.  
  
--mirror: It is used to mirror the repository to the remote. Updated or Newly created local refs will be pushed to the remote end. It can be force updated on the remote end. The deleted refs will be removed from the remote end.  
  
--dry-run: Dry run tests the commands. It does all this except originally update the repository.  
  
--tags: It pushes all local tags.  
  
--delete: It deletes the specified branch.  
  
-u: It creates an upstream tracking connection. It is very useful if you are going to push the branch for the first time.  
  
Git Push Origin Master  
  
Git push origin master is a special command-line utility that specifies the remote branch and directory. When you have multiple branches and directory, then this command assists you in determining your main branch and repository.  
  
Generally, the term origin stands for the remote repository, and master is considered as the main branch. So, the entire statement "git push origin master" pushed the local content on the master branch of the remote location.  
  
$ git push origin master  
  
Git Force Push  
The git force push allows you to push local repository to remote without dealing with conflicts. It is used as follows:  
  
$ git push <remote><branch> -f OR $ git push <remote><branch> -force  
  
How to Safe Force Push Repository:  
  
There are several consequences of force pushing a repository like it may replace the work you want to keep. Force pushing with a lease option is capable of making fail to push if there are new commits on the remote that you didn't expect. If we say in terms of git, then we can say it will make it fail if remote contains untracked commit. It can be executed as:  
  
$git push <remote><branch> --force-with-lease  
  
Delete a Remote Branch  
  
We can delete a remote branch using git push. It allows removing a remote branch from the command line. To delete a remote branch, perform below command:  
$ git push origin -delete edited