# DEV OPS

## GIT BASICS

### Git Terminologies

Before starting with the basics, let's explore a few terminologies:

* **Git Repository**: A directory with **.git folder**, where all the contents are tracked for changes.
* **Remote**: It refers to **a server** where the project code is present. For example, Github and Gitlab.
* **Commit**: It is similar to version. When you make changes in a file, you need to **commit the changes** in order **to save and create its new version**, which will create a unique commit hash (like version number).
* **Origin**: It is a variable where Git stores the URL of your remote repository. For example, origin => [www.github.com/username/myrepo](http://www.github.com/username/myrepo)

### Git Basic Commands

* **git init** adds .git folder and **initializes the current folder to track its changes**
* **git status** displays the current state of the staging area and the working directory, that is, which files are added/removed/modified
* **git diff** **shows the exact changes** with line and column number
* **git add** adds the changes to the staging area. If you have added a new file, this command **starts tracking** the file for modifications.
* **git commit** will **save all the changes** with a unique hash number in the local repository
* **git push** sends the changes to the remote repository (server)

### Stages in Git

The following are the **three stages** in the Git workflow:

* **Working Area**: You can **edit files** using your favorite editor/Integrated Development Environment (IDE).
* **Staging Area**: You have made the changes and **added the changes to Git**. You can **still make changes** here. (From the analogy explained in the next card) It is like taking an item out of the box, where the box is the staging area. (git add)
* **Local Repository**: You have finalized the changes and **committed** them with a new hash and proper message. (git commit)

**Remote Repository**: You can now **push the changes to online platforms** like Github or Gitlab from where others can collaborate. (git push)

### Git Stages: Analogy

Git works in three stages known as **The Three Trees**: Working Area, Staging Area, and Local Repository, which means Git maintains three states of a file. To understand this better, **let us take an analogy of a box.**

Assume you are **packing items in the house in different boxes** and labeling them to identify it later.

* Get a table and keep all the items that you want to pack underneath. (git init)
* Now you might select specific kitchen items, dust them, and club similar items (like spoons) together. (*doing changes - Working area*)
* Add the items that are ready to the box. (git add - Staged)
* Seal the box and add a label - 'Kitchen Items'. (git commit - Committed)

After git commit, a unique hash is created and the changes are saved.

### Ignore or Keep

If you **do not want Git to track any file/directory**, you can add the file/directory to .gitignore file. To track an empty directory, you need to add .gitkeep to that empty directory, as Git ignores them by default.

Ls  
git status  
git add hungry.py  
git commit -m 'first version og hungry code'  
git log  
git difftool HEAD  
echo '\*.tmp' > .gitignore  
git add .gitignore  
git commit -m "gitignore file"

## Git History

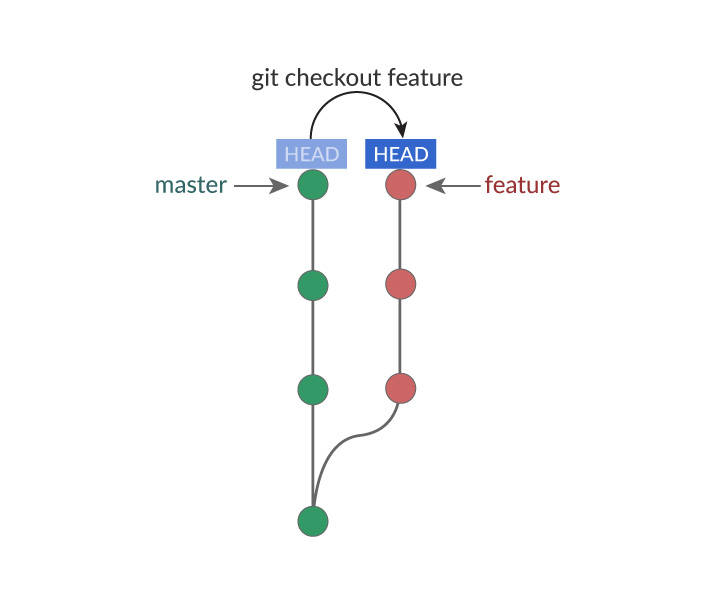
With multiple developers pushing new commits and changes, you might want to view the history of everything that happened with the repository.

Git provides the following commands, which allows you to **read and review your repo's history**:

* git log
* git show
* git diff

Before learning more about these tools, **you must first understand the HEAD and Dot operators**.

### What is HEAD in Git?

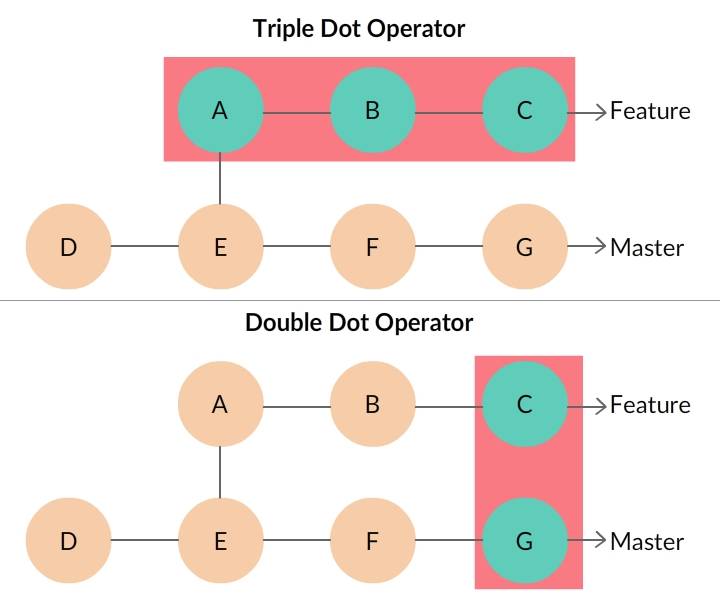


**HEAD** is a reference variable that always **points to the tip of your current branch, that is, recent commit of your current branch**.

**HEAD** can be used with the following symbols to refer to other commits:

* Tilde symbol (~): Used to point to the **previous commits from base HEAD**
* Caret symbol (^): Used to point to the **immediate parent commit** from the current referenced commit

### Dot Operators



#### Double Dot Operator

* It is the default operator in git diff
* git diff master..feature or git diff master feature command will display all the **differences from G to C** (that is, including **F and G**)

#### Triple Dot Operator

* It shows the difference between master and feature branch **starting at the last common commit E**.
* git diff master...feature command's output would be the difference in feature branch (that is, only **A, B, and C**)

### Git Log

**Git log** command shows the list of commits in the current branch. You can use it in the following ways:

* git log -2 displays the history of **last two commits**
* git log commit\_id shows the history **starting from commit\_id**
* git log filename displays the list of commits for the file

**Flags**

You can enhance the output of git log command using these optional flags:

* --oneline: Fits the log output to a single line
* --decorate: Adds a symbolic pointer to the output
* --graph: Gives a graphical representation to the log output
* --grep=<pattern>: Filters the log output and displays the output which matches the specified pattern

For example, git log --oneline

### Git Show

git show is a versatile command which is similar to git log, but with few additional features.

You can pass different arguments like:

* commit\_id
* tree

It shows the author's name, timestamp, commit message, and difference from the previous commit  
For commit range git show --oneline HEAD~2..HEAD, it gets resolved, and each commit is displayed individually.

### Git Diff

**git diff** function takes two input datasets and **outputs the changes** between them.

Datasets can be commits, branch, files and more.

git diff HEAD~1 HEAD shows the difference between two commits.

Let us take a sample **git diff** output as follows.

diff --git a/sample.txt b/sample.txt

index 8d3g6cv..f94e50c 574641

--- a/sample.txt

+++ b/sample.txt

@@ -1 +1 @@

-this is a git diff test example

+this is a diff example

* The linediff --git a/sample.txt b/sample.txt displays the input file for git diff
* --- a/sample.txt +++ b/sample.txt

The changes from a/sample.txt are marked with --- and changes from b/sample.txt are marked with +++ symbol.

### Cheat Sheet

* git log -p: Prints full details of each commit
* git log --grep-reflog=<pattern>: Shows the list of commits when commit message matches regular expression pattern
* git log –follow ./path/to/filename: Shows the history for the current file
* git show: Outputs content changes of the specified commit
* git diff --color-words: Output has only the color-coded words that have changed
* git diff –staged: Shows the file differences between staging and the last committed version
* git diff .path/to/file: Shows changes in a file compared to the previous commit

## Local to Remote

In the previous topic, whatever you have learned was mostly limited to your local machine. **To collaborate with other developers, you need to push** your work to the remote repository, and vice-versa, you need to **pull** others' work from remote to contribute your work to the project.

**In Git, remote is a repository on a server where all your team members can place the code to collaborate**

### Remote URL Types

You can keep your project code in remote servers like GitHub, GitLab or on a self-hosted server. Git sets a **default name for your remote URL as origin**

Remote URL can be one of the two types:

* **HTTPS URL** like https://github.com/user/repo.git: You can clone/push using your user name and password
* **SSH URL**, like git@github.com:user/repo.git: You need to configure SSH keys in Github/Gitlab account and local machine

Keep reading to find out the Git commands used to collaborate with other developers.

### Git Clone

**To get source code of an already existing project from remote repo** (For example, Github), you can use git clone <url> command.

For example, git clone https://github.com/facebook/react.git

This command **downloads the complete project**, all branches, commits and logs from the given remote URL (react repo here) to your local machine.

### Git Pull

Your teammate has pushed the changes to the project's remote repo where you are also working. You can now **pull the changes to your local machine** using **any one of the following commands.**

* git pull is the convenient shortcut key to fetch and merge the content.
  + git pull <remote\_name> <branch\_name>
* git fetch command downloads the remote content to your local repo, **without changing your code changes**.
  + git fetch <remote\_name> <branch\_name> fetches the content from that specific branch in remote to your current working area
* git merge command merges the fetched remote content to the local working tree.
  + git merge <remote\_name>/<branch\_name> merges the content to the specified branch.

### Git Push

To **keep your changes and work in remote repo**, you need to **push the branch** using the command **git push <remote\_name> <branch\_name>**

Git push takes two arguments, namely:

* <remote\_name>
* <branch\_name>

For example, git push origin master, where:

* origin will contain the remote URL
* master is the branch that is pushed (We shall discuss branches later in this course)

##### Working with Existing Projects

Peter is new to the project. For a particular task, he created ten new files in his local machine. His technical lead said there is a common repo where all the team members place their code. He asked Peter to push his files to the same repo. What should Peter do?

In such a scenario, you can **connect your local repo with an existing remote repo** using git remote add command.

##### Git Remote

The syntax to link your local repo with remote repo is:

git remote add <remote\_name> <remote\_url>

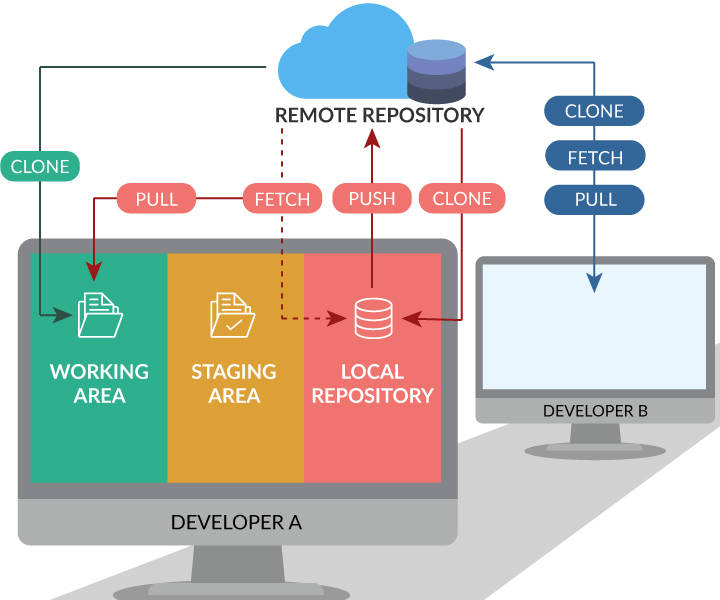
It takes two arguments, namely:

* <remote\_name>, let us take default name **origin**
* <remote\_url>, let us take https\_url **https://github.com/play/repo.git**

For example: git remote add origin https://github.com/play/repo.git

###### Note: Your local repository can be linked to multiple remote repositories as **git remote add**origin1**<url>**, **git remote add**origin2**<url>**

Cheat Sheet



* git clone: Get the complete project from remote to your local machine
* get pull origin <branch\_name>: Get the new changes from remote branch to local branch
* git push origin <branch\_name>: Send your local branch changes to the remote branch
* git remote add <name> <url>: Add a new remote repo link to your local repo
* git remote -v: List all the remote repo URLs linked to your local repo

image: git collaboration commands

##### How to Build Features?

**To build new features** without affecting the current working code, you need to:

* **Create new branch** from the master git branch <branchname>. Here you will write code for the new feature.
* **Merge** the feature branch with the master (or other branch where you want it to be). You can merge two branches locally or in remote.

Keep reading to learn more about branches in Git.