**Difference between Git push and Git pull**

Hey there! I can help you understand the difference between git push and git pull.

git push is a command used to send your committed changes from your local repository to a remote repository. It's like uploading your changes to a shared repository so that others can access them.

On the other hand, git pull is used to fetch and download content from a remote repository and immediately update your local repository to match that content. It's like downloading changes made by others and updating your local files with those changes.

So, in a nutshell, git push is for sending your changes to the remote repository, while git pull is for bringing changes from the remote repository to your local repository. Hope that clears things up for you! Let me know if you need more info.

**Difference between Git pull and Git fetch**

Hey! I can explain the difference between git pull and git fetch for you.

When you use git pull, it's like doing a git fetch followed by a git merge. It automatically fetches the changes from the remote repository and merges them into your current branch.

On the other hand, when you use git fetch, it only brings the changes from the remote repository to your local repository but doesn't automatically merge them into your current branch. This allows you to review the changes fetched before deciding to merge them.

So, in summary, git pull fetches and merges changes automatically, while git fetch only fetches changes for you to review before merging. Hope that clarifies things for you! Let me know if you need more information.

**Git Commands: -**

git config –global user.name “[name]”

git config –global user.email “[email address]”

git init (To Change a Normal directory into git repository)

rm -rf .git (To convert git repository to Normal directory)

git clone remote URL

git add . (for all files)

git add file1 (for particular file)

git commit -m “1st commit” (this command records or snapshots the file permanently in the version history)

git status (list the all files that have to be committed)

git log (To know the history of commits done)

git log -2 (for last two commits)

git log –onelline

git diff: is is used to show the changes between two commits and the data difference can be understood with the below points.

New lines added will be In Green colour

Old lines will be displayed in White colour

Deleted lines will be displayed in Red colour

git diff 1st commit id 2nd commit id (difference between two commits)

newly added shown in green & deleted lines in Red

git log -p (Display the full difference of each commit)

git log –-stat (Include which files were altered and the relative number of lines that were added or deleted from each of them)

git log –-author = “<pattern>” (search fo commits by a particular author)

git log –grep = “<pattern>” (search for commits with a commit message that matches <pattern>

git log <since>…<until> (Show commits that occur between <since> and <until>. Args can be a commit ID, branch name, HEAD or any other kind of version reference.

git add . && git commit -m “1st commit” (combined two commands by &&)

git fetch (from remote to Local Repo)

git fetch = git fetch + git merge

git merge (From Local Rep to PWD)

git pull (To pull file from remote Repo to PWD)

It only brings the changes from the remote repository to your local repository but doesnot automatically merge them into your current branch. This allows you to review the changes fetched before deciding to merge them.

git branch (to list all branches)

git branch branch-name (to create branch)

git checkout branch-name (to navigate particular branch)

git merge branch-name (This will help to combine the changes from two or more branches into a single branch)

**Note: -** File created in workspace will be visible in any of the branch workspace. Until you commit once you commit, then that file belongs to that particular branch.

git branch -D branch-name (To delete branch)

git reflog (To bring back the deleted branch)

git checkout -b branch-name HEAD@{1}

**git reflog:** **-** If we delete branch unfortunately & to retrive that branch data by using git reflog.

**cherry pick: -** Process of picking a particular commit from one-branch to antoher-branch.

git cherry-pick commit ID

Note:- If you want to move sequence of commits from one-branch to another-branch we always goes through git rebase. But if you want to get particular commit from one-branch to another-branch we always go through Cherry-pick.

**git rebase: -** Process of moving or combining a sequence of commits to a new base commit.

git rebase from-branch to-branch

**Merge conflict:-** This merge conflict arises when git cannot automatically resolve code differences between two commits.

**Difference between fork and clone:**

Developers who work on a common codebase will clone the repository and then perform push and pull operations to synchronize their changes.

In contrast, a fork creates a new codebase and updates to the fork are not synchronized with the original repository.

**HEAD, git checkout, git revert, git reset.**

git show HEAD (To see the latest commit)

git checkout commit id (This command is used to see what changes we did in that particular stage)

**Note:-** git checkout is not deleting our commit this command just go back to their particular commit.

**git revert: -** Undo the changes.

git revert commit-ID

git reset (mixed, soft, hard)

**git reset:** **-**By default it is taken git reset --mixed

**git reset --mixed: -** This is used to move file back from “Committed stage” to “working directory stage”.

**git reset –soft: -** this is used to move file back form “Committed stage” to “staging area”.

**git reset –hard: -** This is let you go back to where you were, but it will discard your local changes, which you do not want.

git reset --mixed commit-ID (COMMIT ID IS GIVEN THE BELOW ID WHICH YOU WANT)

git reset --soft commit-ID

git reset –hard commit-ID

**git amend: -** if you want to modify the latest commit. For Example I have committed some code. Then after found that I forgot to modify or I forgot to commit to another files, instead of committing a new file. Why can’t I modify this latest commit? So we have option called amend.

git amend is used to modify the most recent commit.

git show commit-ID (Details about particular commit-ID)

git diff –cached (You can see the changes between commits)

git commit –amend (It is used to modify the most recent commit)

**Stashing: -** it is useful when we need to switch branches without commit. And current branch also.

Only applies modified files not new files.

Whenever doing stash the file should be in staging area.

git stash save, list, apply, pop, drop, clear

git stash save -m “this is my 1st stash” (Stash the uncommitted changes i.e. it is stored in a “stash stack” or This is used to save the file in a temporary area and later it can be reworked)

git stash list (Show all the stashes)

git stash apply stash id (git stash apply stash@{0}) (This is used to reapply the changes in the working copy without removing the changes from the stash)

git stash pop stash id (git stash pop stash@{0}) (This removes the changes from the stash and reapplies the changes in the working copy)

**git stash pop:** git stash pop command helps us to remove or throw away the latest or the topmost stash. The git stash pop command will take the content from the stash file and then apply those changes to our current working file. The git stash pop command enables users to re-apply the commits using only the git stash pop command. So (git stash pop = git stash apply + git stash drop)

git stash drop (This is used to delete the unwanted stash id’s which are not applied)

git stash clear (To delete all the saved stashes at once irrespective of they are applied or not)