**Git Client:**

<https://www.git-tower.com/windows?utm_source=learn-website&utm_campaign=git-faq&utm_medium=easy-in-tower&utm_content=git-stash>

<https://git-scm.com/downloads>

-------------------------------------------------------------------------------------------------------------------------------------

**Push a folder with its content**

Ex: C:\Users\Prakash\_pc\git\test\demo1

Inside demo1 if you have multiple files (file1, file2, file3, file4 ……)

Then how to push whole demo1 folder to remote repo

git add demo1

git add demo1/\*

git push

-------------------------------------------------------------------------------------------------------------------------------------

**Git Clone (Remote repo to Local Repo)**

Create a new folder or directory **gitdemoclonelab**

**mkdir gitdemoclonelab**

git clone https://github.com/prakashk0301/devops-nov-18.git

cd devops-nov-18

https://git-scm.com/docs/git-clone

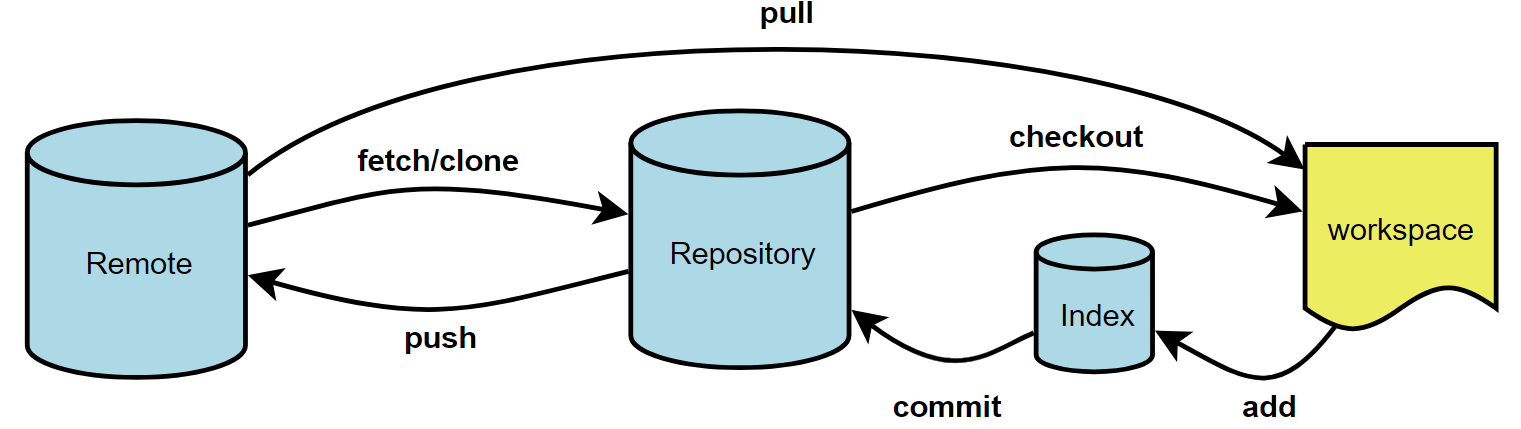
-------------------------------------------------------------------------------------------------------------------------------

**Difference between Git Pull and Git Fetch:**

GIT Pull (Remote Repo to Working)

Git Pull=Git Fetch + Git Merge

**git pull** does a **git fetch** followed by a **git** merge. This operation never changes any of your own local branches under refs/heads, and is safe to do without changing your working copy.



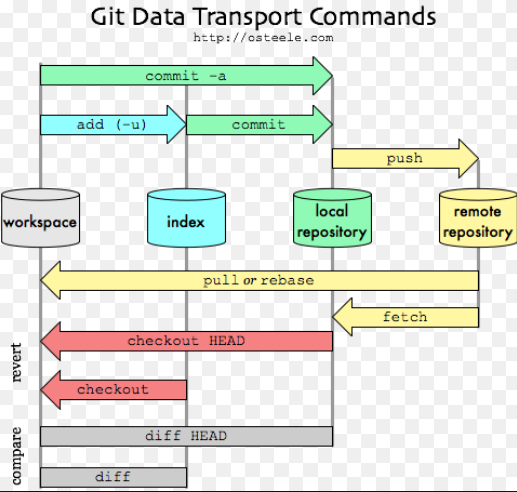
**Demo:** Create a directory then use git pull command

git pull <repo url>

Ex: git pull https://github.com/prakashk0301/devops-nov-18.git

When downloading content from a remote repo, git pull and git fetch commands are available to accomplish the task. You can consider git fetch the 'safe' version of the two commands. It will download the remote content but not update your local repo's working state, leaving your current work intact. git pull is the more aggressive alternative, it will download the remote content for the active local branch and immediately execute git merge to create a merge commit for the new remote content. If you have pending changes in progress this will cause conflicts and kickoff the merge conflict resolution flow.

**git clone vs pull:**

**Git pull** means you are fetching the last modified repository. **Git** push means you are returning the repository after modifying it. In Layman's term **Git clone** is downloading and **Git pull** is refreshing. **clone**: copying the remote server repository to your local machine. 

* **git revert** is used to **undo a previous commit**. In git, you can't alter or erase an earlier commit. (Actually, you can, but it can cause problems.) So instead of editing the earlier commit, revert introduces a new commit that reverses an earlier one.

EX: create a new directory, create a new file and **commit** it.

Now if you don’t want those changes then use revert command

**Syntax: git revert <commit ID>**

**Lab:**

*git init*

*vi index1.html*

*git add index1.html*

*git commit -m "created index1.html"*

*vi index2.html*

*git add index2.html*

*git commit -m "created index2.html"*

*git status*

*git log --oneline*

*git revert <first commit ID>*

*ls ----->we can see only index2.html. Git has undo chnages of first commit ID*

*vi index3.html*

*git add index3.html*

*git commit -m "created index3.html"*

*git log --oneline*

*ls*

*git revert <reverted commit ID> ---------> Git has undo changes of first revert commit.*

*ls --------------------> Now we can index1.html*

*===========================================================*

**git reset** :

* Use this to return the entire working tree to the last committed state. This will discard commits in a branch or throw away uncommitted changes!
* Changes which commit a branch HEAD is currently pointing at. It alters the existing commit history.
* Can be used to unstage a file.

**Use Cases:**

1. **Unstage the changes and file moved to working area.**

**Lab**: vi <filename>

git add <filename> ------------file moved to index area.

git reset HEAD <file name>

git status -------------file moved to working area.

1. **Is used to completely delete particular commit ID**

**Lab:** vi <file1>

git add <file1>

git commit -m “committing file1”

vi <file2>

git add <file2>

git commit -m “committing file2

git log --oneline

**Here you will get 2 commit ID.**

**Now if you want to delete particular commit then,**

git reset HEAD ~1 **---------where ~1 is pointing to latest commit and we will be having only file2. (if you put HEAD~4 means it remove latest 4 commits----commit 1,2,3,4)**

1. **is used to undo changes in your Index area that haven't been committed yet.**

**Lab**: create a new directory, create a new file and save it.

git add

don’t commit it.

Now if you don’t want those changes then use reset command

**Syntax:**

**git reset --hard -------->Its very dangerous.**

**git reset --soft --------> Safe to use.**

Reset hard: Reset the index and working tree. Any changes to tracked files in the working tree since <commit> are discarded.

Reset Soft: Does not touch the index file or working tree at all, but reset the head to <commit>. This leaves all your changed files “changes to be committed” as git status would put it.

**=======================================================================**

* **git checkout** Clear (discard the changes of working area) your **working area**.

git checkout -- <file name>

discard all the changes of multiple files: git checkout .

**Git rebase**

With the rebase command, you can take all the changes that were committed on one branch and replay them on another one.

Ex: you can take the patch of the change that was introduced in b1 and reapply it on top of b2. In Git, this is called *rebasing*

**Demo**: Create a new branch, create a file and commit it.

**Switch** back to your **origin** branch. Now if you want to take all the changes which you made in your new branch then-

**git rebase <new branch name>**

<https://git-scm.com/book/en/v2/Git-Branching-Rebasing>

-------------------------------------------------------------------------------------------------------------------------------

**Git Merge vs Rebase**

The major benefit of rebasing is that you get a much cleaner project history. First, it eliminates the unnecessary merge commits required by git merge. Second, as you can see, rebasing also results in a perfectly linear project history

------------------------------------------------------------------------------------------------------------------------------

**Git Fetch vs Pull:**

The git fetch command downloads commits, files, and refs from a remote repository into your local repo. Fetching is what you do when you want to see what everybody else has been working on. It lets you see how the central history has progressed, but it doesn’t force you to actually merge the changes into your repository. Git isolates fetched content as a from existing local content, it has absolutely no effect on your local development work. Fetched content has to be explicitly checked out using the git checkout command. This makes fetching a safe way to review commits before integrating them with your local repository.

Step 1. Create a folder-create a file then commit it.

Step 2. Create a remote repo

Step 3. At local

**git remote add origin <repo url>**

**git branch -a**

Step 4. Then push to remote repo

git push origin master

Step 5. Create a file at remote repo.

Step 6. Use git fetch to fetch updates

**git fetch origin master -→**(Still you can’t see updates, file which we created at remote repo)

**git branch –a** --→to list all branches

(You can see updates are not merged with our master branch, updates are available only at remote branch)

git log origin/master ----→(You can see commits of remote branch)

git merge origin/master

Reference URL

<https://git-scm.com/book/en/v2>

**Git Stash: Command that allows you to store the modifications you’ve made in our working directory and go back to a clean working directory.**

**git stash -------🡪** Clean your working directory

**git stash pop -------🡪** Continuing where you left off

**Lab:**

*vi index1.html*

*git add index1.html*

*git status*

*git stash*

*git status*

*vi index2.html*

*git add index2.html*

*git commit -m ""committing index2.html"*

*git status*

*cat index1.html*

*git stash pop*

*cat index1.html*

*git status*

*git add index1.html*

*git commit -m "committing index1.html"*