# Git VCS

| | |
|---|---|
| Created by | Aadeesh Bali |
| Author | Aadeesh Bali |
| Last edited | @July 1, 2024 12:58 PM |
| Tags | Strategy doc |

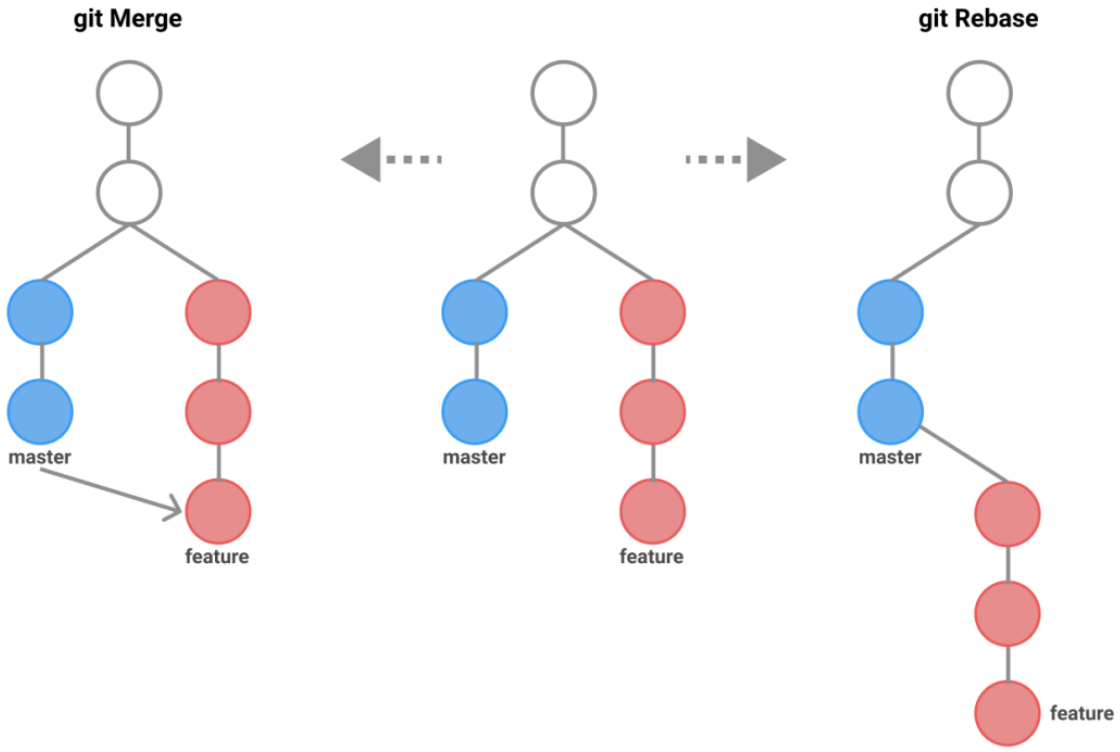
# Git Commands

# Local Branches

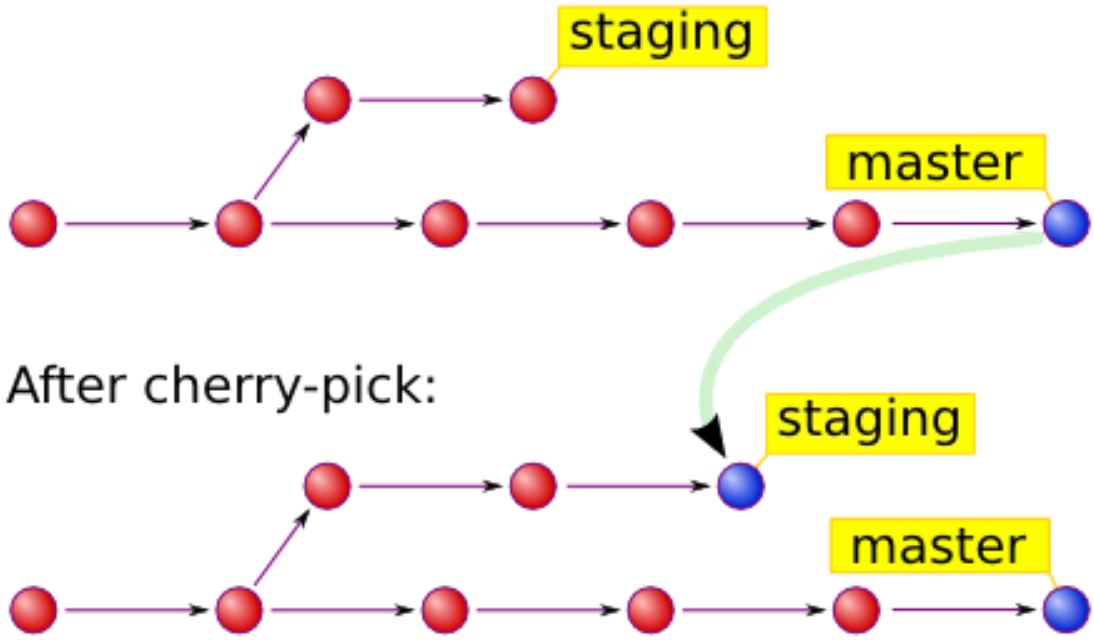
## Branching and merging

1. git commit - taking a snapshot of the changes made.
2. git checkout -b branchName - create and checkout a new branch.
3. git merge branch_name - you need to checkout the branch on which you want to merge first then use this command to merge another branch onto the checked out branch. Creates a new commit which points to both the branches latest commits.
4. git rebase branch name- you need to checkout the branch you want to rebase first.Creates a copy of the commit and puts it on the other branch as latest commit.
5. git rebase <destination> <source> - stacks the commits of branch2 on branch1, no need to checkout branch2.

**git merge is a way of combining changes from one branch (source branch) into another branch (target branch) whereas git rebase is a way of moving the changes from one branch into another branch.**




**Use cherry-pick when you want to apply specific commits from one branch onto another.**



# Moving around in git

## In order to use relative refs you don’t need to checkout to any branch or commit if you have a starting point use it as reference.

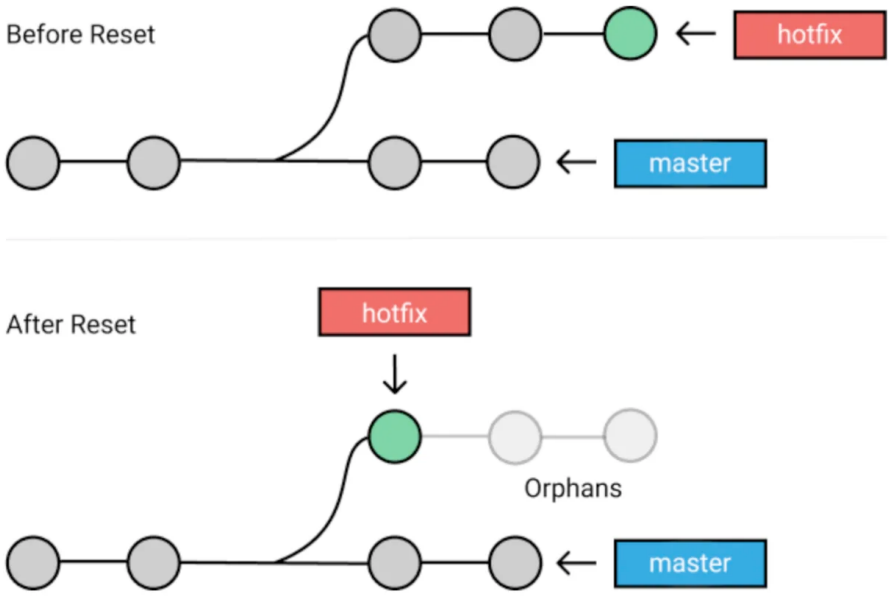
## Eg: If my current checked out branch is main and you want to move to parent commit of branch bugFix just use ‘git checkout bugFix^’.

1. git checkout <commitHash> - to move the head to that commit, makes a new branch.
2. git checkout <commitHash> files - restores the files as in mentioned commit without making a new branch.
3. Using ^ operator git checkout <branch/commit/HEAD>^ - will move the head one commit directly above the mentioned branch/commit/HEAD.
4. Using ~ operator git checkout <branch/commit/HEAD>~<num> - will move the head <num> commit directly above the mentioned branch/commit/HEAD.

## Branch forcing - assigning a branch to a particular commit using -f operator. No need to checkout on that branch.

## Eg: git branch -f main HEAD~3 - moves the main branch to three commits above the HEAD no matter where the head currently resides.

# Reversing changes in git




1. git reset HEAD~1 - moves the current branch to the mentioned state.Works only for local repository
2. git revert HEAD - makes a new commit which happens to have all the changes that are supposed to reverse the changes of the HEAD commit.
3. git reset filename - takes the file out of the staging area.
4. git checkout - - filename - takes the file out of the working area.Reverts back all the files to previous state.

**This is enough to handle 90% of the changes.**

## Moving around work in git

1. git cherry-pick <commit hash code> - adds the mentioned commits to the branch in the given order. You need to first checkout the branch on which you want to add the commits.

Eg. git checkout main →git cherry-pick C3 C4 C5 →Output : commit C3 C4 C5 will be added to main in the mentioned order.

1. git rebase -i HEAD~<num> - opens up a dialog box to select the commits to be moved.

# Remote Branches

## Remote branches reflect the state of remote repositories (since you last talked to those remote repositories). They help you understand the difference between your local work and what work is public.

1. git fetch - performs two main functions:

→downloads all the commits from our remote repository

→reflects the current state of the remote branches i.e on which commit does our local repository points.

1. git pull - does the combination of git fetch and git merge/rebase/cherry-pick.
2. git pull -rebase - adds the latest commit to our local repository and updates the origin/main in our local repository.
3. git checkout -b local_branch remote_mainbranch- sets the local_branch branch to track the remote main branch.
4. git push origin <source>:<destination> - pushes commits from source i.e local to destination i.e remote.
5. git fetch origin <source>:<destination> - fetches commits from source i.e remote to destination i.e local.
6. git push origin :branch - keeping the source empty i.e pushing nothing to a remote branch deletes it .
7. git fetch origin :branch - just creates a new branch on local.
8. git pull origin <source>:<destination> - same as fetch just merges the fetched into the currently checked out branch.
9. git push - -set-upstream origin branch_name - It makes a new branch on the remote and sets the local branch to track the remote branch.

## Why arrows point to the previous commit →It represents that the current commit holds the changes and the rest is similar to the previous commit.

## Merge conflicts

- Why →Happens when there are differences on the same component reflected by two commits belonging to the different branches.
- Solution →You can choose to keep the desired changes in the VS code directly.
- How to resolve merge conflicts on computer:
  - First checkout the main branch
  - Pull the changes from the remote
  - Checkout the conflicting branch
  - Merge the main into the conflicting branch
  - Keep the code you want to keep using the VS code.
  - Push the conflicting branch to the origin

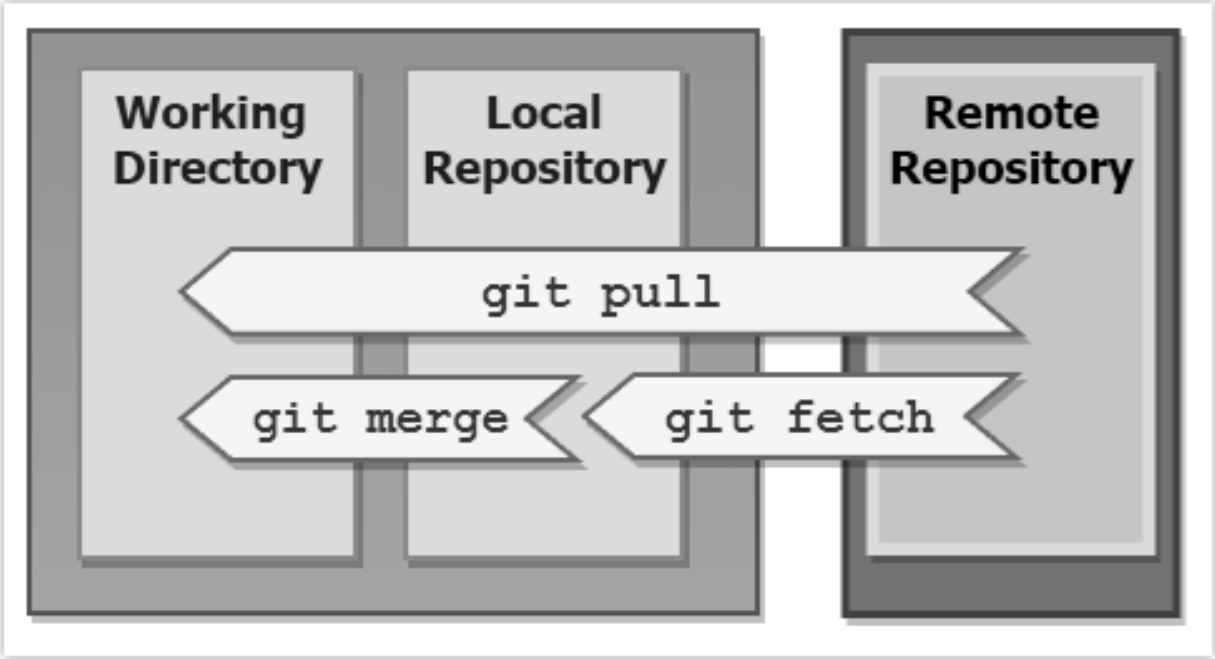
## Some frequently used commands

1. git stash - useful when you're working on a branch and you want to switch to another branch without committing your work on the current branch.
2. git stash pop - get those uncommitted changes when you switch back to the branch.
3. git stash apply - to get the changes from the stash without emptying the stash.
4. git stash -u - to stash untracked files.

## Difference between git pull and git fetch

| git pull | git fetch |
| --- | --- |

| | |
|---|---|
| Downloads the files from our remote repository as well as puts it onto our working directory i.e local repository | Downloads the files from our remote repository but doesn't puts it onto our working directory i.e local repository |
| git pull = download + merge | git fetch = download |

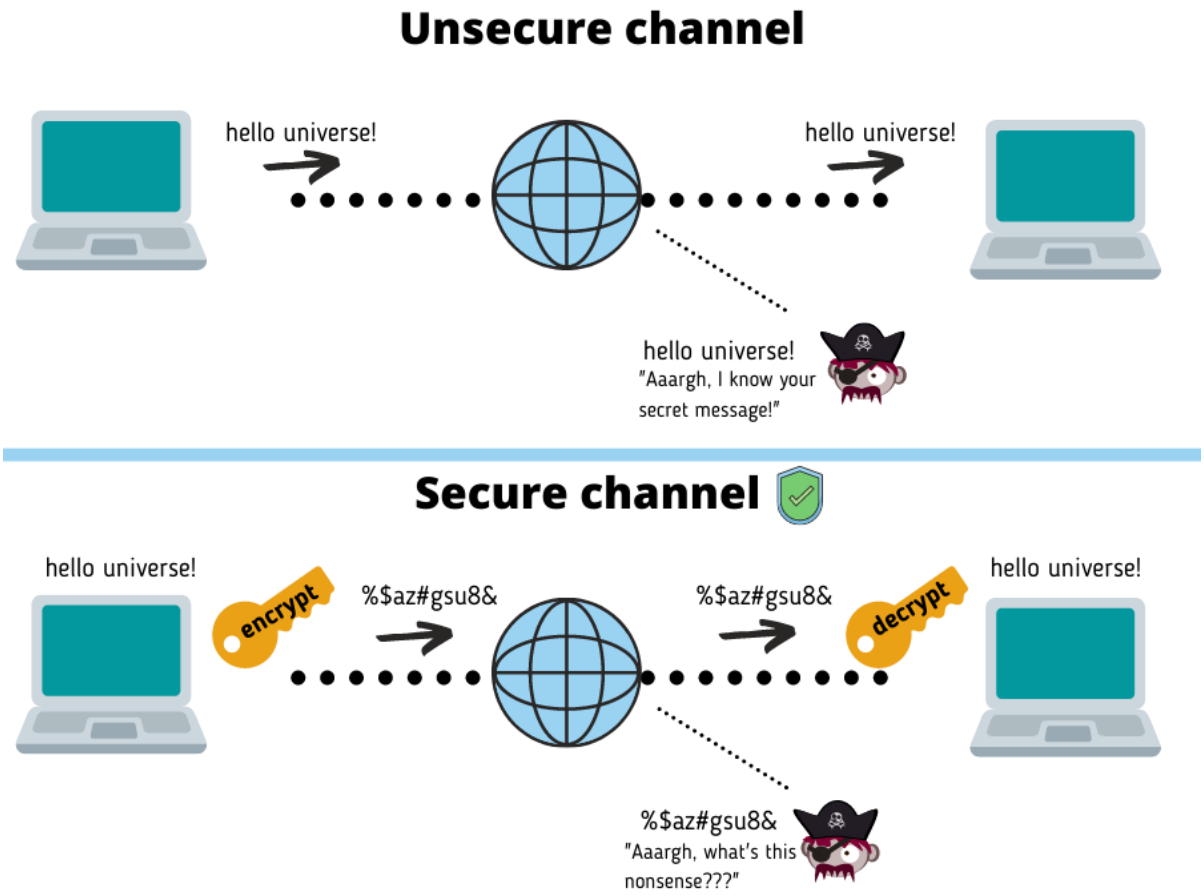



## Merge strategy options

git merge -s <stratergy> branch1 branch2

1. Recursive - Default merge strategy while merging and pulling operation
2. 3 way merge
3. Fast-forward merge

## SSH authentication




# Resources

Git Commands Visualization

Git cheat sheet