# Git architecture



# Setting user name

To generate SSH keys

<https://docs.github.com/en/authentication/connecting-to-github-with-ssh>

once SSH keys are generated then u have to upload that key to ur github account

|  |  |
| --- | --- |
| If u want to see which username and email has been configured | git config user.email |
| If u want to see which name has been configured | git config user.name |
|  |  |
|  |  |

# Reference urls

<https://www.asciiart.eu/mythology/fantasy>

This have lot of diagrams as content, u can happily use

# Git terminal commands

Git is called as distributed because , we have both local and remote

Svn doesn’t have local git repo, it will have only remote repo, since git is distributed we have both local and remote

| **Centralized** | **Distributed** |
| --- | --- |
| You can keep changes only in the server | You can keep changes locally (commit) as well |
| Changes can be merged in the server (remote) alone | Changes can be merged locally as well as remotely |

* git clone: Get the complete project from remote to your local machine
* git 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

## set the email after installing git

$ git config --global user.name "First Last"

$ git config --global user.email "myemail@domain.com"

* **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)
* git log
* git show
* git diff

**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
* 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
* 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.
* 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>**

Delete a folder

Rm –rf <folder name>

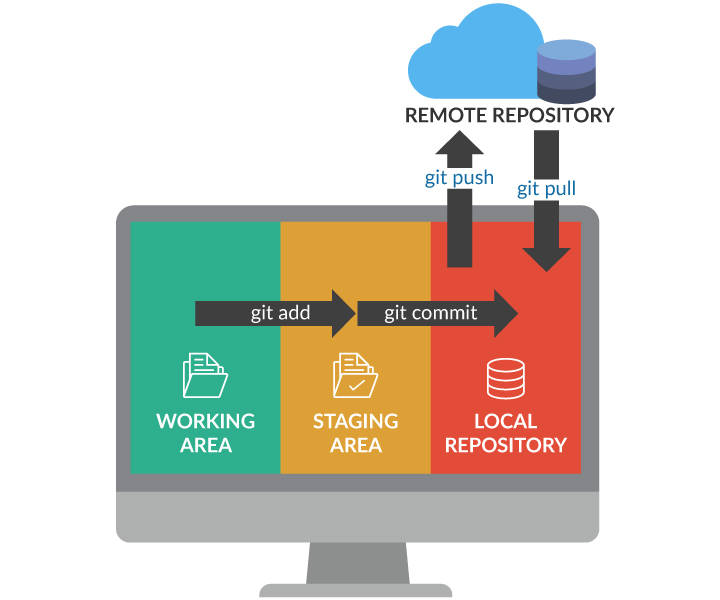
//here –r means –recursively, -f means force

Most used Git commands

**git status**

it will show the files which files are changed on the system

**git add**



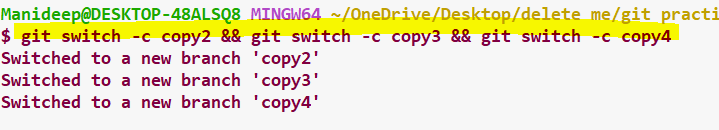
All GIT commands

Execute all commands at once

1. We can execute multiple commands using “&&” operator

touch two.txt && git add . && git commit -m "c2"

// here 3 commands are executed using && operator



Here 3 commands are executed at once using && operator

To initialise a local git repository

// This will create a local git repository

git init

// this will create a local empty git repo inside the specified folder

git init <folder name>

To open windows folder from current git command prompt

start . or explorer.exe .

here dot represents current directory

this start . will even work in windows application also

Connect local repo to remote repo

Lets say u initialized a empty git repo using “git init” and If u want to connect that local repo to remote git repository then follow

We should point the local repo to remote repo by creating a remote

& create an alias name for the remote git URL and point our local repo to remote git url

//pointing to a remote git URL

|  |  |
| --- | --- |
| Creating a remote  //we can create many remotes | git remote add <alias name for the url> <remote git url>  git remote add origin <https://github.com/manideep-vv/Forked-Junit-5-samples.git>  git remote add canadaorigin <ur git url>  here we have set origin as an alias name for the git url, so for everytime for git pull and push we can use that alias name instead of git url |
| See the remote repositories list | git remote //this will give only alias name, it wont give url  or  git remote –v //// this –v will give alias name and remote repository pointing url |
| pulling a specific branch | git pull <remote-alias-name> <branch name>  git pull canadaorigin b1 |
| Set the upstream and  Push to origin | git push –u <remote name> <branch name> // set the upstream//here –u means upstream  git push -u usa copy1//here usa is remote repository name, copy1 is branch name  if u use –u (save the upstream) flag for first time, then next time onwards it won’t ask u the remote name and branch name  u can straight away do git push  git push <remote name> <branch name>  git push canadaorigin  way-2: local branch may be different and remote branch name can be different  git push origin <local branch>:<remote branch>  // Map local dogs branch to remote cats  Git push –u origin dogs:cats // here local dogs branch will be mapped to remote cats branch  //when u do git clone, remote will be automatically set |
| Renaming the origin | git remote rename <current alias name> <new alias name>  git remote rename origin Canada  after that u can pull as  git pull Canada b2 |
| Flow | //here usa is the remote repository url alias name  git remote add usa <https://github.com/manideep-vv/github-demo-novel.git>  git pull usa  git push usa master |
| Deleting remote | git remote remove <remote name>  git remote remove usa |

Generally if u want to pull remote changes to local,

1st best way is git clone <git url> // this will fetches or downloads all branches

2nd option is create a empty git repo and add a remote and pull using



**Clone- pull or check out the code**

1. Best way is git clone 2) another way is git pull <remote url alias >

git clone <url ends with .git>

1. Initialize an empty git repository using (git init) and do

git pull <https://github.com/manideep-vv/Forked-Junit-5-samples.git>

1. With out using url everytime , set a remote and create a alias for above url

|  |  |
| --- | --- |
| Cloning a git repo  **This command will download complete project,** all branches, commits and logs | git clone <git url name> - the problem with this is it will fetch all git history  git clone --depth 1 <repo-url> -- use this if u don’t want git history – in this case it will fetch only latest commit  This saves **only the latest snapshot**, drastically reducing .git size  git count-objects –vH -- this will calculate the size of .git folder which contians the history  git clone <https://github.com/manideep-vv/SpringPaintBrushes-1.git>  when u clone a repo, remote repo alias name.. everything will be autoconfigured, u can get remote repo details from  git remote -v |
| Initialise an empty git repo | git init |
| Configure the remote url to our local repo and create alias name for that url | git remote add <desired alias name for the url> <remote git url>  git remote add maniorigin https://github.com/manideep-vv/Forked-Junit-5-samples.git  //here maniorigin is an alias name for the git url |
| Pull specific branch | git pull <remote-alias-name> <branch name>  git pull maniorigin b1  // pull from that url and mention that specific branch   * // this will fetch and merge the content * Approach- 3-without giving branch name   git pull  the current branch code will be pulled |

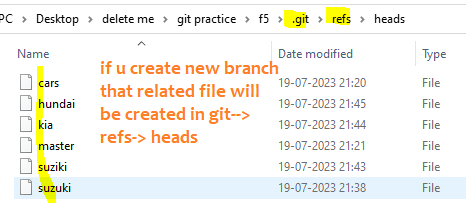
Branches

1. If u want to see what are all the branches available in local then type “git branch –r” (here r stands for recursive)

|  |  |
| --- | --- |
| If u want to know on which branch you are | git status |
| To see the list of all branches and  if u want to see on which branch u are then | 1) git branch  2) git branch –r  // here r stands for recursive TO SEE The remote tracking branches and all.. |
| Create a branch from HEAD | 1. Way-1   git branch <branch name>  git branch Suzuki//now new branch Suzuki will be created  2)way-2  git checkout -b <branch-name>  3) Way-3// while switching if branch is not there it will create the branch  git switch -c <desired new branch name > |
| To see all the commits on that branch | Git log --oneline |
| clone with latest commits alone | git clone --depth 10 <repository\_url> |
| Switch to another existing branch  // this will not only switch and it will connect local branch to remote branch | git switch <existing-branch-name>  2)git switch cars  3)git checkout <branch-name> |
| Create new branch from HEAD and switch to new branch | git switch -c <desired new branch name >  git switch -c hundai  git switch - //to switch back to where u were earlier |
| Checkout- this is also used to switch to another branch= checkout means switch & many other functionalities | git checkout <existing-branch-name>  git checkout cars |

Deleting a branch

|  |  |
| --- | --- |
| Delete branch | git branch –d <branch-name>  git push origin -d <branch-name>// after deleting locally u have to push the deleted branch to remote, so that in remote also that branch will be deleted automatically  ex:- if u want to reset all changes done locally, then delete and re pull |
| If u want to see all commits and their names.  If u see the commit names- later u can checkout that particular commit | git log --oneline |
| checkout particular commit -- then head will be detached (because at that particular commit, many branches could be there, many branches would have merged at that point) | git checkout <particular-name>  // if u want to checkout latest-1  git checkout HEAD~1 |
| Switch to earlier branch where u were on  Or  If u want to re-attach head | git switch - |
| Re-attach head-when head is detached then again u have to switch to reattach the head, some other branch –when u checkout a particular commit –then ur head will be detached | git switch master |
| Renaming a branch | git branch -m old-name new-name  // rename current branch to main  git branch -M main //here M means modify |



Git help

|  |  |
| --- | --- |
| To get imp commands | git help |
| To get all commands | git help –a  Or  git help –all  give down all to go through all the list,  press q to exit |
| If u want to specific command in detail  // here the commands will be shown in console | git <command> help  git init help  Or  git remote help |
| To see the documentation for that command | git help <command name>  git help merge  git help push |

To see all changes in a tool

Git difftool HEAD

To discard all the changes in the working directory

Git checkout -- <file name>

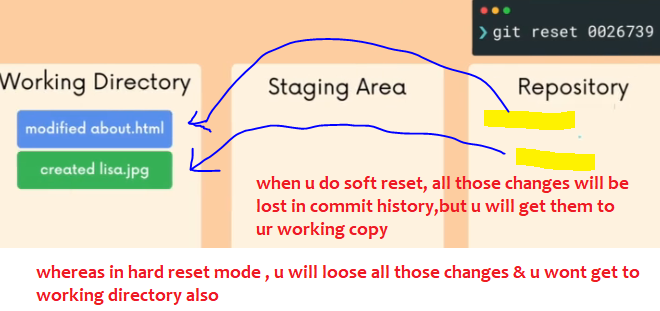
The above commands will work only for staging area,

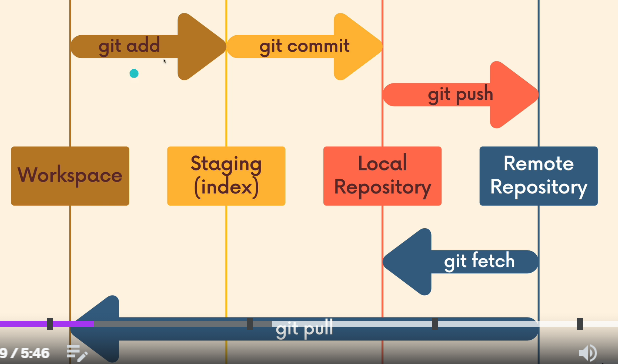
I mean it will revert those files present in staging area.

Commit related

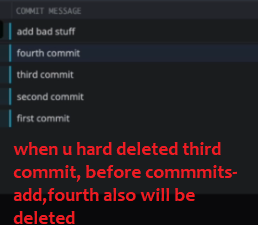
|  |  |
| --- | --- |
| If u want to see the commit history, with commit hashes | git log ---oneline |
| To see all modified files, what all are staged and un-staged changes | git status |
| To discard all changes made to that file and if u want to have all changes as per HEAD | git restore <filename.txt> (this might work only when file is not in staging area)  --way-1  git checkout -- kiaengine.txt  git checkout –- <filename.txt>  or –way -2  git checkout HEAD <file-name>  git checkout HEAD kiaengine.txt |
| Navigate to a particular commit | git checkout <commit-hash>  // if u want to see all the changes made till that commit, then use above  This will leave us in detached stage, then u should use git switch <branch name>  Git switch - /. To go where we were earlier |
| To revert all the local files or reset all local changes | git reset –-hard |
| Add all files from workspace to staging area | git add .  // to add single file to stage area  git add <file-name> |
| Commit – this will move that files from staged are to local repository | git commit –m “fc”  //here –m means commit message  git commit –a –m “fc”  //to add all files present in stage area(not clear) & commit at once |
| Un-staging file | git restore –-staged <file-name>  git restore --staged vehicle.txt  // command says restore the staged file and move back to un-staged area |
| **GIT Reset Soft *– to undo the commits-***  ***If u want to reset/delete that particular commit in commit history in soft way*** –if u reset, that commit from the commit history will be deleted, but all those changes u will get into ur working directory (because it is soft way of doing it)  when u mistakenly committed then u can clear that commit in remote repo- | git reset <commit-hash>  means reset to that point, all commits above that id will be removed  U can see commit hashes using git log –oneline  //it removes the commit from commit history, but it won’t remove the changes from working directory  //remember when u did a mistake in a commit(some wrong in that file), then reset that commit  //when u do soft reset- u will not loose those changes  //remember when u want to delete c1, u have to reset below commit |
| **Git reset hard*-Delete the commit and delete the changes from working directory*** | git reset –-hard <commit-hash name>  //if u delete it ,this commit won’t be deleted , till that commit all previous commit will get deleted |
| **GIT Revert- creates a new revert commit**  git reset & git revert both are to undo the changes but they accomplish in diff way | git reset <commit-hash>  // git reset- this will delete the existing commit & moves the branch pointer backwards  //git revert – this will create a new revert commit & undo all the changes |
| ***Push – this will move the files from local repo to remote repo*** | If remote and upstream is already configured just “git push” is enough   1. if upstream is not configured then use below   git push <remote\_name> <branch\_name>  git push canadaorigin b1  //to the Canada origin remote the code will be pushed to b1 branch  // if u cloned a repo,then remote and upstream will be set automatically  Whereas if u manually set the remote and pulled, then u have to set the upstream  git push -u <remote name> <local branch name>:<remote branch name>  here –u stands for setting upstream  ex:-1 set the upstream and push  git push -u us master:cellphone  //in above master will be local branch name and cellphone will be remote repo name  Ex:2  git push -u us master:main  in local current branch name is master, this branch will be synced to remote main |
| ***Git fetch*** will update the local remote tracking branch with latest changes from remote repository (local branch is separate from remote tracking branch but both are local branches only) | This command will download the changes from remote repo to local repo (not into our workspace)- its like go and download the latest changes ,but don’t screw up my working directory   * ur entire remote repository will be synced to local repository  1. Git fetch <remote> //which branch will be downloaded and kept to local repo? 2. Git fetch <remote> <branch name> // fetching a specific branch 3. Git fetch |
| ***Git pull =git fetch +git merge***  ***Git merge=*** means it will update my current branch with whatever the changes present in locals remote tracking branch | This command will get & integrate these changes from remote repo to local workspace directly  • git pull <remote\_name> <branch\_name> // this will fetch and merge the • content • git fetch <remote\_name> <branch\_name>//this will fetch whereas it won’t merge the content , but git pull will fetch and merges the content  If u just do git pull, current branch (output of git branch command) will be pulled  ***git pull*** //-- if u didn’t mention <remote name> then by default it is origin, if u didn’t mention branch name, then by default it is remote tracking branch/current branch where u are |
| ***Checkout- To view remote files here*** | Git checkout remote/branch name  Git checkout origin/food this will checkout the as it is remote version and sometimes this will leave us in detached state, u can come out using “git switch –“  Then u will be where u are / u will be back to ur previous state/ branch |
| ***Git merge*** | ***Git merge <target branch name>***  The current branch changes will be merged with mentioned branch name  git branch // 2022- says u will be on branch 2022  ***git merge develop*** means current 2022 branch will be merged with develop branch code, means 2022 branch will have develop branch code |

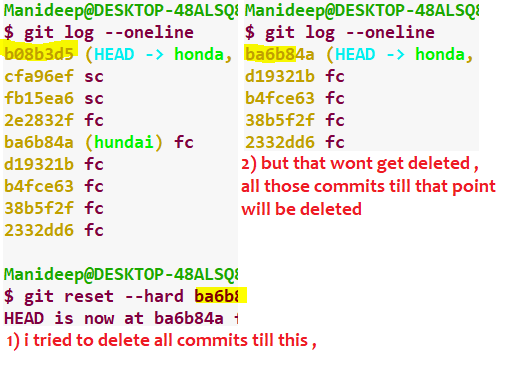
Git soft reset- here commits will be cleaned but changes will still be there





***Delete all commits and delete changes from working directory***





#### Add single file to staging area

//when u are adding means u are adding that file to staging area.

git add < file name abcd.txt >



if u want to pull back the file from staging area and keep in working area then

git restore –-staged <staged-file-name.txt>

#### To check whether file added to stage area or not

It will simply give the status of the branch

If u want to check what file are added to stage area, and which files are not in stage area.

git status

#### To check for modified files

git status

#### Add all files to staging area

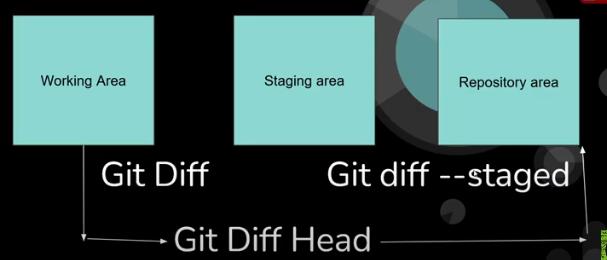
git add .

here “.” Means current directory

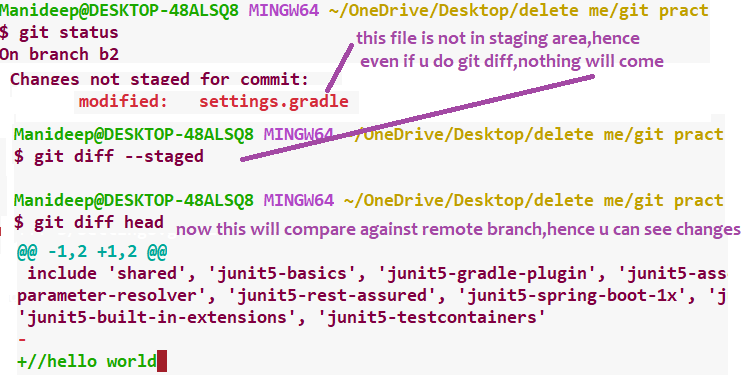
To remove a file from staging area

git restore --staged "1.unstaged file.txt"

#### **Diff**



If that file is not present in staging area, then it will simply it will not show anything



#### **Commit**

|  |  |
| --- | --- |
| To add files into stage area and commit at once | git commit -a -m "commit message"  //here a stands for all , -m stands for commit message  git commit -a -m "sc" |
|  |  |

git commit -m "first commit" or **git commit -m m1**

// here double quotes are not mandatory, “m” stands for commit message

git commit -m “Initital commit message”

it will commit the files only present in staging area.

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

#### To check whether committed or not

git log

// this will show u the commit history

To revert all modified files in staging area

git reset –hard

it will revert all the files in staging area only and

It will not impact to any un staged file ,

Push to upstream

Remember when u push, always u should mention the url + branch name

Here origin was an alias name given to the url(while checking out the code, this would have automatically happenned)

Ex:- git remote add origin <git url>

//here with this origin was set as an alias name our git remote url

git push <git remote url> <branch name>

git push <git remote url alias name(usually origin )> <branch name>

git push origin <branch name>

git push origin master

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

git reset/restart from that

git reset <commit-hash-reset all commits above this commit id>// means commits will be resetted to that point

all the above commits will be deleted (files will not be there in case of hard reset, in case of soft reset files will exists)

reset means resetting to that point, EX:- reset to childhood means are young age everything will be gone, u will restart from that point

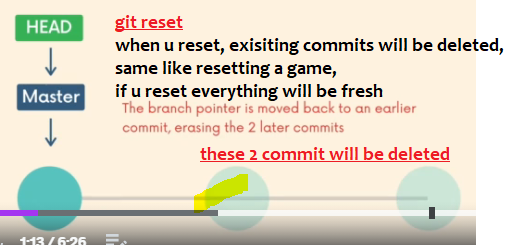
git reset <commit hash> //means u will be resetted to this point, all commits above that are gone and u will be at there

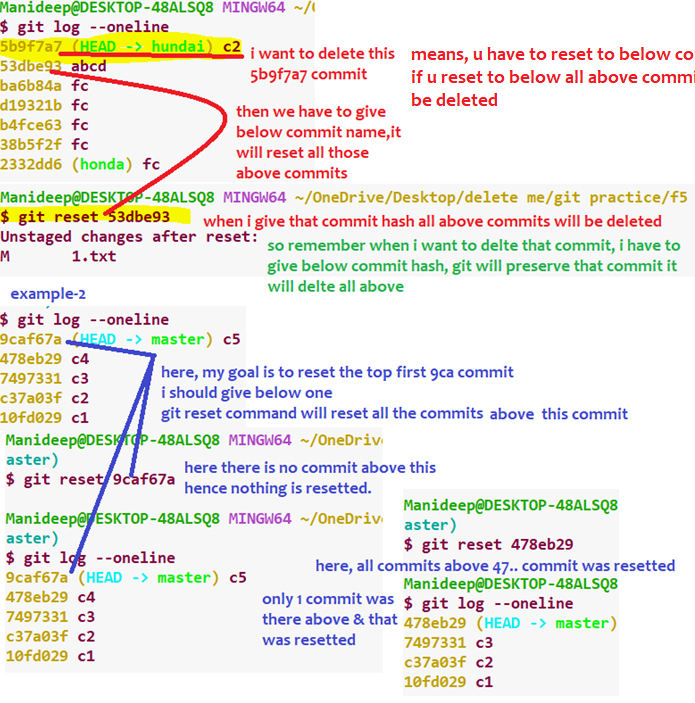
if u reset, everything will be fresh, we will loose those commits, those commits are entirely gone

when only u are working in that branch, when u did a mistake u can use reset, & it will delete the existing commit

whereas, when many people are working on same branch, when u reset then as the commit will be deleted

there might be occurring some problems when other pull bec that previous commit will be deleted.





Git Revert

When many people are working on same branch, then prefer git revert bec it will create a new commit with all reverted changes

So when ur friends pull they will get a new commit, if u use reset then existing commit will be lost, they would have made additional commit

On top of urs and when u merge they might loose existing commit, so it will creates mess better, when many people are working on same branch better use GIT REVERT over git reset



When u revert only that particular commit will be reverted and a new commit will be created

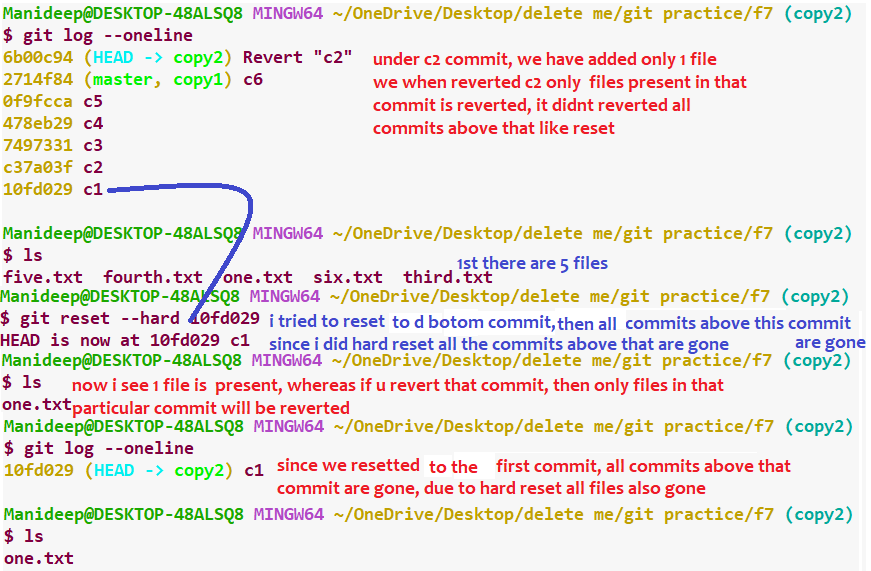
#### Reset vs revert

Git reset <commit-hash>

Git revert <commit-hash>

Reset means, when u reset to a particular commit, it will delete all the commits mentioned above that commit

EX:- when u reset to the bottom most commit, all the commits till top will be deleted (files will be there in case of soft commit)



Git fetch

Git fetch commands fetches branches and history from remote repository to local repository, it only updates the remote tracking branches

It won’t update the workspace

How to create a git text file using cmd

touch .gitignore.txt

the above will create a text file

to exit from command prompt

:q

To Link local repo to remote repo

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

git remote add origin git@github.com:StephenGrider/docker-react.git git push-u origin master

### Pull fetch

* 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**.

*Merge*

* git merge <branch-name-nokia> command merges the nokia remote branch code to the local working tree.

This merges the current branch with nokia branch code

Merges the specified branch code into current branch

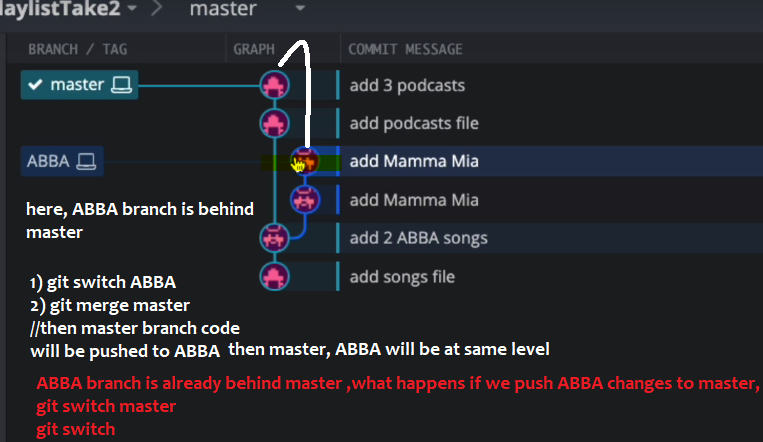
* + git merge <branch\_name> merges the current branch with specified branch.

Ur friends would have committed to the remote branch, if u want to have all those changes locally then use git merge

Ex:- git switch chennai // switches to branch name chennai

git merge <branch name>// means new branch code will be pushed to current branch

git merge hyd // hyd branch code will be pushed to current branch/Chennai



Ex:-

git switch master

git merge bugfix // here bugfix code will be pushed to master branch

Rebase

* git rebase --abort: Cancels rebasing and goes back to the previous commit
* git rebase --continue: Continues rebasing after resolving the rebase conflicts
* git rebase <base>: Rebases the current branch onto the base (branch name, commit ID, tag)
* git rebase -i <base>: Performs interactive rebase. Launches editor where you can specify command on each commit to transform it.

Stash

Stash means save something in some secret area.

|  |  |
| --- | --- |
| Create a stash(it will take both staged and un-staged changes & staged changes from working copy and creates a stash) | git stash save |
| Apply the stashed changes to current branch(it will remove the stashed changes and apply to working copy) | git stash pop |
| Apply the stashed changes without removing from stash memory | git stash apply  //when u use git stash pop, the content will be removed from the stashed are and it will be applied to current branch  Whereas when u used git stash apply the changes will not be removed from stash area and will be applied to current working directory |
| To see the list of all stashed list | git stash list |
| Applying a particular stash | git stash apply <stash name>  ex:-git stash apply stash@{0} |
| To drop / delete the stashed version | git stash drop <stashed-name>  git stash drop stash@{3}  // this will delete stashed version present at index 3  // if u are using pop, then stash will be automatically removed from the list, when u used git stash apply then stash won’t be removed |
| To delete all stashed versions | git stash clear |

## 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 patter

# Some questions

1. staging area means

only the files moved to staging area will be committed to local git repo

1. see which email ocnfigured

git config user.email