what is git

-----------

Version control system for code

Master/Main

-----------

both are same branch only

git --version

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

gives the git version

git help

---------

Syntax:  
git <verb> --help

git help <verb>

gives a documentation of how to use the command  
  
example: git help add

git status

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

gives the status of the changed files, and staged/unstaged files

git init

--------

creates a git repo

creates a .git file which is used to track the project changes

git config --global user.name "shakthi saravanaa"

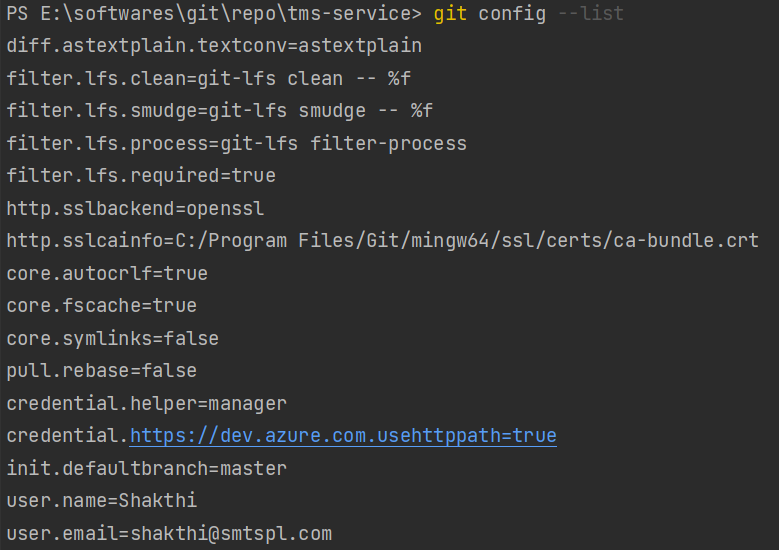
git config --global user.email [shakthi@smtspl.com](mailto:shakthi@smtspl.com)

Omit --global to set the identity only in this repo

git config –list

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

Lists all configurations



git remote add origin git@gitlab.com:vdespa/my-cool-website.git

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

git remote -v

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

gives you the origin address :

origin git@gitlab.com:vdespa/my-cool-website.git(fetch)

origin git@gitlab.com:vdespa/my-cool-website.git(push)

git add

---------

adds files to staging area to commit

git add <filename> {adds a specific file to staging area}

git add --all {adds all files to staging area}

git add . {adds all files to staging area}

git commit

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

commits the staged changes

only files in staging area will be committed

git commit {commits all changes without message}

git commit -m "first change" {commits all changes with message}  
  
git commit --amend {opens an interactive terminal where you can edit the commit message} {edits the recent commit by adding more changes to the recent commit}

how to commit a folder

````````````````````````````````

git does not commit empty folders

so, for git to commit an empty folder keep a file called .gitkeep inside the empty folder

how to delete a folder from git repo

``````````````````````````````````````````````````

delete the folder locally and then commit the change

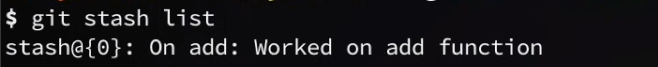
git stash

-----------

git stash {stashes without message}

git stash save “Worked on add function” {stashes with message}

git stash list {lists all stashes}



Stashes all the changes

git stash pop {pops out the recent stash and deletes It off the stash stack}

git stash apply <stash-id> {applies the changes in that particular stash}

ex: git stash apply stash@{0}

git stash show {shows the difference in the latest stash}

git stash show “stash@{2}” {shows the difference in the stash with id “stash@{2}”}

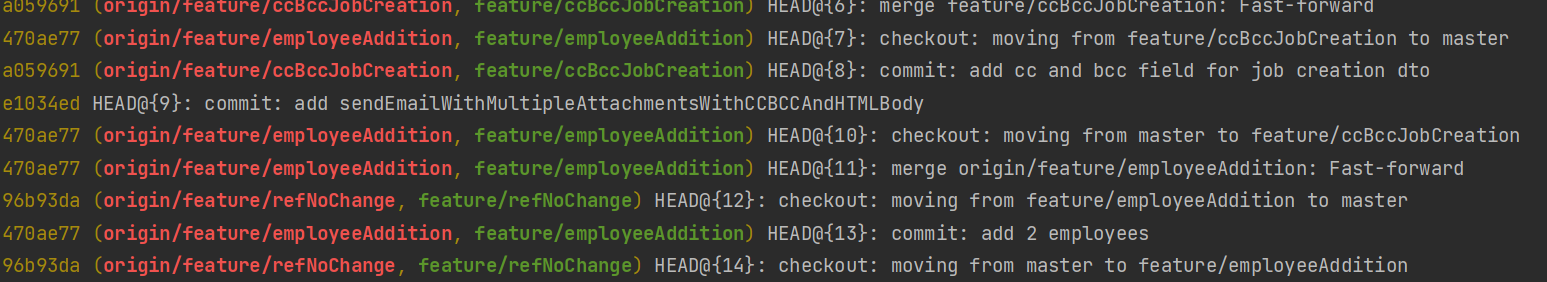
git stash show “stash@{2}” -p {shows detailed difference in the stash with id “stash@{2}”}

git stash drop <stash-id> {drops the stash without bringing the changes}

git stash clear {drops all stashes}

git reflog

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



Gives history of all commits including the operation done

git reset (unstage)

----------

unstages the specified file from staging area

git reset HEAD <filename>

git reset <filename>

unstage a file while retaining the changes in working directory

reset the commits from branch

```````````````````````````````````````````

git reset --soft <commit-hash> {reset the branch upto the <commit\_hash> and puts all other commit changes in the staging area}

git reset --hard <commit-hash> {reset the branch upto the <commit\_hash> and discards all other commit changes }

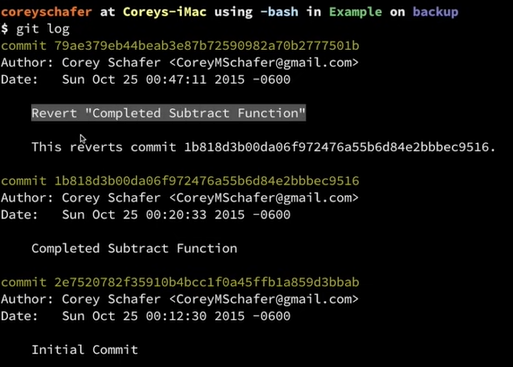
git revert

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

git revert <commit-hash>

Resets the given <commit-hash> alone

Adds a commit on top of the reverted commit instead of resetting up to the given <commit-hash>. Rest of the commits made after the <commit-hash> remains the same



git cherry-pick

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

git cherry-pick <commit-hash>  
brings the changes on <commit-hash> onto the current branch

creates a new commit

git log

--------

gives history of the changes/commits made

git log --patch {gives more information}

git log --stat {gives more info on which files the changes have occurred}

git log <branchB>..<branchA>

show the commits on branchA that are not on branchB

git log --follow <filename>

show the commits that changed file, even across renames

git diff

--------

git diff

shows the changes made on that branch

git diff <commit-hash1> <commit-hash2>

shows the differences between the two commits

git diff --staged

diff of what is staged but not yet commited

git diff <branchB>...<branchA>

show the diff of what is in branchA that is not in branchB

.gitignore

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

file that is used to ignore files that do not want to be committed

how to use : create a .gitignore file inside the project, add the names of folders and files that need not be commited, commit the .gitignore file

git branch

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

git branch - Lists all local branches

git branch -a – Lists all remote and local branches

git branch --merged – Lists all merged branches

git branch --no-merged – Lists all unmerged branches

git checkout

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

create a new branch with changes from the existing table

````````````````````````````````````````````````````````

git checkout -b feature/new-api

git checkout -b bugfix/new-fix

go to an existing branch

````````````````````````

git checkout master

git checkout feature/new-api

restore a file

`````````````````

git checkout <file-name>  
discards changes and revert the file back to the state it had in the last commit or the HEAD revision

also used to bring back a deleted file

create a new detached branch with a commit-hash

```````````````````````````````````````````````````````````````````````

git checkout <commit-hash> {creates a new branch called <commit-hash> which is detached from HEAD. It has all changes upto <commit-hash>}

{it will be garbage collected after 30 days or so..}  
{so, it is better to transfer the changes to a branch for backup}

We can do this by

git branch backup {it will create a new branch called backup, which is a copy of current <commit-hash> branch}

delete a branch

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

git branch -d feature/new-api – deletes locally only (will still be in remote)  
git push origin --delete feature/new-api – deletes in remote

git branch -D feature/new-api (force delete)

git merge

---------

merge the commits of branch A onto branch B

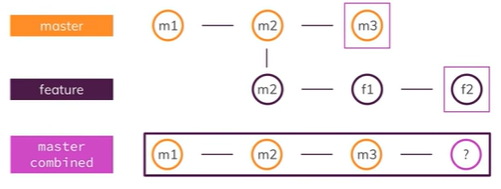
how to merge feature/new-api onto master (if no other commits were made on master after feature/new-api was created)

````````````````````````````````````````

git checkout master (go to the branch on top of which u want to merge the changes)

git merge feature/new-api

this does not maintain commit history of feature/new-api in master, instead it combines all commits into a single commit called merge commit



? – new merge commit on master after merge

git rebase

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

how to merge feature/new-api onto master (if commits were made on master after feature/new-api was created)

````````````````````````````````````````

1)rebase feature/new-api

on branch feature/new-api run, [git rebase master] (this would add all commits made till date from master to feature/new-api)

the commits would be in sequential order, as if the changes made in feature/new-api were made after the commits in master

2)Now merge

on branch master run, [git merge feature/new-api]

this creates a new commit called “merge commit” on master

how to handle merge conflicts

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

1)accept which version of changes u want in an ide

2)commit the changes

Or

git merge --abort { aborts merge }

git rebase --abort { aborts rebase }

git push

----------

pushing all the changes that were committed in local to remote repository

remote repository -> (origin),(git@gitlab.com:vdespa/my-cool-website.git)

git push origin master (pushing the master branch to remote)  
  
git push origin feature/new-api (pushing feature/new-api branch to remote)

No need to specify branch name always, if the branch is pushed like this at starting  
git push -u origin feature/new-api  
--------------  
  
Next time we can do just  
git push , git pull  
Git remembers the local feature/new-api branch is associated with remote feature/new-api

git pull

--------

pulling all the changes from remote to local

git pull origin master (pulling changes made on master branch in remote to local)

Suppose there is a file called readme.md in master branch

Two lines have been added to readme.md in remote

1 line has been modified in local

After you pull remote master branch to local, merge commit will take place

merge commit - annoying commit like message in list of all commits which can be seen when u enter git log

This can be avoided by keeping all changes in remote and discarding all changes in local

(not a good way)

git reset --hard origin/master

(best way)

git pull origin master --rebase

git clone

---------