**Version Control System:**

Management of changes to anything

Way to store files in central location accessible to all, enabling to keep track of changes done in source code by whom, when, what and why

Recover from inevitable circumstances

**Before VCS:**

-File Renaming

-Directories

-Zip Files

-Nothing

**VCS is also helpful in**

-Backups & Restoration

-Synchronization

-Reverts

-Track changes

-Parallel Development (working on different releases or code at same time)

**Types of VCS**

-Centralized VCS

-Distributed VCS

**Centralized VCS:**

Manages all changes or history

Every transaction need active connection

Backups are important

EX: -CVS

-SVN

-TFS

-VSS

**Distributed VCS:**

Authoritative server used to sync and work together

Every working checkout is a repository

Get version control even when detached (Different network)

**EX:**

-GIT (supports 60 to 70% open source are being managed)

-Mercurial

-Bit Keeper

-Darcs

-Bazaar

**Note:** GITHub and BitBucket –every active user is seen as a connection and if it is out of capacity it will throw errors.

Empty folders are not tracked in GIT

Four folders are there in GIT LocalWorkingDirectory, cache, LocalRepo, RemoteRepo

Best practice is to include .git file into the repo so that it will keep the garbage away.

Projects, Issues, Documentation, permissions can be managed in git (based on the requirement we can enable or disable

Web hooks are to deal with the restapi’s

Review code before merge, GUI (benefits of GIT)

GIT can also be configured as a remote repo. GITHub is built on top of GIT

**Distributed V/s Centralized:**

Every Checkout/Clone is Entire Repository

Better Access Management along with Self-service

Inbuilt Hooks Management (By repo admin and does not involve any third parties)

-Pre-commit

-Post-Commit

Better Branch Management

-local

-Remote

(Create branch in local, match it and then push it to remote)

Access Management on Branches in much easier as compare to Centralized VCS

In-built review system (Pull requests, making everyone accountable)

**GIT History:**

Linus uses BitKeeper to manage Linux code

**GIT is not a SCM**

Linus Torvalds, 07 Apr 2005

**Why GIT?**

-Branching: More Flexibility to work on Master Branch

-Distributed Architecture: no requirement of network connection to central repository

-Open Source: Basic Products are free (restricted access is charged or any advanced features let’s say Enterprise GIT)

-Integration with CI: Jenkins, IDE, TFS etc.

**GIT Installation:**

GIT comes default for all major Linux Flavors

Can be downloaded from Below link

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

GIT bash, GIT cmd, GIT GUI will be installed by default.

After installation open the GIT bash and enter the following commands

* git –version (gets the version of GIT installed in the system)

git version 2.21.0.windows.1

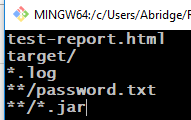
* git config –global user.email “[rajinirede@gmail.com](mailto:rajinirede@gmail.com)” (For configuring the users globally)
* git config –global user.name “Rajini Reddy”
* cat /c/Users/Abridge/.gitconfig ( To get the information of the users)
* Create a folder Rajini/ repo
* ls –la (To get the folder details along with the permissions
* git init (For creating an empty repo will create a .git file and it is by default a master branch, LR )

Author and committer we have both here

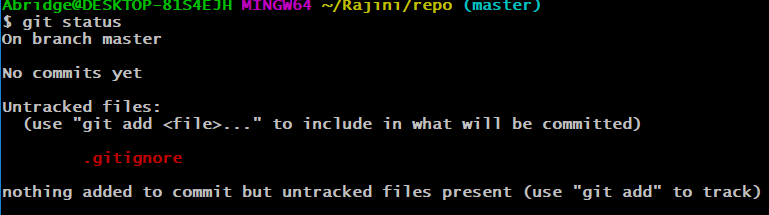
Author is the person who makes changes, committer is the sender

.git file will be updated for all the changes

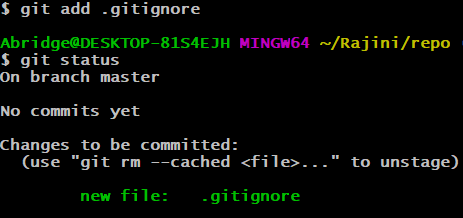
* git –help (show all the commands that we can execute)
* git status (whatever we have in the local will be compared with the commits or files in the folder, LWD)
* vi .gitignore ( To ignore the files named or mentioned in that file, will ignore the files with following extensions or type of files)



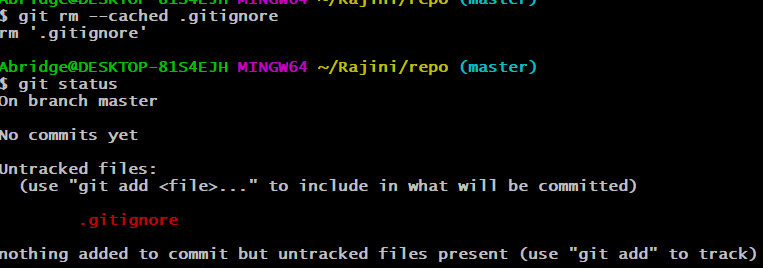
* git status ( will give us the following output showing the untacked files mentioned in the .gitignorefile)



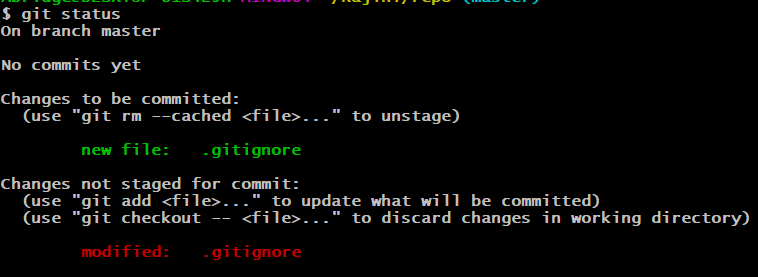
* git add ., git add \*, git add --all, git add file1 file2 file3..
* git add .gitignore(When executing this command it will move the file from LWD to Cache)



* git rm –cached .gitignore( we are not discarding but removing from cache which have been placed mistakenly cache to LWD)



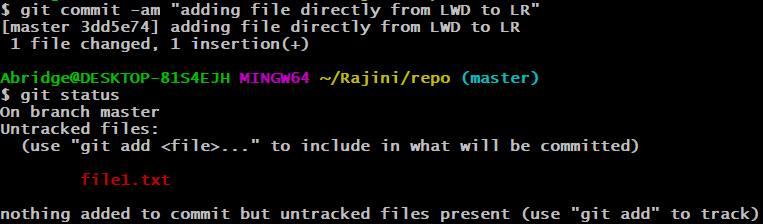
* if we have made any changes to the file (.gitignore) will get the following



* git commit –m “adding .gitignore” (-m is message is mandatory) We have tracked the change cache to LocalRepo

Note: The newly made changes will not be moved to LocalRepo as it is not yet been added to cache yet

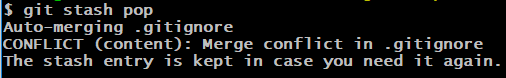
* git commit –am “adding .gitignore”( -a for adding the file first to cache and then committing it to LR, Do this when u r confident of the commit, otherwise first execute git add and then git commit) the changes have not been moved to LR yet



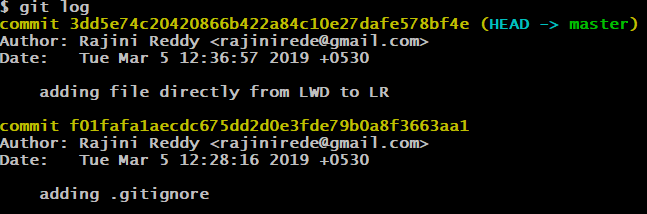
* git checkout .gitignore ( this command will discard the changes in the LWD compare LWD and LR ) then edit
* git stash (will keep the local changes without sending to the Remote repo, recalling them whenever required) To check the stashed files need to checkout with id 3dd5e74. It is not merging, but just bringing it back from stash index or storing it there



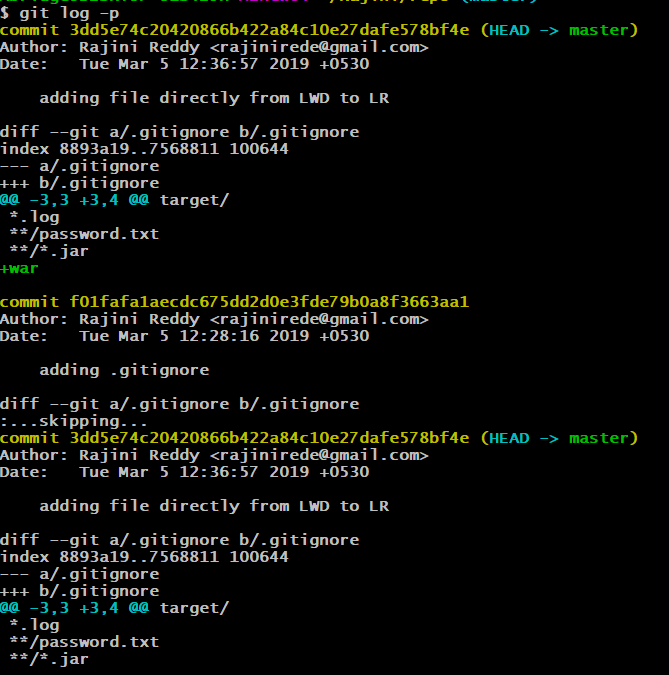
* git stash pop



* git log (will get the details of the commits whom, when, why)



* git log –p ( will give you all the changes along with files(what))



* git branch branch1( Create a new branch)
* git branch( Lists the branches)
* git checkout branch1(switch to new branch mentioned from master)
* git checkout –b new-branch(-b will create an switch to new branch)

**Merging**

There are 3 types of merges

* Auto Merge
* Recursive
* Conflicts

git checkout master

git branch branch-1

git checkout branch-1

nano .gitignore

git commit -am "merge1"

git checkout master

git checkout master

git branch branch-2

git checkout branch-2

touch new-file

git add new-file

git commit -m "merge 2"

git checkout master

nano .gitignore

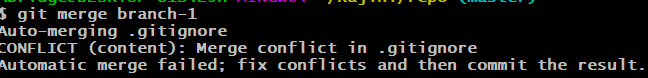
git commit -am "merge 3"

git merge branch-2 (merges with the master branch)

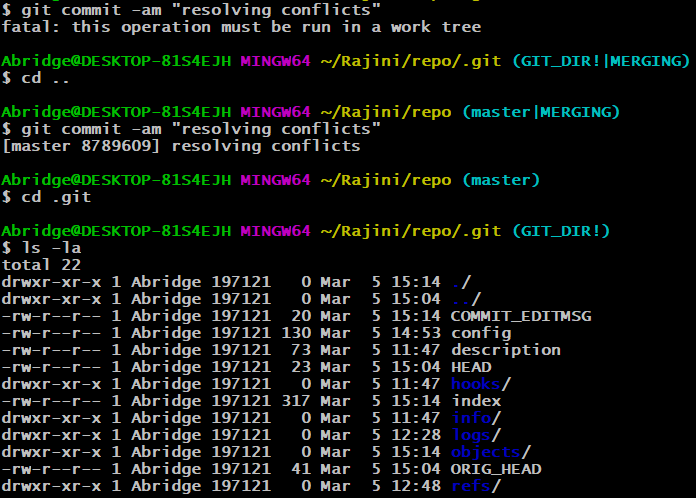
git log

git checkout master

git merge branch-1(Conflict in merge)



To resolve the merge conflict have to correct the conflict file manually for changes and commit again



**Tagging:**

Once the branch is tagged it will become read-only

git tag

git tag ver-1.0

git tag

ver-1.0

git checkout ver-1.0

**Deleting branches:**

git branch –d branch-name

If you are unable to delete with some error use the below

git branch –D branch-name

or

git branch –delete –force branch name

git mv filename with path <new file path>

git rm file-name

**Create a remote Repository**

Login to <https://github.com>

Click on create a new repository, enter the repo name and click on create repository

git remote add origin https://github.com/RajiniRede/Repo1.git

origin here is the name of the URL by default, we can change it to anything, pushing or publishing the data to remote

git push origin master(LR to RR)

**Note:** It will be prompted for username and password, after authentication it will pushed.

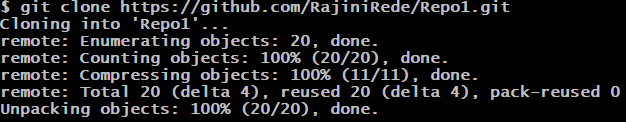
Local Master Branch is pushed to the remote repo Repo1.

**Cloning the existing Remote Repo:**

Click on clone or download and copy the URL

And move out of the master branch in git bash and give the following

git clone <https://github.com/RajiniRede/Repo1.git>



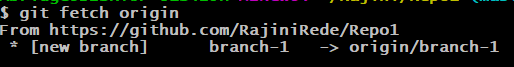
git remote –v



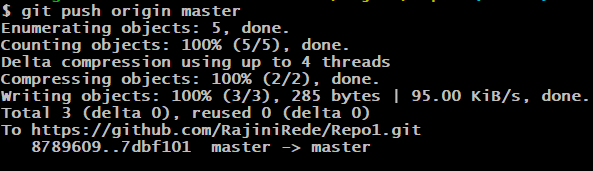
**Fetch:** Fetch the entire data (files, directories) and bring it from RR to LR and then merge based on the differences

**Pull:** Pull the data and put it in the LWD, in pull the merge happens automatically.

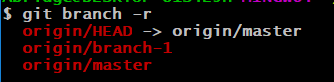
git fetch origin



git push origin master ( even the versions can be pushed)



git branch -r ( remote branch, here origin is the URL of repo)



**Types of Git hub remote merge**

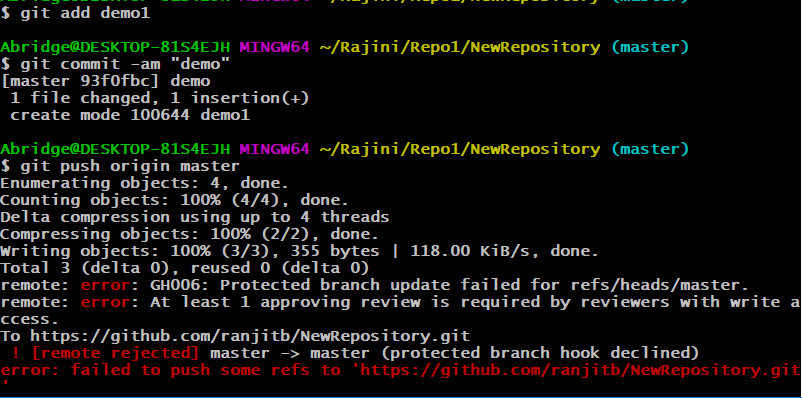
Auto, squash, rebase

**Note:** for viewing or giving access to others or getting access to the repositories, have to go to the collaborators and give the username of that individual and send invite.

Once they accept the invite we will be able to see that in the repositories

If the access is restricted we will not be able to push files to the remote repo shared

Will get the following error:



The reviews can be approved, changed or commented by sending a review request.

**Forking in git:**

Forking is always done on remote. A **fork** is a copy of a repository. **Forking** a repository allows you to freely experiment with changes without affecting the original project. Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea

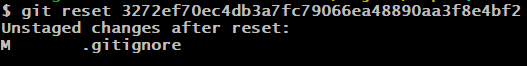
When we enter into other’s repo there we will see a fork option click on that and it will be created.

**Note:** can I fork a repository using only the command line: No

**Advanced GIT commands:**

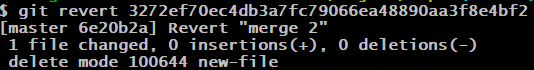
Have to be careful while executing

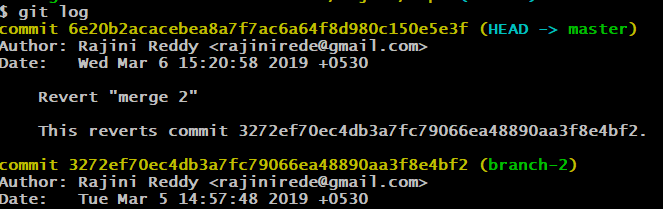
* git reset commit id(once done cannot be brought back , changes are there but history is gone it is for LocalRepo)



* git revert commit id (commonly used, keeping the history but discarding(rollback the changes))

git revert –continue





* git cherry-pick ( Not used commonly, add and commit)
* git rebase and git merge (the purpose is the same, but the way the history is maintained is different, rewriting the history)
* git rebase –i HEAD ~2
* git commit –amend (change the comments )
* git rebase –continue
* ssh-keygen.exe(is the default feature)

Configure the generated key in the git-hub.

* git diff branch-1 branch-2 ( compare two files or revisions)

**BEST PRACTICES with GIT:**

Keep Repositories as small as possible.

Have one to one mappings with build or CI.

For better merge management use development or feature branches.

Always merge to main branches using pull requests.

BitBucket/ GITHub/ GitLab: third party web based Git UI repository manager, issue tracking and ci/cd pipelines

Cloud instances are

* <https://gitlab.com>
* <https://bitbucket.org>
* <https://github.com>

We can host these on our servers and use it for trial version