GIT

Source code management:

- Source code management (SCM) is used to track modifications to a source code repository.
- Source control is also called as Revision Control or Version Control
- Managing the process of Source code is called as Source Code Management

Version control system:

- VCS is used in software industry, where developers/teams keep Source code changes at one place
- VCS helps to keep the track of collection of files (Track Changes)
- Supports creation of different version of collection of files
- Allows to switch between version of files
- These are stored at a place called “REPOSITORY”, in short as REPO.
- Experimentation of Code Changes (Feature/Development)

SCM tools:

- Team Foundation Server (TFS)
- Subversion
- Mercurial
- GIT

Repository/depot:

- Repository is a collection of track Changes at a centralised Server

Work dir/work tree:

Work dir:

- The working directory is a single checkout of one version of the project.

- This essentially means if you checkout a branch (e.g. master) and are sat on a particular commit (e.g. HEAD), your working directory is the "umbrella" term for all your files and folders.

Work tree:
- Git worktree allows you to have multiple working directories associated with one git repo.
- It has been useful for me to look at, copy code between, and run two different branches of my code.
- A working tree is a set of files for a branch that is linked to the repository.

Branch/trunk/code line:

Branch:
- Branch is created from master repo from Server
- Branch is created to do code experimentation
- Or add any functionalities to present code
- Or to modify existing code
- Or to bug fix the issues

Trunk:
- trunk is a place where main development happens, and branches are places where different developers work on different functionalities.
- A trunk in SVN is main development area, where major development happens

Code line:
- A codeline is a set of versions of a software component and other configuration items on which that component depends.

commit/check-in:

commit:
- The "commit" command is used to save your changes to the local repository.
- Using the "git commit" command only saves a new commit object in the local Git repository.

<u>Check in:</u>

- Checking-in code means to upload code to mentioned/required branch repository so that its administrator/reviewer can review the code and finally update/merge the project version.
- Check in" in GIT is something like pushing your local changes to remote branch/repo.
- "Check out" in GIT is something like pulling your remote changes to local branch/repo.
- In Git terms, a "check in" is the act of adding something to versions of a target entity. There is no such git checkin command in git.

version/version-ID/commit-id,tag:

<u>Version:</u>

GitVersion is a tool to help you achieve Semantic Versioning on your project.

Version sources:

There are a number of sources GitVersion can get its versions from, they include:

- Tags
- Version numbers in branches (e.g. release/2.0.0)
- Merge messages (for branches with versions in them, e.g. Merged branch release/2.0.0' into master)
- Track version of another branch (e.g. develop tracks master, so when master increments so does develop)
- GitVersion.yml file (e.g next-version: 2.0.0)

<u>Commit-id:</u>

A Git commit ID is a SHA-1 hash of every important thing about the commit. I'm not going to list them all, but here's the important ones...

- The content, all of it, not just the diff.
- Commit date.
- Committer's name and email address.
- Log message.
- The ID of the previous commit(s).

This serves three purposes.

- First, it means the system can tell if a commit has been tampered with. It's baked right into the architecture.
- Second, one can rapidly compare commits just by looking at their IDs. This makes Git's network protocols very efficient.
- Third, and this is the genius, two commits with the same IDs have the same history. That's why the ID of the previous commits are part of the hash. If the content of a commit is the same but the parents are different, the commit ID must be different

Tag:

- It provides a tag name with -a option and provides a tag message with –m option.

Advantages of git and git snapshots:

Advantages of git:

- One of the biggest advantages of Git is its branching capabilities.
- Unlike centralized version control systems, Git branches are cheap and easy to merge
- Data redundancy and replication
- Superior disk utilization and network performance
- Feature branches provide an isolated environment for every change to your codebase.

Advantages of git snapshots:

- One of the main advantages remains in term of merging, a domain where SVN struggles at time.
- Git does a simple three-way merge, using the two snapshots pointed to by the branch tips and the common ancestor of the two.
- This makes Git more like a mini filesystem with some incredibly powerful tools built on top of it, rather than simply a VCS.

Work space,staging area,buffer area:

Workspace:

- A workspace is simply a set of git repositories.
- You can have several different sets of repositories, each one stored in a different workspace file.

- They are listed in the "Repos" tab in the main window's left pane.

Staging area:

- Staging is a step before the commit process in git.
- To "stage" is to do git add file.ext for a specific file, or git add . to affect all modified and untracked files.
- Files that have been added in this way are said to be "staged" and they will be included in the next "commit".

a commit in git is performed in two steps:

- staging
- actual commit

Repository (local/non-bare):

- Git local repository is the one on which we will make local changes, typically this local repository is on our computer.
- The local repo is on your computer and has all the files and their commit history, enabling full diffs, history review, and committing when offline.

Steps to create local repository:

- Create a new repository on GitHub. To avoid errors, do not initialize the new repository with README, license, or gitignore files. You can add these files after your project has been pushed to GitHub.

Open Terminal.

- Change the current working directory to your local project.
- Initialize the local directory as a Git repository.
- Add the files in your new local repository. This stages them for the first commit.
- Commit the files that you've staged in your local repository.
- At the top of your GitHub repository's Quick Setup page, click to copy the remote repository URL.
- In Terminal, add the URL for the remote repository where your local repository will be pushed.
- Push the changes in your local repository to GitHub.

Repository (central,bare):

- Repositories created with git init --bare are called bare repos.
- They are structured a bit differently from working directories.
- First off, they contain no working or checked out copy of your source files.
- second, bare repos store git revision history of your repo in the root folder of your repository instead of in a .git subfolder.
- bare repositories are customarily given a .git extension.

Installation and configuration:

Git for windows:

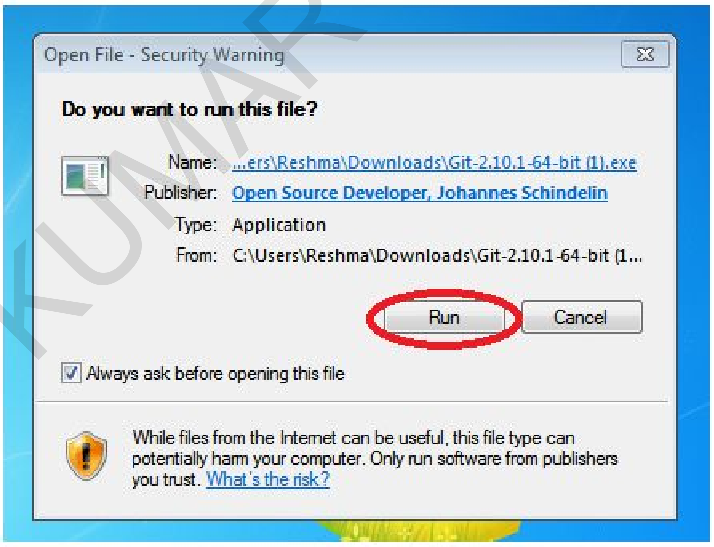
Step 1:

To download the latest version of Git, click on the link below:

Download Git for Windows

Step 2:

After your download is complete, Run the .exe file in your system.



Open File - Security Warning

Do you want to run this file?

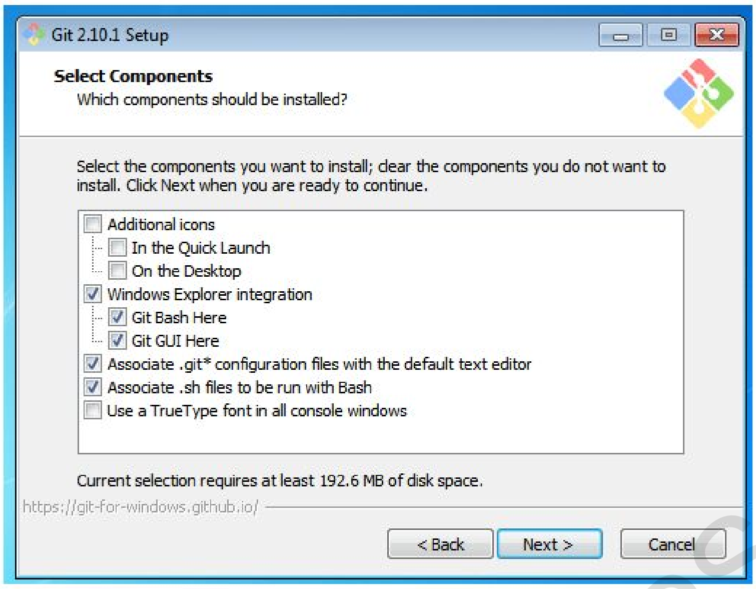
Name: ...ers\Reshma\Downloads\Git-2.10.1-64-bit (1).exe
Publisher: Open Source Developer, Johannes Schindelin
Type: Application
From: C:\Users\Reshma\Downloads\Git-2.10.1-64-bit (1...

☑ Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Step 3:

After you have pressed the Run button and agreed to the license, you will find a window prompt to select components to be installed.


Git 2.10.1 Setup

Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [ ] Additional icons
  - [ ] In the Quick Launch
  - [ ] On the Desktop
- [✓] Windows Explorer integration
  - [✓] Git Bash Here
  - [✓] Git GUI Here
- [✓] Associate .git* configuration files with the default text editor
- [✓] Associate .sh files to be run with Bash
- [ ] Use a TrueType font in all console windows

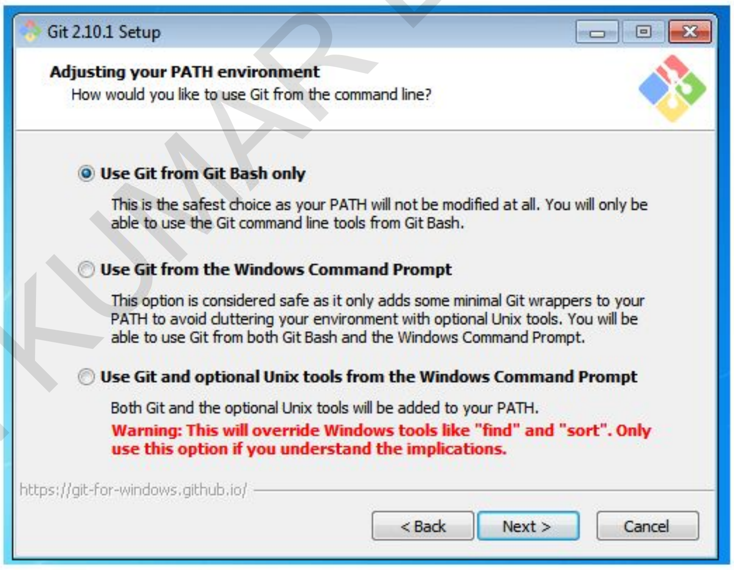
Current selection requires at least 192.6 MB of disk space.

https://git-for-windows.github.io/

After you have made a selection of your desired components, click on Next>.

Step 4:

The next prompt window will let you choose the adjustment of your path environment. This is where you decide how do you want to use Git.


Git 2.10.1 Setup

Adjusting your PATH environment

How would you like to use Git from the command line?

[✓ **Use Git from Git Bash only**]

This is the safest choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

Use Git from the Windows Command Prompt

This option is considered safe as it only adds some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.

Use Git and optional Unix tools from the Windows Command Prompt

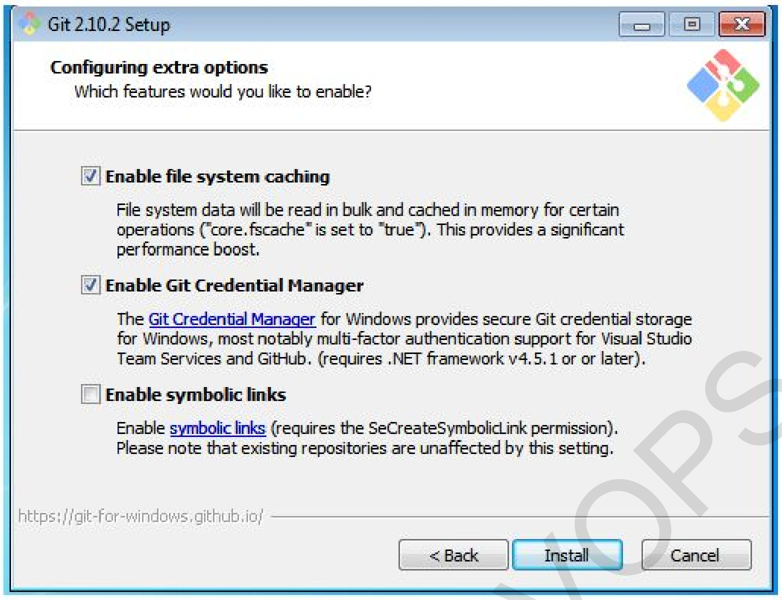
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

https://git-for-windows.github.io/

You can select any of the three options according to your needs. But for beginners, I recommend using Use Git From Git Bash Only

Step 5:

The next step is to choose features for your Git. You get three options and you can choose any of them, all of them or none of them as per your needs. Let me tell you what these features are:



Configuring extra options
Which features would you like to enable?

[✓] **Enable file system caching**
File system data will be read in bulk and cached in memory for certain operations ("core.fscache" is set to "true"). This provides a significant performance boost.

[✓] **Enable Git Credential Manager**
The Git Credential Manager for Windows provides secure Git credential storage for Windows, most notably multi-factor authentication support for Visual Studio Team Services and GitHub. (requires .NET framework v4.5.1 or or later).

[ ] **Enable symbolic links**
Enable symbolic links (requires the SeCreateSymbolicLink permission). Please note that existing repositories are unaffected by this setting.

https://git-for-windows.github.io/

The first is the option to Enable file system caching.

Caching is enabled through Cache manager, which operates continuously while Windows is running. File data in the system file cache is written to the disk at intervals determined by the operating system, and the memory previously used by that file data is freed.

The second option is to enable Git Credential Manager.

The Git Credential Manager for Windows (GCM) is a credential helper for Git. It securely stores your credentials in the Windows CM so that you only need to enter them once for each remote repository you access. All future Git commands will reuse the existing credentials.

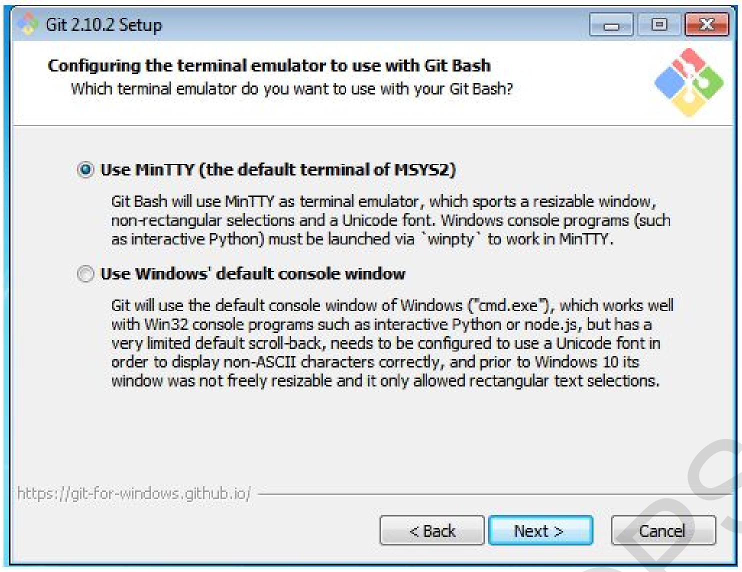
The third option is to Enable symbolic links.

Symbolic links or symlinks are nothing but advanced shortcuts. You can create symbolic links for each individual file or folder, and these will appear like they are stored in the folder with symbolic link.

I have selected the first two features only.

Step 6:

Choose your terminal.



Git 2.10.2 Setup

Configuring the terminal emulator to use with Git Bash

Which terminal emulator do you want to use with your Git Bash?

[✓ **Use MinTTY (the default terminal of MSYS2)**]

Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.

Use Windows' default console window

Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.

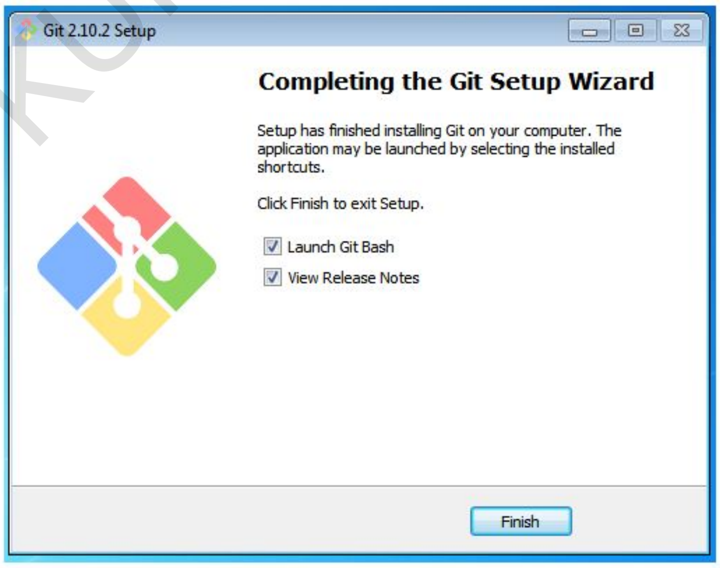
You can choose one from the options.

The default terminal of MYSYS2 which is a collection of GNU utilities like bash, make, gawk and grep to allow building of applications and programs which depend on traditionally UNIX tools to be present.

Or you can choose the window's default console window (cmd.exe).

Step 7:

Now you have got all you need. Select Launch Git Bash and click on Finish.



Git 2.10.2 Setup

Completing the Git Setup Wizard

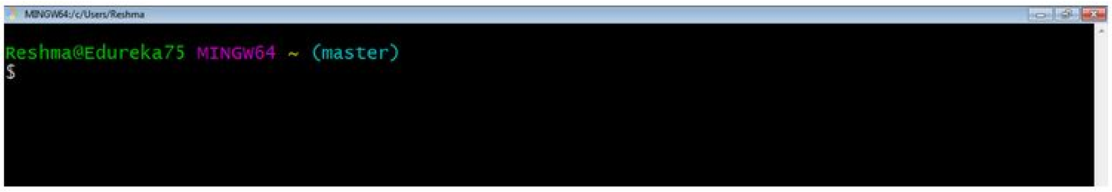
Setup has finished installing Git on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

☑ Launch Git Bash

☑ View Release Notes

This will launch Git Bash on your screen which looks like the snapshot below:



```
MINGW64:/c/Users/Reshma
Reshma@Edureka75 MINGW64 ~ (master)
$
```

SETUP GIT ENVIRONMENT VARIABLES

❖ We need to use the git config command along with flag --global to set the global environment variables.

❖ By default we need to set user.name and user.email environment variables after git is installed.

# git config --global user.name “test”

- This sets the user.name for all projects

# git config --global user.email “test@gmail.com”

- This sets user global email address for all projects

Q. How to check the git config settings ?

# git config --list

Git add,git commit,git status,git log,git push:

Git add:

- Usage - To update changes from working area to Staging area

Syntax :

# git add <file_name> → to add the single file to staging area

# git add <directory> → to add the directory to staging area

# git add . → adds all untracked files under current path or

directory to staging area

# git add -A → adds all untracked files

Git commit:

- Usage : To push the changes from staging area to Local Repository

SYNTAX :

→ to add single file

# git commit -m “commit message” <file_name>

→ to add all staged files to Local Repo

# git commit -a -m “commit message”

→ to commit all files of staging area

# git commit -m “ commit message” .

- Once if we execute this command, git responds by information

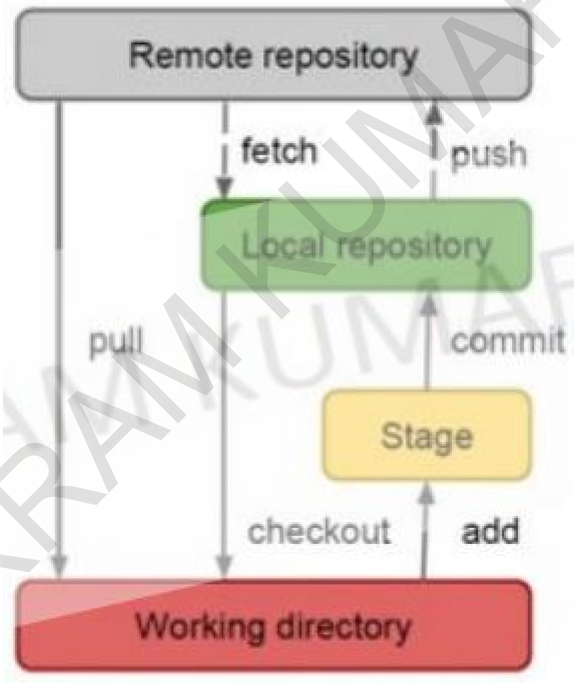
○ SHA1 creation on that particular commit snapshot

○ No. of files committed

○ No. of Insertions

○ No. of deletions

○ Creation mode



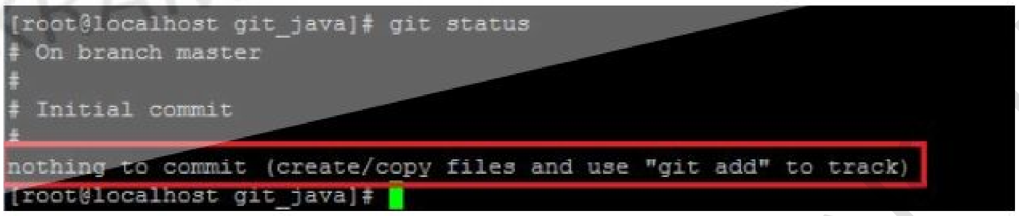
<u>GIT Status:</u>

- Usage - Tells about tracking status of files

SYNTAX :

git status

Example -



```
[root@localhost git_java]# git status
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
[root@localhost git_java]#
```

<u>GIT Log:</u>

- Usage : to check the all the log information of commit on a repository

SYNTAX :

# git log

- Shows the latest commit log at the top
- Log shows - commit id, author, date, commit message details

<u>Git push:</u>

- The git push command is used to upload local repository content to a remote repository.
- Pushing is how you transfer commits from your local repository to a remote repo.
- After a local repository has been modified a push is executed to share the modifications with remote team members.
- After Pull is completed Successfully, now push the changes to git Remote Repository

# git push origin master

Git ignore,git branch,git checkout,git merge:

<u>Git ignore:</u>

- Git ignore file helps to keep track of files that does not need to upload to git remote server
- In a repo, only one git ignore file exist

- For a single repo we will have one git ignore that is on the master branch

<u>GIT branches:</u>

# git branch → Shows the branch of Local Repo only, * symbol indicates the master branch
# git branch <new_branchname> → creates a new branch
# git checkout -b <new_branchname> → to create a new branch and switch to it.
# git checkout <branch_name> → helps to switch to branch_name

Note --** Before switching to new branch make sure that you have clean git status
# git branch -m <oldname> <new_name> → to rename the branches
# git branch -a → shows the branches of Local and Remote
# git merge <branch_name> → it is to merge branch
# git branch -d <branch_name> → Delete the branch

Git snapshots,git conflict,git stash:

<u>Git snapshots:</u>
- Snapshots, in git, are synonymous with “commits”.
- Every time you commit to a git repository, you are saving a snapshot of all the files in your repository.
- Git isn’t saving all the files tracked by your repository every time you commit

<u>Git conflict:</u>
- A merge conflict is an event that occurs when Git is unable to automatically resolve differences in code between two commits.
- When all the changes in the code occur on different lines or in different files, Git will successfully merge commits without your help.
- Merge conflicts can happen when merging a branch, rebasing a branch, or cherry picking a commit.
- If Git detects a conflict, it will highlight the conflicted area and ask which code you wish to keep.

<u>Git stash:</u>
- The git stash command takes your uncommitted changes (both staged and unstaged), saves them away for later use, and then reverts them from your working copy.

- At this point you're free to make changes, create new commits, switch branches, and perform any other Git operations; then come back and re-apply your stash when you're ready.
- The stash is local to your Git repository; stashes are not transferred to the server when you push.

Git reset,git revert:

Git reset:

- Git reset is a powerful command that is used to undo local changes to the state of a Git repo.
- Git reset operates on "The Three Trees of Git". These trees are the Commit History (HEAD), the Staging Index, and the Working Directory.

Git revert:

- Both the git revert and git reset commands undo previous commits.
- But if you've already pushed your commit to a remote repository, it is recommended that you do not use git reset since it rewrites the history of commits.
- Better to use git revert, which undoes the changes made by a previous commit by creating an entirely new commit, all without altering the history of commits.

Git remove,git clean:

Git remove:

- git rm is used to remove a file from a Git repository.
- first remove a target from the filesystem and then add that removal event to the staging index.
- The git rm command operates on the current branch only.
- The removal event is only applied to the working directory and staging index trees.

```
#git rm Documentation/\*.txt
```

Git clean:

- Git clean is to some extent an 'undo' command.
- Git clean can be considered complementary to other commands like git reset and git checkout .
- Whereas these other commands operate on files previously added to the Git tracking index, the git clean command operates on untracked files.

Git fetch,git diff:

Git fetch:

- The git fetch command downloads commits, files, and refs from a remote repository into your local repo.

Options
--all
Fetch all remotes.
-a
--append

Append ref names and object names of fetched refs to the existing contents of .git/FETCH_HEAD. Without this option old data in .git/FETCH_HEAD will be overwritten.

Git diff:
# git diff → it compares the files between working area with Local Repository

# git diff --staged → it compares the files between staged and Local Repository

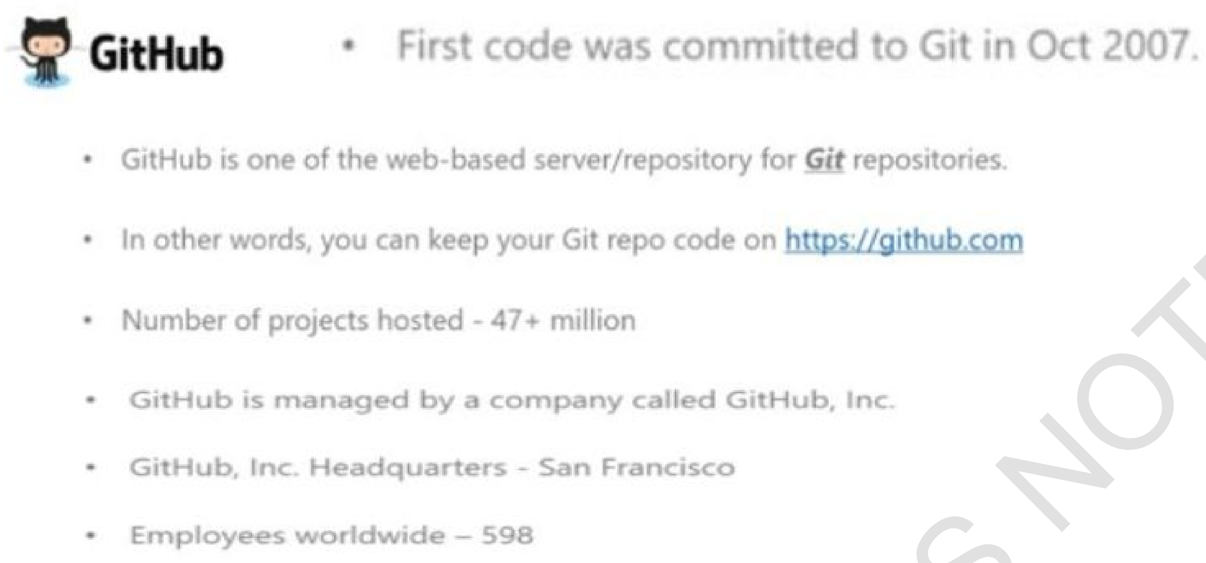
Git cherry-pick,github:

Git cherry-pick:

- git cherry-pick is a powerful command that enables arbitrary Git commits to be picked by reference and appended to the current working HEAD.
- Cherry picking is the act of picking a commit from a branch and applying it to another. git cherry-pick can be useful for undoing changes.
- If you want to cherry pick more than one commit in one go, you can add their commit IDs separated by a space:

```
#git cherry-pick d467740 de906d4
```

Github:

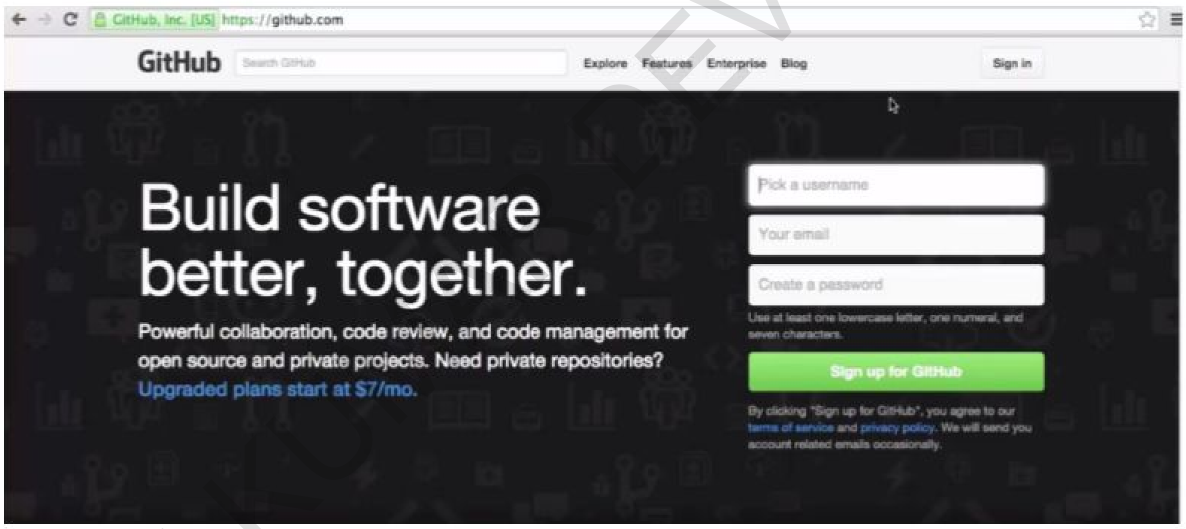


GitHub

- First code was committed to Git in Oct 2007.
- GitHub is one of the web-based server/repository for *Git* repositories.
- In other words, you can keep your Git repo code on https://github.com
- Number of projects hosted - 47+ million
- GitHub is managed by a company called GitHub, Inc.
- GitHub, Inc. Headquarters - San Francisco
- Employees worldwide – 598

Github -

- To open github account open the link https://github.com



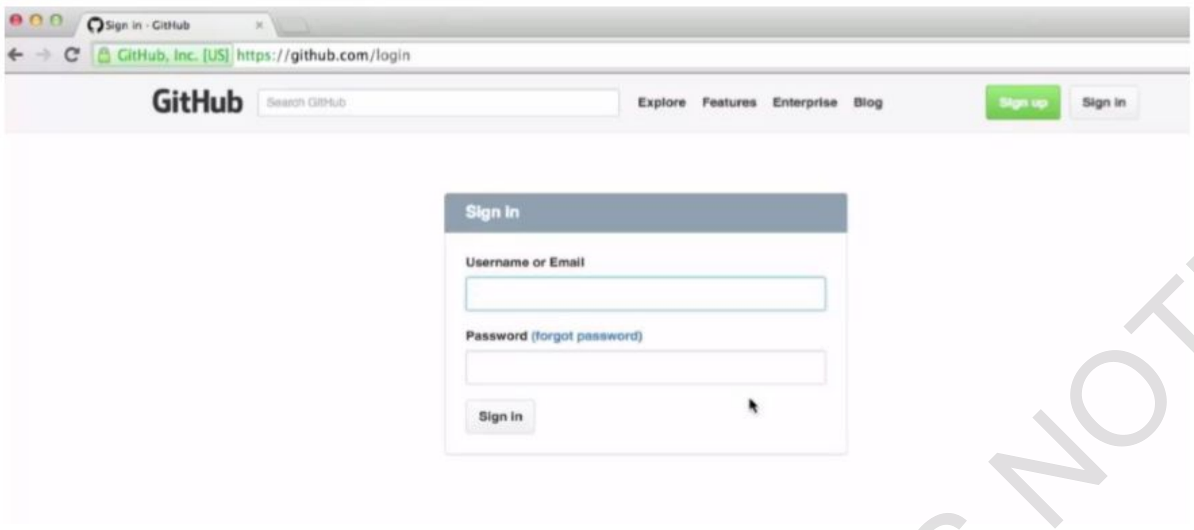
Build software better, together.

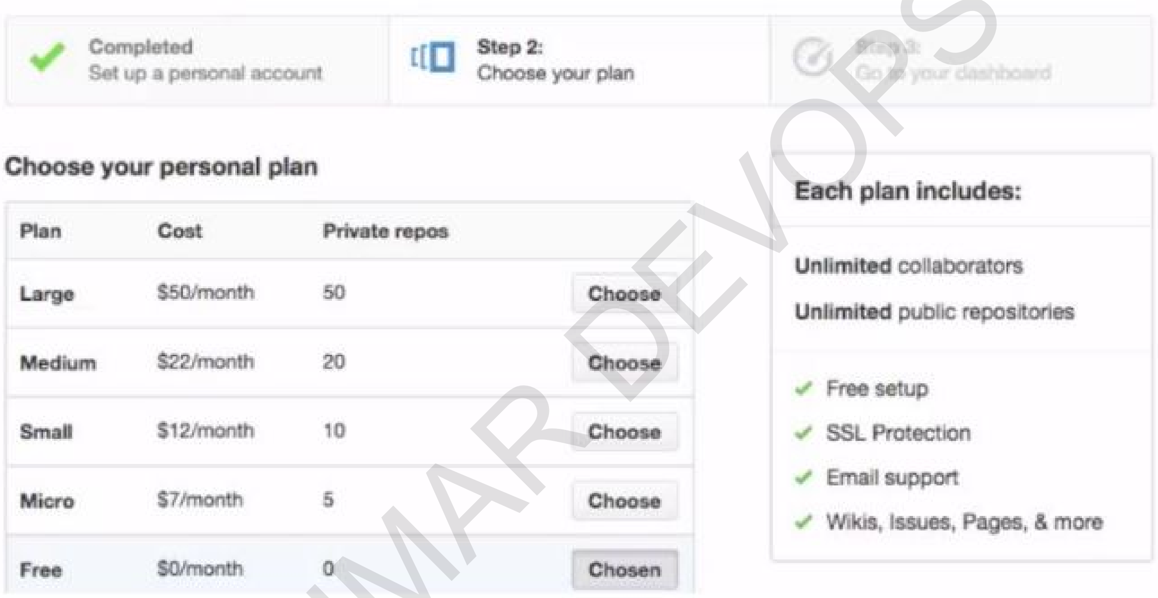
Powerful collaboration, code review, and code management for open source and private projects. Need private repositories? Upgraded plans start at $7/mo.

Use at least one lowercase letter, one numeral, and seven characters.

By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy. We will send you account related emails occasionally.

- If you have already account click on “sign in” or else to create account, click on “signup”





Choose your personal plan

Plan	Cost	Private repos	
Large	$50/month	50	Choose
Medium	$22/month	20	Choose
Small	$12/month	10	Choose
Micro	$7/month	5	Choose
Free	$0/month	0	Chosen

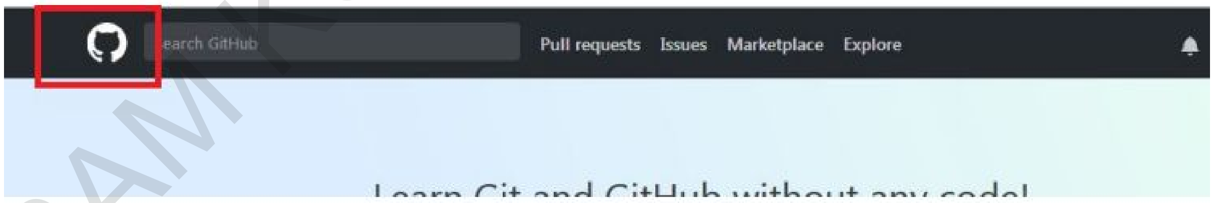
Each plan includes:

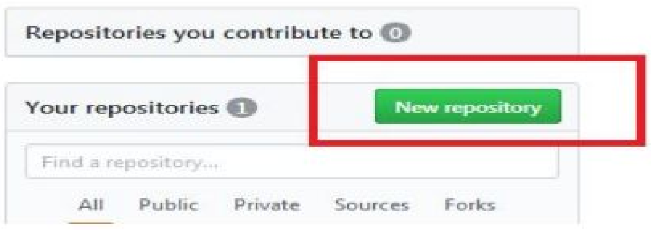
Unlimited collaborators

Unlimited public repositories

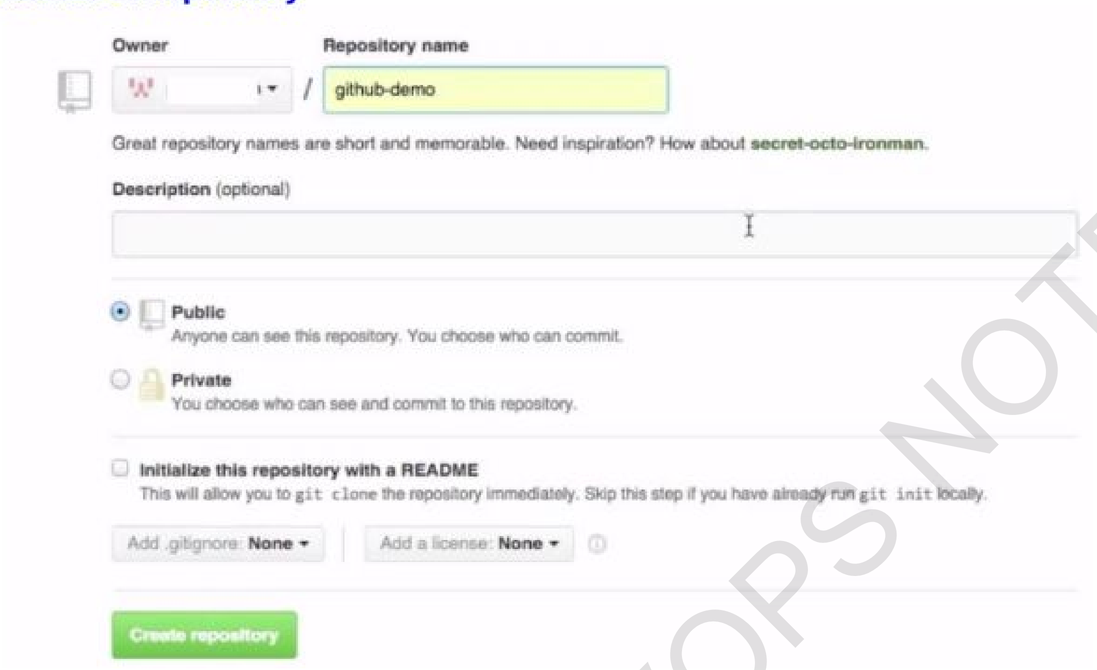
- ✓ Free setup
- ✓ SSL Protection
- ✓ Email support
- ✓ Wikis, Issues, Pages, & more

Click on logo to navigate for creating a new Repository





Fill the details of Repository


Owner
Repository name
github-demo
Great repository names are short and memorable. Need inspiration? How about secret-octo-ironman.
Description (optional)
Public
Anyone can see this repository. You choose who can commit.
Private
You choose who can see and commit to this repository.
Initialize this repository with a README
This will allow you to git clone the repository immediately. Skip this step if you have already run git init locally.
Add .gitignore: None
Add a license: None
Create repository

- Once the Repository is created on Github, now move to Local Machine.
- Create Projects folder in Local Machine to manage and clone the github repository