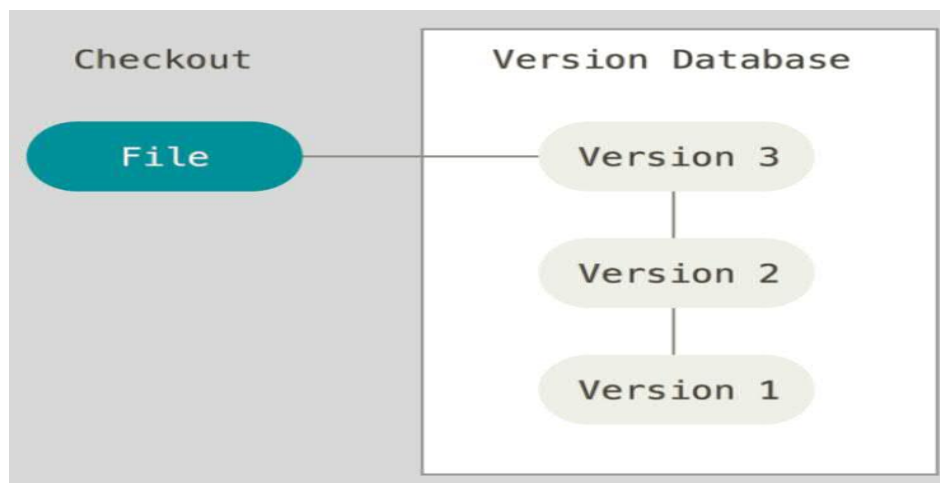


GIT

VCS (Version Control System):

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. For the examples in this book, you will use software source code as the files being version controlled, though in reality you can do this with nearly any type of file on a computer. It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also generally means that if you screw things up or lose files, you can easily recover. In addition, you get all this for very little overhead.



GIT:

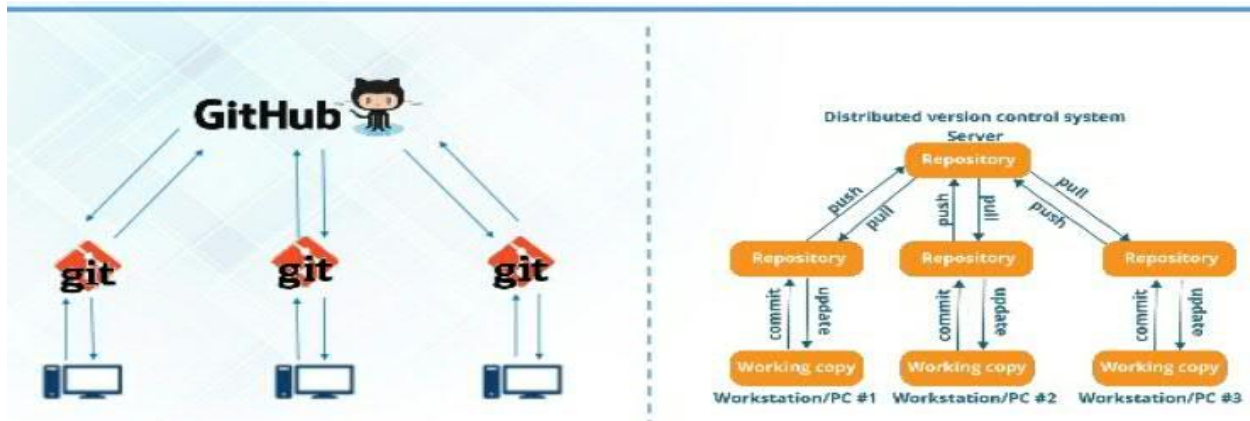
Git is a Version Control System (VCS) and that to be Distributed VCS which was developed by Linus Torvalds, the same engineer who developed Linux Operating System. The situation which made him code Git is the actual definition of VCS. Linus Torvalds thought of a version control system for his

operating system Linux because there did not match any available system to match his needs. This was much needed since Linux was open source and many developers wanted to contribute to Linux. A VCS would bring about a greater pace in the patch release and the development of this OS. All this made him develop GIT which is a version control system.

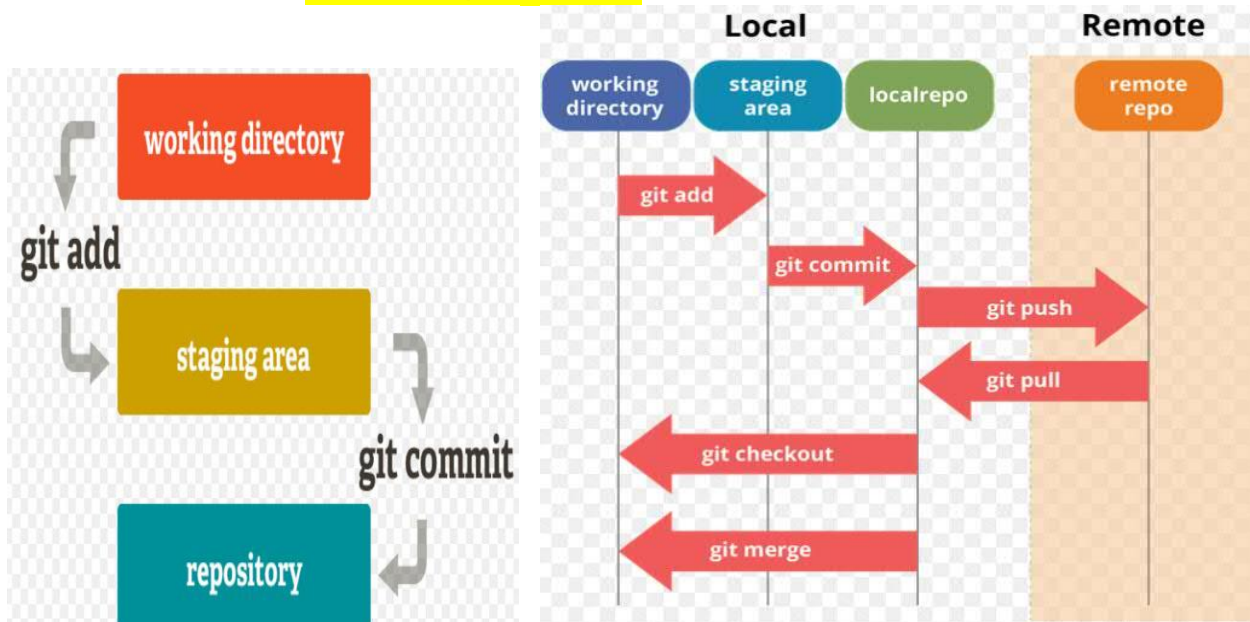
Difference between Git and GitHub:

Git: It is a Distributed Version Control System for tracking versions of files.

Github: It is a web portal and cloud hosting service for your Git repositories.



GIT working flow:



Branch in Git:

A branch in Git is simply a lightweight movable pointer to one of these commits. The default branch name in Git is master. As you initially make commits, you're given a master branch that points to the last commit you made. Every time you commit, it moves forward automatically.

Tag in Git:

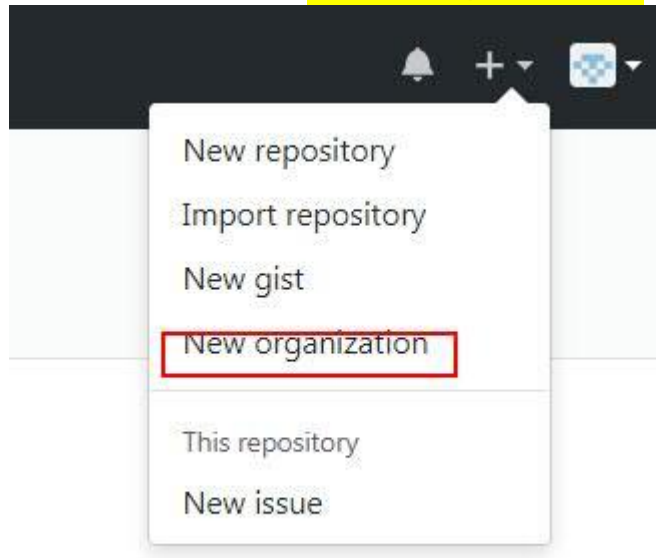
Tags are similar to branches, but the difference is that tags are immutable. It means, tag is a branch, which nobody intends to modify. Once a tag is created for a particular commit, even if you create a new commit, it will not be updated. Usually, developers create tags for product releases.

Working with GitHub Administration:

Creating organization:

Login to GitHub and click on the top right corner.


KC Technologies, Marathahalli, Bangalore. Contact: +91-7013558586/8123997688



Then give the details like Organization name, billing email, then select Free and click on Create Organization.

KC Technologies, Marathahalli, Bangalore. Contact: +91-7013558586/8123997688

Sign up your team

 Step 1: Set up the organization	 Step 2: Invite members	 Step 3: Organization details
--	---	---

Create an organization account

Organization name *

KCTechnologiesDevOps

This will be your organization name on <https://github.com/KCTechnologiesDevOps>.

Billing email *

kctechnologiesdevops@gmail.com

We'll send receipts to this inbox.

Organization accounts allow your team to plan, build, review, and ship software — all while tracking bugs and discussing ideas.

Choose your plan

<input checked="" type="radio"/> Free Unlimited users and public repositories	\$0
<input type="radio"/> Team Starts at \$25 / month which includes your first 5 users. Unlimited public repositories Unlimited private repositories	\$9 per user / month
<input type="radio"/> Business Includes everything in the Team plan, plus: SAML based single sign-on (SSO) Access provisioning 99.95% uptime SLA 24/5 email support with < 8-hour response time Learn more about our Business Plan or contact our team .	\$21 per user / month

The credit card and plan you choose will be billed to the organization — not KCTechnologies (your user account).

☐ This account is owned by a business.
[See our Corporate Terms of Service](#) for details.

By clicking on "Create organization" below, you are agreeing to the [Terms of Service](#) and the [Privacy Policy](#).


Create organization.

Then invite users to that organization.


KC Technologies, Marathahalli, Bangalore. Contact: +91-7013558586/8123997688

KC Technologies, Marathahalli, Bangalore. Contact: +91-7013558586/8123997688


Invite organization members



Step 1:
Set up the organization



Step 2:
Invite members



Step 3:
Organization details

Search by username, full name or email address

Invite

Continue

Organization members


✓ See all repositories ?

✓ Create repositories

✓ Organize into teams

✓ Review code

Then select the required details in the Step 3 (Organization details) or skip it. So that organization will be created.



KCTechnologiesDevOps

Repositories 0

People 1

Teams 0


Projects 0

Settings

This organization has no repositories.

Create a new repository

People 1 >

 KCTechnologies

Invite someone

Click on the “**Create a new repository**” to create new repository.

Then give the details of the repository like, Repository name, description, then select the Public account and other options and click on “Create Repository”.

KC Technologies, Marathahalli, Bangalore. Contact: +91-7013558586/8123997688


KC Technologies, Marathahalli, Bangalore. Contact: +91-7013558586/8123997688

Create a new repository


A repository contains all the files for your project, including the revision history.

Owner

Repository name


 KCTechnologiesDevOps ▾

MySampleRepo




Great repository names are short and memorable. Need inspiration? How about [studious-octo-fortnight](#).

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 let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

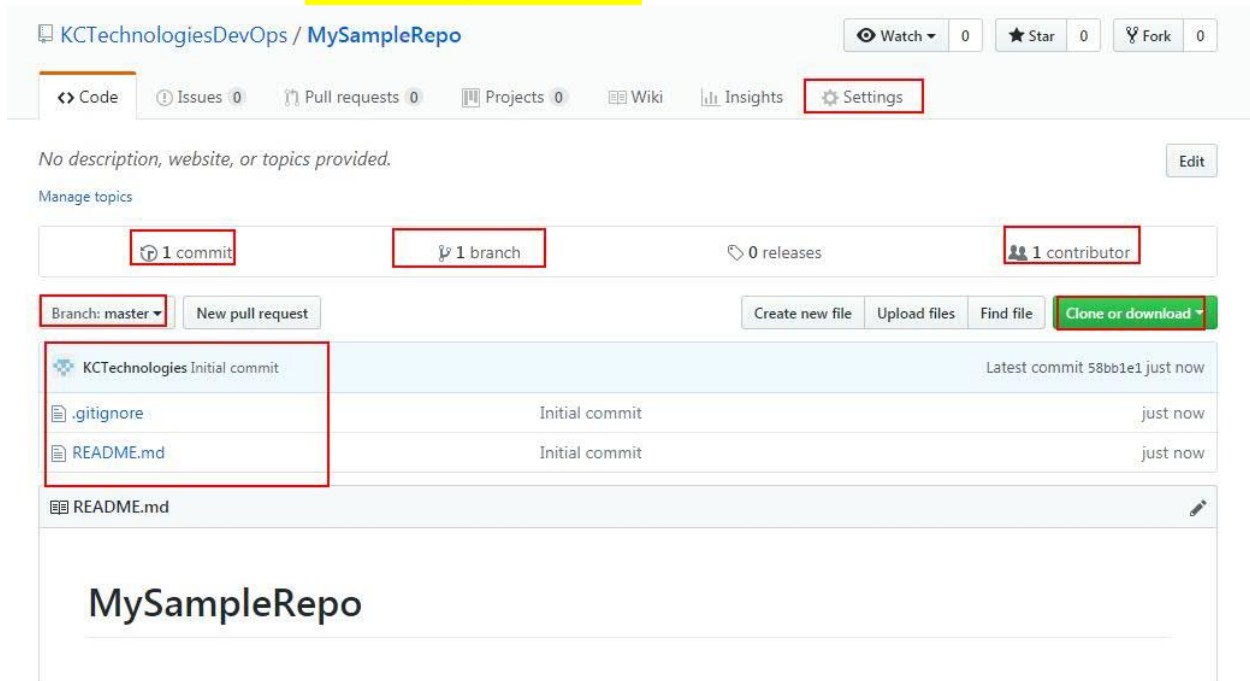
Add .gitignore: Java ▾

Add a license: None ▾ 

Create repository

So that a repository will be created with default master branch.

KC Technologies, Marathahalli, Bangalore. Contact: +91-7013558586/8123997688



Working with GIT Commands:

Reference:

git scm site contains the full documentation about GIT and its commands

<https://git-scm.com/docs>

Below are the sequence of commits needs to use for initial commit:

```
echo "# flipkartrepo" >> README.md
```

```
git init
```

```
git add README.md
```

```
git commit -m "first commit"
```

```
git remote add origin https://github.com/KCTechApps/flipkartrepo.git
```

```
git push -u origin master
```

```
git remote -v
```

To check whether our local repo is connected to origin

git remote -v

To get back file from staging to local workspace

git reset <filename>

KC Technologies, Marathahalli, Bangalore. Contact: +91-7013558586/8123997688

All the files in staging area will come back to Local workspace

git reset

git reset --hard --> Command to hard reset.

This will override the modified (tracked) file changes with the git repo branch code changes (HEAD version) and also removes the untracked files (new files).

git reset HEAD --> Command to hard reset

This will override the modified tracked file changes with the git repo branch code changes.

But will not do anything to untracked files (created in local but not committed to repo). Untracked files have to be removed by us only.

git clean -f

Will remove all the new files from work space.

git ls-files -m

Will give only modified files from working area.

git log

Will give all the commits with commit message, nothing but commit history

Generating SSH key to get password less authentication between local GIT and Remote Repo in HitHub:

To use password less authentication, we have to use SSH while cloning or else, use below line if you have already cloned using https

SSH clone url Syntax: **git@github.com:OrgName /RepoName.git**

Select the ssh url from Github clone or download option

URL syntax to change the origin type:

```
git remote set-url origin git@github.com:<Username>/<Project>.git
```

ssh-keygen -t rsa -b 1024 -c "emailid"

Explanation for above command:

ssh-keygen is a command

-t rsa is an encryption algorithm and it is optional. By default, it takes rsa encryption

-b 1024 indicated no. of bytes and it is optional. By default, it takes 1024 bytes

-c emailid is a comment and it is optional. By default, it will take computer name. But recommended to give the emailid.

ssh-keygen key will generate 2 keys.

Public and Private keys under home directory: ~/.ssh/

id_rsa and **id_rsa.pub** are the 2 keys

.ssh is a hidden directory

git commit filename -m "Message"

To add and commit a file at once

git add -u

To add all modified files to staging area at once.

git add -a or **-A** : To add all files to staging area.

git add -u : To add all modified files --> **this is the difference between add -u and add -a**

git diff filename: is the command which gives the difference between our local file changes and the changes we pulled using pull command. (Use this command before committing to local repo).

Ex: If you want to see what you haven't git added yet:

```
git diff myfile.txt
```

or if you want to see already added changes

```
git diff --cached myfile.txt
```

git checkout <versionid> ---> To get back to the code of given version. Here untracked files (created in local but not committed to repo) will not be removed.

git checkout <branchname> ---> To get the latest code from specified branch. Here untracked files (created in local but not committed to repo) will not be removed.

In **.git** folder, we have many files. Those files are responsible to take care of commits and its versions ids.

git branch

To know the no.of branches present in that git repo.

Here, * indicates the branch we are connected to.

git branch <branchname>: Command to create the branch name

git checkout <branchname>: It will connect to the specified branchname

git checkout -b <newbranchname>:

Single command to create new branch and take the code from connected repository and then switch to the newly created branch.

First connect to one branch. Then

git merge master

This command will merge code from master branch to connected branch

git merge master filename

This command will merge only specified files from master branch to connected branch

git push --all <alias name origin>: Will move all the branches from local to remote repository

git branch -d <branchname> :

It will delete the branch, if the given branch and the branch from which this branched is cloned are merged.

git branch -d -f <branchname>:

It will delete the branch forcibly, even if the given branch and the branch from which this branch is cloned are not merged.

After deleting from local, to delete in remote repo use this command:

git push aliasname :bugfix -> Space is mandatory here after alias name(ex: git push origin :development)

git clone cloneurl: To clone the code (Used only first time)

get fetch: Will get the code from remote repo to local Git repo, but not to local workspace.

Again we have to use **git merge** to get the latest code from local git repo to workspace

git fetch and git pull are the 2 commands to take the updated code from remote repo

To rename Branch:

git branch -m <oldname> <newname>

git branch -m <new branch name> --The current branch name will be renamed.

git revert --no-commit <commitid>..HEAD: To get back code to given commit id.

git reset --hard c14809fafb08b9e96ff2879999ba8c807d10fb07: To get back code to given commit id. Send you back to how your git clone looked like at the time of the checkin.

Difference between git fetch and git pull commands:

git fetch will take the code from Remote repo to local repo only.

git pull will take the code from Remote repo to local workspace directly.

Setting your Git username for every repository on your computer

Setting user email:

git config --global user.name "MyName"

Set a Git username:

git config --global user.email "abc@xyz.com"

git config --list: To list all your configurations related to GIT

New notes:

git status ---> This will give the list of the files added to Staging.

Plumming: Commands used by git for internal process and these can exists in <https://git-scm.com/docs> under plumming commands and we can test those commands.

Ex: **git cat-file -p commit id_id**

tree: if parent structure

id is the file

KC Technologies, Marathahalli, Bangalore. Contact: +91-7013558586/8123997688

Difference between git pull and git checkout: **git pull** will take the latest code from github to local repo, whereas **git checkout** will take the code from given commit id based on history Or switch to another branch. Git checkout is also used to switch between branches.

Create a new branch in local and create a file in that, then if you want to commit to github:

git push origin nameofthebranch --> Newly created branch will be committed to GIT repo.

git log --oneline --graph --> This will give the log history in a graphical manner

OR

git log --graph -->

When we do branch merge, then we see Fast-Forward message. What it is Fast-Forward merge:

Fast forward means merging code from one branch to another existed branch, but not creating a new branch

Branch merging:

Connect to one branch, then use **git merge branchname**. Then code from the given branch will be merged to currently connected branch.

What is detached state or Detached-HEAD:

=====

Always a branch head (commit id) should contain a reference (commit id in the HEAD file). It means your local workspace will be connected to a branch, which means it is connected to that branch.

Whenever our HEAD file contains a commit id directly, then it is called as detached state, this is not a correct process.

Always HEAD should contain a link instead of direct commit id. **That is the reason git checkout version id is not a good practice.**

You cannot do any commits when the HEAD state is Detached state, i.e when HEAD contains commit id.

KC Technologies, Marathahalli, Bangalore. Contact: +91-7013558586/8123997688

Difference between git pull and get fetch:

=====

PULL = FETCH + MERGE

git fetch: Will get code from remote repo to local repo. But it will not tell which branch it has to be part of.

git merge: Will take code from local repo to local branch and workspace.

Tags:

=====

Tag is nothing, but a label given to a commit id while creating a backup branch.

Tag and Branch are similar, but Tag has 1 difference i.e

Whenever you create a branch, its commit id may change, but when you create a Tag, its commit id remains same.

Tag has a static commit id always. Generally, we create Tag when we take a branch after Prod release or Code freeze.

To create tags:

=====

Search in google for "Git tags".

Tags are of 2 types, 1) Lite weighted, 2) Annotated.

Annotated is nothing but:

Connect to a branch, then run below command

git tag -a <tag name> -m <some message>

Then it will create a tag for the connected branch

To see the created tags, run below command

Command: **git tag**

To connect to Tag, we can change in HEAD tag and give the tag name reference, but this is not a correct way even though it works fine by connecting.

To connect to that created tag:

git checkout <tag name>

Tag name can be anything. It is our own name.

To push all created tags to GitHub: **git push --tags**

Git stash:

=====

It is nothing like taking a backup in local repo without committing to GIT remote repo with some name called stash name. For example, half done work is taken backup using stash, and later time we can use and continue working.

Cherry pick:

=====

We made 10 commits to my local git repo, but I want to commit only 1 or 2 to remote repo. If I use git push, then it will commit all commits to remote repo, but I want to commit only 2. In this case we use cherry pick to commit only specific commits from local git repo to remote git repo.

Who don't see .git folder?

=====

Go to Organize on left top, folder and search option, view, then enable "view hidden files and folders options" option. If not, then it will not show .git file

Deleting branch in Git:

Difference between branch and tag in GIT?

How to push the tags to git repo?

Command to see all tags?

Git stashing:

=====

To take backup of uncommitted code in local git repo we use stash.

command: **git stash** --> Currently connected branch will be stashed with existing changes including untracked files in local repo and this will make the working directory clean.

Then you can continue working on something with the same code in local. And you can commit those new changes to remote repo also.

But now, I want to go to the stashed code and not to the new code, then we can use below stash commands.

command: **git stash list** --> will give all the stash list. We can have multiple stashes.

git stash pop --> Pop will apply the changes to local and then removes stash from the stash list.

git stash apply --> Apply will apply the changes to local but will not remove stash from the stash list.

You can try pop and apply with git stash pop and git stash apply commands.

And then verify using git stash list

git rebase:

=====

For example, if we create a new branch from already existing branch at certain commit id.

Later the actual(old) branch is added with some new code with new commit id.

Now the new branch wants to connect to the old branch new commit id instead of old commit id where this new branch was created.

This we can achieve with **git rebase**.

Rebasing: Rebasing is changing the place (commit id) where the branched off (branch created).

To work with rebase, connect to one branch (master) and then create other branch (uat) as master branch as the from branch. Check the base commit of uat branch. Then commit few changes to master.

Then again connect to uat branch and commit few changes.

Then use "**git rebase master**". So the base commit of uat branch will be changes with the latest commit id of master.

Difference between merging and rebase.

Merging and rebase are similar.

Merging will contain history, but rebasing will not contain history.

With merge we will contain the new code from new branch and will merge the new code from old branch to new branch. We may get merge conflicts and we must fix those.

But, if we do rebasing, the new branch will lose the history and new code changes, and will get the new code from old branch only.

The current branch we are on(new branch), from there we have to do rebase to take code from old branch.

Git cherry pick:

=====

For example, I have to work on dev branch, but I worked on master branch and committed my code. Now to move specific commits to master branch, then I can cherry pick those specific commits and move those to dev branch.

.git/objects --> Place where all git commands exists
SHA1 is the algorithm used by git to encrypt its files and store.

Cherry pick example:

cherry-pick is a Git feature. If someone wants to Commit specific commits in one branch to a target branch, then cherry-pick is used.

git cherry-pick steps are as below:

checkout (switch to) target branch.

git cherry-pick <commit id>

Here commit id is activity id of another branch.

Ex: git cherry-pick 9772dd546a3609b06f84b680340fb84c5463264f

Then push to target branch

What is .gitignore file?

Ignored files are tracked in a special file named .gitignore that is checked in at the root of your repository. There is no explicit git ignore command: instead the .gitignore file must be edited and committed by hand when you have new files that you wish to ignore. .gitignore files contain patterns that are matched against file names in your repository to determine whether or not they should be ignored.

Refer below url for more details:

<https://www.atlassian.com/git/tutorials/saving-changes/gitignore>

In google, we can search for online git ignore, if you have java there, then it automatically generates .gitignore file for java projects automatically.

Search with vagrant word for vagrant files

Git hooks:

How to trigger a Jenkins build on git commit

I spent quite some time figuring this out, but was happy when it worked. This is for Windows. In Linux, the process is the same, but it's far easier.

First, you have to know that if you've installed Git in Windows, it comes with [curl](#) pre-installed. So just open the Git bash prompt and start typing your commands.



Let's say you have a Jenkins job setup on localhost, named "someJob". To initiate a build, just type the following command in the Git Bash prompt:

`curl http://localhost:8080/job/someJob/build?delay=0sec`

That's it. Once you've typed the command, just have a look at your Jenkins dashboard and you'll see your project building.

Now to make a Git commit trigger the build, you have to navigate to the hidden ".git" directory in your git repository. Enter the "hooks" directory and you'll see there, a "*post-commit.sample*" file. You can open it with Notepad++ to see the contents of it.

Make a copy of the file and name it "*post-commit*". It does not need an extension. This is the file that Git will invoke whenever you do a commit to this particular Git repository. Don't worry about the fact that it looks like a Linux bash script. When Git triggers it, it will run even in Windows. If the file has a line ": Nothing", you can remove that line and insert this new line:

`curl http://localhost:8080/job/someJob/build?delay=0sec`

Now save the file, make some changes to your git repository and commit the changes. The moment you commit, you'll see the build being triggered in Jenkins! Awesome! :)

Some important points:

try git init command

Then a .git file will be generated.

GIT will track only created/modifies/deleted files, but not any modified folders. GIT tracking is based on files but not on folders.

Search in google with "git cheat sheet git commands" --> you can get all GIT related commands

Search in google with "git cheat sheet linux commands" --> you can get all linux related commands.

Reference url: <http://ndpsoftware.com/git-cheatsheet.html#loc=workspace>;

GitHub Account creation:

Go to GitHub account creation page, then give the details and then select Free account and click on Continue.

KC Technologies, Marathahalli, Bangalore. Contact: +91-7013558586/8123997688

Welcome to GitHub

You've taken your first step into a larger world, @KCTechnologies.

✓ Completed
Set up a personal account

🔧 Step 2:
Choose your plan

⚙️ Step 3:
Tailor your experience

Choose your personal plan

Every plan comes with GitHub's most-loved features: Collaborative code review, issue tracking, the open source community, and the ability to join organizations.

✓
Free

\$0
per month

Includes:
Personal account
Unlimited public repositories
Unlimited collaborators

There are millions of public projects on GitHub. Join one or start your own for free.

Developer

\$7
per month
(view in INR)

Includes:
Personal account
Unlimited public repositories
Unlimited private repositories
Unlimited collaborators

Free for students as part of the [Student Developer Pack](#).

☐ **Help me set up an organization next**
Organizations are separate from personal accounts and are best suited for businesses who need to manage permissions for many employees. [Learn more about organizations](#)

☐ **Send me updates on GitHub news, offers, and events**
Unsubscribe anytime in your email preferences. [Learn more](#)

Then select few details and click on Submit.

Welcome to GitHub

You'll find endless opportunities to learn, code, and create, @KCTechnologies.

✓ Completed
Set up a personal account

🔧 Step 2:
Choose your plan

⚙️ Step 3:
Tailor your experience

How would you describe your level of programming experience?

☐ Totally new to programming ☐ Somewhat experienced ☒ Very experienced

What do you plan to use GitHub for? (check all that apply)

☒ Development ☐ Project Management ☐ Design

☐ School projects ☒ Research ☐ Other (please specify)

Which is closest to how you would describe yourself?

☐ I'm a student ☐ I'm a hobbyist ☒ I'm a professional

☐ Other (please specify)

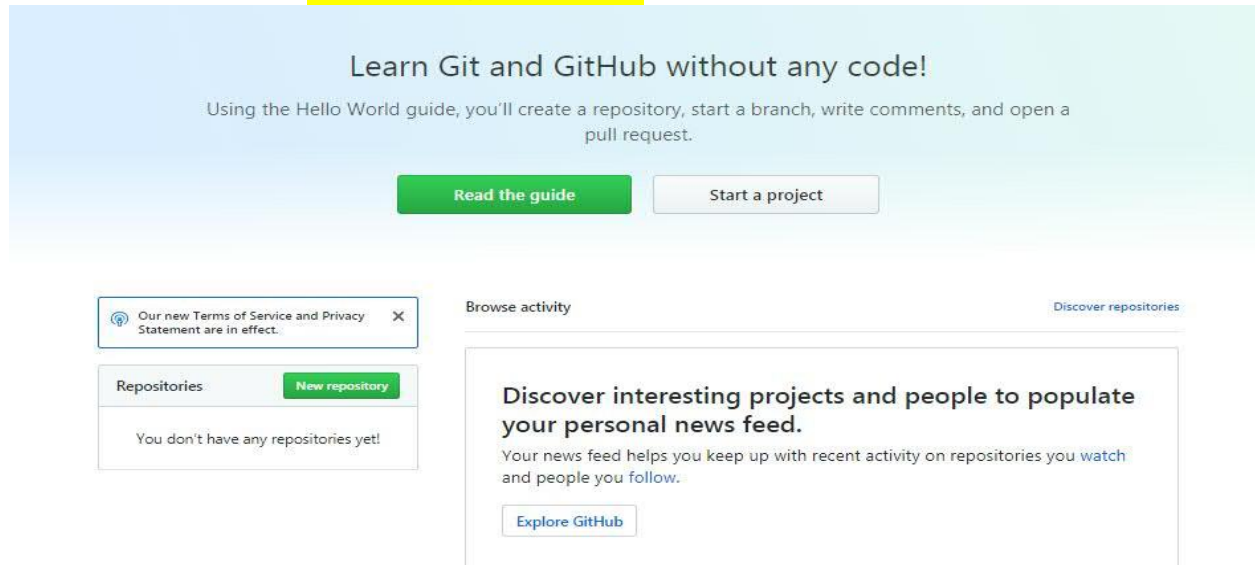
What are you interested in?

e.g. tutorials, android, ruby, web-development, machine-learning, open-source

After this, your GitHub account will be created successfully. But you need to verify your email to create repositories.

KC Technologies, Marathahalli, Bangalore. Contact: +91-7013558586/8123997688

KC Technologies, Marathahalli, Bangalore. Contact: +91-7013558586/8123997688



Then go to your email inbox and verify your email. Then only you will be able to create repositories in GitHub.

Almost done, **@KCTechnologies!** To complete your GitHub sign up, we just need to verify your email address:
kctechnologiesdevops@gmail.com.

Verify email address

Once verified, you can start using all of GitHub's features to explore, build, and share projects.

Button not working? Paste the following link into your browser: https://github.com/users/KCTechnologies/emails/64342749/confirm_verification/df6c0921dcb2f6ab537bf7b0c45e0e1e4dc45cb9

You're receiving this email because you recently created a new GitHub account or added a new email address. If this wasn't you, please ignore this email.

KC Technologies, Marathahalli, Bangalore. Contact: +91-7013558586/8123997688