**Git and GitHub**

**Introduction**

**Git:** git is a version control system. It supports non-linear workflows by providing data assurance for developing quality software.

* To track changes in files / folders
* To collaborate in teams
* It is free and open source.
* We are working on our local system and we have a single file that only you are working on in that case it makes very little sense to use a VCS.



When multiple people working on the same file in that case VCS becomes very important because person A works on that file and makes changes. Person B also works on the same file and makes changes. We have to track all the changes made by all the people on that particular file. So the VCS plays important role in that case.

Two types of VCS:

**Centralized**



The repository is located on remote server. The people who are working on this repository have to be connected always and have to be online to work on their systems because this repository is placed on a server. They work on this repository online and make their changes committed and so they can collaborate. But there is major disadvantage is a single repository is placed on a server. So if anything goes wrong it will be difficult for us to maintain the backup and bring the repository to the normal.

**Distributed**



Here every person can pull and create his local repository so you can see workstation one has a local repository workstation to has his own copy workstation. Major advantage is if anything goes wrong here on the server we can backup the repository from any local system because these are the complete copy of the repository and also do not need to be online. Every time you can pull the repository take your local copy and keep on working on it in offline mode.

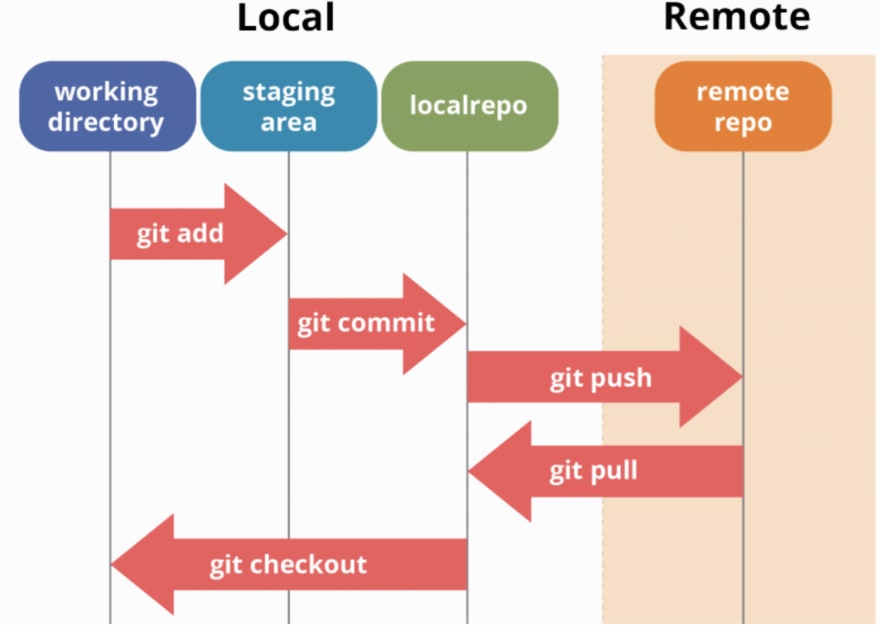
Git is a DVCS.

**GIT HUB** – a website to upload your repositories.

It provides backup, visual interface to your repository. It makes collaboration easier.

Git and GitHub is not same. Git is a VCS whereas GitHub is a website to upload our files/folders.

A simple workflow:



**Git Setup on Local System:**

**Step 1**: check if git is already installed.

git --version

**Step 2:** download Git installer from [https://git-scm.com/](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbmtrdVFJMVNZRmtXTlNXUm1BQWJJS19abERlZ3xBQ3Jtc0tuMGxNUFJXc2h4XzE0TW9yWUd3VmMycnhVcWxMQmduRE5TbXN6SXRYc0dsR0ViQl94RlpGSTVRclB5RGZCc0Z5enE0SzFMQXZsU25Rbnd5QVB5cDJ5U2NrUUVJcG5SUGt5YnNFWl81YVZOWHZtM212RQ&q=https%3A%2F%2Fgit-scm.com%2F&v=B8NurikxEo8)

**Step 3:** Signup and create a account on GitHub [https://github.com/](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbklHQlFILWRiVXp3N1BhelBrTEgtR3BnT3RTd3xBQ3Jtc0tuSjBLZF8zMWtCanFfbXZ4YWk1bm1aTnFWdDJ1X0xFZXJyaFEzQjRrTTc2WlN6WVBHdTZneGpwVWJCT1hRR0RqUVBsUG96MFI0c05ZQVRYZ3NNSm96ZHgweEp1TXk1cWFNN3RyOEw5WU4tMWh3X2dkRQ&q=https%3A%2F%2Fgithub.com%2F&v=0Icla6TVNNo)

**Step 4:** Add your github email and username to git

git config --global user.email "yourGitHub@email.com"

git config --global user.name "yourGitHubusername"

**Step 5:** Add file/folders to git - tracking

**Step 6:** Commands - terminal - goto the location of the folder/project

– git init

- git status

- git add

- git commit -m “…..”

- git remote add origin “location of remote repo”

- git push -u origin master

- git log

- git --help

**Branching and Merging**

**Step 1:** Create branch **git branch “branch name”** **Step 2:** Checkout branch **git checkout “branch name”**

**Step 3:** Merge new branch in master branch **git merge “branch name”**

**Step 4:** Delete branch **git branch -d “branch name”** — delete from local git push origin —delete “branch name” — delete from remote

Branch : work on multiple things at the same time. Branches are master, hotfix, release, develop, feature

1) **git branch** to check the current branch we r in

2) **git branch -A** to check all the branches

3) **git branch -a** short information about current branch

4) **git branch <branchname>** to create a branch

5) **git checkout <branchname**> to switch over to that branch.

6) **git checkout -b <branchname>** both create and switch over to that branch

7) **git merge <old> <new>** to merge the branches

8) **git branch -d <branchname>** to delete a branch

**Git Tags**

Tagging in git or any other VCS refers to creating specific points in history for your repository/data. This is usually done to mark release points. (v1.0, v1.1, …)

Why we create tags?

To mark release points for your code/data.

To create historic restore points.

When to create tags?

When you want to create a release point for a stable version of your code.

When you want to create a historic point for your code/data that you can refer at any future time. (to restore your data)

**How to create Tags in Git**

**Step 1**: Checkout the branch where you want to create the tag.

**git checkout <branch name>**

**Step 2:** create tag with some name**.**

**git tag <tag name>**

**git tag -a v1.0 -m “ver1 of..” ( to create annotated tags)**

**Step 3:** Display or show tags

git tag

**git show v1.0**

**git tag -l “v1.\*”**

**Step 4:** Push tags to remote

**git push origin v1.0**

**git push origin –tags**

**git push –tags**

(to push all tags at once)

**Step 5:** Delete tags (if required only)

To delete tags from local:

**git tag -d v1.0**

**git tag –delete v1.0**

To delete tags from remote:

**git push origin -d v1.0**

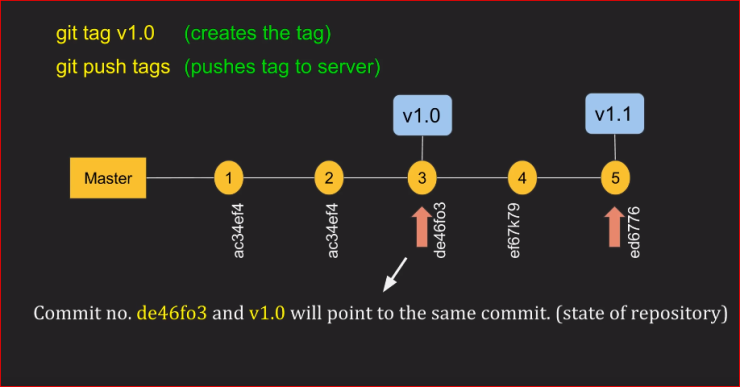
**git push origin --delete v1.0**

**git push origin :v1.0**

To delete multiple tags at once:

**git tag -d v1.0 v1.1 (local)**

**git push origin -d v1.0 v1.1 (remote)**



**How to checkout TAGs**(when required**)**

**We cannot checkout tags in git**

**We can create a branch from a tag and checkout the branch**

**git checkout -b <branch name> <tag name>**

eg: git checkout -b ReleaseVer1 v1.0

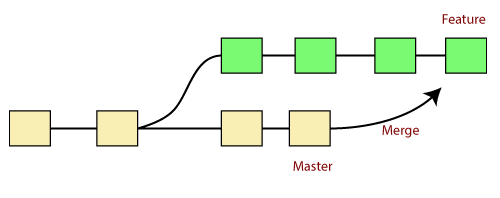
**Can I create a tag from some past commit?**

Yes

**Git tag <tag name> <reference of commit>**

Eg: git tag v1.2 5fcdb03

**git merge**

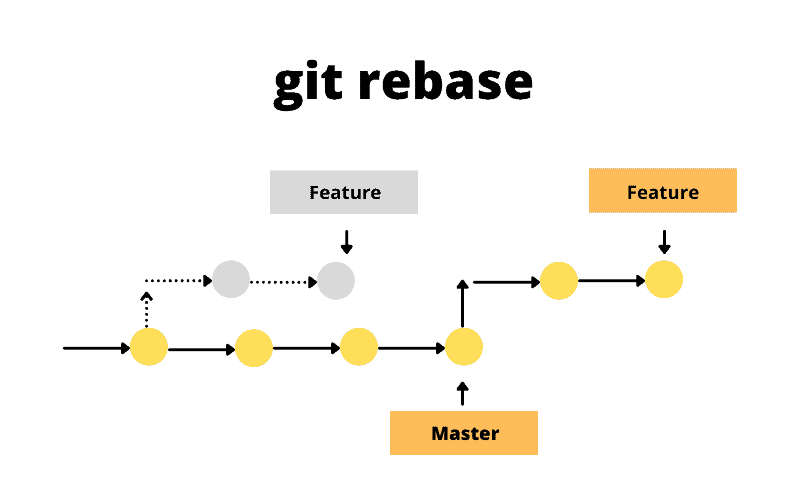


- Is a non-destructive operation

- Existing branches are not changed in any way

- Creates a new merge commit in the feature branch

**git rebase**



- Moves the entire feature branch to begin on the tip of the master branch

- Re-writes the project history

- We get much cleaner and linear project history.

How to access token on GitHub?

While pushing to github you will NOW need to provide ACCESS TOKEN in place of password

To generate Access Token

1. Login to your GitHub account

2. Verify your email address, if it hasn't been verified yet. 3. In the upper-right corner of any page, click your profile photo, then click Settings.

4. In the left sidebar, click Developer settings.

5. In the left sidebar, click Personal access tokens.

6. Click Generate new token.

7. Give your token a descriptive name.

8. To give your token an expiration, select the Expiration drop-down menu, then click a default or use the calendar picker

9. Select the scopes, or permissions, you'd like to grant this token. To use your token to access repositories from the command line, select repo.

10. Click Generate token

# How to Commit, Push, Pull from Eclipse to GitHub?

# Step 1: Create GitHub account and SignIn

# Step 2: Start a Project = Create a repository

# Step 3: Start Eclipse

# Step 4: Goto Perspective - Git Repositories and click on Add Git Repo

# Step 5: Create a project in Eclipse

# Step 6: Do a right click on Project - Team - Share - Add to git repo

# Step 7: Commit and Push the project to the repo

# Step 8: Commit and Push every change to the repo

**Git ignore**

touch .bak .exe .c

{vi .gitignore

.bak

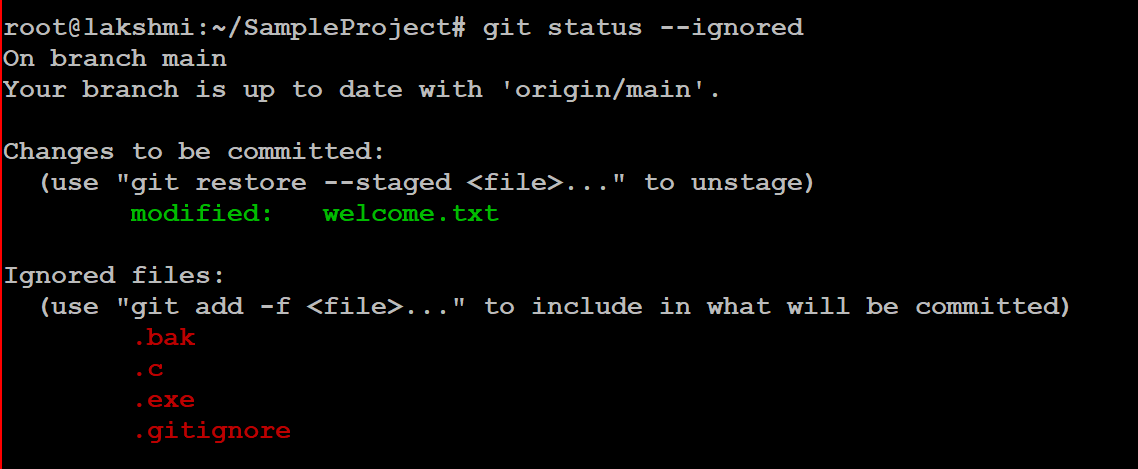
.exe

.c

.gitignore}

git add .

git status –ignored



**Git Stash**

Sometimes you want to switch the branches, but you are working on an incomplete part of your current project.

You don’t want to make a commit of half-done work. Git stashing allows you to do so.

Normally when you switch branch you will commit the code and switch to the new branch.

If you switch branch without committing. Two things will happen.

1. Switches to the branch carrying the changes.
2. Git will not allow to switch the branch and asks to commit or stash the changes.

The git stash command enables you to switch branches without committing the current branch.

The stash’s meaning is “to store something safely in a hidden place.” The sense in Git is also the same for stash; Git temporarily saves your data safely without committing.

**Convert java project to maven**

Right click on the folder package 🡪 configure 🡪 convert to maven project 🡪 right click package 🡪 showin 🡪 terminal 🡪 **mvn clean package** 🡪 **cd target** 🡪 refresh target file from left 🡪 **java -jar filename**

**To change the font size:**

**On the Eclipse toolbar, select Window → Preferences, set the font size (General → Appearance → Colors and Fonts → Basic → Text Font)**. Save the preferences.



How to build a jar file in Jenkins?

Jenkins 🡪 new item 🡪 freestyle project 🡪 scm-git 🡪

Build-invoke top level-goal clean install 🡪 apply and save 🡪 build now.

sudo yum install java-1.8.0-openjdk-devel

