1. **The need for GitHub**
2. **What is GitHub**
3. **Getting started with GitHub**
4. **Create a Repository**
5. **Create a Branch**
6. **Make a commit**
7. **Open and Pull Request**
8. **Git ssh and Branching**
9. **GitHub**
10. **Git Branching Strategy**
11. **Git Flow, stash, cherry-pick, rebase, merge**
12. **GitFlow Bitbucket**
13. **The need for GitHub**

Ans: - It is extremely important for software developers to work on a web-based platform to share their projects, and collaborate with other developers. This platform must be a version control system that is it must enable multiple people to simultaneously work on a single project. Each person edits their own copy of the file and chooses when to share those changes with the rest of the team. This application must also be capable of hosting millions of programmers and hobbies that download and evaluate each other’s work. GitHub is one such platform of choice for developers that can host multiple programmers and review their code.

GitHub has several competitors for instance GitLab – GitLab is an open-source web interface and source control platform based on Git whereas Microsoft teams’ foundation server is an enterprise great server to share code and track work and ship software for any language all in a single package.

Bitbucket on the other hand stores all your git and marketable source code in one place with unlimited private repositories.

So, what really makes GitHub so popular and powerful among developers,

It’s an Opensource platform and the community is really what fuels in. Moreover, GitHub is a platform of choice for developers from various large corporations too. Microsoft is the No 1 contributor to the system.

But there are also other Google Airbnb sap PayPal IBM and many others. Exposure on inside that you get on GitHub is simply unmatchable by any other platform, here you can discover the code written by others and you learn from it and even use it for your own projects.

Versions control on GitHub works very much like Microsoft office or google drive. It Simply tracks all the changes made to your code and who makes them. You can always review the detailed change log that neatly hosts the relevant information.

Using GitHub eliminates the need for complex corporate security solutions because of everything in the cloud. The platforms protect code branches, verifiers commit signing, and control access.

**Setup git workflow in practical by using ssh (secure shell)**

1. **Git On Server**
2. **Distributed workflows**
3. **GitHub**
4. **Advanced Git commands.**

**GitHub, GitLab, Bitbucket –** For all underlying 3 tools have the same open-source Git Engine.

**These are all based on Git, Enterprise paid tools. They have taken the OpenSource Git tool and added all these missing features and started selling it.**

As Enterprise Git = we use central for these activities:

1. To collaborate with other Devs
2. Easy to use UI to view the code
3. To merge the code
4. To Approve or Reject

**Sharing and Updating Projects:**

* + - 1. git fetch
      2. git pull
      3. git push
      4. git remote
      5. git archive
      6. git submodule

**Debugging:**

* + - 1. git bisect
      2. git blame
      3. git grep

**Patching:**

* + - 1. git cherry-pick
      2. git merge
      3. git rebase
      4. git revert

**Email:**

git apply

git am

git format-patch

git imap-send

git send-email

git request-pull

**External Systems:**

1. git svn
2. git fast-import

**Administration**

1. git gc
2. git fsck
3. git reflog
4. git filter-branch

Working with Remotes

To be able to collaborate on any Git project, you need to know how to manage your remote repositories. Remote repositories are versions of your project that are hosted on the Internet or network somewhere. You can have several of them, each of which generally is either read-only or read/write for you. Collaborating with others involves managing these remote repositories and pushing and pulling data to and from them when you need to share work. Managing remote repositories includes knowing how to add remote repositories, remove remotes that are no longer valid, manage various remote branches and define them as being tracked or not, and more. In this section, we’ll cover some of these remote-management skills.

Remote repositories can be on your local machine.

It is entirely possible that you can be working with a “remote” repository that is, in fact, on the same host you are. The word “remote” does not necessarily imply that the repository is somewhere else on the network or Internet, only that it is elsewhere. Working with such a remote repository would still involve all the standard pushing, pulling and fetching operations as with any other remote.

Showing Your Remotes

To see which remote servers you have configured, you can run the git remote command. It lists the shortnames of each remote handle you’ve specified. If you’ve cloned your repository, you should at least see origin — that is the default name Git gives to the server you cloned from:

Setup Git workflow in practical by using .ssh (secure shell)

Create one local directory

>Cd 730am

Open gitbash and generate a ssh keys:

$ ssh-keygen

/c/users/new/.ssh/id\_rsa // private key

/c/users/new/.ssh/id\_rsa.pub // public key

$ cat .ssh/id\_rsa.pub

//copy public key and paste in

Github.com/account

* settings=> sshkeys and gpg => sshkeys => add new=>
* paste here.

Now create a repository in github: add readme

$ git clone [git@github.com:vinodh/git730amrepo.git](mailto:git@github.com:vinodh/git730amrepo.git)

$ ls

Git730amrepo

$ cd git730amrepo

git730amrepo $ vim file1.java

$ git add file1.java

$ git commit -m “file1added”

$ touch sample.txt

$ git add .

$ git commit -m “commit sample”

$ git log

$ git status

$ git push

Accesss from another directory, clone it.

Work1> touch sample.log

$ git add. && git commit -m “add sample”

$ touch f1 f2 f3

$ git status

$ git add \*

$ git status

$ git commit -m “add inn files”

$ git push

$ cd git730repo

Update the repo now.

$ git pull

$ git log

$ git status

$ git log –oneline

$ git pull

$ touch f5

$git add f5

$git commit -m “f5”

$ git push

// without updating work1 person pushing files

Work1> git push

Rejected

Work1$ git pull

$ git push

// now everyone has same files

How we rare doing in real time? In companies we follow branching strategy.

Branching strategy

git flow

github flow

trunk based

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

What is a branch?

$ git branch

\*main

// now we are in main branch

$ git branch feature/auth

$ git branch

feature/auth

\*main

$ git checkout feature/auth

// now all the files form main branch come to feature/auth

$ touch file100

$ git status

$ git add .

$ git commit -m “added”

$ git branch

\* feature/auth

$ git checkout main

$ ll

// there is no file100

// branching promotes parallel development

$ git checkout feature/auth

$ touch file101

$ git add . && git commit -m “file101 added”

$ ll

$ git checkout main

// merging should be done from main branch

$ git merge feature/auth main

$ git push

$ git branch -d feature/auth

// gitflow model is important

// so far what we done is trunk based, we developed and push the code to repository.

$ git branch win

$ git branch linux

$ ll

$ git branch

Linux

\*main

Win

$ git checkout win

$ touch winfile

$ git checkout linux

$mkdir linux

$cd linux

$touch linuxtest

$ git add . && git commit -m “linux committed”

$ git checkout main

$ git merge win linux

$ git merge win linux

$ git push

// observe in remote merge happens but branches not created.

// but in local we have win and linux branches.

$ git push - - all

// the branches also pushed with above command.

$

// now we are working on command prompt, let us use some client.

Note: **pullrequest**

**Code review**

**Crating branches**

// branch is nothing but the exact copy of from where you are creating.

//

================

Git branching strategy:

1. Git flow model:

Some devops projects followed using trunk-based model, the new projects trunk based model and old projects using gitflow model.

The story begins:

=================

In gitflow model default branch is main/master:

Create a repository in github you will get automatically main branch.

Create another branch develop in github:

Now create other branches for each feature.

Develop => feature/f1 , feature/f2, feature/f3

Now 3 branches out of develop branch got created.

Now, merge the branches to develop after work got completed.

Now each branch is tested by testers separately and

Needs to merge now. How to do that?

Now, the pullrequest concept came into scene.

Go to => Pull requests tab

Base: develop <= compare: feature/f1

// You can add reviewers, assignees, any seniors

// Normally team lead will merge the code.

// now pr got approved and feature/f1 got merged.

You can delete the branch here.

Similarly, you can raise pr to feature/f2 and feature/f3

Now, in develop branch has all features available,

Now client is forcing to release the product /project

Now we have to crate another branch called release/1.0 branch from develop branch.

Now the testing team performs the end-to-end testing on the release branch.

If everything is ok, testing, quality, code, functionality

Then we raise pull request to the main branch from release branch.

If any bug found by testing team, now create bugfix branch from release branch.

Now inform to testing team to test the bug fixing branch, Now the branch is ok, now merge the bugfix into release branch.

Now merge the bugfix branch also to develop branch also.

Bugfix branch merged to both release and develop branches.

Now merge the release branch in to main branch.

(Release and develop have same code, so, merge the release to main branch).

Now tags ….

Now from the main branch, you have to tag it.

Why tagging is required because this is production major release that’s why.

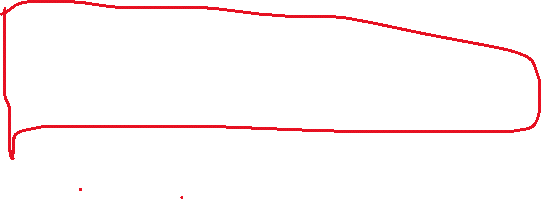
Using tagging I can identify how many features I am adding to production.

Go to tag, create a tag

Releases => TAG-1.0

TAG1.0 <= Target: main

Release-1.x => This is the name to customer.



// signoff



After signoff then the Publish the code.

Now in Release-1.x

Under TAG-1.0

Now, another feature needs to be developed,

Create, feature/f4 from develop branch.

Add the file4 . java

Create a pr to develop branch .

Now , develop have featrure/f4 content ..

Now plan for release .

Now create release2.0 branch from develop

Now pr from release2.0 to main branch.

Now got tag => releases => choose a tag => release2.0

Now publish code and consider bug found in production code.

Now create hotfix branch form main branch.

Now add code to fix the code.

Commit the changes.

Merge to main branch.

Create pull request => approved

Now in the main branch all the code available.

Now go to tag => releases => tag-2.0-HFIX

Tag: Releases-HFIX

Read: Gitflow workflow from bitbucket page

Merge hotfix to development and main branch.

Pull requests are done form github and development (commands) from the command line.

====

// developer responsibility is to create a repositoy and establish the build starategy:

Create remote repository=>Transactionservice2

Create the branch => develop

Now developer,

Clone the repository:

Go to delelop branch

$ git checkout -b feature/transac

$ touch f1

$ git add .

$ git commit -m “f1 added”

$ touch f2

$ git add .

$ git commit -m “f2 added”

// now the push the branch it will try to create a branch in remote repo

$ git push feature/transac origin

// now you will get error saying that no branch in remote.

$ git remote -v

// switch to main branch

$ git checkout main

$ git push feature/transac origin

// now also giving error

$ git push origin feature/transac

// now the branch pushed to remote

/ / once push the branch

// create a pull request and from feature to develop branch .

// add reviewers and merge it with develop branch

// There are 3 options are there

// 1. Crate a merge commit

// 2. Squash and merge

// 3. Rebase and merge

// pull request can be

// we work almost 45 services, the same services we work for 2-3 years

// devops role here is:

// clone the repository,

$ git checkout develop

$ git checkout -b devops/jenkins

// create some files and commit

// now push

$ git push origin devops/Jenkins

// raise pull request, and merge it and delete the devops/Jenkins branch.

/ /within 30 mins the code will be build and deploy into testing server, up to deploying into testing server is automation.

// when you are doing form github to git

$ git remote -v // added automatically

//when you are doing from git to github, you have to add remote repository explicitly

Master> $ git branch -M main

$ git remote -v

$ git remote add origin git@github------

$ git remote -v

$ git push -u origin main

// like above we push branches to remote from local.

//Now you can add commit push the files to remote.

$ git push -u origin main

-u => user

$ git add file1

$ git commit -m “firl1 added”

$ git push -u origin main

// IS IT POSSIBLE TO MOVE CONTENTS FROM ONE REPOSITOY ANOTHER REPOSITORY

// It’s simple

$ git remote add origin1 repository-url

$ git remote -v

$ git push -u origin1

// now go and see the github

// this is how you can work with multiple repositories.

// form one repository two projects can also possible

$ git remote remove test

$ git remote remove origin1

$ git remote -v

// As a devops engineer one time I got a chance to rename the repository,

// but for some reasons the rename is disabled.

// then I cloned and deleted the old project.

=============

$ git add . && git commit -m “commited file a”

$ touch b

$ git add . && git commit -m “commiter b file”

$ git checkout -b test

=================

// Understand How merge working:

$ git log –oneline

Add files between main and branches and files separately

On different time stamps:

When merge happens from

$ git log – oneline

// which gives order based on time stamp. Merge works based on timestamp.

// while doing pull request we have seen 3 options, merge, rebase and merge, and squash and merge.

// rebase means

$ git rebase master

$ git checkout master

$ git merge win master

$ git log - - oneline

// the rebase is a merge without following timestamp.

$ git checkout HEAD

$ git checkout HEAD

========================================

**Bit bucket:**