- Git is **decentralized or distributed (DVCS)** system but can be used as centralized
- The **Repository** : Collection of files managed by git and all version (history)
- Folder where we initialized git or content files in our storage known as **Working Directory or workspace**
- **Commits :** snapshot of file, create a timeline of changes in **branch,** git repo have atleast one branch known as **master branch**
- **GitHub:** git hosting/cloud service for unlimited public repo with no private repo but in business model, we can choose to have private repo also

Install GIT 2:
rpm -ivh endpoint-repo.1.7.rpm
 # yum install git
git --version

Create GIT repo or initialized git:
mkdir myprojects
cd myprojects
 # git init demo
cd demo
ls -a (there folder .git, that is git repo)

Set identity to git, that recorded in version log
 # git config --global user.name "Vimal Daga"
 # git config --global user.email "vdaga@lwindia.com"

Git States with Local:
Working Directory : all files present in above "demo" folder
Staging Area : file preparing for commit, first store in staging area, then only we can commit
Repository/Commit History : committed with version or save changes to git repo

Git States with Remote:
Remote State : repo shared to centralized system like GITHUB

cat > test.txt
First line

 # git status

Put file in the staging area, without putting file here, we can't create commit history
 # git add test.txt

Send file to repo/commit area, here we have created snapshot of file means version of file
 # git commit -m "first comment" text.txt

To check all version of file, every commit they provide commit id
 # git log test.txt

See details of log
 # git show test.txt

We can use commit id's to see the diff in 2 version
 # git diff id1 id2

Remove git repo
 # rm -rf .git

Again create git repo in existing project, go to project folder
 # git init

Add all file in staging area
 # git add .

Add all file in commit area
 # git -m "comment" commit .

Create new file, that is not by default tracked by git
touch new.txt
Show tracked file by git
 # git ls-files

Staging and Committing file in one command : Express commit
 # git commit -a -m "comment" file.txt
Note: not all version of git support this

HEAD Markers : Special Markers Like pointers, points to last commit of current branch

How to make file untracked or removed from staging area:
 # git reset HEAD new.txt

Rollback file data to any point in time:
git log file.txt
Get commit id, at what point we want to rollback
 # git reset commitID file.txt
 # git checkout -- file.txt

Get clone of project over the network using ssh:
 # git clone root@remoteip:/myprojects/demo

Get git help of subcommand:
 # git help log
 # git log --oneline --graph --decorate --all

Create alias in git:
 # git config --global alias.hist “log --oneline --graph --decorate --all”
 # git config --global --list
 # git hist
 # git hist test.txt

Rename file in git
 # git mv test.txt testnew.txt
 # git commit -m “comment rename” testnew.txt
 # git commit -m “commit delete” test.txt

Delete file in git
 # git rm file.txt
 # git commit -m “comment delete” file.txt
 # git commit -m “commit delete” file.txt

Managing file deleting or copy from outside working directory:
cp outfile.txt demo/
rm text.txt

To update git for deletion
 # git add -u

For all modification
 # git add -A

 # git commit -m “comment” .

Excluding file to untracked by git:
Let say, we want to ignore all file with png extension

 # cat > .gitignore
***.png**
 # git add .gitignore
 # git commit -m “ignored” .gitignore

 # touch my.png
 # git status

Branching: timeline of commits, by default branch is master, we can create several branch from master branch
Merge: merge other branch to master
Fast forward merge: simplest case, like never branched , commits on destination, can be disabled
Automatic merge: non-conflicting merge detected, preserves both timelines, merge commit on destination
Manual Merge: automatic merge not possible, conflicting merge state, changes saved in merge commit

List of all branches:
 # git branch
Note: “*” here means current branch name

cat > web.txt
First line

git add .
git commit -m comment1 .

Create new branch “dev2” and its pull all file data from master branch
 # git checkout -b dev2
cat >> web.txt
Second line

Commit data in current branch “dev2”
git commit -m comment2 .

Switch to master branch
 # git checkout master

Note: u wont able to see, data commit in “dev2” branch

If you want to pull data from “dev2” branch and merge in master branch
Its will do fast forward merge
 # git merge dev2

Delete branch:
 # git branch -d dev2

Manual Merge - Conflicting merge resolution:

cat > app.txt
Data1
Data2

git add .
git commit -m com1 .

git checkout -b dev2
vim app.txt
Changes data1 to newdata(in first line)

git commit -m com2 .

git checkout master
vim app.txt
Changes data1 to masterdata(in first line)

git commit -m com3 .

 # git merge dev2
Its shows error auto merging conflict

Note: For this we have to do manual merge, for this we can use any merge tool like p4merge, etc

----

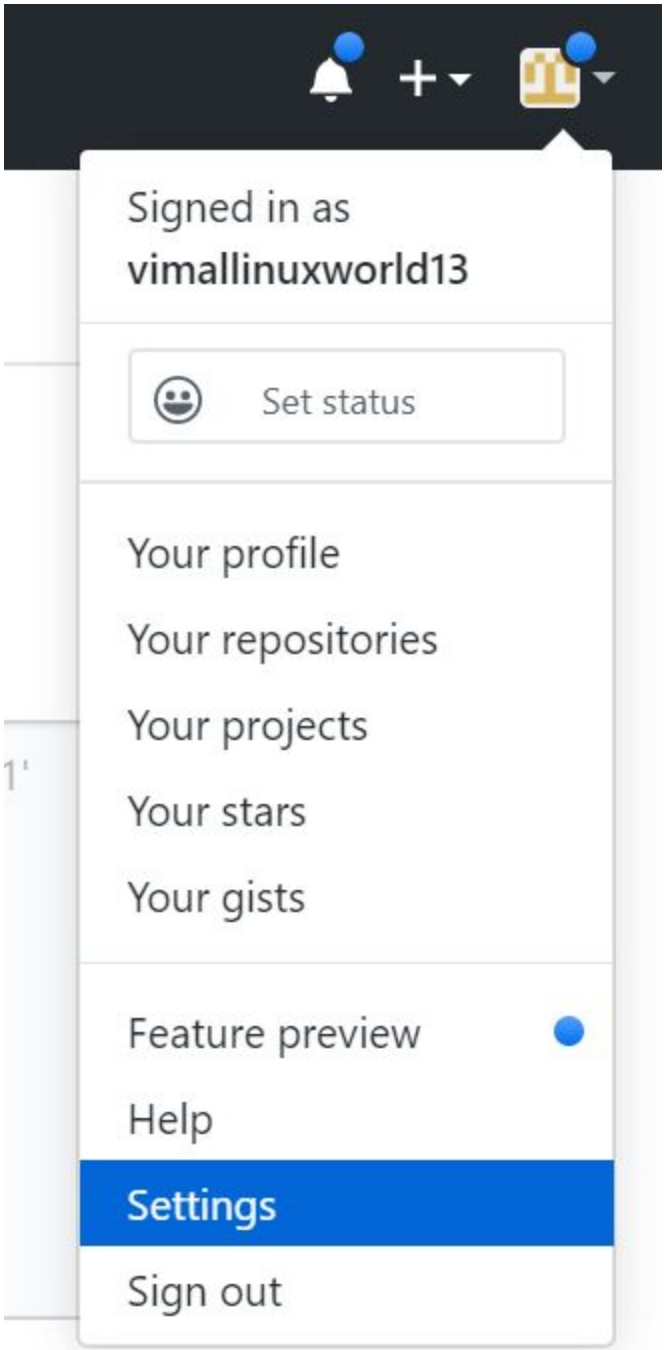
Setup SSH protocol to github for authentication, that give u passwordless authentication:

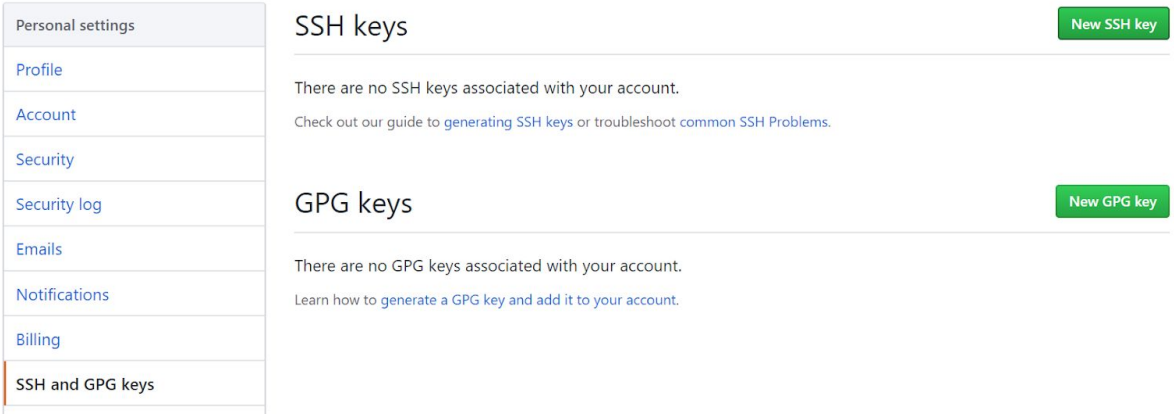
At workstation, where we have local git repo, create private and public ssh key :
 # ssh-keygen

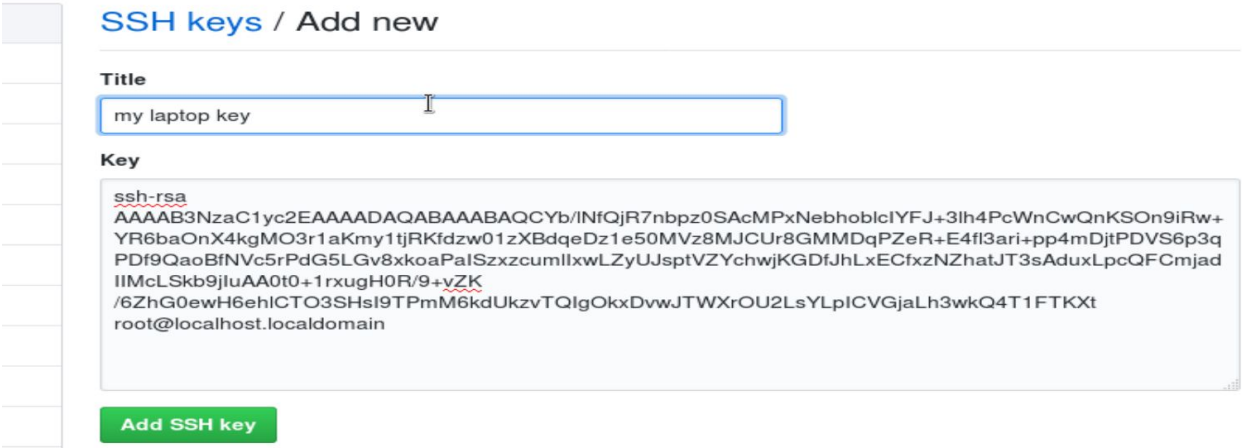
Get public key from here:

```
[root@localhost ~]# cat /root/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCYb/lNfQjR7nbpz0SAcMPxNebhobl
cIYFJ+3lh4PcWnCwQnKSOn9iRw+YR6baOnX4kgMO3r1aKmy1tjRKfdzw01zXBdqeDz1
e50MVz8MJCUr8GMMDqPZeR+E4fl3ari+pp4mDjtPDVS6p3qPDf9QaoBfNVc5rPdG5LG
v8xkoaPaISzxzcumlIxwLZyUJsptVZYchwjKGDfJhLxECfxzNZhatJT3sAduxLpcQFC
mjadIIMcLSkb9jIuAA0t0+1rxugH0R/9+vZK/6ZhG0ewH6ehlCTO3SHsI9TPmM6kdUk
zvTQIgOkxDvwJTWXrOU2LsYLpICVGjaLh3wkQ4T1FTKXt root@localhost.locald
omain
[root@localhost ~]#
```

Keep your private key here at local and send public to github.com, to authorized your local workstation, for this go to github.com : login to account :

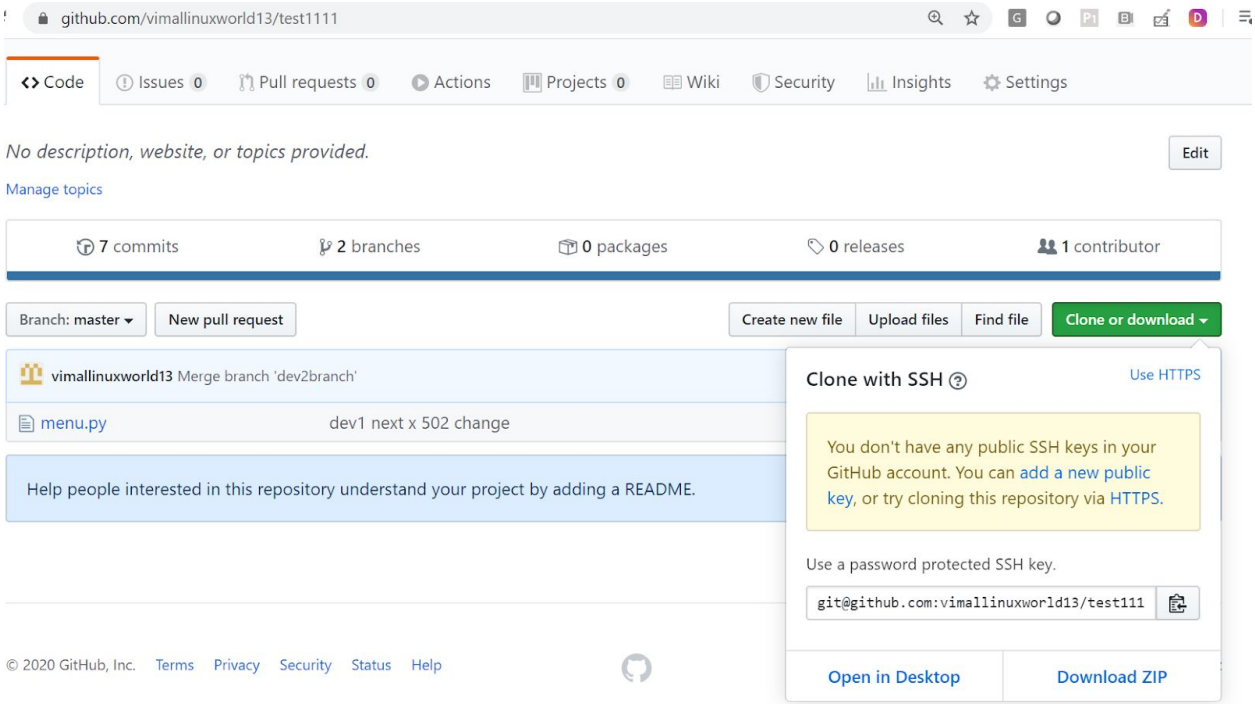

Signed in as
vimallinuxworld13
Set status
Your profile
Your repositories
Your projects
Your stars
Your gists
Feature preview
Help
Settings
Sign out





For testing, go to workstation:
 # ssh -T git@github.com

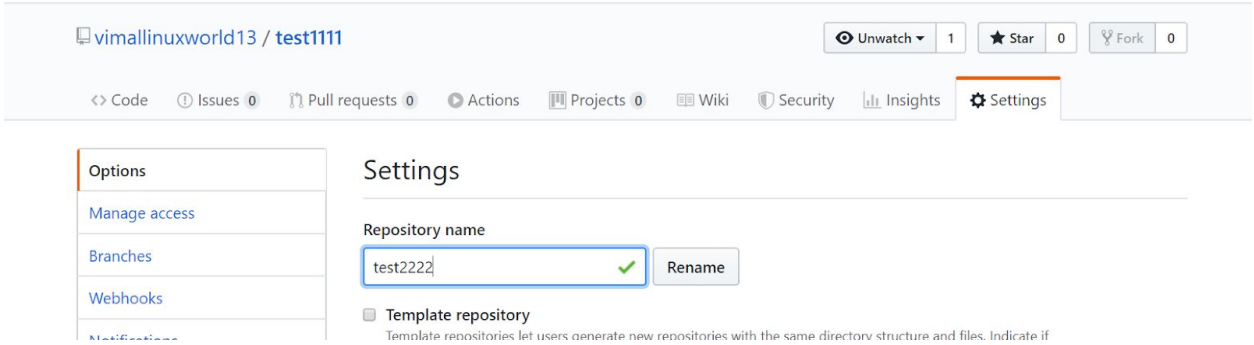
Clone repo from github, to local system using SSH :



We are cloning from github to new dir called mylocalrepodir, it will clone without passphrase :
 # git clone git@github.com:vimallinuxworld13/test1111.git mylocarepodir

---

Change remote github repo name:

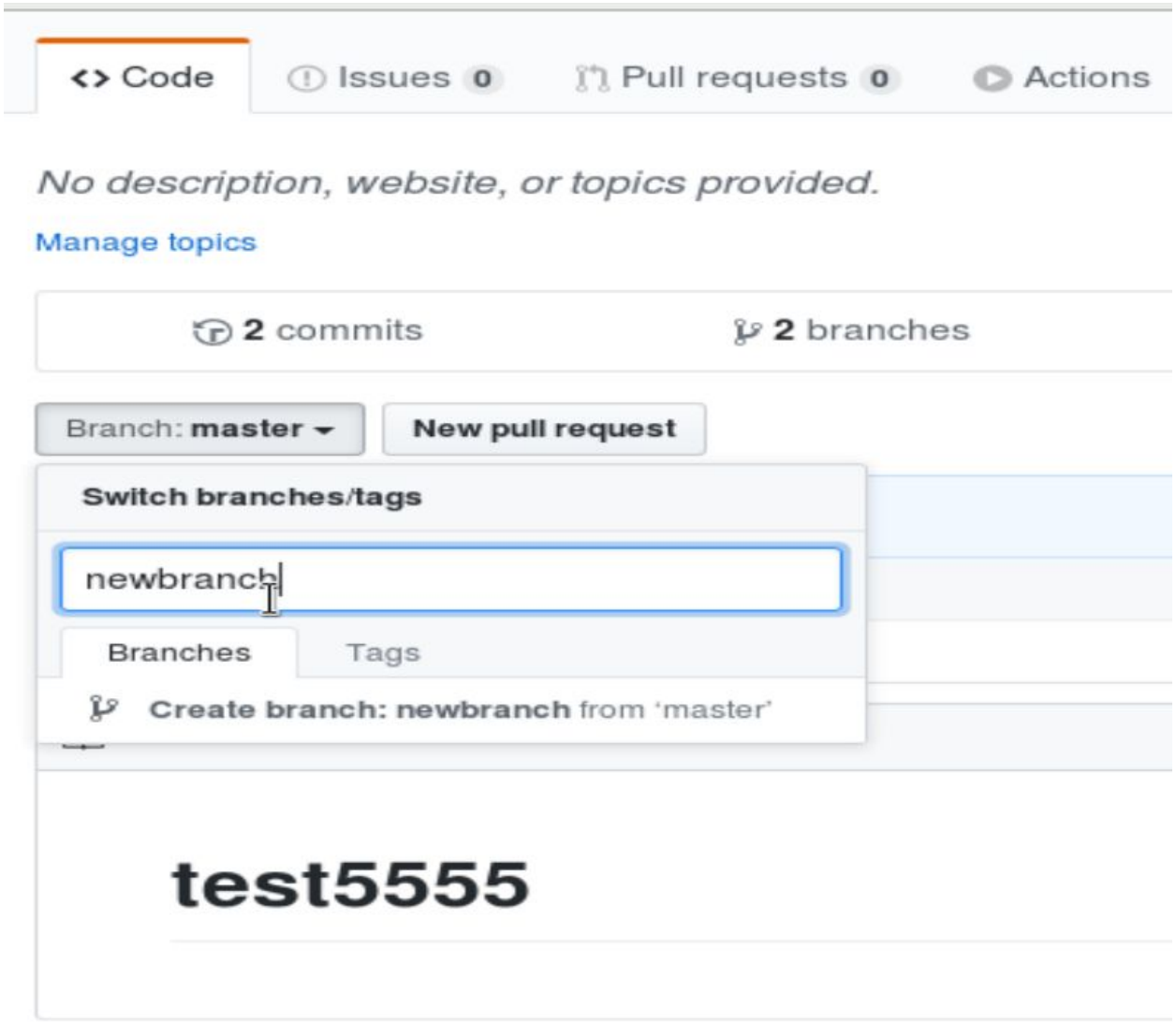


Change remote url in local repo:
 # git remote set-url origin git@github.com:vimallinuxworld13/test2222.git
git remote -v
 # git remote show origin

Delete remote ref :
 # git remote remove origin

Show data from repo at that particular point in time
 # git show commitID (get from git log)

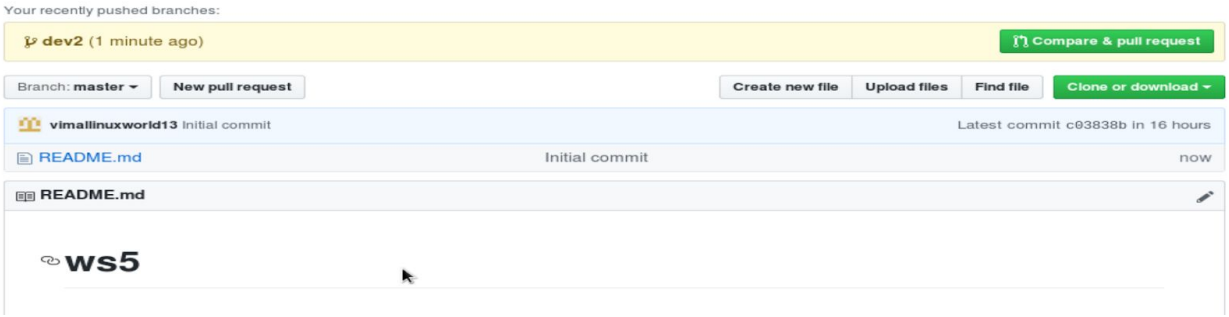
Create new branch from GITHub:



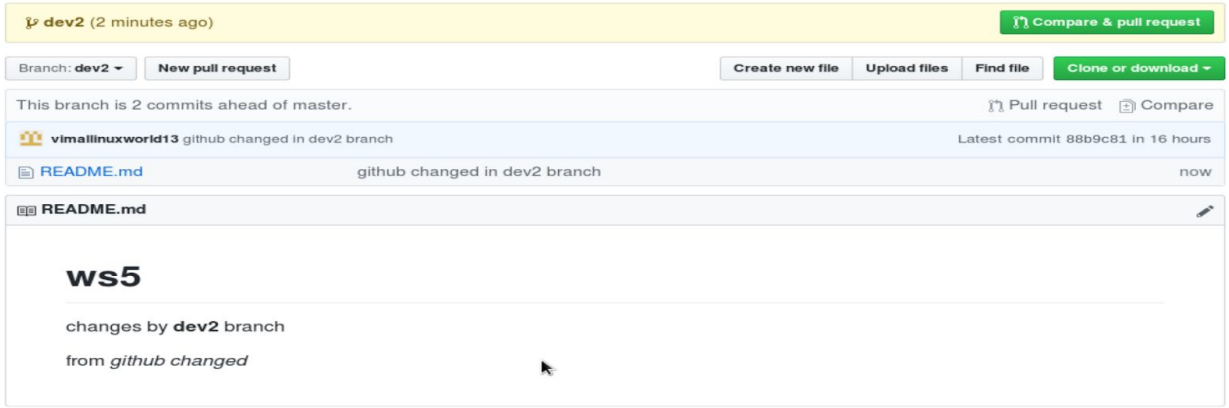
Create local branch and sync to github:
 # git checkout -b testbranch1

It creates new branch in github:
 # git push -u origin testbranch1

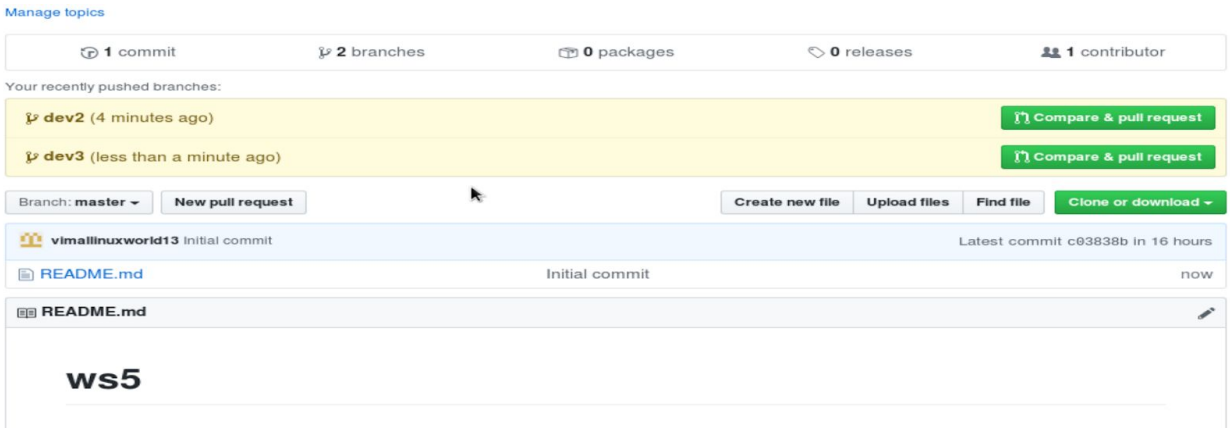
How to do pull request and merge in master branch from other branch in github :
Eg:
This is data in master branch:



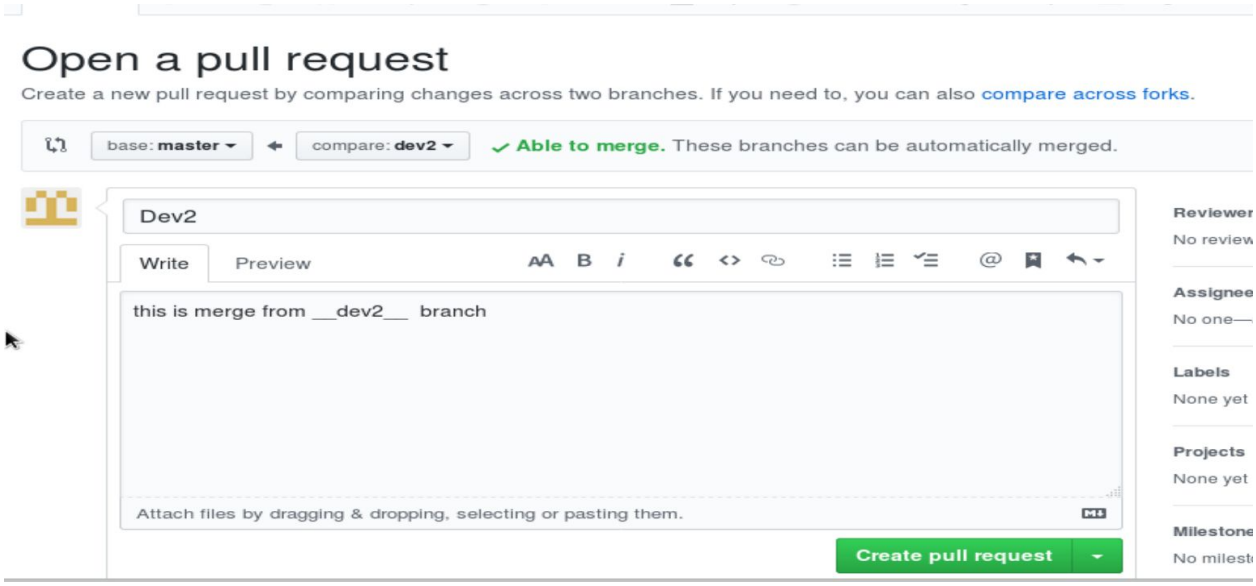
Other branch here, named dev2 changes data:



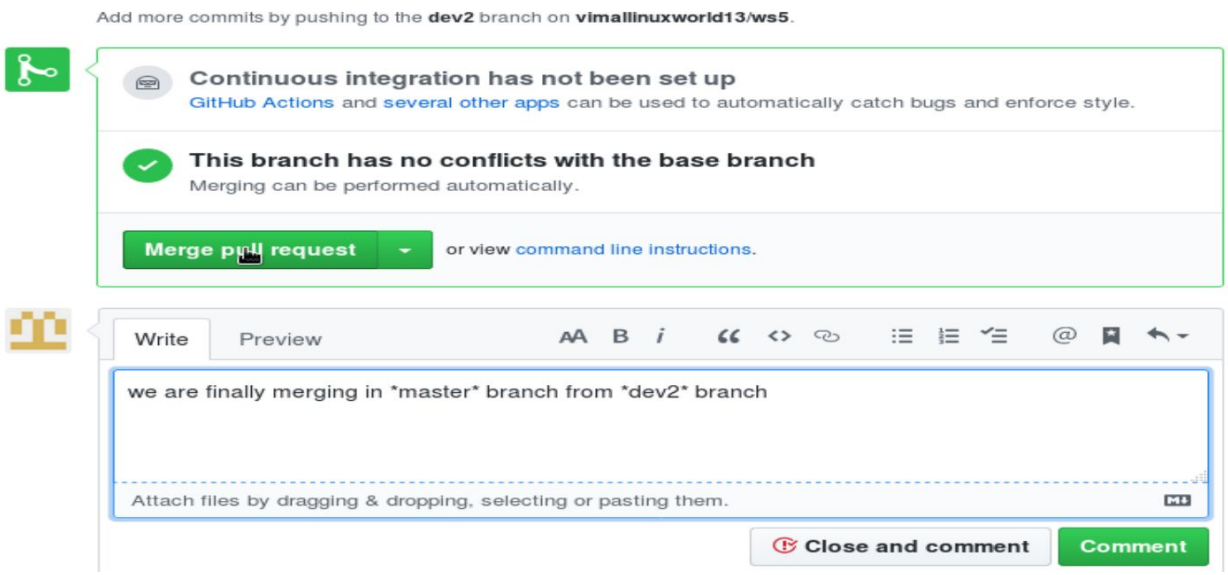
This shows, we have 2 branch ahead of master branch:



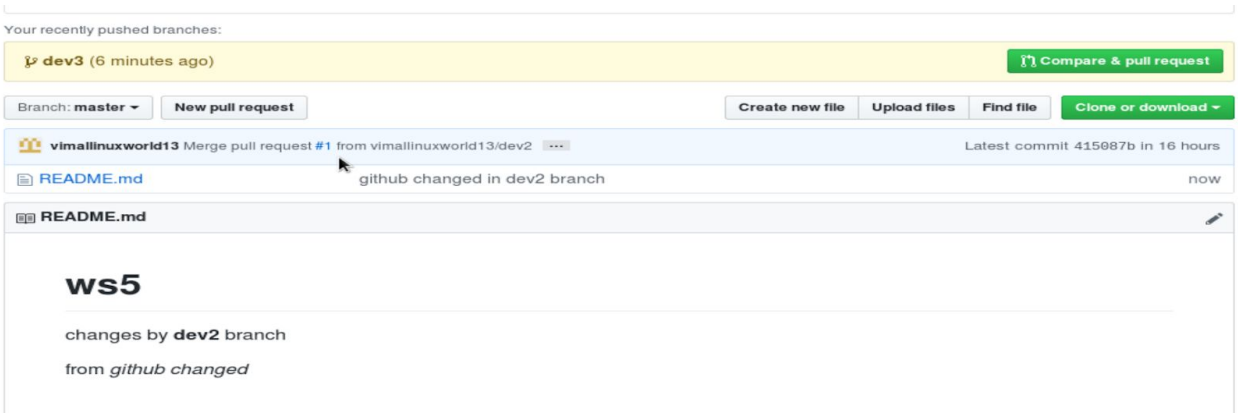
Now you can click on “compare and pull request” button, from where to get the merge in master branch, we are pulling from dev2 in master



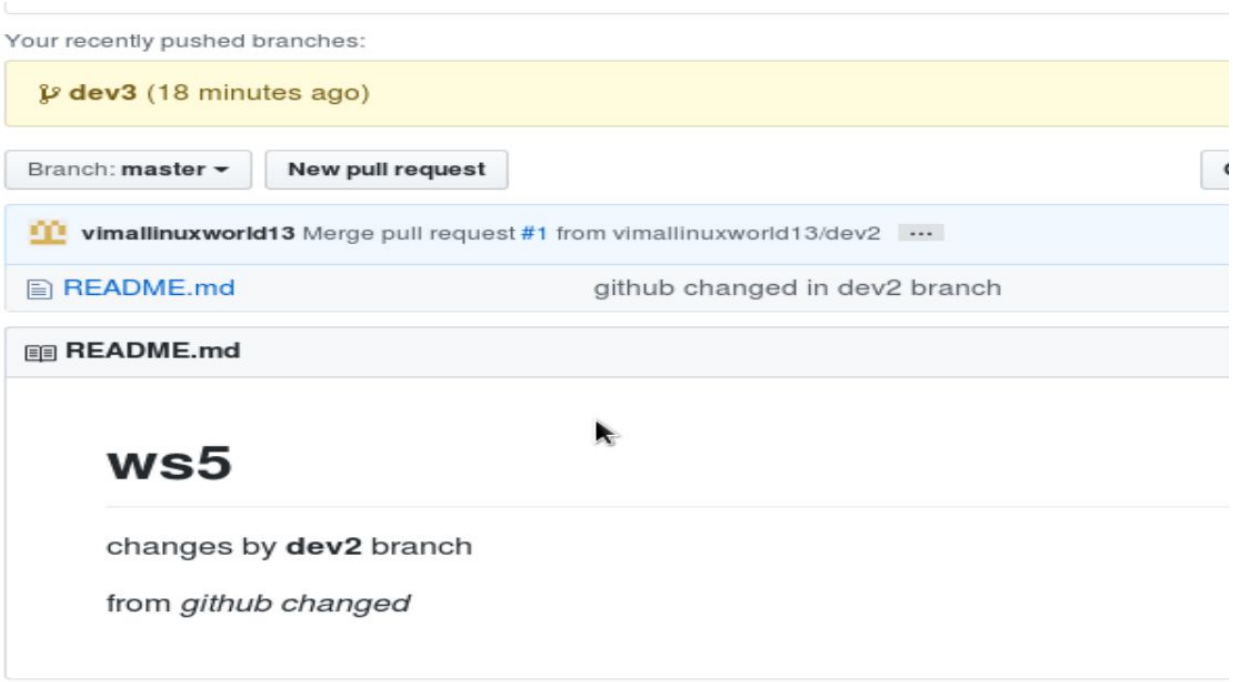
Finally merge from dev2 to master branch:



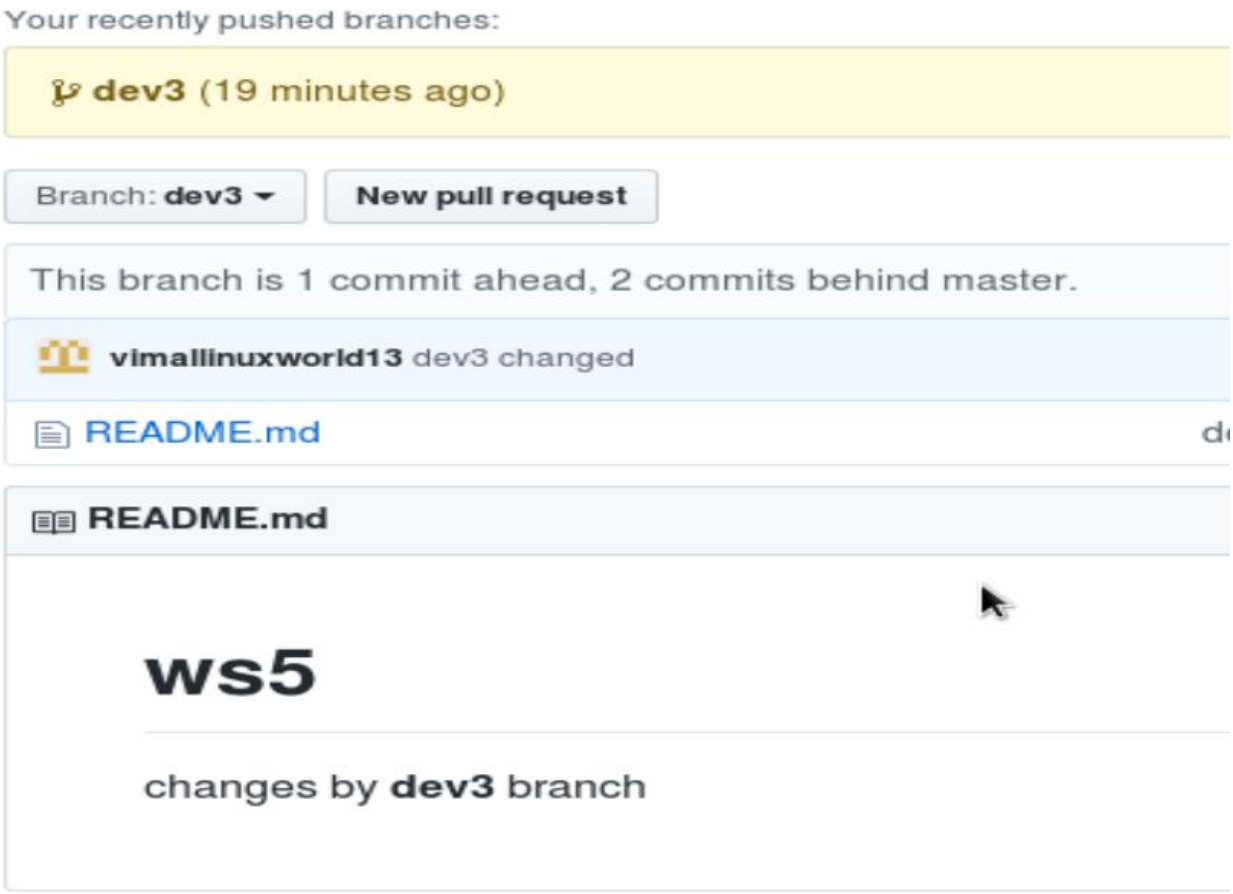
Finally in master branch, we merged from dev2 branch:



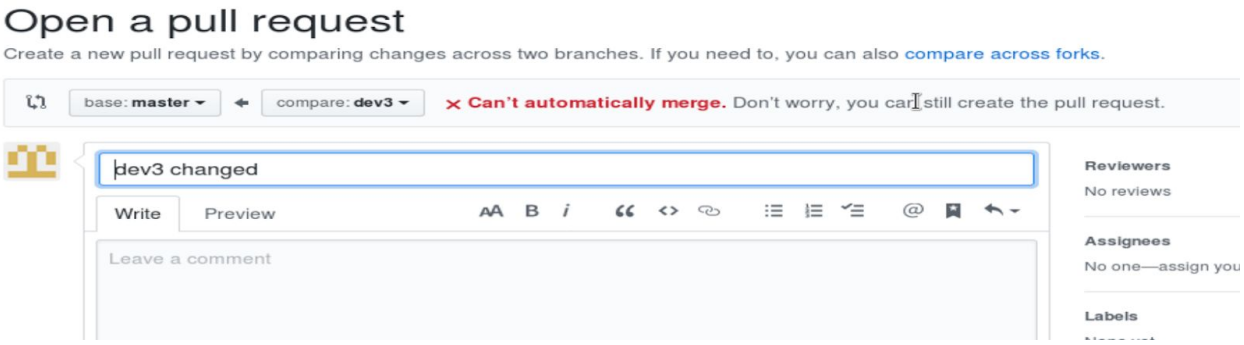
Example of merge conflict:
This is data in master:



This is data in dev3 :

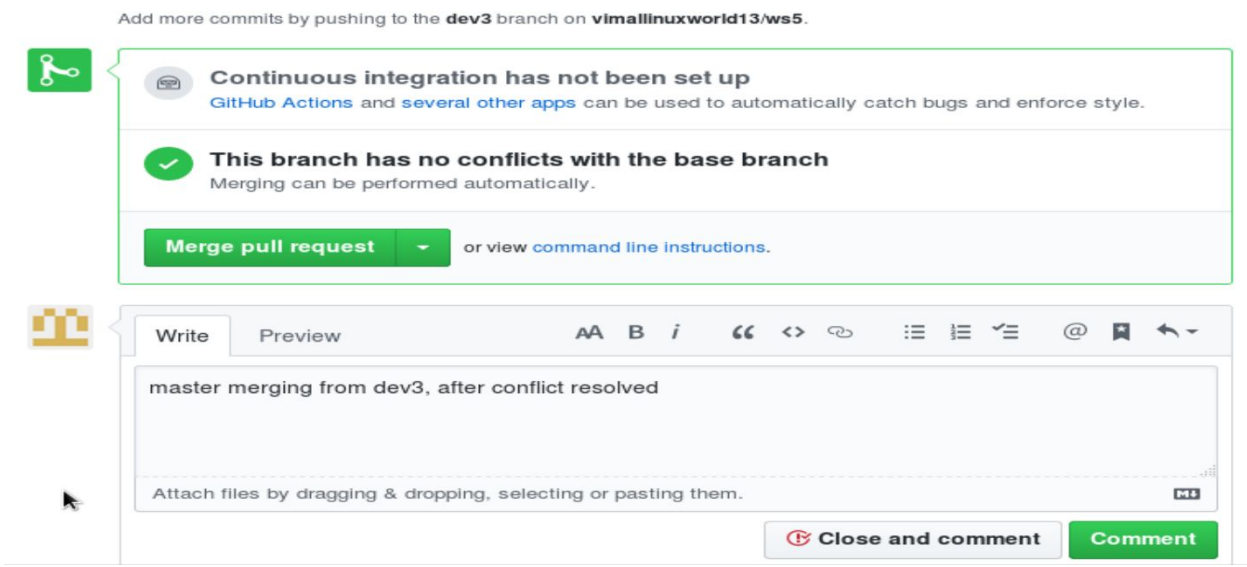


Now if we merge, it wont allow, as conflict :



But we can request pull:

We have to do manual resolve, click resolve conflicts button

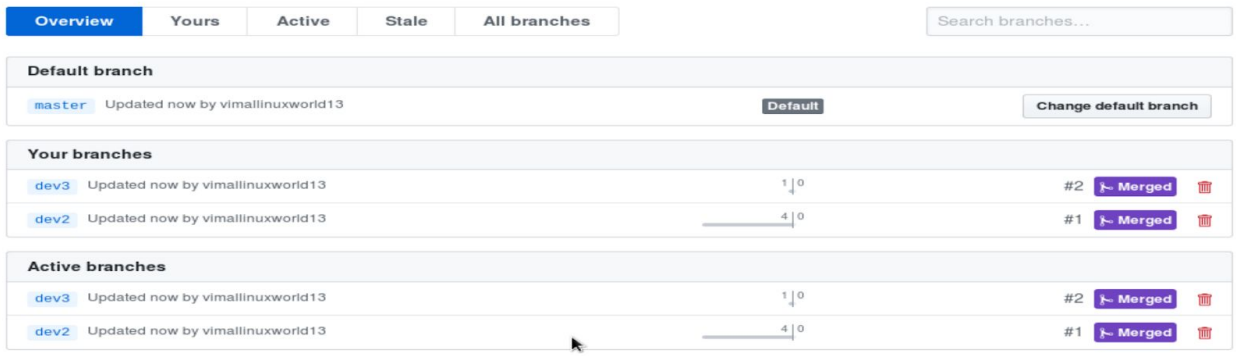


Get local repo synced from github:

```
[root@localhost ws5]# git fetch
remote: Enumerating objects: 14, done.
remote: Counting objects: 100% (14/14), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 8 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (8/8), done.
From github.com:vimallinuxworld13/ws5
   969e549..0b1ffcc  dev3       -> origin/dev3
   c62f7ed..88b9c81  dev2       -> origin/dev2
   c03838b..6185ff0  master     -> origin/master
[root@localhost ws5]# git status
On branch dev3
Your branch is behind 'origin/dev3' by 3 commits, and can be fast-f
orwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean
[root@localhost ws5]#
```

```
[root@localhost ws5]# git pull
Updating 969e549..0b1ffcc
Fast-forward
 README.md | 4 ++++
 1 file changed, 4 insertions(+)
[root@localhost ws5]# cat README.md
# ws5


changes by __dev3__ branch

from *github changed*
```

Remove branches “dev3” from github and update it ref to local:



 # git branch -d dev3

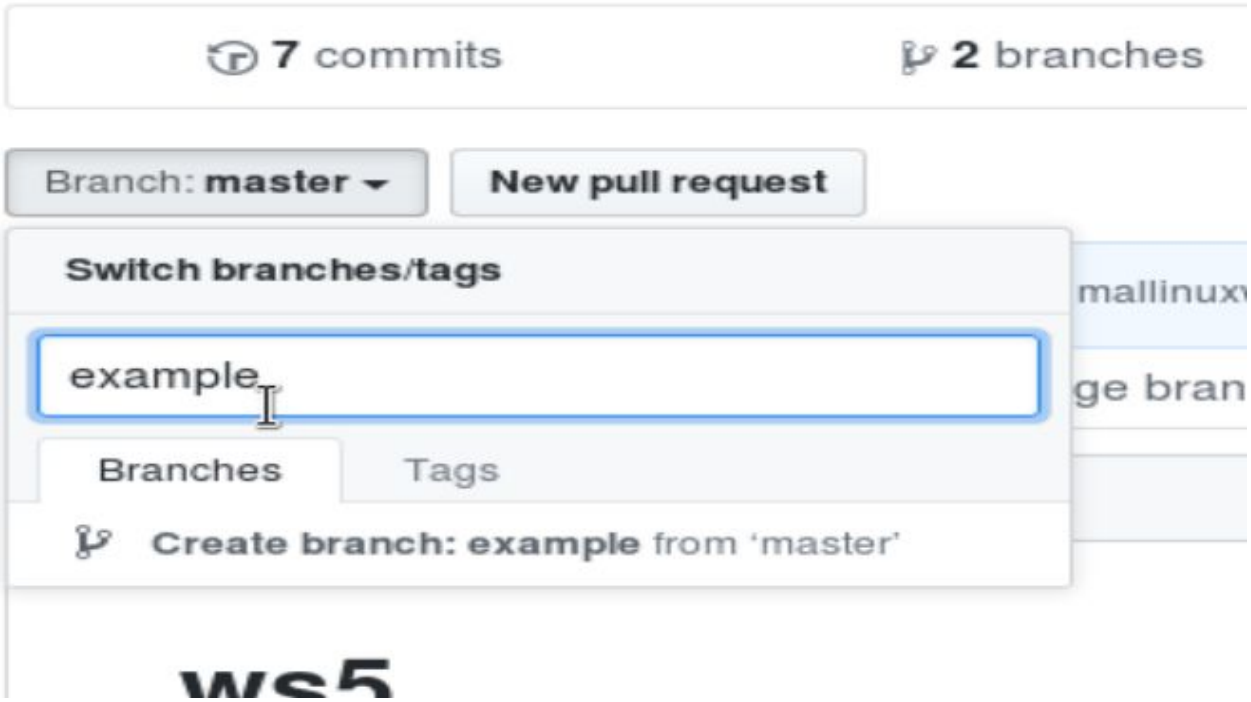
```
[root@localhost ws5]# git branch -a
  dev2
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/dev2
  remotes/origin/dev3
  remotes/origin/master
[root@localhost ws5]#
```

Remove ref of dev3 branch from local to origin (prune branch)
 # git fetch -p

```
[root@localhost ws5]# git fetch -p
From github.com:vimallinuxworld13/ws5
 - [deleted]         (none)     -> origin/dev3
[root@localhost ws5]# git branch -a
  dev2
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/dev2
  remotes/origin/master
[root@localhost ws5]#
```

How to sync branch create in github, synced with local repo:

Create example branch from github:



Changed in example branch:

Branch: **example** ▾ **ws5** / **README.md**

vimallinuxworld13 changed from github example branch

1 contributor

7 lines (3 sloc) | 63 Bytes

ws5

changes by **example** branch

from *github changed*

```
[root@localhost ws5]# git branch -a
  dev2
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/dev2
  remotes/origin/master
[root@localhost ws5]# git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:vimallinuxworld13/ws5
 * [new branch]      example    -> origin/example
[root@localhost ws5]# git branch -a
  dev2
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/dev2
  remotes/origin/example
  remotes/origin/master
[root@localhost ws5]# git checkout example
```

```
  remotes/origin/master
[root@localhost ws5]# git checkout example
Branch 'example' set up to track remote branch 'exampl
in'.
Switched to a new branch 'example'
[root@localhost ws5]# git branch -a
  dev2
* example
  master
  remotes/origin/HEAD -> origin/master
  remotes/origin/dev2
  remotes/origin/example
  remotes/origin/master
[root@localhost ws5]# cat README.md
# ws5


changes by __example__ branch

from *github changed*

[root@localhost ws5]#
```

Here:
 # git checkout example
Automatic create local branch and start tracking it

Sync all repo data from all branch:
 # git pull --all

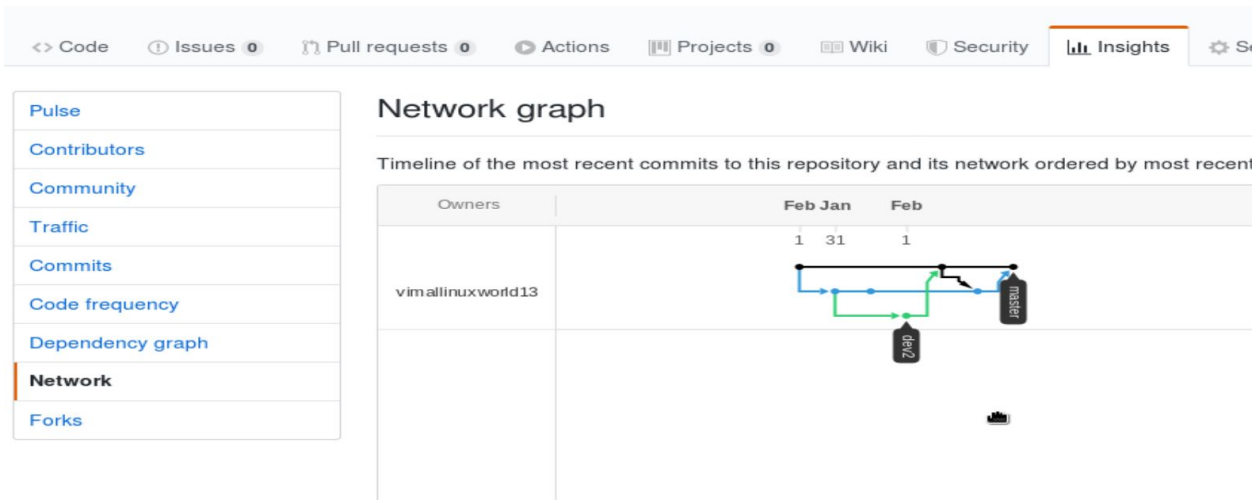
Delete branch from local cli to github:
 # git branch -d example
 # git push origin :example
Note: here ":" means delete example branch

Git graph, show all commit and merge in all branches in graphical form :

 # git log --graph

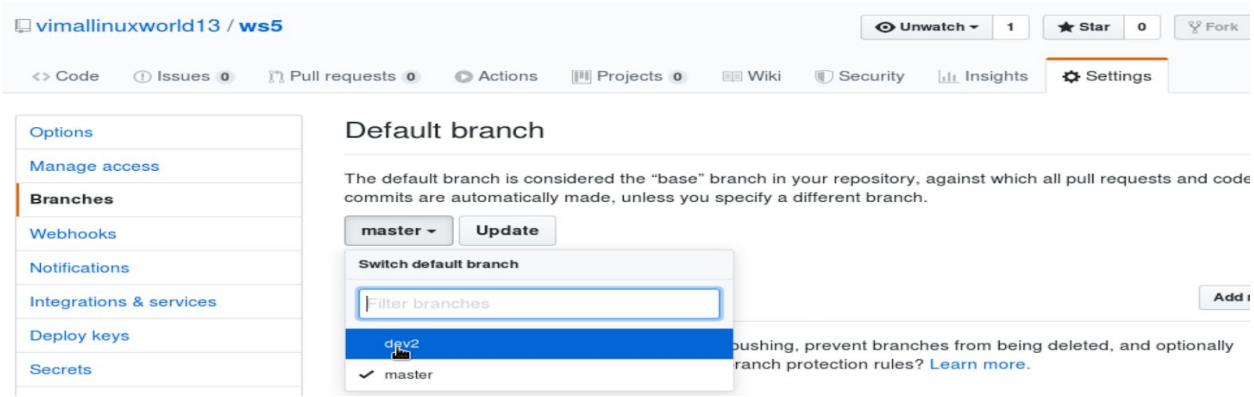
```
[root@localhost ws5]# git log --graph --oneline
* 69fb1fa (HEAD -> example) wdfg
* f3ca012 dfdg
* e6cbb02 changes all
* af23182 changed from github example branch
*   6185ff0 (origin/master, origin/HEAD) Merge pull request #2 fr
 vimallinuxworld13/dev3
|\
| *   0b1ffcc Merge branch 'master' into dev3
| |\
| |/
|/|
* |   415087b Merge pull request #1 from vimallinuxworld13/dev2
|\ \
| * | 88b9c81 (origin/dev2) github changed in dev2 branch
| | * 969e549 dev3 changed
| |/
| * c62f7ed (dev2) dev2 changed
|/
* c03838b (master) Initial commit
[root@localhost ws5]#
```

From github, graph (name changed to insight):



How to change default branch:
Benefit of making other branch default:

- When do git clone, it get from other branch
- When we create other branch, it used base branch from that default branch
- It good for production use case, make production data comes from master branch, and all development work on that new default branch, so accidently nobody changes or merge in master branch



When we have trouble of conflict with pulling :
This scenario, comes when we changed at github, then using "git fetch" at local, and before "git pull" , we changed locally and commit:

```
[root@localhost wsnew]# git status
On branch dev2
Your branch and 'origin/dev2' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

nothing to commit, working tree clean
[root@localhost wsnew]# git push
To https://github.com/vimallinuxworld13/ws5.git
 ! [rejected]        dev2 -> dev2 (non-fast-forward)
error: failed to push some refs to 'https://github.com/vimallinuxwo
rld13/ws5.git'
hint: Updates were rejected because the tip of your current branch
is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for d
etails.
[root@localhost wsnew]# ls
```

```
[root@localhost wsnew]# git pull
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

We have to use mergetool, to merge manually :

```
[root@localhost wsnew]# git mergetool

This message is displayed because 'merge
```

```
[root@localhost wsnew]# git commit
[dev2 704c091] Merge branch 'dev2' of https://github.com/vimallinux
world13/ws5 into dev2
```

```
[root@localhost wsnew]# git push
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 407 bytes | 407.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/vimallinuxworld13/ws5.git
   0a28466..704c091  dev2 -> dev2
[root@localhost wsnew]# git status
On branch dev2
Your branch is up to date with 'origin/dev2'.

nothing to commit, working tree clean
```

Steps to solve “diverged” conflict issue:

 # git pull
manual merge
 # git mergetool
 # git commit -m "comment"
 # git push

Get source branch or base branch(master) update into sec branch(dev1): rewind sub branch

In master branch:
cat >> test.txt
Changes from master

git commit -m “changed from master” .

git checkout dev1

Get dev1 updated data from master branch (something like opposite of merge):

 # git rebase master
cat test.txt (sec branch dev1 get updated data from master branch)

In conflict of rebasing:
git rebase --abort

git mergetool
git add .
git commit

git rebase --continue

If local repo and git hub changes and commit, and we want to update local repo from github, before it push to github:

 # git pull --rebase origin master
 # git push

If we don't want to commit changes in file, as we are in work in progress(WIP), but want to save state of file, and want to work in other file to commit changes :

 # git stash save (or git stash)

git stash list

Go back to the same state, where we saved file, to continue your work from there
 # git stash apply

git stash drop

Stash doesnt stash untracked file by default , if you want to stash all file from staging and working area:
 # git stash -u

And to apply and drop in single command, we can use “pop”:
 # git stash pop

If we want to create multiple stash, having multiple WIP :

```
[root@localhost test1]# git status
On branch master
nothing to commit, working tree clean
[root@localhost test1]# vim h.txt
[root@localhost test1]# git stash save "first"
Saved working directory and index state On master: first
[root@localhost test1]# vim new.txt
[root@localhost test1]# git stash save "sec"
Saved working directory and index state On master: sec
[root@localhost test1]# git stash list
stash@{0}: On master: sec
stash@{1}: On master: first
[root@localhost test1]# git shash show stash@{1}
git: 'shash' is not a git command. See 'git --help'.

The most similar command is
        stash
[root@localhost test1]# git stash show stash@{1}
 h.txt | 1 +
 1 file changed, 1 insertion(+)
[root@localhost test1]#
```

```
[root@localhost test1]# cat new.txt
[root@localhost test1]# git stash apply stash@{1}
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working direct
ory)
        modified:   h.txt

no changes added to commit (use "git add" and/or "git commit -a")
[root@localhost test1]# cat h.txt
fri
sec  hi  from master`
stash
ew
last
[root@localhost test1]# git stash drop stash@{1}
Dropped stash@{1} (be6598540866ee71be50a765f6a1e333f30fd28d)
[root@localhost test1]#
```

Clear all stash, also remove all WIP data:
 # git stash clear

If you changed or create something in master branch, then release it might create issue in code, then we can take all these changes to new branch, and make master branch clean:

```
[root@localhost test1]# vim h.txt
[root@localhost test1]# touch n.txt
[root@localhost test1]# git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working direc
ory)
        modified:   h.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        n.txt

no changes added to commit (use "git add" and/or "git commit -a")
[root@localhost test1]# git stash -u
Saved working directory and index state WIP on master: 7214bf2 f
[root@localhost test1]# git status
On branch master
nothing to commit, working tree clean
[root@localhost test1]#
```

```
[root@localhost test1]# git stash branch newbranch
Switched to a new branch 'newbranch'
On branch newbranch
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working direct
ory)
        modified:   h.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        n.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (ce02008f87943b3dd08825377aaf61d137bfef19)
[root@localhost test1]# git stash list
[root@localhost test1]# git branch
  b1
  master
* newbranch
[root@localhost test1]#
```

How to amend previously committed ID, new data or files:
Always good practise to do in new branch, as while doing amend, it changes commit ID:
git checkout -b newb
 # git commit --amend -m “new changes”

Its shows only reachable history:
 # git log --oneline

But if we want to see all history, even that is unreachable because of amend:
 # git reflog

To get and start working again in orphaned branch:
git checkout commitID
git checkout -b newbranch

Go to history at particular time:
 # git reflog HEAD@{2.days.ago}

 # git reflog master@{2020-01-13}

 # git diff HEAD@{2} HEAD@{3}

git reflog expire --expire-unreachable=now --all
git gc --prune=now

Git GUI client :
---
https://www.gitkraken.com/download