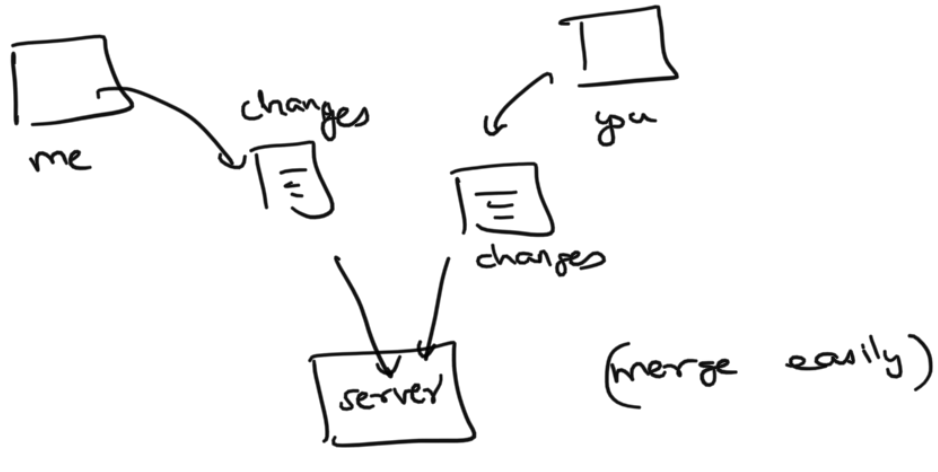
Lecture: Git

Version control (Linus Torvalds, 2005 - current maintainer is Junio Hamano)

Git → keeps track of changes to code

→ keeps collaborators in sync



(merge easily)

→ Tests new changes to code and then only merge new changes

→ allows you to go back to older versions of your code

GitHub → website that stores Git repositories

① Create a GitHub repository

Go to GitHub

repo now exists online
not yet on your computer

② git clone

git clone <url>

remote version downloaded to your computer

③ touch <filename>

creates a file with <filename>

Text editors: atom, Sublime, VS Code

④ add content to your file <filename>

⑤ git add

git add <filename>

tells git that next time I commit, to add this file next time I save the changes to the repository

⑥ git commit

git commit -m "message"

takes snapshot of repository and saves the new snapshot

git commit -am "message" [adds and commits]

(7) git status

gets current status

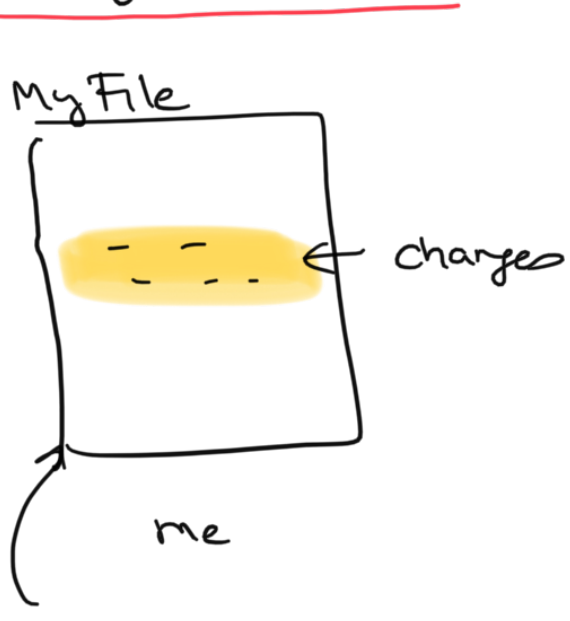
(8) git push

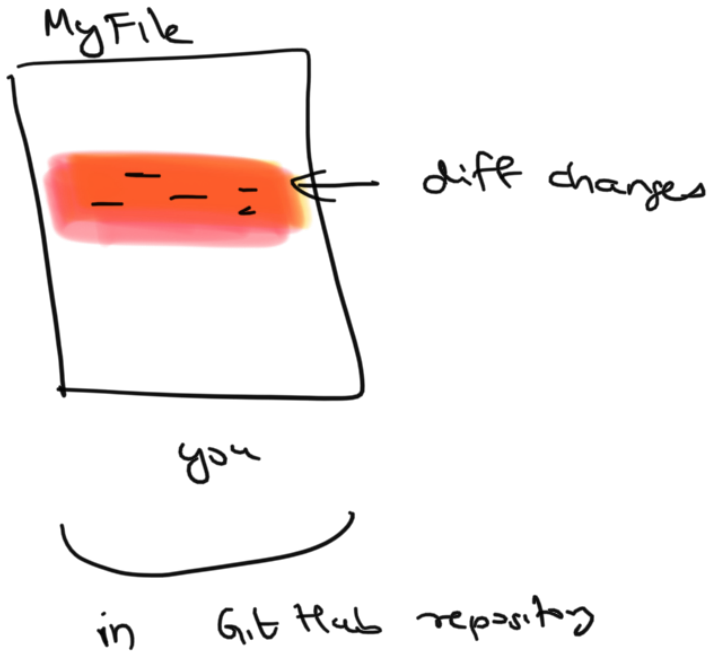
push your commit to GitHub / remote server

(A) git pull

download latest changes from GitHub / remote server to your computer

(B) Merge conflicts

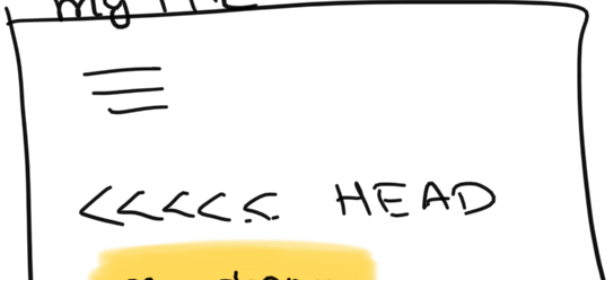




have commited my changes but not yet pushed them

me: git pull

git: merge conflicts !



my changes

=======

git hub change

>>>>>> conflicting commit ← my commit which caused conflict

(a long string of alphanumeric chars)

→ Remove <<<<<< HEAD

=======

>>>> .....

and remove which changes need to be deleted

and then add, commit, push

③ git log

A list of all commits made so far in that repository.

commit <Commit hash>

Author : me

Date :

what changes happened in the commit

} one list entry looks like

④ git reset

git reset --hard <commit hash>

resets git repository to the snapshot taken by the commit with <commit hash>

(Guess this is just on local machine)

" HEAD is now at <commit hash>
" commit message "

git reset --hard origin/master

resets to version on Git Hub ?
(is it the current version or the first version on GitHub from which I cloned ?)

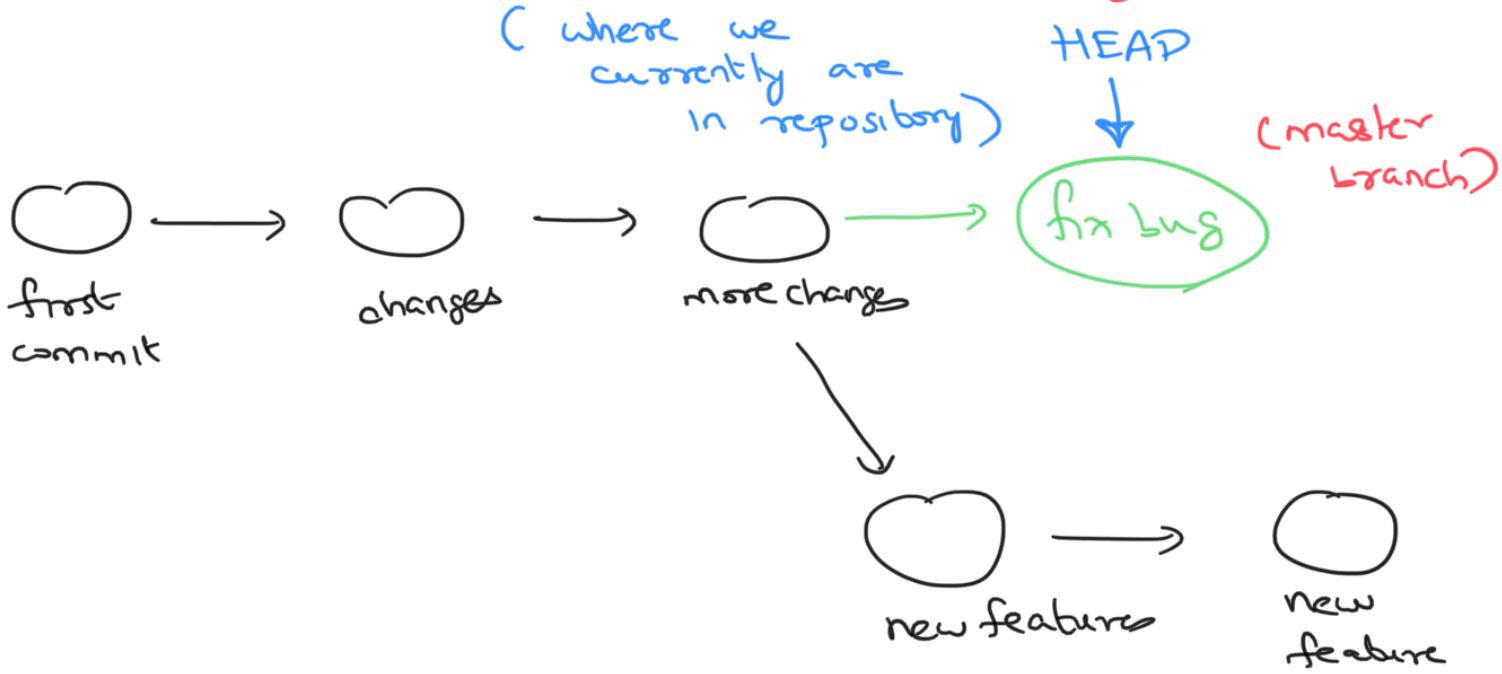
(E) git reflog

(list of all references to logs)

Branching

main branch " Master " (original code always usable)

other branches (work on new features as an aside and later work them into original code)



→ can move "HEAD" to (feature branch)

feature branch to check out feature!

→ merge branches at a later point...

git branch

– listing of all branches currently in my repository

written in green with *

[* currentbranch - I am on
other branches:]

git branch name-of-branch

– creates a new branch called name-of-branch

git checkout name-of-branch

– moves from current place to name-of-branch

to combine these two

git checkout -b new-branch

– creates a new branch called new-branch and moves there

while on master branch,

say you want to merge

feature branch with master branch.

git merge feature_branch

To push contents of branch to GitHub

git checkout -b new-branch

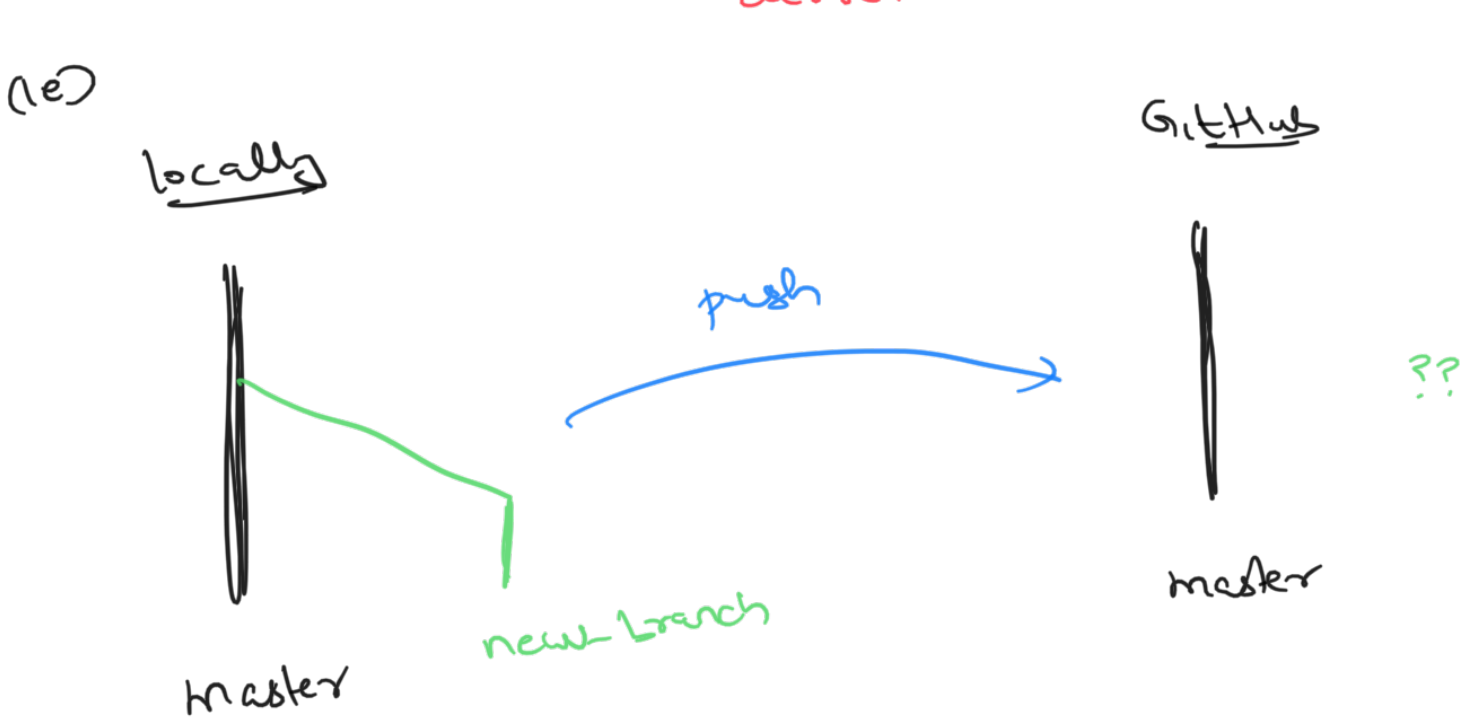
say you modify a file

git commit -am "file changed on new branch"

git push

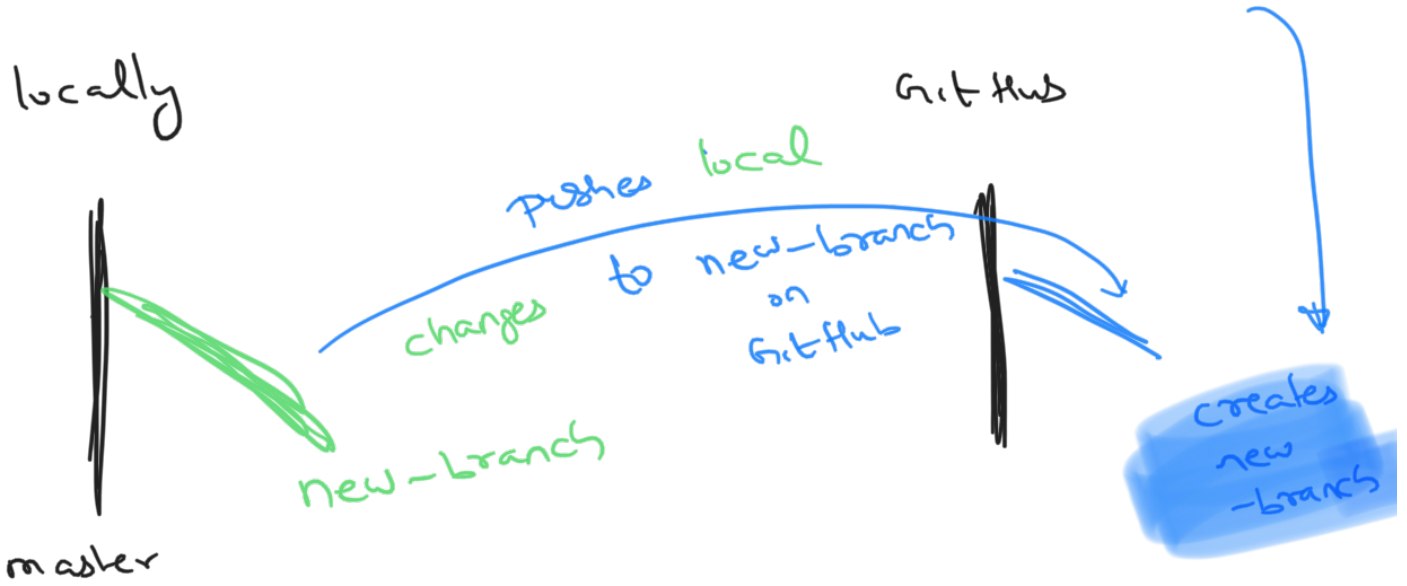
Error! "No upstream branch for current branch new-branch."

(ie)



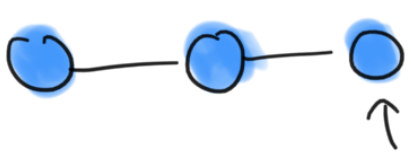
no branch in GitHub for new-branch to go to.

git push -- set-upstream origin new-branch

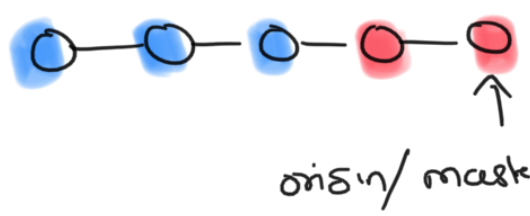


git pull

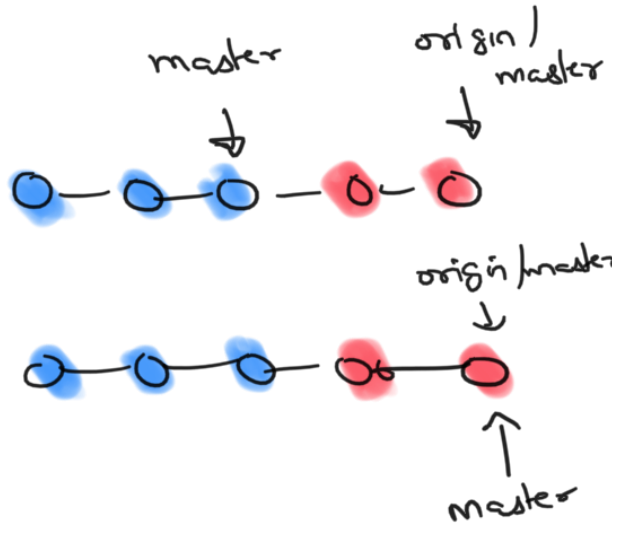
local



remote



git pull = { git fetch ; git merge origin/master }



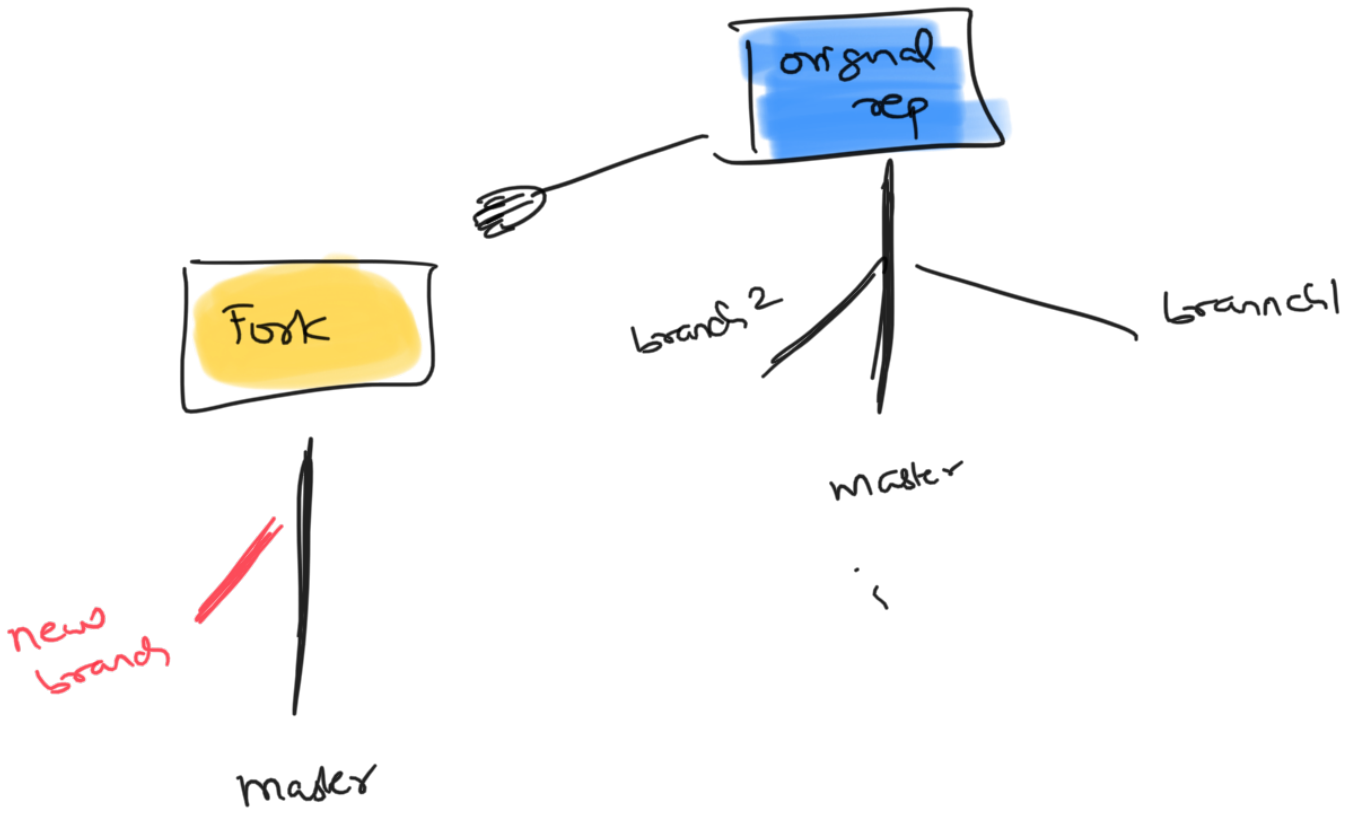
Forks

→ entirely separate version of original rep that gets copied

→ fork belongs to you.

→ changes to fork won't affect original.

Submit pull request



pull request :→ " please commit new_branch changes to original rep"

→ ask for a review of your code to rep_owner.

On Github, in your branch

compare and pull request ...

[unclear how to pull request your files to a different rep... ??]