Feb. 03rd 2015
Feb. 04th 2015.

Git enable tab completion → git-completion.bash

→ git-prompt

Configuration file. → .bash-profile

Staging Area : Key idea : one commit per logical change.

A

B

$

C

#

%

E

Lesson 1 — commit history

✓ git log [- - stat]

✓ git diff

✓ git checkout

git ~~commit~~ clone (create from remote)

git checkout master

↳ back to head

↳ escape from detached head state.

git diff → compare working dir & staging area.

git diff -- staged

→ compare staging area with most recent commit in repo.

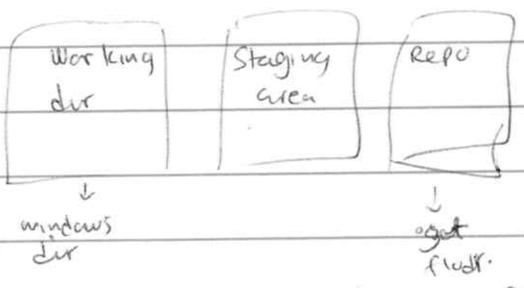
git diff commit1 commit2

Lesson 2

git init (~~stops~~ create ~~remotely~~ locally)

git status → what was changed since last commit

git add (to add staging)



git reset (remove changes in a file ~~from~~ from staging) . ~~fro~~

git reset -- hard

↳ remove changes in all files from staging area & working directory.

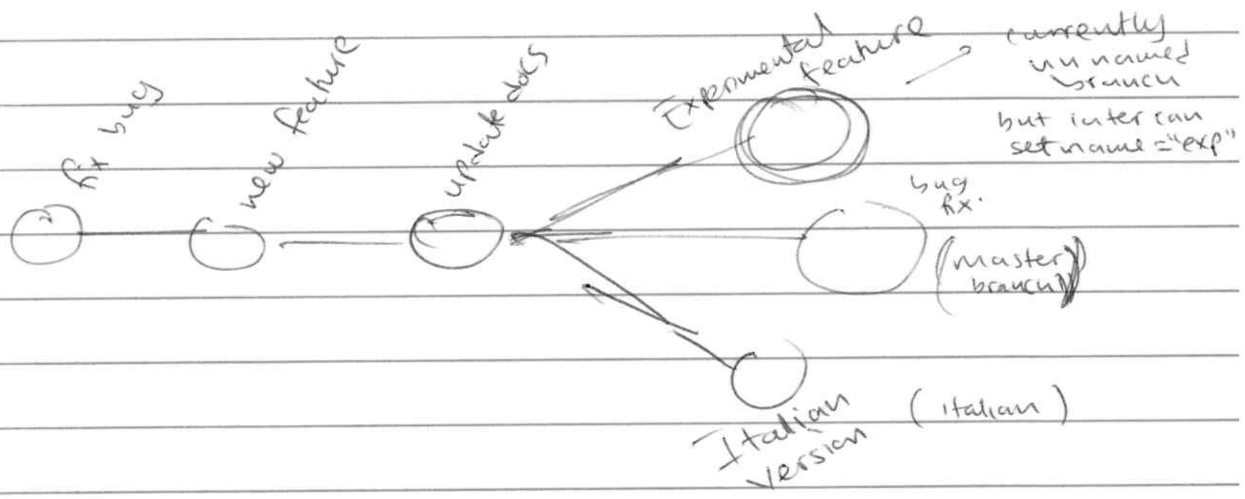
git branch

git branch <name>

Lesson 3

Feb 04th 2015

Branches



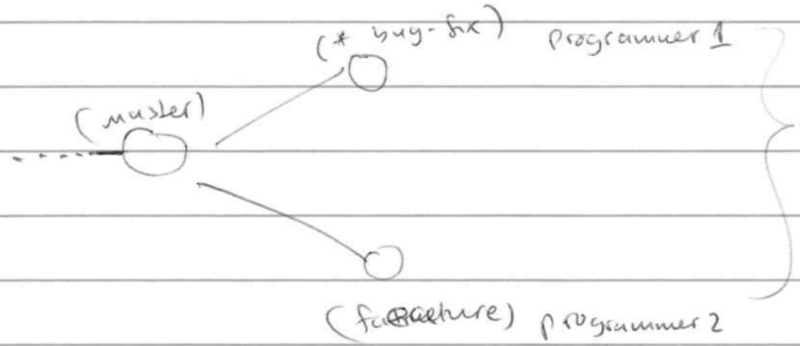
- Linear Commit history : fix bug new feature update docs.
- Branches : Test experimental features.
- Master Branch → git creates for you when you create a new repo.
- Detached HEAD state : You are looking @ a commit on an unnamed branch
- Can check out a "branch" just like you checked out a "commit". ↳ basically pick what branch you are working on.
- If you check out a branch, then commit, the branch lable updates to the new commit
- Tip → most recent commit on a branch.
- Can have multiple branch lables attached to a single commit

git checkout -b newbranchname

① create a new branch
→ git branch branchname
② checkout (or switch) to new branch
→ ② git checkout branchname

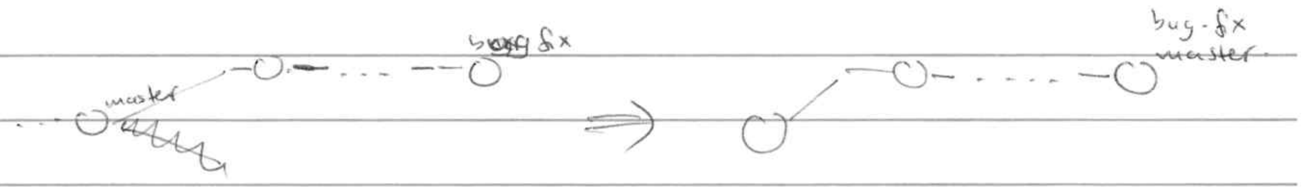
- git ~~branch~~ branch → show me all branches
- git branch <name> → create a new branch with name = <name>;
- * <name> → indecate which branch is checked out.
- git log --graph --oneline master coins
  options — branches I want to see.

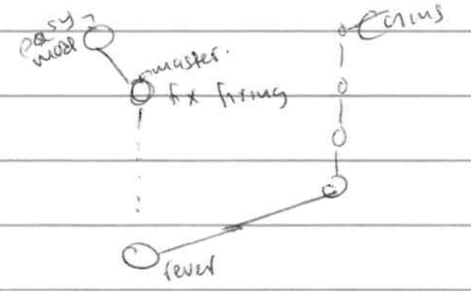
Branch for collaberation / or bug fix.



Good for collaberation.

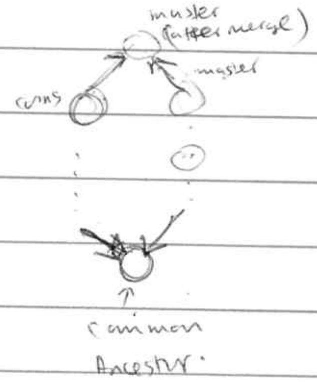
- Idea : you can update master, to point to tip of new branch (if no ~~other~~ parrallel changes had been made to commit "master" later points to).
  i.e





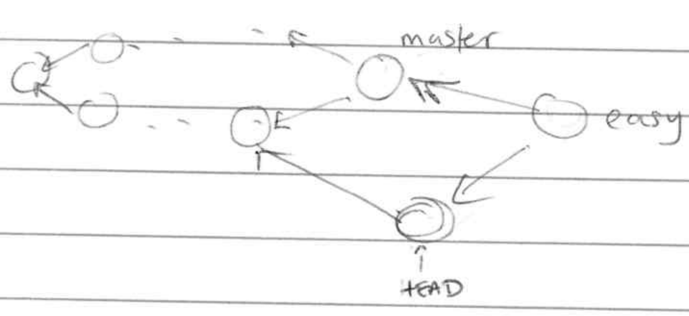
- HEAD → points to the currently checked out commit.

- git merge : • merges of the specified branches into the currently checked out ~~b~~branch.
  - ~~cr~~creates a <u>new</u> commit
  - updates checked out branch lable to pt to this commit
- git merge --abort → restore files to their state before starting a merge.
- git show <<commit id>> : compares a commit to its parent.



- git config --global → Edit git configuration
- git branch -d <<branch name>> → removes label for branch (commits are not deleted).

So now you may or may not be able to reach these commits.



~~H~~HEAD → my code easy.

{ what both modified }

master → from master.

Feb 05th 2015

Lesson 3.

Github : Platform ~~Platform~~ for sharing your repo. → just host for repo (no working directory or staging area)

Popular Github projects. public repos examples:
- Python, Bootstrap, mathquill, jquery atom ↳ text editor.

remote : url for remote repository
can push / pull to/from □ $\xrightarrow{push}$ github repo $\xleftarrow{pull}$

- 1st need to create a repository on github.
  name. ~~reflections~~ Reflections
  - public
  - □ → Do NOT initialize with README
  ~~[illegible]~~
- 2nd create a remote ~~locally~~ in local repo.
  Just like you used git branch ~~to~~ → view and create branch

use git remote to view and create remotes.

- * git remote → to view
- * git remote add origin git@github.com:(... /repo.git
  origin ↓ name; git@github.com:(... /repo.git ↓ copy. url from github.
- * git remote -v → verbose. option to view
- * git push origin master → Push a local branch to a remote (repo
  origin ↳ where to push branch.
  master ↓ copy all commits reachable from master.

- Can add/update file to repo directy from Github web interface.

- Fork → Clone a ~~prepo~~ repo in one account on Github directly into another account on github.

  ↳ github keep count of how many times your repo is forked
  ↳ The forked clones ~~have~~ maintain a reference to original repo (makes it easier to sugest changes from forked clone back to original repository).

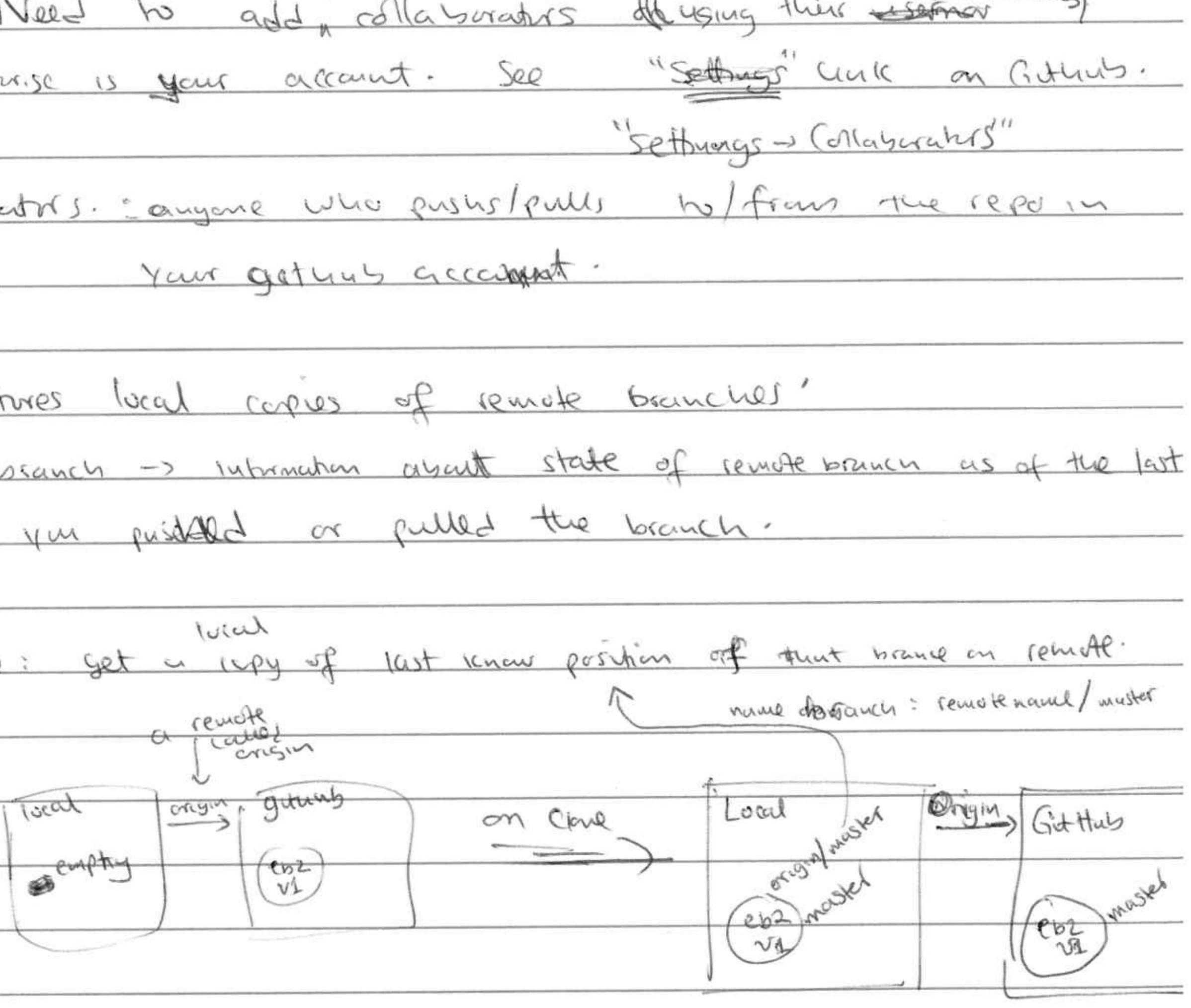
- ~~Can~~ Need to add your collaborators (using their user names) to repositories in your account. See "Settings" link on Github.
  "Settings → Collaborators"

- Collaborators : anyone who pushs/pulls to/from the repo in your github account.

- Git stores local copies of remote branches'
  remote branch → information about state of remote branch as of the last time you pushed or pulled the branch.

on push
on pull
on Clone : get a local copy of last know position of that branch on remote.

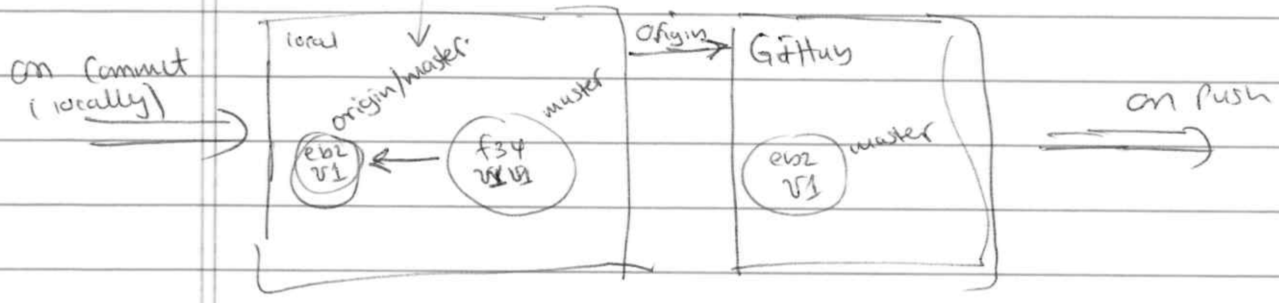
ie.

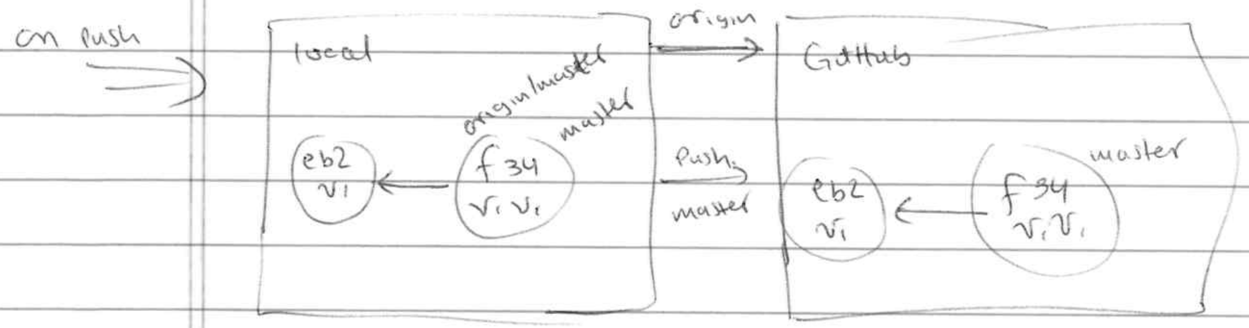


Remember can have multiple remotes setup on a local repo.

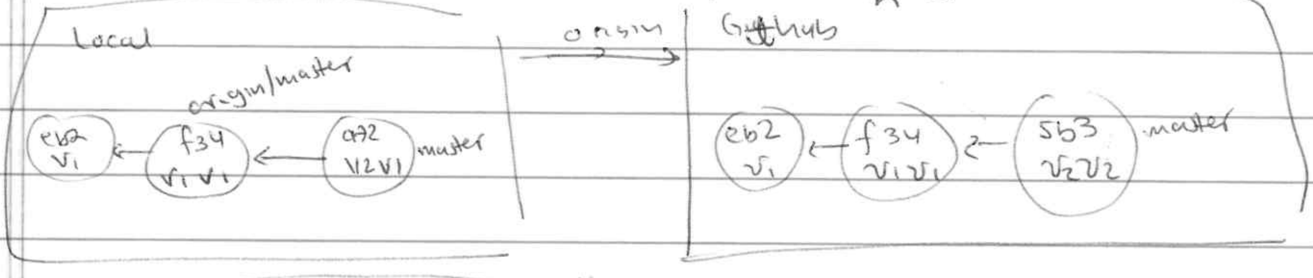
on local commit

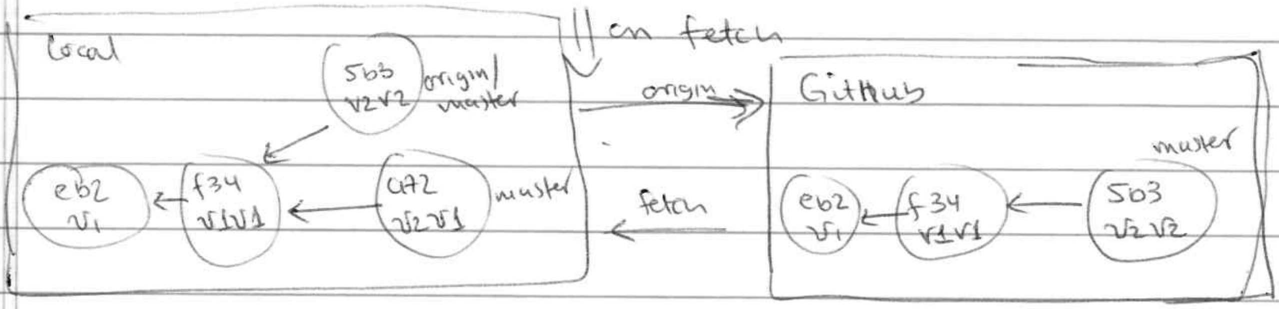
* This is saying, the last time I interacted with the remote, this is where the master was.





git fetch: like git pull without trying to merge.
→ update local copy of the remote branch
↳ use when potential conflicting changes may arise from pull.

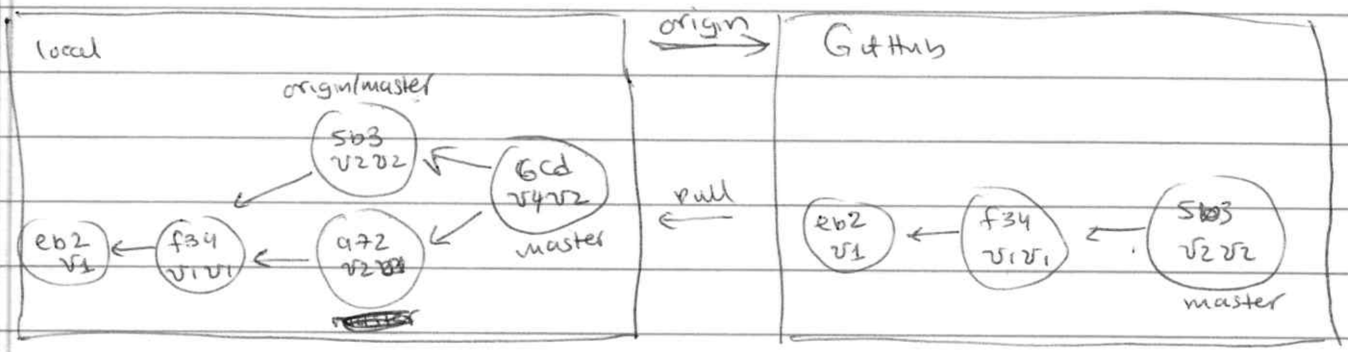




origin/master : local copy of remote ~~m~~master branch.

git ~~fetch~~ pull = git ~~pull~~ fetch + git merge.

git pull = git fetch + git merge.



git fetch origin :→ update all of the local copy of every branch on the origin remote. (in this case just one origin/master)

inspect {

git log origin/muser ← ~~show~~ commit log from origin/master commit
$\underbrace{\text{origin/muser}}_{\text{branch name}}$

git diff origin/master master ← compare these two commits.

local repo

- local master ⇒ master
- Github master ⇒ origin/master

} There are remote branches and local branches in the repo.

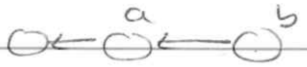
* After resolving merge conflict → git add «file»
git commit
← no message, git will add a default merge message.

Fast Forward Merge:
Criteria: The branch you are merging into is an ancestor of the branch you are merging from.

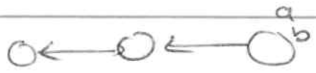
Feb 06th 2015

Fast forward merge example.

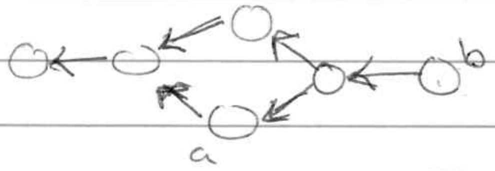
Occurs when you merge two commits where one commit is an ancestor of another. ie a is an ~~ancestor~~ ancestor of b.



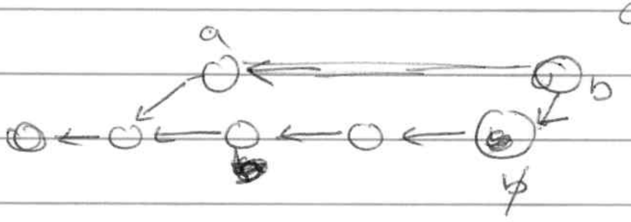
if you wanted to merge b and a : just update a to



also

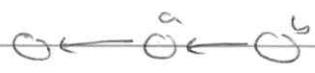


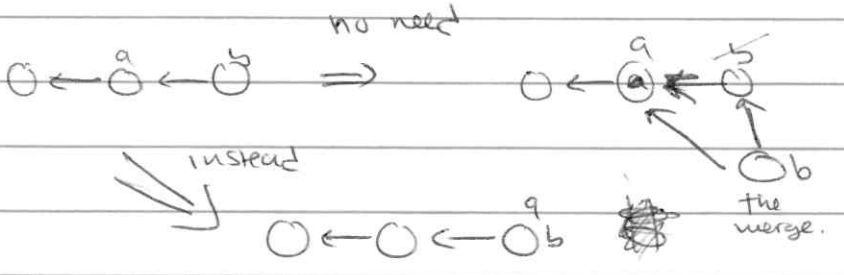
If a and b are not related by ancestry then a new merge commit is create. ie



On b branch, then merg a into b.

(The branch that was checked out gets updat to the merge commit

If we tried to handle  with a merge commit instead, what would happen



$\cancel{b}$ and b contain exactly the same information

So no need of the merge commit.

- Work flow you can use to obtain feedback on your changes before you update the master branch.

  ↳ use Pull Requests.

  (a better name might be: Merge Requests)

  Pull Requests: Asking collaborator/or other to incorporate your changes (i.e. your branch) into the main project branch (i.e. master branch) ON GITHUB

  "Merge Pull Request" button ⇒ only shows up if merge can happen without conflicts.

- In pull request page on github, can comment on whole change or place an inline comment.

- Apparently: $ git pull origin master : ~~also~~ updates → (1) local ~~master~~ branch in repo, → (2) staging area, → (3) working directory

  ↳ I guess this is because pull = fetch + merge and the merge ~~will~~ update ~~the~~ commit that the master branch reference)

  ↳ To avoid this, use git fetch then merge when ready.

- Pull requests → A merge resulting from a pull request updates the specified branch ON GITHUB.

- Pull request → like asking ~~for~~ a code review.

- Merges on Github always creates a new commit. (i.e. no fast-forward merges).
- Want to ~~delete~~ branch that was merged into master after merging as a result of a pull request on github.

- In a collaborative environment it is often only acceptable to make changes ~~from the~~ to the master branch through pull request.
  ↳ ∴ code change are visible to others before being incorperated into master branch.

- if two pull requests conflict then,

(1) ~~One~~ pull requests gets incorperated (i.e. merged) into the master branch on github (remember, a merge commit is alway created here).

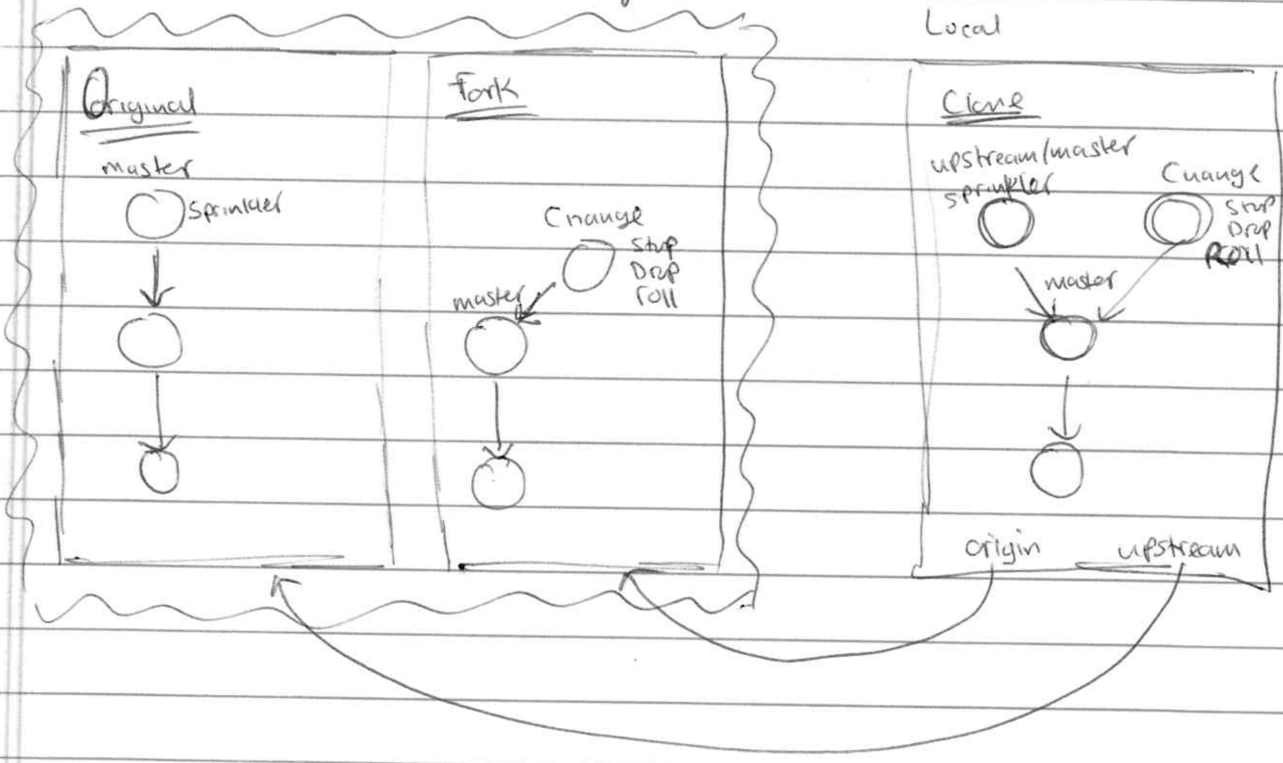
(2) The originator of the other pull request, then pulls the master branch from github into their local repo. (master & origin master) get ~~ed~~ updated

(3) Check out branch the pull request was made for → merge master branch into your local pull request ~~branch~~. → there by fixing any conflicts

(4) Push the newly merged pull request branch to Github.
  ↳ This will update ~~the~~ pull request itself on Github.

IDEA: Merges into to master branch only happens on Github via pull requests.

Merge Conflicts in Pull Request



Upstream: Name of remote typically given to reference original Repository that was forked.

~~Mrg~~ picture: merge conflict - fix by

↳ pull from upstream into local master
↳ ~~update~~. merge upstream change into local master. ← done automatically by pull
↳ merge. master branch into change branch
↳ push change branch into ~~For~~ Fork @ origin → This updates the pull request.