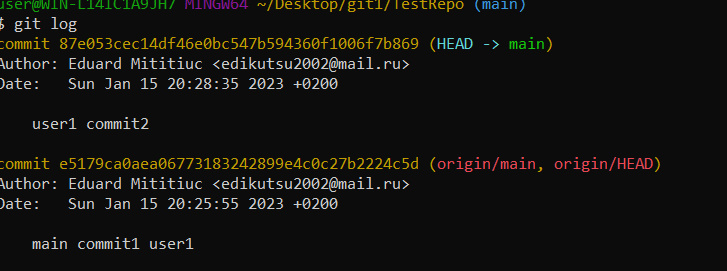
**ATENTIE!**

Nici-una din comenzile git status, git log, git log origin/main, etc. nu ne va arata date reale ACTUALE din remote repo pana nu vom da git pull sau git fetch(daca vrem intai sa analizam si apoi sa punem totul la noi).

**Unica comanda care ne arata daca remote branch e cu vreun commit inainte sau in urma e git remote show origin, chiar si fara git fetch**

* git remote – vedem lista de rep remote
* git clone urlRepo – cloneaza repozitoriul pe pc al nostru
* git log – Arata logurile doar pentru local branch. In cazul in care avem local repository copiat de la unul remot, git log va arata mereu si care e commit ultimul in branch curent din remote, in cazul in care noi suntem inainte cu un commit sau mai multe. Daca insa remote branch e inainte cu un commit,chiar si dupa fetch, nu vom vedea asta in git log, ci in git log origin/main doar. Asta e din cauza ca atunci cand branch a noastra e inainte cu un commit, ea deja a avut ca HEAD commit actual din remote si deci stie bine de el.



(origin/main) – spune ca acest commit e prezent si in main branch din origin(remote)

(origin/HEAD) – spune ca HEAD commit din main branch din origin e anume acest commit

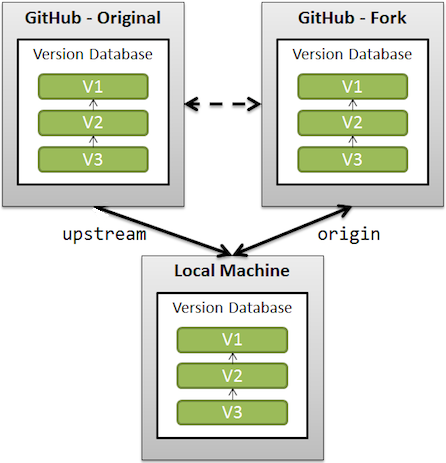
Insa, HEAD -> main arata ca main branch din local repo are alt commit ca HEAD

* git push – trimite toate snapshots facute de noi in branch din repozitoriul clonat pe pcul nostru in branch de pe repozitoriu de pe github. Nu va functiona daca nu avem cel mai nou commit din remote branch in local branch
* git push --set-upstream origin test1 – pune noua branch test1 in origin. Folosim comanda asta daca cream un nou branch in local repo care nu exista in remote si vrem sa dam git push la ea.
* git config --global credential.helper cache – face ca pentru 15 minute, dupa ce va cere sa introducem iar login si parola la vreo comanda, sa nu trebuiasca sa o tot introducem.
* git pull – face ca toate snapshots de pe repozitoriul remote de pe git, care sunt nesalvate, sa fie puse pe repozitoriul local, e un copy. E combinatia dintre git fetch si git merge
* Pentru a salva parola:

[Caching your GitHub credentials in Git - GitHub Docs](https://docs.github.com/en/get-started/getting-started-with-git/caching-your-github-credentials-in-git)

[Connecting to GitHub with SSH - GitHub Docs](https://docs.github.com/en/authentication/connecting-to-github-with-ssh)

* git remote -v – arata care e repository de unde vin datele si care e repsitory unde ele se duc.
* **git remote add upstream linkRepoOriginal – facem ca local repo sa fie syncronizat cu online original repo si sa se adauge upstream ca referinta la el.Daca de ex avem un forked,origin va fi referinta la el si e bine sa il syncronizam cu repo original, ca daca acolo apar modificari, sa le putem obtine cu pull upstream. Orice git push tot se va duce la noi in forked, dar de acum in loc de origin putem pune upstream, ca sa facem referinta nu la forked repo, ci original!**

****

* git remote show origin – arata mai multe info despre origin repository(inclusiv si branches existente)
* git branch -r – vedem doar remote branches la care repository al nostru e conectat:
* git fetch – sincronizeaza local branch in care suntem cu cea din remote repository. Totusi, noile commituri nu sunt **automat** puse in local branch. Necesita merge mai apoi
* git log origin/master – vedem log pentru master din origin
* git merge origin/master – adauga noile date din master branch din remote repository origin in master branch de la noi din local repository. Folosim asta dupa git fetch
* git checkout numeBranch – va copia un nou branch din remote repo in local repo si ne va pune in ea.
* git remote update – da fetch la toate local branches ca sa le conecteze cu cele remote, dar nu copie nimic in ele! Va necesita merge

**Rebase**

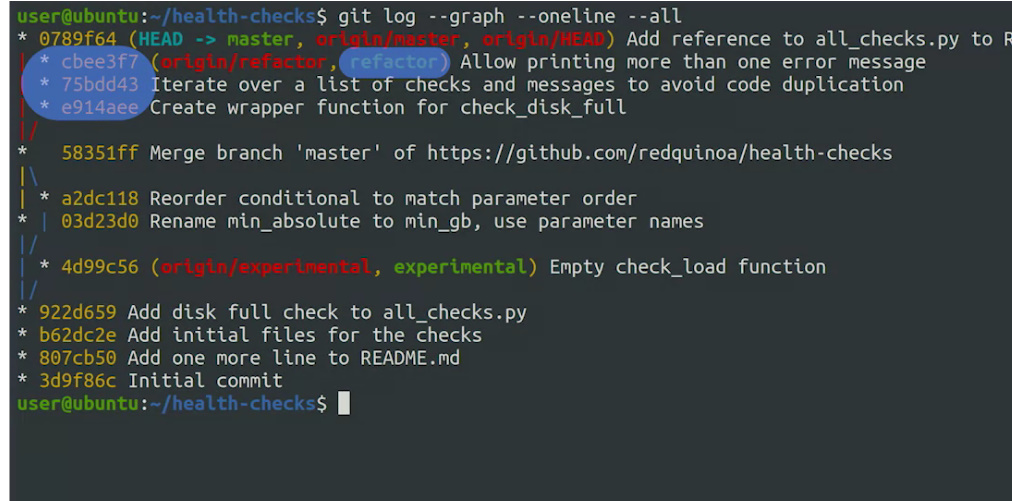
* git rebase NumeBranch – scriem numele la branch pe baza la care vrem sa se faca rebase, adica de la care sa ia cel mai recent commit/uri si sa fie pus/e ca principal, si il pune unde el ar trebui sa fie in istorie, adica dupa fostul cel mai recent commit, si sa recreeze commiturile din istorie
* Daca apar merge conflict:

le rezolvam

git add

git rebase --continue – continuam mai departe cu urmatorul commti, daca exista. Daca nu, se va recrea cel actual si gata

* git rebase --abort – anulam



Din grafic vedem ca refactor are 3 commits create pe baza la commitul care nu mai e cel mai recent in master. Deci, refactor are ca commit de baza 58351ff, care a fost ultimul candva in in master, dar acum dejaa ultimul e 0789f64, dar nu e el de baza in refactor.

Experimental inca are ca commit de baza 922d659

922d659 a fost commit de baza si pentru refactor si a fost ultimul si pentru master

Apoi, refactor a creat inca 2 commituri, si pe baza lor s+a creat unul nou in master

* git rebase -i NumeBranch – ne permite sa alegem actiuni ce sa fie facute pentru fiecare commit dupa ce acesta se va combina cu cel venit din alt branch
* git rebase -i HEAD~X – face ca sa nu combinam commiturile cu vreunul din alt branch, dar sa facem actiuni doar asupra commiturilor din branch actual, dica HEAD. Trebuie neaparat sa specificam dupa ~ cate commituri modificam.
* git push --delete origin NumeBranch – sterge o remote branch