**Head – se refera la ceva recent, de ex cel mai recent commit**

**-p – de la patch**

**Configurare**

* Git --version – vedem versiunea
* Git config --global user.name “Nume” – setam numele celui ce foloseste git
* Git config --global user.email “email”

--global – inseamna ca setam aceste optiuni pentru toate git repozitoriile ce le folosim. Putem seta diferite valori pentru diferite repositorii.

* Git config -l – info despre configuratia git in repozitoriul local,sau curent
* Git init – creaza un git repository gol in folder. Incepem a lucre cu git.

Repozitoriul e .git, un fel de baza de date ce va pastra istoria versiunilor si modificarile.

Cand copiem un repozitoriu deja existent, copiem si tot ce contine el.

Cand cream unul nou, el nu contine date salvate.

* Git clone – creaza o clona a unui repository deja existent undeva

**Salvare fisiere**

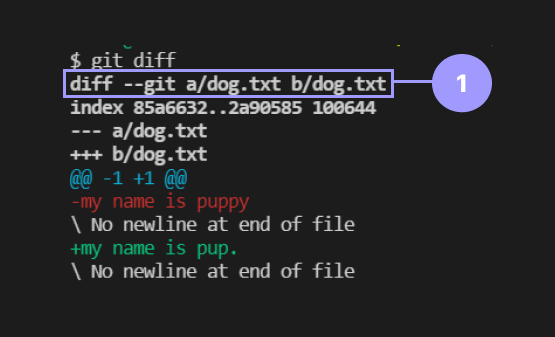
* Git add file – git incepe sa urmareasca fisierul dat si il pune in **staging area.** Git mereu vede ce modificari facem in orice fisier, dar nu vom putea sa salvam modificarile aseta in .git pana fisierul nu va fi in staging area, unde se salveaza modificarile facute, ca mai apoi sa fie salvate in .git. Asta si face comanda data, pune fisierul in staging area. De fiecare data cand apelam comanda, git salveaza schimbarile ce s-au facut pana la apelarea ei, adica face cate o poza.
* Git add . – adauga toate fisierele care au . in nume in stagging area.
* Git add \* - adaugam absolut totul in staging area
* Git status – ne arata info despre working tree. Aici vedem daca s-au facut ceva modificari la fisiere sau nu, indifferent ca sunt tracked sau nu.
* Git add -p – ne arata ce modificari au fost facute intre file din stagging area si cel cu care lucram acum. Apoi ne intreaba daca vrem sa salvam ori nu In staging area modificarile
* Git commit – pune toate modificarile noastre din Stagging area in .git. Ii spune lui git ca noi vrem sa salvam modificarile noastre. Dupa ne va cere sa scriem un commit message. Scriem mesajul in primul rand,CTRL + O,ENTER si dam Shift+Z pentru a salva. ***Q pentru back***
* Git commit -m “comenatriu” – adaugam deodata si comentariu
* Git commit -a -m – adauga in commit toate modificarile facute la fisierele tracked fara a mai scrie git add file din nou. Nu adauga in commit fisierele untracked! Adica fisierele care nu au fost niciodata puse in staging area cu git add si nu exista nicio poza de a lor, nu sunt afectate de comanda data. Asa nu trebuie sa scrie git add pentru toate fisierele tracked. Se vor salva modificarile cele mai curente ale fisierelor tracked, chiar daca in staging area sunt modificari mai vechi ale lor. Se vor pune in commt si modificarile din staging area ce nu au fost modificate.Comanda data trece peste staging area, nemaiputand schimba ceva inainte de a crea commitul.
* git reset --hard commitId- sterge un commit

**Informarmatii despre cele salvate**

* Git log – ne arata commiturile cu mesajele lor
* Git log -p – ne arata modificarile salvate in **fiecare commit**. Arata ce modificari facute in fisiere a salvat fiecare commit. Fiecare commit va arata ce are el si ce a avut commitul anterior, nu arata totul impreuna ca git diff. Ce e comun, ramane aratat la commitul anterior.
* Git log -p -n – ne arata salvarile salvatele in n fisiere
* Git log --stat – Nu ne arata ce anume s-a modificat in fiecare commit, dar ne arata cate inserturi si deleturi s-au realizat in fiecare file si care file anume.
* Git show - ne arata modificarile salvate in head commit si ne Arata ce modificari facute in fisiere a salvat acest commit. **Comanda e doar pentru commitul head**
* **git log --graph –** arata istoria la fiecare commit, cum au interactionat branch cu ele
* **git log --graph** **--oneline** – arata ca si comanda de sus, doar ca fiecare commit ocupa o linie doar, adica se arata doar id si atat.
* Git show id – arata modificarile salvate de un anumit commit. Id apare dupa cuvantul commit in git log. Aici vedem

**Git Diff**

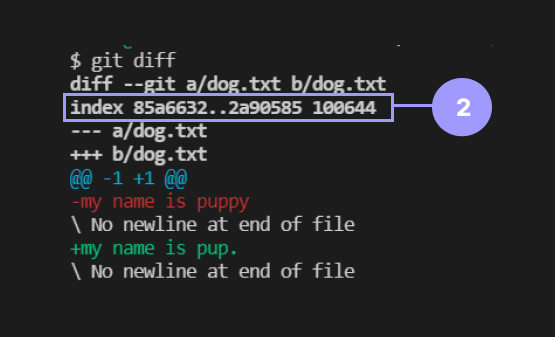
* **Git diff** –arata diferentele intre Working Tree si Staging Area. Daca staging area e goala, atunci se uita in commit HEAD.



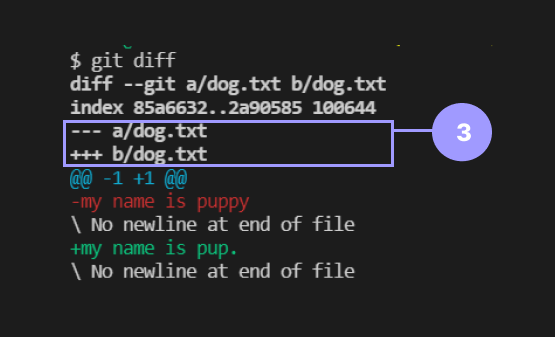
Linia asta ne arata ca se face comparatia intre 2 versiuni ale fisierului.

A – versiunea veche a fisierului

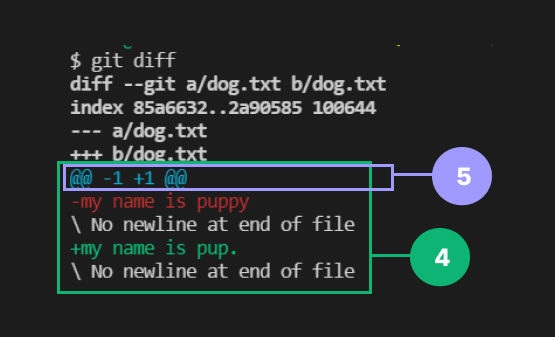
B – Versiunea Noua



Acestea sunt metadate despre fisierul nostru, Primele doua denumiri sunt hashes care ne compara cele 2 fisiere. 100644 e fisierul care contine datele despre versiuni.



Git atribuie – versiunii A(vechi) a fisierului si + versiunii B(noi).



Git in mod normal arata o parte din fisier, si anume cel care au fost modificat, nu fisierul intreg.

Linia cu – vine de la versiunea A(veche) a fisierului

Linia cu + vine de la versiunea B(noua) a fisierului.

-1 inseamna ca se extrage o linie din A, si anume linia 1, care a fost modificata

+1 inseamna ca se extrage o linie din B si anume linia 1, care a inlocuit-o pe cea din A

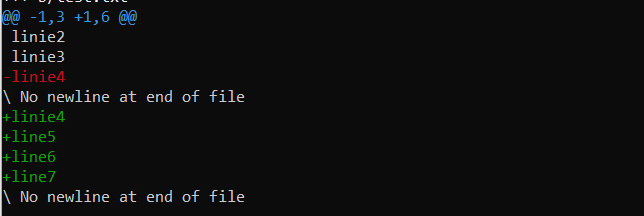
-3,4 – din A se extrag 4 linii incepand cu linia 3, si printre ele unele neaparat au fost modificate, si deci 4 linii din cele afisate apartin lui A

+3,2 – din B se extrag 2 linii incepand cu linia 3, si printre ele unele neaparat le-au modificat pe cele din A. Sau putem spune ca din liniile afisare, 2 apartin lui B.

Liniile necolorate din A tot se socot la B! Pe ecran e pus partea modificata din fisier, si ne arata cate linii din cele aratate apartin lui A si cate lui B

**Putem spune ca – arata linia stearsa din A si + linie adaugata in B**

Comanda data recreaza ca si cum fisierul, si ne arata pe linii ce a fost modificat, nu ne arata aparte la fiecare fisier!



Liniile incolore arata ca acelea nu au fost modificate.

Liniile necolorate tot se calculeaza la cele din B

Cand Dupa o linie cu – merge una cu +, e clar ca cea cu + a inlocuit-o pe cea de mai sus in fisier.

No newline and end of file – inseamna ca nu mai sunt linii modificate dupa linia de mai sus. Ce e in staging area si work tree vor fi luate impreuna.

* Git diff --staged - compara modificarile din staging area cu cele din commit
* Git diff HEAD – compara modificarile din work tree si staging area cu cele din commit
* Git diff commit1 commit – compara 2 commituri

**Modificare Fisiere fizice**

* Git rm fisier – sterge un fisier tracked din repositoriul nostru. Fisierul trebuie sa fie tracked, adica ori sa fie in vreun commit, ori sa fie in staging area. Cand fisierul se sterge, si daca el era in stagging area, in stagging area el devine deleted, si se va salva informatia ca el a fost sters si deci tot ce a fost salvat in stagging area despre el e sters, cu exceptia informatiei ca el a fost sters.Nu am avea cum sa il adaugam acolo altfel, asa cum el e sters, de aia Git o face automat. Daca vom da git commit, fisierul sters nu va mai fi in commit, caci el nu mai exista propriu zis nici in staging area, ci a ramas acolo doar informatia ca el a fost sters, deci se va crea un commit fara acel fisier. Daca asta e unica modificare facuta, commitul va fi ca cel HEAD doar ca fara acel file.
* Git rm fisier -f – forteaza stergerea unui fisier.
* Git mv file nume\_nou – modifica numele unui fisier tracked. Daca el e salvat in vreun commit, in staging area el va aparea ca renamed automat. Noul commit va avea fisierul cu nume modificat.

**Undo si reset**

* Git checkout – restabilim tot ce este in staging area sau commit. Daca nu e nimic in staging area, se ia din commit, daca e ceva, doar sdin staging area. Se foloseste cea mai recenta poza, fie ca e din staging area sau commit.
* Git checkout file – facem ca un fisier sa revina la o stare salvata de mai inainte, care fie e commited, fie staged. Daca avem ceva in staging area, se restabileste de acolo, daca nu, din commit atunci se restabileste.
* Git checkout file -p – ne arata intai ce a fost modificat in noul fisier, fata de ceea ce se gaseste in cea mai recenta salvare a lui si ne intreaba ce vrem sa facem mai departe.
* Git checkout Hash – facem ca un anumit commit sa fie restabilit
* Git reset – resetam staging area, in caz ca am adaugat ceva acolo ce nu vrem sa commitem.
* Git reset -p – arata ce e pe cale sa fie sters si intreaba ce vrem sa facem mai departe
* Git reset file – stergem un file din staging area
* Git reset HEAD file – stergem un anumit fisier din staging area HEAD(actualele poze luate)

**Modificarea commiturilor existente**

* Git commit --amend – git va actualiza cel mai recent commit salvat cu ceea ce e nou in staging area. E bun in caz ca dam git commit si apoi ne dam seama ca am uitat sa punem ceva. **ATENTIE**! Comanda data creaza un nou commit, si il sterge pe cel mai precedent. CTRL + O, ENTER, SHIFT + Z pentru a salva. Nu il folosi in public commits!Asa se modifica un commit folosit si de altii.
* Git commit --amend -m “Nou mesaj” – fara sa mai lucram cu meniul

**Rollback(reintoarcerea) commit**

* Git revert HEAD – Setam ca HEAD commit un nou commit diferit de ultimul care e HEAD. Sa zicem ca modificam un commit si apoi il facem commited, si facem ceva gresit. Revert nu inseamna doar undo.. Inseamna crearea unui commit care sa contina exact ceea ce nu contine commitul rau,adica ultimul pus. Adica, se creaza un commit identic cu penultimul, si asa noul commit nu va avea ce s-a salvat in ultimul. Se va crea un nou commit, dar cel rau oricum va ramane salvat. Ca si cum dam undo la commit HEAD.

HEAD – pointer la actualul commit

E bine sa scriem la mesaj si de ce facem rollback

CTRL + O(SAVE) +ENTER + si SHIFT + Z pentru a iesi si salva din meniu.

**Dupa asta, se vor modifica deci si fisierele din working tree ca sa fie identice cu ce pastreaza noul commit.**

* **Git revert id –** cream un commit care contine tot ce are HEAD commit, dar nu contine modificarile salvate de commit al carui id il scriem.
* sa zicem ca avem 4 commituri:

4commit – am creat fisierul 4.txt

3commit – am creat fisierul 3.txt

2commit – am creat fisierul 2.txt

1commit – am creat fisierul 1.txt

Deci, 4commit are 4 fisiere. Daca dam **git revert 2commit**, se va crea un nou commit, dar ce nu va avea ceea ce a salvat 2commit, adica fara fisierul 2.txt, insa vor exista fisierele 4.txt,3.txt,1.txt

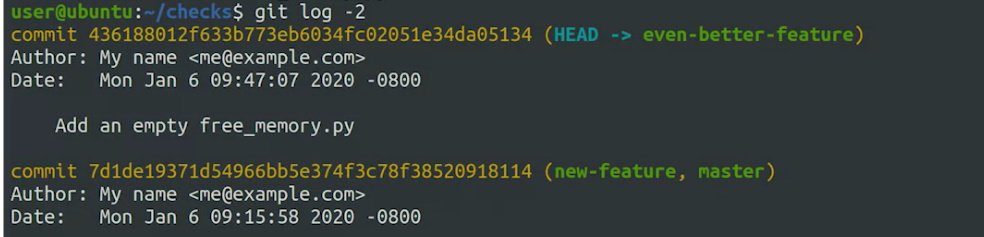
**Nu se sterge si ce salveaza commiturile existente dupa acel commit al caruit il indicam!**

**Branches**

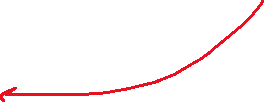
* **git branch –** arata o lista cu toate branches din repozitoriu.

Cea verde cu \* e cea la care suntem conectati la moment.

* **git branch Nume –** cream un nou branch
* **git checkout NumeBranch –** trecem la un nou branch si face ca woking tree sa fie actualizat la branch selectat. Adica, sa se contina tot ce este in branch selectat, inclusiv fisierele cu hitory. Comanda e si pentru a reintoarce fisiere, si pentru a trece la alt branch!
* **git checkout -b NewBranche –** creaza un nou branch si ne muta deodata la el.

****

**HEAD –** branch la care suntem conectati acum



Asta ne arata ca new-feature si master pointeaza la acea poza sau commit

* Cand dam checkout sau commit intr-un branch, aceste modificari vor fi adaugate doar in istoria acelui branch
* **Git branch -d Nume** – sterge un branch

Daca in branch ce vrem sa il stergem sunt lucruri ce nu au fost puse in branch parinte, vom primi un avertisment.

* **git branch -D Nume** – sterge fortat un branch
* **git merge numeBranch** – ia snapshoots dintr-un branch si le combina cu cele din alt branch. Cand folosim comanda, se vor copia datele stocate in branch scris in comanda in cel in care ne aflam cand scriem comanda. Gen daca scriem comanda git merge test cand suntem in Master, tot ce este in test branch se va combina cu ce este in Master(vezi in week2 cum)
* **git merge --abort** – pentru a anula merge daca apare conflict(vezi in week2)
* **Pentru a rezolva conflictul, trebuie sa alegem noi in file ce vrem sa pastram si ce nu, caci git ne va marca in fisier zona cu conflict**.

Linia1

<<<<<<< HEAD

Linia2

=======

ERROR

>>>>>>> new

* Apoi, dam **git add numeFile** pentru a arata ca am rezolvat noi conflctul si **git commit**