GIT

# Local

**Initializare**:

git init initializare repository pt un anumit folder ( creaza un repository gol ".git" si ia folderul in curs ca locatie pentru do

**Status**

git status - arata starea modificarilior din fisiere

Statusuri :

* Untracked = fisier abia creat
* Unstaged = fisier cu modifcari care nu au fost inca pregatite pt commit
* Staged = fisier gata pt commit ( in staging area)
  + Ex : git add numeFisier.text
  + git add ‘\*.txt’ (are efect si pt subdireactoare pt ca trece intai prin git si pe urma prin shell)
  + git add \*.txt (doar pt directorul curent)
* Deleted = fisier sters din folder, dar nu inca din repository

Staging area= loc unde se grupeaza toate fisierele pt a fi commited

Commit = un snapshot al repository-ului

Git commit –m  "Mesaj descriptiv" -adauga la repository toate modificarile din staging area

**Istoric**

git log

git log – summary

**Diferente**

git diff HEAD - arata diferentele intre ultimul commit si fisierele de pe disk

git diff --staged -arata diferentele dintre ultimul commit si ce e in staging area

git diff -din poza vad ca arata diferentele dintre staging area si ce e pe disk

**Unstage**

git reset octofamily/octodog.txt -scoate din staging area

**Undo**

git checkout -- numeFisier.txt

-readuce fisierul la forma ultimului comit

**Atentie** !! comanda checkout face si schimbarea intre branch-uri. pt asta folosim -- pt a vedea ca nu mai e nicio optiune dupa -- , si deci face undo le fisier chiar daca avem un branch cu numele numeFisier.txt

**Remove files**

= sterge fisierele de pe disk si le pune in staging area ca fiind sterse( mai trebuie sa facem commit pt a le sterge si din repository)

git rm ‘\*.txt’

git rm –r numeFolder -sterge recursiv toate folderele si fisierele din folderul curent

# Operatiuni intre branch-uri

= crearea a mai multe branch-uri permite lucrul in paralel pe mai multe issues. Se copiaza branch-ul curent in noul branch

**Creare**

git branch alDoileaBranch -creaza branch-ul cu numele dat care va avea continutul branc-lui curent

**Permtuare**

git checkout alDoileaBranch –se muta pe branchul dat

**Merge branches**

Dupa ce am facut modificari intr-o ramura putem sa ne intoarcem pe master (sau pe alta) cu comanda checkout si sa le aplicam

git merge alDoileaBranch

**Delete**

git branch –d alDoileaBranch – sterge branch-ul dat. Nu functioneaza daca branch-ul nu e merged

git branch –d --force alDoileaBranch - forteaza stergerea chiar daca nu e merged

# Server

**Remote repository**

= repository pe server (ex pe GitHub) cu care sincronizam continutul repository-ului local

git remote add origin https//adresa\_server.git

-creaza o conexiune la repository-ul de pe server, cu numele by default origin de unde putem obtine(pull/fletch) si adauga(push) modificari

**Push**

git push –u origin master

-adauga modificarile din repositoryul local de pe branch-ul curent in repository-ul remote origin pe ramura master (ramura principala by default)

-u salveaza parametrii pt ca dupa sa putem pune doar git push

**Pull**

git pull origin master

obtine si fuzioneaza modificarile din repository-ul remote in repository-ul local si in workspace(datele fizice de pe disk)

Caz posibil : am in local niste modificari pe care vreau sa le pastrez si care nu sunt inca commited. Pot sa folosesc : - git stash – pt a ascunde modificarile locale

* git stash apply – pot re-aplica modifcarile dupa ce am facut pull

**Fletch**

=obtine si fuzioneaza modificarile din repository-ul remote in repository-ul local dar **NU** si in workspace(datele fizice de pe disk)

TEST DIF