

Git

Ce este git ?

Git este un software pentru urmărirea modificărilor în orice set de fișiere, folosit de obicei pentru coordonarea muncii între programatori care dezvoltă codul sursă în colaborare în timpul dezvoltării software. Obiectivele sale includ viteza, integritatea datelor și suport pentru fluxuri de lucru distribuite, neliniare.



De ce îl utilizăm?

Scopul acestui tool este de a putea urmări cine, ce și când a făcut anumite modificări într-un cod.

Dacă ceva nu este bine în acel cod, putem face revert la versiunea anterioară foarte ușor . (sau de exemplu putem vedea cum arată codul nostru în urmă cu 2 ani)

Dacă nu există acest tool fiecare ar fi trimis unul altuia codul lui și neavând o sursă centrală unde să îl ținem ar complica foarte mult lucrurile mai ales când numărul de programatori este mare.



De ce îl utilizăm?

Cele două mari avantaje ale git sunt următoarele:

Putem lucra împreună la un singur cod

Putem face track la istoricul acelui cod

Este **gratuit**, foarte **eficient** și **rapid**

Este foarte **popular**

Este foarte **căutat** pe piața muncii



Tool-uri utile pentru GIT

VSCode este printre cele mai populare tool-uri. Conține foarte multe plugin-uri printre care foarte multe pentru git.

Acesta se poate instala de <https://code.visualstudio.com/>

Link-ul pentru git se poate găsi <https://git-scm.com/downloads>

Git cheatsheet: <https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet>



Configurare git

```
git config --global user.name "yourname"  
git config --global user.email "youremail"
```

Detalii suplimentare se pot găsi [aici](#)

Inițializare

Vom crea un director nou și îl vom accesa. O dată ce suntem în directorul nou creat vom rula comanda **git init**.

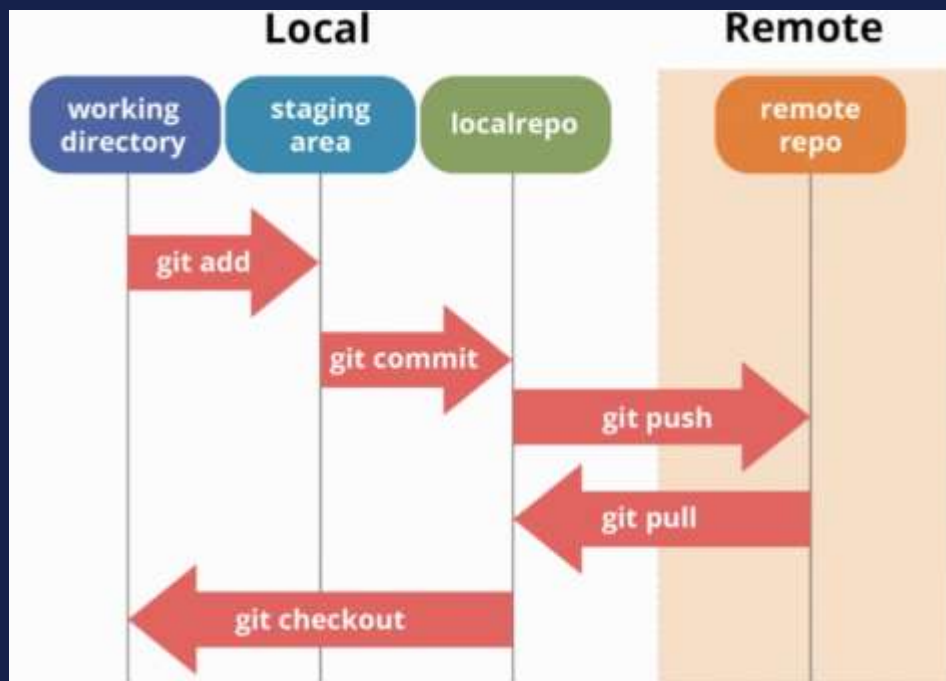
Vom putea observa că un fișier ascuns **.git** a fost creat.

Aici vor fi toate informațiile despre istoricul proiectului nostru (ex despre branches). De aceea nu trebuie să ștergem acest director sub nicio formă.



Cum funcționează git?

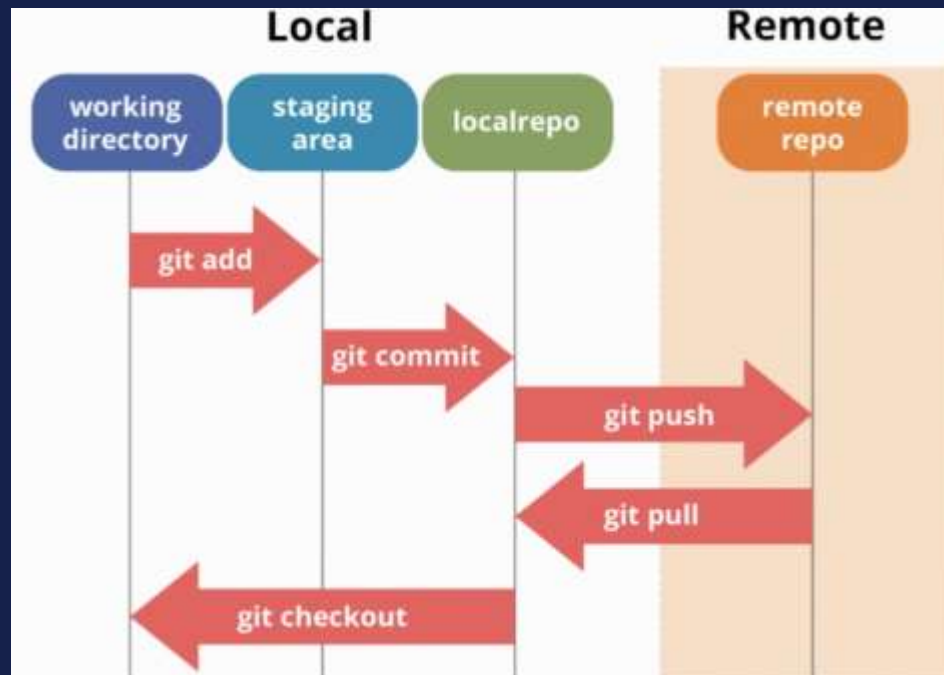
De exemplu, lucrăm la un script de python și modificăm anumite variabile. Putem trimite acele fișiere în “**staging area**” cu git add .



Sursa: [Link](#)

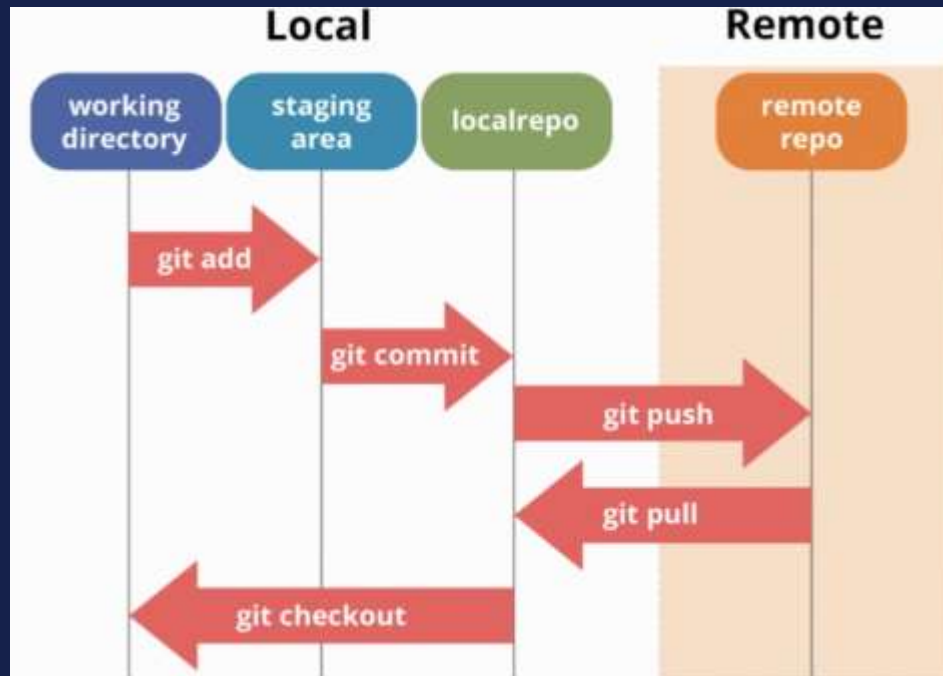
Cum funcționează git?

Când suntem singuri că dorim acele schimbări vom face un **commit** pe acele schimbări (facem un snapshot la acel cod din acel moment). Fiecare commit trebuie să conțină și un mesaj pentru a descrie ce schimbare am dorit să facem astfel încât toată lumea să le poată urmări mai ușor ulterior.



Git add

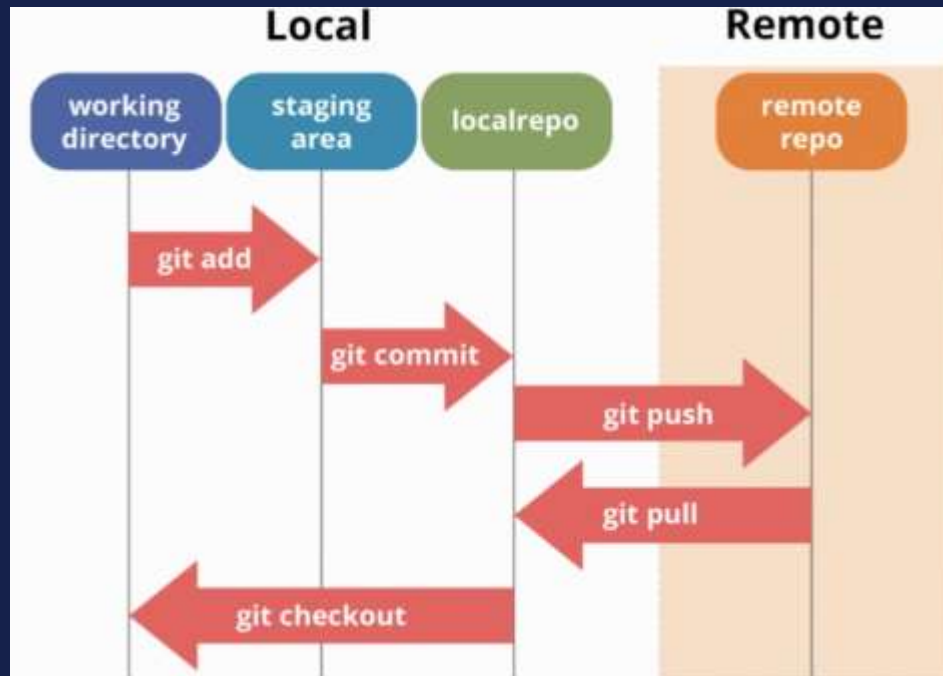
Comanda git add adaugă o modificare în directorul de lucru în zona de staging. Astfel, îi poți spune lui Git că doriți să includeți actualizări pentru un anumit fișier în urmatorul commit. Cu toate acestea, git add nu afectează cu adevărat repository-ul într-un mod semnificativ - modificările nu sunt de fapt înregistrate până când rulați git commit .



Git add

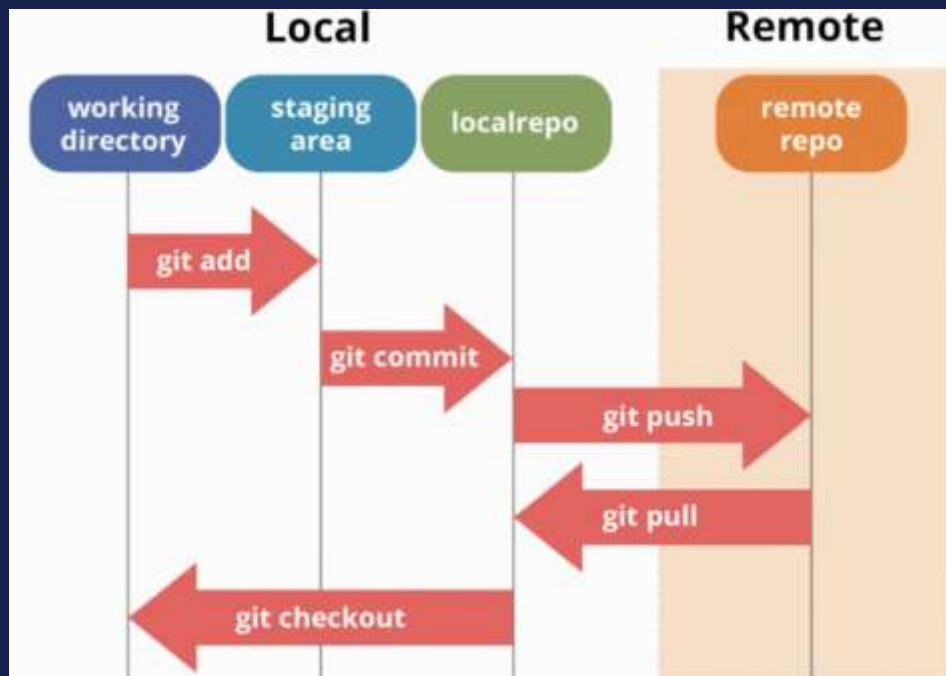
Pentru a adăuga un fișier din working directory în staging area folosim:
git add <file>

Undo git add (pentru modificările la care nu am dat commit):
git reset <file>



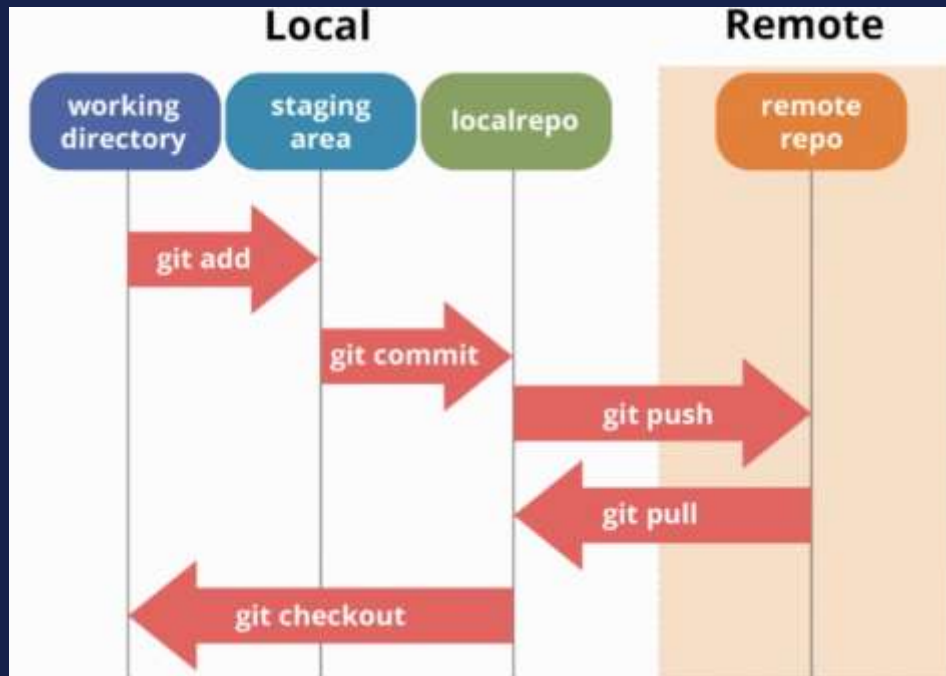
Git commit

Comanda git commit captează un instantaneu(snapshot) al modificărilor realizate în prezent în proiect. Aceste snapshot-uri pot fi considerate versiuni „sigure” ale unui proiect.



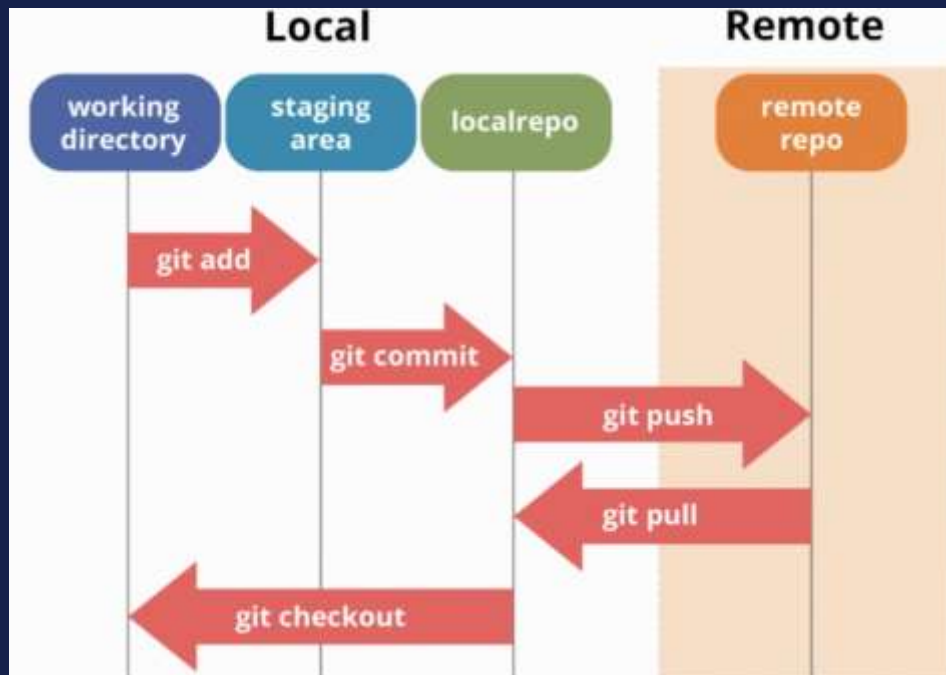
Git commit

De exemplu, putem modifica mai multe fișiere, le dam add pe rând și când suntem siguri de modificări putem face un commit. De fiecare dată un mesaj explicativ trebuie să însoțească git commit. Fiecare commit va conține informații precum un id unic, mesajul introdus anterior, data când a fost făcut și autorul .



Git commit

Este util să facem commit-uri des, însă este de evitat să facem asta la orice simplă modificare. Este bine să facem un commit atunci când considerăm că suntem mulțumiți cu stadiul modificărilor. Aceste commit-uri voi apărea în history. Putem vedea acest history cu comanda: **git log**



Crearea și ștergerea unui branch

Dacă dorim să creăm un nou branch, vom rula comanda urmatoare:

git checkout -b <new-branch-name>

Dacă doar vrem să ne mutam pe un alt branch folosim doar:

git checkout <existing-branch-name>

Dacă dorim ștergem un branch, vom rula comanda:

git branch --delete old-branch

Pentru a afișa branch-urile existente, vom folosi comanda:

git branch



Gitignore

Fișierul .gitignore este un fișier text plasat în repository care îi spune lui git să nu urmărească anumite fișiere și foldere pe care nu doriți să le încărcați în respectivul repository .

Aceste fișiere vor fi pur și simplu ignorate. Un fișier din GIT poate avea dimensiunea maximă de 100Mb. Și în acest caz poate fi util să utilizăm git ignore.

Exemplu:

```
echo "build/" > .gitignore
```

Efectul va fi că tot ce avem în folderul "build" va fi ignorat de git.
Ghid util pentru [gitignore](#)

Cum functioneaza git?

Când suntem singuri că dorim acele schimbări vom face un **commit** pe acele schimbări (facem un snapshot la acel cod din acel moment). Fiecare commit trebuie să conțină și un mesaj pentru a descrie ce schimbare am dorit să facem astfel încât toata lumea să le poată urmări mai ușor ulterior .

Sursa: [Link](#)

Github

GitHub este un serviciu de hosting ce vă permite să gestionați repository-urile Git. Dacă aveți proiecte open source care folosesc Git, atunci GitHub este conceput pentru a vă ajuta să le gestionați mai bine

Este necesar să vă creați un cont pe <https://github.com>

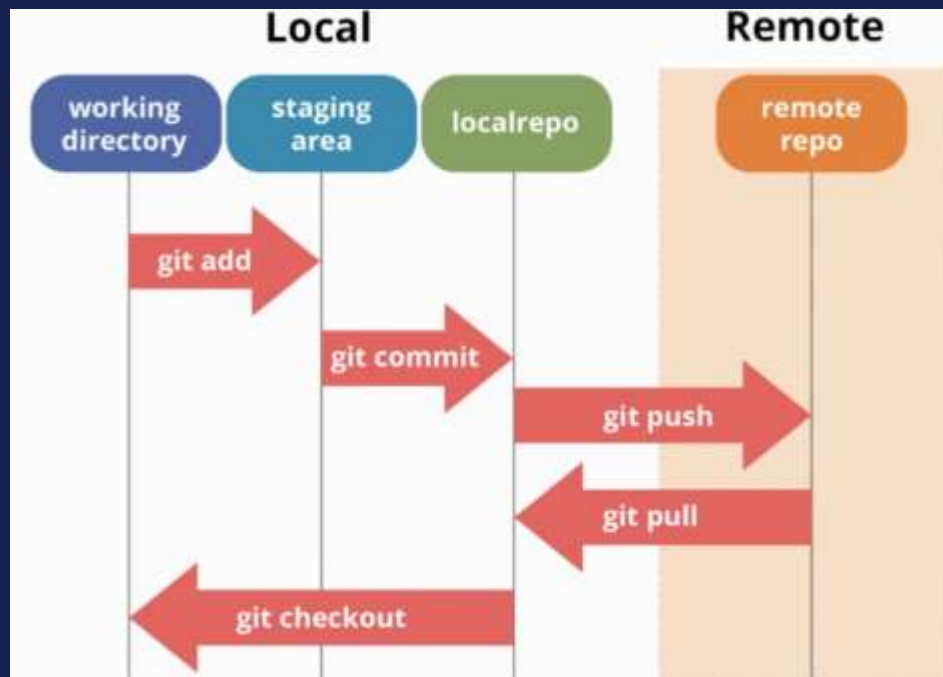
Acolo putem crea un repository și pentru a ne conecta la el va fi nevoie să ne generăm o cheie de acces sau un token

Repository public vs privat

Repository-urile publice sunt accesibile tuturor pe internet. Repository-urile private sunt accesibile unei singure persoane, persoanelor cărora le partajați în mod explicit accesul și pentru repository-urile organizației, anumitor membri ai organizației.

Git push

Comanda git push este folosită pentru a uploada conținutul repository-ului local într-un repository remote. Prin comanda de push puteam spune că este modul în care transferați toate commit-urile din repository-ul local în cel remote.



Git branch

În Git, branch-urile fac parte din procesul de dezvoltare de zi cu zi. Branch-urile din Git sunt efectiv un indicator către un snapshot al modificărilor în cod.

Pentru a vedea toate branch-urile curente folosim comanda:
git branch

Convenția este să avem un branch principal (main sau master) și să creăm pe lângă alte branch-uri pentru a dezvolta feature-uri noi. Când știm sigur că modificările sunt bune, putem merge codul din cele 2 branch-uri, astfel încât noile modificări să fie incluse în branch-ul de main.



Git branch



Sursa [imaginii](#)

Git fetch

Este folosită în cazul în care un coleg crează un nou branch (să zicem **DNRQ_minor_website_fix**) , îl pune pe repository și vrei să actualizezi asta și la nivel local pentru a-l testa modificările. După ce dăm git fetch vom putea da git checkout **DNRQ_minor_website_fix**

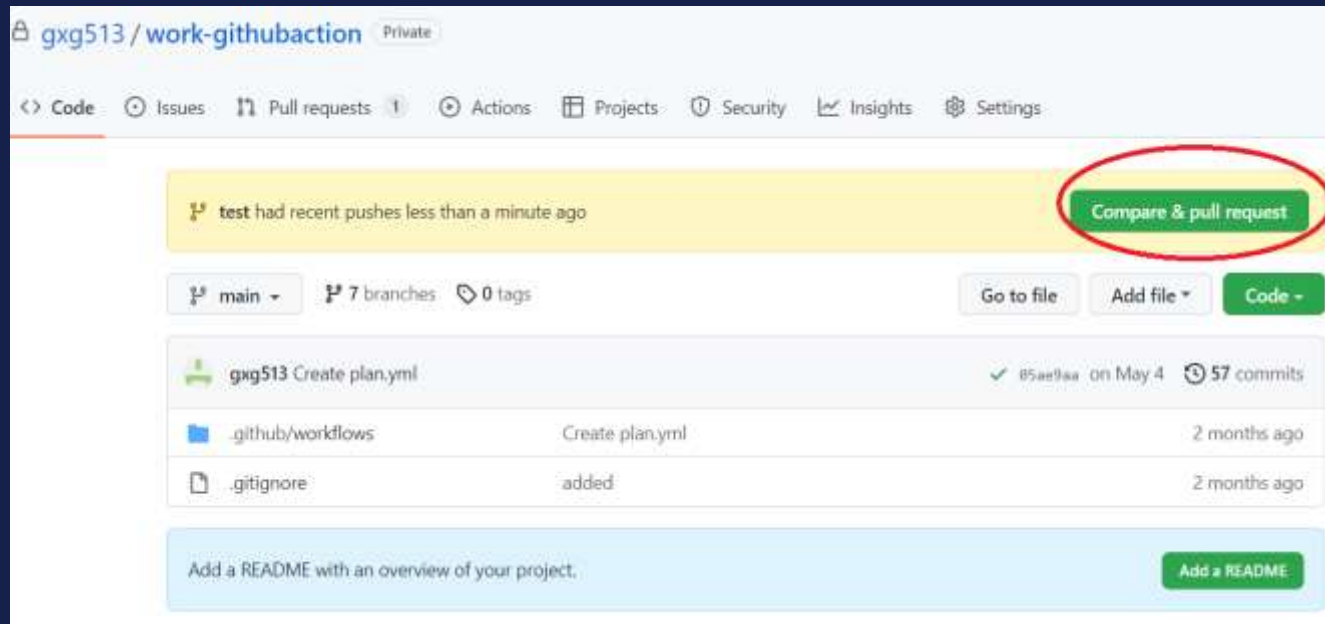
Pull request

Pull request – denumită și **merge request** – este un eveniment care are loc în dezvoltarea de software atunci când un colaborator/dezvoltator este gata să înceapă procesul de îmbinare a noilor modificări de cod cu depozitul principal al proiectului.

În practica, atunci când cineva implementează noile schimbări, să spunem că își crează un branch test care modifică culoarea header-ului de pe un website. Acesta testează noile modificări și când consideră că este totul în regulă va dori să facă **merge** între branch-ul creat și branch-ul MAIN . Va face git push și va pune noile modificări pe branch-ul test.

Pull request




După ce modificările pe noul branch **test** au fost puse pe repository, ne va apărea un buton **Compare & pull request**




Pull request

Open a pull request










Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

 base: main  compare: test  **Able to merge.** These branches can be automatically merged.


 Update .gitignore

Write

Preview

H B I         

Leave a comment

Attach files by dragging & dropping, selecting or pasting them. 

Create pull request

Pull request

După ce creăm acel PR este nevoie ca altcineva să ne dea approve, altfel nu vom putea da merge to master. De obicei este nevoie ca 1-2 persoane să-și de acordul. Acest lucru se întâmplă pentru a preveni anumite erori să ajungă pe branch-ul de main. Astfel, dacă 1-2 persoane vor analiza și ei modificările înainte ca acestea să ajungă în main branch, minimizăm eroarea și nu lăsăm totul doar pe o singură persoană .

Desigur, asta este metoda recomandată, dar se poate configura repository-ul să nu fie nevoie de approve-ul altcuiva, dar în organizații mari, așa este cea mai bună variantă pentru a minimiza potențialele erori .



Pull request

Update .gitignore #21

Open gkg513 wants to merge 1 commit into main from test

Conversation 0 Commits 1 Checks 1 Files changed 1 +1 -2

Changes from all commits File filter Conversations Jump to

0 / 1 files viewed Review changes

3 .gitignore

1	2	*.terraform
3	3	# Compiled files
4	4	*.tfstate
5	5	*.tfstate.backup
6	5	
7	6	# Decrypted data
8	3	.decrypted_userdata.tpl
9	4	
10	5	*.garn-error.log*
58	57	l.elasticsearch/*_global.yml
58	58	
60	59	# security_group_rules_script/output.json #file
61	60	- /apps/security_group_rules_script/output.json
62	61	+ /apps/security_group_rules_script/output.json

Finish your review

Write Preview H B I E C P L I @ C

Leave a comment


Attach files by dragging & dropping, selecting or pasting them.


- ☒ Comment
Submit general feedback without explicit approval
- ☐ Approve
Submit feedback and approve merging these changes
- ☐ Request changes
Submit feedback that must be addressed before merging

Submit review


Pull request

Add more commits by pushing to the **test** branch on [gxx513/work-githubaction](#).




**All checks have passed**[Show all checks](#)

1 successful check

**This branch has no conflicts with the base branch**

Merging can be performed automatically.

Merge pull request 

You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

Abia apoi se poate face merge în main cu noile modificări

Terraform branch #20

Edit <> Code

Open gkg513 wants to merge 5 commits into main from terraform_branch

+7 -4

Conversation 0 Commits 5 Checks 0 Files changed 2

Changes from all commits File filter Conversations

0 / 2 files viewed Review changes

Filter changed files

.github/workflows

plan.yml

main.tf

github/workflows/plan.yml

Viewed

```
@@ -11,7 +11,7 @@ on:
11 11 # Allows you to run this workflow manually from the Actions tab
12 12 workflow_dispatch:
13 13
14 - # A workflow run is made up of one or more jobs that can run sequentially or in parallel
14 + # A workflow run is made up of one or more jobs that can run sequentially or in parallel
15 15 jobs:
16 16 # This workflow contains a single job called "build"
17 17 terraform_plan:
@@ -36,5 +36,8 @@ jobs:
36 36 - name: Terraform Init
37 37 run: terraform init
38 38
39 - - name: Terraform plan
40 - run: terraform plan
39 + # - name: Terraform plan
40 + # run: terraform plan
41 + # - name: Setup Symfony
42 + # working-directory: ./app
43 + # run: cp .env.dev .env
```

Best practices

- 1) Mereu adaugă un mesaj explicativ când faci un commit. Acesta te va ajuta pe viitor cand vei încerca să vezi ce modificări au fost făcute la cod în acel commit și mai ales de ce .
- 2) Nu este o idee bună să faci push sau face modifica direct pe branch-ul de master
- 3) Mereu înainte să începi să lucrezi cu un repo este indicat să faci un git pull pentru a te asigura că ai ultima versiune de cod
- 4) Folosește diferite tool-uri ca VSCode unde vei putea configura plugin-uri de git. Acestea îți vor ușura munca foarte mult.

Sfârșit

