**UNIVERSITATEA DE STAT DIN MOLDOVA**

**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

**DEPARTAMENTUL DE INFORMATICĂ**

**GRĂDINARU MĂDĂLINA**

**Cloud Computing**

**Lucrare de Laborator nr.3:**

**“Bazele Git. Comenzi Git”**

Profesor: Aurelia Prepeliță, dr., conf.univ.

**CHIȘINĂU – 2024**

Cuprins

[Partea I: Crearea imaginilor și containerelor Docker. Comenzi Docker 3](#_Toc178432204)

[1.1 Instalare Docker 3](#_Toc178432205)

[1.2 Instalalre Git 4](#_Toc178432206)

[1.3 Crearea imaginilor și containerelor Docker pe baza acestor imagini 5](#_Toc178432207)

# Noțiuni de bază

**Git** este un sistem de control al versiunilor distribuit, creat pentru a gestiona proiecte software complexe și pentru a permite dezvoltatorilor să colaboreze eficient. Git permite urmărirea modificărilor în cod, revenirea la versiuni anterioare și colaborarea simultană între mai mulți dezvoltatori fără a afecta munca altora.

**GitHub** este o platformă web care utilizează Git pentru stocarea și gestionarea codului sursă în cloud. Pe GitHub, dezvoltatorii pot partaja proiecte și colabora în mod transparent. Platforme similare includ Bitbucket și GitLab.

**Sisteme de control al versiunilor**

**Sisteme locale:** Toate versiunile sunt stocate pe calculatorul unui singur utilizator. Nu este potrivit pentru colaborare între dezvoltatori, deoarece fiecare are o copie locală separată și nu există un punct centralizat de sincronizare.

**Sisteme centralizate :** Exista un server central unde sunt stocate toate versiunile fișierelor, iar toți utilizatorii trimit și primesc modificări de la acest server (ex. Subversion, CVS). Dezavantajul major este că dacă serverul cade, echipa nu mai poate lucra.

**Sisteme distribuite**: Git este un exemplu de astfel de sistem, unde fiecare dezvoltator are o copie completă a repository-ului. Fiecare copie a repository-ului este o copie completă și independentă, astfel încât colaborarea este mai robustă și mai sigură. În cazul unei defecțiuni a serverului principal, oricare dintre copiile dezvoltatorilor poate deveni un nou server.

**Stări principale**

* **Comise** (comitted) înseamnă că fișierul este deja salvat în baza de date locală -- presupune că datele sunt în siguranță în baza de date locală.
* Fișierele **modificate** includ fișiere care s-au modificat, dar nu au fost încă comise.
* Fișierele **în așteptare** sunt fișiere modificate care sunt marcate pentru a fi incluse în următoarea comitere.

**Fișierele urmărite** sunt acele fișiere care au fost în ultimul instantaneu al proiectului; pot fi neschimbate, modificate sau pregătite pentru comitere. Pe scurt, fișierele urmărite sunt acele fișiere despre care Git știe.

**Fișierele neurmărite** sunt orice altceva, orice fișiere din directorul de lucru care nu au făcut parte din ultimul instantaneu și nu sunt gata să fie comise.

**Operatii locale**

* **Directorul Git** este locația unde Git își stochează metadate și baza de date cu obiecte a proiectului dumneavoastră. Aceasta este partea cea mai importantă a Git, și reprezintă ceea ce este copiat atunci când clonați un repository de la un alt calculator.
* **Directorul de lucru** reprezintă un singur checkout al unei versiuni a proiectului. Aceste fișiere sunt preluate din baza de date comprimată din directorul Git și plasate pe discul dumneavoastră pentru a le putea modifica.
* **Zona de așteptare** (staging [en]) este un simplu fișier, de obicei conținut in directorul Git, și stochează informații despre ce va fi folosit pentru urmatorul commit. Este uneori denumit și **index**, dar devine obișnuit să i se spună zona de așteptare.

**Securitatea Git**

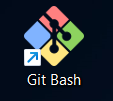
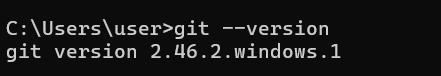
Git asigură integritatea codului sursă folosind algoritmul de criptare SHA-1. Fiecare commit este asociat cu un hash unic, protejând astfel codul împotriva modificărilor accidentale sau malițioase.

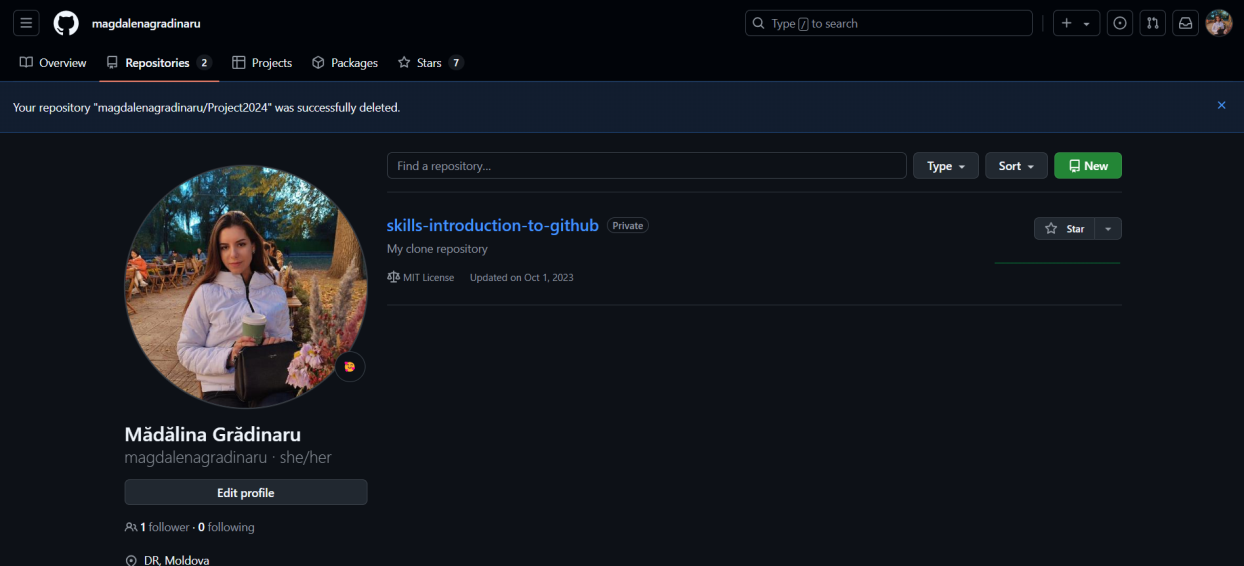
**Performanță Git**

Git este optimizat pentru a lucra cu proiecte mari și suportă operațiuni rapide de creare și comutare între ramuri, îmbinări și reconstituirea istoricului fișierelor. Nu este afectat de modificări minore, cum ar fi redenumiri de fișiere, și este eficient chiar și în echipe mari, distribuite.

# Partea I: Crearea unui cont și repozitoriu pe platforma Git Hub

## Instalalre Git

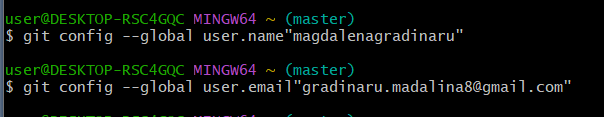




## Executarea comenzilor Git

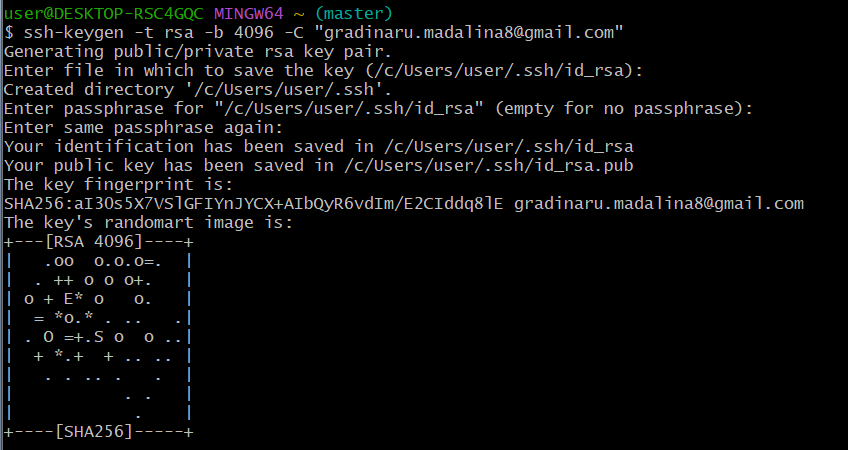
**Configurăm numele și adresa de e-mail care vor fi asociate commit-urilor noastre**

* git config –global



**Generăm o pereche de chei SSH pe sistem.**

* generarea ssh keygen



**ssh-keygen**: Este comanda care inițiază procesul de generare a unei perechi de chei SSH (una publică și una privată).

**-t rsa**: Specifică tipul cheii generate. RSA este un algoritm de criptare cu chei publice. RSA este cel mai utilizat algoritm pentru generarea cheilor SSH.

**-b 4096**: Specifică lungimea cheii în biți. 4096 este o dimensiune foarte sigură, indicând o criptare puternică.

**-C "gradinaru.madalina8@gmail.com":** Aceasta adaugă un comentariu (în acest caz, adresa ta de e-mail) pentru a identifica cheia, în mod obișnuit utilizată pentru conturi de GitHub sau alte platforme.

**Proces:**

1. Sistemul îți oferă opțiunea de a salva cheia într-o locație specifică. În mod implicit, cheia privată este salvată în fișierul id\_rsa, iar cheia publică în id\_rsa.pub, în folderul ascuns .ssh din directorul tău home.

C:\Users\user\AppData\Local\Packages\MicrosoftWindows.Client.CBS_cw5n1h2txyewy\TempState\ScreenClip\{9D3B2993-7511-4CFF-B122-2F6DE9CEF99C}.png

1. Utilizatorul este întrebat dacă vrea să adauge o parolă pentru a proteja cheia privată, ceea ce oferă un strat suplimentar de securitate, însă am ales opțiunea de a lăsa acest câmp gol pentru a nu folosi o parolă.

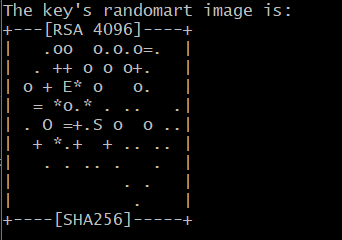
C:\Users\user\AppData\Local\Packages\MicrosoftWindows.Client.CBS_cw5n1h2txyewy\TempState\ScreenClip\{F9FBEAD2-6442-4633-8376-4177DAB60F28}.png

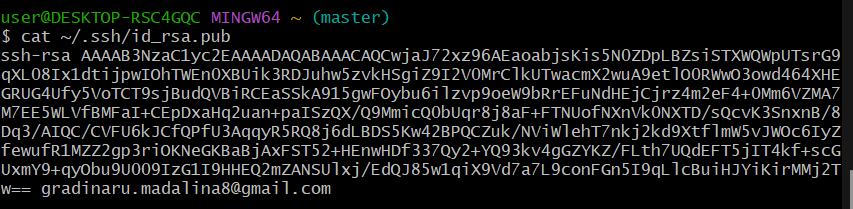
1. SHA256:aI30s5X7VSlGFIYnJYCX+AIbQyR6vdIm/E2CIddq8lE – Este o amprentă unică a cheii mele SSH. Fingerprint-ul ajută să verific că utilizez cheia corectă atunci când o compar cu ceea ce am încărcat pe server.

C:\Users\user\AppData\Local\Packages\MicrosoftWindows.Client.CBS_cw5n1h2txyewy\TempState\ScreenClip\{95385499-F14C-418D-B304-9297F48868D7}.png



1. **Randomart Image**: Imaginea este o reprezentare grafică unică a cheii SSH. Deși nu are o valoare practică de securitate, oferă o metodă vizuală de a verifica cheia generată.



După generarea cheii SSH, trebuie să adaug cheia publică pe platforma GitHub pentru a permite autentificarea, astfel, afișăm conținutul fisierului respectiv  


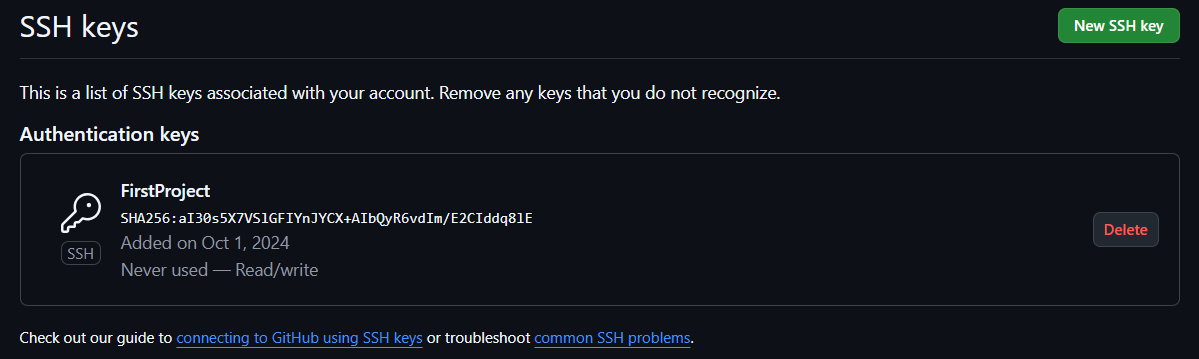
Urmăm pașii următori pentru a crea cheia ssh in interfața grafică ssh și a asigura conexiunea

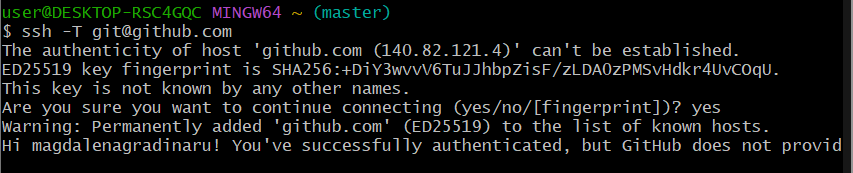
 Accesăm [GitHub](https://github.com), ne autentificăm și accesăm **Settings**.

 În secțiunea **SSH and GPG keys**, facem clic pe **New SSH Key**.

 În câmpul „Title”, dăm un nume descriptiv cheii, adică FirstProject, apoi lipim cheia publică copiată în câmpul „Key”.

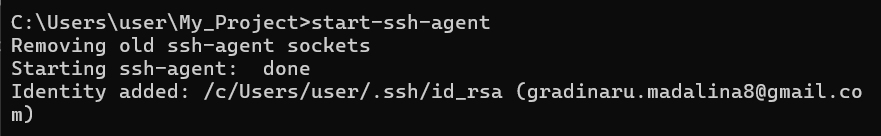
 Apăsăm **Add SSH key**.

Rezultatul obținut  


Verificăm conexiunea  


*Hi magdalenagradinaru! You've successfully authenticated, but GitHub does not provide shell access.*

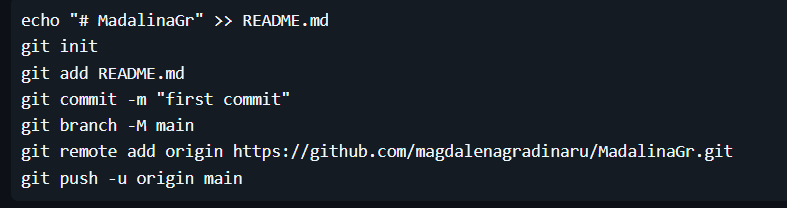
**Adaugam local cheia in ssh-agent, cu comanda**

****

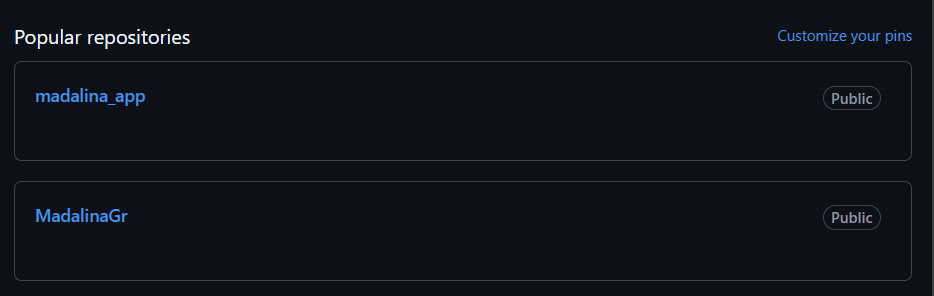
**Inițializăm un nou repository Git în directorul curent**

* git init

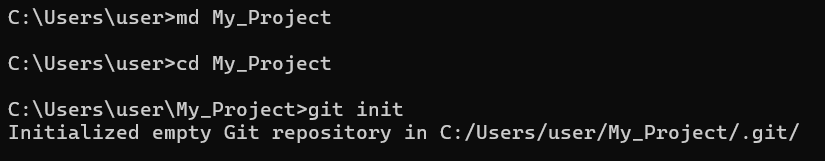
1. creează un repository Git gol în acest director, care va permite urmărirea versiunilor fișierelor din acest folder. Directorul .git este creat automat și conține toate fișierele necesare pentru urmărirea modificărilor, istoricul commit-urilor, etc.

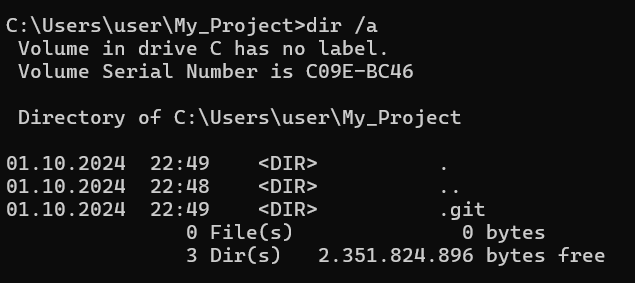


- comenzile pentru crearea unui repozitoriu din cmd (linia de comanda)



-in rezultat, repozitoriul MadalinaGr s-a creat.



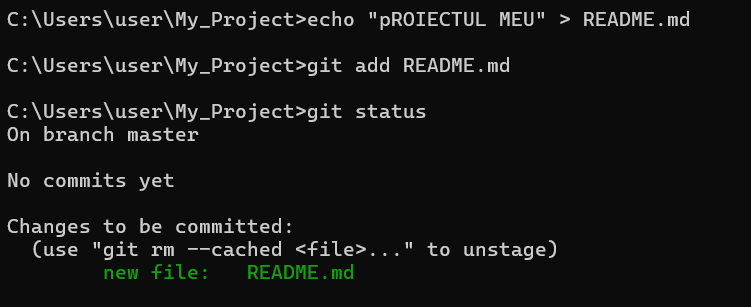


-Creăm un director din ierarhia de fișiere în care vom inițializa repository-ul Git

Am creat până în acest moment un repository local și unul remote. Trebuie să le interconectăm.



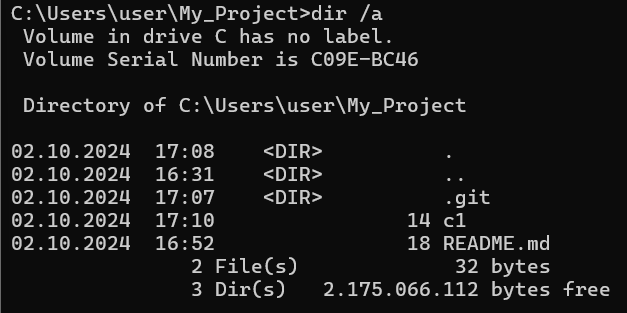
Inițial, vedem că avem un director de lucru curat, cu alte cuvinte — nu există fișiere modificate urmărite în el. De asemenea, Git nu a găsit niciun fișier neurmărit, altfel ar fi fost listate aici. În cele din urmă, comanda vă spune în ce ramură ne aflăm și vă spune că este în concordanță cu ramura de pe server.



- adăugăm la proiect un fișier nou “add README”

-Putem vedea că fișierul este indexat deoarece se află în secțiunea „Modificări care urmează să fie comise”.

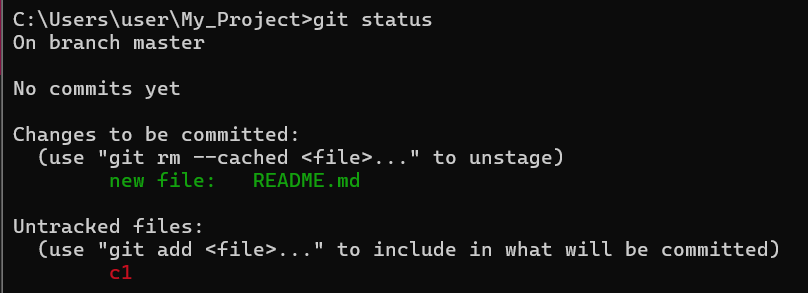
Afișăm lista fișierelor și directoarelor din directorul curent, inclusiv cele ascunse



-Se afișează două fișiere: README.md și c1, precum și directorul .git, care conține metadatele repository-ului Git.

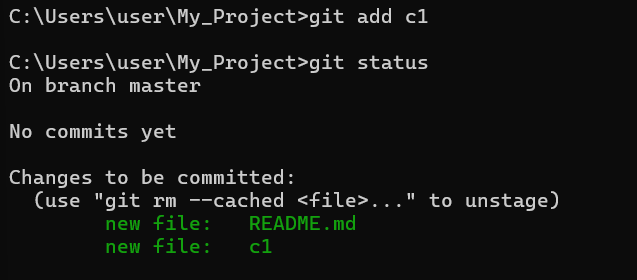
Afișează starea repository-ului

* git status

 -Fișierul README.md este pregătit pentru commit, iar c1 este un fișier neurmărit.

* git add

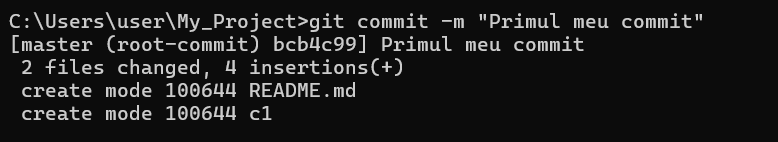
Adaugă fișierul c1 la stagiul de commit (staging area), ceea ce înseamnă că fișierul va fi inclus în următorul commit.



**Git commit**

calculează sumele de control ale fiecărui subdirector (în cazul nostru, doar directorul principal al proiectului) și îl stochează în depozit ca obiect -- **arbore de directoroare**. Git creează apoi un obiect commit cu metadate și un pointer către arborele principal al proiectului, astfel încât să poată fi recreat dacă este necesar.

* git commit -a -m

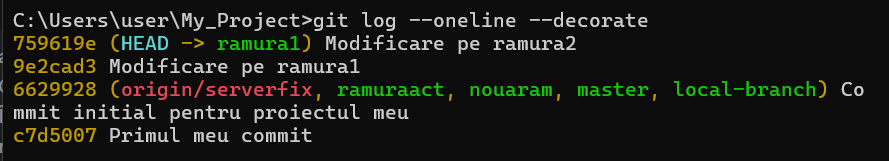


**[master (root-commit) bcb4c99]**: Aceasta este ramura pe care te afli (master), iar commit-ul realizat este primul din acest repository (denumit „root-commit”). Hash-ul commit-ului este bcb4c99, un identificator unic pentru acest commit.

**2 files changed, 4 insertions(+)**: Indică faptul că două fișiere au fost modificate (sau create) și au fost adăugate patru linii noi în total în aceste fișiere.

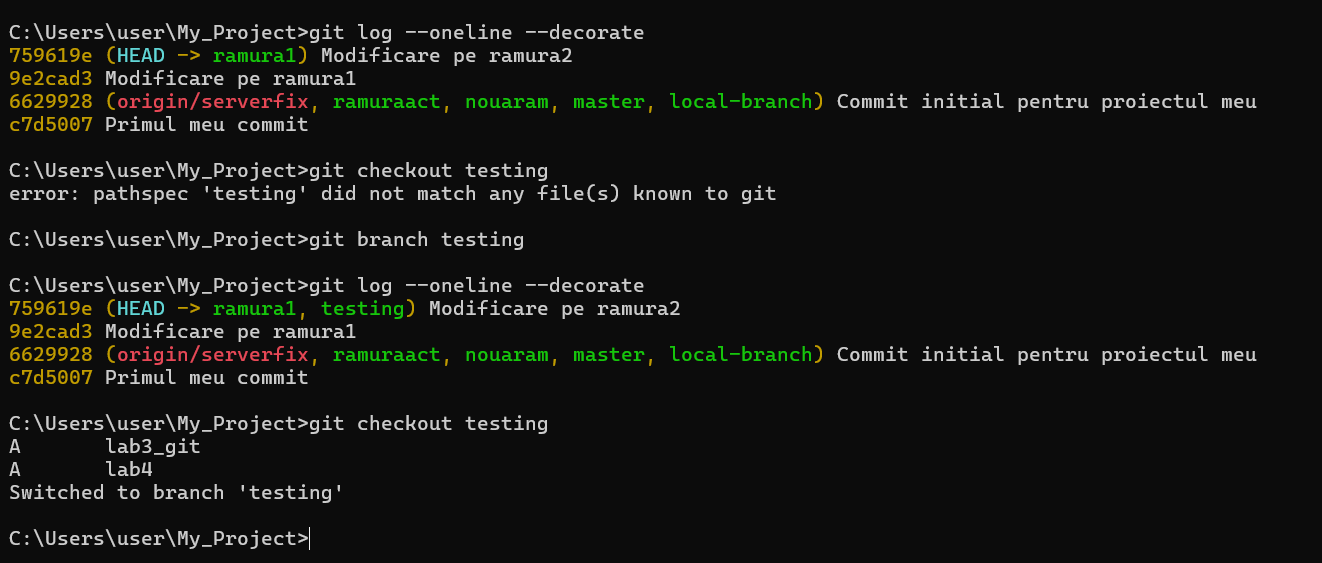
**create mode 100644**: Acest mesaj indică faptul că două fișiere noi au fost create (README.md și c1) și au permisiuni de acces standard pentru un fișier (644, adică fișierul poate fi citit și scris de către proprietar și doar citit de alții).

Cum determină Git ramura în care ne aflăm?  
Stochează un indicator HEAD special. Comanda simplă **git log**, care vă va arăta unde indică indicatorii de ramuri. Această opțiune se numește **--decorate**.

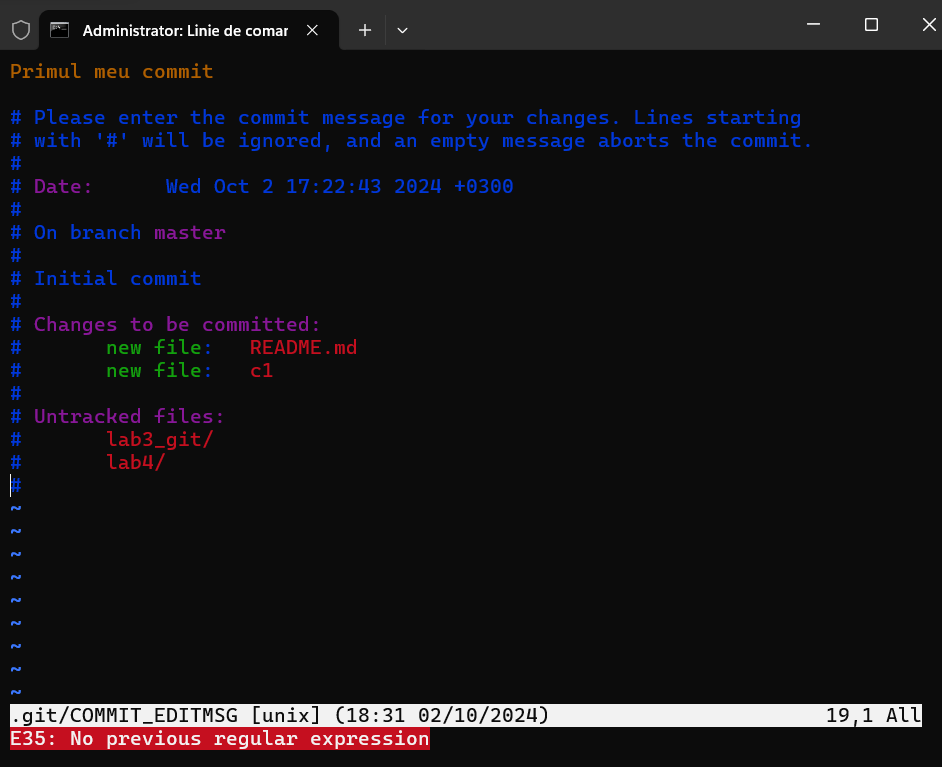
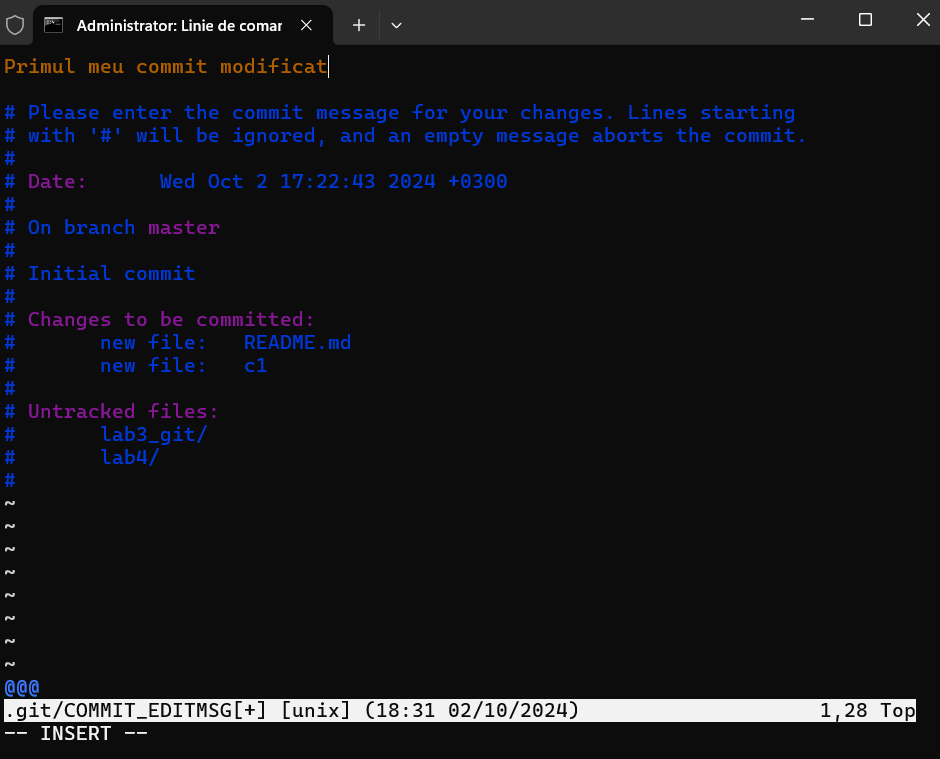


Pentru a trece la o ramură de testare anterior creată, am utilizat **git checkout.**

Ca rezultat indicatorul HEAD se va muta pe ramura de testare.

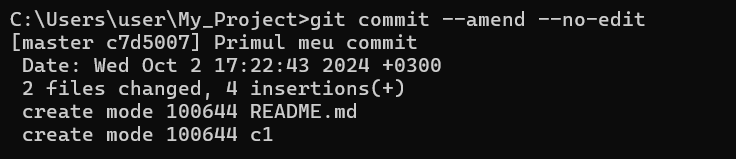


* + Git commit –amend

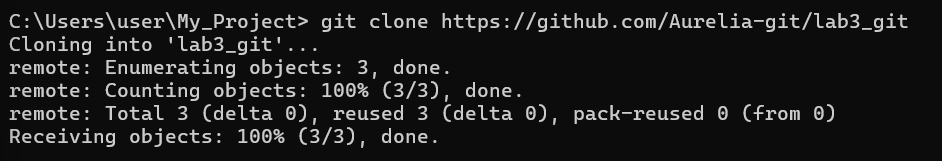
 

- Această comandă deschide editorul de text (de obicei, editorul implicit configurat pentru Git) pentru a permite utilizatorului să modifice mesajul ultimului commit.

* + Git commit –amend –no-edit

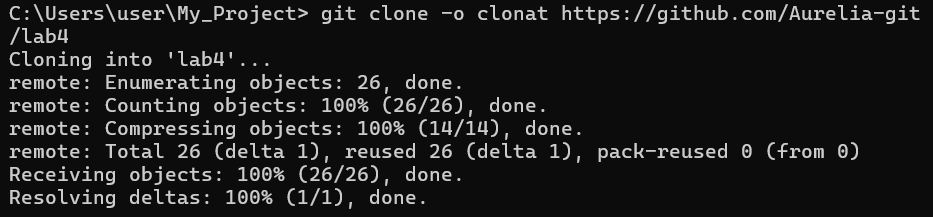


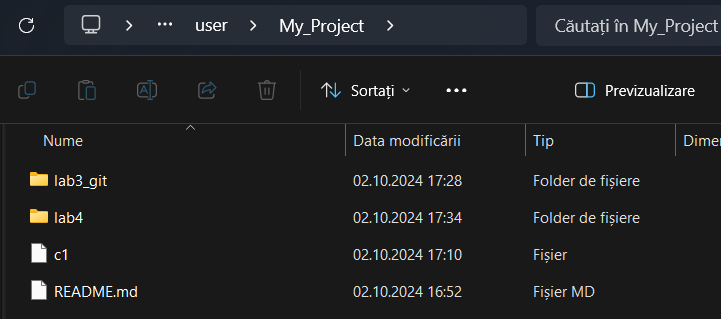
* git clone - creează o copie a codului și a întregului istoric de versiuni al repository-ului în directorul specificat.

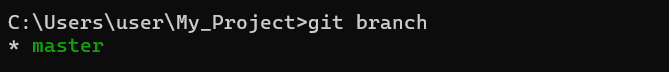


- Comanda git clone https://github.com/Aurelia-git/lab3\_git a descărcat o copie a repository-ului remote în directorul local lab3\_git. Git a clonat toate fișierele și istoria repository-ului, pregătindu-le pentru utilizare locală.

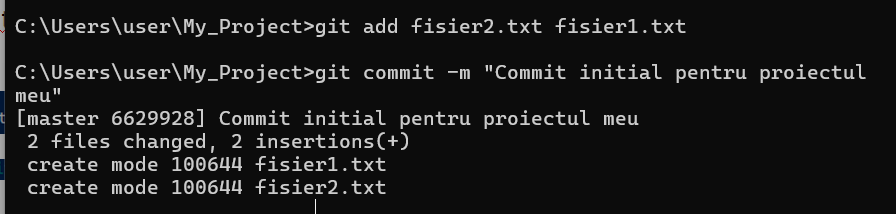
* + git clone -o first - este utilizată pentru a clona un repository Git și a-i atribui un nume personalizat pentru remote, în loc de numele implicit origin

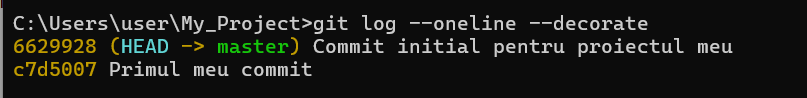




* git branch

- afișează lista tuturor ramurilor din repository-ul curent. Ramura activă va fi marcată cu un asterisc (\*).



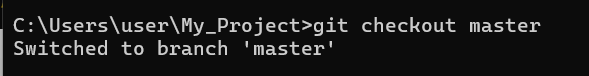


* + git branch -d testing/ git branch -D testing (fortat)

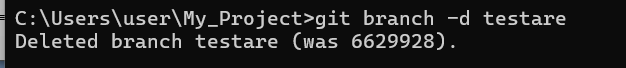
Am creat un nou branch

C:\Users\user\AppData\Local\Packages\MicrosoftWindows.Client.CBS_cw5n1h2txyewy\TempState\ScreenClip\{67FD3BBA-8EDE-417C-A88D-BDFA861967E2}.png

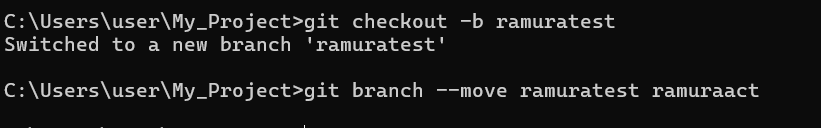
Am comutat pe aceasta ramura master pentru a putea sterge ramura testare



Am sters

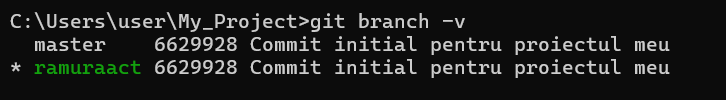


* + git branch --move bad-branch-name corrected-branch-name



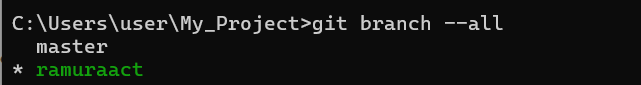
-am utilizat comanda pentru a schimba numele unei ramuri existente. ramuratest este numele vechi al ramurii, iar ramuraact este noul nume

* + git branch –v



- Afișează o listă de ramuri locale cu ultimele commit-uri asociate fiecărei ramuri.

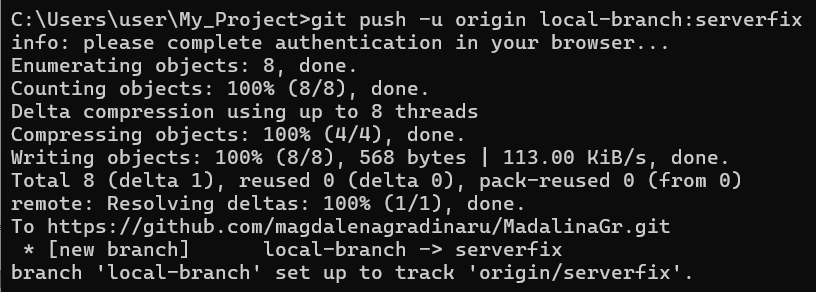
* + git branch –all



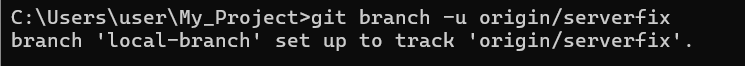
- listează toate ramurile, atât locale, cât și remote. Este utilă pentru a obține o vedere de ansamblu asupra ramurilor disponibile.

* + git branch -u origin/serverfix

-are rolul de a seta ramura curenta sa urmareasca ramura remote specificata.

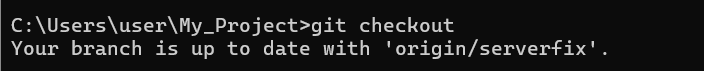


- Această comandă împinge ramura local-branch către repository-ul remote origin, creând o nouă ramură numită serverfix.



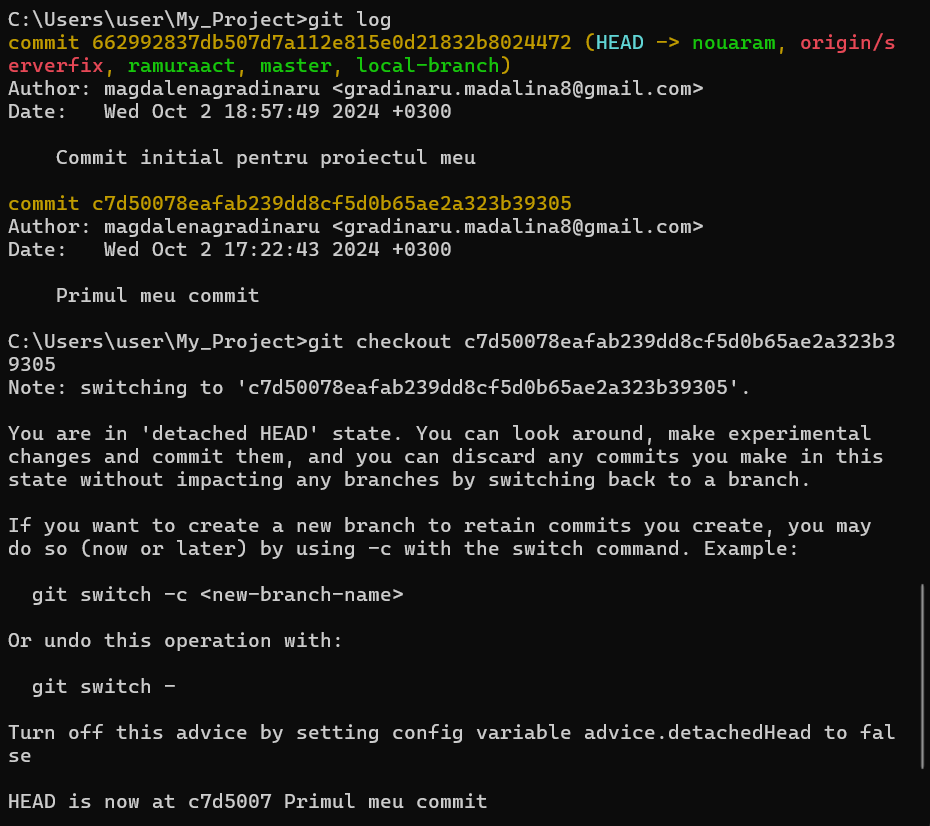
- confirmă că ramura local-branch este acum setată să urmărească ramura remote origin/serverfix.

* git checkout



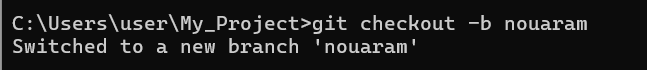
- folosind comanda fara parametri, va afisa informatii despre ramura curenta.

* + git checkout 163b3891d66a75baa4f7adcd7802a098aa3ffc4e



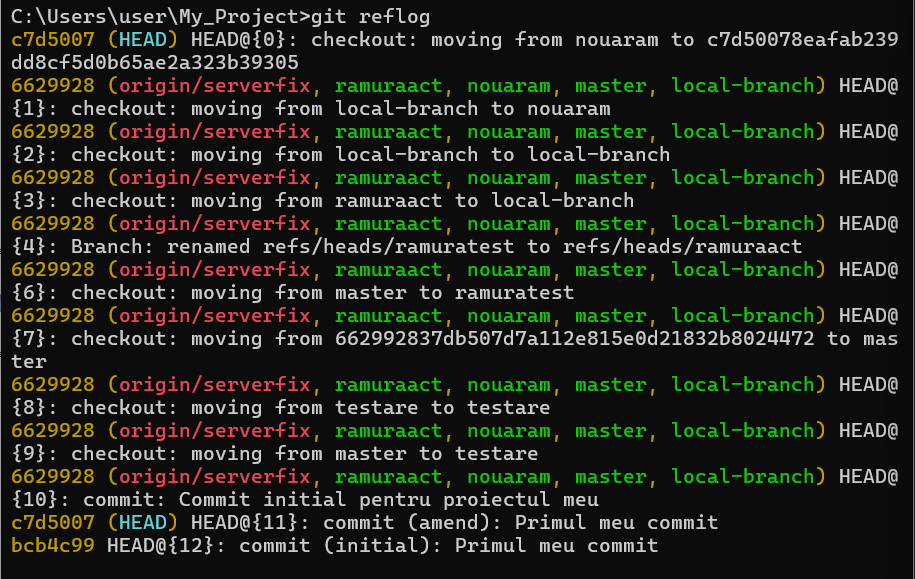
- După rularea acestei comenzi, Git va muta pointer-ul HEAD la acel commit, ceea ce înseamnă că te afli într-o stare „detached HEAD”. Aceasta înseamnă că nu ești pe o ramură activă, ci direct pe un commit. Modificările ulterioare nu vor fi legate de o ramură existentă, deci este important să creezi o ramură nouă dacă dorești să salvezi modificările.

* + git checkout -b <newbranchname>



- Git va crea ramura specificată, va muta HEAD pe acea ramură nouă și va permite lucrul direct pe ea. Este utilă pentru dezvoltarea de caracteristici noi sau pentru experimentare, fără a afecta ramura principală.

* Git reflog



**Identificatorul Commit-ului:** Fiecare intrare conține un hash unic (ex: c7d5007) pentru commit-uri.

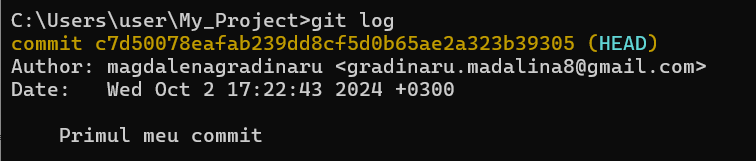
**Starea HEAD:** Arată poziția curentă a HEAD, indicând ramura activă și commit-urile recente.

**Indexul Acțiunii:** Fiecare acțiune are un număr (ex: HEAD@{0}), unde 0 este cea mai recentă.

**Tipul de Acțiune:** Descrie acțiuni precum checkout (schimbarea ramurii) și commit (salvarea modificărilor).

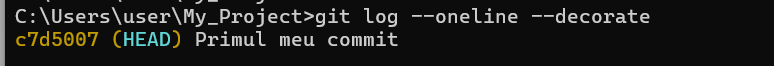
**Ramuri Implicate:** Indică ramurile asociate cu fiecare commit sau acțiune.

* Git log



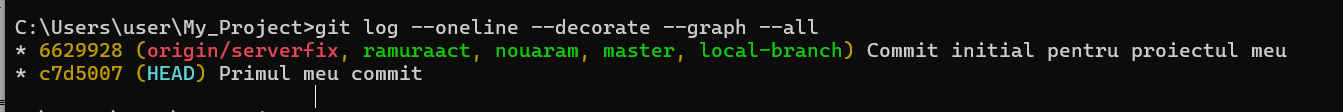
- Aceste informații oferă detalii despre cine a făcut modificarea, când a fost făcută și o descriere a acesteia.

* + git log [--oneline --decorate]



- Afișează o listă compactă a commit-urilor, arătând ramurile și tag-urile.

* + git log --oneline --decorate --graph –all



Oferă o vedere de ansamblu vizuală a întregii istorii a commit-urilor din repository, inclusiv ramuri și îmbinări.

* git reset

- Această comandă se folosește pentru a reseta indexul (staging area) și, opțional, și directorul de lucru la starea anterioară a unui commit. Comanda poate modifica sau păstra modificările neadăugate în directorul de lucru, în funcție de opțiunile specificate.

* + git reset --hard <commit-hash>

- Această comandă resetează atât indexul, cât și directorul de lucru la starea specificată de commit-ul indicat prin <commit-hash>. Toate modificările din directorul de lucru care nu au fost comise vor fi pierdute

* + git reset --hard HEAD@{n}

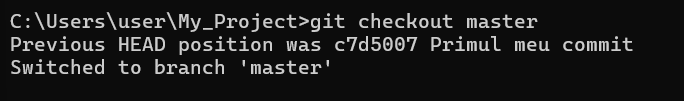
- Această comandă resetează indexul și directorul de lucru la starea unui commit specificat prin referința HEAD@{n}, unde n reprezintă un număr care se referă la o stare anterioară a HEAD-ului din reflog. La fel ca în cazul precedent, toate modificările necomise vor fi pierdute.

* git rm

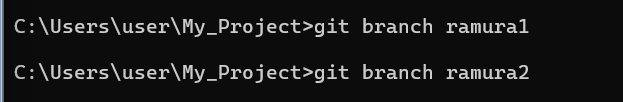
- este utilizată pentru a șterge fișiere din directorul de lucru și din indexul (staging area) Git.

* git merge

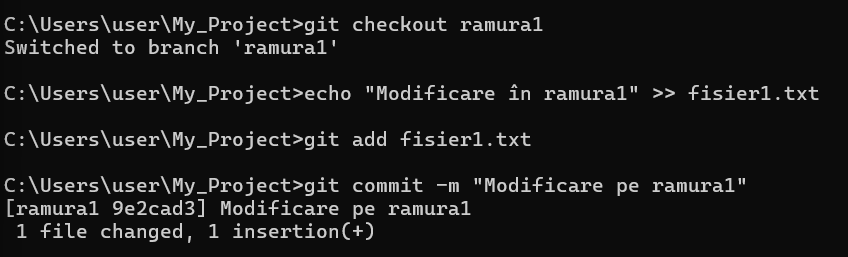
1. Ne asigurăm că suntem pe ramura principală



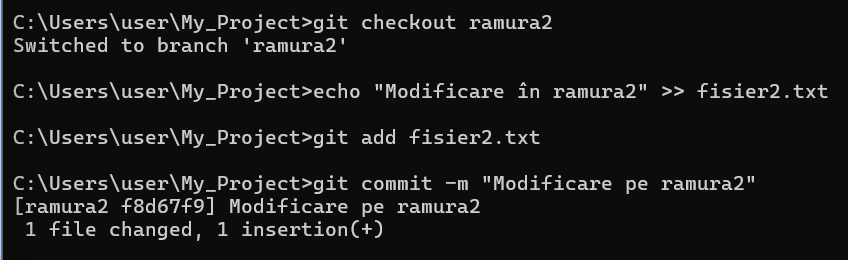
1. Creăm prima si a doua ramură



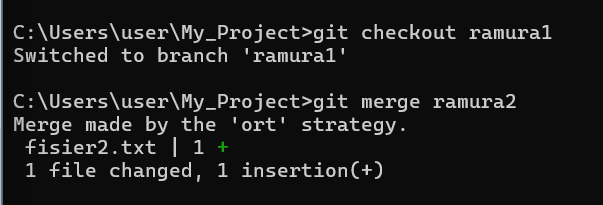
1. Ne mutam pe prima ramura. Facem o modificare într-un fișier. Adăugăm modificarea și facem commit.



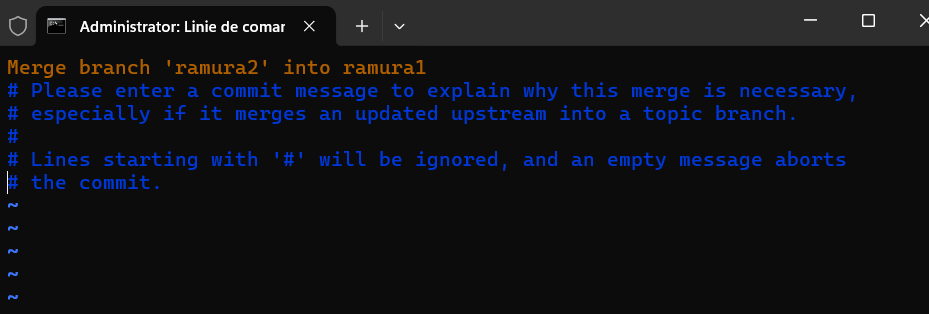
1. Ne mutam pe ramura 2 si facem acelasi lucru.



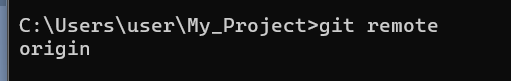
1. Ne intoarecm pe ramura 1 si aplicam merge pentru a fuziona ramura 1 in ramura 2



- Comanda git merge ramura2 a finalizat un merge utilizând strategia 'ort', adăugând o linie în fișierul fisier2.txt, cu o modificare în acest fișier.

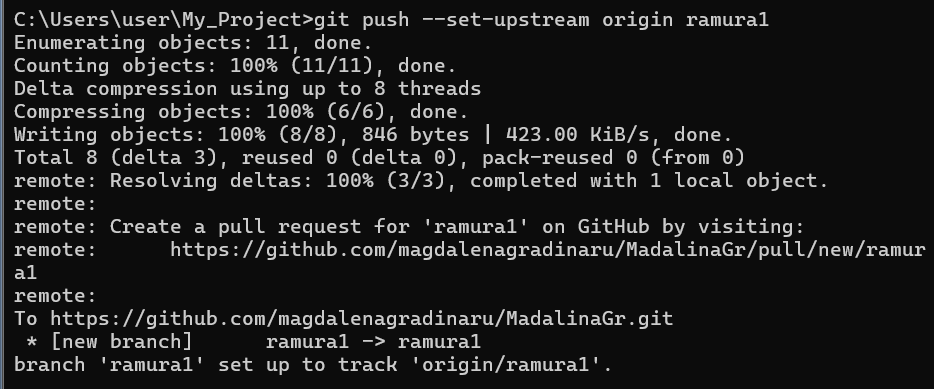


* git remote

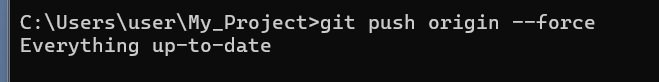


- s-a afișat un singur remote configurat, denumit origin. Acesta este numele implicit dat repository-ului remote atunci când am clonat proiectul. origin reprezintă legătura dintre repository-ul meu local și cel de pe serverul GitHub.

**git push**

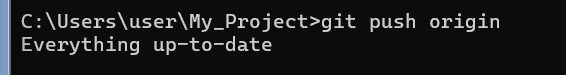
a creat ramura ramura1 pe remote (GitHub) și a setat-o pentru a urmări ramura locala****

* + git push origin –force (after rebase)



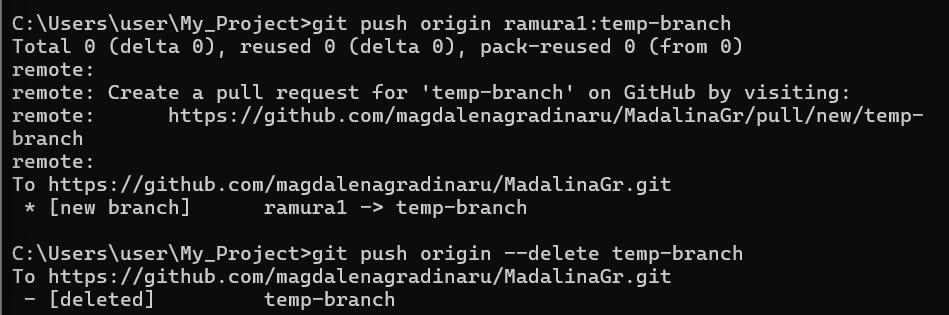
- Această comandă a forțat trimiterea modificărilor către remote, dar a indicat că totul este deja actualizat.

* + git push origin



- A trimis modificările către remote, dar a confirmat că totul este deja actualizat, ceea ce înseamnă că nu erau modificări noi de trimis.

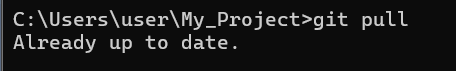
* + git push origin --delete temp-branch



- A trimis ramura ramura1 pe remote și a creat o nouă ramură numită temp-branch.

- Această comandă a șters cu succes ramura temp-branch de pe remote.

* git pull



- aduce modificările de la un repository remote și le integreaza în ramura curentă din repository-ul local.

* git stash



- este utilizată pentru a salva temporar modificările necomise (uncommitted changes) dintr-o ramură de lucru, astfel încât să poți reveni la un stadiu curat al codului. Aceasta este utilă în situațiile în care vrei să schimbi ramurile sau să realizezi o operațiune care necesită un spațiu de lucru curat, fără a pierde lucrările curente.

* git fetch
  + git fetch origin

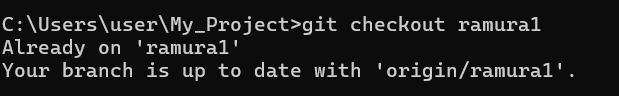


- descarcă actualizările de la remote-ul specificat.

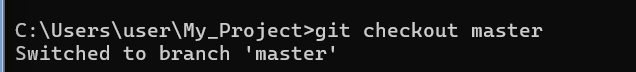
* git rebase

este un instrument esențial în Git care permite mutarea sau combinarea unui set de commit-uri de pe o ramură pe alta.

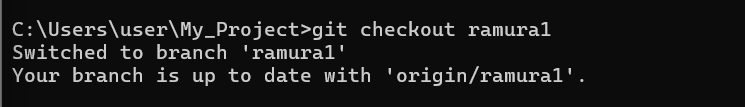
1. **Navighează la ramura pe care dorești să o rebasezi**:



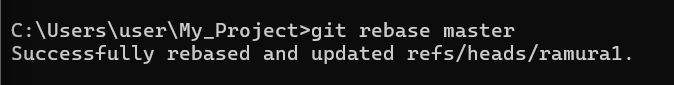
1. **Asigură-te că ramura de bază (master) este actualizată**:



1. **Revenire la ramura pe care dorești să o rebasezi**:



1. **Aplică rebase-ul**:



-Git va lua toate commit-urile din ramura ramura1 și le va aplica pe ramura master, ca și cum ar fi fost făcute direct pe acea ramură.

* git remote add origin

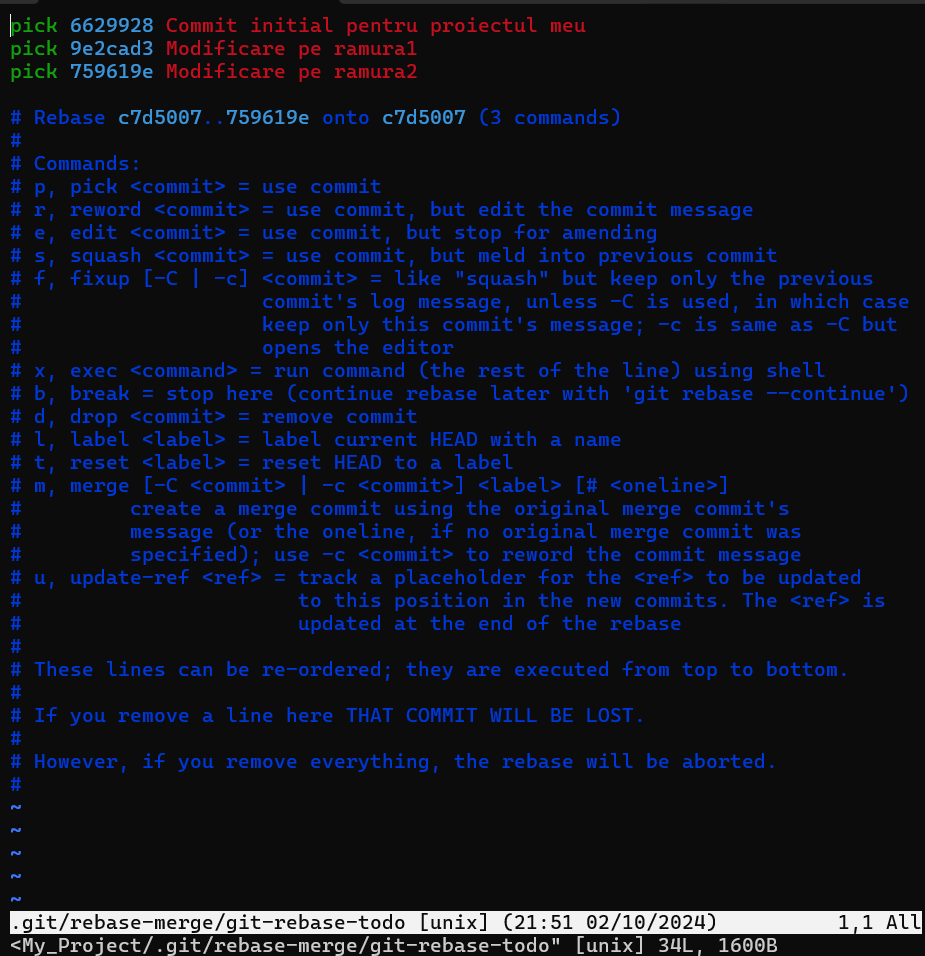
- adăuga un nou remote repository la un proiect Git local, asociindu-l cu numele „origin”. Asta a fost realizat in pasii initiali.



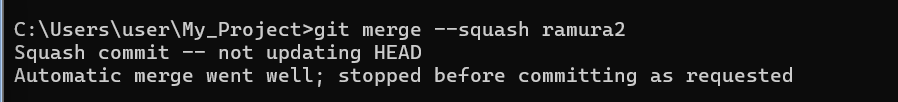
* git squash
  + git rebase -i HEAD~3

- git rebase -i HEAD~3 este utilizată pe

ntru a face un rebase interactiv pe ultimele 3 commit-uri din ramura curentă



* + git merge –squash



Git a combinat modificările din ramura ramura2 în ramura curentă (pe care ești) fără a crea imediat un commit.