FACULTATEA MATEMATICĂ ȘI INFORMATICĂ
DEPARTAMENTUL INFORMATICĂ

CAZACU ANASTASIA, IA 2301

LUCRAREA DE LABORATOR NR. 3

TEMA: “Bazele Git. Comenzi Git.”

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

Autor :Anastasia Cazacu, gr. IA2301

CHIȘINĂU – 2025

Cuprinsul:

Cuprinsul:..... 2
Bazele Git. Comenzi Git..... 5
Instalarea Git. Creați un cont și un repozitoriu pe Git Hub..... 5
Raspuns:..... 5
1. FROM..... 5
Anexa 2 imaginile..... 5
Concluzia:..... 6
Bibliografie..... 7

Bazele Git. Comenzi Git.

Instalarea Git. Creați un cont și un repozitoriu pe Git Hub.

- Accesez https://git-scm.com/downloads
- Descarcă versiunea pentru Windows și rulează installer-ul
- Verific instalarea cu:

git --version

```
PS D:\Anastasia\Study\USM\Anul3\Semestru1\CloudComp\Lab2\exemplu1\cloudlab2> git --version
git version 2.45.2.windows.1
PS D:\Anastasia\Study\USM\Anul3\Semestru1\CloudComp\Lab2\exemplu1\cloudlab2>
```

Creem un cont pe git:
Accesează https://github.com Creează un cont nou (username, email, parolă)

- click pe **New repository**
- Completează:
  - Nume: `lab3-git`
  - Descriere: „Repo pentru laboratorul 3”
  - Opțiuni: Public/Private, cu README
- Click pe **Create repository**

Executați următoarele comenzi Git:

git config --global

Configurează numele și emailul pentru commit-uri:

git config --global user.name "anastasiaCazacu" git config --global user.email "anastasia.cazacu@gmail.com"

```
PS C:\Users\anast> git config --global user.name "Anastasia"
```

Setează numele autorului pentru toate commit-urile

git config --global user.email “anastasia.cazacu@gmail.com” - configureaza emailul

Generarea cheii SSH

Permite autentificarea securizată cu GitHub:

ssh-keygen -t ed25519 -C "anastasia.cazacu@gmail.com"

- Cheia va fi salvată în `~/.ssh/id_ed25519`
- Afișează cheia publică:

cat ~/.ssh/id_ed25519.pub

- Adaug cheia în GitHub:
  GitHub → Settings → SSH and GPG keys → New SSH key

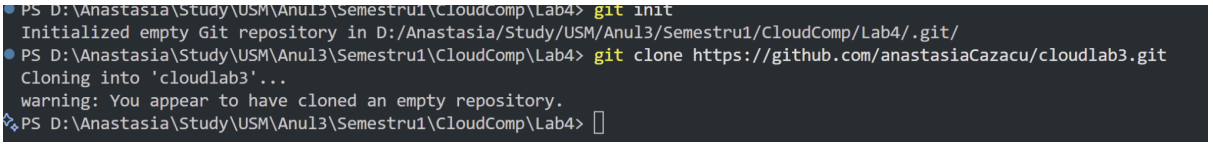
`git init`

Inițializează un repository local:

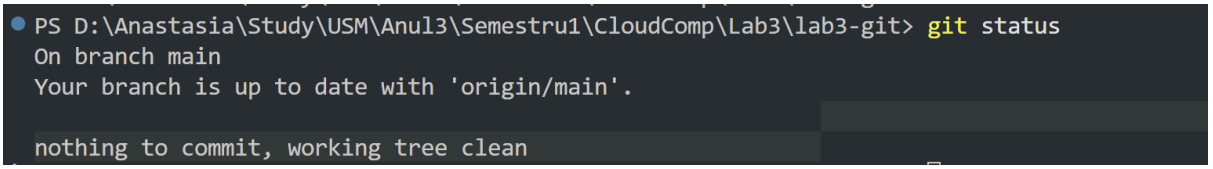
git init

git clone

Clonează un repository existent de pe GitHub:



git clone https://github.com/anastasiaCazacu/cloudlab3.git

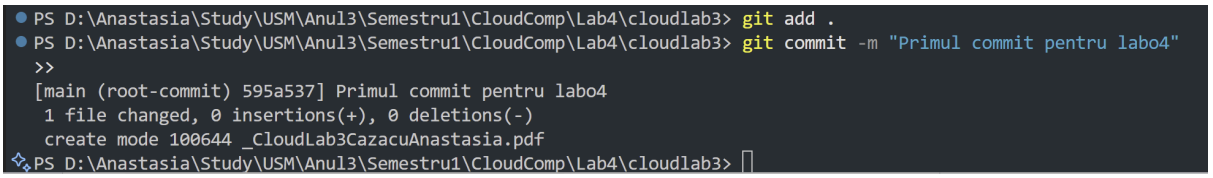


Am clonat repository-ul creat pe GitHub

git status

Am verificat ce fișiere sunt modificate sau neadăugate

PS D:\Anastasia\Study\USM\Anul3\Semestru1\CloudComp\Lab4\cloudlab3> git add .
PS D:\Anastasia\Study\USM\Anul3\Semestru1\CloudComp\Lab4\cloudlab3> git commit -m "Primul commit pentru labo4"



Am reușit să fac commit-ul inițial în subfolderul `cloudlab3`, ceea ce înseamnă că acel folder este acum un repository Git separat.

Comanda `git commit -m` înseamnă că salvez modificările în Git și le însoțesc cu un mesaj explicativ.

`git commit --amend -m “”`

Când folosesc `git commit --amend -m`, modific ultimul commit pe care l-am făcut. Pot să **schimb fie mesajul, fie să adaug fișiere uitate**

```
PS D:\Anastasia\Study\USM\Anul3\Semestru1\CloudComp\Lab4\cloudlab3> git commit --amend -m "Primul Comentariu la lab
4"
[main c528d09] Primul Comentariu la lab4
 Date: Sun Oct 19 00:32:19 2025 +0300
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 _CloudLab3CazacuAnastasia.pdf
PS D:\Anastasia\Study\USM\Anul3\Semestru1\CloudComp\Lab4\cloudlab3>
```

Să zicem că am uitat să adaug un fișier

```
PS D:\Anastasia\Study\USM\Anul3\Semestru1\CloudComp\Lab4\cloudlab3> git add _CloudLab3CazacuAnastasia.pdf
PS D:\Anastasia\Study\USM\Anul3\Semestru1\CloudComp\Lab4\cloudlab3> git commit --amend
[main 74d874b] Primul Comentariu la lab4
 Date: Sun Oct 19 00:32:19 2025 +0300
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 _CloudLab3CazacuAnastasia.pdf
PS D:\Anastasia\Study\USM\Anul3\Semestru1\CloudComp\Lab4\cloudlab3>
```

Astfel eu fac: Git deschide editorul și îmi permite să modific mesajul commitului
Daca nu vreau sa modific mesajul, doar sa adaug fisierul folosesc:
git commit --amend --no-edit

git status

demonstrez ca ma aflu pe branch-ul main

```
PS D:\Anastasia\Study\USM\Anul3\Semestru1\CloudComp\Lab4\cloudlab3> git status
On branch main
Your branch is based on 'origin/main', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

nothing to commit, working tree clean
PS D:\Anastasia\Study\USM\Anul3\Semestru1\CloudComp\Lab4\cloudlab3>
```

Tipuri principale de `git merge`

1. Fast-forward merge

- **Ce este**: Git „avansează” direct pointerul branch-ului curent, fără a crea un commit de îmbinare.
- **Când apare**: Dacă branch-ul curent nu are commituri proprii și poate fi „aliniat” cu cel de îmbinat.
- **Comandă**: **git merge feature-branch**

2. Non-fast-forward merge

- **Ce este**: Git creează un commit de îmbinare, chiar dacă ar putea face fast-forward.
- **Când apare**: Când se folosește explicit `--no-ff` pentru a păstra clar istoricul.
- **Comandă**:

git merge --no-ff feature-branch

3. Îmbinare conflictuală (three-way merge)

- **Ce este**: Git nu poate decide automat cum să îmbine fișierele modificate în ambele branch-uri.
- **Când apare**: Dacă același fișier a fost modificat diferit în ambele branch-uri.
- **Rezolvare**: Editez manual fișierul, apoi:

git add nume_fisier git commit

4. Anularea îmbinării

- **Ce este**: Dacă apare un conflict și nu vreau să continui, pot anula procesul de merge.
- **Comandă**:

git merge --abort

5. Squash merge

- **Ce este**: Git combină toate commit-urile din branch-ul îmbinat într-un singur commit.
- **Când apare**: Dacă vreau un istoric curat, fără commituri intermediare.
- **Comandă**:

git merge --squash feature-branch git commit -m "Squash merge: toate modificările într-un singur commit"

FROM

Raspuns:

Anexa 2 imaginile

Concluzia:

În cadrul acestei teme, am explorat în profunzime ecosistemul Docker, de la conceptele fundamentale până la aplicații practice. Am înțeles cum imaginile Docker sunt construite folosind instrucțiuni precum FROM, RUN, CMD, COPY, WORKDIR și altele, fiecare contribuind la definirea mediului de execuție al containerului. Prin exemple concrete, am învățat să construim imagini personalizate, să le rulăm local și să le gestionăm eficient cu ajutorul comenzilor client precum docker build, docker run, docker ps, docker stop, docker rm, și docker exec.
Un accent important a fost pus pe persistența datelor prin utilizarea volumelor Docker. Am creat volume, le-am montat în containere și am verificat că datele rămân intacte chiar și după oprirea sau ștergerea containerelor. Acest mecanism este esențial pentru aplicații care necesită stocare durabilă, precum baze de date, fișiere de configurare sau loguri. Am demonstrat și cum se pot copia fișiere între sistemul gazdă și container, consolidând înțelegerea interacțiunii între mediile izolate.
În final, am publicat imagini pe Docker Hub, facilitând distribuirea și reutilizarea aplicațiilor containerizate. Întregul parcurs a evidențiat avantajele Docker în dezvoltarea modulară, portabilă și scalabilă a aplicațiilor. Tema a oferit o bază solidă pentru înțelegerea containerizării, pregătind terenul pentru integrarea cu orchestratori precum Kubernetes și pentru dezvoltarea de arhitecturi moderne bazate pe microservicii.

Bibliografie

1. Kubernetes Authors. (n.d.). Kubernetes Documentation. Cloud Native Computing Foundation. Retrieved September 15, 2025, from https://kubernetes.io/docs/home/
2. GitHub - kubernetes/kubernetes: Production-Grade Container Scheduling and Management
   Repozitoriul oficial al codului sursă Kubernetes. Include versiuni, issue-uri și contribuții comunitare.
3. Cloud Native Computing Foundation
   Organizația care susține Kubernetes și alte proiecte cloud-native. Oferă resurse, certificări și evenimente.
4. Kubernetes Documentation
   Secțiunea oficială de documentație, găzduită de Cloud Native Computing Foundation (CNCF). Include concepte, exemple și referințe detaliate.
5. **kubernetes.io**
   Site-ul oficial al proiectului Kubernetes, cu documentație completă, tutoriale, referințe API și ghiduri de instalare.