Git și GitHub - Ghid Complet pentru Echipă

Creat de echipa Mirfak-PTH-27-11-2025 pentru a aduce toată lumea la același nivel

---

📍 PARTEA 1: CE SUNT GIT ȘI GITHUB?

1.1 Diferența fundamentală

Termen	Ce este	Analogie
Git	Tool-ul (comenzile pe care le scrii în terminal)	Aplicația din telefon pentru poze
GitHub / GitLab	Platforma (cloud provider unde stochezi)	Google Photos / iCloud

Imaginează-ți că ai poze pe telefon:
- **Git** = aplicația care face pozele și le organizează
- **GitHub** = cloud-ul unde urci pozele ca să nu le pierzi

Poți folosi Git fără GitHub (lucrezi doar local), dar GitHub fără Git nu prea are sens.

1.2 De ce avem nevoie de Git?

Problemele FĂRĂ Git:
- Salvezi 10 versiuni: `proiect_final.py`, `proiect_final_v2.py`, `proiect_CHIAR_final.py` 😅
- Nu știi ce ai schimbat între versiuni
- Dacă lucrezi în echipă, vă încurcați reciproc
- Pierzi totul dacă ți se strică calculatorul

Cu Git:
- O singură versiune cu tot istoricul salvat
- Vezi exact ce ai schimbat și când

- Echipa lucrează în paralel fără conflicte
- Totul e backup-uit în cloud (GitHub)

📍 PARTEA 2: STRUCTURA UNUI PROIECT GIT

2.1 Repository (Repo)

Ce este: Containerul principal care conține tot proiectul tău.

Analogie: Imaginează-ți un **SPITAL** 🏥:
- Repository = spitalul întreg (clădirea cu tot ce conține)
- Aici se află tot codul, toate fișierele, tot istoricul

```
📦 Repo-ul meu
├── 📁 src/
├── 📁 docs/
├── 📄 README.md
└── 📄 .gitignore
```

2.2 Main (sau Master)

Ce este: Branch-ul principal, codul "oficial" care funcționează.

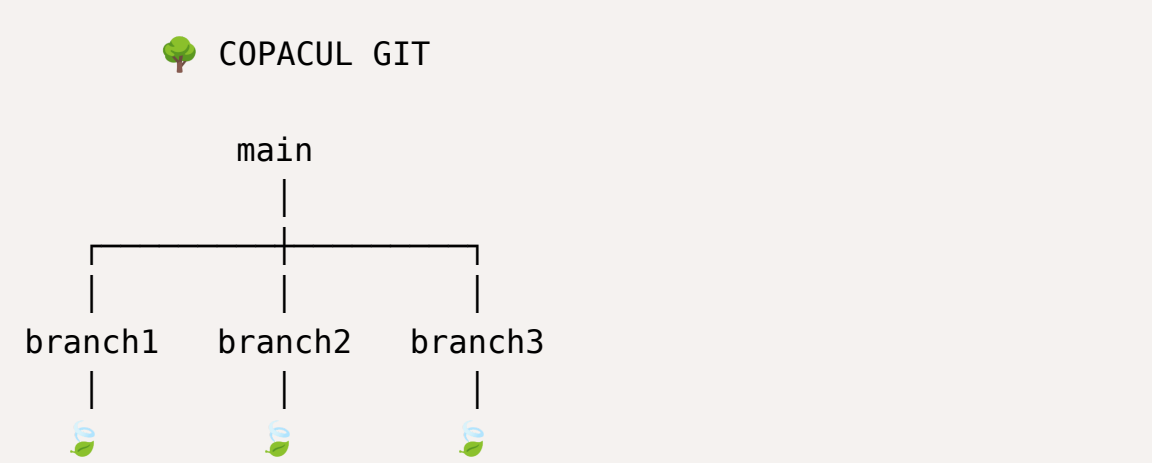
Analogie: Camera de gardă a spitalului
- Aici vine totul după ce e gata și verificat
- E codul stabil, care funcționează
- NU experimentezi direct aici!

> ⚠️ **Notă:** GitHub folosește `main`, versiunile mai vechi de Git folosesc `master`. Sunt același lucru!

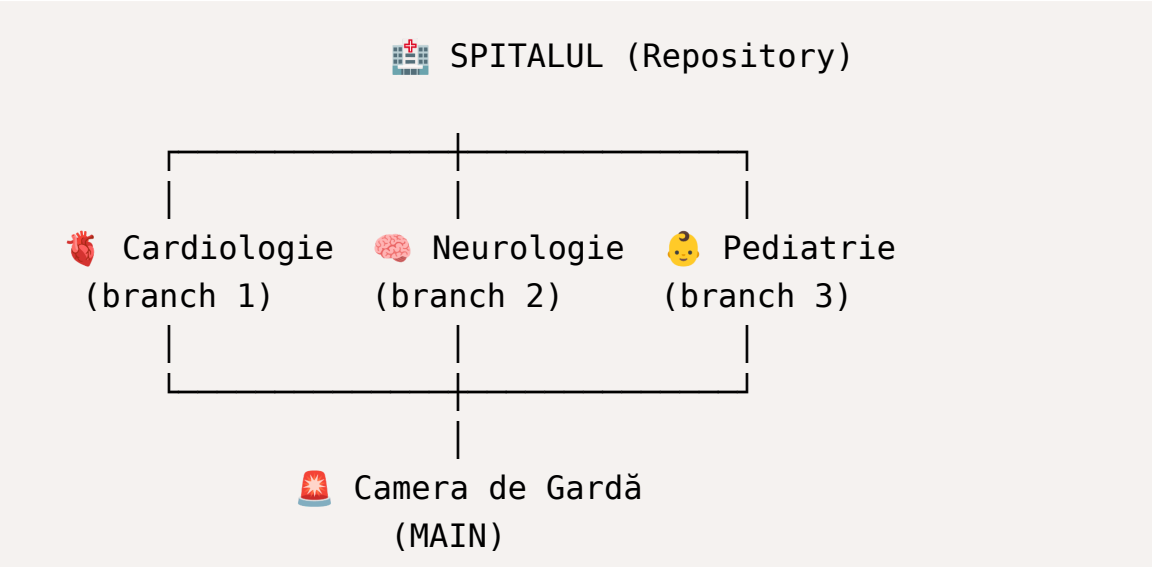
2.3 Branch-uri (Ramuri)

Ce este: O copie a codului pe care lucrezi separat, fără să afectezi main-ul.

Analogia 1: Copacul 🌳



Analogia 2: Secțiile spitalului 🏥



Fiecare secție lucrează separat, dar la final pacientul (codul) vine la gardă cu tratament!

De ce folosim branch-uri?

Motiv	Explicație
Izolare	Testezi features noi fără să strici main-ul
Colaborare	Mai mulți programatori pe același cod
Backup	Experimentezi pe copie, nu pe original

Exemplu practic:

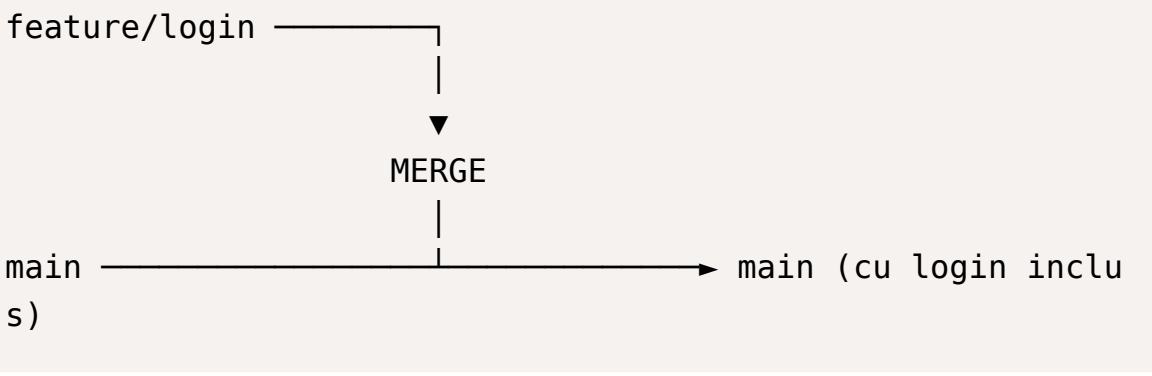
```
main (codul stabil)
  │
  ├── feature/login      ← Ion lucrează la login
  │
  ├── feature/dashboard  ← Maria lucrează la dashboard
  │
  └── bugfix/eroare-x    ← Alex repară un bug
```

Fiecare lucrează pe branch-ul lui. Când termină, face **merge** (unire) în main.

2.4 Merge

Ce este: Procesul de a aduce modificările dintr-un branch înapoi în main.

Analogie: Pacientul revine la camera de gardă cu tratamentul aplicat!



📍 PARTEA 3: CUM FACEM UN REPO - PAȘI PRACTICI

Metoda RECOMANDATĂ: Creezi întâi pe GitHub

```
# 1. Creezi repo-ul pe github.com (cu README sau gol)
#    → Buton verde "New repository"

# 2. Clonezi pe PC

#    Deschide CMD sau PowerShell în folderul unde vrei să c
```

```
opiezi
#    sau
#    Copiem adresa unde se afla folderul in care vrem sa co
pie si mergem cu
#    cd C:\Users\Cata\Desktop\Newfolder ( exemplu nostru de
astaseara)

git clone https://github.com/USERNAME/REPO_NAME.git ( linku
l copiat din GitHub)

# 3. Intri în folder
cd REPO_NAME ( Numele folderului care s-a creat dupa clone)

# 4. Adaugi fișierele tale
#    (copiezi fișierele în folder sau le creezi)

# 5. Add + Commit + Push
git add .
git commit -m "Primul commit"
git push
```

Avantaj: Remote-ul (conexiunea cu GitHub) e configurat automat!

📍 PARTEA 4: COMENZI ESENȚIALE

4.1 Ciclul de bază: Add → Commit → Push

Analogia: Nașterea unui copil 👶

Etapă	Comandă	Ce face	Analogie
1	`git add .`	Pregătește fișierele	Contracțiile încep
2	`git commit -m "mesaj"`	Salvează LOCAL	Copilul vrea să vină pe lume
3	`git push`	Trimite pe server	Nașterea propriu-zisă! 🎉

Etapă	Comandă	Ce face	Analogie
4	`git pull`	Trage de pe server	Mama ține copilul în brațe

⚠️ FOARTE IMPORTANT DE REȚINUT:

```
┌──────────────────────────────────────────────────────────────────────
──────┐
│  ❌ Commit FĂRĂ Push = rămâne LOCAL (nu ajunge pe serve
r!)  │
│  ❌ Push FĂRĂ Commit = NU SE ÎNTÂMPLĂ NIMIC!
│
└──────────────────────────────────────────────────────────────────────
──────┘
```

Trebuie MEREU în ordine: `add` → `commit` → `push`

4.2 Comenzile detaliate

git add

```
git add .                    # Adaugă TOATE fișierele modif
icate
git add fisier.py            # Adaugă doar un fișier specif
ic
git add folder/              # Adaugă un folder întreg
```

Ce face: Pregătește fișierele pentru commit (le pune în "cutie").

git commit

```
git commit -m "Am adăugat funcția de login"
```

Ce face: Salvează modificările LOCAL cu un mesaj descriptiv.

> ⚠️ **Commit-ul rămâne PE CALCULATORUL TĂU! Nu s-a dus încă pe GitHub!**

git push

```
git push                    # Trimite pe branch-ul curent
git push origin main        # Trimite pe main
git push -u origin main     # Prima dată când trimiți (set
ează tracking)
```

Ce face: Trimite commit-urile pe GitHub (în cloud).

---

git pull

```
git pull                    # Trage ultimele modificări de
pe server
```

Ce face: Aduce modificările făcute de alții (sau de tine de pe alt calculator).

4.3 Comenzi de verificare (FOARTE UTILE!)

```
git status                  # Vezi ce fișiere ai modificat
(CEA MAI IMPORTANTĂ!)
git fetch                   # Vezi ce s-a schimbat pe serv
er
git log                     # Vezi istoricul commit-urilor
git log --oneline           # Istoric scurt (o linie per c
ommit)
git checkout nume-branch    # Schimbă pe alt branch
```

git status - CEA MAI IMPORTANTĂ!

```
git status
```

Ce-ți arată:
- Pe ce branch ești
- Ce fișiere ai modificat
- Ce fișiere sunt pregătite pentru commit
- Ce fișiere nu sunt tracked

> 💡 **Sfat:** Dă `git status` ÎNAINTE și DUPĂ fiecare comandă până te obișnuiești!

📍 PARTEA 5: FORK VS CLONE VS BRANCH

5.1 Tabel comparativ rapid

Concept	Ce este	Unde creează copia	Conexiune cu original
Branch	Ramură în ACELAȘI repo	În repo-ul tău	Directă (merge)
Clone	Copie locală a repo-ului	Pe calculatorul tău	Directă (push/pull)
Fork	Copie pe CONTUL TĂU GitHub	Pe GitHub (contul tău)	Prin Pull Request

---

5.2 Clone

```
git clone https://github.com/user/repo.git
```

Ce face:
- Copiază repo-ul pe calculatorul tău
- Rămâi CONECTAT la repo-ul original
- Dacă ai permisiuni, poți face push direct
- Dacă NU ai permisiuni, nu poți face push

Când folosești:
- Proiect propriu
- Proiect de echipă unde ai acces

5.3 Fork

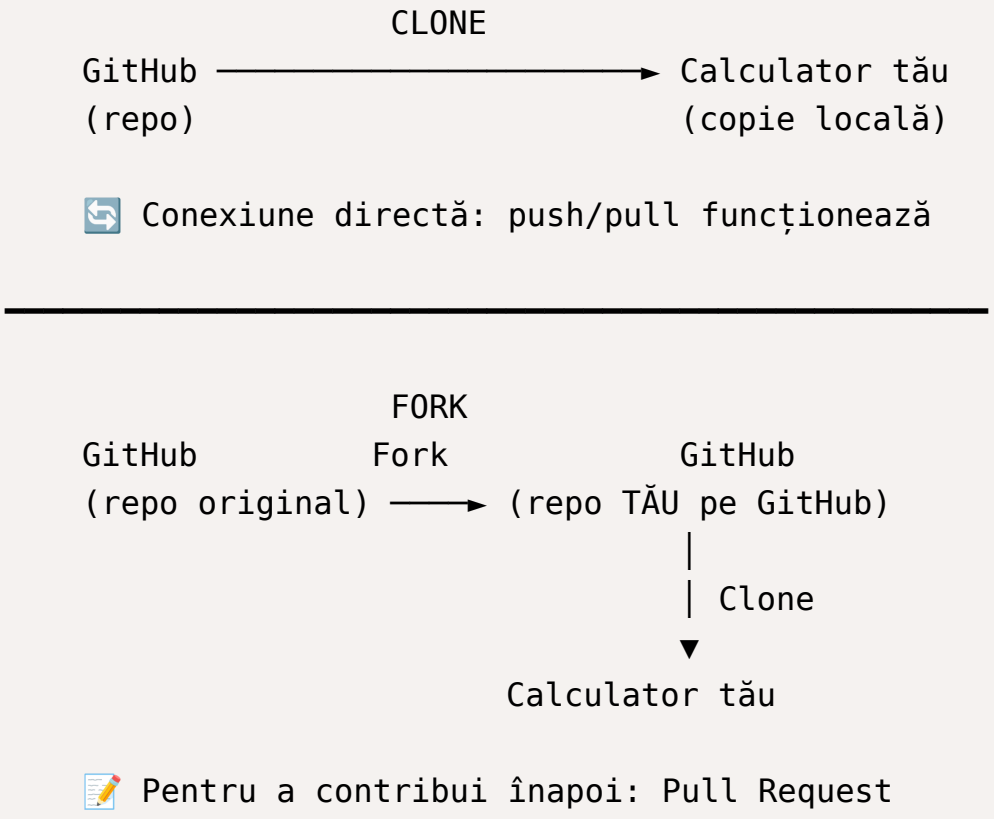
Cum se face: Din GitHub, butonul “Fork” din dreapta sus (NU e comandă în terminal!)

Ce face:
- Creează O COPIE a repo-ului în CONTUL TĂU pe GitHub
- Repo-ul tău e separat, dar “legat vizual” de original

- Poți face orice modificări vrei
- Pentru a contribui înapoi, faci **Pull Request**

Când folosești:
- Proiecte open-source unde NU ai acces direct
- Vrei să experimentezi fără să afectezi originalul

5.4 Diferența Fork vs Clone (VIZUAL)



5.5 Diferența Fork vs Branch

Aspect	Branch	Fork
Unde există	În același repo	Repo separat
Cine o face	Colaboratori cu acces	Oricine
Pentru ce	Lucru în echipă	Contribuții externe
Merge	Direct	Prin Pull Request

📍 PARTEA 6: WORKFLOW-UL ZILNIC RECOMANDAT

6.1 Când începi să lucrezi

```
# 1. Verifică pe ce branch ești
git status

# 2. Trage ultimele modificări
git pull

# 3. (Opțional) Creează branch pentru task-ul tău
git checkout -b feature/task-nou
```

6.2 Când termini de lucrat

```
# 1. Vezi ce ai modificat
git status

# 2. Adaugă fișierele
git add .

# 3. Commit cu mesaj clar
git commit -m "Descriere clară a ce am făcut"

# 4. Push pe server
git push
```

6.3 Reguli de aur pentru echipă

Regulă	De ce
📌 Push cel puțin o dată pe zi	Backup + colegii văd progresul
📣 Anunță echipa când dai push	Evită conflicte
✅ Verifică cu `git status`	Să știi mereu unde ești

Regulă	De ce
🔒 Folosește `.gitignore`	Nu urca chei private, fișiere temporare
📝 Mesaje de commit clare	Să înțelegi peste 6 luni ce ai făcut

📍 PARTEA 7: .GITIGNORE - CE NU TREBUIE URCAT

7.1 Ce este?

Un fișier special `.gitignore` care spune Git-ului ce să IGNORE.

7.2 Ce NU urcăm niciodată

```
# Chei și secrete (FOARTE IMPORTANT!)
*.env
secrets.py
config_private.py
api_keys.txt

# Fișiere temporare Python
__pycache__/
*.pyc
*.pyo

# Fișiere IDE
.vscode/
.idea/
*.swp

# Fișiere sistem
.DS_Store
Thumbs.db

# Dependențe (se reinstalează)
node_modules/
venv/
```

📍 PARTEA 8: ERORI COMUNE ȘI SOLUȚII

8.1 “fatal: not a git repository”

Problemă: Nu ești într-un folder Git.

Soluție:

```
cd nume-proiect              # Intră în folderul corect
# SAU
git init                     # Inițializează repo nou
```

8.2 “error: failed to push some refs”

Problemă: Cineva a făcut push înainte ta.

Soluție:

```
git pull                     # Trage modificările lor întâi
# Rezolvă conflictele dacă apar
git push                     # Apoi push din nou
```

8.3 Branch-ul e “master” dar vrei “main”

Problemă: Ai creat local cu `master`, dar GitHub vrea `main`.

Soluție:

```
git branch -M main           # Redenumește branch-ul
git push -u origin main      # Push cu noul nume
```

8.4 “No configured push destination”

Problemă: Nu ai conectat repo-ul local la GitHub.

Soluție:

```
git remote add origin https://github.com/USER/REPO.git
git push -u origin main
```

📍 PARTEA 9: CHEAT SHEET - TOATE COMENZILE

Comenzi de bază

```
git init                    # Inițializează repo nou
git clone URL               # Clonează repo existent
git status                  # Vezi starea curentă
git add .                   # Adaugă toate fișierele
git commit -m "mesaj"       # Salvează local
git push                    # Trimite pe server
git pull                    # Trage de pe server
```

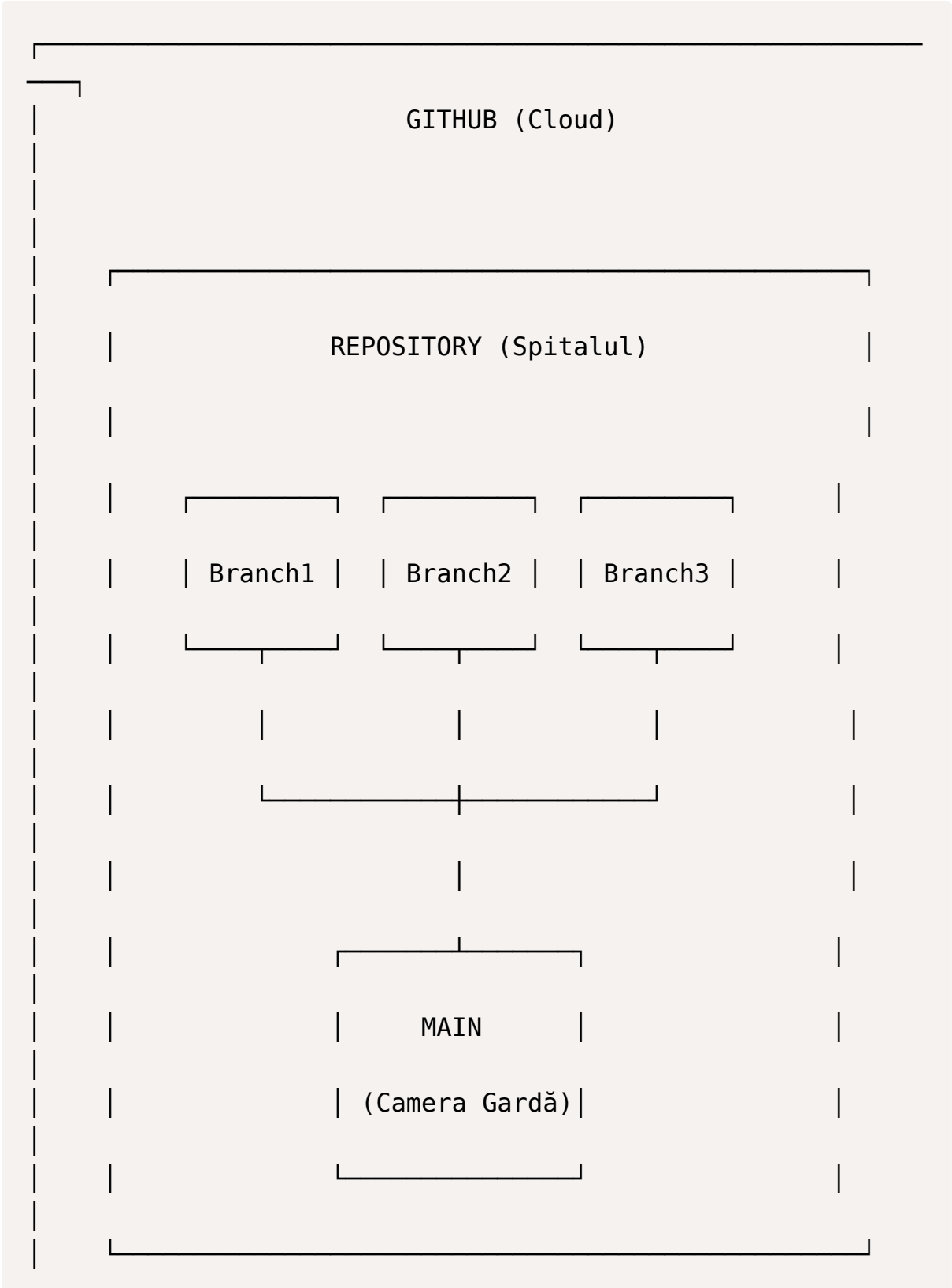
Comenzi pentru branch-uri

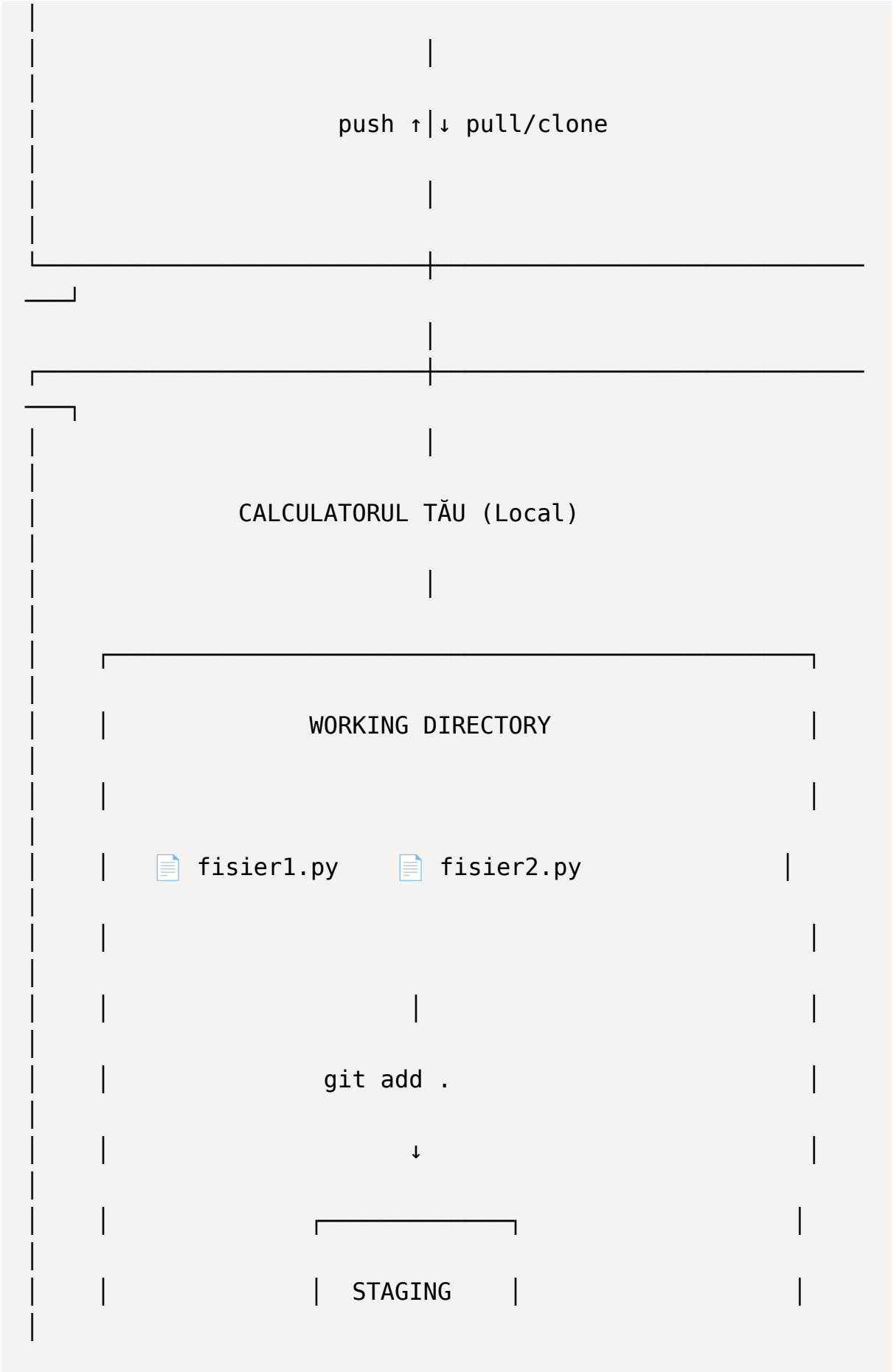
```
git branch                  # Vezi branch-urile
git branch nume             # Creează branch
git checkout nume           # Schimbă pe branch
git checkout -b nume        # Creează + schimbă (2 în 1)
git merge nume              # Unește branch în cel curent
git branch -d nume          # Șterge branch
```

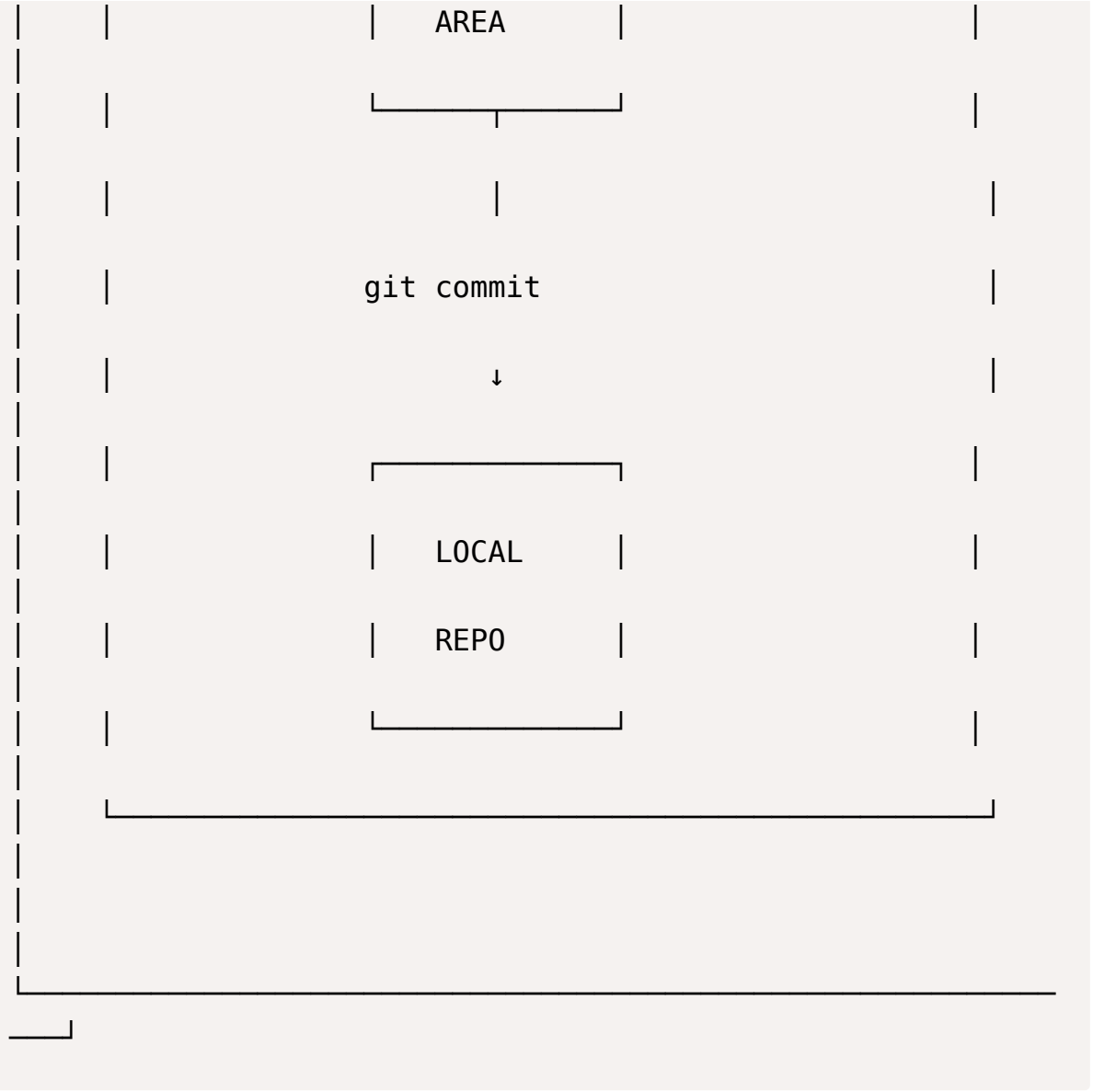
Comenzi de verificare

```
git status                  # Stare fișiere
git log                     # Istoric commits
git log --oneline           # Istoric scurt
git fetch                   # Verifică schimbări pe server
git remote -v               # Vezi conexiunile
git branch -a               # Toate branch-urile
```

📍 PARTEA 10: SCHEMA VIZUALĂ COMPLETĂ




push ↑|↓ pull/clone
CALCULATORUL TĂU (Local)
WORKING DIRECTORY
fisier1.py
fisier2.py
git add .
STAGING



📍 PARTEA 11: ÎNTREBĂRI PENTRU LAURENȚIU

1. **Fork vs Clone:** Care e diferența exactă și când folosim fiecare în proiecte reale?
2. **Pull Request:** Cum funcționează procesul complet de aprobare?
3. **Conflicte de merge:** Cum le rezolvăm când apar?
4. **Branch protection:** Cum protejăm main-ul de push-uri accidentale?

📍 PARTEA 12: EXERCIȚII PRACTICE

Exercițiul 1: Primul repo (5 minute)

```
# 1. Creează folder și intră în el
mkdir primul-repo
cd primul-repo

# 2. Inițializează Git
git init

# 3. Creează un fișier
echo "# Primul meu repo" > README.md

# 4. Add + Commit
git add .
git commit -m "Initial commit"
```

Exercițiul 2: Conectare la GitHub (5 minute)

1. Pe github.com, creează repo GOL (fără README)
2. Copiază URL-ul
3. Rulează:

```
git remote add origin https://github.com/USER/REPO.git
git branch -M main
git push -u origin main
```

Exercițiul 3: Clone și modificare (10 minute)

1. Clonează repo-ul unui coleg
2. Modifică un fișier
3. `git add .` → `git commit -m "Test"` → `git push`
4. Vezi pe GitHub dacă a apărut modificarea

Exercițiul 4: Lucru cu branch-uri (10 minute)

```
# 1. Creează branch
git checkout -b test-branch

# 2. Adaugă un fișier
echo "Test file" > test.txt

# 3. Add + Commit + Push
git add .
git commit -m "Adaugat fisier test"
git push -u origin test-branch

# 4. Pe GitHub, vezi branch-ul nou
```

📋 REZUMAT FINAL

```
┌──────────────────────────────────────────────────────────────────────┐
│                        REZUMAT GIT                                   │
├──────────────────────────────────────────────────────────────────────┤
│                                                                      │
│  Repository = Spitalul întreg (containerul)                          │
│                                                                      │
│  Main = Camera de gardă (codul principal)                            │
│                                                                      │
│  Branch = Secții (izolare pentru testare)                            │
│                                                                      │
│  Merge = Pacientul revine la gardă cu tratament                      │
│                                                                      │
│                                                                      │
│                                                                      │
│  ────────────────────────────────────────────────────────────────    │
```

```
  │
  │
  │
  │  add → commit → push  (în această ordine MEREU!)
  │
  │
  │
  │  Commit fără Push = rămâne LOCAL
  │
  │  Push fără Commit = NU SE ÎNTÂMPLĂ NIMIC
  │
  │
  │
  │  ──────────────────────────────────────────────────────────
  │
  │
  │
  │  Clone = Copie locală, conexiune directă
  │
  │  Fork = Copie pe contul tău, contribui prin Pull Request
  │
  │
  │
  └──────────────────────────────────────────────────────────────
  ──┘
```

Document creat de echipa Mirfak-PTH-27-11-2025- Februarie 2026
Bazat pe discuția de grup + notițele individuale + ideile principale
Pentru întrebări, contactați-l pe Laurențiu sau pe Cata