

# Git și puțin GitHub.



Nu doar salvează. Creează  
versiuni, urmărind tot ce ai  
schimbat.

# Cuprins

---

3..... Este Git același lucru cu GitHub?

4..... Ce este Git?

5..... Ce este GitHub?

6-7..... Sisteme de Control al Versiunilor

8..... Funcționalități Git

9-21...Concepte de bază ale sistemelor de versiuni

22-30. Flux de lucru și comenzi esențiale Git. (simplificat)

31..Subiecte Avansate cuprins

32-34...Commit-ul perfect

35-43..Strategii de Ramificare

44-47.. Pull Requests



48-50..Merge Conflicts

51-54..Merge vs Rebase

55...Mulțumiri

# Este Git același lucru cu GitHub?

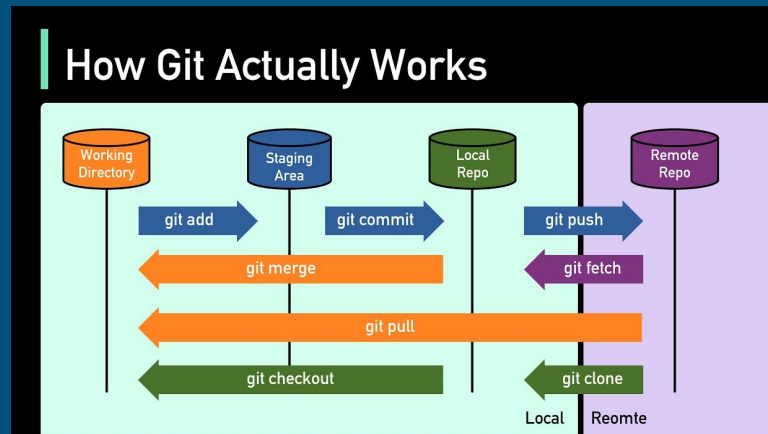
Nu. Git este un software care funcționează **LOCAL** si GitHub este un site unde poți încărca și descărca fișiere scrise de tine și/sau de alte persoane **REMOTE**.

	
1. It is a software	1. It is a service
2. It is installed locally on the system	2. It is hosted on Web
3. It is a command line tool	3. It provides a graphical interface
4. It is a tool to manage different versions of edits, made to files in a git repository	4. It is a space to upload a copy of the <b>Git</b> repository
5. It provides functionalities like Version Control System Source Code Management	5. It provides functionalities of Git like VCS, Source Code Management as well as adding few of its own features

# Ce este Git?

Pe scurt, Git este un software de control al versiunilor (VCS) care îți dă acces la funcționalitățile unei **salvări** și acces la **istoricul complet** al *salvărilor*. Aceasta este funcția sa primară.

Poți crea snapshots (commits) al stării proiectului și te poți întoarce la un snapshot anterior oricând îți dorești.



# Ce este GitHub?

---

**GitHub** este o platformă online care folosește Git pentru a găzdui codul sursă și a facilita colaborarea între dezvoltatori. Poți încărca propriile proiecte, accesa codul altora, clona repository-uri local, contribui prin pull request-uri și colabora în echipă.

- Alte platforme similare sunt **GitLab** și **Bitbucket**, care oferă funcționalități aproape identice.



# Ce face un Sistem de Control al Versiunilor (VCS)?

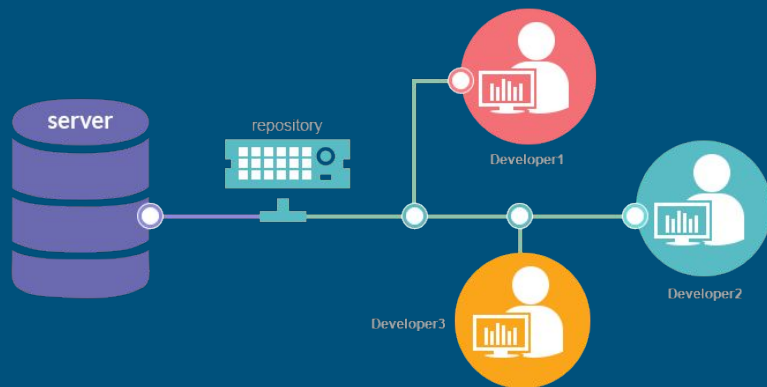
**Ține evidența modificărilor** aduse fișierelor dintr-un proiect (de obicei cod sursă).

**Permite salvarea „versiunilor” codului** în diferite momente (prin commit-uri).

**Permite revenirea** la o versiune anterioară dacă apare o problemă.

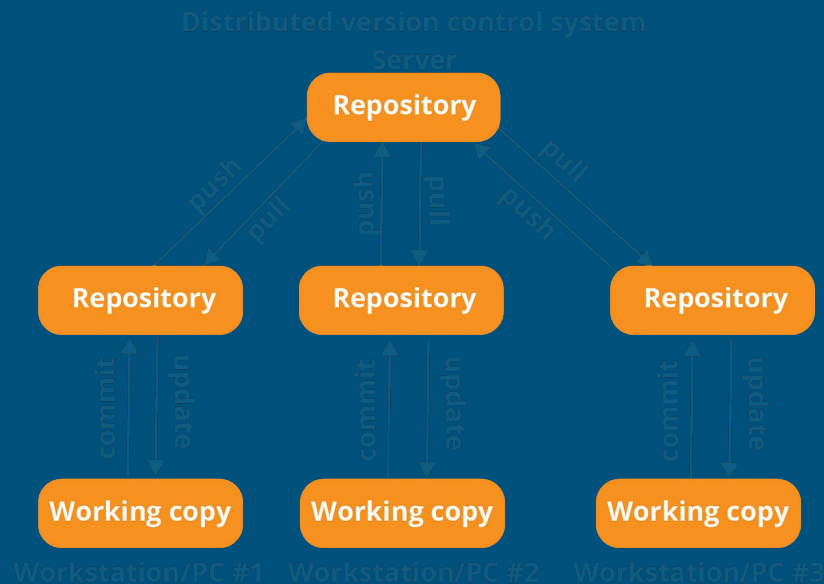
**Păstrează istoricul complet** al modificărilor și **cine le-a făcut**.

**Facilitează colaborarea** între mai mulți oameni care lucrează pe același proiect.



# De ce este important?

- Te ajută să nu pierzi modificări importante.
- Poți experimenta fără teama de a „strica” codul.
- E esențial în echipe: toți pot lucra simultan pe același proiect, fără a se călca pe cod.



# Funcționalități Git

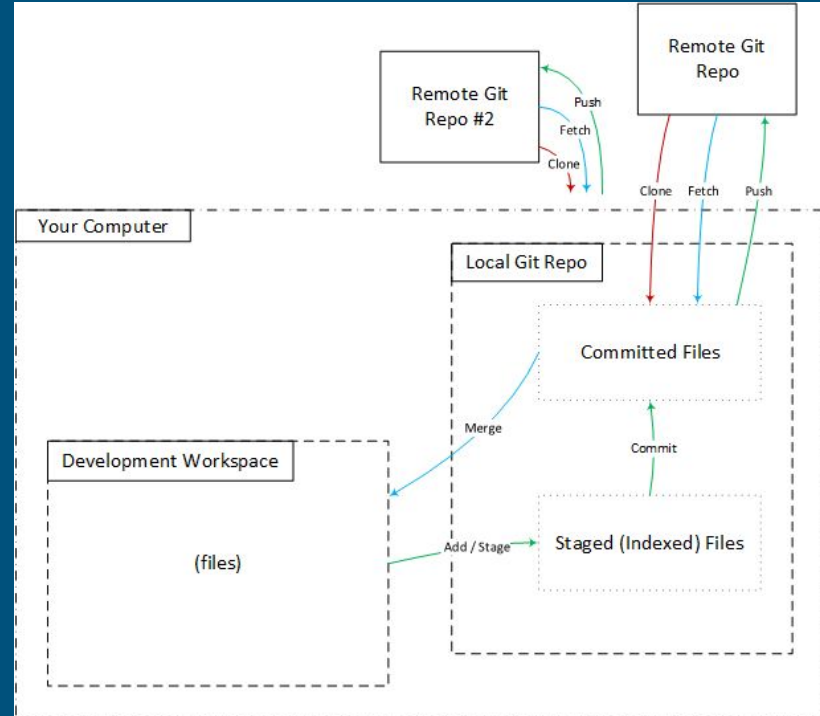
- **Salvarea versiunilor și revenirea la o stare anterioară**  
Permite crearea de "snapshot-uri" ale codului, facilitând revenirea la versiuni stabile în caz de probleme.
- **Istoric complet al modificărilor și al autorilor**  
Git păstrează un jurnal detaliat al fiecărei modificări, indicând ce a fost schimbat, când și de către cine.
- **Crearea de ramuri (branches)**  
Posibilitatea de a dezvolta funcționalități noi sau de a testa cod fără a afecta codul principal (de exemplu main sau master).
- **Îmbinarea ramurilor (merge)**  
Combinarea modificărilor din diferite ramuri într-o versiune finală coerentă.
- **Conectarea cu un repository remote (ex: GitHub)**  
Permite colaborarea cu alți dezvoltatori prin sincronizarea codului local cu un repository la distanță.





# 1. Concepte de bază ale sistemelor de versiuni

1. Repository
2. Branch
3. Commit
4. Merge
5. Rebase
6. Clone & Pull
7. Push
8. Staging Area
9. History / Lot
10. Tag
11. Reset / Revert / Checkout
12. Conflicts

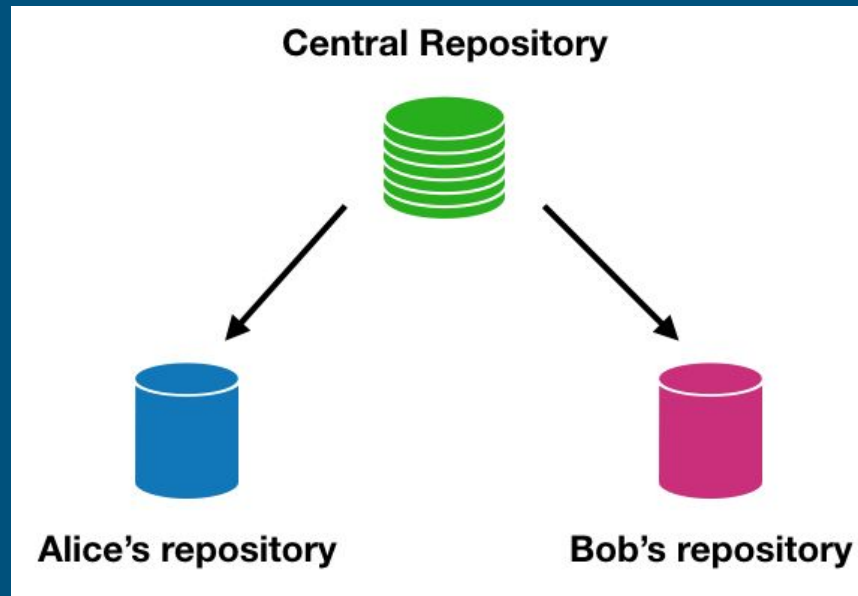


# 1.1 Repository

Un repository este spațiul de stocare al proiectului tău și istoricul schimbărilor

În Git ai:

- Repository **LOCAL** ( pe mașina ta )
- Repository **REMOTE** ( GitHub, GitLab etc )

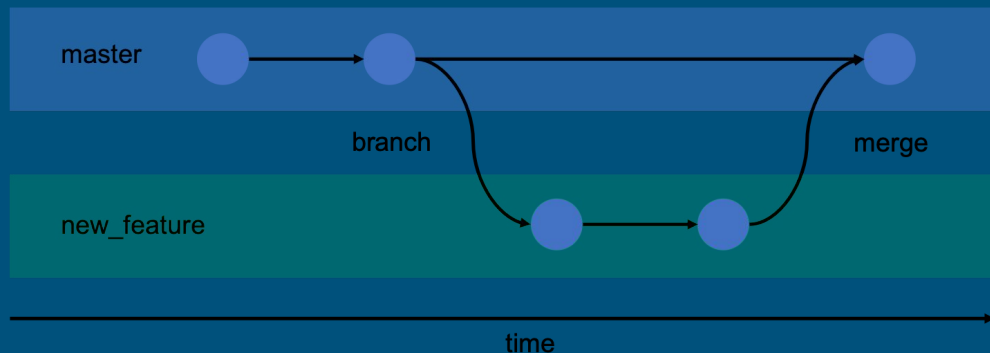


# 1.2 Branch

O ramură este o linie **PARALELĂ** de dezvoltare.

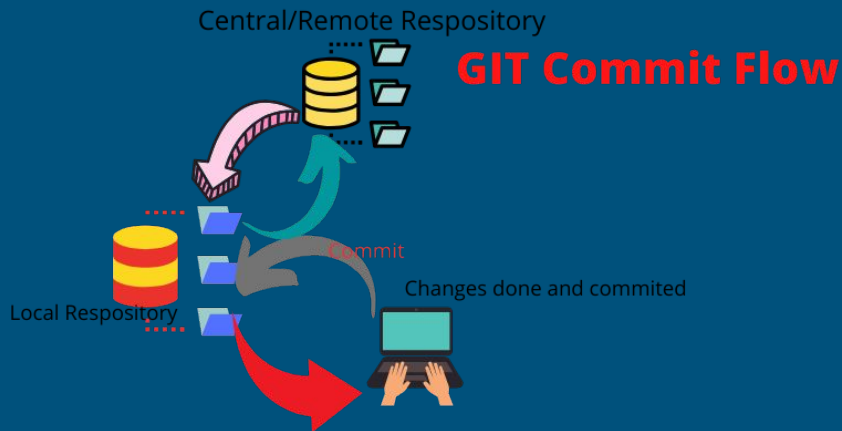
Tipuri de ramuri comune:

- ***main*** sau ***master*** : codul de producție
- ***dev, feature/xy, hotfix/12***



# 1.3 Commit

- Un snapshot al codului dvs. la un moment dat.
- Fiecare comitere are un hash SHA unic, un autor, marca temporală și un mesaj.
- Vă permite să urmăriți și să anulați modificările în timp.

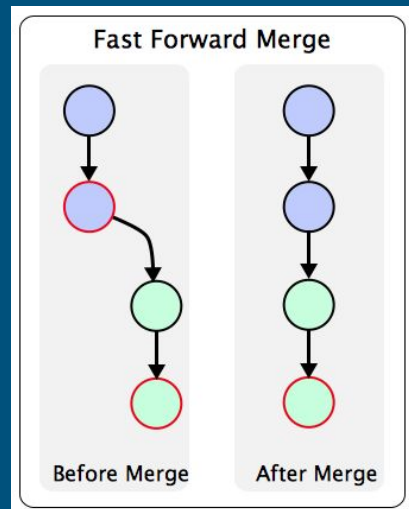
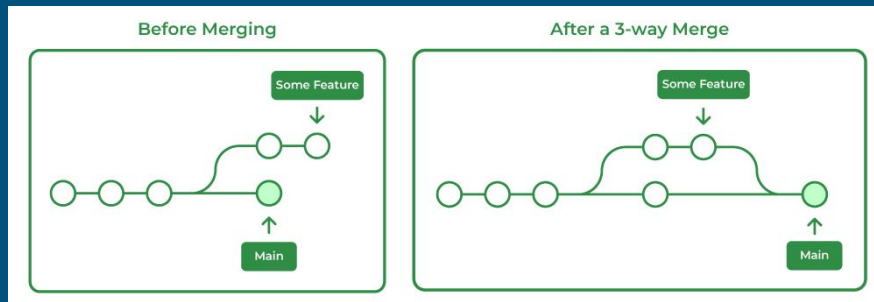


# 1.4 Merge

Combină schimbările de la o ramură în alta.

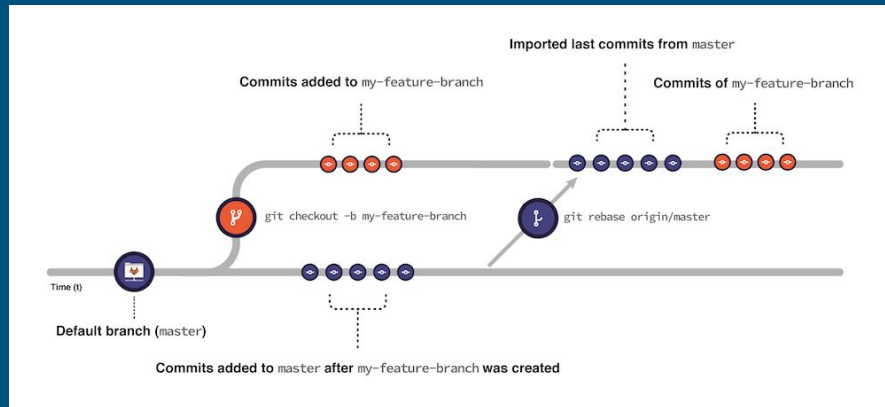
Poate fi :

- Fast Forwarding (istoric liniar)
- Îmbinare în trei căi (creează un commit de îmbinare)



# 1.5 Rebase

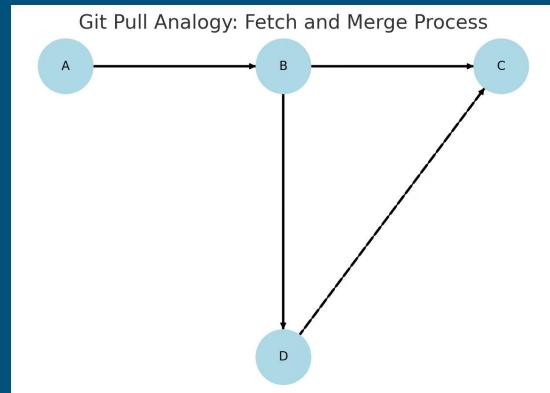
- Reaplica comiterile de la o ramură peste alta.
- Folosit pentru a crea un istoric liniar al comiterilor.
- Rescrie istoricul comiterilor



# 1.6 Clone & Pull

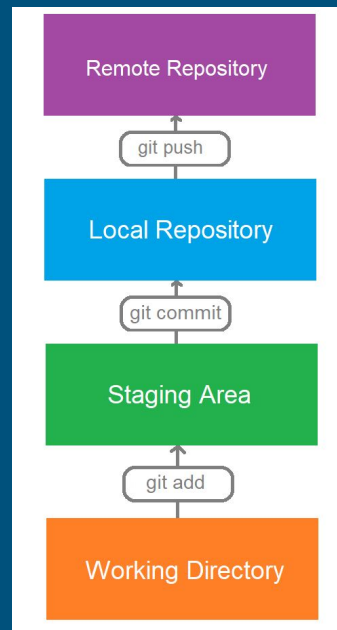
Clone: Creeaza o copie locală a unui repository remote

Pull: preia și îmbină modificările de la repository remote la ramura locală.



# 1.7 Push

Trimite comiterile locale la repository remote.

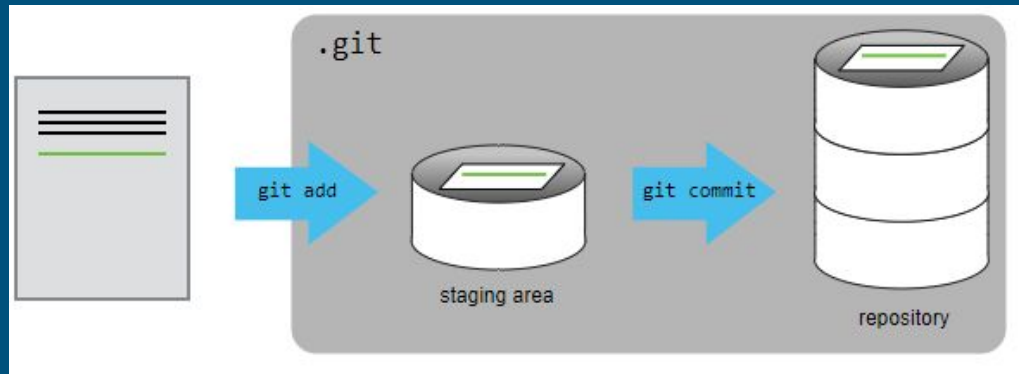




## 1.8 Staging Area (index)

Un spațiu intermediar unde pregătești modificările înainte să dai commit.

- **`git add`** mută modificările în staging area.
- **`git commit`** le finalizează de salvat.



# 1.9 History & Log

Vizualizați secvența commit-urilor folosind `git log`.

Vă ajută să auditați modificările și să urmăriți problemele.

```
commit 434de730f5abe43c8f6f8e32247f2e04d31635f6 (HEAD -> newversion, origin/master, origin/develop)
Author: fyodor <fyodor@e0a8ed71-7df4-0310-8962-fdc924857419>
Date:   Sun Dec 9 02:00:55 2018 +0000

    Update copyright year for Ncat and Ncat Guide

commit 6d420e82b2c55b6c7723c07e33771c52ad193b5e
Author: dmiller <dmiller@e0a8ed71-7df4-0310-8962-fdc924857419>
Date:   Sun Dec 2 05:54:58 2018 +0000

    Changelog for #1227

commit 1ba01193725f4c83bf9e4b4cd589dbc9fc626152
Author: dmiller <dmiller@e0a8ed71-7df4-0310-8962-fdc924857419>
Date:   Sun Dec 2 05:48:27 2018 +0000

    Add a length check for certificate parsing. Fixes #1399

commit b1efd742499b00eef970feef84dc64f301db61f
Author: dmiller <dmiller@e0a8ed71-7df4-0310-8962-fdc924857419>
Date:   Thu Nov 29 20:27:05 2018 +0000

    Warn for raw scan options without needed privileges

commit b642dc129c4d349a849fb0eb055cde263d9d3eb6
Author: dmiller <dmiller@e0a8ed71-7df4-0310-8962-fdc924857419>
Date:   Thu Nov 29 17:42:09 2018 +0000

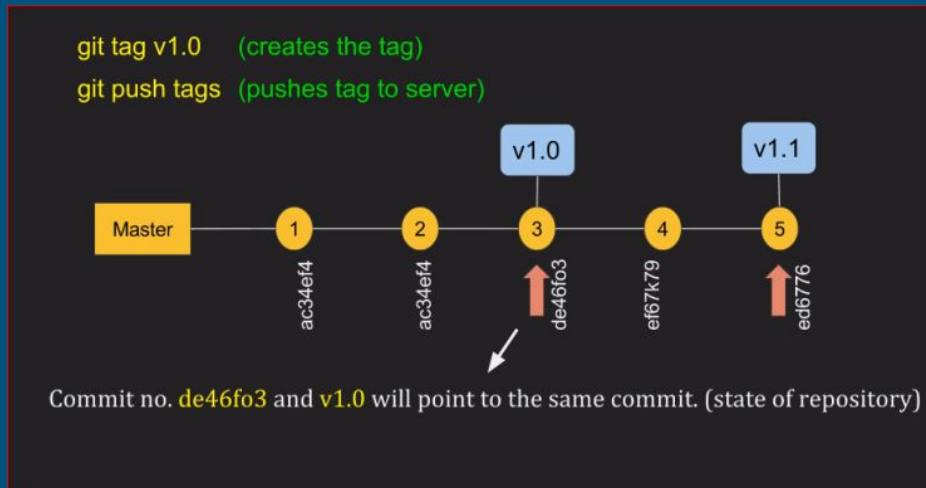
    Fix a bug in the fix. https://github.com/nmap/nmap/commit/ebf083cb0bfc239a000aea7764cc

commit 350bbe0597d37ad67abe5fef8fba984707b4e9ad
Author: dmiller <dmiller@e0a8ed71-7df4-0310-8962-fdc924857419>
Date:   Thu Nov 29 17:42:09 2018 +0000

    Avoid a crash (double-free) when SSH connection fails
```

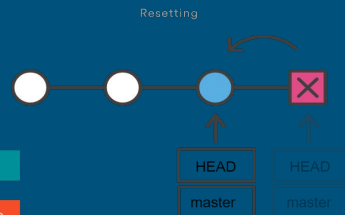
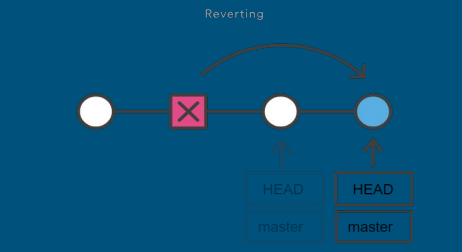
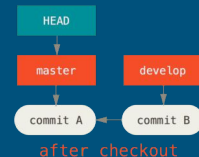
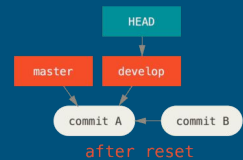
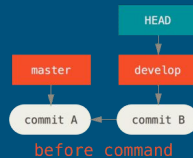
# 1.10 Tag

- Un pointer numit către o comitere specifică.
- Adesea folosit pentru a marca versiunile de lansare (v1.0.0 etc.)



# 1.11 Reset / Revert / Checkout

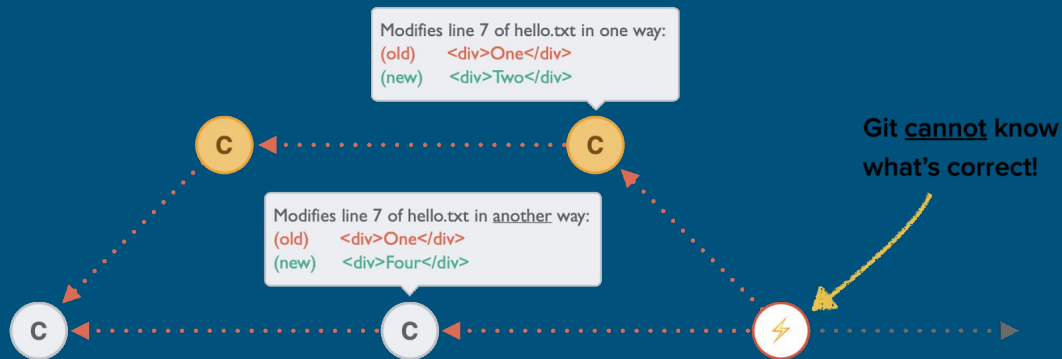
- Reset: Mută HEAD și, opțional, schimbă directorul de lucru și de ședere.
- Revert: anulează în siguranță o comitare creând una nouă.
- Checkout: comută ramuri sau restaurează fișierele.



# 1.12 Conflicts

Apar atunci când două ramuri au modificări incompatibile.

- Trebuie rezolvat manual în timpul merge sau rebase.



## 2. Flux de lucru și comenzi esențiale Git. (simplificat)

---

1. Configurare inițială
2. Crearea și clonarea unui repository
3. Stagiile de lucru ( Working Directory -> Staging -> Repository )
4. Istoricul și jurnalul de commit-uri
5. Ramuri
6. Sincronizare cu remote (GitHub, GitLab etc)
7. Undo & Corectări
8. Alte comenzi utile

## 2.1 Configurare inițială

Comanda	Ce face	Cand o folosești
<code>git config --global <u>user.name</u> "Nume"</code>	Seteaza numele utilizatorului	La prima utilizare Git pe o masina
<code>git config --global user.email "exemplu@gmail.com"</code>	Seteaza email-ul utilizatorului	La prima utilizare Git pe o masina
<code>git config --global core.editor "editor"</code>	Seteaza editorul implicit	Daca vrei sa folosesti alt editor (ex: code, nano)

```
user@DESKTOP-NRPE00H MINGW64 ~  
$ git config --global user.name "exemplu"  
  
user@DESKTOP-NRPE00H MINGW64 ~  
$ git config --global user.email "exemplu@gmail.com"
```

## 2.2 Crearea și clonarea unui repository

Comanda	Ce face	Cand o folosești
git init	Creeaza un nou repository Git gol	Cand vrei sa începi un nou proiect local
git clone <url>	Cloneaza un repozitoriu remote de pe GitHub/GitLab etc.	Cand vrei sa contribui la un proiect deja existent

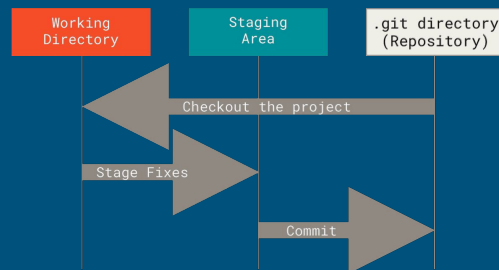
```
user@DESKTOP-NRPE00H MINGW64 ~/desktop/exemplu
$ git init
Initialized empty Git repository in C:/Users/user/Desktop/exemplu/.git/
```

```
user@DESKTOP-NRPE00H MINGW64 ~/desktop/exemplu
$ git clone https://github.com/oveandrei/IntelligentPython.git
Cloning into 'IntelligentPython'...
remote: Enumerating objects: 60, done.
remote: Counting objects: 100% (60/60), done.
remote: Compressing objects: 100% (54/54), done.
remote: Total 60 (delta 6), reused 53 (delta 3), pack-reused 0 (from 0)
Receiving objects: 100% (60/60), 32.48 KiB | 1.08 MiB/s, done.
Resolving deltas: 100% (6/6), done.
```



## 2.3 Stagiile de lucru

Comanda	Ce face	Cand o folosești
git status	Arată fișierele modificate/adaugate	Verifici ce urmează sa comiti
git add <fișier>	Adaugă un fișier în 'staging'	Pregatești fișierul pentru commit
git add .	Adaugă toate fișierele modificate	Cand vrei sa incluzi tot ce e modificat
git reset <fișier>	Scoate fișierul din 'staging'	Dacă ai adaugat ceva din greșeală
git commit -m "mesaj"	Salveaza modificarile din repository	Cand finalizezi o etapă de lucru
git commit -am "mesaj"	Adaugă și comite fișierele deja tracked	Shortcut rapid (nu funcționează pentru fișiere noi)



## 2.4 Istoricul și Jurnalul de commits

Comanda	Ce face	Cand o folosești
git log	Arată istoria commits	Cand vrei sa vezi cine, ce și când a modificat
git log --oneline	Versiune scurta a log	Pentru o privire de ansamblu rapidă
git show <hash>	Arată detaliile unui commit	Pentru a inspecta un commit specific

```
$ git show 635aad5a4b95756bdfabc1bb3f21b33b38b13be1
commit 635aad5a4b95756bdfabc1bb3f21b33b38b13be1 (HEAD -> master)
Author: Bicu Andrei Ovidiu <ove.2k15@gmail.com>
Date:   Wed May 28 16:29:28 2025 +0300

    Added ex.txt file

diff --git a/ex.txt b/ex.txt
new file mode 100644
index 0000000..e69de29
```

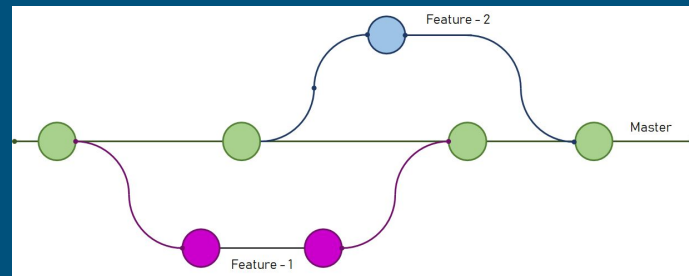
```
$ git log --oneline
635aad5 (HEAD -> master) Added ex.txt file
```

```
$ git log
commit 635aad5a4b95756bdfabc1bb3f21b33b38b13be1 (HEAD -> master)
Author: Bicu Andrei Ovidiu <ove.2k15@gmail.com>
Date:   Wed May 28 16:29:28 2025 +0300

    Added ex.txt file
```

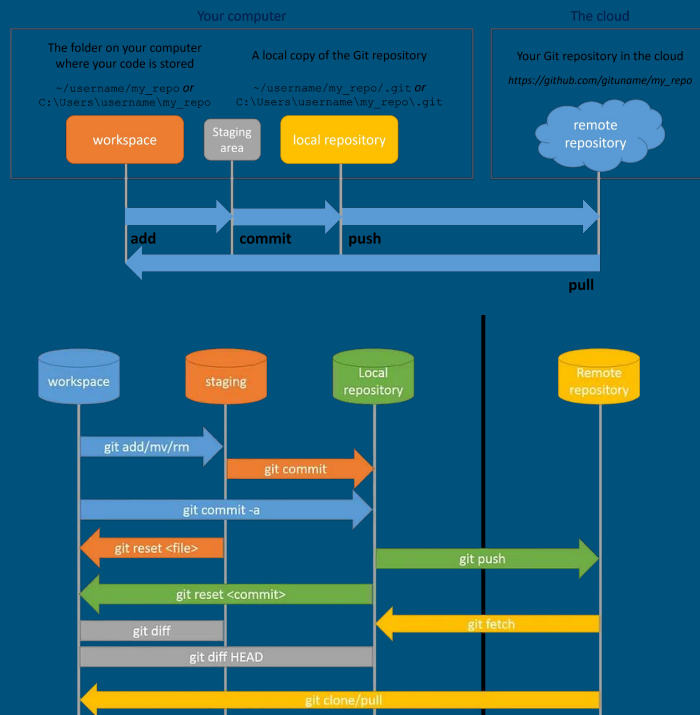
## 2.5 Ramuri

Comanda	Ce face	Cand o folosești
git branch	Arată ramurile existente	Vezi pe ce ramura este
git branch <nume>	Creeaza o nouă ramură	Începi o nouă funcționalitate
git checkout <nume>	Comuti pe alta ramura	Lucrezi pe alta funcționalitate
git checkout -b <nume>	Creeaza si comuta pe o ramura nouă	Shortcut frecvent folosit
git merge <branch>	Unește o ramură în cea curentă	După ce ai finalizat lucrul pe o ramură
git branch -d <nume>	Șterge o ramură	După ce ai făcut un merge



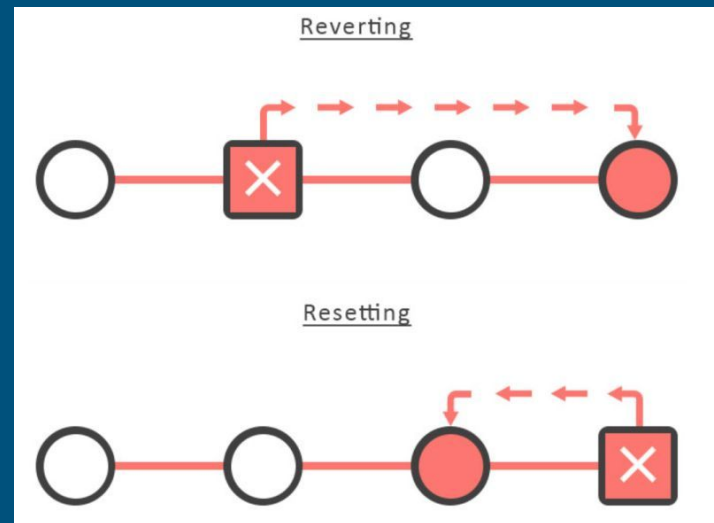
## 2.6 Sincronizare cu remote

Comanda	Ce face	Cand o folosești
<code>git remote add origin &lt;url&gt;</code>	Leaga repo-ul local de unul remote	La începutul proiectului
<code>git push -u origin main</code>	Trimite ramura main și seteaza default	Prima data cand faci push
<code>git push</code>	Trimite commiturile locale pe remote	După fiecare set de modificări
<code>git pull</code>	Aduce ultimele modificări de pe remote	Cand alții au lucrat la ceva și vrei sa fii up-to-date
<code>git fetch</code>	Aduce referințele remote dar nu face merge	Pentru a vedea ce s-a schimbat fără a modifica localul



## 2.7 Undo & Corectări

Comanda	Ce face	Cand o folosești
git checkout -<fișier>	Revine la versiunea salvată	Dacă ai modificat ceva greșit
git reset Head~1	Revine la commitul anterior ( pastreaza modificările )	Pentru a corecta un commit recent
git revert <hash>	Creeaza un commit care inversează altul	Dacă vrei sa anulezi ceva dar pastrezi istoricul
git stash	Salvează temporar modificările necomise	Cand trebuie sa comuti rapid pe alta ramura
git stash pop	Replica modificările salvate	Cand vrei sa revii la ce lucați



## 2.8 Alte comenzi utile

Comanda	Ce face	Cand o folosești
git diff	Arata diferentele intre fisiere	Verifici ce s-a schimbat înainte de commit
git tag <nume>	Adaugă un tag la un commit (versiuni)	Folosit pentru release-uri
git clean -fd	Șterge fișierele untracked	Curată directorul de lucru



```
$ git diff
diff --git a/file.txt b/file.txt 1
index 5d34e82..ba8cc3d 100644 2
--- a/file.txt 3
+++ b/file.txt 4
@@ -1,2 +1,4 @@ 5
 hello 6
 -more text 7
 +more text!!! 8
 +a new line 9
 +yet another line 10
```

# 3. Subiecte avansate

---

1. Commit-ul perfect
2. Strategii de ramificare
3. Pull requests
4. Merge conflicts
5. Merge vs Rebase



„The perfect  
commit”



## 3.1.1 Include doar modificările relevante

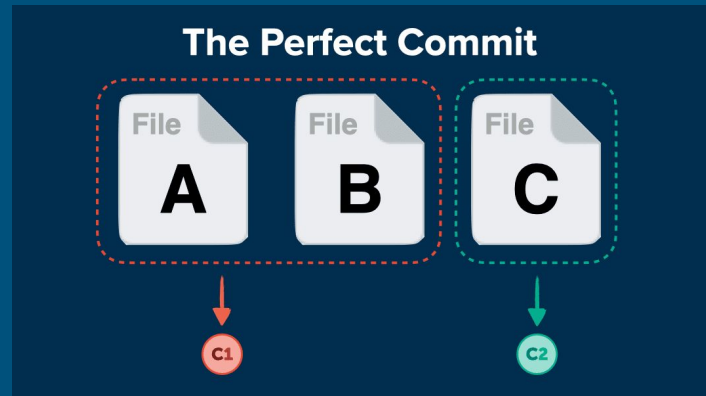
- Fiecare commit ar trebui sa conțină o singura idee / functionalitate
- Evita commit-uri mari și amestecate - sunt greu de înțeles și urmărit.
- Folosește staging pentru a selecta exact fișierele relevante.

- Exemplu:

Fișiere modificate: A, B, C

A și B -> parte din aceeași funcționalitate -> commit 1

C -> alta functionalitate -> commit 2



## 3.1.2 Scrie un mesaj de commit clar si util

- Subiect (maxim 72 caractere): rezumat concis al modificării.
- Descriere: detalii suplimentare despre ce, de ce și cum.

*Hint:* Dacă îți e greu să scrii un mesaj clar, poate ai inclus prea multe schimbări într-un singur commit.

În concluzie respectarea acestor principii îți va menține istoricul de commit-uri **clar, ușor de urmărit și ușor de înțeles** – atât pentru tine, cât și pentru echipă



# Strategii de ramificare

## 3.2.1 Strategii de ramificare

---

1. O convenție scrisă care stabilește modul de lucru cu ramurile Git în echipă.

- a. De ce este importantă?

Evită coliziuni și erori, asigură un flux de lucru clar și coerent, ușurează integrarea noilor membri și ajută la gestionarea lansărilor de software.

- b. Ce influențează strategia?

Mărimea echipei, complexitatea proiectului, modul de colaborare și frecvența lansărilor.

- c. Recomandare

- Documentează strategia (ex. în README sau [CONTRIBUTING.md](#)).
- Respectă convenția pentru eficiență



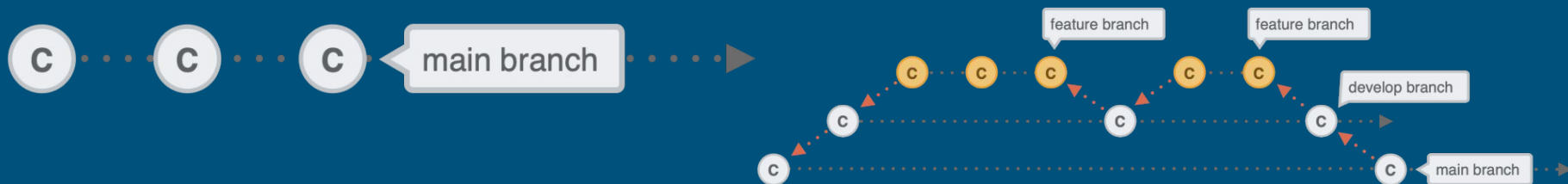
## 3.2.2 Integrarea schimbărilor și structurarea lansărilor

1. Mainline Development - „Integrează întotdeauna”
  - Folosește mai puține ramuri.
  - Are mai puține commit-uri
  - Se bazează pe testare riguroasă și standarde stricte de calitate

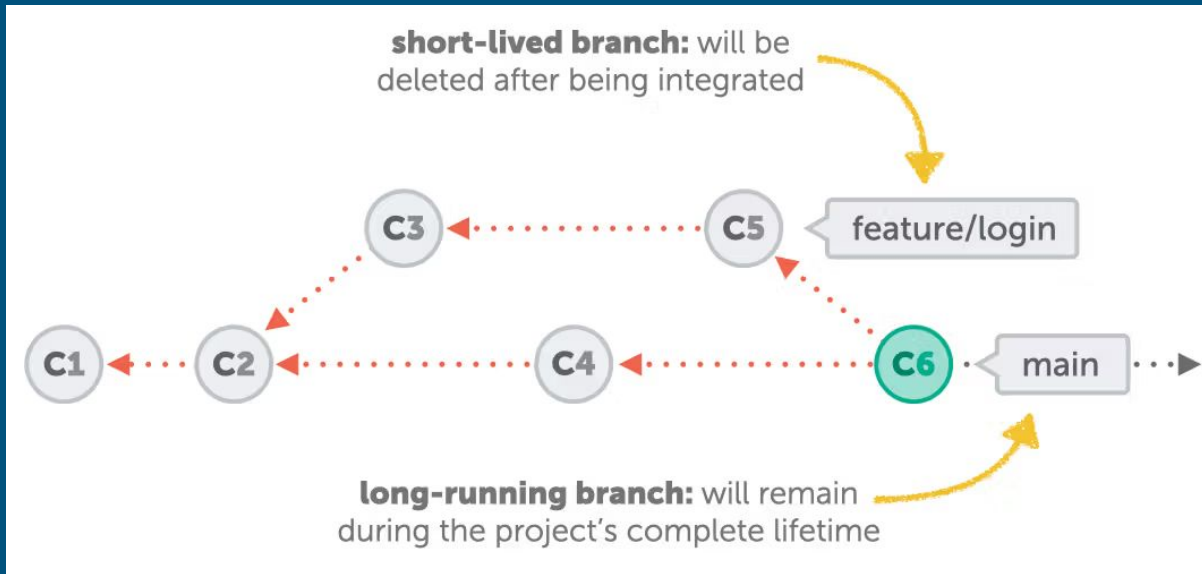
### 2. State, Release și Feature Branches

- a. Există tipuri diferite de ramuri, fiecare cu un rol clar:
  - Feature branches pentru dezvoltarea noilor funcții sau experimente
  - Release branches pentru planificarea și gestionarea lansărilor
- b. Ajută la organizarea clară a muncii și a procesului de lansare

Strategia aleasă depinde întotdeauna de nevoile și cerințele echipei și proiectului!



# Ramuri de lungă durată și de scurtă durată



## 3.2.3 Ramuri de lungă durată

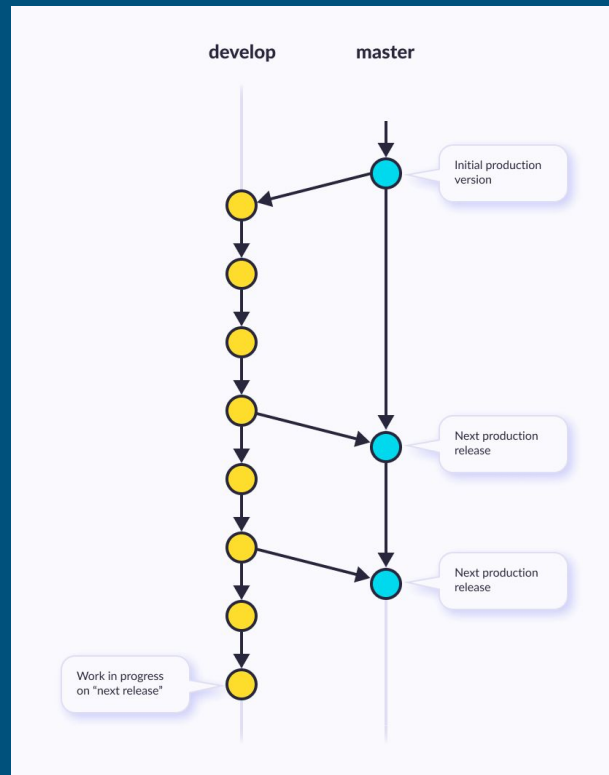
- Există pe toată perioada proiectului.
- Oglindesc etapele ciclului de viață al dezvoltării
- **Regulă de bază:** Niciun commit direct! Modificările ajung aici doar prin integrare ( merge sau rebase)

Motive:

- a. Asigură calitatea codului – doar cod testat și revizuit ajunge în aceste ramuri (exemplu - producție)
- b. Facilitează gruparea și programarea lansărilor planificate

Exemple:

- main ( ramura principală)
- Ramuri de integrare develop, staging – reflectă stări din procesul de dezvoltare/lansare.



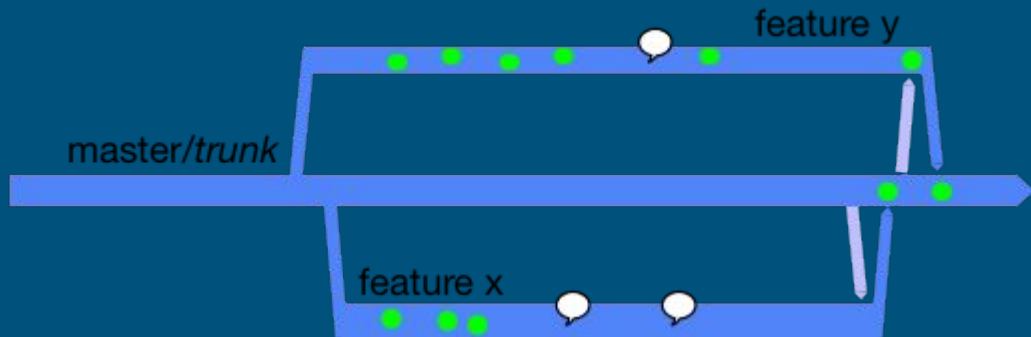
## 3.2.4 Ramuri de scurtă durată

- Create pentru scopuri specifice și **șterse după integrare.**
- **Se bazează pe o ramură de lungă durată.**

Exemple de utilizare:

- Funcționalități noi (feature branches)
- Corecții de bug-uri
- Refactorizări sau experimente

- După finalizarea muncii și integrarea prin merge / rebase, aceste ramuri se șterg.





# Concluzie

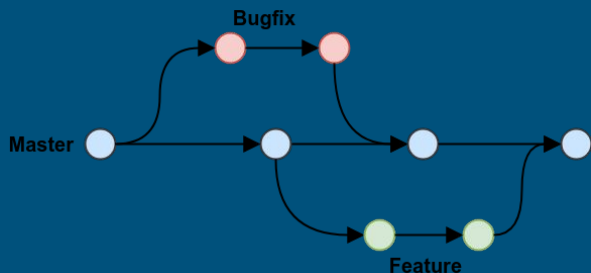
---

Structurarea clară a ramurilor ajută la menținerea calității codului și la o gestionare eficientă a procesului de dezvoltare.

## 3.2.5 Exemple de Strategii de Ramificare

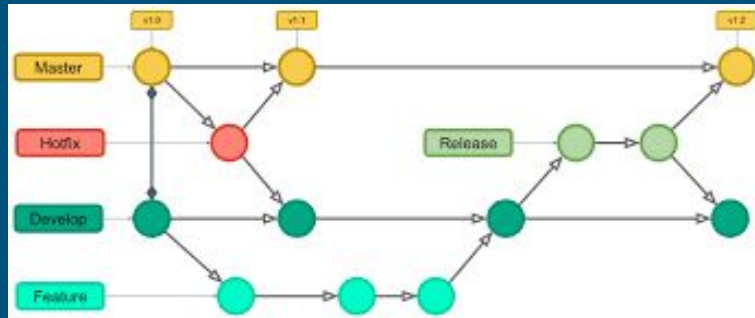
### GitHub Flow

- Recomandat pentru un flux de lucru simplu și flexibil
- **O singură ramură de lungă durată**: main
- Ramuri de scurtă durată pentru orice modificare: funcționalități, bugfix-uri, refactorizări
- Fiecare lucru activ se face într-o ramură separată, temporară



### GitFlow

- Oferă mai multă structură și reguli.
- Ramuri de lungă durată:
  - a. main
  - b. develop
- Ramuri de scurtă durată
  - a. funcții (feature branches)
  - b. lansări (release branches)
  - c. corecturi urgente (hotfix branches)



## 3.2.6 Care este cel mai „bun” model de ramificare?

---

**Nu există un model universal perfect.** Alegerea depinde de:

- **Proiectul** – dimensiunea, complexitatea, tehnologia
- **Ciclul de lansări** – frecvența și modul în care livrezi software
- **Echipa** – mărime, experiență, stil de colaborare

Recomandări:

- **Inspiră-te** din modele consacrate
  - GitHub Flow - simplu, rapid
  - GitFlow - structurat, stabil
  -
- **Adaptează și creează** un model propriu, care se potrivește echipei și nevoilor voastre.
- **Iterează și ajustează** modelul pe parcurs. **Flexibilitatea** este cheia!

# Pull Requests

## 3.3.1 Ce sunt?

---

Pull requests nu sunt o funcționalitate nativă Git, ci o opțiune oferită de platformele de găzduire Git – GitHub, GitLab, Bitbucket etc.

Arată diferit pe fiecare platformă, dar *principiul este același*.

## 3.3.2 Scop principal

Revizuirea și comunicarea despre cod.

- În loc să integrezi direct în main, poți cere colegilor să revizuiască modificările.
- Ideal pentru modificări importante sau complexe.
- După feedback și aprobare -> codul este integrat (merge/rebase)



## 3.3.3 Contribuții externe

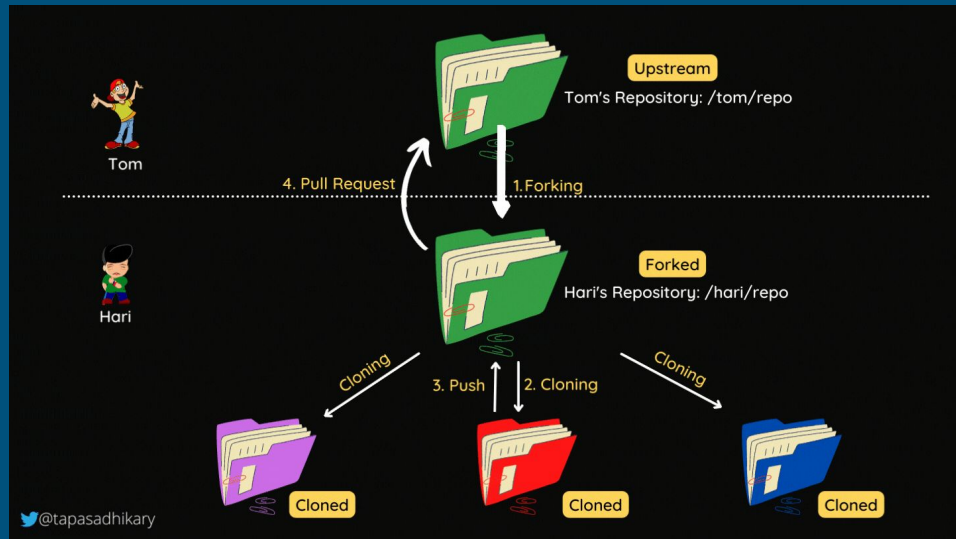
Pull request-urile sunt esențiale când vrei să contribui la un repository la care nu ai acces direct. Pentru a face asta vei putea face un fork.

Fork este copia ta personală a repository-ului.

1. Poți modifica liber codul din această copie
2. După ce ai terminat, creezi un Pull Request către repository-ul original pentru a propune modificările.
3. Apoi, un contributor principal poate:
  - a. Revizui codul tău
  - b. Cere modificări
  - c. Acceptă și integrează contribuția

Concluzie:

Este un mod controlat și sigur de a colabora fără a oferi acces direct la codul principal.



# Merge Conflicts



## 3.4.1 Când apar și ce sunt mai exact?

---

Apar atunci când Git încearcă să îmbine modificări din surse diferite și nu doar la merge, ci și la:

- rebase
- cherry-pick
- pull cu modificări
- stash apply

Nu apar mereu, ci doar când există modificări contradictorii și Git nu poate decide singur care modificare este cea corectă

Aceste conflicte sunt de fapt situații unde aceeași parte din cod a fost schimbată diferit în două locuri.

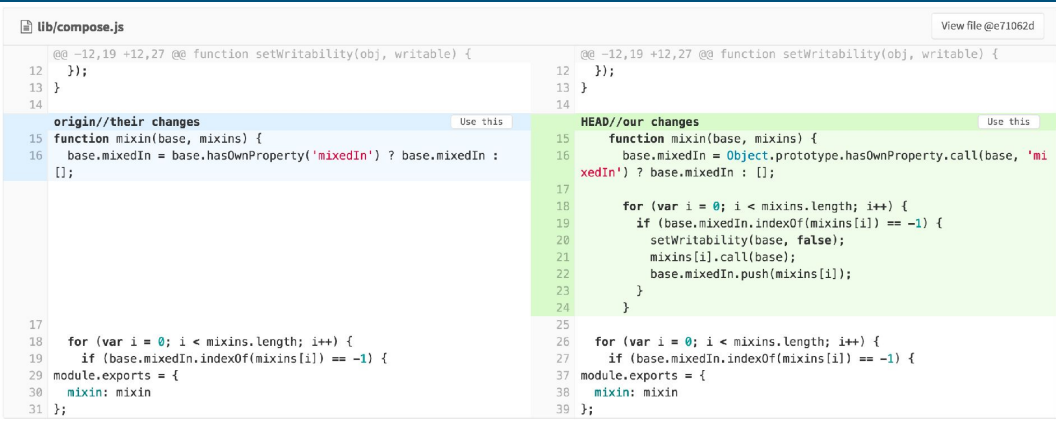
Exemple:

- Aceeași linie de cod modificată în două ramuri
- Un fișier modificat într-o ramură și șters în alta

Git o să te anunțe clar atunci când apare un conflict.

## 3.4.2 Cum se rezolvă?

1. Deschizi fișierele marcate ca fiind în conflict
  2. Git marchează în cod blocurile conflictuale
  3. Alegi versiunea corectă sau le combini manual
  4. Salvezi fișierul, adaugi și finalizezi.
- De asemenea poți folosi un tool vizual de comparare/rezolvare ca Kaleidoscope, VS Code, Meld, etc.



```
lib/compose.js
@@ -12,19 +12,27 @@ function setWritability(obj, writable) {
12  });
13  }
14
15  origin//their changes
16  function mixin(base, mixins) {
17    base.mixedIn = base.hasOwnProperty('mixedIn') ? base.mixedIn :
18    [];
19
20    for (var i = 0; i < mixins.length; i++) {
21      if (base.mixedIn.indexOf(mixins[i]) == -1) {
22        module.exports = {
23          mixin: mixin
24        };
25      }
26    }
27  }
28
29  HEAD//our changes
30  function mixin(base, mixins) {
31    base.mixedIn = Object.prototype.call(base, 'mixedIn') ? base.mixedIn : [];
32
33    for (var i = 0; i < mixins.length; i++) {
34      if (base.mixedIn.indexOf(mixins[i]) == -1) {
35        setWritability(base, false);
36        mixins[i].call(base);
37        base.mixedIn.push(mixins[i]);
38      }
39    }
40  }
41
42  for (var i = 0; i < mixins.length; i++) {
43    if (base.mixedIn.indexOf(mixins[i]) == -1) {
44      module.exports = {
45        mixin: mixin
46      };
47    }
48  }
49  }
50  }
```

# Merge vs Rebase

## 3.5.1 Merge

---

### Cum funcționează un merge?

1. Git caută 3 commit-uri importante:
  - Strămoșul comun: ultimul commit care apare în ambele ramuri.
  - Capetele celor două ramuri care trebuie îmbinate.
2. Când să folosești merge?
  - Când vrei să păstrezi istoricul real al dezvoltării.
  - În echipe mari – oferă transparență despre când și cum au fost combinate ramurile.

### Tipuri de merge

1. Fast-Forward Merge
  - Ramura de destinație nu are commit-uri proprii față de ramura sursa
  - Git mută pur și simplu pointerul – fără creare de commit nou
    - Simplu, clar, dar cu istoric mai puțin explicit.
2. Merge obișnuit (cu commit de merge)
  - Există divergențe între ramuri.
  - Git creează un commit automat, numit merge commit, care îmbină modificările ambelor ramuri
    - Istoricul rămâne ramificat, păstrând ordinea cronologică reală

## 3.5.2 Rebase

---

Reorganizează istoricul commit-urilor pentru a crea o linie dreaptă, mai curată.

Cum funcționează?

1. Git identifică commit-urile care aparțin doar ramurii curente – cele făcute după ultimul strămoș comun.
2. Git le salvează temporar.
3. Git aplica commit-urile de pe ramura țintă peste care vrem să „așezăm” munca noastră.
4. Git „reaplică” commit-urile salvate, ca și cum ar fi fost făcute după cele din main.

Rezultatul: o istorie liniară, ca și cum modificările au fost făcute direct peste ramura principală.

Când folosim și când nu?

- Fiecare commit reaplicat primește un hash nou.
- **NU face** rebase pe commit-uri deja împinse într-un repository partajat.
- **IDEAL** pentru curățarea istoricului local înainte de a face merge într-o ramură importantă.

## 3.5.3 Rebase vs Merge

---

	Rebase	Merge
Istoric	Liniar, curat	Ramificat, realist
Commit nou?	NU (rescrie commit-urile)	DA (merge commit automat)
Ușor de urmărit?	Bun pentru proiecte mici	Bun pentru colaborări mari
Siguranță	Doar local, înainte de push	Oricând, fără ris

Mulțumesc pentru vizionare!

---