Curs GIT, Virtualizare, Containerizare, Jenkins

Curs 4. Git2

Coordonator:
dr. conf. ing. Serban Obreja

Autori:
ing. Cosmin Cimpoeru, ing. Ciprian Chende

Versiune

Versiune	Cine	Data	Descriere schimbare
v3	Ciprian Chende	2023/03/19	Adaugat continut. Varianta completa
v2	Ciprian Chende	2023/02/16	Completat continut.
v1	Ciprian Chende	2023/02/18	Cuprins si continut

Curs 4
Git 2

Curs 4 GIT 2. Cuprins

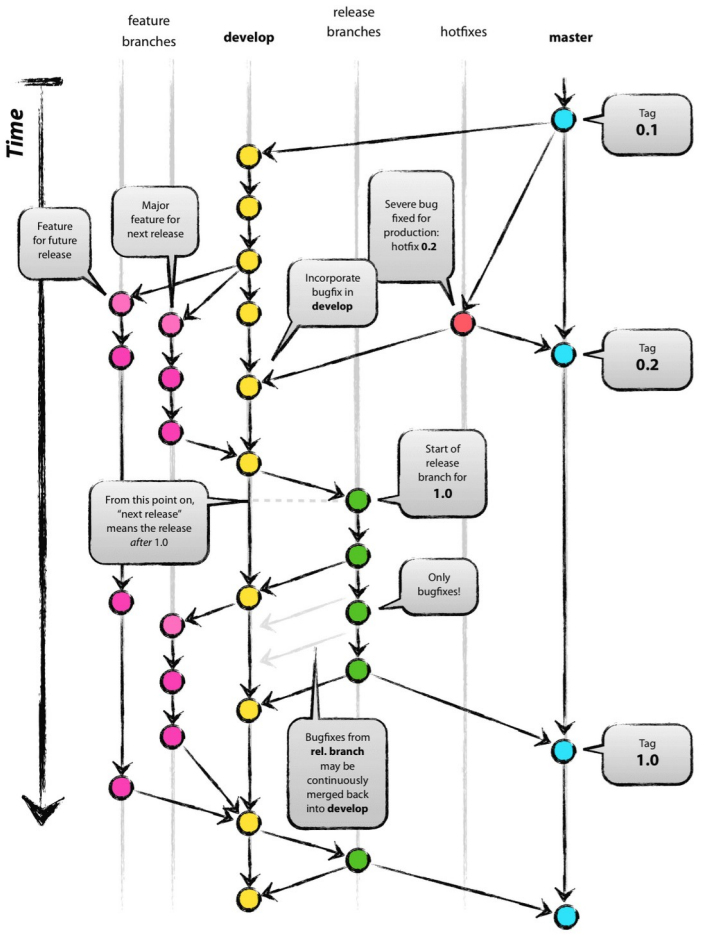
- Configuratie necesara pentru curs
- Scheme / strategii de branching
  - “git-flow”
  - “GitHub flow”
  - Alte scheme de branching. Ce alegem?
- Servere GIT: GitHub, Bitbucket, GitLab etc. De Retinut!
- Integrare modificare dintr-un branch in altul cu `git rebase`
- Configurare git sa ignore fisiere / directoare care nu sunt de interes
- Lucru colaborativ in GitHub
  - Adaugare colaboratori la repository
  - Adaugare cod de catre mai multi programatori
  - Creere “Pull Request – PR” pentru integrare modificari
  - Review si aprobare PR
  - Discutie despre reguli de limitare a accesului la branch-uri – ex. Integrare in ‘main’ / ‘master’ doar cu Pull Request (PR)
  - Utilizare site ‘github.com’ pentru a semnala probleme (issues) si le aloca spre rezolvare programatorilor care contribuie la cod
- Recaptiulare
- Utilizare IDE cu extensie pentru git. Exemplu Visual Code Studio.
- Tematica examen
- Bibliografie

Configuratie necesara pentru curs

- Laptop cu Linux sau masina virtuala linux
- Acces la internet pentru instalare
- Cont GitHub: https://github.com
- Editor cod: Linux: vi / vim, gedit (www.gedit.org) (se pot instala cu apt sau din Synaptic),
  Windows: notepad, notepad++ (Windows)
  Linux, Windows, Mac:
  Visual Studio Code: https://code.visualstudio.com
  (Download fisier `.deb`; Instalare `cu apt install <fisier>`)

Scheme de ‘branching’. “`git-flow`”

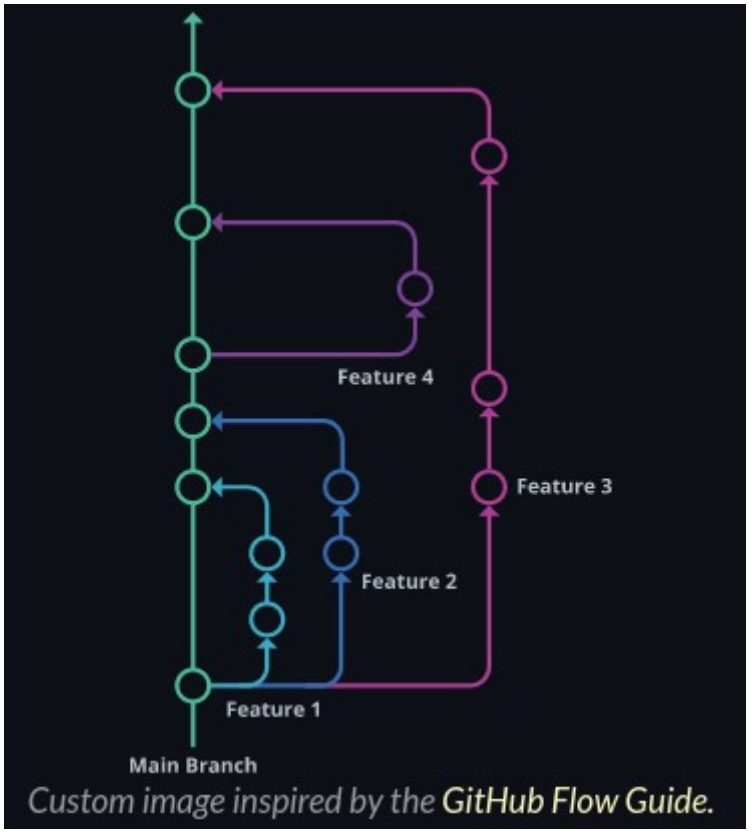
- Schema de branching Vincent Driessen in 2010
- https://nvie.com/posts/a-successful-git-branching-model/
- 5 tipuri de branch-uri:
  - master / main (noua denumire)
  - develop
  - release
  - Hotfixes
  - Feature
- Schema care se preteaza pentru proiecte care trebuie sa suporte mai multe versiuni sau versiuni explicite
- Pune anumite dificultati pentru proiecte puternic orientate pe ‘continuous integration (CI)’ datorita faptului ca contine multe tipuri de branch-uri



Scheme de ‘branching’. “GitHub-flow”

- Schema de branching recomandata de GitHub: https://docs.github.com/en/get-started/quickstart/github-flow
- Reprezentare schematica:

  https://www.gitkraken.com/learn/git/best-practices/git-branch-strategy ...
- Doua tipuri de branch-uri:
  - master / main
  - Feature
- Se preteaza mai bine la proiecte puternic orientate pe integrare continua (CI).
  Sunt mai putine branch-uri si mai putine ‘locuri’ unde sa inseram actiuni de CI.
  ex. Executie pipeline CI la PR in main din oricare branch de tip feature



Scheme si … scheme ... de ‘branching’

- Exista mai multe scheme / strategii de branching
- Putem mentiona:
  - Trunk-based
  - GitLab-flow

  etc.

  (https://www.flagship.io/git-branching-strategies/)
- Ce alegem?

- Ar fi de urmat sfatul celui care a crea git-flow:

  “*To conclude, always remember that panaceas don't exist.*
  Consider your own context.
  Don't be hating.
  Decide for yourself.”

  Vincent Driessen (*https://nvie.com/posts/a-successful-git-branching-model/*)

Servere GIT: GitHub / Bitbucket / GitLab etc. ***De retinut!***

- Pun la dispozitie un mediu de lucru care permite:
  - **Partajare resurse**
  - **Colaborare**

Integrare modificari prin ‘`git rebase`’

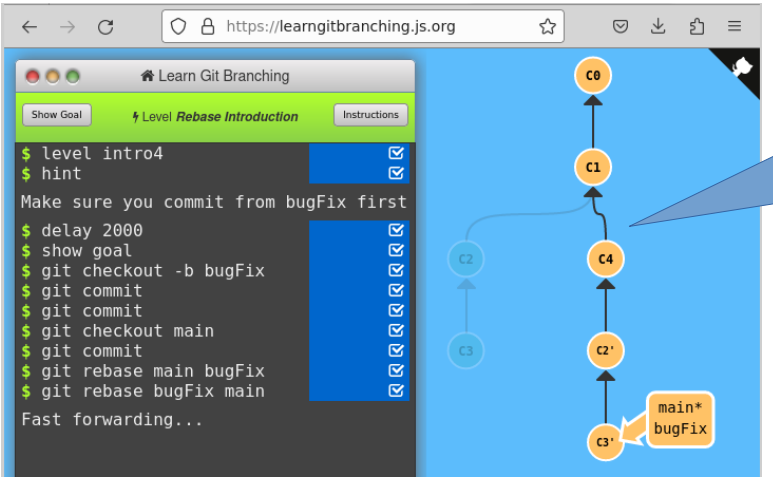
- Facem modificari in **devel** si le adaugam cu `git add si git commit`
- Trecem pe **master**/**main**, facem modificari si le adaugam din nou cu `git add si git commit`
- Putem adude modificarile din devel in master si cu comanda `git rebase master devel`, nu doar cu comanda `git merge devel` (data cu branch-ul master activ)
- Prin rebase nu se mai creaza ‘bucle’ in istoricul commit-urilor. Commiturile sunt luate din branch-ul sursa si adaugate in branch-ul destinatie.
- Daca sunt conflicte la unul din commit-uri, acestea trebuie rezolvate la acel commit iar apoi rebase-ul trebuie continuat cu comanda `‘git rebase –continue’`.
  Daca mai sunt commit-uri de integrat dupa rezolvarea conflictului, se poate folosi:
  `’git commit --amend’` pentru a amenda commit-ul curent
- Ex exercitiu: Repository: dir_local_cu_git
  - in branch-ul ‘devel’ facem doua modificari:
    - adaugat linia: print('modificare 2 - branch devel') + git add + git commit
    - adaugat linia: print('modificare 2 - branch devel') + git add + git commit
  - in branch-ul ‘master’ facem o modificare:
    - adaugat linia: print(“modificare 2 branch master') + git add + git commit
  - in acest moment avem o ramificatie – punct de divergenta
  - Comanda de integrare este `git rebase master devel`
  - La integrarea primului commit din devel in master apare un conflict care trebuie rezolvat
  - Se continua apoi rebase-ul cu `git rebase --continue`

    `(se poate folosi si git commit --amend pentru a modifica mesajul de commit)`

```
cip@cipasus:~/git/dir_local_cu_git$ git log --all --oneline --graph
* 985dad1 (HEAD -> devel, master) rebase continue
* 327ef2e modificare 2 branch master
*   f8d3c51 Merge branch 'devel'
|\
| * 6e04a20 modificare 1 devel
* | f964fc3 modificare 1 branch master
|/
* 4cc3a94 adaugare in master
* 02a941e adaugare line 2
* 0cde98c adaugare linie 1
* 304771f test
cip@cipasus:~/git/dir_local_cu_git$
```

Integrare modificari din devel in master cu git rebase

Integrare cu git merge. Se observa bucla in istoricul commit-urilor



Vizualizare grafica si comenzi git pentru rebase pe learngitbranching.js.org

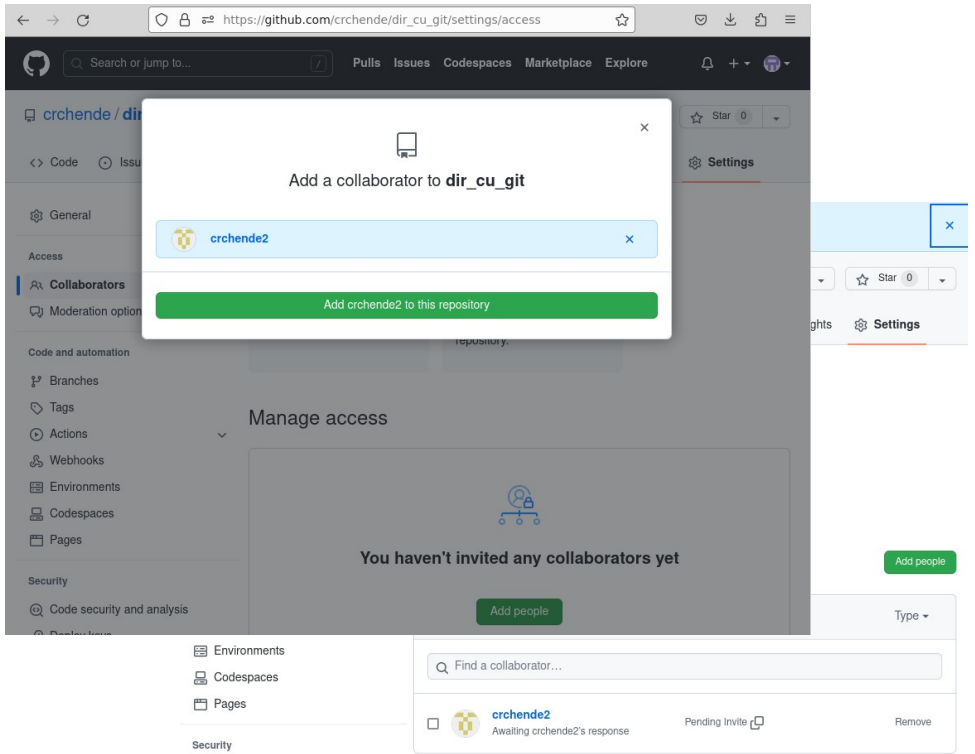
Configurare git sa ignore anumite fisiere / directoare

- Fisier `.gitignore` in directorul de lucru – acelasi director care are si directorul `.git`
- Fisier `.git/info/exclude`
- Aceste fisiere cuprind sabloane de nume fisier/director care vor fi ignorate de git
- MOTIV – Nu vrem sa adaugam pe git fisiere backup, fisiere compilate etc
- Exemple intrari in aceste fisiere:
  - *.o
  - *.[oa]
  - __pycache__
  - .gitignore
  - .venv

    etc
- Testare functionare:
  - Creati fisierul .gitignore si adaugati in el una din liniie de mai sus
    - Creati un fisier care sa aiba, de exemplu extensia '.o' – test.o
    - Dati comanda `git status`. Fisierul test.o nu ar trebui sa apara ca fisier modificat
    - Stergeti linia din .gitignore. Dati comanda `git status.` Ar trebui sa vedeti fisierul test.o ca fisier nou adaugat.
  - O alta optiune este adaugarea / stergerea de linii similare din fisierul `.git/info/exclude`
  - Oricare varianta este buna. Varianta cu fisierul .gitignore in directorul de lucru este in opinia mea, mai explicita.

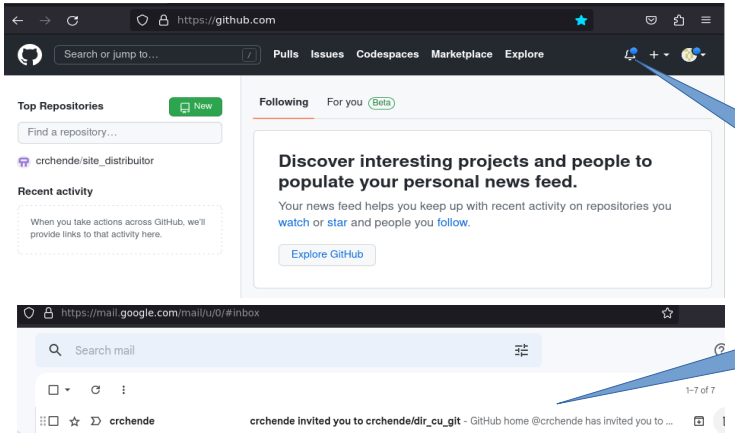
Lucru colaborativ. Adaugare colaboratori

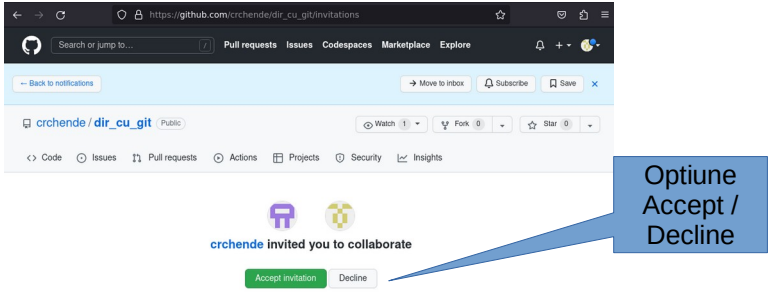
- Programatorul 1:
- Login pe GitHub, selectare repository
- Adaugare colaborator



- Programatorul 2:
- Login pe GitHub. Click pe pictograma GitHub
- Notificare in GitHub ca este invitat sa fie colaborator

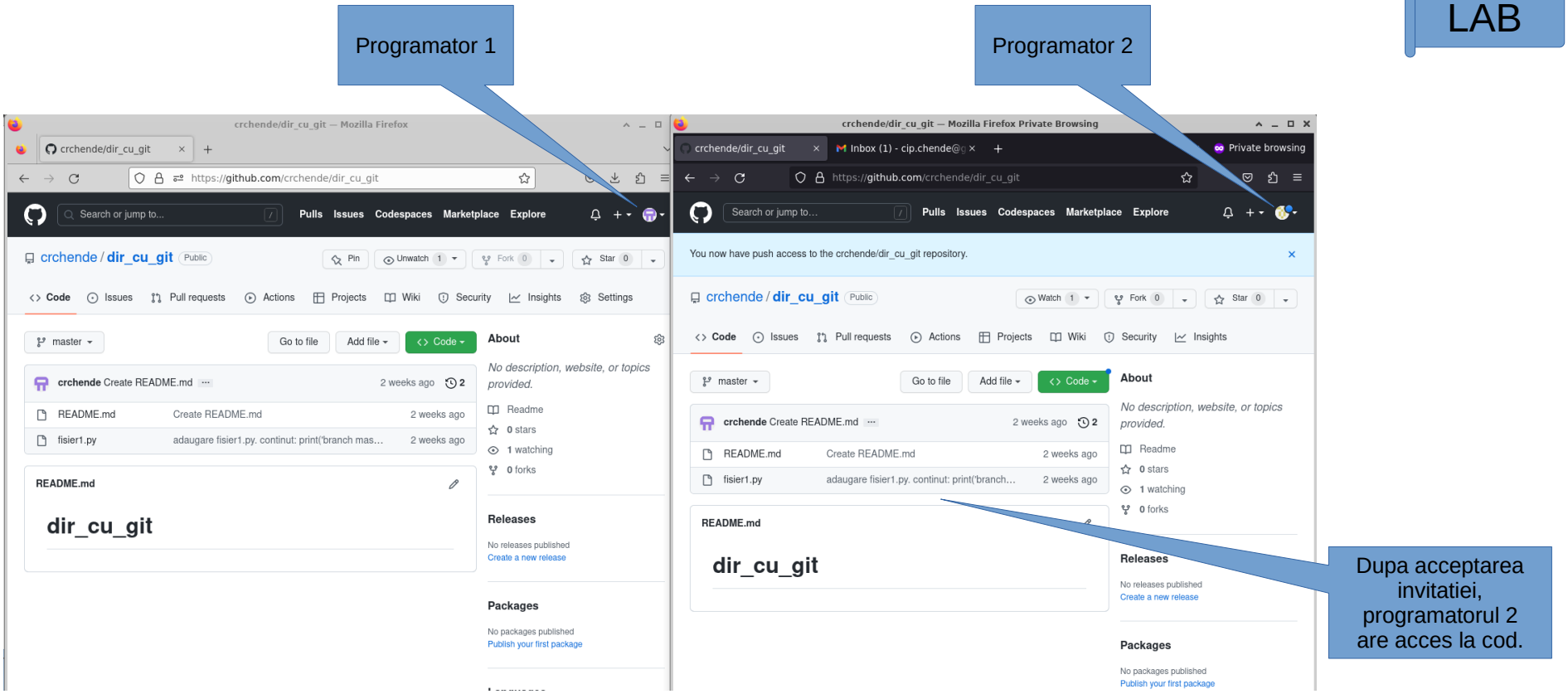
LAB





Lucru colaborativ. Adaugare colaboratori (2)

LAB



Lucru colaborativ. Adaugare colaboratori

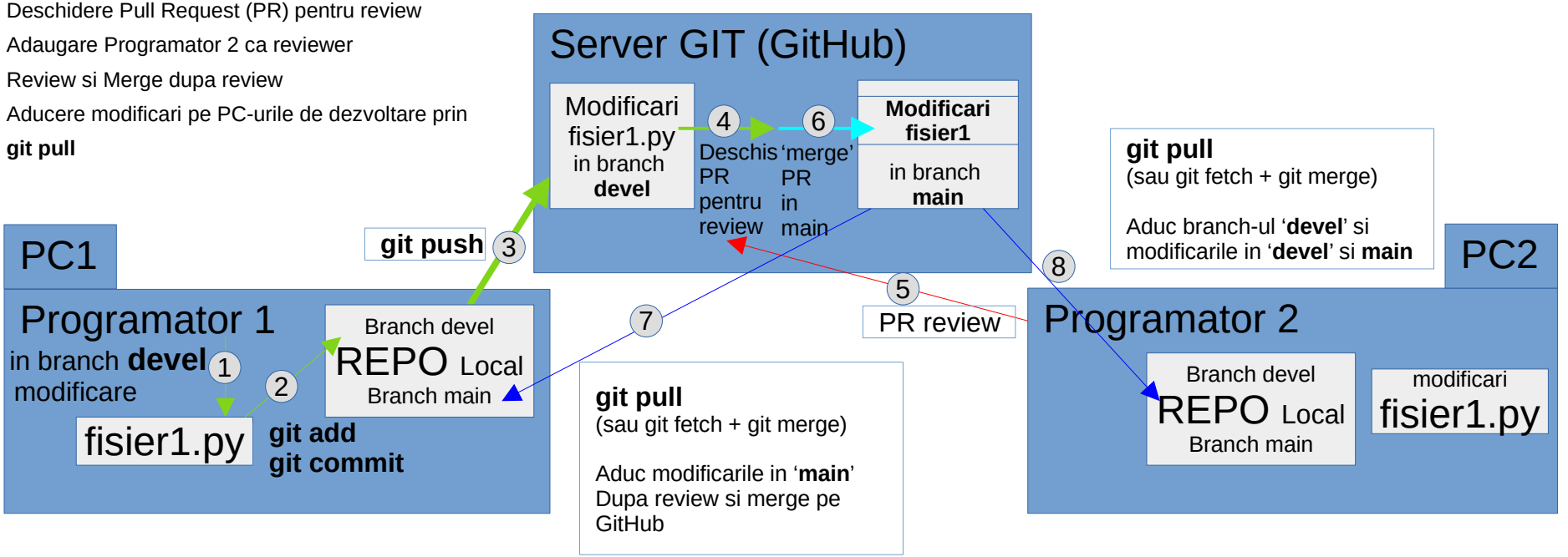
LAB

- Mai multi oameni pot contribui la dezvoltarea continutului unui repository
- Proprietarul repository-ului, poate sa-i invite
- Este nevoie de cont GitHub
- Cei invitati primesc notificare in GitHub si pe email
- Cei care accepta invitatia primesc acces la cod – pot vedea repository-ul partajat
- Pot clona repository-ul local si pot sa adauge continut
- Pot avea doar rol de review / mici modificari – actiuni care se fac din GitHub. Clonarea repository-ului fiind optionala in acest caz.
- Pentru cei care adauga continut – programatori, pentru proiecte de prograre, fiecare colaborator trebuie sa-si configureze ‘personal access token’ pentru a putea adauga codul pe GitHub
- Pentru repository-urile private – doar colaboratorii le vad
- Pentru repository-urile publice – toata lumea le vede, poate face clone. Doar colaboratorii pot modifica cod.
- Conturile ‘Enterprise’ (platite) de GitHub au posibilitatea de a configura accesul si drepturile colaboratorilor pentru a se asigura ca se respecta regulile de colaborare agreate de grup
- Conturile ‘Publice’ (pe gratis) au optiuni limitate de a configura accesul colaboratorilor

Lucru colaborativ. Adaugare cod.
Reprezentare schematica

LAB

- Modificare fisier in branch ‘devel’
- Adaugare modificare pe server: **git push**
- Deschidere Pull Request (PR) pentru review
- Adaugare Programator 2 ca reviewer
- Review si Merge dupa review
- Aducere modificari pe PC-urile de dezvoltare prin **git pull**



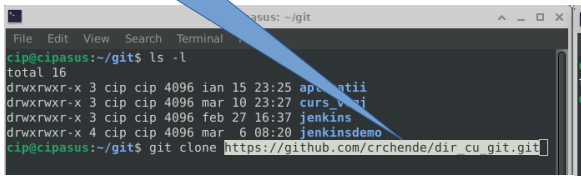
Lucru colaborativ. Adaugare cod (1)

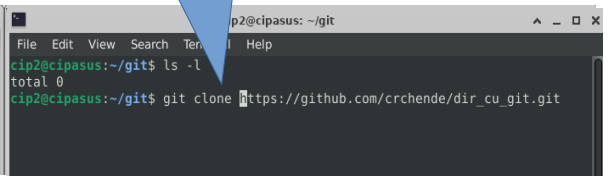
Modificare fisier in ‘devel’ si adaugare modificare pe server.

LAB

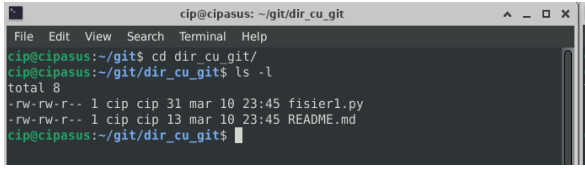
Programator 1
Clonare
repository

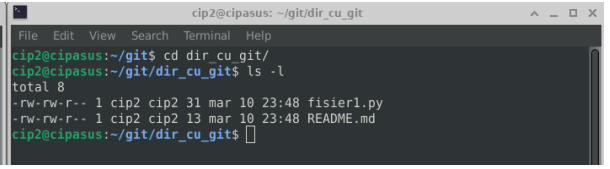
Programator 2
Clonare
repository



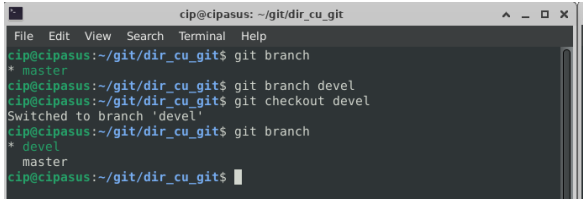


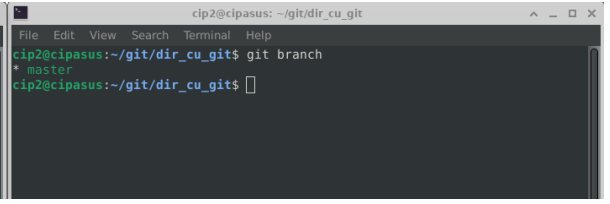
Listare
continut
director
clonat





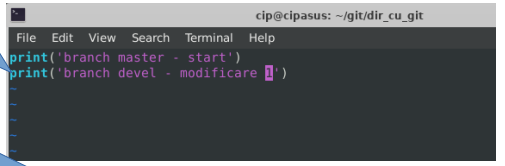
Prog. 1
Creaza
branch-
ul devel



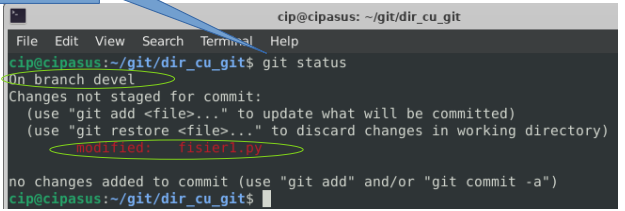


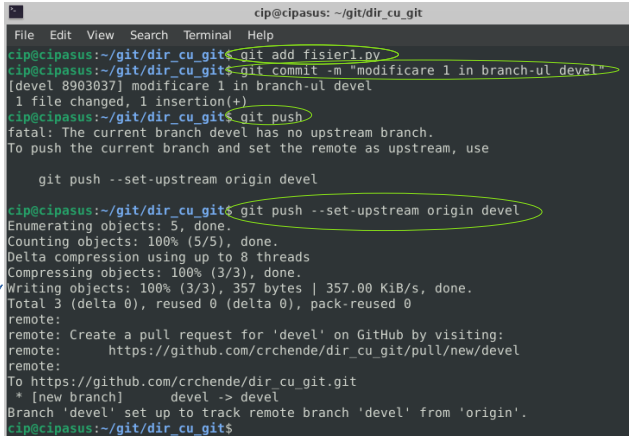
Programator 1
git add
git commit
git push –set-upsteram origin devel # doar prima data dupa adaugare branch nou
(dupa adaugarea branch-ului devel pe server, este suficient doar ‘**git push**’)

Prog. 1
Adauga
linia 2



Prog. 1
git status





Lucru colaborativ. Adaugare cod

LAB

- `Vizualizare modificari in GitHub si creere Pull Request`

Prog. 1 Vizualizare repo in GitHub

Branch-uri

Mesaj ca sunt diferente in tre branch-uri. Link creere PR

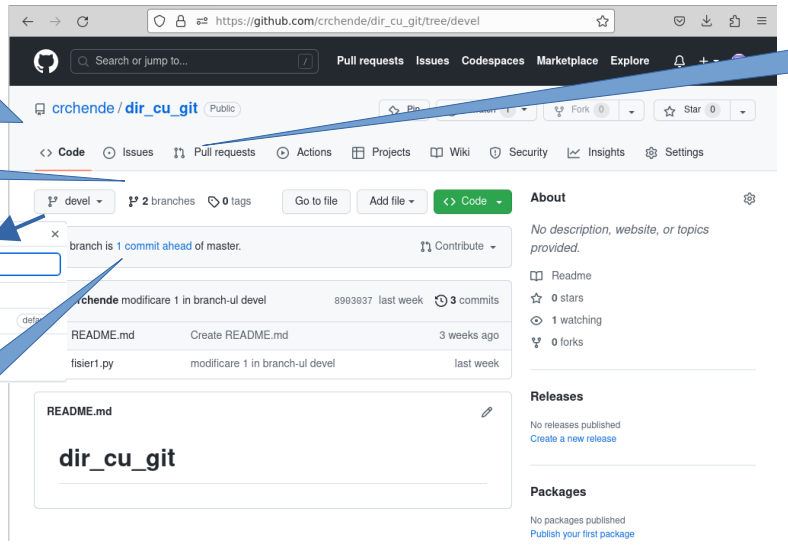
Prog. 2 Vizualizare repo in GitHub

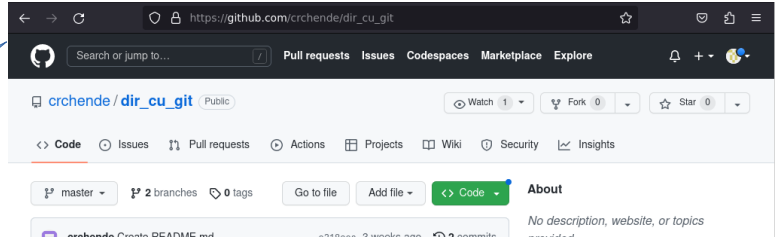
Meniu Pull Requests

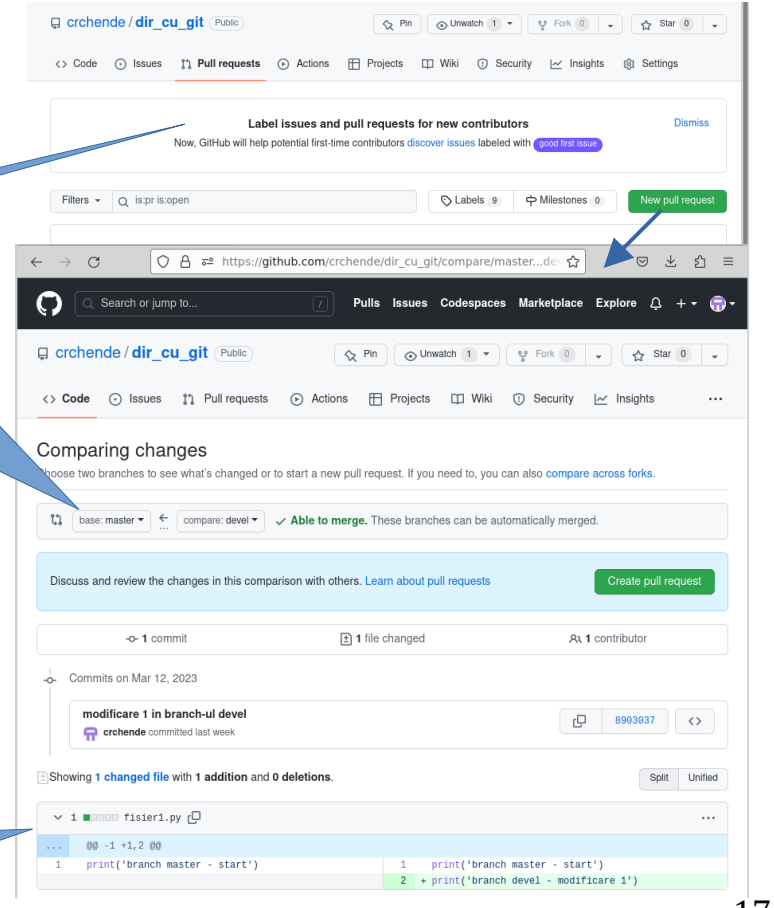
Pagina PR

Selectare sursa → destinatie pentru Pull Request din devel in Master

Vizualizare Diferente



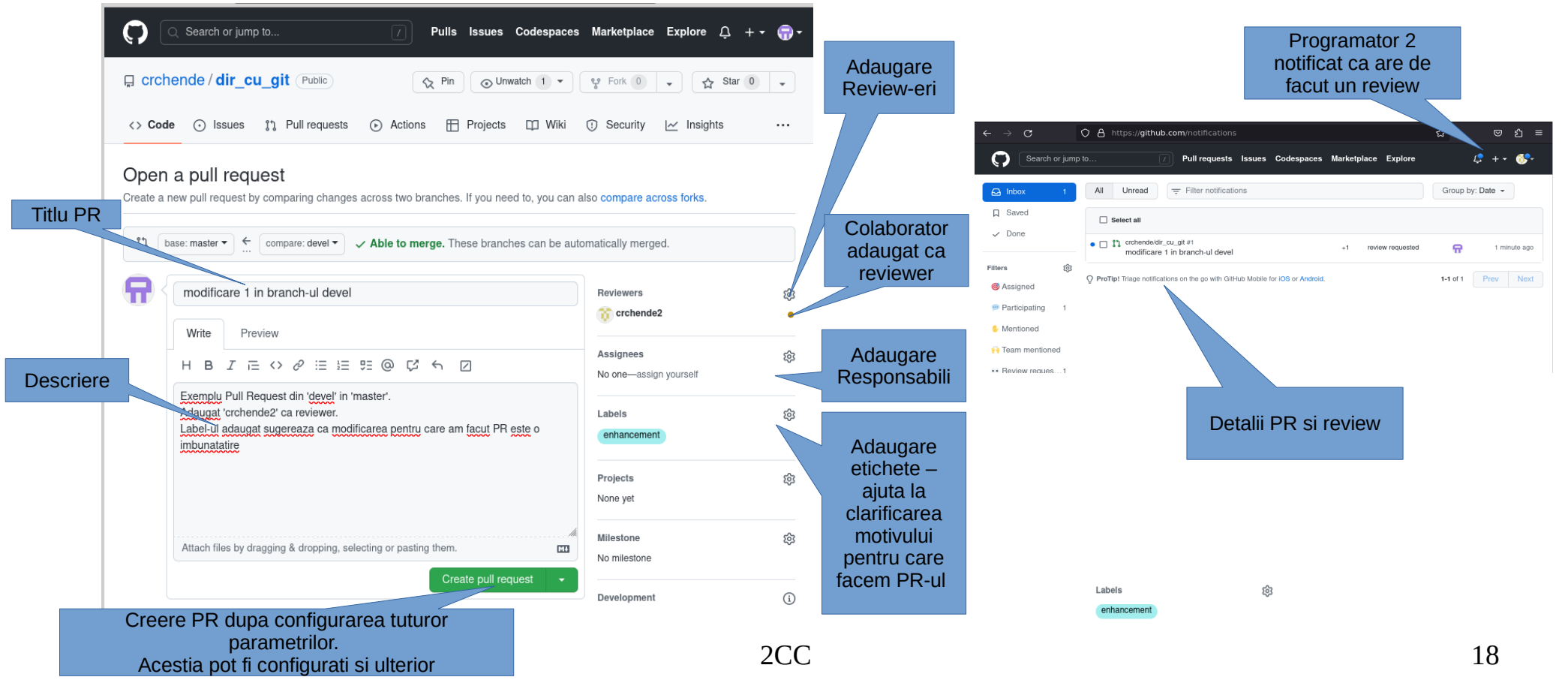




Lucru colaborativ. Pull Request (PR)

LAB

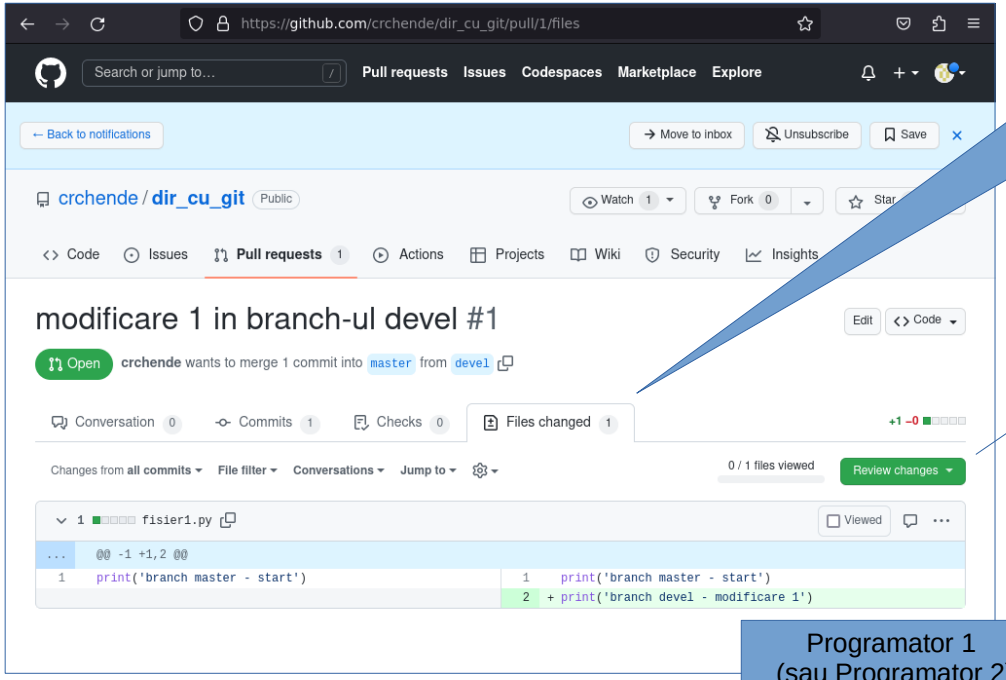
- `Pagina creere Pull request (PR)`



Lucru colaborativ. Review cod in “PR”

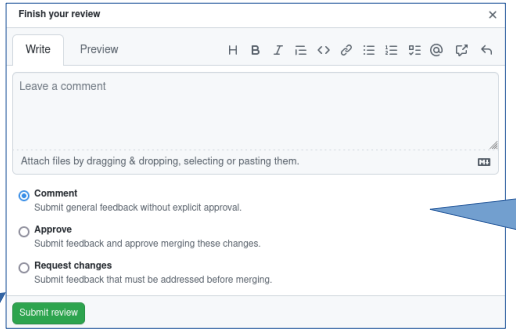
- Adaugare review si vizualizarea acestuia

LAB



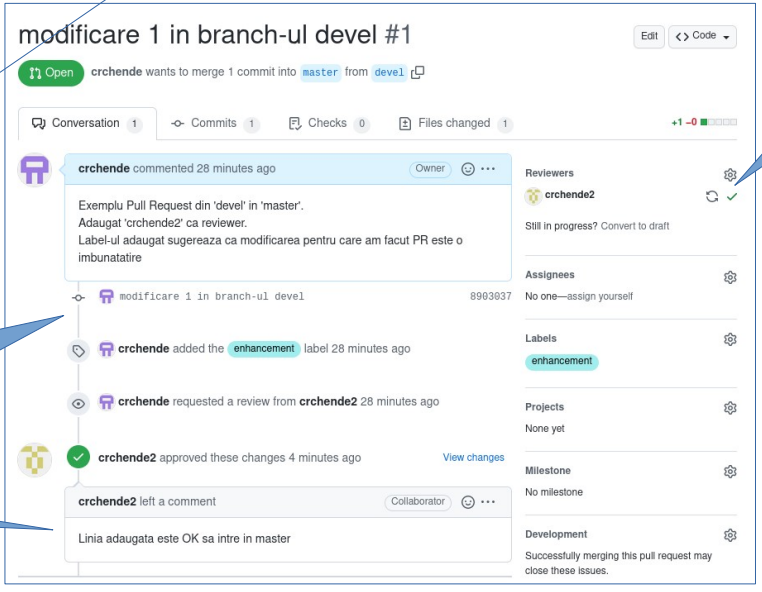
Programator 2 (reviewer-ul) Vizualizare PR

Vizualizare fisiere modificate



Programator 2 (reviewer-ul) Fereastra Review

Confirmare aprobare la review

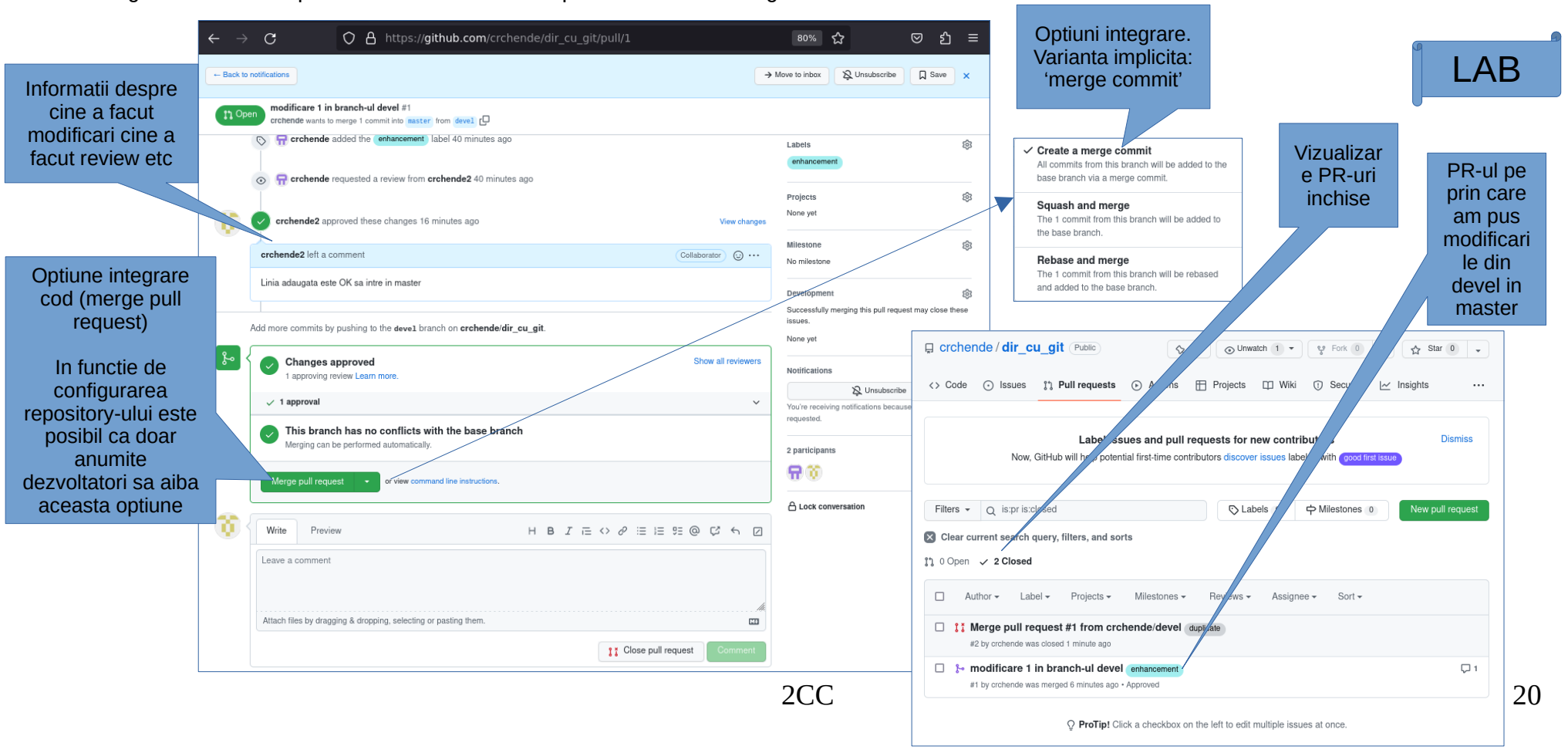


Programator 1 (sau Programator 2) Pagina dupa adaugare review

Comentariu adaugat la review

Lucru colaborativ. Integrare cod / ‘merge’ PR

- Pagina Pull Request. Vizualizare aprobari si integrare cod



Sincronizare directoare de lucru dupa ’merge’ ‘Pull request’

Programator 1

Programator 2

Vizualizare diferente intre GitHub si repository si director de lucru local

```
cip@cipasus: ~/git/dir_cu_git
File  Edit  View  Search  Terminal  Help
cip@cipasus:~/git/dir_cu_git$ git remote show origin
* remote origin
  Fetch URL: https://github.com/crchende/dir_cu_git.git
  Push  URL: https://github.com/crchende/dir_cu_git.git
  HEAD branch: master
  Remote branches:
    devel  tracked
    master tracked
  Local branches configured for 'git pull':
    devel  merges with remote devel
    master merges with remote master
  Local refs configured for 'git push':
    devel  pushes to devel  (up to date)
    master pushes to master (local out of date)
cip@cipasus:~/git/dir_cu_git$
```

```
cip2@cipasus: ~/git/dir_cu_git
File  Edit  View  Search  Terminal  Help
cip2@cipasus:~/git/dir_cu_git$ git remote show origin
* remote origin
  Fetch URL: https://github.com/crchende/dir_cu_git.git
  Push  URL: https://github.com/crchende/dir_cu_git.git
  HEAD branch: master
  Remote branches:
    devel  new (next fetch will store in remotes/origin)
    master tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (local out of date)
cip2@cipasus:~/git/dir_cu_git$
```

Aducem modificarile de pe server local cu ‘git pull’

```
cip@cipasus: ~/git/dir_cu_git
File  Edit  View  Search  Terminal  Help
cip@cipasus:~/git/dir_cu_git$ git pull
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 635 bytes | 635.00 KiB/s, done.
From https://github.com/crchende/dir_cu_git
   e218cea..e2f337e  master     -> origin/master
Updating e218cea..e2f337e
Fast-forward
 fisier1.py | 1 +
 1 file changed, 1 insertion(+)
cip@cipasus:~/git/dir_cu_git$
cip@cipasus:~/git/dir_cu_git$
cip@cipasus:~/git/dir_cu_git$
```

```
cip2@cipasus: ~/git/dir_cu_git
File  Edit  View  Search  Terminal  Help
cip2@cipasus:~/git/dir_cu_git$ git pull
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 960 bytes | 960.00 KiB/s, done.
From https://github.com/crchende/dir_cu_git
   e218cea..e2f337e  master     -> origin/master
 * [new branch]      devel      -> origin/devel
Updating e218cea..e2f337e
Fast-forward
 fisier1.py | 1 +
 1 file changed, 1 insertion(+)
cip2@cipasus:~/git/dir_cu_git$
```

Integrare modificari din branch-uri diferite
Rezolvare conflicte de integrare

- Scenariu:
  - Programatorul 1 modifica un fisier: fisier1.py
    - Adauga fisierul local: `git add, git commit`
    - Adauga fisierul pe server cu `git push`
  - Programatorul 2 modifica acelasi fisier si-l adauga local (`git add, git commit`)
    - Imediat dupa ce Programatorul 1 a adaugat modificarea e server, incearca sa adauge modificarea cu `git push`
    - Problema la adaugarea pe server cu `git pull`
    - Programatorul 2 trebuie sa faca ia modificarile de pe server cu ‘`git pull`’ (care este echivalent cu `git fetch + git merge`)
    - Trebuie sa rezolve posibilele conflicte
    - Deabia dupa acesti pasi, modificarea poate fi adaugata pe server cu ‘`git push`’

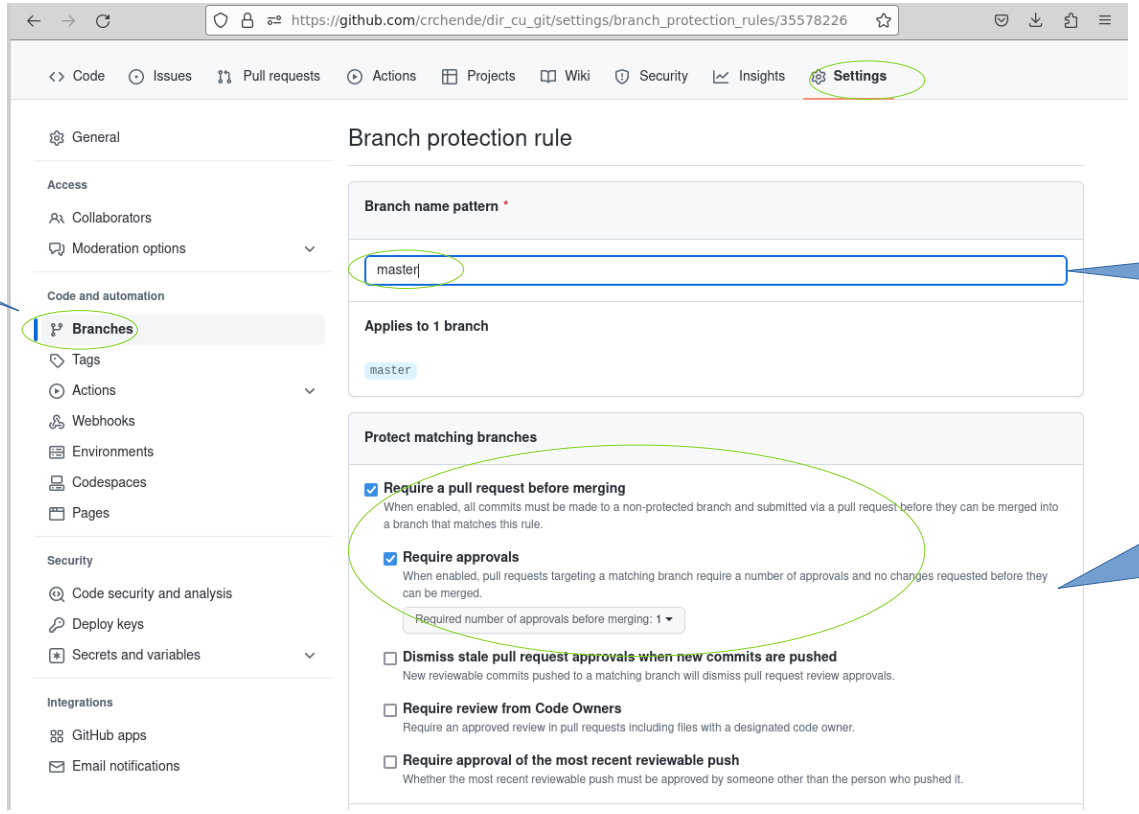
NOTA: `git pull = git fetch + git merge`

`git pull` aduce modificarile de pe server local si face si merge

`git fetch` aduce modificarile de pe server local. Nu integreaza aceste modificari cu cele locale. Integrarea trebuie facuta printr-o alta comanda `git merge sau git rebase,` explicit de catre programator.

Lucru colaborativ. Reguli protectie branch-uri

Repository-ul poate fi configurat cu reguli de protectie a branch-urilor



LAB

Alegem branch-ul pe care dorim sa configuram reguli

Configuram regulile. Ex: nu se poate face integrare in ‘master’ direct, fara PR. Este obligatoriu un review

Lucru colaborativ. Alte functionalitati:

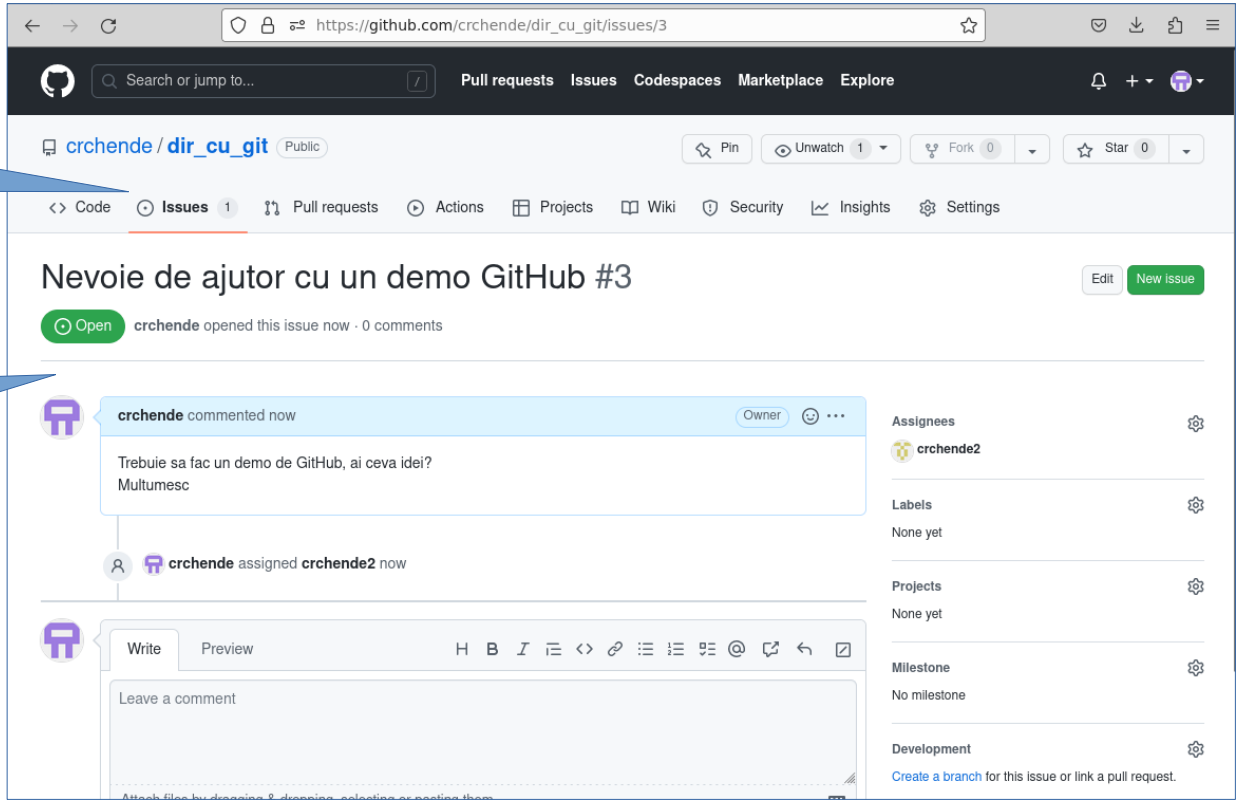
`Pagina 'Issues' – adaugare cereri catre grupul de lucru`

LAB

Un programator are nevoie de ajutor. Poate folosi pagina 'Issues'

Creaza un 'issue' si-l poate aloca altor utilizatori

Pagina 'Issues'
- Tine evidenta problemelor
- Permite distributia acestora catre membrii grupului



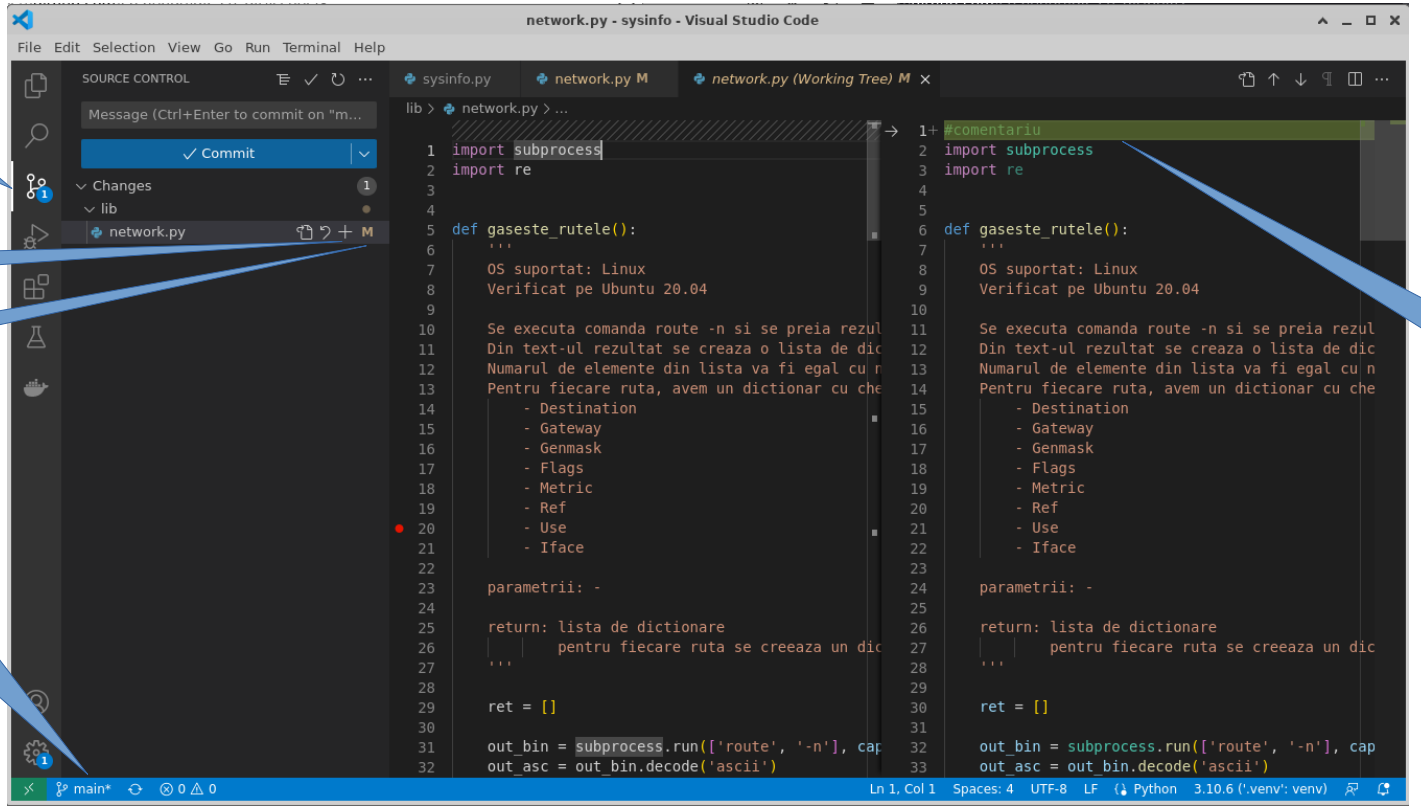
Recapitulare. GitHub

- Modificarile locale trebuie adaugate in repository-ul local cu ‘`git add’ si ‘git commit’`
- Adaugarea pe server se face cu ‘`git push’`
- ‘`git pull’ = ‘git fetch + git merge’` aduce local modificarile de pe server
- Conflicte de ‘merge’ - Programatorii trebuie sa rezolve aceste conflictele, cand apar (modificari in aceeazi zona a acelorasi fisiere)
  - Primul programator care adauga modificarea nu are conflicte de rezolvat
  - Ceilalti trebuie sa preia modificarea cu git pull / git fetch + git merge, sa rezolve conflictul de ‘merge’ iar apoi sa adauge fisierul modificat local cu git push.
- GitHub permite creerea de ‘Pull request’ pentru integrarea ‘pe server’ si pentru review de cod
  - Branch-urile pot fi configurate sa nu permita merge / rebase daca nu exista un ‘Pull request’ deschis
  - Se forteaza astfel un proces care sa includa si partea de review de cod (de dorit)
- Toate commit-urile facute pe durata cat Pull request-ul este deschis vor fi luate in calcul pentru merge
- Pe fiecare Pull Request pot fi adaugati review-eri
- Reguli de protectie a branch-urilor
- Git Hub are functionalitati care permit colaborarea, inglobate atat in Pull Request-uri cat si in Issue-uri. Toti colaboratorii pot sa vada ce se intampla, care sunt problemele care trebuie rezolvate, cine a ridicat problema, daca lucreaza cineva la problema etc.
- Varianta Enterprise a GitHub permite mai multe metode de protectie, restrictii acces etc
- GitHub Actions – este echivalentu-ul Jenkins

Utilizare IDE cu extensie pentru git.

Exemplu Visual Studio Code (vscode)
Modificare in repo: https://github.com/crchende/sysinfo.git

LAB



Vizualiizare GIT

'git add'

Notificare fisier modificat

Branch-ul curent

Adaugare linie: '#comentariu' vizualizare diferente

Tematica examen

1. Creere repository local: `git init.` Director `.git`
2. Adaugare modificari cod in repository local: `git add, git commit`
3. Vizualizare stare director de lucru: `git status`
4. Creere / vizualizare branch-uri: `git branch`
5. Trecere de pe un branch pe altul: `git checkout`
6. Integrare modificari intre branch-uri: `git merge`
7. Clonare repository din GitHub: `git clone <URL>`
8. Configuratie pentru a putea pune modificarile pe server: ‘token’
9. Adugare modificari locale pe server cu ‘`git push`’
10. Aducere locala a modificarilor de pe server cu ‘`git pull`’ - echivalet cu - `git fetch + git merge`
11. Rezolvare probleme de merge – in cazul in care cineva a modificat aceeazi zona de cod ca si dezvoltatorul care tocmai vrea sa faca push - prima data va trebuie sa faca `git pull` sau `git fetch + git merge`, sa rezolve problema de merge, sa faca un nou commit la care sa-i dea push)
12. Vizualizare stare ‘remote’: `git remote show origin` – se vede daca repository-ul local este sincronizat cu cel remote
13. Vizualizare loguri: `git log / git log --all --oneline –graph`
14. Vizualizare diferente: `git diff`
15. Vizualizare configurare `git config –list –show-origin`
16. Vizualizare ‘remote’ `git ls-remote`
17. Pull request – la ce voloseste
18. Reguli de protectie branch-uri
19. Mijloace de cloaborare si organizare a activitatilor pe GitHub: Pull Request / revieweri, Issues
20. Ajutor in linia de comanda

Bibliografie

Git	https://git-scm.com/book/en/v2
Aplicatie WEB invatare git	https://learngitbranching.js.org/
Git flow	http://datasift.github.io/gitflow/IntroducingGitFlow.html
Git flow	https://nvie.com/posts/a-successful-git-branching-model/
GitHub flow	https://docs.github.com/en/get-started/quickstart/github-flow
GihHub flow	https://www.gitkraken.com/learn/git/best-practices/git-branch-strategy ...
Git,GitHub,Trunk flows	https://www.flagship.io/git-branching-strategies/
Exemplu aplicatie 1	https://github.com/crchende/sysinfo
Exemplu aplicatie 2	https://github.com/crchende/site_distribuitor