

Curs GIT, Virtualizare, Containerizare, Jenkins

Curs 4. Git2

Coordonator:

dr. conf. ing. Serban Obreja

Autori:

ing. Cosmin Cimpoeu, 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

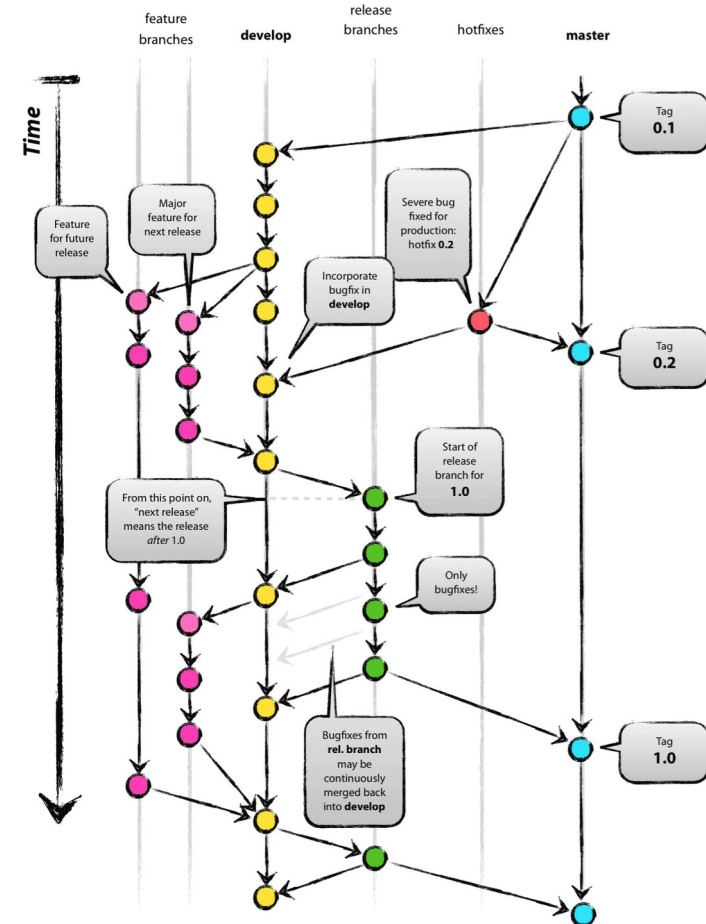
- Configuratie necesara pentru curs
- Scheme / strategii de branching
 - “git-flow”
 - “GitHub flow”
 - Alte scheme de branching. Ce alegem?
- Servere GIT: GitHub, Bitbucket, GitLab etc. De Retinut!
- Integrare modificare dintr-un branch in altul cu `git rebase`
- Configurare git sa ignore fisiere / 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>)

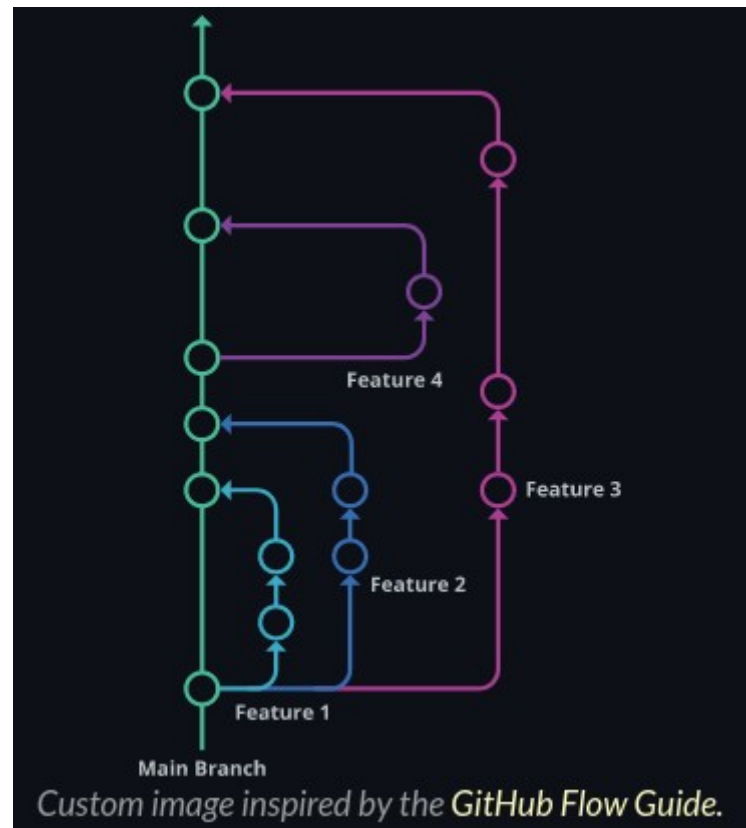
Schema de 'branching'. "git-flow"

- 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 ...](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

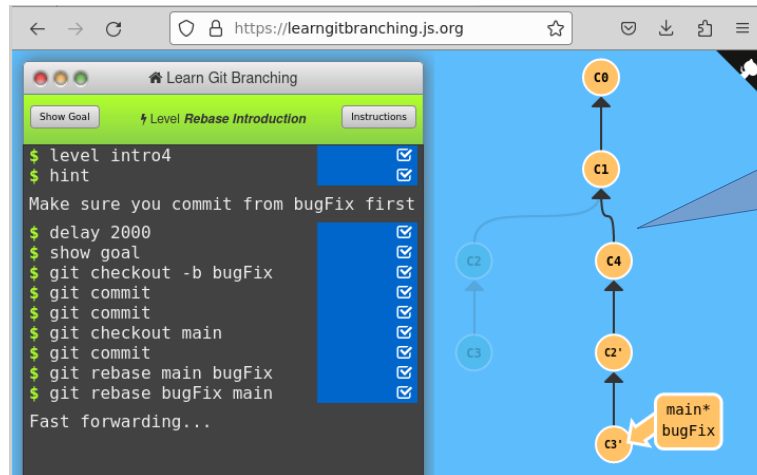
Integrare modificari prin 'git rebase'

- 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 aduce 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



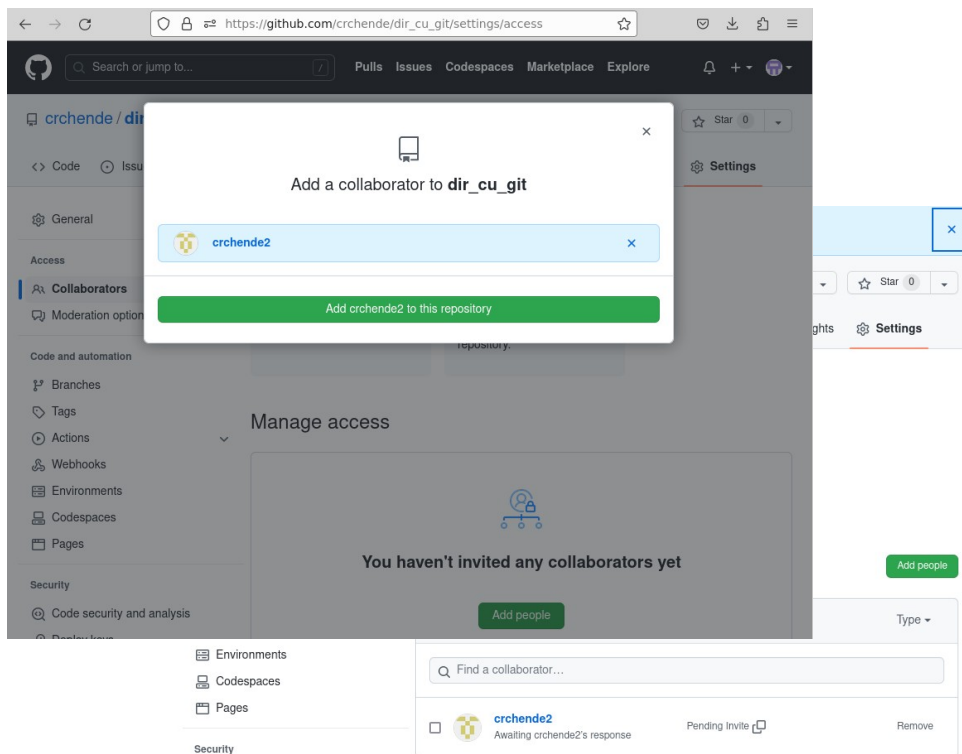
Vizualizare
grafica si
comenzi git
pentru rebase pe
learnitbranching
.js.org

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 liniile 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.

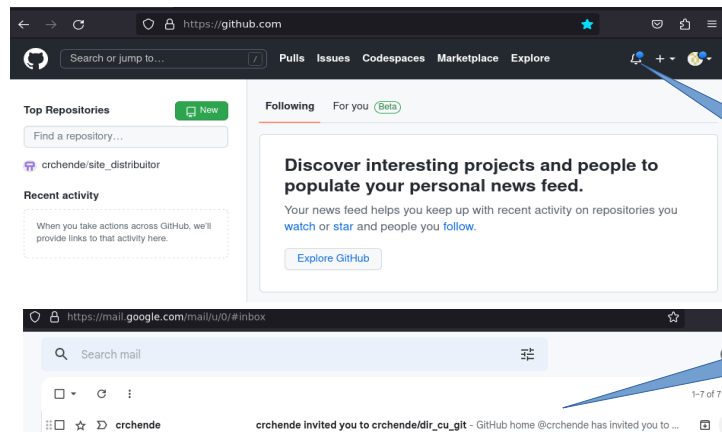
Lucru colaborativ. Adaugare colaboratori

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



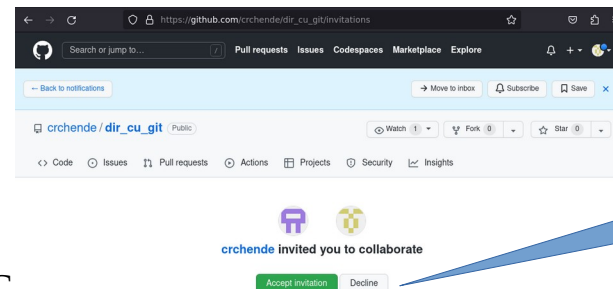
- Programatorul 2:
- Login pe GitHub. Click pe pictograma GitHub
- Notificare in GitHub ca este invitat sa fie colaborator

LAB



Notificare
pe
GitHub

Email cu
invitatie



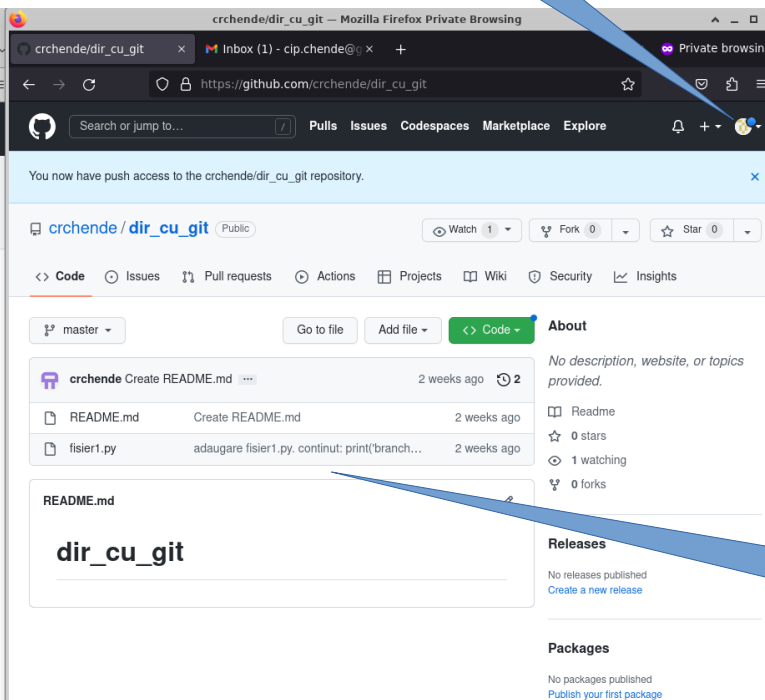
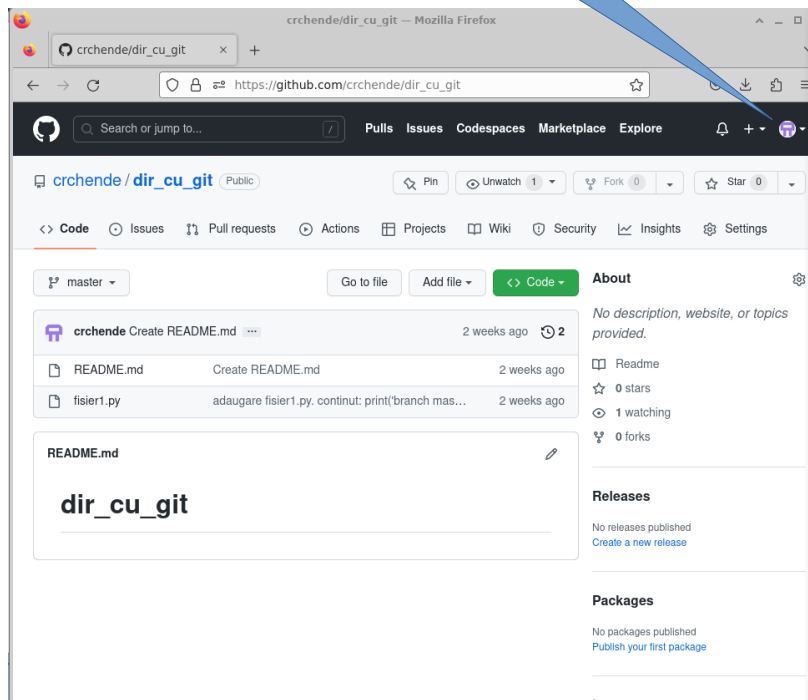
Optiune
Accept /
Decline

Lucru colaborativ. Adaugare colaboratori (2)

LAB

Programator 1

Programator 2



Dupa acceptarea invitatiei, programatorul 2 are acces la cod.

Lucru colaborativ. Adaugare colaboratori

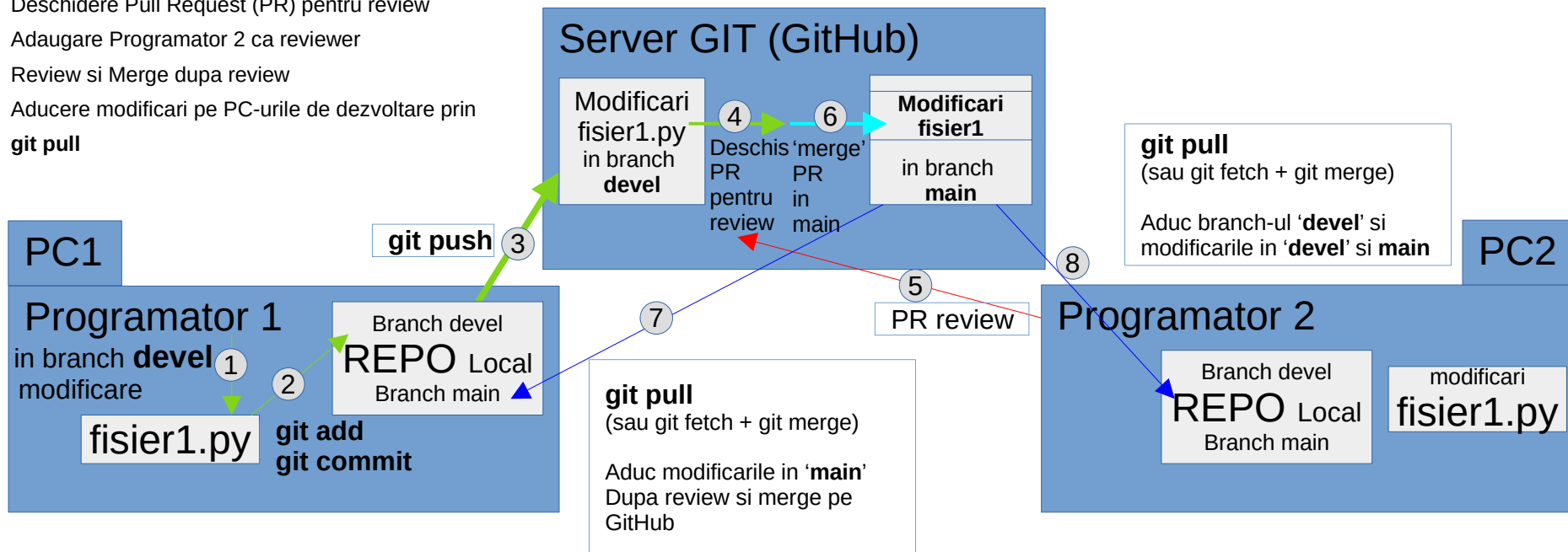
LAB

- 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

- 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

Prog. 1
Adauga
linia 2

Prog. 1
git status

Listare
continut
director
clonat

Prog. 1
Creaza
branch-
ul devel

Programator 1
git add
git commit

git push --set-upstream origin devel # doar prima data dupa adaugare branch nou
(dupa adaugarea branch-ului devel pe server, este suficient doar 'git push')

```
cip@cipasus: ~/git
cip@cipasus:~/git$ ls -l
total 16
drwxrwxr-x 3 cip cip 4096 ian 15 23:25 .
drwxrwxr-x 3 cip cip 4096 mar 10 23:27 curs
drwxrwxr-x 3 cip cip 4096 feb 27 16:37 jenkins
drwxrwxr-x 4 cip cip 4096 mar 6 08:20 jenkinsdemo
cip@cipasus:~/git$ git clone https://github.com/crchende/dir_cu_git.git
```

```
cip2@cipasus: ~/git
cip2@cipasus:~/git$ ls -l
total 0
cip2@cipasus:~/git$ git clone https://github.com/crchende/dir_cu_git.git
```

```
cip@cipasus: ~/git/dir_cu_git
cip@cipasus:~/git$ cd dir_cu_git/
cip@cipasus:~/git/dir_cu_git$ ls -l
total 8
-rw-rw-r-- 1 cip cip 31 mar 10 23:45 fisier1.py
-rw-rw-r-- 1 cip cip 13 mar 10 23:45 README.md
cip@cipasus:~/git/dir_cu_git$
```

```
cip2@cipasus: ~/git/dir_cu_git
cip2@cipasus:~/git/dir_cu_git$ cd dir_cu_git/
cip2@cipasus:~/git/dir_cu_git$ ls -l
total 8
-rw-rw-r-- 1 cip2 cip2 31 mar 10 23:48 fisier1.py
-rw-rw-r-- 1 cip2 cip2 13 mar 10 23:48 README.md
cip2@cipasus:~/git/dir_cu_git$
```

```
cip@cipasus: ~/git/dir_cu_git
cip@cipasus:~/git/dir_cu_git$ git branch
* master
cip@cipasus:~/git/dir_cu_git$ git branch devel
cip@cipasus:~/git/dir_cu_git$ git checkout devel
Switched to branch 'devel'
cip@cipasus:~/git/dir_cu_git$ git branch
* devel
  master
cip@cipasus:~/git/dir_cu_git$
```

```
cip2@cipasus: ~/git/dir_cu_git
cip2@cipasus:~/git/dir_cu_git$ git branch
* master
cip2@cipasus:~/git/dir_cu_git$
```

```
cip@cipasus: ~/git/dir_cu_git
File Edit View Search Terminal Help
print('branch master - start')
print('branch devel - modificare 1')
```

```
cip@cipasus: ~/git/dir_cu_git
File Edit View Search Terminal Help
cip@cipasus:~/git/dir_cu_git$ git status
On branch devel
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   fisier1.py
no changes added to commit (use "git add" and/or "git commit -a")
cip@cipasus:~/git/dir_cu_git$
```

```
cip@cipasus: ~/git/dir_cu_git
File Edit View Search Terminal Help
cip@cipasus:~/git/dir_cu_git$ git add fisier1.py
cip@cipasus:~/git/dir_cu_git$ git commit -m "modificare 1 in branch-ul devel"
[devel 8903037] modificare 1 in branch-ul devel
1 file changed, 1 insertion(+)
cip@cipasus:~/git/dir_cu_git$ git push
fatal: The current branch devel has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin devel

cip@cipasus:~/git/dir_cu_git$ git push --set-upstream origin devel
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 357 bytes | 357.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'devel' on GitHub by visiting:
remote:   https://github.com/crchende/dir_cu_git/pull/new/devel
remote:
To https://github.com/crchende/dir_cu_git.git
 * [new branch]   devel -> devel
Branch 'devel' set up to track remote branch 'devel' from 'origin'.
cip@cipasus:~/git/dir_cu_git$
```


Lucru colaborativ. Adaugare cod

- Vizualizare modificari in GitHub si creare Pull Request

LAB

Prog. 1
Vizualizare
repo in
GitHub

Branch-uri

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

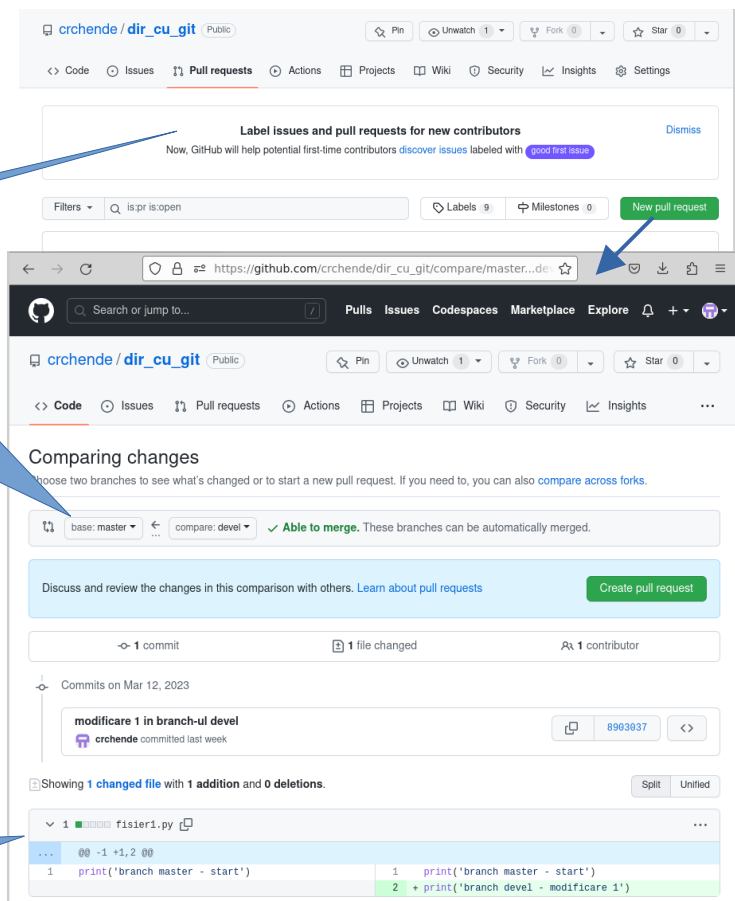
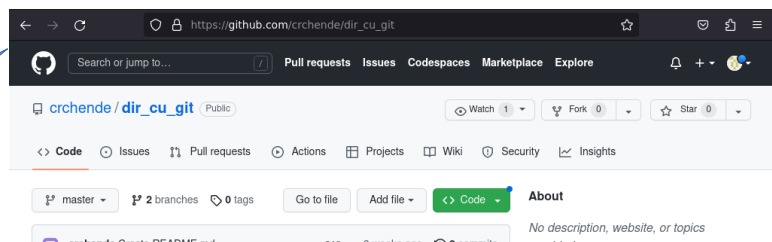
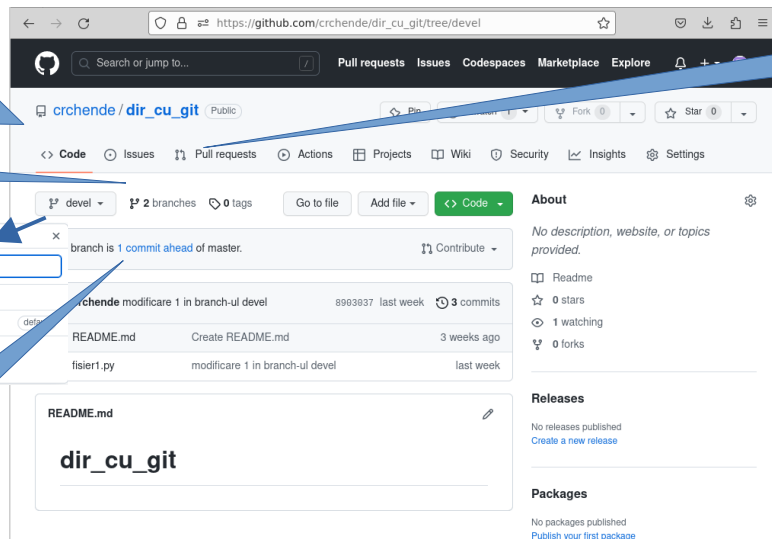
Prog. 2
Vizualizare
repo in
GitHub

Meniu Pull
Requests

Pagina PR

Selectare
sursa →
destinatie
pentru
Pull
Request
din devel in
Master

Vizualizare
Diferente



Lucru colaborativ. Pull Request (PR)

- Pagina creare Pull request (PR)

LAB

The screenshot shows the GitHub interface for creating a pull request. At the top, there's a search bar and navigation links for Pulls, Issues, Codespaces, Marketplace, and Explore. Below this, the repository name 'crchende/dir_cu_git' is displayed with options to pin, unwatch, fork, and star. The main heading is 'Open a pull request' with a subtext explaining how to create a PR by comparing changes across two branches. A dropdown menu shows 'base: master' and 'compare: devel', with a green checkmark indicating 'Able to merge'. Below this, there's a text area for the PR title and description. The title is 'modificare 1 in branch-ul devel'. The description includes a preview of the PR content, which mentions 'Exemplu Pull Request din 'devel' in 'master'. Adaugat 'crchende2' ca reviewer. Label-ul adaugat sugereaza ca modificarea pentru care am facut PR este o imbunatatire'. At the bottom right, there's a green 'Create pull request' button.

Titlu PR

Descriere

Creere PR dupa configurarea tuturor parametrilor.
Acestia pot fi configurati si ulterior

Adaugare Review-eri

Colaborator adaugat ca reviewer

Adaugare Responsabili

Adaugare etichete – ajuta la clarificarea motivului pentru care facem PR-ul

The screenshot shows the GitHub notifications page. At the top, there's a search bar and navigation links for Pull requests, Issues, Codespaces, Marketplace, and Explore. Below this, there's a notification for a pull request. The notification includes the repository name 'crchende/dir_cu_git #1', the title 'modificare 1 in branch-ul devel', and the status 'review requested'. It also shows the time '1 minute ago'. Below the notification, there's a section for filters, including 'Assigned', 'Participating', 'Mentioned', and 'Team mentioned'. At the bottom, there's a section for labels, with 'enhancement' selected.

Programator 2 notificat ca are de facut un review

Detalii PR si review

Lucru colaborativ. Review cod in "PR"

- Adaugare review si vizualizarea acestuia

LAB

Programator 2
(reviewer-ul)
Vizualizare PR

Vizualizare
fișiere modificate

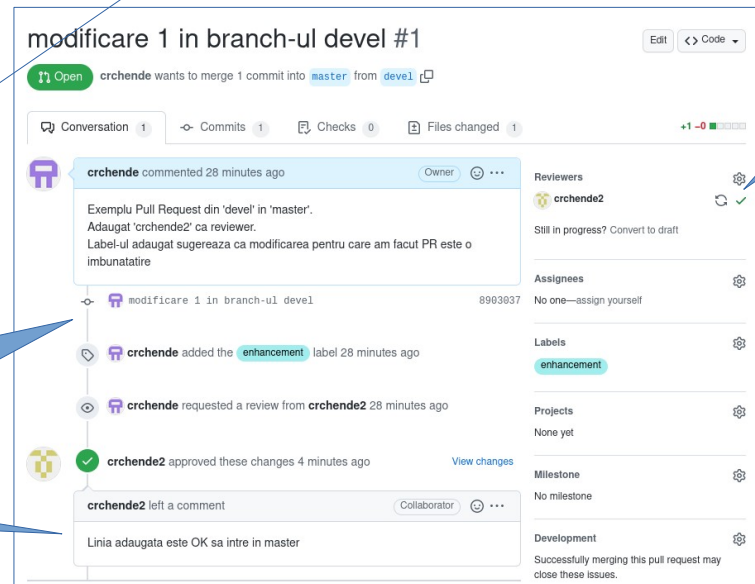
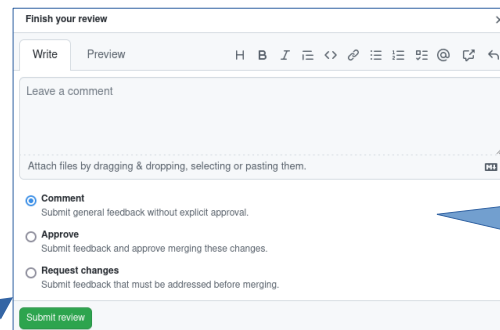
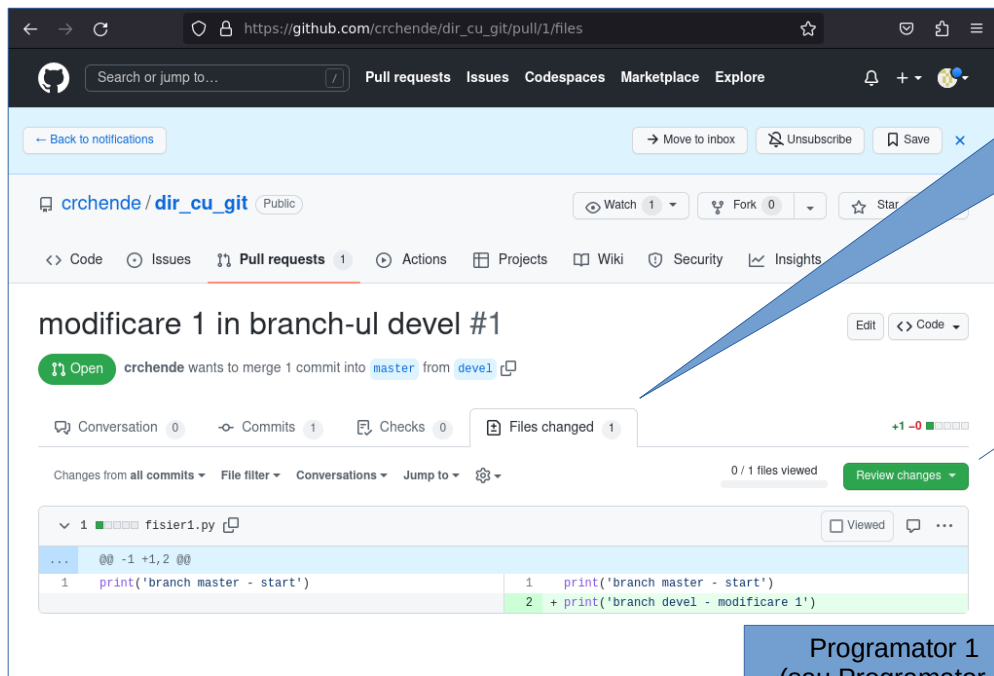
Programator 2
(reviewer-ul)
Fereastra Review

Confirmare
aprobare la
review

Programator 1
(sau Programator 2)
Pagina dupa adaugare
review

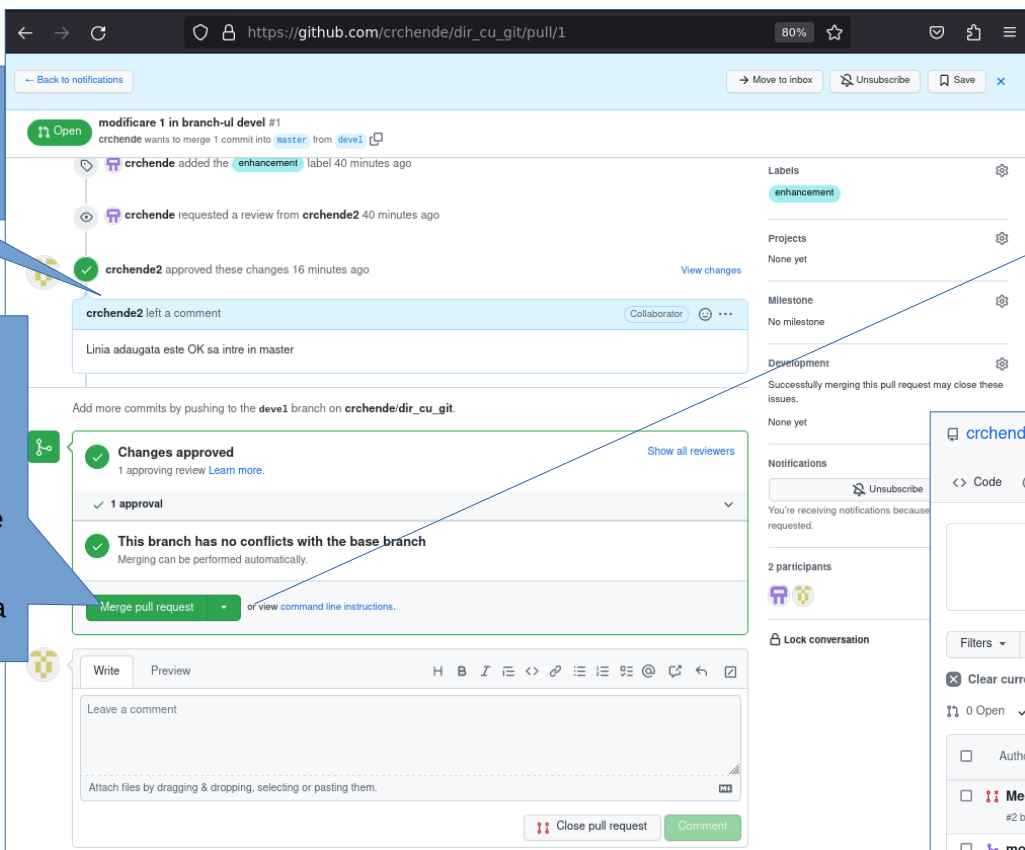
Comentariu adaugat la review

2CC



Lucru colaborativ. Integrare cod / 'merge' PR

- Pagina Pull Request. Vizualizare aprobari si integrare cod



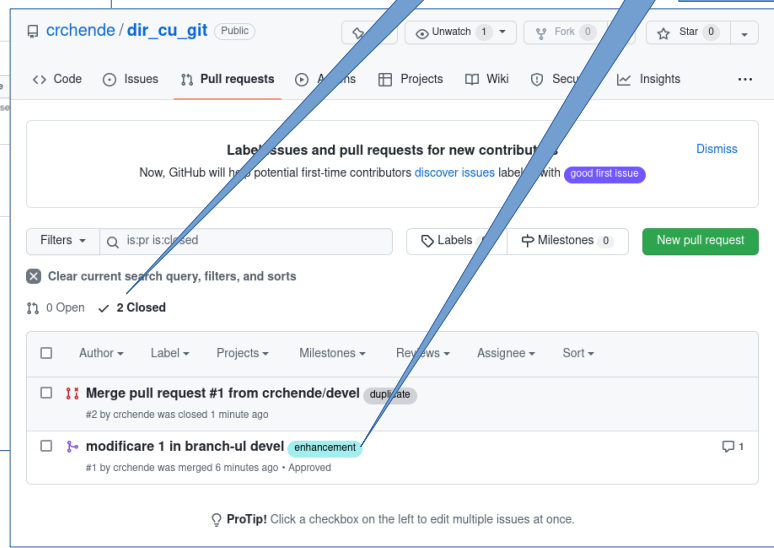
Optiuni integrare. Varianta implicita: 'merge commit'

- ✓ **Create a merge commit**
All commits from this branch will be added to the base branch via a merge commit.
- Squash and merge**
The 1 commit from this branch will be added to the base branch.
- Rebase and merge**
The 1 commit from this branch will be rebased and added to the base branch.

Vizualizare PR-uri inchise

PR-ul pe prin care am pus modificari le din devel in master

LAB



Sincronizare directoare de lucru dupa 'merge' 'Pull request'

Programator 1

Programator 2

Vizualizare
diferente intre
GitHub si
repository si
director de
lucru local

```
File Edit View Search Terminal Help
cip@cipasus: ~/git/dir_cu_git
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$
```

```
File Edit View Search Terminal Help
cip2@cipasus: ~/git/dir_cu_git
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'

```
File Edit View Search Terminal Help
cip@cipasus: ~/git/dir_cu_git
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$
```

```
File Edit View Search Terminal Help
cip2@cipasus: ~/git/dir_cu_git
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

Branch protection rule

Branch name pattern *

master

Applies to 1 branch

master

Protect matching branches

- ☒ **Require a pull request before merging**
When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.
- ☒ **Require approvals**
When enabled, pull requests targeting a matching branch require a number of approvals and no changes requested before they can be merged.
Required number of approvals before merging: 1
- ☐ **Dismiss stale pull request approvals when new commits are pushed**
New reviewable commits pushed to a matching branch will dismiss pull request review approvals.
- ☐ **Require review from Code Owners**
Require an approved review in pull requests including files with a designated code owner.
- ☐ **Require approval of the most recent reviewable push**
Whether the most recent reviewable push must be approved by someone other than the person who pushed it.

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

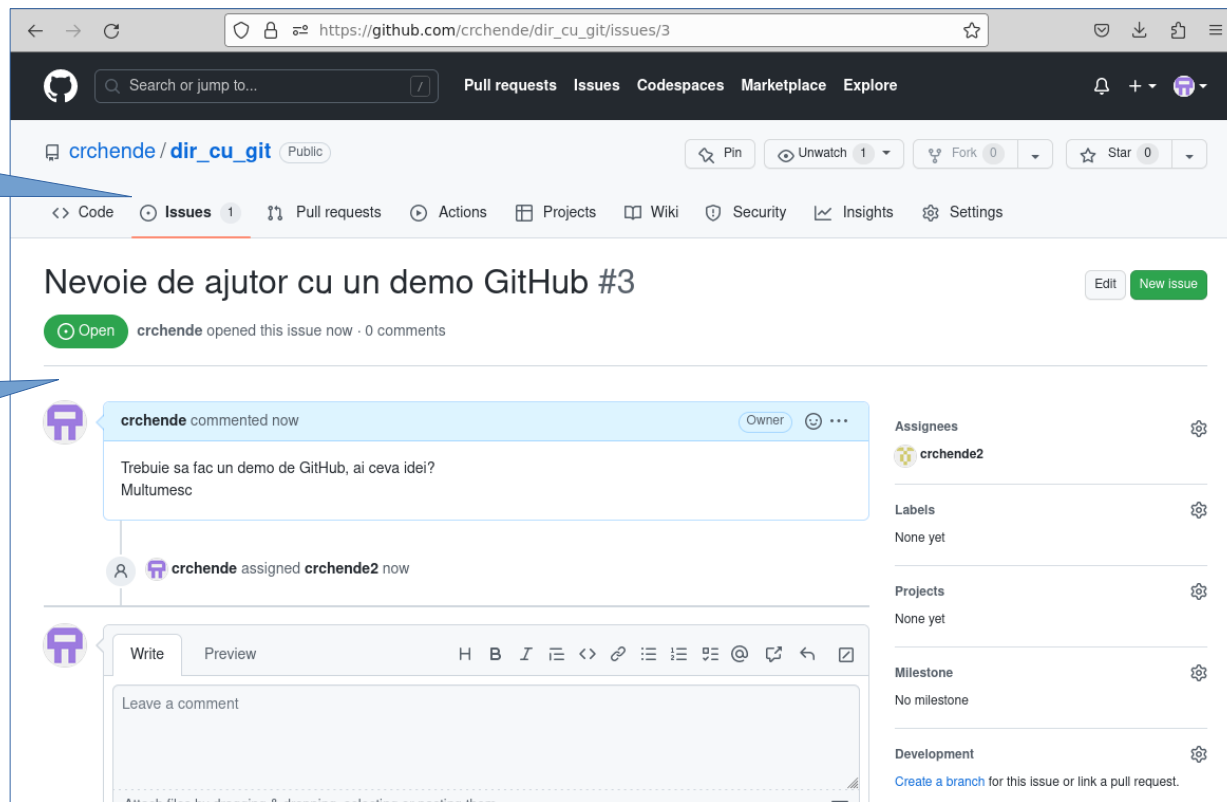
LAB

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 aceeași zona a aceluși fișier)
 - 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 echivalentul Jenkins

Utilizare IDE cu extensie pentru git.

Exemplu Visual Studio Code (vscode)

Modificare in repo: <https://github.com/crchende/sysinfo.git>

LAB

The screenshot shows the Visual Studio Code interface with the 'network.py' file open. The Source Control panel on the left shows the 'main' branch and a file named 'network.py' that has been modified. The editor shows the Python code for the 'gaseste_rutele()' function. A callout points to the 'Commit' button in the Source Control panel, labeled 'Vizualizare GIT'. Another callout points to the 'git add' button, labeled 'git add'. A third callout points to a notification icon in the bottom left, labeled 'Notificare fisier modificat'. A fourth callout points to the 'main' branch in the Source Control panel, labeled 'Branch-ul curent'. A fifth callout points to a new line being added to the code, labeled 'Adaugare linie: #comentariu vizualizare diferite'.

Vizualizare GIT

'git add'

Notificare fisier modificat

Branch-ul curent

Adaugare linie: '#comentariu' vizualizare diferite

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' - echivalent cu - `git fetch + git merge`
11. Rezolvare probleme de merge – in cazul in care cineva a modificat aceeași zona de cod ca și dezvoltatorul care tocmai vrea să facă push - prima dată va trebuie să facă `git pull` sau `git fetch + git merge`, să rezolve problema de merge, să facă un nou commit la care să-i dea push)
12. Vizualizare stare 'remote': `git remote show origin` – se vede dacă repository-ul local este sincronizat cu cel remote
13. Vizualizare loguri: `git log` / `git log --all --oneline -graph`
14. Vizualizare diferite: `git diff`
15. Vizualizare configurare `git config -list -show-origin`
16. Vizualizare 'remote' `git ls-remote`
17. Pull request – la ce folosește
18. Reguli de protecție branch-uri
19. Mijloace de colaborare și organizare a activităților pe GitHub: Pull Request / revieweri, Issues
20. Ajutor în linia de comandă

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_distributor