

Desenvolupament col·laboratiu (II): Branques de Git remotes i fluxos de treball

Albert Clapés (aclapes@ub.edu)

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona (UB)

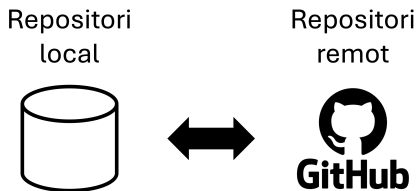
Projecte Integrat de Software (2024/25)

Objectius

Els **objectius** són:

- ① Saber manegar branques de Git remotes.
- ② Conèixer diversos fluxos de treball.
 - Saber fer *pull requests*.

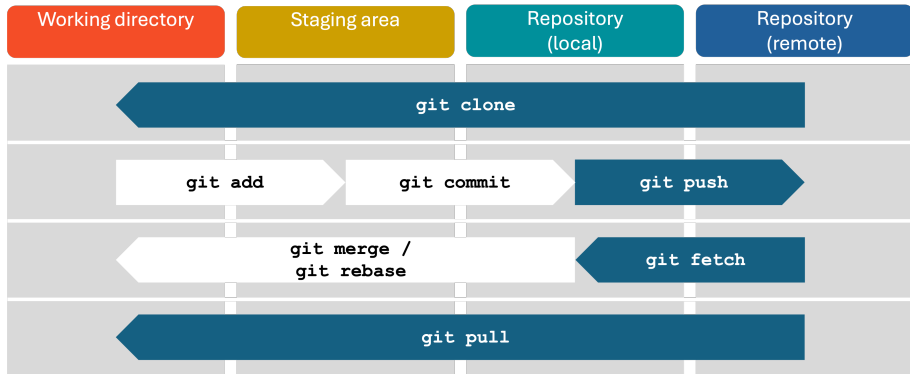
Git remot



Git permet sincronitzar canvis entre el repositori local i un repositori remot (GitHub, GitLab, Bitbucket, etc).

Entren en joc les *branques remotes*.

Àrees de Git i operacions remotes



Branques remotes

Les **branques remotes** "viuen" al repositori remot. Localment, només tenim accés a les respectives *branques de seguiment remot*:

```
# LListar totes les braques
$ git branch --all
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/dev
  remotes/origin/main
  remotes/origin/test
```

En aquest exemple, tenim una branca local (`main`) i quatre de seguiment remot (`remotes/origin/*`).

Comandes de Git: `clone`

git clone <url>

Crea un repositori local còpia d'un repositori remot. Funcionament:

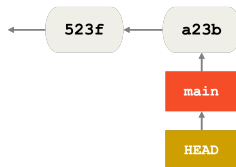
- 1 Estableix el remot (`origin`) per apuntar a <url>.
- 2 Descarrega els commits del remot.
- 3 Crea les branques de seguiment remot (p. ex. `origin/main`).
- 4 Crea una branca local a partir del meta-punter remot `origin/HEAD`.
Si `origin/HEAD -> origin/main`, crearà la branca local `main` i
farà `HEAD -> main`.
- 5 Crea els fitxers/directoris en el *working directory* per a reflectir el
`HEAD`.

Funcionament de `clone`

Punt de partida

Local

Remote (origin)

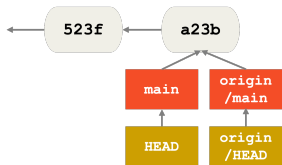


Abans de fer `clone`.

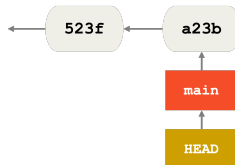
Funcionament de clone

```
$ git clone <remote-url>
```

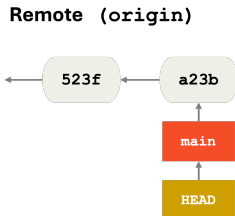
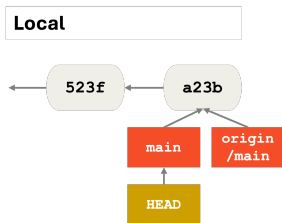
Local



Remote (origin)

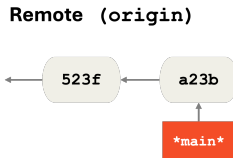
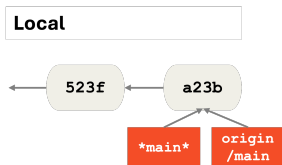


Funcionament de `clone`



El meta-punter a la branca de seguiment remot `origin/HEAD` no es fa servir més. Només indicava quina era la branca a la que fer `switch` després del `clone`.

Funcionament de `clone`



Simplificació: indicarem la branca apuntada pels HEAD amb `*nom-branca*`.

Branques de seguiment remot

Inicialment, només es crea la branca local `<nom-branca>` si `origin/HEAD -> origin/<nom-branca>`).

Com es creen la resta de branques locals a partir de les branques de seguiment remot?

Branques de seguiment remot

Fent `switch` a una branca de seguiment remot, automàticament, es crea la branca local corresponent en cas de no existir:

```
$ git switch dev  
branch 'dev' set up to track 'origin/dev'.  
Switched to a new branch 'dev'
```

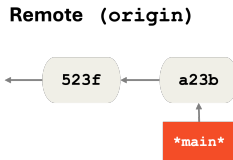
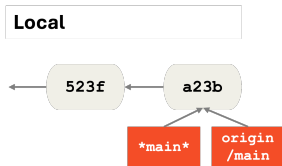
```
$ git branch --all  
* dev  
main  
remotes/origin/HEAD -> origin/main  
remotes/origin/dev  
remotes/origin/main  
remotes/origin/test
```

Branques de seguiment remot

Les **branques de seguiment remot** no permeten fer-hi `commit`.
Només a la corresponent branca local.

Canvis a branca remota

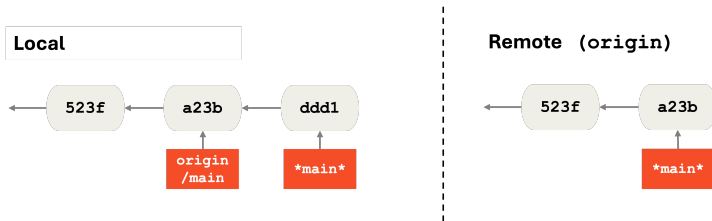
Punt de partida



Suposem que hem fet ja el `clone`.

Canvis a branca remota

```
$ git commit -m "..."
```



El `commit` només ha fet avançar `main`.

Com reflectim els canvis a `origin/main` i a la `main` remota?

Comandes de Git: push

git push origin <nom-branca>[:<nom-branca-remota>]

Reflecteix els canvis de <nom-branca> del repositori local a la branca <nom-branca-remota> del repositori remot (origin). Per defecte:

- Si no existeix <nom-branca-remota>, la crea al remot.
- Sí només s'indica <nom-branca>, s'agafa el mateix nom per <nom-branca-remota>.

Funcionament:

- 1 Localment i de manera transparent, push fa merge de la branca <nom-branca> (origen) a origin/<nom-branca-remota> (destí).
- 2 Després, puja els commits de origin/<nom-branca-remota> al remot.
- 3 Si la branca remota <nom-branca-remota> ha avançat i divergit, haurem de baixar els canvis primer (veurem fetch i pull).

Comandes de Git: `push` (*cont.*)

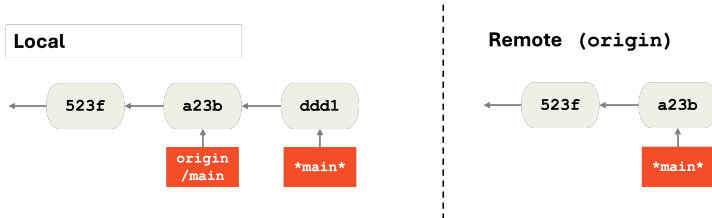
Es recomana fixar l'associació entre cada branca local i el seu *upstream* (branca remota) amb **git push -u ...** per:

- No haver de passar paràmetres. Fer, directament, `git push`.
- Veure les diferències entre la branca local i el seu *upstream* (branca de seguiment remot associada) sempre que fem `git status`.

Per comprovar els upstream, podem fer **git branch -vv**.

Funcionament de `push`

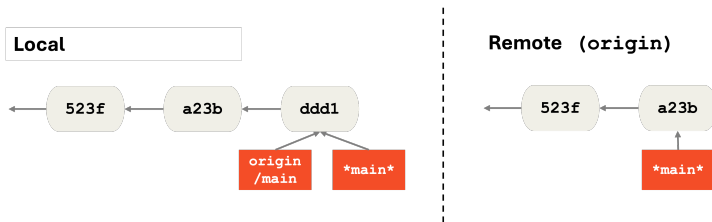
Punt de partida



Cal fer `push` per "pujar" els canvis.

Funcionament de `push`

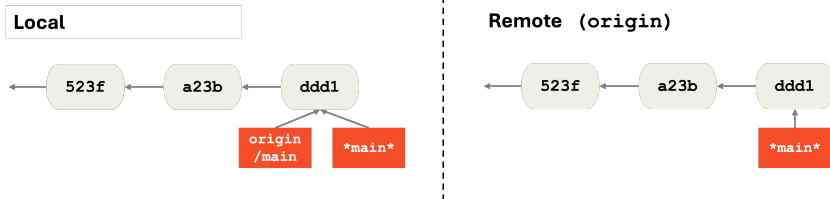
```
$ git push origin main
```



Quan fem el `push`, passen dues coses internament:
(1) merge (sempre fast-forward) de `main` a `origin/main` i...

Funcionament de `push`

```
$ git push origin main
```



... (2) es pugen els nous commits a la `main` del remot.

Comandes de Git: `fetch`

```
git fetch [origin [<nom-branca-remota-1> ...  
<nom-branca-remota-N>]]
```

Baixa els canvis del repositori remot (`origin`) al local, creant branques de seguiment remot per les noves branques remotes o actualitzant les branques de seguiment remot preexistents.

- Si no indiquem branques remotes, aplica a totes.
- Igual que `clone`, no crea branques locals. Per crear-les, cal fer `switch`.
- Tampoc no actualitza branques locals. Per actualitzar-les, cal fer-lis `merge` de les branques de seguiment remot.

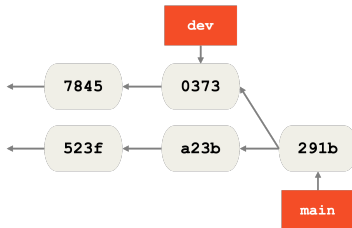
Funcionament de `fetch`

Punt de partida

Local

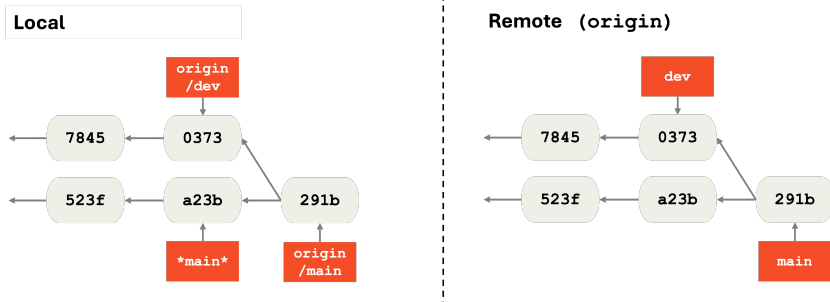


Remote (origin)



Funcionament de `fetch`

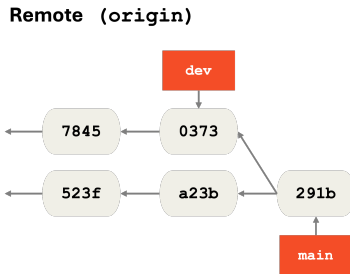
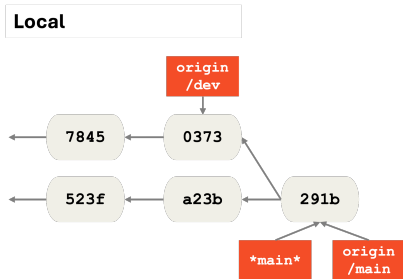
```
$ git fetch
```



❗ No es crea una branca local `dev` automàticament. Caldria fer:
`git switch dev`.

Funcionament de fetch

```
$ git merge origin/main
```



merge ha incorporat els canvis d'origin/main en el main (HEAD).

Comandes de Git: pull

git pull [origin <nom-branca>]

Fa `fetch` i – si cal – `merge` en una sola comanda. Per defecte:

- Si no indiquem <nom-branca>, el `fetch` és fa a la branca HEAD.
- Tampoc crea la branca local <nom-branca>. Per a crear-la, cal fer-hi `switch`.

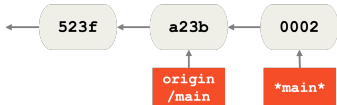
git pull [origin <nom-branca>] **--rebase**

Substitueix `merge` pel `rebase`.

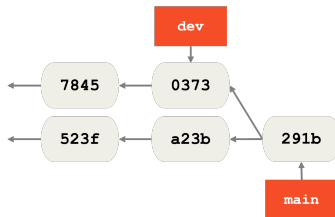
Funcionament de `pull`

Punt de partida.

Local



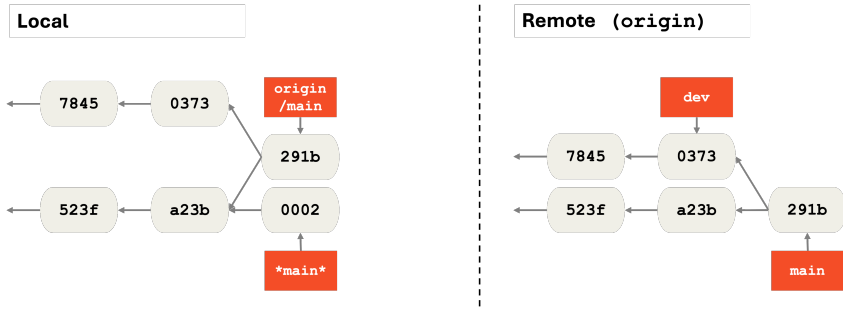
Remote (origin)



La branca local `main` i la branca remota `main` han **divergit**.

Funcionament de `pull`

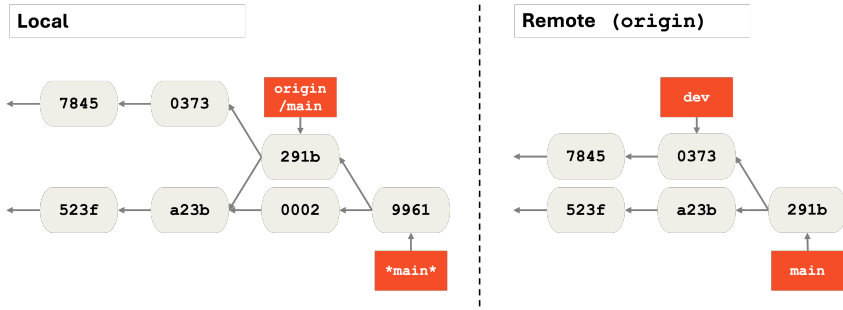
```
$ git pull origin main
```



El `pull` fa (1) el `fetch` de la branca remota indicada (`main`) creant només la branca de seguiment remot `origin/main` i...

Funcionament de pull

```
$ git pull origin main
```



... (2) merge de la branca `origin/main` a `main`.
Què podríem fer ara?

Simulador de Git

Recursos. Existeixen simuladors de Git que simulen local i remot i permeten provar l'efecte de les comandes en termes de branques:

- <https://github.com/initialcommit-com/git-sim>
- <http://git-school.github.io/visualizing-git/>
(on-line)

Desenvolupament col·laboratiu en repositori remot

Quan es desenvolupa conjuntament en un repositori remot, les branques tenen un paper important. Hem de decidir com les utilitzem.

Preguntes freqüents

- 1 Quines hem de crear i per què serveixen?
- 2 Quant temps les mantenim?
- 3 Qui les crea i hi treballa?
- 4 Com integrem canvis de diverses branques?
- 5 ...

Fluxos de treball

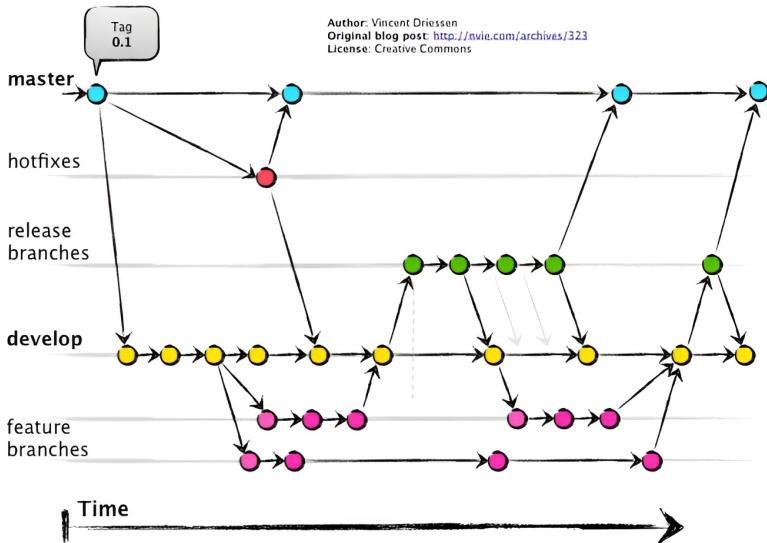
Un **flux de treball** estableix una metodologia de desenvolupament en un repositori de codi compartit, que – entre d'altres coses¹ – com es creen, gestionen i fusionen les branques.

Exemples

- *Git flow*
- *Github flow*
- *Trunk-based development*
- *Master-only flow*
- ...

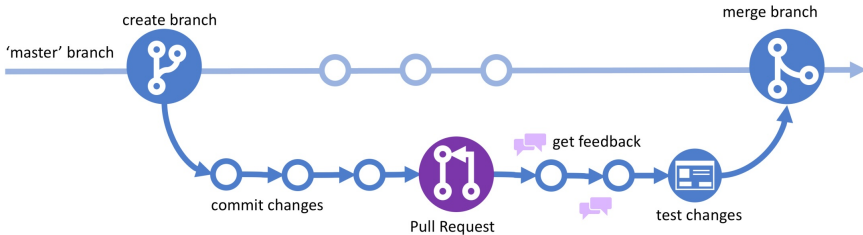
¹Inclouen altres aspectes a banda del maneig de branques, que no cobrirem.

Git flow



Github flow

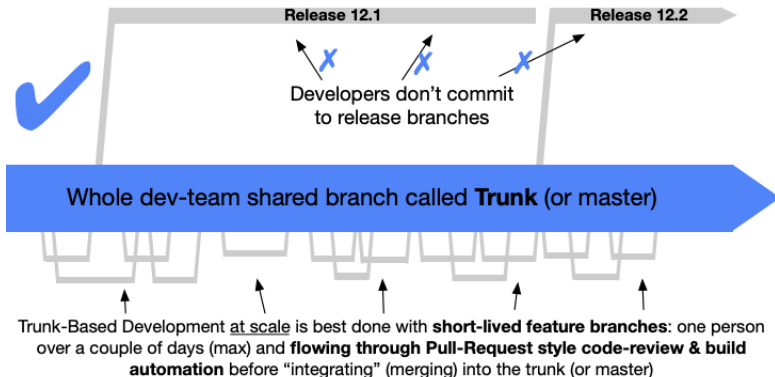
GitHub Flow



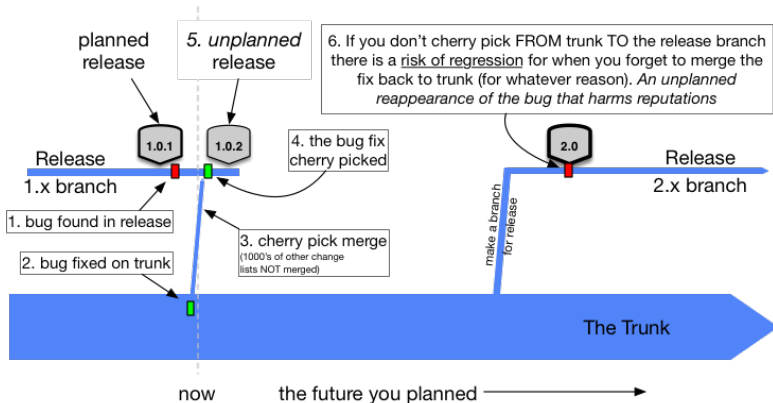
Copyright © 2018 Build Azure LLC

<http://buildazure.com>

Trunk-based development



Trunk-based development (cont.)



Comandes de Git: `cherry-pick`

git cherry-pick <commit-sha-1> ... <commit-sha-N>

Aplica els canvis introduïts pel commits identificat per <hash-commit> a la branca actual.

- Permet portar canvis concrets d'una branca a una altra sense fusionar tota la branca.
- És útil per corregir errors o portar funcionalitats específiques sense arrossegar altres modificacions.
- Els commits 1 ... N s'apliquen, un a un, en l'ordre especificat.

Si algún commit genera conflictes: resoldre'ls i fer `git cherry-pick --continue`, saltar-se el cherry-pick del commit conflictiu (`--skip`) o cancel·lar tot el procés (`--abort`).

Master-only flow



Les **feature flags** permeten activar/desactivar funcionalitats que encara no estiguin llestes:

```
public Database getDatabase() {  
    FeatureManager fm = FeatureManager.getInstance();  
    if (fm.isEnabled("firebase_database"))  
        return new FirebaseDatabase();  
  
    return new MockDatabase();  
}
```

Pull request

Un **pull request (PR)** és una sol·licitud per fusionar remotament els canvis d'una branca en una altra.

- Permet revisar, discutir i/o aprovar els canvis abans de fusionar-los.
- És una funcionalitat de plataformes com GitHub, GitLab o Bitbucket, no una comanda de Git.
- Inclou detalls sobre els canvis, la descripció, i una llista de commits i fitxers modificats.
- Es poden afegir comentaris a cada línia de codi per discutir millores o correccions.

No és exclusiu de Github flow. Utilitzat, per integrar canvis de branques exclusives de desenvolupador en branques compartides.

Passos per fer un *pull request*

(1) Creem una branca on afegir-hi la funcionalitat o la obtenim del remot si ja existeix al remot:

```
# Crear branca localment
git switch -c <nom-branca>
# O bé... Obtenim branca del remot
git fetch origin <nom-branca>
```

(2) Fem la feina en el repositori local i sincronitzem amb el remot:

```
# ... Fem la feina ...

# Preparar i crear commit
git add .
git commit -m "... "

# Sincronitza amb el remot
git push origin <nom-branca>
```

Passos per fer un *pull request*

(3) Crear **pull request** des de la plataforma web. Amb GitHub:

aclapes / GitExample

Q Type [17](#) to search

<> Code Issues **1** Pull requests Actions Projects **1** Wiki Security Insights Settings


Label issues and pull requests for new contributors [Dismiss](#)

Now, GitHub will help potential first-time contributors [discover issues](#) labeled with [good first issue](#)

Filters [Labels 9](#) [Milestones 0](#) [New pull request](#)

☒ Clear current search query, filters, and sorts

☐ [0 Open](#) ☒ [3 Closed](#) Author Label Projects Milestones Reviews Assignee Sort



There aren't any open pull requests.

You could search [all of GitHub](#) or try an [advanced search](#).

Passos per fer un *pull request*

(4) Seleccionar de destí (main) i origen (5-uc2-log-in):

The screenshot shows the GitHub 'Compare changes' interface. At the top, the repository is 'aclapes / GitExample'. The navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The 'Compare changes' section is active, showing 'base: main' and 'compare: main'. A dropdown menu for 'Choose a head ref' is open, displaying a search bar and a list of branches. The 'main' branch is selected with a checkmark and labeled 'default'. Other branches listed include '5-uc2-log-in', 'dev', 'newbranch', 'newbranch2', 'newbranch3', 'prova-remota', and 'test'. The 'prova-remota' branch is highlighted. Below the dropdown, the 'Create pull request' button is visible. The background shows the start of a commit history table with columns for commit message, author, and time ago.

Commit message	Author	Time ago
...	...	20 hours ago
...	...	2 weeks ago
...	...	2 weeks ago
...	...	2 weeks ago

Passos per fer un *pull request*

(5) Revisió de canvis abans de crear-lo i fer `Create pull request`:

The screenshot shows the GitHub web interface for a repository named 'GitExample' by user 'aclapes'. The top navigation bar includes links for Code, Issues (1), Pull requests, Actions, Projects (1), Wiki, Security, Insights, and Settings. The main content area is titled 'Comparing changes' and provides instructions on how to use the comparison tool. It shows a comparison between the 'base: main' branch and a branch named 'compare: 5-uc2-log-in'. A green checkmark indicates that the changes are 'Able to merge'. Below this, there is a green 'Create pull request' button. A summary bar shows '1 commit', '1 file changed', and '1 contributor'. A commit history section shows a commit from 'Feb 25, 2025' titled 'Created User class' by 'aclapes', with a commit hash of '3900a48'. At the bottom, it shows 'Showing 1 changed file with 5 additions and 0 deletions', specifically 'User.java' with 5 lines of code. The interface is in dark mode.

Passos per fer un *pull request*

(6) Emplenar detalls i meta-informació:

The screenshot shows the GitHub interface for a pull request in the repository 'aclapes / GitExample'. The top navigation bar includes links for Code, Issues (1), Pull requests, Actions, Projects (1), Wiki, Security, Insights, and Settings. The main heading is 'Open a pull request', with a subtext explaining that a new pull request is created by comparing changes across two branches. Below this, a comparison bar shows 'base: main' and 'compare: 5-uc2-log-in', with a green checkmark indicating 'Able to merge'. The 'Add a title' section contains the text '[enhancement] UC2: log in'. The 'Add a description' section has a rich text editor with a 'Description' tab selected, containing the following text: 'This PR enhances the user login process by implementing the following improvements: - [] User with username `admin` can log-in. - [] Any user with credentials can log-in.' Below the description, there are sections for 'Related Issue' and 'Motivation & Context'. On the right side, there are sections for 'Reviewers' (No reviews), 'Assignees' (@aclapes), 'Labels' (enhancement), 'Projects' (@aclapes's untitled project), and 'Milestone' (No milestone).

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#). [Learn more about diff comparisons here](#).

base: main ← compare: 5-uc2-log-in ✓ Able to merge. These branches can be automatically merged.

Add a title

[enhancement] UC2: log in

Add a description

Write Preview

📄 Description

This PR enhances the user login process by implementing the following improvements:

- [] User with username `admin` can log-in.
- [] Any user with credentials can log-in.

📌 Related Issue

Closes #XX (if applicable)

📌 Motivation & Context

Reviewers

No reviews

Assignees

@aclapes

Labels

enhancement

Projects

@aclapes's untitled project

Milestone

No milestone

Issues

Un **issue** és un *registre* (amb identificador numèric, p. ex. #X) que informa d'un problema, demana funcionalitat o denota una tasca pendent d'adreçar.

- Permeten organitzar i fer-ne seguiment.
- Pot ser comentat, assignat a desenvolupadors i etiquetat per categoritzar-lo.
- Pot estar vinculat a un o diversos PR.
- Es pot tancar. O la pot tancar un PR fent ús de la paraula clau² `closes` i l'id de l'*issue* en el cos de la PR. Per exemple, `Closes #5`.

²Veure altres paraules clau: [Using keywords in issues and pull requests](#).

Bibliografia

- **Pro Git** (llibre electrònic, lliure).
`https://git-scm.com/book/en/v2`
- **Documentació de Git.** `https://git-scm.com/docs`
- **Documentació de GitHub (*pull requests*).**
`https://docs.github.com/en/pull-requests`