

# Projecte (material de suport) - Desenvolupament basat en branques

Projecte Integrat de Software (2024/25)  
Facultat de Matemàtiques i Informàtica  
Universitat de Barcelona

## Introducció

Inicialment, desenvolupareu en branques de funcionalitat (*feat*) i, quan sigui necessari, en branques de correcció d'errors (*bugfix*). Les branques hauran d'estar constantment sincronitzades amb la seva versió del remot, perquè els/les altres membres de l'equip puguin estar al corrent de la vostra feina.

Us mostrem un exemple del flux de desenvolupament a la Fig. 1.

## Prerequisits

- Git instal·lat a l'ordinador.
- Conèixer el funcionament de les comandes bàsiques de Git.
- Accés al repositori del projecte al GitHub, havent-vos afegit a un equip del Classroom<sup>1</sup> i havent acceptat l'*assignment* "Projecte".

## 1 Clonar el Repositori

Potser haureu clonat el repositori del projecte amb l'Android Studio directament. Si encara no ho heu fet, cloneu el repositori del projecte. Per fer-ho amb la línia de comandes:

```
1 # Clonar, assumint que sou l'equip "ABCD4"  
2 git clone https://github.com/ProjecteIntegratDeSoftware-24-25/  
   projecte-abcd4.git  
3 # Accedir al directori del projecte clonat  
4 cd projecte-abcd4
```

---

<sup>1</sup>L'enllaç al GitHub Classroom: <https://campusvirtual.ub.edu/mod/url/view.php?id=6134521>

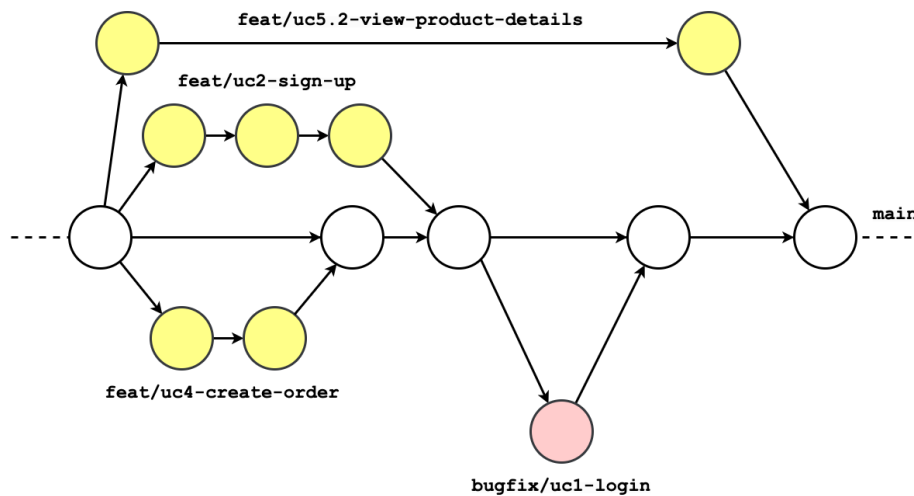


Figure 1: Exemple de desenvolupament basat en branques. Es tracta de crear una branca per funcionalitat (*feature*) o correcció d'error (*bugfix*) que us proposeu implementar. Un cop hagueu acabat, fareu merge a la branca main. Totes les branques, inclosa la principal, les tindreu constantment sincronitzades amb el remot.

Obriu el projecte a l'Android Studio i el terminal integrat (Alt + F12). O, si ho preferiu, obriu un terminal independent. En qualsevol cas, necessitarem un terminal per seguir la resta de passos.

## 2 Crear una branca

Abans de començar a desenvolupar res, creeu una nova branca per mantenir la vostra feina actual separada de la branca principal (main). Si, per exemple, hem de treballar en el log-in de l'aplicació:

```
1 git switch -c feat/uc1-log-in
```

Això crea una nova branca local anomenada `feat/uc1-log-in` i hi fa el canvi (hi situa el HEAD).

D'altra banda, us preguntareu què és el prefix `feat` en el nom de la branca. Els prefixos serveixen per distingir quin tipus de canvis aportarà la branca. Normalment, es fan servir aquests:

- **feat:** Una nova funcionalitat.
- **fix (o bugfix):** Correcció d'un error.
- **refactor:** Modificació del codi que no corregeix errors ni afegeix funcionalitats. Només afecta la organització i/o llegibilitat.

- **style:** Canvis molt menors que no afecten el comportament del codi, ni canvien substancialment la organització/llegibilitat del codi (espais en blanc, formatació, punts i comes absents, etc.).
- **perf:** Canvi en el codi per millorar el rendiment.
- **test:** Afegir proves mancants.
- **chore:** Canvis en el procés de construcció o en eines auxiliars i llibreries (integrar la Firebase al projecte).
- **docs:** Canvis només en la documentació.

Abans de continuar, podríeu decidir pujar ja la branca al remot ara 4, tot i ser buida (sense *commits*), o esperar-vos a fer un primer *commit*.

### 3 Treballar a la branca: fer canvis i **commit**

Estant a la branca de treball, implementeu el codi amb l'Android Studio (això vol dir fer canvis en els fitxers del *working directory*), poseu-los a l'*staging area* i feu *commit*:

```

1 # ... Crear p. ex. la LoginActivity ...
2
3 # Afegir canvis a l'staging area
4 git add .    # Afegir tots els canvis
5
6 # Afegir canvis a la branca local
7 git commit -m "Creada LoginActivity"
```

### 4 Pujar la branca local al repositori remot

Per a reflectir els vostres canvis en remot, pugeu la nova branca amb *push*:

```

1 git push origin feat/uc1-log-in
```

Tal com sabeu, això actualitza la branca de seguiment remot *origin/feat/uc1-log-in* a partir de la local *feat/uc1-log-in* i puja els canvis (*commits*) al remot. Si la branca no existeix a l'*origin*, el primer que farà serà crear-la.

Fent això, els/les companys/es de l'equip veuran la feina que esteu fent a la branca.

### 5 Merge de la branca remota amb el main

Arribats al moment de voler incorporar els canvis de la branca de *feat/uc1-log-in* a la branca *main*. Per això, abans de res haurem d'assegurar-nos que la branca *main* està al dia amb els canvis del remot, sinó no ens deixarà fer-hi merge. Un cop fet, podem fer fusionar-hi *feat/uc1-log-in*. Amb més detall:

1. Actualitzar main amb els últims canvis del remot que no tinguem localment (en cas d'haver-n'hi). En cas contrari, no podrem incorporar-hi canvis de la nostra branca de treball. Per això, canviarem a la branca local main, actualitzarem la branca de seguiment remot origin/main amb els canvis del remot (fetch) i comprovarem si, efectivament, hi ha canvis del remot que no tenim actualment:

```
# Canviem a la branca local de destí (main)
git switch main
# Actualitza la branca de seguiment remot (origin/main)
git fetch origin main
# Comprovar l'estat de main respecte origin/main
git status
```

Si no hi ha canvis, l'status dirà `Your branch is up to date with 'origin/main'`. Si n'hi ha, els haurem d'aplicar a la branca local main (destí), agafant-los de la branca de seguiment remot origin/main (origin):

```
git merge origin/main
```

Aquest pas no ens generarà mai conflictes i serà sempre *fast-forward merge*, perquè no farem mai modificacions directes a la branca main del local.

2. A continuació, havent sincronitzat main amb el remot, apliqueu-hi els canvis de la branca de treball feat/ucl-log-in:

```
git merge feat/ucl-log-in
```

En aquest cas, sí que podríem tenir conflictes que el mateix merge ens llistarà. Si calgués, també podríem fer `git status` i buscar la secció "Unmerged paths" per veure-hi la mateixa llista.

## 6 Resoldre conflictes

Si seguint els pas (2) de l'apartat anterior us trobeu amb conflictes, no haureu pogut fusionar els canvis i obtenir un *merge commit*. Haureu, primer, de resoldre els conflictes i tornar a fer merge un cop resolta:

1. Resoldre els conflictes en el codi. Busqueu les seccions de codi amb els marcadors de conflicte:

```
Hello,
<<<<<< HEAD:README
this is an example.
=====
this is just an example.
>>>>>> 77976da35a11db4580b80ae27e8d65caf5208086:README
Bye.
```

El primer bloc (entre <<<<<< i =====) és el codi conflictiu de la branca de destí (la que teniu seleccionada, és a dir, HEAD) i el segon bloc (entre ===== i >>>>>>) és la versió de l'últim commit de la branca d'origen (la que heu especificat a la comanda merge). Heu de decidir quines línies manteniu o no de cada bloc, eliminar els marcadors i guardar els canvis.

2. Tornar a fer merge:

```
git add .
git commit -m "Resolt conflicte en merge commit"
```

Que ara sí, generarà un *merge commit*.

3. Pujar, finalment, els canvis a la branca remota main.

```
git push origin main
```

## 7 Visualitzar branques

En qualsevol moment, i després d'un merge especialment, podria convenir-vos visualitzar l'arbre de branques. Una de les formes més amigables de fer-ho és aquesta:

```
git log --oneline --graph --decorate --all
```

## 8 Esborrar branques

Un cop acabada la feina en una branca i fet el *merge commit* a main, podríeu voler esborrar-la per tenir una visualització de les branques del repositori (Sec. 7) més clara i comprensible.

La comanda és la següent:

```
# Esborrar la branca local un cop fusionada
git branch -d feat/uc1-log-in
# Esborrar la branca del remot
git push origin :feat/uc1-log-in
```

Tingueu en compte que `-d` només esborrarà la branca si ja ha estat fusionada amb alguna altra branca (en aquest cas, main) i, per tant, no té *commits* pendents de fusionar. En cas contrari, per forçar l'esborrat i perdre els *commits* no fusionats, haureu d'utilitzar `-D` (enlloc de `-d`) i, seguidament, esborrar-la remot amb el mateix push.

## Preguntes i respostes

Per acabar, algunes preguntes que creiem que us podríeu estar plantejant:

**P: He d'implementar un cas d'ús sencer en una branca de funcionalitat abans de fusionar-la amb el main?** R: No necessàriament. La qüestió important és el temps que trigueu per implementar el que us haguéssiu proposat implementar-hi des que vàreu crear la branca; perquè tenir més temps la branca sense fusionar, vol dir – inevitablement – més conflictes de merge quan fusioneu. Per tant, podeu pensar en estratègies per partir els casos d'ús en parts de funcionalitat (flux principal del cas d'ús i fluxos alternatius) o dividir la funcionalitat en tasques d'implementació. Un exemple de divisió en tasques podria ser la següent: `feat/uc1-log-in/activity` (que implementi l'activitat `LogInActivity`), `feat/uc1-log-in/viewmodel` (que implementi el `LogInViewModel`), `feat/uc1-log-in/client` (per crear la classe `Client`), etc. No hi ha una única manera vàlida de fer-ho. Poseu-vos d'acord en la vostra manera de fer i, això sí, sigueu consistents dins de l'equip.

**P: Podem utilitzar altres eines que no siguin només la línia de comandament de git?** R: Si voleu, sí. No us ho podem, ni volem prohibir. Nosaltres expliquem com fer les coses amb la línia de comandament perquè és el que us demanem aprendre a l'assignatura (i sobre el que se us avaluarà). I us ho demanem perquè és la manera de fer que proporciona més control i està menys subjecte al canvi. Ara bé, entenem que arribats a cert punt, podrien resultar-vos més còmodes altres opcions, clar que sí.

**P: No cal que fem *pull-requests*, doncs?** R: No encara. Els *pull-requests* els reservarem per més endavant, per quan tingueu més agilitat programant amb Android. Ara bé, si hi ha algun equip que s'hi vegi amb cor, endavant.