VNIVERSITAT DE VALÈNCIA

Exercicis del curs

Raúl Salinas Monteagudo

Taula de continguts

1 Abans de començar

Durant la sessió es fan els exercicis 1, 2, 3, 3b i 5. Tots necessiten només Git i accés al repositori: **no cal muntar cap entorn ni instal · lar cap base de dades**.

L'exercici 4 es veu **en demostració**: el ponent arranca l'Oracle efímer en pantalla i vosaltres mireu. Està escrit ací amb totes les passes per si el voleu repetir després amb calma; per a fer-ho farà falta Docker amb `compose` i baixar la imatge, que són uns 2 GB:

```
docker pull gvenzl/oracle-free:23-slim-faststart
```

L'exercici 6 tampoc es fa a classe: queda com a pràctica posterior, i és el que converteix el curs en una manera de treballar.

1.1 Dues coses abans de la primera comanda

Digueu-li a Git quin editor voleu. En algun moment l'obrirà a soles —per exemple, en tancar una integració amb conflicte—. Si no li ho dieu, obrirà `vi`:

```
git config --global core.editor nano
```

En compte de `nano`, poseu el que useu: `vim`, `gedit`, `kate`, o en Windows `notepad`. Amb Visual Studio Code cal `git config --global core.editor "code --wait"`.

Açò és **configuració de Git**, no del terminal: es guarda i **no cal repetir-ho** cada vegada que obriu una finestra nova.

Poseu el vostre identificador de la Universitat en una variable. Treballem tots en el mateix repositori, així que les branques han de dur el vostre nom o xocareu entre vosaltres:

```
export UVLOGIN=raulsa8      # ← el vostre, no el meu
```

Si obriu un terminal nou, torneu a executar-ho.

> **i** Si no useu un terminal tipus Unix
>
> L'enunciat dona per fet un terminal tipus Unix: **Git Bash** (ve amb Git per a Windows), WSL, MobaXterm o un Linux de veres. És el que recomanem, perquè tot funciona tal com està escrit. Si esteu en **PowerShell**, la variable es posa d'una altra manera:
>
> ```
> $env:UVLOGIN = "raulsa8"
> ```
>
> I si no voleu complicar-vos, **escriviu el vostre identificador a mà** allà on l'enunciat pose `$UVLOGIN`: la branca es dirà `validacio-cites-raulsa8`.

> **i** En Windows
>
> Tot el que hi ha ací són comandes de **Git**, així que funcionen igual en Git Bash, en PowerShell o en `cmd`. Si teniu Git Bash —ve amb Git per a Windows—, useu-lo: és el més paregut al que veureu en pantalla.

2 Exercici 1: primer commit

Objectiu: entendre el cicle `modificar → revisar → commit`.

Este primer commit el fareu en el vostre `main` local, sense branca. És a posta: les branques venen en l'exercici 2, i no té sentit crear-ne una abans de saber què és.

I que ningú patisca: el `main` del servidor **no el toca ningú**. El vostre és una còpia vostra; a partir de l'exercici 2, tot va en la vostra branca personal, que és com es treballa de veres.

Feu les passes d'una en una i mireu l'eixida de cadascuna. El valor de l'exercici està justament en el que diu Git entremig.

Primer, on estem:

```
git status
```

Obriu el fitxer:

```
$EDITOR examples/consulta_tablespaces.sql
```

I afegiu-hi **esta línia dalt de tot**, canviant les inicials per les vostres:

```
-- Ocupació dels tablespaces, de major a menor. Revisat per: XX
```

Guardeu i tanqueu l'editor.

Ara mireu què heu tocat:

```
git diff
```

Trieu eixe canvi per al pròxim commit, i torneu a mirar l'estat: ara el fitxer apareix davall de *Changes to be committed*.

```
git add examples/consulta_tablespaces.sql
git status
```

I consolideu-lo, amb el motiu:

```
git commit -m "Documenta consulta de tablespaces"
```

3 Exercici 2: branca de treball

Objectiu: modificar sense tocar la branca principal.

A partir d'ací, **tot en la vostra branca**. Creeu-la i comproveu que hi esteu:

```
git switch -c validacio-cites-$UVLOGIN
git branch --show-current
```

Obriu `examples/validacio_cites.sql`:

```
$EDITOR examples/validacio_cites.sql
```

L'última línia és esta:

```
HAVING COUNT(*) > 1;
```

Substituïu-la exactament per esta altra, que és la mateixa comprovació però ignorant les files sense centre:

```
HAVING COUNT(*) > 1 AND centre IS NOT NULL;
```

Fixeu-vos que és **l'última línia**: això importa per a l'exercici 3b.

Guardeu, i feu el commit:

```
git add examples/validacio_cites.sql
git commit -m "Afig validació de cites duplicades"
```

Fins ací **només existix en la vostra màquina**. Per a publicar la branca:

```
git push -u origin validacio-cites-$UVLOGIN
```

4 Exercici 3: revisió

Objectiu: obrir una pull request i comentar un canvi d'una altra persona.

Comprovacions que cal fer:

- El canvi té tiquet o explicació?
- El SQL té `WHENEVER SQLERROR` si modifica dades?
- Hi ha validació prèvia?
- Hi ha validació posterior?
- S'ha evitat incloure secrets?
- El `WHERE` és prou restrictiu?

5 Exercici 3b: conflicte i recuperació

Objectiu: veure què fa Git quan dues persones toquen la mateixa línia, i recuperar una versió anterior.

Partiu de la vostra branca `validacio-cites-$UVLOGIN` de l'exercici 2, on vàreu canviar la línia del `HAVING`. Si no la vàreu tocar **eixa línia exacta**, torneu arrere i feu-ho ara: un conflicte només apareix quan les dues bandes han tocat **la mateixa línia**, i això ja diu molt de com funciona.

Ara toca l'altra banda del conflicte. I convé dir una cosa: **això no és una simulació**. Un conflicte apareix de dues maneres, i les dues són igual de reals:

- un **company** ha tocat eixa mateixa línia;
- o l'heu tocada **vosaltres mateixos en una altra branca**.

La segona passa més del que sembla: estàs fent una cosa, et diuen que una altra corre pressa, tanques el que tens en un commit, obris una branca nova partint de `main`… i quan tornes a ajuntar-ho, les dues toquen el mateix. Git no distingix qui va fer cada banda: veu **dos canvis damunt del mateix punt de partida**.

Ací fareu la segona, que a més és l'única que podeu provar sense dependre de ningú. Tot es queda en la **vostra còpia**: no pujarem res, i el vostre `main` local se separarà del del servidor sense afectar ningú.

Passeu al vostre `main`:

```
git switch main
```

Obriu el mateix fitxer i canvieu **l'última línia** per esta altra —la mateixa comprovació, escrita d'una altra manera:

```
HAVING COUNT(*) >= 2;
```

I feu el commit:

```
git commit -am "Reescriu la condició de duplicats (INC-1234)"
```

Torneu a la vostra branca i proveu d'integrar `main`:

```
git switch validacio-cites-$UVLOGIN
git merge main
```

Git s'atura i deixa els dos candidats marcats dins del fitxer:

```
<<<<<<< HEAD
... la vostra versió
=======
... la versió de main
>>>>>>> main
```

Obriu-lo, deixeu la versió correcta, lleveu les tres línies de marcadors i tanqueu la integració:

```
git add examples/validacio_cites.sql
git commit
```

> ⚠ Ací s'obri un editor
>
> El `git commit` d'una fusió **no du `-m`**: Git ja ha escrit el missatge tot sol i només vol que el confirmes. Si s'obri una pantalla estranya de text, és Vim: polsa `:wq` i Intro (dos punts, ve, cu). Si prefereixes evitar-ho, fes `git commit --no-edit`, que accepta el missatge que Git proposa sense obrir res.

Fixeu-vos en el que acaba de passar: **ningú ha perdut res**. En una carpeta compartida, el segon que hagués guardat s'hauria emportat el canvi de l'altre sense cap avís. La contrapartida és que algú ha hagut de decidir, que és exactament el que una màquina no pot fer per tu.

Proveu ara què passa si oblideu llevar els marcadors i feu `git commit` igualment: la pipeline de comprovacions estàtiques els busca, i el canvi es queda en roig.

Per acabar, recupereu una versió anterior del fitxer. Primer, l'historial:

```
git log --oneline --all -- examples/validacio_cites.sql
```

El `--all` hi és a posta. Sense ell **no ix** el commit del company que acabeu de fusionar, i val la pena entendre per què: com que la resolució es va quedar amb la vostra versió, per a Git el camí que explica el fitxer és el de la vostra branca, i simplifica l'historial seguint-lo només a ell. No és cap error; amb `--all` es veuen els dos costats.

Ara copieu el **hash** del commit que vulgueu mirar —els set caràcters del principi de la línia— i useu-lo:

```
git show a183fe2:examples/validacio_cites.sql     # com estava el fitxer
git diff a183fe2 -- examples/validacio_cites.sql  # què ha canviat des d'aleshores
```

Copiar el hash en compte de comptar posicions amb `HEAD~2` no és manies: `HEAD~2` depén de quants commits hàgeu fet pel camí, i a cada u li apuntarà a un lloc distint. El hash no depén de res.

I compareu les tres operacions que se solen confondre: `git show` mira, `git restore --source=a183fe2 examples/validacio_cites.sql` torna el fitxer enrere en el vostre directori, i `git revert` desfà un canvi ja integrat amb un commit nou. En una branca compartida, la bona és sempre l'última.

6 Exercici 4: entorn de proves efímer

A la sessió es veu en demostració. Ací queda per a repetir-lo després.

Objectiu: comprovar que l'entorn és reproduïble sense instal · lar res.

Arranqueu l'entorn:

```
docker compose -f examples/docker-compose.yml up -d
```

I mireu com arranca, que la primera vegada tarda:

```
docker compose -f examples/docker-compose.yml logs -f oracle
```

Quan aparega `DATABASE IS READY TO USE`, l'esquema i la càrrega sintètica ja estan aplicats. Executeu la validació de negoci i comproveu que retorna zero files:

```
docker compose -f examples/docker-compose.yml exec -T oracle \
    sqlplus -S matricula/local_throwaway@localhost/FREEPDB1 \
    < examples/validacio_cites.sql
```

No fa falta instal · lar cap client: el contenidor ja en porta un.

Després afegiu al final d'`examples/02_seed_matricula.sql` una fila que duplique la franja d'una altra:

```
INSERT INTO matricula.centre_franja VALUES (3, 'FIS', 'G1100', TO_DATE('2026-09-01 09:00', 'YYYY-M
```

Recreeu l'entorn i torneu a validar. Ara la consulta ha de retornar una fila, i la pipeline hauria de fallar pel mateix motiu:

```
docker compose -f examples/docker-compose.yml down -v
docker compose -f examples/docker-compose.yml up -d
```

Per acabar, canvieu la versió de la imatge en `examples/docker-compose.yml` i torneu a arrancar l'entorn. Provar una actualització és un canvi d'una línia: si les validacions passen, el salt està provat; si no, s'esborra la branca i no ha passat res.

7 Exercici 5: CI

Objectiu: provocar una fallada i corregir-la.

Idees:

- Afegir una cadena que parega una contrasenya.
- Posar un `DELETE` sense `WHERE`.
- Llevar el `README.md` d'un procés.
- Afegir un marcador de conflicte `<<<<<<<`.

8 Exercici 6: un script nou naix versionat

Objectiu: crear des de zero el primer patró reutilitzable per als scripts nous del servei.

Passos:

1. Crear una branca.
2. Afegir un script de comprovació (per exemple, de tablespaces o d'objectes invàlids).
3. Afegir un `README.md` amb l'objectiu i els paràmetres.
4. Incorporar una comprovació prèvia i una posterior.
5. Afegir una plantilla de configuració **sense secrets**.
6. Provocar una fallada de CI (per exemple, un dels patrons de l'exercici 5).
7. Corregir-la.
8. Obrir una pull request i revisar-la.

El resultat és un primer patró reutilitzable: la manera com naixeran els scripts operatius nous del servei.