

---

# Control de versions amb Git

---



**git**

## CONCEPTES PREVIS I FONAMENTS

**Repositori.** És un magatzem on podem guardar i organitzar informació digital i des d'aquest magatzem fer-ne difusió.

**Sistema de Control de versions de software (VCS).** Són sistemes pensats per a facilitar les tasques de desenvolupament de software ajudant als equips de desenvolupadors a tenir un control eficient sobre les diferents versions del codi i altres documents.

N'hi ha diversos com **Subversion**, **Mercurial**, **Bazaar**, etc. Nosaltres farem servir **Git** que és probablement el més utilitzat.

N'hi ha de dos tipus:

- **Centralitzats.** Hi ha un sol repositori al que accedeixen tots els membres de l'equip de desenvolupament. **Subversion** pertany a aquest grup.
- **Distribuïts.** Cada membre de l'equip de desenvolupament treballa en la seva còpia local del repositori. Aquestes còpies se sincronitzen amb el repositori central. **Git** pertany a aquest grup. Els sistemes de control de versions distribuïts també s'identifiquen amb les sigles **DVCS**.

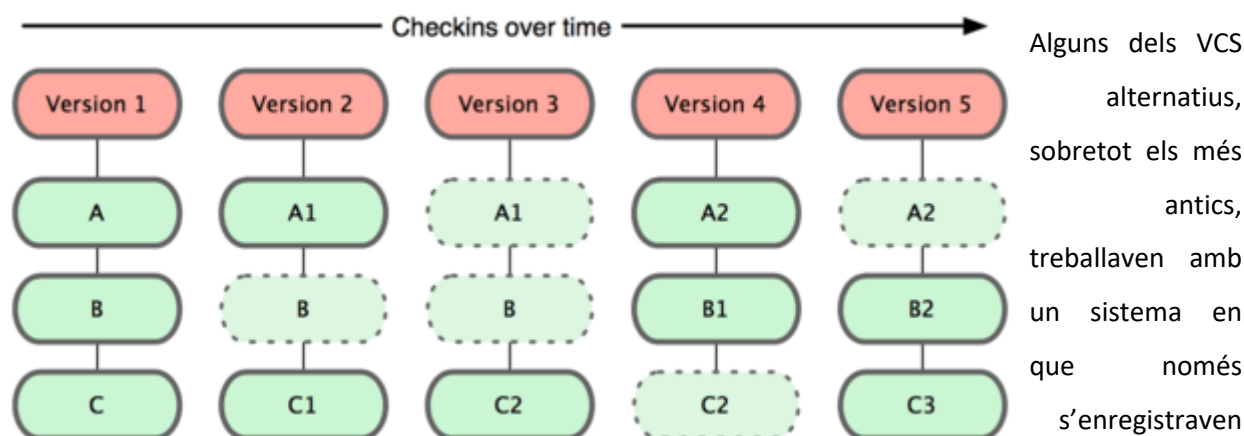
**Origen de Git.** Git va ser creat el 2005 per Linus Torvalds quan es va privatitzar **BitKeeper** que era el sistema de control de versions que estava utilitzant la comunitat de desenvolupadors de Linux.

Els objectius principals que es van plantejar en el desenvolupament de Git van ser:

- Rendiment alt. Que fos un sistema ràpid.
- Senzillesa.
- Orientat al desenvolupament no lineal. Es va establir un sistema de branques que permetés el desenvolupament en paral·lel.
- Completament distribuït.
- Orientat a grans projectes.

## Instantànies

Git treballa amb snapshots o instantànies la qual cosa ens permet accedir a l'estat en que estava el projecte sencer en un moment determinat tal com mostra la figura. Els mòduls en línia discontinua són els que s'han modificat respecte a la versió anterior.



les modificacions fetes als fitxers tal com mostra la figura següent.

## Integritat

Tot allò que es guarda en els repositoris de Git és verificat utilitzant un CheckSum basat en SHA-1

<http://www.sha1-online.com/>

## GitHub

GitHub és una de les plataformes web que permet centralitzar repositoris Git. És molt flexible i fàcil d'utilitzar. Des de que l'octubre de 2018 Microsoft la va comprar molts dels desenvolupadors han migrat a GitLab que és d'Atlassian.

## Conceptes

**Commit** És una instantània de l'estat d'un projecte en un moment determinat.

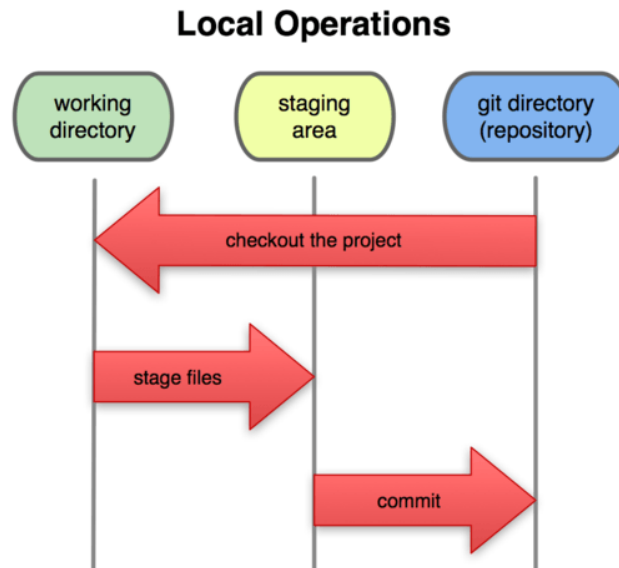
**Branca** Seqüència de commits ordenada cronològicament. La branca principal d'un projecte s'acostuma a anomenar **master**

**Workspace** Directori o àrea de treball on es van creant les diferents versions del projecte, vindria a ser el directori arrel del repositori

**Índex** També s'anomena **àrea de canvis** o **staging area** i és un registre de tots els canvis que s'han fet des del darrer commit i que seran implantats en el pròxim commit.

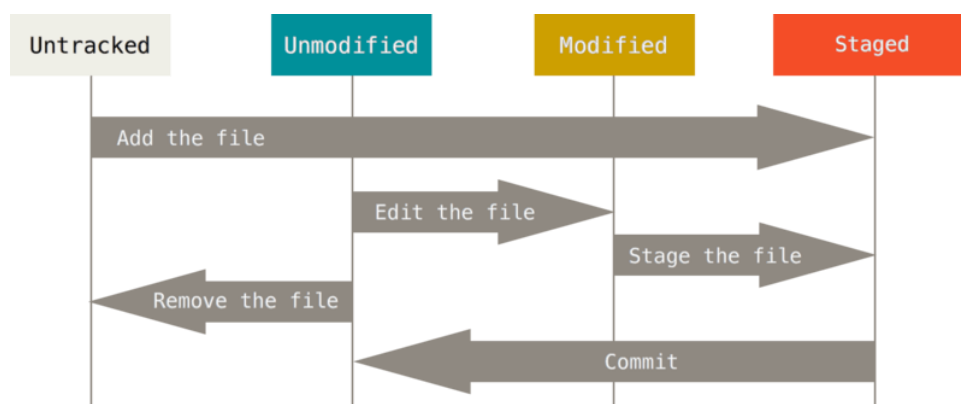
Allò que no estigui indexat (enregistrat) no es tindrà en compte en el pròxim commit. Els canvis no s'afegeixen a l'índex si no ho fem de manera explícita mitjançant `git add`

## Flux dels projectes Git



## Estats dels arxius

Els arxius que tenim en un projecte Git poden estar en un dels següents estats:



Podem veure l'estat dels arxius amb l'ordre `git status`

## Referències

<https://git-scm.com/doc>

<https://git-scm.com/book/es/v1/Empezando>

<https://git-scm.com/downloads>

<https://es.atlassian.com/git/tutorials>

<https://es.atlassian.com/git/tutorials/atlassian-git-cheatsheet>

<https://guides.github.com/introduction/flow/>

<http://rogerdudler.github.io/git-guide/index.es.html>

## Instal·lar Git Bash

El Git Bash és el Shell que ens permet treballar amb els repositoris de Git a la nostra màquina. El descarreguem de

<https://git-scm.com/downloads>

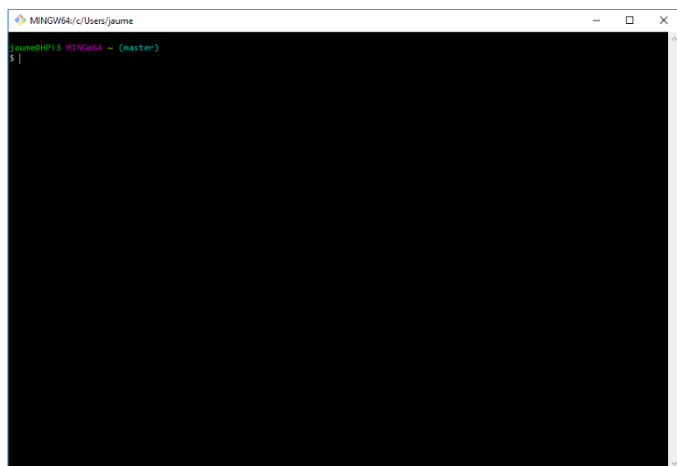
Quan ja hem instal·lat el Git Bash, en una de les carpetes tindrem tot de pàgines html amb el manual. Aquestes pàgines són les que es mostren quan a la consola fem servir l'opció `--help` de qualsevol ordre i són a la carpeta

`C:/Program Files/Git/mingw64/share/doc/git-doc/`

Amb el Git Bash només podem treballar a nivell local, com que volem disposar de repositoris que ens permetin treballar en equip necessitarem crear un compte, en el nostre cas, a GitHub ja que podem sol·licitar el pack d'educació.

## Configuració

Entrem al git bash que és semblant a Linux. Si fem ls veurem les carpetes que tenim.



Primer de tot hem de configurar les nostres credencials perquè no podem ser anònims. Per a veure els valors de totes les variables de configuració fem

```
git config --list
```

Si no surten les variables de l'usuari o estan buides les hem de configurar.

```
user.name=  
user.mail=  
user.email=
```

Les configurarem amb

```
git config --global user.name "nom alumne"  
git config --global user.email "adreça de mail"
```

global indica que afecta a tot el conjunt de repositoris Git, local afectaria només al repositori en que estiguem treballant

Tornem a fer un list per a veure si s'han agafat els valors

```
user.name=Jaume Fadó  
user.mail=jfado@educem.com  
user.email=jfado@educem.com
```

Podem consultar el valor de configuració d'una variable determinada fent

```
git config user.name
```

## Primer projecte – dam2m5uf2/projecte\_zero

mkdir dam2m5uf2 cd dam2m5uf2	Fem la carpeta <b>dam2m5uf2</b> i ens hi situem
mkdir projecte_zero cd projecte_zero	Fem la carpeta <b>projecte_zero</b> i ens hi situem
git init	Inicialitzem el projecte. <code>git init</code> converteix el directori on som (o opcionalment l'indicat) en un espai de treball de Git. Es genera l'arxiu <code>.git</code> que podrem veure amb <code>ls -a</code>
Fem el fitxers <code>LICENSE</code> i <code>README.TXT</code> . A dins hi escrivim qualsevol cosa. Ho podem fer amb el nano a la consola però serà més fàcil fer-ho gràficament anant a la carpeta del repositori que serà a <code>C:\Users\compte</code> .	
git status git status --s (-short)	<p>Consultem l'estat i ens surten els arxius que hem creat indicant que estan <i>untracked</i> que vol dir que no hi era en el darrer commit i que no han estat enregistrats a l'índex de canvis. En format curt ens surt</p> <pre>?? LICENSE ?? README.TXT</pre> <p>Els estats dels fitxers, a banda d'<i>untracked</i> també tenim <i>modified</i> (modificat des del darrer commit) i <i>staged</i> (enregistrat en l'índex de canvis preparat per al pròxim commit).</p>
git add LICENSE README.TXT	<p>Enregistrem a l'índex de canvis els dos fitxers al repositori. Si volguéssim enregistrar tots els que hi ha a la carpeta hauríem de posar <code>git add .</code></p> <p>Si volem treure del registre un o més fitxers ho hem de fer amb <code>git reset</code></p>
git status git status --s (-short)	<p>Consultem l'estat de les tasques pendents des del darrer commit (encara no n'hem fet cap). En format curt ens surt</p> <pre>A LICENSE A README.TXT</pre> <p>La lletra A en verd indica que hem fet Add.</p>
mv README.TXT README.MD.TXT git status --s  git add .	<p>Canviem el nom del fitxer README.TXT amb l'ordre <code>mv</code> (sense el prefix git) perquè ens hem equivocat. L'status ens mostra que s'ha eliminat l'arxiu README.TXT i el nou surt amb ?? perquè no està enregistrat a l'índex de canvis.</p> <p>Tornem a afegir els arxius a l'índex. Posant el punt enlloc de noms de fitxers afegirem tots els arxius i carpetes que han canviat.</p>

<code>git status -s</code>	<p>Veiem que els dos arxius estan correctament indexats.</p> <p>També ho podríem haver fet, però és més feina, amb <code>git reset</code> que el que fa és eliminar el registre que hem fet amb <code>git add</code>. Llavors hauríem d'enregistrar novament l'arxiu a l'índex.</p> <pre>git reset README.MD.TXT git status -s mv README.MD.TXT README.MD git add README.MD</pre>
<code>git mv README.MD.TXT README.MD</code> <code>git status -s</code>	Si el canvi de nom del fitxer el fem amb el prefix <code>git</code> els canvis s'apliquen directament sobre l'índex, tot és més coherent i, sobretot, tenim menys feina.
<code>git commit -m "Llicència i fitxer ReadMe"</code>	Fem un <code>commit</code> , és a dir, una instantània del repositori. Equival a un punt de tall.
<code>git status -s</code>	Ara l'estat no ens mostra cap tasca perquè acabem de fer el commit.
<code>git log</code> <code>git log --oneline</code>	<p>Mostra la relació de commits en format llarg o format curt.</p> <p>Cada commit té un identificador únic de 40 dígits generat com a clau de hashing SHA1 <a href="http://www.sha1-online.com/">http://www.sha1-online.com/</a></p> <p>Com es veurà més endavant Git treballa amb branques que són rutes diferenciades de desenvolupament. Tots els repositoris tenen per defecte una primera branca que és la branca <code>master</code>. A més de l'identificador del commit també ens surt el nom de la branca a que està associat.</p> <p><code>HEAD</code> és el punter arrel que indica el punt on estem treballant que és a partir del darrer commit.</p> <p>Més endavant veurem que <code>git log</code> té molts paràmetres que permeten triar el format i la quantitat d'informació que volem que es mostri.</p>

## Llicències

<https://choosealicense.com/>

Al fitxer LICENSE hi posem el text de la llicència del MIT i també modifiquem el fitxer README.MD posant-hi un text explicatiu sobre el projecte.



<code>git status -s</code>	La lletra <b>M</b> es mostra en vermell perquè no l'hem enregistrat a l'índex.
<code>git add LICENSE</code> <code>git status -s</code>	Ara l'estat ens mostra que l'arxiu LICENSE s'ha modificat però la lletra <b>M</b> és verda. Al fitxer README.MD també hi ha la <b>M</b> però és vermella perquè no l'hem afegit a l'índex.
<code>git diff</code>	Mostra les diferències de l'arxiu README.MD respecte el commit anterior. El fitxer LICENSE no surt perquè està enregistrat a l'índex.
<code>git diff --cached</code> <code>git diff --staged</code> (opció també vàlida)	Ara veiem les modificacions dels arxius enregistrats, en aquest cas només el LICENSE. També podríem haver fet <code>git diff --cached LICENSE</code>
A GitHub fem un nou repositori que es digui <code>nomalumne_dam2m5uf2projecte_zero</code> Una vegada creat tindrem una URL del repositori que acaba amb <code>.git</code> i l'hem de copiar. A la consola posarem <code>git remote add origin URL</code>	
<code>git push http://.... master</code> <i>master és la branca principal del repositori</i>	Pugem el repositori a GitHub. Fixem-nos que no hem fet cap altre commit, per això el fitxer LICENSE de GitHub conté el text antic.
Crea un tercer fitxer que es dirà <code>AUTHOR.TXT</code> en el que hi poses les teves dades	

**EXERCICI.** Puja a GitHub la versió actual, la que té l'autor, amb un nou commit.

**EXERCICI.** Canvia el nom del fitxer `AUTHOR.TXT` per `AUTHOR` i fes un commit i puja-ho a GitHub.

<code>git log</code> <code>git log --oneline</code>	Se'ns mostra l'historial de commits en format llarg o curt. HEAD indica el punt on estem situats, en aquest cas la branca <code>master</code> i el darrer commit que hem fet.
<code>git log --graph</code>	Ens surt una línia discontinua que fa de graf indicant les branques. Ara només hi ha una línia perquè només tenim la branca master.

**EXERCICI.** Situa't a la carpeta arrel del teu Git Bash i utilitza `git clone` per a baixar-te el repositori remot <https://github.com/jfedcmnet/webzero> que conté una pàgina web que faràs servir en els pròxims exercicis.

```
git clone https://github.com/jfedcmnet/webzero clon_del_web0
```

`git clone` ens permet fer una còpia des de repositoris que són públics perquè el propietari els ha posat a disposició de tothom o també podem clonar des d'un repositori privat si el propietari ens ha posat com a col·laboradors o en sabem les dades del compte GitHub associat.

El nom `clon_del_web0` és el nom que li volem donar localment al repositori un cop clonat. Si no posem nom agafarà el del repositori remot, en aquest cas `webzero`.

Com alternativa podíem haver posat l'ordre utilitzant el `protocol git` enlloc d'`https`

```
git clone git://github.com/jfedcmnet/webzero clon_del_webzero
```

**EXERCICI.** Al repositori `projecte_zero` crea una carpeta `web0` i posa-hi els arxius que tens dins la carpeta de `clon_del_webzero`. **Alerta!!! No hi posis la carpeta `.git`.**

Fes els passos necessaris per a actualitzar el repositori `projecte_zero` a GitHub.

**EXERCICI.** Modifica els estils per a que la pàgina web tingui un altre tipus de lletra i els colors siguin uns altres.

Busca una imatge per a canviar el logo del cohet. Guarda-la sense perdre el cohet i modifica els arxius necessaris per a que mostri el nou logo sense aplicar-li cap rotació.

Comprova les diferències entre els arxius del darrer commit i els actuals.

Puja els canvis a GitHub i visualitza quin commit ha modificat cada arxiu.

**EXERCICI.** Fes una versió del web0 traduïda al català modificant el mínim possible d'arxius i posa-la en una carpeta `web0\webcat`.

Comprova les diferències entre els arxius del darrer commit i els actuals.

Puja els canvis a GitHub i visualitza quin commit ha modificat cada arxiu.

Dins la carpeta <code>web0</code> fem una carpeta <code>img</code> i movem els arxius d'imatge a aquesta carpeta. Després modifiquem els <code>index.html</code> per a que trobin correctament el logo.	
<code>git status</code>	Veiem els canvis que s'han fet i destaquem que surten en vermell perquè no hem aplicat <code>git add</code>

<code>git diff</code>	Mostra els canvis fets
<code>git add .</code>  <code>git diff</code>  <code>git status</code>	Afegim els canvis a l'índex  Ja no es mostren diferències perquè no n'hi ha des de que hem afegit els arxius a l'índex, caldria posar el paràmetre <code>--cached</code>  L'estat surt en verd i les operacions han variat una mica des de la consulta anterior de l'estat. Ara tenim <code>renamed</code> i abans teníem <code>deleted</code> i una <code>nova carpeta</code>
<code>git commit -m "Carpeta imatges"</code>	Fem el commit dels darrers canvis
<code>git log</code>	Mirem la relació de commits
<p>Imaginem que ens hem equivocat i volem desfer aquest darrer commit. Ho farem amb <code>git reset --soft HEAD~1</code></p> <p>Aquesta ordre desfà el darrer commit. Si haguéssim posat <code>HEAD~2</code> desfaria el darrer i el penúltim, si posem <code>HEAD~3</code> desfaria el darrer, el penúltim i l'anterior al penúltim,...</p> <p>El paràmetre <code>--soft</code> fa que els arxius quedin a l'índex tal com estaven abans del commit que estem eliminant.</p>	
<code>git log</code>  <code>git status</code>	Comprovem que el commit "Carpeta imatges" ja no hi és Comprovem que l'estat dels arxius és el d'abans de fer el commit eliminat.
<code>git commit -m "Un altre cop carpeta imatges"</code>	Repetim el commit que havíem eliminat amb un altre nom
<code>git log</code>	Comprovem que el commit "Un altre cop Carpeta imatges" s'ha fet bé.
<code>git reset --hard HEAD~1</code>	Desfem el darrer commit però aquest cop amb el paràmetre <code>--hard</code>
<code>git log</code>  <code>git status</code>	Comprovem que el commit s'ha eliminat  Veiem que també ha desaparegut la carpeta d'imatges i que les imatges han tornat a la carpeta anterior. Utilitzar <code>--hard</code> és arriscat perquè també podem perdre codi dels arxius que hem modificat.

Hem pogut desfer el commit anterior amb <code>git reset</code> perquè no l'havíem pujat a GitHub. Si el push del commit ja estigués fet enlloc de <code>git reset</code> farem servir <code>git revert</code> . Tant una opció com l'altra són arriscades, sobretot si utilitzem <code>--hard</code> o volem desfer un commit que no és l'últim.	
Tornem a fer la <code>carpeta img</code> i els canvis indicats i també fem el <code>git add .</code> i el commit de la <code>carpeta img</code> però <b>no fem el push</b> .	
Ara crearem les carpetes <code>css</code> i <code>js</code> dins de <code>web0</code> i hi posarem els arxius que pertoca i modificarem convenientment el fitxer <code>index.html</code>	
<code>git add .</code> <code>git commit -m "Carpetes CSS i JS"</code> <code>git push .....</code>	Fixarem els canvis al repositori i els pujarem a GitHub. Després del push comprovem a GitHub els commits que hi tenim.
<code>git revert HEAD~1</code>	S'obre l'editor de textos per defecte del Windows mostrant-nos un script. No fem cap canvi i guardem l'script
<code>git push ....</code>	Després del push comprovem a GitHub els commits que hi tenim i veurem que n'hi ha un més que és el Revert de l'anterior.

Tot i que es pot fer, no és aconsellable fer `reset` o `revert` de commits que no siguin el darrer que hem fet ja que això podria deixar moltes incoherències en el codi. Val més que fem nous commits per a revertir la situació que serà menys arriscat.

## Exclusió d'arxius amb `.gitignore`

De vegades hi ha arxius que són temporals o són de dades o són versions antigues o són d'algun altre tipus que tot i que els tenim dins de les carpetes del repositori no volem que Git els tingui en compte.

Per a això el que hem de fer és crear l'arxiu `.gitignore` a l'arrel del repositori. Dins d'aquest arxiu hi escriurem els arxius que volem excloure.

Exemple d'arxiu `.gitignore`

```
*.dat
*.bak
tmp/
```

**EXERCICI.** Al Git Bash crea l'arxiu `.gitignore` de l'exemple anterior i puja'l a GitHub.

**EXERCICI.** A la carpeta del repositori local crea un arxiu que es digui `dades.dat` , un altre arxiu que es digui `dades.dat.old` i crea també una carpeta que es digui `tmp`. Dins de la carpeta tmp posa-hi un parell d'arxius qualsevols. Després ves a la carpeta on hi ha l'`index.html` del web0 i fes-ne una còpia que es digui `index.html.bak`

Fes `git add` . i comprova quins de tots els arxius que has creat en aquest exercici s'afegeixen a l'índex. Després de fer la comprovació esborra aquests arxius.

<code>git remote -v</code>	<p>Amb aquesta ordre podem veure la relació de repositoris remots amb els que el repositori local s'ha connectat.</p> <p>Situa't al repositori <code>clon_web0</code> i fes <code>git remote -v</code> Ens surt que l'origen és el repositori remot amb la url des d'on l'hem clonat.</p> <p>Surt una línia <code>fetch</code> que indica el repositori remot des del que carreguem branques (fetch) i una <code>push</code> indicant el repositori remot on pugem els canvis fets al local.</p>
<code>git remote add ElMeuRepositori https://.....</code>	<p>Amb aquesta ordre el que fem és crear una mena d'alias a partir del qual ja no haurèm d'escriure la url del repositori remot sinó que ens hi podrem referir amb el nom <code>ElMeuRepositori</code>.</p> <p>Així doncs, quan haguem de fer push podrem posar <code>git push ElMeuRepositori master</code></p>

## Branques

Un dels avantatges de Git sobre molts altres sistemes de control de versions és la possibilitat de treballar amb **branques**.

Les branques permeten que en un moment determinat el desenvolupament es pugui dividir ja sigui perquè es fa en equip i cada membre de l'equip treballa en la seva pròpia part de codi o perquè un mateix desenvolupador es divideix la feina en tasques diferents. Quan el codi de branques diferents ja està acabat es poden fusionar branques (merge) i seguir treballant conjuntament amb el mateix codi.

Al principi del dossier hem explicat que Git treballa amb instantànies la qual cosa facilita la recuperació de versions. Si hi ha més d'una branca cada branca té les seves instantànies.

Per defecte, tots els repositoris comencen amb una sola branca que és la branca `master` o `main`. En el projecte zero has estat treballant amb aquesta branca única.

Depenent de la ubicació parlem de **branques locals** o **remotes**.

Les branques poden treballar de manera aïllada i llavors en diem **branques sense tracking**.

O poden treballar interrelacionades amb altres de remotes amb les quals se sincronitzen, llavors en diem **branques amb tracking**.

Seguim amb el projecte zero	
<b>git branch</b>	Consultem les branques locals que tenim, només ha de sortir la master. Surt en color verd perquè és la que està activa, si n'hi hagués d'altres sortien en blanc.
<b>git branch -r</b>	Mostra les branques remotes en color vermell. Situa't al repositori <b>clon_web0</b> i fes <b>git branch -r</b> Aquí sí que haurien de sortir branques remotes perquè les hem clonat d'un repositori remot. La línia <b>origin/HEAD</b> -> origin/master no és una branca sinó que indica quina és la branca inicial (HEAD és l'arrel del repositori).
<b>git branch -a</b> <b>git branch -a -vv</b>	Mostra totes les branques: locals i remotes. L'opció amb <b>-vv</b> ens dóna més informació indicant també el nom del darrer commit.
<b>git branch vespanyol</b> <b>git checkout vespanyol</b>	Fem una branca nova que es dirà <b>vespanyol</b> i amb el <b>checkout</b> ens hi situem.  Alternativa: Podíem haver-ho fet amb una sola ordre posant el paràmetre <b>-b</b> (build) a checkout <b>git checkout -b vespanyol</b>
<b>git branch -a -vv</b>	Veiem la nova branca i ens fixem que surt en verd perquè és la branca activa, la branca master ara surt en blanc. També veurem que master i vespanyol tenen el mateix commit com a darrer commit ja que la branca vespanyol comença en el punt en que ens trobem.

Si ens hem equivocat podem esborrar branques amb **git branch -d** (delete) o canviar-ne el nom amb **git branch -m** (move)

**EXERCICI.** Fes la carpeta **webes** dins de **web0** i posa-hi l'arxiu **index.html** convenientment modificat per a tenir la versió en espanyol.

Assegura't que la branca en la que estàs treballant és la nova i puja a GitHub la nova versió.

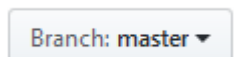
Ves al web de GitHub i fixa't que ara tens 2 branques. Mira'n les diferències.

<code>git log --online</code>	Veiem els commits que s'han fet i ens surt la branca on s'han aplicat.
<code>git checkout master</code>	Ens tornem a situar a la branca master
<code>git merge vespanyol</code>	Fusionem les dues branques
<code>git log --online</code>	Veiem les diferències respecte el git log anterior.
<code>git push ....</code>	Ho pugem a GitHub i veurem que ara tenim la versió en espanyol al web0 de la branca master i que la branca vespanyol es manté, no s'elimina.

<code>git branch -d vespanyol</code>	Suposem que la branca vespanyol era una branca només de treball i que ara que ja hem fet la fusió amb la master no la necessitem més. Amb aquesta ordre l'eliminem del repositori local.
<code>git push origin --delete vespanyol</code>	Per a eliminar la branca remota.
<code>git push origin :vespanyol</code> <code>git push ElMeuRepositori :vespanyol</code>	Si fem un push veurem que la branca remota no ha desaparegut. Per a eliminar-la cal fer servir aquesta ordre. Després d'executar-la comprova a GitHub que ja només tens la branca master. <b>origin</b> és l'alias que git ha assignat per defecte al repositori remot però es pot canviar pel d'un altre dels repositoris que tenim vinculats depenent d'on estigui la branca que volem eliminar.

**EXERCICI.** Crea una carpeta **webpt** amb la versió portuguesa de l'**index.html** però **assegura't que no la crees dins de la carpeta web0 del projecte\_zero sinó en una altra carpeta qualsevol del teu disc.**

Al repositori remot de GitHub crea una branca que es digui **vportuguesa**. Les branques es creen fent clic al botó



Puja-hi la carpeta **webpt** fent-ho a través del web, no des de la consola.

Ara el repositori local té només la branca master i el remot té la master i la vportuguesa.

Si a la consola fem **git branch -a** veurem que no surt la nova branca remota **vportuguesa**. Ara fem **git fetch** i tornem a consultar les branques. Veurem que ja surt la branca remota **vportuguesa** i si fem **git checkout vportuguesa** veurem que en local també la tenim.

De tota manera, si mirem a la carpeta web0 veurem que no hi tenim la carpeta **vportuguesa**. Això és perquè **git fetch** no descarrega els fitxers, només actualitza l'estructura de branques. Per a descarregar els fitxers hem de fer **git pull**.

Ens assegurem que som a la consola dins la branca vportuguesa i fem **git pull**.

Si estem situats a la branca master el que es farà és un merge de la master amb la vportuguesa

**git pull origin vportuguesa**  
**git pull ElMeuRepositori vportuguesa**

Amb **git pull**, a més d'actualitzar les branques, també descarreguem els fitxers.  
Amb **git checkout vportuguesa** ens hi podem situar.

**EXERCICI.** De manera semblant a com has fet abans amb la versió portuguesa ara crea una carpeta **webfr** en una carpeta que no sigui la del repositori i fes l'index.html de la versió francesa. A GitHub crea una branca **vfrancesa** i puja la carpeta **webfr** a aquesta branca.

Després fes els següents passos a la consola

**git checkout -b vfrancesa**  
**git pull origin vfrancesa**

Crearem manualment una branca vfrancesa al repositori local sense fer el fetch.  
  
Aquest pull farà un merge entre la branca remota i la local. Se'ns mostrarà un editor preguntant si volem posar un missatge. És un editor de tipus **vim** de Linux.

El que hem fet és semblant al que havíem fet amb la versió portuguesa però creant la branca local enlloc de fer el fetch.

**EXERCICI.** Situa't a la branca master de la consola i integra les versions francesa i portuguesa a la branca master amb **git merge** i puja els canvis a GitHub. Veuràs que no has de fer **git add** ni **git commit**, només el **git push**.

Comprova que tot està bé i ja podràs eliminar les branques portuguesa i francesa.

## Etiquetes



Git ofereix la possibilitat de posar etiquetes a punts concrets del desenvolupament. Això ens permetrà localitzar un commit etiquetat de manera ràpida.

Hi ha dos tipus d'etiquetes les **lleugeres** i les de **notes**. Nosaltres treballarem amb les de notes que ens seran més fàcils d'utilitzar, les lleugeres utilitzen el números de hashing dels commit i potser no són tan clares.

<code>git tag -a v1.0 -m "primera versió"</code>	Crea una etiqueta en el punt en que estem.
<code>git show w1.0</code>	Mostra la informació del darrer commit que s'havia fet en el moment de crear l'etiqueta.

## Git diff i Git log

Si volem veure les diferències entre els continguts de dues branques podem posar

`git diff branca1 branca2`

També podríem mirar les diferències entre dos commits posant

`git diff id_commit1, id_commit2`

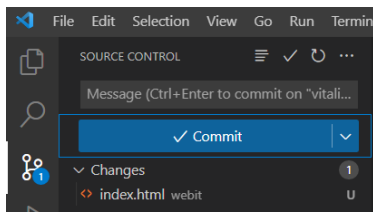
`git log` Fins ara hem utilitzat `git log` per a veure la història dels commits que hem fet però aquesta ordre admet diversos paràmetres que faran que la informació es mostri de manera més detallada o menys:

<code>git shortlog</code>	Versió resumida del log
<code>git log --online</code>	Resumeix la informació però menys que l'opció anterior
<code>git log -p</code>	Mostra les diferències del codi entre commits
<code>git log --stat</code>	Mostra el número de modificacions, insercions o eliminacions fetes en cada commit
<code>git log --graph</code>	Mostra la informació però de manera que podem seguir els commits per branques
<code>git log --graph --online</code>	Mostra una versió del graph més resumida i que acostuma a ser més manejable.

Hi ha molts altres paràmetres per a `git log` que permeten veure el log a partir d'una data, filtrant per l'autor del commit, etc. però aquests exemples anteriors són els que s'utilitzen més.

## Git i Visual Studio Code

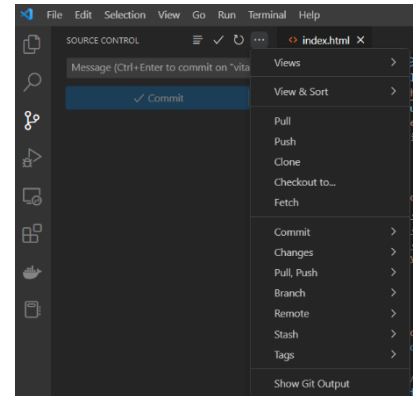
**EXERCICI.** Crea una branca **vitaliana** i fes la versió en italià del web creant una carpeta **webit**. Fes un commit



amb aquesta nova versió.

Puja-la a GitHub. Ho pots fer amb el botó **<Publish>** o anant al menú dels puntets com mostra la figura i triar **<Push>**

En aquest menú trobem altres opcions i subopcions





## Git merge

Fem un merge de la branca **vitaliana** per a que es fusioni amb la branca **master**. Ens situem a la branca **master** i anem a **<...><Branch><Merge>** i

triem la branca **vitaliana**. Si ara mirem els arxius veurem que a la branca **master** ja tenim la carpeta **webit**.

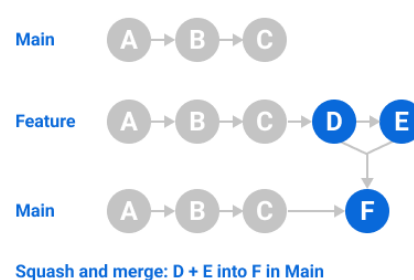
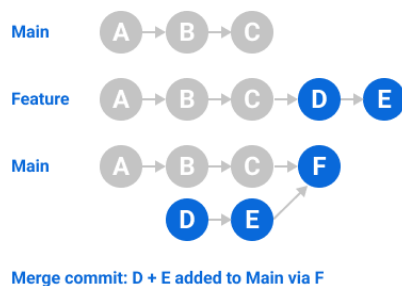
A Github encara no ho tenim fusionat, caldrà fer clic al botó **<Publish>** per a que es pugi.

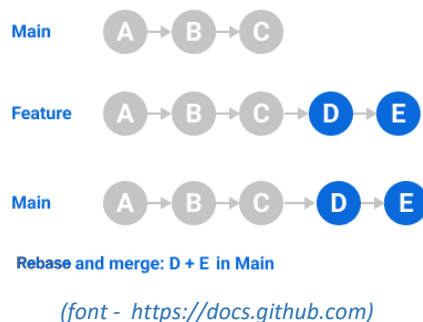
**Eliminar la branca vitaliana local i sincronitzar canvis.** Ens mantenim a la branca **master** i amb el menú **<...><Branch>** eliminem la branca **vitaliana**. Veurem que a Github no s'ha eliminat i que, encara que sincronitzem amb el botó tampoc  **master**  s'eliminarà.

Per a eliminar-la de Github ho podem fer des del terminal (del mateix Visual Studio Code o Git CMD o Git Bash) posant l'ordre **git push origin -d vitaliana**

## Git Squash i Git Rebase

Aquestes dues ordres són alternatives a Merge. La diferència principal és la manera com gestionen l'historial de commits una vegada feta la fusió d'una branca amb un altra (generalment fusionem amb la branca master o main però no té perquè ser sempre així)





## Altres ordres

**Commit amend.** Permet modificar el darrer commit.

**Commit signoff.** En projectes antics o dependents d'alguns kernels de Linux s'utilitza aquest mètode menys segur ja que no valida les credencials de manera tan exhaustiva.

**Fetch prune.** Neteja les branques que s'han eliminat al repositori local o al remot.

## Treball en equip

**Git clone** és l'ordre que fem servir per vincular un repositori local a un repositori remot. Ho podem fer a un repositori propi o a un de públic. Si ho fem en un repositori públic que no és nostre no hi podrem introduir modificacions, només les podrem treballar a nivell local.

**Git fork** és l'ordre que fem servir per a fer una còpia d'un repositori al que hi tenim accés perquè és públic o perquè ens han convidat a col·laborar-hi. Aquesta còpia es fa a través de GitHub i queda ubicada allà. Si fem modificacions a la còpia del repositori podem fer una **pull request** per a comunicar-nos amb el propietari i proposar-li l'acceptació dels canvis.

**EXERCICI.** Connecta't al teu compte de GitHub i fes un **fork** del repositori [jfedcm/equipdam2](#).

Quan ja tinguis la còpia del repositori en el teu compte fes un **clon** per a tenir aquesta còpia localment. La pots fer dins del **Visual Studio Code**.

Crea una **branca** que porti el teu nom i posa't a treballar en aquesta branca.

Fes el teu avatar amb <https://www.southparkstudios.com/info/lv0nha/avatar> i després afegeix el fitxer html que et correspon.

Fes un **commit** i puja la feina que has fet a **GitHub**.

Torna a GitHub i fes una **pull request** per a que el propietari del repositori pugui valorar si accepta els teus canvis. Al fer aquesta **pull request** has de posar en la descripció tota una explicació sobre els canvis o afegits que proposes.

Es podrà establir un diàleg amb el propietari que és qui finalment decidirà si fusiona la branca amb la `master` (`main`) o no. Un cop fusionada, podràs actualitzar la branca `master` (`main`) de la teva còpia del repositori.

## Referències

<https://visualstudio.microsoft.com/vs/github/>

[https://yourbrainoncomputers.com/using-git-with-visual-studio-code-the-ultimate-guide/#Merge\\_a\\_Branch\\_Into\\_Master](https://yourbrainoncomputers.com/using-git-with-visual-studio-code-the-ultimate-guide/#Merge_a_Branch_Into_Master)

<https://dev.to/thenomadevel/top-5-best-git-extensions-for-vs-code-you-must-have-40b6>