

Utilitza repositoris de versions de codi com Git i Github



Ajuntament de
Barcelona



Barcelona
Activa

Índex

1 INTRODUCCIÓ I INSTAL·LACIÓ	3
1.1 CONCEPTES PRINCIPALS	3
1.2 QUÈ ÉS GIT I GITHUB?	4
1.3 INSTAL·LACIÓ	6
1.4 CONFIGURACIÓ	9
1.5 IDEES CLAU: INTRODUCCIÓ I INSTAL·LACIÓ.....	12
 2 CONEIXENT GIT	 13
2.1 DISTRIBUCIÓ DELS REPOSITORIS GIT	13
2.2 CREANT I PARTICIPANT EN UN PROJECTE JA EXISTENT	14
2.3 <i>COMMITS</i> I MISSATGES	15
2.4 UTILITZANT BRANQUES.....	17
2.5 IDEES CLAU: CONEIXENT GIT.....	19
 3 TREBALLANT AMB REPOSITORIS GITHUB	 19
3.1 INTRODUCCIÓ A GITHUB	20
3.2 CREACIÓ D'UN REPOSITORI.....	23
3.3 ENLLAÇANT AMB EL NOSTRE REPOSITORI GITHUB	25
3.4 <i>PUSHEANT</i> I VERIFICANT CANVIS A GITHUB.....	29
3.5 ACTUALITZACIÓ DE CANVIS EN REPOSITORI LOCAL.....	30
3.6 CREANT NOUS FITXERS A GITHUB	32
3.7 TREBALLANT AMB FITXERS A GITHUB	33
3.8 REVISANT <i>COMMITS</i>	34
3.9 COMPARTINT REPOSITORIS	36
3.10 <i>PULL REQUESTS</i> A GITHUB.....	37
3.11 IDEES CLAU: TREBALLANT AMB REPOSITORIS GITHUB	40
 4 ALTRES REPOSITORIS	 41
4.1 GITLAB	42
4.2 BITBUCKET.....	43
4.3 SOURCEFORGE	44
4.4 ALTRES REPOSITORIS <i>ONLINE</i>	46
4.5 IDEES CLAU: ALTRES REPOSITORIS	47

1 INTRODUCCIÓ I INSTAL·LACIÓ

En aquest primer mòdul, ens endinsarem en el món de programes de control de versions, de la mà de Git i GitHub: sistemes per gestionar els canvis que es van aplicant a un conjunt d'arxius; que resulten sobretot útils quan es treballa en equip.

Començarem familiaritzant-nos amb els programes a través d'un exemple que ens permetrà veure les possibilitats que ofereixen aquests programes. A continuació, definirem els dos programes detalladament i veurem les seves característiques principals.

Finalment, explicarem les instruccions d'instal·lació i configuració als nostres ordinadors, per poder començar a treballar amb els programes nosaltres.

Comencem?

1.1 CONCEPTES PRINCIPALS

Benvinguts i benvingudes!

Començarem aquest vídeo parlant sobre el control de versions de Git i GitHub, dues eines que ens ajudaran a organitzar l'evolució dels nostres projectes i el treball en equip.

Comencem!

Què és el control de versions? Es tracta d'un sistema per gestionar els canvis que es van aplicant a un conjunt d'arxius. Quan estem treballant en un projecte, aplicar-li un control de versions farà que es registrin les modificacions dels seus arxius al llarg del temps. Això ens permet, per exemple, tornar a una versió anterior dels arxius, examinar-la i tornar després a l'última versió.

En un sistema de control de versions, tots els canvis queden registrats en un espai anomenat *repositori*. Cada vegada que vulguem registrar un canvi en els nostres arxius, el guardarem dins del repositori. Aquest albergarà en el seu interior tot l'històric de modificacions del nostre projecte. Aquest repositori pot estar sincronitzat amb un repositori *online*, allotjat al núvol, i també pot estar sincronitzat amb els repositoris de qui estigui col·laborant en el mateix projecte.

Il·lustrem-ho amb un exemple: l'Eva, en Joan i la Núria estan treballant en els estils d'una botiga virtual i estan utilitzant Git com a sistema de control de versions. A cadascun dels ordinadors hi ha la carpeta amb els arxius del projecte i, a més, cada carpeta té associat un repositori on estan registrats tots els canvis. Cada membre de l'equip té, a la seva màquina, una còpia de la mateixa carpeta i el mateix repositori que, a més, està sincronitzat amb un repositori *online* allotjat a GitHub, al qual té accés tot l'equip.

L'Eva crea un nou arxiu per als estils del carret de la compra. Atès que la creació d'un arxiu és un canvi en el projecte, decideix registrar aquest canvi i guardar-lo al repositori. A Git, quan registrem un canvi en l'històric de modificacions, a aquest canvi l'anomenem *commit*. Cada entrada a la llista de modificacions d'un projecte és un *commit*.

Aleshores l'Eva sincronitza el repositori de la seva màquina amb el de GitHub. D'aquesta manera, la modificació es propagarà per la resta d'ordinadors de l'equip quan la Núria i en Joan sincronitzin els seus repositoris locals amb el de GitHub. En el moment en el qual el facin, en les seves respectives carpetes locals, es descarregarà automàticament el nou arxiu que l'Eva havia afegit al projecte.

Posteriorment, en Joan també crea un nou arxiu per als estils del formulari de comanda i la Núria afegeix nous estils a l'arxiu que va crear l'Eva. En Joan i la Núria sincronitzen els seus repositoris amb el repositori a GitHub, i els *commits* que han realitzat arribaran a la resta de l'equip:

- La Núria rebrà el nou arxiu d'en Joan amb els estils del formulari de comanda.
- En Joan rebrà l'arxiu modificat per la Núria amb els nous estils del carret de la compra.
- L'Eva rebrà tots dos arxius quan sincronitzi el seu repositori local amb el de GitHub. Tot l'equip acaba tenint una còpia del repositori, que a més continua estant al núvol a disposició de qualsevol que s'incorpori al projecte més tard.

Ara imaginem que la Núria necessita corregir alguns errors durant un viatge amb avió. Amb Git, la majoria d'accions les realitzem a la nostra pròpia màquina, perquè treballem en un repositori local. Per tant, encara que la Núria no disposi de connexió a internet, pot registrar tots aquests canvis al seu repositori.

Si utilitzem Git, no només tindrem la possibilitat de crear l'històric de versions al nostre gust (com a Google Drive o Dropbox), sinó que, a més, podrem fer-ho sense connexió a internet, compartint el codi amb altres persones i sense por de què una altra persona de l'equip estigui modificant el mateix arxiu en el qual estiguem treballant.

1.2 QUÈ ÉS GIT I GITHUB?

Hola!

Quan ens introduïm en el món de Git, més tard o més d'hora ens trobem amb GitHub. Però, Git i GitHub és el mateix? Quines diferències hi ha entre els dos sistemes? En aquest vídeo resoldrem aquests i més dubtes. Som-hi!

Git és una eina de control de versions. Ens la instal·lem al nostre ordinador, funciona per línia d'ordres i ens serveix per aplicar control de versions a qualsevol carpeta albergada a la nostra pròpia màquina.

A més, podem utilitzar altres programes com SourceTree o GitKraken, també instal·lats al nostre ordinador, que ens proporcionen una interfície gràfica per treballar amb Git sense necessitat d'utilitzar la consola d'ordres. Tenir Git instal·lat a l'ordinador –amb interfície gràfica o sense ella– ens permet, per tant, anar guardant un històric de canvis d'un projecte, examinar versions anteriors i compartir el nostre codi amb altres persones.

Però com compartim el codi amb altres persones? Git treballa amb un model de repositoris distribuïts, en el qual teòricament no fa falta un repositori central. No obstant això, a la pràctica, tots els equips utilitzen un repositori central al qual es connecten les persones que participen en el desenvolupament del projecte. Per què? Doncs perquè facilita molt el flux de treball en equip i perquè els repositoris centrals solen tenir eines extra per revisar les modificacions en el codi.

I és aquí on entren sistemes com GitHub. **GitHub** és una plataforma de serveis al núvol que ens ofereix un allotjament per albergar el repositori central del nostre projecte. Imaginem que un equip de cinc persones està treballant en un mateix desenvolupament. Cada un dels ordinadors tindrà el seu repositori local i, a més, l'equip tindrà un repositori central allotjat a GitHub, amb el qual aniran sincronitzant les seves carpetes locals per tal que el codi pugui estar distribuït entre totes les màquines de l'equip.

Aquest repositori central que allotgem a GitHub pot ser:

- **Públic:** qualsevol persona tindrà accés al nostre codi, com a mínim a veure'l i a descarregar-lo. També podem, opcionalment, obrir més l'accés i permetre que qualsevol pugui contribuir al codi amb els seus propis *commits*.
- **Privat:** només les persones usuàries de GitHub a qui donem accés podran veure el codi. Aquest accés pot tenir més o menys privilegis: des de només veure el codi fins a contribuir directament amb *commits* al repositori.

GitHub no només ens dona, per tant, un allotjament *online* per al repositori central, sinó que a més ens proporciona un sistema de gestió i autenticació d'usuaris i usuàries que ens permetrà poder establir diferents polítiques d'accés i contribució al codi del projecte.

Ara que sabem què és Git i què és GitHub, quines són les diferències entre aquests dos sistemes?

- Git és una eina que t'instal·les a l'ordinador, mentre que GitHub és una eina al núvol a la qual pots connectar els teus repositoris Git locals.
- Git és l'eina que et permet crear branques i commits, i mantenir un repositori local. GitHub és l'eina que et permet tenir un repositori central i visualitzar tot el que l'equip va afegint al codi.
- Així com Git està orientat al treball individual de cada membre de l'equip, GitHub concentra la gestió de la part col·laborativa d'un projecte.
- Per utilitzar Git, no necessites tenir un compte enlloc. Per utilitzar GitHub has d'haver-te registrat prèviament a la plataforma.

En la majoria dels casos, quan apliques el control de versions a un projecte, estaràs usant tant Git com alguna plataforma que ofereixi repositoris *online* (per exemple, GitHub). Potser estàs pensant que a vegades treballes en projectes en els quals no hi ha més persones col·laborant. Doncs bé, en aquests casos, GitHub t'ofereix un lloc per tenir una còpia centralitzada del teu repositori local i, si a més vols que el teu codi estigui disponible per a tothom, tindràs a GitHub una plataforma perfecta de distribució.

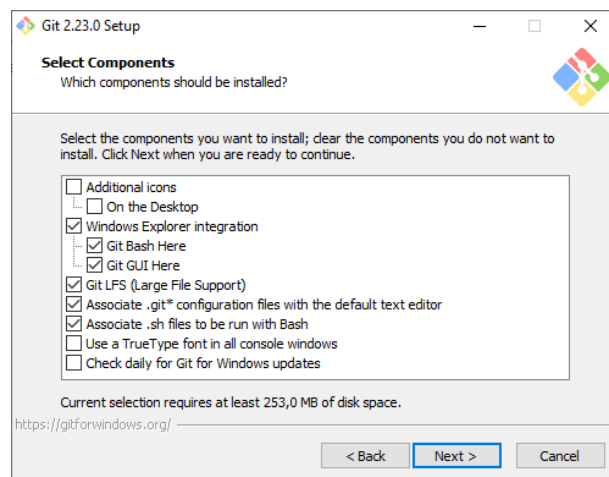
Seguim!

1.3 INSTAL·LACIÓ

Instal·lació a Windows

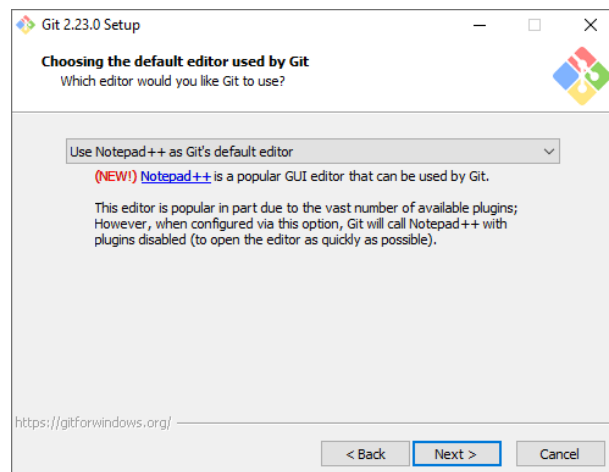
Per instal·lar Git, primer ens descarregarem l'instal·lador des de la pàgina web <<https://git-scm.com>> i l'executarem. L'assistent d'instal·lació per a Windows té diversos passos, de tots ells ens detindrem en els més importants.

Pas 1: "Select Components"



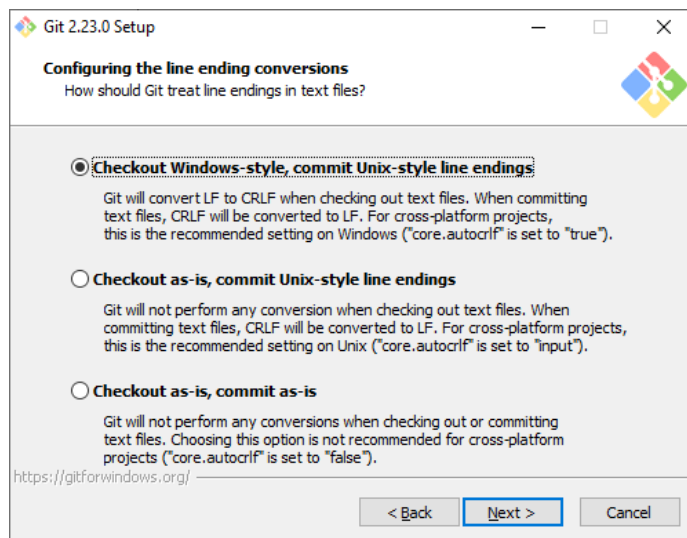
El més recomanable per llançar ordres de Git és utilitzar Git Bash, que és una terminal que s'instal·la amb Git. En aquest pas de la instal·lació, és recomanable confirmar que està marcada l'opció *Git Bash Here*. Això ens afegirà una opció al menú contextual de Windows per al botó dret, de manera que, quan tinguem una carpeta a l'explorador, podrem fer-hi clic amb el botó dret i obrir Git Bash ja amb la ruta corresponent, sense haver de navegar mitjançant ordres.

Pas 2: "Choosing the default editor used by Git"



En alguns casos, Git necessitarà utilitzar un editor de text. El cas més típic és quan vulguem realitzar un *commit* i hàgim d'escriure el missatge associat. En aquest cas, Git obrirà l'editor de text que li hàgim configurat per defecte. L'opció més còmoda és [Notepad++](#) (que hauràs d'instal·lar prèviament, si no ho tens ja instal·lat). Git l'obrirà sense *plugins*, de manera que s'iniciarà molt ràpid i ens permetrà escriure el missatge del *commit* de manera àgil.

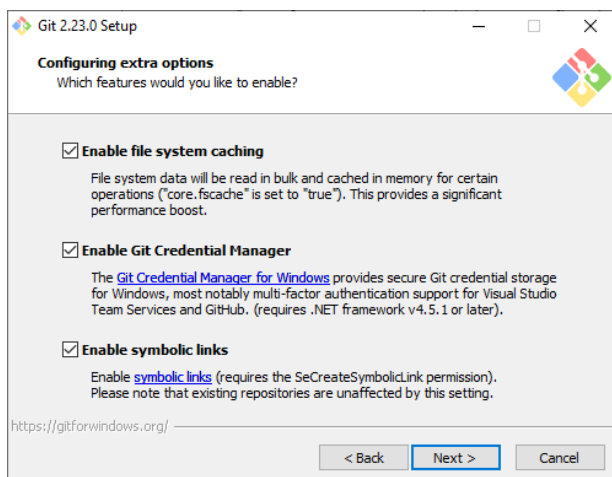
Pas 3: "Configuring the line ending conversions"



En els sistemes de Windows, un arxiu de text pla té codificats els seus salts de línia amb els codis CRLF (*Carriage Return + Line Feed*). En el cas d'un sistema Unix (com un Linux o un Mac OS), els salts de línia es codifiquen amb el codi LF (*Line Feed*).

Quan estan treballant en un mateix projecte diverses màquines amb diferents sistemes operatius, aquestes diferències entre els codis de salt de línia solen ser sempre font de conflictes a Git. En marcar aquesta opció, li estem dient a l'eina que quan els arxius estiguin al nostre directori de treball, els codis de final de línia siguin els de Windows (CRLF), però, en enviar-los mitjançant un *commit* al repositori, els codis de final de línia siguin els d'Unix (LF), de manera que no sorgeixin problemes entre màquines amb sistemes operatius diferents.

Pas 4: "Configuring extra options"



En aquest pas, és important comprovar que hi ha marcada l'opció “Enable Git Credential Manager”. Mitjançant aquesta opció, a l'hora de connectar-nos a un repositori remot que ens demani credencials, només haurem d'introduir-les una vegada i ja es quedaran emmagatzemades en el gestor de credencials de Windows.

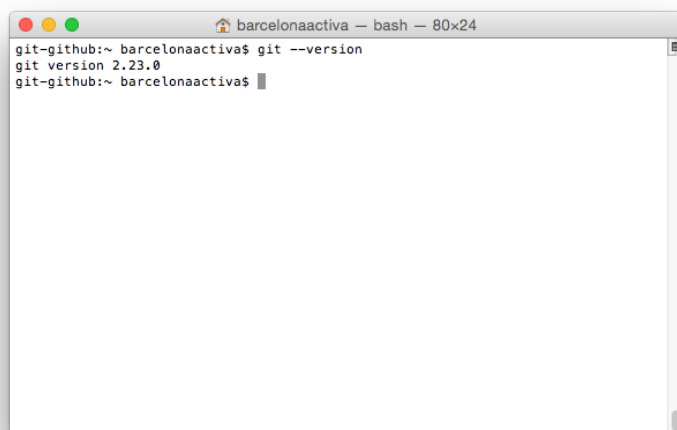
Instal·lació a Mac OS

El mètode més ràpid i recomanable per instal·lar Git en un Mac és utilitzant el gestor de paquets [Homebrew](#). L'únic que cal fer per instal·lar el programa és llançar des d'una terminal la següent ordre:

```
brew install git
```

En tots dos casos, després podrem comprovar si s'ha instal·lat, llançant la següent ordre des de la terminal:

```
git --version
```

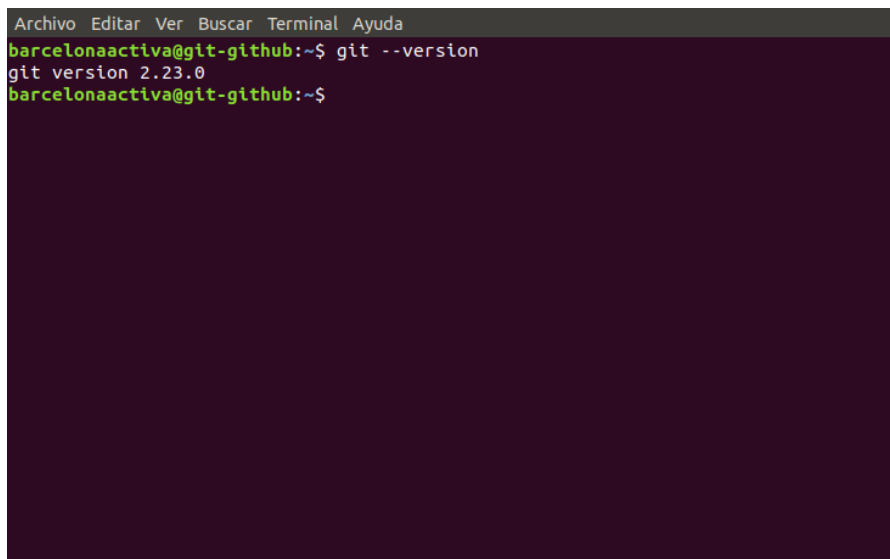


Instal·lació a Linux

Per instal·lar Git a Linux, hem d'utilitzar el gestor de paquets de la distribució en la qual estiguem. A la web <https://git-scm.com/download/linux> hi ha recopilades les ordres que hauràs de llançar, depenent de la teva distribució. Per exemple, si tens Ubuntu o Debian, pots instal·lar-ho amb la següent ordre:

```
apt-get install git
```


Després, pots comprovar si s'ha instal·lat, llançant el següent comando des de la terminal:
git --version



```
Archivo Editar Ver Buscar Terminal Ayuda
barcelonaactiva@git-github:~$ git --version
git version 2.23.0
barcelonaactiva@git-github:~$
```

1.4 CONFIGURACIÓ

Una vegada instal·lat Git al nostre ordinador, podem procedir a configurar-lo. Les opcions de configuració de l'eina s'emmagatzemen en forma de variables, que tindran un nom i un valor. L'ordre per treballar-hi és *git config*.

Per exemple, per dir-li a Git com ens diem, llançarem l'ordre:

git config user.name "El meu nom"

On el nom de l'opció de configuració és *user.name* i el valor és *"El meu nom"*.

Les opcions de configuració de Git poden existir simultàniament en tres àmbits diferents:

- **Sistema:** l'àmbit d'una variable de sistema és tot el sistema operatiu. Per aquesta mateixa raó, per establir, modificar o esborrar una variable de configuració a Linux o a Mac OS, haurem de fer-ho amb permisos de superusuari.
- **Global:** l'àmbit d'una variable global és el nostre compte d'usuari en el sistema operatiu. Per tant, una mateixa opció de configuració global pot tenir diversos valors diferents, depenent de a quin compte d'usuari estigui associat. Si una opció de configuració està

establerta en l'àmbit del sistema i també en l'àmbit global, sempre guanyarà l'àmbit global (si és que estem usant el compte d'usuari associat).

- **Local:** l'àmbit d'una variable local és una carpeta que està essent versionada amb Git, generalment la carpeta d'un projecte concret. Una opció de configuració local pot tenir valors diferents en diferents projectes de la nostra màquina i sempre guanyarà enfront dels valors que tingui la mateixa opció de configuració en altres àmbits més alts (global o sistema).

Seguint l'exemple anterior, si volem establir el nostre nom per a tot el sistema operatiu, escriurem:

```
git config --system user.name "El meu nom en el sistema"
```

Si volem establir el nostre nom per al compte d'usuari del sistema operatiu, escriurem:

```
git config --global user.name "El meu nom en el meu compte d'usuari"
```

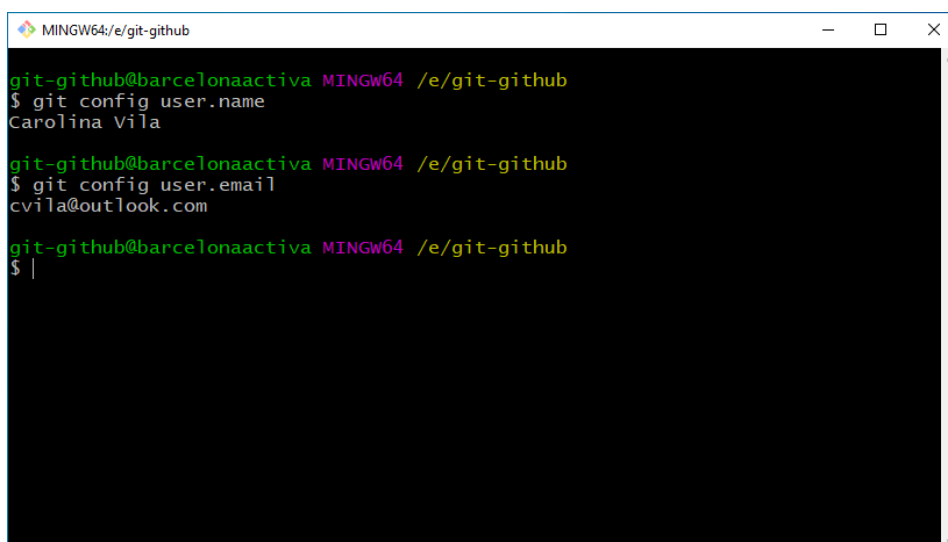
I, si volem establir-lo per a un projecte concret, escriurem, des d'una carpeta versionada amb Git:

```
git config --local user.name "El meu nom en aquest projecte"
```

Si estem dins d'un projecte, el paràmetre `--local` és opcional i, si estem fora d'un projecte, el paràmetre `--global` és opcional.

Per consultar el valor d'una opció de configuració, només hem de llançar l'ordre:

```
git config [--system/--global/--local] user.name
```



```
MINGW64:/e/git-github
git-github@barcelonaactiva MINGW64 /e/git-github
$ git config user.name
Carolina Vila

git-github@barcelonaactiva MINGW64 /e/git-github
$ git config user.email
cvila@outlook.com

git-github@barcelonaactiva MINGW64 /e/git-github
$ |
```

En la captura anterior no s'usa cap àmbit: pel fet d'estar dins de la carpeta d'un projecte, Git buscarà el valor de la variable local. Si no existeix, buscarà el de la variable global i, si no existeix, buscarà el de la variable de sistema.

Si volem consultar totes les opcions de configuració que hi ha establertes per a un àmbit, farem:

```
git config --list [--system / --global / --local]
```

Quines opcions són obligatòries?

Quan creem un *commit*, Git l'emmagatzema amb unes dades associades, entre elles qui l'ha realitzat. Per aquesta raó, no podrem fer un *commit*, si prèviament no hem dit a Git com ens diem i quina és la nostra adreça de correu electrònic.

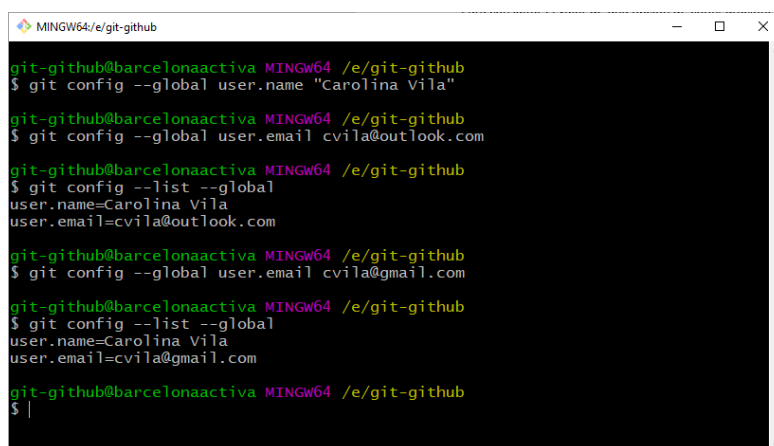
Les opcions de configuració per establir aquests dos valors són *user.name* i *user.email*. Com ja hem vist, podem establir aquests valors en qualsevol dels tres àmbits. El més habitual és establir-los en l'àmbit global, així, si una altra persona utilitza el mateix ordinador amb un altre compte d'usuari del sistema operatiu, pot establir els seus propis valors.

```
git config --global user.name "Carolina Vila"
```

```
git config --global user.email cvila@outlook.com
```

Si el valor té un espai al mig, hem d'utilitzar les cometes, però, si no, no fan falta. Si en algun moment volem modificar el valor d'una opció, n'hi ha prou amb llançar la mateixa ordre amb un altre valor, i la nova sobreescriurà l'antiga:

```
git config --global user.email cvila@gmail.com
```



```
MINGW64/e/git-github
git-github@barcelonaactiva MINGW64 /e/git-github
$ git config --global user.name "Carolina Vila"
git-github@barcelonaactiva MINGW64 /e/git-github
$ git config --global user.email cvila@outlook.com
git-github@barcelonaactiva MINGW64 /e/git-github
$ git config --list --global
user.name=Carolina Vila
user.email=cvila@outlook.com
git-github@barcelonaactiva MINGW64 /e/git-github
$ git config --global user.email cvila@gmail.com
git-github@barcelonaactiva MINGW64 /e/git-github
$ git config --list --global
user.name=Carolina Vila
user.email=cvila@gmail.com
git-github@barcelonaactiva MINGW64 /e/git-github
$ |
```

Com veus, la configuració de Git és molt fàcil, el més important és no perdre de vista en quin àmbit hi ha cada variable.

1.5 IDEES CLAU: INTRODUCCIÓ I INSTAL·LACIÓ

Hola!

Aquest primer mòdul ha servit per familiaritzar-nos amb Git i GitHub, dos programes de control de versions per treballar arxius en conjunt.

- **Git:** és una eina de control de versions. Ens l'instal·lem al nostre ordinador, funciona per línia d'ordres i ens serveix per aplicar control de versions a qualsevol carpeta albergada en el nostre dispositiu.
- **GitHub:** és una plataforma de serveis al núvol que ens ofereix un allotjament per albergar el repositori central del nostre projecte. Imaginem que un equip de cinc persones està treballant en un mateix desenvolupament. Cadascun dels ordinadors tindrà el seu repositori local i, a més, l'equip tindrà un repositori central allotjat a GitHub, amb el qual aniran sincronitzant les seves carpetes locals perquè el codi pugui estar distribuït entre totes les màquines de l'equip. Aquest pot ser públic o privat.

Per instal·lar Git, ens descarreguem l'instal·lador des de la pàgina web: <<https://git-scm.com>> i l'executem. Hem de seguir els passos que ens indica.

Una vegada instal·lat el programa, caldrà configurar-lo. Les opcions de configuració de Git poden existir simultàniament en tres àmbits diferents:

- **Sistema:** l'àmbit d'una variable de sistema és tot el sistema operatiu.
- **Global:** l'àmbit d'una variable global és el nostre compte d'usuari en el sistema operatiu. Per tant, una mateixa opció de configuració global pot tenir diversos valors diferents, depenent de a quin compte d'usuari estigui associat.
- **Local:** l'àmbit d'una variable local és una carpeta que està essent versionada amb Git; generalment la carpeta d'un projecte concret.

Seguim en el mòdul 2!

2 CONEIXENT GIT

Git és conegut per la seva particular manera de gestionar les branques i per afegir una zona anomenada *stage* en el procés de creació de *commits*. Quan parlem de ramificacions, significa que hem pres la branca principal de desenvolupament (*master*) i a partir d'aquí continuem treballant sense seguir la branca principal de desenvolupament.

Aquestes dues característiques poden imposar una mica de respecte en acostar-nos per primera vegada a l'eina, així que, en aquest mòdul, veurem com funcionen, per perdre'ls la por.

En un projecte versionat amb Git, estarem manejant diversos repositoris distribuïts per diversos ordinadors. Veurem com s'estructuren aquests repositoris i com interactuen entre ells.

Git ens ofereix l'avantatge de crear l'històric de canvis totalment al nostre gust, i ho fa donant-nos la possibilitat de decidir selectivament quins canvis s'afegeixen a cada commit. Per a això, l'eina fa ús d'una zona anomenada *stage*, que ens servirà per anar preparant un commit abans de registrar-lo. Aprendre a interactuar amb l'*stage* per mitjà d'ordres com *git add* o *git status*.

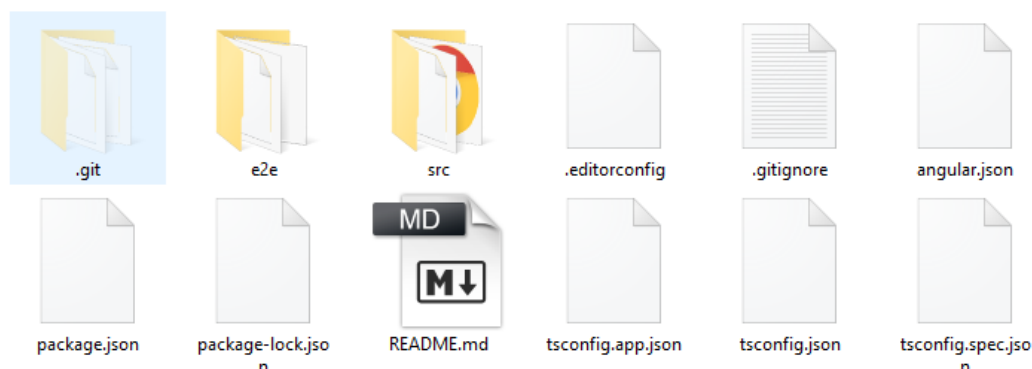
Una altra de les característiques més potents de Git és el maneig de branques. Veurem com mantenir diferents itineraris de codi gràcies a l'ús de branques locals, que ens permetran barrejar diferents versions del codi sense por al fet que es perdi el nostre treball.

Comencem?

2.1 DISTRIBUCIÓ DELS REPOSITORIS GIT

Git és conegut per ser un sistema de control de versions distribuït. Què significa això exactament?

Quan tenim un projecte versionat amb Git, anem registrant l'històric de canvis en un espai anomenat **repositori**. Aquest repositori està emmagatzemat dins de la subcarpeta *.git* del nostre projecte. És a dir, el nostre repositori està allotjat al nostre ordinador, al costat dels arxius que estan essent versionats. Diem, per tant, que tenim un repositori **local**.



Si estem treballant en un equip, cadascuna de les persones de l'equip també tindrà un repositori al seu ordinador, és a dir, un repositori local.

A més, el més habitual és que hi hagi un repositori **central**, que és un repositori que està allotjat en un servidor. Aquest servidor pot estar a la nostra xarxa o al núvol. És el cas de plataformes com **GitHub** les que ens ofereixen un servidor per allotjar els nostres repositoris centrals.

Així que, si hi ha un equip de cinc persones treballant en un projecte, el codi d'aquest projecte estarà **distribuït** en sis repositoris: cinc repositoris locals –un per persona de l'equip– i un repositori central.

Aquesta estructura té alguns avantatges:

- Facilita que totes les persones de l'equip puguin connectar-se fàcilment. Si bé es pot interactuar amb un altre repositori local directament, la infraestructura de xarxa seria molt més complexa i hauríem d'estar sincronitzant amb tants repositoris com ordinadors hi hagués a l'equip de treball.
- Aporta la seguretat de tenir diverses còpies del codi distribuïdes per diverses màquines.

En el nostre treball quotidià amb Git, realitzarem la majoria d'accions contra el repositori local. Afegir canvis, crear *commits*, crear i fusionar branques, revisar codis antics, canviar opcions de configuració... són accions bàsiques que farem dins de la nostra màquina. Per aquesta raó, podem treballar amb Git encara que no tinguem connexió a internet: només haurem de connectar-nos al repositori central, quan el vulguem sincronitzar amb el nostre.

En el nostre repositori local hi tindrem les branques principals, que compartirem amb la resta de l'equip a través del repositori central. També anirem creant branques locals per anar fent avançar el projecte, desenvolupant noves característiques o arreglant *bugs*, però aquestes branques només existiran en el nostre repositori. Quan estiguin llestes, les abocarem en alguna de les branques comunes.

És així com, a la nostra màquina i sense necessitat d'estar connectats a la xarxa, podem anar registrant nous *commits*, que quedaran emmagatzemats en el nostre repositori local i que compartirem amb l'equip per mitjà d'un repositori *online*.

2.2 CREANT I PARTICIPANT EN UN PROJECTE JA EXISTENT

Hola, benvinguts i benvingudes! En aquest vídeo veurem com començar a treballar amb un repositori, tant si l'hem de començar nosaltres com si ja existeix i ens hi hem d'unir.

Vegem primer com començar un repositori des de zero. Imagina que estàs en un equip de desenvolupament que està creant una web per visualitzar les prediccions del temps. L'equip

pren la decisió de començar a fer una altra versió del projecte amb Git i a tu se t'encarrega la tasca de crear el repositori.

Com que Git treballa amb repositoris distribuïts, el lloc on crear el primer repositori és en un dels ordinadors de l'equip, el teu en aquest cas. Obre la consola d'ordres i entra a la carpeta del projecte, on ja hi ha els arxius de la web, encara sense versionar. Per iniciar un repositori, has de llançar l'ordre *git init*.

Quan llancis l'ordre, Git crearà a la carpeta del teu projecte una subcarpeta denominada *.git*. Si treballes amb Windows, l'hauràs de tenir configurat perquè et mostri els arxius ocults, sinó no la veuràs. Dins d'aquesta subcarpeta, Git emmagatzemarà tota la informació del repositori.

A més de la subcarpeta, Git t'ha creat una branca, anomenada *master*. Si estàs usant el terminal Git Bash, veuràs que al costat del *prompt* t'indica sempre en quina branca estàs. En aquest cas, ja estàs a la branca *master* recentment creada.

A partir d'aquest moment, ja pots començar a registrar els teus canvis afegint *commits* al repositori.

Ara veurem com unir-nos a un repositori ja existent. Imagina que t'incorpores a un equip que ja porta un temps desenvolupant i versionant la web amb Git. Totes les persones que formen part de l'equip tenen un repositori local a les seves màquines, però també hi ha un repositori central allotjat a GitHub.

El que hauràs de fer per tenir una còpia del repositori al teu ordinador és **clonar-lo**. Per fer-ho, ves a una carpeta on vulguis allotjar el projecte i llança la següent ordre: *git clone <URL-REPOSITORIO>*.

I quina URL has de posar a l'ordre? L'URL del repositori allotjat a GitHub. Per obtenir aquesta URL, obre la pàgina del repositori a GitHub i copia-la de la barra de direccions del navegador o del botó "Clone or download".

Quan llancis l'ordre, crearà una subcarpeta amb el nom del repositori i, a dins, hi veuràs que estan tots els arxius del projecte i també la subcarpeta *.git*.

Com veus, tant si has d'iniciar tu el repositori com si t'has d'unir a un de ja creat, amb pocs passos el tindràs funcionant.

2.3 COMMITS I MISSATGES

Hola! En aquest vídeo veurem com anar registrant els nostres canvis en forma de *commits*.

Git ens ajuda a mantenir un registre cronològic dels canvis que va experimentant el nostre projecte. Però, lluny d'anar guardant canvis automàticament, Git ens proporciona la possibilitat de decidir quins canvis quedaran registrats a cada punt de l'històric. Què significa això? Que tenim a les nostres mans una eina ideal per escriure un històric al nostre

gust: una llista de canvis que doni tota la informació necessària de quan s'ha fet cada cosa. Cada entrada a la llista de canvis serà un *commit*.

Imagina que tens un desenvolupament versionat amb Git i introdueixes una sèrie de canvis a l'aplicació. Per exemple, els canvis consisteixen en:

- Modificar l'arxiu *style.css*.
- Modificar l'arxiu *functions.php*.
- Crear un nou arxiu anomenat *style-header.css*.

En aquest punt, decideixes que vols registrar un nou canvi en l'històric, és a dir, vols fer un *commit*. En el projecte, tens tres canvis, però només vols registrar els canvis en els estils i deixar el canvi en *functions.php* per a un altre *commit* posterior.

Per poder incloure a cada *commit* els canvis que prefereixis, Git ens ofereix una zona virtual anomenada *stage*. L'*stage* és la zona on anirem preparant tot el que anirà inclòs en el pròxim *commit*. De manera que si introduïm a l'*stage* únicament els canvis dels arxius *style.css* i *style-header.css*, aquests seran els que s'inclouran en el pròxim *commit*.

Sempre que vulguis veure quins canvis hi ha pendents d'afegir a l'*stage* o quins canvis has afegit ja, pots llançar l'ordre *git estatus*. Aquesta ordre et marca en vermell els canvis que hi ha pendents d'afegir.

Per afegir arxius a l'*stage*, utilitza l'ordre *git add*, seguit de tot el que vulguis afegir.

Si ara llances *git estatus*, veuràs que et marca en vermell el canvi que encara queda per afegir i, en verd, els canvis que has afegit a l'*stage*. Són aquests canvis en verd els que aniran inclosos al *commit*.

Per fer un *commit*, has de llançar l'ordre *git commit*. Aquesta ordre t'obrirà l'editor de text que tinguis configurat per defecte. Això és perquè un *commit* sempre ha de portar un missatge associat. És important que triïs un missatge descriptiu i evitis textos genèrics com "Canvis" o "Correccions", que no proporcionaran cap informació, quan algú estigui navegant per l'històric i hi hagi centenars de *commits*.

Quan escriguis el missatge, guarda i tanca l'editor. En aquell moment ja s'haurà generat el *commit*. Si ara llances *git estatus*, veuràs que ja no hi ha res de color verd, és a dir, l'*stage* està buit. Has registrat els canvis de dos arxius en un *commit*.

Ara imagina que vols crear un altre *commit* amb el canvi que tens pendent. Com sempre que vulguis fer un *commit*, primer has d'afegir a l'*stage* els canvis que vulguis que hi estiguin inclosos.

Per afegir l'únic arxiu amb canvis a l'*stage*, utilitza l'ordre *git add*. En aquest cas, pots utilitzar *git add*., ja que només tens aquest canvi.

Si ara llances *git estatus*, veuràs que ja no et queda cap canvi en vermell pendent d'afegir a l'*stage*, i que el teu canvi està de color verd, esperant a ser inclòs en el pròxim *commit*.

Per crear el *commit*, torna a llançar l'ordre *git commit*, però aquesta vegada afegeix el missatge en la mateixa ordre. Amb el modificador -m pots incloure, entre cometes, el missatge associat al *commit*, i no necessitaràs passar per l'editor de text.

Per veure els *commits* que has creat, llança l'ordre *git log*. Aquesta ordre et mostra l'històric de canvis. El *log* et mostra, per a cada *commit*:

- Un identificador únic.
- Qui l'ha creat.
- Quan s'ha creat.
- Quin és el missatge associat.

Quan l'històric sigui més llarg, veure el *log* en aquest format no serà gens pràctic. Si a l'ordre *git log* li afegeixes el modificador --oneline, veuràs els *commits* en format compacte. Serà molt més útil per navegar per un històric de setmanes o mesos de durada.

2.4 UTILITZANT BRANQUES

Benvinguts i benvingudes! En aquest vídeo veurem com utilitzar branques amb Git.

Sabem que Git ens facilita la tasca de treballar en equip, però com? Una de les característiques que ens ajudaran més en aquest sentit són les branques.

Una branca ens permetrà realitzar itineraris en l'històric de canvis que són independents entre si. Així podrem desenvolupar característiques del nostre projecte en paral·lel amb el codi principal o el codi que estiguin desenvolupant altres persones de l'equip.

Però il·lustrem-ho amb un exemple: estàs a la branca *master* i vols començar a desenvolupar una nova característica a la teva aplicació. Per fer-ho, crea una branca anomenada *bootstrap* amb l'ordre *git checkout -b bootstrap*. Aquesta ordre crea la branca i salta cap a ella. A partir d'ara, quan vagis creant *commits* anirà avançant la branca *bootstrap*.

Els canvis a realitzar consisteixen en la càrrega del *framework bootstrap* a l'HTML i de la tipografia *Montserrat* de Google Fonts. Per fer un *commit*, recorda afegir primer tots els canvis a l'*stage* amb *git add* i després realitzar el *commit* amb *git commit*, afegint un missatge que descrigui els canvis que hi ha.

Ara has de realitzar un canvi més: canvies el CSS perquè el teu web utilitzi la tipografia *Montserrat* que acabes de canviar. Per registrar aquest canvi en un nou *commit*, primer l'afegeixes a l'*stage* amb *git add* i després el registres amb *git commit*. Si ara mires el *log*, veuràs que la branca *bootstrap* està dos *commits* per davant de la branca *master*,

Imagina que en aquest moment t'arriba un avís d'un *bug* que cal corregir amb urgència. L'avantatge d'haver obert una branca és que no has d'aplicar la correcció sobre un codi que encara no està acabat, pots tornar a la branca *master* i aplicar-hi la correcció.

Salta a la branca *master* amb `git checkout master`, i des d'allà crea una nova branca que es digui *fix-ruta-material*. Fes la correcció i, quan la tinguis, afegeix-la a l'*stage* i crea-li un *commit*.

Si mires el *log*, veuràs que ara tens una divergència, a partir de la branca *master*, amb dues bifurcacions: *bootstrap* i *fix-ruta-material*. Com que ja tens llest el *fix*, és el moment d'abocar-lo a la branca principal. Per això, hauràs d'abocar la branca *fix-ruta-material* a la branca *master*, per mitjà de l'ordre `git merge`. Aquesta ordre farà que la branca *master* tingui els *commits* que tenia la branca *fix-ruta-material*. Una vegada fet això, pots esborrar la branca auxiliar mitjançant l'ordre `git branch -d fix-ruta-material`. No et preocupis: en esborrar la branca, no s'esborrarà el treball que has fet, aquests *commits* ja formen part de la branca *master*. Pots comprovar-ho mitjançant `git log`.

Ara pots reprendre la branca *bootstrap*, per continuar la feina des d'on la vas deixar, i, quan l'acabis, pots fer el mateix que has fet amb l'altra branca: abocar-la a la branca principal i després esborrar-la. D'aquesta manera, la branca principal tindrà el *fix* anterior i tot el treball de *bootstrap*.

Models de *branching*

Hem utilitzat un enfocament concret a l'hora de gestionar les branques, però hi ha moltes maneres de fer-ho. Als diferents enfocaments sobre quina política de branques seguir se'ls anomena models de *branching*.

La majoria de models de *branching* que existeixen es basen en l'existència de dues branques principals, que no finalitzen mai mentre el projecte estigui en desenvolupament. Una branca de producció, que se sol dir *master*, i una branca d'integració, que se sol dir *develop*.

La branca *master* només avança quan s'allibera una nova versió del projecte a producció. Per avançar la branca *develop*, es van creant branques auxiliars de desenvolupament cada vegada que volem avançar amb una nova característica o arreglar un *bug*. Aquestes branques es bolquen a *develop* una vegada finalitzades i després s'esborren. Les úniques branques que es pugen al repositori central i es comparteixen amb l'equip són *master* i *develop*.

Aquest model bàsic d'ús de branques ens garantirà l'aparició de molts menys conflictes a l'hora de treballar en equip. Segueix!

2.5 IDEES CLAU: CONEIXENT GIT

Git és un sistema distribuït en el qual, quan hi treballem, el codi del projecte estarà repartit per diverses màquines. És important tenir clar que quan fem un *commit* l'estem registrant en el nostre repositori local. La majoria de les ordres de Git treballen contra el repositori local.

Quan necessitem sincronitzar el nostre codi amb el de la resta de l'equip, llavors interactuarem amb el repositori central, enviant-li o demanant-li nous canvis.

A Git no es guarden els canvis automàticament. Tu decideixes quan es registra un canvi i què conté. Per fer-ho, sempre has de preparar el *commit* utilitzant l'*stage*. L'*stage* és la zona on aniràs afegint els canvis que seran inclosos en el pròxim *commit*.

L'ordre *git estatus* et dona informació valuosa sobre quins canvis estan pendents d'afegir-se a l'*stage* i quins estan preparats per ser inclosos al *commit*.

Cada vegada que hakis de programar una nova característica o corregir un error, obriràs una nova branca local a partir d'una de les branques principals. Quan hakis acabat la tasca, serà el moment d'abocar la branca local a la branca principal. Recorda que les branques principals són les úniques que es comparteixen amb les altres persones de l'equip.

Una vegada que hakis abocat una branca en una altra, pots esborrar la primera sense por. No es perdran els *commits* que hakis registrat, perquè ja hauran passat a formar part de la branca principal.

L'ordre *git log* et mostrarà l'històric de *commits*. Aquí pots veure els modificadors que admet, per poder personalitzar el seu aspecte i així poder rebre la informació que necessites: <https://git-scm.com/docs/git-log>.

3 TREBALLANT AMB REPOSITORIS GITHUB

GitHub és una de les plataformes més conegudes per allotjar repositoris Git. En aquest mòdul, aprendrem a crear un compte d'usuari de GitHub i començarem a treballar amb el programa.

Veurem com, mitjançant una sèrie d'ordres, podem sincronitzar el nostre repositori local amb el repositori remot. Això ens permetrà:

- Pujar la feina que tinguem en local i el que vulguem compartir amb les altres persones usuàries del programa.
- Baixar-nos al nostre dispositiu la feina de la resta de l'equip que ja estigui pujada al repositori remot.

Repassem quins passos cal fer per començar a treballar amb un repositori remot. A l'hora de crear un repositori, decidirem si és públic o privat i podrem autoritzar altres persones a contribuir al codi.

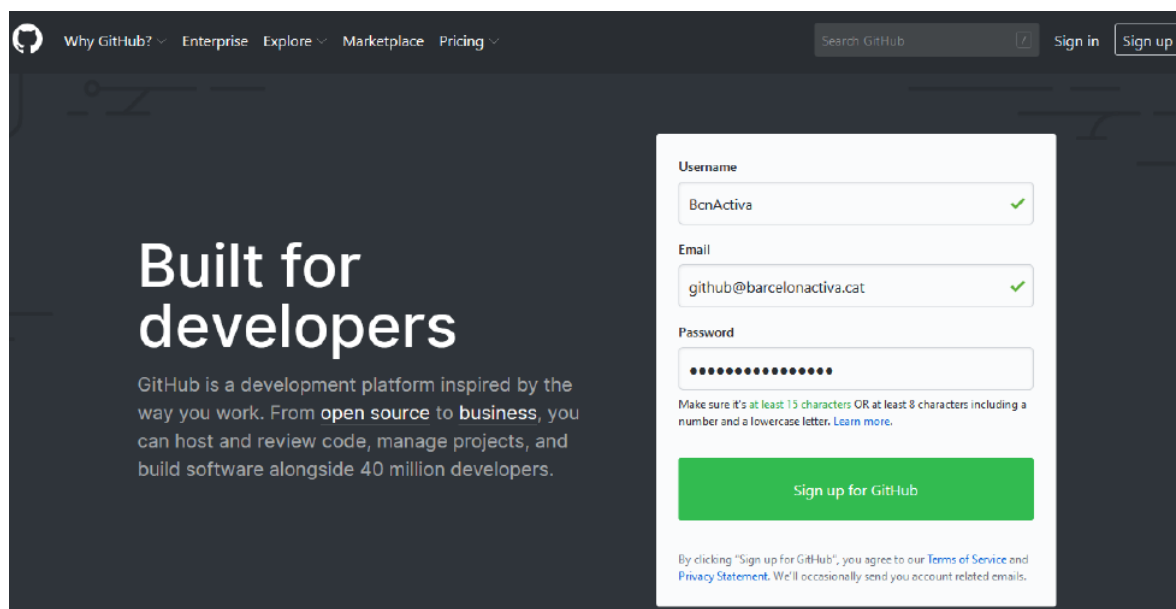
Una vegada ja hi hagi un repositori buit, veurem com pujar el codi per primera vegada des del nostre dispositiu i com anar actualitzant el remot amb els canvis que fem.

No obstant això, també pot ser que ens incorporem a un equip que ja faci temps que treballa amb un repositori a GitHub. En aquest cas, veurem com podem connectar-nos al repositori remot, descarregar tots els *commits* que ja s'hagin pujat i començar a treballar.

Finalment, repassem quines maneres de col·laborar amb altres persones ens permet GitHub. Passarem per les invitacions als nostres repositoris, explicarem què és un *fork* i veurem un dels sistemes més estesos de col·laboració: les *Pull Requests*.

3.1 INTRODUCCIÓ A GITHUB

GitHub ens ofereix un servei d'allotjament per als nostres repositoris Git. En aquest article, veurem quins són els primers passos per obrir un compte a GitHub i que comenci a funcionar amb la plataforma.



Why GitHub? Enterprise Explore Marketplace Pricing

Search GitHub

Sign in Sign up

Built for developers

GitHub is a development platform inspired by the way you work. From **open source** to **business**, you can host and review code, manage projects, and build software alongside 40 million developers.

Username

BcnActiva ✓

Email

github@barcelonactiva.cat ✓

Password

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)


Sign up for GitHub

By clicking "Sign up for GitHub", you agree to our [Terms of Service](#) and [Privacy Statement](#). We'll occasionally send you account related emails.

La creació d'un compte és un procés ràpid. Només cal disposar d'una adreça de correu electrònic. Quan entrem a la pàgina de GitHub (<https://github.com>), ens apareixerà el formulari de registre, que ens demana el nom d'usuari, l'adreça de correu electrònic i una contrasenya, de les tres coses sol el nom d'usuari serà públic.

Individuals

Teams




Free
\$0 USD

The basics of GitHub for every developer

Choose Free

- ✓ Unlimited public repositories
- ✓ Unlimited private repositories
- ✓ Limited to 3 collaborators for private repositories
- ✓ Issues and bug tracking
- ✓ Project management



Pro
\$7 USD

Per month

Pro tools for developers with advanced requirements

Choose Pro

- ← Includes everything in Free
- ✓ Unlimited collaborators
- ✓ GitHub Pages
- ✓ Wikis
- ✓ Protected branches
- ✓ Code owners
- ✓ Repository insights

Quan introduïm les dades, ens arribarà un correu electrònic a l'adreça indicada, amb un enllaç per confirmar el compte. Mentrestant, podem continuar amb el procés d'alta.

Després d'un primer control per verificar que no som un *bot*, GitHub ens dona a triar entre els diferents plans que ofereix. Ofereix serveis tant gratuïts com de pagament. Si necessitem crear repositoris privats, la plataforma ens ho ofereix gratuïtament sempre que no hi hagi més de tres persones assignades a cada repositori. En els repositoris públics, no hi ha límit de col·laboradors ni col·laboradores.

Després de triar el pla, ja podrem començar a crear repositoris. Però abans veurem com omplir el nostre perfil i també repassarem algunes opcions importants de la configuració del nostre compte.

A dalt a la dreta, trobem l'avatar que ens ha generat GitHub per defecte. Si hi premem, veurem l'opció "Your profile" al menú que es desplega. A l'hora d'accedir al perfil, veurem que està bastant buit. Vegem com omplir les nostres dades.

Personal settings

- Profile
- Account
- Security
- Emails
- Notifications
- Billing
- SSH and GPG keys
- Blocked users
- Repositories
- Organizations
- Saved replies
- Applications

Developer settings

Public profile

Name

Barcelona Activa

Your name may appear around GitHub where you contribute or are mentioned. You can remove it at any time.

Public email

Select a verified email to display

You have set your email address to private. To toggle email privacy, go to [email settings](#) and uncheck "Keep my email address private."

Bio

Barcelona Activa és l'agència de desenvolupament local de l'Ajuntament de Barcelona.

You can @mention other users and organizations to link to them.

URL

<https://barcelonactiva.cat>

Company

Barcelona Activa

You can @mention your company's GitHub organization to link it.

Location

Barcelona

All of the fields on this page are optional and can be deleted at any time, and by filling them out, you're giving us consent to share this data wherever your user profile appears. Please see our [privacy statement](#) to learn more about how we use this information.

[Update profile](#)

Profile picture

Edit

Fent clic a l'avatar, accedirem a la pàgina per omplir la informació del perfil. Aquí podrem:

- Establir un nom públic en el camp "Name" (que es mostrarà a més del nom d'usuari).
- Escriure un paràgraf sobre nosaltres en el camp "Bio".
- Introduir dades com una web de referència (camp "URL"), l'empresa en la qual treballem (camp "Company") o el nostre lloc de residència (camp "Location").
- Canviar el nostre avatar o imatge de perfil (fent clic a sobre de la imatge actual).

Per veure com queda el teu perfil públicament, pots obrir una sessió d'incògnit al navegador i anar a <<https://github.com/tu-nombre-de-usuario>>.

A continuació, explicarem com configurar el compte. Si t'hi fixes, a la mateixa pàgina d'edició del perfil, hi ha un menú lateral de navegació. Si surts de l'opció "Profile" i vas a les altres, podràs anar configurant el teu compte. Vegem quines són les opcions més importants.

Secció “Account”

En aquesta secció, pots canviar el nom d'usuari o eliminar el compte. En tots dos casos et demanarà una confirmació.

Secció “Security”

En aquesta secció, pots modificar la teva contrasenya actual per una de nova i també pots activar l'autenticació en dos passos, si vols blindar una mica més l'accés al teu compte.

Secció “Emails”

En aquesta secció, pots afegir adreces de correu electrònic addicionals mitjançant el camp “Add email address”, i també pots establir si les teves adreces són visibles públicament o no, amb el camp “Keep my email addresses private”.

Secció “Notifications”

En aquesta secció, podràs configurar quines alertes vols rebre al nostre correu electrònic.

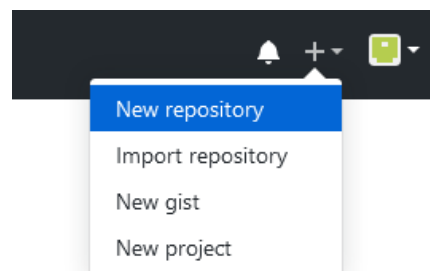
Secció “SSH and GPG keys”

Si vols que l'accés a algun repositori sigui mitjançant SSH, aquí podràs generar les claus corresponents.

Si bé pots començar a utilitzar GitHub i a crear repositoris sense repassar aquesta informació, sempre ve bé que dediquis una mica de temps a donar forma al teu perfil i a configurar el teu compte després de donar-lo d'alta.

3.2 CREACIÓ D'UN REPOSITORI

En aquest article descobrirem com crear un repositori a GitHub. El botó per crear un repositori el trobarem sempre a la part superior dreta, en el menú desplegable de la icona “+”. Si fem clic a l'opció “New repository”, ens portarà a la pàgina de creació del nou repositori.



En el formulari per crear un nou repositori, haurem d'indicar el seu nom i si serà públic o privat. El nom formarà, al costat del nostre nom d'usuari, l'URL del repositori. Opcionalment, podem escriure una descripció del que contindrà el repositori.

Create a new repository
A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner: BarcelonaActiva / Repository name: temps-app ✓

Great repository names are short and memorable. Need inspiration? How about [psychic-guacamole](#)?

Description (optional): PWA de prediccions meteorològiques

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

Add .gitignore: None | Add a license: None ⓘ

Create repository

Un repositori **públic** serà visible per a tothom, és a dir, qualsevol podrà veure tant el codi que hi ha allotjat, com els *commits* i les branques. Però que sigui públic no implica que qualsevol pugui contribuir-hi realitzant *commits*. Després, podrem decidir si l'accés és només de lectura o si donem permís a alguna persona, o a diverses, perquè puguin afegir *commits* al repositori.

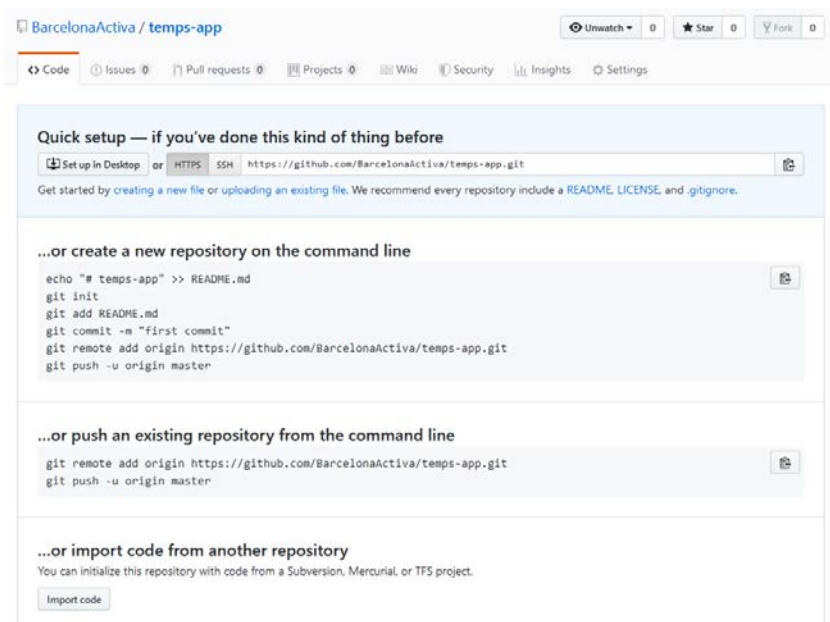
Un repositori **privat** només serà accessible per a les persones que indiquem. Ningú que no estigui a la llista de col·laboradors o col·laboradores podrà veure el repositori, ni tampoc contribuir-hi. Igual que amb el públic, podrem donar permís a les persones que ens interressi.

Quan creem un repositori, està completament buit i actua com a mer contenidor per a què després hi pugem els arxius que toquin. No obstant això, si estem creant un repositori i encara no hi ha cap arxiu al projecte, és a dir, estem creant el repositori al mateix temps que iniciem el projecte, GitHub ens ofereix la possibilitat de crear tres arxius inicials: el *README*, el *.gitignore* i l'arxiu de llicència. En el formulari de creació del repositori, podem marcar-los perquè GitHub els creï automàticament, tot i que els tres són totalment opcionals.

L'arxiu *README* és un arxiu de text que conté la descripció del projecte, i GitHub mostrarà el seu contingut automàticament a la pàgina inicial del repositori.

L'arxiu *.gitignore* és un arxiu de text que conté tots els arxius i carpetes del nostre projecte que no volem que es versionin. Tot i que això depèn del projecte i les tecnologies que utilitzem, aquí solen anar les carpetes de codi compilat i les dependències del projecte. Tot el que no estigui versionat per Git s'ignorarà i no es pujarà al repositori *online*.

Finalment, l'arxiu de llicència conté la llicència d'ús que donem al nostre projecte.



Una vegada creat el repositori, podrem accedir a la seva pàgina principal. Si ja conté arxius, la pàgina principal ens els llistarà i, si està buit, a la pàgina principal, veurem les instruccions per començar a utilitzar el repositori.

3.3 ENLLAÇANT AMB EL NOSTRE REPOSITORI GITHUB

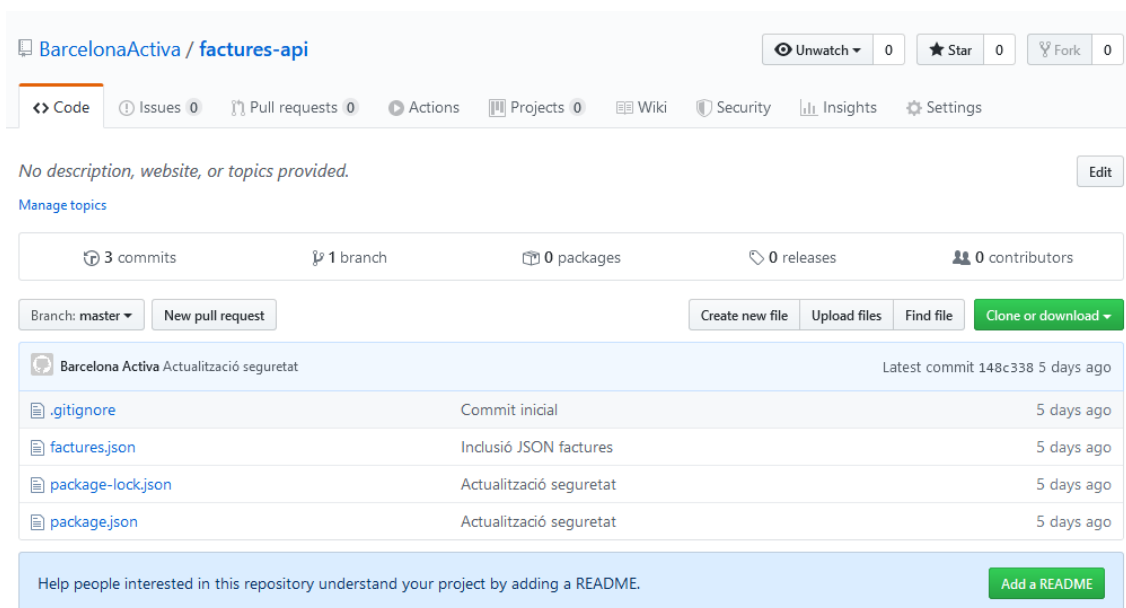
Per enllaçar el nostre dispositiu amb un repositori *online* de GitHub, haurem d'utilitzar dos procediments diferents, depenent de si ja existeixen arxius en el projecte o no: enllaçant amb un repositori de contingut o enllaçant amb un repositori buit.

Enllaçant amb un repositori de contingut

Aquest sol ser el cas en el qual ens unim a un projecte ja existent, i en el nostre dispositiu encara no hi ha els arxius del projecte. Per poder tenir els arxius, que estan en el repositori central i en els dispositius de la resta de l'equip, **clonarem** el repositori que hi ha a GitHub. Això farà quatre coses:

- Ens crearà una subcarpeta al nostre dispositiu amb el nom del repositori.
- Ens crearà un *remote* anomenat *origin* amb l'URL del repositori.
- Es descarregarà tots els arxius que hi ha al repositori central.
- Es descarregarà tot el repositori, és a dir, l'històric de *commits* i les branques.

Un *remote* és un àlies a un repositori *online*. Consta del propi àlies i de l'URL del repositori. L'utilitzarem posteriorment per referir-nos al repositori al qual volem connectar-nos des del nostre repositori local.



Per clonar el repositori, primer hem de copiar la seva URL. L'URL la podem agafar de dos llocs: de la barra d'adreces del navegador o del requadre de text que apareix quan premem en el botó "Clone or download".

Un cop tinguem l'URL al portapapers, podem obrir una terminal i llançar l'ordre següent: *git cloni url-repositorio*.

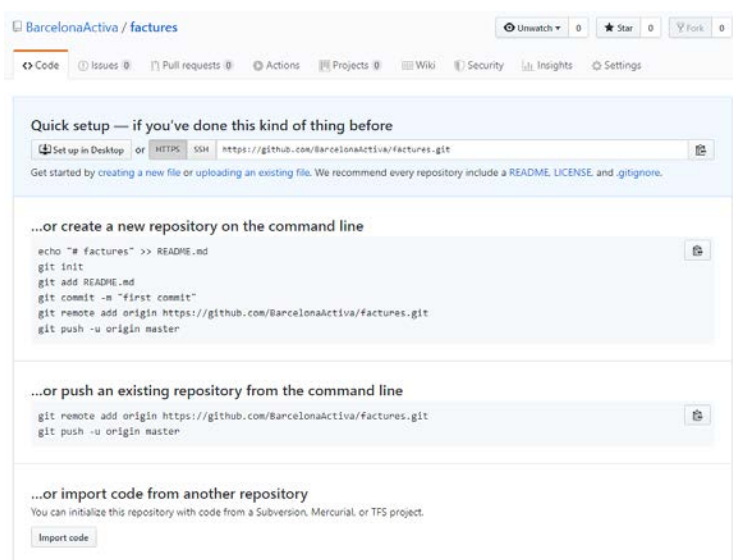
```
MINGW64/e/git-github/projects
git-github@barcelonaactiva MINGW64 /e/git-github/projects
$ git clone https://github.com/BarcelonaActiva/factures-api.git
Cloning into 'factures-api'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 12 (delta 4), reused 12 (delta 4), pack-reused 0
Unpacking objects: 100% (12/12), done.

git-github@barcelonaactiva MINGW64 /e/git-github/projects
$ |
```

Recorda que aquesta ordre et crearà una subcarpeta, per tant, has de llançar-la des de la carpeta del nivell immediatament superior a la que albergarà el projecte. Si ja tenies la subcarpeta creada, pots entrar a dins i llançar la mateixa ordre seguida d'un punt, per a què no creï la subcarpeta: *git clone url-repositorio*.

Un error molt comú és llançar l'ordre *git cloni* i després començar a llançar la resta d'ordres. Recorda que, tret que hagis optat per crear tu la subcarpeta, quan llancis *git clone* encara no estaràs a dins de la carpeta del projecte, així que el primer que has de fer és entrar i després ja pots començar a treballar amb Git.

Enllaçant amb un repositori buit



Aquest és habitualment el cas en el qual, o bé el projecte encara no ha començat i no existeix cap arxiu, o ja s'ha començat a crear el projecte en un únic dispositiu, però encara no s'ha pujat a cap repositori central. Si el projecte encara no ha començat, almenys ha d'existir la carpeta del projecte i s'ha d'haver inicialitzat el repositori local amb *git init*.

Per enllaçar un repositori local amb un repositori central buit, haurem de crear el *remote* manualment. Per fer-ho, llançarem l'ordre següent: *git remote add origin url-repositorio*.

```
MINGW64:/e/git-github/projectes/factures
git-github@barcelonaactiva MINGW64 /e/git-github/projectes/factures (master)
$ dir
angular.json karma.conf.js README.md tsconfig.json
browserslist package.json src tsconfig.spec.json
e2e package-lock.json tsconfig.app.json tslint.json

git-github@barcelonaactiva MINGW64 /e/git-github/projectes/factures (master)
$ git remote add origin https://github.com/BarcelonaActiva/factures.git

git-github@barcelonaactiva MINGW64 /e/git-github/projectes/factures (master)
$ |
```

git remote és l'ordre que utilitzarem per treballar amb *remotes* i l'acció *add* és per crear un *remote*. L'anomenarem *origin* perquè, per convenció, se sol anomenar així al repositori remot principal (n'hi pot haver més d'un), però en realitat podríem anomenar-lo com volguéssim.

Aquesta ordre ens crea un àlies, que podrem utilitzar a partir d'ara per pujar i baixar arxius. Quan vulguem fer la primera pujada d'arxius, que sol ser de la branca *master*, farem:

git push origin master

```
MINGW64:/e/git-github/projectes/factures
git-github@barcelonaactiva MINGW64 /e/git-github/projectes/factures (master)
$ git push origin master
Enumerating objects: 168, done.
Counting objects: 100% (168/168), done.
Delta compression using up to 4 threads
Compressing objects: 100% (152/152), done.
Writing objects: 100% (168/168), 125.57 KiB | 2.41 MiB/s, done.
Total 168 (delta 30), reused 0 (delta 0)
remote: Resolving deltas: 100% (30/30), done.
To https://github.com/BarcelonaActiva/factures.git
 * [new branch]      master -> master

git-github@barcelonaactiva MINGW64 /e/git-github/projectes/factures (master)
$ |
```

L'ordre *git push* es connecta a un repositori remot per pujar els arxius. El remot al qual es connecta és *origin* (l'àlies que hem creat abans) i puja la branca *master*.

Un cop llancem aquesta ordre per primera vegada, el repositori remot deixarà d'estar buit i contindrà el primer lot d'arxius del projecte.

3.4 PUSHEANT I VERIFICANT CANVIS A GITHUB

Benvinguts i benvingudes! En aquest vídeo veurem algunes de les operacions bàsiques que farem quan tinguem el nostre repositori local connectat amb un repositori *online* a GitHub.

Imaginem que tenim un repositori local amb una aplicació de prediccions meteorològiques, que està connectat amb un repositori remot allotjat a GitHub, encara buit. Per comprovar la connexió amb el repositori remot, llistarem els *remotes* que hi hagi registrats, amb l'ordre `git remote -v` (el modificador `-v` és perquè ens ensenyi l'URL del repositori).

Veurem el nom del *remote*, en aquest cas *origin*, i l'URL associada. Git ens llista dues URL diferents, una per pujar i una altra per baixar, però a la pràctica sempre utilitzarem la mateixa per a les dues coses.

Abans de pujar al repositori *online*, revisarem el nostre històric de *commits*. Si mirem el *log* amb `git log --oneline`, veurem que actualment hi ha dos *commits* a la branca *master*. Això és el que es pujarà al repositori remot.

Per pujar-ho, llancem l'ordre `git push origin master`, essent *origin* el nom del *remote* i *master* la branca que volem pujar.

Ara comprovem si s'ha pujat tot bé a GitHub.

El primer que apreciem és que si anem a la pàgina del repositori ja no apareixen les instruccions inicials, sinó els arxius del nostre projecte. Repassem la informació que ens dona aquesta pàgina, per verificar que la pujada ha anat bé.

Aquí podem veure el nombre de *commits* que té el repositori, que efectivament coincideix amb els del nostre repositori local, com hem comprovat abans. Si hi fem clic, podrem veure el *log* que ja havíem vist en el nostre repositori local. I aquí el nombre de branques, que també coincideix perquè només n'hem pujat una, *master*.

A sobre de la llista d'arxius, podem veure informació sobre l'últim *commit*: qui ho va registrar, quant temps fa que es va registrar i quin és el missatge associat. A la dreta de cada arxiu, veurem quin és l'últim *commit* en el qual l'arxiu va patir canvis. Fent clic a un arxiu, podem accedir al seu contingut. I, si cliquem al botó "History", podem veure tots els *commits* en els quals aquest arxiu ha patit canvis.

Ara generarem nous canvis en el repositori local per veure com s'actualitza posteriorment el remot.

Imagina que fem un canvi en els estils CSS i un altre en el manifest. Generarem un *commit* per a cadascun d'ells.

Si ara llancem l'ordre *git estatus*, veurem que els dos canvis estan pendents d'afegir-se a l'*stage*. Afegim el primer canvi, el dels estils CSS, mitjançant l'ordre *git add*. Una vegada hem preparat el *commit* amb els canvis d'estils, podem registrar-lo mitjançant l'ordre *git commit*.

Si ara llancem *git status*, veurem que ja només queda un canvi pendent de pujar a l'*stage*, el del manifest. El pugem amb *git add* i després podem registrar el *commit*.

Ara podem mirar l'històric de canvis per veure els dos nous *commits*. Si llancem l'ordre *git log --oneline*, veurem no sols que hi ha dos *commits* més, sinó que la branca *master* està dos *commits* per davant de la branca *origin/master*. Les branques el nom de les quals va precedit d'*origin/* són les branques del repositori remot. Per tant, aquí podem veure que el nostre repositori local està dos *commits* per davant del repositori remot. Quan pugem mitjançant *git push origin master*, el que es pujarà és aquesta diferència de *commits* entre *origin/master* i *master*.

Si, després de realitzar la pujada, comprovem el *log*, veurem que la branca *master* i *origin/master* estan a la mateixa altura. Hem sincronitzat la nostra branca local amb la branca remota.

Per comprovar-ho, anem de nou a la pàgina del repositori a GitHub. Podem veure que ara apareixen quatre *commits* en lloc de dos i que tant la carpeta *CSS* com l'arxiu *manifest.json* tenen un altre missatge diferent a l'anterior, perquè hi ha hagut *commits* nous que han modificat el seu contingut.

Com pots veure, des de la consola, pots monitorar els canvis que hi ha entre el repositori local i el remot, però la interfície de GitHub et presenta tota aquesta informació d'una manera molt més pràctica i còmoda.

3.5 ACTUALITZACIÓ DE CANVIS EN REPOSITORI LOCAL

Hola! En aquest vídeo, veurem com portar fins al nostre repositori local els nous canvis que hi hagi en el repositori remot. Som-hi!

Moltes vegades, ens connectarem al repositori *online* per pujar canvis que hi ha a la nostra màquina i que encara no estan al central, però altres vegades ens hi connectarem per al contrari: baixar-nos canvis que estiguin en remot i no tinguem encara en local. I també és habitual connectar-se per fer les dues coses alhora.

Anem a revisar el nostre repositori local. Si fem *git log --oneline*, veiem que tenim una branca *master* amb dos *commits*. Encara que vegem que la branca *origin/master* està a la mateixa altura que la branca *master*, hem de tenir en compte que aquesta informació es basa en un

caixet i no és a temps real. La branca *master* del remot pot estar més avançada, com veurem a continuació.

Si veiem el repositori a GitHub i anem a la llista de *commits*, podrem comprovar que hi ha dos *commits* més que en el nostre repositori local. Anem a veure dos mètodes per baixar-nos al nostre dispositiu tot aquest treball que no tenim en local.

git fetch + git merge

Amb el primer mètode, realitzarem la baixada en dos passos. El primer que farem serà llançar l'ordre *git fetch*. Aquesta ordre es connecta al servidor remot i comprova si hi ha canvis respecte al nostre repositori local. Aquí podem veure que la branca *master* ha canviat d'un *commit* a un altre. Aquest era l'últim que teníem en local i aquest és l'últim que hi ha en remot.

Si ara mirem el *log*, veurem que continuem tenint els nostres dos *commits*, però ja no ens apareix la branca *origin/master*. Si volem veure els *commits* de la branca *origin/master*, hem d'afegir el modificador *--all*. Ara podem veure com la branca *origin/master* té dos *commits* més. Aquests *commits* encara no estan al nostre espai de treball, però ja s'han descarregat en el nostre dispositiu, en una zona de caixet.

L'única cosa que hem de fer per portar aquests *commits* a la nostra branca *master* local és fusionar totes dues branques mitjançant l'ordre *git merge origin/master*. Mitjançant aquesta ordre, abocarem els *commits* de la branca *origin/master* a la branca *master*.

Si ara mirem el *log*, podem comprovar com ja tenim a la nostra branca tot el que hi havia a la branca *master* del repositori remot.

git pull

El mètode anterior ens serveix per a quan necessitem comprovar primer quines diferències hi ha entre el repositori remot i el local. Després, podem decidir si fusionem o no, mitjançant *git merge*.

Però la majoria de les vegades que vulguem baixar-nos treball del repositori ho voldrem fer en un sol pas. Ho podem fer amb l'ordre *git pull*. Aquesta ordre no és res més que la suma de les dues anteriors.

L'única diferència és que hem d'explicitar el nom del *remote* i de la branca que volem actualitzar. Així, per baixar-nos tot el que hi hagi de nou a la branca *master*, haurem d'escriure *git pull origin master*. *origin* és l'àlies del remot al qual ens volem connectar, i *master* la branca que volem baixar.

Si no volem haver d'escriure tots dos paràmetres cada vegada, podem vincular la nostra branca local *master* a la branca *master* del *remote origin* mitjançant l'ordre *git branch --set-upstream-to origin/master*.

Aquesta ordre mantindrà les dues branques vinculades, de manera que la pròxima vegada que vulguem descarregar-nos la branca *master* remota, només haurem d'escriure *git pull*.

3.6 CREANT NOUS FITXERS A GITHUB

Hola a tots i a totes! En aquest vídeo coneixerem el procés per crear nous fitxers des de la interfície de GitHub. Som-hi!

Si bé la manera normal d'afegir arxius a un projecte és des dels repositoris locals, GitHub ens ofereix l'opció de crear-los directament al repositori remot.

Si anem a la pàgina d'un repositori que conté un projecte, veurem la llista dels seus arxius i carpetes. Per crear un arxiu, el primer que hem de fer és navegar fins a la carpeta on vulguem situar-lo. Imagina que volem crear un nou arxiu d'estils CSS. Anem primer a la carpeta *css*, i quan en veiem el contingut, el creem.

Per fer-ho, tenim dos botons: "Create new file" i "Upload files". El primer és per crear directament el contingut de l'arxiu escrivint el codi des de la interfície. El segon és per pujar un arxiu ja existent al nostre ordinador.

"Create new file"

Si premem aquest botó, se'ns obre una pàgina amb un editor de text. El primer que hem de fer és posar un nom a l'arxiu nou, mitjançant la caixa de text que hi ha damunt de l'editor.

Per escriure el contingut a l'editor, podem ajustar primer les tabulacions: si volem que siguin *tabs* o espais, i quants seran (2, 4 o 8). Un cop tinguem el contingut del nostre arxiu llest, podem anar al formulari de *commit*, que és a sota l'editor. Un arxiu nou és un canvi en el repositori, per tant, ha d'estar registrat en un *commit*. En aquest formulari, hem d'escriure el missatge associat. Recordem que el nom d'usuari que quedarà registrat al *commit* és aquell amb el qual hàgim iniciat sessió a GitHub.

Una vegada creat el *commit*, veurem l'arxiu a la interfície de GitHub i ja el podrem baixar als nostres repositoris locals.

"Upload files"

Si l'arxiu ja existeix en el nostre dispositiu, el procés serà més ràpid: només hem de prémer aquest botó, buscar l'arxiu que volem pujar i omplir les dades per al nou *commit*. El resultat final serà el mateix que amb el mètode anterior.

Seguim!

3.7 TREBALLANT AMB FITXERS A GITHUB

Benvinguts i benvingudes. En aquest vídeo, explicarem com realitzar alguns canvis en els fitxers del nostre repositori local i sincronitzar-los amb el repositori remot a GitHub.

La primera modificació que veurem és el canvi de nom d'un arxiu. Podem modificar el nom d'un arxiu, per qualsevol mètode que ofereixi el sistema operatiu, i, si aquest arxiu ja estava en els anteriors *commits* del repositori, Git ho interpretarà així: fixem-nos en quina informació ens dona quan llancem *gitstatus*. Ens diu que s'ha esborrat un arxiu i que ha aparegut un arxiu nou. Això és perquè, d'una banda, Git veu que en els *commits* anteriors hi havia un arxiu amb aquest nom i a la carpeta local ja no existeix. I, d'altra banda, perquè ara a la carpeta local hi ha un arxiu amb aquest nom i en els anteriors *commits* no existia.

Però vegem què passa si pugem tots dos canvis a l'*stage* i tornem a llançar *gitstatus*: ara sí que veiem que la modificació consisteix en un canvi de nom del fitxer.

Després, ja podem realitzar el *commit* i registrar aquest canvi al repositori.

Ara veurem quin és el procediment per registrar l'esborrat d'un arxiu. Podem fer-ho de dues maneres.

La primera és esborrant l'arxiu de qualsevol de les maneres convencionals que el sistema operatiu permeti. Si fem *git estatus* després d'haver-ho esborrat, veurem que hi ha un canvi pendent de pujar a l'*stage*, ja que l'esborrat d'un arxiu també és un canvi en el nostre projecte.

Com amb qualsevol altre canvi, podem pujar-lo a l'*stage* amb *gitadd* i després realitzar el *commit* amb normalitat

Però Git ens ofereix una ordre per agilitzar aquest procediment. Podem esborrar l'arxiu amb *gitrm archivo*. Veiem que l'arxiu s'ha esborrat i, si llancem *gitstatus* després, comprovarem que la modificació ja està pujada a l'*stage*, per la qual cosa ens estalviem un pas.

Ara tenim dos *commits* amb modificacions. Les pujarem al repositori remot i després les verificarem des de GitHub. Amb *git log* podem veure quins *commits* es pujaran, si són els dos que hi ha a *master* i no a *origin/master*.

Una vegada aplicats els canvis, anem a la pàgina del repositori remot a GitHub. Aquí podem veure el nou nom de l'arxiu que volem canviar de nom. Si fem clic al *commit* corresponent, GitHub ens mostra en què va consistir la modificació, és a dir, el canvi de nom.

D'altra banda, també podem comprovar que l'arxiu que hem esborrat al nostre repositori local ja no està entre la llista d'arxius. Si fem clic a l'últim *commit*, Git ens mostra que la modificació ha consistit en l'eliminació de l'arxiu.

Sempre podem anar a un *commit* anterior a aquests canvis i veure com estava la llista de carpetes i arxius, i comprovar així l'antic nom de l'arxiu que ha canviat de nom i també l'existència prèvia de l'arxiu ara esborrat.

Una vegada que aquestes modificacions són al repositori central, quan una altra persona de l'equip s'hi connecti per sincronitzar el seu repositori local, en el seu dispositiu es produiran tant el canvi de nom d'un arxiu com la desaparició de l'altre.

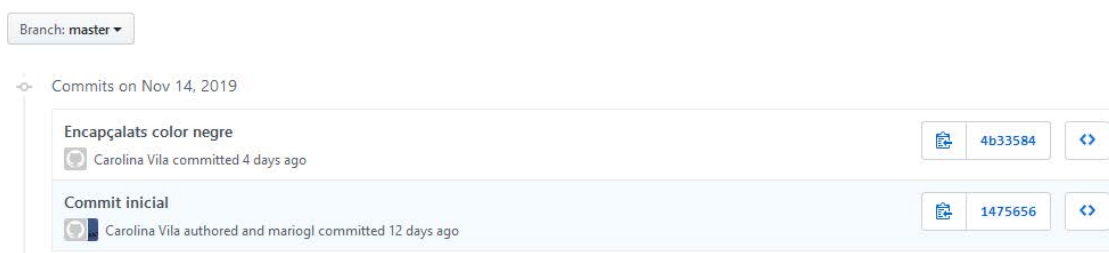
3.8 REVISANT *COMMIT*S

En aquest article aprendrem a revisar l'històric de canvis d'un repositori remot i com revisar el contingut i la informació de cada *commit* que s'ha registrat.

A la pàgina del nostre repositori remot podem fer clic al nombre de *commits*, a dalt a l'esquerra, que ens conduirà a la llista de *commits* de la branca actual, agrupats per dia. Aquesta llista és l'equivalent al *log* que pots veure en el teu repositori local.

Per cada *commit* de la llista, podem veure:

- El seu missatge associat.
- Qui l'ha registrat.
- Quan s'ha registrat.
- Quin és el seu identificador.



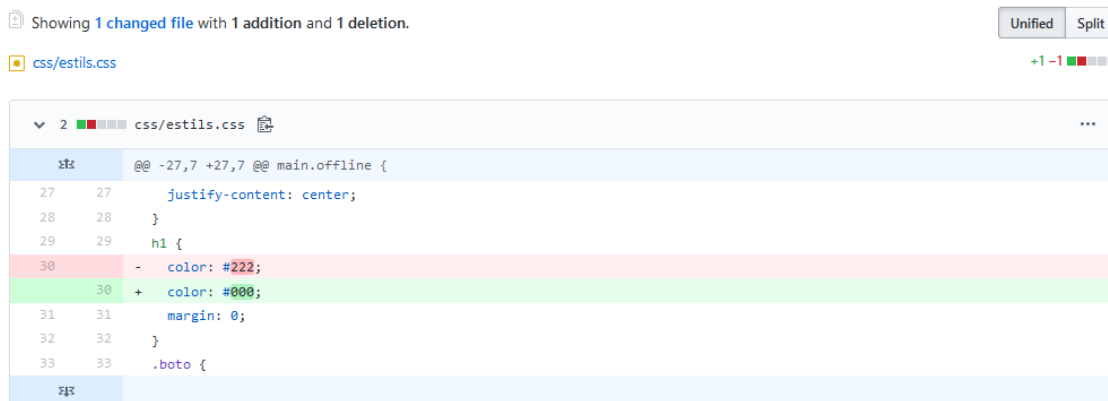
Per veure quines modificacions conté un *commit*, podem fer clic tant al seu missatge, com al seu identificador. A la pàgina que se'ns obre, a més de tenir la mateixa informació que a la pantalla anterior, hi ha una part on ens diu quantes modificacions hi ha: "Showing 1 changed file with 1 addition and 1 deletion". En aquest missatge ens indica el nombre d'arxius modificats i el nombre de línies modificades.

Fent clic en el nombre d'arxius modificats, ens mostrarà la llista d'aquests arxius, acompanyats d'un símbol.

- Si el símbol és un "+", es tracta d'un nou arxiu.
- Si el símbol és un "-", es tracta d'un arxiu eliminat.
- Si el símbol és un ".", es tracta d'un arxiu modificat.

Si fem clic en qualsevol dels arxius, ens mostrarà els canvis que hi ha hagut dins d'un arxiu. Aquí només hi ha dues opcions:

- Si una línia va precedida d'un símbol "+", és una línia nova.
- Si una línia va precedida d'un símbol "-", és una línia eliminada.



Aquí cal tenir en compte que, per a Git, una línia modificada sempre és una línia eliminada i després una línia nova, encara que només hàgim modificat un caràcter. Per tant, si teníem una línia així:

```
background-color: #000;
```

i canviem el color:

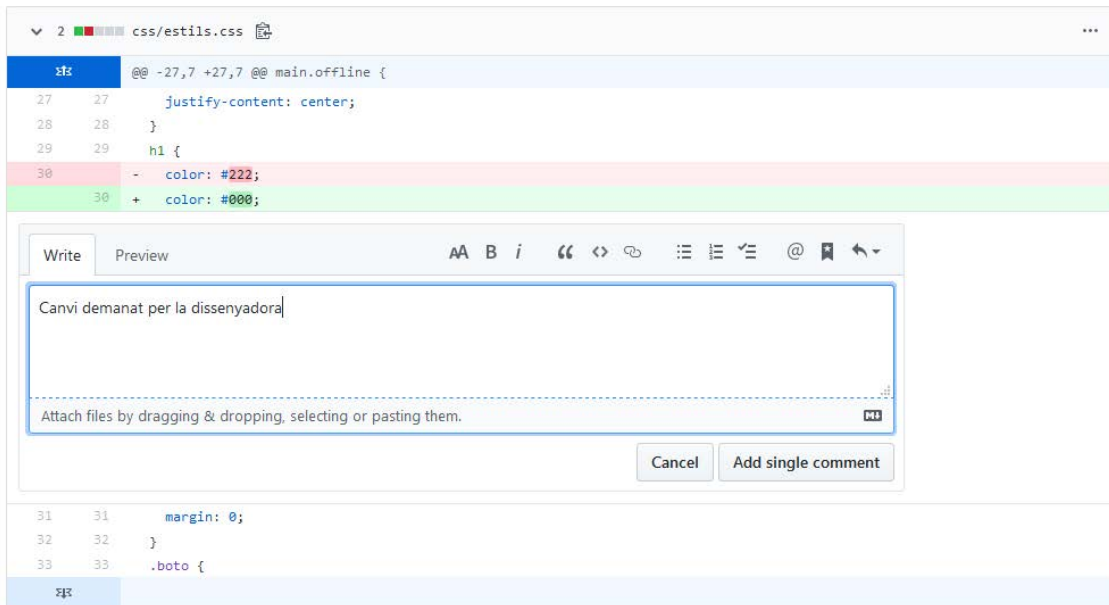
```
background-color: #333;
```

Git ens mostrarà els canvis com si s'hagués esborrat una línia i se n'hagués afegit una de nova:

```

-background-color: #000;
+background-color: #333;
  
```

La interfície de GitHub ens permet afegir comentaris a les línies d'aquests arxius modificats. Si passem el punter del ratolí per sobre d'una línia, ens apareix a l'esquerra una icona amb el símbol "+". Quan fem clic en aquesta icona, se'ns obre un editor per escriure-hi el nostre comentari, que quedarà registrat i associat a aquesta línia.



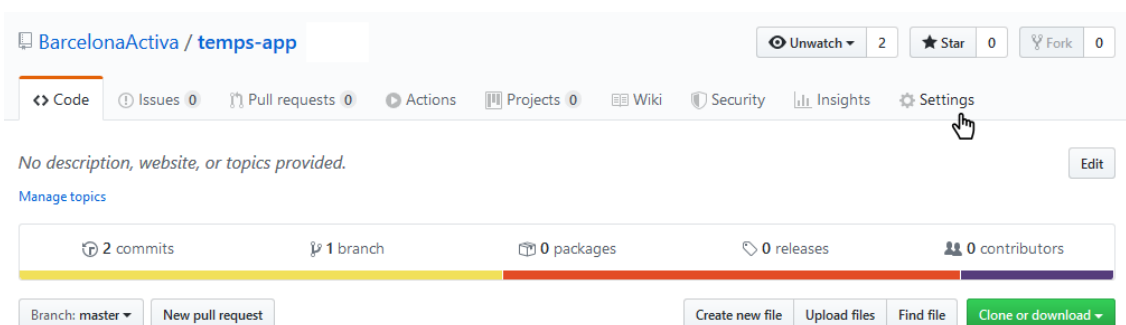
També podem escriure comentaris en format de *commit*, al formulari que ens apareix al final de la pàgina del *commit*. És important tenir en compte que tots aquests comentaris pertanyen a l'àmbit de GitHub i no arribaran als repositoris locals.

3.9 COMPARTINT REPOSITORIS

Si bé podem ser les úniques persones que treballen en un projecte, quan tenim un repositori a GitHub generalment és perquè hi haurà més gent contribuïnt-hi. En aquest article, parlarem sobre com compartir el nostre repositori amb les altres persones.

Si el nostre repositori és públic, qualsevol podrà veure'l a través de la seva URL. Si és privat, només el veuran les persones que decidim. I en tots dos casos només podran realitzar operacions d'escriptura les persones que registrem com a col·laboradors o col·laboradores.

Per donar permisos d'escriptura a algú, anem a la configuració del repositori, a la qual podem accedir mitjançant l'enllaç "Settings" que apareix a dalt a la dreta.



Una vegada siguem a la configuració, premem a l'enllaç “Collaborators”, que ens mostrarà una pàgina amb un cercador. En aquest cercador, podem afegir persones pel seu nom d'usuari a GitHub. Quan afegim algú, cal esperar que aprovi la invitació.

A la persona a qui hàgim afegit, li arribarà un correu electrònic amb un enllaç perquè accepti la invitació. Una vegada acceptada, es convertirà en col·laborador o col·laboradora del repositori. En aquest moment, pot sincronitzar el seu repositori local i realitzar operacions d'escriptura, com, per exemple, pujar *commits* mitjançant *git push*.

Una altra manera de compartir els nostres repositoris és fer-los públics, però sense afegir col·laboradors ni col·laboradores. En aquest cas, qualsevol pot fer un *fork* del nostre repositori. Quan això passa, es crea un repositori nou en el seu compte de GitHub, i aquest repositori és una còpia del nostre. Quan aquesta persona pugi *commits* a la seva còpia, podrà crear una *Pull Request* al nostre repositori, que és una petició per introduir en el nostre repositori els canvis que s'hagin pujat al seu. Podem revisar, primer, els canvis i, després, si ho veiem tot bé, acceptar la *Pull Request*.

Com veiem, tenim opcions per restringir totalment l'accés al nostre repositori, donar accés directe a qui vulguem o permetre la col·laboració mitjançant *Pull Requests*.

3.10 PULL REQUESTS A GITGITHUB

GitHub ha estat, durant anys, la plataforma de referència de l'*Open Source*, i ha allotjat múltiples projectes en els quals col·laboren multitud de persones des de diferents punts del planeta. Per poder coordinar el treball de tanta gent en el mateix projecte, s'utilitzen les *Pull*

Requests. Com veurem en aquest article, les *Pull Requests* permeten a qualsevol persona fer una contribució al nostre repositori, però sempre passant per un filtre de revisió.

El nostre repositori podrà rebre PR (*Pull Requests*) de dues maneres:

- Quan algú realitza un *fork* a partir del nostre repositori. Després podrà pujar canvis al seu *fork* i sol·licitar una PR al nostre repositori, per abocar-hi aquests canvis des del seu *fork*.
- Quan tenim col·laboradors i col·laboradores, però hem restringit l'accés a alguna branca de manera que no s'hi puguin pushar commits directament.

Per a la primera opció, no haurem de realitzar cap acció especial. Qualsevol podrà fer un *fork* del nostre repositori. Anem a veure com faríem la segona opció.

Si anem a la configuració del nostre repositori, al menú lateral veurem una opció anomenada “Branches”. A la configuració de branques, veurem un apartat de regles de protecció: “Branch protection rules”. Aquí és on restringirem una branca perquè només s'hi pugui contribuir mitjançant *Pull Requests*.

Si premem el botó “Add rule”, ens portarà a un formulari on podrem establir un nom de branca (camp “Branch name pattern”). Més avall, tenim una casella amb el text “Require pull request reviews before merging”. Si la marquem, estem restringint la branca de manera que ningú no podrà fer hi *push* directament a sobre. Per contribuir a aquesta branca, caldrà fer-ho mitjançant *Pull Requests*.

Imagina que estàs col·laborant en un repositori i vols contribuir mitjançant PR a la branca *develop*. El teu treball local estarà desenvolupat en una branca diferent, per exemple, *canvi-*

nom-pwa. Primer, hauràs de pujar aquesta branca auxiliar al remot. Un cop pujada, hauràs de crear una PR mitjançant el botó “New pull request”.

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](#).

base: develop ← compare: canvi-nom-pwa ✓ Able to merge. These branches can be automatically merged.

Nou nom PWA

Write Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Reviewers
No reviews—at least 1 approving review is required.

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

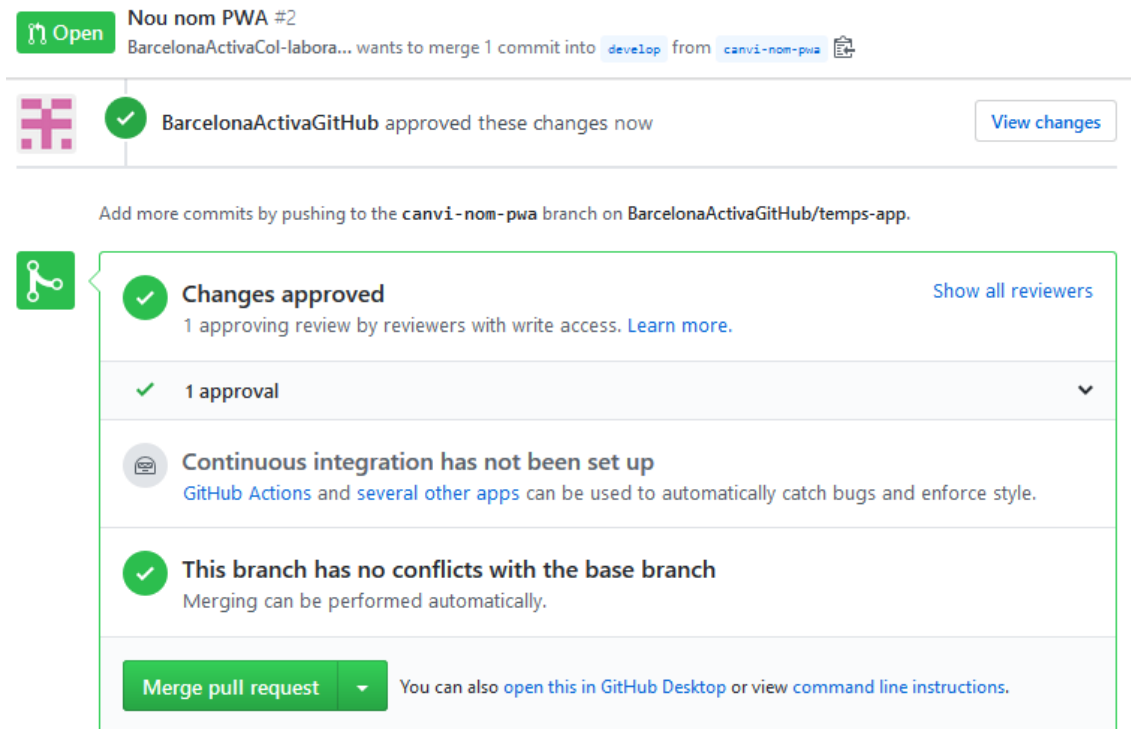
Milestone
No milestone

A la pàgina per crear PR, cal triar la branca *origen* i la branca *destino*. En aquest cas, estem sol·licitant abocar des de la branca *canvi-nom-pwa* a la branca *develop*. Quan triem aquestes dues branques, GitHub ens mostrarà les diferències que hi ha entre totes dues, és a dir, què és el que s’abocarà, si s’accepta la *Pull Request*. Si cliquem en el botó “Create pull request”, haurem creat la petició, que podrà ser revisada per l’equip de revisió del repositori.

Si els nostres canvis passen el filtre de revisió, s’acceptarà la *Pull Request* i automàticament s’abocarà el nostre treball de la branca *canvi-nom-pwa* a la branca *develop*.

Si, en lloc d’estar col·laborant, ets encarregat o encarregada de revisar les *Pull Requests*, hauràs de buscar-les a la pestanya superior anomenada “Pull requests”. T’apareixerà una llista amb totes les PR pendents de revisar i aprovar. Si entres en una, veuràs tots els detalls de la PR, podràs revisar els *commits* i comprovar els canvis a l’interior dels arxius. Si veus algun error, pots escriure un comentari. La PR és un espai de discussió i revisió de codi, on la persona que revisa pot anar comunicant-se amb qui està sol·licitant la inclusió del seu treball.

Quan hakis comprovat que tot és correcte, pots aprovar la revisió a través de l'enllaç “Add your review”. Un cop aprovada, s'habilitarà el botó “Merge pull request”, que és el que finalment realitzarà l'abocament.



Després d'aprovar i *mergear* una de les *Pull Requests*, pots esborrar la branca que la va originar (*canvi-nom-pwa*, en aquest cas), ja que tots els seus *commits* han passat a formar part de la branca *develop*.

3.11 IDEES CLAU: TREBALLANT AMB REPOSITORIS GITHUB

La majoria d'ordres que llançarem a Git treballaran contra el nostre repositori local, però en aquest mòdul hem vist tres ordres clau per a la sincronització amb el repositori remot: *git fetch*, *git pull* i *git push*.

Hem vist com el *log* ens mostra on hi ha les branques remotes (les que comencen per origin /), però això sempre es basa en un caixet. Per actualitzar la informació d'aquest caixet, utilitzarem l'ordre *git fetch*.

Si el que volem és baixar-nos treball del repositori que no tenim en el nostre dispositiu, utilitzarem l'ordre *git pull*. Aquesta ordre farà dues coses: actualitzarà la informació del caixet sobre branques remotes i abocarà a la nostra branca local els nous *commits* que hagi

descarregat del remot. L'ordre ha d'anar acompanyada del nom del *remote* i de la branca: *git pull origin master*. Si no volem haver d'escriure tot això, podem vincular les branques, llançant l'ordre *git branch --set-upstream-to origin/master*.

Si volem pujar al repositori remot els *commits* que hem creat en local, utilitzarem l'ordre *git push*, acompanyada del nom del *remote* i de la branca (*git push origin master*). Si vinculem les branques, ja no serà necessari, i podrem escriure només *git push*.

Quan creem un repositori, per permetre que altres persones puguin contribuir-hi, tenim dues opcions:

- Les convidem, a través del seu nom d'usuari. Quan acceptin la invitació, podran realitzar *push* al repositori.
- Deixem el repositori sense col·laboradors ni col·laboradores i qui vulgui contribuir haurà de fer un *fork* del repositori. Una vegada hagi pujat canvis al seu repositori *fork*, podrà crear una *Pull Request* al nostre repositori, per sol·licitar que s'incloguin els seus canvis.

Les *Pull Requests* són un sistema molt estès de col·laboració. Mitjançant aquest sistema, algú sol·licita la inclusió dels seus *commits* al nostre repositori, però no s'inclouran fins que no passin un sistema de revisió. Seran les persones encarregades de la revisió les que finalment faran el pas d'incloure aquests canvis al repositori remot.

4 ALTRES REPOSITORIS

Per què GitHub?

Quan estem desenvolupant en equip un projecte versionat amb Git, l'habitual és tenir un repositori centralitzat al qual es connectaran totes les persones que hi col·laboren. Avui en dia, hi ha diverses alternatives a l'hora de triar un servei per allotjar el nostre repositori *online*. No obstant això, GitHub és el més conegut i també el més utilitzat de tots.

GitHub és una plataforma per a la gestió col·laborativa de projectes de programari. El seu cor està en els repositoris Git que ofereix, però en els últims anys ha anat incloent eines per a *Continuous Integration / Continuous Delivery*. D'aquesta manera, es col·loca a l'altura dels seus competidors i competidores més pròximes, com Bitbucket i GitLab, que li portaven avantatge quant a automatització de processos. No obstant això, aquestes eines d'automatització només són gratuïtes en repositoris públics. A diferència de les altres plataformes, si volem utilitzar-les en repositoris privats, haurem de pagar pel servei.

El **gran avantatge de GitHub**, i potser el que més la diferencia de la resta d'alternatives, és el seu vessant de xarxa social de desenvolupadors i desenvolupadores. Què volem dir? La plataforma ens ofereix un entorn per al nostre repositori que està molt orientat a presentar-lo al públic. Podem crear un *Wiki* amb informació sobre el projecte, i fins i tot podem allotjar una

web simple amb *GitHub Pages*. També ofereix un gestor d'*Issues*, on la resta de les persones usuàries podran proposar millores o reportar *bugs*. Finalment, el seu sistema de *Gists* (plataforma per a *snippets*) és molt utilitzat per compartir *snippets* (petites parts reutilitzables de codi font, codi màquina o text) de codi amb altres persones.

El **principal inconvenient de GitHub** és que el seu compte gratuït només ofereix un màxim de tres persones que poden col·laborar en els repositoris privats. Per això només poden utilitzar aquesta característica persones individuals, sense equip, o equips molt petits. Plataformes com GitLab o Bitbucket no tenen un límit tan baix en comptes gratuïts.

Un altre dels inconvenients que té GitHub és que la seva interfície d'usuari és bastant menys intuïtiva que la de Bitbucket o GitLab i costa més temps familiaritzar-se amb l'entorn.

Des que Microsoft es va convertir en propietària de GitHub, el seu objectiu és aconseguir fer-lo més atractiu per a les empreses. Això farà que progressivament vagin desapareixent les restriccions que fins ara feien del servei quelcom més orientat a l'*Open Source*.

Si treballes com a desenvolupador o desenvolupadora que només es dedica a projectes oberts, GitHub és probablement la plataforma amb més detalls pensats per a tu. Si, per contra, treballes amb repositoris privats i en equips mitjans o grans, Bitbucket i GitLab són probablement plataformes més ben preparades per al que necessites.

4.1 GITLAB

Benvinguts i benvingudes!

Tot i que feia anys que estava a l'ombra de plataformes com GitHub i Bitbucket, GitLab ha passat a ser dels serveis *online* més utilitzats per albergar repositoris Git.

L'adquisició de GitHub per part de Microsoft el 2018 va provocar una migració massiva d'usuaris i usuàries de GitHub a GitLab. Des de llavors, aquesta plataforma s'ha convertit en una de les més utilitzades i la més ben valorada. Es caracteritza i es diferencia de la resta de les seves competidores per ser l'única eina *Open Source*.

GitLab ofereix un entorn integral de *DevOps* en el qual podem trobar eines típiques de *Continuous Integration / Continuous Delivery*, gestió de tasques i repositoris Git *online*. El servei ve, com en la resta de plataformes, de dues maneres:

- **Gitlab.com:** és la solució al núvol, una plataforma a la qual ens podrem connectar des de qualsevol lloc amb accés a internet.
- **Self-Managed:** és el programari preparat per instal·lar al nostre servidor, per a quan vulguem mantenir els nostres repositoris en una xarxa controlada.

Totes dues solucions ens ofereixen comptes gratuïts i comptes de pagament. En tots els casos podrem crear un nombre il·limitat de repositoris públics o privats. Una de les majors

diferències amb plataformes com GitHub o Bitbucket és que GitLab no ens limita el nombre de persones que poden col·laborar en els nostres projectes, ni tan sols en el compte gratuït.

I com ens donem d'alta en un compte a GitLab? Vegem-ho!

El formulari de registre de GitLab et demanarà un nom d'usuari (que formarà part de l'URL dels teus repositoris), una adreça de correu electrònic i una contrasenya.

Una vegada omplert aquest formulari, et demanarà el nom complet i algunes dades estadístiques. Després, ja pots començar a utilitzar la plataforma, tot i que t'arribarà una confirmació del compte al correu que hagi proporcionat.

I si volguessis modificar les dades del teu compte i l'avatar? Pots fer-ho des de la configuració del teu compte, clicant a l'opció "Settings" del menú que es desplega a sota del teu avatar. Aquí podràs canviar tant la imatge com l'adreça de correu electrònic i altres dades que formen part del teu perfil. Des de l'opció "Account" del menú lateral podràs canviar el teu nom d'usuari o tancar el compte. Però, alerta!, tingues en compte que si canvies el nom d'usuari, les URL dels teus repositoris canviaran i totes les persones que hi estiguin col·laborant hauran d'actualitzar els seus *remotes*.

Per canviar a un compte de pagament, has d'anar a l'opció "Billing" del menú lateral. Allà podràs triar qualsevol dels plans de pagament que ofereix la plataforma.

En definitiva, GitLab és una molt bona alternativa a GitHub i, tot i que està mancat de la potència que té aquesta altra com a xarxa social de desenvolupadors i desenvolupadores, ja és la primera opció per a molts equips de treball. Seguiu!

4.2 BITBUCKET

Hola a tots i a totes!

Si bé GitHub és la plataforma més utilitzada quant a repositoris Git *online*, el segon lloc l'ocupa Bitbucket. Mentre que GitHub és més conegut per ser referent en el món de l'Open *Source*, Bitbucket està més estès entre les empreses.

Bitbucket és un servei orientat a la planificació de projectes en equip. Els repositoris Git són només una de les característiques que té, però també està preparat per a Mercurial (un altre sistema de control de versions) i té eines per a *Continuous Integration / Continuous Delivery*. També té totalment integrats els gestors de tasques Jira –que és de la mateixa empresa que Bitbucket– i Trello.

Podem utilitzar Bitbucket de dues maneres diferents:

- *Cloud*: és el servei de Bitbucket al núvol i podrem connectar-nos als nostres repositoris des de qualsevol lloc amb accés a internet.

- *Self-hosted*: és una versió de Bitbucket per instal·lar al nostre servidor, per a quan vulguem que els nostres repositoris estiguin dins d'una xarxa controlada.

Si bé cadascun dels dos productes té plans amb preus diferents, la versió *cloud* és l'única que ens ofereix un compte gratuït. Amb aquest compte, podem crear tots els repositoris que vulguem, tant públics com privats, amb capacitat per convidar fins a cinc persones per col·laborar-hi. Amb els comptes de pagament no existeix aquest límit i podem convidar a totes les persones que vulguem.

Ara anem a veure com podem obrir un compte a Bitbucket. La plataforma no té un sistema de comptes propi, sinó que utilitza els comptes d'usuari d'Atlassian, la marca propietària del servei. D'aquesta manera, tenint un compte per a Bitbucket, també el tindràs per a Jira i viceversa.

Per crear el nostre compte, el primer que demana Bitbucket és una adreça de correu electrònic. Una vegada que la introdueixis, et demanarà el teu nom complet i una contrasenya d'accés.

Després de validar les dades, la plataforma t'enviarà un correu de confirmació a l'adreça que hagi introduït en el primer pas.

Quan iniciïs sessió per primera vegada després d'haver confirmat l'adreça, Bitbucket et demanarà un nom d'usuari. Aquest nom formarà part de la teva URL a la plataforma. Després d'omplir (o ometre) una petita enquesta, ja tindràs el teu compte a punt.

A partir d'aquí, pots modificar les dades i l'avatar del teu compte en qualsevol moment. La configuració la tens a la icona amb el teu avatar, prement a "Bitbucket settings". Clicant a "Atlassian Account", podràs accedir a totes les dades del teu compte.

Si en algun moment vols passar a un compte de pagament, és tan fàcil com anar a l'opció "Plan details" del menú lateral i prémer el botó "Upgrade plan".

Com veus, malgrat ser una interfície d'usuari bastant diferent a la de GitHub, pots trobar funcionalitats molt semblants en totes dues plataformes.

4.3 SOURCEFORGE

Hola!

En aquest vídeo parlarem d'una plataforma que és probablement més coneguda com a centre de descàrregues de programari que com a repositori *online*: SourceForge.

SourceForge fa molts anys que ofereix un servei d'allotjament de projectes *Open Source*, incloent sistemes de control de versions com Git, Mercurial o Subversion. També té *Pull Requests* (anomenades *Merge Requests*, en aquesta plataforma) i eines per gestionar *Issues*.

A diferència de molts dels seus competidors i competidores, no compta amb un entorn per a *Continuous Integration / Continuous Delivery*.

SourceForge continua allotjant multitud de projectes *Open Source*, però els seus múltiples intents de monetitzar els serveis de la plataforma han fet que molts usuaris i usuàries hagin deixat d'utilitzar-lo. És un bon lloc per allotjar els nostres repositoris, però, si l'utilitzem, haurem d'estar una mica més pendents de quins serveis que fins ara eren gratuïts es converteixen en serveis de pagament.

Obrir un compte és gratuït i podem fer que els nostres repositoris siguin privats, però només durant la fase de desenvolupament. SourceForge exigeix que tots els projectes finalment estiguin oberts al públic i siguin *Open Source*, per la qual cosa, si necessitem que els nostres repositoris estiguin ocults, és recomanable que utilitzem altres plataformes.

Anem a veure ara com crear un compte a SourceForge. Veuràs que el formulari de registre té bastants camps, tot i que no tots són obligatoris. Els camps que hauràs d'omplir, com a mínim, seran:

- El teu nom complet
- Una adreça de correu electrònic
- Un nom d'usuari
- Una contrasenya d'accés
- Una via de contacte

Quan enviïs aquest formulari, t'arribarà una confirmació del compte al correu que hagi proporcionat.

Una vegada iniciada la sessió, després de confirmar el compte, podràs crear el teu primer projecte. A SourceForge no creem repositoris directament, com se sol fer en altres plataformes. El que es fa és crear un projecte de programari i, dins del projecte, podem crear el repositori Git.

Per crear un projecte, prem el botó "Create Your Project Now". Això et portarà al formulari de creació, on has d'introduir un nom de projecte. L'URL del projecte es basarà, per defecte, en el nom que hagi triat, però la pots canviar, si vols. Després, hauràs de triar quines eines tindrà el teu projecte, d'entre les que ofereix SourceForge. Les que ens interessin en aquest curs són les relacionades amb el control de versions. Veuràs que SourceForge et permet crear repositoris a Git, a Mercurial i a Subversion.

Per modificar les dades del teu compte, has de fer clic a l'opció "Account Settings", que hi ha a sota de l'enllaç "Me". Aquí podràs modificar el teu nom d'usuari, afegir o eliminar adreces de correu electrònic i canviar la teva contrasenya d'accés a la plataforma.

4.4 ALTRES REPOSITORIS *ONLINE*

Encara que el mercat de l'allotjament *online* està encapçalat per GitHub, GitLab i Bitbucket, hi ha moltes altres solucions. En aquest article en comentarem algunes.

[Microsoft Azure](#)

La plataforma Azure de Microsoft ofereix un entorn complet basat en *DevOps*. Dins de totes les eines que ofereix hi ha els repositoris *online* per a Git. Aquí podrem trobar gairebé totes les característiques de la resta de plataformes: repositoris públics o privats (il·limitats), *Pull Requests*, permisos per decidir la manera de col·laborar en un repositori... Tot això integrat amb eines de *Continuous Integration / Continuous Delivery* i de gestió de tasques.

[Launchpad](#)

Launchpad té al darrere l'equip d'Ubuntu Linux i és una plataforma discreta però pràctica per allotjar els nostres repositoris Git. Admet repositoris públics i privats, i també es pot contribuir a un projecte mitjançant *Pull Requests*.

El seu principal inconvenient resideix en una pobra interfície d'usuari que dificulta la navegació per la plataforma i els repositoris.

[Beanstalk](#)

La plataforma Beanstalk ens ofereix un entorn de desenvolupament de projectes amb allotjament per a repositoris Git o Subversion. Està molt orientada a empreses i l'únic compte gratuït que ofereix només permet un usuari o usuària i un repositori. Tots els altres comptes són de pagament, però donen dues setmanes de prova gratis.

[Cloud Source](#)

Cloud Source forma part del servei de Google Cloud i ens ofereix exclusivament repositoris privats però il·limitats. L'avantatge d'aquesta plataforma és que, pel fet d'estar integrada a Google Cloud, permet tenir accés des del nostre repositori a la majoria d'eines que ofereix Google.

[AWS CodeCommit](#)

CodeCommit és un servei gratuït d'AWS que ofereix repositoris Git, exclusivament privats. L'avantatge d'aquesta plataforma és que està integrada amb la resta de serveis d'AWS, amb la qual cosa podrem automatitzar fàcilment el *deployment* del nostre projecte. A més, compta amb tot el que solen portar els repositoris *online* per col·laborar: permisos de branques, *Pull Requests* i gestor d'*issues*.

4.5 IDEES CLAU: ALTRES REPOSITORIS

En aquest mòdul hem vist quins són els avantatges i inconvenients d'utilitzar GitHub com a allotjament per als nostres repositoris *online*. GitHub ha estat, durant molts anys, un referent en el món de l'*Open Source* i l'han seguit de prop plataformes com Bitbucket i GitLab. SourceForge ja no s'utilitza tant com fa anys, però continua essent una alternativa perfectament vàlida.

Les diferències entre GitHub i els seus dos competidors pròxims sempre s'han basat en el fet que GitLab i Bitbucket oferien repositoris privats il·limitats i un entorn de *Continuous Integration / Continuous Delivery*. Aquestes diferències s'han anat escurçant des que Microsoft va adquirir GitHub i va començar a fer-lo més atractiu per a les empreses. Ara les tres plataformes s'assemblen bastant quant a les funcionalitats que ofereixen, però GitHub segueix essent la que més limitacions té per a comptes gratuïts.

Bitbucket és la referència en l'entorn empresarial. La seva integració amb eines com Jira i Trello i els il·limitats repositoris privats han fet que molts equips de desenvolupament de l'entorn corporatiu hagin optat per aquesta plataforma. Ofereix un entorn de CI/CD per ajudar en els processos de *deployment* i també té les típiques opcions de col·laboració: *Pull Requests* i permisos de branques.

GitLab ha estat la destinació de moltes persones que van sortir de GitHub després de la seva adquisició per part de Microsoft. GitLab és la plataforma que ofereix menys restriccions amb el seu compte gratuït i també ofereix eines de CI/CD i de col·laboració, com *Pull Requests* (anomenades *Merge Requests* a GitLab) i permisos de branques.

SourceForge té menys persones usuàries que les altres tres, però encara continua essent un lloc bastant central per al món de l'*Open Source*. Si bé ofereix *Pull Requests* per a entorns col·laboratius, no té ni eines de CI/CD ni permisos de branques.

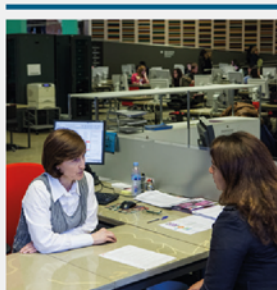
Però les quatre plataformes esmentades no són les úniques. També tenim altres serveis per allotjar els nostres repositoris *online*, com ara:

- [Microsoft Azure](#)
- [Launchpad](#)
- [Beanstalk](#) (amb gairebé totes les opcions de pagament)
- [Cloud Source](#) (només amb repositoris privats)
- [AWS CodeCommit](#) (només amb repositoris privats)

Pots veure com el ventall d'opcions a l'hora de triar on residirà el teu repositori *online* és molt ampli i gairebé totes les plataformes t'oferiran opcions gratuïtes per a què puguis provar i quedar-te amb la qual més et convingui.

- Bitbucket: <<https://bitbucket.org>>
- GitLab: <<https://gitlab.com>>
- SourceForge: <<https://sourceforge.net>>

Descobreix tot el que Barcelona Activa pot fer per a tu



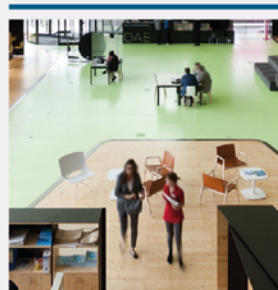
Acompanyament durant tot el procés de recerca de feina

barcelonactiva.cat/treball



Suport per posar en marxa la teva idea de negoci

barcelonactiva.cat/emprenedoria



Serveis a les empreses i iniciatives socioempresarials

barcelonactiva.cat/empreses

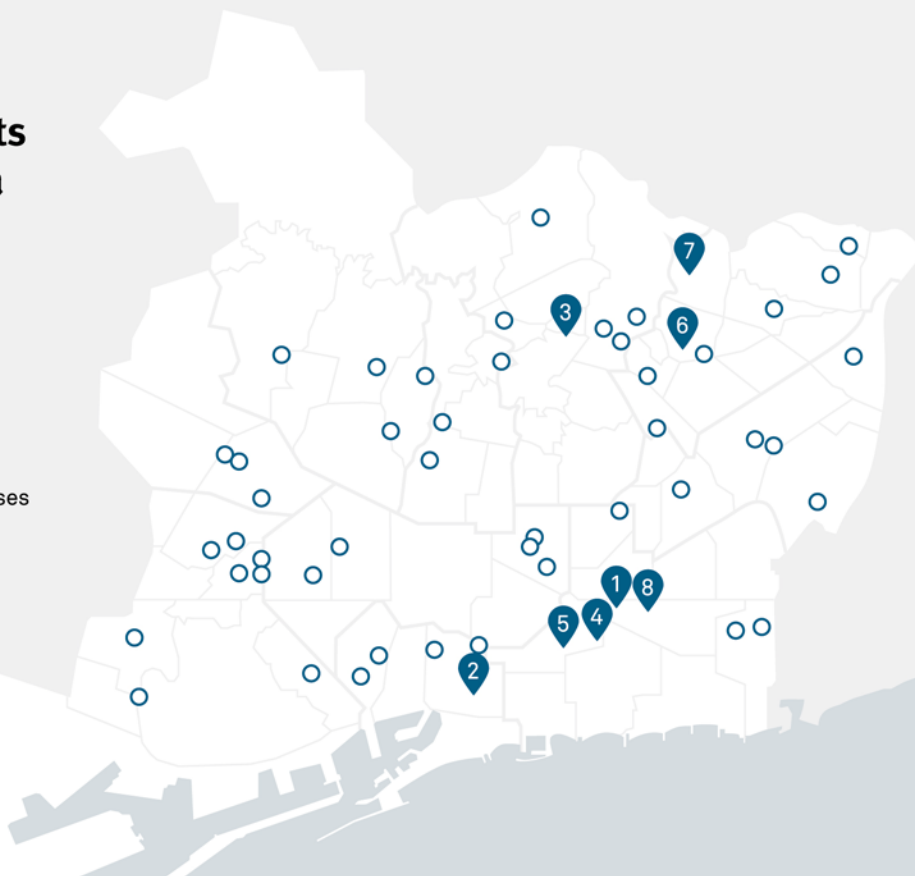


Formació tecnològica i gratuïta per a la ciutadania

barcelonactiva.cat/cibernarium

Xarxa d'equipaments de Barcelona Activa

- 1 Seu Central Barcelona Activa
Porta 22
Centre per a la Iniciativa
Emprenedora Glòries
Incubadora Glòries
- 2 Convent de Sant Agustí
- 3 Ca n'Andalet
- 4 Oficina d'Atenció a les Empreses
Cibernàrium
Incubadora MediaTIC
- 5 Incubadora Almogàvers
- 6 Parc Tecnològic
- 7 Nou Barris Activa
- 8 innoBA
- Punts d'atenció a la ciutat



© Barcelona Activa
Darrera actualització 2020

Cofinançat per:



UNIÓ EUROPEA
Fons Europeu de Desenvolupament Regional

Segueix-nos a les xarxes socials:



barcelonactiva.cat/cibernarium



[barcelonactiva](https://facebook.com/barcelonactiva)



[barcelonactiva](https://twitter.com/barcelonactiva)



company/barcelona-activa