GIT & GITHUB

GIT

QUÈ ÉS GIT?

Git és un sistema de control de versions (VCS=Version control System), de codi obert.
Permet rastrejar versions anteriors dels arxius amb els quals estem treballant, i per tant permet:

- CONSULTAR versions anteriors dels arxius
- DESFER canvis en els arxius
- MANTENIR diferents versions d'un mateix projecte al mateix temps

Git és un sistema distribuït i descentralitzat, és a dir, les persones que treballen en el mateix projecte poden clonar el projecte, que es guarda en un repositori, treballar de forma individual en una part del codi, per després enviar aquests canvis al repositori, que actualitza el moment perquè siguin excesibles a la resta de persones que treballen en el projecte. Per tant permet treballar de manera fàcil, a moltes persones en un mateix projecte ja que permet combinar tots els canvis que es realitzen de forma individual, en un mateix repositori.
Hi ha diversos serveis que s'utilitzen per guardar aquests repositoris BitBucket, GitLab, GitHub, ... En aquests serveis es pot guardar codi de forma pública i també de forma privada. Nosaltres farem servit GitHub, que és el més popular per a projectes de codi obert, en parlarem més endavant.

INSTAL·LACIÓ DE GIT

Instalar Git és força fàcil en totes les plataformes:

Windows

Es pot descarregar aquí: https://gitforwindows.org/

MAC OSX

Primer s'ha d'instalar brew: https://brew.sh/

Un cop instal·lat brew s'ha d'introduir per consola la següent ordre:

```
brew install git
```

Linux

Sempre per consola :

- Per distribucions basades en Debian, per exemple Ubuntu:

```
$ apt-get install git
```

- Per a Fedora:

```
$ yum install git-core
```

Comprovació

Desde el terminal escriviu l'ordre:

```
$ git --version
```

Això us hauria de donar la versió de Git que teniu instalada :)

CONFIGURACIÓ BÀSICA I ORDRES D'AJUDA

Ordres d'Ajuda

`$ git help -a`

Dona una llista força llarga de l'ajuda de totes les ordres diponibles.

`$ git help <ordres>`

Ens proporciona ajuda sobre una ordre concreta.

`$ git help tutorial`

Llistat d'ordres basiques amb una breu descripció.

`$ git help everyday`

Llistat de les ordres més usades amb una breu descripció.

`$ git help revisions`

Llistat d'ordres que s'usen per gestionar les revisions amb una breu descripció.

`$ git help workflows`

Llistat d'ordres que s'usen per gestionar lfluxos de treball, ja siguin projectes individuals o projectes en grup.

Configuració Bàsica

De moment no podem configurar els nostres dades en Git, ja que encara no hem creat el nostre compte a Github. Perquè és important configurar-lo? Doncs perquè Git, pren "*fotografies instantànies"* del nostre codi, i emmagatzema qui fa el canvi.
Les ordres per configurar les nostres dades són:

`$ git config –global user.name "El vostre nom"` (amb " ")

Configurem el nostre nom d'usuari, perque se'ns identifiqui correctament.

`$ git config –global user.mail "email@exemple.com"` (sense " ")

Configurem el nostre correu electrònic, perque ens arribin les notificacions.

 Tot això ho farem un cop haguem creat el nostre compte (amb nom d'usuari i correu electrònic) a GitHub, ja que han de coincidir.

GITHUB

QUÈ ÉS GITHUB?

GitHub és una pàgina web, que serveix d'allotjament per a milions de projectes. En general el seu ús és d'emmagatzematge de codi i per col·laborar en el codi d'altres persones, de manera publica.
Github és una poderosa eina ja que permet:

• Compartir el codi amb un altre usuari o la comunitat de Github
• Col • laborar en projectes interessants d'altres usuaris
• Creació d'una còpia (fork) d'un projecte que vol millorar
• Fluxos de treball fàcils per equipsde treball
• Permet actualitzar el codi directament des del terminal

Com es relacionen Git i GitHub

Git va de la mà amb Github, però són coses diferents i la gent els confon:

• Git és un programa que funciona localment, a l'ordinador, i emmagatzema instantàniament el seu codi en una base de dades que com ja hem dit anomenarem repositoris.

• Github és un Servidor Central VCS, un servei d'allotjament per a dipòsits. Es tracta d'una aplicació web que permet als usuaris carregar els seus dipòsits i guardar-los en un emmagatzematge centralitzat.

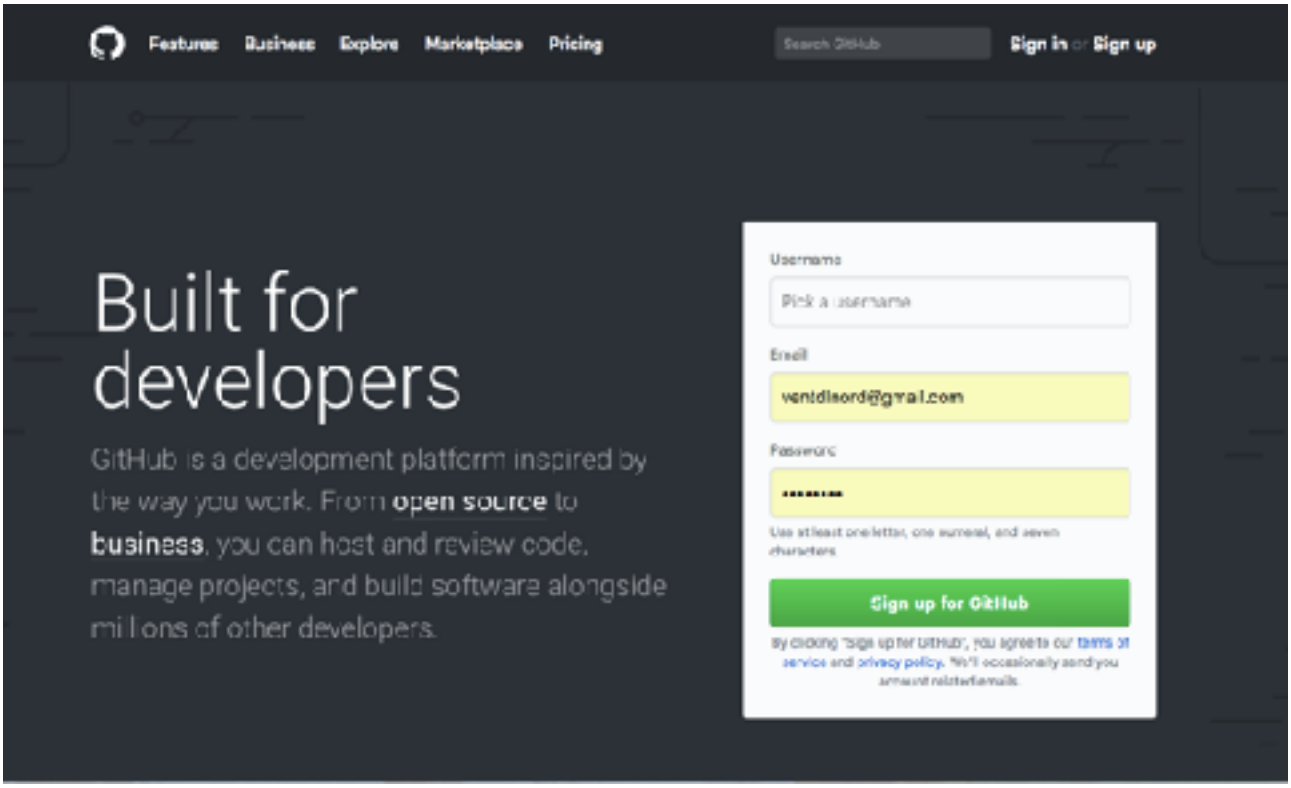
El octocat és la mascota i el logotip de Github. Va ser dissenyat per Simon Oxley, el mateix noi que va dissenyar l'ocell de Twitter.

És un pop amb el cap d'un gat i els usuaris de Github han estat creant versions del octocat des que Github va néixer. Si voleu fer una ullada a totes les versions, aneu a OctoDex.

CREAR UN COMPTE DE GITHUB

Per crear un compte de GitHub sols hem d'anar a la pàgina web i crear un compte, que serà gratuit ;)

Recordau que hem de donar un nom d'usuari i un correu electrònic que haurem de recordar per a poder configurar les nostres dades a Git, un cop creat el compte de GitHub.



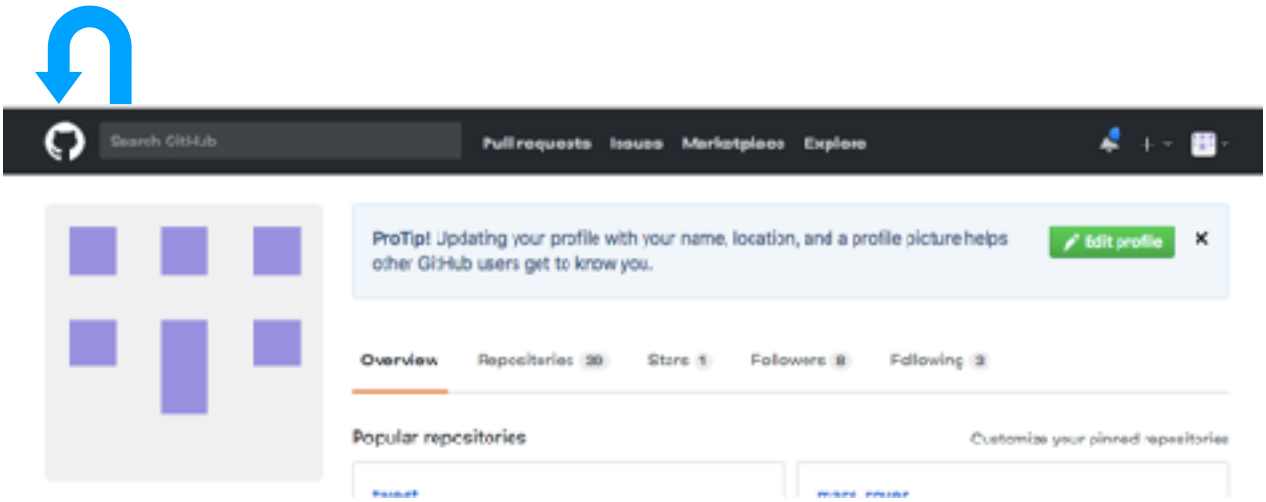
CREAR UN REPOSITORI

Suposem que volem iniciar un projecte: un notebook en R, una applicació web, una entrega d'un programa d'alguna assignatura, … Qualsevol tipus de projecte.

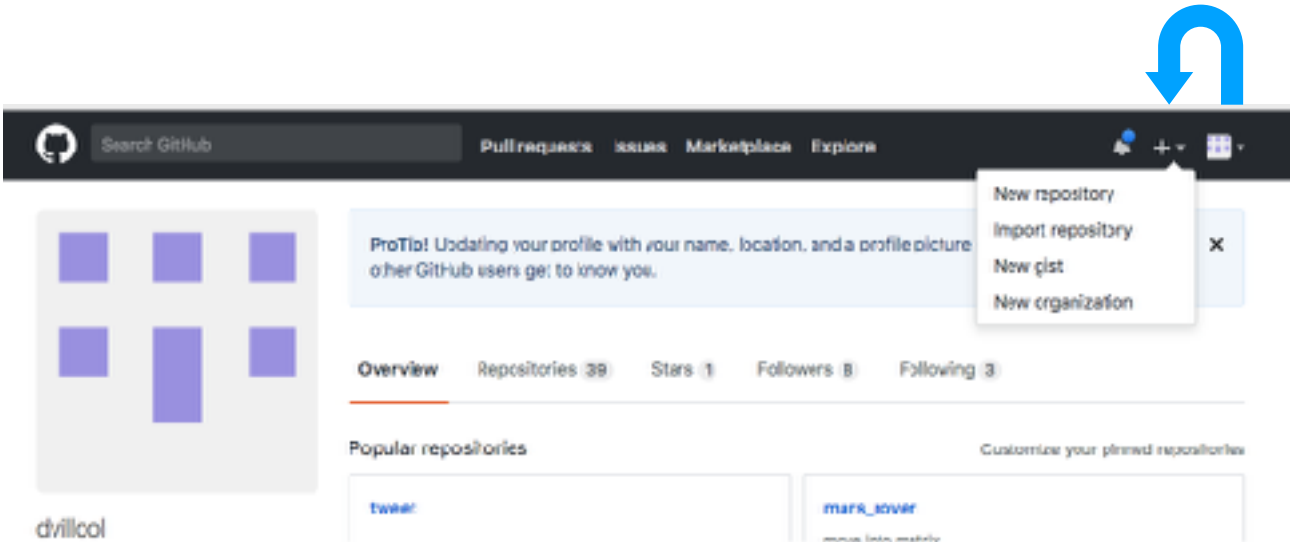
Abans de res haurem de crear el repositori en el nostre compte de **GITHUB.**

Si no ho heu fet abans, feu-ho ara ;)

Podeu anar a la vostre pàgina principal clicant l'icona de l'octocat (a dalt a l'esquerra):



El següent pas es clicar en el simbol + que hi ha a dalt a la dreta. Si ho feu s'obrirà un desplegable, on haureu de triar l'opció "New Repository":

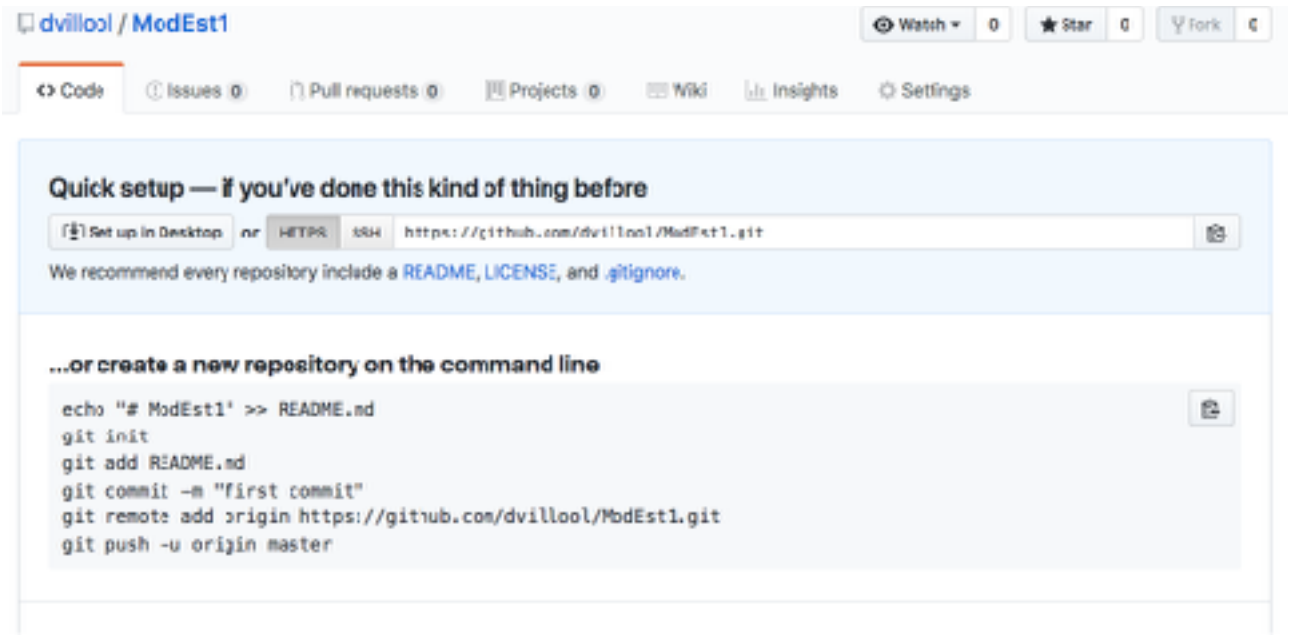


Això ens durà a una nova pàgina on ens permetrà posar el nom al nostre repositori

⚠️ El nom hauria de ser fàcil i curt possible, i per una bona pràctica es recomana que sigui d'una sola paraula (encara que aquesta sigui una fusió de dues paraules), ja que serà el nom que li haurem de posar a la carpeta del projecte en el nostre ordinador.

A l'indicar-li el nom del nou repositori, es desbloqueja el botó que ens permet crear-lo, i per descomptat el cliquem.

Ara, ens permet triar entre diverses maneres d'afegir o crear un nou repositori, en aquest cas i per començar, el que seguirem són les instruccions del primer cas, *"or create a new repository on the command line":*



Farem els passos per les següents raons:

—>Primer haurem de crear la carpeta del projecte en el directori que més ens agradi, recordeu que li ha de posar el nom del repositori creat en Github.
Entrem dins de la carpeta, i seguidament a través del terminal vam entrar les ordres que ens donen en les instruccions del seu compte de GitHub. Les ordres són:

```
$ git init
```

Aquesta ordre crea un fitxer o directori ocult (amb extensió .git) en la nostra carpeta del projecte. En aquest directori ocult és on Git opera i desa les vostres dades, de manera que ara mateix està buit, ja que no tenim res a la carpeta del nostre projecte i encara no hem fet cap canvi.

—>La segona ordre que ens dóna GitHub és la que ens permetrà afegir aquest arxiu, sempre que ho desitgem (no és obligatori afegir un arxiu de README.txt, però en aquest cas ho farem, ja que el farem servir d'arxiu de proves).

Però primer haurem de crear-lo:

`$ touch arxiu.ext` (ordre de consola “touch”)

Com afegim UN arxiu del projecte **a Git**? amb la següent ordre:

```
$ git add arxiu.ext
```

 Si volguessim afegir tots els arxius modificats o creats hauriem de donar l’ordre següent: `$ git add .`

—>Ara ens diuen que notifiquem aquest canvis de manera official. Aquest pas és el més important de tots, ja que cada vegada que ho fem quedarà guardat per a la posteritat, i perquè quedi constància haurem de posar un missatge entre cometes (simples o dobles no importa), on haurem d'indicar el que estem notificant de manera específica, ja que en un futur ens servirà de guia o índex per a nosaltres mateixos, o per a les altres persones que també treballen en el projecte. **Notifiquem els canvis**:

```
$ git commit –m ‘Add the file arxiu.ext’
```

—>El quart pas, és crear un vincle entre el nostre repositori en GitHub i el projecte en el nostre ordinador, això ho podem fer gràcies a Git que és el programa que ens fa d'enllaç:

```
$ git remote add origin “direcció URL que ens doni GitHub”
```

La direcció URL ens la donarà la pàgina de GitHub que estem seguint. ;)

—> Finalment, només ens queda actualitzar el repositori en GitHub, amb tots els canvis que fins ara hem registrat a la carpeta del projecte en el nostre ordinador, l'ordre és:

```
$ git push -u origin master
```

 També podríem haver creat el projecte primer al nostre ordinador, guardar-ho tot en git (és a dir seguir igualment totes les ordres menys la ultima git push), crear el repositori en GitHub i finalment fer el git push per actualitzar el repositori.

TREBALLAR INDIVIDUALMENT

Ara que ja hem creat un projecte, treballarem amb ell. En la realitat els nostres projectes tindran diversos arxius amb codi, però en el nostre cas només tenim un arxiu de text.

El procés que sempre seguirem serà el següent:

Treballar en un arxiu i guardar els canvis (també podem crear altres arxius, carpetes, ...)

Guardar els canvis a Git

Actualitzar els canvis en el repositori de GitHub

Fem-ho amb el nostre projecte i al mateix temps anirem comprovant i descobrint algunes curiositats:

Primer anem a veure l'estat del projecte, per a això li donarem la següent ordre:

```
$ git status
```

- Obrim l'arxiu de text i escrivim qualsevol frase, i ho guardem.
- Ara, tornem a consultat l'estat del projecte: `$ git status`

Podem veure que Git ens diu que s'ha modificat un arxiu, però que no li hem dit res, fem-ho:

```
$ git add .
$ git commit –m ‘First mod in README.txt’
```

- Si tornem a consultar l'estat, Git ens dirà que tot està correcte.
- El següent pas és consultar el nostre repositori en GitHub. Obrim el nostre repositori (clicant sobre el nom d'aquest), i veurem la llista de fitxers del nostre projecte (en el nostre cas només hi ha un README.txt). Si l'obrim podem comprovar que està buit !!
- Però com pot ser, si nosaltres ho hem guardat i li hem notificat a Git? No cal que els corrents vagi a revisar el seu arxiu, encara que si desitja pot fer-ho. ;)
- El que ha passat és que no hem actualitzat els canvis que li hem notificat a Git en el nostre repositori, per tant GitHub, no sap res i per això no ho mostra.
- Per tant el que ara farem es actualitzar el nostre repositori amb la següent ordre:

```
$ git push origin master
```

El que hem fet amb aquesta ultima ordre, és prendre tot els canvis que hem fet i notificat a Git, i els hem actualitzat a GitHub, per tant si ara tornem al nostre repositori del projecte i actualitzem la pàgina podrem veure com si es mostren els nostres canvis.

En aquest punt cal destacar un parell de coses:

-> Si treballem sense connexió, però anem guardant els canvis a Git, i finalment donem l'ordre de actalització del repositori, les actualitacions no es mostraran fins que no tornem a tenir connexió, però no caldrà tornar-li a repetir l'ordre per a terminal, Git s'activarà quan torni a tenir connexió i executés l'ordre d'actualització per si sol.

-> Ara que ja tenim notificat un canvi, podem **consultar el nostre index de canvis**:

```
$ git log
```

Aquesta ordre ens obre un arxiu de text, del qual podrem sortir amb l'ordre q(= quit). En aquest arxiu de text ens dóna una llista de tots els canvis que portem fets fins aquell moment, amb la data, hora, usuari que ha fet els canvis i el Id del canvi.

Tots els canvis tenen el seu propi Aneu, per la qual cosa a través d'aquest Aneu podrem consultar-los. Això ens permet, en el cas que ho necessitem, consultar com estava el codi en un cert moment del projecte.

Tornem a treballar amb el nostre projecte:

Vam consultar l'estat del projecte i mirem l'índex de canvis, és a dir, donem pel terminal les següents ordres:

```
$ git status
$ git log
```

La primera ordre ens diu que tot està correcte, la segona ens obre un arxiu de text on podrem veure que fins ara només hem fet 2 canvis, el primer és la creació del projecte, i el segon és la modificació del nostre arxiu de text.

Ara farem un parell de canvis més i els notificarem per separat a Git, per entendre quin és el flux de treball adequat:

Vam crear un nou arxiu de text i escrivim alguna cosa, i ho guardem.

Li notifiquem a Git el canvi que hem fet:

```
$ git add .
$ git commit –m ‘Creation arxiu.txt and add text’
```

Anem al primer arxiu de text, l'obrim i el modifiquem. El guardem i li notifiquem a Git de la mateixa manera.

Ara si consultem l'índex de canvis (recorda: `$ git log`) veurem que ens apareixen 4 canvis, els 2 que teníem anteriorment, i els 2 que acabem de fer.

Ara si anem al nostre repositori, a GitHub, podem veure que els canvis no s'han actualitzat, fem-ho: `$ git push origin master`

Tornem al repositori i actualitzem la pàgina ... .. de cop tots els canvis s'actualitzen.

Per tant la manera de treballar, es fent les modificacions necessàries en cada un dels arxius del nostre projecte, al nostre ordinador, i anar notificant cada vegada que creem conveninent a Git, per finalment actualitzar els canvis en el repositori (GitHub) sempre que una funció, o un bloc de codi ens funcioni.

Ara bé, que passaria si tenim un projecte a mitges, hem fet alguns push (actualitzem el nostre repositori a GitHub) i treballem en el nostre ordinador en una part del projecte, hem fet commits d'aquestes actualitzacions en Git (però no hem fet push) i volem canviar-ho?

Tenim diverses opcions:

1era—> podem borrar els commits a Git amb la següent ordre

`$ git reset arxiu.ext` (molt poc recomenable)

2ona—> l'altra cosa que podem fer és recuperar la versió dels arxius en un moment anterior i que estan guardats en Git. Realment és un nivell molt més avançat i necessiteriemmolt més temps per poder-lo treballar.

3era—> De la mateixa manera que podem empènyer o actualitzar els nostres canvis al repositori en GitHub, també podem empènyer o actualitzar el nostre projecte en el nostre ordinador amb la següent ordre:

`$ git pull origin master` (És una de les opcions recomenables)

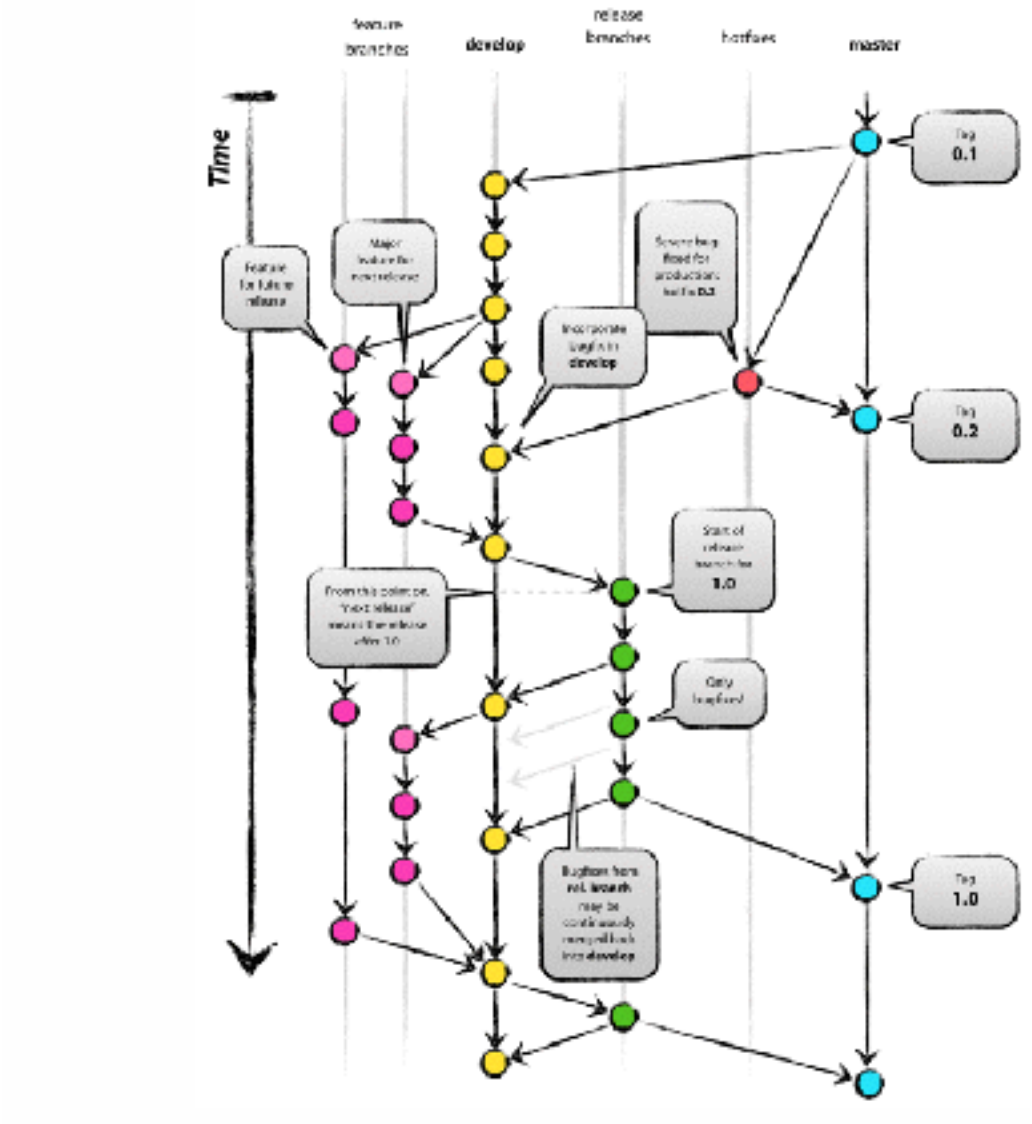
Aquesta ordre el que fa és actualitzar el nostre projecte en el nostre ordinador amb la informació que tenim guardada a GitHub.

4ta—> Podem treballar en branques. És una mica complicat al principi però molt útil si volem desenvolupar diferents versions dels nostres projectes. També serveix per treballar en projectes comunitaris, o en grans projectes individuals. Vegem-ho.

TREBALLAR AMB BRANQUES (BRANCH)

Git i GitHub ens permeten treballar al mateix temps en diferents branques (branch), tantes com vulguem. Aquestes branques representen diferents línies de desenvolupament del projecte, i cadascuna d'elles es bifurca de la branca principal, aquesta es crea per defecte que s'hanomena màster.

Git és molt flexible: pot mantenir múltiples branques actives al mateix temps, i cadascuna d'elles es pot desenvolupar de forma independent fins que desitgem fusionar-les.



Aquest diagrama mostra la complexitat de treballar en branques, alguns fins i tot dirien que és tot un art. Però no és tan complicat com sembla. ;)

Clarifiquem el diaframa:

L'eix I representa el temps, i aquest passa a mesura que avança cap avall.

Cadascuna de les línies verticals és una branca del Git.

Aquest diagrama mostra bàsicament que en una configuració d'un equip de treball sobre un mateix projecte, l'ús de git no és tan senzill com fer-ho en la nostra màquina. Cadascuna de les branques té un propòsit i s'utilitzen per les següents raons:

Màster conté una versió completa del projecte, sense cap tipus d'error, i s'usa per desplegar l'última versió del nostre projecte.

Hotfix (= revisió) aquesta branca serveix per a resoldre un error que hàgim trobat en la branca **master**, quan l'haguem resolt la fusionarem amb la branca **dev** tan ràpid com puguem.

Dev és la còpia principal de la branca **master** i tindrà totes les funcions que ja hem implementat.

Features són un conjunt de branques que es van creant per implementar cadascuna de les noves característiques del nostre projecte. Aquestes branques ens ajuden a fer tants canvis com necessitem per implementar una funció. Un cop implmentada, fusionarem la branca amb **dev**.

Release (= d'alliberament) Aquesta branca conté tots els canvis que hem anat fent durant el desenvolupament del projecte i es desplegarà per intentar fer funcionar totes les funcions alhora, i assegurar-nos que estiguin treballant. Un cop la branca funcioni la podem fusionar amb la **master.**

Per a un petit projecte individual, amb les branques **master** i **dev**, hauríem de tenir suficient, però en un projecte grupal, la millor opció és que cada component de l'equip tingui la seva pròpia branca de treball, com a mínim, a més de les branques **master** i **dev** .

Cada persona treballarà en la branca que porta el seu nom (per exemple), i a mesura que acabi el seu treball o parts del seu treball, l'anirà fusionant amb la branca **dev**.

Quan la branca **dev** funcioni, llavors es fusionarà amb la branca **master.**

Molta feina, oi? Al principi sembla que si, però a la llarga, a més d'evitar problemes en la codificació, facilita la resolució de conflictes entre les diferents parts del codi.

SOLUCIONANT CONFLICTES

Tan les ordres *push* com *pull*, i l'ordre de fusionar branques poden ocasionar conflictes entre diferents parts del codi. Aquests conflictes s'han de solucionar el moment per evitar mals majors, a més Git no ens deixarà fer cap altre commit a menys que no els solucionem ... a Git li agrada l'ordre.

En el cas de *push*, normalment es resol fent canvis, analitzant els conflictes i després fent un nou commit que resol els conflictes.

En el cas de *pull*, la nostra còpia on treballem, immediatament editarà indicant els canvis conflictius, li indicarà el codi antic i el nou, i l'haurem de resoldre, notificar-ho a Git i actualitzar-lo a GitHub.

En cas que tinguem conflictes en fusionar dues branques, els solucionarem com si s'haguessin creat amb un pull. És a dir, la nostra còpia de la branca editarà, triarem amb quin codi ens vam quedar (l'antic o el nou), i quan hàgim resolt tots els conflictes, ho notificarem a Git i el actualitzarem en GitHub.

COM TREBALLAR AMB BRANQUES

Suposem que estem treballant en un projecte, i volem crear una branca nova per desenvolupar una funció addicional nova, ho farem de la seguent manera:

```
$ git branch NewBranch
```

Aquesta ordre ens creara la nova branca, però continuarem estant en la branca master. Per canviar de branca haurem de fer:

```
$ git checkout NewBranch
```

Aixo ens farà canviar de branca, de la master (on erem) a la NewBranch(on volem estar).

 Si volem crear una branca i alhora anar a aquesta branca l'ordre seria la següent: `$ git checkout –b NewBranch`

Ara treballem una mica en aquesta branca, però hem d'anar a classe, haurem de notificar-ho a Git de la mateixa manera que hem fet fins ara, l'únic canvi serè el nom de la branca a la qual ens referirem, es a dit, fare :

```
$ git add.
$ git commit -m 'comentari'
```

I si estem satisfets amb el nostre treball fins ara, podem fins i tot fer:

```
$ git push origin NewBranch
```

 Això actualitzarà el nostre repositori en GitHub, creant una nova branca al repositori i afegint els arxius afegits en aquesta nova branca.

Quan tornem de classe, el professor ha donat indicacions sobre el projecte a lliurar, que volem implmentar avui. Com en la nova branca NewBranch treballavavem en una funció addicional que el professor no havia demanat, volem treballar en la versió a lliurar, per tant, farem:

`$ git checkout master` (canvi de branca)

Ara si consultem els arxius que tenim en aquesta branca (els) veurem que els arxius de la funció addicional han desaparegut !! Com? Perquè?

No cal entrar en pànic, això és degut a que no hem fusionat les branques ;)

Suposem que treballem en la funcionalitat del projecte a lliurar (màster) i al mateix temps treballem en la funcionalitat addicional (NewBranch) i abans de la data de lliurament veiem que tot ens funciona, i per tant volem fusionar les branques, per fer-ho tindrem de donar les ordres següents:

`$ git checkout master` (canvi de branca en el cas que estiguem en NewBranch)

`$ git merge NewBranch`

Aquesta última ordre ens fusionarà la branca master amb la NewBranch, s'afegirà a màster els arxius que estan en NewBranch i màster no té, així com les modificacions en els arxius que les dues branques tenen.

És per això que en fusionar branques podem tenir conflictes de codi.

En aquest cas, els alxius que no presentin conflictes màster els acceptarà sense dir res, però en cas que hagui conflictes de codi, Git ens editarà el projecte i ens anirà marcant un a un els conflictes que puguem tenir, i nosaltres haurem de triar amb quina part de codi ens vam quedar.

 Tot el codi que tingui conflictes que no s'hagin solucionat, Git el marqués com a "no fusionat" (unmerged)

A mesura que hem solucionat tots els conflictes haurem de notificar-ho a Git:

`$ git add.`

Això marcarà cada arxiu modificat, i finalment:

`$ git commit -m 'merge ok'`

Indicarà que els conflictes derivats de la fusió estan solucionats i tindrem Git actualitzat

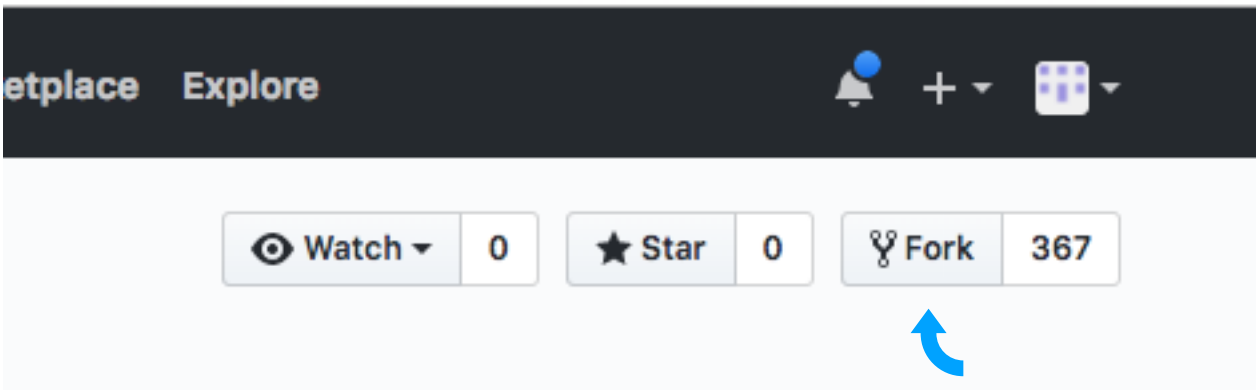
Suposem que hem resolt tots els conflictes, ara només ens queda un últim pas, com màster ja té tots els canvis, la branca NewBranch ja no ens serveix per a res i la podem esborrar, per fer-ho tenim el següent ordre:

`$ git branch -d NewBranch`

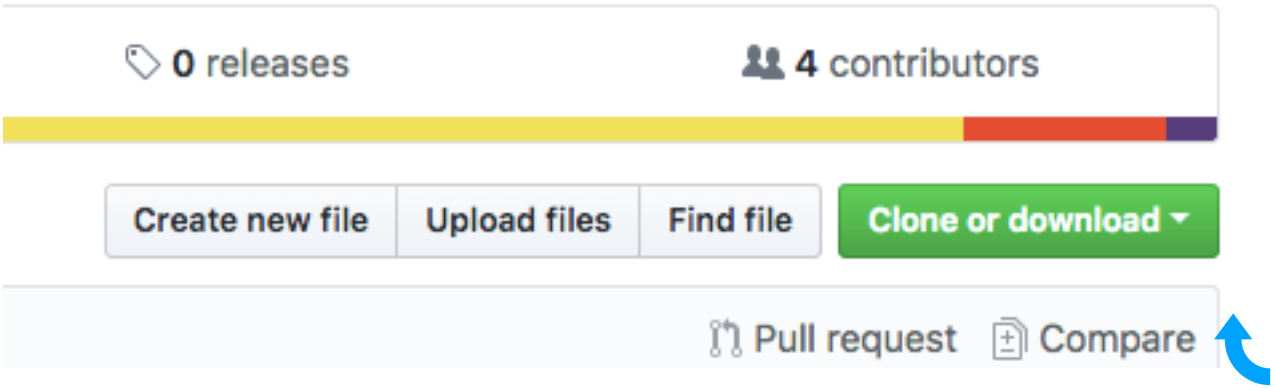
TREBALLAR AMB UN REPOSITORI CLONAT

Ara volem treballar amb un repositori que ja existeix a GitHub, per exemple un projecte amb el que voldríem contribuir. En aquest cas haurem de seguir els següents passos:

- Anirem a la pàgina del repositori (ha de ser un repositori públic) i premerem albotó **"Fork"** del repositori



- El següent pas és obtenir la direcció "SSH o HTTP" del repositori. Clicarem el botó **"Clone or Download"** i copiarem la direcció, la necessitarem per poder clonar el repositori al nostre ordinador.



- Amb el terminal anem a la carpeta o directori on volem clonar el repositori, i un cop dins introduïm la ordre:

```
$ git clone direcció SSH o HTTP copiada
```

- Aquesta ordre fa que Git apunt al repositori que volem clonar, però encara no tenim els arxius del repositori al nostre ordinador, per tenir-los haurem de fer:

```
$ git pull origin master
```

Si el que volem és simplement actualitzar els canvis que s'han fet en el repositori haurem de fer: `$ git pull -r upstream màster`

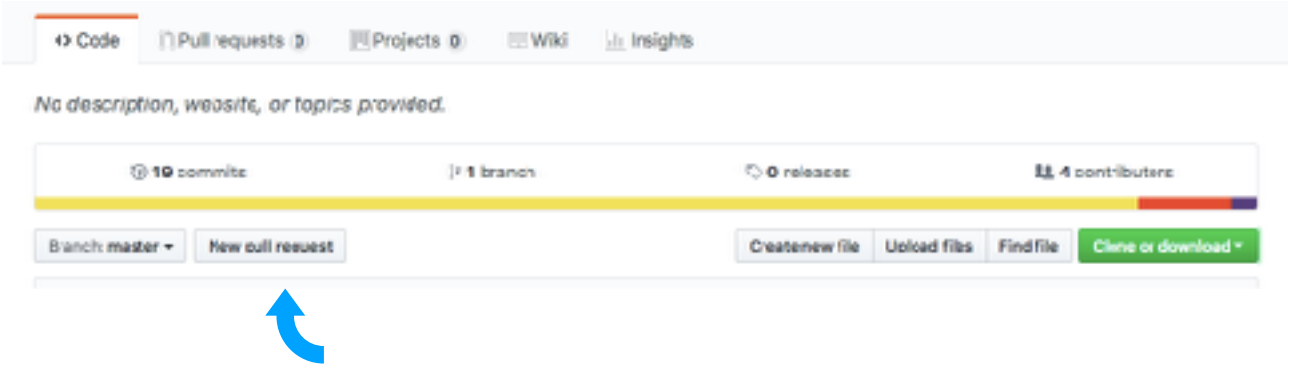
Crearem una branca complementària, que serà a la branca que treballarem:

```
$ git checkout -b ComplemetaryBranch
```

Arribats a aquest punt ja podem fer tots els canvis que vulguem, amb:

```
$ git add.
$ git commit -m 'All changes "
$ git push origin ComplementaryBranch
```

Per actualitzar aquests canvis en el repositori original que hem clonat, haurem de tornar a anar a la pàgina del repositori, i prémer el botó "New pull request"



En fer clic a "New pull Request", ens permetrà indicar els canvis que hem fet, indicar elnostre nom de ususari a la persona, ...

Finalment enviarem la nostra proposta amb el botó "Send pull Request"

Ara només ens queda esperar, al fet que l'usuari propietari del repositori original, el revisi, ho accepti i el fusioni amb la branca master.

Aquesta, és una manera poc convencional, però efectiva de treballar en grup, minimitzant els problemes de múltiples fusions de branques, i de push i pull que es puguin donar.

Hi ha moltes altres maneres de treballar en grup, es pot crear un repositori grupal, on tots els usuaris treballen simultàniament, però només és recomanable en cas que ja es domini el treball individual.

També es pot treballar en grup a través de 2 dipòsits que autoritzen l'altre, és a dir 2 o més usuaris de GitHub, creen un repositori (poden tenir noms diferents), però autoritzen l'altre, és a dir, l'usuari1 pot accedir al repositori del usuari2 i viceversa, però en aquest cas a part necessités un bon coneixement ipractica de Git, també convé mantenir una molt fluida i correcte comunicació entre usuaris, i això no sempre és possible.

Per finalitzar m'agradaria donar suggeriments sobre recursos addicionals per continuar treballant i aprenent més de Git i GitHub.

RECURSOS ADDICIONALS

. Aplicació web per apendre les comandes de Git:
https://try.github.io/levels/1/challenges/1

- Git Pro Book : el llibre més complet sobre git que hi ha, és gratuït i de codi obert , podeu trobar el codi, per descomptat, a github

- Basic Git commands :
https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html

- Git Rebase :
https://help.github.com/articles/about-git-rebase/

- Using git rebase :
https://help.github.com/articles/using-git-rebase-on-the-command-line/