



UNIVERSITAT ROVIRA I VIRGILI

git & GitHub

Aleix Mariné-Tena
Professor substitut DEIM URV
ICIQ data steward
aleix.marine@urv.cat



Objectius

- Saber què és git, què és GitHub, per a que serveix cadascun i com estan relacionats entre ells.
- Aprendre a fer servir git de forma efectiva:
 - Inicialització d'un repositori: init, clone, fetch, fork
 - Visualitzadors gràfics de git
 - Cicle de vida del desenvolupament amb git: add -> commit -> push. .gitignore
 - Fusions i resolució de conflictes: pulls & merges
- Aprendre a fer servir GitHub de forma efectiva:
 - Inicialització de repositoris: Markdown & READMEs, profile displays, llicències, pre-built .gitignores
 - GitHub services: Wikis, Security, Insights
 - PR: Issues, Pull Requests (Projects)
 - Automatitzacions i CI&CD: Actions, Secrets

git?

Software que implementa un sistema de control de versions per a administrar canvis a fitxers i carpetes, especialitzat en fitxers de text.



Què és un sistema de control de versions?

És un software que permet gestionar versions de diferents fitxers i carpetes.

Aquesta gestió consisteix en crear noves versions, modificar-les, fusionar-les...



Altres sistemes de control de versions

Hi ha altres sistemes de control de versions:

- subversion
- mercurial
- CVS

Però aquests:

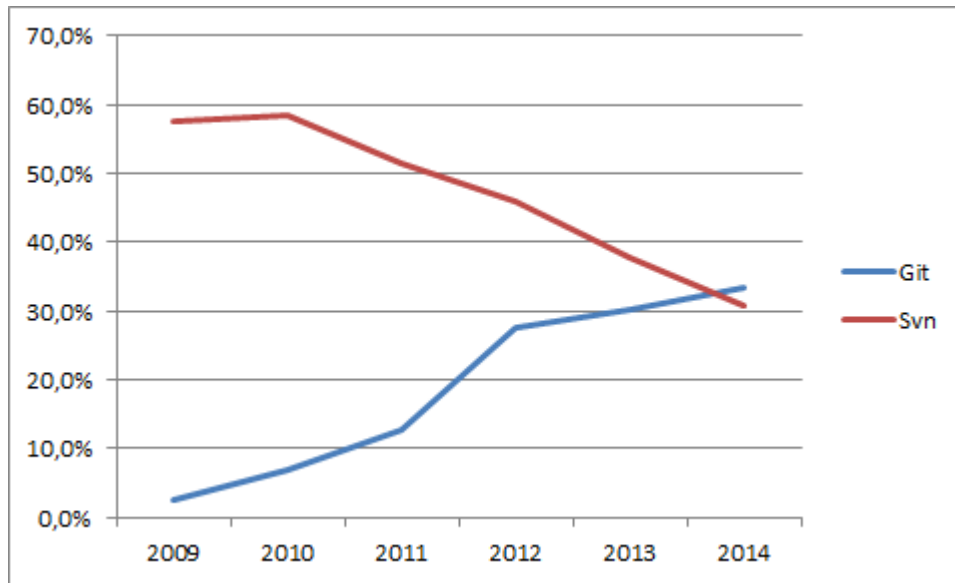
- Enfoc centralitzat, no distribuït
- Software tancat, no software lliure (o obert)
- Software de pagament, no ofereix una “tier” gratis per particulars
- Producte enfocat a empreses

Això ha fet que git sigui el més “popular”
no?

August 2019

- Git: 913,378 repositories (70% of total)
- SVN: 324,629 repositories (25% of total)

Clara tendència cap a l'ús en augment de git, però l'ús de subversion segueix sent significatiu



Què pot fer?

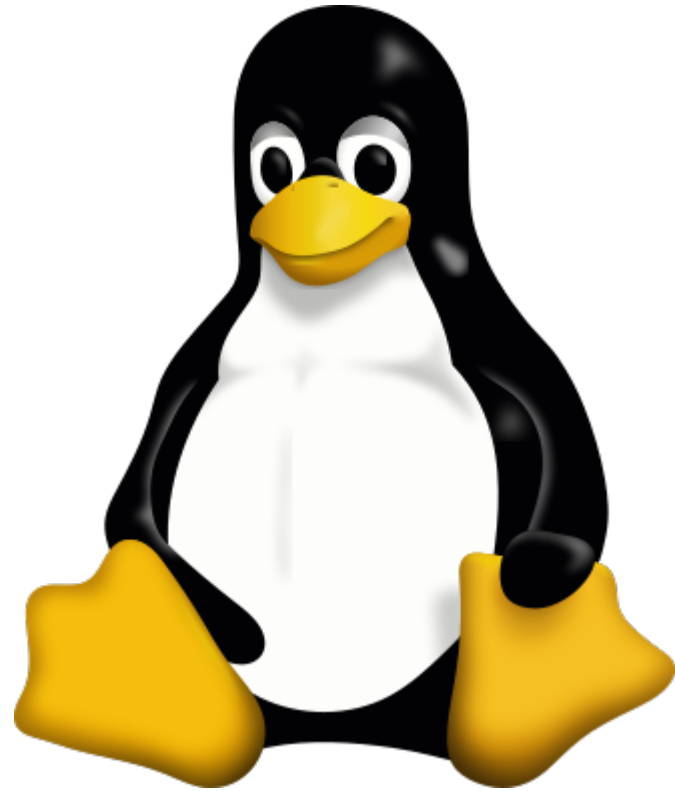
- Suport per a canvis no lineals: Es poden fer canvis concurrents sobre la mateixa versió.
- Gestió distribuïda: Tothom té una còpia completa dels canvis no només el servidor.
- Multiprotocol: Els canvis poden publicarse per protocols com **HTTP(s)**, SSH, FTP o **amb el protocol natiu de git amb SSH** o amb TCP/IP simple.

Si és un software lliure, qui el va crear?



Linus Torvalds

I per què?



GNU/Linux

Per què m'hauria d'interessar aprendre'l?

Per què m'hauria d'interessar aprendre'l?

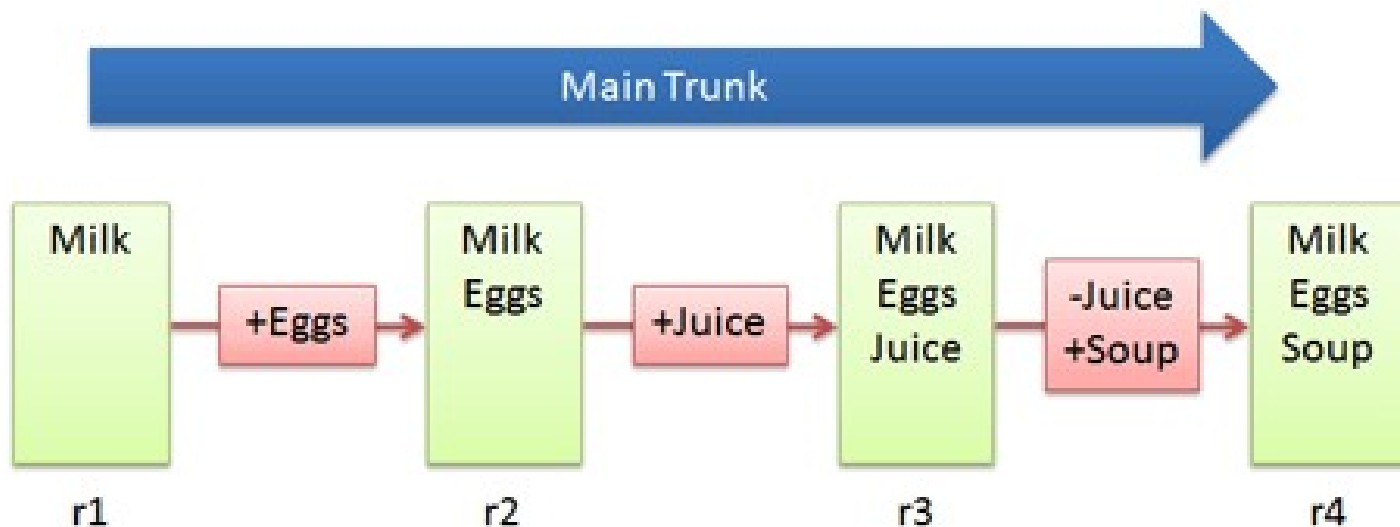
- Si treballes amb software (no només escrivint-lo!) gairebé tots els softwares open-source es troben en repositoris git.
- Et permet manipular fitxers de forma concurrent i eficient amb persones treballant en el mateix projecte.
- Permet fusionar canvis a fitxers de text.
- Al ser distribuït tenim redundància de dades (còpies de seguretat) però sense la ineficiència que sol comportar això.

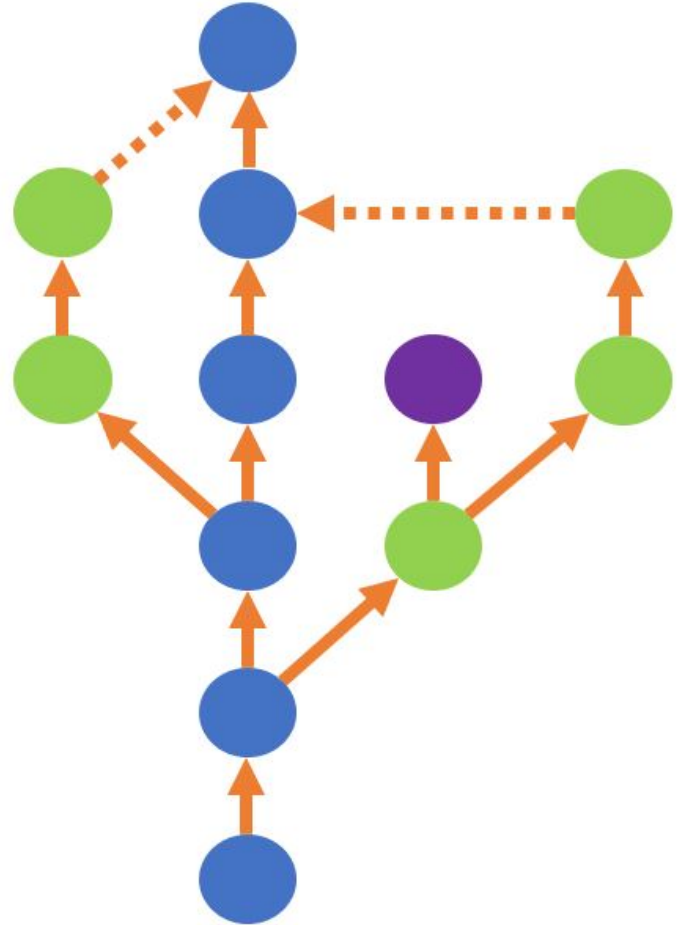
Com funciona?

- Model client-servidor:
 - Aplicació client que farem servir a l'ordinador on estiguem desenvolupant: **git CLI**, IDE plugins, GitHub desktop...
 - Servidor: Servidor privat on-premises, servidor públic (automantigut) a cloud provider, proveïdor de repositoris git (**GitHub**, GitLab, BitBucket...)
- Creem repos al servidor i treballem amb ells al client amb el nostre sistema de fitxers.
- Si fem servir proveïdors de repositoris, les funcionalitats d'aquests augmenten a través dels serveis addicionals que ofereix el proveïdor.

Com funciona? (2)

Basic Diffs





Operacions cicle de vida (treballant sol des d'un sol pc)

Configurar git (git config) (offline) una vegada per màquina

0. “Clonar” un repositori (git clone) (online) una vegada per repo!

Inicialitzar un repo localment (git init) (offline) una vegada per repo!

- 0. Treballar amb branques i desplegar versions al nostre sistema de fitxers (git branch, git checkout) (offline)
 - 1. Fer canvis als fitxers
 - 2. “Afegir” o “Trackejar” fitxers nous (git add / git rm) (offline)
 - 3. Crear una nova versió (git commit) (offline)
 - 4. Penjar una nova versió (git push) (online)

Operacions cicle de vida (treballant amb altres o des de diversos ordinadors)

- Descartar canvis locals (git reset) (offline)
- Crear nova branca (git branch) (offline)
- Canviar de branca o commit (git checkout) (offline)
- Actualitza la informació del servidor (git fetch) (online)
- Fusionar versions (git merge) (offline)
- git fetch seguit de git merge: git pull (online)

Hi ha alguna alternativa als sistemes de control de versió?

Hi ha alguna alternativa als sistemes de control de versió?

Clar. Això no ha existit des de sempre! Pots tornar al mètode clàssic...



Documento
final



Documento
final 2



Documento
finalmente
definitivo



Documento
versión
definitiva



Documento
definitivo
por favor



Por lo
menos



Será el
definitivo



Seguro que
este es el
documento
final



Por favor,
que esta
sea la
versión
definitiva



El único
y definitivo

No, ara de veritat, no hi ha alguna alternativa als sistemes de control de versió?

Podriem fer servir una carpeta compartida en un sistema de fitxers al Cloud (Dropbox, Google Drive, MEGA...) que suporti edició a temps real amb una carpeta local del nostre sistema de fitxers de forma semblant a com ho fa git, però les limitacions són moltes:

- No es suporta edició paral·lela dels fitxers de text, normalment només dels tipus “built-in” del cloud que fem servir (Google documents, sheets, etc.)
- Por haver-hi historial de versions però és lineal. No hi ha branques.
- No hi ha suport explícit per a projectes de programació.
- No es poden fusionar fitxers en funció de les seves versions.
- Només estem al nivell amb tema de backups.

FAQ: Puc perdre les dades si treballo amb git?

No, precisament el contrari: El propòsit principal de git és mantenir la integritat del que es guarda a cada commit.

Si treballes amb git “en local” segueixes sent vulnerable a lo de sempre: Clicks o esborrats accidentals, virus, aplicacions no desitjades, robatoris, errors de hardware o de sistema. És a dir, el que fa “perdre” dades amb git es no fer-lo servir per penjar versions al servidor per fer-ne la còpia de seguretat!

De fet, un dels “problemes” de git, es que esborrar alguna cosa és una tasca enormement tediosa i difícil. MAI HI PENGIS CAP PASSWORD NI DADA PERSONAL O SENSIBLE!

Què és GitHub?



Què és GitHub?

GitHub és un proveïdor de repositoris de git

GitHub ~~The Fellowship~~ at 100% strength



GitHub ~~The Fellowship~~ at 99% strength



Per a que serveixen els proveïdors de git?

- Un proveïdor de repositoris ens permet crear i fer servir repos de git sense haver de gestionar la infraestructura del servidor.
- A més, afegeixen funcionalitats addicionals als repositoris com documentacions, pàgines online, fòrums de discussions, sistemes de proposta discussió i revisió de canvis, informes automàtics de seguretat, Pipelines de CI & CD, integracions amb serveis de tercers...
- Alternativament: Xarxa social de repositoris de git
- Alternativament: CV “pràctic” (portfolio). Prova demostrable de les teves habilitats.

Altres proveïdors de repositoris

Hi ha altres proveïdors de versions a banda de GitHub (closed-source i de Microsoft):

- GitLab (open-source). On-premises és gratuït.
- BitBucket (closed-source, Atlassian)
- Gitea (open-source)

Força semblants entre ells en quant a funcionalitats.

Tot i que el debat sol ser GitHub vs GitLab (estàndards de-facto)



GitHub

vs

GitLab

- Es pot desplegar on-premises amb GitHub Enterprise pagant.
- Més present en projectes personals.
- En general més visibilitat ja que hi ha molts repos de software open-source.
- Més intuïtiu i senzill (però van afegint coses contínuament)
- Closed-source (propietat de Microsoft)
- Controvèrsies per l'ús de codi de repos privats per l'entrenament de la IA copilot

- Es pot desplegar on-premises gratuïtament ja que el codi es open-source.
- Més present en projectes empresarials.
- En general menys visibilitat ja que molts dels repos estan a clouds privats
- Lleugerament més complex
- Open-Source
- Sense controvèrsies per IA a destacar

Exemples pràctics del que es pot fer a GitHub (coses meves)

- Personalitzar el teu perfil de GitHub (miniportfolio)
- Crear el teu portfolio (a la teva propia pàgina gratuïtament) (no funcionarà)
- Escriure documentacions en forma de Wiki (2)
- Gestionar les propostes i idees d'un software i la col·laboració per portar-les a terme.
- Treballar, treballar i treballar
- Conservar projectes vells
- Hostejar fixers
- Curar dades
- Automatitzar, automatitzar i automatitzar (de vegades al 100 %)
- Col·laborar amb els creadors de les teves eines preferides i de vegades queixar-te i obtenir suport.
- I molt, molt, però que molt més...

Exercicis pràctics

Exercici 0: Crear compte de GitHub i instal·lar git

- Anar a github.com, fer click a sign up, i crear un nou compte, preferentment amb el correu de la URV.
- (opcional) afegir 2FA i/o mètodes addicionals d'inici de sessió
- Instal·lar git CLI:
 - Linux: apt update; apt install -y git
 - Mac: brew install git
 - Windows: Descarregar instal·lador, executar i seguir els passos: <https://git-scm.com/downloads/win>
- Jo treballo amb Linux i recomano Linux per a programar (milax?)
- Instal·lar IDE: [VSCode](#), [Pycharm](#), nvim... Recomano Pycharm o altres de JetBrains (teniu llicència amb la uni)

Llicències de software

Serveixen per a “protegir” un software legalment.

Indiquen què es pot fer amb el teu software i baix quins termes es pot fer.

Llicències comuns:

- Copyright: En general no es pot fer res sense permís explícit de l'autor. Hi ha algunes excepcions, però.
- **Llicència GPL, la llicència del software lliure:** Centrada en les llibertats de l'usuari. Principalment es distingeix de les altres perquè obliga a fer servir la mateixa llicència en cas de basar-se en el teu software.
- Llicències open-source: Hi ha més o menys permissives amb respecte a que es pot fer amb un software basat en el teu.
- Creative Commons: No són adequades per a software.

Exercici 1: Inicialitzar repo hello-world


- Un cop loggats a GitHub, fer click al signe “+” per crear un nou repositori.
- Familiaritzar-vos amb la interfície del vostre nou repositori. (destacar Security i Insights)
- Escriure un programa “Hello World” en el vostre llenguatge preferit (preferentment llenguatge compilat o que generi un binari final si us quedeu fins al final de la xerrada per l'exercici 11) al repo clonat i les eines necessàries per treballar amb ell fàcilment. Jo ho faré en C.

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner *

 CODE-URV ▾

Repository name *

hello-world

✔ hello-world is available.

Great repository names are short and memorable. Need inspiration? How about **reimagined-octo-memory** ?

Description (optional)

A simple hello world program with C to showcase the basic capabilities of git and GitHub



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:



Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore


.gitignore template: C ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: GNU General Public License v3.0 ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  master as the default branch. Change the default name in CODE-URV's [settings](#).

Exercici 1: Inicialitzar repo hello-world

En el meu cas

git clone <https://github.com/CODE-URV/hello-world>

En el vostre:

git clone [https://github.com/\[NOM_PERFIL\]/hello-world](https://github.com/[NOM_PERFIL]/hello-world)

Treballar en el codi fins aconseguir versió funcional.

Exercici 2: Crear .gitignore

.gitignore:

Fitxer especial que pot trobar-se als repositoris git. Indica al client de git quins fitxers ha d'ignorar a l'hora de monitoritzar la carpeta del projecte.

S'indiquen els fitxers a ignorar amb el path relatiu des d'aquell .gitignore fins al fitxer a ignorar. També es poden fer servir comodins.

Normalment n'hi ha un a l'arrel del projecte. GitHub ens dona plantilles específiques en funció del llenguatge.

```
# Executables
*.exe
*.out
*.app
*.i*86
*.x86_64
*.hex

# Debug files
*.dSYM/
*.su
*.idb
*.pdb

# Kernel Module Compile Results
*.mod*
*.cmd
.tmp_versions/
modules.order
Module.symvers
Mkfile.old
dkms.conf

.idea/
```

Exercici 2: Crear `.gitignore`

Què hauríem de posar a un `.gitignore`?

hauria de ser: Què hauríem de penjar a un repositori de git?

Codi. Hem d'intentar penjar-hi només codi. Però cada projecte és un món i no sempre podrem penjar només codi...

Exercici 2: Crear .gitignore

A un repo de git NO hi hem de penjar:

- Productes de la compilació del nostre projecte.
- Fitxers autogenerats per l'IDE (no, no m'interessa el teu IDE), compilador, sistema operatiu...: Logs, fitxers de configuració interns, temporals...
- Dades personals, dades sensibles, passwords i secrets

A un repo de git hauríem de procurar no penjar:

- Binaris llibreria o eines. Sempre es pot fer un script que les baixi i gestionar el codi de l'script.
- Dades: El mateix que al cas anterior, però hi ha excepcions si les dades son lleugeres i no canviantes. (imatges) També git LFS.

Exercici 2: Crear .gitignore

A la pràctica, la forma de treballar amb el gitignore és tenir ignorats sempre els fitxers que no es vulguin penjar. Això es pot veure si fem `git status` i hi ha fitxers a la secció `untracked` que realment no volem penjar. Hem d'escriure la regla al `.gitignore` i veure el que el fitxer no desitjat tampoc està a la secció `untracked` al fer un `git status`, i així per a cada fitxer. Amb això podrem fer `git add .` tranquilament.

```
23:04 $ git status
En la rama master
Tu rama está actualizada con 'origin/master'

Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo
    hello-world
```

```
23:06 $ git status
En la rama master
Tu rama está actualizada con 'origin/master'.

Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar
  (usa "git restore <archivo>..." para descar
  trabajo)
    modificados:      .gitignore
```


Exercici 3: Penjar feina hello-world

Cicle de vida del software amb git (treballant sols amb un sol PC):

Fem canvis als fitxers fins aconseguir una versió funcional

git config --global user.email "el teu email" # Un sol cop

git config --global user.name "El teu nom o nom d'usuari" # Un sol cop

Fem els canvis

git add # Afegir fitxers que volem que estiguin a cada versió (un per fitxer)

git status # Comprovem que hem afegit correctament els fitxers

git commit -am "Missatge on expliques quins canvis has fet"

git status # Comprovem que el commit s'ha creat correctament

git push origin master # main # Publiquem el commit

git status # Comprovem que s'ha fet el push

Exercici 4: Afegir README.md en markdown / HTML

Crear / Editar el fitxer README.md a l'arrel del nostre repositori hello-world, per explicar com funciona el nostre software (com a mínim).

Hem d'estar “nets” de canvis:

```
22:22 $ git status
En la rama master
Tu rama está actualizada con 'origin/master'.

nada para hacer commit, el árbol de trabajo está limpio
```

Si no ho estem:

```
22:22 $ git status
En la rama master
Tu rama está actualizada con 'origin/master'.

Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado
  (usa "git restore <archivo>..." para descartar los cambios en el d
trabajo)
    modificados:    README.md

sin cambios agregados al commit (usa "git add" y/o "git commit -a")
```

Exercici 4: Afegir README.md en markdown / HTML

Si tenim canvis bruts vol dir que tenim canvis en local que el servidor no coneix.

Hem de fer add del README.md, commit i push per guardar els canvis o bé fer `git reset --hard` (i esborrar fitxers nous) si volem descartar-los.

Exercici 4: Afegir README.md en markdown / HTML

Markdown: Llenguatge de marcat per a maquetació de pàgines web que representa un subset del HTML amb el propòsit de ser fàcil de llegir en la seva versió de codi font.

Markdown també permet fer servir HTML sense cap constructe especial, permettent-nos accedir a totes les característiques d'HTML.

Extensió .md

GitHub té suport built-in d'aquest format, pel que si visualitzem [un fitxer markdown](#) amb la interfície web el podrem veure renderitzat.

Exercici 4: Afegir README.md en markdown / HTML

Algunes coses de sintaxi:

- Capçaleres amb #. Més # menys importància.
- [text a mostrar](enllaç)
- ![Alt text](URL_or_path_to_image)
- **cursiva**, ****negreta****
- ``codi``

``lenguatge

bloc de codi

```

# Exercici 4: Afegir README.md en markdown / HTML

Recursos:

- [Què hi hauria d'haver a un README?](#)
- [Què hi hauria d'haver a un README \(segons GitHub\)?](#)
- [Tota la sintaxi bàsica de markdown](#)
- [Markdown Cheat Sheet](#)
- [Projects that have an awesome README](#)
- [Best README template](#)

Quan acabem pugem el fitxer i anem al nostre repo amb al navegador a la pàgina de GitHub per a veure el resultat.

També podem veure el resultat al nostre PC si tenim algun plugin o software per a renderitzar markdown.

## Exercici 5: Documentar amb GitHub Issues / GitHub Projects

Suposem que algú (o nosaltres mateixos) volem suggerir una modificació al nostre software hello-world.

Podem escriure aquesta idea en un issue amb markdown:

<https://github.com/CODE-URV/hello-world/issues/new>

Un issue és l'abstracció que fa GitHub d'una tasca a resoldre relacionada amb el nostre projecte. Podem assignar una persona responsable, uns tags, podem discutir aquesta idea, donada per completada, etc.

Cada issue té de forma interna un identificador del tipus #NOMBRE que l'identifica i es pot fer servir en altres issues o punts del projecte per a referenciar-lo fàcilment.

## Exercici 5: Documentar amb GitHub Issues / GitHub Projects

També podem organitzar de forma més complexa els issues per a implementar un sistema SCRUM amb [GitHub Projects](#)



## Exercici 6: Fusionar i resoldre conflictes

Ara simularem que estem col·laborant amb algú altre i crearem un conflicte que resoldrem per a entendre en què consisteixen els conflictes i per què si entenem el que estem fent no hem de tenir-los-hi por.

Tenim el nostre codi font de hello-world. Suposarem que vam penjar una versió que diu “hello-world” sense fer un salt de línia.

Suposem que algú que té accés al nostre repo fa una millora al programa hello-world i afegeix el salt de línia “\n” al final i penja una nova versió a la branca `new-line`. Concurrentment, nosaltres ja havíem vist que faltava el salt de línia, però afegim “\r\n” per a fer-ho multi-plataforma a la branca `master`. Al fer un git merge de la nostra branca i la seva, hi haurà un conflicte, ja que els dos hem modificat la mateixa línia i cal escollir manualment quina modificació es queda a l'arbre. Això es fa fent un commit anomenat commit de fusió o resolució, on editem els fitxers amb el conflicte per a escollir nosaltres quina de les dues modificacions conservar (o “unir” manualment)

# Exercici 6: Fusionar i resoldre conflictes

#Partint de master amb arbre de treball net.

```
git checkout -b new-line
```

# modifiquem hello-world.c i afegim “\n” al final

```
git commit -am “Afegit sal de línea \n a l’script ”
```

```
git push origin new-line # Si es queixa seguim instruccions i afegim --set-upstream
```

```
git checkout master # o main
```

# modifiquem hello-world.c i afegim “\n\r” al final

```
git commit -am “Afegit sal de línea \n\r a l’script ”
```

```
git push origin master # o main
```

```
git pull origin new-line o git merge new-line # conflicte!
```

## Exercici 6: Fusionar i resoldre conflictes

# Resoldre conflicte,

git add . # Amb .gitignore ben configurat

git commit -am "commit de fusió"

git push origin master

Podrem veure els resultats amb:

gitk --all --date-order &

O podem fer servir algun plugin del nostre IDE per a visualitzar l'arbre del git

## Exercici 7: Forks i profile display

Per a personalitzar la vista del nostre perfil podem fer crear el repositori especial “EL NOSTRE NOM D’USUARI”.

Podriem crear-lo desde 0 com hem fet abans, però és més fàcil agafar un perfil ja creat i modificar-lo.

- [Aquí una llista awesome de perfils](#)
- [I aquí una llista awesome de plantilles de perfils](#)

[He trobat aquest que pot ser un bon començament](#)

Procedirem a fer-li un fork (una còpia) que anomenarem igual que el nostre nom d’usuari. Ho clonem al nostre ordinador i modifiquem el README.md per a posar la nostra info.

## Exercici 8: Plantilles i pàgina web amb GitHub Pages

Podem crear la nostra pàgina web (front-end only) allotjada als servidors de GitHub de forma gratuïta. Podem fer-ho una vegada per perfil / organització.

Per a fer-ho hem de crear el repositori especial [EL NOSTRE NOM D'USUARI].github.io

Aquest repositori està vinculat a una pàgina web de domini [EL NOSTRE NOM D'USUARI].github.io

Igual que abans: Podem agafar un simple markdown o HTML per a fer-la desde 0, però també podem fer servir alguna plantilla o la pàgina d'algú altre i modificar-la.

# Exercici 8: Plantilles i pàgina web amb GitHub Pages

Plantilla [Just The Docs](#)

[Click aquí per a fer-la servir](#)

Creem amb ella el repo [EL NOSTRE NOM D'USUARI].github.io

Clonem el repo i modifiquem el contingut i posem les nostres dades.

Potser hem de marcar alguna opció a la pestanya Settings del repo, però en principi, en quan es publiqui el commit hauriem de tenir la pàgina disponible a [https://\[EL NOSTRE NOM D'USUARI\].github.io](https://[EL NOSTRE NOM D'USUARI].github.io)

# Exercici 9: Pull Requests (PR)

La Pull Request (PR) és el procés fonamental que vertebrava la col·laboració a GitHub

Aquest procés permet proposar canvis a un projecte quan no es tenen permisos suficients per a fer els canvis tu mateix. A diferència dels Issues, que serveixen pel mateix, la PR va acompanyada de commits que implementen la idea que es proposa.

De la mateixa manera que amb els Issues, els propietaris poden discutir la implementació, proposar millores, afegir els seus canvis o requerir que el que fa la PR completi tasques addicionals.

El tancament d'una PR implica la fusió (merge) dels canvis proposats a alguna de les branques del projecte.

Una PR sol començar amb el fork del projecte al qual es volen incorporar els canvis, ja que tenim tots els permisos al nostre fork, però no al projecte original.

## Exercici 9: Pull Requests (PR)

Farem servir la PR per a que les persones que vulguin certificat d'assistència s'apuntin en [aquest](#) repositori.

Cal crear un fork, clonar aquest fork, afegir el fitxer assistents/[EL TEU NOM D'USUARI] amb el teu email a dins.



# Exercici 10: Hosting de fitxers

Els fitxers dels repositoris públics de GitHub es troben localitzables amb una URL directa de descarrega.

Podem pujar fitxers que volguem disponibles públicament (com logos que fem servir a firmes corporatives) o fitxers dels que volguem facilitar la descàrrega.

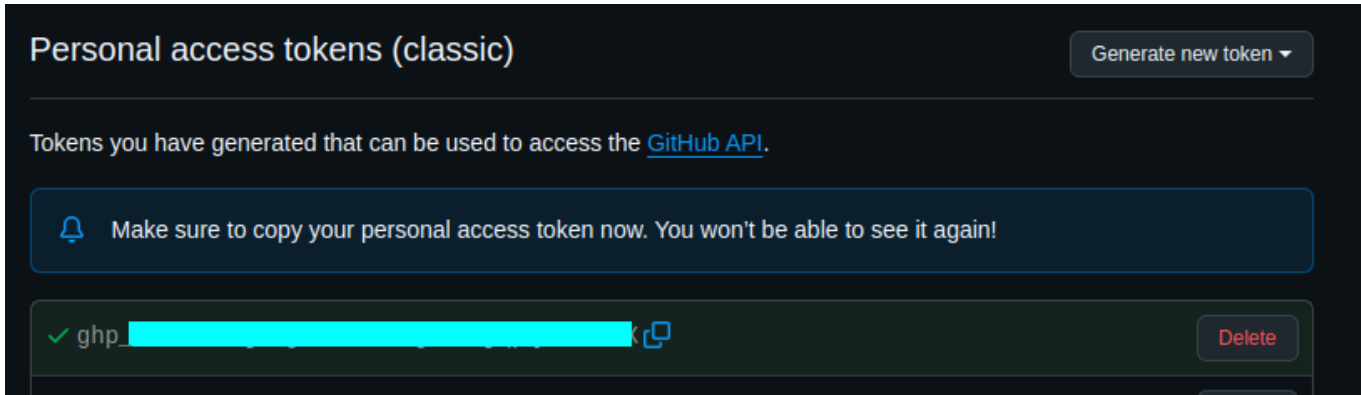
Aquí un exemple: <https://github.com/CODE-URV/hosting>

Al README.md està explicat com funciona ;)

# Exercici 11: Automatitzacions GitHub Actions & Secrets

Tornar al repo hello-world i crear un workflow d'automatització amb GitHub Actions que compili el nostre software i pengi el binari a l'apartat "Releases" del repo.

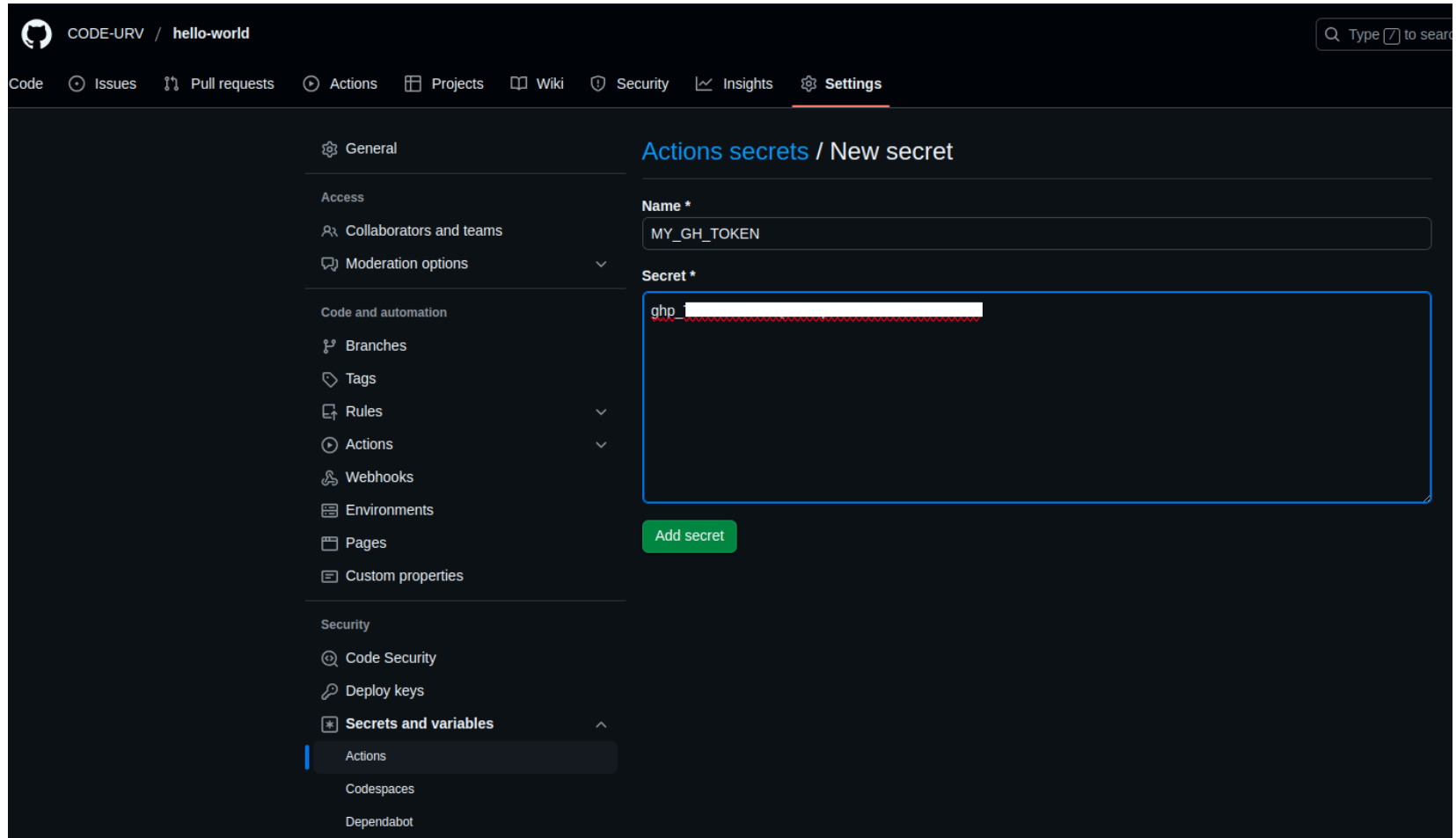
Per a fer-ho cal anar a "Settings" del nostre perfil, "Developer Settings", "Personal Access Tokens" i "Tokens (classic)", "Generate new token". Copiar el password i guardar-lo bé (no el podrem tornar a mirar).



# Exercici 11: Automatitzacions GitHub Actions & Secrets

Tornar al repo hello-world, “Settings”, “Secrets and variables”, “New repository secret”. Enganxar el token i com a nom posar MY\_GH\_TOKEN.

# Exercici 11: Automatitzacions GitHub Actions & Secrets



The screenshot shows the GitHub web interface for the repository **CODE-URV / hello-world**. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The **Settings** tab is active, and the left sidebar shows the **Secrets and variables** section expanded, with **Actions** selected.

The main content area is titled **Actions secrets / New secret**. It contains two input fields:

- Name \***: A text input field containing the text `MY_GH_TOKEN`.
- Secret \***: A large text area for the secret value. It starts with the prefix `ghp_` followed by a redacted portion of the token (indicated by a white box and a red dotted line).

Below the input fields is a green button labeled **Add secret**.

# Exercici 11: Automatitzacions GitHub Actions & Secrets

Ara crearem el fitxer `.github/workflows/ci.yml` a l'arrel del repositori.

[Aquí](#) teniu un exemple funcional del contingut del fitxer.

Afegim el fitxer, fem un commit i el pugem al servidor.

Ara, hauríem de veure a la secció “Actions” del nostre repositori que s'està executant un workflow, quan acabi, si acaba bé, veurem una nova release a la secció releases del nostre repositori hello-world.

## Altres Recursos:

- [Aprende a Branchear en git](#): Entorn interactiu per aprendre git
- [Oh my git](#): Joc de git

Preguntas?



UNIVERSITAT ROVIRA I VIRGILI

# git & GitHub

Aleix Mariné-Tena  
Professor substitut DEIM URV  
ICIQ data steward  
[aleix.marine@urv.cat](mailto:aleix.marine@urv.cat)

