CONTROL DE VERSIÓNS

Introdución

Os sistemas de control de versións son unha categoría de ferramentas de software que axudan a rexistrar os cambios realizados nos ficheiros mantendo un control das modificacións feitas no código.

Un produto de software está normalmente desenvolvido en colaboración por un grupo de desenvolvedores que poden estar situados en diferentes lugares. Cada compoñente do grupo pode ter que facer modificacións no código fonte por iso, un sistema de control de versións axuda ao equipo a xestionar de forma eficiente todos os cambios que se van facendo no código fonte xunto coa información de quen o fixo e cando se produciu a modificación.

No desenvolvemento dun proxecto créase un código fonte principal e cada colaborador crea unha rama para o seu traballo. Só cando o traballo colaborativo estea validado, pódese fusionar co código fonte principal.

Actualmente, na creación de software é necesario dar respostas rápidas a cambios durante o desenvolvemento, sendo habitual repetir varias veces o proceso do ciclo de vida dunha aplicación antes da súa entrega ou publicación debido a distintas modificacións que se poden producir antes de finalizar o proxecto. Para estas situacións é imprescindible a xestión do software con un sistema de control de versións que permite modificar o código en calquera punto da súa liña de desenvolvemento e dar resposta rápida a novas versións (desenvolvemento, probas).

Destacamos as seguintes avantaxes do uso dun sistema de control de versións:

- Optimiza a velocidade de desenvolvemento do proxecto proporcionando unha colaboración eficiente,
- Mellora a produtividade, acelera a entrega de produtos e as habilidades dos empregados a través dunha mellor comunicación e asistencia,
- Reduce as posibilidades de erros e conflitos do desenvolvemento do proxecto mediante a trazabilidade de cada pequeno cambio,
- Permite a colaboración entre distintas persoas en diferentes localizacións xeográficas.
- Cada colaborador xestiona a súa copia de traballo da que se pode seguir o rastro en todo momento.
- Axuda na recuperación en caso de distintos imprevistos.
- Informan sobre quen, que, cando e por que se fixeron cambios.

Conceptos básicos

Un **sistema de control de versións** permite a modificación das distintas partes dun proxecto de software. Una **versión dun proxecto** é o estado actual en que se atopa un produto que se está a desenvolver nun punto do seu ciclo de vida.

A continuación, defínense distintos conceptos que se utilizarán no control de versións:

- **Repositorio**. O lugar onde se almacenan todos as distintas versións dun proxecto que se está a desenvolver por un equipo de persoas.
- **Rama**. Cada persoa implicada no desenvolvemento do proxecto ten a súa copia local de traballo e realiza modificacións sobre unha rama do proxecto.
- **Cambio** (*change*). Prodúcese un cambio cando un programador sube o seu traballo despois de facer modificacións.
- **Revisión** (*revision*). Unha revisión é unha versión do software e permite ao sistema de control de versións controlar as distintas revisións do software realizadas. Normalmente, identifícase con un contador ou tamén con unha etiqueta identificativa. Soe ter asociado metadatos como:
    - Nome da persoa que fixo a revisión.
    - Data e hora na que se almacenan os cambios.
    - Motivo das modificacións da revisión.
    - De que revisión e/ou rama se deriva a revisión.
    - Palabras ou termos asociados a revisión.
- **Confirmar** (*commit*). Un *commit* é unha confirmación de que se realizaron cambios nun arquivo e implica o seu almacenamento no repositorio local.
- **Importar** (import). Cando queremos incluír un proxecto por primeira vez nun repositorio a operación que temos que facer é a de *importar***.**
- **Despregar** (*t*). Ao facer *checkout* créase unha copia local dunha rama do repositorio.
- **Publicar** (*commit ou* check-*in*). Permite actualizar un repositorio co contido local.
- **Sincronizar** (*update*). Actualiza un directorio local de traballo cos cambios realizados no repositorio desde a última vez que se fixo esta operación.
- **Rotular** (*tag*). Ao crear un *tag* creamos unha nova versión do proxecto e lle damos un nome, estas versión etiquetadas poden ser accedidas en calquera momento.
- **Abrir rama** (*branch*). Abrimos unha rama cando se quere bifurcar o proxecto para desenvolver código con unha evolución diferenciada da rama da que partiu.
- **Integración ou fusión** (*merge*). Unha rama pode ser fusionada con calquera outra rama do repositorio. A operación **patch** realiza o mesmo que *merge* pero limítase a modificacións en ficheiros mentres que *merge* permite directorio e ficheiros.
- **Cambios** (*diff*). Mostra os cambios nos ficheiros en calquera momento e desde onde queiramos da liña de tempo da árbore de traballo.

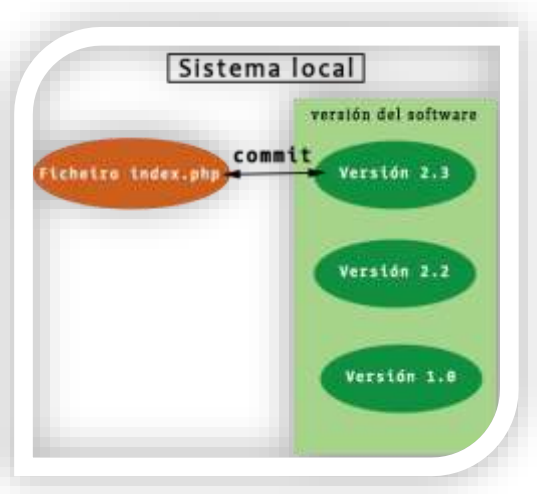
Tipos de sistemas de control de versións

Un sistema de control de versións consta de un repositorio local ou remoto que é onde se atopa todo o código do proxecto, e un cliente que se conecta a ese repositorio.

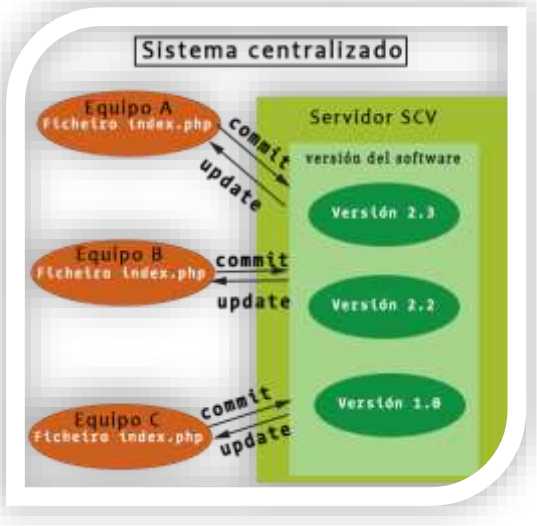
O cliente traballa no seu espazo propio e conectase ao repositorio para enviar as súas actualizacións sobre os arquivos/carpetas do repositorio (*commit*).

Dependendo da arquitectura usada podemos falar de distintos tipos de sistema de control de versións:

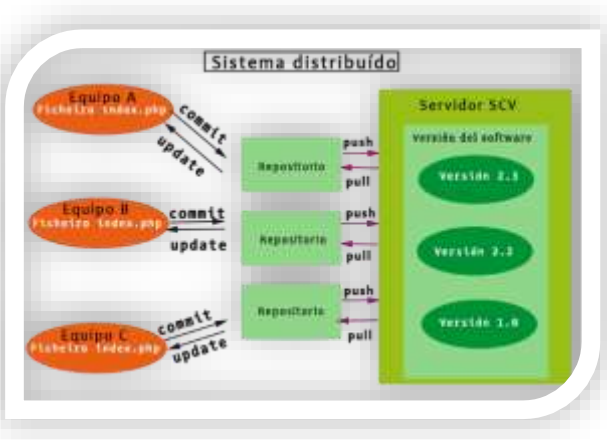
- **Locais**. As copias non se comparten con ninguén, e equivalente a un *backup* local.



- **Centralizados**. Un servidor central contén o repositorio de todo o código e un usuario, normalmente o xefe de proxecto, é o encargado de realizar a xestión do servidor. Todas as operacións que se fagan como ramificacións ou modificación de gran calado necesitan a autorización do encargado da xestión. Esta é a forma de traballo do sistema SCV.

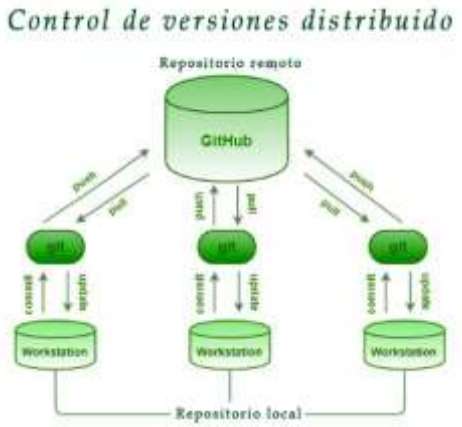


- **Distribuídos**. Non existe un repositorio central, cada equipo de traballo ten o seu repositorio. Os repositorios que existen en cada equipo pódense intercambiar e fusionar. Todos os usuarios teñen a copia de todo o repositorio de forma local. *Git* pode traballar desta forma.



GIT

Git foi creado por Linus Torvals como un sistema de control de versións distribuído gratuíto e de código aberto deseñado para xestionar todo tipo de proxectos de software.



Git facilita o desenvolvemento distribuído de software onde máis dun desenvolvedor pode ter acceso ao código fonte dunha aplicación específica e pode facer modificacións que poden ver todos os participantes no proxecto. Permite ao usuario ter *versións* dun proxecto e retroceder para ver ou desfacer eses cambios.

Git é multiplataforma e os seus repositorios poden acceder a repositorios doutros sistemas de control de versións como SVN.

Git permite aos seus usuarios traballar nunha liña paralela aos ficheiros principais do proxecto. Estas liñas chámanse **ramas**. As ramas en **Git** ofrecen unha función para facer cambios no proxecto sen afectar á versión orixinal. A **rama mestra** dunha versión sempre conterá o código en **produción**. Calquera nova característica pódese probar e traballar na súa rama e tamén pode ser combinada coa rama mestra (*fusión*).

Git permite o traballo local e a sincronización co repositorio remoto no momento que se desexe, este funcionamento permite poder traballar sen estar conectado a rede en todo momento.

Git comproba todos os datos mediante SHA1 para manter a integridade da información en todo momento.

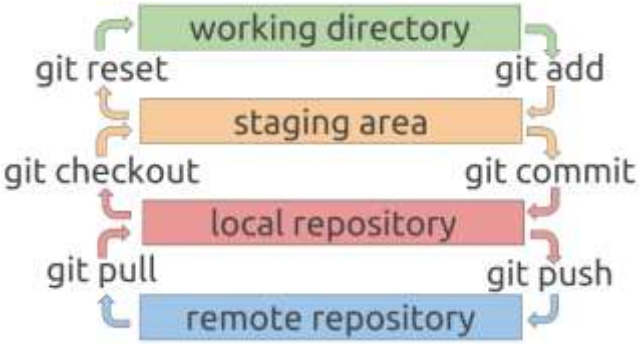
Git, a diferenza doutros sistemas de control de versións, almacena instantáneas dos arquivos e non so ás modificacións que se producen.

O repositorio local de *Git* chámase ***git*** e pode utilizar distintos repositorios remotos, como ***BitBucket***, pero o propio é ***GitHub***.

Zonas de Git

En **Git** existen tres zonas físicas/virtuais de traballo onde se pode atopar os arquivos:

- **Workging directory**: Directorio de traballo onde inicializamos *git*, é a carpeta onde traballamos de forma local. Estes arquivos non teñen seguimento.
- **Staging area**: Esta zona intermedia é onde colocamos os arquivos dos que queremos ter seguimento. Cando os arquivos se atopan nesta zona, *git* indícanos se o arquivo é novo para el, se foi modificado ou si foi borrado por completo.



- **Repository**: O repositorio é a zona onde se confirman os cambios despois da verificación da operación a realizar na zona *Staging area*. Este repositorio pode ser ter dúas zonas local e remoto.

Estados de Git

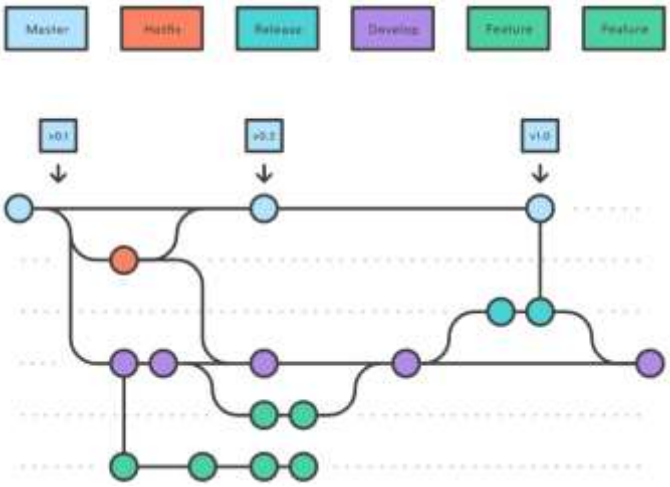
Existen distintos estados nos que poden estar os arquivos cando traballamos con G*it*:

- **untracked**: son os arquivos novos sobre os que *Git* non está a facer ningún seguimento. Isto pode ser tanto porque é un arquivo que agregamos ou ben porque son arquivos excluídos do noso repositorio (binarios, dependencias, etc.)
- **unmodified**: neste estado estarán todos os arquivos cuxo estado non cambiou con respecto ao que está confirmado no repositorio local
- **modified**: dáse pola modificación do contido de arquivos que están nos repositorios *Git*.

- **staged** (*preparado*): neste estado poderemos colocar os arquivos locais que estean listos para confirmar.
- **commited** (*confirmado*): estado no que os arquivos están almacenados de forma segura no repositorio local.

Boas prácticas ao traballar: Git Flow

Git Flow é unha metodoloxía de traballo baseada no división das distintas etapas de produción de software en distintas ramas do repositorio:



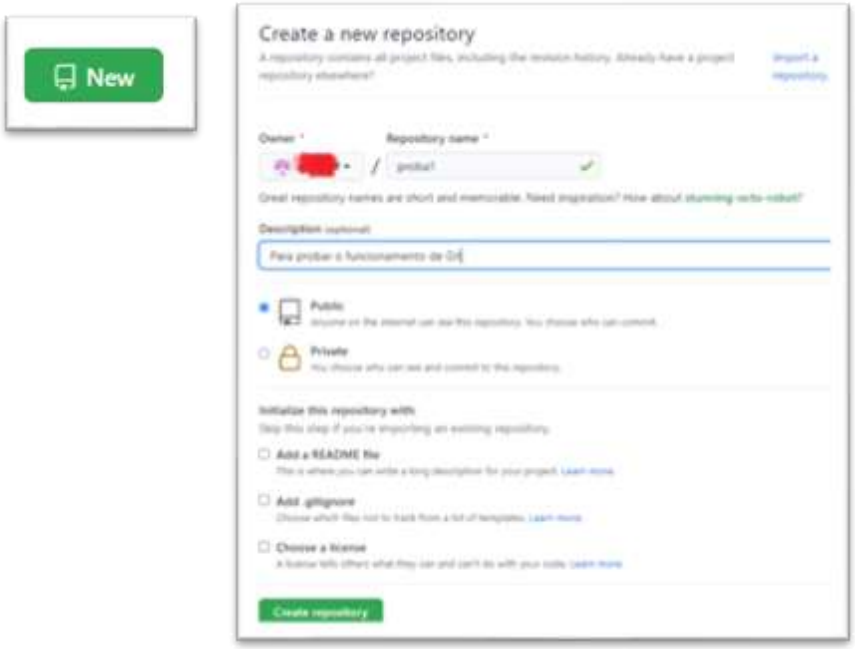
- **master/main**: Na rama *máster* ou *main* atópanse as *releases*[1] do noso proxecto. O código desta rama debe ser estable pois será o que se descarguen os usuarios. Débese etiquetar todas as confirmacións da rama *main* con un nº de versión.
- **develop**: Esta rama parte da última *release*. Nela vanse integrando todas as novas características ata a seguinte *release*.
- **feature-X**: Cada nova mellora ou característica que vaiamos introducir no noso software terá unha rama que conterá o seu desenvolvemento. As ramas de *feature* saen da rama *develop* e unha vez completado o desenvolvemento da mellora, volven integrar en *develop*.
- **release-X**: As ramas de *release* parten da rama *develop* e créanse cando se vai a publicar a seguinte versión do software co obxectivo de arranxar *bugs* e xerar documentación. Unha vez listo para a publicación, intégrase en *master* e etiquétase co número de versión correspondente. Intégranse tamén con *develop*, xa que o seu contido puido cambiar debido a novas melloras.
  Utilizar unha rama específica para preparar publicacións fai posible que un equipo perfeccione a publicación actual mentres outro equipo segue traballando nas funcións para a seguinte publicación.
- **hotfix-X**: Se o noso código contén *bugs* críticos que é necesario parchear de maneira inmediata, é posible crear unha rama *hotfix* a partir da publicación correspondente na rama *master*. Esta rama conterá unicamente os cambios que haxa que realizar para parchear o *bug*. Unha vez arranxado, integrarase en *master* (*merge*), coa súa etiqueta de versión correspondente, tamén se ten que fusionar en *develop*.
  Ter unha liña de desenvolvemento específica para a corrección de erros permite que o teu equipo aborde as incidencias sen interromper o resto do fluxo de traballo nin esperar ao seguinte ciclo de publicación

[1] Versión testada e final dun proxecto

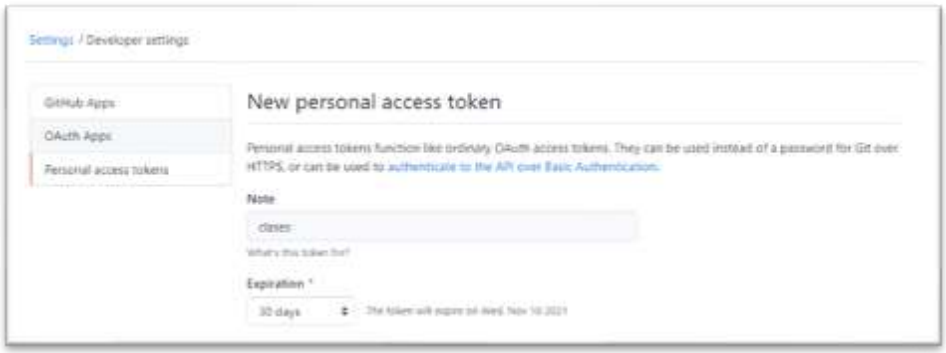
Titorial básico de Git en Ubuntu

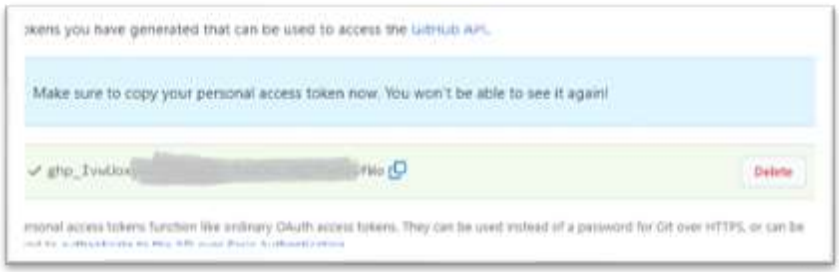
CREAR UN REPOSITORIO EN GITHUB

1. Creamos un repositorio en **GitHub** chamado **proba1**



2. Crear un **token** para poder acceder desde o repositorio local. Información





3. Temos que copiar o **token** para usar como contrasinal á hora de facer ***push*** desde o repositorio **git** local.

CREAR UN REPOSITORIO LOCAL GIT

1. Para instalar **Git** no equipo local:

```
# apt-get install git
```

   Procedemos a instalación no noso equipo.

2. Para actualizar á última versión estable de **Git**.

```
# add-apt-repository ppa:git-core/ppa # apt update; apt install git
```

   Procedemos a actualización no noso equipo.

3. Podemos coñecer a versión de **Git** instalada.

```
# git --version
```

   Consultamos a versión instalada no noso equipo.

```
docente@serv:~$ git --version
git version 2.25.1
```

4. En primeiro lugar, é necesario definir cal vai ser a carpeta onde se vai a localizar o repositorio local.
   No noso exercicio, a carpeta onde se vai localizar o repositorio é ***/home/****usuario****/mirepositorio***

   Activamos a carpeta do repositorio

```
docente@serv:~$ cd mirepositorio
docente@serv:~/mirepositorio$
```

5. Para crear un novo repositorio **Git** na carpeta activa:

```
# git init
```

   Ao crear un repositorio local nunha carpeta, crease unha carpeta oculta chamada **`.git`**

   No noso exercicio, entramos na carpeta mireposgit--itorio e creamos un novo repositorio

```
docente@servidor-DNS:~/mirepositorio$ git init
Initialized empty Git repository in /home/docente/mirepositorio/.git/
```

Comprobamos que a carpeta **mirepositorio** é una repositorio de **Git**

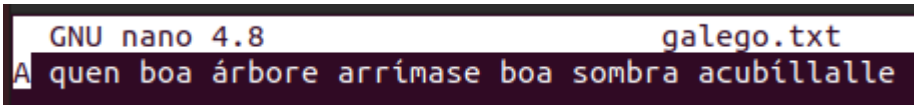
```
docente@servidor-DNS:~/mirepositorio$ ls -a
.  ..  .git
```

Si se borra a carpeta **.git**, perderase o historial do proxecto

6. No caso de existir xa un repositorio **Git** para o proxecto, temos que introducir o comando **`git clone`** ***`url do repositorio`***. Por exemplo:

```
# git clone https://mi.website/git-repository
```

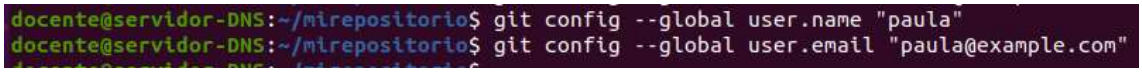
7. Imos crear unha a carpeta **/home/usuario/mirepositorio/traducciones** e creamos dentro dela un arquivo chamado **galego.txt** que conteña o seguinte texto:



8. Configuramos a nosa conta en **Git** aportando os datos seguintes:

    a. nome de usuario: **git config --global user.name** *"nomeUsuario"*
    b. email de usuario: **git config --global user.email** *"emailUsuario"*

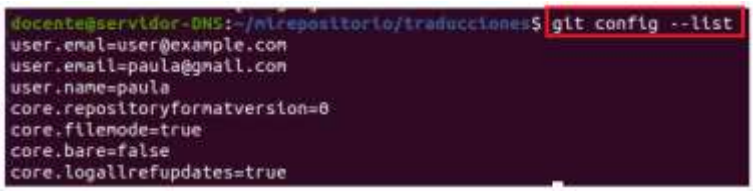
    No vos exercicio tedes que poñer o voso nome e xa que ese nome de usuario asociarase coa túa actividade posterior de Git. O email debe ser o mesmo que usaches no perfil de GitHub.



9. Podes consultar a configuración de **Git**.

```
# git config --list
```

    Consulta a túa configuración de Git.



10. Para coñecer os cambios que non están actualizados no repositorio utilizamos:

```
# git status
```

    Preguntámoslle a Git cales foron os cambios feitos no repositorio (hai unha nova carpeta).

11. Para que teña en conta os cambios feitos, dirémoslle que os engada ao repositorio co comando:

```
# git add .
```

    Engadimos os cambios feitos

E preguntamos de novo polo estado dos cambios pendentes:

```
docente@servidor-DNS:~/mirepositorio$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   traducciones/galego.txt
```

12. Pódense etiquetar os cambios para apuntar a natureza da modificación, para elo utilizamos o comando

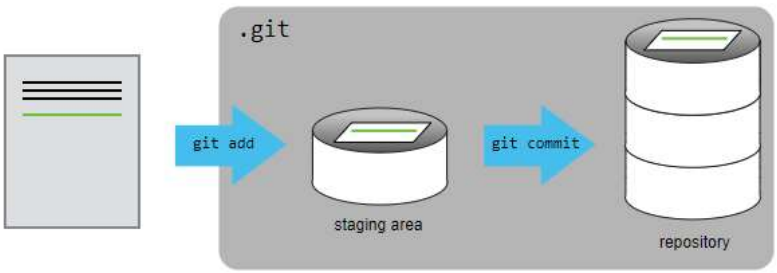
```
# git commit -m "comentario"
```

No noso exemplo:

```
docente@servidor-DNS:~/mirepositorio$ git commit -m "engadimos 1ª de galego"
[master (root-commit) 1328c66] engadimos 1ª de galego
 1 file changed, 1 insertion(+)
 create mode 100644 traducciones/galego.txt
```

Desta forma xa lle dixemos a **Git** que temos modificacións no nosos proxecto, pero aínda estes non saíron da nosa máquina, só están aplicados en local.

Ata aquí a operación que temos feita pódese representar coa seguinte figura, tendo en conta que o repositorio é o local.



13. Todo repositorio crea de forma automática a rama ***master*** , como podemos ver ao consultar cal é a rama activa co seguinte comando:

```
# git branch
```

```
docente@servidor-DNS:~/Privado$ git branch
* master
```

14. Consulta o enlace para coñecer as causas polas que en **GitHub** a rama principal xa non se chama **master** senón que se chama ***main***. Debido ao cambio feito en **GitHub** temos que mover toda a rama **master** a unha rama nova chamada ***main*** antes de continuar o traballo

```
# git branch -M main
```

A necesidade deste comando é temporal.

15. Temos que ter creado en **GitHub** un repositorio remoto coa URL https://github.com/estherff/proba1. O que imos facer é relacionar esta URL con un identificador (***origen***).

```
# git remote add etiqueta URL_do_repositorio_GitHub
```

No noso exemplo:

```
docente@servidor-DNS:~/Privado$ git remote add origen https://github.com/estherff/proba1
```

16. Podemos consultar a etiqueta de referencia do repositorio remoto.

```
# git remote
```

No noso exemplo:

```
docente@servidor-DNS:~/Privado$ git remote
origen
```

17. Para subir as carpetas/arquivos sobre os que fixemos ***commit*** temos que utilizar o comando

```
# git push -u etiqueta_URL_do_repositorio_GitHub rama
```

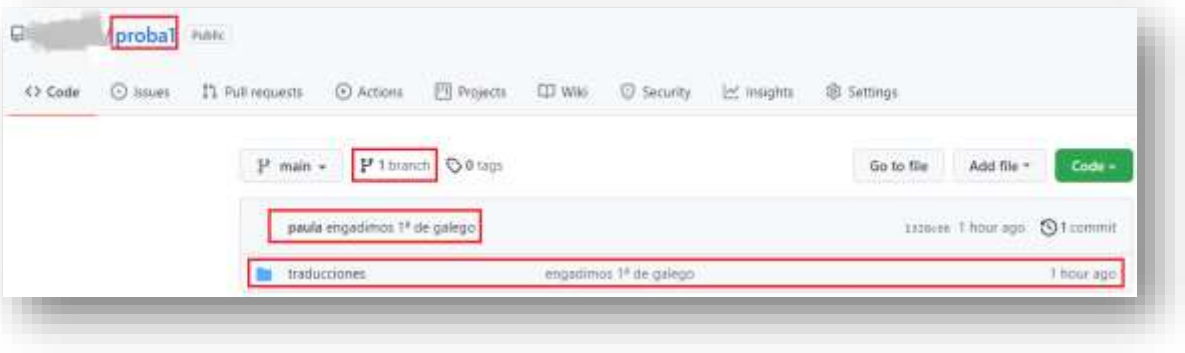
O parámetro **-u** vincula de forma permanente o repositorio local co repositorio remoto e permite usar ***git pull*** (*sincronización do repositorio local co remoto*) sen ningún argumento.

No noso exercicio, subimos os cambios ao repositorio de **GitHub**. Nesta operación teremos que indicar o nome de usuario e como contrasinal introducir un ***token*** xerado desde **GitHub**.

```
docente@servidor-DNS:~/mirepositorio/.git$ git push -u origen main
Username for 'https://github.com': estherff
Password for 'https://estherff@github.com':
Enumerating objects: 4, feito.
Counting objects: 100% (4/4), feito.
Compressing objects: 100% (2/2), feito.
Writing objects: 100% (4/4), 316 bytes | 316.00 KiB/s, feito.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/estherff/proba1.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origen'.
```

En lugar de poñer a etiqueta ***origen*** pódese poñer a URL ***https://github.com/estherff/proba1***

Comprobamos en **GitHub** que a carpeta e os arquivos foron incluídos no repositorio.



18. Comprobamos o estado despois de facer *commit* e *push*.

```
docente@servidor-DNS:~/mirepositorio$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

19. Modificar o arquivo **galego.txt** para que o seu contido coincida co da imaxe.

```
A quen boa árbore arrímase boa sombra acubillalle
Non por moito madrugar, amence mais cedo
```

20. Consulta o status do repositorio para confirmar que **Git** detectou os cambios realizados, que aínda non están actualizados en **GitHub**.

```
docente@serv:~/mirepositorio$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working direct
        modificado: traducciones/galego.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

21. De novo, fai os pasos 10 e 11 e 13 adaptando o comentario ao facer *commit* a nova situación. Confirma as actualización realizadas en GitHub.

22. Para ver un historial dos cambios feitos no repositorio podemos utilizar o comando

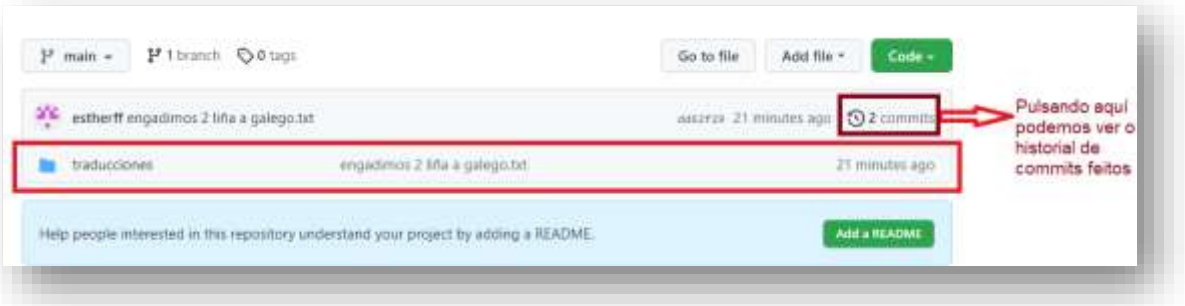
```
# git log
```

No noso exercicio:

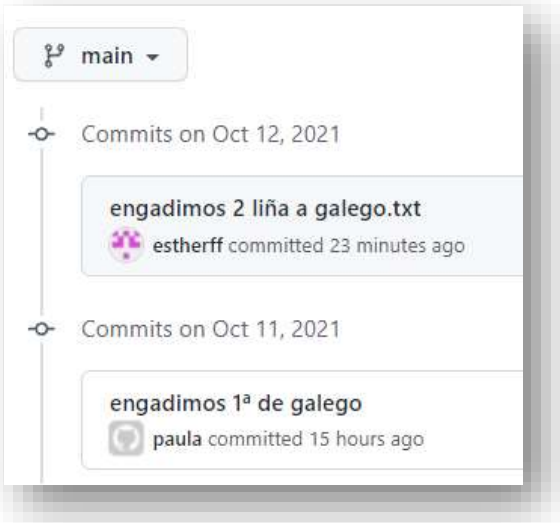
```
docente@serv:~/mirepositorio$ git log
commit dd62f29bc48f0ba82d958da8eede23f3c2f3c941 (HEAD -> main)
Author: estherff <estherff.edu@gmail.com>
Date:   Tue Oct 12 11:33:29 2021 +0200

    engadimos 2 liña a galego.txt

commit 1328c66c32dd06aaa431e86d3478f026d52e564c
Author: paula <paula@example.com>
Date:   Mon Oct 11 20:47:14 2021 +0200

    engadimos 1ª de galego
```

Fixen os *commits* dando distintos nomes de usuario desde o mesmo repositorio local, por iso aparecen dous nomes distintos, no voso exercicio apareceranvos os usuarios que indicástedes ao facer *push.*

23. Pasamos a ver o contido de **GitHub**



e o seu historial de *commits* feitos



Axuda de Git. Para obter axuda dun comando da seguinte forma:

```
$ git comando -h
$ git comando --help
```

Aviso: *GitHub* no autoriza el uso de contrasinais desde **git** a non ser que teñamos activada a autenticación en dous pasos.
`Settings > Acount security > Enable two-factor authentication.`

Tamén podemos crear un **token** desde GitHub que utilizaremos como contrasinal en Git.

Información

Referencias

www-geeksforgeeks-org

https://david-estevez.gitbooks.io/the-git-the-bad-and-the-ugly/content/es/buenas-practicas-al-trabajar-con-git.html

Traballo derivado por Marta Rey López con Licenza CC: BY-NC-SA

https://swcarpentry.github.io/git-novice-es/02-setup/index.html