Git

Git

Git é un sistema de control de versións distribuído, libre e de código aberto.

Foi creado por Linus Torvalds en 2005. Orixinalmente foi deseñado para que varias persoas traballasen simultaneamente no desenvolvemento do **kernel de Linux**.

Actualmente utilízase en proxectos reais de desenvolvemento de software nos que traballan múltiples persoas en paralelo. Nestes proxectos é necesario un sistema de control de versións, como Git, para asegurar que non haxa conflitos de código.

O propósito de Git é levar o rexistro dos cambios nos arquivos e coordinar o traballo que varias persoas realizan sobre ficheiros compartidos.

Ademais, os requisitos dun proxecto cambian a miúdo, polo que un sistema de control de versións permite volver a unha versión máis antiga do código, en caso de que sexa necesario.

Repositorios Git

Un repositorio é un espazo de almacenamento que contén múltiples ficheiros.

Hai dúas formas de almacenar repositorios:

- online
- offline (nun servidor local).

Os tres servizos máis populares para almacenar repositorios online son:

- GitHub
- GitLab
- BitBucket

Creación dun repositorio remoto con GitHub

GitHub é unha plataforma que permite crear repositorios remotos.

O IES San Clemente forma parte del programa GitHub Campus que dá acceso gratuíto a ferramentas útiles de desenvolvemento de software. O alumnado ten que rexistrarse para obter acceso ao paquete de ferramentas de desenvolvemento de software para estudantes.

Exercicio: crea unha conta en GitHub usando o correo electrónico do IES San Clemente.
NOTA: Usar como nome de usuario o do IES San Clemente

Unha vez creada a conta, aparecerá a páxina para obter acceso ao paquete de ferramentas de desenvolvemento. É probable que sexa necesario escanear a matrícula para xustificar ser estudante do IES San Clemente.

Exercicio:

- Crea un novo repositorio: páxina principal -> Repositories -> New.
- Ponlle o nome "**2022-modulo-login**", (no meu caso sería **2022-DWCC-cparis** ou **2022-DI-cparis**). Engade unha descrición e determina o nivel de visibilidade do repositorio.
- Inicializa o repositorio cun arquivo **README**, o que permitirá clonar inmediatamente o repositorio.

Creación dun repositorio remoto con GitLab

GitLab é unha plataforma que permite crear repositorios remotos.

Exercicio: crea unha conta en GitLab usando o correo electrónico do IES San Clemente. Para iso inicia sesión con Google dende a páxina de rexistro de GitLab.

Unha vez creada a conta imos establecer o contrasinal, para poder interactuar con GitLab a través da liña de comandos. Para iso debemos acceder ao menú Editar perfil -> contrasinal.

Tamén é posible modificar o idioma da interface web. Para iso hai que acceder ao menú Editar perfil -> Preferencias -> Localización.

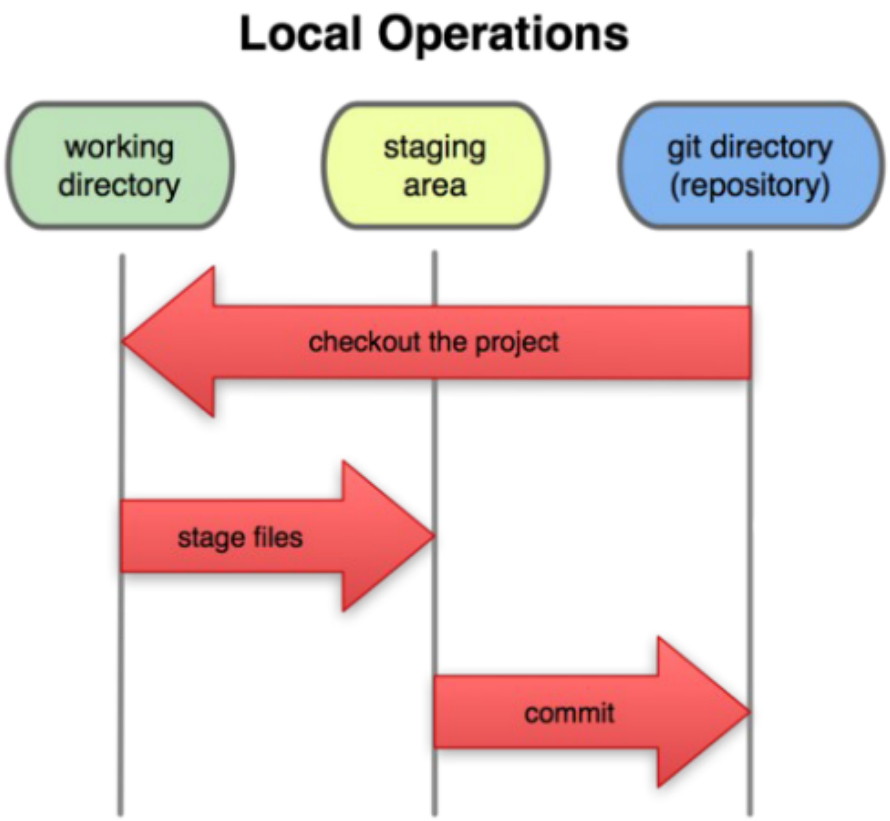
Exercicio:

- Crea un novo repositorio, que GitLab denomina "crear un proxecto en branco".
- Ponlle o nome "**2022-modulo-login**", (no meu caso sería **2022-DWCC-cparis** ou **2022-DI-cparis**). Engade unha descrición e determina o nivel de visibilidade do repositorio.
- Inicializa o repositorio cun arquivo **README**, o que permitirá clonar inmediatamente o repositorio.

Funcionamento de Git

Git proporciona tres espazos de traballo:

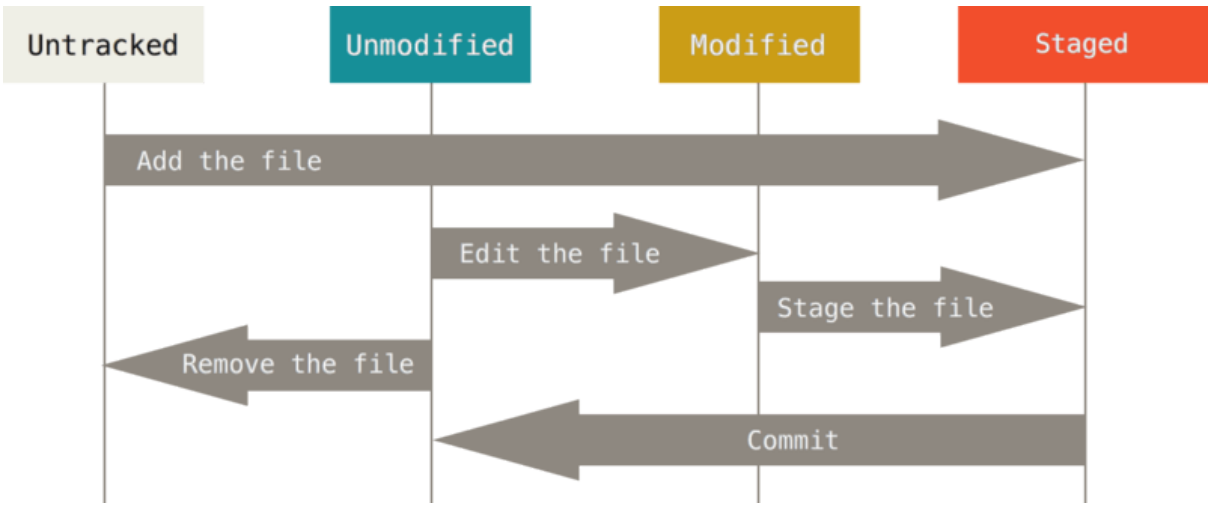
- O **directorio de traballo** (working directory) é o directorio no equipo local que contén a versión dos ficheiros cos que se está traballando actualmente.
- A **área de preparación** (*staging area*), tamén denominada **índice** (*INDEX*), é un área intermedia na que están os arquivos listos para actualizar no repositorio no seguinte commit.
- O directorio ou **repositorio** é onde Git almacena os metadatos coa instantánea dunha versión do proxecto. Contén o historial de versións de todos os arquivos.



Todo repositorio ten unha **rama principal** (master/main branch) que almacena os cambios con marcas temporais. A medida que se use Git, poden crearse novas ramas.

En Git, os arquivos poden estar nun dos seguintes estados:

- **Untracked**: o ficheiro non se engadiu a ningún repositorio.
- **Non modificado**: o ficheiro está sen modificar.
- **Modificado**: arquivo modificado e sen confirmar.
- **Preparado (*staged*)**: ficheiro marcado como preparado para enviarse ao repositorio.
- **Confirmado (*commited*)**: ficheiro almacenado de forma segura no repositorio.



Instalación de Git

Para poder traballar con repositorios, ben de forma local ou remota, hai que instalar Git na máquina física. Para iso hai que descargar da páxina oficial a versión adecuada para o sistema operativo.

Esta instalación, permitirá traballar e interactuar con repositorios usando a liña de comandos.

Configurar a contorna de traballo local

Git trae unha ferramenta chamada **git config** que permite obter e establecer variables de configuración que controlan o aspecto e funcionamento de Git.

A información de configuración pode almacenarse:

- **De forma global no directorio /etc/gitconfig:** contén a configuración de git no sistema que é común a todas as persoas usuarias. Configúrase usando o parámetro **--system.** Son necesarios privilexios administrativos para cambiar esta configuración.
- **De forma persoal no directorio persoal do usuario** (~/.gitconfig ou ~/.config/git/config). Arquivo de configuración específico para un usuario. Para que Git lea e escriba este arquivo úsase o parámetro **--global**
- **Individualmente, no directorio do repositorio** (ficheiro config no directorio de Git: .git/config). É un arquivo específico para o repositorio actual. Configúranse para o repositorio actual sen necesidade de indicar ningún parámetro específico.

Cada nivel sobrescribe os valores do nivel anterior, polo que os valores de **.git/config** teñen prioridade sobre os de **~/.gitconfig**

NOTA: En sistemas Windows, Git busca o arquivo **.gitconfig** no directorio $HOME (normalmente en c:\Users\$User). Tamén busca o arquivo …/etc/gitconfig (ruta relativa ao directorio de instalación de Git).

Antes de comezar a traballar con repositorios, deben configurarse algunhas variables, entre elas o nome de usuario e correo electrónico, que se rexistrarán por cada operación realizada. Se non se configuran, Git intenta descubrilas automaticamente en función do nome de usuario (login) e do nome da máquina (hostname), aínda que non se asegura que a configuración sexa correcta. Para evitar erros recoméndase establecelas manualmente.

Poden configurarse manualmente cos seguintes comandos:

git config --global user.name "nome"
git config --global user.email "correoElectronico"

Comprobar que están correctamente configuradas co comando:

git config -l

Cando algún comando de git necesita abrir un editor, abre o editor Nano, que é o editor por defecto que usa Git. É posible configurar un editor de texto diferente. Ligazón con exemplos de configuración de diferentes editores.

Configuración Visual Studio Code como editor por defecto de git:

NOTA: asegurarse de que está instalado VS Code no sistema: **"code --version"**.

- Configurar VS Code como editor por defecto:

  git config --global core.editor "code --wait --new-window"

- Configurar VS Code como a ferramenta por defecto para diff e merge:

  git config --global diff.tool vscode
  git config --global difftool.vscode.cmd 'code --wait --diff $LOCAL $REMOTE'
  git config --global difftool.prompt false
  git config --global merge.tool vscode
  git config --global mergetool.vscode.cmd 'code --wait $MERGED'
  git config --global mergetool.prompt false

Tamén é posible configurar o símbolo do sistema para que mostre o nome da rama actual. Para iso, hai que engadir ao ficheiro ~/.bash_profile o seguinte código:

```
parse_git_branch() {
    git branch 2> /dev/null | sed -e '/^[^*]/d' -e 's/* \(.*\)/(\1)/'
}
export PS1="\[\e[91m\]\$(parse_git_branch)\[\e[00m\] \w $ "
```

Conexión ao repositorio por SSH

Git permite dous tipos de acceso a un repositorio remoto: HTTPS ou SSH.

Cando se usa HTTPS, se o repositorio non é de escritura pública, sempre que se faga unha operación de escritura haberá que autenticarse, o que pode ser molesto e pouco práctico.

As URLs de SSH proporcionan acceso a un repositorio Git mediante SSH, un protocolo seguro. Para usar estas URLs haberá que xerar un par de chaves SSH no equipo e agregar a chave pública á conta de GitHub.

Con SSH é posible engadir unha chave pública ssh ao perfil da persoa usuaria para que cada vez que se envíe unha solicitude de escritura no repositorio utilizando este protocolo,

non sexa necesario indicar as credenciais de forma manual, xa que para a autenticación se utilizará o arquivo coa chave privada.

Hai dous tipos de chaves: ED25519 e RSA.

O libro Practical Cryptography With Go suxire que as chaves ED25519 son máis seguras que as RSA.

Antes de xerar as chaves, pode comprobarse se hai chaves SSH existentes. O seguinte comando lista os ficheiros no directorio .ssh, se existe:

ls -al ~/.ssh

Por defecto, os nomes dos arquivos de chaves públicas son **id_rsa.pub**, **id_ecdsa.pub** ou **id_ed25519.pub.**

Pode usarse unha chave existente ou xerarse unha nova.

Para xerar as claves ED25519 usarase o seguinte comando:

ssh-keygen -t ed25519 -C "your_email@example.com"

Se pide o nome dun arquivo onde gardar a chave, pulsar Intro para establecer o nome por defecto.

Tamén pedirá unha frase de contrasinal segura.

A opción -C con un comentario como un correo electrónico é unha forma opcional de etiquetar as chaves SSH.

O comando anterior crea as chaves ED25519 no directorio **.ssh** da carpeta persoal do usuario.

Os arquivos xerados son:

- **id_ed25519**: chave privada. Esta nunca debe saír do noso computador.
- **id_ed25519.pub**: chave pública. Esta é a chave que debemos publicar no servidor para permitir a autenticación segura.

Máis información sobre a xeración de chaves.

Engadir a chave SSH á conta de GitHub:

- Copiar o contido do ficheiro coa **chave pública** no portapapeis
- Iniciar sesión en GitHub.
- Seleccionar o avatar na esquina superior dereita -> Settings
- Pulsar SSH and GPG keys
- Pulsar "**New SSH Key**"
    - No campo "Título" engadir un nome descritivo.
    - No campo "Key type" seleccionar "Authentication Key".

- Pegar o contido da chave pública no campo “**Key**”.
- Pulsar o botón **“Add SSH key”**.

Para comprobar que todo funciona correctamente, executar o seguinte comando:

ssh -T git@github.com

Copiar a chave pública ao perfil do GitLab.

- Copiar o contido do ficheiro coa **chave pública** no portapapeis
- Iniciar sesión en GitLab
- Seleccionar o avatar na esquina superior dereita -> Editar perfil
- Pulsar en chaves SSH.
- Pegar o contido da chave pública na caixa de texto.
- Asignarlle un título descritivo
- Opcionalmente pode engadirse unha data de caducidade.
- Pulsar o botón engadir.

Para comprobar que todo funciona correctamente, executar o seguinte comando:

ssh -T git@gitlab.com

A primeira vez que se conecta a GitLab por SSH, debe verificarse a autenticidade o host ao que nos conectamos. Contestando que si, engádese GitLab.com na lista de hosts verificados.

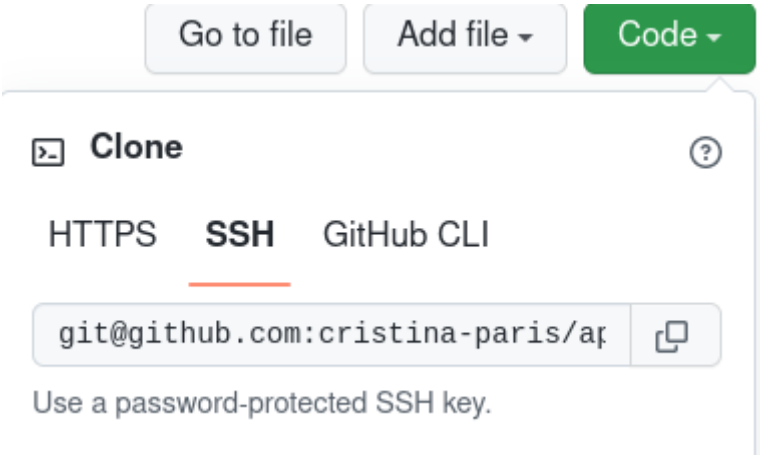
Máis información: Generación de una nueva clave SSH y adición al agente SSH

Clonación dun repositorio remoto

Unha vez creado o repositorio remoto, para poder traballar en local cos ficheiros que almacena, necesitamos facer unha copia no equipo local. Para logralo, hai que clonar o repositorio.

A clonación dun repositorio remoto é unha operación que descarga os arquivos remotos no equipo local.

Para facer a clonación é necesario saber a dirección do repositorio. Para iso pulsamos no botón **Code** do repositorio de GitHub e copiamos a dirección HTTPS ou SSH.



Para clonar o repositorio no equipo local executarase o seguinte comando:

git clone <enderezo HTTPS ou SSH> <directorio>

NOTA: para poder clonar por SSH hai que ter configurada a conexión ao repositorio e as xerar as chaves.

NOTA: GitHub deshabilitou a autenticación mediante contrasinais en agosto de 2021 para que se usen métodos de autenticación máis seguros como a creación dun token de acceso persoal.

O comando git clone crea unha copia dun repositorio no equipo local. Este repositorio almacénase nun directorio do equipo físico na ruta actual. Se se omite <directorio> o nome do directorio creado será igual ao nome do repositorio remoto. Se se indica un directorio, o repositorio almacenarase dentro do directorio especificado (debe estar baleiro).

O repositorio local terá un subdirectorio chamado **.git** con toda a información de trazabilidade dos arquivos.

Operacións no repositorio local

Cando se traballa no repositorio local e se parte dende un commit previo, o fluxo de traballo soe ser:

Cambios en local -> engadir ficheiros -> commit local -> push remoto

A continuación detállanse as principais operacións a realizar nun repositorio local, xunto co comando usado:

- **Comprobar o estado do repositorio**: o seguinte comando mostra un resumo dos estado do repositorio.

  git status

```
/media/MV-Linux/repositorios/demo$ git status
En la rama main
Tu rama está actualizada con 'origin/main'.

nada para hacer commit, el árbol de trabajo está limpio
/media/MV-Linux/repositorios/demo$
```

  Observar que a imaxe indica que:

  - a rama activa é a rama **main** -> “En la rama main”
  - a rama local está actualizada/sincronizada con “origin/main” que fai referencia á rama main do repositorio en GitHub/GitLab.
  - a árbore/directorio de traballo está limpa, non hai cambios.

  O comando **git clone** establece automaticamente unha relación do repositorio local co repositorio web e asígnalle a esta relación o nome **origin.**

- **Engadir ficheiros**: se se crean novos ficheiros nun repositorio, eses ficheiros estarán en estado “untracked” ata que se notifique a git.

  A seguinte imaxe mostra a creación dun ficheiro. Posteriormente, a consulta do estado do repositorio indica que hai un ficheiro sen seguimento.

```
/media/MV-Linux/repositorios/demo$ echo "Proba" >> proba.txt
/media/MV-Linux/repositorios/demo$ ls
proba.txt  README.md
/media/MV-Linux/repositorios/demo$ git status
En la rama main
Tu rama está actualizada con 'origin/main'.

Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
        proba.txt

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa
"git add" para hacerles seguimiento)
/media/MV-Linux/repositorios/demo$
```

Para que os ficheiros sen seguimento pasen á *staging area* debe usarse o comando **git add <nomeFicheiro>**

```
/media/MV-Linux/repositorios/demo$ git add proba.txt
/media/MV-Linux/repositorios/demo$ git status
En la rama main
Tu rama está actualizada con 'origin/main'.

Cambios a ser confirmados:
  (usa "git restore --staged <archivo>..." para sacar del área de stage)
        nuevos archivos: proba.txt

/media/MV-Linux/repositorios/demo$
```

Observar na imaxe que o estado actual do repositorio indica que hai un ficheiro na área de preparación ou índice (*staging area*) mediante a mensaxe “Cambios a ser confirmados”. Nesta área estarán os ficheiros preparados para o commit (confirmación).

git add . (engade todos os ficheiros sen seguimento do directorio actual e dos seus subdirectorios á *staging area*)

NOTA: O comando **git add** pode ser executado múltiples veces antes dun commit.

- **Confirmar os cambios:** despois de engadir ficheiros á área de preparación pode facerse un commit:

  git commit -m “mensaxe”

  Recoméndase escribir unha mensaxe explicativa dos cambios realizados.

```
/media/MV-Linux/repositorios/demo$ git commit -m "Engadir primeiro ficheiro"
[main 8f9bb07] Engadir primeiro ficheiro
 1 file changed, 1 insertion(+)
 create mode 100644 proba.txt
/media/MV-Linux/repositorios/demo$
```

A imaxe anterior amosa que se fixo un commit da rama “main” identificado co SHA-1 8f9bb07, que serve para identificar de forma unívoca o commit.

Pode volver a comprobarse o estado do repositorio:

```
/media/MV-Linux/repositorios/demo$ git status
En la rama main
Tu rama está adelantada a 'origin/main' por 1 commit.
  (usa "git push" para publicar tus commits locales)

nada para hacer commit, el árbol de trabajo está limpio
/media/MV-Linux/repositorios/demo$
```

A imaxe anterior indica que o directorio de traballo está limpo. Observar ademais que di que a “rama está adiantada … por 1 commit”. É dicir, non hai cambios

pendentes no repositorio local, sen embargo este non está sincronizado co remoto. Pode comprobarse accedendo cun navegador ao repositorio vía web e observarase que o último ficheiro creado non está na web.

- **Enviar os cambios ao repositorio remoto:**

  git push [nomeRemoto] [nomeRama]

  git push origin main

  - **origin**: é o nome por defecto do repositorio remoto. Este nome establécese automaticamente ao clonar o repositorio. O nome pode descubrirse co comando **git remote -v**
  - **main**: é o nome da rama á que se queren enviar os cambios.

  Se non houbo erros, o repositorio remoto debería estar sincronizado co local. Pode comprobarse nun navegador web que os ficheiros novos xa aparecen no servidor remoto.

  Cando se quere subir o traballo ao servidor remoto hai que usar o comando **push**. Por defecto, Git non sincroniza automaticamente as ramas locais cos repositorios remotos, senón que hai que facer un push expresamente.

  Este comando só funcionará se se clonou dun servidor sobre o que se ten permisos de escritura e se ninguén máis enviou datos polo medio. Se alguén máis clona o mesmo repositorio e envía información, o novo envío será rexeitado. Haberá que traer o traballo e combinalo antes de poder enviar datos ao servidor. Por iso ás veces é aconsellable facer un **git pull** antes de facer un **git push**.

- **Actualizar o repositorio local coas últimas modificacións producidas no repositorio remoto:**

  git pull -all

  git pull origin main

  Este comando incorpora os cambios do repositorio remoto na rama actual. En realidade, primeiro executa un **git fetch** e despois **git merge** para incorporar os cambios remotos na rama actual.

O comando **git fetch** é un comando non destrutivo que simplemente actualiza as referencias entre os repositorios remoto e local e o comando **git merge** fusiona os cambios.

NOTA: recoméndase facer un pull antes de facer un push para asegurarse que en local están as últimas modificacións do repositorio.

- **Mostrar o rexistro dos cambios realizados no repositorio:**

  git log

  Mostra o historial dos cambios feitos no repositorio dende o último ata o máis antigo.

  O comando **git log --oneline --decorate --graph** mostra o historial de confirmacións, indicando tamén onde están os apuntadores das ramas. Os cambios aparecen por orde cronolóxica, sendo o primeiro que aparece o máis recente.

  - **oneline**: mostra a información nunha única liña.
  - **decorate**: mostra etiquetas ou información adicional dos commits
  - **graph**: mostra un gráfico das ramas

```
(main) /media/MV-Linux/repositorios/starter-web $ git log --oneline --graph --decorate
* 1a8dbd3 (HEAD -> main) renaming
* 4b24ff1 (origin/main) Segundo
* 053d8df Adding more ipsum text
* 930912f My first commit
```

Na imaxe anterior obsérvase que a rama main local está apuntando ao último commit, mentres que a rama main do repositorio remoto apunta ao penúltimo commit.

- **Mostrar información dun commit específico:**

  git show <obxecto>

```
(main) /media/MV-Linux/repositorios/starter-web $ git show 1a8dbd3
commit 1a8dbd397579673ac20a4cd3efd2c2288101eb58 (HEAD -> main)
Author: Cristina París <cparis@iessanclemente.net>
Date:   Fri Aug 12 22:28:00 2022 +0200

    renaming

diff --git a/level1/listado.txt b/level1/lista.txt
similarity index 100%
rename from level1/listado.txt
rename to level1/lista.txt
(main) /media/MV-Linux/repositorios/starter-web $
```

A imaxe anterior mostra a información do commit identificado co código 1a8dbd3.

- **Ignorar arquivos:** cando se quere configurar o repositorio para que non rastrexe certos arquivos crearase un ficheiro **.gitignore** co listado de patróns que coincida co nome de ficheiros a ignorar.

  O formato deste ficheiro contén en cada liña unha expresión que pode ser:

  - Un ficheiro específico: **ficheiro.txt**
  - un patrón: ***.class**
  - unha carpeta ou directorio: **directorio/**

  Exemplo:

```
# ignora os arquivos que teñan extensión .class
*.class
# ignora todos os arquivos do directorio build/
build/
```

  GitHub mantén unha extensa lista de arquivos .gitignore adecuados para decenas de proxectos e linguaxes en https://github.com/github/gitignore.

  Por exemplo, no caso de traballar en proxectos Java sería interesante incluír no .gitignore os ficheiros apropiados de Java e tamén os do IDE a usar:

  - Modelo Java.gitignore
  - Modelo NetBeans.gitignore

- **Manter as credenciais de acceso en caché:**

  Cada vez que se fai un envío ao repositorio remoto é necesario introducir o nome de usuario e contrasinal para validar o envío.

  O seguinte comando permite asignar un tempo de permanencia das credenciais de acceso na caché local para evitar que se pida o usuario e contrasinal de cada vez:

  git config --global credential.helper 'cache --timeout=3600'

  O comando anterior establece unha permanencia en caché de 3600 segundos (1 hora) das credenciais de acceso Por defecto este valor está establecido a 300 segundos (5 minutos).

- **Comando git diff:**

  Úsase para ver os cambios que se produciron no repositorio. Poden compararse commits, ramas, ficheiros, etc.

  Exemplo: edita un ficheiro do repositorio e engade ou elimina contido. Executa despois o comando:

  git diff

O comando anterior mostra as liñas exactas que foron engadidas e eliminadas.

No apartado Comparacións móstrase unha ferramenta gráfica para ver as diferencias entre versións de arquivos.

- **Repositorios remotos:**

  É posible ter varios repositorios remotos conectados a un mesmo repositorio local. Para ver os repositorios remotos que hai configurados, debe executarse o comando **git remote -v**. Isto mostrará o nome de cada un dos repositorios remotos especificados. Se se clonou o repositorio, debería mostrarse, polo menos, o repositorio ***origin*** que é o nome que por defecto asigna Git ao servidor dende o que se clonou.

  Para engadir un repositorio remoto úsase o comando:

  git remote add [nome] [url]

- **Alias:** cando un comando é moi longo e necesita ser escrito moitas veces resultará útil crear un alias. Alias de Git.

Creación dun repositorio local

Ás veces é necesario crear un repositorio local, traballar nel localmente e subir os ficheiros a un repositorio remoto.

Pasos a realizar:

- Calquera directorio local pode ser convertido en repositorio co comando

  git init

  Isto creará un directorio **.git** no directorio de traballo que servirá para almacenar a información das diferentes versións dos arquivos.

- Poden crearse ficheiros no repositorio local e engadilos co comando **git add .**
- Unha vez engadidos os ficheiros á área de preparación, hai que confirmar os cambios cun commit

  git commit -m “mensaxe”

Para subir este repositorio local a un remoto, necesitamos crear un repositorio remoto **SEN crear o ficheiro README.**

- Copiar a dirección HTTPS de GitLab.
- Asociar o repositorio local co remoto

  git remote add origin [dirección HTTPS]

- Comprobar que o comando anterior funcionou executando

  git remote -v

  Se o resultado é “origin”, o comando executouse correctamente.

- **Enviar os cambios ao repositorio remoto:**

  git push origin main

  origin: é o nome por defecto dos repositorios remotos. O nome pode descubrirse co comando **git remote**

  main: é o nome da rama á que se queren enviar os cambios.

Comparacións

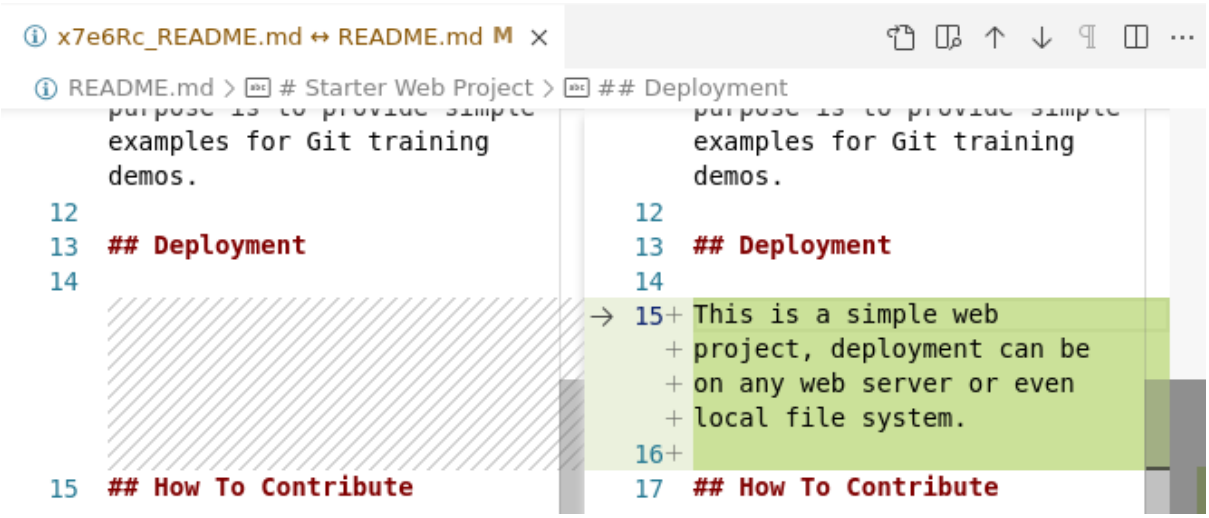
O comando **git status** informa se houbo cambios no directorio de traballo actual, mais non permite saber cales foron os cambios. Para iso utilízanse o comando **git diff**. Tamén é posible utilizar unha ferramenta gráfica para ver os cambios de forma máis sinxela visualmente.

Nestes exemplos vaise usar o Visual Studio Code como editor por defecto, configurado no apartado Configurar a contorna de traballo local.

- Comparar as diferencias entre o directorio de traballo e o *staging area*:

  git difftool

  Este comando abre o editor por defecto con dúas pantallas: na esquerda está a versión do ficheiro na *staging area* e á dereita está o ficheiro no directorio de traballo actual. O editor mostra visualmente as diferencias entre os ficheiros.



- Tamén é posible comparar as diferencias entre o directorio actual e o último commit co comando:

  git difftool HEAD

  Igual que antes, abrirase un editor con dúas ventás: na esquerda estará a versión do ficheiro no último commit e á dereita a versión no directorio actual.

- Comparar o que está na *staging area* co último commit:

  git difftool --staged HEAD

- Para comparar dous commits hai que indicar os seus identificadores, que se poden obter con **git log --oneline**

  Exemplos:

  git difftool HEAD^ HEAD ## Compara o penúltimo commit co último

  A ferramenta visual mostrará á esquerda o penúltimo commit e á dereita o último.

  git difftool d68506f 4b24ff1

  A ferramenta visual mostrará á esquerda o commit con id **d68506f** e á dereita o commit con id **4b24ff1**.

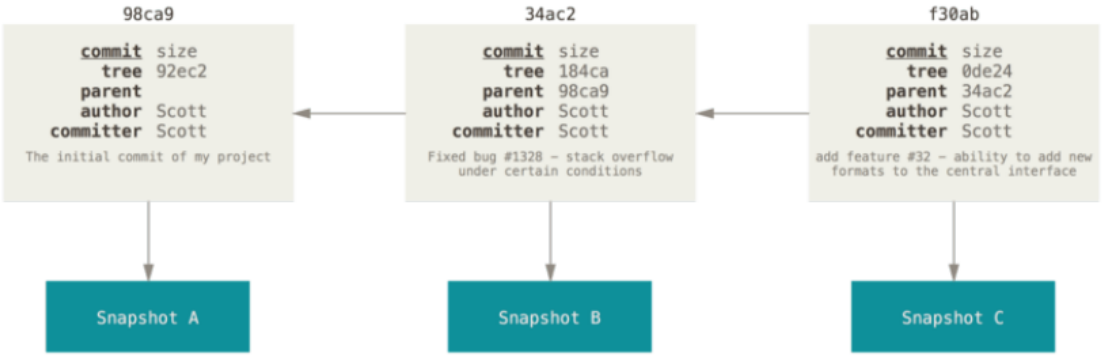
- Comparación entre ramas. O seguinte exemplo compara a rama master de GitHub coa rama master local

  git difftool origin/master master

NOTA: cando hai varios arquivos cambiados pode limitarse a busca mediante o parámetro **-- <path>**. Exemplo: **git difftool -- README.md**

Ramas

En cada confirmación de cambios (commit) Git almacena unha instantánea do traballo. Dita instantánea contén, ademais, metadatos con información da autoría dos cambio, a mensaxe explicativa e un ou varios apuntadores ás confirmacións (commit) que sexan pais directos desta (un pai, no caso de confirmación normal, e múltiples pais no caso de realizar unha fusión de varias ramas).



Unha rama Git é simplemente un apuntador móbil apuntando a unha desas confirmacións. A rama por defecto de Git chámase **master** ou **main**.

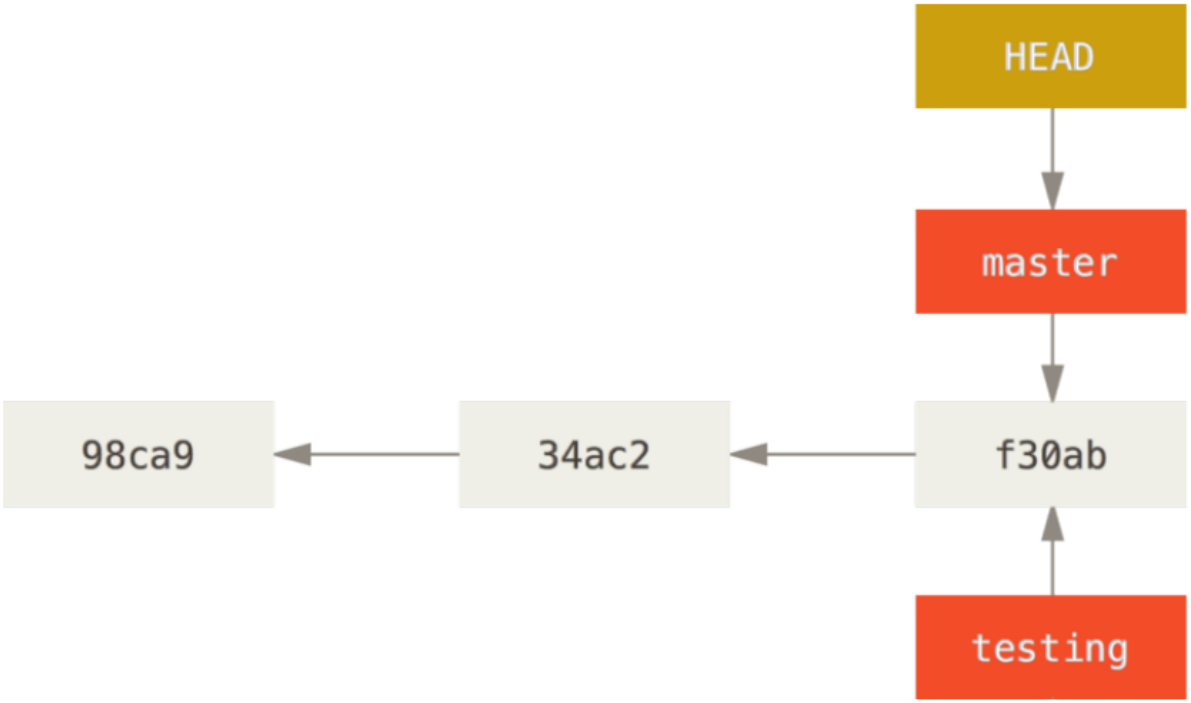
As ramas utilízanse para crear unha nova versión de código, por exemplo para facer probas. É común que exista unha rama en produción e unha rama en desenvolvemento.

As ramas permiten traballar con diferentes versións dun mesmo arquivo e, cando se queira, pode facerse unha fusión para incorporar os cambios realizados.

¿E como sabe Git en que rama estamos en cada momento?. Git utiliza un apuntador especial denominado HEAD que apunta ao último commit da rama actual.

NOTA: cando se fai un salto nun repositorio a un commit antigo, que non é o último, éntrase nun estado chamado "**detached head state**", que indica que o punteiro HEAD non contén o identificador do último commit da rama actual.

Na seguinte imaxe observamos un repositorio con dúas ramas (master e testing). O apuntador HEAD apunta á rama **master**, polo que o directorio de traballo correspóndese co desta rama.



Comandos para traballar con ramas:

- Visualizar as ramas existentes:

  git branch -a

  Este comando mostra as ramas locais e remotas e aparece un ***** ao lado da rama actual activa.

```
(main) /media/MV-Linux/repositorios/starter-web $ git branch -a
* main
  remotes/origin/main
(main) /media/MV-Linux/repositorios/starter-web $
```

- Creación dunha nova rama:

  git branch <nomeRama>

```
(main) /media/MV-Linux/repositorios/starter-web $ git branch mynewbranch
(main) /media/MV-Linux/repositorios/starter-web $ git branch -a
* main
  mynewbranch
  remotes/origin/main
(main) /media/MV-Linux/repositorios/starter-web $
```

- Cambiarse á nova rama:

  git checkout <nomeRama>

  Este comando realiza dúas accións: mover o apuntador HEAD e actualizar o directorio de traballo coa versión dos arquivos e directorios que había na rama á que se vai mover.

  NOTA: Se houbese cambios pendentes, non confirmados, Git non permitiría saltar de rama.

```
(main) /media/MV-Linux/repositorios/starter-web $ git checkout mynewbranch
Cambiado a rama 'mynewbranch'
(mynewbranch) /media/MV-Linux/repositorios/starter-web $ git branch -a
  main
* mynewbranch
  remotes/origin/main
(mynewbranch) /media/MV-Linux/repositorios/starter-web $
```

  A seguinte imaxe mostra como se ve o cambio realizado no historial. O último commit ten varias etiquetas asociadas que indican que o HEAD apunta tanto á nova rama, como á rama main do repositorio remoto e tamén á local. Isto é así porque aínda non hai ningún cambio realizado.

```
(mynewbranch) /media/MV-Linux/repositorios/starter-web $ git log --oneline --decorate
b1ae1a4 (HEAD -> mynewbranch, origin/main, main) Updating repo with changes from compare section
d9f4787 Adding introduction text
```

- Cambiar o nome dunha rama:

  git branch -m <nomeVello> <nomeNovo>

```
(mynewbranch) /media/MV-Linux/repositorios/starter-web $ git checkout main
Cambiado a rama 'main'
Tu rama está actualizada con 'origin/main'.
(main) /media/MV-Linux/repositorios/starter-web $ git branch -m mynewbranch newbranch
(main) /media/MV-Linux/repositorios/starter-web $ git branch -a
* main
  newbranch
  remotes/origin/main
(main) /media/MV-Linux/repositorios/starter-web $
```

- Eliminar unha rama

  git branch -d <nomeRama>

```
(main) /media/MV-Linux/repositorios/starter-web $ git branch -d newbranch
Eliminada la rama newbranch (era b1ae1a4).
(main) /media/MV-Linux/repositorios/starter-web $ git branch -a
* main
  remotes/origin/main
(main) /media/MV-Linux/repositorios/starter-web $
```

  NOTA: non se pode borrar a rama actual, primeiro haberá que moverse a outra.

- Crear unha rama e ao mesmo tempo cambiarse a esa rama.

  git checkout -b <nomeRama>

```
(main) /media/MV-Linux/repositorios/starter-web $ git checkout -b title-change
Cambiado a nueva rama 'title-change'
(title-change) /media/MV-Linux/repositorios/starter-web $
```

Cando se está traballando nunha rama poden utilizarse os comandos básicos de git para engadir ficheiros (add) e confirmar os cambios (commit).

Para actualizar o repositorio remoto coa nova rama creada en local utilizarase o comando:

git push origin <nomeRama>

Cando se queiran fusionar os cambios de dúas ramas, primeiro haberá que situarse na rama á que se queren incorporar os cambios e despois facer a fusión.

git checkout main

git merge <nomeRama>

Desafortunadamente non sempre será posible fusionar os cambios e produciranse conflitos que haberá que resolver manualmente.

Podes encontrar aquí máis información sobre conflitos con merge.

Fusión e resolución de conflitos

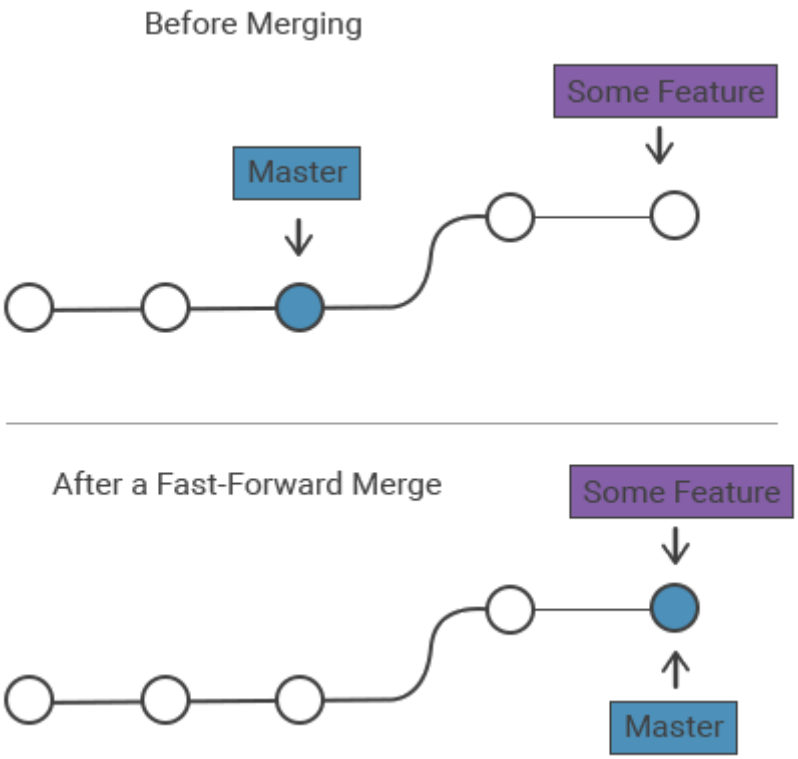
Imaxina que se crea unha rama separada do proxecto e nun momento dado se quere fusionar coa rama master.

Git determina o algoritmo de fusión automaticamente, sen embargo nalgunhas ocasións os procesos de fusión teñen conflitos, por exemplo cando hai modificacións dispares nunha mesma porción dun arquivo nas dúas ramas a fusionar. Neste caso Git non será capaz a de facer a fusión automaticamente.

Fast forward

Unha fusión con avance rápido prodúcese cando hai un proceso lineal dende o extremo da rama actual ata a rama destino. En lugar de fusionar “realmente” as ramas, o único que ten que facer Git para integrar os historiais é mover o extremo da rama actual ao extremo da rama destino, é dicir, facer un “avance rápido”. Información de git merge.

A seguinte imaxe mostra dúas ramas antes de facer a fusión: a rama “Master” e a rama “Some Feature”. Como hai un proceso lineal dende a rama actual ata a rama destino pódese facer unha fusión con avance rápido que consistirá en mover o extremo da rama Master ao extremo da rama “Some Feature”, como se visualiza na parte de abaixo da imaxe.



Operacións para exemplificar unha fusión de tipo Fast forward, ou avance rápido empezando nunha árbore de traballo limpa:

```
# Comprobar que a árbore de traballo está limpa
git status

git checkout -b title-change

# Modificar o ficheiro simple.html
code simple.html

# Comprobar que o ficheiro foi modificado
git status

git add .
git commit -m “Changing title of HTML file”
```

git log --oneline

```
(title-change) /media/MV-Linux/repositorios/starter-web $ git log --oneline
003d4c5 (HEAD -> title-change) Changing title of HTML file
b1ae1a4 (origin/main, main) Updating repo with changes from compare section
d9f4787 Adding introduction text
```

git checkout main

Antes de facer a fusión, comprobar cales son os cambios
git diff main title-change
git difftool main title-change

git merge title-change

```
(main) /media/MV-Linux/repositorios/starter-web $ git merge title-change
Actualizando b1ae1a4..003d4c5
Fast-forward
 simple.html | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
(main) /media/MV-Linux/repositorios/starter-web $
```

git log --oneline

```
(main) /media/MV-Linux/repositorios/starter-web $ git log --oneline
003d4c5 (HEAD -> main, title-change) Changing title of HTML file
b1ae1a4 (origin/main) Updating repo with changes from compare section
d9f4787 Adding introduction text
```

git branch -d title-change

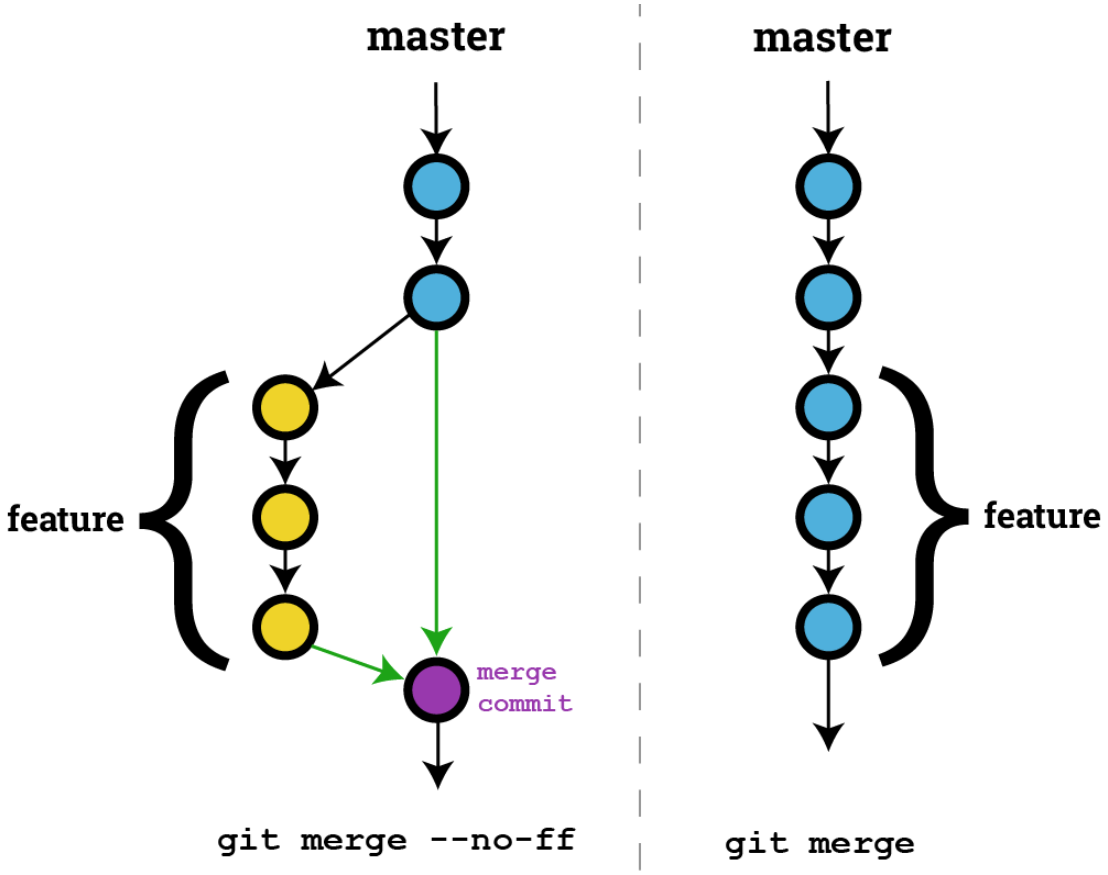
git log --oneline

```
(main) /media/MV-Linux/repositorios/starter-web $ git log --oneline
003d4c5 (HEAD -> main) Changing title of HTML file
b1ae1a4 (origin/main) Updating repo with changes from compare section
```

Fusión deshabilitando Fast forward

Pode interesar facer a fusión deshabilitando o Fast forward, o que significará que se manterán os rexistros da creación da rama e dos cambios realizados.

A seguinte imaxe compara a fusión deshabilitando o fast-forward (esquerda) ou fusión con fast-forward (dereita).



Exemplo de fusión deshabilitando Fast forward empezando dende unha árbore de traballo limpa:

```
# Comprobar que a árbore de traballo está limpa
git status

git checkout -b add-copyright

# Modificar o ficheiro simple.html
code simple.html

git status

git add .
git commit -m "Adding copyright notice"

# Modificar o ficheiro README.md
code simple.html

git status

git add .
```

git commit -m "Adding copyright notice to README”

git log --oneline --graph --decorate

```
(add-copyright) /media/MV-Linux/repositorios/starter-web $ git log --oneline --graph --decorate
* 89916d1 (HEAD -> add-copyright) Adding copyright notice to README
* 4d3c40b Adding copyright notice
* 003d4c5 (main) Changing title of HTML file
* b1ae1a4 (origin/main) Updating repo with changes from compare section
```

git checkout main

Ao deshabilitar o fastforward, en realidade créase un novo commit, polo que o comando
merge solicitará unha mensaxe para o commit. Pode deixarse o que aparece por
defecto
git merge add-copyright --no-ff

git log --oneline --graph --decorate

```
(main) /media/MV-Linux/repositorios/starter-web $ git log --oneline --graph --decorate
*   b70f1ff (HEAD -> main) Merge branch 'add-copyright' into main
|\
| * 89916d1 (add-copyright) Adding copyright notice to README
| * 4d3c40b Adding copyright notice
|/
* 003d4c5 Changing title of HTML file
* b1ae1a4 (origin/main) Updating repo with changes from compare section
```

git branch -d add-copyright

git log --oneline --graph --decorate

```
(main) /media/MV-Linux/repositorios/starter-web $ git log --oneline --graph --decorate
*   b70f1ff (HEAD -> main) Merge branch 'add-copyright' into main
|\
| * 89916d1 Adding copyright notice to README
| * 4d3c40b Adding copyright notice
|/
* 003d4c5 Changing title of HTML file
```

Fusións automáticas

Exemplificación dunha fusión automática, é dicir, sen conflitos. Neste caso créase unha rama nova e fanse cambios. Ademais, na rama main tamén se fai algún cambio. Como os cambios non entran en conflito, pode facerse a fusión de forma automática.

Comprobar que a árbore de traballo está limpa
git status

git checkout -b simple-changes

Facer cambios en ficheiros
code humans.txt

git status

git add .
git commit -m “Adding team member to humans.txt”

git checkout main

Antes de facer a fusión vanse facer cambios na rama main
code README.md

git status

git add .
git commit -m "Adding instructions on how to contribute"

git log --oneline --graph --decorate --all

```
(main) /media/MV-Linux/repositorios/starter-web $ git log --oneline --graph --decorate --all
* 64a5b5a (HEAD -> main) Adding instructions on how to contribute
| * a6bde70 (simple-changes) Adding team member to humans.txt
|/
*   b70f1ff Merge branch 'add-copyright' into main
```

Observar que na imaxe anterior o commit da rama **simple-changes** é diferente
do commit feito na rama **main**

Como a fusión vai solicitar unha mensaxe, inclúese co parámetro **-m**
git merge simple-changes -m "Merging changes from simple-changes branch"

```
(main) /media/MV-Linux/repositorios/starter-web $ git merge simple-changes -m "Merging changes f
rom simple-changes branch"
Merge made by the 'recursive' strategy.
 humans.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
(main) /media/MV-Linux/repositorios/starter-web $
```

Observar que se crea un novo commit
git log --oneline --graph --decorate --all

```
(main) /media/MV-Linux/repositorios/starter-web $ git log --oneline --graph --decorate --all
*   37aac9d (HEAD -> main) Merging changes from simple-changes branch
|\
| * a6bde70 (simple-changes) Adding team member to humans.txt
* | 64a5b5a Adding instructions on how to contribute
|/
*   b70f1ff Merge branch 'add-copyright' into main
```

git branch -d simple-changes

Fixarse que só se eliminou a etiqueta “simple-changes”, a rama segue no grafo
git log --oneline --graph --decorate --all

```
(main) /media/MV-Linux/repositorios/starter-web $ git log --oneline --graph --decorate --all
*   37aac9d (HEAD -> main) Merging changes from simple-changes branch
|\
| * a6bde70 Adding team member to humans.txt
* | 64a5b5a Adding instructions on how to contribute
|/
*   b70f1ff Merge branch 'add-copyright' into main
```

Resolución de conflitos

Git intenta sempre facer a fusión automaticamente, sen embargo non sempre será posible e aparecerán conflitos.

Prodúcese un conflito cando un arquivo foi modificado en dous commits que se queren fusionar e Git non sabe decidir con que versión quedar.

O seguinte exemplo mostra unha fusión con conflitos. Para simulalo realizaranse cambios na mesma zona do mesmo arquivo en dúas ramas diferentes e intentarase facer a fusión ao final.

```
# Comprobar que a árbore de traballo está limpa
git status

git checkout -b realwork

# Facer varios cambios no arquivo simple.html
code simple.html

git status

git add .
git commit -m "Making changes to simple.html"

git status

git checkout main

# Facer varios cambios no arquivo simple.html na rama main. Para crear un conflito,
# facer cambios na mesma zona que se fixeron antes.
# A versión do arquivo é a da rama main e non ten os cambios da rama realwork
code simple.html

git status

git add .
git commit -m "Adding conflicting changes (on purpose) for Git example."

git log --oneline --graph --decorate --all
(main) /media/MV-Linux/repositorios/starter-web $ git log --oneline --graph --decorate --all
* 6039716 (HEAD -> main) Adding conflicting changes (on purpose) for Git example.
| * a46e9f5 (realwork) Making changes to simple.html
|/
*   37aac9d Merging changes from simple-changes branch

git branch

# Ver as diferencias
git diff main realwork
git difftool main realwork
```

Observar que non se fai a fusión porque hai conflito
git merge realwork

```
(main) /media/MV-Linux/repositorios/starter-web $ git merge realwork
Auto-fusionando simple.html
CONFLICTO (contido): Conflicto de fusión en simple.html
Fusión automática falló; arregle los conflictos y luego realice un commit con el resultado.
(main) /media/MV-Linux/repositorios/starter-web $
```

Abrir o ficheiro en conflito cun editor
code simple.html

```
<<<<<<< HEAD
  <title>A Very Respectfull Website</title>
=======
  <title>A Great website</title>
>>>>>>> realwork
```

Git engade aos arquivos conflitivos uns marcadores especiais de resolución de conflitos como guía para facer a edición manualmente e decidir con que versión quedarse. Todo o que está por encima de "=======" é o contido do ficheiro na rama actual (neste caso main, lugar dende onde se intentou a fusión) e todo o que hai debaixo é o contido do arquivo na rama a fusionar. Pode editarse o arquivo con calquera editor de texto e decidir cal debe ser o seu contido. Eliminar por completo as liñas <<<<<<<, ========= e >>>>>>>>.

Unha vez resoltos os conflitos só queda facer:

git add .
git commit -m "Done resolving merge conflicts"

```
(main) /media/MV-Linux/repositorios/starter-web $ git commit -m "Done resolving merge conflicts"
[main 59ae3e5] Done resolving merge conflicts
(main) /media/MV-Linux/repositorios/starter-web $
```

Tan só falta eliminar a rama creada para exemplificar o conflito:
git branch

git branch -d realwork

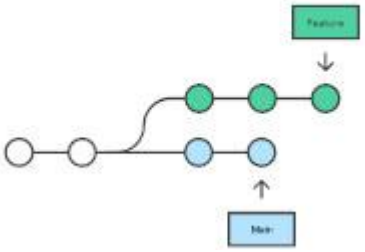
git log --oneline --graph --decorate --all

```
(main) /media/MV-Linux/repositorios/starter-web $ git log --oneline --graph --decorate --all
*   59ae3e5 (HEAD -> main) Done resolving merge conflicts
|\
| * a46e9f5 Making changes to simple.html
* | 6039716 Adding conflicting changes (on purpose) for Git example.
|/
*   37aac9d Merging changes from simple-changes branch
```

Rebase

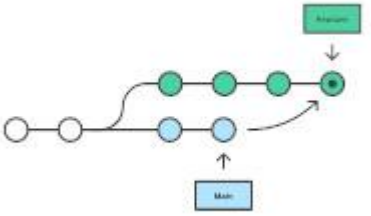
O comando **git rebase** soluciona o mesmo problema que **git merge**, xa que ambos comandos están deseñados para integrar cambios dunha rama noutra, sen embargo fano de forma distinta.

Imaxinar que se empeza a traballar nunha nova función nunha rama específica e despois, outra persoa do equipo actualiza a rama **main** con novas confirmacións. O resultado é un historial bifurcado.

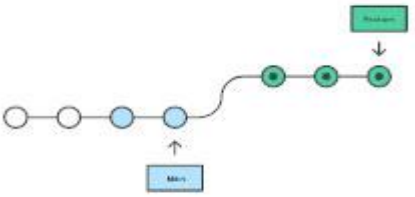


Para incorporar as confirmacións da rama **main** á nova rama hai dúas opcións: facer a fusión con **merge** ou facer unha fusión mediante cambio de base (**rebase**).

Cando se opta por unha fusión, créase unha confirmación de fusión na nova rama que une os dous historiais de ambas ramas, o que proporciona unha estrutura de rama cun aspecto similar ao da seguinte imaxe:



Como alternativa á operación merge, pode facerse unha **fusión mediante cambio de base da nova rama**. Desta forma, a nova rama móvese e empeza no extremo da rama **main** incorporándose de forma eficaz as novas confirmacións. Esta operación reescribe o historial do proxecto creando novas confirmacións para cada confirmación na rama orixinal.



Os seguinte comandos exemplifican unha operación de rebase:

```
git checkout -b myfeature

# Facer cambios no arquivo humans.txt
code humans.txt

git status

git add .
git commit -m "Saying thanks to all my students"

git checkout main

# Facer cambios no arquivo README.md
code README.md
```

git status

git add .
git commit -m "Adding oneline to README for rebase example"

git log --oneline --graph --decorate --all

```
(main) /media/MV-Linux/repositorios/udemy/starter-web $ git log --oneline --graph --decorate --all
* 811a069 (HEAD -> main) Adding oneline to README for rebase example
| * be1806d (myfeature) Saying thanks to all my students
|/
*   1fccc0c (origin/main) Done resolving merge conflicts
```

A imaxe anterior mostra unha bifurcación no historial para as ramas main e myfeature.

Imaxinar que non se rematou o traballo na rama **myfeature** e sen embargo queren incorporarse os cambios feitos na rama **main.** Para iso utilízase a operación **rebase**.

git checkout myfeature

git rebase main

git log --oneline --graph --decorate --all

```
(myfeature) /media/MV-Linux/repositorios/udemy/starter-web $ git log --oneline --graph --decorate --all
* 2b4e0c3 (HEAD -> myfeature) Saying thanks to all my students
* 811a069 (main) Adding oneline to README for rebase example
*   1fccc0c (origin/main) Done resolving merge conflicts
```

Facer cambios no arquivo README.md
code README.md

git status

git add .
git commit -m "Adding another change after rebase"

git log --oneline --graph --decorate --all

```
(myfeature) /media/MV-Linux/repositorios/udemy/starter-web $ git log --oneline --graph --decorate --all
* 179696d (HEAD -> myfeature) Adding another change after rebase
* 2b4e0c3 Saying thanks to all my students
* 811a069 (main) Adding oneline to README for rebase example
*   1fccc0c (origin/main) Done resolving merge conflicts
```

git checkout main

git status

git diff main myfeature

Observar que finalmente se fai un fast-forward merge.
git merge myfeature

```
(main) /media/MV-Linux/repositorios/udemy/starter-web $ git merge myfeature
Actualizando 811a069..179696d
Fast-forward
 README.md  | 2 +-
 humans.txt | 2 +-
 2 files changed, 2 insertions(+), 2 deletions(-)
(main) /media/MV-Linux/repositorios/udemy/starter-web $
```

git log --oneline --graph --decorate --all

```
(main) /media/MV-Linux/repositorios/udemy/starter-web $ git log --oneline --graph --decorate --all
* 179696d (HEAD -> main, myfeature) Adding another change after rebase
* 2b4e0c3 Saying thanks to all my students
* 811a069 Adding oneline to README for rebase example
*   1fccc0c (origin/main) Done resolving merge conflicts
```

git branch -d myfeature
git branch

Stash

O comando **git stash** almacena temporalmente os cambios efectuados no código no que se está traballando para poder facer outra cousa e, máis tarde, regresar e recuperar os cambios nos que se estaba traballando. Gardar os cambios en stashes resulta práctico cando hai que cambiar rapidamente de contexto e poñerse a facer outra cousa urxente (como arreglar un erro), cando aínda non estaba rematado o código no que se estaba traballando.

Exemplo de realización dun stash:

Comezando coa árbore de traballo limpa, modifícase un ficheiro
code simple.html

Comprobar o estado do repositorio
git status

Gardar os cambios no stash
git stash

```
(main) /media/MV-Linux/repositorios/udemy/starter-web $ git stash
Directorio de trabajo y estado de índice WIP on main: 4e590b8 local:updating simple.html copyright notice guardados
(main) /media/MV-Linux/repositorios/udemy/starter-web $
```

Comprobar que o directorio de traballo volveu ao estado inicial, é dicir unha árbore
limpa sen os cambios feitos ao ficheiro simple.html
git status

Facer cambios no ficheiro README.md
code README.md

Facer un commit
git add .
git commit -m "Quick fix in production to improve copyright notice"

Recuperar os cambios gardados no stash para seguir traballando neles
Observar que informa do estado do repositorio
git stash apply

```
(main) /media/MV-Linux/repositorios/udemy/starter-web $ git stash apply
En la rama main
Tu rama está adelantada a 'origin/main' por 1 commit.
  (usa "git push" para publicar tus commits locales)

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 directorio de trabajo)
        modificado:      simple.html

sin cambios agregados al commit (usa "git add" y/o "git commit -a")
(main) /media/MV-Linux/repositorios/udemy/starter-web $
```

Continuar traballando ata facer o commit
git add .
git commit -m "Done with simple.html updates"

Os cambios do stash permanecen na área de almacenamento reservada:
git stash list

```
(main) /media/MV-Linux/repositorios/udemy/starter-web $ git stash list
stash@{0}: WIP on main: 4e590b8 local:updating simple.html copyright notice
(main) /media/MV-Linux/repositorios/udemy/starter-web $
```

Para eliminar o stash executar o comando
git stash drop

```
(main) /media/MV-Linux/repositorios/udemy/starter-web $ git stash drop
Descartado refs/stash@{0} (822141661bc655a0c0abdb45961554432924570e)
(main) /media/MV-Linux/repositorios/udemy/starter-web $
```

O comando **git stash** non inclúe a información relacionada cos ficheiros en estado "untracked". Para que se inclúan estes ficheiros hai que usar o parámetro **-u:**

git stash -u

O comando **git stash pop** é equivalente á execución dos comandos **git stash apply** e **git stash drop**. Pop fai referencia a eliminar o elemento superior da pila

Referencias

Para a elaboración deste material utilizáronse, entre outros, os recursos que se enumeran a continuación:

- https://manuais.iessanclemente.net/index.php/Control_de_versiones_con_Git_y_Git_Hub

- Repositorio exemplos: https://github.com/awesomejt/starter-web
- https://www.freecodecamp.org/news/the-beginners-guide-to-git-github/
- https://git-scm.com/book/es/v2
- https://git-scm.com/book/es/v2/Ramificaciones-en-Git-Procedimientos-B%C3%A1sicos-para-Ramificar-y-Fusionar
- https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts