Universidad de Murcia

Facultad de Informática

Universidad Politécnica de Cartagena

Escuela Técnica Superior de Ingeniería de Telecomunicación

TÍTULO DE GRADO EN
CIENCIA E INGENIERÍA DE DATOS

Fundamentos de Computadores

Práctica 2: Control de versiones con Git

Boletines de prácticas

CURSO 2022 / 23

Índice general

Boletines de prácticas

B2.1. La bitácora del alumno de Fundamentos de Computadores.

Durante el trascurso de este curso, cada estudiante va a utilizar un repositorio Git en el que registrar de manera periódica su trabajo en el laboratorio de prácticas (seguimiento de los ejemplos de los boletines, resolución de los ejercicios propuestos, etc.) con el fin de permitir al profesor supervisar el grado de seguimiento y consecución de los objetivos de aprendizaje. Este repositorio funcionará, por tanto, como una bitácora personal de todas las prácticas que el alumno también podrá consultar en cualquier momento para recordar los ejercicios realizados. En este boletín aprenderás a crear y utilizar tu repositorio-bitácora en cada sesión de prácticas.

NOTA IMPORTANTE SOBRE LA EVALUACIÓN.
En la evaluación de la bitácora del alumno se tendrá en cuenta tanto el **contenido** de las contribuciones como la **periodicidad** con que cada alumno contribuye a su repositorio. Igualmente, la **organización** adecuada del contenido registrado en la bitácora resulta imprescidible para que el alumno pueda alcancar la puntuación asignada a esta herramienta de evaluación continua.
En cuanto al contenido, se valorará la corrección de las respuestas a los ejercicios y preguntas formuladas en los boletines. En cuanto al número de ejercicios registrados, es necesario realizar como mínimo el **80 % de los ejercicios propuestos** para que la bitácora se considere apta.
Con respecto a la periodicidad de las contribuciones, dado que se trata de una herramienta de evaluación continua, es imprescindible que las contribuciones a la bitácora se realicen en el plazo máximo marcado, que con carácter general se establece en **dos semanas a partir de la sesión de laboratorio** en la que se da por concluida la práctica correspondiente, atendiendo a lo dispuesto en la planificación de la asignatura para cada subgrupo.
En lo que se refiere a la organización, es fundamental seguir las siguientes instrucciones a la hora de estructurar el contenido del repositorio así como **documentar** los ejercicios de cada boletín (p.ej., `#EJERCICIO 1`), tal y como se describe en los mismos. La bitácora debe tener un directorio por cada práctica, dentro del cual habrá un subdirectorio por cada boletín; en la carpeta de cada boletín se ubicará tanto el fichero con la sesión de terminal grabada mediante el comando `script`, como cualquier otro fichero solicitado en los ejercicios o que el alumno considere relevante.

B2.2. Tutorial de Git

B2.2.1. Control de Versiones con Git

Un sistema de control de versiones (o CVS -Control Version System- de sus siglas en inglés) es simplimente una forma de mantener un seguimiento de cambios en ficheros y carpetas, lo cuales pueden ser modificados por uno o varios autores. Un CVS debe tener como mínimo la funcionalidad básica de:

- Registrar quién hizo qué y cuándo
- Revertir ficheros a un estado previo
- Revertir un proyecto entero a un estado previo
- Comparar los cambios hechos a lo largo del tiempo

La mayor ventaja de un sistema de control de versiones es la confianza y tranquilidad de poder recuperar nuestros datos a un estado previo si, por cualquier motivo, éstos se pierden o machacan con valores incorrectos. Existen dos tipos de CVS:

- Centralizados. Utilizan un servidor central que almacena los datos y ningún participante tiene una copia de estos datos. Los participantes se descargan los datos, los modifican y vuelven a volcar al servidor central.
- Distribuidos. No tienen necesidad de un servidor central. Cada participante clona los datos en su ordenador local y la historia completa de estos datos, lo cual en terminología CVS se conoce como *repositorio*. No existe un repositorio maestro, todos los repositorios locales son igualmente válidos. Los CVS distribuidos son más rápidos y flexibles ya que nos permiten trabajar offline y hacer cambios localmente.

En 2005, Linux Torvalds desarrolló Git como un CVS distribuido para el kernel de Linux. De hecho, el kernel 2.6.12 de Linux fue la primera distribución manejada mediante Git. Git es gratuito, de código abierto y diseñado bajo la premisa de eficiencia:

- Distribuido: No existe una copia "maestra"del repositorio, sino que cada copia contiene toda la información.

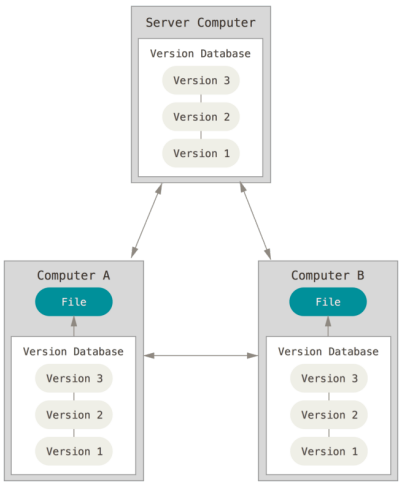


Figura I.1: Ejemplo de repositorio distribuido

- Simple: Mantiene instantáneas o versiones (snapshots) de un árbol de directorios que contiene objetos en lo que llamamos *repositorio*.

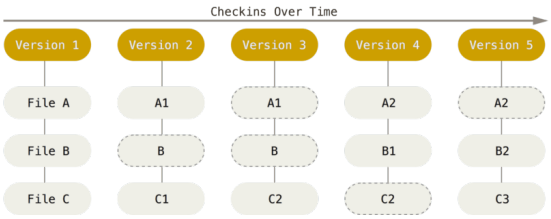


Figura I.2: Snapshots en Git

- Seguro: Cada objeto es referenciado mediante una suma de comprobación, checksum o hash (SHA-1). Además, las operaciones sobre el repositorio normalmente sólo añaden datos.
- Rápido: La mayor parte de las operaciones sobre el repositorio son locales y los objetos están comprimidos. Unión automática de múltiples cambios en uno.
- Gran facilidad para aislar tu trabajo en ramas. Soporte para desarrollo no lineal en grandes proyectos (miles de ramas).
- Aceptación mundial. Git se ha convertido en el CVS de facto.

B2.2.2. Conceptos Básicos de Git

Áreas

Un *repositorio* es una colección de instantáneas o versiones de un árbol de directorios (proyecto). Una *instantánea* (*snapshot*) es una copia del proyecto en un punto del tiempo, conteniendo una copia completa de todos los ficheros, excepto aquellos que ya existían. La Figura I.3 muestra las diferentes areas de trabajo en Git.

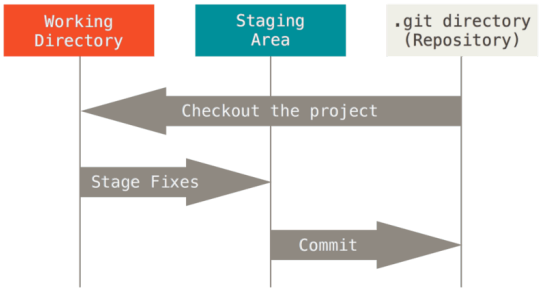


Figura I.3: Flujo de trabajo básico en Git

- El **directorio .git** contiene la base de datos del repositorio, es decir, todas las instantáneas o versiones del mismo.
- El **directorio de trabajo** (*working directory*) es una copia de una instantánea o versión del repositorio. A nivel práctico, es donde los ficheros de un usuario residen y son modificados.
- El **índice** *(staging area)* incluye aquellos archivos que han sido modificados y que son candidatos a ser incluidos en la próxima instantánea o versión del repositorio.

El flujo general de trabajo en Git, como se muestra en Figura I.3 es el siguiente:

1. El proyecto del usuario es contenido en un repositorio Git (asumimos por ahora que este repositorio es local). El usuario *vuelca* este repositorio a su área de trabajo, lo cual se realiza mediante una operación de **checkout**.
2. El usuario trabaja modificando ficheros en su directorio de trabajo.
3. Llegado a un punto, el usuario *registra* los cambios hechos hasta el momento, mediante una operación de **stage**.
4. Finalmente, el usuario *publica* los cambios registrados actualizando el repositorio mediante una operación de **commit**.
5. El usuario repite sucesivamente los pasos 2-4 conforme trabaja en su directorio de trabajo.

Estados

Los archivos pueden estar en varios estados antes de estar confirmados (committed), es decir, antes de formar parte de una instantánea o versión del repositorio:

- Ignorado (untracked): No forma parte del repositorio y no está en el índice para ser incluido en él.
- No modificado (unmodified): Forma parte del directorio de trabajo pero no ha sido alterado.
- Modificado (modified): Forma parte del directorio de trabajo y ha sido actualizado.
- Candidato (staged): Forma parte del índice (ha sido modificado) y es candidato a formar parte de la próxima instantánea o versión del repositorio.
- Confirmado (committed): Se incluye en la última instantánea o versión del repositorio.

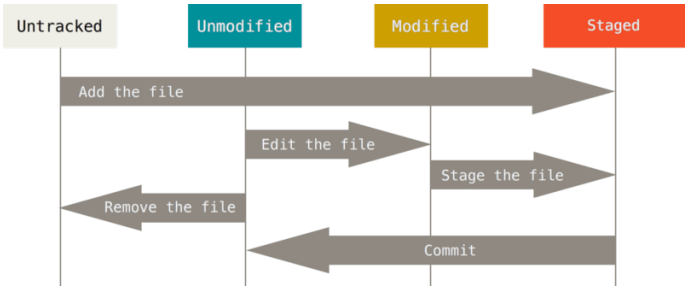


Figura I.4: Estados de un fichero en Git

Repositorio remoto

Un repositorio remoto es una copia de un repositorio que se encuentra en un servidor remoto. Aunque Git no requiere utilizar un repositorio remoto, éste simplifica la colaboración entre varios usuarios además de ofrecer tolerancia a fallos. La Figura I.5 muestra el flujo de trabajo básico con un repositorio remoto así como sus principales comandos.

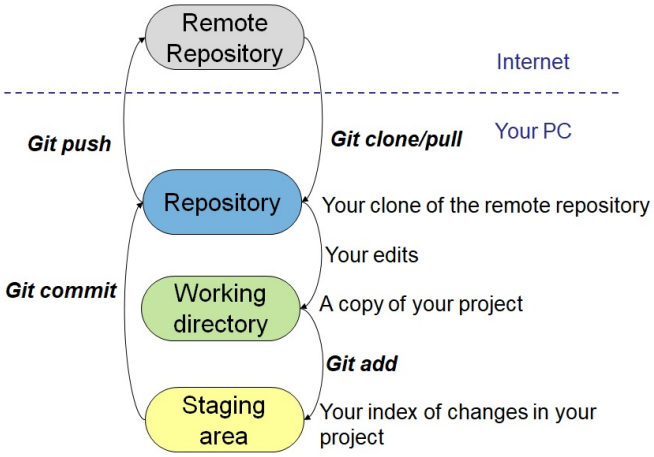


Figura I.5: Flujo de trabajo general en Git con repositorio remoto

El comando `git clone` clona un repositorio remoto en la máquina local y al mismo tiempo crea el directorio de trabajo y el índice del repositorio. El comando `git pull` nos permite obtener los últimos cambios que han sido publicados en el repositorio remoto (por ejemplo, por otros participantes del repositorio), y mezclarlos con el contenido de nuestro directorio de trabajo. El comando `git add` nos permite actualizar el índice de nuestro repositorio, es decir registrar los cambios que hemos hecho en nuestros ficheros locales. El comando `git commit` actualiza los cambios en el repositorio local y el comando `git push` actualiza el repositorio remoto según el estado del repositorio local.

La Figura I.6 clarifica la secuencia típica de trabajo entre un repositorio local y remoto. Una vez se ha clonado el repositorio en local, éste pasará por una serie de estados "Modificado" según vayamos modificando archivos. El índice se irá actualizando con estos cambios y se vaciará una vez hagamos el *commit* a nuestro repositorio local. Este comando registra los cambios localmente, por lo que nuestro repositorio local pasa a estar "desincronizado" con el repositorio remoto ya que este último no ha sido aún actualizado con estos cambios. Finalmente, una operación de *push* actualizará el repositorio remoto con todos los cambios confirmados.

Es importante destacar que aunque esta es la secuencia típica, un usuario podría decidir editar sus ficheros sin hacer nunca un *commit*, pero en este caso el usuario no tendrá la "historia" del repositorio y en caso de pérdida de datos no podrá volver a ninguna versión previa. El usuario podría decidir también no hacer nunca un *push* y en este caso todos los datos serían locales, no tendría ninguna copia de respaldo, y en caso de ser un proyecto colaborativo, no permitiría a otros usuarios ver sus contribuciones.

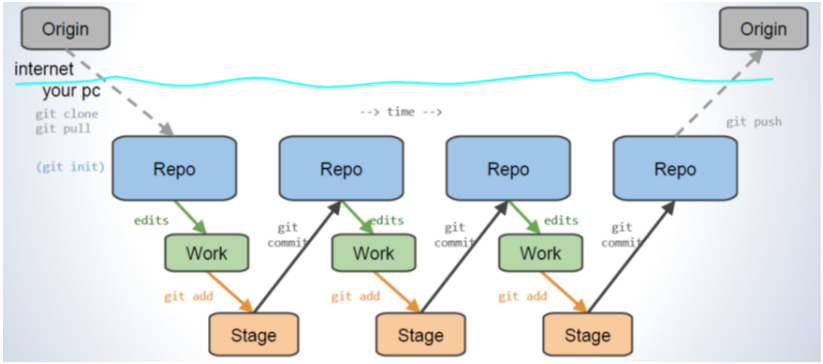


Figura I.6: Secuencia de comandos con repositorios Git

Ramas

El concepto de rama en un repositorio Git nos permite aislar una "línea de trabajo" experimental mientras mantenemos la línea principal inalterada. Por convenio, la rama principal de un repositorio se llama `main` [1] y el repositorio se conoce como `origin`. Por ejemplo, para actualizar el repositorio remoto utilizamos el comando `git push origin main` para indicar a Git que debe actualizar la rama `main` del repositorio remoto origin. Un usuario puede crear otras ramas para trabajar de forma temporal o experimental y finalmente actualizar la rama `main` con los cambios desarrollados en la rama experimental. Figura I.7 muestra el concepto de ramas en Git de forma conceptual. No obstante, en esta asignatura, nuestro uso de Git es básico y por tanto nos vamos a limitar sólo al uso de la rama principal.

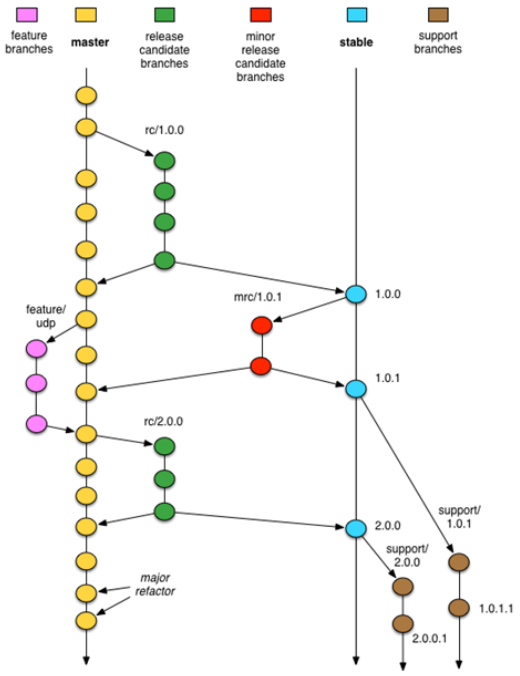


Figura I.7: Ejemplo de ramas en Git

[1] La rama principal de muchos repositorios todavía se sigue llamando `master`, aunque existe un compromiso global por evitar términos relacionados con el racismo.

B2.2.3. Listado de comandos Git

Aunque los comandos Git básicos que vamos a utilizar se ven después en los ejercicios del boletín de esta sesión, aquí resumimos los principales:

- `git clone url` : Clona un repositorio remoto con la dirección *url* en el directorio actual.
- `git add nombre_fichero` : Inserta el fichero llamado *nombre_fichero* en el índice del repositorio.
- `git commit` : Registra los cambios insertados en el índice del repositorio y vacia éste.
- `git push` : Actualiza el repositorio remoto con los cambios registrados en nuestro repositorio local.

B2.3. Boletín 1: Mi repositorio Git para FC

B2.3.1. Objetivos

El objetivo de esta primera sesión de prácticas es conseguir que el alumno se familiarice con la herramienta de control de versiones Git para crear su propio repositorio, mantener y gestionar las versiones de ficheros en él y ser capaz de revertir cambios cuando sea necesario. Este repositorio nos servira a modo de bitácora para registrar nuestro progreso en las prácticas de esta asignatura. Además, para almacenar nuestro repositorio en el cloud, utilizaremos **Github**, que es por mucho el servicio de alojamiento (*hosting*) de repositorios Git más utilizado a nivel mundial.

Para esta práctica necesitas utilizar el software instalado en los ordenadores del laboratorio de la asignatura o alternativamente haber previamente seguido los pasos del boletín anterior para instalar en la máquina virtual de Ubuntu la herramienta Git.

B2.3.2. Plan de trabajo

El plan de trabajo de esta sesión será el siguiente:

1. Lectura del boletín por parte del alumno.
2. Seguimiento de ejemplos presentados por el profesor.
3. Realización de los ejercicios propuestos en el boletín (con supervisión del profesor).

B2.3.3. Ejercicios a realizar durante la sesión

1. **Accede a la máquina virtual de Ubuntu**. Abre la aplicación *VirtualBox* e inicia la máquina virtual de Ubuntu preconfigurada. Una vez que estés en el escritorio gráfico, abre una consola de comandos tal y como describimos en el boletín anterior (e.g. con `Ctrl + Alt + T`)

2. **Comienza a registrar tu actividad**. Ejecuta el comando script para empezar a registrar esta sesión de prácticas en un fichero llamado `typescript-prac2`. Teclea `### PRÁCTICA 2 ###` para documentar en tu fichero los ejercicios del boletín en el que nos encontramos.

3. **Configura Git**. La herramienta 'git' se puede configurar de forma extensa. En nuestro caso, vamos a configurar sólo el nombre y el email del usuario con el comando `git config`, asumiendo que esta configuración es la misma para todos los repositorios del usuario.

   Warning! Usa tu nombre y apellido(s), y tu dirección de correo electrónico universitario (de la UMU o la UPCT).

   ```
   $ git config --global user.name "Nombre Apellido1 [Apellido2]"
   $ git config --global user.email "alumno@um.es"
   ```

La configuración actual se puede consultar también con el comando `git config`:

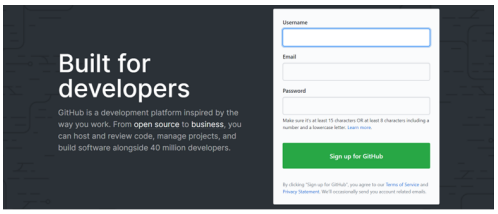
```
$ git config --global --list
user.name=Nombre Apellido1 [Apellido2]
user.email=alumno@um.es
```

La configuración se almacena en el archivo `$HOME/.gitconfig`:

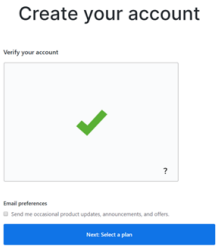
```
$ cat $HOME/.gitconfig
[user]
    name = Nombre Apellido1 [Apellido2]
    email = alumno@um.es
```

4. **Creación de una cuenta en GitHub**. Accede a `github.com` • `SignUp` desde el navegador de tu máquina virtual, y crea una cuenta con tu nombre, apellidos y dirección de correo electrónico (de la UMU o UPCT) (ver Figura I.8).

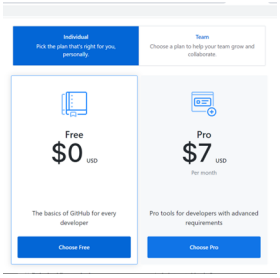
   NOTA: Si tu dirección de correo electrónico es 'alumno@um.es' o 'alumno@upct.es', el nombre de usuario debe ser 'alumno-um-es' o 'alumno-upct-es' respectivamente.



(a) Introduce tus datos



(b) Verifica tu cuenta



(c) Elige el plan gratuito

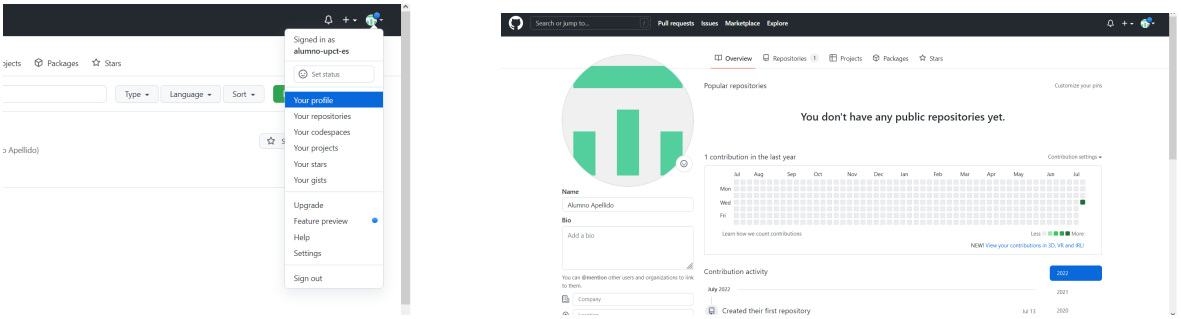
Figura I.8: Creación de cuenta en Github

Una vez que has creado tu cuenta, configura tu nombre y primer apellido haciendo click en el icono de la esquina superior derecha y seleccionando 'Tu perfil'. En tu perfil, haz click en el botón "Editar perfil" y escribe tu nombre y apellido en el campo 'Nombre' (ver Figura I.9).

5. **Creación de un repositorio para FC en GitHub**. En tu cuenta de Github, para crear un nuevo repositorio, haz click en el desplegable de la esquina superior derecha con el simbolo '+' y selecciona 'Nuevo repositorio' (ver Figura I.10).

   Como se muestra en la Figura I.11, nombra el repositorio como "fc-alumno". En la descripción del repositorio, indica "Fundamentos de Computadores" seguido de tu nombre y apellido entre paréntesis. Selecciona la visibilidad del repositorio como 'Privado' y haz click en el botón "Crear repositorio".

   Esto te llevará a la página home de tu repositorio (ver Figura I.12), la cual te indica la URL de éste.



(a) Edita tu perfil (b) Indica tu nombre y apellido

Figura I.9: Configuración de nombre y apellido en cuenta Github

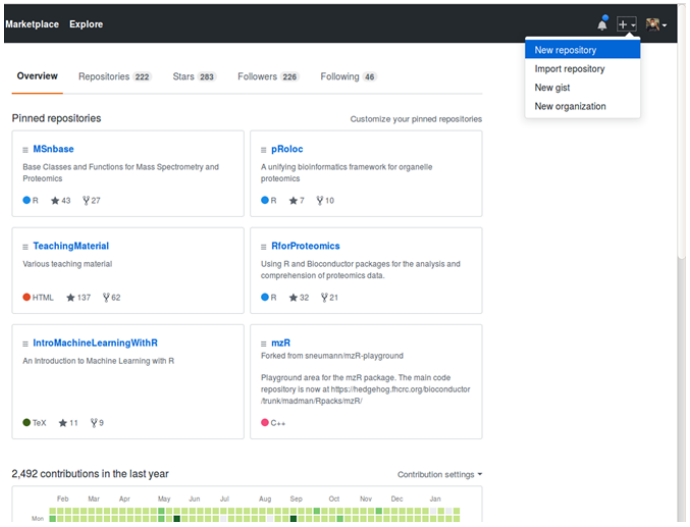


Figura I.10: Crear nuevo repositorio en Github

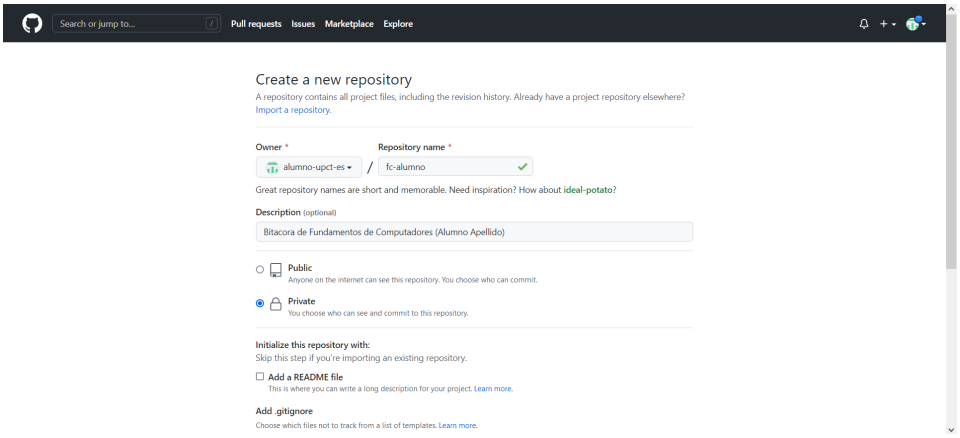


Figura I.11: Configuración de repositorio para FC

6. **Crea un subdirectorio en tu directorio HOME de Ubuntu**. Vuelve a la consola de comandos de tu máquina virtual Ubuntu y en tu directorio home de Ubuntu (e.g. */home/alumno*) crea una carpeta que se llame FC mediante el comando **mkdir FC**.

7. **Intenta clonar tu repositorio remoto en local**. Para clonar el repositorio remoto que has creado en Github en tu máquina local, vamos a utilizar el comando `git clone`. Para ello, copia la URL de tu repositorio haciendo

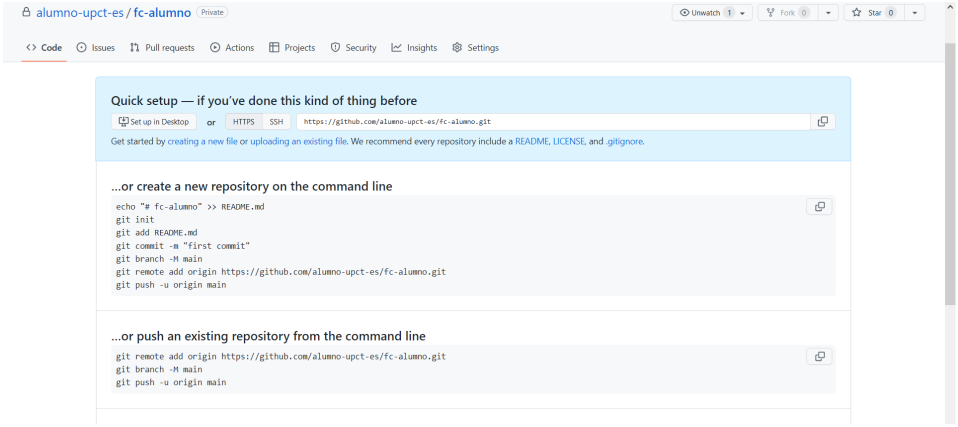


Figura I.12: Página home de repositorio Git de FC

click en el icono de pegar en el extremo derecho de la sección "Quick Setup" de la página home de tu repositorio Github (ver Figura I.12). En tu carpeta FC, ejecuta el siguiente código:

```
$ git clone https://github.com/alumno-upct-es/fc-alumno.git
Cloning into 'fc-alumno'...https://github.com/alumno-upct-es/fc-alumno.git
Username for 'https://github.com':
Password for 'https://alumno-upct-es@github.com':
```

¿Por qué no puedes clonar tu repositorio?

Github ha eliminado la autenticación mediante usuario y contraseña desde 2021 por motivos de seguridad. Ahora, necesitamos configurar un token y utilizarlo en nuestros comandos Git, lo cuál vamos a hacer en el siguiente punto.

8. **Obtención de un token de acceso personal**. Haz click en el icono de la esquina superior derecha de tu cuenta Github y selecciona 'Settings'. Al final del todo del panel izquierdo, selecciona 'Preferencias de desarrollador', y en estas preferencias, selecciona 'Personal access tokens' en el panel izquierdo (ver Figura I.13).

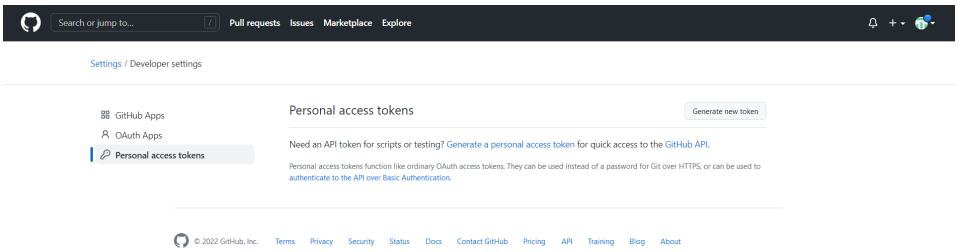


Figura I.13: Crear Personal Access Token en Github

Haz click en "Generate new token" e indica los siguientes campos como se muestra en la Figura I.14. En Nota, indica "PAT para FC", cambia el número de días de validez del token de 30 días a 90 días y selecciona los ámbitos (scopes) del token como "repo" y "workflow".

Ahora haz click en "Generate token". Aparecerán los datos del token que acabas de crear (ver Figura I.15). Copia tu token haciendo click en el icono de copiado a su derecha.

Atención! Aseguráte de guardar en un lugar seguro este token, ya que Github no te permitirá volver a visualizarlo. Te recomendamos que lo guardes en un fichero de algún espacio de almacenamiento cloud como OneDrive o GoogleDrive. Si prevees que vas a utilizar siempre tu ordenador portátil personal, puedes almacenarlo de forma

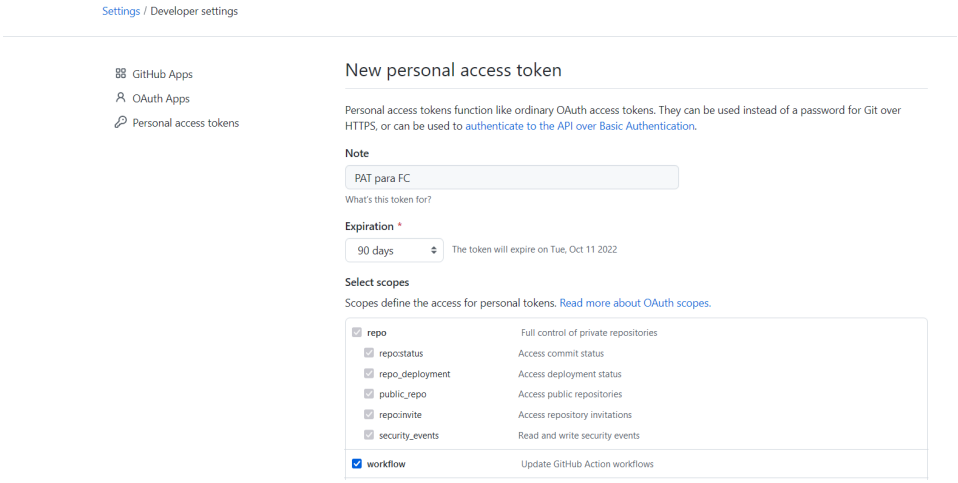


Figura I.14: Configración de Personal Access Token en Github

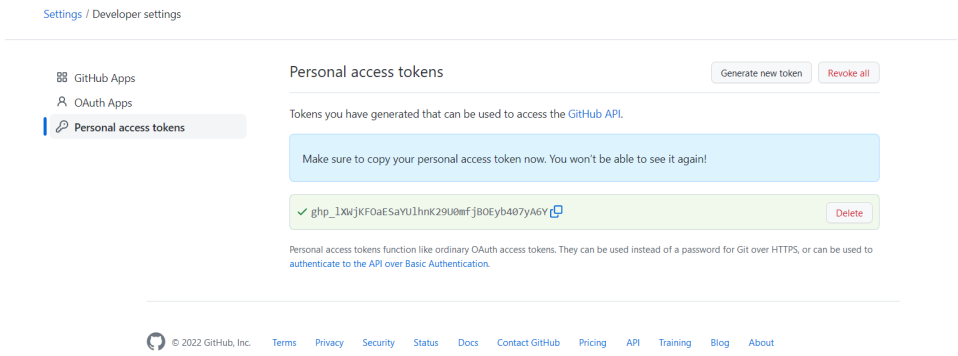


Figura I.15: Personal Access Token creado en Github

local mediante los siguientes pasos: copia el token y vuelve a tu línea de comandos de Ubuntu, crea un archivo oculto que se llame *pat* ejecutando **nano .pat**, pega el token haciendo click en el boton derecho del ratón y seleccionando 'Pegar', y por último guarda y sal de la herramienta Nano con `CTR + o` y `CTR + x`. Podras visualizar este fichero oculto con el comando *ls -la* y mostrar el token con **cat .pat**. Nunca sigas estos pasos en un ordenador de laboratorio puesto que son compartidos y otros estudiantes podrían ver, borrar o modificar tu token.

9. **Clona tu repositorio remoto en local con tu token personal**. Vuelve a tu consola de comandos y ejecuta de nuevo el comando `git clone` pero esta vez con tu token personal. Para evitar volver a teclear el comando, pulsa la flecha de arriba de tu teclado para acceder a los últimos comandos tecleados en la consola hasta encontrar el comando `git clone` y pulsa "Enter". Esta vez, cuando Git solicite el nombre de usuario, pega tu token personal haciendo click en el botón derecho del ratón. Cuando solicite tu contraseña, simplemente pulsa la tecla de "Enter".

```
$ git clone https://github.com/alumno-upct-es/fc-alumno.git
Cloning into 'fc-alumno'...https://github.com/alumno-upct-es/fc-alumno.git
Username for 'https://github.com': [PAT]
Password for 'https://[PAT]@github.com': [Enter]
warning: You appear to have cloned an empty repository.
```

10. **Entra a tu directorio de trabajo local 'fc-alumno'**.

11. **Examina la configuración de tu repositorio**. Git crea una carpeta oculta *.git* que contiene toda la configuración del repositorio. Para ver el contenido de esta carpeta ejecuta los siguientes comandos en tu directorio de trabajo ('fc-alumno'):

```
$ ls .git
branches        config       HEAD   index  logs     refs
COMMIT_EDITMSG  description  hooks  info   objects
$ cat .git/config
[core]
repositoryformatversion = 0
filemode = true
bare = false
logallrefupdates = true
[remote "origin"]
url = https://github.com/alumno-upct-es/fc-alumno.git
fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
remote = origin
merge = refs/heads/main
```

¿Cómo se llama la rama principal por defecto de nuestro repositorio?

12. **Crea el fichero README de tu repositorio**. Utiliza el comando *echo* para crear el fichero 'README.md' en el repositorio 'fc-alumno' desde la línea de órdenes. Este fichero contendrá tu nombre y apellido:

```
$ echo -e "# FC\n\n- Alumno Apellido" > README.md
```

13. **Haz un commit de tu fichero README**. Para hacer un commit del cambio que acabas de hacer, primero tienes que añadir este cambio al índice de tu repositorio local mediante el comando `git add`. Después, necesitas actualizar tu repositorio local mediante el comando `git commit`.

```
$ git add README.md
$ git commit -m "Creación de README.md"
```

Nota que la operación de *commit* necesita una pequeña nota que indique muy brevemente el cambio realizado (tras el parámetro -m).

14. **Actualiza tu repositorio remoto**. Para actualizar el repositorio remoto en Github con los cambios que hemos hecho localmente utilizamos el comando `git push` con nuestro token personal:

```
$ git push --set-upstream origin main
Username for 'https://github.com': [PAT]
Password for 'https://[PAT]@github.com': [Enter]
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 245 bytes | 81.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/alumno-upct-es/fc-alumno.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Esta operación de push actualiza la rama principal (main) del repositorio remoto y la opción *–set-upstream* establece la rama por defecto en la que se está trabajando para futuras operaciones de *pull* y *push*.

15. **Verifica que el repositorio en Github ha sido actualizado con éxito**. Para ello, ve a tu cuenta en Github y en tus repositorios, haz click en el repositorio. Como se muestra en Figura I.16, podrás ver el fichero README.md que has creado. Haz click en la pestaña "1 branch"para ver todas las ramas del repositorio, en este caso, sólo tenemos una que es la principal y se llama *main* por defecto.

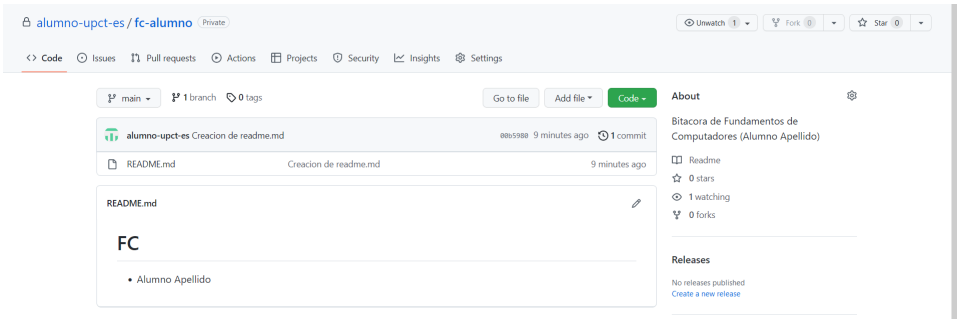


Figura I.16: Repositorio actualizado en Github

16. **Modifica archivos y comprueba los cambios en tu repositorio local**. Cambia tu archivo README.md añadiendo al final de tu nombre, tu segundo apellido. Para ello, en tu directorio de trabajo abre el fichero mediante *nano*, añade tu segundo apellido, pulsa `CTR + o` para guardar y `CTR + x` para salir. Ahora comprueba el estado de tu repositorio con el comando `git status`:

```
$ git status
On branch main
Your branch is up to date with 'origin/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 directory)
modified:   README.md

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

Git te está indicando que el fichero README.md ha sido modificado pero, puesto que los cambios no han sido registrados en el índice, el repositorio está actualizado (no hay nada que actualizar). Ahora observa las diferencias en el fichero que ha cambiado con el comando `git diff`:

```
$ git diff
diff --git a/README.md b/README.md
index ef33406..3bc127f 100644
--- a/README.md
+++ b/README.md
@@ -1,3 +1,3 @@
 # FC

-- Alumno Apellido
+- Alumno Apellido Apellido2
```

Vemos que ha cambiado una línea del fichero, la antigua está precedida por '- -' y su valor actual por '+-'.

Ahora crea otro fichero en tu directorio de trabajo que se llame "hello.txt" con el comando **touch hello.txt**, después vuelve a comprobar el estado de tu repositorio.

```
$ touch hello.txt
$ git status
On branch main
Your branch is up to date with 'origin/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 directory)
modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
```

```
hello.txt

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

Como podemos ver, ahora tenemos dos ficheros con dos estados diferentes: README.md sigue "Modificado" (modified) y hello.txt está "Ignorado" (untrucked) porque nunca ha sido insertado en el índice del repositorio. Los dos constituyen cambios en el directorio de trabajo, pero no están preparados para ser registrados en el repositorio.

17. **Actualiza tu repositorio local con varios cambios**. Para preparar todos los cambios de nuestro repositorio en el índice, utilizamos el comando `git add -A`:

```
$ git add -A
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
modified:   README.md
new file:   hello.txt
```

Ahora los dos ficheros han sido insertados en el índice del repositorio, y por tanto han pasado al estado "Candidato" (staged) (es decir, están preparados para ser registrados o confirmados en el repositorio).

18. **Extracción de un fichero del índice del repositorio**. Si por cual motivo nos hemos equivocando añadiendo un fichero a nuestro índice, podemos revertir está acción con el comando `git restore`. Vamos a eliminar el archivo *hello.tx*t de nuestra area de staging:

```
$ git restore --staged hello.txt
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
hello.txt
```

El fichero hello.txt vuelve a tener el estado de "Ignorado" (Untracked).

19. **Eliminación de un fichero del directorio de trabajo**. Para eliminar un archivo de nuestro directorio de trabajo utilizamos el comando `git clean`. Ten en cuenta que este comando eliminará todos los archivos que tienen el estado Ignorado, por lo que hay que ser muy cuidadoso con él. Es recomendable utilizarlo primero con el parámetro '-n' el cual lo ejecuta en modo simulado (o dry-run).

```
$ git clean -n
Would remove hello.txt
$ git clean -f
Removing hello.txt
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
modified:   README.md
```

El archivo hello.txt no sólo ha sido extraído del índice del repositorio sino que también ha sido eliminado del sistema de archivos.

20. **Haz un commit del fichero README.md modificado.** Para registrar este cambio en nuestro repositorio local ejecutamos `git commit` y comprobamos el estado del repositorio tras este registro con `git status`.

```
$ git commit -m "README modificado con segundo apellido"
[main eedac85] README modificado con segundo apellido
1 file changed, 1 insertion(+), 1 deletion(-)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to be commited, your working ares is clean
```

Como puedes ver, el comando `git status` te está informando de que los repositorios locales y remotos no coinciden y debes hacer un push para publicar tus cambios locales.

21. **Publica de nuevo tu fichero README.** Para actualizar tu repositorio local con el fichero README modificado y registrado en tu repositorio local, utiliza el comando `git push`. Recuerda que debes utilizar tu token personal:

```
$ git push origin main
Username for 'https://github.com': [PAT]
Password for 'https://[PAT]@github.com': [Enter]
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 296 bytes | 2967.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/alumno-upct-es/fc-alumno.git
  00b5980..eedac85      main -> main
```

Si compruebas ahora el estado del repositorio veras que ahora sí está actualizado con la rama remota.

22. **Termina de registrar tu actividad y copia tu fichero *typescript-prac2* a tu repositorio local**. Ejecuta el comando exit para terminar de registrar tu actividad. Mueve el fichero generado a tu repositorio local con el comando `mv`, el cual tiene como primer parámetro el fichero a mover y como segundo parámetro la carpeta destino. Recuerda usar nombres distintos del fichero generado para cada práctica.

23. **Publica tu fichero *typescript-prac2* en tu repositorio remoto**. Para ello, tal y como hemos hecho en ejercicios anteriores, utiliza los comandos *git add*, `git commit` y `git push`.

24. **Ahora comparte tu repositorio con los profesores de la asignatura**. Ve a tu repositorio en tu cuenta de Github y selecciona Settings (ver Figura I.17). Una vez allí, haz click en Collaborators en el panel izquierdo y haz click en el botón Add people. Aparecerá una ventana, dónde debes introducir el email de tu profesor y seleccionarlo cuando aparezca su usuario Github automáticamente. Haz click en el botón para añadirlo.

25. **Importante! Finalmente, elimina tu repositorio local.**. Para borrar tu repositorio local, simplemente utiliza el comando `rm` para borrar el directorio de tu repositorio. Deberás usar los parametros *r* y *f* para borrar todos los ficheros incluidos en el directorio de forma recursiva y forzar su borrado (el SO no pedirá confirmación), respectivamente. No olvides este último paso en cada sesión de prácticas ya que otros usuarios podrían modificar tu bitácora!

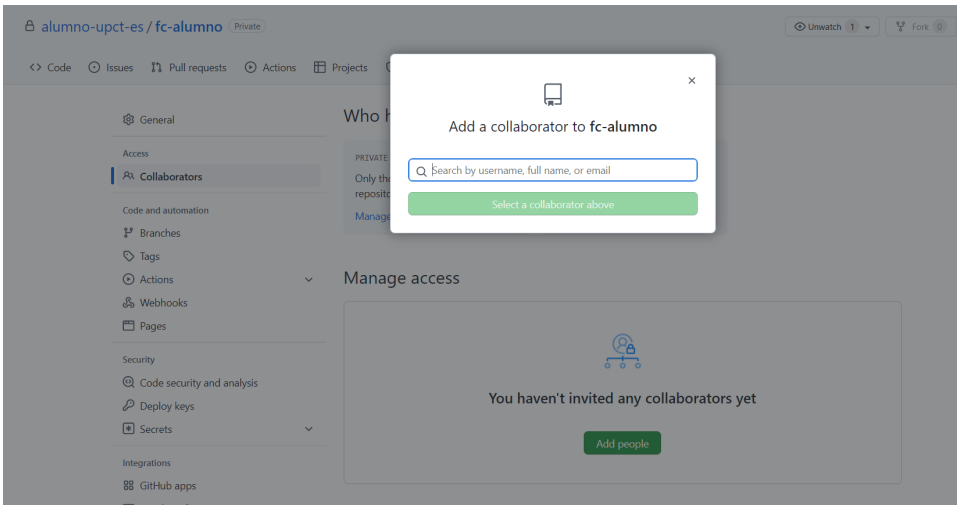


Figura I.17: Añade un colaborador en un repositorio de Github