Práctica de Git: Instalación e uso básico

Nome e apelidos:

Esta práctica será avaliada individualmente observando o traballo nos repositorios públicos da conta de GitHub que especifiques a continuación:

URL á túa conta de GitHub:

Por exemplo: https://github.com/very-serious-dev

SECCIÓN 1: CREACIÓN DUNHA CONTA EN GITHUB

GitHub é a plataforma web de compartición de código máis popular a día de hoxe.

Git é unha ferramenta de control de versións creada por Linus Torvalds en 2005.

Non son o mesmo.

Vamos a aprender sobre o uso de Git, e aínda que empregaremos GitHub para publicar os nosos repositorios de código, existen moitas alternativas.

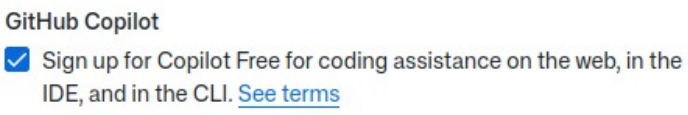
⚠ Se xa tes unha conta de GitHub e queres usar esa conta para traballar nesta práctica podes facelo. Pega a URL no recadro inicial, e salta á Sección 2.

Para empezar, **accede** á web de GitHub:

- https://github.com

Despois, na parte superior dereita clica en “Sign up” e comeza o proceso de rexistro.

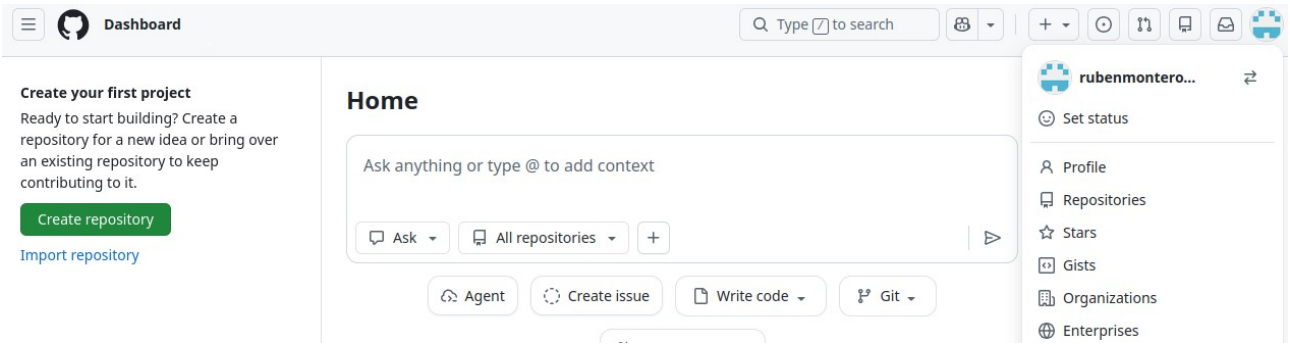
Non é preciso que te rexistres a partir dunha conta de Google ou de Apple. Se escolles a posibilidade de usar GitHub Copilot (asistente de IA):



...ten en conta que os termos de uso permiten a GitHub empregar os teus datos (o teu código) de forma libre para entrenar modelos de IA e existen riscos éticos e de seguridade asociados.

Despois de rexistrarte, noraboa! Tes unha conta de GitHub. Emprega esta conta máis aló das necesidades desta asignatura e pensa nela coma un “portfolio” do teu traballo e logros que será de utilidade, por exemplo, na búsqueda de emprego. Subir código “ben feito” non é o máis importante (ao igual que un músico non toca o instrumento de xeito perfecto dende o primeiro momento). O importante é empregalo con constancia para demostrar o teu interese e esforzo ao longo do tempo.

Agora, clica no botón dereito superior e verás un menú desplegable.



Clica en “Profile” para ver a páxina do teu perfil.

Unha vez esteas ahí, **copia** e **pega** a URL da páxina web no recadro inicial deste PDF, para entregalo.

SECCIÓN 2: INSTALACIÓN DE GIT

Descarga a ferramenta Git dende a páxina web oficial:

- https://git-scm.com/

...e despois, **instálaa** completando todos os pasos.

> 🧠 Durante a instalación de Git, especialmente en Windows, é posible teñas que tomar varias decisións de configuración. Investiga cada unha delas!

Se estás en Linux, é posible que a ferramenta xa veña instalada ou que a poidas instalar executando[1] “sudo apt install git” dende un terminal.

En calqueira caso, o obxectivo é que, dende un terminal calqueira, o comando “git” estea operativo no teu ordenador:

```
$ git -v
git version 2.43.0
```

1 Asumindo que aptitude é o teu xestor de paquetes

SECCIÓN 3: CREACIÓN DUN REPOSITORIO E CONFIGURACIÓN INICIAL

En Git, un "repositorio" é unha carpeta que contén o teu código.

Pero non unha carpeta calqueira. Unha carpeta con metadatos especiais (recopilados baixo unha subcarpeta oculta chamada ".git") que recopilan todos os cambios que se foron producindo no código ao longo do tempo.

Vamos a crear o noso primeiro repositorio.

Abre un terminal e crea unha nova carpeta chamada "practica_git". Podes facelo con **`mkdir practica_git`**.

Despois, cambia o directorio activo a dentro desa carpeta con **`cd practica_git`**.

E agora, inicializa esa carpeta coma repositorio Git executando **`git init`**.

❓ Eres capaz de averiguar que cambiou exactamente na túa carpeta "practica_git" despois de executar o comando "git init"?

Se executas **`git status`** obterás información do repositorio (información do control de cambios na carpeta). Seguramente verás algo coma isto:

```
C:\Users\Ruben\practica_git>git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Como primeiro paso vamos a configurar o noso nome e correo electrónico. Dese xeito, cando fagas un commit[2], este terá automáticamente a túa información reflectida no autor.

Supoñendo que te chamas Rubén Montero e o teu correo electrónico é ruben@edu.xunta.gal, podes facelo con estes dous comandos:

```
git config --global user.name "Rubén Montero"
git config --global user.email "ruben@edu.xunta.gal"
```

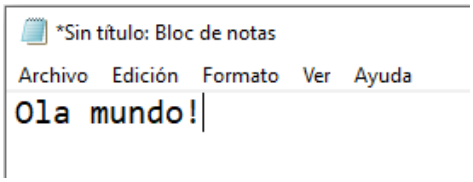
Executa eses dous comandos "git config" no teu terminal, adaptando o teu nome e correo aos teus datos persoais.

2 Pronto falaremos do que é un "commit"!

SECCIÓN 4: FACENDO ALGÚNS COMMITS

Agora vamos a cambiar o contido no repositorio “practica_git”.

Crea un novo arquivo de texto empregando o teu editor favorito (gedit, nano, Bloc de notas,…) e nese novo arquivo, escribe unha soa liña con *Ola mundo!*



Chama ao arquivo saudo.txt e **gárdao**.

Agora executa `git status` no terminal. Que é o que ves?

Git marca como “Untracked files” aqueles ficheiros que aínda non están rexistrados no control de versións. É dicir, son archivos normais. Non están salvados no “diario” do repositorio.

Vamos a salvar o noso valioso arquivo.

Executa `git add saudo.txt`. Con ese comando, acabamos de indicarlle a Git que temos interese en salvar o arquivo saudo.txt nun novo commit no control de versións.

❓ Executa de novo “git status”. De que cor sae agora o arquivo? Entendes o que indica Git?

Agora o arquivo xa está listo para pasar a ser parte dun novo commit.

Executa `git commit -m "Primeiro commit"`.

Seguramente verás algo como esto:

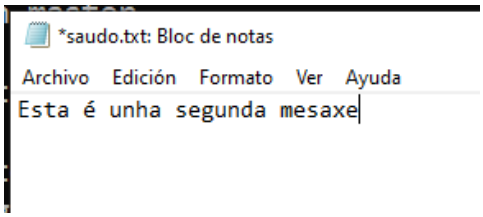
```
C:\Users\Ruben\practica_git>git commit -m "Primeiro commit"
[master (root-commit) 1503246] Primeiro commit
 1 file changed, 1 insertion(+)
 create mode 100644 saudo.txt
```

❓🧠 Con toda certeza, o número “1503246” é diferente no teu ordenador. Por que? De que se trata ese número? Cales son os datos que inflúen na súa xeración? Esta é unha pregunta digna de poñer a proba os teus coñecementos de Git. Asegúrate de averigualo, preguntando á IA se fai falla!

Xa tes feito o teu primeiro commit!

Posiblemente te preguntes: “Para que o fixen?”. Non cho preguntes. Mellor, fagamos outro commit.

Abre o ficheiro saudo.txt novamente e edítao. Agora, en lugar de *Ola mundo!* vamos poñer o contido: *Esta é unha segunda mesaxe*.



Gárdao e observa con `git status` que información obtés.

"Changed not staged for commit" é algo así coma "Untracked files". Nesta ocasión, fala de cambios feitos sobre arquivos xa rexistrados nalgún commit. Pero, en si, os novos cambios non están rexistrados.

Cando hai cambios en ficheiros, con `git diff` podemos ver sinxelamente que foi o que cambiou. Isto vai ser moi útil no futuro. Próbao, e verás algo coma o seguinte:

```
C:\Users\Ruben\practica_git>git diff
diff --git a/saudo.txt b/saudo.txt
index d4afd80..4cab513 100644
--- a/saudo.txt
+++ b/saudo.txt
@@ -1 +1 @@
-Ola mundo!
\ No newline at end of file
+Esta é unha segunda mesaxe
\ No newline at end of file
```

Repite os pasos `git add saudo.txt` e fai `git commit -m "Un segundo commit de XXXX"`, sendo **XXXX** o teu nome.

Agora terás dous commits no teu repositorio local.

SECCIÓN 5: QUE É REALMENTE O CONTROL DE VERSIÓNS?

Agora vamos a entender un poquiño mellor que é realmente un sistema de control de versións, e por que se fan commits.

Executa **`git log`**.

Verás un historial de todos os commits que se fixeron no repositorio, xunto a información de identificador (SHA) do commit, autor, data e mesaxe.

Asegúrate de que tes pechado o teu editor de texto con saudo.txt.

Agora, **copia** o identificador (SHA) do "Primeiro commit". É un numeriño moi largo. En moitos terminais basta con que fagas clic dereito para copialo ao portapapeis.

A continuación executa **`git show YYYY`**, sendo **`YYYY`** o identificador (SHA) que acabas de copiar.

Verás un informe dos cambios feitos nese commit.

> ⚠ Agora parece trivial ver que se engadiu a liña "Ola mundo!", pero conforme un proxecto avanza e se volve máis complexo, é moi útil ter a información que como se cambiaron os ficheiros, por quen, a que hora, e que mesaxe descritiva escribiu cando o fixo

Non só podemos "ver" os cambios salvados en dito commit. Tamén podemos revertir o contido do noso sistema de ficheiros (a nosa carpeta) ao estado no que se atopaba nese intre.

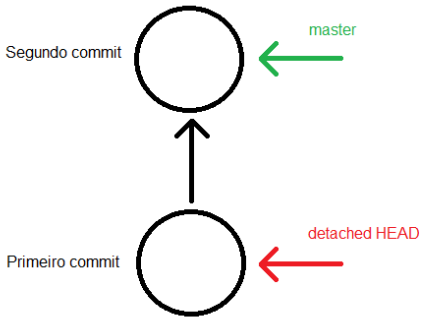
Executa **`git checkout YYYY`** sendo **`YYYY`** o identificador (SHA) anterior.

Aparecerá o seguinte aviso:

```
C:\Users\Ruben\practica_git>git checkout  1503246fe580510d51782fdde8f41ad251e189
62
Note: switching to '1503246fe580510d51782fdde8f41ad251e18962'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 1503246 Primeiro commit
```

Este aviso indícanos que o sistema de arquivos non está no punto máis recente da historia, o cal non é habitual ao traballar con Git.

O habitual é traballar no commit máis recente, porque o código vai evolucionando cara adiante. Se, temporalmente, "exploramos" un commit anterior, non hai ningún problema en facer iso pero estamos a viaxar "ao pasado", o cal chámase "detached HEAD" en Git.



Agora: **Abre** o arquivo saudo.txt. Cal é o seu contido?

Sorpréndete?

Non cambies nada. Non hai nada estropeado. Despois de pechar o arquivo, podes volver ao commit máis "recente" con `git checkout master`. Estás a traballar nunha rama (unha historia) chamada master, e master é, polo tanto, sempre a referencia ao commit máis recente.

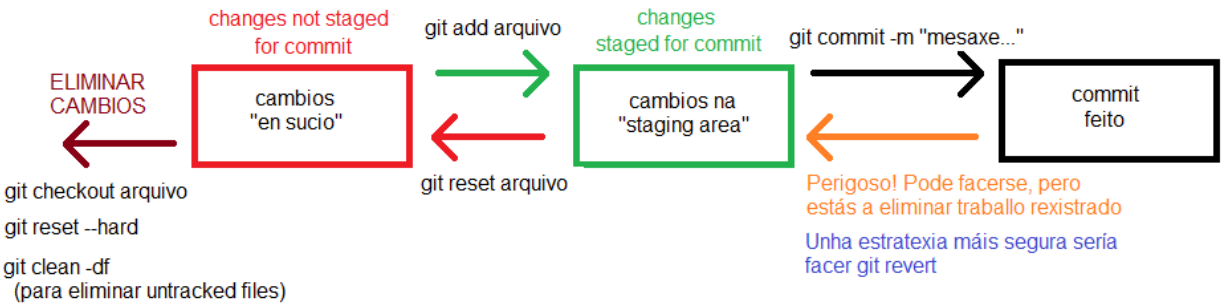
Abre de novo o arquivo saudo.txt e averigua o que contén.

Repasa a Sección 4 e a Sección 5 se cres que non interiorizaches ben o que se fixo.

SECCIÓN 6: COMPRENDENDO A STAGING AREA

Se xa entendes ben que un repositorio Git contén información de commits que almacenan cambios en ficheiros e se van producindo ao longo do tempo, entón vamos a profundizar un pouco máis niso da "staging area".

A grandes rasgos, o fluxo de traballo ten os seguintes estados cando se traballa en Git:



Vamos a comprobalo.

Modifica saudo.txt e pon a mesaxe: *Este texto vai durar pouco*. **Garda** o ficheiro

Que mostra `git status`?

Agora executa[3] `git checkout saudo.txt`

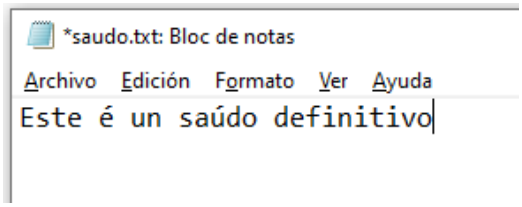
```
C:\Users\Ruben\practica_git>git checkout saudo.txt
Updated 1 path from the index
```

Recarga o arquivo saudo.txt desde o editor de texto (péchao e ábreo se o precisas). Que contén agora? Que indica novamente `git status`?

❓ Verifica que as modificacións en saudo.txt tamén se perden se fas git reset --hard. Despois, xa sabes! Non executes git reset --hard sen pensalo con coidado! Sempre mide dúas veces e corta unha

Agora, comprobaremos cómo funciona “git reset”.

Modifica saudo.txt e pon a seguinte mesaxe:



Gárdao.

A continuación, **crea** outro novo arquivo despedida.txt e escribe o contido *Ata pronto!*

Para engadir os dous arquivos á “staging area” nun só comando, executa `git add *`

Se executas “git status”, verás algo coma isto:

```
C:\Users\Ruben\practica_git>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   despedida.txt
        modified:   saudo.txt
```

3 Dícheste conta de que o comando “checkout” pode cumplir dous propósitos diferentes? Este uso non é o mesmo que cando o fixemos na Sección 5

Supoñamos agora que “non queremos meter os dous arquivos no mesmo commit”. Queremos facer dous commits separados. Vaia! Como executar agora “git commit” crearía un commit co cambio nos dous ficheiros, vamos a modificar o contido do “staging area”.

Fai **`git reset despedida.txt`**

Agora, “git status” darache algo coma o seguinte. **Compróbao**:

```
C:\Users\Ruben\practica_git>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   saudo.txt

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

Fai un **`git commit -m "Modificado saúdo"`**

Despois, **executa** os dous comandos necesarios para (1) engadir despedida.txt novamente á “staging area” (cor verde) e (2) crear un commit con ese arquivo, poñendo a mesaxe de commit “Engadida despedida.txt”

SECCIÓN 7: MÁIS COMMITS

Se xa entendiches ben o poder da “staging area” e como nos permite decidir os cambios da área de traballo a incluir no seguinte commit, entón completa as seguintes tarefas:

Fai 1 commit engadindo o arquivo “musica.txt” co contido *MELLORES CANCIÓNS*

Fai 5 commits, engadindo en cada un unha nova liña a musica.txt co nome dunha canción

Análogamente, **fai** outros 6 commits cun arquivo “pelis.txt” que comece por MELLORES PELIS e que teña 5 liñas de texto co título de 5 películas.

⚠ Se te equivocas nalgún commit, borra “practica_git” e volve a empezar o traballo desde a Sección 4, pois vaise avaliar que o repositorio ten exactamente os commits que se piden coa forma que se pide. Alternativamente, busca algún xeito de arreglalo (e.g.: git reset --hard HEAD~1), pero ten coidado!

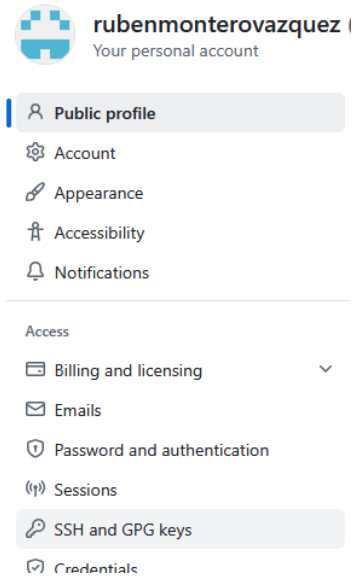
SECCIÓN 8: CONFIGURACIÓN SSH EN GITHUB

Agora xa debes ter algo máis de soltura cos comandos necesarios para traballar con git en local. Pero git non está pensado para usarse de xeito individual!

Abre https://github.com e loguéate coa túa conta.

No menú superior dereito, busca a opción “Settings”.

Dentro desa páxina, **clica** en SSH and GPG keys:



Dentro, verás un botón verde “New SSH Key”.

Se o clicas, haberá un campo de texto grande no que podemos pegar a nosa clave pública SSH.

Pero, tes unha clave pública?

Se non é o caso, abre un novo terminal e executa `ssh-keygen -t rsa`

Con ese comando, creas un novo par de claves SSH (pública/privada).

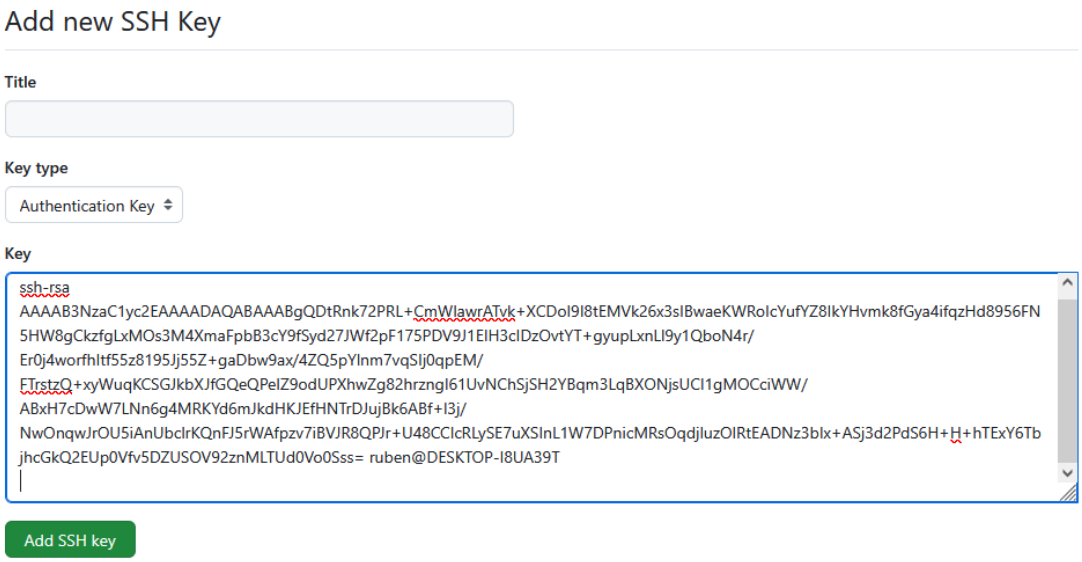
> ❓🧠 Por que hai dúas claves, pública e privada? É un bó momento para indagar sobre “arquitectura de clave asimétrica” no teu LLM favorito

Preguntaráche pola carpeta no disco na que se gardará o novo par de claves (deixa a ubicación por defecto) e por un novo contrasinal para cifrar con seguridade a clave privada (é aconsellable que o poñas, aínda que o podes deixar en branco).

Unha vez completes o comando, terás no teu directorio persoal unha subcarpeta .ssh na que hai dous novos arquivos: “id_rsa” e “id_rsa.pub”. Son, respectivamente, a túa clave privada e a túa clave pública SSH.

Abre cun editor de texto o ficheiro “id_rsa.pub”. **Copia** o contido (é longo).

Despois, volve á páxina de GitHub de engadir unha clave SSH e **pega** a esa clave pública no recadro de texto:

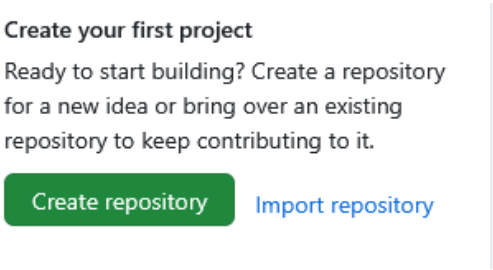


E fai **clic** en “Add SSH key”.

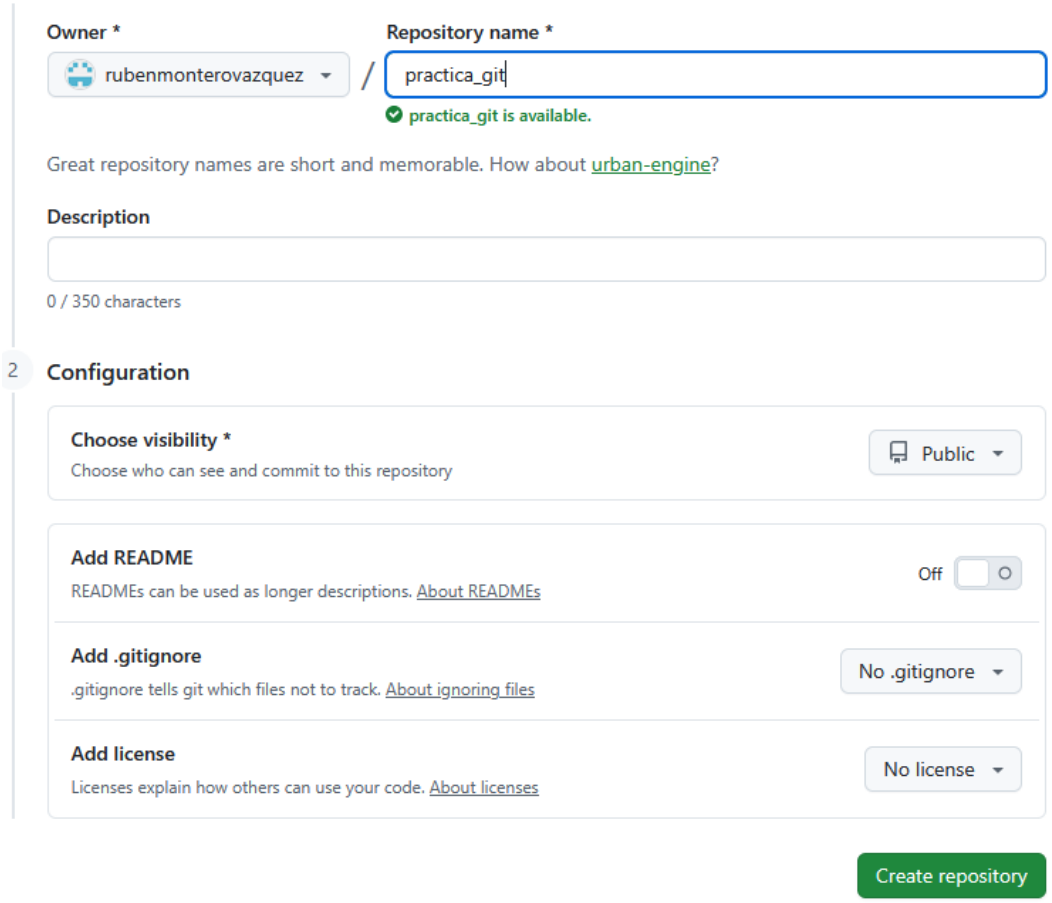
Se todo saíu ben, entón GitHub xa ten a túa clave pública e confía, polo tanto, en que os commits e a información que lle mandes son teus. Isto é así porque git emprega SSH por debaixo, e vaise encargar de utilizar a túa clave privada (“id_rsa”) para cifrar a información que envíes a GitHub.

SECCIÓN 9: SUBIR A REPOSITORIO REMOTO

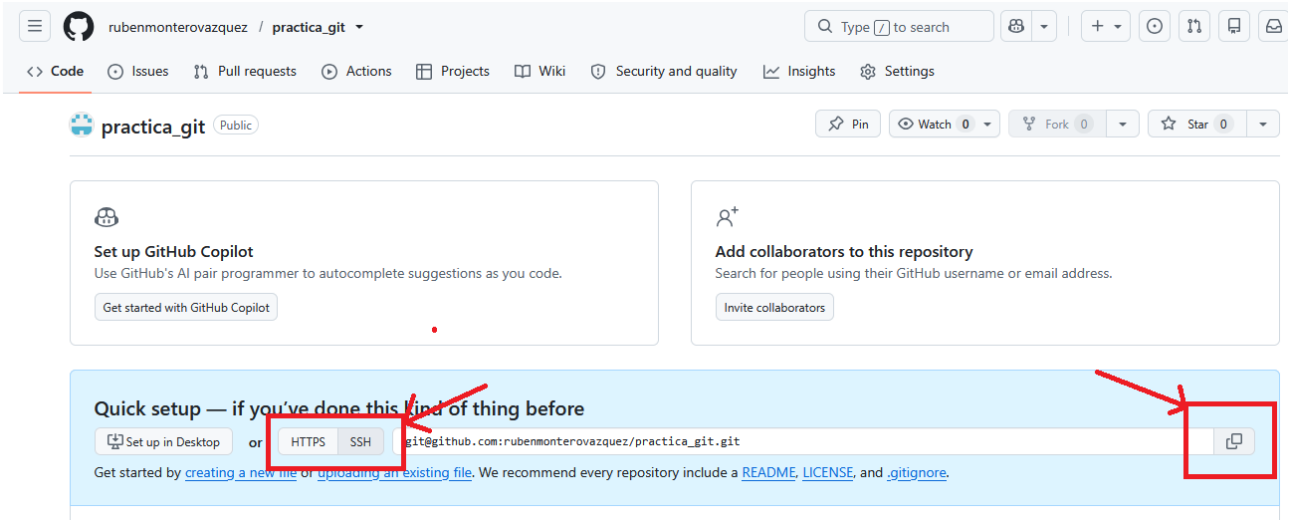
Desde https://github.com, logueado coa túa conta, **crea** un novo repositorio:



Vamos a chamalo “practica_git” (non sería obrigatorio que se chamase igual ca nosa carpeta local) e manter as opcións por defecto:



Cando estea creado, dentro da páxina do repositorio, **copia** a URL SSH (non a HTTP) do repositorio:



Vamos a regresar ao noso terminal e ao noso repositorio local, “practica_git”.

Executa `git remote -v`. Isto mostra os repositorios remotos aos que o noso repositorio local (carpeta) está vinculado. De xeito pouco sorprendente, o comando polo de agora non mostra nada.

Agora, executa `git remote add origin ZZZZ` sendo **ZZZZ** a URL SSH que copiaches no paso anterior de GitHub.

"git remote -v" mostra un resultado distinto agora?

Xa tes vinculada a túa carpeta local "practica_git" co repositorio remoto en GitHub.

Agora, tes dúas opcións:

(1) Facer `git push origin master`

...ou (2) facer `git push --set-upstream origin master` e despois `git push`.

> 🧠 Podes preguntarlle á IA a diferencia entre ambas, e que implicacións ten cada unha

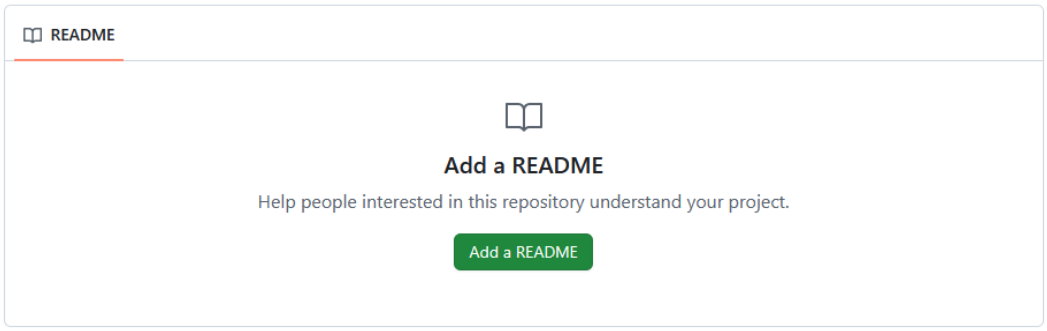
Se o fixeches con éxito, verás en GitHub os teus cambios dentro da páxina do proxecto.

Para rematar esta Sección, **engade** o texto que ti queiras a saudo.txt, **fai** un commit coa mesaxe que consideres oportuna e **sube** ("git push") os cambios ao repositorio remoto. Despois, **verifica** en GitHub que se subiu con éxito.

SECCIÓN 10: BAIXAR DO REPOSITORIO REMOTO

Agora vamos a probar a operación inversa a "push". Chámase "pull".

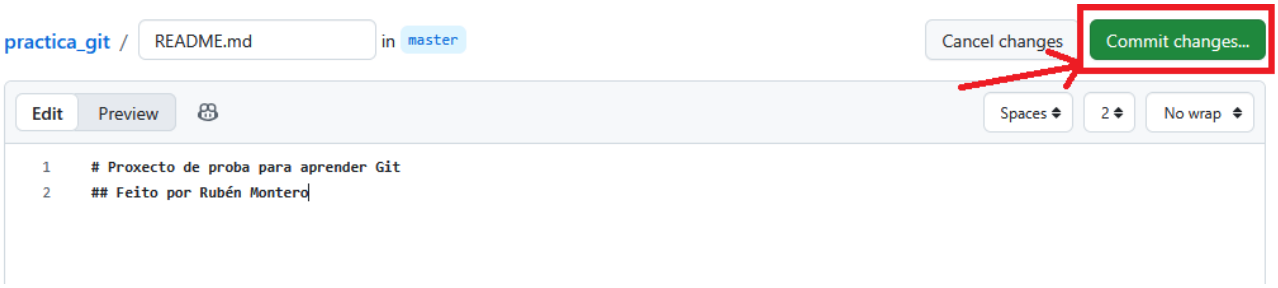
Desde https://github.com, navega á páxina do teu proxecto "practica_git" e verás isto:



Os arquivos README.md son unha convención para especificar en texto plano[4] as características do teu proxecto no repositorio: Que é, para que serve, como funciona, que tecnoloxías usa, etc.

4 En realidade, empregando formato Markdown

Agora, **clica** no botón verde “Add a README”. Despois, no editor web **escribe** un texto coma o seguinte (poñendo o teu nome en vez de Rubén Montero) e dalle a “**Commit changes**”



No diálogo que vai aparecer deixa a escolla “Commit directly to branch master” e **pulsa** no botón verde.

Acabas de engadir un novo commit á historia do repositorio! Contén un ficheiro README.md. Pero ese arquivo non está na carpeta do teu ordenador, verdade?

Desde un terminal na carpeta “practica_git”, executa `git branch -set-upstream-to=origin/master master` se non o seguiches o método (2) ao remate da Sección 9.

Despois executa `git pull`

Verás que traes os cambios do remoto a local.

Está agora o arquivo README.md na túa carpeta “practica_git” local?

SECCIÓN 11: GIT DESDE VSCODE

Agora vamos a probar a usar Git de forma visual.

Abre VSCode e **abre** a carpeta “practica_git”. Se inicialmente a carpeta ábrese en “Restricted Mode”, preme no botón que che permita configurar o proxecto coma “Trusted”

No panel da esquerda hai varios botóns útiles.

O primeiro botón, que xa coñeces, permíteche explorar a xerarquía de carpetas da carpeta que tes aberta.

O terceiro botón é o control de versións:

Preme nese botón e observa que, despois de cargar, dispós dunha ventaniña inferior que che mostra a historia de commits e unha ventaniña superior que che permite facer commit.

Modifica o arquivo README.md (engade unha liña co texto que ti queiras) e **gárdao**.

Agora, cal é o botón que che permite pasar ditos cambios á “staging area”? Cal é o botón que che permitiría devolver os cambios á “unstaged”? Se tes os cambios na “staging area”, como podes facer un commit? Como podes facer “push”? E “pull”?

Sube a modificación ao repositorio remoto nun novo commit feito dende VSCode.

SECCIÓN 12: EXTRA (NON AVALIABLE)

Que é unha rama (branch)? Como se crea unha?

Por que existen as ramas? Podes crear unha rama nova e facer varios commits nela?

Como se cambia entre ramas en local? Cando cambias de rama, aos cambios en sucio ou na “staging area” que lles pasa?

Que é “git merge”? Que estratexias de “merge” existen?

Que é unha “pull request”? (noutras plataformas coma GitLab chámase merge request)

Que é “git stash”?

Que é “git cherry-pick”?