

# Tutorial GIT básico

Este tutorial es una traducción resumida de [Git Tutorial](#), de [Lars Vogel](#).

## Contenido

Tutorial GIT básico.....	1
1. Git .....	3
1.1. ¿qué es Git?.....	3
1.2. Descarga & Instalación .....	4
1.3. Ejecutar & Utilizar GIT .....	4
1.4. Terminología importante .....	7
1.1. Staging index (el índice) .....	8
2. Configuración.....	8
2.1. Configuración del usuario .....	8
2.2. Crear el repositorio y hacer un commit .....	8
3. Bitbucket.....	10
3.1. ¿Qué es Bitbucket? .....	10
3.2. Registro en Bitbucket.....	10
3.2.1. Registro .....	10
3.2.2. Creando un Repositorio.....	11
3.2.3. Proyecto existente vs Empezando desde cero.....	12

## 1. Git

### 1.1. ¿qué es Git?

Git es un sistema de control de versiones distribuido (scvd) escrito en C. Un sistema de control de versiones permite la creación de una historia para una colección de archivos e incluye la funcionalidad para revertir la colección de archivos a otro estado. Otro estado puede significar a otra colección diferente de archivos o contenido diferente de los archivos.

puedes, por ejemplo, cambiar la colección de archivos a un estado de 2 días atrás, o puedes cambiar entre los estados para características experimentales y problemas de producción.

Esta colección de archivos generalmente es llamada "código fuente". En un sistema de control de versiones distribuido todos tienen una copia completa del código fuente (incluyendo la historia completa del código fuente) y puede realizar operaciones referidas al control de versiones mediante esa copia local. El uso de un scvd no requiere un repositorio central.

Si haces cambios al código fuente, haces esos cambios relevantes para el control de versiones (los agregas al staging index) y después los agregas al repositorio (commit).

Git mantiene todas las versiones. Por lo tanto, puedes revertir a cualquier punto en la historia de tu código fuente usando Git.

Git realiza los commits a tu repositorio local y es posible sincronizar ese repositorio con otros (tal vez remotos) repositorios. Git te permite clonar los repositorios, por ejemplo, crear una copia exacta de un repositorio incluyendo la historia completa del código fuente. Los dueños de los repositorios pueden sincronizar los cambios vía `push` (transfiere los cambios al repositorio remoto) o `pull` (obtiene los cambios desde un repositorio remoto).

Git soporta el concepto de `branching`, es decir, puedes tener varias versiones diferentes de tu código fuente. Si quieres desarrollar una nueva característica, puedes abrir un `branch` en

tu código fuente y hacer los cambios en este `branch` sin afectar el principal desarrollo de tu código.

## 1.2. Descarga & Instalación

El sitio oficial de GIT es <https://git-scm.com/download/win> donde se puede descargar la última versión, para la configuración que tengas, ya sea Windows, Linux, OS, etcétera.

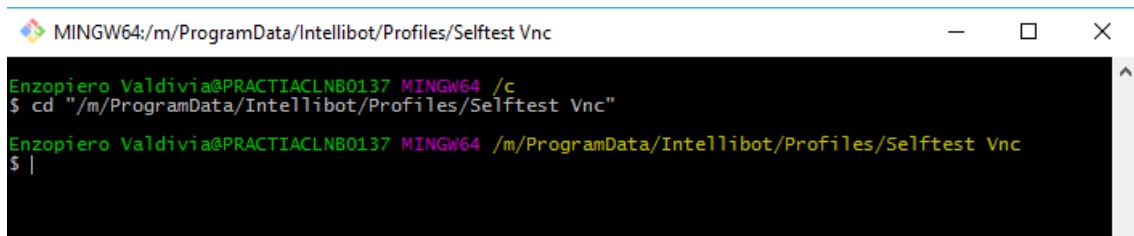
La instalación es como cualquier otro software y no tiene mayor complejidad por lo que no se verá en mayor detalle en este manual

## 1.3. Ejecutar & Utilizar GIT

Una vez hayamos instalado GIT en nuestro equipo y con la variación de si decidimos permitir que GIT se pueda ejecutar solo desde el bash de GIT o también desde la consola de Windows, tenemos 3 formas de ejecutar GIT para poder utilizar el control de versiones en nuestro PC.

Si es que seleccionamos durante la instalación que GIT se ejecutara desde la consola de Windows (*recomiendo hacer esto solo si tienen Windows 10 y en vez de la consola pueden utilizar **powershell**, ya que de otra manera la consola de Windows tiende a marearse con los comandos de GIT*), entonces para poder utilizar el mismo es tan simple como abrir la consola de Windows (Windows + R, powershell y ejecutar, o inicio -> ejecutar, escribir powershell y ejecutar), o abrir el bash de GIT en inicio -> GIT -> git Bash (se logra el mismo efecto yendo a la carpeta física que se quiere respaldar, click derecho “git bash here”).

Navegamos hasta la carpeta que queremos hacer control de versiones en este ejemplo iré al selftest de “VNC”:



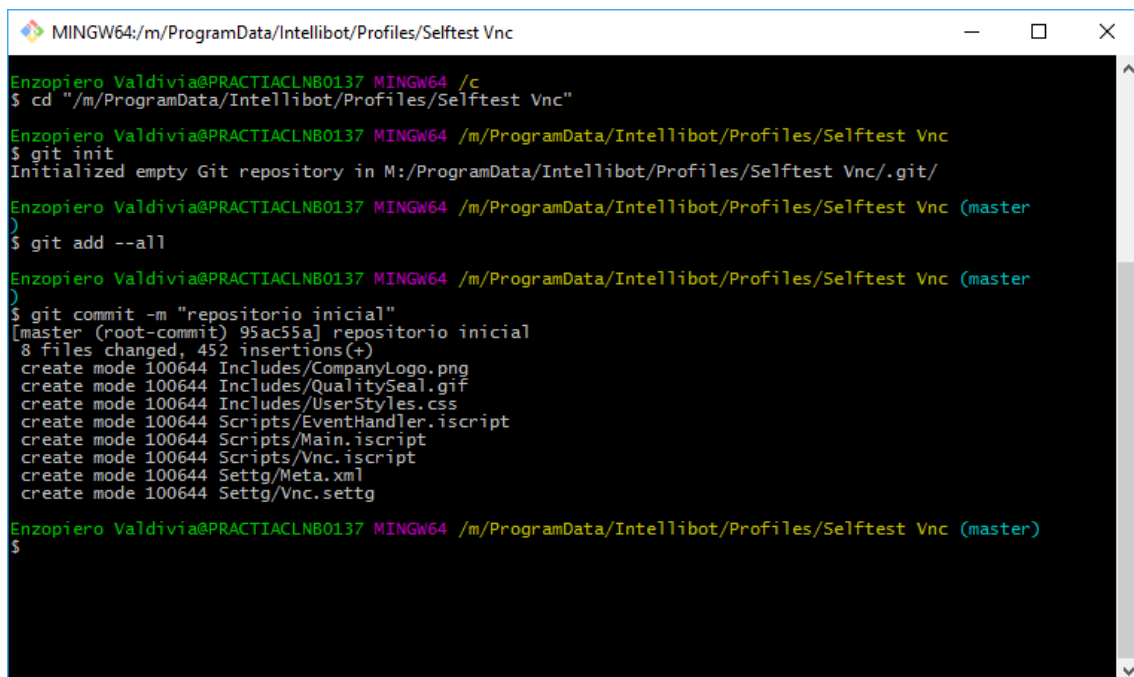
```

MINGW64:/m/ProgramData/IntelliBot/Profiles/Selftest Vnc
Enzopiero Valdivia@PRACTIACLN0137 MINGW64 /c
$ cd "/m/ProgramData/IntelliBot/Profiles/Selftest Vnc"
Enzopiero Valdivia@PRACTIACLN0137 MINGW64 /m/ProgramData/IntelliBot/Profiles/Selftest Vnc
$ |

```

Una vez hemos hecho la navegación hasta la carpeta podemos notar que ahora la ruta que sale en el encabezado ya no es la por defecto “/c” si no que nos marca a donde navegamos en este caso el perfil “Selftest VNC” (con toda su ruta).

Ahora que estamos en la carpeta solo nos queda iniciar un nuevo GIT



```

MINGW64:/m/ProgramData/IntelliBot/Profiles/Selftest Vnc
Enzopiero Valdivia@PRACTIACLN0137 MINGW64 /c
$ cd "/m/ProgramData/IntelliBot/Profiles/Selftest Vnc"
Enzopiero Valdivia@PRACTIACLN0137 MINGW64 /m/ProgramData/IntelliBot/Profiles/Selftest Vnc
$ git init
Initialized empty Git repository in M:/ProgramData/IntelliBot/Profiles/Selftest Vnc/.git/
Enzopiero Valdivia@PRACTIACLN0137 MINGW64 /m/ProgramData/IntelliBot/Profiles/Selftest Vnc (master)
$ git add --all
Enzopiero Valdivia@PRACTIACLN0137 MINGW64 /m/ProgramData/IntelliBot/Profiles/Selftest Vnc (master)
$ git commit -m "repositorio inicial"
[master (root-commit) 95ac55a] repositorio inicial
 8 files changed, 452 insertions(+)
 create mode 100644 Includes/CompanyLogo.png
 create mode 100644 Includes/QualitySeal.gif
 create mode 100644 Includes/UserStyles.css
 create mode 100644 Scripts/EventHandler.iscript
 create mode 100644 Scripts/Main.iscript
 create mode 100644 Scripts/Vnc.iscript
 create mode 100644 Settg/Meta.xml
 create mode 100644 Settg/Vnc.settg
Enzopiero Valdivia@PRACTIACLN0137 MINGW64 /m/ProgramData/IntelliBot/Profiles/Selftest Vnc (master)
$

```

Para explicar que hice iré por partes desde el “git init” hacia delante.

- git init: inicia una nueva instancia de git en la carpeta para generar el control de versiones, esta línea se ejecuta solo una vez y es como dije para iniciar el sistema.
- git add --all: lo que hace este comando es agregar todo lo que haya en este momento específico al “*staging index*” que es básicamente decir a git que saque una fotografía de todo lo que haya en el repositorio para guardar. A diferencia de “*git init*” este comando se utiliza cada vez que quieras hacer un commit y guardar una nueva versión, siempre habrá que

hacer un “*git add --all*” para agregar todo lo que haya en el repositorio a la versión.

- *git commit -m “<mensaje>”*: Al igual que “*git add --all*” el commit se ejecuta cada vez que quieras guardar una nueva versión, si “*add*” era sacar una fotografía del momento “*commit*” es guardar esa misma fotografía en el repositorio, al agregar el “*-m ‘<mensaje>’*” agregas de manera inmediata el mensaje que va asociado a esa versión (git no te permite commitear una versión sin asignar un comentario a este).

Para guardar este mismo repositorio en bitbucket sigue los pasos en el [punto 3](#).

## 1.4. Terminología importante

<b>Término</b>	<b>Definición</b>
Repositorio	Un repositorio contiene la historia, las diferentes versiones en el tiempo y todas los distintos branch y etiquetas. En Git, cada copia del repositorio es un repositorio completo. El repositorio te permite obtener revisiones en tu copia actual.
Branch	Un branch es una línea separada de código con su propia historia. Te es posible crear un nuevo branch a partir de uno existente y cambiar el código independientemente de otros branches. Uno de los branches es el original (generalmente llamado master). El usuario selecciona un branch y trabaja en ese branch seleccionado, el cual es llamado copia actual (working copy). Seleccionar un branch es llamado "obtener un branch" (checkout a branch)
Etiquetas (Tags)	Un tag (una etiqueta) apunta a un cierto punto en el tiempo en un branch específico. Con un tag, es posible tener un punto con un tiempo al cual siempre sea posible revertir el código, por ejemplo, ponerle el tag "testing" al código 25.01.2009
Commit	haces un commit de los cambios a un repositorio. Esto crea una revisión nueva la cual puede ser obtenida después, por ejemplo, si quieres ver el código fuente de una versión anterior. Cada commit contiene el autor y el que realizó el commit, siendo así posible identificar el origen del cambio. El autor y el comiteador pueden ser diferentes personas.
URL	Una URL en Git determina la ubicación de un repositorio.
Revisión	Representa una versión del código fuente. Git identifica revisiones con un id SHA1. Los id son de 160 bits de largo y son representados en hexadecimal. La última versión puede ser direccionada a través de "HEAD", la versión anterior mediante "HEAD-1" y así siguiendo.

## 1.1. Staging index (el índice)

Git requiere que los cambios sean marcados explícitamente para el próximo commit. Por ejemplo, si haces un cambio en un archivo y quieres que ese cambio sea relevante para el próximo commit, es necesario que agregues el archivo a lo que se llama comúnmente "staging index" mediante el comando `git add`. Dicho índice será una imagen completa de los cambios.

Los nuevos archivos siempre deben ser explícitamente añadidos al índice. Para archivos que ya han sido comiteados puedes usar la opción `-a` durante el commit.

## 2. Configuración

Git te permite almacenar configuraciones globales en el archivo `gitconfig`. Ese archivo queda ubicado en tu directorio de usuario (el home). Git almacena el comiter y el autor para cada commit. Esta y otra información adicional puede ser almacenada en la configuración global.

### 2.1. Configuración del usuario

Configura tu usuario e email para Git mediante los siguientes comandos:

```
# Configura el usuario que será usado por git
# Obviamente deberías usar tu nombre
git config --global user.name "John doe"
# Lo mismo para el correo electrónico
git config --global user.email "correo@mail.com "
```

### 2.2. Crear el repositorio y hacer un commit

Cada repositorio Git es almacenado en la carpeta `.git` del directorio en el cual el repositorio ha sido creado. Este directorio contiene la historia completa del repositorio. El archivo `.git/config` contiene la configuración local del repositorio.



## 2.3. Crear repositorio GIT

```
#Navegamos hasta la carpeta a iniciar
cd M:\programdata\intellinot\profile\perfil
# Inicializamos el repositorio local Git
git init
#Agregamos todo (archivos y directorios) al repositorio
git add --all
# Hacemos un commit al repositorio
git commit -m "Commit inicial"
# Muestra el log (un historial)
git log
```

### 3. Bitbucket

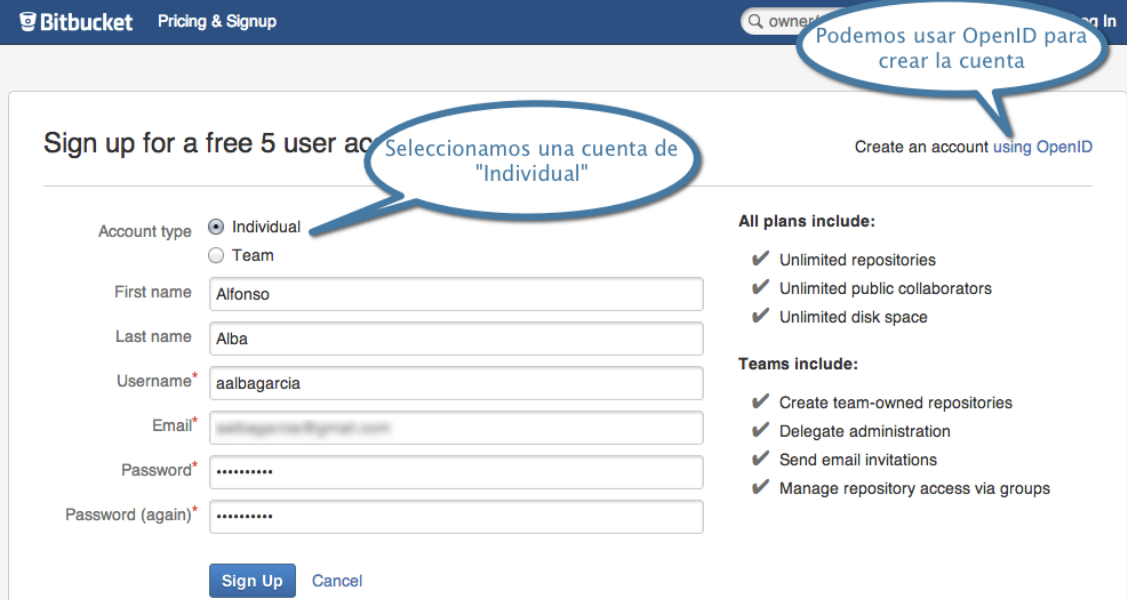
#### 3.1. ¿Qué es Bitbucket?

**Bitbucket** es un servicio de alojamiento basado en web, para los proyectos que utilizan el sistema de control de versiones Mercurial y Git. Es gratuito y te permite una cantidad ilimitada de repositorios privados y hasta 5 usuarios compartidos de manera gratuita.

#### 3.2. Registro en Bitbucket

##### 3.2.1. Registro

Si ya dispones de una cuenta, puedes pasar directamente al siguiente paso. Lo primero que haremos será acceder a la página de registro y completar el formulario que nos aparece en pantalla:



The screenshot shows the Bitbucket 'Sign up for a free 5 user account' page. The header includes the Bitbucket logo, 'Pricing & Signup', and a search bar. A blue callout bubble points to the 'Sign up' link in the top right, stating 'Podemos usar OpenID para crear la cuenta'. Another blue callout bubble points to the 'Individual' radio button under 'Account type', stating 'Seleccionamos una cuenta de "Individual"'. The form fields include: 'First name' (Alfonso), 'Last name' (Alba), 'Username\*' (aalbagarcia), 'Email\*' (aalbagarcia@gmail.com), 'Password\*', and 'Password (again)\*'. To the right, under 'All plans include:', there are three checked items: 'Unlimited repositories', 'Unlimited public collaborators', and 'Unlimited disk space'. Under 'Teams include:', there are four checked items: 'Create team-owned repositories', 'Delegate administration', 'Send email invitations', and 'Manage repository access via groups'. At the bottom are 'Sign Up' and 'Cancel' buttons.

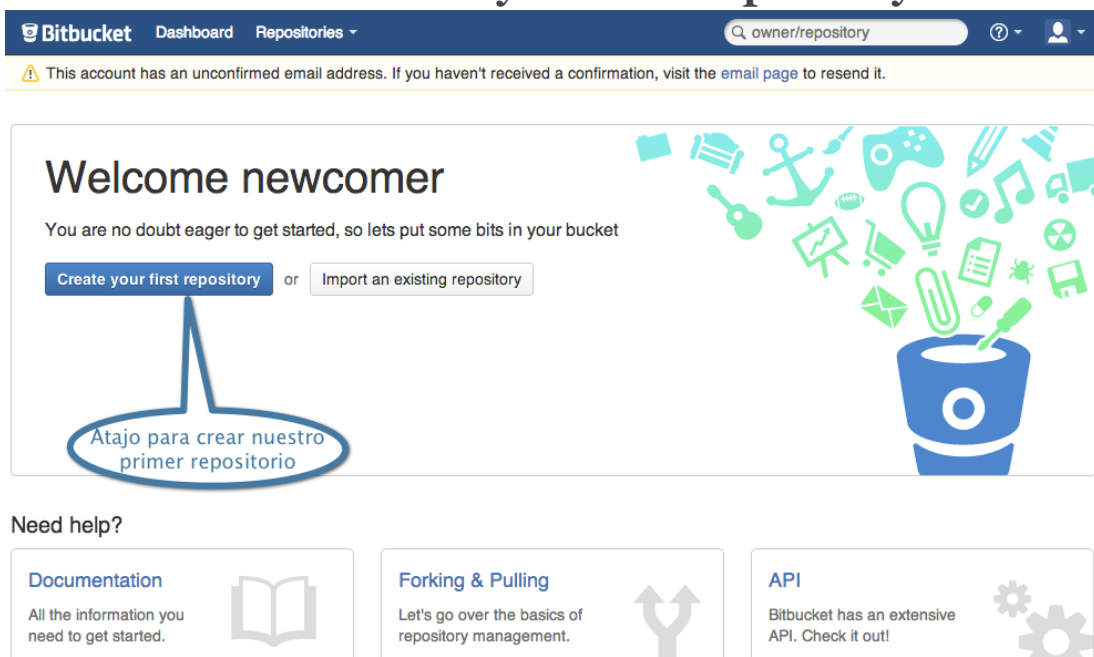
- El nombre de usuario que escojas es el que te identificará en la aplicación. Cualquier actividad que realices en Bitbucket, tanto pública como privada, mostrará este nombre de usuario como un enlace a tu perfil. Actualmente estamos utilizando de manera general el mismo usuario en nuestro correo electrónico de Practia más “Practia” por ejemplo: en mi caso mi usuario sería “EvaldiviaPractia”.
- Como lo que queremos es crear nuestra cuenta de usuario personal, vamos a seleccionar “Individual”.

Sigamos con el proceso de registro normal rellenando el formulario y enviándolo, haciendo clic en SignUp. Cuando lo hagamos, ocurrirán dos cosas:

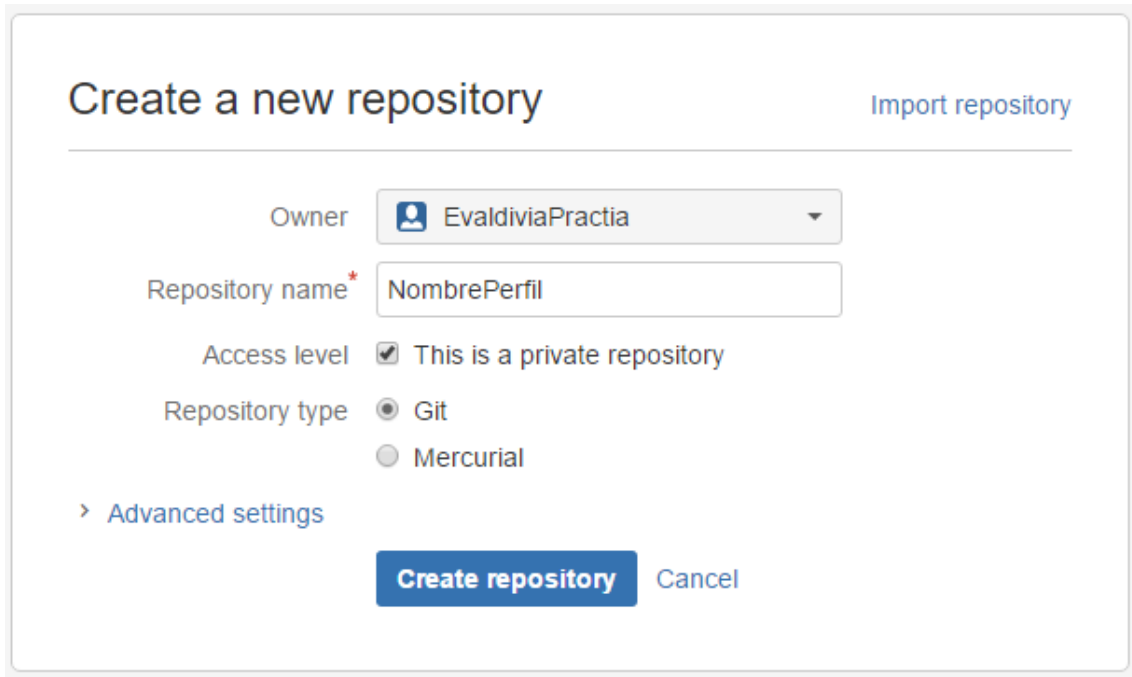
- Se nos remitirá un correo electrónico para confirmar nuestra cuenta, cosa que deberemos hacer para acceder a toda la funcionalidad de Bitbucket
- El sistema nos logeará automáticamente, presentándonos la pantalla de bienvenida

### 3.2.2. Creando un Repositorio

Antes de seguir, debemos revisar el correo con el cual nos creamos la cuenta y confirmar la cuenta que hemos utilizado. Una vez lo hagamos, volvemos a esta pantalla en la que haremos clic sobre el botón **“Create your first repository”**.



Llegamos a la página de creación de repositorios donde veremos algo así:



The screenshot shows a web form titled "Create a new repository" with a link for "Import repository". The form contains the following fields and options:

- Owner:** A dropdown menu showing "EvaldiviaPractia".
- Repository name\*:** A text input field containing "NombrePerfil".
- Access level:** A checkbox labeled "This is a private repository" which is checked.
- Repository type:** Two radio buttons, "Git" (selected) and "Mercurial".
- Advanced settings:** A link with a right-pointing arrow.
- Buttons:** "Create repository" (blue) and "Cancel" (grey).

- **Owner:** es el dueño del repositorio en este caso quién está creando el mismo
- **Repository Name:** es el nombre del repositorio en este caso sería útil usar el nombre del perfil que vamos a respaldar.
- **Access Level:** si es para PosRobot siempre usaremos repositorio privado.
- **Repository Type:** BitBucket trabaja con Git y con Mercurial, de momento nosotros solo usaremos "Git".

Una vez que creamos nuestro repositorio podemos empezar a usar para respaldar nuestro perfil.

### 3.2.3. Proyecto existente vs Empezando desde cero

Una vez creado nuestro repositorio podemos dentro de nuestra configuración hacer 2 cosas crear un proyecto desde cero "[I'm starting from scratch](#)" o "[I have an existing project](#)" como nosotros lo que queremos hacer es respaldar nuestros perfiles y mantener un control de versiones del mismo lo que haremos será subir a bitbucket un repositorio creado en nuestra maquina eso quiere decir que por ejemplo subiremos a bitbucket el repositorio que creamos para el perfil de "easy".

### Get started the easy way

Creating a README or a .gitignore is a quick and easy way to get something into your repository.

Create a README

Create a .gitignore

### Get started with command line

▼ I have an existing project

Step 1: Switch to your repository's directory

```
1 cd /path/to/your/repo
```

Step 2: Connect your existing repository to Bitbucket

```
1 git remote add origin https://EvaldiviaPractia@bitbucket.org/EvaldiviaPractia/nombreperfil.git
2 git push -u origin master
```

> I'm starting from scratch

Acá es la parte importante, como llegamos a este punto luego de ya haber creado nuestro repositorio lo que tenemos que hacer es en GIT navegar hasta el repositorio que vamos a respaldar en este ejemplo “easy” como se ve en el primer paso “cd /path/to/your/repo” luego es tan solo correr las dos líneas que bitbucket nos entrega. La primera lo que hará será asignar a mi repositorio local una dirección remota para respaldar en la nube, y la siguiente línea lo que hace es pasar todos estos datos a bitbucket.

De esta forma cuando ya trabajemos y queramos guardar una nueva versión del repositorio solo tenemos que hacer “git push origin master” para enviar los datos de manera automática a bitbucket.