



# FACULTAD DE INGENIERÍA

---

## INGENIERÍA DE SISTEMAS COMPUTACIONALES

### “Guía: Primeros pasos con Git”

#### Práctica de Campo 1

#### Grupo 12:

ASMAT DE LA CRUZ, ANTHONI BRYAN (N00405749)

CAMPOS LIPA, JOHN LENIN (N00400607)

LACHIRA PINGO, MIGUEL ORLANDO (N00268489)

ROJAS YSLA, DAVID ALEXANDER (N00331763)

ROMERO BARRETO, JUAN LUIS (N00399596)

#### Curso:

Técnicas de Programación Orientada a Objetos

#### Docente:

Martín Eduardo Torres Martínez

Lima – Perú

2025-2

Contenido

GIT .....2

1. Descarga de Git.....3

2. Creación y clonación de un repositorio local .....5

    Crear un repositorio local de GitHub.....5

    Clonar un repositorio de GitHub en nuestra maquina local .....6

3. Navegación básica por el Repositorio .....7

4. Commit (realizar cambios en el código).....8

5. Comandos *git add* y *git commit*. .....9

6. Historial de commits. ....10

7. Manejo de Branches y Merges.....11

    Crear y Cambiar entre Ramas (branches) .....11

    Fusión de Branches. ....12

URL web - Repositorio “Spoon-Knife” clonado desde GitHub:.....12

## GIT

Git es un sistema de control de versiones que permite la colaboración simultánea de archivos, principalmente códigos fuentes en diferentes lenguajes de programación. Fue creada en el año 2005 por Linus Torvalds, con el objetivo de gestionar proyectos de software con múltiples colaboradores distribuidos a nivel internacional.



Es una herramienta fundamental para cualquier desarrollador hoy en día, ya que nos permite tener un control de las versiones sobre la realización de un proyecto, es decir, es como “historial” o “backup inteligente” para nuestros archivos de código.

¿Para qué nos sirve?

- Guardar diferentes versiones del proyecto.
- Trabajar en equipo sin pisar el trabajo de otros miembros del proyecto.
- Experimentar sin miedo a romper el código.
- Volver atrás si cometes un error.

Asimismo, nos permite trabajar con plataformas en la nube basadas en Git, tales como GitHub, Gitlab, etc. Estas plataformas en la nube vienen a ser como una red social para desarrolladores y herramientas de colaboración para proyectos.

**GitHub**, al ser una plataforma online en la nube, permite la colaboración a través de *pull request*, revisión de códigos, entre otras acciones; asimismo, permite guardar *commits* subidos desde la computadora (se verá más adelante).

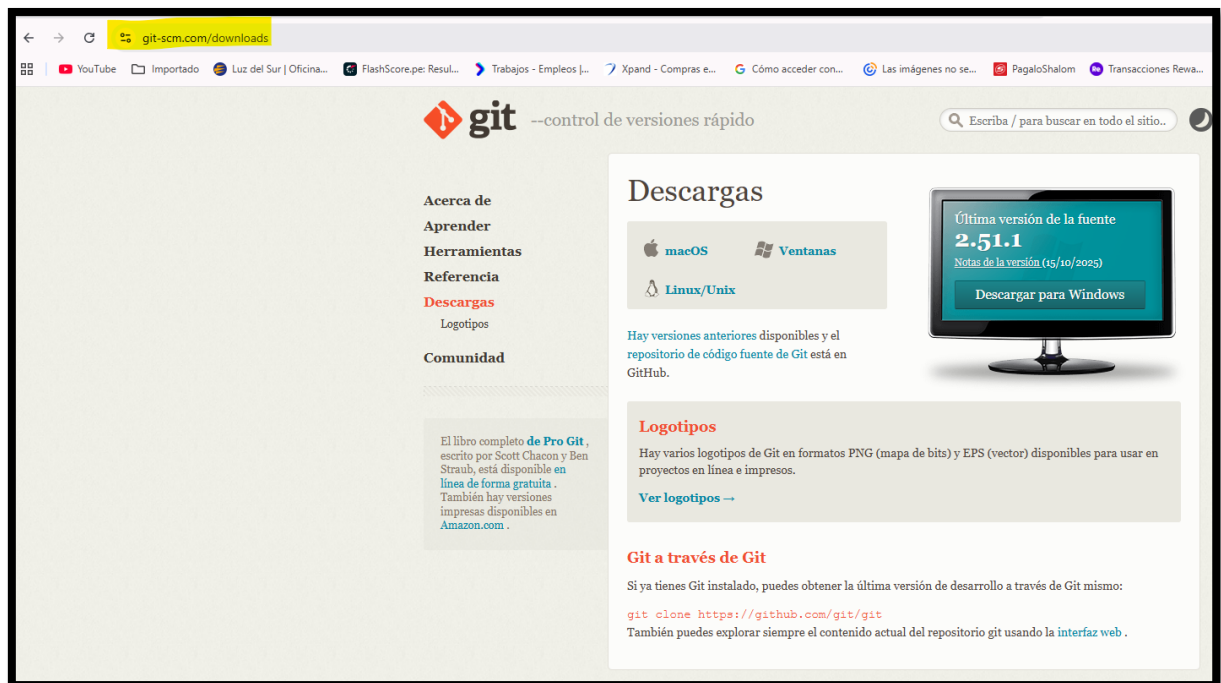
A continuación, comenzaremos la Guía de los primeros pasos para utilizar Git:

## 1. Descarga de Git

Primero empezaremos descargando Git en nuestra estación de trabajo, a partir del siguiente link:

<https://git-scm.com/downloads>

Aquí descargamos la versión según la plataforma de nuestro equipo.



Terminada la instalación, buscamos el terminal Gitbash, es con esa terminal donde iniciaremos a utilizar los comandos, ahora ¿porque trabajar con Gitbash y no con otras opciones con interfaces graficas de Git como Git GUI, GitHubDesktop?



A continuación, mencionamos los puntos más importantes de utilizar Gitbash y no un entorno grafico:

### Gitbash

- Mismos comandos en Windows/Mac/Linux
- Más rápido que interfaces graficas
- Acceso a todos los comandos de Git
- Entiendes lo que realmente pasa

## Interfaces Graficas

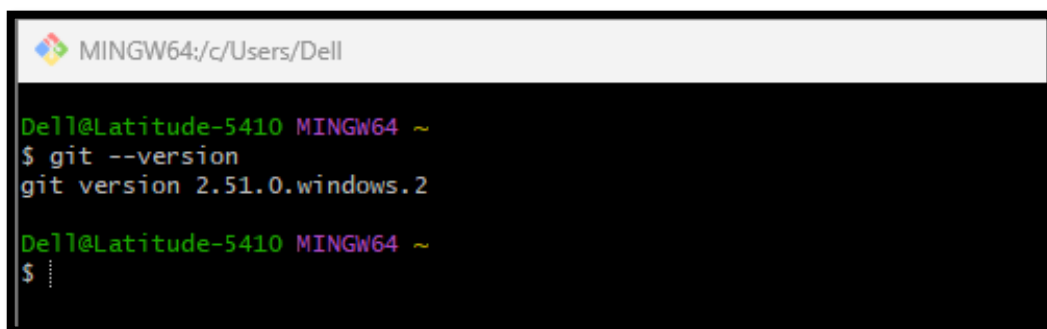
- No muestra todos los conceptos
- Limitada en funciones avanzadas

Mencionado estos puntos, iniciamos la terminal Gitbash



Git necesita saber quiénes somos y cómo preferimos trabajar. Para ello, necesitamos configurar Git:

- Primero necesitamos validar la versión de Git y ver si está correctamente instalado.

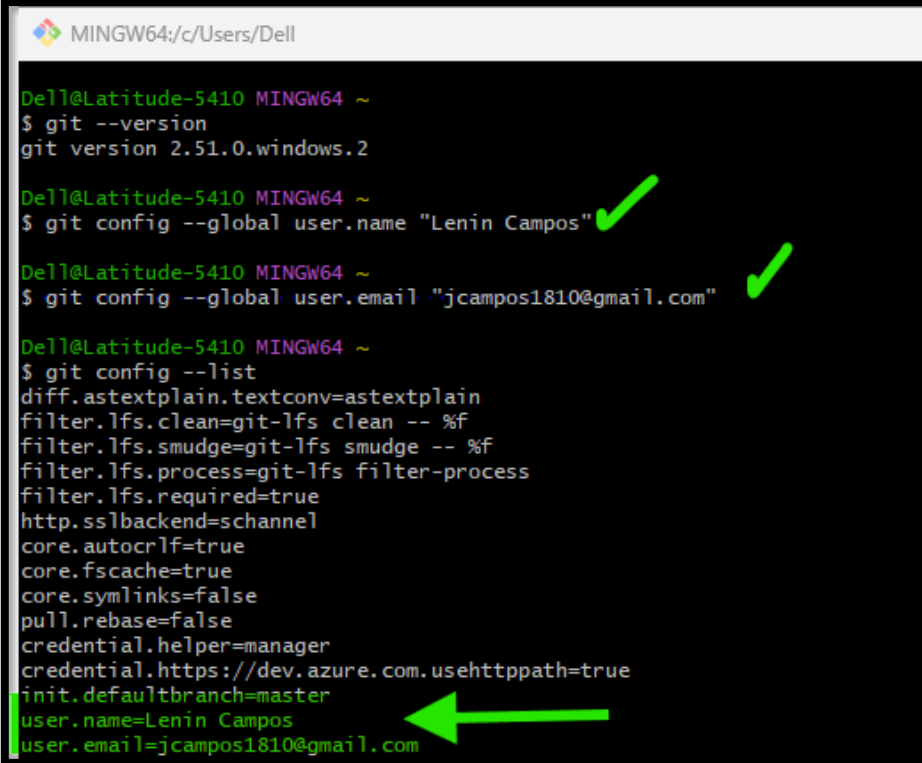


Comandos usados:

### ➔ *git --version*

(Este comando nos permite saber la versión de git instalada y nos indica si Git se encuentra instalado correctamente. Tomar en cuenta que luego de “git” se coloca un “espacio” seguido por dos guiones “--”).

- Ahora configuramos Git por primera y única vez en el equipo. Para efectos prácticos, se observará el paso a paso, con el ejemplo del alumno Lenin Campos. Tal como se puede observar:



```

MINGW64:/c/Users/Dell

Dell@Latitude-5410 MINGW64 ~
$ git --version
git version 2.51.0.windows.2

Dell@Latitude-5410 MINGW64 ~
$ git config --global user.name "Lenin Campos" ✓

Dell@Latitude-5410 MINGW64 ~
$ git config --global user.email "jcampos1810@gmail.com" ✓

Dell@Latitude-5410 MINGW64 ~
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=schannel
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=Lenin Campos
user.email=jcampos1810@gmail.com
  
```

Configuración por parte del alumno Lenin Campos

Comandos usados:

- ➔ *git config --global user.name "Lenin Campos"*
- ➔ *git config --global user.email "[jcampos18@gmail.com](mailto:jcampos18@gmail.com)"*

(Colocar el email con el que nos registramos en la plataforma GitHub).

- ➔ *git config --list*

(Nos permite validar la configuración).

## 2. Creación y clonación de un repositorio local

### Crear un repositorio local de GitHub

Ahora crearemos nuestro primer repositorio local:

```
Dell@Latitude-5410 MINGW64 ~/Desktop/helloGit
$ touch Holagit.py

Dell@Latitude-5410 MINGW64 ~/Desktop/helloGit
$ git init
Initialized empty Git repository in C:/Users/Dell/Desktop/HelloGit/.git/
```

Comandos usados:

➔ **Mkdir “HelloGit”**

(Primero creamos un directorio(carpeta), en este caso la ruta es la siguiente c:/Users/Dell/Desktop/), (Ingresamos al directorio **“HelloGit”**, este será nuestro repositorio local).

➔ **Git init**

(Aquí estamos creando un repositorio de Git, ósea el directorio **“HelloGit”** ahora trabajara con un control de versiones).

Donde está el (.git) nos indica que hemos creado un repositorio de Git

*Initialized empty Git repository in C:/Users/Dell/Desktop/HelloGit/.git/*

### Clonar un repositorio de GitHub en nuestra maquina local

Para ellos tenemos que crear una carpeta nueva carpeta, en este caso la llamaremos “Repositorio GitHub”, aquí es donde clonaremos el repositorio de GitHub.

```
Dell@Latitude-5410 MINGW64 ~/Desktop
$ mkdir "Repositorio GitHub"

Dell@Latitude-5410 MINGW64 ~/Desktop
$ ls
Administrador de configuración de SQL Server 2022.Ink*  PowerPoint.Ink*          'Visual Studio 2022.Ink*'
Excel.Ink*      'Proteus 9 Demonstration.Ink*'  'Visual Studio Code.Ink*'
'Files Compartido/'  'Repositorio GitHub/'      Word.Ink*
'LINDO 6.1.Ink*'    'SQL Server Management Studio 21.Ink*'  desktop.ini
'Office 2021 LTSC/'  'Tiempo de Servicio.pdf'
'Utilitarios Lenin/'
```

Comandos usados:

➔ **mkdir**

(Crea una nueva carpeta, ya que es ahí donde se encontrará el repositorio clonado de GitHub).

➔ **ls**

(Me permite ver el listado de un directorio en este caso “Desktop”).

➔ **cd**

(Me permite cambiar a un directorio, navegar entre carpetas en un mismo sistema de archivos, en este caso nos desplazamos a la carpeta “Repositorio GitHub”).

Ahora clonaremos el repositorio público creado por GitHub.

```
Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub
git clone https://github.com/octocat/Spoon-Knife.git
Cloning into 'Spoon-Knife'...
remote: Enumerating objects: 16, done.
remote: Total 16 (delta 0), reused 0 (delta 0), pack-reused 16 (from 1)
Receiving objects: 100% (16/16), done.
Resolving deltas: 100% (3/3), done.
```

→ **git clone**

(Mediante este comando + la URL(<https://github.com/octocat/Spoon-Knife.git>))

Como se puede observar, Git confirma que está creando una carpeta llamada “Spoon-Knife” y copiara los archivos del proyecto; asimismo, muestra el total de objetos que se enviaron desde GitHub hasta la maquina local. Como es un repositorio de ejemplo muy pequeño, son solo 16 objetos en total. Por último, indica que nuestro equipo descargó completamente los datos del repositorio “100% transferencia con éxito”.

### 3. Navegación básica por el Repositorio

Ahora navegaremos por el repositorio clonado desde GitHub:

```
MINGW64:/c/Users/Dell/Desktop/Repositorio GitHub/Spoon-Knife

Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub
$ ls
Spoon-Knife/

Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub
$ cd Spoon-Knife

Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub/Spoon-Knife (main)
$ ls -la
total 14
drwxr-xr-x 1 Dell 197121  0 Oct 17 12:15 ./
drwxr-xr-x 1 Dell 197121  0 Oct 17 12:15 ../
drwxr-xr-x 1 Dell 197121  0 Oct 17 12:15 .git/
-rw-r--r-- 1 Dell 197121 789 Oct 17 12:15 README.md
-rw-r--r-- 1 Dell 197121 375 Oct 17 12:15 index.html
-rw-r--r-- 1 Dell 197121 273 Oct 17 12:15 styles.css

Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub/Spoon-Knife (main)
$ git branch
* main
```

Comandos usados:

→ **ls**

(Validamos que este el repositorio Spoon-Knife se encuentre en la carpeta “Repositorio GitHub”, **ls** nos permite mostrar los directorios).

→ **cd**

(Ingresaremos al repositorio Spoon-Knife, repositorio clonado desde GitHub)



**→ *ls -la***

(Nos permite visualizar todos los archivos del proyecto, incluyendo los ocultos, aquí encontramos el (.git) y nos indica que es un repositorio en Git).

**→ *git branch***

(Para validar en que rama nos encontramos, en este caso estamos en “main”, el asterisco (\*) indica la rama actual, por defecto, cuando clonas un repositorio, estas en la rama principal (main o master)).

```
Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub/Spoon-Knife (main)
$ git remote -v
origin https://github.com/octocat/Spoon-Knife.git (fetch)
origin https://github.com/octocat/Spoon-Knife.git (push)
Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub/Spoon-Knife (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.
nothing to commit, working tree clean
```

**→ *git remote -v***

(Ver la información del repositorio remoto, este comando que nuestro repositorio local está conectado al de GitHub, Origen es el nombre por defecto del repositorio remoto).

**→ *git status***

(Ver el estado del repositorio, como observamos nos indica que no hay cambios pendientes y estamos sincronizado con el repositorio).

#### **4. Commit (realizar cambios en el código)**

El commit permite guardar uno o más cambios (versiones) en el historial del proyecto o repositorio trabajado.

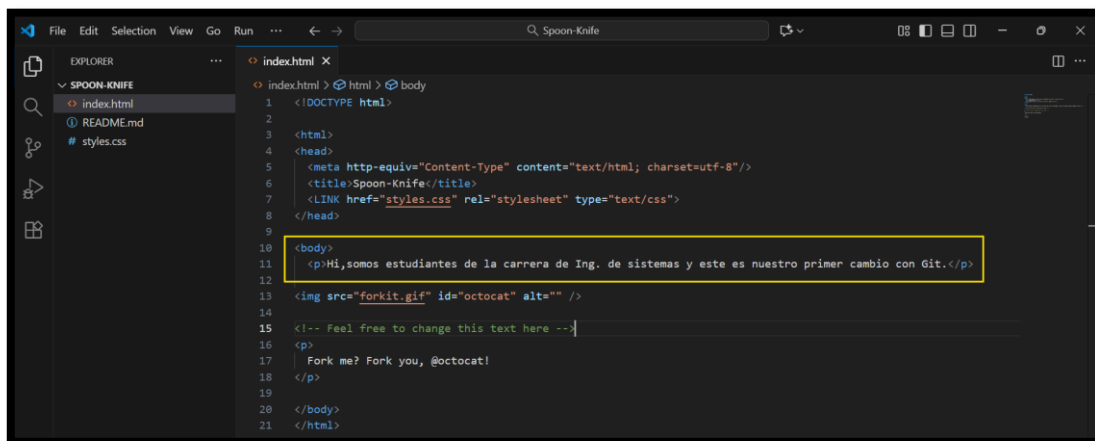
Ahora realizaremos nuestro primer commit, para ello necesitamos abrir el proyecto clonado desde GitHub “Spoon-Knife”.

```
Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub/Spoon-Knife (main)
$ code .
```

**→ *code .***

(Eso abrirá la carpeta Spoon-Knife en visual studio code)

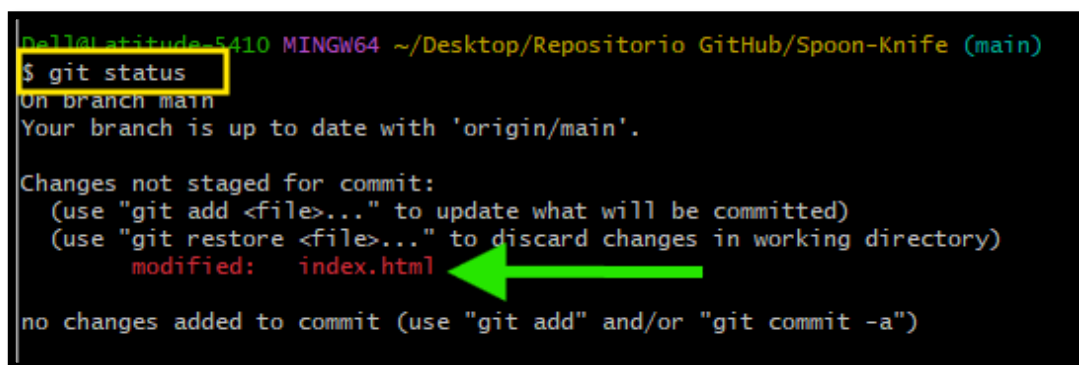
Aquí modificaremos el archivo index.html y agregaremos dentro de <body> una línea:



```

1 <!DOCTYPE html>
2
3 <html>
4 <head>
5   <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
6   <title>Spoon-Knife</title>
7   <link href="styles.css" rel="stylesheet" type="text/css">
8 </head>
9
10 <body>
11   <p>Hi, somos estudiantes de la carrera de Ing. de sistemas y este es nuestro primer cambio con Git.</p>
12
13   
14
15   <!-- Feel free to change this text here -->
16   <p>
17     Fork me? Fork you, @octocat!
18   </p>
19
20 </body>
21 </html>
  
```

Guardamos los cambios (Ctrl + S), ahora nos dirigimos a la terminal Git Bash.



```

Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub/Spoon-Knife (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

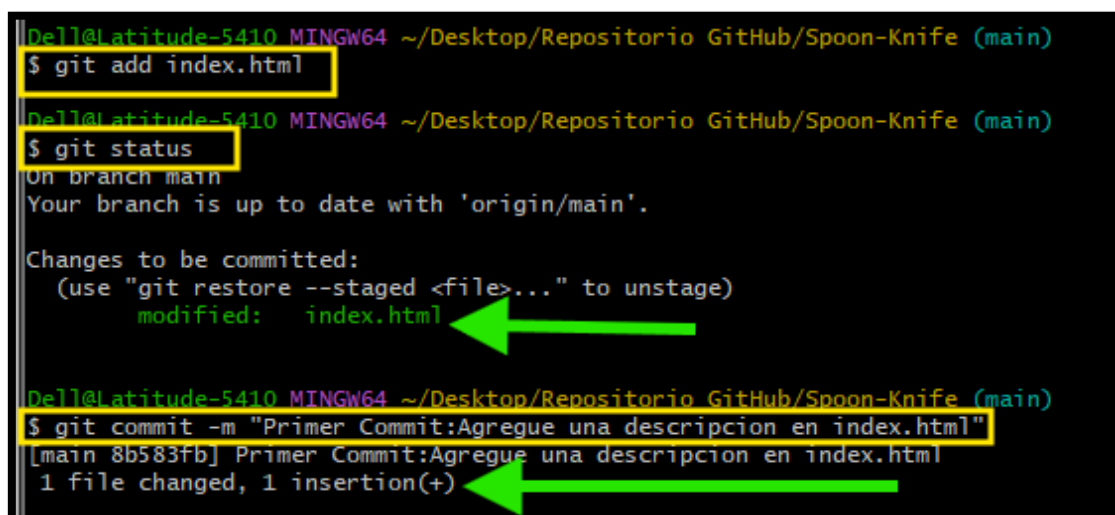
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html
no changes added to commit (use "git add" and/or "git commit -a")
  
```

➔ *git status*

(Verificamos los cambios en Git, que detecta que el archivo index.html fue modificado).

## 5. Comandos *git add* y *git commit*.

Ahora que Git detecto que el archivo index.html fue modificado procederemos a realizar nuestro primer commit, pero antes de ello tenemos que realizar lo siguiente:



```

Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub/Spoon-Knife (main)
$ git add index.html

Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub/Spoon-Knife (main)
$ 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:   index.html

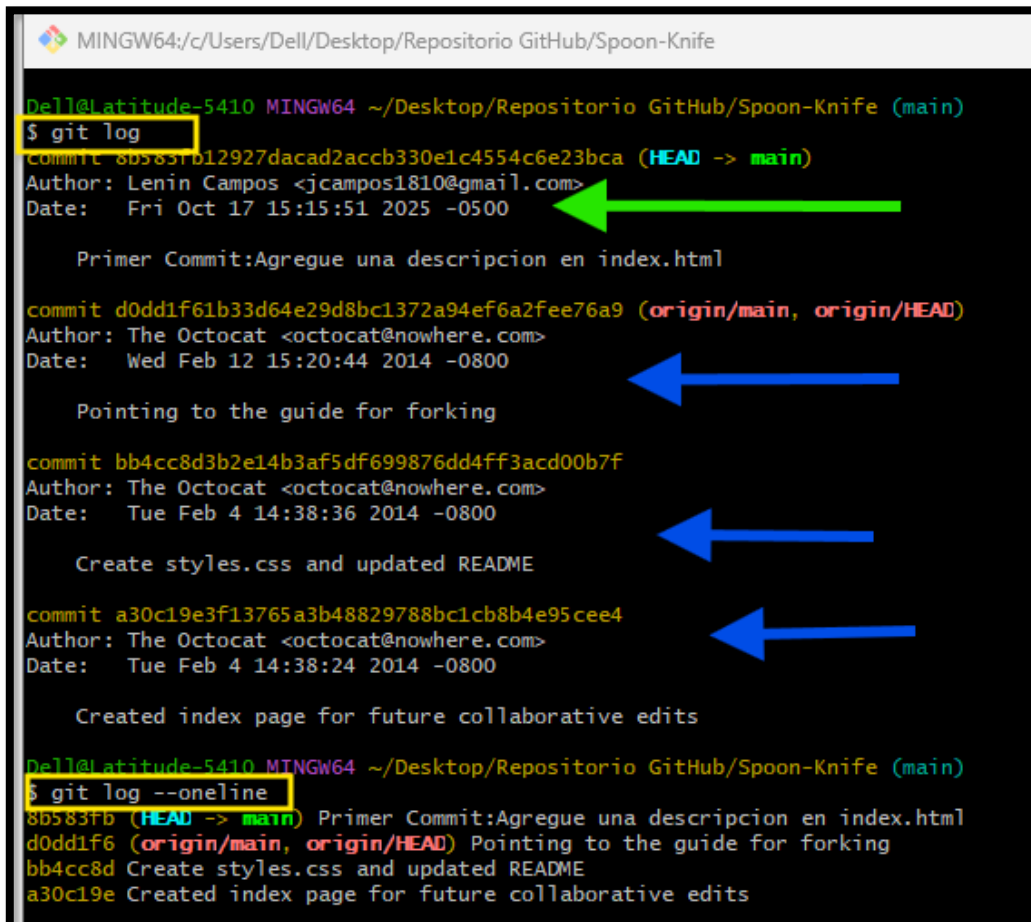
Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub/Spoon-Knife (main)
$ git commit -m "Primer Commit:Agregue una descripcion en index.html"
[main 8b583fb] Primer Commit:Agregue una descripcion en index.html
1 file changed, 1 insertion(+)
  
```

Comandos usados:

- ➔ **git add**  
 (Este comando nos permite agregar los cambios al área de preparación (Staging), en este caso el archivo indes.html no se encuentra en STAGING).
- ➔ **git status**  
 (Validamos nuevamente el estado del repositorio y podemos verificar que el archivo index.html se encuentra en STAGING, es decir, ese archivo está listo para ser incluido en el próximo commit).
- ➔ **git commit -m "comentario"**  
 (Con este comando registramos oficialmente el cambio, junto a ello un comentario descriptivo de dicho cambio. 1 fila modificado, 1 inserción).

## 6. Historial de commits.

Ahora verificamos el historial del commit:



```

MINGW64:/c/Users/Dell/Desktop/Repositorio GitHub/Spoon-Knife
Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub/Spoon-Knife (main)
$ git log
commit 8b583fb12927dacad2accb330e1c4554c6e23bca (HEAD -> main)
Author: Lenin Campos <jcampos1810@gmail.com>
Date:   Fri Oct 17 15:15:51 2025 -0500

    Primer Commit:Agregue una descripcion en index.html

commit d0dd1f61b33d64e29d8bc1372a94ef6a2fee76a9 (origin/main, origin/HEAD)
Author: The Octocat <octocat@nowhere.com>
Date:   Wed Feb 12 15:20:44 2014 -0800

    Pointing to the guide for forking

commit bb4cc8d3b2e14b3af5df699876dd4ff3acd00b7f
Author: The Octocat <octocat@nowhere.com>
Date:   Tue Feb 4 14:38:36 2014 -0800

    Create styles.css and updated README

commit a30c19e3f13765a3b48829788bc1cb8b4e95cee4
Author: The Octocat <octocat@nowhere.com>
Date:   Tue Feb 4 14:38:24 2014 -0800

    Created index page for future collaborative edits

Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub/Spoon-Knife (main)
$ git log --oneline
8b583fb (HEAD -> main) Primer Commit:Agregue una descripcion en index.html
d0dd1f6 (origin/main, origin/HEAD) Pointing to the guide for forking
bb4cc8d Create styles.css and updated README
a30c19e Created index page for future collaborative edits
  
```

- ➔ **git log**  
 (Nos muestra el historial de los commit)  
 El ultimo commit que observamos es justamente el que acabamos de realizar y lo observamos en el primer lugar con los datos de autor y datos como fecha y hora del commit.

Los otros commit que observamos son el historial de commit que tiene el repositorio Spoon-Knife, esta fue clonada de GitHub.

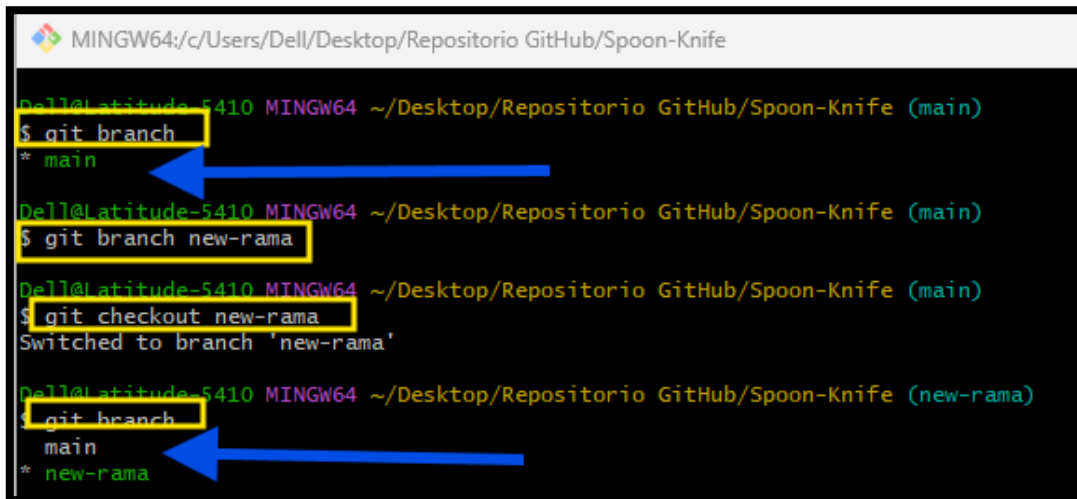
➔ ***git log –oneline***

(Esta es una versión compacta y más legible del historial de commits, es decir, nos permite ver una línea por commit, adicional nos muestra una legibilidad rápida y clara).

## 7. Manejo de Branches y Merges

### Crear y Cambiar entre Ramas (branches)

Ahora comenzaremos creando una nueva rama(branch), esto lo realizaremos en el repositorio clonado de GitHub “Spoon-Knife”,



```

MINGW64:/c/Users/Dell/Desktop/Repositorio GitHub/Spoon-Knife
Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub/Spoon-Knife (main)
$ git branch
* main

Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub/Spoon-Knife (main)
$ git branch new-rama

Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub/Spoon-Knife (main)
$ git checkout new-rama
Switched to branch 'new-rama'

Dell@Latitude-5410 MINGW64 ~/Desktop/Repositorio GitHub/Spoon-Knife (new-rama)
$ git branch
main
* new-rama
  
```

Comandos usados:

➔ ***git branch***

(Nos permite validar en que rama nos encontramos, en este caso en la rama “main”, adicional el asterisco (\*) nos indica que es la rama actual).

➔ ***git branch new-rama***

(Nos permite crear una nueva rama, pero aún no nos cambiamos a ella)

➔ ***git checkout new-rama***

(Este comando nos permite cambiarnos a la nueva rama, en este caso “new-rama”)

➔ ***git branch***

(Validamos nuevamente en que rama nos encontramos y efectivamente ya nos encontramos en la nueva rama “new-rama”)

### Fusión de Branches.

La fusión de branches (merge) es un proceso de combinar los cambios de dos ramas diferentes en una sola, integrando el trabajo realizado por separado, es decir, unir el trabajo de dos líneas de desarrollo en una versión unificada del proyecto.

#### **Propósito principal:**

Integrar cambios de ramas de desarrollo (features, fixes, experimentos) a la rama principal (main o master) una vez que están probados y listos.

**URL web - Repositorio “Spoon-Knife” clonado desde GitHub:**

<https://github.com/octocat/Spoon-Knife.git>