




Bootcamp de Desarrollo Web

Bootcamp Desarrollo Web 1

1



Bootcamp de Desarrollo Web

Clase 4

Configuración del entorno de trabajo

Bootcamp Desarrollo Web 2

2



¿Qué es la línea de comandos?

La línea de comandos, también conocida como interfaz de línea de comandos (**CLI por sus siglas en inglés, Command Line Interface**), es una forma de interactuar con una computadora a través de texto, mediante comandos escritos en un intérprete de comandos o shell.

En lugar de utilizar interfaces gráficas como ventanas, iconos y menús (**GUI, Graphical User Interface**), la línea de comandos permite a los usuarios enviar instrucciones al sistema operativo o a programas específicos escribiendo comandos directamente en una ventana de terminal o consola.



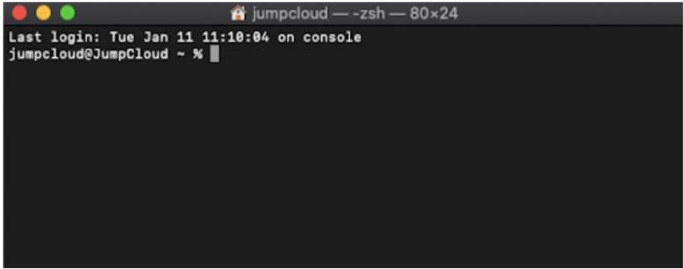
```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\Users\JaviPas> echo "hola hatakeros"
hola hatakeros
PS C:\Users\JaviPas>
```

Windows
PowerShell

Linux
CLI

```
john@ubuntu: ~/john_directory
john@ubuntu:~$ pwd
/home/john
john@ubuntu:~$ ls
john_directory john_file
john@ubuntu:~$ cd john_directory
john@ubuntu:~/john_directory$ history
1  pwd
2  ls
3  cd john_directory
4  history
john@ubuntu:~/john_directory$
```




A screenshot of a terminal window titled "jumpcloud — zsh — 80x24". The terminal shows the text "Last login: Tue Jan 11 11:10:04 on console" and the prompt "jumpcloud@JumpCloud ~ %".

MAC OS CLI

Bootcamp Desarrollo Web 5

5



A small icon of a terminal window with a green background and a white cursor.

¿Cómo funciona?

En un entorno de línea de comandos, los usuarios escriben comandos específicos, seguidos de argumentos o parámetros opcionales, para realizar diversas tareas, como manipular archivos, ejecutar programas, administrar el sistema, configurar opciones y más.

Cada sistema operativo tiene su propio intérprete de comandos, como Bash en sistemas basados en Unix/Linux, Command Prompt en Windows, PowerShell (más avanzado) en Windows, entre otros.

Bootcamp Desarrollo Web 6

6



Texto plano vs. texto enriquecido

En el mundo de la programación, cómo almacenamos y representamos datos es esencial.

Dos métodos comunes son el texto plano y el texto formateado, ambos con sus propias características y usos específicos.



Texto plano

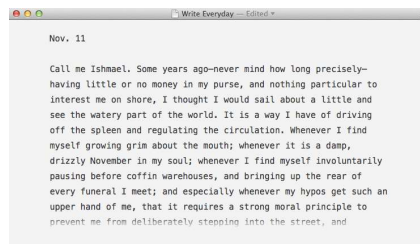
En el mundo de la programación, cómo almacenamos y representamos datos es esencial.

Dos métodos comunes son el texto plano y el texto formateado, ambos con sus propias características y usos específicos.



Texto plano

- El texto plano se refiere a datos sin formatos específicos.
- Es una secuencia de caracteres sin ningún tipo de formato adicional.



Texto enriquecido

- El texto formateado incluye estilos, colores, tamaños de fuente y otros elementos visuales.
- Se utiliza comunmente para mejorar la presentación visual y destacar información específica.

FOR RENT

1488 Villa Piña Way #201

2 bed 1.5 bath 900ft²

\$1,550 per month

Spanish-style condo in great location

Won't last! Available June 1

Contact Claire Vasquez @ [\(double-check cell #\)](#)



Resumen

- El texto plano contiene solo texto sin formato, con saltos de línea y espaciado para separar ideas o secciones, pero no incluye estilos visuales como negritas o colores.
- El texto plano no puede ser marcado con estilos visuales o formatos específicos para resaltar o dar énfasis a partes específicas del texto.
- En contraste, el texto enriquecido se compone de texto que ha sido estilizado con elementos como variaciones en el tamaño de la fuente, formatos como negritas o cursivas, y colores, agregando un aspecto visual al contenido.
- Determinar la diferencia entre un documento de texto plano y uno de texto enriquecido se basa en observar si el texto presenta estilos visuales y formatos o si simplemente se compone de texto sin este tipo de añadidos.

11



Bootcamp de Desarrollo Web

Clase 4

Símbolo del Sistema

12



Símbolo del Sistema

- El símbolo del sistema, representado como el símbolo "\$", es el espacio donde introduces los comandos que deseas que tu computadora ejecute.
- Es importante tener en cuenta que el símbolo puede variar ligeramente entre diferentes sistemas operativos. A veces, puede ser un "%" o un "#". Este símbolo, conocido como command prompt en inglés, nos indica que el ordenador está listo para recibir y ejecutar comandos.

13



¡Conócete a ti mismo!

Whoami – regresa el nombre de usuario

```
ebanos@MacBook-Pro-de-Eduardo:~  
Last login: Tue Dec 19 23:09:35 on console  
ebanos@MacBook-Pro-de-Eduardo ~ % whoami  
ebanos  
ebanos@MacBook-Pro-de-Eduardo ~ %
```

14



Comando **MAN**

Para obtener información y detalles sobre casi cualquier comando en el sistema, existe una herramienta clave, el comando **man**. Su uso es simple; basta con escribir **man** seguido del nombre del comando que desees investigar.

Por ejemplo, **man whoami** te proporcionará información detallada sobre el comando **whoami**. Incluso puedes utilizar **man man** para conocer más acerca de cómo utilizar esta útil herramienta.

```
man whoami
WHOAMI(1)                                General Commands Manual                                WHOAMI(1)

NAME
  whoami - display effective user id

SYNOPSIS
  whoami

DESCRIPTION
  The whoami utility has been obsoleted by the id(1) utility, and is equivalent to "id -un". The command "id -p" is suggested for normal interactive use.

  The whoami utility displays your effective user ID as a name.

EXIT STATUS
  The whoami utility exits 0 on success, and >0 if an error occurs.

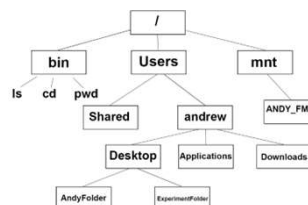
SEE ALSO
  id(1)
```



Sistema de archivos

En tu computadora, los archivos están estructurados en un sistema jerárquico de archivos. Esto implica la existencia de una carpeta principal o "raíz" en tu sistema. Esta carpeta principal contiene otras carpetas, y a su vez, estas carpetas contienen más carpetas, creando así una estructura en niveles.

Este sistema se compone de una organización en capas, donde cada carpeta puede contener archivos o subcarpetas, formando una estructura de árbol que facilita la organización y el acceso a la información en tu computadora.





PWD

Deberías obtener una salida como **/Users/tu-nombre-de-usuario**.

Eso significa que estás en el directorio

tu-nombre-de-usuario en la carpeta **Users** dentro del directorio **/** o raíz. Este directorio se llama a menudo el directorio **"home"**.

En Windows, tu salida sería en cambio **C:/Users/tu-nombre-de-usuario**. La carpeta en la que se encuentra se llama **directorio de trabajo**, y **pwd** significa **"imprimir el directorio de trabajo"**. La palabra **"imprimir"** puede ser algo engañosa.

El comando **pwd** no imprimirá realmente nada excepto en tu pantalla. Este comando es más fácil de entender cuando interpretamos **"print"** como **"display"** en inglés.



PWD

```
ebanos@MacBook-Pro-de-Eduardo:~  
ebanos@MacBook-Pro-de-Eduardo ~$ pwd  
/Users/ebanos  
ebanos@MacBook-Pro-de-Eduardo ~$
```



Explorando Archivos y Carpetas

Una vez ubicados en el directorio `tu-nombre-de-usuario` o directorio de trabajo, ¿cómo podemos conocer su contenido?

Utilizamos el comando `ls`, que enumera las carpetas y archivos presentes en ese directorio.

Podrás observar una lista de carpetas como Documentos, Desktop, entre otras, y posiblemente algunos archivos. Este es el contenido del directorio actual.

```
ebanos@MacBook-Pro-de-Eduardo:~$ pwd
/Users/ebanos
ebanos@MacBook-Pro-de-Eduardo:~$ ls
Applications Developer  Downloads  Movies  Desktop  Documents  Library  Music  Public  src  bash_profile  upResume
```



Explorando Archivos y Carpetas

Ahora, si queremos explorar la carpeta Desktop, utilizamos el comando `cd Desktop`. El comando `cd` nos permite "cambiar de directorio". Es importante escribir correctamente el nombre del directorio, respetando mayúsculas y minúsculas. Si el comando fue exitoso, no verás ninguna salida en pantalla, lo cual es normal en la línea de comandos.

¿Cómo sabemos si la navegación fue exitosa? Volvemos a utilizar nuestro comando `pwd`.

```
ebanos@MacBook-Pro-de-Eduardo:~/Developer$ ls
Applications Developer  Downloads  Movies  Desktop  Documents  Library  Music  Public  src  bash_profile  upResume
ebanos@MacBook-Pro-de-Eduardo:~/Developer$ cd Desktop
ebanos@MacBook-Pro-de-Eduardo:~/Developer/Desktop$ pwd
/Users/ebanos/Desktop
```



Explorando Archivos y Carpetas

Para desplazarte por el sistema de archivos, puedes utilizar el comando `cd ..`. Este comando te permite moverte hacia el directorio superior en la jerarquía del sistema de archivos. La expresión `..` siempre señala al directorio inmediatamente superior al actual.

Por ejemplo, si te encuentras en un subdirectorio, al ejecutar `cd ..`, ascenderás al directorio principal de ese subdirectorio.

```
shanos@MacBook-Pro-de-Eduardo:~
Applications  Developer  Downloads  Movies      Public      src
Desktop      Documents  Library    Music       Pictures    task_profile
shanos@MacBook-Pro-de-Eduardo  cd Developer
shanos@MacBook-Pro-de-Eduardo  ls
cssCrashCourse  test_site  typescript
shanos@MacBook-Pro-de-Eduardo  pwd
/Users/shanos/Developer
shanos@MacBook-Pro-de-Eduardo  cd ..
shanos@MacBook-Pro-de-Eduardo  pwd
/Users/shanos
shanos@MacBook-Pro-de-Eduardo  ls
```



Touch

El comando touch se utiliza para crear archivos vacíos. Este comando es útil cuando necesitas crear un archivo, pero aún no tienes ningún contenido que almacenar en él.

Al ejecutar el comando `touch foo.txt` desde la terminal, se creará un archivo de texto llamado `foo.txt` sin contenido. Si el comando es exitoso, no se mostrará ninguna salida en la terminal.

The screenshot shows the 'About This Mac' window with the 'Storage' tab active. The storage is at 85% capacity (10.5 GB of 12.3 GB). A list of files and folders is shown, including 'Applications', 'Developer', 'Downloads', 'Music', 'Pictures', 'Public', 'src', and 'ufpsname'. The file 'bash_profile' is highlighted, showing it is 1.5 KB and located in the 'Public' folder.



MKDIR

El comando **mkdir** en sistemas se utiliza para crear directorios o carpetas en el sistema de archivos. Su función es simple: generar una nueva carpeta con el nombre especificado.

Supongamos que queremos crear una carpeta llamada "**consola**" en el directorio actual. Para ello, ejecutamos el siguiente comando en la terminal o símbolo del sistema:

```
ebanos@MacBook-Pro-de-Eduardo:~/Developer$ ls
total 0
drwxr-xr-x  5 ebanos  staff  1600 Nov  6 12:41 cssCrashCourse
drwxr-xr-x  17 ebanos  staff  5440 Dec 14 18:56 test_site
drwxr-xr-x  4 ebanos  staff  1280 Nov  6 12:18 typescript
ebanos@MacBook-Pro-de-Eduardo:~/Developer$ mkdir consola
ebanos@MacBook-Pro-de-Eduardo:~/Developer$ ls
total 0
drwxr-xr-x  2 ebanos  staff   640 Dec 20 01:33 consola
drwxr-xr-x  5 ebanos  staff  1600 Nov  6 12:41 cssCrashCourse
drwxr-xr-x  17 ebanos  staff  5440 Dec 14 18:56 test_site
drwxr-xr-x  4 ebanos  staff  1280 Nov  6 12:18 typescript
ebanos@MacBook-Pro-de-Eduardo:~/Developer$
```



Eliminar archivos y carpetas

El comando **rm** en la consola se utiliza para eliminar archivos o directorios. La función principal del comando **rm** es borrar archivos o directorios de forma permanente del sistema de archivos.

Eliminar archivo

```
rm archivo.txt
```

Eliminar carpeta

```
rm -r directorio
```

-r se utiliza para eliminar directorios de forma recursiva, lo que significa que elimina el directorio especificado y todo su contenido.



ECHO

El comando echo se utiliza en sistemas para mostrar mensajes o imprimir texto en la pantalla. Además de mostrar texto simple, puede ser utilizado para redirigir texto a archivos.

```
ebanos@MacBook-Pro-de-Eduardo:~/Developer/consola
ebanos@MacBook-Pro-de-Eduardo:~/Developer/consola$ echo "Hola Mundo"
Hola Mundo
ebanos@MacBook-Pro-de-Eduardo:~/Developer/consola$
```

25



ECHO y CAT




Además de simplemente mostrar texto, el comando echo puede redirigir la salida hacia un archivo. Por ejemplo, para guardar "Hola, mundo!" en un archivo llamado saludo.txt:

```
ebanos@MacBook-Pro-de-Eduardo:~/Developer/consola
ebanos@MacBook-Pro-de-Eduardo:~/Developer/consola$ cat foo.txt
Hola Mundo
ebanos@MacBook-Pro-de-Eduardo:~/Developer/consola$ echo "Soy Lalo" > foo.txt
ebanos@MacBook-Pro-de-Eduardo:~/Developer/consola$ cat foo.txt
Soy Lalo
```

El comando cat, se utiliza principalmente para mostrar el contenido de archivos de texto en la terminal. Además, puede concatenar varios archivos y mostrar su contenido combinado en la salida estándar.

```
ebanos@MacBook-Pro-de-Eduardo:~/Developer/consola
ebanos@MacBook-Pro-de-Eduardo:~/Developer/consola$ ll
total 8
-rw-r--r--@ 1 ebanos  staff   98 Dec 30 01:40 foo.txt
ebanos@MacBook-Pro-de-Eduardo:~/Developer/consola$ cat foo.txt
Soy Lalo
```

26



Bootcamp de Desarrollo Web

Clase 4

Git

Bootcamp Desarrollo Web 27

27



Controlador de versiones

Controladores de Versiones Centralizados: Estos sistemas tienen un repositorio central donde se almacena todo el historial de versiones y los archivos. Los usuarios obtienen la versión más reciente del repositorio central y luego pueden trabajar en sus copias locales.

Controladores de Versiones Distribuidos: Estos sistemas almacenan copias locales completas de un repositorio, lo que permite a los usuarios trabajar de manera independiente y realizar cambios sin conexión a un servidor central.

Bootcamp Desarrollo Web 28

28



Tipos de Controlador de versiones

Un controlador de versiones es un sistema que registra los cambios realizados en un conjunto de archivos a lo largo del tiempo.

Estos sistemas permiten rastrear las modificaciones, facilitando la colaboración en proyectos, el seguimiento de versiones y la reversión a estados anteriores si es necesario.

Hay dos tipos principales de controladores de versiones: sistemas **centralizados** y **sistemas distribuidos**.



Beneficios de los controladores de versiones

Control de Historial: Mantienen un registro detallado de cada cambio realizado en los archivos a lo largo del tiempo.

Colaboración: Facilitan la colaboración entre múltiples personas que trabajan en un proyecto al permitirles trabajar simultáneamente en diferentes partes del código o archivos.

Rastreo de Cambios: Permiten comparar versiones anteriores, fusionar cambios entre diferentes ramas y revertir a versiones anteriores si es necesario.

Seguimiento de Autoría: Registran quién realizó cada cambio, cuándo se hizo y qué se modificó, proporcionando transparencia y responsabilidad en los desarrollos.



¿Qué es Git?

Git es un sistema de control de versiones distribuido ampliamente utilizado para el seguimiento de cambios en archivos de código fuente durante el desarrollo de software.

Git es conocido por su velocidad, eficiencia y capacidad para manejar proyectos de cualquier tamaño.



Características clave de Git

- **Ramas (Branches):** Permite trabajar en diferentes líneas de desarrollo de forma simultánea, lo que facilita la experimentación y el desarrollo paralelo sin afectar la rama principal (master o main)
- **Historial de Versiones:** Almacena el historial completo de cambios con metadatos detallados, lo que facilita la revisión de versiones anteriores y la gestión de cambios.
- **Gestión de Conflictos:** Ayuda a resolver conflictos que puedan surgir al fusionar cambios de diferentes ramas o cuando varios colaboradores modifican el mismo archivo.



Términos clave en Git

- **Repositorio (Repository):** Es el lugar donde se almacenan los archivos y el historial de versiones. Puede ser local (en la computadora del usuario) o remoto (en un servidor).
- **Commit:** Un commit representa un cambio específico en el código. Cada commit tiene un mensaje descriptivo que explica qué cambios se han realizado.
- **Pull y Push:** Pull se utiliza para obtener cambios del repositorio remoto y actualizar el repositorio local. Push se usa para enviar cambios locales al repositorio remoto.



¿Qué es Github?

GitHub es una plataforma de desarrollo de software que facilita la colaboración entre desarrolladores, permitiendo el alojamiento de proyectos, el control de versiones y herramientas de colaboración.

Características Principales

Control de Versiones: Permite realizar un seguimiento de los cambios en el código a lo largo del tiempo, facilitando la colaboración y la gestión de cambios.

Repositorios: Espacios donde se almacena el código de un proyecto, incluyendo archivos, historial y ramas.

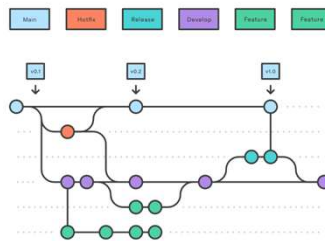
Colaboración: Facilita la colaboración entre equipos de desarrollo, permitiendo a múltiples personas trabajar juntas en un proyecto.





Git Flow

- Gitflow es un modelo de flujo de trabajo basado en Git que se utiliza para gestionar y organizar el desarrollo de software utilizando ramas específicas y una estructura definida, se ha convertido en un enfoque popular para colaborar en proyectos de software con múltiples versiones y equipos de desarrollo.



Git Flow

- Rama Master (master):** Esta rama representa la rama principal del proyecto y contiene el código estable y listo para producción. Normalmente, los despliegues a producción se realizan desde esta rama.
- Rama de Desarrollo (develop):** Esta rama es donde se integran todas las características desarrolladas. Es una rama de trabajo compartida donde se combinan los cambios de las distintas ramas de características (feature branches).
- Ramas de Características (feature branches):** Para desarrollar nuevas características o realizar trabajos específicos, se crean ramas separadas llamadas "feature branches". Estas ramas se desprenden de develop y se fusionan nuevamente en develop una vez que la característica está completa.



Git Flow

- **Ramas de Publicación (release branches):** Antes de llevar a cabo un lanzamiento o una versión, se crea una rama de publicación desde develop. En esta rama se realizan las últimas pruebas, correcciones de errores menores y preparativos para la versión. Una vez finalizada, se fusiona con master y develop.
- **Ramas de Corrección de Errores (hotfix branches):** Estas ramas se utilizan para corregir problemas críticos o errores que se descubren en producción. Se crean a partir de master, se corrige el error y luego se fusionan tanto con master como con develop.



Comandos mas usados de Git

git init : es un comando fundamental en Git que se utiliza para iniciar un repositorio nuevo y vacío. Este comando se emplea típicamente al comenzar un nuevo proyecto o al convertir un directorio existente en un repositorio Git.

Cuando ejecutas git init en un directorio, Git crea un nuevo repositorio en esa ubicación y comienza a realizar un seguimiento de los cambios en los archivos que se encuentran en ese directorio.

```
ebanos@facbook-brande-Eduardo:~/Developer/consola$ git init
ayuda: Usando 'master' como el nombre de la rama inicial. Este nombre de rama predeterminado
ayuda: está sujeto a cambios. Para configurar el nombre de la rama inicial para usar en todos
ayuda: de sus nuevos repositorios, reprimiendo esta advertencia, llama a:
ayuda:
ayuda: git config --global init.defaultBranch <nombre>
ayuda:
ayuda: Los nombres comúnmente elegidos en lugar de 'master' son 'main', 'trunk' y
ayuda: 'development'. Se puede cambiar el nombre de la rama recién creada mediante este comando:
ayuda:
ayuda: git branch -m <nombre>
ayuda:
Iniciado repositorio Git vacío en /Users/ebanos/Developer/consola/.git/
```



Comandos mas usados de Git

git status : Al ejecutarlo en la terminal, obtendrás una salida que te mostrará información importante, como:

- Los archivos modificados en tu directorio de trabajo que no se han agregado al área de preparación
- Los archivos que han sido agregados al área de preparación y están listos para ser incluidos en el próximo commit
- Los archivos que aún no están bajo seguimiento de Git
- Información sobre la rama actual en la que te encuentras y si tienes cambios pendientes para hacer commit o si estás al día con la rama remota.



Comandos mas usados de Git

git status

```
Inicializado repositorio Git vacio en /Users/ebanos/Developer/consola/.git/
ebanos@MacBook-Pro-de-Eduardo ~/Developer/consola master$ git status
En la rama master

No hay commits todavía

no hay nada para confirmar (crea/copia archivos y usa "git add" para hacerles seguimiento)
ebanos@MacBook-Pro-de-Eduardo ~/Developer/consola master$

ebanos@MacBook-Pro-de-Eduardo ~/Developer/consola master$ touch foo.txt
ebanos@MacBook-Pro-de-Eduardo ~/Developer/consola master$ git status
En la rama master

No hay commits todavía

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



Comandos mas usados de Git

git add : El comando git add en Git se utiliza para agregar cambios específicos o archivos al área de preparación (staging area), lo que prepara los cambios para ser incluidos en el próximo commit en el repositorio.

```
ebanos@MacBook-Pro-de-Eduardo ~ % git status
En la rama master

No hay commits todavía

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

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles seguimiento)
ebanos@MacBook-Pro-de-Eduardo ~ % git add foo.txt
ebanos@MacBook-Pro-de-Eduardo ~ % git status
En la rama master

No hay commits todavía

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

ebanos@MacBook-Pro-de-Eduardo ~ %
```



Comandos mas usados de Git

git restore : El comando git reset se utiliza en Git para descartar cambios en los archivos o para restaurar archivos a un estado específico. Es útil para deshacer modificaciones no deseadas en el directorio de trabajo o para revertir archivos a un estado previo.

```
ebanos@MacBook-Pro-de-Eduardo ~ % git status
En la rama master

No hay commits todavía

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

ebanos@MacBook-Pro-de-Eduardo ~ % git reset -- foo.txt
ebanos@MacBook-Pro-de-Eduardo ~ % git status
En la rama master

No hay commits todavía

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

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles seguimiento)
```



Comandos mas usados de Git

git commit : El comando git commit es fundamental en Git, ya que nos permite guardar los cambios en el repositorio de manera permanente. Es esencial entender su uso para mantener un registro claro de las modificaciones realizadas en un proyecto.

Syntaxis : **git commit -m "mensaje"**

Con esta orden, podemos registrar los cambios que han sido previamente preparados (staged) con git add. El mensaje proporcionado describe de manera breve y clara los cambios realizados en este commit.

El mensaje de commit es crucial. Debe ser descriptivo pero conciso, explicando qué cambios se han realizado. Un buen mensaje de commit facilita la comprensión de los cambios realizados por otros colaboradores en el proyecto.



Comandos mas usados de Git

```
ebanos@MacBook-Pro-de-Eduardo ~/Developer/consola master > git status
En la rama master

No hay commits todavía

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

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles seguimiento)
ebanos@MacBook-Pro-de-Eduardo ~/Developer/consola master > git add .
ebanos@MacBook-Pro-de-Eduardo ~/Developer/consola master > git status
En la rama master

No hay commits todavía

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

ebanos@MacBook-Pro-de-Eduardo ~/Developer/consola master > git commit -m "Actualizados los contenidos de ejemplo.txt"
[master (commit-raíz) d019a77] Actualizados los contenidos de ejemplo.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 foo.txt
ebanos@MacBook-Pro-de-Eduardo ~/Developer/consola master >
```



Comandos mas usados de Git

git push : Sutiliza para enviar los cambios locales que has realizado en tu rama hacia un repositorio remoto. Sin embargo, el comando git push solo puede ejecutarse correctamente si tienes configurado un repositorio remoto y has vinculado tu rama local a una rama remota.

Si ejecutas simplemente git push sin argumentos adicionales, Git intentará enviar los cambios de la rama local actual hacia la rama correspondiente en el repositorio remoto.

Aquí hay una secuencia de pasos típicos para utilizar git push:

- Asegúrate de haber realizado tus cambios y haber ejecutado git commit para confirmar tus cambios localmente en tu rama.
- Luego, ejecuta **git push**. Si es la primera vez que ejecutas este comando, puede que necesites especificar la rama remota y local con el comando:

```
git push -u <nombre-remoto> <rama-local>
```



Comandos mas usados de Git

<nombre-remoto> es el nombre del repositorio remoto, comúnmente llamado "origin".

<rama-local> es el nombre de tu rama local que deseas subir.

Comando : **git push -u <nombre-remoto> <rama-local>**

```
ebanos@MacBook-Pro-de-Eduardo ~/Developer/consola $ git push -u bootcamp main
Enumerando objetos: 3, listo.
Contando objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (3/3), 226 bytes | 226.00 KiB/s, listo.
Total 3 (delta 0), reusados 0 (delta 0), pack-reusados 0
To https://github.com/eederz/bootcamp.git
* [new branch] main -> main
rama 'main' configurada para rastrear 'bootcamp/main'.
ebanos@MacBook-Pro-de-Eduardo ~/Developer/consola $
```



Comandos mas usados de Git

git pull: se utiliza en Git para actualizar tu repositorio local con los últimos cambios desde un repositorio remoto.

Esencialmente, combina dos acciones en una: obtiene los cambios del repositorio remoto y luego ejecuta automáticamente fusión esos cambios en tu rama actual.

Cuando ejecutas git pull, Git intentará fusionar automáticamente los cambios remotos en tu rama local actual.



Comandos mas usados de Git

git pull: se utiliza en Git para actualizar tu repositorio local con los últimos cambios desde un repositorio remoto.

Esencialmente, combina dos acciones en una: obtiene los cambios del repositorio remoto y luego ejecuta automáticamente fusión esos cambios en tu rama actual.

Cuando ejecutas git pull, Git intentará fusionar automáticamente los cambios remotos en tu rama local actual.

```
lebanos@MacBook-Pro-de-Eduardo ~/Developer/console main git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Desempaquetando objetos: 100% (3/3), 651 bytes | 100.00 KiB/s, listo.
Desde https://github.com/ederz/bootcamp
d019a77..eafbb19 main -> bootcamp/main
Actualizando d019a77..eafbb19
Fast-forward
 readme.md | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 readme.md
```




git clone: se utiliza para crear una copia local (clon) de un repositorio remoto. Esta copia contiene todos los archivos, ramas y el historial de versiones del repositorio remoto, permitiéndote trabajar en tu máquina local con una réplica del proyecto.

```
ebanos@MacBook-Pro-de-Eduardo ~ % Developer/tempClone % git clone https://github.com/eederz/bootcamp.git
Clonando en 'bootcamp'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 3 (delta 0), pack-reused 0
Recibiendo objetos: 100% (6/6), listo.
ebanos@MacBook-Pro-de-Eduardo ~ % Developer/tempClone % ll
total 0
dwxr-xr-x@ 5 ebanos  staff  160B Dec 20 15:38 bootcamp
ebanos@MacBook-Pro-de-Eduardo ~ % Developer/tempClone % cd bootcamp
ebanos@MacBook-Pro-de-Eduardo ~ % Developer/tempClone/bootcamp % main % ll
total 8
-rw-r--r--@ 1 ebanos  staff   0B Dec 20 15:38 foo.txt
-rw-r--r--@ 1 ebanos  staff  2B Dec 20 15:38 readme.md
ebanos@MacBook-Pro-de-Eduardo ~ % Developer/tempClone/bootcamp % main %
```



Bootcamp Desarrollo Web