¿Qué es Git?

Git es un popular sistema de control de versiones. Fue creado por Linus Torvalds en 2005, y ha sido mantenido por Junio Hamano desde entonces.

Se utiliza para:

* Seguimiento de cambios en el código
* Seguimiento de quien realizó cambios
* Colaboración de codificación

¿Qué hace Git?

* Administrar proyectos con repositorios
* Clonar un proyecto para trabajar en una copia local
* Controlar y realizar seguimientos de cambios en modo provisional y confirmado
* Bifurcar y combinar para permitir al trabajo diferentes partes y versiones de un proyecto
* Extraer la versión más reciente del proyecto en una copia local
* Insertar actualizaciones locales en el proyecto principal

¿Por qué Git?

* Más del 70% de los desarrolladores usan Git
* Los desarrolladores pueden trabajar juntos desde cualquier parte del mundo
* Los desarrolladores pueden ver el historial completo del proyecto
* Los desarrolladores pueden versiones anteriores de un proyecto

¿Qué es GitHub?

* Git es lo mismo que GitHub
* GitHub crea herramientas que utilizan Git
* GitHub es el mayor hospedaje de código fuente en el mundo y ha sido propiedad de Microsoft desde 2018
* Este curso se centra en utilizar Git

**Iniciando con Git**

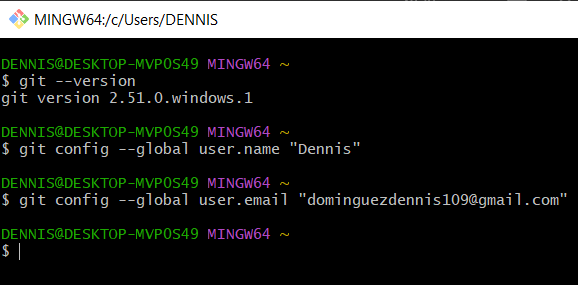
git --version



**Configurar Git**

git config –global user.name “Dennis”

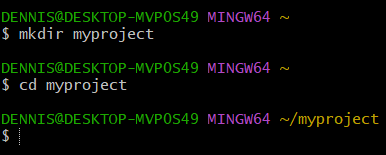
git config –global user.email “dominguezdennis109@gmail.com”



**Creación de una carpeta Git**

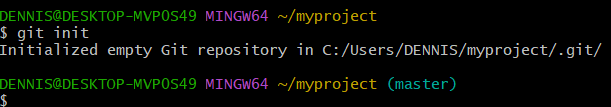
mkdir myproject

cd myproject



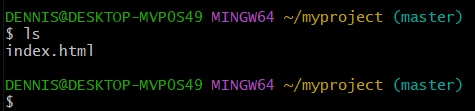
**Inicializar Git**

git init



**Agregando nuevos archivos en Git**

ls



**Entorno de preparación de Git**

* Una de las misiones centrales de Git son los conceptos del entorno de preparación y la configuración (Commit).
* Mientras se trabaja es posible que se agreguen, editen o eliminen archivos. Pero cada vez que se alcanza un hito (milestone) o se completa una parte del trabajo, es importante agregar los archivos al entorno de preparación
* Los archivos preparados (staged files) son aquellos que están listos para ser confirmados (Committed) en el repositorio en el que se está trabajando. Más adelante se considera la información acerca de las confirmaciones (Commit).

git add index.html



**Entorno de preparación de Git**

git status

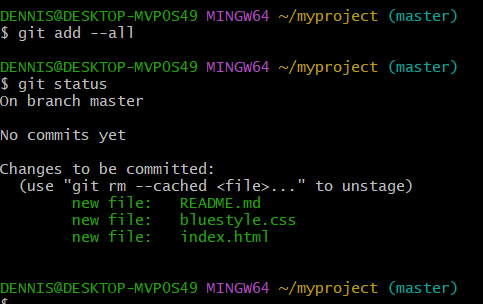


**Agregar más de un archivo en Git**

git add –all

git status

Nota El comando abreviado para git add –all es git add -A

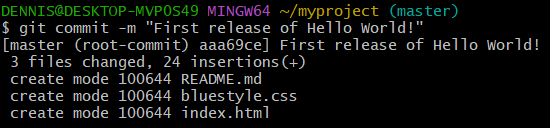


|  |
| --- |
| Ejercicios   * Agregar index.html al entorno de preparación * git add index.html * Preparar todos los archivos nuevos, modificados y eliminados.   Utilizar el comando abreviado   * git add -A |

**Confirmación en Git (Commit)**

* Dado que se ha terminado el trabajo, es posible pasar de la etapa de preparación (stage) a la confirmación (commit) en el repositorio.
* Agregar confirmaciones (commit) ayuda a hacer un seguimiento del progreso y cambios a medida que se trabaja. Git considera cada confirmación como un punto de cambio o “punto de guardado”. Es un punto del proyecto al que se puede volver si se encuentra un error o se desea realizar un cambio
* Cuando se confirma (commit) siempre se **debe incluir un mensaje**

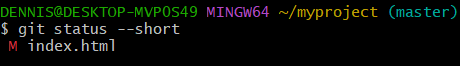
git commit -m “First reléase of Hello World!”



**Confirmación en Git sin preparación**

* En ocasiones, cuando se realiza cambios pequeños, utilizar el entorno de preparación puede parecer una pérdida de tiempo
* Es posible confirmar cambios directamente, omitiendo el entorno de preparación
* La opción -a preparará automáticamente cada archivo modificado que ya esté siendo rastreado

git status –short



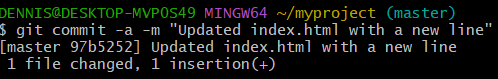
Nota Las banderas de estado abreviadas (short) son:

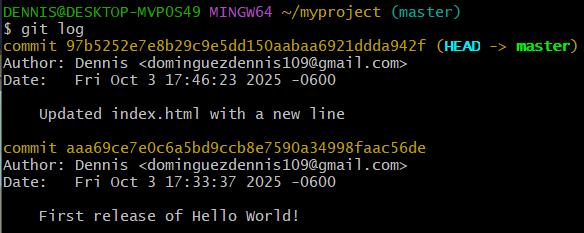
* ?? - Archivos no rastreados (Untracked files)
* A - Archivos añadidos al entorno de preparación (Files added to stage)
* M - Archivos modificados (Modified files)
* D - Archivos eliminados (Deleted files)
* Se puede observar que el archivo que se esperaba está modificado.

Así que se va a confirmar directamente:

git commit -a -m “Updated index.html with a new line”

git log





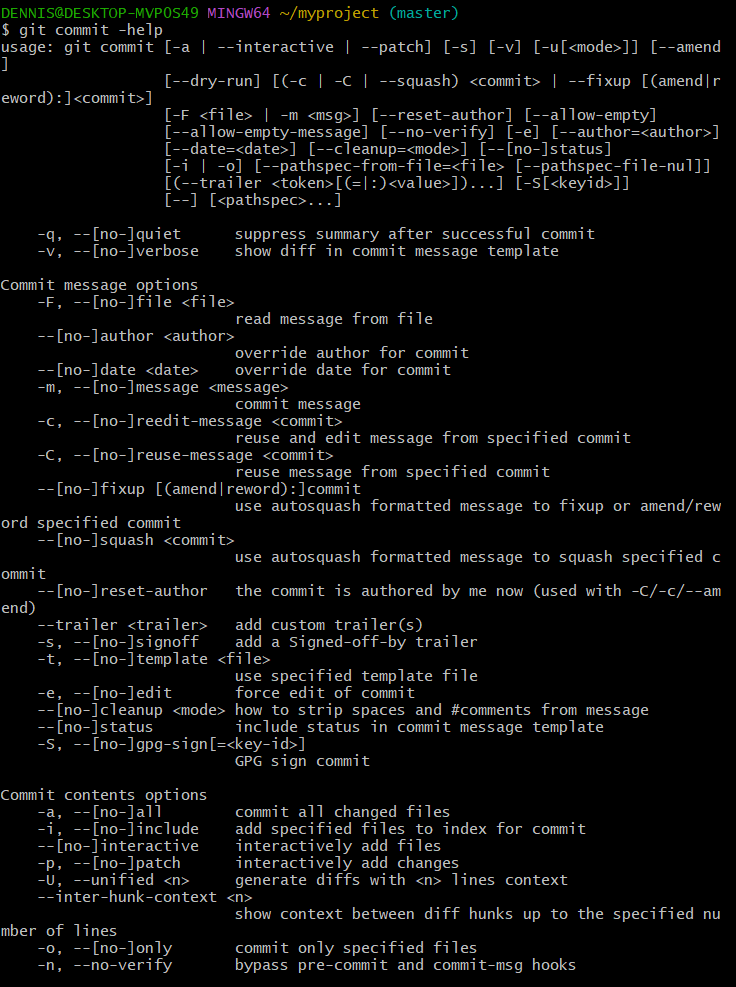
|  |
| --- |
| Ejercicios   * Confirma los cambios en el repositorio actual con el mensaje “First release! * git commit -m “First reléase!” * Revisa la versión compacta del estado del repositorio * git status –short * Confirma los archivos actualizados directamente, omitiendo el entorno de preparación * git commit -a -m “Nueva línea añadida” * Ver el historial de confirmaciones para el repositorio * git log |

**Ayuda de GIT (Git-help)**

* Si se tiene dificultades para recordar comandos u opciones para comandos, se puede utilizar la ayuda de Git (help)
* Hay varias formas de utilizar el comando de ayuda (help) en la línea de comandos
* git command -help: Muestra todas las opciones disponibles para el comando específico
* git help –all: Muestra todos los comandos posibles

**Git-help**

git commit -help

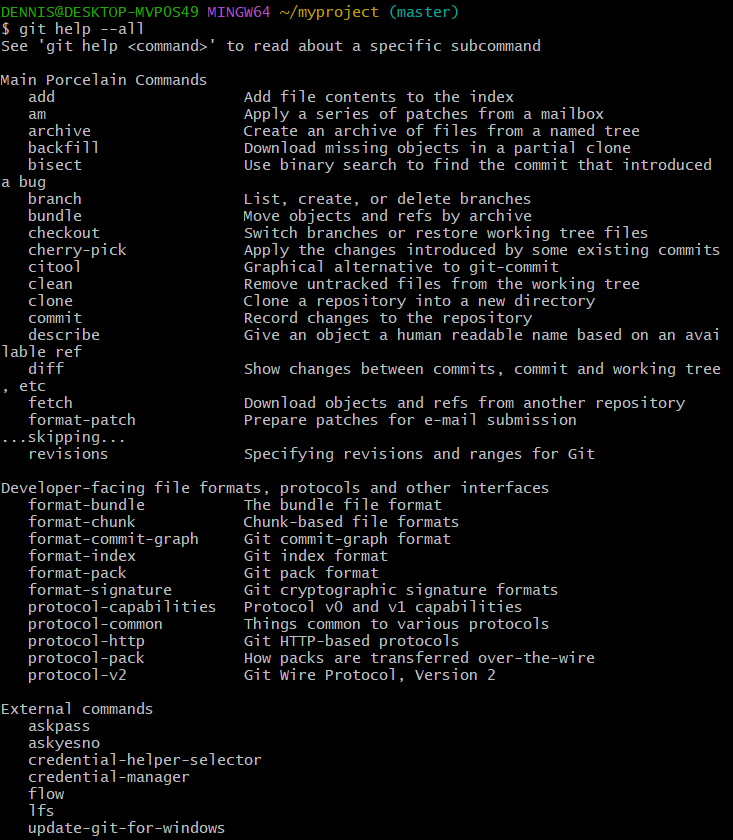


**Git help –all**

**Ver todos los comandos posibles**

* Para listar todos los comandos posibles cómo utilizar el comando help –all
* Advertencia: Esto mostrará una lista muy larga de comandos
* Nota Si se encuentra uno atascado en la vista de la lista utilizar SHIFT + g para ir al final de la lista, luego q para salir de la vista

git help –all



|  |
| --- |
| Ejercicios   * Muestra las posibles opciones para el comando de estado (status) en línea de comandos * git status -help * Muestra todos los comandos posibles de Git en la línea de comandos * git help --all |

**Git Branch (rama)**

Trabajando con ramas en Git (Branches)

En Git, una rama (Branch) es una versión nueva/separada del repositorio principal (main repository)

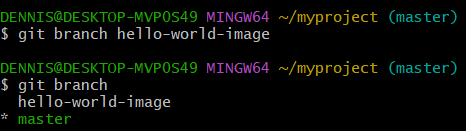
**Git Branch (Rama de GIT)**

* Ejemplo. Si se contara con un proyecto extenso y se necesitará actualizar el diseño…
* ¿Cómo funcionaría eso sin Git y con Git?
* Sin Git:
* Hacer copias de todos los archivos relevantes para evitar afectar la versión en vivo.
* Comenzar a trabajar en el diseño y descubrir que el código depende de código en otros archivos que también deben cambiarse.
* Hacer copias de los archivos dependientes también, asegurándose de que cada dependencia de archivo haga referencia al nombre de archivo correcto.
* ¡EMERGENCIA! Hay un error no relacionado en alguna otra parte del proyecto que debe corregirse lo antes posible.
* Guardar todos los archivos, tomando nota de los nombres de las copias en las que estabas trabajando.
* Trabajar en el error no relacionado y actualizar el código para corregirlo.
* Regresar al diseño y terminar el trabajo allí.
* Copiar el código o renombrar los archivos para que el diseño actualizado esté en la versión en vivo.
* (2 semanas después, se da uno cuenta de que el error no relacionado no se corrigió en la nueva versión del diseño porque se copiaron los archivos antes de la corrección).
* Ejemplo. Si se contara con un proyecto extenso y se necesitará actualizar el diseño…
* ¿Cómo funcionaría eso sin Git y con Git?
* Con Git:
* Con un Branch (rama) llamado “nuevo-diseño”, editar el código directamente sin afectar la rama principal.
* ¡EMERGENCIA! Hay un error no relacionado en alguna otra parte del proyecto que necesita ser corregido lo antes posible.
* Crear un nuevo Branch (rama) desde el proyecto principal llamada “corrección-error-pequeño”.
* Corregir el error no relacionado y fusionó la rama “corrección-error-pequeño” con la rama principal.
* Regresar a la rama “nuevo-diseño” y terminar el trabajo allí.
* Fusionar la rama de “nuevo-diseño” con la principal (recibiendo una alerta sobre la corrección del error pequeño que te faltaba).
* Las ramas (Branch) le permiten a uno trabajar en diferentes partes de un proyecto sin afectar la rama principal.
* Una vez que el trabajo está completo, una rama (Branch) puede fusionarse con el proyecto principal.
* Incluso se puede cambiar entre ramas (Branches) y trabajar en proyectos diferentes sin que interfieran entre sí.
* La creación de ramas (Braching) en Git es muy ligera y rápida.

**Nuevo Git Branch (nueva rama)**

git branch hello-world-image

git branch

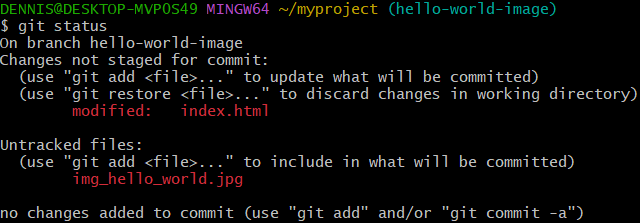


* Se puede observar la nueva rama con el nombre “hello-world-image”, pero el \* junto a “master” especifica que actualmente se está en esta rama.
* “checkout”. Comando utilizado para cambiar a una rama. Este mueve desde la rama actual a la que se especifica el final del comando:

git checkout hello-world-images

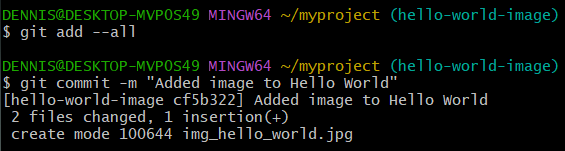


git status



git add –all

git commit -m “Added image to Hello World”



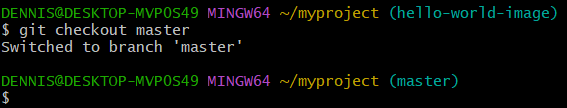
Nota utilizar la opción -b en checkout creará un nuevo Branch (rama) y se moverá a este si no existe

**Switching Between Branches (Cambiar entre ramas)**

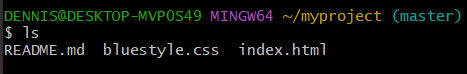
Ls



git checkout master

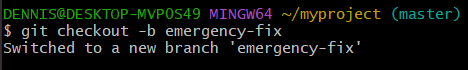


ls

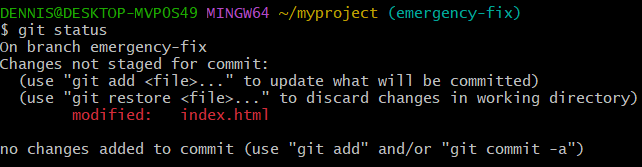


**Rama de Emergencia (Emergency Branch)**

git checkout -b emergency-fix

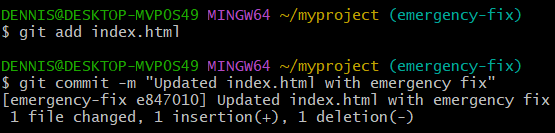


git status



git add index.html

git commit -m “Updated index.html with energebcy fix”



|  |
| --- |
| Ejercicio   * Crear un nuevo branch llamado hello-world-images: * git branch hello-world-image * Enlistar los branches existentes * git branch * Ubicarse en el Branch hello-world-image * git checkout * Crear y ubicarse en mi nuevo branch con el nombre “hello-you” * git checkout -b hello-you |