

Comandos de GIT Básicos – Guía Completa



¿Necesitas aprender algunos comandos de [GIT](#) básicos? Has venido al lugar correcto. Sigue leyendo para descubrir nuestra práctica hoja de trucos que puedes utilizar como referencia diaria.

¡Empecemos!

Entendiendo cómo funciona GIT

Comandos de GIT básicos

Hoja de trucos de los comandos de GIT en .pdf

Entendiendo cómo funciona GIT

GIT es el SCV (sistema de control de versiones) de código abierto [más utilizado](#) que te permite rastrear los cambios realizados en los archivos. Las empresas y los programadores suelen utilizar el GIT para colaborar en el desarrollo de software y aplicaciones.

El directorio de trabajo es donde se agregan, borran y editan los archivos. Luego, los cambios son preparados (indexados) en el área de preparación. Después de que confirmes tus cambios, la instantánea de los cambios se guardará en el directorio git.

Todo el mundo puede usar GIT ya que está disponible para [Linux](#), [Windows](#), [Mac](#) y [Solaris](#). El software puede tener una fuerte curva de aprendizaje, pero hay muchos [tutoriales](#) disponibles para ayudarte.

Comandos de GIT básicos

Aquí hay algunos comandos básicos de GIT que debes conocer:

- **git init** creará un nuevo repositorio local GIT. El siguiente comando de Git creará un repositorio en el directorio actual:

```
git init
```

- Como alternativa, puedes crear un repositorio dentro de un nuevo directorio especificando el nombre del proyecto:

```
git init [nombre del proyecto]
```

- **git clone** se usa para copiar un repositorio. Si el repositorio está en un servidor remoto, usa:

```
git clone nombredeusuario@host:/path/to/repository
```

- A la inversa, ejecuta el siguiente comando básico para copiar un repositorio local:

```
git clone /path/to/repository
```

- **git add** se usa para agregar archivos al área de preparación. Por ejemplo, el siguiente comando de Git básico indexará el archivo temp.txt:

```
git add <temp.txt>
```

- **git commit** creará una instantánea de los cambios y la guardará en el directorio git.

```
git commit -m "El mensaje que acompaña al commit va aquí"
```

- Ten en cuenta que los cambios confirmados no llegarán al repositorio remoto.

- **git config** puede ser usado para establecer una configuración específica de usuario, como el email, nombre de usuario y tipo de formato, etc. Por ejemplo, el siguiente comando se usa para establecer un email:

```
git config --global user.email tuemail@ejemplo.com
```

- La opción -global le dice a GIT que vas a usar ese correo electrónico para todos los repositorios locales. Si quieres utilizar diferentes correos electrónicos para diferentes repositorios, usa el siguiente comando:

```
git config --local user.email tuemail@ejemplo.com
```

```
git status
```

- **git push** se usa para enviar confirmaciones locales a la rama maestra del repositorio remoto. Aquí está la estructura básica del código:

```
git push origin <master>
```

- Reemplaza <master> con la rama en la que quieres enviar los cambios cuando no quieras enviarlos a la rama maestra.

- **git checkout** crea ramas y te ayuda a navegar entre ellas. Por ejemplo, el siguiente comando crea una nueva y automáticamente se cambia a ella:

```
command git checkout -b <branch-name>
```

- Para cambiar de una rama a otra, sólo usa:

```
git checkout <branch-name>
```

- **git remote** te permite ver todos los repositorios remotos. El siguiente comando listará todas las conexiones junto con sus URLs:

```
git remote -v
```

- Para conectar el repositorio local a un servidor remoto, usa este comando:

```
git remote add origin <host-or-remoteURL>
```

- Por otro lado, el siguiente comando borrará una conexión a un repositorio remoto especificado:

```
git remote <nombre-del-repositorio>
```

- **git branch** se usa para listar, crear o borrar ramas. Por ejemplo, si quieres listar todas las ramas presentes en el repositorio, el comando debería verse así:

```
git branch
```

- Si quieres borrar una rama, usa:

```
git branch -d <branch-name>
```

- **git pull** fusiona todos los cambios que se han hecho en el repositorio remoto con el directorio de trabajo local.

```
git pull
```

- **git merge** se usa para fusionar una rama con otra rama activa:

```
git merge <branch-name>
```

- **git diff** se usa para hacer una lista de conflictos. Para poder ver conflictos con respecto al archivo base, usa:

- El siguiente comando se usa para ver los conflictos que hay entre ramas antes de fusionarlas:

```
git diff <source-branch> <target-branch>
```

- Para ver una lista de todos los conflictos presentes usa:

```
git diff
```

- **git tag** marca commits específicos. Los desarrolladores lo usan para marcar puntos de lanzamiento como v1.0 y v2.0.

```
git tag 1.1.0 <instert-commitID-here>
```

- **git log** se usa para ver el historial del repositorio listando ciertos detalles de la confirmación. Al ejecutar el comando se obtiene una salida como ésta:

```
commit 15f4b6c44b3c8344caasdac9e4be13246e21sadm
Author: Alex Hunter <alexh@gmail.com>
Date:   Mon Oct 1 12:56:29 2016 -0600
```

- **git reset** sirve para resetear el index y el directorio de trabajo al último estado de confirmación.

```
git reset - -hard HEAD
```

- **git rm** se puede usar para remover archivos del index y del directorio de trabajo.

```
git rm filename.txt
```

- **git stash** guardará momentáneamente los cambios que no están listos para ser confirmados. De esta manera, puedes volver al proyecto más tarde.

```
git stash
```

- **git show** se usa para mostrar información sobre cualquier objeto git.

```
git show
```

- **git fetch** le permite al usuario buscar todos los objetos de un repositorio remoto que actualmente no se encuentran en el directorio de trabajo local.

```
git fetch origin
```

- **git ls-tree** te permite ver un objeto de árbol junto con el nombre y modo de cada ítem, y el valor blob de SHA-1. Si quieres ver el HEAD, usa:

```
git ls-tree HEAD
```

- **git cat-file** se usa para ver la información de tipo y tamaño de un objeto del repositorio. Usa la opción -p junto con el valor SHA-1 del objeto para ver la información de un objeto específico, por ejemplo:

```
git cat-file -p d670460b4b4aece5915caf5c68d12f560a9fe3e4
```

- **git grep** le permite al usuario buscar frases y palabras específicas en los árboles de confirmación, el directorio de trabajo y en el área de preparación. Para buscar por www.hostinger.com en todos los archivos, usa:

- **gitk** muestra la interfaz gráfica para un repositorio local. Simplemente ejecuta:

```
gitk
```

- **git instaweb** te permite explorar tu repositorio local en la interfaz GitWeb. Por ejemplo:

```
git instaweb -http=webrick
```

- **git gc** limpiará archivos innecesarios y optimizará el repositorio local.

```
git gc
```

- **git archive** le permite al usuario crear archivos zip o tar que contengan los constituyentes de un solo árbol de repositorio. Por ejemplo:

```
git archive - -format=tar master
```

- **git prune** elimina los objetos que no tengan ningún apuntador entrante.

```
git prune
```

- **git fsck** realiza una comprobación de integridad del sistema de archivos git e identifica cualquier objeto corrupto

```
git fsck
```

- **git rebase** se usa para aplicar ciertos cambios de una rama en otra. Por ejemplo:

```
git rebase master
```

Conclusión

Aprender los comandos básicos de GIT será de gran ayuda para los desarrolladores, ya que pueden controlar fácilmente el código fuente de los proyectos. Puede que te lleve algo de tiempo recordarlos todos, por eso nuestra hoja de trucos de GIT podría resultarte útil.

¡Practica estos comandos de GIT y aprovecha al máximo tus habilidades en desarrollo! ¡Buena suerte!