TRUCOS Y CONSEJOS DE GIT

presentado por Tower - el mejor cliente Git para Mac y Windows

CREAR

Clonar un repositorio existente

```
$ git clone ssh://user@domain.com/repo.git
```

Crear un nuevo repositorio local

```
$ git init
```

CAMBIOS LOCALES

Ficheros modificados en el directorio de trabajo

```
$ git status
```

Cambios en los ficheros bajo seguimiento

```
$ git diff
```

Añadir todos los cambios actuales en el siguiente commit

```
$ git add .
```

Agrega algunos cambios en <fichero> al siguiente commit

```
$ git add -p <fichero>
```

Confirmar todos los cambios locales en los ficheros bajo seguimiento

```
$ git commit -a
```

Hacer commit de los cambios previamente realizados

```
$ git commit
```

Modificar el último commit

¡No modifiques los commits ya publicados!

```
$ git commit --amend
```

HISTORIAL DE COMMITS

Mostrar todos los commits, comenzando por el más nuevo

```
$ git log
```

Mostrar cambios previos para un fichero específico

```
$ git log -p <file>
```

¿Quién cambió qué y cuándo en <fichero>?

```
$ git blame <fichero>
```

RAMAS & ETIQUETAS

Mostrar un listado de todas las ramas existentes

```
$ git branch -av
```

Cambiar a la rama HEAD

```
$ git checkout <rama>
```

Crea una nueva rama basada en tu HEAD actual

```
$ git branch <nueva-rama>
```

Crear una nueva rama de seguimiento basada en una rama remota

```
$ git checkout --track <rama/remota>
```

Eliminar una rama local

```
$ git branch -d <rama>
```

Marcar el commit actual con una etiqueta

```
$ git tag <etiqueta>
```

ACTUALIZACIÓN & PUBLICACIÓN

Listar todos los repositorios remotos configurados actualmente

```
$ git remote -v
```

Mostrar información sobre un repositorio remoto

```
$ git remote show <remoto>
```

Agregar nuevo repositorio remoto, llamado <nombre-corto>

```
$ git remote add <nombre-corto> <url>
```

Descargar todos los cambios desde el repositorio <remoto> (sin integrar en HEAD)

```
$ git fetch <remoto>
```

Descargar cambios y directamente integrar en HEAD

```
$ git pull <remoto> <rama>
```

Publicar los cambios locales en el repositorio remoto

```
$ git push <remoto> <rama>
```

Eliminar una rama en el repositorio remoto

```
$ git branch -dr <remoto/rama>
```

Publicar tus etiquetas

```
$ git push --tags
```

MERGE & REBASE

Integrar <rama> en el HEAD actual

```
$ git merge <rama>
```

Hacer rebase del HEAD actual a <rama>

No hagas rebase de commits publicados!

```
$ git rebase <rama>
```

Abortar un rebase

```
$ git rebase --abort
```

Continuar con un rebase después de resolver conflictos

```
$ git rebase --continue
```

Usa el editor para resolver conflictos manualmente y (después de resolver) marcar el archivo como resuelto

```
$ git mergetool
```

Usa el editor para resolver conflictos manualmente y (después de resolver) marcar el archivo como resuelto

```
$ git add <fichero-resuelto>
```

```
$ git rm <fichero-resuelto>
```

DESHACER

Descartar todos los cambios locales en el directorio de trabajo

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

Descartar cambios locales en un fichero específico

```
$ git checkout HEAD <fichero>
```

Revertir un commit (produciendo un nuevo commit con los cambios inversos)

```
$ git revert <commit>
```

Resetea el HEAD a un commit anterior ... y descarta todos los cambios desde entonces

```
$ git reset --hard <commit>
```

...y preservar todos los cambios como cambios pendientes

```
$ git reset <commit>
```

... y preservar cambios locales no commiteados

```
$ git reset --keep <commit>
```

CONTROL DE VERSIÓN

BUENAS PRÁCTICAS

HAZ COMMIT SÓLO DE LOS CAMBIOS RELACIONADOS

Un commit debe ser un envoltorio para los cambios relacionados. Por ejemplo, la corrección de dos errores diferentes debería producir dos commits separados. Hacer pequeños commits facilita a los demás desarrolladores la comprensión de los cambios y el retroceso de los mismos en caso de que algo saliera mal.

Con herramientas como el área de staging y la capacidad de incluir sólo partes de un archivo, Git facilita la creación de commits muy granulares.

HAZ COMMITS FRECUENTEMENTE

Hacer commits muy a menudo ayuda mantener tus commits pequeños y, de nuevo, te ayuda a commitear sólo los cambios relacionados. Además, te permite compartir tu código con otros con más frecuencia. De esta manera es más fácil para todos integrar los cambios regularmente y evitar conflictos de integración. Tener pocos commits grandes y compartirlos de forma no habitual, en cambio, dificulta la resolución de conflictos.

NO HAGAS COMMIT DE TRABAJO HECHO A MEDIAS

Sólo debes integrar el código cuando esté completo. Esto no significa que tengas que completar una función completa y grande antes de hacer un commit. Todo lo contrario: divide la implementación de la función en partes lógicas y recuerda hacer commit pronto y con frecuencia. Pero nunca debes hacer commit sólo para tener algo en el repositorio antes de salir de la oficina al final del día. Si estás tentado a hacer un commit sólo porque necesitas una área de trabajo limpia (para revisar una rama, hacer cambios, etc.) considera usar la función «Stash» de Git en su lugar.

PRUEBA TU CÓDIGO ANTES DE HACER COMMIT

Resiste la tentación de hacer commit de algo que «creas» que se ha completado. Pruébelo a fondo para asegurarte de que realmente está completo y no causa efectos secundarios (hasta donde alcanza tu conocimiento). Mientras integrar cosas hechas a medias en tu repositorio local sólo te causa problemas a ti, tener tu código probado es aún más importante cuando se trata de compartir tu código con otros.

ESCRIBE TÍTULOS DESCRIPTIVOS

Comenza tu mensaje con un breve resumen de tus cambios (hasta 50 carácteres como guía). Sepáralo del siguiente cuerpo incluyendo una línea en blanco. El cuerpo de tu mensaje debe proporcionar respuestas detalladas a las siguientes preguntas:

› ¿Cuál es fin del cambio?

› ¿En qué se diferencia de la implementación anterior?

Usa el imperativo, presente («cambiar», en lugar de «he cambiado» o «cambios en...») para ser consistente con los mensajes generados desde comandos como git merge.

EL CONTROL DE VERSIONES NO ES UN SISTEMA DE COPIAS DE SEGURIDAD

Hacer una copia de seguridad de tus archivos en un servidor remoto es un agradable efecto secundario de tener un sistema de control de versiones. Pero no deberías usar tu VCS como si fuera un sistema de respaldo. Al realizar el control de versiones, debes prestar atención a hacer commits semánticamente (ver «cambios relacionados») - no debes simplemente poner más ficheros.

UTILIZA RAMAS

La ramificación es una de las características más potentes de Git, y esto no es casualidad: desde el primer día, la ramificación rápida y sencilla fue un requisito fundamental. Las ramas son la herramienta perfecta para evitar que se mezclen diferentes líneas de desarrollo. Deberías utilizar las ramas de forma extensiva en tus flujos de trabajo de desarrollo: para nuevas funciones, correcciones de errores, ideas....

ACORDAD UN FLUJO DE TRABAJO

Git te permite elegir entre una gran variedad de flujos de trabajo: ramas de larga duración, ramas temáticas, merge o rebase, git-flow... La elección depende de un par de factores: tu proyecto, tus flujos de trabajo generales de desarrollo e implementación y (quizás lo más importante) tus preferencias personales y las de tus compañeros de equipo. Independientemente de cómo se decida trabajar, aseguraos de acordar un flujo de trabajo común que todos sigan.

AYUDA Y DOCUMENTACIÓN

Obtén más ayuda en la línea de comandos

```
$ git help <comando>
```

RECURSOS GRATUITOS EN LÍNEA

http://www.git-tower.com/learn

http://rogerdudler.github.io/git-guide/

http://www.git-scm.org/