

# CÁBULA DE GIT

Cortesia da Tower - o melhor cliente de Git para Mac e Windows



## CLONAR

Clona um repositório existente

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

Criar um novo repositório local

```
$ git init
```

## MODIFICAÇÕES LOCAIS

Ver ficheiros modificados no diretório de trabalho

```
$ git status
```

Ver alterações nos ficheiros monitorizados

```
$ git diff
```

Adicionar todas as alterações atuais ao próximo commit

```
$ git add .
```

Adicionar as alterações efetuadas a determinado ficheiro ao próximo commit

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

Adicionar um commit com todas as alterações efetuadas em ficheiros monitorizados

```
$ git commit -a
```

Adicionar um commit com alterações previamente preparadas

```
$ git commit
```

Alterar o último commit  
*Não modifique commits já publicados!*

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

## HISTÓRICO DE COMMITS

Mostrar todos os commits, começando pelo mais recente

```
$ git log
```

Mostrar as modificações de um determinado ficheiro

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

Ver quem mudou o quê e quando num determinado ficheiro

```
$ git blame <ficheiro>
```

## RAMOS E ETIQUETAS

Listar todos os ramos existentes

```
$ git branch -av
```

Mudar de ramo

```
$ git checkout <ramo>
```

Criar um novo ramo a partir da HEAD atual

```
$ git branch <novo-ramo>
```

Criar um novo ramo de rastreamento baseado num ramo remoto

```
$ git checkout --track <remoto/ramo>
```

Eliminar um ramo local

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

Associar uma etiqueta ao commit atual

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

## ATUALIZAR E PUBLICAR

Listar todos os remotos configurados

```
$ git remote -v
```

Mostrar informações sobre um remoto

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

Adicionar um novo repositório remoto

```
$ git remote add <nome> <url>
```

Descarregar todas as alterações de um repositório remoto, sem as integrar na HEAD

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

Descarregar todas as alterações de um repositório remoto, integrando-as na HEAD

```
$ git pull <remoto> <ramo>
```

Publicar todas as alterações locais num repositório remoto

```
$ git push <remoto> <ramo>
```

Eliminar um ramo de um repositório remoto

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

Publicar etiquetas

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

## MERGE E REBASE

Fazer merge de determinado ramo no HEAD atual

```
$ git merge <ramo>
```

Fazer rebase do seu HEAD em determinado ramo *Não faça rebase com commits publicados!*

```
$ git rebase <ramo>
```

Abortar um rebase

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

Continuar um rebase depois de resolver conflitos

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

Utilizar a ferramenta de merge configurada para resolver conflitos

```
$ git mergetool
```

Utilizar o editor para resolver conflitos manualmente e marcar o ficheiro como resolvido

```
$ git add <ficheiro-resolvido>
```

```
$ git rm <ficheiro-resolvido>
```

## DESFAZER

Descartar todas as mudanças locais no diretório de trabalho

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

Descartar todas as alterações locais de um ficheiro específico

```
$ git checkout HEAD <ficheiro>
```

Reverter um commit (criando um novo com as alterações inversas)

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

Repõe o ponteiro HEAD para um commit anterior  
... e descarta as alterações desde então

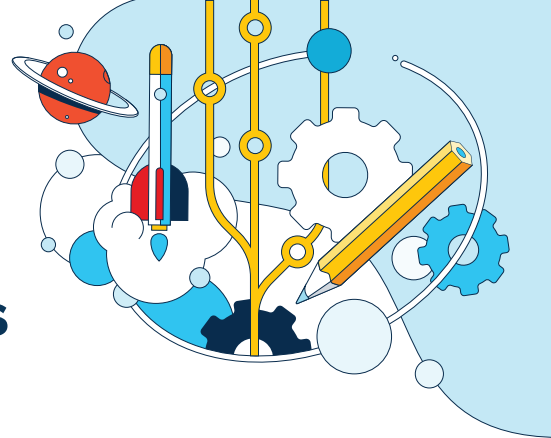
```
$ git reset --hard <commit>
```

...e preserva todas as modificações como modificações não preparadas

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

...e preserva as modificações locais não adicionadas a commits

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



# Sistema de Controle de Versões

## Boas Práticas

### COMMITTS PEQUENOS

Um commit deve conter um conjunto de modificações relacionadas. Por exemplo, uma correção de dois bugs diferentes deve produzir dois commits separados. Pequenos commits facilitam a vida aos programadores no momento de entender o que mudou ou de desfazer algo, se necessário. Com a possibilidade de adicionar apenas partes de um ficheiro, o Git facilita a criação de commits granulares.

### COMMITTS FREQUENTES

Commits frequentes ajudam a manter os seus commits pequenos e, novamente, a incluir apenas modificações relacionadas. Assim, poderá partilhar o seu código mais frequentemente com a equipa, facilitando a vida de todos na integração das mudanças por evitar conflitos no momento do merge. Ter poucos commits, partilhados raramente e com muitas alterações apresentadas, dificulta a resolução de conflitos.

### COMMITTS COMPLETOS

Só deve fazer o commit do código quando este estiver concluído. Tal não significa que tenha de completar uma funcionalidade inteira antes de fazer o commit—muito pelo contrário. Divida a implementação da funcionalidade em partes lógicas e lembre-se de fazer commits cedo e com frequência. Não faça o commit apenas para ter algo no repositório antes de sair do escritório no final do dia. Se se sentir tentado a fazer um commit só porque precisa de uma cópia de trabalho limpa (para verificar um ramo, fazer pull de alterações, etc.) considere usar a funcionalidade "Stash" do Git.

### TESTE O CÓDIGO ANTES DO COMMIT

Resista à tentação de adicionar algo que "pensa" estar concluído. Teste o seu código exaustivamente para se certificar de que está realmente finalizado e que não aparenta trazer efeitos colaterais. No seu repositório local só precisará de se perdoar a si próprio, mas quando partilha o seu código com outros, é especialmente importante que este esteja devidamente testado.

### ESCREVA BOAS MENSAGENS DE COMMIT

Comece a sua mensagem com um breve resumo das suas alterações (até 50 caracteres como linha de orientação). Separe-a do corpo utilizando uma linha em branco. O corpo da sua mensagem deve responder às seguintes perguntas:

- › O que motivou esta mudança?
  - › Em que é que ela difere da implementação anterior?
- Use o imperativo, no presente ("change", não "changed" ou "changes") para ser consistente com as mensagens geradas por comandos como `git merge`.

### CONTROLE DE VERSÃO NÃO É BACKUP

Ter cópias de segurança dos seus ficheiros num servidor remoto é um bom efeito secundário de utilizar um sistema de controle de versões. Mas não deve ser encarado como um sistema de backup. Ao fazer o controle de versões, deve ter em atenção a semântica dos commits, evitando o simples amontoar de ficheiros.

### UTILIZE RAMOS

A ramificação é uma das funcionalidades mais poderosas do Git - e isto não é por acaso: a ramificação rápida e fácil foi um requisito central desde o primeiro dia. Os ramos são a ferramenta perfeita para evitar a mistura de diferentes linhas de desenvolvimento. Deve utilizar extensivamente as ramificações nos seus fluxos de trabalho para desenvolver novas funcionalidades, corrigir erros, testar ideias...

### SIGA UM FLUXO DE TRABALHO

O Git permite-lhe escolher entre uma série de fluxos de trabalho diferentes: ramos de longa duração, ramos de tópicos, merge ou rebase, git-flow... A escolha depende de alguns factores: o seu projeto, os seus fluxos de trabalho gerais de desenvolvimento e implementação e (talvez o mais importante) as suas preferências pessoais e as dos seus colegas de equipa. Independentemente da forma como escolher trabalhar, certifique-se de que chega a acordo sobre um fluxo de trabalho comum que todos seguem.

### AJUDA E DOCUMENTAÇÃO

Obtenha ajuda por linha de comando

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

### RECURSOS ONLINE GRÁTIS

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

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

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