

# GIT CHEAT SHEET

presented by **TOWER** > Version control with Git - made easy



## CRIAR

Clona um repositório existente

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

Cria um novo repositório local

```
$ git init
```

## MODIFICAÇÕES LOCAIS

Arquivos modificados no diretório ativo

```
$ git status
```

Modificações em arquivos versionados

```
$ git diff
```

Add all current changes to the next commit

```
$ git add .
```

Adiciona as mudanças do <file>  
no próximo commit

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

Commita todas as modificações de  
arquivos versionados

```
$ git commit -a
```

Commita modificações preparadas

```
$ git commit
```

Modifica o último commit

*Não modifique commits publicados!*

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

## LINHA DO TEMPO

Mostra todos os commits, começando  
pelo mais novo

```
$ git log
```

Mostra as modificações para um arquivo  
específico

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

Quem mudou o quê e quando em um arquivo

```
$ git blame <file>
```

## BRANCHES E TAGS

Lista todas as branches existentes

```
$ git branch -av
```

Muda a branch atual

```
$ git checkout <branch>
```

Cria uma branch a partir do HEAD atual

```
$ git branch <new-branch>
```

Deleta uma branch local

```
$ git checkout --track <remote/branch>
```

Delete a local branch

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

Marca o commit atual com uma tag

```
$ git tag <tag-name>
```

## ATUALIZAR E PUBLICAR

Lista todos os remotes configurados  
atualmente

```
$ git remote -v
```

Mostra informações sobre um remote

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

Adiciona um novo repositório remoto,  
nomeado <remote>

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

Baixa todas as modificações do <remote>,  
mas não integra ao HEAD

```
$ git fetch <remote>
```

Baixa as modificação e automaticamente  
faz o merge

```
$ git pull <remote> <branch>
```

Publica as modificações locais em um remote

```
$ git push <remote> <branch>
```

Deleta uma branch no remote

```
$ git branch -dr <remote/branch>
```

Publica suas tags

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

## MERGE E REBASE

Fazer merge da <branch> no HEAD atual

```
$ git merge <branch>
```

Fazer rebase do seu HEAD na <branch>

*Não faça rebase com commits publicados!*

```
$ git rebase <branch>
```

Abortar um rebase

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

Continuar um rebase depois de resolver  
conflitos

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

Usar a sua ferramenta de merge  
configurada para resolver conflitos

```
$ git mergetool
```

Use seu editor para resolver conflitos manual-  
mente e marcar o arquivo como resolvido

```
$ git add <resolved-file>
```

```
$ git rm <resolved-file>
```

## DESFAZER

Descarta todas as mudanças locais  
no diretório atual

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

Descarta mudanças locais em  
um arquivo específico

```
$ git checkout HEAD <file>
```

Reverte um commit (criando um novo com as  
modificações ao contrário)

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

Reseta o ponteiro do HEAD para um  
commit anterior

...e descarta as modificações desde então

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

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

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

...e preserva modificações locais não commitadas

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

# VERSION CONTROL

## MELHORES PRÁTICAS



### COMMITTE MODIFICAÇÕES RELACIONADAS

Um commit deve ser uma pacote de modificações relacionadas. Por exemplo, corrigindo dois bugs diferentes deve produzir dois commits separados. Pequenos commits tornam fácil para outros desenvolvedores entenderem as modificações e as desfazerem caso algo saia errado.

Com ferramentas como área de staging e a habilidade de enviar para stage apenas partes de um arquivo, o Git torna fácil a criação de commits bem granulares.

### COMMITTE SEMPRE

Commitar com frequência ajuda manter seus commits pequenos e, novamente, ajuda você commitar apenas modificações relacionadas. Além disso, isso permite você compartilhar seu código mais frequentemente com os outros. Dessa forma é mais fácil para todos integrarem as mudanças regularmente, evitando clonfritos no merge. Tendo poucos e grandes commits, e compartilhando-os raramente, em contraste, torna a resolução de conflitos difícil.

### NÃO COMMITTE TRABALHO PELA METADE

Você só deve commitar o código quando ele estiver pronto. Isso não significa que você deve completar um grande feature inteira antes de commitar. Pelo contrário: Divida a implementação das features blocos lógicos e lembre-se de commitar cedo e com frequência. Mas não committe apenas para ter algo no repositório antes de deixar o escritório do final do dia. Se você está tentado a commitar apenas pela necessidade de uma cópia de trabalho limpa (para fazer checkout em uma branch, pull em modificações, etc.) considere usar a funcionalidade de stash do Git.

### TESTE O CÓDIGO ANTES DE COMMITAR

Resista a tentação de commitar algo que você «acha» que terminou. Teste completamente para ter certeza que está pronto e não tem efeitos colaterais (as far as one can tell). Enquanto que commitar coisas pela metade, exige apenas que você se perdoe, ter seu código testado é muito mais importante quando ele é publicado/compartilhado com seus amigos.

### ESCREVA BOAS MENSAGENS DE COMMIT

Comece sua mensagem com um pequeno sumário de sua modificações (até 50 letras como orientação). Separe isso do corpo do commit com uma linha em branco. O corpo da mensagem deve prover respostas detalhadas para as seguintes perguntas:

- › Qual foi a motivação para a modificação?
- › Quais as diferenças da última implementação?

Use o imperativo, no tempo presente («muda», não «mudou» ou «mudanças») para ser consistente com mensagens geradas a partir de comandos como git merge.

### CONTROLE DE VERSÃO NÃO É BACKUP

Ter um backup dos seus arquivos em um servidor remoto é um bom efeito colateral do uso de sistemas de controle de versão. Mas você não deveria usar um SCV como backup. Fazendo o controle de versão, você deve prestar atenção em commitar semanticamente - não amontoe apenas um monte de arquivos.

### USE BRANCHES

Branches são umas das funcionalidades mais poderosas do Git - e não é por acidente: branches de uma forma fácil e rápida foi um requisito central desde o início. Branches são ferramentas perfeitas para ajudar você a não misturar diferentes frentes de desenvolvimento. Você deve usar branches várias vezes no seu fluxo de desenvolvimento: para novas funcionalidades, correções de bugs, ideias...

### SIGA UM WORKFLOW

O Git permite você escolher entre várias formas de desenvolvimento: branches de longa duração, branches como tópicos, merge ou rebase, git-flow...

A sua escolha depende de alguns fatores: seus projeto, seu desenvolvimento global e fluxo de deploy e (talvez o mais importante) nas preferências suas e do seu time. Independente do que você escolha, apenas garanta que todos concordem em seguir o fluxo de trabalho.

### AJUDA E DOCUMENTAÇÃO

Obtenha ajuda por linha de comando

```
$ git help <command>
```

### RECURSOS ONLINE GRÁTIS

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

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

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