# COMANDOS INICIAIS DO GIT

Conheça os comandos básicos do Git nesse artigo simples.

por Candido Souza 09/01/2015 **Seja o primeiro a comentar!**

~ 5 min. / 1013 palavras

O Git é um sistema de controle de versão. Com o Git você não perderá seu trabalho, vai poder voltar para a versões anteriores, recuperando a versão do seu código de antes de ter cometido o erro e poderá criar e trabalhar diversas versões em paralelo.

Uma ótima leitura que indico é o livro Pro Git, escrito por Scott Chacon. Ele descreve corretamente sobre o controle de versão. Olhe só:

**O que é controle de versão, e por que você deve se importar?**

"O controle de versão é um sistema que registra as mudanças feitas em um arquivo ou um conjunto de arquivos ao longo do tempo, de forma que você possa recuperar versões específicas.

Se quer manter todas as versões de uma imagem ou layout, usar um Sistema de Controle de Versão (Version Control System ou VCS) é uma decisão sábia. Ele permite reverter arquivos e projetos inteiros para um estado anterior, compara mudanças feitas ao decorrer do tempo, vê quem foi o último a modificar algo que pode estar causando problemas, quem introduziu um bug e quando, e muito mais. Usar um VCS normalmente significa que caso tenha estragado ou perdido algum arquivo, poderá facilmente reavê-los. Além disso, você pode controlar tudo sem maiores esforços."

## Vamos lá!

Bom, depois dessa aula com Scott Chacon, vamos ver alguns códigos!

Lembrando que todos os comandos aqui devem ser feitos pelo Terminal, Console, GitBash, entre outros e o não recomendado Prompt de Comando do Windows, incluindo entradas e saídas de pastas, tudo por comandos!

# Iniciando o Git

Entre no diretório que deseja controlar a versão e inicie o Git assim:

```
git init
```

Feito isso, seus arquivos ainda não estão sendo versionados, mas eles estão esperando para serem adicionados no estágio de controle. Para fazer isso digite o comando

```
git add nome-do-arquivo-incluindo-extensão
```

Se você precisa adicionar todos os arquivos do diretório, basta digitar:

```
git add .
```

Saber o status do projeto é importante. Com o comando abaixo você consegue ver quais arquivos estão fora do controle, quais foram modificados e estão esperando por uma descrição de modificação etc:

```
git status
```

Voltando ao estágio anterior do adicionamento:

Commit – Comitando:

```
git commit -m "Mensagem do commit"
```

Adicionando e comitando ao mesmo tempo:

```
git commit -a -m "Mensagem do commit"
```

# Voltando commits a versões anteriores

Voltar um commit:

```
git reset HEAD~1
```

Voltar dois commits:

```
git reset HEAD~2
```

Voltando um commit e deixando o arquivo no estagio anterior:

```
git reset HEAD~1 --soft
```

Voltando um commit e excluindo o arquivo, deixando no estágio anterior:

```
git reset HEAD~1 --hard
```

Verificando o histórico de commits:

```
git log
```

Verificando o que foi mudado, diferença entre um arquivo e outro:

```
git log -p
```

Verificando os 2 últimos commits:

```
git log -p -2
```

Mostrando as estatísticas de todos os commits:

```
git log --stat
```

Mostrando todos os commits, cada um em uma linha:

```
git log --pretty=oneline
```

Mostrando todos os commits dos últimos 2 dias até o momento atual

```
git log --since=2.days
```

Criando um branch – uma ramificação

```
git checkout -b nome-do-branch
```

```
git branch
```

Voltando para o branch master

```
git checkout master
```

# Jogando o branch criado no branch master

Entre como branch master:

```
git merge nome-do-branch-que-foi-criado
```

# Grudando o branch criado no branch master sem o commit

Somente localmente – localhost, entre como branch master:

```
git rebase nome-do-branch-que-foi-criado
```

Removendo um branch:

```
git branch -D nome-do-branch
```

Vendo branchs remotos:

```
git branch -a
```

Mostrando o início do hash, quem comitou, quanto tempo atrás, mensagem: descrição do commit:

```
git log --pretty=format: "%h - %an, %ar : %s"
```

Deletando arquivos:

```
git rm nome-do-arquivo
```

Deletando todos os aquivos removidos ao mesmo tempo:

```
git ls-files --deleted | xargs git rm
```

# Ignorando arquivos

Existem alguns arquivos que muito provavelmente você não vai precisar versionar, como por exemplo os arquivos de cache do SASS, arquivos de configuração e etc. Nesse caso você precisa fazer com que o controle de versão ignore estes arquivos. Para tanto, crie um arquivo chamado **.gitignore**. Feito isso, dentro deste arquivo, digite o nome ou o endereço das pastas que você quer ignorar. Um exemplo:

```
# See https://help.github.com/ignore-files/ for more about ignoring files.
#
# If you find yourself ignoring temporary files generated by your text editor
# or operating system, you probably want to add a global ignore instead:
# git config --global core.excludesfile ~/.gitignore_global

# Ignore bundler config
/.bundle

# Ignore the build directory
/build

# Ignore Sass' cache
/.sass-cache
```

```
.cache
.rvmrc

vendor/*

.DS_Store

# Vim
*.swp
*.swo

Gemfile.lock
.vagrant
Vagrantfile

# rbenv
.ruby-version

# Ignore deploy related files
deploy

Gemfile.lock
```

O arquivo **.gitignore** fica na raiz do projeto.

# Clonando e puxando alterações de projetos

Clonando um projeto remoto:

```
git clone url-do-projeto
```

Fazendo um clone de outros branchs:

```
git checkout -b nome-do-branch origin/ nome-do-branch
```

Trazendo, puxando as alterações feitas por outros usuários:

Sincronizando tudo que está no repositório remoto:

```
git pull
```

Enviando o(s) projeto(s), arquivo(s) para o repositório:

```
git push origin master
```

Enviando um branch para o repositório:

```
git push origin nome-do-branch
```

# Tags

As tags servem para marcar uma etapa. Imagine que você vai lançar uma versão, que resolve uma série de problemas. Você pode marcar aquela etapa criando uma tag. Assim fica simples de fazer qualquer rollback do projeto para uma tag específica em vez de voltar para um commit. Você sabe que tudo o que foi feito até aquela tag está funcionando.

Criando tags:

```
git tag versão-da-tag
```

Listando tags:

```
git tag -l
```

```
git push origin master --tags
```

Removendo as tags criadas localmente:

```
git tag -d versão-da-tag
```

Removendo tag no repositório remoto:

```
git push origin :refs/tags/versão-da-tag
```

# Concluindo

Se você quer continuar ou iniciar seus estudos com Git, indico o link do livro citado acima, é um ótimo começo, se tiver problemas com o inglês, encontrará várias versões em português.

O Akita fez um screencast para quem está começando com Git. Vale a pena ver.

O pessoal da CodeSchool juntamente com o GitHub fizeram uma página exclusivamente para ensinar Git na prática. Visite aqui.

Há também a documentação do Git que é bastante completa.

## Leia mais aqui no Tableless:

- Deploy automático com GIT e Netlify
- Deploy usando git pull e hooks
- Plugins de GIT para Sublime e Brackets
- Slides para devs #10 – GIT
- Criando páginas web para seus repositórios com o GitHub Pages

## Você vai gostar de ler:

Deploy automático com GIT e Netlify

Deploy usando git pull e hooks

Plugins de GIT para Sublime e Brackets

Slides para devs #10 – GIT

Criando páginas web para seus repositórios com o GitHub Pages

## Categorias

tecnologia-e-tendências (315)

geral (297)

artigos (290)

técnicas-e-práticas (269)

javascript (264)

código (212)

browsers (206)

html (190)

css (183)

mercado (121)

## SOBRE

SOBRE O TABLELESS

CONTATO

ANUNCIE

SEJA UM AUTOR

FAZEMOS CÓDIGO FRONT-END

## ACOMPANHE

WEBINARS

FÓRUM

CANAL NO MEDIUM

## COMUNIDADE

UDACITY | CURSO DE REACT

FEMUG

MEETUPCSS

PODCAST ZOFE

BRAZILJS

DEVNAESTRADA

FRONT-END BRASIL

**Escrito pela e para a comunidade web brasileira. Ajude.**