# Quais são os comandos git que você precisa saber?

Então, vamos mencionar e desenvolver os principais comandos git que você precisa conhecer para gerenciar seus projetos, seja no github, seja em outras plataformas.

Para executar esses comandos git, basta abrir o prompt de comando ou CMD no seu computador. Usaremos o github como exemplo de repositório.

## 1. git commit

O **commit** é um comando importantíssimo. Ele leva as mudanças de um ambiente local para o repositório no git, permitindo ainda a inserção de uma mensagem descritiva. Assim, a cada mudança ou finalização de uma tarefa, a pessoa desenvolvedora pode submeter seus feitos e deixar claro para as outras pessoas o que ela fez.

```
$ git commit -m "Commit for the people of medium"
[develop f3dab06] Commit for the people of medium
 1 file changed, 6 insertions(+)
```

Fonte: Towards Data Science

## 2. git add

Um comando muito similar ao “commit” e que trabalha com ele é o “**add**”. Com essa palavra-chave, nós preparamos arquivos para o próximo “commit”, ou seja, para subir para o repositório na web. É possível adicionar um único arquivo ou todos os arquivos modificados de uma única vez.

Para um único arquivo, use “git add nome_do_arquivo”. Para preparar todos os arquivos para atualização (incluindo as exclusões), use “git add -A”. Para preparar somente as adições, use “git add .”

## 3. git init

O **init** é o primeiro dos comandos git que se usa para começar um repositório. Isto é, o que ele faz é transformar uma pasta com códigos no seu HD em uma pasta monitorada pelo git, que será

carregada para a plataforma e estará visível para outras pessoas. Ou então cria um repositório novo, do zero. Exemplo: “git init”

## 4. git clone

Para começar, muitas pessoas optam por uma alternativa ao init: o git clone. A partir dele, você clona um código de um repositório para a sua máquina para então começar a trabalhar nele. Pode ser um projeto de uma pessoa da sua empresa, um projeto de colegas da faculdade ou **até mesmo uma aplicação open-source** para a qual você julgou interessante colaborar.

```
git clone git://github.com/schacon/grit.git
```

Fonte: Comandos Git

## 5. git status

Para saber algumas informações sobre a ramificação na qual você está trabalhando agora, use o “**status**”. Esse comando esclarece quais arquivos foram alterados e faz uma comparação com relação à ramificação principal. Exemplo: “git status”

## 6. git branch

Aliás, falando em ramificações, precisamos falar logo sobre o termo branch. Para trabalhar em equipe, você pode criar diferentes branches, e o git administra todas elas em paralelo para evitar problemas de versão. Então, posteriormente, com um comando que veremos, é possível unificar as ramificações.

O **comando “git branch”** cria novas branches. Mas também pode funcionar como uma forma de verificar as ramificações já existentes.

Depois de criar uma, você precisa de um “push” para subir essa ramificação. Assim:

“git push -u <remote> <nome-da-branch>”.

Por sua vez, para deletar uma branch, use:

git branch -d <nome-da-branch>

## 7. git merge

Depois de programar em uma branch, você tem que fazer uma conjunção dela com outras para de fato subir as alterações. É só colocar o nome da branch que desejamos mesclar com a principal depois do termo **merge**.

```
$ git merge develop
Updating d6288c9..f3dab06
Fast-forward
 test.py | 7 +++++++
 1 file changed, 7 insertions(+)
```

Fonte: Towards Data Science

## 8. git checkout

Para fazer o merge corretamente, é preciso olhar esse outro comando, o **checkout**. O objetivo dele é fazer a pessoa programadora mudar de branch. Você pode usar o “git branch” para saber quais existem e depois trocar de uma para outra.

É importante destacar que é preciso fazer um checkout para a master branch quando queremos captar as mudanças de outra ramificação.

```
$ git checkout develop
Switched to branch 'develop'
Your branch is up to date with 'origin/develop'.
```

Fonte: Towards Data Science

## 9. git revert

O revert é um dos comandos git aplicados para garantir a segurança dos nossos projetos. Permite desfazer algum commit e recuperar uma versão saudável, seja localmente, seja remotamente.

Para usá-lo, é preciso primeiro executar um “git log--oneline” para obter o número do hash. Com o hash, então, é possível digitar: “**git revert 'nº do hash'”.**

## 10. git rm

O git rm é um comando muito útil para remover arquivos do git e parar de monitorá-los, ou seja, de associá-los ao repositório.

```
git rm -f {arquivo}
```

Fonte: Comandos Git

## 11. git pull

Antes de começar a programar em algum repositório, é bom também executar um "**pull**". Esse comando traz para a sua máquina todas as mudanças que foram realizadas na plataforma. Ou seja, é uma forma de atualizar a sua versão da aplicação com o que foi alterado remotamente.

```
git pull origin develop
```

Fonte: Comandos Git

## 12. git stash

O **stash** serve para criar uma pilha de alterações que serão enviadas posteriormente para o repositório. É uma boa forma de guardar algumas mudanças em espera enquanto você muda de branch para trabalhar em outros aspectos do sistema. É ideal para sistemas grandes, com muitas ramificações que demandam essa flexibilidade da pessoa programadora. Exemplo: "git stash"

## 13. git config

O **config** é um comando inicial para vincular o trabalho no repositório com sua conta no github. Assim, é configurado com o nome e com o e-mail.

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

Fonte: Comandos Git

### 14. git reset

O **reset** é outra forma de voltar ao último estado saudável do seu sistema, uma alternativa ao revert. Funciona assim: “git reset --hard HEAD~1”.

### 15. git push

O **push** serve para subir as alterações de uma ramificação para um certo repositório. Ele entrega todos os commits e a mensagem. Exemplo: “git push”

```
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 459 bytes | 459.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/jaimezorno/test.git
   6f92286..f3dab06  develop -> develop
```

Fonte: Towards Data Science

## Por que usar o git?

Agora, vamos examinar alguns motivos pelos quais toda pessoa profissional de tecnologia deve usar o git, independentemente se sua área é desenvolvimento web, ciência de dados ou outra.

### Documentação

Um dos pontos é a documentação, uma prática fundamental no universo de TI. O ato de documentar sistemas é importante para torná-los transparentes e fáceis de entender para outras pessoas. Se uma pessoa trabalha em uma empresa e documenta bem seus projetos, outras pessoas têm fácil acesso e conseguem prosseguir com o trabalho, mesmo sem ter contato direto com a equipe.

Ou seja, **a documentação favorece a comunicação** e deve ser encorajada. Com as ferramentas que implementam o git, como o github, você dispõe de uma plataforma para hospedar o código, criar

arquivos de explicação/tutorial, comentar os códigos e as alterações e criar diversas informações úteis para quem for ler depois.

Ou seja, é possível pensar na continuação e na manutenção de uma aplicação de forma segura e precisa.

## Redução no tempo de deploy

Outro motivo para considerar o uso de um sistema de versão como o git é a redução do tempo de deploy e simplificação desse processo. Atualmente, diversas demandas em TI fazem com que as pessoas tenham que atualizar os sistemas rápida e constantemente, de forma integrada, como **os ideais de DevOps e MLOps.**

Assim, ter um deploy rápido e eficiente é necessário. O git ajuda nessa questão oferecendo às pessoas profissionais a capacidade de subir alterações com facilidade, controlando as ramificações, como já vimos.

## Exibição do seu trabalho

Para as pessoas que aspiram por boas oportunidades de emprego e até para as que buscam melhorias na carreira, o git e as plataformas que utilizam o git são essenciais. Elas permitem mostrar o seu trabalho, os seus projetos, a frequência de codificação, as suas soluções e a sua organização como pessoa desenvolvedora.