**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.

commit

Fonte: [Towards Data Science](https://towardsdatascience.com/git-for-noobs-7a846ce98c9a)

**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.

clone

Fonte: [Comandos Git](https://comandosgit.github.io/)

**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**.



Fonte: [Towards Data Science](https://towardsdatascience.com/git-for-noobs-7a846ce98c9a)

**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.

checkout

Fonte: [Towards Data Science](https://towardsdatascience.com/git-for-noobs-7a846ce98c9a)

**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.



Fonte: [Comandos Git](https://comandosgit.github.io/)

**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.

pull

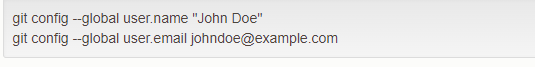
Fonte: [Comandos Git](https://comandosgit.github.io/)

**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.



Fonte: [Comandos Git](https://comandosgit.github.io/)

**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”

