10 Principais comandos Git

Git init

Para começar um projeto que ainda não seja um repositório (ou repo), o Git Init costuma ser o primeiro comando que você vai usar, pois vai precisar de um subdiretório .git na raiz do seu projeto.

Esse comando cria um repositório vazio ou transforma uma pasta que você já tem, e que não está com controle de versão, em um repositório.

```
git init
```

Com sua pasta de trabalho devidamente iniciada, é hora de começar a preenchê-la.

Git clone

O Git clone é um comando para baixar o código-fonte existente de um repositório remoto (como o Github, por exemplo).

Existem algumas maneiras de baixar o código-fonte, mas eu prefiro o clone com o modo https:

```
git clone <https://url-do-link>
```

Quando você clonar um repositório, o código é copiado para a o seu computador e continua linkado ao original, como foi explicado lá na descrição do que é um sistema distribuído.

Se você quiser desvincular a sua cópia do original, rode o comando abaixo:

```
git remote rm origin
```

Git branch

Com *branches* (ou ramificações), vários desenvolvedores podem trabalhar paralelamente no mesmo projeto. Assim, cada um pode codar a sua parte sem se atrapalharem.

Por isso, esse é um dos comandos Git mais importantes. Pode-se usar o comando **git branch** para criar, listar e excluir *branches*.

Criando uma nova *branch*:

```
git branch <nome-da-branch>
```

Este comando criará uma *branch* local. Para upar a nova *branch* para o repositório remoto, você precisa usar o seguinte comando:

```
git push -u <remote> <nome-da-branch>
```

Para ver as ramificações:

```
git branch
```

ou

```
git branch --list
```

Deletando uma *branch*:

```
git branch -d <nome-da-branch>
```

Git checkout

Este é um dos comandos Git mais usados. Para trabalhar em uma *branch*, primeiro você precisa mudar para ela. Não ir para a *branch* que você acabou de criar e na qual quer trabalhar é um erro bastante comum no começo.

Então, usamos o *git checkout* principalmente para mudar de um *branch* para outro. Também podemos usá-lo para verificar arquivos e *commits*:

```
git checkout <nome-da-ramificação>
```

Há ainda um comando de atalho que te permite criar e ir para um *branch* de uma vez só:

```
git checkout -b <nome-da-branch>
```

Git status

O comando status do Git fornece algumas informações sobre a *branch* em que você estiver no momento, como seu nome, se ela está atualizada em relação à master e quais arquivos foram alterados.

```
git status
```

Git add

Quando criamos, modificamos ou excluímos um arquivo, essas alterações ocorrerão em nosso ambiente local e não serão incluídas no próximo *commit* (a menos que alteremos as configurações).

Precisamos usar o comando **git add** para incluir as alterações de um arquivo em nosso próximo *commit*.

Para adicionar apenas um arquivo:

```
git add <arquivo>
```

Para adicionar, de uma vez, todos os arquivos modificados:

```
git add -A
```

O comando **git add** não altera o repositório e as alterações não são salvas até usarmos o *git commit*.

Git commit

Este comando é como definir um ponto de verificação no processo de desenvolvimento, para o qual você pode voltar mais tarde, se necessário.

```
git commit -m "mensagem explicando a mudança no código"
```

Git push

Após confirmar as alterações, a próxima coisa que você deseja fazer é enviar as alterações para o servidor remoto.

O comando **git push** envia e salva suas confirmações no repositório remoto.

```
git push <remote> <nome-do-branch>
```

No entanto, se seu branch for criado recentemente, você também precisará fazer upload do branch com o seguinte comando:

```
git push -u origin <nome-do-branch>
```

Git pull

O comando **git pull** é usado para obter atualizações do repositório remoto. O comando de pull depende do referencial de onde ele foi feito, ou seja, um git pull feito da sua máquina vai puxar informações do repositório original para ela.

Mas um git pull feito a partir do repositório original vai puxar as informações da sua máquina. Percebe?

Este comando é uma combinação de *git fetch* (baixa as alterações do repositório remoto mas não mescla elas com o seu) e *git merge* (que mescla tudo junto), o que significa que, quando usamos o **git pull**, ele recebe as atualizações do repositório remoto (**git fetch**) e aplica imediatamente as alterações mais recentes no seu local (**git merge**).

```
git pull <remote>
```

Git revert

Existem várias maneiras de desfazer nossas alterações local ou remotamente (dependendo da necessidade), mas devemos usar esses comandos com cuidado para evitar problemas.

Uma maneira segura de desfazer os *commits* é usando **git revert**.

```
git revert 'número do hash'
```

O número do hash pode ser conseguido pelo comando:

```
git log -- oneline
```

Git merge

Quando você conclui o desenvolvimento em sua *branch* e tudo funciona bem, sem conflitos, a etapa final é mesclar as *branches*, isso é feito com o comando **git merge**.

Como falamos no tópico sobre git pull, esse comando vai mesclar, no seu repositório local, todas as alterações feitas.

```
git merge <nome-da-branch>
```