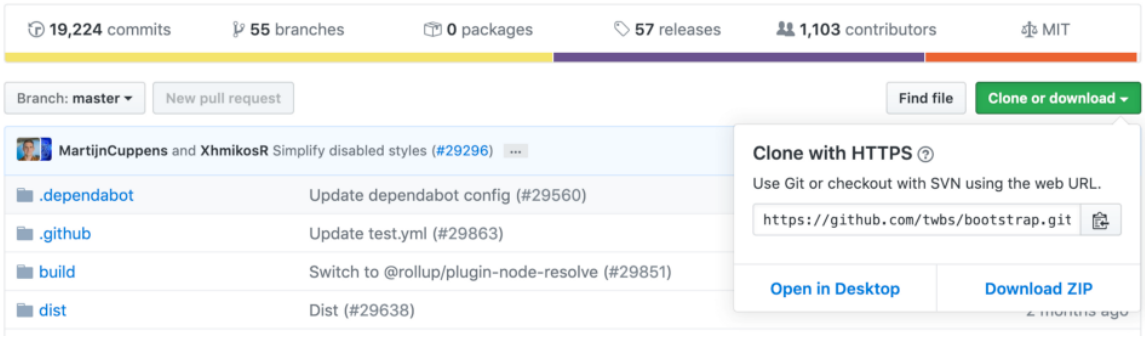
1. Git clone

Git clone é uma comando para baixar o código-fonte existente de um repositório remoto (como, por exemplo, o Github). Em outras palavras, git clone, basicamente, faz uma cópia idêntica da versão mais recente de um projeto em um repositório e a salva em seu computador.

Há algumas maneiras de baixar o código-fonte, mas, em geral, eu prefiro a maneira de **clonar com https**:

```
git clone <https://link-com-o-nome-do-repositório>
```

Por exemplo, se eu quiser baixar um projeto do Github, tudo o que eu preciso fazer é clicar no botão verde (que diz "Clone or download"), copiar o URL da caixa logo abaixo e colá-lo após o comando git clone que mostrei logo acima.



Exemplo com o código-fonte do Bootstrap no Github

Isso fará uma cópia do projeto no seu espaço de trabalho local para que você possa começar a trabalhar nessa cópia.

2. Git branch

Branches (algo como ramificações, em português) são altamente importantes no mundo do git. Usando as *branches*, vários desenvolvedores conseguem trabalhar em paralelo no mesmo projeto simultaneamente. Podemos usar o comando git branch para criar, listar e excluir as *branches*.

Como criar uma branch:

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

Esse comando criará uma branch **em seu local de trabalho**. Para fazer o push (algo como enviar) da nova branch para o repositório remoto, você precisa usar o comando a seguir:

```
git push -u <local-remoto> <nome-da-branch>
```

Como ver as branches:

```
git branch ou git branch --list
```

Como excluir uma branch:

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

3. Git checkout

Esse também é um dos comandos do Git mais usados. Para trabalhar em uma branch, primeiro, é preciso "entrar" nela. Usamos **git checkout**, na maioria dos casos, para trocar de uma branch para outra. Também podemos usar o comando para fazer o checkout de arquivos e commits.

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

Existem alguns passos que você precisa seguir para trocar de branch com sucesso:

- As alterações em sua branch atual devem estar em um commit ou em um stash antes de você fazer a troca
- A branch na qual você quer fazer o checkout deve existir no seu espaço de trabalho local

Também existe um comando de atalho que permite criar e automaticamente trocar para a branch criada ao mesmo tempo:

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

Esse comando cria a branch em seu espaço de trabalho local (a flag -b representa a branch) e faz o checkout na nova branch logo após sua criação.

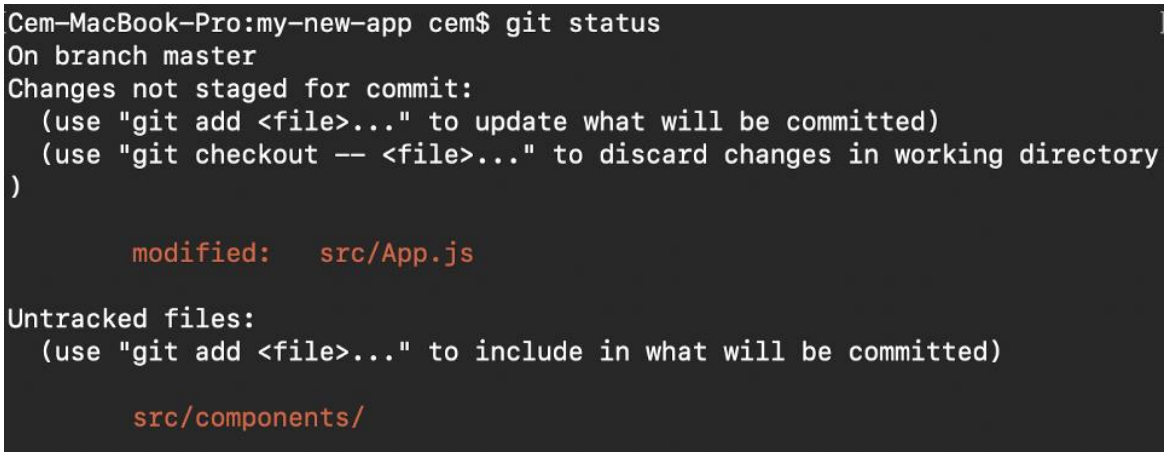
4. Git status

O comando git status nos dá todas as informações necessárias sobre a branch atual.

```
git status
```

Obtemos as seguintes informações:

- Se a branch em que estamos no momento está atualizada
- Se precisamos fazer o commit, push ou pull de algo
- Se os arquivos estão em fase de stage, fora dessa fase ou se não estão sendo rastreados
- Se arquivos foram criados, modificados ou excluídos



Git status nos dá informações sobre a branch e os arquivos

5. Git add

Ao criarmos, modificarmos ou excluirmos um arquivo, essas alterações acontecerão em nosso espaço de trabalho 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 ou vários arquivos em nosso próximo commit.

Para adicionar um único arquivo:

```
git add <arquivo>
```

Para adicionar tudo ao mesmo tempo:

```
git add -A
```

Ao ver a imagem acima, na 4ª seção, você verá que existem nomes de arquivo que estão em vermelho - isso significa que os arquivos ainda não estão em fase de stage. Esses arquivos não serão incluídos em seus commits.

Para incluí-los, precisamos usar git add:

Os arquivos em verde agora estão em fase de stage com o git add

Importante: o comando git add não altera o repositório. As alterações não são salvas até que se use o git commit.

6. Git commit

Talvez esse seja o comando mais usado do Git. Quando chegamos a determinado ponto em desenvolvimento, queremos salvar nossas alterações (talvez após uma tarefa ou resolução de problema específica).

Git commit é como definir um ponto de verificação no processo de desenvolvimento. Você pode voltar a esse ponto mais tarde, se necessário.

Também precisamos escrever uma mensagem breve para explicar o que desenvolvemos ou alteramos no código-fonte.

```
git commit -m "mensagem do commit"
```

Importante: git commit salva suas alterações no espaço de trabalho local.

7. Git push

Após fazer o commit de suas alterações, a próxima coisa a fazer é enviar suas alterações ao servidor remoto. Git push faz o upload dos seus commits no repositório remoto.

```
git push <repositório-remoto> <nome-da-branch>
```

Entretanto, se a sua branch foi recém-criada, também é preciso fazer o upload da branch com o seguinte comando:

```
git push --set-upstream <repositório-remoto> <nome-da-branch>
```

ou

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

Importante: git push somente faz o upload das alterações que já estão em um commit.

8. Git pull

O comando **git pull** é usado para obter as atualizações de um repositório remoto. Esse comando é uma combinação de **git fetch** e **git merge**, o que significa que, quando usamos git pull, ele recebe as atualizações do repositório remoto (git fetch) e aplica imediatamente as alterações mais recentes em seu espaço de trabalho local (git merge).

```
git pull <repositório-remoto>
```

Essa operação pode causar conflitos que você precisará resolver manualmente.

9. Git revert

Às vezes, precisamos desfazer as alterações que fizemos. Existem várias maneiras de se desfazer as alterações em nosso espaço de trabalho local ou remotamente (dependendo do que você necessita), mas devemos usar esses comandos com cuidado para evitar exclusões indesejadas.

Uma maneira segura de desfazer nossos commits é usando **git revert**. Para ver nosso histórico de commits, primeiro, precisamos usar **git log--oneline:**

```
Cem-MacBook-Pro:my-new-app cem$ git log --oneline
3321844 (HEAD -> master) test
e64e7bb Initial commit from Create React App
```

Histórico de commits da minha branch master

Em seguida, precisamos apenas especificar o código hash ao lado do commit que desejamos desfazer:

```
git revert 3321844
```

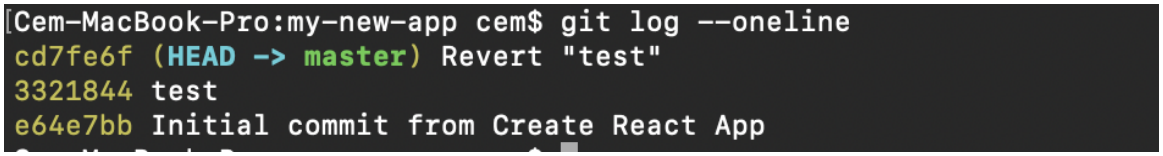
Depois disso, você verá uma tela igual ao que vemos abaixo - basta pressionar **shift + q** para sair:

```
Revert "test"

This reverts commit 332184490ef2b5db289d85ed3f1a13ff2d5f94b9.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Committer: Cem <cem@Cem-MacBook-Pro.local>
#
# On branch master
# Changes to be committed:
#       modified:   src/App.js
#       deleted:    src/components/myFirstComponent.js
#
~
~
~
~
~
~
~
~
~
~
"~/Desktop/my-new-app/.git/COMMIT_EDITMSG" 14L, 384C
```

O comando git revert desfará o commit especificado, mas criará outro commit sem excluir o antigo:



Novo commit do "revert"

A vantagem de se usar **git revert** é não tocar no histórico de commits. Isso significa que você ainda pode ver todos os commits do seu histórico, mesmo os revertidos.

Outra medida de segurança está no fato de que tudo acontece em nosso sistema local, a menos que façamos o push de tudo para o repositório remoto. É por isso que o uso de git revert é mais seguro e é a forma preferida de desfazer nossos commits.

10. Git merge

Quando você concluir o desenvolvimento em sua branch e quando tudo funcionar bem, a etapa final é fazer o merge (mesclar ou unir, em português) da branch com a branch pai (dev ou master/main, em geral). Isso é feito com o comando `git merge`.

Git merge, basicamente, integra sua branch com o recurso e todos os seus commits na branch de desenvolvimento (dev) ou na branch principal (master ou main). É importante lembrar que, primeiro, você precisa estar na branch específica na qual você quer fazer o merge de sua branch com o recurso.

Por exemplo, ao querer fazer o merge de sua branch do recurso na branch dev:

Primeiro, troque para a branch dev:

```
git checkout dev
```

Antes do merge, atualize sua branch dev local:

```
git fetch
```

Por fim, faça o merge da sua branch do recurso em dev:

```
git merge <nome-da-branch-com-o-recurso>
```

Dica: certifique-se de que sua branch dev tem a versão mais recente antes de fazer o merge de suas branches de recurso. Do contrário, você pode ter que lidar com conflitos e outros problemas indesejados.

Esses são os meus 10 comandos mais usados do Git, com os quais eu lido diariamente ao programar. Existe muito mais para se aprender sobre o Git e eu tentarei explicar o resto em artigos separados.

Se quiser aprender mais sobre desenvolvimento para a web, não se esqueça de [seguir o autor no Youtube]!