DigitalHouse>
Coding School

Certified Tech Developer

The Ultimate Degree

git

GitHub

>_índice

O que é Git?

É um **software** de **controle** de **versões** que **registra** as **alterações** realizadas em um arquivo ou conjunto de arquivos ao longo do **tempo.** Desta forma, podemos recuperar e ter **acesso** a versões específicas quando quisermos.

Por que usar um sistema de controle de versões?

Usar um sistema de controle de versões (*VSC*), permite **reverter arquivos** e **projetos inteiros** a um estado anterior, **comparar alterações** ao longo do tempo, ver **quem modificou** pela última vez, descobrir quando um erro foi introduzido e muito mais.

Instalação do Git

- ⇒ Vá ao site oficial e baixe o executável.
- ⇒ Execute o arquivo que baixamos.
- ⇒ Se o sistema operacional for o Windows, além de instalar o Git, um terminal chamado **Git Bash** será instalado em nossa máquina.
- ⇒ Uma vez instalado o Git, estará disponível o comando `git` para executar no terminal.
- ⇒ Para verificar se a instalação foi realizada corretamente, abra um terminal e execute o comando `git --version`.

O que é GitHub?

GitHub é um **plataforma** onde podemos **armazenar** os **arquivos** e **projetos** de programação de maneira **gratuita**. Para utilizar os seus benefícios, basta criar uma conta.

O que é um repositório?

É o local onde serão **armazenados** os **arquivos** de nosso projeto. No GitHub podemos ter a quantidade de projetos que quisermos, onde **cada projeto** corresponderá a **um repositório**.

>_

Tipos de repositórios

Os repositórios hospedados no GitHub são chamados de repositórios **remotos**, enquanto os que estão armazenados em nosso PC, são chamados de repositórios **locais**. É necessário criar um **vínculo** entre ambos, para que possamos manter os arquivos locais **atualizados** com os que estão conectados na nuvem.

>_

Criar um repositório remoto (GitHub)

⇒ Uma vez iniciada a sessão no GitHub, clique no ícone **+** localizado na barra principal e escolha a opção **New repository**.

⇒ Veremos na tela um formulário.

⇒ Escolha o nome para o repositório. Podemos nomeá-lo como quisermos, mas deve ser um nome que não tenhamos usado para outro repositório.

⇒ Desça a tela e clique no botão **Create Repository**.

>_

Criar um repositório local (PC)

⇒ Crie uma pasta no computador para armazenar o projeto. Esta pasta será o **repositório local**.

⇒ Dentro da pasta, abra um terminal e execute o comando `git init`.

⇒ Este comando **inicializa** um **repositório local** na pasta do projeto.

>_

Adicionar nossa identidade ao repositório local

Para que o Git **acompanhe totalmente** as **alterações** feitas, precisamos informar ao repositório quem somos. Para isso:

⇒ Abra um terminal no mesmo caminho do nosso repositório local.

- ⇒ Execute o comando `git config user.name "nomeDeUsuario"` , onde, entre aspas, devemos digitar nosso nome de usuário, tal qual está definido no GitHub.
- ⇒ Para verificar se adicionamos corretamente nosso nome de usuário, execute o comando `git config user.name` e pressione *Enter*.
- ⇒ Execute o comando `git config user.email "nome@email.com"` , onde, entre aspas, devemos digitar o email que registramos em nossa conta do GitHub.
- ⇒ Para verificar se adicionamos corretamente nosso email, execute o comando `git config user.email` e pressione *Enter*.

Para configurar nossa identidade de maneira global e não ter que declarar sempre nosso email e nome de usuário, adicione o termo `--global` .

- ⇒ `git config --global user.name "nomeDeUsuario"`
- ⇒ `git config --global user.email "nome@email.com"`

>_

Conectar o repositório local com o repositório remoto

Para que nosso repositório local saiba onde queremos enviar nossos arquivos, é necessário especificá-lo.

- ⇒ Ter criado previamente um repositório no GitHub.
- ⇒ Vá ao local do repositório remoto e copie a URL.
- ⇒ Digite o comando `git remote add origin` .
- ⇒ Cole a **URL** após a palavra *origin* (deixando um espaço no meio) e pressione *Enter*.
- ⇒ Para verificar se o passo anterior foi realizado corretamente, execute o comando `git remote -v`. No Terminal deve aparecer a palavra `origin` seguida da URL.

>_

O que é um commit?

Cada vez que subimos arquivos (novos ou modificados) a um repositório remoto, eles são carregados em forma de um pequeno **pacote de arquivos.** Cada pacote tem uma **data de criação** (*timestamp*) e um **autor**.

É através dos **commits** que vamos fazer o acompanhamento das alterações que vão sendo realizadas nos projetos, já que cada um deles gera um **ponto cronológico** na linha do tempo do projeto.

>_

O que significa quando um arquivo está em acompanhamento?

Quando enviamos um **arquivo** ao repositório, estamos dizendo ao Git que queremos fazer um **acompanhamento dele** através do tempo,ou seja, necessitamos que se guarde o **estado atual** desse arquivo para que cada vez que fizermos uma alteração nova e a enviemos, possamos **comparar estados** e ver como estava em determinado momento. Também é uma forma de **segui-lo** ao longo do projeto.

>_

Subindo arquivos a um repositório remoto

Para subir nossos arquivos para a nuvem, devemos seguir os seguintes passos:

- ⇒ Abra um terminal no caminho de nosso repositório local.
- ⇒ Execute o comando `git status` para ver o **estado** de nossos arquivos (aqueles em vermelho são os arquivos que ainda não estão em acompanhamento).
- ⇒ Execute o comando `git add .` para indicar que queremos **adicionar todos** os arquivos ao repositório.
- ⇒ Para **adicionar somente um arquivo**, execute o comando `git add arquivo.extensao` (`ex: git add texto.txt`) onde deveremos indicar tanto o nome como a extensão do arquivo.
- ⇒ Execute o comando `git status` para ver o **estado** de nossos arquivos novamente (aqueles em verde são os arquivos que serão adicionados ao repositório, portanto estarão em acompanhamento).
- ⇒ Para **confirmar** que queremos subir de maneira definitiva aqueles arquivos que adicionamos, executamos o comando `git commit -m "mensagem"`, onde, entre aspas, deveremos digitar, se possível, uma mensagem curta que resuma o trabalho que estamos enviando.
- ⇒ Para **enviar** os arquivos ao repositório remoto, execute o comando `git push origin master`.

>_

Atualizando arquivos de um repositório local

Para atualizar os arquivos de nosso repositório local a respeito dos que estão no repositório remoto, devemos executar o comando `git pull origin master`.

>_

Clonando arquivos de um repositório remoto

Para **baixar** pela primeira vez um repositório remoto em nossa máquina, teremos que **cloná-lo**.

⇒ Abra um terminal no caminho onde quer clonar o projeto.
⇒ Copie a **URL** do repositório que queremos clonar.
⇒ Digite o comando `git clone` .
⇒ Cole a **URL** depois da palavra *clone* (deixando um espaço no meio) e pressione *Enter*.

>_

Resolvendo conflitos

Uma das grandes vantagens do Git e do GitHub é que **muitas pessoas** podem trabalhar em um **mesmo projeto** em paralelo. Neste cenário, é bem possível que duas ou mais pessoas modifiquem o mesmo arquivo e é aí que geralmente aparecem os **conflitos**.

➔ Detectando o conflito

Imaginemos que estamos trabalhando nos estilos de um website, no arquivo `styles.css`. Ao mesmo tempo, outro dos membros da equipe — vamos chamá-la Ana — decide fazer uma modificação **nesse mesmo arquivo** e sobe imediatamente suas alterações ao repositório:

```
body {
  background-color: yellow;
  font-family: monospace;
}
```

Nós, sem sabermos que Ana subiu essas alterações, seguimos trabalhando em nosso arquivo normalmente, mas, diferente dela, na linha 2, atribuímos um fundo azul ao body:

```
body {
  background-color: blue;
  font-family: monospace;
}
```

Minutos mais tarde, quando terminamos nossas alterações e fazemos o commit e o push, o console nos devolve um lindo erro que parece estar escrito em “élfico antigo”:

```
To https://github.com/whesslleyrimar/projeto.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/whesslleyrimar/projeto.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Mas, o que quer dizer esta mensagem?

Que Ana foi mais rápida que nós e, há **novas alterações** no GitHub. Estas alterações não coincidem, ou seja, a versão que temos no repositório local não é a mesma que está no repositório remoto. O Git nos diz: "Talvez queira integrar primeiro as alterações remotas antes de voltar a fazer o push".

Portanto, sabendo que no repositório remoto existem alterações que **não temos**, o próximo passo será atualizarmos e trazermos essas alterações com o comando:

```
git pull origin master
```

E agora, o console nos devolve outra mensagem em "hieróglifo":

```
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 691 bytes | 28.00 KiB/s, done.
From https://github.com/whesslleyrimar/projeto
 * branch            main       -> FETCH_HEAD
   5037c50..71a17c1  main       -> origin/main
Auto-merging styles.css
CONFLICT (content): Merge conflict in styles.css
Automatic merge failed; fix conflicts and then commit the result.
```

Desta mensagem, somente nos importam as três últimas linhas:

```
Auto-merging styles.css
CONFLICT (content): Merge conflict in styles.css
Automatic merge failed; fix conflicts and then commit the result.
```

O que aconteceu?

O Git tentou mesclar nossas alterações junto com as que trouxemos com o pull e não foi possível: nós e Ana editamos a mesma linha e o Git não sabe qual das duas alterações tem que ficar. **É nosso trabalho esclarecer isso ao Git**.

→ **Como se analisa o conflito**

Em nosso editor de código veremos o seguinte trecho **para cada conflito que haja no projeto**: neste caso sabemos que o conflito está em `styles.css`.

```
<<<<<<< HEAD (Current Change)
body{
    background-color: blue;
    font-family: monospace;
}
=======
body{
    background-color: yellow;
    font-family: monospace;
}
>>>>>>> 71a17c1f95fbc08f142b2f29a54a2c2294c35847 (Incoming Change)
```

A **primeira parte**, entre `<<<<<<< HEAD` e `=======` será nossa **versão local**,com a alteração que fizemos e queremos subir ao GitHub.
A **segunda parte**, entre `=======` e `>>>>>>>` será **a versão que nos veio do GitHub** (em outras palavras, as alterações de Ana).

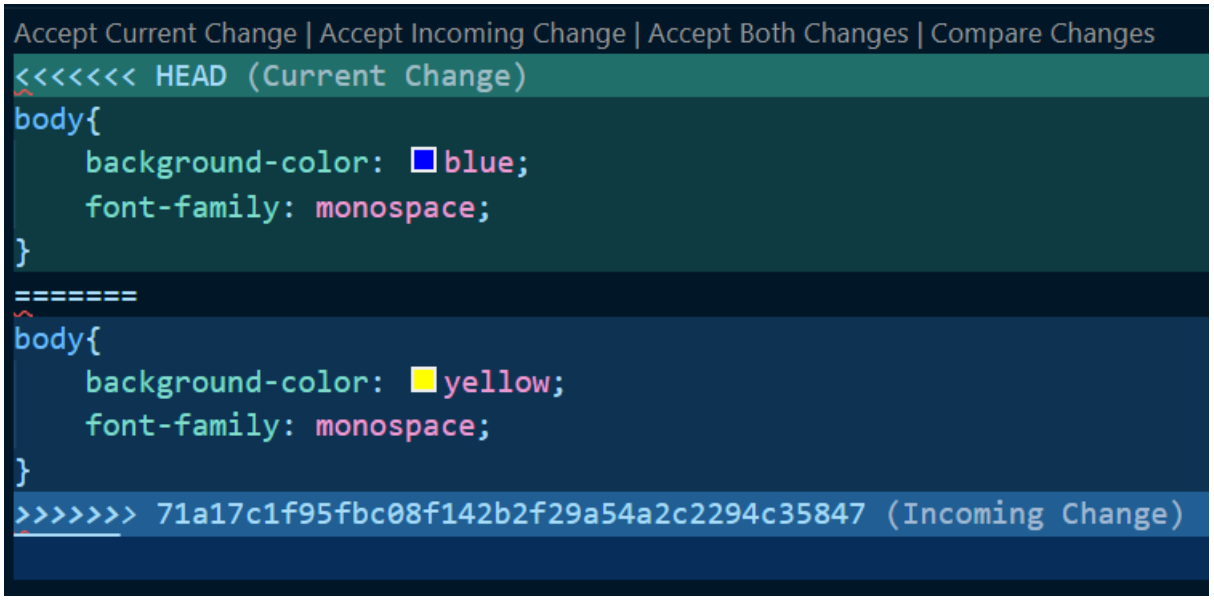
➔ Resolvendo o conflito

Para cada um destes conflitos teremos que decidir por alguma destas três opções:

⇒ Deixamos nosso código, versão local.
⇒ Deixamos o código que vem do GitHub.
⇒ Unimos ambos.

Sem nos importarmos com qual dos três caminhos devemos tomar, não podemos esquecer de apagar as três linhas de texto que o Git adicionou para identificar a zona de conflito.

Se estivermos trabalhando com um editor moderno, como o VS Code, o mais provável é que ele compreenda este formato e nos dê alguma opção para resolver o conflito **com somente um clique**:



Para alguns conflitos, o Git poderá determinar como unificar as alterações de maneira **automática**. Nestes casos todo este trabalho será feito automaticamente.
Pode acontecer de o Git nos pedir que adicionemos, se assim desejarmos, uma mensagem que explique porque esse *merge* foi necessário. Se não digitarmos nada, por padrão deixará registrada a mensagem "Merge branch…".
Para sair do console sem digitar uma mensagem em particular, teremos que pressionar `:q`, ou `Ctrl+x`.
Se nos interessa aprender como digitar uma mensagem, podemos investigar por nossa conta os editores de texto **Nano** e **Vim**.

➔ Subindo os arquivos

Assim que estivermos satisfeitos com a alteração, vamos repetir o processo como se fosse uma nova mudança, ou seja, será necessário voltar a **adicionar** o arquivo com `git add`, neste caso `styles.css`, incluí-lo em um commit e enviá-lo ao servidor com um `git push`.

➔ Evitando conflitos

Uma boa maneira de evitar conflitos é manter os commits relativamente pequenos e subir ao repositório frequentemente. Desta maneira, temos menos probabilidades de que ocorram conflitos e, se ocorrerem, serão **pequenos**.

Outra maneira um pouco mais avançada é o uso de **branches** para trabalhar em paralelo com a versão principal do projeto. O convidamos a pesquisar este assunto por conta própria.

CONFLITOS: FLUXO COMPLETO

⇒ Prepare as alterações que quer enviar para o repositório

⇒ Executa o add, o commit e o push.
⇒ O console nos devolve uma mensagem de erro. Não se pode fazer o push porque no repositório remoto existem alterações que não temos em nosso repositório local e não coincidem as versões.
⇒ Execute um pull para atualizar nosso repositório local e trazer essas alterações.
⇒ Se não há conflitos entre o que tínhamos e o que trouxemos, o Git vai mesclar todas as mudanças de forma automática.
    - Neste caso, o console vai nos pedir que digitemos uma mensagem de commit para explicar o *merge* que acaba de acontecer. Podemos digitar uma mensagem ou sair do console pressionando as teclas `:q` ou `Ctrl+x`.
    - Só precisamos fazer o push para enviar nossas alterações ao repositório remoto.
⇒ Se não puder ser resolvido automaticamente, o console retornará uma nova mensagem nos informando que há conflitos que devemos resolver manualmente antes de carregarmos nossas alterações para o repositório remoto.
⇒ Vá ao editor de código para verificar quais são os arquivos em conflito.
⇒ O Git nos deixa ver a zona de conflito da seguinte maneira:

    <<<<<<< HEAD
    Tudo o que está entre estas duas linhas são nossas **alterações locais**, *o que estivemos trabalhando e queremos integrar com o resto do projeto.*
    =======
    Tudo o que está entre estas duas linhas são as alterações que trouxemos do repositório remoto com o pull.
    >>>>>>> 3216f3fd5ca65cfd3252ae76808d8f659a715fa6

⇒ Por cada conflito que encontramos, temos que decidir se:
    - Mantemos nossa alteração e apagamos o que trouxemos.
    - Mantemos o que trouxemos e apagamos nossa mudança.
    - Ficamos com ambas as mudanças.
⇒ Resolvemos o conflito tomando algum destes três caminhos e apagamos as linhas de texto que o Git adicionou: <<<<<<< HEAD, ======= e >>>>>>> 3216f3fd5ca65cfd3252ae76808d8f659a715fa6
⇒ Uma vez resolvidos todos os conflitos, fazemos o add, o commit e o push para subir estas novas alterações ao repositório remoto.