Como compartilhar o código e não o projeto usando git

Abrir o Github – hospedagem de projetos

Repositório é onde eu guardo os códigos de um determinado projeto

O nome do repositório tem que ser único na minha conta

Para criar um repositório ir no mais do canto superior da tela e clicar em new repositor

Colocar o nome do projeto

Definir se ele é público ou privado

Clicar abaixo para criar o repositório

O primeiro comando git init, precisa instalar primeiro o git

Depois instalar a variável de ambiente – para colocar o caminho da pasta CMD e depois reiniciar o VSCode

/////////

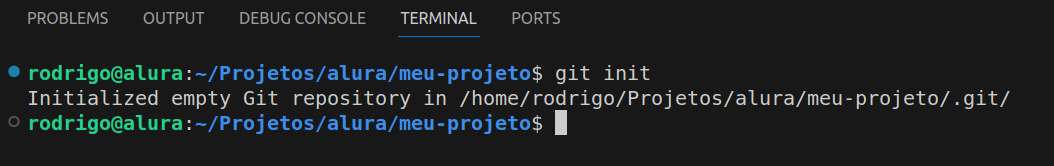
O comando **git init** deve ser utilizado para converter um diretório existente no computador, que geralmente é o diretório de algum projeto, em um repositório Git. No entanto, seu uso requer atenção e cuidado para evitar problemas indesejados. Vamos entender melhor o que o comando git init faz.

**O que é o comando git init?**

Para que possamos realizar o controle de versão de um projeto, registrando as mudanças realizadas nele ao longo do tempo, devemos, primeiramente, transformar o diretório do projeto em um repositório Git. O comando git init é utilizado para esse objetivo, devendo ser executado apenas **uma vez**. Quando executado, ele configura o diretório atual para ser rastreado pelo Git, inicializando um repositório vazio.

**Executando o comando git init**

Suponha que no seu computador exista um diretório chamado meu-projeto, que represente um projeto pessoal seu e que você deseja transformar em um repositório do Git. Para isso, você pode abrir esse diretório no VSCode, abrir uma janela do terminal e executar o comando git init. A saída será algo como:



Repare na imagem anterior que o comando git init foi executado no terminal do VSCode. Observe também que no terminal é indicado qual o diretório no qual o comando foi executado, que no exemplo foi em: ~/Projetos/alura/meu-projeto. É importante se atentar a isso, pois o comando git init transforma o diretório atual em um repositório do Git, logo ele deve ser executado **dentro** do diretório do projeto e não em outros diretórios do computador.

Ao executar o comando, note que a saída no terminal foi a mensagem **Initialized empty Git repository in /home/rodrigo/Projetos/alura/meu-projeto/.git/**. Essa mensagem indica que o comando foi executado corretamente e um repositório local do Git foi criado com sucesso nesse diretório. A partir desse ponto, já podemos trabalhar no projeto, adicionando arquivos, realizando modificações e registrando as mudanças no Git.

**Cuidados com o comando git init**

Aprendemos que o comando git init serve para criar um novo repositório Git e por isso deve ser executado apenas uma única vez. Ou seja, se um diretório já for um repositório Git, não faz sentido rodar novamente o comando git init. Esse é um erro bastante comum de ser cometido.

Se você executar o comando git init em um diretório que já foi inicializado como um repositório Git, a seguinte mensagem será exibida:

Reinitialized existing Git repository **in** /home/rodrigo/Projetos/alura/meu-projeto/.git/

Copiar código

Isso indica que o Git **reinicializou** um repositório já existente, ou seja, o comando git init foi executado em um diretório que já era um repositório Git.

Caso você tenha cometido esse equívoco, não precisa se preocupar, pois todo o histórico de mudanças e commits no projeto não será apagado. O Git detecta que o diretório já era um repositório Git e com isso o comando não tem efeito nenhum.

Na dúvida, antes de executar o comando git init, execute primeiramente o comando git status. Se aparecer a mensagem fatal: not a git repository (or any of the parent directories): .git, isso significa que o diretório atual **não é** um repositório Git e você pode então executar o comando git init.

//////////

**Gabrielle**: Já instalamos o Git e inicializamos o repositório local com o comando git init. Mas agora, como podemos prosseguir para **conectar esse repositório local ao repositório remoto** no GitHub.

**Rodrigo**: Isso é muito importante. O Git possui esse conceito de **repositório local e repositório remoto**. Quando criamos o repositório no **GitHub**, o repositório que se encontra lá é denominado repositório remoto. No entanto, em nosso **computador**, também precisamos criar um repositório, que será o local.

Esses dois repositórios não estão conectados. Agora que instalamos o Git e criamos o repositório local, podemos seguir o tutorial que foi apresentado para estabelecer essa conexão entre o meu repositório local e o repositório remoto no GitHub.

**Adicionar arquivos e vincular repositório**

**Rodrigo**: Vamos retornar ao site do GitHub. Já executamos o comando git init. O próximo comando que aparece no tutorial é git add, que mostra como exemplo um arquivo README.md que não possui existe no nosso projeto. Ao invés disso, precisamos adicionar os arquivos do nosso projeto.

Vamos copiar o comando git add e o colar no terminal integrado do VS Code. No entanto, precisamos adicionar cada um dos arquivos do projeto. Mas, temos vários arquivos neste projeto, como o app.js, index.html e outros. Precisaremos adicionar cada um manualmente?

**Gabrielle**: Temos a opção de adicionar manualmente, informando o nome de cada um dos arquivos, mas às vezes pode ser um pouco trabalhoso, especialmente quando temos um projeto grande com muitos arquivos.

Para facilitar esse processo, temos um atalho no comando do Git, onde podemos **adicionar todos os arquivos de uma só vez**. É o comando git add .. Esse ponto vai adicionar todos os arquivos do repositório.

git **add** .

Copiar código

**Rodrigo**: Após pressionar "Enter", apareceram alguns alertas, porque existem algumas diferenças entre Windows e Linux, dependendo do sistema operacional que você estiver utilizando. No entanto, são só alertas e não vão afetar nada.

Retornando ao GitHub, o próximo comando que encontramos é o git commit -m e temos que inserir uma mensagem entre aspas. Copiamos até o -m, porque vamos alterar a mensagem. Colocaremos a mensagem projeto inicial entre aspas duplas.

git **commit** -m "projeto inicial"

Copiar código

Estes comandos, add e commit, foram somente copiados e executados, mas nas próximas aulas, vamos explicar detalhadamente o que cada um faz. No momento, o objetivo é apenas conectar os dois repositórios: o local no computador com o remoto no GitHub.

*Author identity unkown*

Após pressionar "Enter", houve um erro ao tentar fazer o *commit*.

**Gabrielle**: Apareceu uma mensagem indicando que a identidade do autor desse *commit* é desconhecida. Precisamos fazer uma configuração para informar quem é a pessoa responsável por esse *commit*.

No terminal, é exibida uma dica indicando quais comandos devemos usar para fazer essa configuração:

git config --**global** user.email "your@example.com"

git config --**global** user.name "Your Name"

Copiar código

**Rodrigo**: No GitHub, criamos uma nova conta e fiz o login. Portanto, o GitHub sabe quem somos, porque estamos logados no navegador. No entanto, instalamos o Git no computador e o Git não sabe quem somos, pois não possui vínculo direto com o GitHub.

Esses comandos são justamente para nos identificar ao Git no computador.

Copiamos apenas comando git config --global user.email, mas só até email. Vamos colar no terminal e incluir o e-mail que configuramos no GitHub. No nosso caso, é o rodrigo.alura87@gmail.com entre aspas duplas.

git config --**global** user.email "rodrigo.alura87@gmail.com"

Copiar código

**Atenção**: Você deve substituir pelo e-mail usado para criar sua conta no GitHub!

Após pressionar "Enter", vamos utilizar a seta para cima para resgatar o último comando digitado. Alteraremos o user.email para user.name, pois além do e-mail, o Git também precisa saber o seu nome.

Entre as aspas, inserimos o Rodrigo Ferreira e apertamos "Enter". Você deverá colocar seu nome e sobrenome.

git config --**global** user.name "Rodrigo Ferreira"

Copiar código

Pronto. Realizamos a configuração de maneira global, devido a esse parâmetro --global. Agora, neste computador, qualquer repositório que criamos, o Git já nos reconhece.

**Gabrielle**: Agora, podemos prosseguir e tentar realizar esse *commit*.

**Rodrigo**: Vamos apertar novamente a seta para cima três vezes para voltar para o comando de *commit* e dar um "Enter". Agora o *commit* foi realizado.

[master (root-commit) 250c665] projeto inicial

10 files changed, 334 insertions (+)

Ele **registrou esses arquivos no nosso repositório local**. Ótimo, vamos voltar para o tutorial no GitHub.

O próximo comando, git branch -M main. Vamos copiar, colar e executar. Não deu nenhum alerta.

git branch -M **main**

Copiar código

E o próximo comando é o principal, pois é o comando que vai fazer o **vínculo do repositório do GitHub com o meu repositório local**, que é esse git remote add.

**Gabrielle**: Sim, após executar esse comando, vamos conseguir que esse repositório que temos localmente em nosso computador esteja conectado com esse repositório do GitHub.

**Rodrigo**: Vamos apenas fazer uma alteração, porque por padrão ele vem com o protocolo HTTPS. Mas acima, onde temos instruções para o *Quick Setup* (configuração rápida), tem um botão chamado "SSH". Se clicamos neste botão, o GitHub troca para o protocolo para o SSH.

Geralmente, o **protocolo SSH** é o mais recomendado para fazer esse ligação com o GitHub.

O comando git remote add origin já foi alterado, e agora a URL não está com http://. Ela está usando o formato SSH. Vamos copiar e colar no terminal integrado:

git remote add origin git**@github**.com:rodrigoalura87/numero-secreto.git

Copiar código

Lembre-se que a URL do GitHub vai variar de acordo com o nome do seu usuário e projeto!

**Gerar chave SSH**

Por fim, temos o comando para enviar esses arquivos, ou seja, esse *commit* que fizemos localmente para o GitHub com o comando git push. O remote add não envia automaticamente, ele só faz a ligação.

git push -u origin **main**

Copiar código

Vamos copiar e colar esse último comando. Com isso, ele vai tentar sincronizar, de fato, com o GitHub. Vamos conferir se vai funcionar.

The authenticity of host 'GitHub.com (20.201.28.151)' can't be established

Deu outro erro! Foi um erro de autenticação. Afinal, instalamos o Git no computador, colocamos o e-mail e nome, mas como é que o GitHub vai saber que somos nós mesmos - e não é outra pessoa que está usando nosso nome e e-mail?

De alguma forma, precisamos fazer essa configuração de segurança para ele saber que neste computador, é, de fato, o Rodrigo com e-mail específico e está vinculado com a conta do Rodrigo no GitHub.

Como copiamos o protocolo SSH, o Git já nos dá a sugestão de **gerar uma chave SSH** no computador porque ainda não temos uma. Ele faz uma pergunta que precisamos responder com "sim" ou "não".

Are you sure you want to continue connecting (yes/no/[fingerprint])?

Em inglês, ele pergunta: quer continuar a conexão? Respondemos com um yes (sim). Feito isso, ele manda uma mensagem dizendo que não reconhece esse computador, portanto, não está autorizado para se conectar no GitHub.

Warning: Permanently added 'github.com' (ED25519) to the list of known hosts

git@github.com: Permission denied (publickey)

Isso é importante para garantir a segurança. Ou seja, vamos ter que fazer essa configuração no GitHub.

**Gabrielle**: Isso mesmo. Vamos ao GitHub finalizar essa configuração.

**Rodrigo**: Essas são configurações iniciais, pois como acabamos de instalar o Git, não tem nada disso configurado. Será apenas uma vez que precisamos fazer todos esses comandos.

De volta ao GitHub, no canto superior direito da página, vamos clicar no ícone do avatar para abrir o menu lateral e selecionar a opção "*Settings*". Assim, entraremos nas configurações da nossa conta e não do repositório.

Na página de configurações da nossa conta, no menu à esquerda, existe a opção "SSH and GPG keys" onde podemos fazer a configuração do SSH.

Em seguida, vamos clicar no botão verde "*New SSH key*" (nova chave SSH). Ao clicar nesse botão, devemos colocar um título. É como se estivéssemos criando uma chave para conectar esse nosso computador com o GitHub. Vamos colocar notebook pessoal, apenas para identificar que chave é essa.

Ele também solicita um tipo, nesse caso, é uma chave de autenticação (*Authentication key*). Além disso, nos orienta a colar a chave. Mas, de onde vamos conseguir essa chave, Gabi?

**Gabrielle**: Devemos gerar essa chave em nosso computador.

**Rodrigo**: Assim, no nosso computador, teremos que acessar essa chave que foi gerada com o SSH.

No terminal integrado, quando pressionamos "yes", ele deve ter criado uma chave que fica salva dentro do computador na pasta do usuário. Podemos tanto fazer isso pelo terminal ou navegar até a pasta, o que é mais fácil.

Portanto, vamos abrir o explorador de arquivos e entrar na pasta do usuário. Nesse casso, a pasta Rodrigo. Agora, vamos procurar uma pasta oculta chamada .ssh. Entrando nela, há um arquivo chamado known\_hosts, que são os hosts conhecidos, mas a chave não foi gerada.

Para gerar a chave, existe um comando que temos que copiar. Inclusive, existe um tutorial no próprio site do GitHub, que vamos disponibilizar no curso como uma atividade. Ele tem um tutorial ensinando [como gerar uma chave SSH](https://docs.github.com/pt/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent) para cada sistema operacional. No nosso caso, é o Windows.

Nessa documentação, temos a explicação do que é e como gerar uma chave SSH. Vamos copiar o comando sugerido ssh-keygen -t ed25519 -C seguido do e-mail da conta do GitHub.

No terminal do Visual Studio Code, vamos colá-lo. Porém, temos que alterar o e-mail de exemplo para o nosso e-mail, que é rodrigo.alura87@gmail.com. Você deve substituir com seu e-mail pessoal.

ssh-keygen -t ed25519 -C "rodrigo.alura87@gmail.com"

Copiar código

Após apertar "Enter", ele informa que vai gerar um par de chaves SSH. E nos pergunta onde queremos salvar esse arquivo.

Enter file in which to save the key (C:\User\Rodrigo/.ssh/id\_ed25519):

Você pode simplesmente pressionar "Enter" para salvá-lo na própria pasta SSH, que é a pasta padrão. Ele pergunta se você quer digitar uma senha.

Enter passphrase (empty for no passphrase):

Se digitamos uma senha, toda vez que eu for sincronizar com o GitHub, ele vai nos pedir essa senha. Por isso, vamos apenas pressionar "Enter" para não ter uma senha. Depois, devemos digitar a mesma senha novamente, damos "Enter" novamente.

Pronto. Agora a chave foi gerada.

Vamos até a pasta "Rodrigo > .ssh". Agora, contém dois arquivo chamamos id\_ed25519, um com a chave privada, outro com a chave pública. Devemos abrir o da chave pública, vamos conferir.

Vamos dar dois cliques e mandar abrir com o bloco de notas. Estava escrito *private key*, era a chave privada.

Vamos abrir o outro arquivo. Podemos clicar com o botão direito e selecionar "Abrir com" e escolher o aplicativo de bloco de notas. Agora, temos o código da chave começando com ssh ed-25519 que precisamos copiar e colar no site do GitHub.

No campo *key*, vamos colá-lo e apertar o botão "*Add SSH Key*" para adicionar a chave SSH. O GitHub já adicionou e listou a chave do notebook pessoal em "SSH keys".

Agora o GitHub sabe que neste notebook tem uma chave e já nos autenticamos, pois geramos essa chave com o nosso e-mail. Assim, o GitHub confia neste computador. Desse modo, conseguiremos sincronizar com a nossa conta no GitHub.

**Sincronizar repositórios**

**Rodrigo**: Vamos voltar para o Visual Studio Code e tentar fazer novamente aquele git push:

git push -u origin **main**

Copiar código

Vamos conferir se agora vai funcionar.

Total 14 (delta 0), reused 0 (delta 0), pack-reused 0

To github.com: rodrigoalura87/numero-secreto.git

* [new branch) main -> main

branch 'main' set up to track 'origin/main'.

Finalmente, o upload dos arquivos e do *commit* para o nosso repositório no GitHub foi bem-sucedido, ao que tudo indica. Será que está sincronizado, Gabi?

**Gabrielle**: Sim, aparentemente funcionou, mas vamos abrir nosso repositório no navegador e atualizar a página para verificar se encontramos nosso projeto lá?

**Rodrigo**: No navegador, precisamos voltar para a página do repositório. Basta clicar no ícone do avatar e acessar "*Your repositories*" (seus repositórios) e escolher o numero-secreto. Em seguida, vamos atualizar a página (atalho "F5") e o projeto apareceu.

O tutorial que estava antes sumiu, porque agora tudo já está sincronizado e os arquivos estão sincronizados no repositório remoto.

Temos o index.html, o app.js e o sistema identificou o *commit* e usuário, o rodrigoalura87. Podemos navegar pelos arquivos e ter acesso ao código-fonte.

Logo, nosso repositório está sincronizado e na barra de endereços temos a URL do repositório. No nosso caso, é github.com/rodrigoalura87/numero-secreto.

Podemos compartilhar esse link com outras pessoas e elas podem acessar o repositório, inclusive você, Gabi. Eu posso te enviar este repositório e fica aí essa pergunta: será que você conseguirá baixar o projeto no seu computador?

**Gabrielle**: O GitHub nos proporciona essa possibilidade de outras pessoas não apenas acessarem nosso código, mas também contribuírem para ele. Vamos explorar isso em seguida.

////////

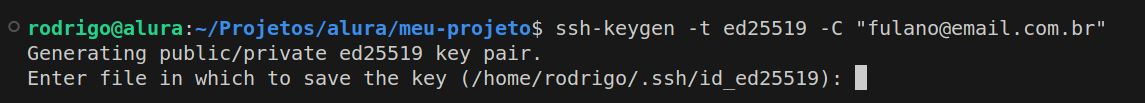
O comando git push deve ser executado para **sincronizar** as mudanças do repositório local com o repositório remoto, ou seja, quando desejamos enviar os novos commits que realizamos em nosso repositório local para o repositório remoto. No entanto, para garantir uma conexão segura, é essencial configurar uma **chave SSH** no computador antes de executar esse comando.

**Chave SSH**

Ao vincular um repositório remoto ao nosso repositório local, via comando git remote add, precisamos utilizar algum protocolo seguro, como HTTPS ou SSH. No caso de se utilizar o protocolo SSH, escolha realizada neste curso, devemos gerar uma chave SSH em nosso computador, além de cadastrá-la em nossa conta do GitHub. Isso é necessário para garantir a autenticação, pois o GitHub checa se quem está realizando o push dos commits tem permissão para realizar tal ação.

**Geração de uma chave SSH**

Antes de executar o comando git push, precisamos gerar uma chave SSH. A geração da chave é feita via terminal, com o comando ssh-keygen -t ed25519 -C "SEU EMAIL AQUI":



Repare, na imagem anterior, que ao executar o comando para gerar uma chave SSH, uma pergunta foi feita e o terminal fica travado esperando nossa resposta:

Generating **public**/**private** ed25519 **key** pair.

Enter file **in** which **to** save the **key** (/home/rodrigo/.ssh/id\_ed25519):

Copiar código

Essa primeira pergunta é para indicarmos o **diretório** em nosso computador no qual a chave será salva, sendo que entre parênteses é indicado o diretório padrão. O recomendado é apenas apertar a tecla enter no teclado para que a chave seja salva no diretório padrão, pois assim o Git consegue encontrar essa chave automaticamente sempre que executarmos o comando git push.

Após apertar a tecla enter, uma nova pergunta será apresentada no terminal:

Enter **passphrase** (enter **for** no passphrase):

Copiar código

Essa segunda pergunta é para indicarmos se desejamos adicionar uma senha à chave SSH que será gerada. Caso você digite uma senha, toda vez que executar o comando git push será necessário digitar tal senha. Ao não digitar nada e apenas apertar a tecla enter, a chave será gerada sem a proteção de uma senha.

Por fim, a terceira e última pergunta é apenas para confirmar a senha anterior:

Enter same passphrase again:

Copiar código

A chave será gerada e a seguinte mensagem será exibida no terminal:

Your identification has been saved **in** /home/rodrigo/.ssh/id\_ed25519

Your **public** **key** has been saved **in** /home/rodrigo/.ssh/id\_ed25519.pub

The **key** fingerprint is:

SHA256:jxAkhGR7NHm/0fcmyPnErZxSKr+ObsH7r4AC/vUNvPY fulano@email.com.br

The **key**'s randomart image is:

+--[ED25519 256]--+

| .oo=.. |

| ..o.+. |

| . .... . |

| . .o . . |

| . ..S+ = o |

| . . ++o+ = + |

| . . o =o.\* = |

| . o .=\*o = |

| . +=\*E=. |

+----[SHA256]-----+

Copiar código

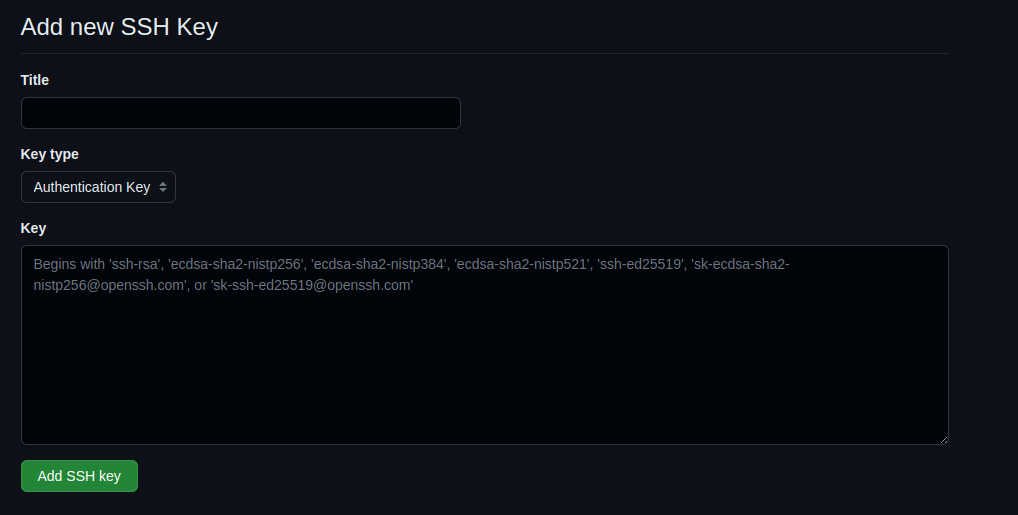
Na primeira linha da mensagem você consegue identificar o diretório no seu computador no qual a chave foi salva. Agora, basta acessar tal diretório para ter acesso à chave SSH.

**Observação:** Nesse diretório serão gerados dois arquivos que representam a chave SSH, sendo um para a chave privada (arquivo id\_ed25519) e o outro para a chave pública (id\_ed25519.pub).

**Cadastrando a chave SSH no GitHub**

Após gerar a chave, precisamos cadastrá-la em nossa conta do GitHub, para que assim o GitHub consiga nos identificar e autenticar ao executar o comando git push de nosso computador.

Acesse a [página de chaves SSH](https://github.com/settings/keys) de sua conta no GitHub e clique no botão **New SSH key** ou **Nova chave SSH** para realizar o cadastro da chave:



Repare que o formulário exibido na imagem anterior contém três campos:

* **Title** ou **Título**: Informe um apelido para sua chave SSH (por exemplo: computador casa)
* **Key type** ou **Tipo de chave**: Escolha o tipo **Authentication Key** ou **Chave de autenticação**
* **Key** ou **Chave**: Nesse campo você deve colar o **conteúdo** do arquivo da sua chave SSH **pública** (arquivo id\_ed25519.pub)

Após realizar esse procedimento, será possível sincronizar o repositório local com o remoto, enviando os novos commits com o comando git push.

////////////////

1. Vimos no vídeo anterior que para fazer o link entre o repositório local, que está em nosso computador, com o repositório remoto, que está no GitHub, devemos utilizar o comando git remote.
2. Esse comando tem algumas variações e parâmetros opcionais que podem ser úteis em certas situações. Confira a seguir exemplos de uso desse comando:
3. 1 - Adicionar um repositório remoto:
4. Quando você deseja estabelecer uma conexão entre seu repositório local e um repositório remoto, como aquele hospedado no GitHub, o comando git remote add é a ferramenta essencial. Essa etapa é crucial para possibilitar a colaboração e o compartilhamento de código com outras pessoas, bem como para fazer backup de seu trabalho em um servidor remoto.
5. A sintaxe básica do comando é a seguinte:

git remote **add** apelido url

Copiar código

1. 'apelido': Este é um nome que você atribui ao repositório remoto. Geralmente, se utiliza nomes descritivos, como "origin" para o repositório principal no GitHub, mas você pode escolher nomes que façam sentido para o seu projeto.
2. 'url': Aqui, você insere a URL do repositório remoto. Esta URL é única para cada repositório remoto e serve como o endereço para acessar e interagir com ele pela internet. Certifique-se de copiar a URL correta do repositório que deseja adicionar.
3. 2 - Listar os repositórios remotos:
4. Para listar os repositórios remotos associados ao seu projeto local, você pode utilizar o comando git remote -v. Isso exibirá uma lista de todos os repositórios remotos vinculados ao seu projeto, juntamente com suas URLs. Veja um exemplo:

git remote -v

Copiar código

1. A saída será algo semelhante a:

origin https://github.com/seu-usuario/seu-projeto.git (fetch)

origin https://github.com/seu-usuario/seu-projeto.git (**push**)

Copiar código

1. Essa lista é útil para verificar se os repositórios remotos estão configurados corretamente. Obs: Vai aparecer duplicado mesmo, pois o Git separa a url de envio de commits (push) da url de baixar commits (fetch).
2. 3 - Remover um repositório remoto:
3. Caso você não precise mais de um repositório remoto específico, pode removê-lo com o comando git remote remove apelido. Substitua 'apelido' pelo nome do repositório remoto que deseja remover. Aqui está um exemplo:

git remote **remove** origin

Copiar código

1. Após a execução deste comando, o repositório remoto chamado "origin" será desvinculado do seu projeto local.
2. 4 - Alterar a URL de um repositório remoto:
3. Às vezes, é necessário atualizar a URL de um repositório remoto, como quando a URL do servidor do GitHub é modificada ou quando você copiou a URL incorreta ao adicionar o repositório remoto. Use o comando git remote set-url apelido nova\_url para realizar essa atualização. Substitua 'apelido' pelo nome do repositório remoto e 'nova\_url' pela nova URL que você deseja associar a ele. Aqui está um exemplo:

git remote set-url origin https://github.com/seu-usuario/seu-repositorio.git

Copiar código

1. Isso atualizará a URL do repositório remoto "origin" para a nova URL especificada.
2. 5 - Alterar o apelido de um repositório remoto:
3. Se você deseja renomear um repositório remoto, use o comando git remote rename apelido novo\_apelido. Substitua 'apelido' pelo nome atual do repositório remoto e 'novo\_apelido' pelo novo nome que você deseja atribuir a ele. Aqui está um exemplo:

git remote rename origin novo-origin

Copiar código

1. Isso renomeará o repositório remoto de "origin" para "novo-origin".
2. Lembre-se de que o comando git remote é fundamental para a gestão de conexões entre seu repositório local e repositórios remotos, permitindo a colaboração eficiente e o controle de versão. Praticar esses comandos em seu ambiente de desenvolvimento ajudará a consolidar seu

///////

1. Nesse curso, aprendemos como realizar a autenticação no Github em seu computador pessoal através de uma chave SSH. Entretanto, essa não é a única forma de se autenticar no Github.
2. Caso você tente conectar seu repositório local com o repositório remoto através da conexão HTTP, será pedido uma senha no momento de realizar o seu login. Porém, desde 2021, por motivos de segurança, o Github não aceita mais que essa autenticação seja feita por meio de sua senha.
3. Caso você tente usar sua senha, acontecerá um erro como o mostrado a seguir:

Logon failed, use ctrl+c to cancel basic credential prompt.

remote: Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.

remote: Please see https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information.

fatal: unable to access ‘<repositório Git>’: The request URL returned error: 403

Copiar código

1. Assim, a outra opção de autenticação para usar o Github em seu terminal é através de um **token para o git**.
2. O artigo [Nova exigência do Git de autenticação por token, o que é e o que devo fazer?](https://www.alura.com.br/artigos/nova-exigencia-do-git-de-autenticacao-por-token-o-que-e-o-que-devo-fazer) te ensinará a realizar as configurações necessárias para criar e utilizar esse token.

//////////////////////////////////////

O Git é uma ferramenta excelente para acompanhar a mudança entre versões de um mesmo projeto e, dentre vários benefícios, nos ajuda a observar de perto o desenvolvimento do seu aprendizado. Tudo isso de uma forma organizada através dos commits.

Além disso, algo que é essencial no universo da tecnologia é apresentar o seu progresso para a comunidade e montar um portfólio dos seus projetos para demonstrar suas habilidades. Dessa forma, o GitHub é essencial para compartilhar e colaborar em projetos de programação de todo o mundo.

Pensando nisso, criamos uma lista de atividades (não obrigatórias) focada em prática para melhorar ainda mais sua experiência de aprendizagem. Bora praticar então?

**Desafios**

1. Crie uma conta no GitHub
2. Crie um repositório para um projeto pessoal.
3. Faça a instalação do Git
4. Crie um repositório localmente para o seu projeto pessoal
5. Adicione alguns arquivos no seu repositório local
6. Faça o commit das alterações
7. Configure a identidade do autor do commit.
8. Crie a branch Main
9. Realize a conexão do seu repositório local com o remoto
10. Envie as alterações no repositório local para o remoto
11. Utilize o comando git status

Caso precise de ajuda, opções de solução das atividades estarão disponíveis na seção “Opinião da pessoa instrutora”.

///////////////////////////

Para acessar um projeto de outro usuário ele precisa estar publico, basta localizar o nome do repositório, e clicar na opção de clonar, que vai abrir o terminal e dai digitamos git clone e colocamos a url que ficou opiada. Assim acessamos o repositório e podemos abrir o projeto pelo vs code abrindo o file/ open folder.

//////////////////////

Para saber quais alterações foram feitas no projeto basta fazer um git status e vai listar todas as alterações que foram realizadas e em quais arquivos

/////////////////////////

**Rodrigo:** Gabi, você já conseguiu baixar o projeto do repositório público da minha conta do GitHub e também importou na IDE, no Visual Studio Code.

Agora, vamos descobrir se você consegue trabalhar no mesmo projeto que desenvolvi e fiz upload para o repositório no GitHub.

O que você acha de aumentarmos a dificuldade do jorgo Número Secreto? Atualmente, ele sorteia um número de 1 a 10. Que tal modificarmos para escolher um número aleatório entre 1 e 100?

**Gabrielle:** Essa é uma alteração interessante, vamos lá!

**Realizando um commit**

index.hmtl

Começamos abrindo o arquivo index.html. Feito isso, na linha 23, no texto Escolha um número entre 1 a 10, mudamos para 10 a 100.

//Código omitido

<p **class**="texto\_\_paragrafo">Escolha um número entre 1 a 100</p>

//Código omitido

Copiar código

app.js

Em seguida, abrimos o arquivo app.js. Na linha 2, mudamos a variável numeroLimite para 100.

**let** numeroSecreto = parseInt(**Math**.**random**() \* 11)

**let** tentativas = 1

**let** chute

//código omitido

Copiar código

**Rodrigo:** Estamos simulando uma mudança no código do projeto, como se tivesse sido solicitada por um usuário ou cliente.

Sendo assim, realizamos as alterações, porém ainda não estão registradas no repositório local do computador.

**Gabrielle:** Isso mesmo, precisamos registrar essas alterações. Para isso, na barra de menu superior do VS Code, clicamos em "Terminal" e depois em "New terminal" para abrí-lo.

**Rodrigo:** Nessa funcionalidade você precisou mexer em dois arquivos, certo? Suponhamos que isso foi feito em uma sexta-feira e quando você voltou ao trabalho na semana seguinte, não lembra onde parou no projeto.

Tem algum comando no GitHub para sabermos o estado atual do código?

**Gabrielle:** Sim, o comando é o git status.

git status

Copiar código

Ao executá-lo é exibido na tela informações sobre as alterações feitas nos arquivos app.js e index.html do projeto e que essas mudanças que precisam ser registradas no commit.

**Rodrigo:** Então, se esquecermos qual arquivo foi modificado ou qual precisa ser adicionado, basta executar esse comando que será exibido o status atual do repositório local.

No nosso caso, temos dois arquivos modificados que estão destacados em vermelho porque ainda não foram adicionados para fazer o commit na sequência.

**Gabrielle:** Repare que o próprio git status nos fornece uma dica para usar o comando git add seguido do nome do arquivo para fazer essa adição.

Anteriormente descobrimos que temos a opção de adicionar arquivo por arquivo, escrevendo o nome de cada um, ou podemos adicionar todas as mudanças de uma vez com o ponto. Portanto, usaremos o git add ..

git **add** .

Copiar código

Feito isso, provavelmente os arquivos foram adicionados Para conferir, rodamos novamente o git status.

git status

Copiar código

Repare que ao fazer isso, notamos no terminal que a ferramenta indica que existem alterações para serem comitadas e quais foram os arquivos modificados. Além disso, a cor mudou para verde, isso significa que os arquivos foram adicionados corretamente.

**Rodrigo:** Muito legal, Gabi. O GitHub tem essa sinalização pelas cores, se os arquivos modificados, mas não foram adicionados ficam vermelhos quando são ficam verdes.

Agora que você já adicionou, podemos fazer o commit, que é o comando responsável por registrar uma modificação no código.

**Gabrielle:** Para isso, passamos o comando git commit. Lembrando que todo commit sempre terá uma mensagem que informará qual alteração foi feita no projeto.

Para adicionar essa mensagem, na mesma linha de comando escrevemos -m "alterando limite para 100" e apertamos "Enter".

git **commit** -m "alterando limite para 100"

Copiar código

Como retorno é informado que o commit foi realizado com sucesso.

**Rodrigo:** Importante ressaltar que, quando fazemos um commit é como se estivéssemos registrando uma versão do nosso sistema. Se estávamos na versão 1, passamos para a versão 2, depois para a versão 3, 4, 5, e assim por diante.

Porém, precisamos saber o que é a versão 1, o que mudou na versão 2, 3 e 4. É justamente a mensagem que proporciona esse indicativo.

Disponibilizamos no Para Saber Mais dicas de **boas práticas** para desenvolver essa mensagem para que fique compreensível e as pessoas entendam as mudanças e o que cada alteração muda no projeto.

Inclusive, Gabi, existe algum comando em que conseguimos identificar quais commits foram realizados no projeto e quem fez?

**Gabrielle:** Tem, sim, Rodrigo! Para isso, podemos utilizar o comando git log.

git log

Copiar código

Ao executá-lo é exibido o histórico dos commits que foram feitos ao longo do projeto.

Se analisarmos o retorno, retonhamos o primeiro commit que o Rodrigo fez para subir o projeto inicial. Identificamos o nome do autor, data seguido da mensagem. Logo acima temos o commit mais atual que é o que acabamos de fazer.

**Rodrigo:** Observe que o git log mostra o log, ou seja, o histórico de commits realizados no projeto. Assim, podemos identificar quem foi que fez cada registro, a mensagem, o significado de cada registro, a data e toda essa sequência de alterações no repositório.

Ótimo! Gabi, você conseguiu fazer uma modificação no projeto e efetuou o commit. Porém, é importante lembrar que o commit está apenas no seu repositório local, ainda não foi enviado para o *GitHub*.

Na sequência descobriremos como enviar esse commit para o repositório no *GitHub*.

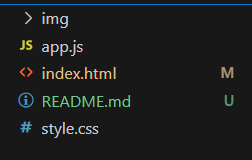
////////////////////////

git add insere mudanças individuais e git add . insere todas as mudanças de uma vez.

Commit é como se eu tivesse colocando versões no meu sistema

Git log mostra todas as informações de todos os commits que foram feitos

////////////////////////////

1. Quando estamos trabalhando em um projeto utilizando o versionamento Git e a IDE VSCode, ao adicionar ou alterar algum arquivo aparece uma sinalização ao lado do nome desses arquivos no VSCode, como podemos ver na imagem abaixo:
2. 
3. Mas o que isso significa?

* **M**: A letra M representa o estado *Modified*, do português modificado. Isso significa que o arquivo já existia no repositório, mas que recebeu alguma modificação que ainda não foi registrada no Git.
* **U**: A letra U representa o estado *Untracked*, do português não rastreado. Isso significa que o arquivo ainda não existia no repositório e que ainda não teve seu registro (commit) feito no Git.

1. Essa sinalização nos ajuda a entender o estado atual dos nossos arquivos do projeto no versionamento Git.

/////////////////////////////

1. Apesar de não existir uma regra universal para a escrita das mensagens de commit, algumas boas práticas podem ser seguidas para garantir que outras pessoas, e até mesmo você no futuro, entendam que alterações foram feitas e por quê.
2. As mensagens dos commits devem ser **simples e objetivas**. A seguir, listamos algumas orientações para isso:

* **Mantenha a mensagem curta e concisa:** A primeira linha da mensagem deve conter, no máximo, 72 caracteres. Caso seja necessário uma descrição adicional, você deve pular uma linha e adicionar os detalhes, como o contexto, da mudança realizada.
* **Uso de verbo no infinitivo:** É comum que a mensagem do commit inicie com um verbo no infinitivo que descreva a alteração feita, como “adicionar”, “corrigir” ou “atualizar”. Em sequência, são adicionados detalhes concisos da mudança. Por exemplo: “Atualizar texto do título da página”.
* **Evite detalhes técnicos**: Não inclua detalhes técnicos complexos na mensagem de commit. Esses detalhes podem ser adicionados nos comentários de código ou na documentação.

1. É importante ter em mente que a mensagem do commit é uma forma de documentação do histórico das mudanças que ocorreram ao longo do código. A pessoa que ler essa mensagem pode não ter conhecimento do contexto original. Assim, garanta que suas mensagens de commit tenham clareza e sejam suficientemente descritivas.

////////////////////////////

1. Um commit deve ser realizado sempre que você **finalizar uma tarefa** específica ou **resolver algum bug**. Isso mantém o histórico de commits claro e rastreável, de modo que seja possível entender o que foi feito em cada commit.
2. Assim, é importante realizar commits frequentemente. Porém, evite realizar commits muito pequenos ou muito grandes, pois isso pode tornar difícil o seu entendimento.
3. Lembre-se de **nunca realizar um commit de um código que você sabe que contém bugs**. O ideal é que o commit contenha somente código funcional.

///////////////////////////

1. O controle de mudanças do Git é feito através dos **commits**. Cada commit armazena o **estado completo** do projeto em um determinado momento por meio de um **snapshot**. Ou seja, cada commit é um registro completo do repositório no momento em que esse commit foi criado.
2. Como cada commit é uma representação completa e consistente do estado do projeto em um determinado ponto no tempo, isso facilita a rastreabilidade e o entendimento de como se deu a evolução do código ao longo do tempo.
3. Todo commit conta com um id único e traz uma referência aos commits anteriores. Assim, através dessa cadeia de commits, o Git registra um **histórico completo** de todos os commits realizados no repositório.
4. Caso queira conhecer melhor sobre esse processo, acesse a [documentação oficial do Git](https://git-scm.com/book/pt-br/v2/Come%C3%A7ando-O-B%C3%A1sico-do-Git).

///////////////////////////////////////

O commit sob as alterações para o projeto local, ele ainda não subiu para o remoto

**Rodrigo:** Gabi, você conseguiu clonar o repositório para o seu computador local, fazer uma alteração no código e o commit.

Porém, essa alteração só existe no seu repositório local. Então, tentaremos fazer o mesmo processo realizado anteriormente, enviaremos o commit para o GitHub utilizando o comando git push.

**Gabrielle:** Mas, nesse caso, precisamos repetir a conexão com o repositório?

**Rodrigo:** Anteriormente, quando seguimos o tutorial, primeiro utilizamos o comando git remote add para adicionar no repositório local um link com o repositório remoto. Só depois fizemos o git push. Porém, você não precisará repetir esses passos, pois rodou o comando clone do repositório.

**Gabrielle:** O comando clone já realiza automaticamente essa conexão entre o repositório remoto e o repositório local.

**Rodrigo:** Inclusive, você pode verificar isso rodando o comando git remote sem passar nenhum parâmetro. Ao fazer isso, será listado os repositórios remotos no seu repositório local.

**Gabrielle:** Então, vamos passar o comando git remote seguido de "Enter".

git remote

Copiar código

Feito isso é exibido o origin.

**Rodrigo:** O origin é o nome que o GitHub atribui quando fazemos o clone, é como se referisse a origem do repositório.

Como você já tem o repositório remoto adicionado, agora pode executar o comando git push para tentar enviar esse commit para o meu repositório no GitHub.

**Gabrielle:** Passamos git push e em sequência precisamos informar para onde enviaremos o commit. Escrevemos origin que é o apelido do repositório remoto, seguido de main que é a branch e apertamos "Enter".

Embora não vamos nos aprofundar nesses termos agora, em breve você encontrará um material explicativo para aprofundar ainda mais seu conhecimento sobre o tema.

git push origin **main**

Copiar código

Ao executar o comando notamos um erro.

**Rodrigo:** É um erro de permissão, certo? Já esperava que isso acontecesse, afinal, se o erro não aparecesse teria algo errado.

O repositório foi criado na minha conta do GitHub com meu e-mail pessoal e você tem outra conta. Embora você tenha conseguido baixar o repositório, criado como público, esse é um material público apenas para leitura.

Isso significa que as pessoas podem acessá-lo, baixar o código, fazer mudanças e commits, porém apenas localmente. Quando alguém tentar fazer um push, o próprio GitHub irá barrar.

Isso porque você é outra pessoa usuária que está tentando fazer um push para um repositório da minha conta, Rodrigo. Isso não é permitido, a não ser que eu dê permissão.

**Adicionando colaboradores ao projeto**

**Rodrigo:** Se quisermos que outras pessoas colaborem nesse projeto, é preciso adicioná-las manualmente no projeto. Para isso, no navegador, logamos no Github e acessamos a página do repositório numero-secreto.

Na barra de menu superior, clicamos no botão "Settings", que se refere a configuração. Em seguida, somos encaminhados para uma nova tela.

No menu lateral esquerdo, clicamos em "Collaborators". Feito isso, notamos que não há nenhum colaborador. Para adicionarmos, no fim da tela, clicamos no botão verde chamado "Add people".

Na nova janela, há um campo referente ao username da pessoa que adicionaremos como colaboradora. Preenchemos com o user da Gabi e depois clicamos no botão "Add".

Ao fazer isso, a ferramenta indica que o pedido de acesso está pendente, pois a Gabi precisa aceitá-lo. Sendo assim, esse processo não é automático.

Então, Gabi, esse é seu momento de compartilhar a tela.

**Gabrielle:** No navegador, acesso meu e-mail do GitHub. Feito isso, notamos o e-mail do GitHub com o convite. Para visualizá-lo, clicamos no botão "View invitation".

**Rodrigo:** Repare que, se voltarmos a minha tela, a Gabi agora aprece como colaboradora do repositório.

Isso é o que você fará no dia a dia de trabalho. Quando estiver em um repositório e quiser que outras pessoas colaborem, será necessário adicioná-las como colaboradoras.

Gabi, agora, se você tentar rodar o comando push é para funcionar. Vamos testar?

**Gabrielle:** Vamos nessa. No terminal do VS Code, digitamos novamente o comando git push origin-main e pressionamos *Enter*.

git push origin-**main**

Copiar código

Feito isso, é exibido uma mensagem que o commit foi enviado para o repositório.

**Rodrigo:** Vamos conferir. No site do GitHub, na barra de menu superior, clicamos em "Code" para voltarmos para a página inicial do repositório.

No arquivo app.js conseguimos notar a mudança de "alterando o limite para 100" e ao lado a foto da Gabi, que fez a alteração.

////////////////////////////

**Rodrigo:** Gabi, você conseguiu clonar o repositório para o seu computador local, fazer uma alteração no código e o commit.

Porém, essa alteração só existe no seu repositório local. Então, tentaremos fazer o mesmo processo realizado anteriormente, enviaremos o commit para o GitHub utilizando o comando git push.

**Gabrielle:** Mas, nesse caso, precisamos repetir a conexão com o repositório?

**Rodrigo:** Anteriormente, quando seguimos o tutorial, primeiro utilizamos o comando git remote add para adicionar no repositório local um link com o repositório remoto. Só depois fizemos o git push. Porém, você não precisará repetir esses passos, pois rodou o comando clone do repositório.

**Gabrielle:** O comando clone já realiza automaticamente essa conexão entre o repositório remoto e o repositório local.

**Rodrigo:** Inclusive, você pode verificar isso rodando o comando git remote sem passar nenhum parâmetro. Ao fazer isso, será listado os repositórios remotos no seu repositório local.

**Gabrielle:** Então, vamos passar o comando git remote seguido de "Enter".

git remote

Copiar código

Feito isso é exibido o origin.

**Rodrigo:** O origin é o nome que o GitHub atribui quando fazemos o clone, é como se referisse a origem do repositório.

Como você já tem o repositório remoto adicionado, agora pode executar o comando git push para tentar enviar esse commit para o meu repositório no GitHub.

**Gabrielle:** Passamos git push e em sequência precisamos informar para onde enviaremos o commit. Escrevemos origin que é o apelido do repositório remoto, seguido de main que é a branch e apertamos "Enter".

Embora não vamos nos aprofundar nesses termos agora, em breve você encontrará um material explicativo para aprofundar ainda mais seu conhecimento sobre o tema.

git push origin **main**

Copiar código

Ao executar o comando notamos um erro.

**Rodrigo:** É um erro de permissão, certo? Já esperava que isso acontecesse, afinal, se o erro não aparecesse teria algo errado.

O repositório foi criado na minha conta do GitHub com meu e-mail pessoal e você tem outra conta. Embora você tenha conseguido baixar o repositório, criado como público, esse é um material público apenas para leitura.

Isso significa que as pessoas podem acessá-lo, baixar o código, fazer mudanças e commits, porém apenas localmente. Quando alguém tentar fazer um push, o próprio GitHub irá barrar.

Isso porque você é outra pessoa usuária que está tentando fazer um push para um repositório da minha conta, Rodrigo. Isso não é permitido, a não ser que eu dê permissão.

**Adicionando colaboradores ao projeto**

**Rodrigo:** Se quisermos que outras pessoas colaborem nesse projeto, é preciso adicioná-las manualmente no projeto. Para isso, no navegador, logamos no Github e acessamos a página do repositório numero-secreto.

Na barra de menu superior, clicamos no botão "Settings", que se refere a configuração. Em seguida, somos encaminhados para uma nova tela.

No menu lateral esquerdo, clicamos em "Collaborators". Feito isso, notamos que não há nenhum colaborador. Para adicionarmos, no fim da tela, clicamos no botão verde chamado "Add people".

Na nova janela, há um campo referente ao username da pessoa que adicionaremos como colaboradora. Preenchemos com o user da Gabi e depois clicamos no botão "Add".

Ao fazer isso, a ferramenta indica que o pedido de acesso está pendente, pois a Gabi precisa aceitá-lo. Sendo assim, esse processo não é automático.

Então, Gabi, esse é seu momento de compartilhar a tela.

**Gabrielle:** No navegador, acesso meu e-mail do GitHub. Feito isso, notamos o e-mail do GitHub com o convite. Para visualizá-lo, clicamos no botão "View invitation".

**Rodrigo:** Repare que, se voltarmos a minha tela, a Gabi agora aprece como colaboradora do repositório.

Isso é o que você fará no dia a dia de trabalho. Quando estiver em um repositório e quiser que outras pessoas colaborem, será necessário adicioná-las como colaboradoras.

Gabi, agora, se você tentar rodar o comando push é para funcionar. Vamos testar?

**Gabrielle:** Vamos nessa. No terminal do VS Code, digitamos novamente o comando git push origin-main e pressionamos *Enter*.

git push origin-**main**

Copiar código

Feito isso, é exibido uma mensagem que o commit foi enviado para o repositório.

**Rodrigo:** Vamos conferir. No site do GitHub, na barra de menu superior, clicamos em "Code" para voltarmos para a página inicial do repositório.

No arquivo app.js conseguimos notar a mudança de "alterando o limite para 100" e ao lado a foto da Gabi, que fez a alteração.

Na lateral superior direita, a ferramenta indica que há dois commits. Se clicarmos, visualizamos o primeiro que é o projeto inicial e o segundo o atualizado.

Se clicarmos no commit encontramos os arquivos modificados e as alterações realizadas em cada um deles. Então, deu certo, Gabi. Você conseguiu colaborar comigo neste projeto!

**Gabrielle:** Ótimo, Rodrigo! Porém, agora que enviei as alterações, imagino que você queira continuar trabalhando no seu projeto. Como podemos trazer essas modificações que estão no GitHub para o seu computador?

**Rodrigo:** Isso é muito importante! O último commit existe no seu repositório local, porque você o fez no seu computador.

Embora você tenha feito o push, que está no GitHub, no meu repositório local ele ainda não existe, pois não é sincronizado automaticamente.

No vídeo seguinte vamos descobrir como baixar os novos commits para o repositório local.

**Até lá!**

Na lateral superior direita, a ferramenta indica que há dois commits. Se clicarmos, visualizamos o primeiro que é o projeto inicial e o segundo o atualizado.

Se clicarmos no commit encontramos os arquivos modificados e as alterações realizadas em cada um deles. Então, deu certo, Gabi. Você conseguiu colaborar comigo neste projeto!

**Gabrielle:** Ótimo, Rodrigo! Porém, agora que enviei as alterações, imagino que você queira continuar trabalhando no seu projeto. Como podemos trazer essas modificações que estão no GitHub para o seu computador?

**Rodrigo:** Isso é muito importante! O último commit existe no seu repositório local, porque você o fez no seu computador.

Embora você tenha feito o push, que está no GitHub, no meu repositório local ele ainda não existe, pois não é sincronizado automaticamente.

No vídeo seguinte vamos descobrir como baixar os novos commits para o repositório local.

**Até lá!**

///////////////////////////////////////

1. Cada commit possui por padrão um autor, que é a pessoa que realizou aquelas alterações no código.
2. Entretanto, quando trabalhamos em equipe pode ser que algum trecho de código seja escrito em dupla ou trio. Assim, como definir a autoria dessas outras pessoas no commit?
3. O Git oferece a possibilidade de adicionar **mais de um autor** a um commit. Para isso, após escrever a mensagem do commit, pulamos duas linhas e usamos a palavra-chave Co-authored-by:, seguido do nome e e-mail associado ao GitHub (entre < >) de cada pessoa colaboradora.
4. Cada coautor deve estar em uma linha diferente, como é mostrado no exemplo a seguir:

$ git commit -m "Adicionar nova funcionalidade.

>

>

Co-authored-by: NOME <nome@email.com>

Co-authored-by: OUTRO-NOME <outro@email.com>"

Copiar código

1. Caso queira entender mais sobre coautoria no GitHub, você pode acessar a [documentação referente ao assunto](https://docs.github.com/pt/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/creating-a-commit-with-multiple-authors).

///////////////////////////////

1. Existem diversos projetos de software com seu código fonte disponível no GitHub e abertos para colaboração de qualquer um que queira contribuir. Esse modelo de desenvolvimento é chamado de **Open Source** ou **Código Aberto**.
2. Caso queira entender mais sobre projetos Open Source, temos aqui na Alura o artigo [Open Source - Uma breve introdução](https://www.alura.com.br/artigos/open-source-uma-breve-introducao), que fala mais sobre o tema.
3. Em um projeto que segue o modelo Open Source, as demandas, como novos recursos e correção de bugs, são descritas e listadas no repositório do GitHub via **issues**. Assim, caso você queira colaborar, é possível escolher uma issue.
4. Você precisará realizar um **fork** do projeto, que é uma cópia do repositório em sua conta. Assim, você poderá escrever o código que soluciona a issue escolhida.
5. Por fim, para enviar sua solução de volta ao repositório fonte, você precisará abrir um **pull request**, que é uma solicitação de pull das suas alterações. Esse pull request passará por um processo de avaliação dos responsáveis pelo projeto, podendo ser aceito ou não.
6. Caso seja aceito, seu código agora fará parte do código fonte desse projeto.
7. É importante sempre se atentar às **regras de contribuição** de cada repositório, que podem variar de acordo com o projeto.
8. Grandes projetos são Open Source e se encontram no GitHub, como a IDE VS Code e o framework React, do JavaScript. Você pode conferir os repositórios desses projetos nos links abaixo:

* [Repositório do VS Code no Github](https://github.com/microsoft/vscode)
* [Repositório do React no Github](https://github.com/facebook/react-native)

.......................//////////////////////////////

**Transcrição**

**Gabrielle:** No vídeo anterior, fizemos alterações no projeto e as enviamos. Porém, ainda não estão no repositório local do Rodrigo. Vamos descobrir como fazer isso.

**Baixando novos commits**

**Rodrigo:** Vamos lá! No GitHub, já temos o seu commit, mas se abrirmos o VS Code, o código continua na versão antiga, na qual o número limite que pode ser sorteado é 10.

Isso significa que o repositório local não atualizou automaticamente o commit feito pela Gabi. Para isso, existe o comando git pull especificamente para isso. Ele funciona como o oposto do push, já que puxa os commits do remoto para o local.

Para isso, além de git pull, na mesma linha de código indicamos qual é o repositório remoto do qual baixaremos esses commits, nesse caso será o origin. Em seguida, passamos a *branch* main, onde ele trará esses commits para o repositório local.

git pull origin **main**

Copiar código

Após dar "Enter", a ferramenta irá sincronizar com o GitHub. Nisso, aparece uma mensagem dizendo que há um commit, porém no remoto há outro, então esse é baixado.

**Gabrielle:** Para verificarmos se isso foi feito, podemo rodar o git log.

`git log`

Copiar código

**Rodrigo:** Assim, temos o commit feito pela Gabi, seguido da data e da mensagem. Se no Explorer abrirmos o arquivo app.js, notamos a mudança no código.

Portanto, o comando git pull tem esse objetivo, baixar os novos commits que outras pessoas colaboradoras do seu repositório enviaram para o GitHub. Com isso, temos um fluxo de trabalho.

**Gabrielle:** Exatamente, Rodrigo. Quando estamos trabalhando com Git, GitHub, utilizamos sempre esses comandos que aprendemos ao longo desta aula. Vamos relembrá-los!

Quando estivermos trabalhando em um projeto e precisarmos realizar mudanças, usaremos o git status para verificar os arquivos modificados.

Adicionaremos essas mudanças com o comando git add, depois, realizaremos um commit com o git commit. Subiremos essas mudanças para o repositório com o git push e eventualmente, conforme outras pessoas forem colaborando com o projeto, traremos essas mudanças novamente para o computador com o git pull.

**Rodrigo:** Esse é o fluxo de trabalho no dia a dia. Cada pessoa faz o clone do repositório local e centralizam no repositório remoto do GitHub, permitindo a colaboração de todos.

Por isso utilizar o GitHub para trabalhar de maneira colaborativa com projetos de *software* é muito interessante.

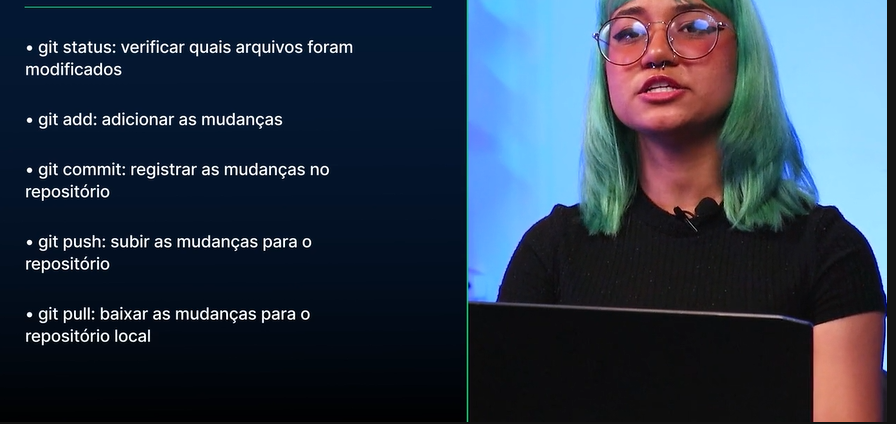
**Gabrielle:** Até o momento, digitamos esses comandos no terminal, mas acredito que há uma forma mais fácil de realizar isso.

**Rodrigo:** É verdade! Nesse caso, estamos usando o VS Code como IDE. Geralmente as IDEs já possuem integração com o Git, pois é o principal sistema de controle de versão.

Então, todo trabalho que fizemos aqui no terminal conseguimos fazer dentro do VS Code de uma maneira visual, tornando esse trabalho muito mais simples.

É isso que estudaremos na aula seguinte!

/////////////////////////////



//////////////////////////

Agora é com você! Baixe uma cópia do repositório remoto no GitHub para o seu computador, realize alguma modificação no projeto, registre essas alterações no Git e as envie novamente para o repositório no GitHub.

Ver opinião do instrutor

**Opinião do instrutor**

Para baixar o repositório em seu computador, você deve realizar um clone do projeto através do comando:

git clone https://github.com/SEU\_USUARIO/SEU\_REPOSITORIO.git

Copiar código

Lembre-se de substituir o SEU\_USUARIO pelo seu próprio usuário do GitHub e o SEU\_REPOSITORIO pelo nome do seu repositório no GitHub.

Caso você já tenha o projeto em seu computador, ao invés de realizar um clone, você pode baixar somente uma atualização dos commits através do comando:

git pull origin **main**

Copiar código

Depois disso, abra o projeto que você acabou de baixar no VS Code. Altere algo no código, como o limite superior de valores de 10 para 100, nos arquivos de *app.js* e *index.html*.

Após salvar as alterações, visualize quais arquivos foram modificados no repositório local com o comando:

git status

Copiar código

Agora, para registrar as alterações feitas, use os comandos a seguir:

git **add** .

git **commit** -m “MENSAGEM”

Copiar código

Lembre-se de substituir o MENSAGEM, por um texto que descreva brevemente qual alteração você realizou.

Para enviar as alterações feitas para o repositório remoto no GitHub, use o comando a seguir:

git push origin **main**

Copiar código

Caso você queira verificar se funcionou, basta abrir o seu repositório no GitHub, atualizar a página e o novo commit deverá aparecer por lá.