course/git-github-compartilhando-colaborando-projetos

1.

05

**Criando um repositório**

**Gabrielle**: Criamos uma conta no GitHub que já está ativa e agora podemos subir o nosso projeto. Entretanto, como podemos **fazer o upload** desse projeto? Como **organizar vários projetos** no GitHub?

**Rodrigo**: Esta é uma questão interessante. Criamos a conta, mas e se tivermos dois, três, cinco, dez ou mesmo cem projetos? É bastante comum desenvolver vários projetos quando estamos aprendendo a programar. Queremos separá-los, porque cada projeto tem seus próprios arquivos específicos, um não tem relação com o outro.

O GitHub tem um conceito conhecido como repositório:

O **repositório** de um projeto é o local onde vamos guardar os códigos de um projeto específico.

Então, a ideia agora é criar um novo repositório e depois fazer o upload do nosso projeto.

Mas antes de fazer essa mudança, vamos fazer uma pequena configuração no GitHub. Vamos alterar o tema do claro para o tema escuro, para melhorar a visualização. Podemos fazer isso, clicando no canto superior direito da tela na foto do seu avatar para expandir o menu.

Em seguida, você vai até a opção de "*Settings*" (configurações). Nessa opção, no menu lateral esquerdo, tem a opção "*Appearance*" (aparência). Ele sincronizou com o sistema operacional, mas vamos mudar essa opção na combobox para "*single theme*" (tema único). E depois selecionar a opção "*dark default*" (escuro padrão).

**Dica**: Você pode escolher opções de alto contraste, clicando em "*Dark high contrast*" ou para daltonismo, clicando em "*Dark Protanopia & Deuteranopia*" ou "*Dark Tritanopia*".

Pronto, alteramos o tema para o escuro. Vamos voltar para a tela principal do GitHub, clicando no ícone do GitHub no canto superior esquerdo da tela.

**Criando um repositório**

**Rodrigo**: Para criar o repositório, podemos clicar no botão verde chamado "*Create Repository*" (criar repositório) na lateral esquerda da página inicial do GitHub. Ou, no ícone de "+" no menu superior, podemos ir em "Create new > New Repository".

Existem várias opções para criar um novo repositório.

Com isso, se abre um novo formulário com a conta logada já identificada. O GitHub pede para digitar o **nome desse repositório**. É como se fosse o nome do projeto e deve ser único.

**Gabrielle**: Da mesma forma que temos o nome de usuário único, também precisamos de um nome único para o nosso repositório.

Esse nome único se refere a um repositório de uma pessoa. Então, poderíamos criar um repositório na sua conta com o mesmo nome que outro repositório em minha conta.

O nome do repositório tem que ser único na sua conta.

**Rodrigo**: Por exemplo, poderíamos colocar projeto. Ele não vai verificar em todos os usuários do GitHub se alguém já escolheu esse nome. Ele vai verificar apenas no nosso usuário se já temos um repositório com esse nome. Se já tiver, ele não permite, caso contrário geraria um conflito de nomes.

Nesse caso, vamos criar com o nome numero-secreto, que é o nome do nosso projeto.

No próximo campo, podemos inserir uma **descrição**. É opcional, caso queira descrever o que é esse projeto. Vamos deixar em branco.

O próximo campo é importante. Devemos colocar se esse repositório será público (*public*) ou privado (*private*).

**Gabrielle**: A visibilidade do nosso repositório é definida nesse passo. Se deixarmos marcado como público, qualquer pessoa que tiver a URL do repositório conseguirá acessar o nosso projeto e visualizar os códigos.

Se deixarmos como privado, será um repositório fechado com informações mais confidenciais.

**Rodrigo**: Essa escolha vai depender de como você deseja compartilhar esse código.

Se for um projeto para compartilhar, montar um portfólio e você quer mostrar o código para outras pessoas, deixe-o como **público**. Agora, se o projeto é pessoal ou da empresa, ou seja, é restrito e outras pessoas não devem acessar o código, deixe-o como **privado**.

Deixaremos o repositório como público no nosso caso.

Existem algumas outras opções, como *README*, *.gitignore*, *license* (licença), mas não modificaremos essas opções agora. Explicaremos o que cada uma delas significa posteriormente no curso.

Para iniciar, inserimos apenas o nome e marcamos como público. Em seguida, clicamos no botão verde "*Create Repository*" (criar repositório) na parte inferior.

O GitHub criará o repositório e nos redirecionará para a página do repositório. Note como a URL do navegador é:

github.com/rodrigoaluro87/numero-secreto

Onde rodrigoaluro87 é o nome do usuário que criamos na conta GitHub e numero-secreto é o nome do repositório.

Agora que já criamos o repositório, agora vamos ter que obter o código do projeto que está no nosso computador e fazer o upload para esse repositório.

**Gabrielle**: Correto. O GitHub até mostra um tipo de tutorial, um passo a passo, sobre como usar esses comandos e fazer o upload do nosso projeto. Veremos isso na sequência.

06

**Para saber mais: repositórios privados**

A **visibilidade dos repositórios no GitHub** serve para controlar quem tem acesso a eles.

Repositórios públicos permitem que qualquer pessoa na internet os acesse, enquanto repositórios privados só podem ser acessados pelo **dono do repositório** e as pessoas que são adicionadas como **colaboradoras do projeto.**

Caso você não saiba o que é um colaborador em um projeto, não se preocupe. Mostraremos como isso funciona no GitHub mais adiante.

Nas contas gratuitas do GitHub existe uma condição para o uso de repositórios privados: a quantidade de pessoas colaboradoras em cada projeto é limitada a três. Para a inclusão de mais pessoas em repositórios privados, é preciso utilizar um plano pago ou manter a visibilidade como pública.

Caso queira se aprofundar mais no assunto, deixamos como recomendação a leitura das documentações a seguir:

* [Sobre repositórios - GitHub Docs](https://docs.github.com/pt/repositories/creating-and-managing-repositories/about-repositories)
* [Definir a visibilidade do repositório - GitHub Docs](https://docs.github.com/pt/repositories/managing-your-repositorys-settings-and-features/managing-repository-settings/setting-repository-visibility)

07

**Para saber mais: GitHub Student Developer Pack**

Caso você seja estudante em uma instituição de ensino, você pode aproveitar os **benefícios de uma conta Pro do GitHub** (conta paga), além de acesso gratuito a diversas ferramentas pagas de desenvolvimento de software, através da **licença para estudantes do GitHub** (a *GitHub Student Developer Pack*).

Para obter essa licença, você precisará do seu e-mail institucional e do comprovante de matrícula. Depois, basta preencher o [formulário disponível no site GitHub Education](https://education.github.com/discount_requests/application) e aguardar a aprovação.

Para mais informações, você pode acessar o site do [GitHub Student Developer Pack](https://education.github.com/pack?ref=producthunt).

**Gabrielle**: Já criamos nossa conta e o repositório. Agora, como faremos essa **conexão do projeto com o repositório do GitHub** que criamos? Existe um tutorial, mas onde e como executamos esses comandos?

**Rodrigo**: Nós já fizemos a parte do GitHub, criamos a conta e o repositório. Entretanto, o projeto está no VS Code. Baixamos o projeto, importamos no VS Code e, portanto, está local no nosso computador.

O que precisamos fazer é criar ligação entre o projeto que está no computador e o repositório do GitHub.

**Diferença entre Git e GitHub**

Esse tutorial que apareceu após criar o repositório, é uma série de comandos que precisamos executar tanto para criar um novo repositório pela linha de comando ou fazer upload de repositório existente pela linha de comando.

São comandos, logo vamos precisar abrir o **prompt de comandos** do Windows.

Porém, já que estamos utilizando o Visual Studio Code, ele tem um **terminal integrado**. No menu superior do VS Code, clicamos na opção "Terminal > Novo terminal" (ou "Ctrl + Shift + '"). No canto inferior, é aberto o prompt de comandos.

Queremos rodar esses comandos por esse terminal integrado ao projeto. Quando abrimos o terminal pelo VS Code, ele já está na pasta do projeto aberto, que no nosso computador se chama C:\Users\Rodrigo\Desktop\numeros-secretos.

Precisamos rodar aqueles comandos na pasta do projeto. Vamos ao GitHub para copiá-los.

Se você ainda não tem um repositório local no seu computador, o GitHub fornece os seguintes comandos:

echo "# numero-secreto" >> README.md

git **init**

git **add** README.md

git commit -m "first commit"

git branch -M main

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

git push -u origin main

Se você já tem um repositório existente local, outros comandos são sugeridos:

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

**git** **branch** **-M** **main**

**git** **push** **-u** **origin** **main**

O comando que começa com echo seria apenas se você quisesse criar um arquivo de exemplo do zero. Não é o nosso caso, já temos um projeto com arquivos. Então, vamos começar a partir do segundo passo.

Precisamos rodar esse comando git init. Vamos copiar esse comando com "Ctrl + C" e colar com "Ctrl + V" no terminal integrado. Vamos pressionar "Enter" e verificar o que acontece.

git **init**

git: O termo 'git' não é reconhecido como nome de cmdlet, função, arquivo de script ou programa operável. Verifique a grafia do nome ou, se um caminho tiver sido incluído, veja se o caminho está correto e tente novamente.

Gabi, não deu certo, deu um erro. Mas, o que aconteceu? Por que não funcionou?

**Gabrielle**: Esse erro está acontecendo porque precisamos instalar o **Git** no nosso computador.

**Rodrigo**: Exatamente, o Git é uma ferramenta que nos permite trabalhar com a parte de repositório e histórico. Mas, qual a diferença entre Git e GitHub?

**Gabrielle**: São ferramentas diferentes.

O **GitHub** é a plataforma onde vamos hospedar nosso código, enquanto o **Git** é a ferramenta onde conseguimos fazer o controle de versões do nosso código, registrar quais as mudanças foram feitas ao longo do tempo.

**Rodrigo**: São duas ferramentas separadas. O GitHub já conhecemos, e a partir de agora, precisaremos ter o Git instalado no computador. Ainda não temos essa ferramenta instalada, por isso que apareceu essa mensagem de erro.

**Instalando o Git**

**Rodrigo**: Portanto, vamos baixar e instalar o Git no computador. No navegador, vamos abrir uma nova aba e pesquisar "git download" no mecanismo de busca. O primeiro link é a [página de download do site do Git](https://git-scm.com/downloads).

Na página de downloads, você deve escolher o seu sistema operacional. No nosso caso, vamos clicar na opção do Windows. Depois, devemos escolher o instalador para 32 ou 64 bits - no geral é 64 bits.

São oferecidas duas opções: o instalador *standalone* (sozinho) e *portable* (portátil). O portátil não instala no computador, e, sim, cria um executável que você executa no computador.

Vamos baixar a primeira opção *standalone* para Windows 64 bits, pois queremos instalar de fato no computador.

Em seguida, abre-se a tela de downloads. Dependendo da sua conexão, pode demorar um pouco. Ele vai baixar um arquivo, provavelmente .exe ou .msi, para fazer a instalação no Windows. No nosso caso, foi um .exe.

Após terminar, você vai até a pasta de downloads e clica duas vezes para abrir o arquivo .exe. Não se preocupe, não haverá vírus desde que você baixe do site oficial do Git.

Vamos clicar em "sim" para permitir que o aplicativo faça alterações no dispositivo. E então vamos para a instalação.

A instalação é bem direta. Basta clicar para avançar em "*Next*" até concluir a instalação com "*Finish*". São várias opções, mas não precisa se preocupar, manteremos todas as opções padrão.

Pode surgir um alerta avisando que já existe essa pasta e perguntado se desejamos sobrescrevê-la. No nosso caso, já tínhamos uma pasta chamada Git nos arquivos de programa, por isso, vamos permitir que ele a sobrescreva.

Agora o Git será instalado no sistema operacional, e, quando ele terminar, voltaremos ao Visual Studio Code, e aquele comando deveria funcionar, certo?

Assim que o Git finaliza a instalação, ele pergunta se queremos abrir um arquivo de notas de lançamento ou abrir o Git Bash, vamos desmarcar ambas opções. Apenas clicaremos em "*Finish*" para finalizar.

**Configurando variáveis de ambiente**

**Rodrigo**: Vamos voltar para o terminal integrado do VS Code. Vamos rodar novamente o comando git init?

git **init**

git: O termo 'git' não é reconhecido como nome de cmdlet, função, arquivo de script ou programa operável. Verifique a grafia do nome ou, se um caminho tiver sido incluído, veja se o caminho está correto e tente novamente.

Mas, ele ainda não reconheceu. O que aconteceu? Como já tínhamos aberto esse terminal, ele não detectou que o Git foi instalado no computador. É preciso fechar essa janela do terminal, clicando no botão "*Close panel*" (fechar painel) no canto direito,

Novamente, vamos lá no menu superior em "Terminal > Novo terminal". O VS Code abre um novo prompt e agora deve pegar as novas instalações. Então, vamos rodar o git init.

E aí não funcionou novamente, Gabi. Isso acontece por causa do Windows. O Windows tem **variáveis de ambiente** que devemos configurar.

No Windows, vamos clicar no botão iniciar e buscar por "variáveis". Vamos escolher a opção "editar as variáveis de ambiente do sistema".

Na janela que se abre, vamos clicar no botão "Variáveis de Ambiente" e outra janela se abrirá com as variáveis do sistema operacional. Na parte de baixo, temos que procurar uma variável chamada Path.

Após selecioná-la, vamos clicar no botão "Editar" para abrir uma nova janela de edição. Essa variável Path tem o caminho dos programas que estão instalados no computador. Até existe um caminho para o Git definido como C:\Programs Files\Git\cmd.

Contudo, será que esse diretório está correto? Precisamos verificar.

Por isso, vamos abrir o explorador de arquivos e fazer o caminho C:\Programs Files\Git. Existe uma pasta cmd, que contém o aplicativo do Git. Na verdade, ele já adicionou a variável de ambiente, só que ainda não a reconheceu por algum motivo.

Talvez seja preciso fechar e abrir novamente o Visual Studio Code para ele detectar essa mudança.

Agora que verificamos que as variáveis estão configuradas, vamos fechar todas as janelas e fechar o Visual Studio Code.

**Gabrielle**: Caso no seu computador essa variável de ambiente não esteja configurada, você vai editar a variável de ambiente Path e adicionar o caminho para onde o seu Git está instalado.

**Rodrigo**: Perfeito. Vamos abrir o Visual Studio Code e também o terminal integrado. Vamos novamente rodar git init e dar "Enter".

Initialized empty Git repository in C:/Users/Rodrigo/Desktop/numero-secreto/.git/

Agora deu certo, pois ele mostrou a mensagem: "Repositório vazio do Git foi inicializado".

**Dica:** Quando for instalar o Git, verifique nas variáveis de ambiente se o caminho foi adicionado corretamente. Às vezes, não é adicionado. Nesse caso, você terá de colocar o caminho daquela pasta cmd do Git no seu computador.

E não basta apenas fechar o terminal integrado no Visual Studio Code. É necessário reiniciar o Visual Studio Code para detectar o Git.

Agora, Gabi, rodamos o primeiro comando. Na sequência, entenderemos quais são aqueles outros comandos, o que eles fazem.

09

**Para saber mais: git init**

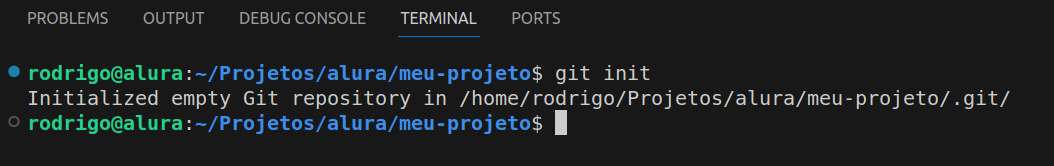
O comando **git init** deve ser utilizado para converter um diretório existente no computador, que geralmente é o diretório de algum projeto, em um repositório Git. No entanto, seu uso requer atenção e cuidado para evitar problemas indesejados. Vamos entender melhor o que o comando git init faz.

**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/

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.

11

**Sincronizando repositórios**

**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** .

**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"

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"

**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"

**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"

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**

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

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**

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"

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**

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.

12

**Para saber mais: Chave SSH**

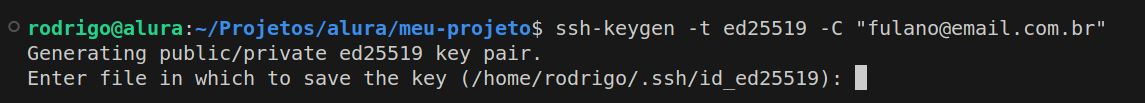
O comando git push deve ser executado para **sincronizar** as mudanças do repositório local com o repositório remoto, ou seja, quando desejamos enviar os novos commits que realizamos em nosso repositório local para o repositório remoto. No entanto, para garantir uma conexão segura, é essencial configurar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

ssa 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):

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:

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]-----+

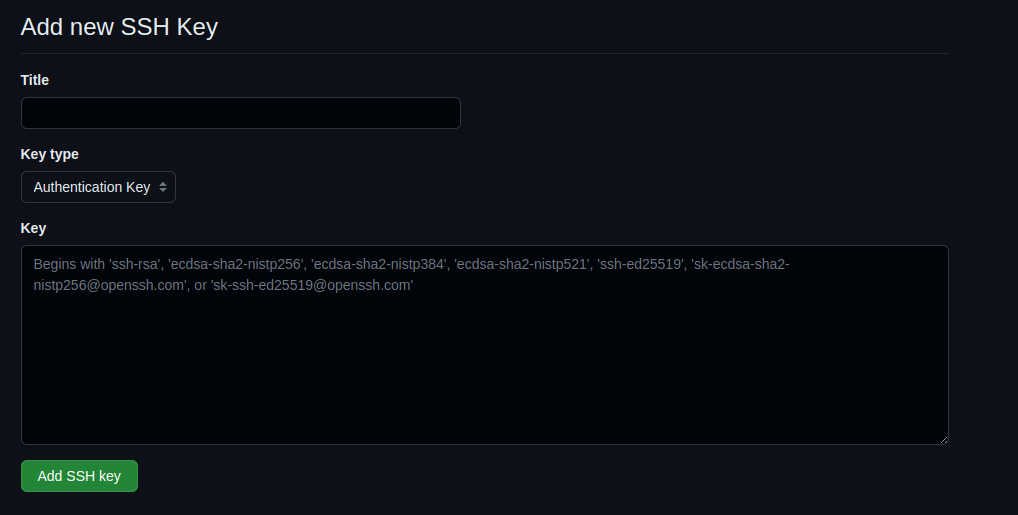
Na primeira linha da mensagem você consegue identificar o diretório no seu computador no qual a chave foi salva. Agora, basta acessar tal diretório para ter acesso à chave SSH.

**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.

13

**Para saber mais: O comando git remote**

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.

Esse comando tem algumas variações e parâmetros opcionais que podem ser úteis em certas situações. Confira a seguir exemplos de uso desse comando:

1 - Adicionar um repositório remoto:

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.

A sintaxe básica do comando é a seguinte:

git remote **add** apelido url

'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.

'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.

2 - Listar os repositórios remotos:

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

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**)

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).

3 - Remover um repositório remoto:

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

Após a execução deste comando, o repositório remoto chamado "origin" será desvinculado do seu projeto local.

4 - Alterar a URL de um repositório remoto:

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

Isso atualizará a URL do repositório remoto "origin" para a nova URL especificada.

5 - Alterar o apelido de um repositório remoto:

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

Isso renomeará o repositório remoto de "origin" para "novo-origin".

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 entendimento.

16

**Para saber mais: autenticação por token no Github**

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.

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.

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

Assim, a outra opção de autenticação para usar o Github em seu terminal é através de um **token para o git**.

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.

17

**Desafio: hora da prática.**

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

**Opinião do instrutor**

1. Crie uma conta no GitHub Você pode começar seguindo os passos mostrados no vídeo, preenchendo o formulário de cadastro, verificando sua conta e explorando a página inicial do GitHub.
2. Criar um novo repositório no GitHub e fazer o upload de um projeto local para esse repositório. Você pode seguir passos mencionados na aula, como acessar as configurações do GitHub, criar um novo repositório com um nome único e escolher se ele será público ou privado. Em seguida, você pode adicionar uma descrição, um README, um .gitignore e uma licença ao repositório.
3. Instalação do Git: Caso você ainda não tenha realizado a instalação, siga os passos na atividade [Faça como eu fiz: instalação do Git](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/144999)
4. Para criar um repositório local você, digite o seguinte comando no terminal: git init
5. Para adicionar os arquivos no repositório local, digite no terminal o comando: git add .
6. Faça um commit com as modificações, digite no terminal o comando: git commit -m "mensagem de commit"
7. Para configurar a identidade do autor do commit, digite no terminal o comando:
8. git -config --**global** user.email "seuemailaqui@example.com"
9. git config --**global** user.name "seu nome aqui"
10. Para criar a branch Main, digite no terminal o comando: git branch -M main

O comando git branch -m é usado para criar uma nova ramificação no repositório Git atual. Neste caso, criamos a branch padrão main, que representa a versão principal do código.

1. Para realizar a conexão do seu repositório local com o remoto via SSH, digite no terminal: git remote add origin https://github.com/seunomedeusuario/seu-repositorio.git

Caso seja necessário, realize a configuração do protocolo SSH através da geração de chave, você pode acompanhar os passos em vídeo na atividade [Sincronizando repositórios](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/139310)

1. Para subir as alterações no repositório local para o remoto, digite o seguinte comando no terminal: git push -u origin main
2. Digite no terminal o comando git status e observe a saída.

O comando git status é uma ferramenta essencial para gerenciar alterações no controle de versão Git. Ele fornece uma visão geral do estado atual do repositório, indicando quais arquivos foram modificados, adicionados ou excluídos desde o último commit. Essa informação é crucial para compreender o progresso do desenvolvimento e tomar decisões de gerenciamento de alterações.

A saída do comando git status geralmente contém três seções principais:

**Modificados:** Lista os arquivos que foram modificados desde o último commit, mas ainda não foram adicionados à área de preparação (Stage).

**Adicionados:** Indica os arquivos que foram adicionados à área de preparação, mas ainda não foram confirmados no histórico de commits.

**Modificados, adicionados ou excluídos:** Exibe os arquivos que não foram rastreados pelo Git, ou seja, que não foram adicionados ao índice de modificações (Staging Area).

Além disso, o comando git status pode fornecer informações adicionais sobre as ramificações atuais, como a ramificação atual e as ramificações que estão à frente ou atrás da atual.

O comando git status é uma ferramenta indispensável para qualquer desenvolvedor que utiliza Git. Ele permite monitorar as alterações no repositório, identificar o estado dos arquivos e tomar decisões de gerenciamento de commits de forma eficiente.

18

**O que aprendemos?**

**Nessa aula, você aprendeu como:**

* Criar sua conta no GitHub para poder compartilhar seus projetos de software e/ou colaborar em projetos de outras pessoas;
* Baixar e instalar o Git em seu computador;
* Configurar sua conta do Git em seu computador, com os comandos git config --global user.name e git config --global user.email;
* Criar um repositório remoto no GitHub;
* Criar e conectar seu repositório local com o repositório remoto por meio dos comandos git init, git add, git commit, git remote add e git push.

2.

02

**Realizando um commit**

**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

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

**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

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** .

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

git status

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"

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

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.

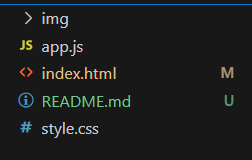
Ótimo! Gabi, você conseguiu fazer uma modificação no projeto e efetuou o commit. Porém, é importante lembrar que o commit está apenas no seu repositório local, ainda não foi enviado para o *GitHub*.

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

03

**Para saber mais: sinalizações em arquivos do VSCode**

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:



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.

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

04

**Para saber mais: mensagens de commits**

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.

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.

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.

06

**Para saber mais: como o Git controla as mudanças?**

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.

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.

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.

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).

09

**Para saber mais: adicionando colaboradores no commit**

Cada commit possui por padrão um autor, que é a pessoa que realizou aquelas alterações no código.

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?

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.

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>"

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).

10

**Para saber mais: outras formas de colaborar**

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**.

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.

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.

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.

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.

Caso seja aceito, seu código agora fará parte do código fonte desse projeto.

É importante sempre se atentar às **regras de contribuição** de cada repositório, que podem variar de acordo com o projeto.

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)

11

**Baixando novos commits**

**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**

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`

**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!

14

**Desafio: hora da prática**

Manipular diferentes versões de um mesmo projeto pode se tornar desafiador. No entanto, o uso eficiente das ferramentas de versionamento elevam a qualidade do código e tornam o fluxo de trabalho muito mais fácil para toda a equipe. Para isso se tornar uma realidade dentro da rotina da pessoa desenvolvedora, é preciso seguir com a prática constante.

Pensando nisso, desenvolvemos uma lista com exercícios não obrigatórios que abrangem desde a criação de um novo repositório remoto até a sincronização com alterações feitas por outros colaboradores. Tudo para que você desenvolva uma compreensão sólida do fluxo de trabalho do Git enquanto explora comandos essenciais.

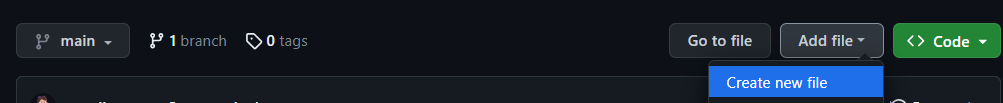
**Desafios:**

1. Crie um novo repositório remoto no Github e insira um arquivo.
2. Faça um clone do seu repositório remoto para o local.
3. Faça uma nova modificação no repositório remoto.
4. Atualize seu repositório local a partir do Remoto.
5. Utilize o comando git remote -v no terminal.
6. Confira as mudanças nos arquivos.

**Opinião do instrutor**

**Desafio 1: Criar um novo repositório remoto no GitHub e inserir um arquivo.**

* Faça login na sua conta do GitHub.
* Clique no sinal de "+" no canto superior direito e escolha "New repository".
* Siga as instruções para criar um novo repositório, dando um nome e uma descrição, se desejar.
* No GitHub, vá para o repositório recém-criado: Clique em "Add file" e escolha a opção que preferir para adicionar um novo arquivo (pode ser um README.md para simplificar).



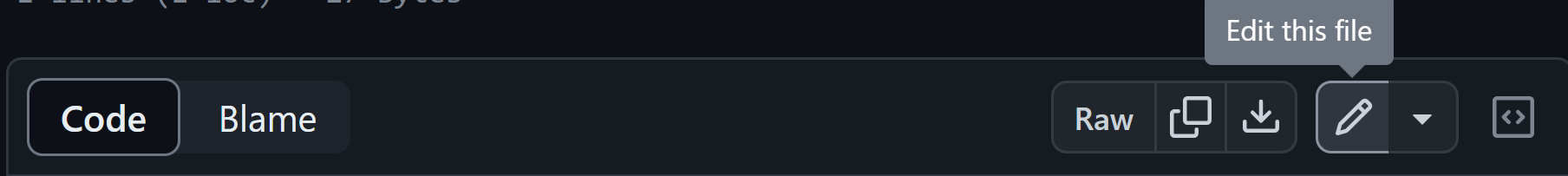
**Desafio 2: Faça um clone do seu repositório remoto para o local.**

* Abra o terminal no seu computador.
* Navegue até o diretório onde deseja clonar o repositório.
* Execute o comando git clone <url-do-seu-repositorio>.

**Desafio 3: Faça uma nova modificação diretamente no GitHub.**

* Acesse o seu repositório no GitHub.
* Abra um arquivo existente ou crie um novo.
* Faça as alterações desejadas diretamente no GitHub e salve.

*Dica: Você pode clicar no arquivo existente e selecionar a opção “Edit this file” (tradução livre para o português: Edite este arquivo)*



**Desafio 4: Atualize seu repositório local a partir do Remoto.**

* Navegue até o diretório do seu repositório local usando o terminal.
* Execute o comando git pull origin main para trazer as últimas alterações do repositório remoto para o seu local.

**Desafio 5: Utilize o comando git remote -v no terminal.**

* Liste repositórios remotos:
* No terminal, navegue até o diretório do seu repositório local.
* Execute git remote -v para listar as entradas remotas configuradas e suas URLs.

O comando é usado para listar todas as entradas remotas configuradas no repositório Git atual. Cada entrada remota é representada por um nome e uma URL. O nome da entrada remota é usado para referenciar outros comandos git, como o git fetch e git push.

Por exemplo, se o seu repositório Git tiver uma entrada remota chamada origin que aponta para o URL https://github.com/meu-usuário/meu-repositório.git, o comando git remote -v exibirá o seguinte:

origin https://github.com/meu-usuário/meu-repositório.git (fetch)

origin https://github.com/meu-usuário/meu-repositório.git (push)

**Desafio 6: Confira as mudanças nos arquivos.**

* No diretório do seu repositório local, digite no terminal o comando git status para ver arquivos modificados

3.

04

**Resolvendo conflitos com Visual Studio Code Merge Editor**

Quando você está trabalhando em um projeto grande, principalmente se muitas pessoas estão envolvidas, é comum aparecerem os chamados "conflitos". Mas o que são esses conflitos?

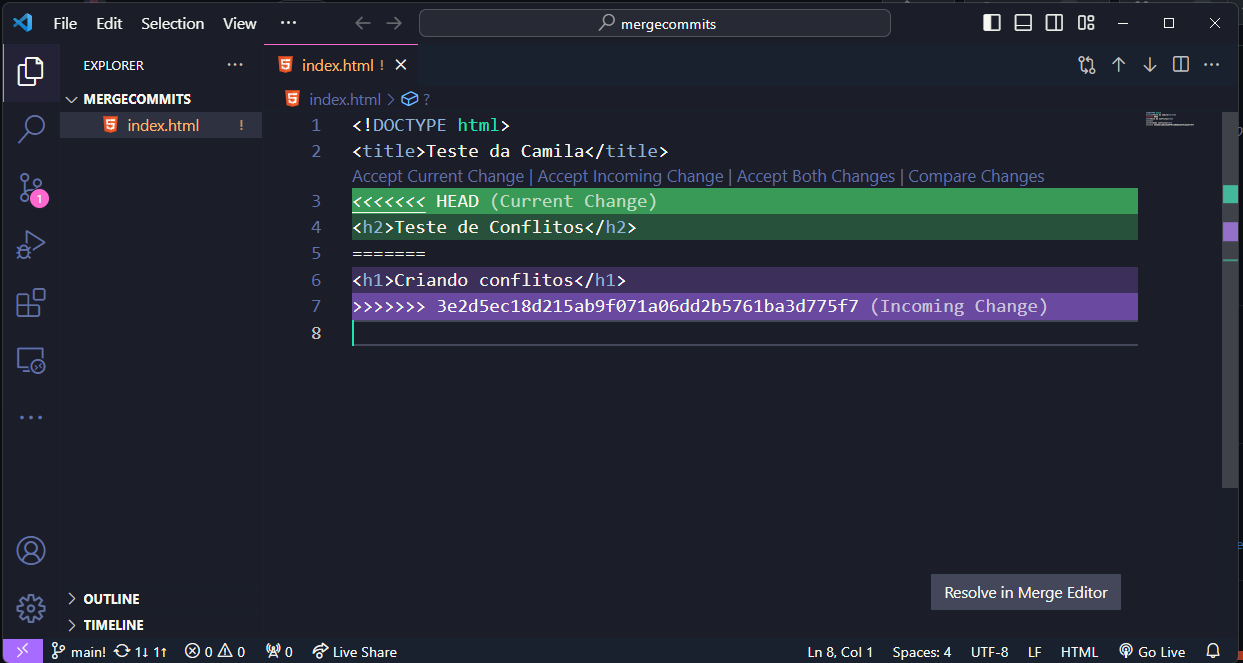
Imagine que o projeto é um grande quebra-cabeça e cada pessoa está trabalhando em uma parte dele. Se duas pessoas tentarem encaixar peças diferentes no mesmo lugar, surge um conflito. No mundo do desenvolvimento de software, isso ocorre quando duas pessoas editam o mesmo pedaço de código de formas diferentes.

Há várias formas de resolver conflitos, podemos utilizar a linha de comando ou o próprio Visual Studio Code. O editor de código fornece mais de uma forma para resolver conflitos de mesclagem entre o código local (o que está na sua máquina) e o remoto(o que está no github, por exemplo).

Uma possibilidade é utilizar a ferramenta **“Merge Editor”.** Vamos conferir seu funcionamento?

**Resolvendo conflitos no Merge Editor**

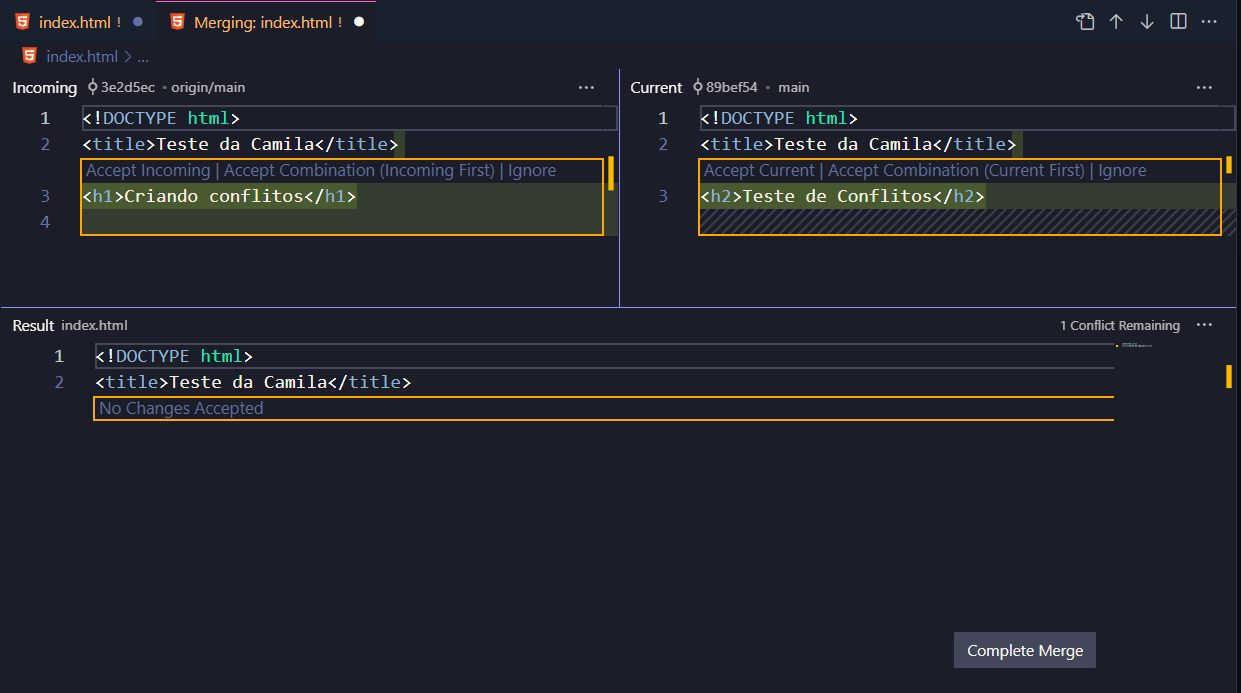
No exemplo a seguir, nós temos duas versões de um código na branch main: uma no repositório do github e outra modificação diferente no ambiente local. Ao realizarmos o git -push para a branch main, ocorreu um **conflito** e precisamos resolvê-lo para que a atualização suba para o repositório no github corretamente, como a imagem nos apresenta:



Para solucionarmos o problema, clicamos na opção “Resolve in Merge Editor”, como no print abaixo:



Após o clique, somos redirecionados para outra aba que apresenta as modificações no arquivo, vamos entender o que cada opção significa:



A tela do Visual Studio Code está dividida em três partes:

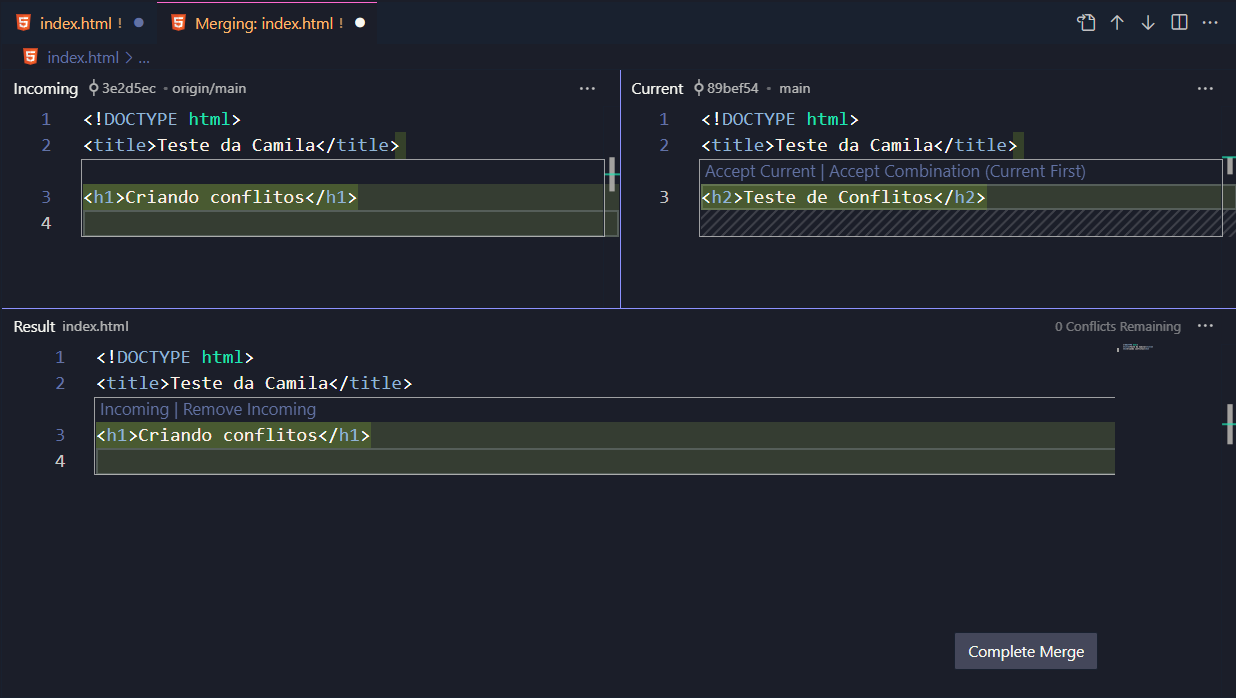
* **Incoming (remoto)**: modificações que chegam do repositório remoto.
* **Current (local)**: modificações locais.
* **Result (resultado)**: resultado do merge, ou seja, a resolução dos conflitos de mesclagem. É o estado atual do arquivo.

Os quadrados na cor amarela em volta do código no campo “Incoming” e “Current” são **marcadores de conflito**: exibem o conteúdo que apresenta conflito no arquivo.

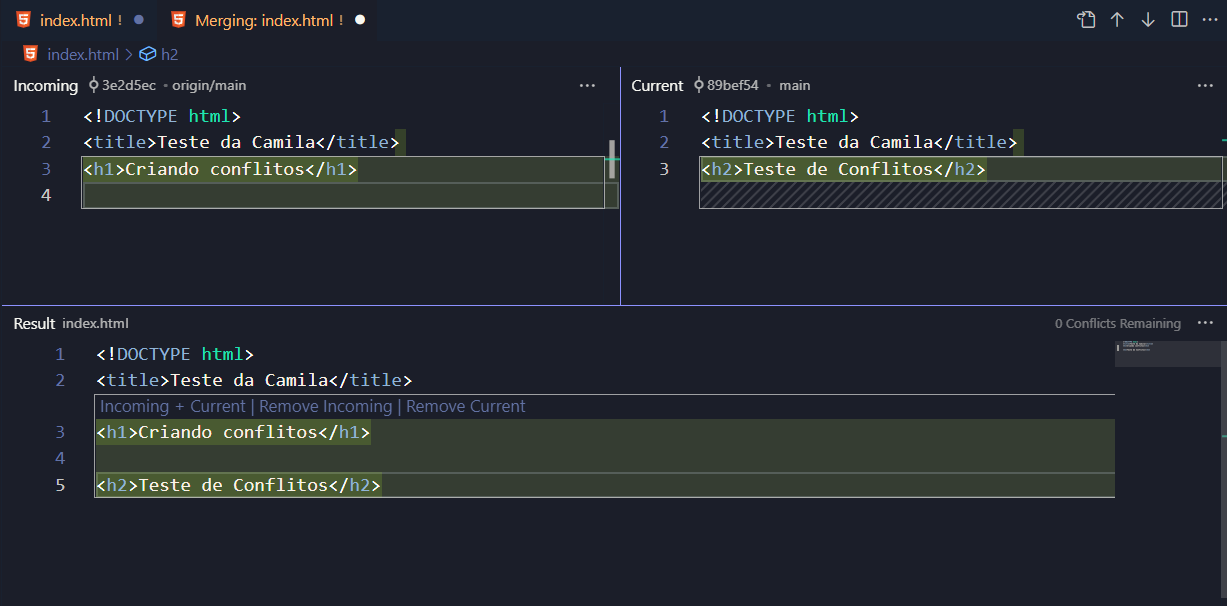
**Campo Incoming**

**->** No **campo “Incoming”,** acima da linha de código dos marcadores de conflito no campo há outras opções que resultam na alteração do código atual:

* **Accept Incoming**: aceita modificações oriundas do **remoto**



* **Accept Combination Incoming First**: realiza a combinação com as linhas do código do repositório remoto no topo.

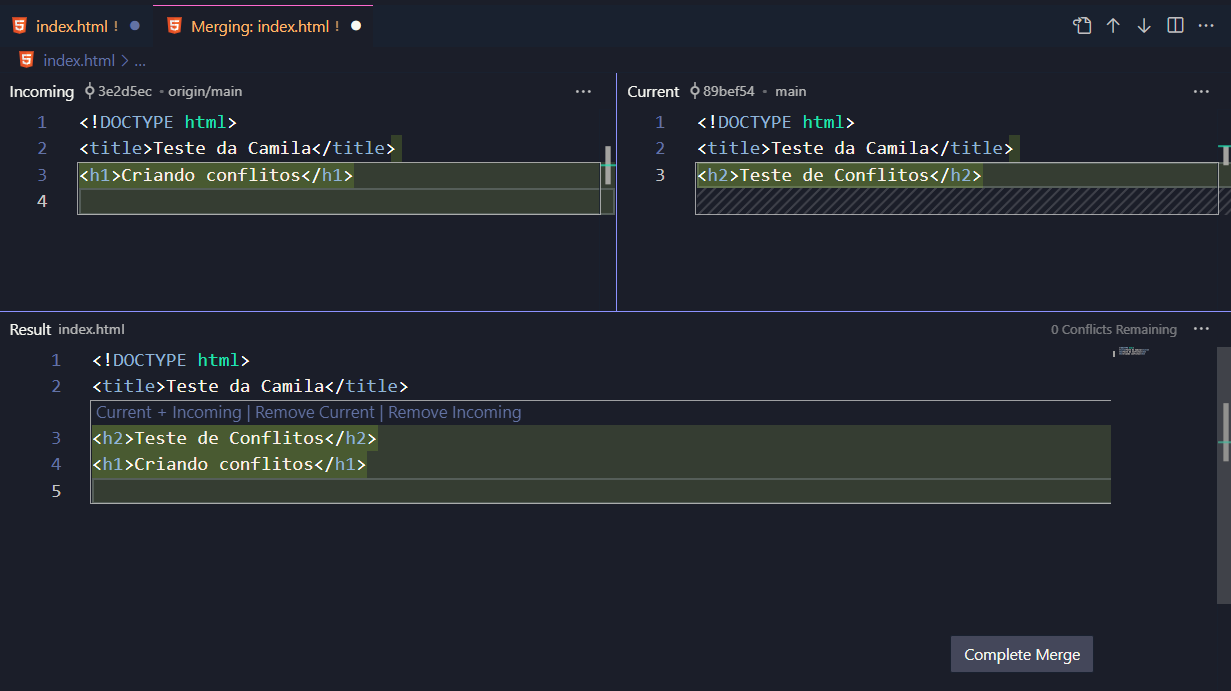


* **Ignore:** ignora as modificações.

**Campo Current**

**->** O campo **“Current”** trabalha com as modificações locais do documento.

* **Accept Current**: Aceita a modificação local no resultado do documento
* **Accept combination Current First**: Aceita a combinação local + remoto. Nos resultados a linha de código com a tag <h2> fica antes de <h1>, comprovando que o código local é inserido primeiro que o remoto.



* **Ignore**: ignora as modificações no resultado no final.

**Após selecionarmos a opção com o resultado desejado, devemos:**

1. Salvar o arquivo
2. Clicar no botão “complete Merge”
3. Realizar o commit das modificações
4. Sincronizar as modificações para realizar o push.

**Para saber mais:**

* [Como o Visual Studio facilita o controle de versão com o Git](https://learn.microsoft.com/pt-br/visualstudio/version-control/git-with-visual-studio?view=vs-2022)
* [Como resolver conflitos de mesclagem no Visual Studio](https://learn.microsoft.com/pt-br/visualstudio/version-control/git-resolve-conflicts?view=vs-2022)
* [Um guia muito útil para mesclar conflitos - em Inglês](https://www.youtube.com/watch?v=HosPml1qkrg)

07

**Desafios: hora da prática**

Conflitos em diferentes versões no código é algo comum no cotidiano da pessoa desenvolvedora e é essencial saber como resolver quando esses conflitos aparecem. A ideia do próximo desafio é gerar e resolver um conflito entre um repositório remoto e local, vamos praticar com os desafios não obrigatórios?

**Desafios**

* Crie um novo repositório local
* Adicione o repositório remoto criado nos exercícios anteriores ao seu repositório local.
* Faça uma alteração no repositório local e envie para o remoto.
* Resolva os conflitos manualmente, escolhendo quais alterações serão mantidas
* Realize um commit para registrar a resolução do conflito.
* Verifique quais arquivos foram adicionados
* Sincronize o repositório local com o repositório remoto no GitHub.

**Opinião do instrutor**

**Desafio 1: Criar um Novo Repositório Local**

* Abra o terminal.
* Navegue até o diretório onde deseja criar o novo repositório.
* Execute git init para inicializar um novo repositório.

**Desafio 2: Adicionar o Repositório Remoto Criado Anteriormente**

* Execute git remote add <nome-remoto> <url-do-repositorio-remoto> para adicionar o repositório remoto ao seu local.

**Desafio 3: Faça uma Alteração no Repositório Local e Envie para o Remoto**

* Abra um arquivo existente ou crie um novo no seu repositório local.
* Faça as alterações desejadas.
* Execute git add . para adicionar as alterações.
* Em seguida, execute git commit -m "Alteração no repositório local" para realizar o commit.
* Utilize git push <nome-remoto> main para enviar as alterações para o repositório remoto.

**Desafio 4: Resolvendo Conflitos Manualmente**

* No GitHub, faça uma alteração no mesmo arquivo que foi modificado localmente.
* Execute git pull <nome-remoto> main no seu terminal para trazer as alterações remotas.
* O Git indicará um conflito. Abra o arquivo afetado e resolva manualmente, escolhendo quais alterações manterá.

Sugestão: você pode seguir os passos demonstrados no vídeo [Resolvendo conflitos](https://cursos.alura.com.br/course/git-github-compartilhando-colaborando-projetos/task/139317)

**Desafio 5: Realize um Commit para Registrar a Resolução do Conflito**

* Após resolver manualmente, execute git add . para adicionar as alterações ao *staged area*.
* Em seguida, execute git commit -m "Resolver conflito manualmente de ‘descrição do conflito’" para registrar a resolução do conflito.

**Desafio 6: Verifique Quais Arquivos Foram Adicionados**

* Execute git status para verificar se há alterações pendentes.
* Se quiser ver as diferenças, utilize git diff.
* Conheça mais sobre o comando git diff na [documentação git diff](https://git-scm.com/docs/git-diff/pt_BR)

**\*\*Desafio 7: Sincronize o Repositório Local com o Repositório Remoto no GitHub\*\***

* Execute git push <nome-remoto> main para enviar as alterações, incluindo a resolução de conflitos, para o repositório remoto no GitHub.
* Verifique o histórico de commits no GitHub e confirme que o conflito foi resolvido corretamente.

Parabéns! Você concluiu o desafio de trabalhar com conflitos no Git, resolvendo manualmente e sincronizando seu repositório local com o remoto. Este processo é crucial ao colaborar em projetos com outros desenvolvedores.

4.

01

**Desfazendo um commit**

**Rodrigo:** Bom, Gabi, já aprendemos vários recursos do GitHub e do Git. Contudo, até agora, focamos mais na parte de **colaboração**.

Criei um repositório no meu usuário do Git, você fez o clone, te adicionei como colaboradora e começamos a trabalhar simulando **rotinas** de desenvolvimento de software. Você realizou mudanças no código, *commits*, assim como eu no meu computador, baixando os seus commits e enviando os commits para o GitHub.

Cada commit representa uma **versão** do código, que fica registrada no histórico. Conhecemos o git-log e conseguimos conferir todo o histórico de versões. E, eventualmente, vamos querer voltar no tempo, pois alguém pode pedir para **desfazer uma alteração**. O Git também nos ajuda com esta situação, correto?

**Gabrielle:** Sim, o Git pode nos ajudar nessas situações. Notei que você fez uma nova modificação no nosso projeto. Havia a imagem de uma menina ao fundo e você a removeu. Agora, vamos supor que queremos trazer essa imagem de volta, retornar para a versão em que tínhamos essa imagem no projeto. Como podemos fazer isso?

**Rodrigo:** Neste caso, poderíamos entrar no site do GitHub, clicar na lista do log para ver os commits e encontrar qual foi o commit que fez essa alteração. Lembre-se de que é possível detalhar o que aquele commit fez no código e quais arquivos foram modificados.

Neste nosso exemplo, seria simples encontrar as alterações, porque a mudança ocorreu apenas no index.html. Porém, e se o commit tivesse alterado 86 arquivos, por exemplo? Não seria nada produtivo ter que verificar arquivo por arquivo, ir ao VS Code e desfazer a alteração.

Presumo que o Git tenha um comando para automatizar esse processo, certo, Gabi?

**Gabrielle:** Isso mesmo. Vamos descobrir qual é esse comando?

Vamos abrir o VS Code e, naquela ferramenta do GitHub que estávamos utilizando no próprio VS Code, para abrir o log, aquele histórico dos nossos commits. A ferramenta é a "Source Control", no terceiro botão do menu lateral esquerdo.

Clicando nessa ferramenta, temos o menu de três pontos (botão "Views and More Actions"). Vamos clicar nele e verificar se temos a opção de log. Procurando nas opções, não vemos nenhuma que mostre nosso histórico de commits.

**Rodrigo:** Verdade. Começamos a usar essa integração do Visual Studio Code com o Git, mas nem todos os comandos estão disponíveis via menu. Os mais comuns, como ver o status, qual arquivo foi modificado, fazer um commit, *ADD*, *push*, *pull*, são exibidos.

Mas, se quisermos executar um log ou outros comandos mais avançados, e que não são tão comuns, o VS Code não mostra. Sendo assim, teremos que usar o terminal novamente.

**Gabrielle:** Vamos abrir novamente o terminal, clicando em "Terminal > New Terminal" no menu superior. Vamos minimizar o menu lateral esquerdo para deixar apenas o terminal na tela.

Agora, vamos rodar o comando git log, que já conhecemos. Como vimos anteriormente, esse comando traz algumas informações dos nossos commits. Cada commit possui um **ID**, o qual identifica esse commit de maneira única.

**Rodrigo:** O ID único é a maneira pela qual o Git diferencia cada commit. É como se fosse um ID de registro no banco de dados. Precisamos saber qual é o commit que queremos desfazer e, para isso, usaremos esse ID.

**Gabrielle:** O commit que queremos desfazer é o "**removendo foto**", então vamos copiar o ID dele (o código de letras e números ao lado de "commit").

**commit** 2ad48c068dc9677fb57efec70620700410f976b0

Em seguida, pressionamos a tecla "Q" para retornar ao nosso terminal. **O comando para reverter um commit é o git revert**, passando ao lado o ID do commit a ser desfeito.

git revert 2ad48c068dc9677fb57efec70620700410f976b0

**Rodrigo:** Executando o comando git revert, é aberta uma pequena janela no topo da tela. Isso acontece porque o git revert é o comando responsável por realizar o processo de identificar o commit pelo ID passado como parâmetro, analisar cada alteração feita por ele e desfazer essas alterações sozinho.

Entretanto, o revert não desfaz as alterações modificando apenas os arquivos. Ele cria um novo commit para registrar esse revert.

**Gabrielle:** Portanto, o revert já nos fornece uma **mensagem** para esse novo commit,dizendo que realizou um revert e traz a mensagem do commit original, por exemplo, além do ID desse commit.

*Revert "removendo foto"*

*This reverts commit 2ad48c068dc9677fb57efec70620700410f976b0.*

Fechando esta janela, essa mensagem já ficará salva para o novo commit.

Agora, vamos executar o comando git log no terminal para verificar o que aconteceu.

Na lista de commits, foi retornado esse novo commit no topo da lista, indicando que ocorreu um revert do commit de "removendo foto".

**Rodrigo:** Então, na verdade, o revert não apaga o commit original. O que ele faz é criar um novo commit que efetua uma espécie de "Ctrl + Z" no commit original. Todos os arquivos modificados pelo commit original têm suas alterações desfeitas, mas o revert fica registrado no histórico.

No entanto, como é um novo commit, ele está apenas no nosso repositório local. Portanto, quisermos enviá-lo para o GitHub, será necessário sincronizar, rodando novamente o seguinte comando no terminal:

git push origin main

Com isso,sincronizamos e enviamos esse commit de revert para o repositório remoto, permitindo que outras pessoas possam baixá-lo.

Será que essa alteração funcionou? Vamos verificar no site se a foto da pessoa foi restaurada

**Gabrielle:** Abrindo o navegador, podemos notar que a imagem da menina já está no plano de fundo novamente. Portanto, deu certo!

**Rodrigo:** Nós fizemos o revert, desfizemos a alteração de um determinado commit. O próprio git se encarregou de olhar cada arquivo, desfazer as respectivas alterações e registrar isso no histórico, revertendo o código para a versão anterior.

Esta é uma situação comum. Eventualmente, as pessoas que usam o sistema podem solicitar uma alteração e, depois de uma semana ou um mês, elas podem mudar de ideia e dizer:

— Olha, testamos a alteração que você fez naquele dia, e não era exatamente o que queríamos. Você pode reverter para o como estava antes?

E nós não lembraremos de cabeça como era o código anterior. Portanto, graças a essa funcionalidade de registro por commits, nós conseguimos localizar este commit e fazer o revert, recuperando a versão do código daquele momento.

**Gabrielle:** O histórico de comandos pode ser útil em **outras situações**. Por exemplo, vamos imaginar que em vez de querer reverter um commit, tenhamos feito algo no repositório local e queiramos **excluir** esse commit, removê-lo do histórico. É possível fazer isso?

**Rodrigo:** Essa é uma situação diferente. Agora não queremos mais apenas desfazer um commit, mas de fato apagá-lo. É possível fazer isso, e aprenderemos a seguir.

02

**Resetando um commit**

**Rodrigo:** No último vídeo, aprendemos como desfazer um commit utilizando o comando git revert. Esse comando gera um novo commit com essa mudança, desfazendo todo o código que foi alterado naquele determinado commit.

No entanto, existem situações em que não queremos, de fato, desfazer um commit. Na verdade, queremos **apagar o commit do histórico**.

Por exemplo, nós podemos estar trabalhando em uma funcionalidade que no meio do caminho é cancelada, sendo que nós já tínhamos alterado os arquivos e feito um ou mais commits. Então, aqueles commits não fazem mais sentido no histórico e queremos apagá-los.

Ou, em outra situação comum, você pode estar realizando uma tarefa e outra pessoa inicia a mesma tarefa sem avisar, termina, faz o commit e faz o push. Então, aquele commit que você havia feito também não faz mais sentido e nós queremos apagá-lo.

Temos um comando para fazer isso?

**Gabrielle:** Sim! Vamos conhecê-lo.

Podemos abrir o nosso arquivo index.html e realizar uma mudança. Na linha 22, nós temos um título H1 "Adivinhe o número secreto":

index.html

<h1>Adivinhe o <span class="container\_\_texto-azul">numero secreto</span></h1>

Vamos apagar essa linha e realizar o *commit* da maneira que já sabemos: clicando em "Source Control > Stage Changes (ícone de adição)" para adicionar o index.html no commit. Depois, escrevemos uma mensagem de commit no campo de texto logo acima:

*Remove título*

Depois, clicamos no botão "Commit". Não vamos mandar essa alteração para o GitHub.

E agora, como podemos fazer para reverter esse commit que acabamos de fazer?

**Rodrigo:** Acabamos de simular a situação em que uma pessoa realizou uma mudança no código, apagando aquele título H1, e fez o commit. Só que ela não sincronizou a atualização e recebemos um pedido para cancelá-la, por exemplo. Mas o commit já está registrado no repositório local dessa pessoa.

Então, podemos rodar um log para visualizar esse commit no histórico, mas terá que ser pelo terminal. Na sequência, aprenderemos a apagar esse commit.

**Gabrielle:** Vamos fechar a aba do Source Control, abrir um novo terminal e rodar o comando git log.

**Rodrigo:** Da mesma forma que o revert, se queremos apagar um commit, vamos precisar do ID do commit em questão para identificá-lo.

**Gabrielle:** Então, vamos copiar o ID do commit "Remove título" que acabamos de fazer:

a3322db2eb6f82162977169f5461fc93b81bfac1

Em seguida, limpamos nosso terminal. **O comando para apagar um commit é o git reset --hard junto do ID do commit**.

git reset --hard a3322db2eb6f82162977169f5461fc93b81bfac1

**Rodrigo:** Um detalhe importante: o parâmetro --hard foi colocado porque o comando reset possui algumas opções, alguns modos de realizar esse reset. O parâmetro --hard serve para apagar o commit e também avisei apagar a mudança no código.

No **Para Saber Mais** desta aula, explicamos outras possibilidades de uso do comando reset.

**Gabrielle:** Será que funcionou? Vamos rodar o git log no terminal para verificar.

Estranhamente, o commit que tentamos apagar, "Remove título", ainda está presente no histórico. Parece que não funcionou.

**Rodrigo:** Tem uma pegadinha aí. Na verdade, com o comando reset utilizando a opção --hard, temos que passar o ID do commit, mas não é o ID do commit que queremos resetar, mas sim o **ID do comando anterior**.

Acabamos copiando o ID do commit "Remove título", mas temos que passar o ID da versão a que queremos retornar. Ou seja, esse comando retorna ao estado do commit indicado pelo ID. Então, seria o ID do commit anterior, "Revert 'remove foto'".

Vamos copiar o ID correto dessa vez e limpar o terminal. Vamos digitar novamente o comando git reset --hard com o ID do commit "Revert 'remove foto'".

git reset --hard 7f69ff8220e093d2c8ad234ceec109a6e794e5f61

Ao dar "Enter", uma mensagem aparece indicando que nosso HEAD agora está no início do ID, que identifica nosso commit.

Vamos rodar novamente o git log. Agora parece que funcionou.

**Rodrigo:** Aquele último commit, "Remove título", sumiu. Mas será que ele desfez as mudanças no código? Vamos ver se voltou aquele H1 que apagamos.

**Gabrielle:** Vamos fechar o terminal e expandir o index.html. E pronto, retornou: na linha 22 temos o nosso H1. Deu certo!

**Rodrigo:** Além do comando revert, que nos permite **reverter** uma mudança mantendo o commit e criando um novo, temos o comando reset para quando queremos **apagar** um determinado commit, excluí-lo do histórico porque ele não faz mais sentido.

Nos últimos dois vídeos, nós apresentamos essas duas situações: em uma delas estamos revertendo um commit, na outra estamos apagando um commit. Mas há uma terceira situação possível.

Imagine que não queremos nem apagar nem desfazer, mas **alterar** um commit. Por exemplo: mudar a mensagem do commit ou incluir um arquivo que esquecemos de adicionar no momento do commit. O Git também nos ajuda com isso.

**Gabrielle:** Mas antes de falar um pouco mais sobre isso, é importante lembrarmos que **o comando reset que acabamos de usar deve ser usado apenas quando tivermos feito esse commit de alteração no nosso repositório local.**

Caso já tenhamos subido esse commit, não é tão interessante usar esse comando e alterar o histórico de versões que já está público.

**Rodrigo:** Essa é mais uma vantagem do Git: nós fazemos os commits e eles só existem localmente até o momento que sincronizamos e enviamos para o GitHub.

No entanto, enquanto o commit está apenas local, as outras pessoas não têm esse commit. Portanto, podemos apagar, desfazer, fazer mudanças. Mas, se já enviamos para o GitHub, não é recomendado apagar um commit, porque isso poderia causar confusão.

**Gabrielle:** Agora podemos continuar e aprender como realizar uma alteração na mensagem ou algum detalhe de um commit. Vamos lá?!

03

**Para saber mais: mais opções do comando git reset**

Aprendemos como utilizar o comando git reset com a opção --hard, que define o modo como o reset será realizado. Entretanto, essa não é a única forma de uso desse comando.

Você pode encontrar as outras opções do comando git reset na [documentação oficial do git](https://git-scm.com/docs/git-reset/pt_BR).

04

**Alterando o último commit**

**Rodrigo:** Já aprendemos alguns comandos para alterar o histórico, reverter um commit e apagar um commit. No entanto, temos também a terceira situação, onde nós queremos apenas **alterar um commit**.

Já fizemos a simulação de um novo *commit*. Vamos rodar um git log no terminal para verificar o que aconteceu.

Lá está o nosso último commit: "Trocando texto do botão *avidinhar*". Vamos simular essa situação.

A Gabi editou o texto do botão, mudando "chutar" para "adivinhar" e fez o commit. Mas ao digitar a mensagem do commit, houve um pequeno erro. Em vez de "adivinhar", ficou "*avidinhar*".

Essa é uma situação em que se pensa: *Já fiz o commit, e agora? Preciso alterar essa mensagem! Será que vou ter que apagar esse commit e fazer um totalmente novo*?

É possível alterar o commit, Gabi?

**Gabrielle:** Sim! Vamos limpar o terminal primeiro. Podemos fazer isso por meio do **comando git commit --amend -m**, seguido pela minha mensagem correta, que seria "Trocando botão para adivinhar".

git **commit** --amend -m "Trocando botao para adivinhar"

**Rodrigo:** É o próprio comando git commit, como se estivéssemos fazendo um novo commit. Mas não é um novo commit por causa do parâmetro --amend.

Ele indica ao Git que não queremos fazer um novo *commit*, mas alterar o anterior. Então é só passar o -m, com o texto da mensagem correta.

**Gabrielle:** Vamos ver se vai dar certo. Após executar o comando acima, temos o retorno de sucesso. Parece que deu certo.

Vamos executar o git log para confirmar se ele realmente alterou o commit anterior em vez de criar um novo commit.

Parece que deu certo! O último commit da lista tem a mensagem "Trocando botao para adivinhar".

**Rodrigo:** Essa é uma situação comum. Eventualmente digitamos com pressa e acabamos errando a mensagem. Mas não queremos deixar aquela mensagem feia e errada.

Ou a situação pode ter sido a seguinte: você editou seis arquivos, por exemplo, e quando foi fazer o commit, você adicionou cinco e deixou um para trás. E então você fez o commit.

Também é possível fazer esse comando -- amend. Nesse caso você não iria alterar a mensagem, mas precisaria apenas adicionar o arquivo e fazer o commit com o amend.

São situações comuns que, de vez em quando, nos deparamos.

Com isso, aprendemos nessa aula alguns comandos para **voltar no tempo**, modificar o histórico de versão. Seja para desfazer, apagar ou mesmo alterar um commit.

Na sequência, vamos aprender outros conceitos e recursos dentro do GitHub.

05

**Para saber mais: cuidados ao mudar o histórico**

Nessa aula, exploramos como é possível alterar o histórico de commits no Git, mas é fundamental exercer cautela ao realizar tais mudanças. Como diz o ditado, "com grandes poderes vêm grandes responsabilidades".

É importante destacar que os comandos do Git que permitem modificar o histórico de commits devem ser utilizados com **prudência** e apenas quando o commit em questão ainda **não foi enviado ao repositório remoto**, ou seja, quando ele existe apenas no seu repositório local.

Modificar um commit que já se tornou público, ou seja, aquele que já foi enviado ao GitHub ou a qualquer outro repositório remoto, pode acarretar problemas consideráveis na colaboração com as outras pessoas e na integridade do histórico de um projeto.

Em situações de colaboração em equipe, é essencial manter a integridade do histórico de commits, pois qualquer modificação em um commit que outras pessoas estejam trabalhando pode resultar em conflitos e dificuldades na colaboração.

É recomendável evitar a modificação excessiva do histórico de commits, uma vez que isso pode tornar o histórico confuso. O histórico deve ser uma representação precisa do progresso do projeto ao longo do tempo.

07

**Faça como eu fiz: modificações no último commit**

Agora é com você! Altere a mensagem do seu último commit, como foi demonstrado no vídeo anterior.

**Opinião do instrutor**

Antes de modificar o último commit, vamos usar o comando a seguir para visualizar informações a respeito desse commit:

git log

Para alterar informações do último commit, como o texto da mensagem, use o comando a seguir:

git **commit** --amend -m “NOVA MENSAGEM”

Não esqueça de substituir o NOVA MENSAGEM por um texto descritivo para aquele commit.

Para verificar se a mensagem foi alterada, use novamente o comando a seguir:

git log

08

**Desafios: hora da prática**

Domine o Controle de Versão com estes desafios práticos! De visualizar e modificar commits a sincronizar com repositórios remotos, estes **exercícios não obrigatórios** proporcionarão uma experiência prática para fortalecer sua compreensão do Git.

A prática constante é a chave para se tornar um mestre no Git!

**Desafios**

1. Acesse todos os commits realizados
2. Modifique o último commit
3. Reverta mudanças no repositório local
4. Apague um ou mais commits
5. Sincronize as modificações com o repositório remoto
6. Analise as mensagens de commits realizados e faça as alterações de acordo com as boas práticas

**Opinião do instrutor**

**Desafio 1: Veja todos os commits realizados**

Para visualizar todos os commits realizados em um repositório Git, você pode usar o comando git log. Este comando exibe o histórico de commits, mostrando informações como o hash do commit, autor, data, e mensagem do commit. Aqui estão algumas variações do comando git log que podem ser úteis:

* Para exibir o histórico completo: git log
* Exibir Alterações Detalhadas: git log -p
* Exibir Apenas Mensagens de Commit: git log --oneline

**Desafio 2: Modificar o Último Commit com git commit --amend:**

* Faça um commit com algumas alterações no seu código.
* Perceba que esqueceu de incluir algumas modificações.
* Para adicionar as modificações esquecidas ao último commit, sem criar um novo commit, utilize o comando: git commit --amend

**Desafio 3: Reverter Mudanças de um Commit com Git Revert:**

* Crie um novo arquivo no seu repositório.
* Realize um commit adicionando esse novo arquivo.
* Para reverter automaticamente as mudanças feitas no último commit sem excluir o histórico: git revert <hash-do-commit>
* Se você precisar reverter uma série de commits, pode utilizar o seguinte comando: git revert -n <hash-do-ultimo-commit-a-manter>
* Lembre-se que após reverter, é necessário realizar um novo commit para aplicar a reversão ao repositório

Lembre-se de que o git revert é uma maneira segura e não destrutiva de desfazer alterações, pois não reescreve o histórico existente. Ele é particularmente útil em ambientes de trabalho compartilhados, onde reescrever o histórico pode causar problemas para outros colaboradores.

**Desafio 4: Apagar um Commit com Git Reset:**

* Faça uma série de commits com alterações no seu código.
* Escolha um commit específico que deseja excluir.
* Use para apagar o commit selecionado, desfazendo automaticamente as mudanças no código: git reset --hard <hash-do-ultimo-commit-a-manter>
* Se você apenas deseja desfazer commits, mas manter as alterações no diretório de trabalho, você pode usar git reset --soft.

Observação: No cotidiano de trabalho em desenvolvimento, a escolha entre reset e revert deve ser tomada com base nas necessidades específicas do seu projeto e na colaboração com as outras pessoas desenvolvedoras. Se você deseja desfazer alterações em um commit específico sem reescrever o histórico, git revert é uma escolha mais segura. Se você precisa reverter completamente para um estado anterior, git reset pode ser apropriado, mas use-o com cautela, especialmente em branches compartilhadas.

**Desafio 5: Sincronize as modificações com o repositório remoto**

* Execute o comando: git push

**Desafio 6: Analise as mensagens de commits realizados e faça as alterações de acordo com as boas práticas:**

* Para visualizar todo o histórico de commits, execute o comando: git log
* Leia a [documentação da *Conventional Commits*](https://www.conventionalcommits.org/pt-br/v1.0.0-beta.4/) para elaborar as mensagens de commits de acordo com as boas práticas
* Reescreva os commits utilizando os comandos já aprendidos: git commit –amendou git revert.

5.

01

**Readme do repositório**

**Gabrielle:** Já aprendemos diversas informações relevantes sobre o uso do Git e do GitHub!

Durante nosso trabalho nesse projeto, entendemos do que se trata e o que o código realiza. Porém, caso outra pessoa que não faça parte de nossa equipe entre nesse repositório, não conseguirá compreender do que se trata esse projeto.

É possível documentar e explicar o que o projeto realiza? Quais tecnologias utilizamos e, enfim, fornecer um resumo sobre esse código?

**Rodrigo:** Ótima questão, Gabi! No futuro, outra pessoa pode ingressar no time ou podemos decidir transformar esse projeto em *open source* (código aberto) e permitir que qualquer pessoa colabore.

Nessas situações, essas pessoas não conseguirão discernir quais são os arquivos e o que cada um deles faz, pois não há um contexto para o projeto além do código.

Acessando a página do repositório no GitHub, parte inferior, aparece uma mensagem indicando precisamente isso:

Help people interested in this repository understand your project by adding a README

Portanto, está perguntando se não gostaríamos de informar às outras pessoas do que se trata esse projeto. À frente, há um botão chamado "Adicionar um README".

Existe esse conceito chamado **README**, que significa "Leia-me" em inglês. Ele atua como uma documentação. Portanto, é comum encontrarmos repositórios do GitHub que possuem esse arquivo como uma documentação, mas ao invés de usar o Word ou um PDF, o arquivo fica dentro do repositório em si. Dessa forma, a documentação permanece próxima ao código.

O GitHub tem um tratamento especial para isso. Se identifica que há um arquivo com esse nome no repositório, o texto é exibido integralmente na parte inferior. É possível formatá-lo utilizando HTML e o formato Markdown.

O que acha de criarmos esse arquivo README e documentarmos nosso repositório?

**Gabrielle:** Vamos lá!

**Rodrigo:** Podemos fazer isso pelo Visual Studio Code. Basta criar um arquivo com esse nome, "README", em maiúsculo, e a extensão .md de Markdown. Podemos fazer isso diretamente na IDE ou pelo próprio site do GitHub. Como estamos com o GitHub aberto, faremos por aqui.

Ao clicarmos o botão "Add a README", ele abrirá um formulário, como se estivéssemos criando o arquivo dentro do GitHub. Perceba que, no topo da página, ele já inseriu o nome do arquivo, README.md, e há um campo para digitarmos nossa documentação.

Nesse campo, precisamos escrever o que é o projeto, quais são as tecnologias, quem são as pessoas que trabalham, podendo ainda formatar esse texto, colocá-lo em negrito, itálico, adicionar fotos, entre outras opções.

Também é possível editar o código e há um botão chamado "Preview" para pré-visualização do texto escrito até o momento. No momento, só temos o texto "numero-secreto", mas a ideia é apagar isso e começar a descrever o projeto.

Podemos usar *tags* HTML para quebrar linhas, colocar tabelas, fazer alinhamentos, entre outras formatações. Também podemos utilizar o próprio Markdown, que é uma linguagem de marcação. Inclusive, vamos disponibilizar um **Para Saber Mais**, explicando como fazer formatações com Markdown e com HTML.

Para não perdermos muito tempo, digitando do zero, já preparamos um README. Vamos apenas colá-lo.

<h1>Jogo do número secreto</h1>

<h2>�� Sobre</h2>

<p>Projeto utilizado nos cursos de lógica de programação da Alura.</p>

## �� Tecnologias

<div>

<img src="https://img.shields.io/badge/HTML-239120?style=for-the-badge&logo=html5&logoColor=white">

<img src="https://img.shields.io/badge/CSS-239120?&style=for-the-badge&logo=css3&logoColor=white">

<img src="https://img.shields.io/badge/JavaScript-F7DF1E?style=for-the-badge&logo=javascript&logoColor=black">

</div>

// Código omitido.

Para consultar o arquivo completo, basta acessar o [GitHub do curso](https://github.com/alura-cursos/3386-git-github/commit/16018ade439fdae0582aa0108f2a94974f9c97ec)

Ele contém algumas *tags* HTML e algumas formatações do Markdown. Vamos pré-visualizá-lo apertando "Preview".



A documentação apresenta o nome do projeto, um breve resumo do que é o projeto, quais são as tecnologias, perfil do time, entre outras coisas.

**Gabrielle:** A documentação do nosso projeto ficou muito bonita!

**Rodrigo:** Passamos o texto no campo "Edit", clicamos em "Preview", visualizamos e gostamos do resultado. Está provado, Gabi?

**Gabrielle:** Aprovado! Ótimo!

**Rodrigo:** Como podemos sinalizar ao GitHub para criar esse arquivo? Precisamos lembrar que se trata de um arquivo novo que estamos adicionando ao projeto. Vamos ter que fazer um *commit* dele. É possível fazer isso diretamente na interface do GitHub.

No canto superior direito da tela, apareceu um botão: "Commit changes". Ele gerará um *commit* criando esse novo arquivo. Ao selecioná-lo, aparecerá uma pequena janela com um campo para digitarmos a mensagem do *commit*. Há outro campo opcional, caso queiramos descrever com mais detalhes esse *commit*.

Ele nos pergunta se queremos fazer esse *commit* na "main branch" (principal). Manteremos essa opção e clicaremos no botão "*Commit*".

Um novo *commit* foi criado. Já é possível visualizar que o último *commit* foi o nosso e a mensagem "create README.md". Agora, quando entramos no repositório, o GitHub detecta que existe um arquivo README.

Abaixo da lista de diretórios e arquivos do projeto, ele já exibe automaticamente a documentação. Nosso projeto está documentado, Gabi.

**Gabrielle:** Maravilha! Agora, quando alguém quiser colaborar com o nosso projeto, fica bem mais fácil entender o que está acontecendo dentro do nosso repositório, certo?

**Rodrigo:** Ótimo! Disponibilizaremos outro **Para saber mais** com dicas de como formatar esse README para torná-lo agradável, repleto de informações sobre o projeto, seguindo o padrão das empresas e dos projetos de tecnologia.

**Gabrielle:** Muito bem, Rodrigo! Surgiu outra dúvida em relação ao Git e GitHub. Quando estamos trabalhando em um projeto no nosso computador, a tecnologia que estamos usando naquele projeto pode gerar alguns arquivos de *cache*, e talvez não seja tão interessante enviá-los para o GitHub.

Podemos até ter arquivos com informações confidenciais. Será que há alguma forma de limitar o que subimos para o GitHub e o que fica apenas no nosso computador?

**Rodrigo:** Muito importante! Alguns arquivos estão na pasta do projeto, mas não queremos enviá-los para o GitHub, como arquivos de senha. Na sequência, entenderemos como esconder determinados arquivos do repositório.

02

**Para saber mais: formatação com markdown**

Vimos que para criar um README, usamos a linguagem de marcação de texto **Markdown**.

Caso você queira aprender mais sobre esse tipo de formatação para criar READMEs incríveis, temos aqui na Alura o artigo [Markdown: como trabalhar com essa linguagem de markup?](https://www.alura.com.br/artigos/como-trabalhar-com-markdown" \t "_blank).

Você também pode acessar a documentação a seguir que mostra os principais códigos em Markdown: [Markdown Quick Reference Cheat Sheet](https://wordpress.com/support/markdown-quick-reference/" \t "_blank).

03

**Para saber mais: melhorando o README**

O README é uma parte importante dos repositórios do Github. Nele serão documentadas informações relevantes do projeto para que pessoas que ainda não o conhecem possam entender melhor os seus detalhes.

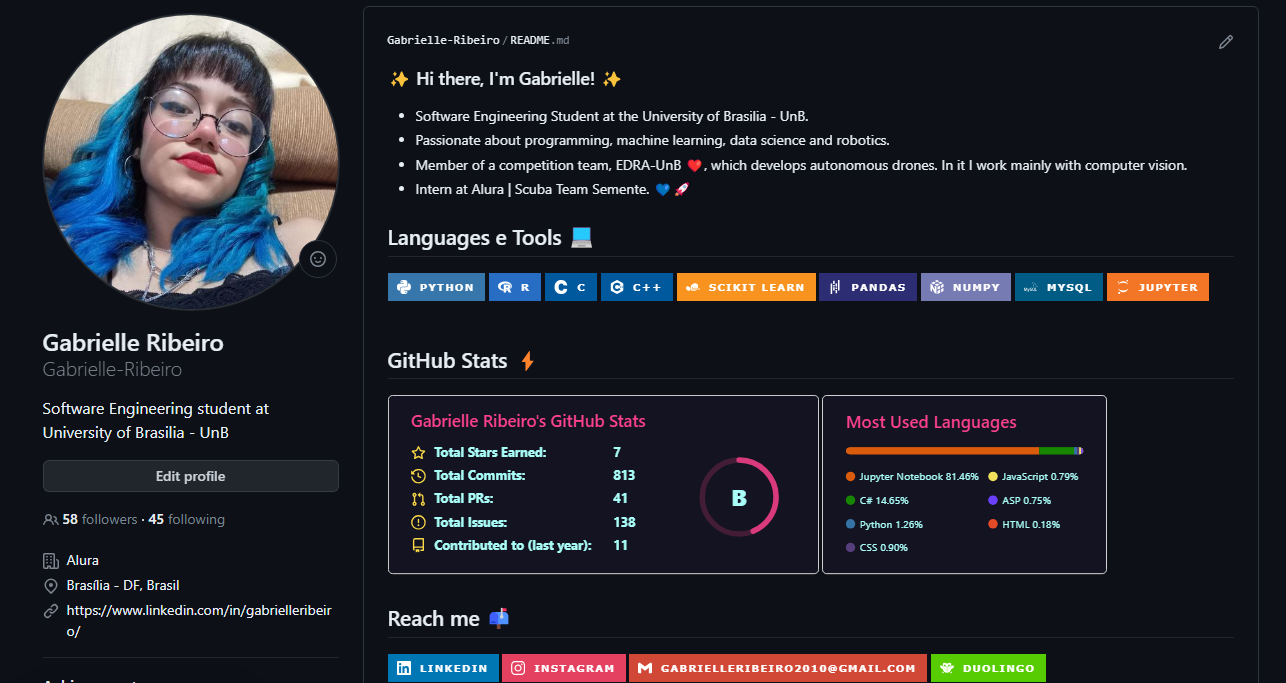
Assim, caso você queira saber como criar um bom README para os seus repositórios, você pode ler o artigo [Como escrever um README incrível no seu Github](https://www.alura.com.br/artigos/escrever-bom-readme).

04

**Para saber mais: README para seu perfil no GitHub**

No GitHub você pode **personalizar seu perfil** com um README especial para mostrar informações que você considera importante que outras pessoas saibam sobre você, como seu trabalho ou estudos, projetos que você faz parte, seus dados de contato, etc.

Um exemplo pode ser visto a seguir:



Para criar o seu próprio README, você pode ler o artigo [Como criar um README para o seu perfil do GitHub](https://www.alura.com.br/artigos/como-criar-um-readme-para-seu-perfil-github).

05

**Ignorando arquivos no repositório**

**Gabrielle:** Já sabemos que nem sempre é adequado enviarmos para o GitHub todos os arquivos e diretórios do nosso projeto local.

Como podemos fazer para que o GitHub ignore esses arquivos quando realizarmos o upload?

**Rodrigo:** Boa pergunta, Gabi! Vamos voltar para o Visual Studio Code.

No caso do nosso projeto, criamos o repositório na pasta "NUMERO-SECRETO", que é a pasta principal do projeto. Todos os arquivos contidos nela, como style.css, index.html, app.js, até mesmo os diretórios, como a pasta .img e todos os arquivos internos dela, serão rastreados pelo Git quando usamos o comando git add..

No entanto, pode haver uma pasta ou arquivo específico que desejamos que o Git ignore. Talvez seja um arquivo ou diretório necessário para a execução do projeto localmente, mas que não faz sentido compartilharmos com outras pessoas. Cada uma terá seu arquivo ou diretório específico, portanto, não queremos enviar para o GitHub.

Porém, se adotarmos essa estratégia, teríamos que ter cuidado ao realizar um commit para não usar git add., caso contrário, o Git incluiria automaticamente o diretório que gostaríamos de ignorar. E então, teríamos que ficar adicionando individualmente cada arquivo, o que não seria nada produtivo.

**Gabrielle:** Isso é verdade, também corremos o risco de nos confundirmos, esquecermos qual arquivo deve ser ignorado e acabar enviando por engano.

**Rodrigo:** Sim. Seria muito mais fácil se o próprio Git pudesse fazer isso automaticamente. Continuaríamos usando git add., mas o Git saberia que deveria ignorar alguns arquivos ou pastas específicos.

Existe um recurso que nos permite criar um arquivo para informar ao Git quais diretórios e arquivos do projeto ele deverá ignorar. Esse arquivo é chamado de .gitignore. Podemos criar um arquivo com esse nome no nosso projeto.

No Visual Studio Code, criaremos um novo arquivo. O nome desse arquivo deve ser .gitignore, com um ponto no início, pois será um arquivo oculto. Após criar o arquivo .gitignore, incluiremos nele quais arquivos e diretórios do projeto queremos ignorar.

Por exemplo, vamos supor que temos uma pasta chamada temp/ no projeto, que é uma pasta temporária, com arquivos temporários. Estamos adicionando essa pasta no arquivo .gitgnore para que o Git ignore esse arquivo. Se tivermos outros arquivos e diretórios, cada um deles estaria separado por linhas.

Portanto, na linha abaixo, poderíamos colocar arquivo.txt passando um nome específico ou padrão, por exemplo, \*.css. Nós usaremos apenas o temp/. Com isso, estamos dizendo, basicamente, para o Git: "Ignore a pasta temp do projeto".

Por fim, vamos fazer um teste, Gabi?

**Gabrielle:** Claro, Rodrigo! Então, vamos criar uma pasta com o nome temp. Dentro dela, criaremos um arquivo teste.txt e adicionaremos um texto qualquer: "nao deveria ir para o git".

Agora, criamos o gitignore e adicionamos a pasta temp (com o arquivo teste.txt) para ser ignorada. Em teoria, se fizermos um commit, o Git deverá ignorar esse arquivo.

Vamos fazer uma mudança no index.html. No trecho de código a seguir, substituiremos a palavra "Adivinhe" por "Descubra":

// Código omitido.

<h1>Descubra o <span **class**="container\_\_texto-azul">numero secreto</span>

// Código omitido.

Realizamos uma alteração no index.html e queremos efetuar o *commit* dessa mudança. Vamos verificar se ele irá ignorar a pasta temp e levar apenas a alteração do index.

Acessaremos o atalho do Git no Visual Studio Code para fazer um novo *commit*. Adicionaremos o index.html e o .gitignore, que precisa ser enviado ao repositório. Também colocaremos uma mensagem: "mudanças no título e gitignore" e apertaremos o botão de "Commit".

Após isso, apertaremos o botão "Sync Chages" (sincronizar), para enviarmos ao GitHub e aparentemente deu certo. Na sequência, verificaremos no site do GitHub se a pasta foi realmente ignorada.

Ele mostrou o arquivo .gitignore, mas não trouxe a pasta temp, somente a pasta img. Deu certo!

**Gabrielle:** Ficou bem legal! Acredito que exista algum recurso ou algo que nos auxilie a escrever esse .gitignore, já que essa é uma ferramenta muito comum no dia a dia das pessoas desenvolvedoras, certo?

**Rodrigo:** Isso varia muito de linguagem para linguagem. Por exemplo,as aplicações em **Java** têm os arquivos .class, .jar, que normalmente não são inseridos no repositório.

Em **PHP**, teremos outros arquivos, assim como em **Python**, **Hornet**, cada linguagem tem seus arquivos temporários - arquivos de compilação - e não faz sentido enviá-los para o repositório.

Ao invés de criarmos o .gitignore manualmente, precisando lembrar quais são os arquivos e diretórios daquela linguagem, daquela tecnologia, existem sites que nos auxiliam nessa tarefa. Existem sites geradores de .gitignore para cada tecnologia.

Um desses auxiliadores é o site [gitignore.io](https://www.toptal.com/developers/gitignore), que é um site utilitário. Nele, indicamos a tecnologia ou a linguagem de programação, por exemplo, Java, e apertamos o botão "criar". Com isso, ele retorna como seria um .gitignore para um projeto com Java.

Ele fornece o arquivo completo, inclusive, listando \*.class, \*.log, .ctxt, e o diretório temp, .jar. Para o Java, esses são os arquivos a serem ignorados.

# Created by https://www.toptal.com/developers/gitignore/api/java

# Edit at https://www.toptal.com/developers/gitignore?templates-java

### Java ###

# Compiled class file

\*.class

# Log file

\*.log

# BlueJ files

\*.ctxt

#Mobile Tools for Java (J2ME)

.mtj.tmp/

# Package Files #

\*.jar

\*.war

\*.nar

\*.ear

\*.zip

\*.rar

# virtual machine crash logs, see http://www.java.com/en/download/help/error\_hotspot.xml

hs\_err\_pid"

replay\_pid\*

#End of https://www.toptal.com/developers/gitignore/api/java

Agora, vamos substituir a linguagem Java para Django, que é uma tecnologia do Python. Neste caso, temos outros arquivos e diretórios para ignorar. Essa ferramenta é muito útil, pois já disponibiliza um .gitignore pronto, bastando apenas copiá-lo e colá-lo em nosso projeto.

**Gabrielle:** Aprendemos muitas coisas interessantes e nosso projeto está bem legal, mas teve um trecho de código que gostei bastante e queria compartilhar com as pessoas só esse pedaço do código. Será que preciso criar um novo repositório para colocar só este trecho?

**Rodrigo:** Caso você queira compartilhar apenas um trecho de código, não um projeto completo, o GitHub também tem uma ferramenta para isso. Em seguida, entenderemos como ela funciona.

06

**Compartilhando códigos com Gist**

**Gabrielle:** Quero compartilhar apenas um trecho do nosso projeto e não ele inteiro. Então, Rodrigo, você me disse que eu não preciso criar um novo repositório, porque existe uma ferramenta que pode me auxiliar nisso.

**Rodrigo:** Exato! O próprio GitHub oferece um recurso para quando desejamos compartilhar trechos de códigos com outras pessoas, sem a necessidade de criar um repositório e realizar todo o processo de commits que realizamos.

Abrindo o site do GitHub, no ícone de "+ (mais)", no canto superior direito, além da opção de criar um novo repositório, encontramos outras opções. Dentre elas, temos uma chamada "New Gist".

Ao selecioná-la, seremos direcionados para um formulário. A ferramenta **GitHub Gist** foi desenvolvida pelo próprio GitHub para compartilhamento de códigos. A descrição inclusive menciona "compartilhe códigos de maneira instantânea, comentários ou trechos de códigos pontualmente".

Nós podemos criar um "Gist" e, assim que a criação for realizada, será gerada uma URL que copiaremos e compartilharemos com outras pessoas. Vamos compartilhar um trecho do nosso projeto?

Então, vamos voltar para o Visual Studio Code. Vamos supor que queremos compartilhar a biblioteca que está sendo utilizada no projeto para a leitura dos trechos da página. Enfim, queremos mostrar para as pessoas que usamos uma biblioteca de inteligência artificial para ler e interpretar o site.

Seria, portanto, a tag script localizada no nosso arquivo index.html.

<script src="https://code.responsivevoice.org/responsivevoice.js"></script>

Vamos copiar esse trecho de código. Agora, voltando para o formulário do Gist, podemos inserir uma descrição, ou seja, um texto explicando o que ele é: "Código de leitura de tela".

Abaixo deste campo, há outro para incluir arquivos. Nele, é possível compartilhar um ou vários arquivos de forma separada, a ferramenta permite isso. Precisamos definir o nome do arquivo: index.html e colar a tag script no campo abaixo.

Ao final da tela, existe uma opção para adicionar arquivos. Se quisermos compartilhar outro arquivo, podemos adicionar usando este botão "Add file". Então, vamos selecioná-lo. Ele abriu um novo formulário para incluirmos um novo arquivo. Vamos incluir o app.js.

Retornando ao nosso código, compartilharemos também uma linha do nosso arquivo app.js, onde configuramos a biblioteca para utilizar a voz do Google na leitura:

responsiveVoice.**speak**(texto, 'Brazilian Portuguese Female', {rate:1.2});

Copiaremos esse trecho de código e colaremos no Gist. Pronto! São dois arquivos, duas linhas de código.

Ao final da página, existe um botão verde, "Create secret gist", que serve para criarmos o Gist. E ele também tem a opção de ser um Gist público ou secreto. No nosso caso, vai ser secreto ou vai ser público?

**Gabrielle:** Vai ser público. Essa biblioteca de voz é uma ferramenta bem legal. Então, vamos torná-lo público.

**Rodrigo:** Funciona da mesma forma que o repositório. Se for público, pessoas podem acessar os Gists que criamos. Se for privado, somente a pessoa responsável pelo arquivo pode compartilhá-lo com outras pessoas. O Gist privado não aparece na página do usuário no GitHub.

O Gist público foi criado! O endereço que está na barra de endereços do navegador é o que compartilharemos com outras pessoas. Vamos testar e verificar se está funcional mesmo.

Abriremos uma janela anônima, sem estarmos logados no GitHub, e colaremos a URL. Deste modo, será possível visualizar como a informação aparecerá para as pessoas.

O nome da pessoa que criou o Gist está sendo apresentado, juntamente com a descrição, "código de leitura de tela", além dos arquivos e dos trechos de código que compartilhamos em cada um dos arquivos.

Assim, Gabi, você pode compartilhar trechos de código instantaneamente, sem a necessidade de criar um repositório do zero, criar arquivos, fazer *commits*.

**Gabrielle:** Esse recurso é bastante interessante! Agora, compartilharei esse link com todas as pessoas. Também vou usar essa ferramenta para compartilhar todos os meus códigos interessantes que eu criar.

07

**Para saber mais: branches e merges**

Neste curso, citamos que no Git existem dois recursos chamados **branch** e **merge**. Entretanto, nesse curso de Git, tais assuntos não serão abordados.

Caso você queira se aprofundar nesses temas, temos aqui na Alura o curso de [Git e GitHub: dominando controle de versão de código](https://cursos.alura.com.br/course/git-github-dominando-controle-versao-codigo" \t "_blank) que te ensinará esses e outros assuntos.

Obs: É valido informar que para um melhor aproveitamento do curso, é necessário realizar antes os cursos indicados na sua lista de pré-requisitos.

09

**Faça como eu fiz: README do repositório**

Agora é com você! Crie o README do seu repositório, da mesma forma que foi demonstrado na aula.

**Opinião do instrutor**

Você deve criar um arquivo chamado **README.md** no diretório raiz do seu projeto, sendo que nesse arquivo você deve escrever os detalhes do seu projeto, utilizando a formatação Markdown e/ou HTML. Fique à vontade para escrever os detalhes que achar mais importantes e relevantes do seu projeto.

Você pode consultar exemplos e dicas nos artigos citados ao longo da aula:

* [Formatação com Markdown](https://wordpress.com/support/markdown-quick-reference)
* [Como escrever um README incrível no seu Github](https://www.alura.com.br/artigos/escrever-bom-readme)

10

**Desafio: hora da prática**

**Hora do Desafio**

Ao realizar esta sequência de **exercícios não obrigatórios**, você não apenas aprimora suas habilidades técnicas no Git, mas também desenvolve uma compreensão profunda de como otimizar seus repositórios para uma colaboração eficaz. Prepare-se para um mundo onde a organização do seu código é tão essencial quanto a sua funcionalidade!

**Desafios**

1. Criar um README.md para documentação de seu projeto
2. Crie um arquivo .txt com os logs de commits no seu projeto
3. Ignore o arquivo de logs durante a sincronização do repositório local para o remoto
4. Crie um README.md para o seu perfil do github

**Opinião do instrutor**

Vamos explorar passo a passo como realizar desafios essenciais para melhorar a documentação e organização do seu projeto no GitHub.

**Desafio 1: Criar um README.md para Documentação do Projeto**

* Na raiz do seu projeto, crie um arquivo chamado README.md. Você pode fazer isso manualmente ou usando um comando no terminal, dependendo do seu ambiente.
* Edite o README.md: Utilize o formato Markdown para estruturar e formatar seu README. Inclua informações como uma breve descrição do projeto, instruções de instalação, exemplos de uso, entre outras informações relevantes.
* Atualizar as alterações: Faça um commit das alterações no README.md e envie para o repositório remoto no GitHub.

**Desafio 2: Criar um Arquivo .txt com os Logs de Commits**

* Execute o Comando git log para ver os registros de commits
* No terminal, execute: git log > logs.txt para criar um arquivo chamado logs.txt contendo os logs de commits

**Desafio 3: Ignorar o Arquivo de Logs Durante a Sincronização**

* Criar um arquivo .gitignore:
* Se ainda não tiver, crie um arquivo chamado .gitignore na raiz do seu projeto.
* Adicionar a entrada para o Arquivo de Logs no .gitignore: Dentro do .gitignore, adicione uma linha com o nome do arquivo de logs (logs.txt). Isso garantirá que o Git ignore esse arquivo durante a sincronização.
* Commit e Push: Faça um commit das alterações no .gitignore e envie para o repositório remoto.

**Desafio 4: Criar um README.md para o seu Perfil do GitHub**

Este é um desafio diferente do anterior, pois é uma forma de criar uma apresentação para o seu perfil no Github, vamos realizá-lo?

Você também pode seguir as orientações no [artigo “Como criar um readme para seu perfil no github”](https://www.alura.com.br/artigos/como-criar-um-readme-para-seu-perfil-github)

* No GitHub, crie um repositório com o mesmo nome do seu nome de usuário.
* Dentro desse repositório, crie um arquivo README.md para documentar quem você é, suas habilidades, projetos favoritos, ou qualquer outra informação que deseje compartilhar.
* Personalizar o README: Utilize Markdown para formatar e personalizar seu README. Adicione links, imagens e qualquer coisa que represente você da melhor forma.
* Salvar as alterações com Commit e Push: Faça um commit das alterações no README.md do seu perfil e envie para o repositório remoto.

Ao concluir esses desafios, você terá melhorado a documentação do seu projeto e do seu perfil no GitHub, além de aprender a ignorar arquivos durante a sincronização. Essas práticas são fundamentais para tornar seus projetos mais acessíveis e seu perfil mais informativo. Continue explorando e aprimorando suas habilidades no GitHub!

13

**Referências**

**1.**[**Controlando versões com Git e GitHub**](https://www.casadocodigo.com.br/pages/sumario-git-github)

Neste livro, Alexandre Aquiles e Rodrigo Ferreira mostrarão como utilizar o Git para controlar as versões do seu projeto. Serão ensinados comandos para criação de repositórios, trabalho local e remoto, branches, tags, conflitos, dentre outros assuntos. Além disso, será mostrado também como criar uma conta no GitHub e utilizá-la para hospedar seus repositórios

**2.**[**Documentação git**](https://git-scm.com/book/pt-br/v2/Come%C3%A7ando-O-B%C3%A1sico-do-Git)

Documentação oficial do Git, onde você encontrará informações importantes sobre comandos e funcionamento da ferramenta.

**3.**[**Documentação github Conventional Commits**](https://www.conventionalcommits.org/pt-br/v1.0.0-beta.4)

A especificação do Conventional Commits é uma convenção simples para utilizar nas mensagens de commit. Ela define um conjunto de regras para criar um histórico de commit explícito, o que facilita a criação de ferramentas automatizadas. Esta convenção segue o SemVer, descrevendo os recursos, correções e modificações que quebram a compatibilidade nas mensagens de commit.

**4.**[**Ferramenta gerador de .gitignore**](https://www.toptal.com/developers/gitignore/)

Ferramenta para criar arquivos .gitignore úteis para o seu projeto.

**5.**[**Ferramenta MarkDown Guide**](https://www.markdownguide.org/)

O Markdown Guide é um guia de referência gratuito e de código aberto que explica como usar o Markdown, a linguagem de marcação simples e fácil de usar que você pode usar para formatar praticamente qualquer documento.

**6. [Markdown: como trabalhar com essa linguagem de markup?](https://www.alura.com.br/artigos/como-trabalhar-com-markdown?_gl=1*j8tztu*_ga*NzU2NTAyMDMyLjE2ODcxOTg5NTE.*_ga_1EPWSW3PCS*MTcwMTI4NjgzMy4yOTAuMS4xNzAxMjg5MzM1LjAuMC4w*_fplc*ZXM3TVQzbmI1bSUyQlhHeUZNY1h6VHFIdHZsc1BidGUwV1FWbFMzUm41VGxpSjk2dGl0M1o4OUVuUkdNMVJMN1p2MEtxWE1TV1ZsdFc0Y3A3cVdPYksxcWdUQllCb25CVDZpTGpIVENaakQ0QXlBRW5BbDQ1VTBkRU1NY00wUnclM0QlM0Q" \t "_blank)**

Você irá entender sobre o funcionamento do Markdown, que é um formato de marcação de texto e como fazer o uso adequado dessa ferramenta.

**7. [Introducing Github O’Reilly](https://www.oreilly.com/library/view/introducing-github/9781491949801/" \t "_blank)**

Livro em **Inglês** para você, que é novo no GitHub. Este livro conciso mostra exatamente o que você precisa para começar e nada mais. É perfeito para gerentes de projetos e produtos e outros membros da equipe que desejam colaborar em um projeto de desenvolvimento, seja para revisar e comentar o trabalho em andamento ou para contribuir com mudanças específicas. Também é ótimo para desenvolvedores que estão aprendendo o GitHub.

**8.**[**Learning Git O'Reilly**](https://www.oreilly.com/library/view/learning-git/9781098133900/?_gl=1*1k0j59z*_ga*MTc2ODczMTk1MS4xNjk4NDI5NjAw*_ga_092EL089CH*MTcwMTcxODU5OC4yLjEuMTcwMTcxODc3NS41My4wLjA.)

Este livro,**em Inglês**, ensina Git de maneira simples, visual e tangível para que você possa construir um modelo mental sólido de como funciona o controle de versão do Git. Através do uso de cores, narrativas e exercícios práticos, você aprenderá a usar essa ferramenta com confiança.

As informações são introduzidas de forma incremental para que você não fique atolado em termos ou conceitos desconhecidos. Aprender Git é ideal para quem precisa usar Git para projetos pessoais ou profissionais: estudantes de bootcamp de codificação, desenvolvedores juniores, profissionais de dados e escritores técnicos, para citar apenas alguns!