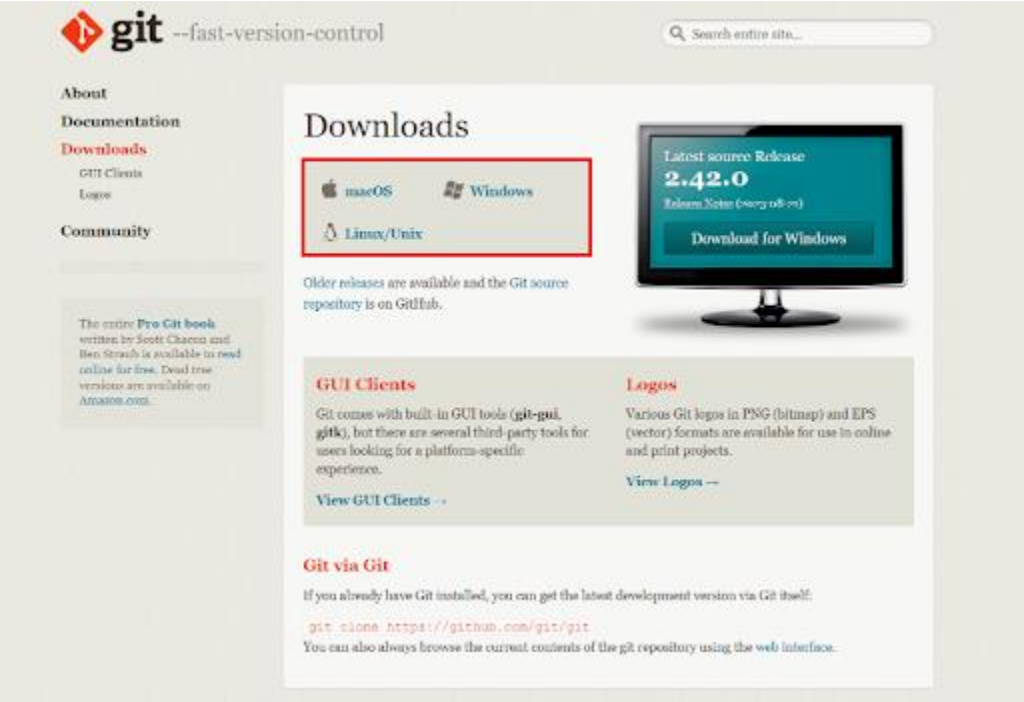
Roteiro 1 –1

Instalando o Git

Para realizar essas atividades, você precisa ter instalado o Git em seu computador. Caso ainda não tenha feito isso, faça o download e a instalação conforme mostramos a seguir:

Para instalar o Git em seu computador, você deve acessar o site de downloads do Git. Em seguida, clique no botão referente ao seu sistema operacional.



Windows

Caso seu sistema operacional seja o Windows, no site do Git será aberta uma lista com várias opções de download.

Se o seu computador tiver a arquitetura de 64bits, que é a mais comum, clique na opção 64-bit Git for Windows Setup. Caso seu computador tenha a arquitetura de 32bits, clique na opção 32-bit Git for Windows Setup.

Assim, baixamos um arquivo com a extensão .exe.

Você deve clicar no arquivo baixado para iniciar a instalação. Provavelmente uma janela será aberta perguntando se deseja permitir que esse programa faça alterações em seu computador; caso esse cenário ocorra, clique em Sim para continuar.

Na janela de instalação, prossiga clicando nos botões de Next. Neste momento, não é necessário realizar nenhuma alteração nas configurações padrão da instalação do Git.

Após uma série de cliques no botão Next, você deve clicar no botão Install, para instalar o Git. Por fim, clique no botão Finish. Assim, o Git estará instalado em seu computador.

Para confirmar a instalação digite no seu command (cmd)

git --version

Distribuições Linux

Caso seu sistema operacional seja alguma distribuição Linux, será aberta uma página com as instruções de instalação por meio do gerenciador de pacotes do seu sistema.

Basta seguir os comandos especificados pelo seu Terminal que o Git será instalado em seu computador.

Roteiro 1 - 2

Criando sua conta no GitHub

Para realizar esse curso, você precisa ter uma conta no GitHub. Caso ainda não tenha, acesse o site do GitHub e clique no botão Sign up.

Isso abrirá um formulário a ser preenchido com suas informações. Também haverá um processo de verificação, para garantir que você não é um robô. :)

Criada a conta, o GitHub enviará um código para o seu e-mail. Verifique seu e-mail, inclusive as abas de “Promoções”, “Social” e “Spam”. Após encontrá-lo, informe o código enviado.

Assim, sua conta no GitHub estará criada e pronta para uso.

Configurando seu usuário do GitHub localmente

Por fim, você precisará configurar seu usuário do GitHub em seu computador. Para isso, abra seu Terminal e execute os seguintes comandos:

git config --global user.name "SEU NOME"
git config --global user.email EMAIL@exemplo.br
Copiar código

Lembre-se de substituir “SEU NOME” pelo seu nome e “EMAIL@exemplo.br” pelo e-mail que você usou na sua conta do GitHub.

Roteiro 1 - 3

Enviando o código para o GitHub

Abrimos o navegador, acessamos a página do GitHub e, para isso, precisamos criar uma conta e autenticar nela.

Criando um repositório

Precisamos criar um repositório. Para isso, clicamos no ícone no canto superior direito, selecionamos "Your Repositories" ("Seus repositórios") e clicamos no botão verde "New" ("Novo") do lado direito.

Isso abrirá opções para a criação do novo repositório em uma página intitulada "Create a new repository" ("Crie um novo repositório").

O campo "Owner" ("Proprietário") indica quem é o dono do repositório, e o campo "Repository Name" é o nome do repositório. Chamaremos o repositório de "AvaliaçãoA3".

As demais configurações para o repositório podem ser mantidas como estão. Clicamos no botão verde "Create Repository" ("Criar repositório") no canto inferior direito para criar o repositório remoto, sendo o que está na web.

O GitHub oferece algumas orientações sobre como podemos enviar o código de nossa máquina para esse repositório remoto.

Iniciando um repositório Git no projeto local

Agora, retornemos ao CMD, navegaremos até o local da nossa pasta de trabalho.

Ex: ... Projeto/AvaliaçãoA3

Precisamos iniciar um repositório Git em nosso projeto local. Para isso, executamos o comando:

c: \ Projeto\AvaliaçãoA3> git init

Esse comando inicializa o repositório Git localmente, dentro da pasta do nosso projeto, criando uma pasta oculta chamada .git.

Definindo branch

Em seguida, definimos um branch com o comando git branch -M main.

```
git branch -M main
```

Conectando o repositório local com o do GitHub

Agora, vamos conectar o repositório que criamos no GitHub com o nosso repositório local em nosso computador. Voltamos ao navegador, para o repositório que criamos e copiamos a URL fornecida no campo "Quick Setup" ("Configuração rápida").

A URL a seguir pertence à instrutora; cada pessoa terá a sua própria. Estamos usando essa apena

De volta ao terminal, executamos o comando

```
git remote add.
```

Precisamos dar um nome a esse repositório remoto, geralmente chamamos de origin. Informamos a URL do repositório.

```
git remote add origin https://github.com/Gabrielle-Ribeiro/allbooks.git
```

Com isso, adicionamos o repositório remoto.

Rodamos o comando cls para limpar a tela e depois git remote.

Conclusão

Estabelecemos a conexão entre o computador e o repositório do GitHub. No entanto, ao atualizar a página do projeto no GitHub clicando no ícone “↻” localizado no canto superior esquerdo, notamos que nada foi alterado; o código ainda não foi enviado. A seguir, você aprenderá como enviar o código do seu computador para o GitHub.

Roteiro 1 - 4

Adicionando, Registrando e Enviando Alterações no GitHub

Abrimos o terminal e executamos alguns comandos. Ao rodarmos o comando git status, poderemos visualizar os arquivos que foram modificados ou adicionados em nosso projeto e ainda não foram incluídos no GitHub - eles serão destacados em vermelho.

```
git status
```

O retorno abaixo foi parcialmente transcrito. Para conferi-lo na íntegra, execute o código na sua máquina

```
Untracked files:

(use "git add `<file>`..." to include in what will be committed)

.gitignore

README.md

package-lock.json

package.json

src/
```

Adicionando arquivos

Em seguida, utilizaremos o comando git add para especificar quais arquivos desejamos adicionar.

```
git add  “nome do artquivo”
```

Podemos simplificar o processo de adicionar arquivos ao nosso projeto. Em vez de digitar manualmente o nome de cada arquivo (por exemplo, git add README.md e teclar "Enter"), podemos usar um atalho chamado git add . .

```
git add .
```

Esse comando adiciona todos os arquivos de uma vez. Ao rodarmos novamente o comando git status, veremos a lista de todos os arquivos que foram adicionados.

Criando um commit

Agora, criaremos um commit para registrar as mudanças realizadas. Utilizamos o comando git commit, incluindo a opção "-m" para adicionar uma mensagem sobre as alterações feitas no projeto.

git commit –m

Após o espaço, colocamos aspas duplas e escrevemos uma breve descrição das modificações. Como mensagem, optamos por "Adiciona projeto inicial". Fechamos as aspas e pressionamos "Enter".

```
git commit -m "Adiciona projeto inicial"
```

O comando de commit registra a alteração realizada. Podemos limpar a tela de novo com cls.

Podemos executar o comando git log para visualizar o registro das alterações feitas em nosso projeto.

git log

Como retorno, temos:

```
commit 2019f9827c7412e3234031491289f35956ee6c25 (HEAD -> main)
```

Author: (seu nome)

Date: Mon Dec 18 17:07:48 2023 -0300

Adiciona projeto inicial

No entanto, é importante notar que essas modificações estão atualmente disponíveis apenas em nosso computador e ainda não foram enviadas para o GitHub.

Enviando para o GitHub

O último passo é utilizar o comando git push, especificando o destino como o nosso origin main.

git push origin main

Verificando no GitHub

No navegador, acessamos nosso projeto no GitHub, atualizamos a página e todos os arquivos presentes em nosso computador também estão disponíveis no GitHub. Observem que temos a branch main na parte superior esquerda, o nome da pessoa proprietária abaixo e os arquivos na sequência.

Compartilhando o projeto

Se desejarmos compartilhar o projeto, basta fornecer a URL que consta no endereço da página na parte superior, permitindo que outras pessoas visualizem e colaborem no código.

Conclusão

É evidente que no GitHub, o trabalho colaborativo desempenha um papel significativo. Quando várias pessoas trabalham simultaneamente, fazendo alterações no mesmo código, é possível que uma modificação feita por alguém não seja automaticamente refletida em nosso computador.

Como podemos incorporar as alterações feitas por outros colaboradores de volta ao nosso projeto local? Abordaremos isso em seguida.

Roteiro 1 - 4

Integrando Modificações no Projeto

Abrimos nosso projeto no GitHub e simulamos a alteração que alguém teria feito.

Nos arquivos, selecionamos o README.mde efetuamos a modificação diretamente no GitHub. Acima do arquivo, encontramos várias opções e no canto direito, identificamos um botão com um ícone de lápis para editar o arquivo (ao colocarmos o mouse por cima obtemos a mensagem "Edit this file"). Clicamos nele e, antes de explicar o que é o projeto, adicionamos um texto de "Boas-vindas".

Conclua fazendo o commit da alteração.

Retornando à pasta raiz do projeto no GitHub, clicando em "Code" no menu superior esquerdo. Assim, retornamos ao início do projeto no GitHub e abaixo do botão verde "Code" do lado direito, encontramos os dois commits, indicando a quantidade de modificações realizadas.

Podemos clicar nele para visualizar o histórico de commits, assim como fazemos no terminal, acompanhando as alterações realizadas no projeto ao longo do tempo.

O commit mais recente está no topo, correspondendo à atualização no README. Ao retornarmos ao terminal e executamos o comando git log.

commit 2019f9827c7412e3234031491289f35956ee6c25 (HEAD -> main, origin/main)

Author: Gabrielle Ribeiro gabrielleribeiro2010@gmail.com

Date: Mon Dec 18 17:07:48 2023-0300

Adiciona projeto inicial

Enquanto no GitHub temos dois commits, no terminal temos apenas um. É necessário sincronizar essas alterações em nossa máquina, utilizando o comando git pull origin main.

git pull origin main

Este comando traz a mensagem, informando que o arquivo README foi modificado e podemos rodar novamente o comando git log. Agora podemos verificar que temos os dois commits no nosso histórico.

Roteiro 2-1

Até o momento, nós estamos trabalhando somente com a branch "main", mas ainda não compreendemos muito bem o que branch significa. O sistema de versionamento Git nos apresenta essa ideia de branches que, traduzindo, significa ramificações. Isso permite que nós criemos várias linhas de desenvolvimento.

Esse recurso é útil quando trabalhamos com um projeto mais robusto, onde teremos várias funcionalidades e adições que são feitas ao longo desse projeto. E é importante que novas alterações que fazemos nesse projeto sejam feitas de forma isolada, porque na nossa branch principal, a chamada main, normalmente temos um código que já estará pronto para a produção. Isso significa que ele já estará pronto para ser usado pelo cliente. Portanto, não podemos alterar aquele código sem ter a garantia de que isso não causará nenhum problema.

As branches fornecem para nós esse recurso, que permite que criemos diversas linhas de desenvolvimento, sem que uma afete o que é feito na outra.

Abrindo o Terminal, podemos rodar o comando git branch. Esse comando mostra para nós quais são todas as ramificações presentes no nosso projeto. No nosso caso, só temos a main, que definimos no início.

Agora, imagine a seguinte situação: queremos adicionar a documentação da nossa aplicação, e não vamos adicionar isso diretamente na branch principal. Criaremos uma nova branch para adicionar essa parte do projeto.

Para isso, ainda no Terminal usamos o comando git checkout -b e passamos o nome da nossa branch

```
git checkout -b Documentação_Requisitos
```

Aparecerá uma mensagem dizendo "Switched to a new branch “Documentação Requisitos" (Mudou para a nova branch ‘Documentação_Requisitos’).
Rode novamente o comando *git branch* e vai listar todas as branchs criadas.

Agora vamos pegar o arquivo de escritas dos requisitos e vamos colar na página do projeto da A3 criada na Roteiro 1.

Agora podemos subir essa nova branch para o nosso GitHub. Então, vamos rodar aqueles mesmos comandos que já estamos acostumados. Começando com o *git status*, que mostra que existem esses dois novos arquivos. Depois rodamos o nosso *git add .*, seguido do git status novamente para ver se deu certo. Recebemos a mensagem de commit com os arquivos de cor diferente, portanto, deu certo.

Agora, criaremos o nosso commit com o comando git commit -m "Adicionado documento", passando o texto entre aspas, para adicionar o texto do commit. Por fim,

podemos rodar o comando *git push origin* Documentação_Requisitos. Assim, ao invés de mandarmos para o origin main, mandamos para uma nova branch que acabamos de criar.

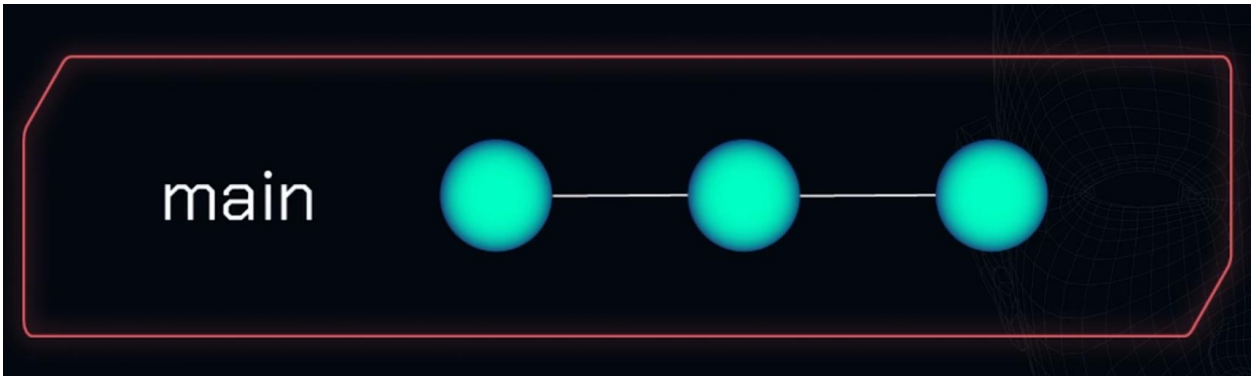
Agora podemos voltar para o nosso navegador e abrir o nosso projeto no GitHub.

Podemos atualizar a página e, na parte superior, aparece uma mensagem falando que essa branch " Documentação Requisitos " teve um push recente alguns segundos atrás.

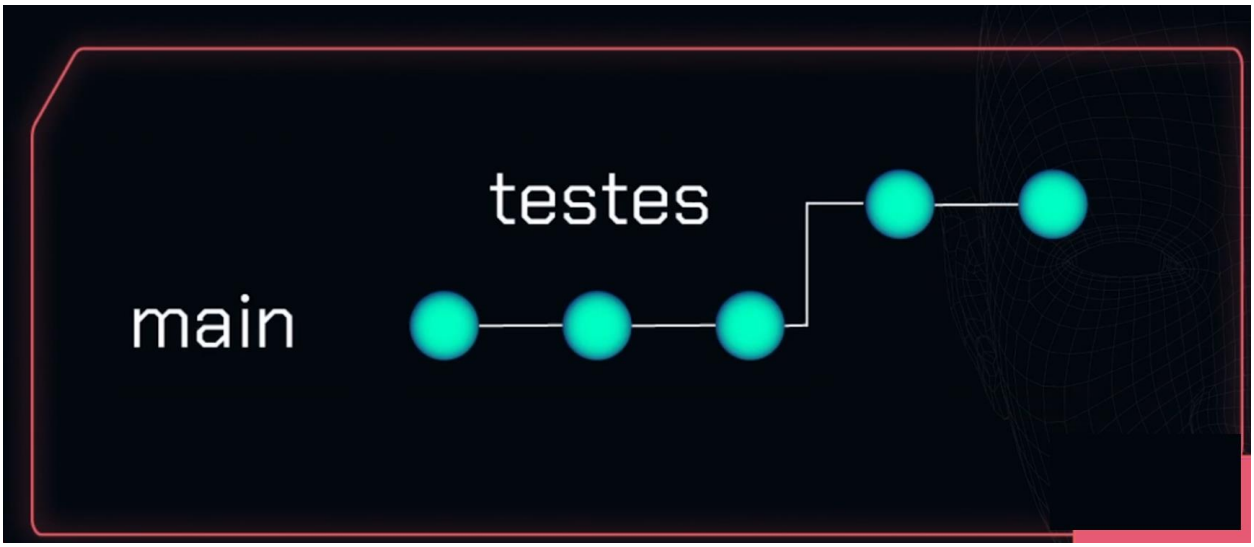
Roteiro 2-2

Entendendo o funcionamento das branches

Inicialmente, temos a branch main, a nossa branch principal. Na imagem abaixo, cada círculo ao lado da main representa um commit que fizemos no repositório. Trabalhamos na main, adicionando commits e, no momento, decidimos criar uma nova branch para o documento, resultando em uma nova ramificação do projeto.



Agora, temos duas linhas de desenvolvimento independentes uma da outra. Isso significa que tudo o que fizermos na main não vai afetar a branch Documentação_Requisitos e todas as mudanças que adicionarmos em Documentação_Requisitos não afetará diretamente o código que está salvo na main. Um detalhe interessante de notar é que a branch Documentação_Requisitos, como foi criada a partir da main, trará tudo que veio anteriormente da main em seu histórico. E, caso achemos necessário, podemos criar uma nova branch para adicionar uma nova funcionalidade no projeto. Podemos criar quantas ramificações quisermos no nosso projeto, conforme acharmos necessário.



Quais são as vantagens de trabalhar com Git e GitHub utilizando as branches?

Roteiro 2-3

Mesclando códigos de duas branches

Abriremos nosso terminal e faremos essa integração do código da branch de Documentação_Requisitos para a branch "main". Esse processo é chamado de merge (mesclagem), que é a mesclagem do código que está em algum lugar, em alguma branch, para uma branch de destino.

No terminal, rodamos o comando git branch. Como ele, percebemos que estamos na branch de documentos e precisaremos mudar para a branch que queremos que receba o código. Então, vamos rodar o comando *git checkout main* para mudarmos para a branch "main". Depois limpamos o terminal, com o comando cls, e precisamos rodar o comando para realizar esse merge, que é o comando *git merge nome-da-branch-origem,* ou seja, passamos o nome da branch de onde vamos trazer esse código, no caso, " Documentação Requisitos ".

git merge Documentação_Requisitos

Realizamos nosso merge, apareceu a mensagem informando que trouxemos aqueles commits e o código que adicionamos.

Rode o comando de log e veja o que aconteceu

Vamos voltar ao terminal para fazermos o push dessas alterações, subindo essa modificação para o GitHub. Para isso, vamos rodar o comando git push origin main. Subimos esse commit e podemos abrir nosso projeto na página do GitHub.

Atualizamos a página e trocamos para a branch "main". Portanto, clicamos em " Documentação_Requisitos ", na parte superior esquerda da lista de arquivos, para abrir a lista de branches, e depois selecionamos "main".

Conclusão

Através do merge, integramos o código de uma branch a outra branch. Vamos entender visualmente o que aconteceu: