Versionamento de código com Git e Git Hub

Introduzir ao versionamento de código com Git e Git hub

Conhecer as ferramentas Instalar, configurar e Autenticar, Primeiros Passos com Git e Git Hub Dicas e Materiais de Apoio

Pré-requisitos

* Computador com acesso à internet
* Vontade de aprender

Percurso

Visão geral do curso e ferramentas

Instalação, configuração e Autenticação

Primeiros passos com Git e Git Hub

Dicas de Materiais de Apoio

O que é versionamento de código

Versionamento é em programação, que a cada alteração/versão de código ser gerado uma nova versão.

Sistemas de Controle de versão

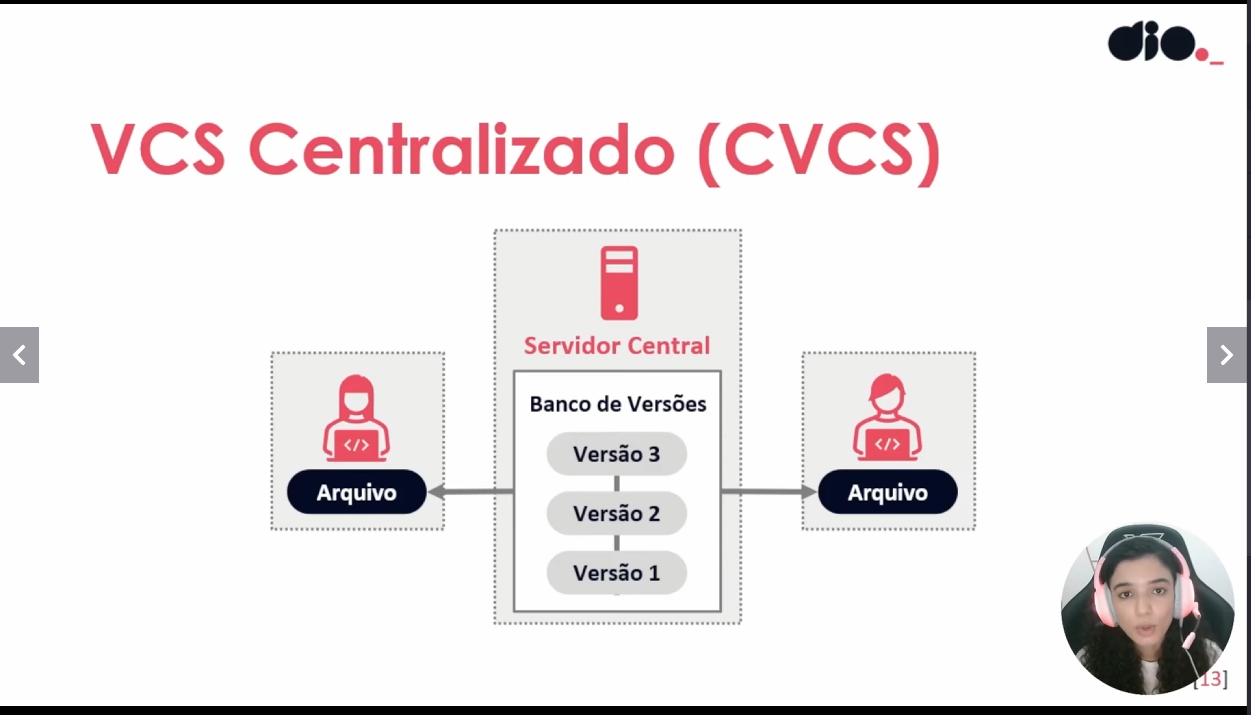
Controlam as versões de um arquivo ao longo do tempo.

* Registra o histórico de atualizações de um arquivo;
* Gerencia quais foram as alterações, a data, autor, etc;
* Organização, controle e segurança;

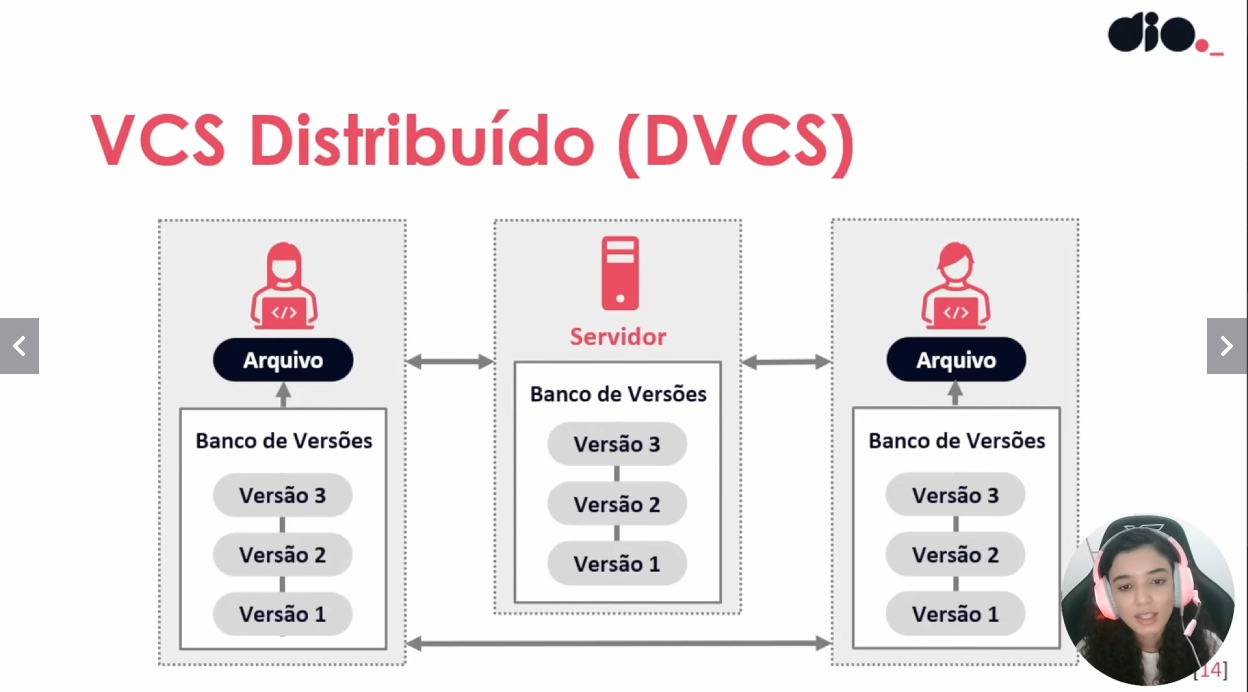
Tipos de Sistemas de Controle de versão

Dentre os sistemas de controle de versão (VCS), temos:

* VCS Centralizado (CVCS); -> CVS, Subversion.
* VCS Distribuídos (DVCS); - > Git, Mercurial



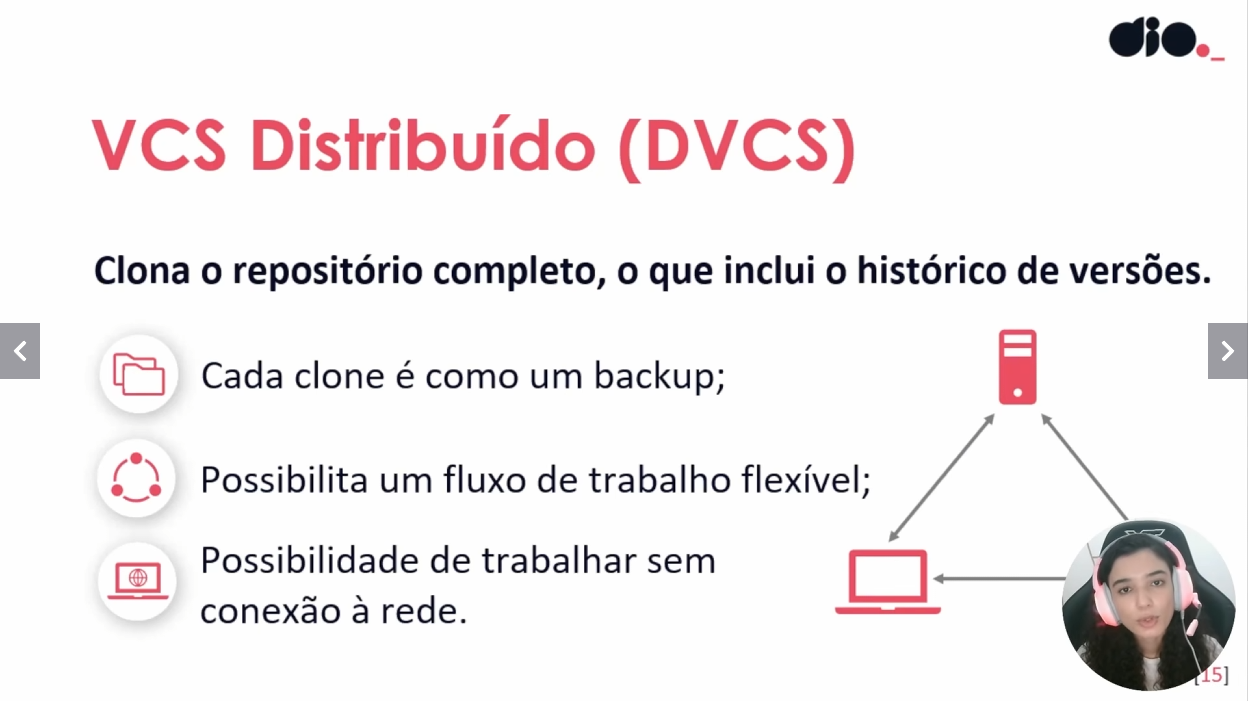
Nesse CVCS, esse modelo é caracterizado por conter apenas um servidor que conterá todos os arquivos responsáveis pelo controle de versão, e sua principal desvantagem, em caso de queda ou problemas com o servidor central você fica impossibilitado de salvar ou alterar alguma versão, e para casos de corrompimento de arquivos ou perda de dados, você perdera o projeto e para solucionar isso sugiram então os VCS Distribuídos (DVCS)



Nesse modelo se caracteriza principalmente que nesse modelo cada repositório ou cada “banco de versão”, sendo duplicado localmente, ou seja, os participantes possuem uma cópia do que está no servidor principal.

O VCS Distribuídos (DVCS), clona o repositório completo, o que inclui o histórico de versões.

* Cada clone é como um backup;
* Possibilita um fluxo de trabalho flexível;
* Possibilidade de trabalhar sem conexão à rede;



O que é Git

Sistema de controle de versão Distribuído.

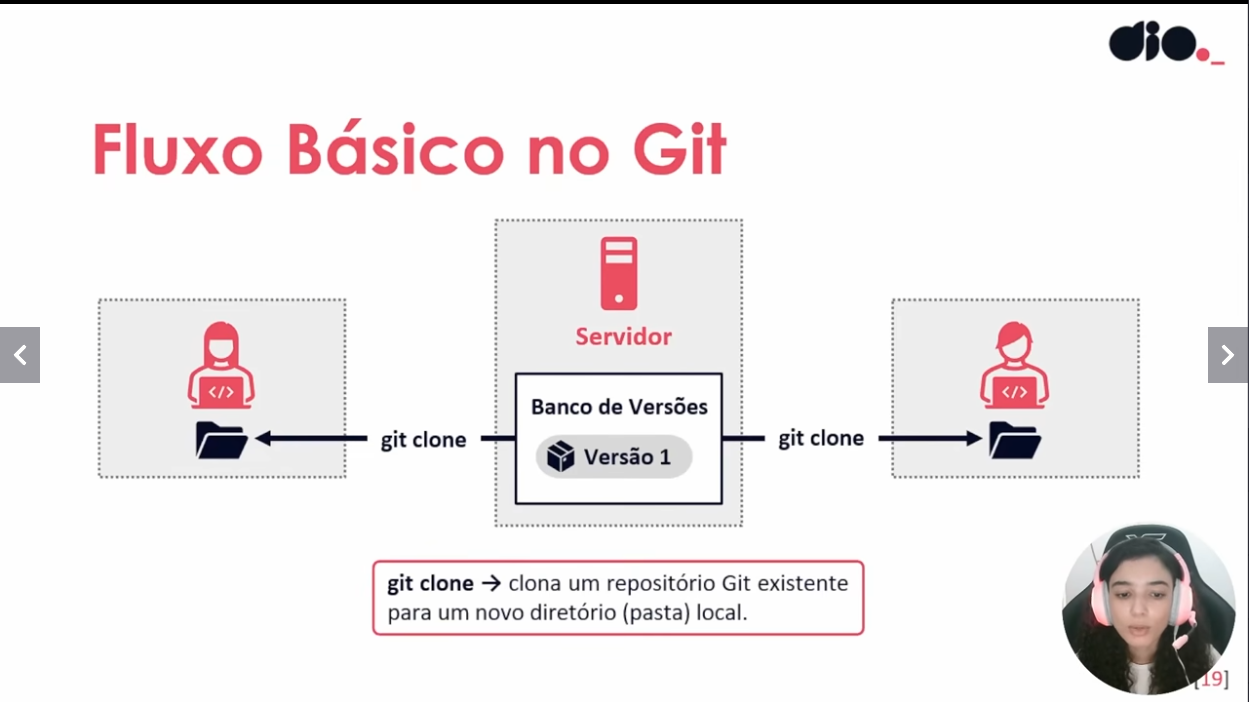
* Gratuito e Open Source (Código Aberto);
* Ramificação (Branching ) e fusões (merging) eficientes);
* Leve e rápido

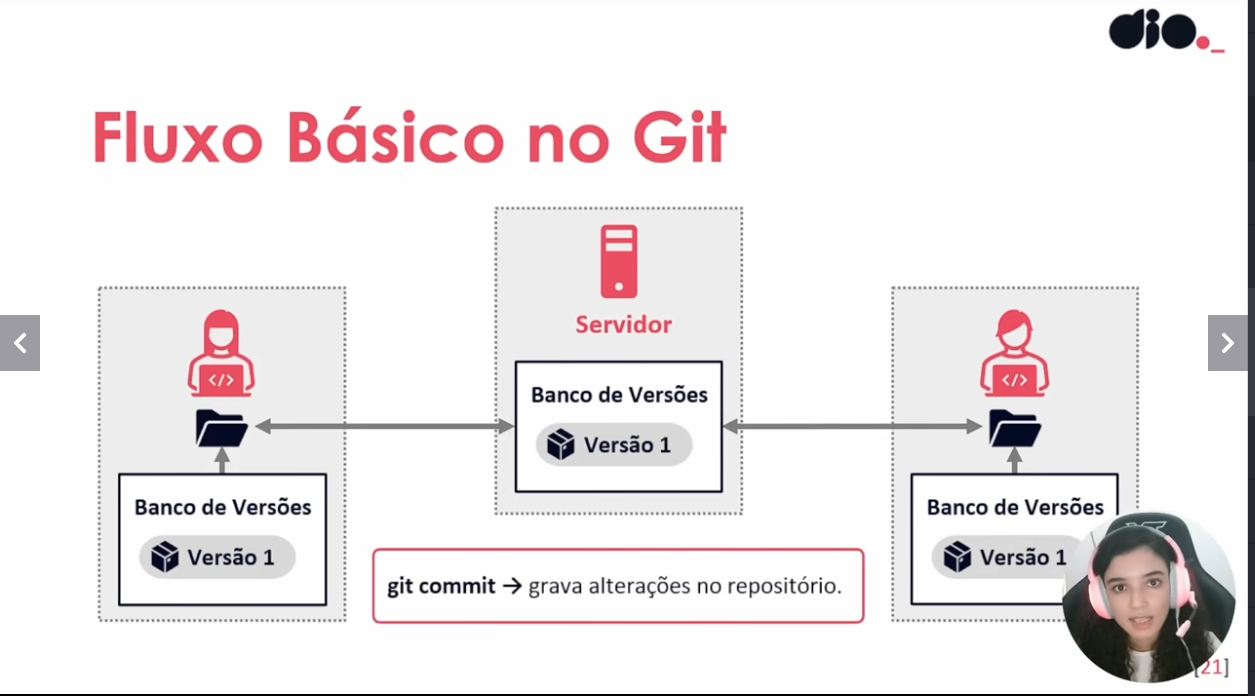
<https://git-scm.com/>

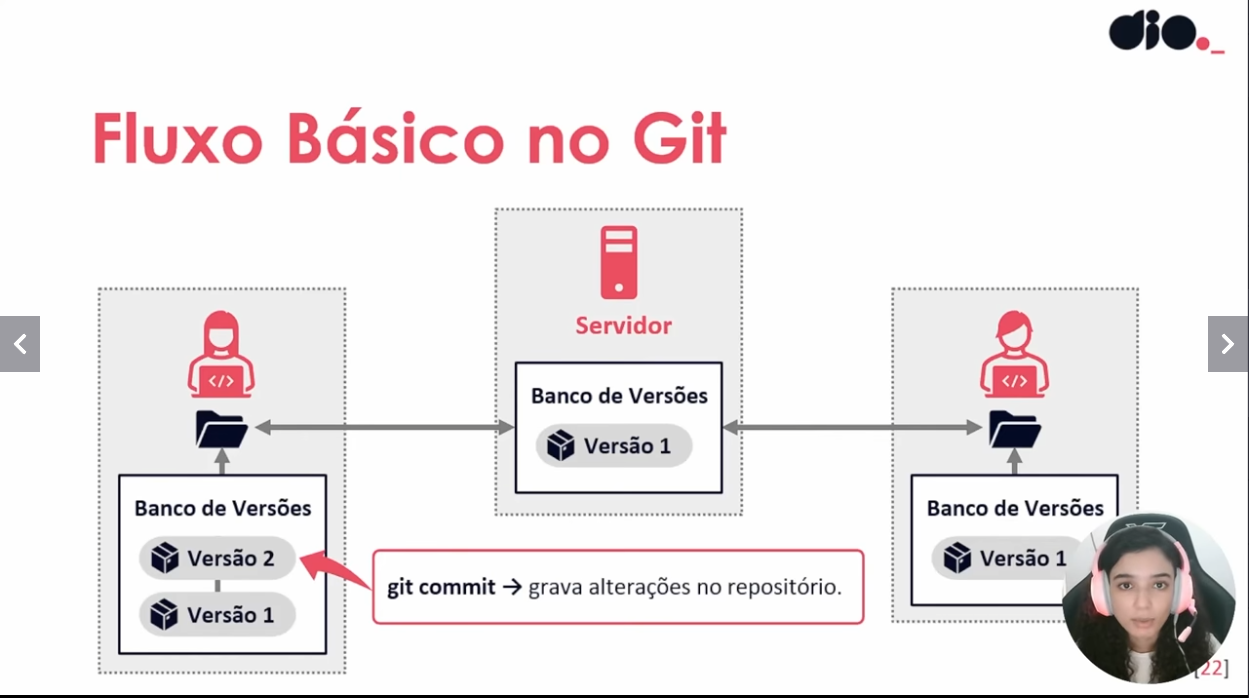
Breve histórico do Git

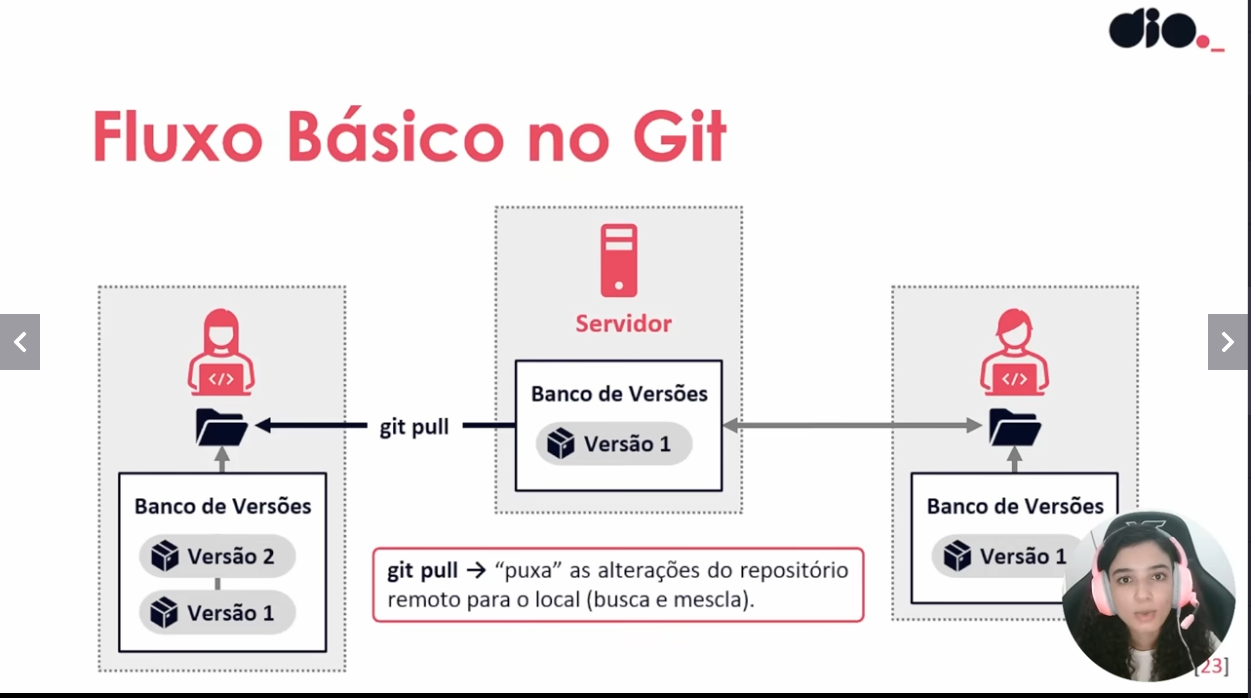
2002 -> O projeto do núcleo (Kernel) do Linux, que é open source, começa a utilizar o bitkeeper, um DVCS proprietário;

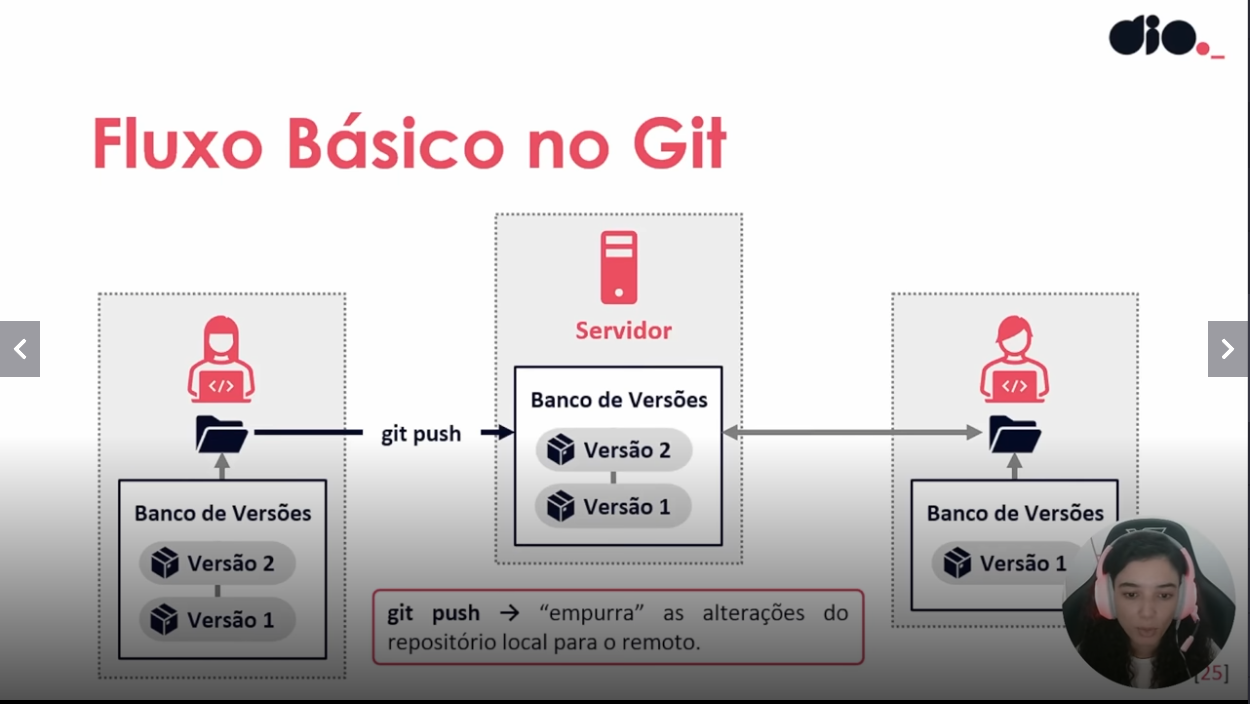
2005 -> Após conflitos com a comunidade, o BitKeeper recinde a licença gratuita. O que leva a Linux Torvalds, o criado do Linux, e sua equipe desenvolverem sua própria ferramenta, o Git.











O que é GitHub?

Plataforma de hospedagem de código para controle de versão com Git, e colaboração

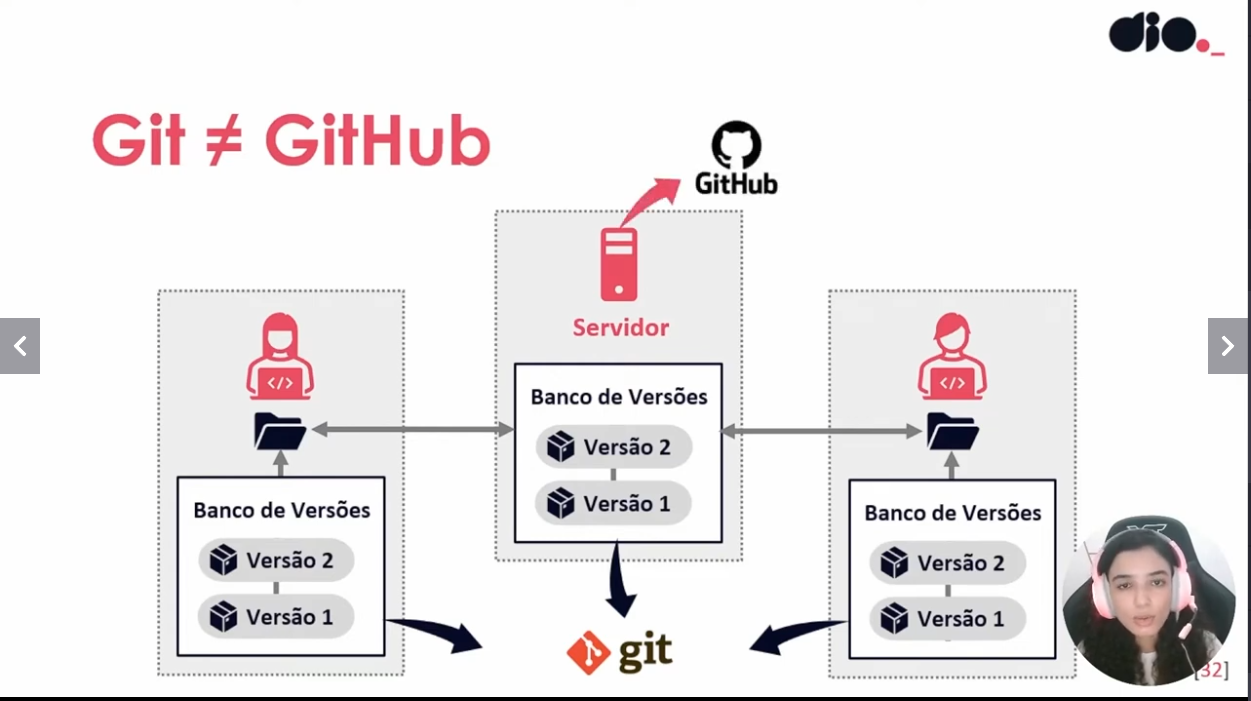
* Comunidade ativa;
* Utilizado mundialmente;
* Mascote “Octocat”.



Breve Histórico do GitHub

2008 -> Desenvolvido por Chris Wanstrath, J. Hyett, Tom preston-Werner e Scott Chacon.

2018 -> Vítima de um dos maiores ataques de DDoS (Ataque distribuído de negação de serviço); comprado pela Microsoft Corporation por US $ 7,5 Bilhões.



Autenticando via Token

Após as configurações padrões do git e github caso deseje realizar a configuração via token que foi gerado no git hub

Siga os seguintes passos

1 gerar o token no git hub

2 inserir o usuário do git hub e email

3 inserir o token que foi gerado

4 dentro do git bash e ou terminal git inserir os seguintes comandos

git config credential.helper cache (para esse caso e a configuração para que essa credencial não fique salva)

git config credential.helper store (para esse caso e para salvar a credencial gerada com a adição do local (global, system ou local ), ele guarda aquelas configurações.)

Criando e clonando repositórios

Parar clonar um repositório, porém modificando o seu nome o comando é

Git clone url e nome da pasta que eu quero que salve

Outro comando para ser utilizado e git remote -v para visualizar os repositórios remotos que esta vinculado

Para vincular diretamente a um repositório romote e fazer

Git remote add origin e a url / chave ssh

Git clone url –branch nome da branch –single-branch esse comando pode ser utilizado para realizar o clone de uma Branch em especifico do repositório remoto que está sendo clonado

Para realizar que o git ignore uma determinada pasta pode-se criar dentro do arquivo. gitignore o nome da pasta precedido pelo /

Caso seja um tipo de extensão em especifico, pode-se visualizar na doc do git

.gitkeep esse arquivo pode ser utilizado como contenção para que um pasta sem arquivos seja salva.

Desfazendo alterações no repositório Local

Caso tenha inicializado uma pasta como repositória git só excluir a pasta git no terminal o comando é:

rm -rf .git

Para que possa realizar a restauração de um arquivo em especifico que foi modificado apagado o conteúdo etc... pode-se utilizar o seguinte comando

git restore e o nome do arquivo assim o git irá restaurar o arquivo de acordo com o ultimo commit.

Para modificar a mensagem que é utilizada em conjunto do commit, pode-se realizar os seguintes comandos git commit –amend -m “adiiona a nova mensagem para o commit.”, caso não adicione o comando -m o git irá realizar a abertura do editor e você o pode editar ali a mensagem

Para desfazer algum commit você deve utilizar alguns dos comandos listados abaixo

git reset –soft hash do commit (Nesse comando em questão, pega os arquivos nos commit que estão posteriores a ao escolhido pela hash e os coloca na área de preparação para que seja commitado novamente, ou seja ele não apaga os arquivos do commit apenas retornas os arquivos ou o commit feito, para que ele possa ser editado e adicionado novamente ao repositorio)

git reset –mixed hash (Como padrão o comando mixed é o padrão realizado pelo git caso não seja adicionado esse argumento o git irá entender que é um git reset –mixed, diferentemente do que é feito no –soft ele não os adiciona a lista de adicionado e sim de não conhecidos no commit, ou seja arquivos e pastas não rastreadas e adicionadas ao repositório)

git reset –hard (Nesse comando em questão ele retorna ao status seja de arquivos ou códigos escritos para o commit selecionado)

Outro comando útil para visualizar quais alterações foram feitas no repositório pode-se utilizar o comando git reflog, esse comando demonstrar todos os comandos que foram realizados, incluindo os git reset, para que um arquivo ou pasta com arquivo não seja adicionado a área de preparação (pre commit), deve-se utilizar o comando git reset e o caminho do arquivo que está no git status, outra opção é

git restore –staged e o caminho desejado

Outro dica importante caso deseje realizar a edição de um arquivo de forma online você pode realizar a edição através do atalho de teclado com a tecla . (ponto), que irá abrir um vscode online

Trabalhando com Branches

De uma maneira simplista, uma Branch (em tradução livre, “ramo”), é uma ramificação do seu projeto.

* É um ponteiro móvel para um commit no histórico do repositório.
* Quando você cria uma nova Branch a partir de outra existente, a nova se inicia apontando para o mesmo commit da Branch que estava quando foi criada.

Em resumo, uma branch pode ser encarada em outras palavras como um universo paralelo da sua branch principal, uma branch criada funciona como um ponteiro pera um determinado commit, fazendo assim alterações naquela branch sem alterar a sua branch ou versão principal do código.

Os comando básicos e funcionais em um cenário ideal são os litados abaixo:

Para realizar a adição de uma nova branch utiliza-se o comando abaixo

git checkout -b nome da nova branch

Para visulizar as branchs do repositório utilzia-se o comando abaixo:

git branch -v

para retornar ou acessar alguma outra branch desejada, utiliza-se o comando descrito abaixo:

branch checkout nome da branch desejada

Para realizar a união dos arquivos/commits realizados na branch criada com a branch principal, utiliza-se o comando abaixo:

git merge nome da branch

Para visualizar qual branch se encontra no momento pode-se utilizar o comando abaixo:

git branch

Para deletar uma branch utiliza-se o comando abaixo:

git branch -d nome da branch

Nos exemplos acima funcionam em caso de não haver conflitos nem alterações concorrentes entre as branchs, em cenários onde se utiliza o repositório remoto, ao se deparar com conflito de commits, deve-se decidir qual será o commit a ser seguido. Para tal deve-se seguir os seguintes passos:

1º Realizar o git pull para sincronizar o repositório local com o repositório remoto.

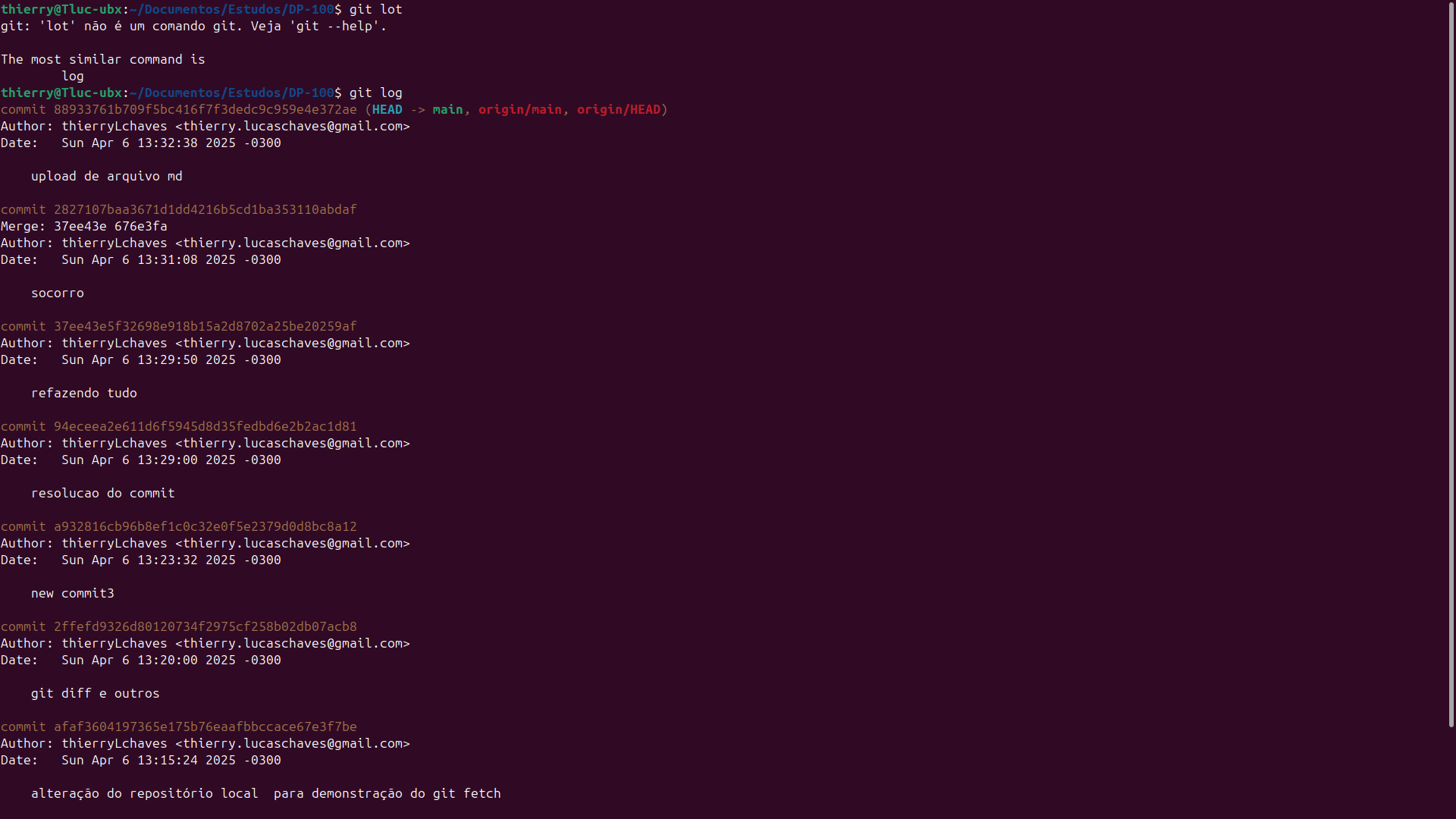
2º Pode-se utilizar o comando git status e observe quais as diferenças entre os arquivos e solucione-as

3º Pode-se também utilizar o comando git diff nome da branch selecionada, qual branch remota será comparada com a branch local ex:

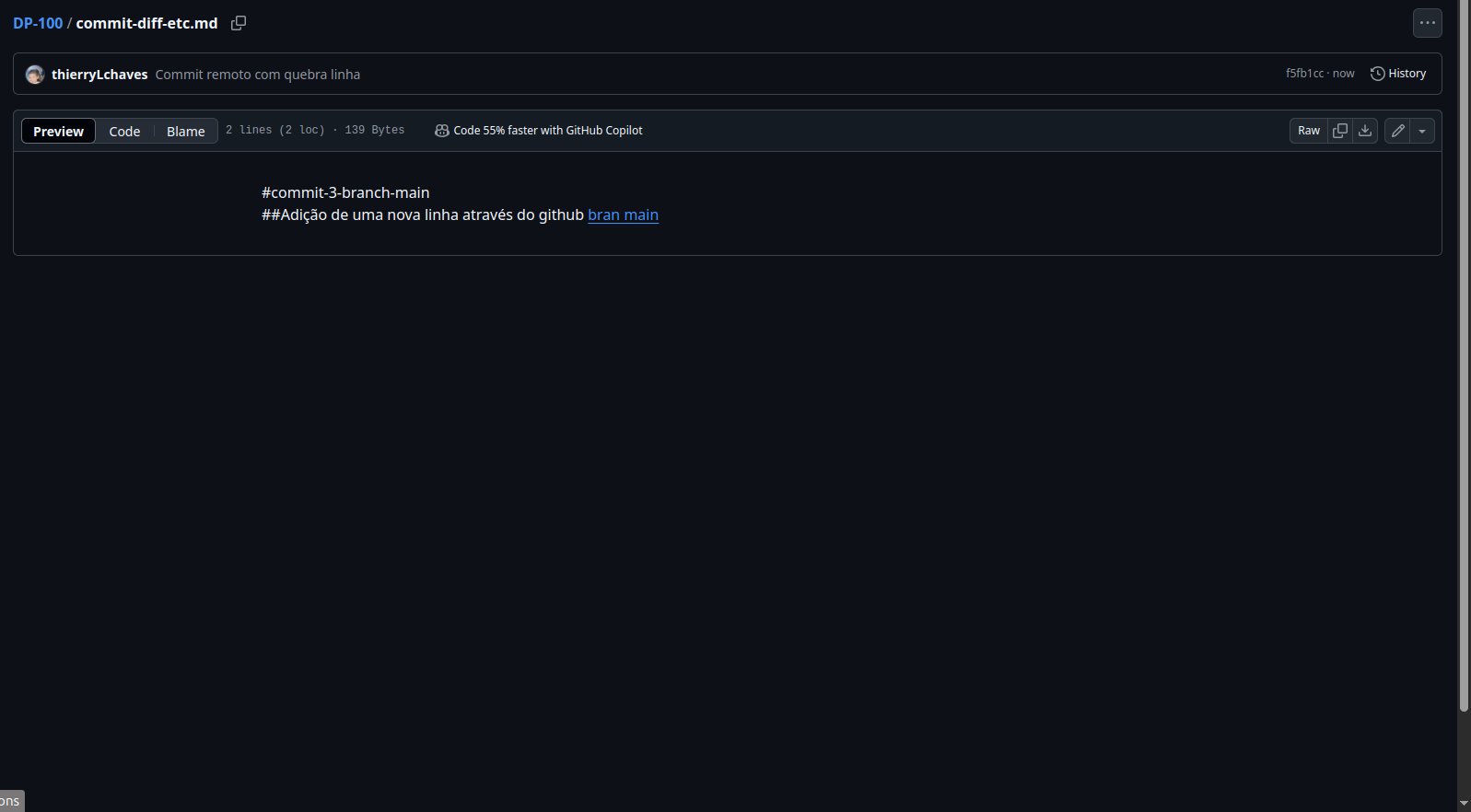
git diff main origin/main

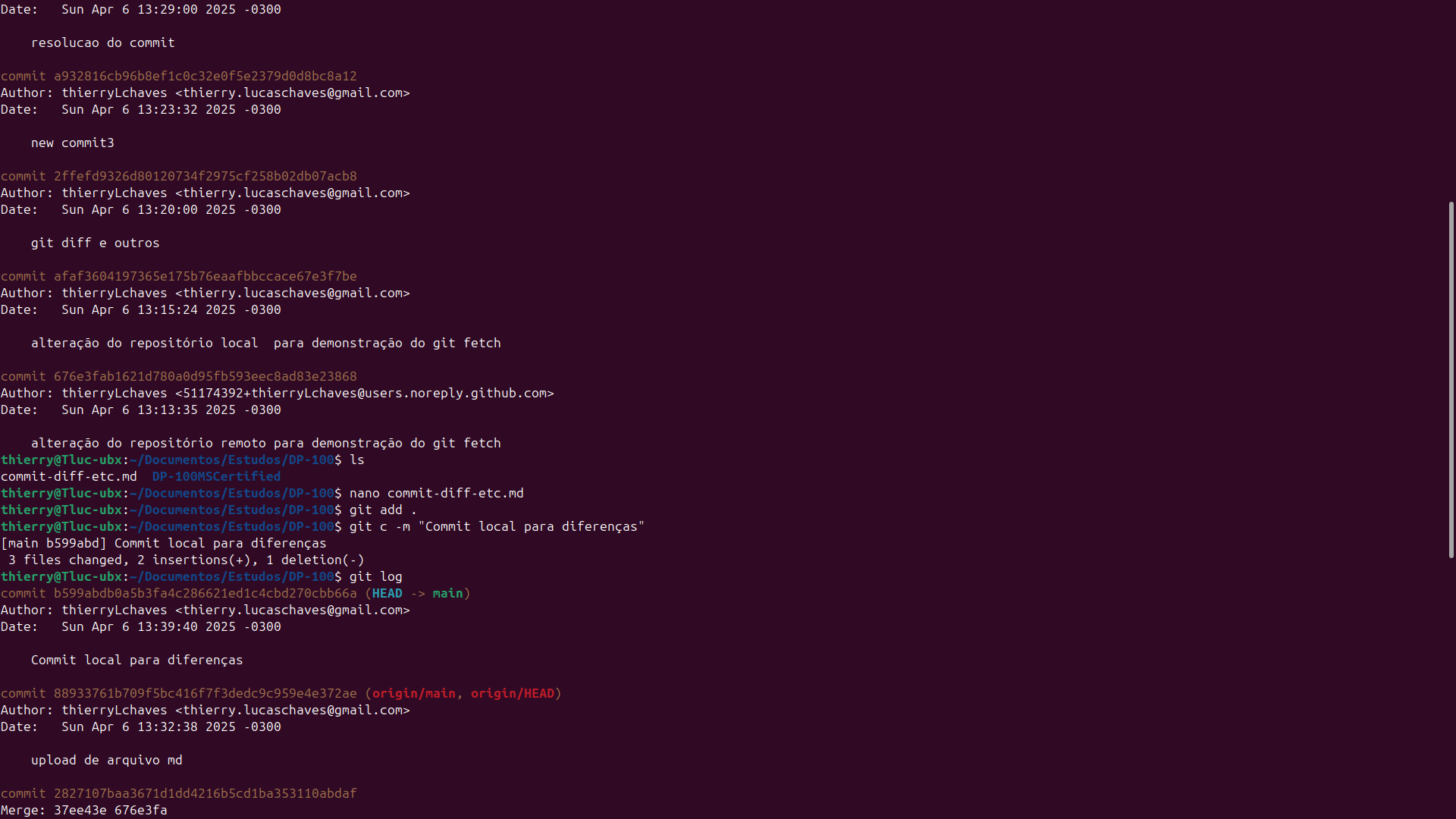
4º será listado algo como o seguinte.

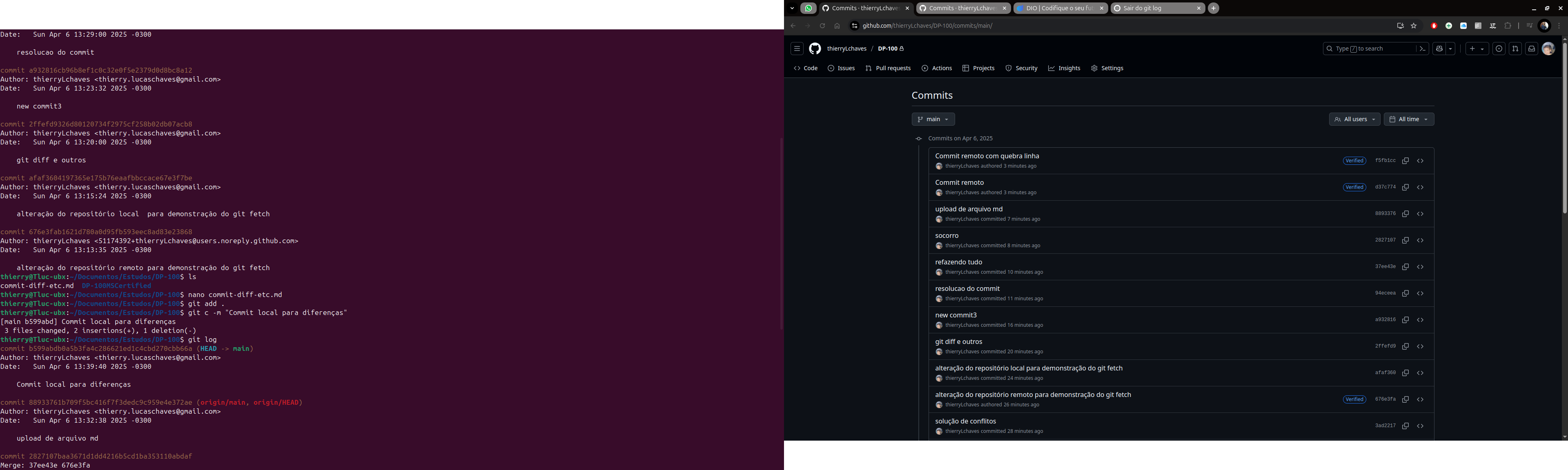
No exemplo a seguir demonstrar através do git log que as branch’s main e master se encontram com arquivos iguais e parelhos

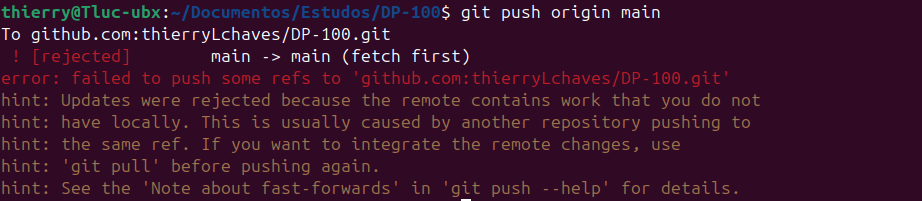


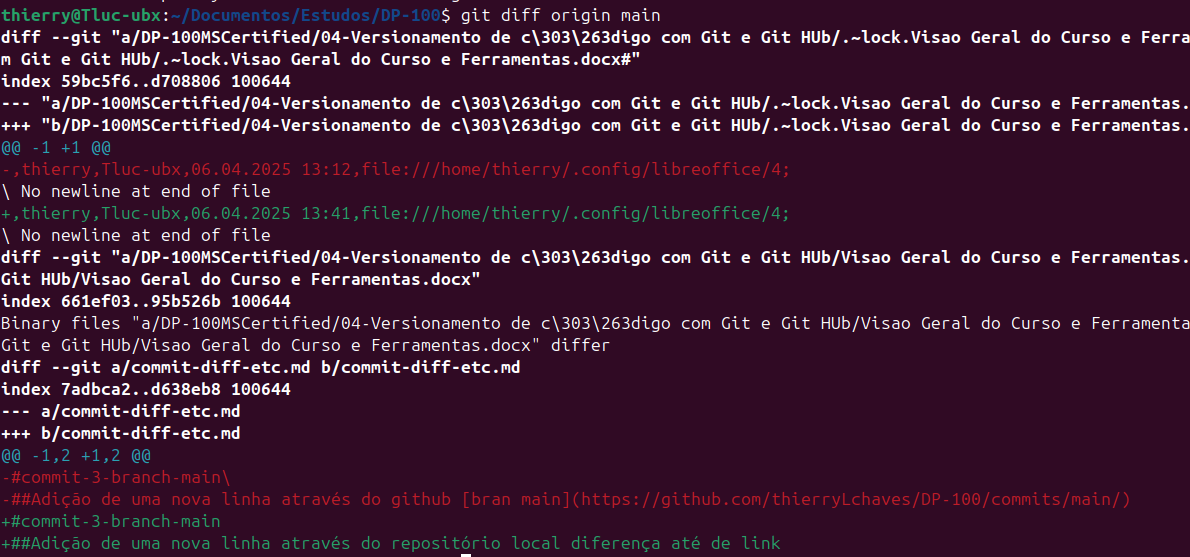
Como estamos realizando alterações no mesmo arquivo com mesmo usuário irei realizar a modificação no arquivo remoto através do github

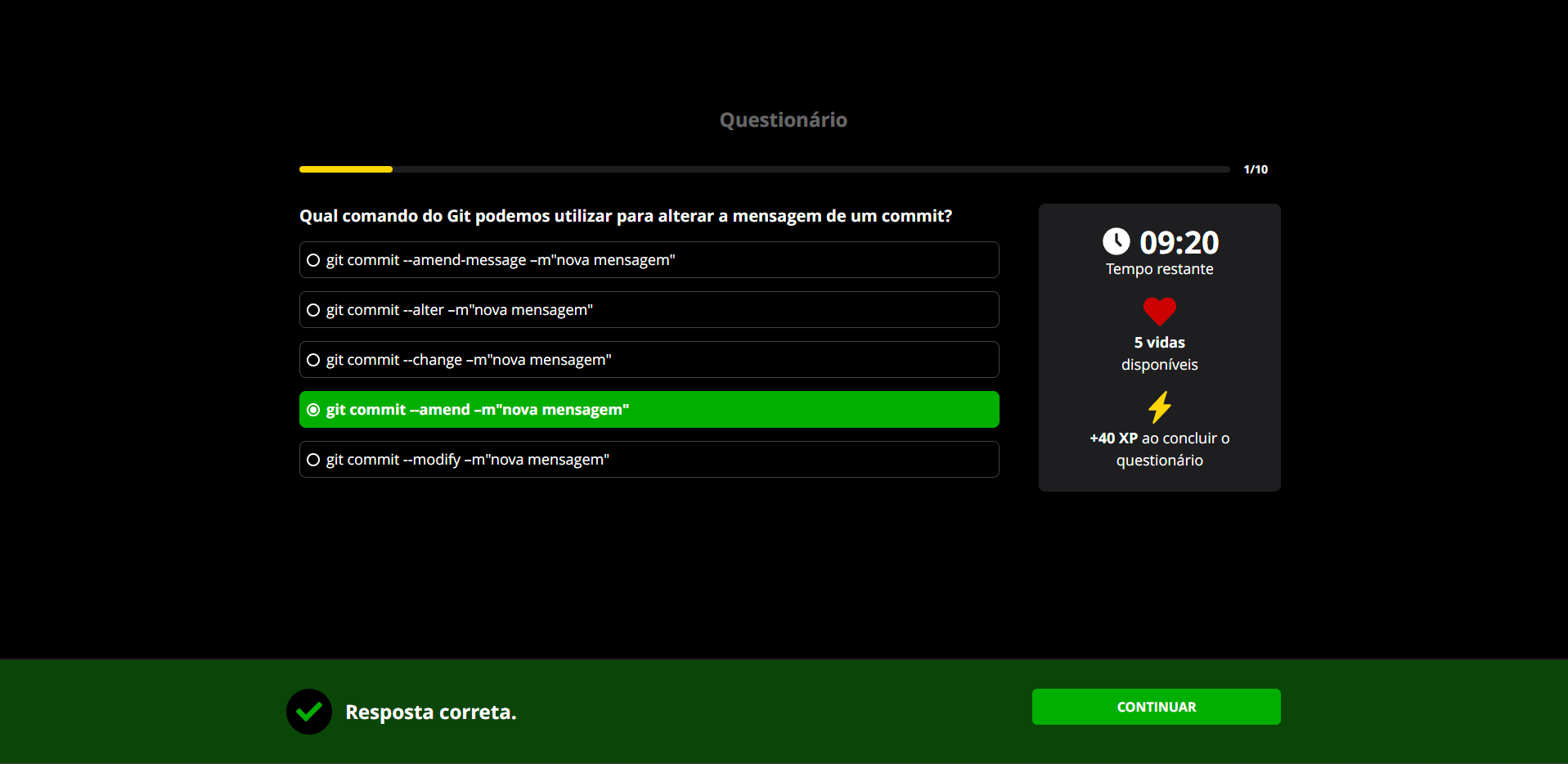
Para realizar o conflito irei realizar a edição do arquivo localmente adicionando uma nova linha 2 diferente da linha 2 do commit remoto sem realizar o git pull

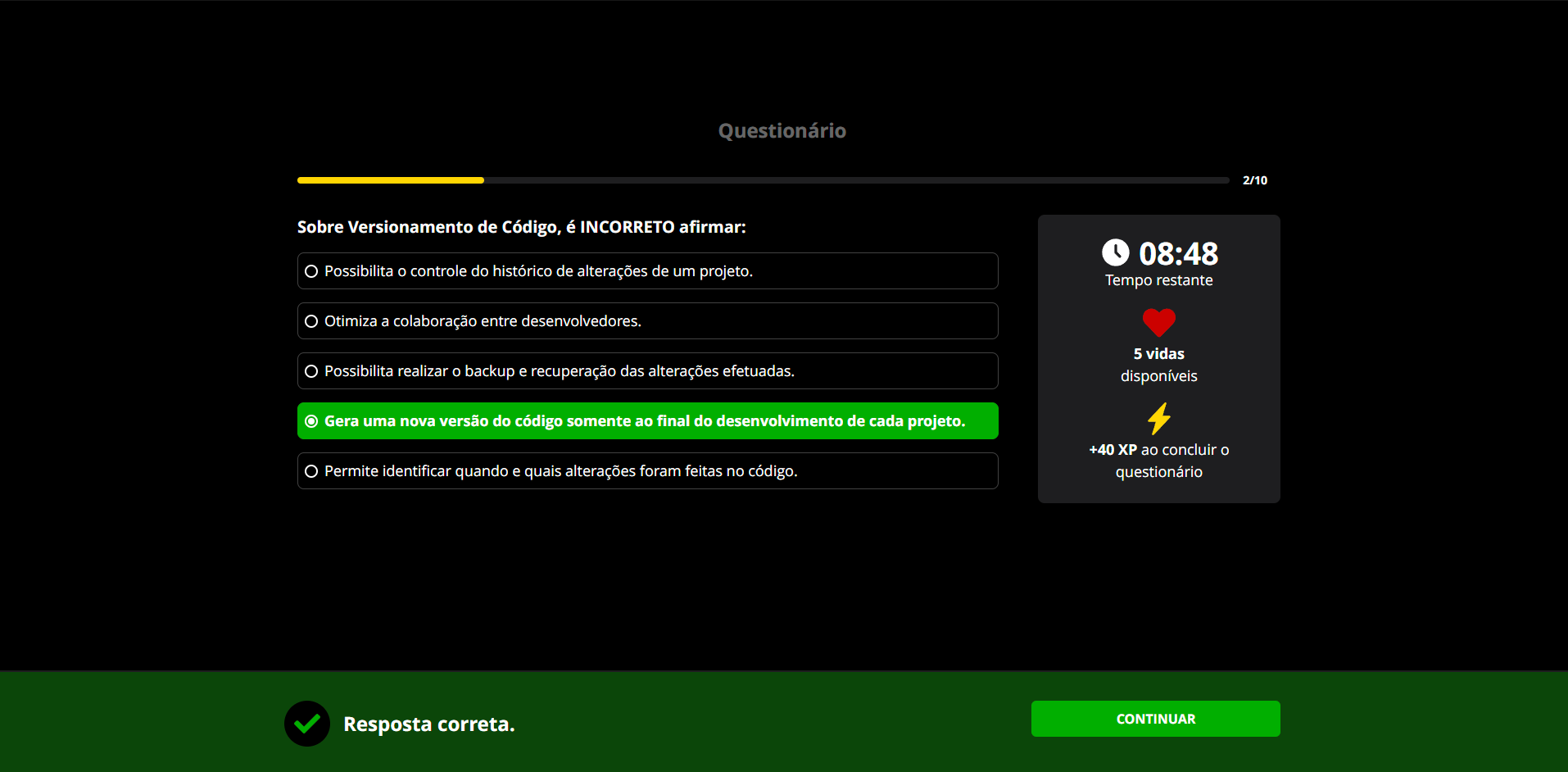
Como podemos visualizar o commit indicado localmente é do commit local para diferenças na main

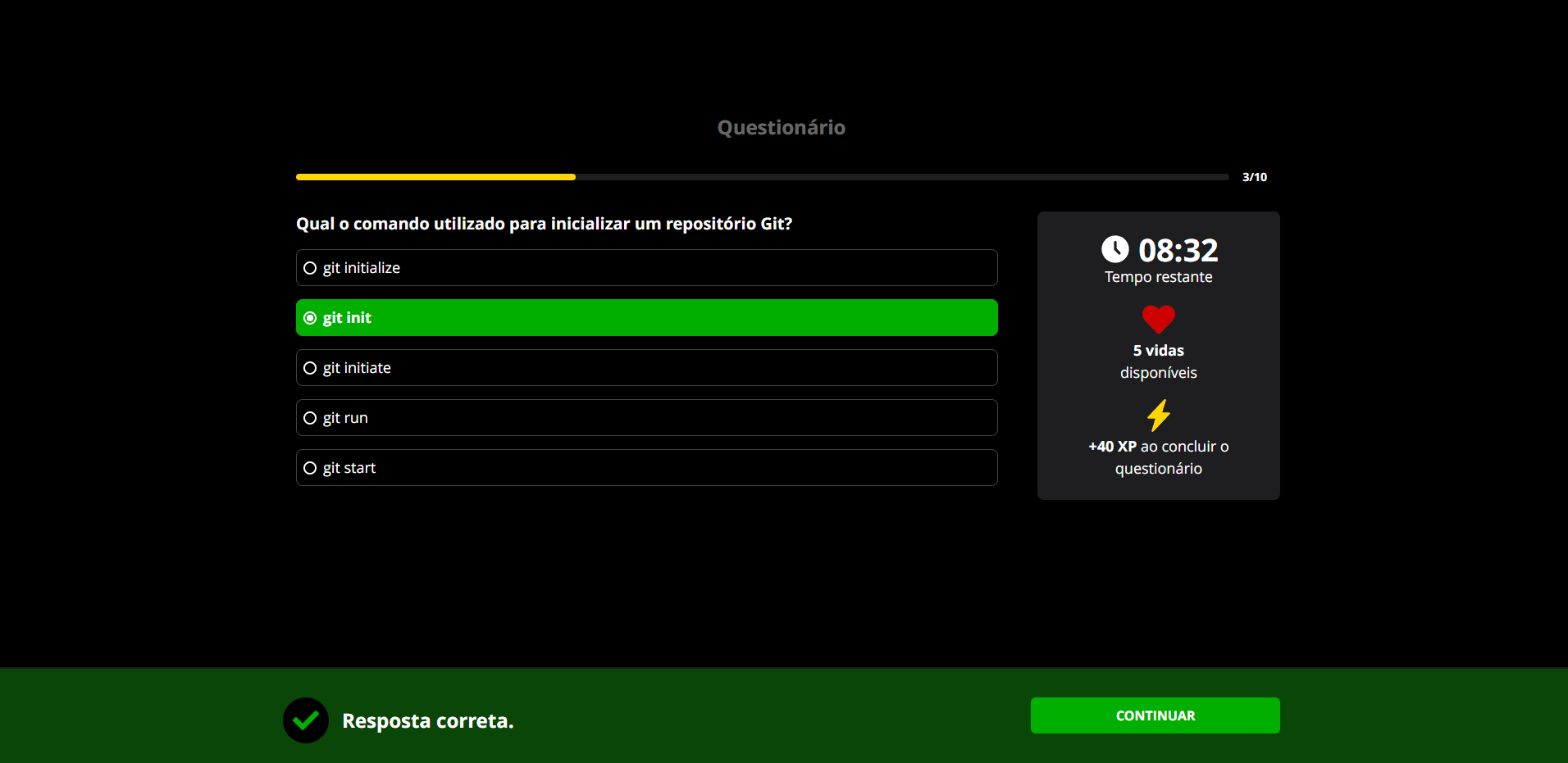
já o repositório remoto aponta pra outro commit. Sendo assim irei realizar o push para tentar enviar o commit

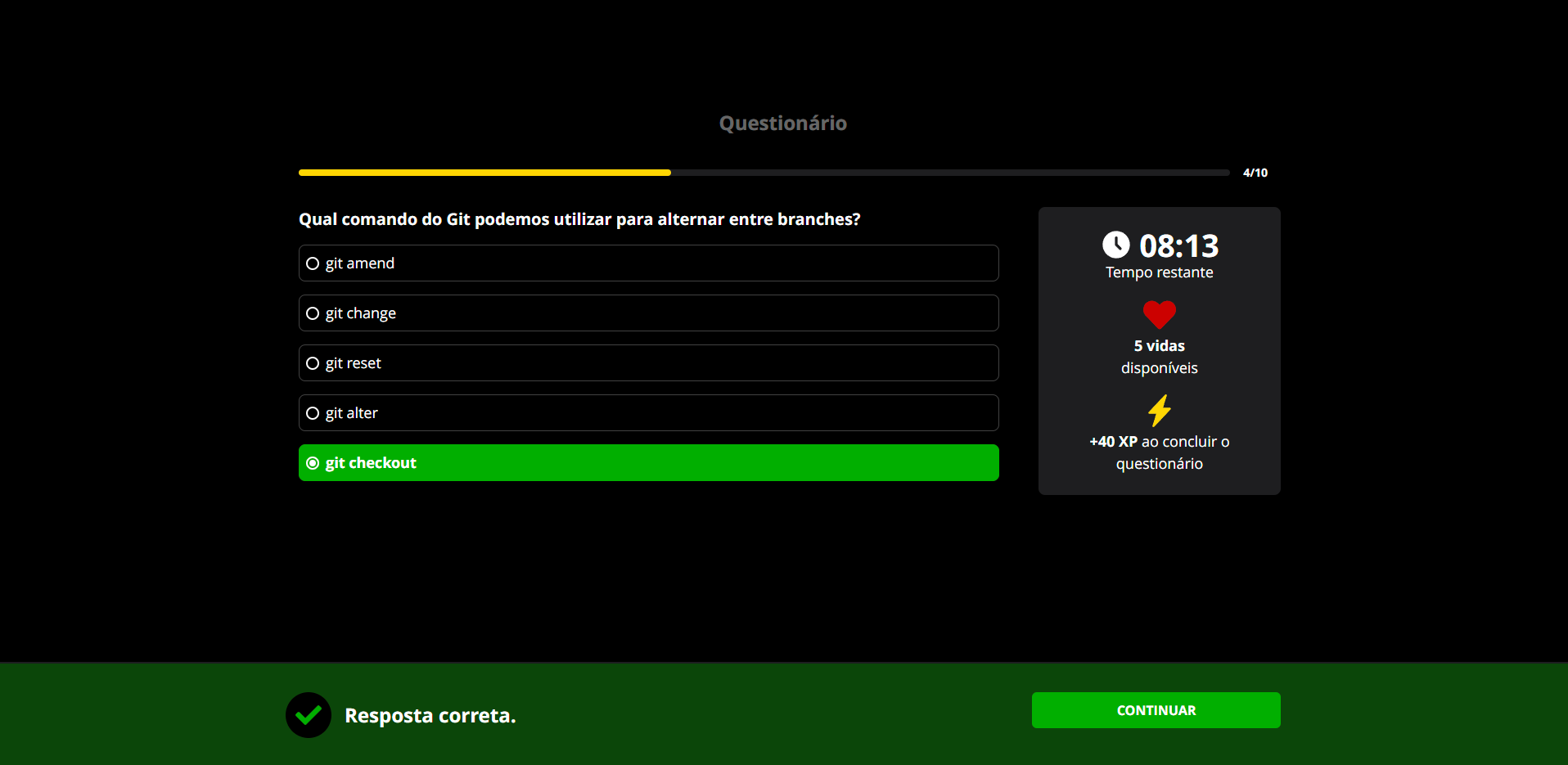
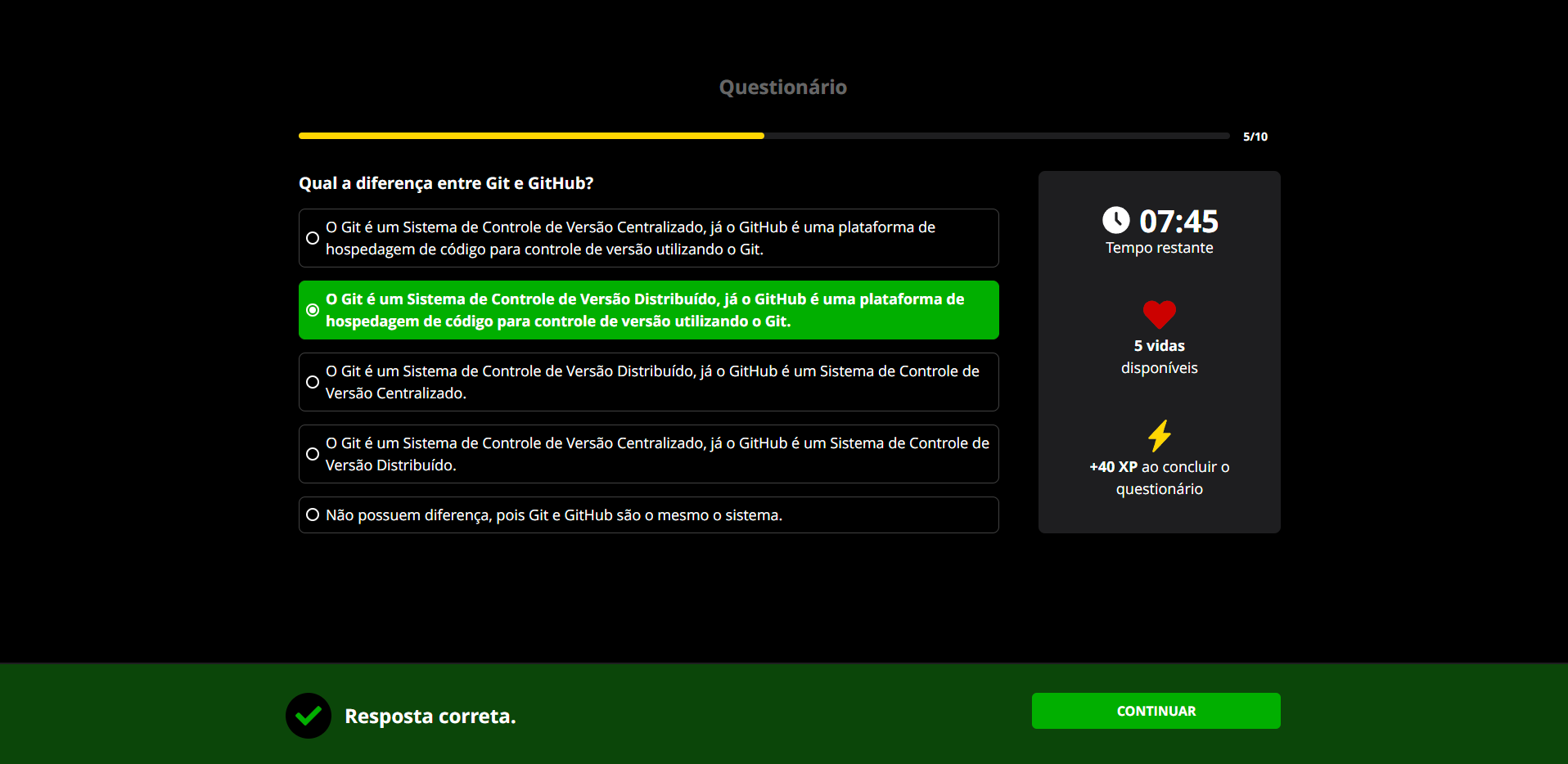
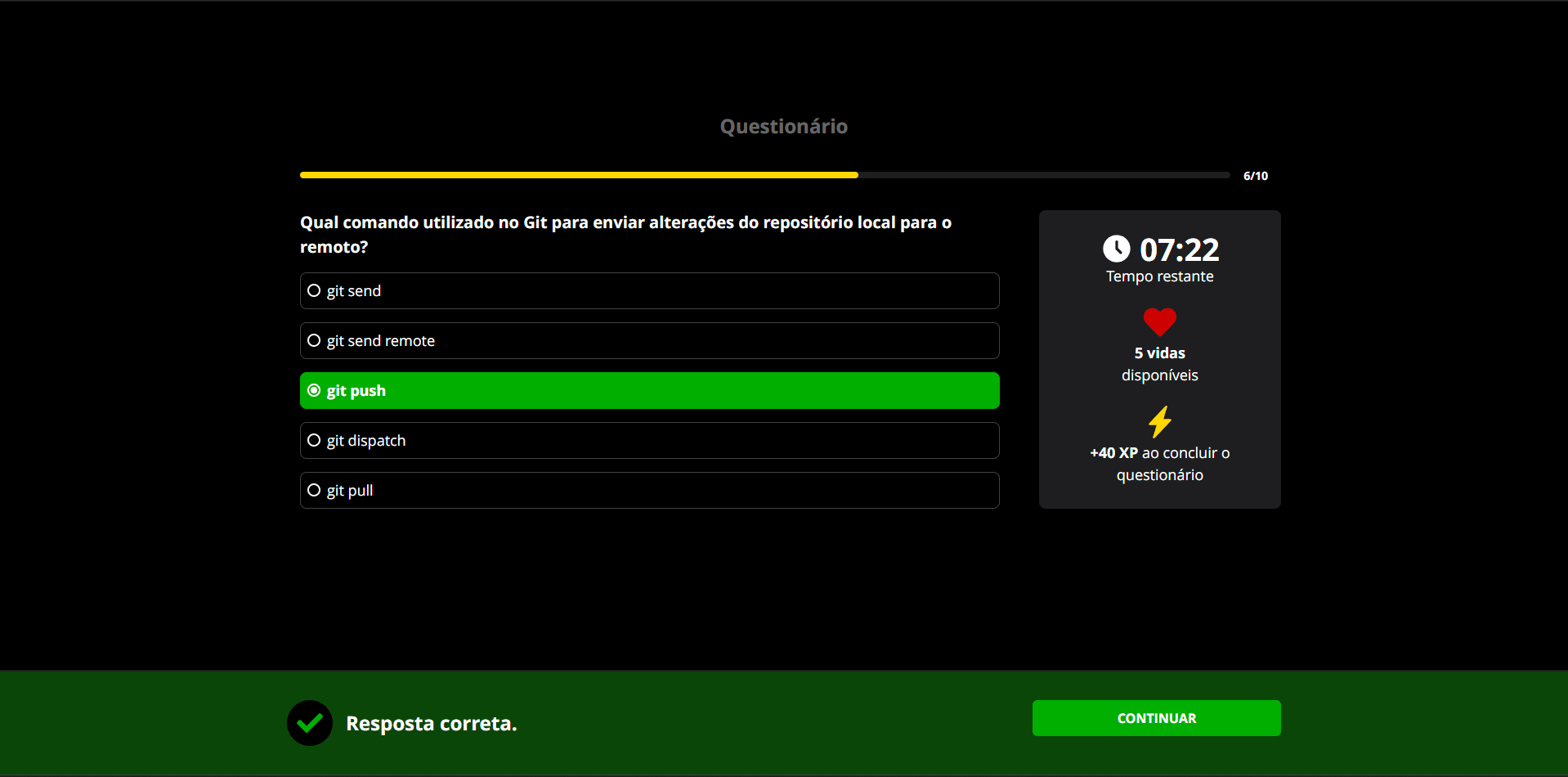
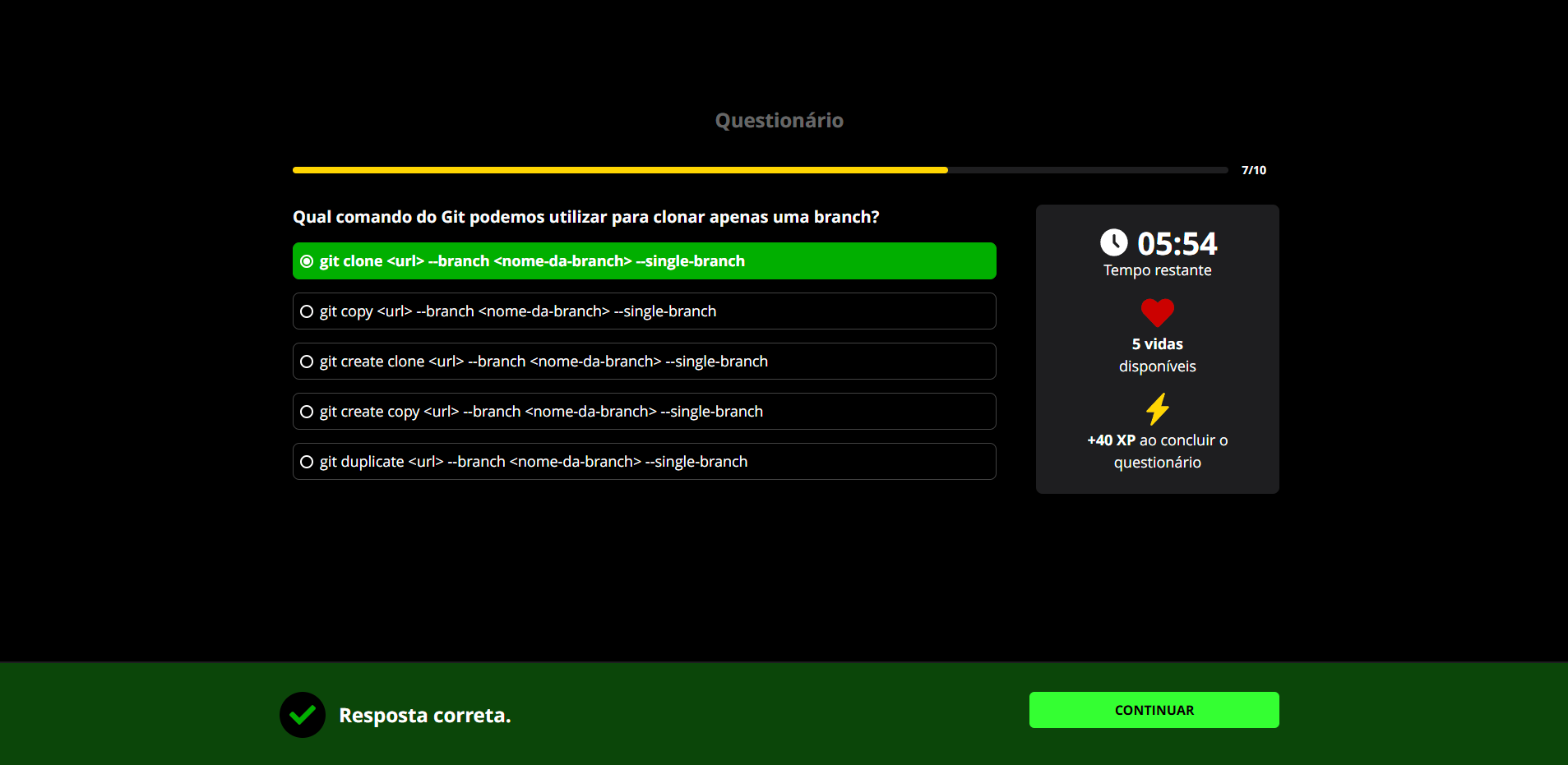
**Nesse caso o git já aponta que a diferenças entre os commit remotos e locais.**

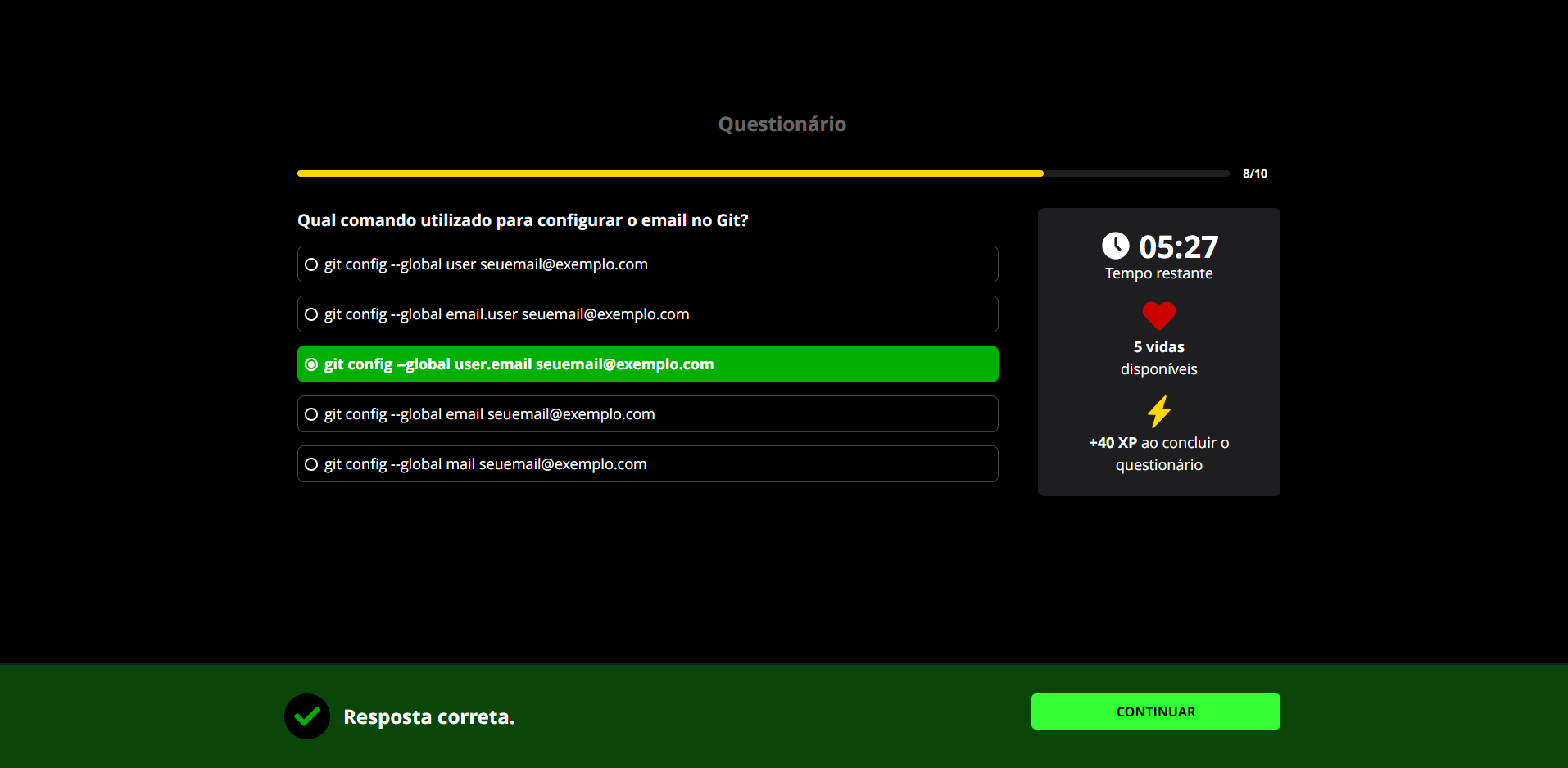
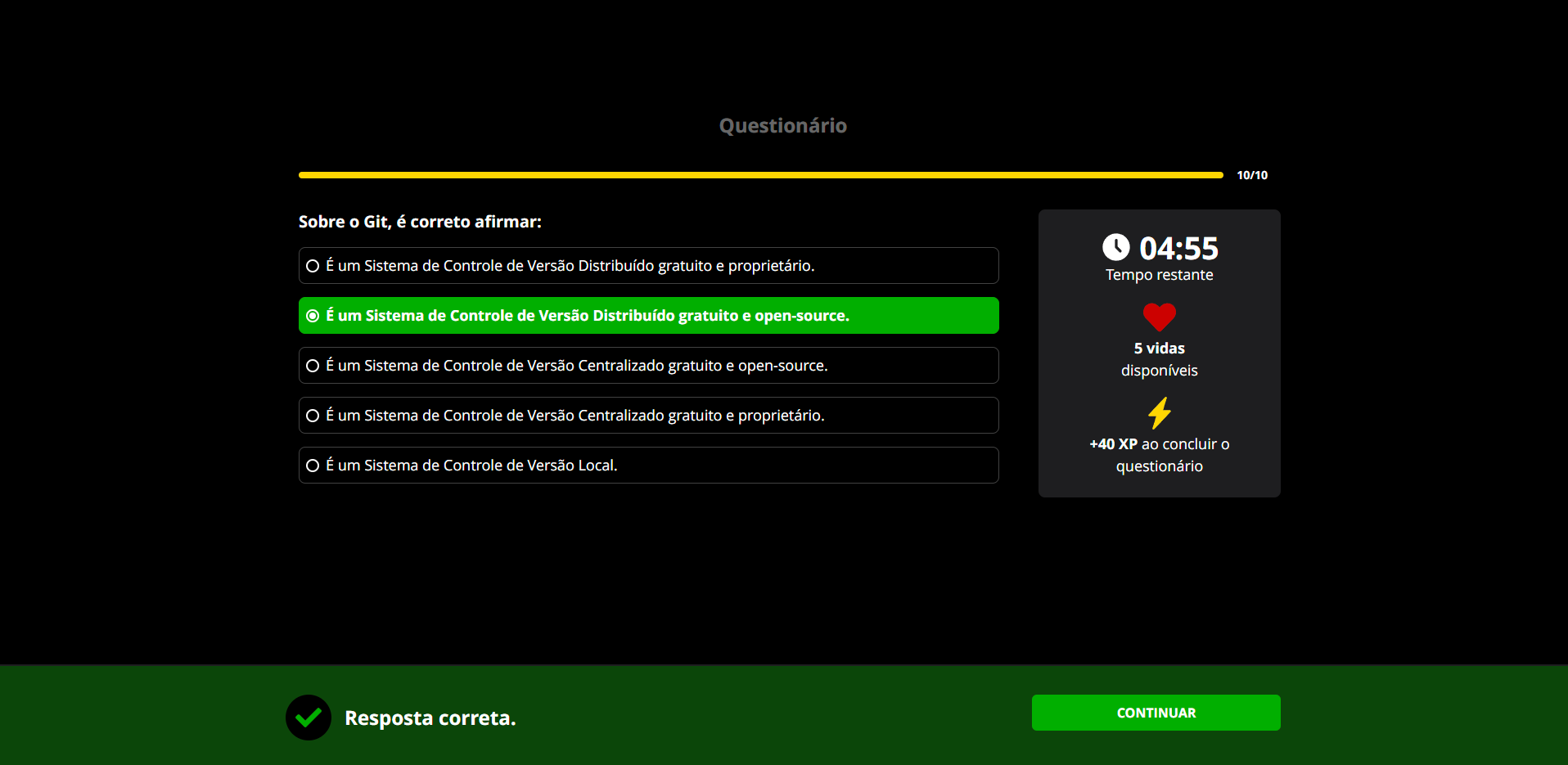
**Ao visulizar o console percebemos que o arquivo em questão contem diferenças da linhas em vermelho o remoto em verde o local**

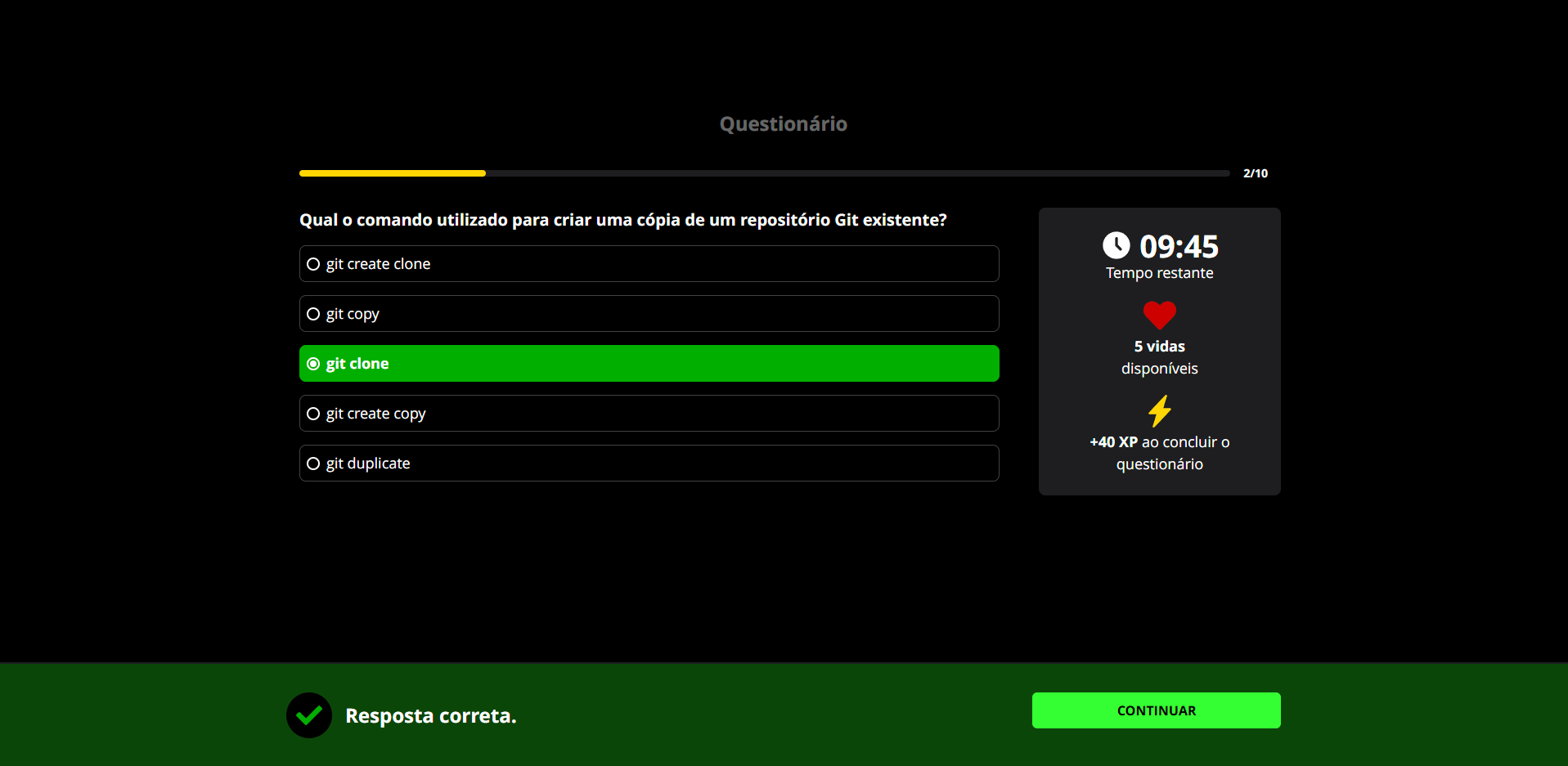










**Materiais Complementares**

Nossos materiais complementares e de apoio têm como objetivo apresentar informações para facilitar e enriquecer a sua jornada de aprendizado. Para isso, links úteis (como slides, repositórios e páginas oficiais) serão disponibilizados, além de dicas sobre como se destacar na DIO e no mercado de trabalho 😉

**Repositório Git**

O Git é um conceito essencial no mercado de trabalho atualmente, por isso sempre reforçamos sua importância em nossa metodologia educacional. Por isso, todo código-fonte desenvolvido durante este conteúdo foi versionado no seguinte endereço para que você possa consultá-lo a qualquer momento:

<https://github.com/elidianaandrade/dio-curso-git-github>

**Slides**

[Versionamento de Código com Git e GitHub.pptx](https://academiapme-my.sharepoint.com/:p:/g/personal/renato_dio_me/EYjkgVZuUv5HsVgJUEPv1_oB_QWs8MFBY_PBQ2UAtLqucg?e=262HGK)

**Dicas/Links Úteis**

Por fim, disponibilizamos alguns links úteis para que você possa se desenvolver ainda mais através de referências oficiais das tecnologias, páginas de documentação e/ou fóruns de discussão relevantes. Nesse contexto, seguem algumas sugestões:

* **Artigos/Fórum**: você pode compartilhar conteúdos técnicos através de [Artigos](https://web.dio.me/articles) (visíveis globalmente na plataforma da DIO). Por outro lado, você também pode compartilhar suas conquistas e dúvidas usando os Fóruns (que são específicos para cada experiência educacional na DIO, como um Bootcamp por exemplo);
* **Rooms**: caso você esteja inscrito(a) em uma experiência educacional na DIO (como um Bootcamp, por exemplo) você terá acesso ao Rooms. O Rooms é uma ferramenta de bate-papo em tempo real onde todos os inscritos podem interagir, compartilhando dúvidas e dicas (que podem conter imagens e snippets de código-fonte);
* **Pesquise na Web**: pode parecer óbvio, mas é importante frisar a importância das engines de busca no dia-a-dia de um profissional de TI. Caso não encontre o que procura dentro da DIO, pesquise sobre o assunto (conceito, dúvida, erro etc) na Internet (dê um Google), pois na maioria das vezes você será levado à páginas incríveis como o [StackOverflow](https://stackoverflow.com/) que salvarão o seu dia 😎