Git

Parte 1: Introdução ao Git

A palavra “versão”, como citado, se refere a uma forma particular de algo, geralmente tendo relação a uma **revisão ou atualização** de alguma coisa existente.

Um sistema de controle de versão armazena um conjunto de arquivos ou, como é comum dizer no mundo do desenvolvimento, o código-fonte do projeto (independentemente da linguagem de programação utilizada).

Ele oferece uma maneira eficiente e concisa de gerenciar as alterações de um mesmo arquivo. Ele torna possível visualizar o histórico de todas as modificações feitas, incluindo data de alteração e qual ponto do arquivo foi editado.

Assim pode-se atuar na criação e/ou alteração dos arquivos do projeto sem conflitos e sem sobreposição das alterações.

E se, porventura, você estiver desenvolvendo um determinado requisito de seu projeto e ele falhar, você pode voltar os seus arquivos a um ponto anterior em que seu projeto esteja estável.

Sendo assim, não é necessário criar múltiplas versões do mesmo arquivo e todos podem usar um único, simplificando o trabalho.

O Git

O Git é um sistema de controle de versões utilizado pelos desenvolvedores de software. Foi criado em 2005 por Linus Torvalds e tornou-se uma ferramenta essencial para as equipes de desenvolvimento de software para registrar as mudanças ao longo do tempo.

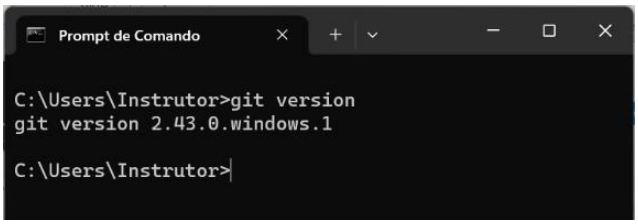
Instalação do Git

Para utilizarmos o git, é necessário à sua instalação em nosso computador. O download do git pode ser feito no link abaixo:

https://git-scm.com/downloads

Escolha o download de acordo com a versão do seu Sistema Operacional.

Para verificarmos se o git foi instalado corretamente vamos executar o seguinte comando: git version



Ao abrir o cmd em seu computador o prompt aparecerá conforme o login em seu Sistema Operacional.

O comando nas imagens aparecerá sempre após o prompt.

Conceitos fundamentais do Git

Fonte: https://www.alura.com.br/artigos/o-que-e-git-github

Para o funcionamento do Git, existem conceitos com nomenclaturas um tanto diferentes, que são:

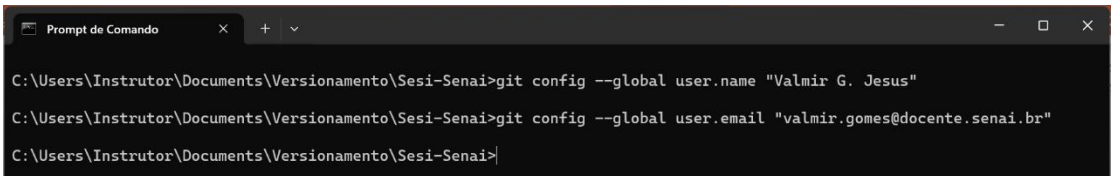
Repositórios, commits e árvores (Trees)

Um repositório é como uma pasta ou diretório que contém todos os arquivos e o histórico de um projeto.

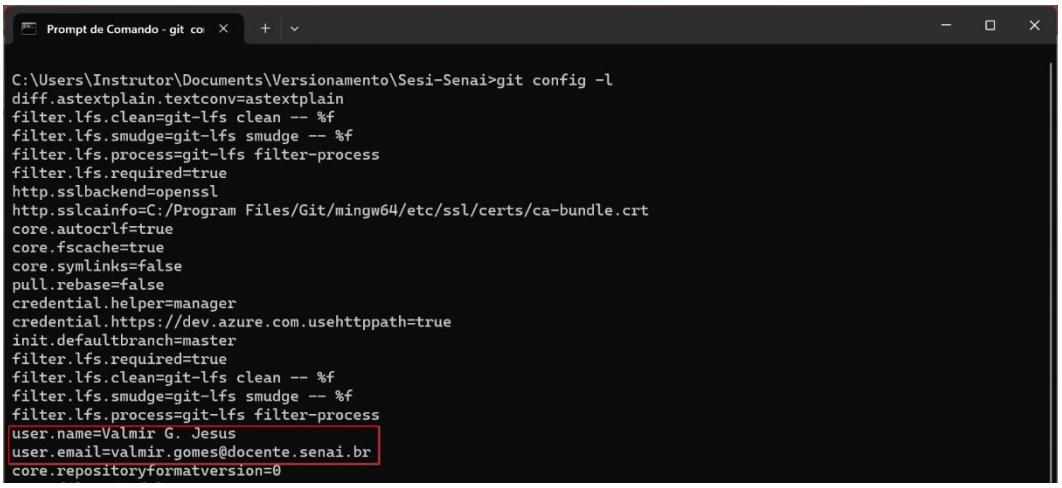
Já o termo **commit** pode ter como tradução literal "compromisso", que seria uma ação em que você faz uma alteração no projeto, se compromete e salva suas alterações no histórico do projeto, ou seja, cada commit é uma entrada no histórico que contém informações sobre as alterações feitas.

Árvores, por último, representam a estrutura do diretório e arquivos em um commit específico, que tem como função registrar a organização do projeto ao longo do histórico de desenvolvimento.

Conte para o Git quem é você.



Vamos ver se o git entendeu quem é você: git config -l



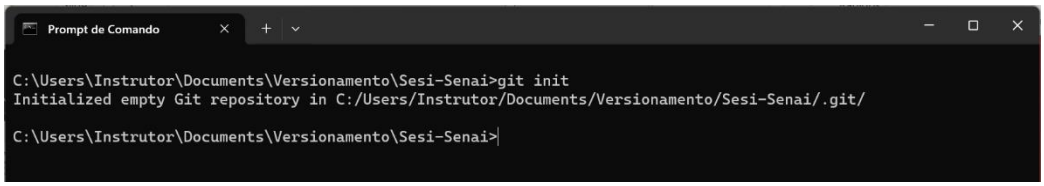
Agora que o Git já te conhece, vamos ao primeiro projeto:

Projeto Sesi-Senai

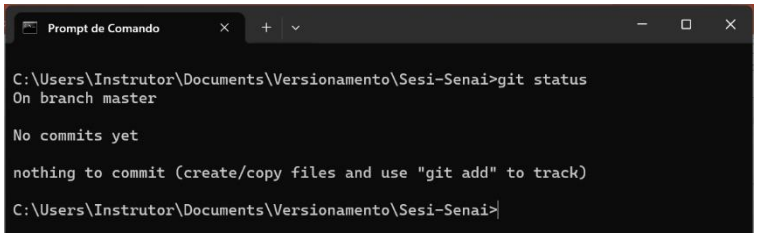
Enquanto explicamos as funcionalidades principais do git, construiremos um projeto de endereçamento.

Etapa 1 | Criar a pasta do projeto

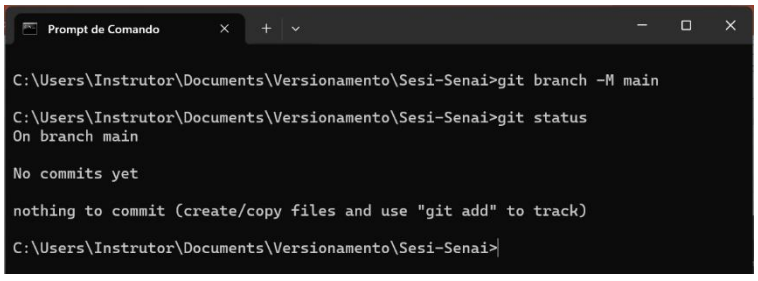
Etapa 2 | Iniciar versionamento.



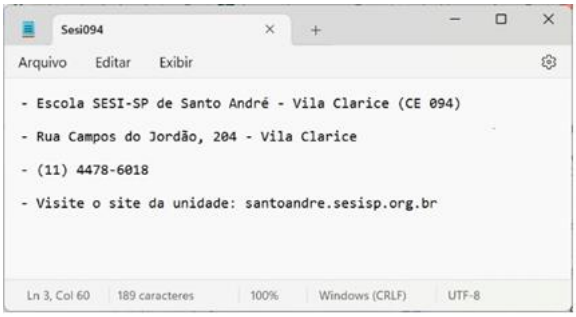
Conferir o status



Etapa 3 | Trocar o nome da branch principal



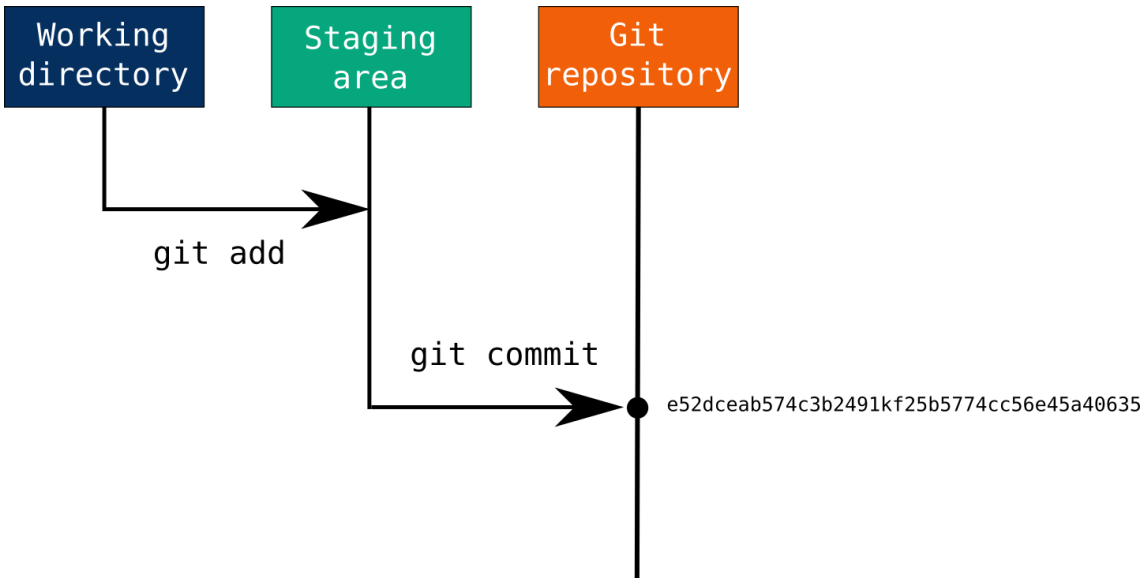
Abrir o “Bloco de notas” e construir o arquivo Sesi094.txt



Salve o arquivo no repositório do projeto.

Os endereços das unidades do Sesi foram retirados do link:
https://www.sesisp.org.br/educacao/educacao/santo-andre

Bom, agora que nosso repositório tem um arquivo, precisamos entender o ciclo de vida dos arquivos que compõe um projeto.



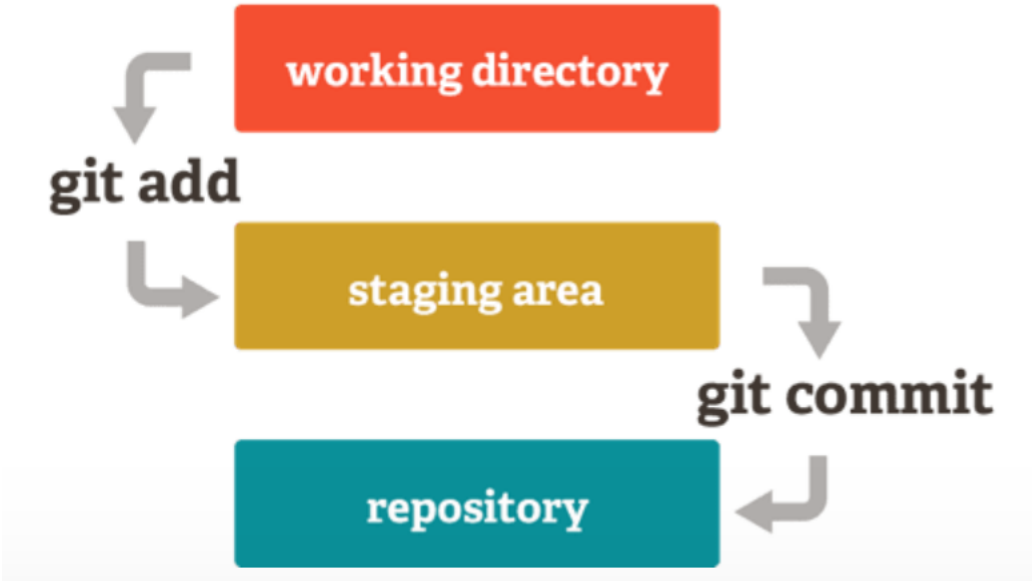
Git Add

Colocar os arquivos do diretório de trabalho em estado de “Preparado” (Staged).

Commit

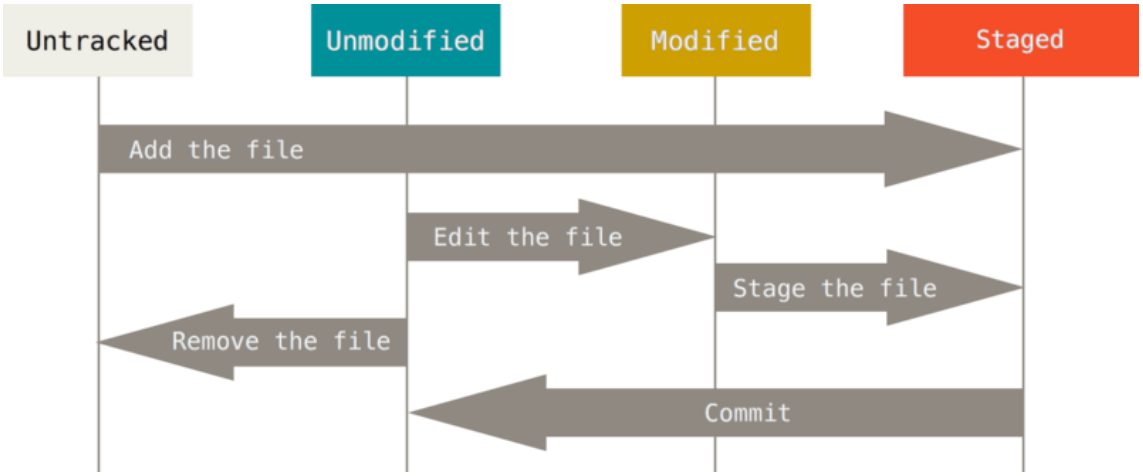
São objetos que guardam o snapshot (instantâneo) de um momento do projeto. Todos os arquivos que se encontram preparados farão parte do commit.

O hashcode SHA-1 que identifica um commit é justamente o que aparece ao utilizar o comando git log, e é essencial para controlar as versões do projeto.



IMPORTANTE:

Situações possíveis do(s) arquivo(s) de um projeto entre o diretório de trabalho e a preparação para o commit:

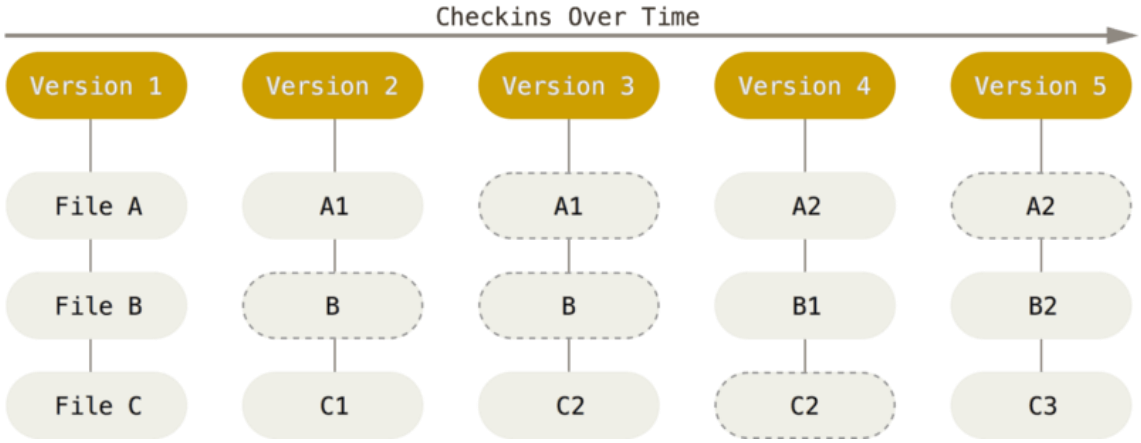


Ao iniciar o versionamento, o comando git status mostra a situação dos arquivos do projeto

Só pode fazer parte de um commit os arquivos que foram adicionados ao “Stage” sejam novos ou modificados.

Não construa frases sem sentido para registrar um commit. Procure informar sobre o término de uma etapa ou sobre o atendimento de um requisito do projeto. Evite sempre o descompromisso com a qualidade da informação da frase do commit.

Exemplo de criação e alteração de arquivos em projetos.



De volta ao projeto Sesi-Senai

Adicionar o arquivo Sesi094 ao Staged. Rodar o comando git add Sesi094.txt

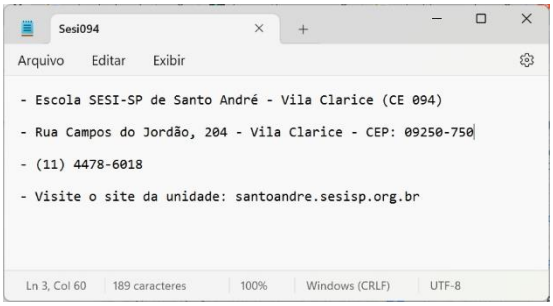
Após adicionar vamos ver o “status” do nosso repositório. Rodar o comando git status

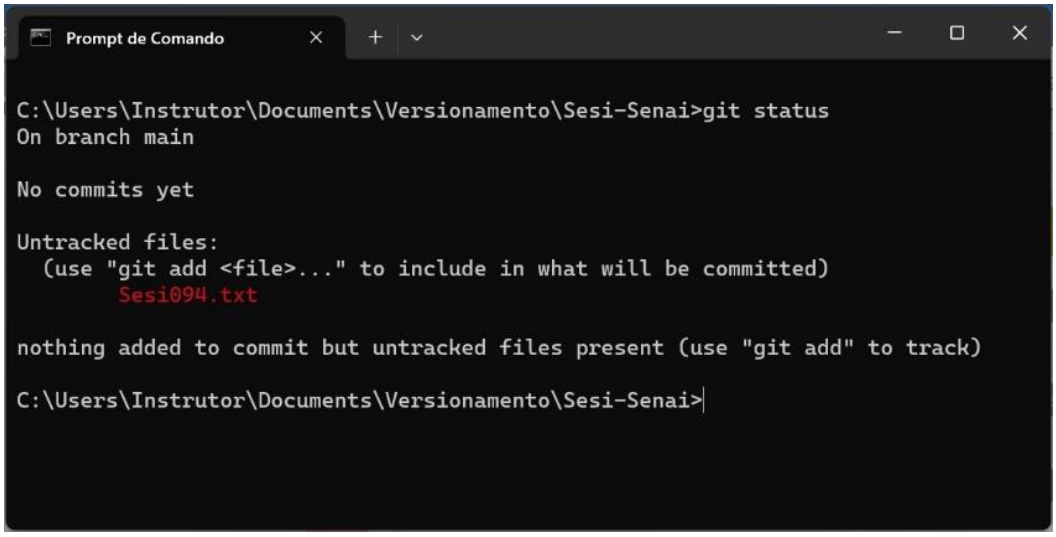
```
C:\Users\Instrutor\Documents\Versionamento\Sesi-Senai>git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Sesi094.txt
```

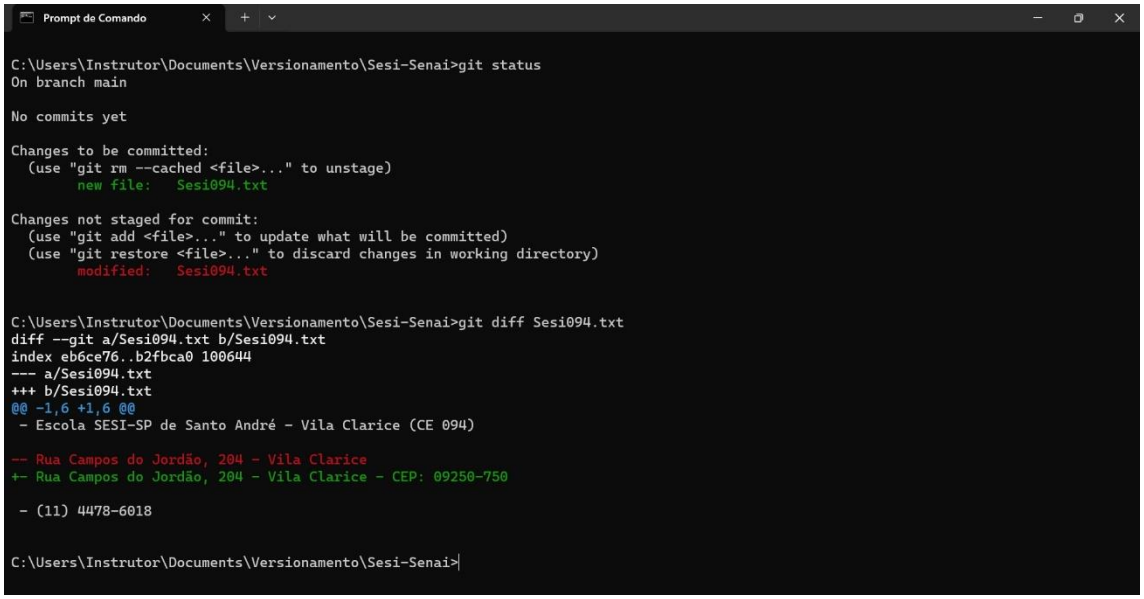
Abrir o arquivo no bloco de notas e adicionar o CEP ao endereço

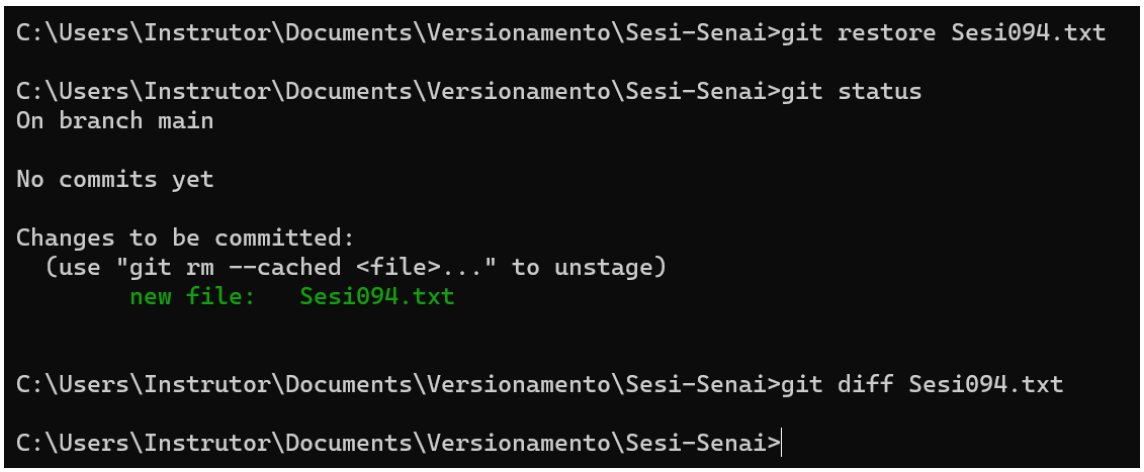




Realizar um restore

O git restore é uma nova opção quando estamos trabalhando e precisamos restaurar algum arquivo ou o projeto por completo.





Primeiro commit

```
C:\Users\Instrutor\Documents\Versionamento\Sesi-Senai>git commit -m "Endereço Sesi 094"
[main (root-commit) 46010fc] Endereço Sesi 094
 1 file changed, 7 insertions(+)
 create mode 100644 Sesi094.txt

C:\Users\Instrutor\Documents\Versionamento\Sesi-Senai>
```

Após primeiro commit, status e log

```
C:\Users\Instrutor\Documents\Versionamento\Sesi-Senai>git status
On branch main
nothing to commit, working tree clean

C:\Users\Instrutor\Documents\Versionamento\Sesi-Senai>git log
commit 46010fc6cc3bf4de10933e1dc720172b1a4e4841 (HEAD -> main)
Author: Valmir G. Jesus <valmir.gomes@docente.senai.br>
Date:   Thu Feb 15 11:58:23 2024 -0300

    Endereço Sesi 094

C:\Users\Instrutor\Documents\Versionamento\Sesi-Senai>
```

Para completar o projeto Sesi-Senai, construir um arquivo de texto para cada unidade do Sesi com dados de endereço nos mesmos moldes do Sesi 094 e realizar commit para cada arquivo. (Sesi 166, 221 e 265).

Os endereços das unidades do Sesi foram retirados do link:
https://www.sesisp.org.br/educacao/educacao/santo-andre

Quando você usa o Git e faz checkout para um commit anterior, altera um arquivo e faz um novo commit, você está efetivamente criando um novo branch a partir do ponto do commit anterior. Isso acontece porque o Git segue um modelo de dados baseado em snapshots, onde cada commit representa um estado completo do repositório naquele momento. Vamos detalhar o processo passo a passo:

1. Fazendo Checkout para um Commit Anterior

Quando você faz checkout para um commit anterior usando algo como git checkout <commit-hash>, o Git move o ponteiro HEAD para esse commit específico. O HEAD é uma referência ao commit atual, então mudá-lo para um commit anterior faz com que o seu diretório de trabalho e o índice (staging area) reflitam o estado do repositório naquele momento no tempo. Neste ponto, você está em um estado "DETACHED HEAD", o que significa que você não está mais em uma branch ativa.

2. Alterando um Arquivo e Fazendo Commit

Se você alterar um arquivo e fizer um commit enquanto está em um estado "DETACHED HEAD", o Git criará um novo commit com essas alterações. Esse novo commit é uma continuação direta do commit ao qual você fez checkout, não da branch de onde você veio originalmente.

3. Consequências

Criação de um Novo Branch: Se você quiser manter as alterações que fez a partir do commit anterior, você deve criar uma nova branch a partir do estado atual do HEAD antes de mudar de volta para sua branch original. Isso pode ser feito com git checkout -b nova-branch. Isso move as alterações para uma nova branch, preservando o trabalho que você fez a partir daquele ponto no histórico.

Histórico Dividido: Ao fazer isso, você efetivamente cria um "fork" no histórico do seu projeto. A linha do tempo original (onde estava a branch antes de você fazer checkout para o commit anterior) continua existindo, mas agora há uma nova linha do tempo iniciada pelo seu novo commit, que diverge a partir daquele ponto.

Possíveis Conflitos de Merge: Se você planeja integrar as alterações dessa nova branch de volta à branch original (ou a qualquer outra branch), pode ser necessário resolver conflitos de merge, especialmente se as mesmas partes dos arquivos foram modificadas em ambas as linhas do tempo.

Importante

É essencial entender que ao trabalhar em um estado "DETACHED HEAD" e fazer commits, esses commits podem se tornar "órfãos" se não forem associados a uma nova branch. Commits órfãos eventualmente podem ser coletados pelo garbage collector do Git, pois não há nenhuma branch apontando para eles, tornando difícil recuperar o trabalho perdido. Portanto, se você deseja manter as alterações feitas a partir de um commit anterior, sempre crie uma nova branch para essas alterações antes de mudar para outra parte do projeto.