Aula Prática: Git e GitHub

Perguntas - Introdução e Conceito

1. Por que o Git é considerado um sistema de controle de versão distribuído?

💡 Resposta:

O Git é considerado **distribuído** porque **cada desenvolvedor que clona** um repositório remoto (como no GitHub) obtém uma **cópia completa** de todo o **histórico do projeto** (incluindo todos os branches e commits). Isso permite que eles trabalhem completamente offline, façam commits e vejam o histórico sem precisar de uma conexão constante com o servidor central. A sincronização (push/pull) só é necessária para compartilhar as mudanças.

2. Qual a diferença entre working directory, staging area e repository?

💡 Resposta:

- **Working Directory** (Diretório de Trabalho): É a pasta no seu sistema de arquivos onde você está **visualmente trabalhando** nos arquivos. São os arquivos atuais, que podem ter sido modificados e ainda não foram rastreados pelo Git.
- **Staging Area** (Área de Preparação, ou Index): É uma **área intermediária** onde você **seleciona** as mudanças específicas (adições, modificações, exclusões) que farão parte do **próximo commit**. É como um rascunho do que você quer salvar.
- **Repository** (Repositório Local): É a **estrutura de dados** (geralmente a pasta .git oculta) que armazena todos os **commits** (versões salvas), histórico do projeto, branches, e metadados. É o **banco de dados** do controle de versão.

3. Para que serve o comando git clone?

💡 Resposta:

O comando **git clone** é usado para **criar uma cópia local** de um repositório Git **existente** (geralmente um repositório remoto, como no GitHub, ou outro repositório local). Ele baixa todo o histórico do projeto e configura o repositório local para rastrear o repositório remoto original.

4. Onde estão implementados fisicamente working directory, staging area e repository?

💡 Resposta:

Working Directory: É a **pasta/diretório** do projeto no seu sistema de arquivos (os arquivos que você edita).
Staging Area: É implementada fisicamente como um arquivo chamado **index** dentro da pasta oculta **.git** do repositório local.
Repository: É implementado fisicamente dentro da pasta **.git** do repositório local, contendo todos os objetos de dados (como os commits) e referências (como branches).

5. Quais os estados de um arquivo no repositório do git?

💡 Resposta:

Untracked (Não rastreado): O Git **ainda não monitora** este arquivo. É um arquivo novo que nunca foi adicionado à Staging Area.
Unmodified (Não modificado): O arquivo **não foi alterado** desde o último commit.
Modified (Modificado): O arquivo foi alterado no **Working Directory** desde o último commit, mas ainda **não foi adicionado** à Staging Area.
Staged (Preparado): O arquivo modificado (ou novo) foi **adicionado à Staging Area** com git add e está pronto para ser incluído no próximo commit.

6. Explique as possíveis transições de estado de um arquivo no repositório do git?

💡 Resposta:

Untracked –> **Staged:** Usando o comando **git add** em um arquivo novo.
Unmodified –> **Modified:** Ocorre quando o arquivo é **editado** no Working Directory.
Modified –> **Staged:** Usando o comando **git add** em um arquivo que já é rastreado (tracked) e foi modificado.
Staged –> **Unmodified:** Ocorre quando o comando **git commit** é executado. A versão do arquivo na Staging Area é salva permanentemente como um commit, e o arquivo volta ao estado "não modificado" em relação ao histórico.
Staged –> **Modified:** Se o arquivo for editado novamente **após** ter sido adicionado à Staging Area, ele estará em ambos os estados: a versão "preparada" e uma nova versão "modificada" no Working Directory.

2. Prática com Git Local

Etapa 1 - 💡 Resposta: A mensagem exibida é: **Initialized empty Git repository in C:/Users/dudus/aula-git/.git/.** Na prática, isso significa que o Git criou a pasta oculta .git dentro do diretório aula-git. É nessa pasta que todo o histórico e

metadados do repositório serão armazenados, e o diretório aula-git agora se torna o Working Directory deste novo repositório Git.

Etapa 2 – Adicionar arquivo e fazer commit

Perguntas:

1. Qual o estado do arquivo antes e depois do git add?

💡 Resposta: **Antes do git add:** O estado do arquivo é **Untracked** (Não rastreado) ou, se já fosse rastreado e modificado, **Modified** (Modificado). Neste caso, o arquivo.txt foi criado, então o estado é **Untracked** (confirmado pelo git status).

2. O que significa o estado untracked e tracked?

💡 Resposta: **Untracked (Não Rastreável):** Significa que o arquivo está no seu **Working Directory**, mas o Git **nunca o viu antes** e não está incluindo nenhuma versão dele em seu histórico.
Tracked (Rastreável): Significa que o Git **já registrou** o arquivo em algum momento no histórico (no último commit). Um arquivo rastreável pode estar nos estados **Unmodified**, **Modified**, ou **Staged**.

3. Qual o objetivo do git commit?

💡 Resposta: O objetivo do comando git commit é **registrar permanentemente** a versão dos arquivos que estão atualmente na **Staging Area** (preparados) como um **novo ponto na história** do projeto. O commit representa uma mudança atômica, coerente e salva, e é a unidade básica do histórico do Git.

4. Qual o estado do arquivo após o git commit?

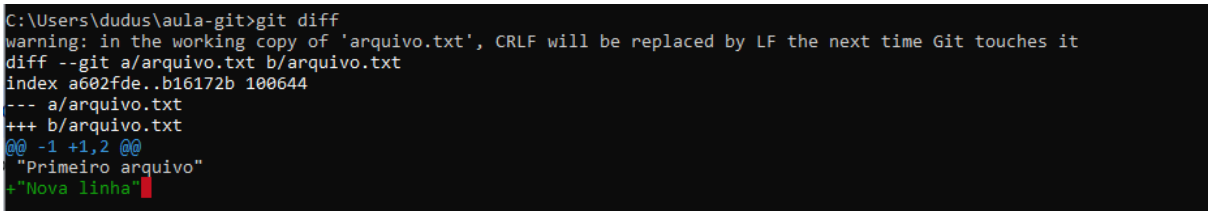
💡 Resposta: O estado do arquivo é **Unmodified** (Não modificado). Ele agora corresponde perfeitamente à versão que acabou de ser salva no histórico.

Etapa 3 – Histórico e alterações

Perguntas:

1. O que o comando git diff mostra?
💡 Resposta: O comando git diff sem argumentos mostra as **diferenças** entre os arquivos no seu **Working Directory** e a versão mais recente dos arquivos

na **Staging Area** (ou, se a Staging Area estiver vazia/os arquivos não estiverem staged, ele compara com o último commit). Ele serve para ver o que você mudou, mas **ainda não preparou** para o commit.



2. Qual commit está atualmente apontado por HEAD?

💡 Resposta: O HEAD aponta para o **commit mais recente** da **branch atual** que é a main. Ele sempre indica a **ponta da linha de trabalho** no seu repositório local.

Etapa 4 – Trabalhando com Branches

Perguntas:

1. Como verificar em qual branch você está?

💡 Resposta: O comando mais comum é **git branch** (que lista todas as branches e destaca a atual), ou **git status** (que geralmente informa a branch atual no topo da saída).

2. O que acontece se você rodar git merge nova-feature estando na branch principal?

💡 Resposta: O comando git merge nova-feature irá **integrar** as alterações da branch nova-feature na **branch atual**. O Git tentará combinar as histórias das duas branches, criando um novo **merge commit** (ou executando um **Fast-Forward** se não houver conflitos e a branch principal não tiver novos commits).

3. Conectando ao GitHub

Perguntas:

1. O que significa o -u no comando git push -u origin main?

💡 Resposta: O -u é a abreviação de --set-upstream. Ele **configura a branch local** (main) para **rastrear (track)** a branch remota (main no origin).

Na prática, isso significa que, nas próximas vezes, você poderá usar apenas git push ou git pull na branch main sem precisar especificar o remoto (origin) e a branch (main), pois o Git já saberá para onde enviar e de onde puxar as alterações.

2. Como verificar os remotes configurados no repositório?

💡 Resposta: O comando para verificar os repositórios remotos configurados é **git remote -v**. O -v (verbose) mostra tanto o nome curto do remoto (como origin) quanto as URLs de busca (fetch) e envio (push) para cada um.

4. Encerramento e Discussão

💡 Respostas:

Qual etapa foi mais difícil?

A etapa mais difícil para mim foi a de **conceituação e a diferença entre os estados dos arquivos (Working Directory, Staging Area e Repository)**, especialmente quando tive que usar o git diff.

Como o Git ajuda na colaboração?

Controle de Versão Confiável: Mantém um histórico detalhado e imutável de todas as alterações, facilitando o rastreamento de quem fez o quê.

Branches Isoladas: Permite que desenvolvedores trabalhem em novas funcionalidades ou correções (branches) **isoladamente** sem quebrar o código principal (main).

Merge e Pull Requests: Fornece ferramentas para **reunir** o trabalho de diferentes desenvolvedores de forma controlada (via merge) e revisada (via Pull Requests em plataformas como o GitHub).

Distribuição: Por ser distribuído, cada desenvolvedor tem uma cópia completa, garantindo **resiliência** e permitindo trabalho paralelo sem depender de uma conexão constante.

Que diferença faz ter um repositório remoto?

Ter um repositório remoto (como no GitHub, GitLab, etc.) é crucial para:

- **Colaboração Centralizada:** Serve como um **ponto de encontro central** para que todos os colaboradores possam compartilhar e sincronizar seu trabalho.
- **Backup e Segurança:** Atua como um **backup** seguro do projeto, protegendo o código em caso de falha do hardware local de qualquer colaborador.
- **Acesso e Implantação:** É frequentemente usado como a **fonte oficial** do código para processos de integração contínua (CI/CD) e implantação em ambientes de teste ou produção.
- **Acessibilidade:** Permite que membros da equipe acessem o código de qualquer lugar.