

UC Gestão e Qualidade de Software

# **Atividade Prática - Git e GitHub**

Isabella Correa de Souza - RA: 32322616

## 1. Introdução e Conceitos

O Git é um sistema de controle de versão **distribuído** criado por **Linus Torvalds** em 2005 para gerenciar o código do **Linux**.

Cada cópia de um repositório Git contém todo o histórico do projeto, permitindo trabalho offline e controle total de versões.

## Perguntas

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

Porque cada usuário possui uma cópia completa de todo o repositório, incluindo o histórico de alterações, em sua máquina local. Isso diferencia o Git dos sistemas centralizados, nos quais existe apenas um servidor principal e os desenvolvedores dependem dele para consultar ou enviar modificações.

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

- Working Directory: local onde os arquivos do projeto ficam acessíveis para edição no seu ambiente local; todas as alterações feitas, como adicionar, modificar ou excluir arquivos, acontecem nele, antes de serem registradas pelo Git.
- Staging Area: funciona como uma ponte entre o working directory e o repository, onde é selecionada quais alterações feitas serão incluídas no próximo commit, utilizando o comando git add; só os arquivos presentes na staging area serão realmente registrados quando o commit for executado.
- Repository: banco de dados interno do Git onde os commits são armazenados; todos os arquivos que passaram pela staging area e foram "comitados" ficam registrados definitivamente nele, formando o histórico de versões do projeto.

### 3. Para que serve o comando git clone?

Serve para criar uma cópia local completa de um repositório Git existente, geralmente hospedado em um serviço remoto, como GitHub ou GitLab. Ao executar git clone, são copiados todos os arquivos, histórico de commits, branches e configurações, permitindo trabalhar localmente em um projeto já existente.

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

O working directory está na pasta principal do projeto, enquanto staging area e repository estão fisicamente no diretório .git, com a staging area representada principalmente pelo arquivo index.

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

Os principais estados são:

- Untracked (não rastreado): arquivo presente no working directory, mas ainda não está sendo monitorado pelo Git porque não foi adicionado à área de staging.
- Modified (modificado): arquivo já rastreado pelo Git, sofreu mudanças após o último commit, mas ainda não foi incluído para o próximo commit na staging area.
- Staged (preparado): arquivo modificado foi adicionado à staging area com git add e está pronto para ser incluído no próximo commit.
- Committed (comitado): arquivo e suas alterações foram armazenados no history do repositório, ou seja, o commit foi finalizado e o arquivo está sincronizado com a última versão registrada.
- Unmodified (não modificado): arquivo rastreado pelo Git que não possui alterações em relação ao último commit; é o estado após um commit bem-sucedido.

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

### Principais Transições

- Untracked → Staged: um arquivo novo começa não rastreado (untracked). Quando você executa git add <arquivo>, ele passa para o estado staged, pronto para receber commit.
- Unmodified → Modified: quando um arquivo que já está sob controle do Git é alterado no working directory, ele sai do estado unmodified (sem alterações) e entra em modified (modificado).
- Modified → Staged: com git add <arquivo>, o arquivo modificado vai para staged, ou seja, está preparado para ser salvo no próximo commit.
- Staged → Committed: após o comando git commit, todas as alterações em arquivos staged são registradas no histórico do repositório; os arquivos voltam ao estado unmodified (sem alterações em relação ao último commit).

### Outras transições relevantes

- Staged → Modified: se modificar novamente um arquivo que está na área staged, ele volta para modified, pois a nova alteração ainda não está preparada para commit.
- Staged → Unstaged: usando git reset <arquivo> ou git restore --staged <arquivo>, o arquivo é removido da área staged e retorna ao estado modified.
- Modified → Unmodified: se descartar as modificações (por exemplo, com git restore <arquivo>), o arquivo retorna ao estado anterior ao commit, ficando unmodified.

## 2. Prática com Git Local

Execute os comandos a seguir e responda às perguntas baseadas no resultado do terminal.

### Etapa 1 – Criar o repositório

```
mkdir aula-git
cd aula-git
git init
```

Qual foi a mensagem exibida após o comando git init e o que ela significa na prática?

Initialized empty Git repository in C:/Users/isabella.batista/aula-git/.git/

Essa mensagem significa, na prática, que o Git criou um subdiretório oculto chamado .git dentro da pasta aula-git, iniciando ali um repositório vazio. A partir desse momento, o projeto passa a ser rastreado, permitindo controle de versão, registros de alterações e uso dos demais comandos do Git.

## Etapa 2 – Adicionar arquivo e fazer commit

```
echo "Primeiro arquivo" > arquivo.txt
git status
git add arquivo.txt
git commit -m "Primeiro commit"
```

### Perguntas:

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

Antes do git add, o arquivo está no estado untracked (não rastreado), ou seja, é um arquivo novo no working directory que o Git ainda não acompanha.

Após o git add, o arquivo muda para o estado staged (preparado), pois agora está na área de staging, pronto para ser incluído no próximo commit.

#### 2. O que significa o estado untracked e tracked?

**Untracked:** arquivos que existem no diretório de trabalho, mas ainda não foram adicionados ao controle de versão pelo Git, portanto não estão sendo monitorados.

**Tracked:** arquivos que o Git já está monitorando, ou seja, que foram previamente adicionados e registrados em commits anteriores; eles podem estar modificados ou não.

#### 3. Qual o objetivo do git commit?

O git commit salva oficialmente as alterações preparadas (staged) no repositório local, criando um snapshot que registra o estado dos arquivos naquele momento e adicionando essas alterações ao histórico do projeto.

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

Depois do commit, o arquivo passa para o estado unmodified (não modificado), pois o Git o reconhece como estando sincronizado com a última versão registrada no histórico do repositório.

## Etapa 3 – Histórico e alterações

```
git log --oneline
echo "Nova linha" >> arquivo.txt
git diff
```

### Perguntas:

#### 1. O que o comando git diff mostra?

O comando git diff mostra as diferenças entre o conteúdo atual do working directory (diretório de trabalho) e o último estado registrado no repositório (HEAD) ou entre o working directory e a área de staging. Ele apresenta linha a linha o que foi adicionado ou removido em arquivos modificados, permitindo visualizar exatamente as alterações ainda não comitadas.

#### 2. Qual commit está atualmente apontado por HEAD?

O commit atualmente apontado por HEAD é o commit mais recente no branch ativo. Ou seja, HEAD refere-se ao snapshot atual do repositório onde o trabalho local está baseado e onde novas alterações serão aplicadas no próximo commit.

## Etapa 4 – Trabalhando com Branches

```
git branch nova-feature
git checkout nova-feature
echo "Linha da nova branch" >> arquivo.txt
git add arquivo.txt
git commit -m "Alteração na nova branch"
```

### Perguntas:

#### 1. Como verificar em qual branch você está?

Para verificar em qual branch você está, você pode usar o comando git branch. Ele lista todas as branches locais e marca com um asterisco (\*) a branch atual. Outra forma mais direta é usar git branch --show-current, que exibe apenas o nome da branch atual.

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

Se você rodar git merge nova-feature estando na branch principal, o Git irá integrar as alterações presentes na branch "nova-feature" na branch principal. Isso significa que todas as mudanças comitadas na "nova-feature" serão incorporadas à branch principal, unificando o histórico e o conteúdo entre as duas ramas.

### 3. Conectando ao GitHub

1. Crie um repositório vazio no GitHub chamado aula-git.
2. Conecte o repositório local ao remoto:

```
git remote add origin https://github.com/<usuario>/aula-git.git  
git branch -M main  
git push -u origin main
```

#### Perguntas:

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

O parâmetro -u no comando git push -u origin main significa --set-upstream. Ele configura o branch remoto (neste caso, origin/main) como o branch upstream padrão para o branch local (main), estabelecendo uma associação entre eles.

##### 2. Como verificar os remotes configurados no repositório?

Para verificar os remotes configurados no repositório Git, você pode usar o comando **git remote -v**. Esse comando lista os nomes dos remotes configurados junto com suas URLs para operações de fetch e push, mostrando todos os repositórios remotos vinculados ao seu repositório local.

### 4. Encerramento e Discussão

#### Qual etapa foi mais difícil?

A etapa mais difícil foi gerenciar corretamente os estados dos arquivos (working directory, staging area e commits) e as transições entre eles, além de aprender a trabalhar com branches e merges, pois envolve conceitos abstratos e comandos que precisam ser dominados para evitar conflitos e perdas de trabalho.

#### Como o Git ajuda na colaboração?

O Git ajuda na colaboração ao permitir que múltiplos desenvolvedores trabalhem simultaneamente no mesmo projeto, com cada um tendo uma cópia completa do repositório local. O sistema distribuído do Git, aliado ao uso de branches, possibilita que mudanças sejam feitas independentemente e depois integradas de forma organizada, com controle de histórico e facilidade para resolver conflitos.

#### Que diferença faz ter um repositório remoto?

Ter um repositório remoto faz grande diferença porque centraliza o código e o histórico, permitindo a sincronização entre diferentes colaboradores. O remoto atua como referência principal para compartilhar, revisar e integrar mudanças, garantindo backup seguro dos dados, facilitando a colaboração remota e possibilitando operações de integração contínua e deploy automatizado em ambientes modernos de desenvolvimento.

Reinitialized existing Git repository in C:/Users/isabella.batista/aula-git/.git/

C:\Users\isabella.batista\aula-git>echo "Primeiro arquivo" > arquivo.txt

C:\Users\isabella.batista\aula-git>git status  
On branch main

No commits yet

Untracked files:  
(use "git add <file>..." to include in what will be committed)  
arquivo.txt

nothing added to commit but untracked files present (use "git add" to track)

C:\Users\isabella.batista\aula-git>git add arquivo.txt  
fatal: pathspec 'arquivo.txt' did not match any files

C:\Users\isabella.batista\aula-git>git commit -m "Primeiro commit"  
On branch main

Initial commit

Untracked files:  
(use "git add <file>..." to include in what will be committed)  
arquivo.txt

nothing added to commit but untracked files present (use "git add" to track)

C:\Users\isabella.batista\aula-git>git add arquivo.txt

C:\Users\isabella.batista\aula-git>git commit -m "Primeiro commit"  
[main (root-commit) c0c2c90] Primeiro commit  
1 file changed, 1 insertion(+)  
create mode 100644 arquivo.txt

C:\Users\isabella.batista\aula-git>git log --oneline  
c0c2c90 (HEAD -> main) Primeiro commit

C:\Users\isabella.batista\aula-git>echo "Nova linha" >> arquivo.txt

C:\Users\isabella.batista\aula-git>git diff  
diff --git a/arquivo.txt b/arquivo.txt  
index a602fde..b16172b 100644  
--- a/arquivo.txt  
+++ b/arquivo.txt  
@@ -1,2 @@  
 "Primeiro arquivo"  
+"Nova linha"

C:\Users\isabella.batista\aula-git>git branch nova-feature

C:\Users\isabella.batista\aula-git>git checkout nova-feature  
M arquivo.txt  
Switched to branch 'nova-feature'

C:\Users\isabella.batista\aula-git>echo "Linha da nova branch" >> arquivo.txt

C:\Users\isabella.batista\aula-git>git add arquivo.txt

C:\Users\isabella.batista\aula-git>git commit -m "Alteração na nova branch"  
[nova-feature da60f06] Alteração na nova branch  
1 file changed, 2 insertions(+)

C:\Users\isabella.batista\aula-git>git remote add origin https://github.com/bellasz13/aula-git.git

C:\Users\isabella.batista\aula-git>git branch -M main

C:\Users\isabella.batista\aula-git>git push -u origin main  
info: please complete authentication in your browser...  
Enumerating objects: 6, done.  
Counting objects: 100% (6/6), done.  
Delta compression using up to 4 threads  
Compressing objects: 100% (3/3), done.  
Writing objects: 100% (6/6), 523 bytes | 523.00 KiB/s, done.  
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)  
To https://github.com/bellasz13/aula-git.git  
\* [new branch] main -> main  
branch 'main' set up to track 'origin/main'.

C:\Users\isabella.batista\aula-git>\_



## Resumo dos Comandos

Ação	Comando
Inicializar repositório	<code>git init</code>
Ver status	<code>git status</code>
Adicionar arquivo	<code>git add</code>
Fazer commit	<code>git commit -m ""</code>
Ver histórico	<code>git log --oneline</code>
Comparar alterações	<code>git diff</code>
Criar branch	<code>git branch nome</code>
Mudar branch	<code>git checkout nome</code>
Unir branch	<code>git merge nome</code>
Conectar remoto	<code>git remote add origin URL</code>
Enviar commits	<code>git push</code>
Atualizar repositório	<code>git pull</code>