🕑 1. Introdução e Conceitos

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

O Git é distribuído porque cada desenvolvedor tem uma cópia completa do repositório na sua máquina, não apenas os arquivos atuais, mas todo o histórico de versões também! Isso é diferente de sistemas centralizados onde só o servidor central tem tudo. Então se o servidor cair, é possível continuar trabalhando normalmente, fazendo commits locais, e depois sincroniza quando voltar.

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

Working Directory é a pasta onde o desenvolvedor realmente trabalha, edita seus arquivos, e escreve o código e realiza alterações.
Staging Area é uma "área de preparação". Quando o desenvolvedor finaliza uma modificação que está pronta para ser entregue, ele sobe seus arquivos para a Staging Area. É onde que foi selecionado para o próximo commit é adicionado.
Já o repository é onde o Git guarda permanentemente o histórico de commits. Quando o comando git commit é feito, o que está na staging area vai pra lá de forma definitiva.

3. Para que serve o comando git clone ?

O git clone serve pra copiar um repositório remoto inteiro pra minha máquina. Ele baixa todos os arquivos, todo o histórico de commits, branches, tags... tudo! É geralmente o primeiro comando que uso quando vou trabalhar num projeto existente.

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

Working Directory são os arquivos e pastas normais que vejo na minha pasta do projeto.
Staging Area fica no arquivo chamado index dentro da pasta .git/. É um arquivo binário que o Git gerencia.
Repository fica tudo dentro da pasta oculta .git/ na raiz do projeto. Lá tem as pastas objects/ (com os commits, árvores, blobs), refs/ (branches e tags), etc.

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

Untracked (não rastreado): arquivo novo que o Git ainda não conhece
Unmodified (não modificado): arquivo já commitado e sem alterações
Modified (modificado): arquivo que foi alterado mas ainda não foi pro stage
Staged (preparado): arquivo na staging area, pronto pro commit

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

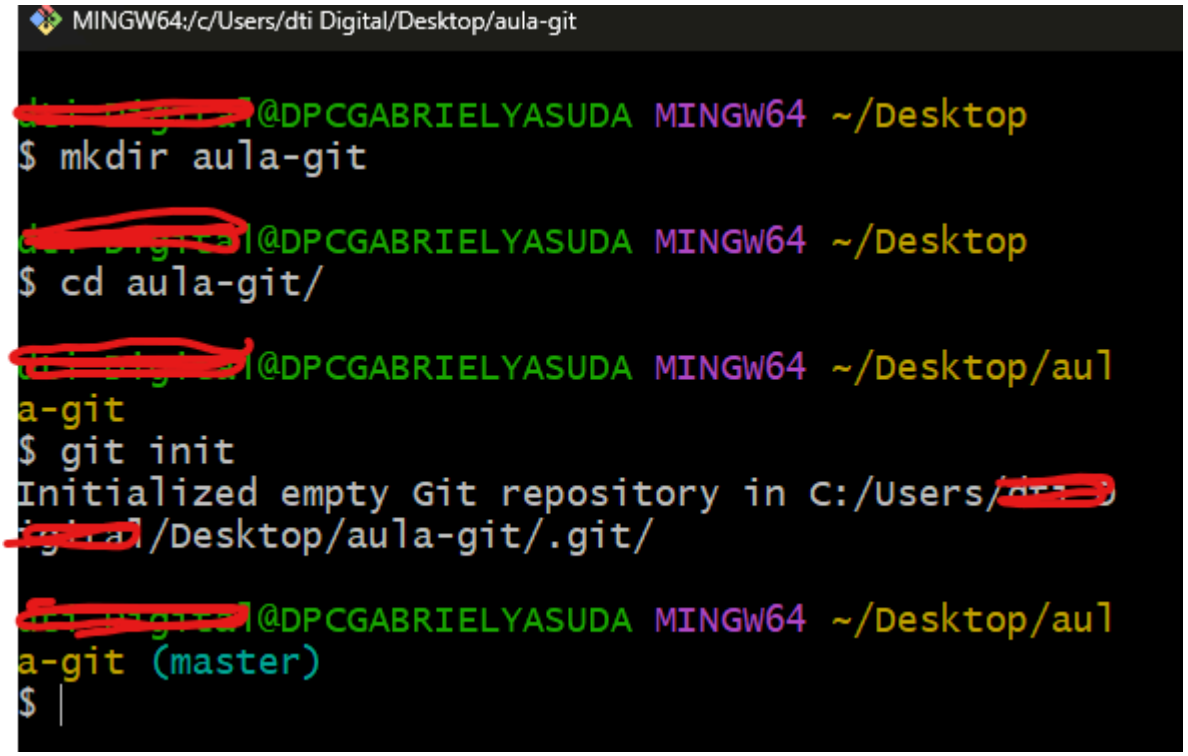
- Untracked -> Staged: uso git add arquivo num arquivo novo
- Unmodified -> Modified: simplesmente edito o arquivo
- Modified -> Staged: uso git add arquivo pra preparar a mudança
- Staged -> Unmodified: faço git commit e o arquivo entra no repositório
- Staged -> Modified: se eu modificar de novo um arquivo já staged
- Unmodified -> Untracked: se eu remover do rastreamento com git rm --cached
- Staged -> Unmodified: posso usar git reset HEAD arquivo pra tirar da staging

💻 2. Prática com Git Local

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

“Initialized empty Git repository in <caminho da pasta>”

Isso significa que foi criado um repositório vazio git. A partir desse momento posso utilizar o git para versionar e gerenciar o conteúdo desse diretório.



1. Qual o estado do arquivo antes e depois do git add ?
Antes do git add o arquivo estava Untracked (não rastreado)
Depois do git add o arquivo passou para Staged (preparado/na staging area)

2. O que significa o estado untracked e tracked ?
Untracked é um arquivo que o Git não conhece ainda, não tá sendo monitorado. Geralmente são arquivos novos que acabei de criar. O Git ignora eles até eu fazer git add. Aparecem em vermelho no git status (na maioria dos terminais)

Tracked são arquivos que o Git já conhece e monitora. Pode estar em 3 sub-estados: unmodified, modified ou staged. O Git detecta qualquer mudança neles. Aparecem em verde quando staged ou em vermelho quando modified

3. Qual o objetivo do git commit ?
O git commit serve para salvar permanentemente as mudanças que estão na staging area no histórico do repositório. É como tirar uma "foto" do projeto naquele momento.

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

Após o git commit, o arquivo fica no estado Unmodified (não modificado) e Tracked. Isso significa que o arquivo agora faz parte do repositório oficialmente e será monitrado.

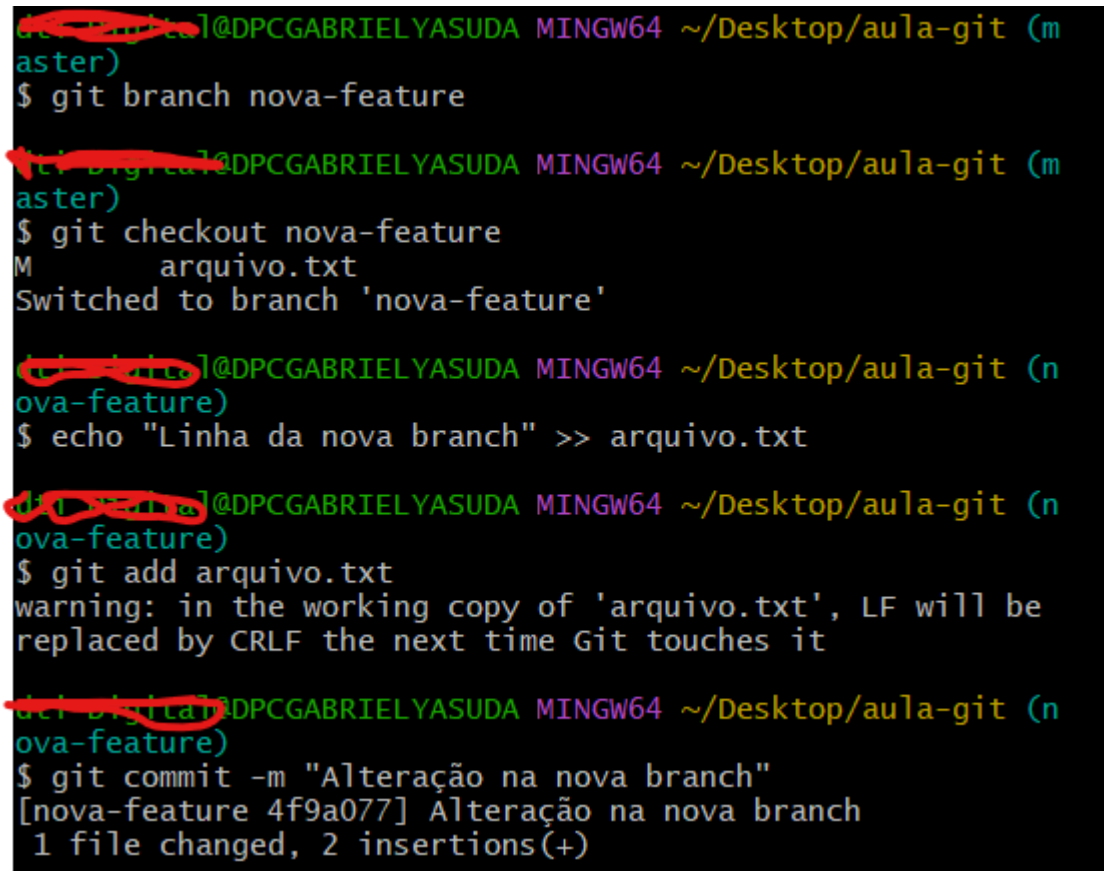
1. O que o comando git diff mostra?
O git diff mostra as diferenças/mudanças entre o working directory e a staging area (ou o último commit se não tiver nada staged).

```
$ git log --oneline
2c54b00 (HEAD -> master) Primeiro commit

[redacted]@DPCGABRIELYASUDA MINGW64 ~/Desktop/aula-git (master)
$ echo "Nova linha" >> arquivo.txt

[redacted]@DPCGABRIELYASUDA MINGW64 ~/Desktop/aula-git (master)
$ git diff
warning: in the working copy of 'arquivo.txt', LF will be replaced by CRLF
 the next time Git touches it
diff --git a/arquivo.txt b/arquivo.txt
index 4ecc55c..376418e 100644
--- a/arquivo.txt
+++ b/arquivo.txt
@@ -1 +1,2 @@
 Primeiro arquivo
+Nova linha
```

2. Qual commit está atualmente apontado por HEAD?
O HEAD está apontando para o "Primeiro commit".

1. Como verificar em qual branch você está?
Utilizando o git branch.

```
[redacted]@DPCGABRIELYASUDA MINGW64 ~/Desktop/aula-git (n
ova-feature)
$ git branch
  master
* nova-feature
```



2. O que acontece se você rodar git merge nova-feature estando na branch principal?
As alterações da nova-feature vão ser trazidas para a branch principal. Portanto, ambas as branches (master e nova-feature) estão apontando para o mesmo commit, e o arquivo.txt na branch principal também vai ter "Linha da nova branch".

1. O que significa o -u no comando git push -u origin main?
O `-u` é uma abreviação de `--set-upstream`.

Ele serve para criar um vínculo entre a minha branch local e a branch remota.

2. Como verificar os remotes configurados no repositório?
Através dos comandos: git remote, git remove -v ou git remote show origin.

Qual etapa foi mais difícil?
A etapa mais díficil foi a etapa de configuração da upstream, ou o vínculo da local com a remota, pois tive dificuldades em autenticar o repositório.

Como o Git ajuda na colaboração?
Em uma equipe com diversos desenvolvedores, cada um pode trabalhar numa branch diferente sem atrapalhar o trabalho dos outros. O desenvolvedor X pode estar fazendo uma feature enquanto o colega Y corrige um bug, e ninguém quebra o código de ninguém.

Que diferença faz ter um repositório remoto?
Ter um repositório remoto garante a integridade digital do código fonte. Caso algum desenvolvedor faça alguma alteração que prejudique o funcionamento do software, é possível reverter, já que o repositório remoto está disponibilizado e intacto. Além disso, ele serve como backup, caso a máquina do desenvolvedor perca seu armazenamento local.