Atividades - Aula Prática: Git e GitHub

1) Introdução e Conceitos

a) Por que o Git é considerado um sistema de controle de versão distribuído?
RESPOSTA:

- Git é distribuído porque cada clone contém o repositório completo, com todo o histórico e branches.
- É possível commitar, branchar e fazer merge localmente, trabalhando offline sem servidor central obrigatório.
- A colaboração ocorre via push/pull entre repositórios remotos, permitindo múltiplos fluxos de trabalho.
- Isso traz redundância, desempenho e tolerância a falhas.

b) Qual a diferença entre working directory, staging area e repository?
RESPOSTA:

- Working directory: seus arquivos no disco; onde você edita e vê mudanças não preparadas.
- Staging area (index): área intermediária; com git add você seleciona exatamente o que vai no próximo commit.
- Repository (.git): banco de dados local com histórico, branches e tags; git commit grava o que está no staging aqui.

c) Para que serve o comando git clone?
RESPOSTA:

- git clone cria uma cópia local de um repositório (remoto ou local), com todo o histórico, branches e tags.
- Configura o repositório remoto “origin” para você fazer pull/push.
- Permite clonar um branch específico ou de forma rasa (opções --branch e --depth).
- É o passo inicial para começar a trabalhar num projeto existente.

d) Onde estão implementados fisicamente working directory, staging area e repository?
RESPOSTA:

- Working directory: os arquivos do projeto no seu sistema de arquivos (cópia de trabalho).
- Staging area (index): arquivo .git/index dentro do diretório oculto .git.
- Repository (local): o diretório .git (objetos em .git/objects, refs em .git/refs).

e) Quais os estados de um arquivo no repositório do git?
RESPOSTA:

- Untracked: arquivo não rastreado (ou ignorado pelo .gitignore).
- Tracked — Unmodified: rastreado e sem mudanças em relação ao último commit.
- Modified → Staged: alterado na working tree e, após add, vai para o index (pronto para commit).
- Committed: gravado no histórico do repositório.

f) **Explique as possíveis transições de estado de um arquivo no repositório do git?**
   - Estados: untracked/ignored, unmodified (committed), modified (working tree), staged (index), unmerged (conflito), deleted/renamed.
   - Fluxo típico: untracked → git add → staged → git commit → unmodified → editar → modified → git add → staged → commit.
   - Reversões: staged —(git reset)→ modified; modified —(git restore/checkout)→ unmodified; untracked —(git clean)→ removido.
   - Remoções/renomes: git rm/git mv marcam como staged (delete/rename) → commit; conflitos ficam “unmerged” até resolver (add) e commitar.

2) Prática com Git Local

a) **Qual foi a mensagem exibida após o comando git init e o que ela significa na prática?**
   RESPOSTA: “Initialized empty Git repository in C:/Users/richa/Programming/Outros/aula-git/.git/”
   Isso significa que o Git criou a pasta oculta .git nesse caminho e passou a versionar o diretório; o repositório está vazio (sem commits e sem remoto configurado) e o HEAD aponta para a branch padrão (geralmente main, conforme sua configuração). A partir daí será possível usar comandos git no repositório, como git add, git commit, etc.

b) **Qual o estado do arquivo antes e depois do git add?**
   RESPOSTA: O estado do arquivo antes do git add é “untracked” e após o git add é “tracked”.

c) **O que significa o estado untracked e tracked?**
   RESPOSTA:
   - Untracked: arquivo na working tree que o Git não monitora; não entra em commits até você usar git add (pode ser ignorado via .gitignore).
   - Tracked: arquivo conhecido pelo Git (já adicionado/commitado); pode estar unmodified, modified ou staged e é considerado nos commits.
   - Transição: git add torna um arquivo tracked; git rm --cached (ou .gitignore + clean) faz o Git parar de rastreá-lo.

d) **Qual o objetivo do git commit?**
   RESPOSTA:
   - Registrar um snapshot das mudanças preparadas (staged) no histórico local do repositório.
   - Criar um ponto de restauração com mensagem e metadados (autor, data, hash) para rastreabilidade.
   - Consolidar e versionar o estado do projeto, permitindo comparar, reverter, ramificar e mesclar.
   - Servir de unidade básica de colaboração a ser compartilhada via push/pull com remotos.

e) **Qual o estado do arquivo após o git commit?**
   RESPOSTA:

- Após git commit, os arquivos que estavam staged passam a “committed” e ficam “unmodified” (rastreados) no working tree.
- O índice (staging area) é limpo e o HEAD aponta para o novo commit.
- git status normalmente mostra “nothing to commit, working tree clean”.

3) Histórico e alterações

a) **O que o comando git diff mostra?**
RESPOSTA:
- Mostra as diferenças entre duas versões de arquivos: linhas adicionadas (+) e removidas (-).
- Sem opções, exibe o que você mudou e ainda não preparou para commit (unstaged) em relação ao último commit.
- Pode comparar commits, branches ou o que está preparado (git diff --staged).
- Serve para revisar o que vai entrar no commit antes de salvar.

b) **Qual commit está atualmente apontado por HEAD?**
RESPOSTA:
- O commit “Primeiro commit”

4) Trabalhando com Branches

a) **Como verificar em qual branch você está?**
RESPOSTA:
- Ao enviar o comando “git branch” uma lista com todos os branches é exibida.
- Aquele que tiver um asterisco (*) à esquerda é a branch atual.

b) **O que acontece se você rodar git merge nova-feature estando na branch principal?**
RESPOSTA:
- Na branch principal, git merge nova-feature traz as mudanças do branch “nova-feature” para ela.
- Como só a nova-feature tem um commit a mais, o Git normalmente só adianta a principal para incluir essa mudança (sem criar commit extra).
- O arquivo.txt passará a conter a linha: Linha da nova branch.
- Só haveria conflitos se ambas as branches tivessem alterado a mesma parte do arquivo.

5) Conectando ao GitHub

a) **O que significa o -u no comando git push -u origin main?**
RESPOSTA:

- git push envia (faz upload) os seus commits do repositório local para o repositório remoto (por exemplo, no GitHub).
- O -u (ou --set-upstream) faz com que a sua branch local "main" fique ligada à branch remota "origin/main".
- Com isso, em futuros pushes/pulls você pode apenas digitar git push ou git pull sem especificar remote/branch.
- Também permite ao Git mostrar status indicando a diferença entre sua branch local e a remota.

b) Como verificar os remotes configurados no repositório?
RESPOSTA:

- O comando “git remote -v” lista os remotes (ex.: origin) e suas URLs para fetch/push.

6) Encerramento e Discussão

a) Qual etapa foi mais difícil?
RESPOSTA:
A etapa de trabalhar com múltiplos branches, pois geralmente com muitos colaboradores quando dá conflito é uma das partes mais custosas de se resolver.

b) Como o Git ajuda na colaboração?
RESPOSTA:

- Git permite que cada pessoa trabalhe numa cópia do projeto e salve mudanças com histórico e autoria.
- Com branches, cada um desenvolve novidades separadamente sem atrapalhar os outros.
- Push/pull sincronizam mudanças entre computadores e servidores para juntar o trabalho.
- O histórico registra quem fez o quê, facilita revisões, resolução de conflitos e voltar atrás em erros.

c) Que diferença faz ter um repositório remoto?
RESPOSTA:

- Um repositório remoto é uma cópia do seu projeto hospedada na internet (ex.: GitHub) que funciona como ponto central para guardar o código.
- Ele permite que várias pessoas colaborem trocando mudanças (push/pull) e mantém um backup acessível de qualquer lugar.
- Também facilita automatizar testes/deploy e sincronizar trabalho entre diferentes computadores.