Atividade avaliativa – Git e GitHub

Parte 1:

1. Principais comandos do Git e suas siglas

O Git é um sistema de **controle de versão distribuído**, que permite registrar o histórico de um projeto, colaborar em equipe e restaurar versões anteriores.

Comandos fundamentais

Comando	Descrição	Significado / Sigla
`git init`	Cria um novo repositório Git no diretório atual.	*init = initialize*
`git status`	Mostra o estado atual dos arquivos (modificados, adicionados, etc).	—
`git add nome-do-arquivo ou git add .`	Adiciona arquivos à **área de stage**.	*add = adicionar*
`git commit -m "mensagem"`	Cria um ponto de salvamento (commit) com mensagem descritiva.	*commit = confirmar/enviar*
`git log`	Mostra o histórico de commits.	*log = registro*
`git branch`	Lista ou cria branches (ramificações).	*branch = ramo*
`git checkout nome-da-branch`	Muda para outra branch.	*checkout = trocar/verificar*
`git merge nome-da-branch`	Mescla o conteúdo de uma branch na atual.	*merge = unir*
`git push origin nome-da-branch`	Envia alterações locais para o repositório remoto.	*push = empurrar/enviar*
`git pull`	Puxa (baixa) as alterações do repositório remoto.	*pull = puxar/atualizar*
`git clone url`	Copia um repositório remoto para a máquina local.	*clone = duplicar/cópia*
`git tag -a v1.0.0 -m "mensagem"`	Marca um ponto importante no histórico (como uma versão).	*tag = etiqueta/marcação*
`git graph`	Exibe um **gráfico visual** no terminal mostrando as branches, merges e histórico de commits. (Usado geralmente com `git log --graph --oneline --decorate --all`)	-
`git branch -D feature/login`	Excluir uma branch	O `-D` ignora verificações de merge — use apenas quando tiver certeza de que o código foi descartado ou é irrelevante.

2. Atalhos, siglas e boas práticas

Atalhos de opções

Atalho	Forma longa	O que faz
`-m`	`--message`	Adiciona uma mensagem ao commit
`-b`	`--branch`	Cria uma nova branch
`-u`	`--set-upstream`	Liga uma branch local com a remota
`-a`	`--annotate`	Cria uma tag com anotação
`-M`	—	Força a renomeação de uma branch existente
`-v`	`--verbose`	Exibe mais detalhes no comando

Boas práticas de Git

- **main** → branch principal (código estável e publicado)
- **develop** → onde o desenvolvimento ativo acontece
- **feature/** → novas funcionalidades
- **hotfix/** → correções urgentes em produção
- **release/** → preparação para o lançamento de nova versão

Exemplo de criação de branch de funcionalidade:

```
git checkout -b feature/tela-login develop
```

---

3. Exercício prático — Projeto com HTML, CSS e JS + Git

Cenário:

Desenvolver uma **página de login com tema escuro**, validação de campos via JavaScript e controle de versões com Git.
A página deve exibir mensagens de erro quando o usuário tentar fazer login com campos vazios.

Passo 1 — Estrutura inicial

Crie os arquivos:

```
index.html
style.css
script.js
```

---

index.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Tela de Login</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="login-container">
    <h1>Login</h1>
    <form id="loginForm">
      <input type="text" id="usuario" placeholder="Usuário">
      <input type="password" id="senha" placeholder="Senha">
      <button type="submit">Entrar
      <p id="mensagem"></p>
    </form>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

---

style.css

```
body {
  font-family: Arial, sans-serif;
  background-color: #121212;
  color: #fff;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}

.login-container {
  background-color: #1f1f1f;
  padding: 30px;
  border-radius: 10px;
  box-shadow: 0 0 15px rgba(255, 255, 255, 0.1);
  width: 300px;
}

.login-container input {
  width: 100%;
  margin: 10px 0;
  padding: 10px;
  border: none;
  border-radius: 5px;
}

button {
  width: 100%;
  padding: 10px;
  background-color: #03dac6;
  color: #000;
  border: none;
```

```
  border-radius: 5px;
  font-weight: bold;
  cursor: pointer;
}

button:hover {
  background-color: #00bfa5;
}

#mensagem {
  margin-top: 10px;
  color: #ff5252;
  font-weight: bold;
}
```

Inicialize o repositório:

```
git init
git branch -M main
git add .
git commit -m "Estrutura inicial do projeto"
```

Passo 2 — Controle de versão

Versionar o progresso (add / commit):

Criar a branch develop (branch mãe)

Criar uma nova branch de validação (checkout):

Adicionar a validação de campos e depois unir ao projeto principal (add script / commit / checkout main /merge feature validação / push)

script.js

```
const form = document.getElementById("loginForm");
const usuario = document.getElementById("usuario");
const senha = document.getElementById("senha");
const mensagem = document.getElementById("mensagem");

form.addEventListener("submit", (e) => {
  e.preventDefault(); // evita o recarregamento da página

  if (usuario.value === "" || senha.value === "") {
    mensagem.textContent = "Preencha todos os campos!";
  } else if (usuario.value === "adm" && senha.value === "123") {
    mensagem.style.color = "#03dac6";
    mensagem.textContent = "Login realizado com sucesso!";
  } else {
```

```
    mensagem.textContent = "Usuário ou senha incorretos.";
  }
});
```

Parte 2

Adicionar 4 novas branches no projeto

1 feature/dashboard

Responsável por criar o **painel principal** do site, exibindo informações após o login (checkout feature/dashboard develop).

Objetivo: criar `dashboard.html` com um menu simples e mensagem de boas-vindas ao usuário autenticado.

---

2 feature/cadastro-usuario

Para implementar o **formulário de cadastro** de novos usuários (checkout feature/cadastro-usuario develop).

Objetivo: adicionar `cadastro.html`

---

3 hotfix/erro-html

Para corrigir um **erro no código HTML** detectado na página de login (`index.html`), onde havia tags mal fechadas ou estrutura incorreta (checkout -b hotfix/erro-html main).

Objetivo: corrigir a estrutura HTML do arquivo principal, garantindo que a página siga o padrão semântico correto e funcione adequadamente nos navegadores.

Exemplo de correção no `index.html`:

Antes (erro comum):

```
<form id="loginForm">
  <input type="text" id="usuario" placeholder="Usuário">
  <input type="password" id="senha" placeholder="Senha">
  <button type="submit">Entrar
</form>
```

O erro está na **tag `<button>` sem fechamento adequado**.

Após correção:

```
<form id="loginForm">
  <input type="text" id="usuario" placeholder="Usuário">
  <input type="password" id="senha" placeholder="Senha">
  <button type="submit">Entrar</button>
</form>
```

Comando para versionar a correção (add / commit / checkout main / merge hotfix/erro-html / push)

---

4 release/v1.1.0

Para preparar o **lançamento da nova versão** do sistema com todas as funcionalidades integradas (checkout release / versão develop).

Objetivo: revisar o código, ajustar detalhes visuais e preparar o `README.md` com a documentação do projeto.

Resumo final

Tipo de Branch	Nome	Base	Finalidade
Feature	feature/dashboard	develop	Criar painel principal
Feature	feature/cadastro-usuario	develop	Criar tela de cadastro
Hotfix	hotfix/erro-html	main	Corrigir erro estrutural no HTML
Release	release/v1.1.0	develop	Preparar nova versão para publicação

Parte 3:

Responder o questionário:

https://forms.office.com/r/He5qzQ3ESX

Entregar no Teams arquivo em PDF contendo:

Link do repositório

Link do site hospedado

Todos os comandos solicitados