Sumário

[Aula 1: Introdução ao Git, GitHub e VSCode 3](#_Toc190546903)

[O que é Git? 3](#_Toc190546904)

[O que é GitHub? 3](#_Toc190546905)

[O que é Visual Studio Code (VSCode)? 3](#_Toc190546906)

[Instalação do Git e Git Bash 3](#_Toc190546907)

[Configuração do Visual Studio Code (VSCode) 3](#_Toc190546908)

[Aula 2: Conceitos Básicos de Git 5](#_Toc190546909)

[Versionamento 5](#_Toc190546910)

[Repositório 6](#_Toc190546911)

[Commit 7](#_Toc190546912)

[Branch 7](#_Toc190546913)

[Criar uma Branch 7](#_Toc190546914)

[Deletar uma Branch 7](#_Toc190546915)

[Para renomear uma branch existente 8](#_Toc190546916)

[Merge 8](#_Toc190546917)

[Aula 3: Trabalhando com GitHub 10](#_Toc190546918)

[Clonar Repositórios 10](#_Toc190546919)

[Forkar Repositórios 10](#_Toc190546920)

[Pull Requests 11](#_Toc190546921)

[Aula 4: Comandos Avançados de Git 11](#_Toc190546922)

[Comandos Avançados 11](#_Toc190546923)

[Resolvendo Conflitos 12](#_Toc190546924)

[Aula 5: Colaboração e Fluxo de Trabalho 12](#_Toc190546925)

[Fluxo de Trabalho Colaborativo 13](#_Toc190546926)

[**Branches Comuns** 13](#_Toc190546927)

[**Exemplo de Fluxo de Trabalho** 13](#_Toc190546928)

[**Comandos** 13](#_Toc190546929)

[Aula 6: Projeto Final 14](#_Toc190546930)

[Criando um Projeto 14](#_Toc190546931)

[Versionamento Completo 14](#_Toc190546932)

[Publicação no GitHub 14](#_Toc190546933)

[Exercício de Colaboração em Squad 15](#_Toc190546934)

[**Como fazer** 15](#_Toc190546935)

[**Passos para Criar um Pull Request** 15](#_Toc190546936)

[**Passos para Revisar um Pull Request** 16](#_Toc190546937)

[**Passos para Consolidar os Exercícios** 16](#_Toc190546938)

[Dicas e Boas Práticas 16](#_Toc190546939)

[Recursos Adicionais 17](#_Toc190546940)

[Perguntas Frequentes (FAQ) 18](#_Toc190546941)

[Exercícios Adicionais 21](#_Toc190546942)

[Resolução de Problemas 21](#_Toc190546943)

[Conclusão 22](#_Toc190546944)

# Aula 1: Introdução ao Git, GitHub e VSCode

**Objetivos:**

* Compreender o que é Git e GitHub.
* Instalar Git e configurar o ambiente de desenvolvimento.
* Compreender o que é o Visual Studio Code (VSCode) e como utilizá-lo.

**Conteúdo:**

## O que é Git?

Git é um sistema de controle de versão distribuído que permite rastrear mudanças no código-fonte durante o desenvolvimento de software. Ele ajuda a gerenciar o histórico de alterações e facilita a colaboração entre desenvolvedores.

**Por que usar Git?**

* **Controle de Versão:** Permite voltar a versões anteriores do código.
* **Colaboração:** Facilita o trabalho em equipe, permitindo que várias pessoas trabalhem no mesmo projeto simultaneamente.
* **Rastreamento de Alterações:** Mantém um histórico detalhado de todas as mudanças feitas no código.

## O que é GitHub?

GitHub é uma plataforma de hospedagem de código-fonte que utiliza Git para controle de versão. Ele permite que desenvolvedores armazenem, compartilhem e colaborem em projetos de software.

**Por que usar GitHub?**

* **Hospedagem de Código:** Armazena repositórios de código na nuvem.
* **Colaboração:** Ferramentas para revisão de código, pull requests e issues.
* **Integração:** Funciona bem com outras ferramentas de desenvolvimento e CI/CD.

## O que é Visual Studio Code (VSCode)?

Visual Studio Code (VSCode) é um editor de código-fonte desenvolvido pela Microsoft. Ele oferece suporte para várias linguagens de programação e integrações com ferramentas como Git e GitHub, tornando-o uma escolha popular entre desenvolvedores.

**Recursos do VSCode:**

* **Extensões:** Suporte para uma ampla variedade de extensões que aumentam a funcionalidade do editor.
* **Integração com Git:** Ferramentas integradas para controle de versão.
* **Depuração:** Ferramentas de depuração para várias linguagens de programação.

## Instalação do Git e Git Bash

1. Acesse o site oficial do Git: git-scm.com.
2. Baixe e instale a versão mais recente do Git.
3. Durante a instalação, habilite a opção "Git Bash Here" para facilitar o uso do terminal.

## Configuração do Visual Studio Code (VSCode)

1. Acesse o site oficial do VSCode: code.visualstudio.com.
2. Baixe e instale a versão mais recente do VSCode.
3. Abra o VSCode e instale a extensão "GitLens" para integração com Git.

**Exercício**

1. Baixe GIT e GIT Bash. Deixe habilitado "Git Bash Here" e a conexão entre Git Bash e VSCode.
2. Crie o diretório localmente: mkdir curso-git-e-github Se o diretório já existir, você verá uma mensagem de erro indicando que o diretório já existe. Nesse caso, você pode ignorar este passo.
3. Liste os arquivos e diretórios no diretório atual para verificar se ele foi criado corretamente: ls
4. Mude o diretório atual para o diretório curso-git-e-github que você acabou de criar: cd curso-git-e-github
5. Inicialize o repositório Git: git init Se o repositório já estiver inicializado, você verá uma mensagem indicando que o repositório Git foi reinicializado.
6. Verifique se você está no diretório correto: pwd Esse comando mostra o caminho do diretório atual. Certifique-se de que você está no diretório curso-git-e-github.
7. Verifique se o repositório Git foi inicializado: ls -Force Esse comando lista todos os arquivos, incluindo os ocultos. Se o repositório foi inicializado corretamente, você verá uma pasta chamada .git.
8. Configure seu nome de usuário do GitHub: git config --global user.name "Seu Nome"
9. Configure seu email do GitHub: git config --global user.email "seu-email@example.com"
10. Esses comandos garantem que suas informações de usuário do GitHub estejam vinculadas ao Git.
11. Verifique se as informações de usuário e email do GitHub estão vinculadas ao Git: git config --list . Esse comando mostrara as informações configuradas. Certifique-se de que o nome de usuário e o email estão corretos.
12. Verifique o status do repositório: git status Esse comando mostra o estado atual do repositório. Se tudo estiver certo, você verá uma mensagem indicando que você está em uma nova branch (geralmente main ou master) e que não há commits ainda.
13. Liste os arquivos e diretórios no diretório atual: ls Você verá uma lista de todos os itens no diretório onde você está.
14. Mude para o diretório curso-git-e-github: cd curso-git-e-github Certifique-se de que esse diretório existe no local onde você está.
15. Atalho que abre o Visual Studio Code no diretório atual (curso-git-e-github): code . É uma maneira rápida de abrir o editor de código no diretório onde você está trabalhando.

Resolução de Problemas

Se você encontrar o erro indicando que o caminho não pode ser localizado, siga estes passos:

Verifique o caminho atual: Use o comando pwd para confirmar onde você está atualmente.

Liste os diretórios no caminho atual: Use o comando ls para listar os diretórios e arquivos no caminho atual. Verifique se o diretório curso-git-e-github está presente na lista.

Corrija o caminho: Certifique-se de que você está especificando o caminho correto. Se você estiver tentando acessar um subdiretório, verifique se ele realmente existe. Por exemplo, se você estiver no diretório C:\Users\letti, o comando correto para mudar para curso-git-e-github seria:

cd curso-git-e-github

Evite caminhos duplicados: Verifique se você não está repetindo o nome do diretório várias vezes. Por exemplo, o caminho C:\Users\letti\curso-git-e-github\curso-git-e-github\curso-git-e-github é incorreto se o diretório curso-git-e-github só existe uma vez.

Exemplo de comandos para verificar e corrigir o caminho:

* # Verifique o caminho atual pwd
* # Volte um nível no diretório cd ..
* # Liste os diretórios no nível anterior Ls
* # Navegue para o diretório correto cd curso-git-e-github

# Aula 2: Conceitos Básicos de Git

**Objetivos:**

* Compreender os conceitos de repositório, commit, branch e merge.
* Criar e inicializar um repositório Git.

**Conteúdo:**

## Versionamento

O versionamento de software é um processo de controle de versões, definido através de “numerações” de históricos diferentes. Nunca se repete a numeração. Isso permite que as pessoas desenvolvedoras saibam quando e quais alterações foram realizadas, acompanhando as mudanças aplicadas no software. O git controla esse histórico de versões de um mesmo código atribuindo uma “numeração” referente ao estado daquele código em determinado momento que foi salvo por uma pessoa específica. O git entende como salvar quando você aciona e comenta suas alterações. Não basta você salvar o arquivo para o git entender que essa mudança vai ser um commit que posteriormente vou enviar para o github. Vamos entender o conceito de "change" (alteração) e "diff" (diferença) no Git.

**O que o Git Entende como Alteração**

No Git, uma alteração pode ser:

Criação de Arquivos: Adicionar novos arquivos ao repositório.

Renomeação de Arquivos: Mudar o nome de arquivos existentes.

Exclusão de Arquivos: Remover arquivos do repositório.

Modificação de Arquivos: Inserir ou excluir linhas em um arquivo. Uma linha modificada é considerada tanto uma inserção quanto uma exclusão.

**Diferença (Diff)**

O termo "diff" refere-se às diferenças entre duas versões de um arquivo. O Git usa essas diferenças para rastrear as alterações. Quando você faz uma alteração em um arquivo, o Git compara a versão atual com a versão anterior e identifica as diferenças. Essas diferenças são chamadas de "diffs".

**Exemplo Prático**

Se você adicionar uma linha a um arquivo, o Git verá isso como uma inserção. Se você remover uma linha, o Git verá isso como uma exclusão. Se você modificar uma linha, o Git verá isso como uma exclusão da linha antiga e uma inserção da linha nova.

Aqui está um exemplo de como o Git mostra um diff:

- Linha antiga

+ Linha nova

No exemplo acima, a linha com o sinal de menos (-) foi removida, e a linha com o sinal de mais (+) foi adicionada.

**Comandos**

Para ver as diferenças entre a versão atual do seu código e a última versão commitada, você pode usar: git diff

## Repositório

É a pasta onde armazenamos o nosso projeto, dentro de um repositório os projetos podem se dividir em módulos e outras divisões especificas para cada tipo de projeto. Todo repositório tem um arquivo .git, este arquivo indica que este repositório é “rastreável” para o git. Ele contém todos os arquivos do projeto e o histórico de alterações.

**Comandos:**

* mkdir nome-do-repositorio: Cria um diretório.
* cd nome-do-repositorio: Entra no diretório.
* git init: Usar quando você está começando um novo projeto e deseja versionar o código com Git ou usar quando você tem um projeto existente que ainda não está versionado com Git e deseja começar a rastrear as mudanças.

**Exemplo Prático:**

mkdir meu-projeto

cd meu-projeto

git init

## Commit

Um commit é um conjunto de alterações no código. Ele registra as mudanças feitas no projeto. Toda vez que você desejar salvar/registrar as alterações no seu projeto, você comita essas alterações. Um commit tem as alterações que são realizadas nele e uma mensagem descritiva, além de informações sobre o autor, a data, etc.

**Comandos:**

* git add .: Adiciona todas as alterações ao índice.
* git commit -m "mensagem do commit": Realiza um commit com uma mensagem descritiva.

**Exemplo Prático:**

git add .

git commit -m "Adicionando arquivo index.html"

## Branch

Branches são ramificações do código que permitem o desenvolvimento paralelo. Uma branch é uma linha de desenvolvimento separada dentro do repositório, permitindo trabalhar em novas funcionalidades ou correções de bugs sem afetar a branch principal (main ou master). Uma pasta no repositório não é uma Branch, só é um diretório dentro do sistema de arquivos do repositório, usado para organizar arquivos. A equipe de desenvolvimento cria branches para trabalhar em novas soluções ou melhorar funcionalidades existentes de forma independente.

Tipos de Branches

* **Branch develop**: Usada durante o desenvolvimento. Após testar o código localmente, ele é disponibilizado nesta branch.
* **Branch homolog**: Após validação no ambiente de desenvolvimento, o código vai para a branch de homologação, onde é validado por pessoas de negócio e testado pela equipe de qualidade.
* **Branch master**: Branch principal do projeto, contendo o código que vai para produção, ou seja, o site disponibilizado ao público.

No Visual Studio Code (VSCode), você pode gerenciar branches usando a interface gráfica ou o terminal integrado.

### Criar uma Branch

* Para criar uma nova branch e mudar para ela: git checkout -b minha-nova-branch

### Deletar uma Branch

* **Para deletar uma branch local:** git branch -d nome-da-branch
* Se a branch ainda não foi mesclada e você deseja forçar a exclusão, use: git branch -D nome-da-branch
* **Para deletar uma branch remota:** git push origin --delete nome-da-branch

### **Para renomear uma branch existente**

1. Certifique-se de estar na branch que deseja renomear: git checkout nome-da-branch-antiga
2. Renomeie a branch: git branch -m novo-nome-da-branch

**Comandos Úteis**

* git branch nome-da-branch: Cria uma nova branch.
* git checkout nome-da-branch: Alterna para a branch especificada.
* Listar as branches atuais: git branch
* Listas as branches remotas: git branch -a

**Exemplo Prático**

git branch nova-feature

git checkout nova-feature

## Merge

Merge é a ação técnica de combinar mudanças de uma branch em outra e Pull Request é o processo colaborativo que envolve revisão, discussão e aprovação antes do merge. Quando você faz um merge, você está integrando o histórico de commits de uma branch na branch de destino. Isso é feito localmente no seu repositório. Um pull request (ou merge request em algumas plataformas) é uma solicitação para que as mudanças de uma branch sejam revisadas e, eventualmente, mescladas (merged) na branch de destino. O pull request é uma ferramenta de colaboração que permite:

* Revisão de Código: Outros membros da equipe podem revisar o código, sugerir melhorias e identificar problemas antes que as mudanças sejam integradas.
* Discussão: Facilita a comunicação e discussão sobre as mudanças propostas.
* Automação: Pode acionar testes automáticos e outras verificações antes do merge.

Fluxo de Trabalho

* Criação de uma Branch: Um desenvolvedor cria uma nova branch para trabalhar em uma funcionalidade ou correção.

git checkout -b nova-feature

* Desenvolvimento: O desenvolvedor faz commits na nova branch.

git add .

git commit -m "Implementa nova funcionalidade"

* Push para o Repositório Remoto: As mudanças são enviadas para o repositório remoto.

git push origin nova-feature

* Criação do Pull Request: No repositório remoto (por exemplo, GitHub), o desenvolvedor cria um pull request para solicitar a revisão e o merge das mudanças na branch principal (geralmente main ou master).
* Revisão e Aprovação: Outros desenvolvedores revisam o pull request, fazem comentários e, se necessário, solicitam mudanças.
* Merge: Após a aprovação, o pull request é mesclado na branch de destino. Isso pode ser feito através da interface do repositório remoto ou localmente.

git checkout main

git merge nova-feature

**Comandos:**

* git merge nome-da-branch: Realiza o merge da branch especificada com a branch atual.
* git checkout main: Realiza a trocha de branch

**Exemplo Prático:**

git checkout main

git merge nova-feature

**Exercício Prático:**

* + Crie um repositório local seguindo os comandos acima.
  + Adicione arquivos e realize commits.
  + Crie uma nova branch `apostila-2` e `apostila-git-github`
  + Troque para a branch `main`: git checkout main
  + Atualize a branch `main` com as últimas mudanças do repositório remoto: git pull origin main
  + Faça o merge da branch `apostila-2` na branch `main`: git merge apostila-2
  + Resolva quaisquer conflitos que possam surgir e finalizar o merge: git add .
  + git commit -m "Merge branch 'apostila-2' into main"
  + Repita o processo para a branch `apostila-git-github`: git merge apostila-git-github
  + Resolva conflitos, se houver, e finalizar o merge: git add .
  + git commit -m "Merge branch 'apostila-git-github' into main"
  + Envie as mudanças para o repositório remoto: git push origin main

# Aula 3: Trabalhando com GitHub

**Objetivos:**

* Compreender como clonar, forkar e fazer pull requests.
* Conectar repositórios locais ao GitHub.

**Conteúdo:**

## Clonar Repositórios

Clonar um repositório significa copiar um repositório remoto para o seu computador. Isso permite que você trabalhe localmente no projeto, podendo realizar as alterações e enviá-las para o github quando desejar.

**Comandos:**

* git clone https://github.com/usuario/nome-do-repositorio.git: Clona o repositório especificado.

**Exemplo Prático:**

Praticar o uso de comandos Git e de manipulação de diretórios no terminal, corrigindo um erro comum ao clonar repositórios.

1. Clone o repositório para o diretório errado: Primeiro, abra seu terminal. Navegue até o diretório onde você deseja clonar o repositório (por exemplo, sua pasta de projetos): cd ~/curso-git-e-github
2. Clone o repositório: git clone https://github.com/WoMakersCode/intro-ao-python
3. Liste os arquivos e diretórios: ls
4. Você verá o diretório ` curso-git-e-github ` listado
5. Percebeu que o repositório foi clonado no lugar errado
6. Voltar um nível no diretório: cd ..
7. Verificar o diretório atual: pwd
8. Crie o diretório correto onde o repositório deve estar: mkdir intro-ao-python
9. Use o comando `mv` para mover o diretório `intro-ao-python` para dentro de `curso-git-e-github`: mv intro-ao-python curso-git-e-github/
10. Navegue até o novo diretório para confirmar que o repositório foi movido corretamente: cd intro-ao-python
11. Verifique se o repositório está lá: ls
12. Entre no diretório do repositório: cd intro-ao-python
13. Liste os arquivos para confirmar: ls
14. Sempre verifique o diretório atual antes de clonar um repositório para evitar erros.
15. Use `pwd` para imprimir o caminho do diretório atual, se necessário.

## Forkar Repositórios

Forkar um repositório cria uma cópia do repositório original na sua conta GitHub. Isso é útil para contribuir com projetos de outras pessoas. Fork é bem semelhante ao clone porém ele so ocorre na interface gráfica do github. O repositório não sera baixado para o seu computador, mas você terá uma cópia dele na sua conta do github. Depois de “forkar” um repositório de outra pessoa, você pode fazer pull-requests para contribuir com o conteúdo no repositório principal.

**Passos:**

Acesse o repositório original no GitHub que você deseja copiar.

Clique no botão "Fork" no canto superior direito da página do repositório. Isso criará uma cópia do repositório em sua própria conta GitHub.

Clone o repositório forked para o seu computador usando o comando git clone:

git clone https://github.com/seu-usuario/nome-do-repositorio.git

Substitua seu-usuario pelo seu nome de usuário no GitHub e nome-do-repositorio pelo nome do repositório que você fez o fork.

Agora você pode fazer alterações no projeto localmente e, quando estiver pronto, pode enviar suas mudanças de volta para o GitHub e abrir um pull request para o repositório original.

## Pull Requests

Pull requests permitem que você envie suas alterações para o repositório de outra pessoa. O dono do repositório pode revisar e aceitar suas mudanças.

**Passos para Criar um Pull Request:**

1. Faça um fork do repositório do colega.
2. Clone o repositório forkado para sua máquina:

git clone https://github.com/seu-usuario/nome-do-repositorio.git

1. Crie uma nova branch para suas alterações:

git checkout -b nome-da-branch

1. Faça as alterações necessárias e realize commits:

git add .

git commit -m "Descrição das alterações"

1. Envie as alterações para o seu fork:

git push origin nome-da-branch

1. No GitHub, abra um pull request do seu fork para o repositório original.

**Passos para Revisar um Pull Request:**

1. Acesse o repositório no GitHub.
2. Vá para a aba "Pull requests".
3. Selecione o pull request que deseja revisar.
4. Adicione comentários com dúvidas ou sugestões.
5. Aprove ou solicite mudanças no pull request.

# Aula 4: Comandos Avançados de Git

**Objetivos:**

* Utilizar comandos avançados de Git para gerenciar repositórios.
* Resolver conflitos de merge.

**Conteúdo:**

## Comandos Avançados

* git pull: Atualiza o repositório local com as mudanças do repositório remoto.
* git push: Envia as alterações locais para o repositório remoto.
* git fetch: Baixa as mudanças do repositório remoto sem aplicá-las.
* git rebase: Reaplica commits de uma branch em outra.

**Exemplo Prático:**

# Atualizar o repositório local

git pull origin main

# Enviar alterações locais para o repositório remoto

git push origin main

# Baixar mudanças do repositório remoto sem aplicá-las

git fetch origin

# Reaplicar commits de uma branch em outra

git rebase main

!Exemplo de Comandos Avançados

## Resolvendo Conflitos

Conflitos de merge ocorrem quando duas alterações incompatíveis são feitas no mesmo arquivo. É necessário resolver esses conflitos manualmente.

**Passos para Resolver Conflitos:**

1. **Identificar Conflitos:**
   * Quando um conflito ocorre, Git interrompe o processo de merge e marca os arquivos conflitantes.
   * Use o comando git status para ver quais arquivos estão em conflito.
2. **Editar Arquivos Conflitantes:**
   * Abra os arquivos conflitantes no seu editor de código (VSCode).
   * Git marca os conflitos com linhas como <<<<<<<, ======= e >>>>>>>.
   * Edite o arquivo para resolver o conflito, escolhendo quais mudanças manter ou combinando as alterações.
3. **Marcar Conflitos como Resolvidos:**
   * Após resolver os conflitos, adicione os arquivos ao índice:
4. git add nome-do-arquivo
5. **Finalizar o Merge:**
   * Realize um commit para finalizar o merge: bash git commit -m "Resolvendo conflitos de merge"

**Exemplo Prático:**

1. Simule um conflito criando duas branches e fazendo alterações conflitantes no mesmo arquivo.
2. Tente fazer o merge das branches e observe o conflito.
3. Resolva o conflito seguindo os passos acima.
4. Realize o commit para finalizar o merge.

**Exercício Prático:**

* + Pratique os comandos git pull, git push, git fetch e git rebase.
  + Crie duas branches e faça alterações conflitantes no mesmo arquivo.
  + Tente fazer o merge das branches e resolva os conflitos.
  + Trabalhe em pares para resolver conflitos em um projeto compartilhado.

# Aula 5: Colaboração e Fluxo de Trabalho

**Objetivos:**

* Compreender o fluxo de trabalho colaborativo com Git e GitHub.
* Utilizar branches para desenvolvimento colaborativo.

**Conteúdo:**

## Fluxo de Trabalho Colaborativo

Branches de desenvolvimento permitem que várias pessoas trabalhem simultaneamente em diferentes partes do projeto. Isso é essencial para equipes que trabalham em funcionalidades diferentes ou em correções de bugs.

## **Branches Comuns**

* develop: Ambiente de desenvolvimento onde novas funcionalidades são integradas.
* homolog: Ambiente de homologação onde as funcionalidades são testadas antes de irem para produção.
* master ou main: Ambiente de produção onde o código estável é mantido.

## **Exemplo de Fluxo de Trabalho**

1. **Criação de Branches:**
   * Cada nova funcionalidade ou correção de bug deve ser desenvolvida em uma branch separada.
   * Nomeie as branches de forma descritiva, como feature/nova-funcionalidade ou bugfix/corrigir-bug.
2. **Desenvolvimento:**
   * Desenvolva a funcionalidade ou correção na branch criada.
   * Realize commits frequentes para registrar o progresso.
3. **Integração:**
   * Quando a funcionalidade estiver pronta, faça um pull request para a branch develop.
   * Outros membros da equipe devem revisar o pull request e fornecer feedback.
4. **Teste e Homologação:**
   * Após a aprovação do pull request, a branch develop é mesclada na branch homolog para testes.
   * Realize testes para garantir que a funcionalidade está funcionando corretamente.
5. **Produção:**
   * Após a homologação, a branch homolog é mesclada na branch master ou main.
   * O código estável é então implantado em produção.

## **Comandos**

* git branch nome-da-branch: Cria uma nova branch.
* git checkout nome-da-branch: Alterna para a branch especificada.
* git push --set-upstream origin nome-da-branch: Envia a branch para o repositório remoto.

**Exemplo Prático:**

# Criar uma nova branch

git branch feature/nova-funcionalidade

# Alternar para a nova branch

git checkout feature/nova-funcionalidade

# Enviar a branch para o repositório remoto

git push --set-upstream origin feature/nova-funcionalidade

!Exemplo de Fluxo de Trabalho

**Exercício Prático:**

* + Crie uma nova branch para uma funcionalidade ou correção de bug.
  + Realize commits frequentes durante o desenvolvimento.
  + Trabalhe em uma funcionalidade em uma branch separada.
  + Faça um pull request para a branch develop e peça revisão de um colega.
  + Divida o projeto em funcionalidades e atribua a cada membro da equipe uma branch.
  + Integre as funcionalidades na branch develop e depois na branch homolog para testes.

# Aula 6: Projeto Final

**Objetivos:**

* Aplicar todo o conhecimento adquirido em um projeto prático.
* Subir um projeto completo para o GitHub.

**Conteúdo:**

## Criando um Projeto

Estruture o projeto com pastas e arquivos necessários.

**Passos:**

1. Crie um diretório para o projeto:

mkdir nome-do-projeto

cd nome-do-projeto

1. Adicione arquivos e pastas conforme necessário:

touch index.html

mkdir css js

touch css/style.css js/script.js

## Versionamento Completo

Realize commits, pushes e merges para manter o histórico de alterações.

**Comandos:**

* Adicionar todas as alterações ao índice:

git add .

* Realizar um commit com uma mensagem descritiva:

git commit -m "mensagem do commit"

* Enviar as alterações locais para o repositório remoto:

git push

* Realizar o merge da branch especificada com a branch atual:

git merge nome-da-branch

## Publicação no GitHub

Configure o repositório remoto e publique o projeto no GitHub.

**Comandos:**

* Adicionar o repositório remoto:

git remote add origin https://github.com/usuario/nome-do-repositorio.git

* Enviar o projeto para o repositório remoto:

git push -u origin master

**Exemplo Prático:**

# Criar um diretório para o projeto

mkdir meu-projeto

cd meu-projeto

# Inicializar um repositório Git

git init

# Adicionar arquivos e pastas

touch index.html

mkdir css js

touch css/style.css js/script.js

# Adicionar todas as alterações ao índice

git add .

# Realizar um commit com uma mensagem descritiva

git commit -m "Iniciando o projeto com estrutura básica"

# Adicionar o repositório remoto

git remote add origin https://github.com/usuario/meu-projeto.git

# Enviar o projeto para o repositório remoto

git push -u origin master

!Exemplo de Publicação no GitHub

**Exercício Prático:**

* + Estruture o projeto com pastas e arquivos necessários.
  + Realize commits frequentes para registrar o progresso.
  + Configure o repositório remoto e publique o projeto no GitHub.
  + Colabore com a squad para revisar e melhorar o projeto.
  + Utilize pull requests para integrar as contribuições dos colegas.

# Exercício de Colaboração em Squad

**Objetivo:**

* Simular um ambiente de trabalho colaborativo onde cada integrante da squad resolve um exercício e compartilha com as demais. A líder da squad consolidará tudo e postará no portal para a professora corrigir.

## **Como fazer**

1. **Divisão dos Exercícios:**
   * A squad tem 7 exercícios para resolver.
   * Cada integrante ficará responsável por um exercício.
2. **Criação de Repositórios Individuais:**
   * Cada integrante deve criar um repositório no GitHub para seu exercício.
   * Compartilhe o link do repositório com a squad.
3. **Desenvolvimento dos Exercícios:**
   * Cada integrante deve desenvolver seu exercício no repositório criado.
   * Realize commits frequentes para registrar o progresso.
4. **Revisão por Pares:**
   * Cada integrante deve revisar o exercício de outro membro da squad.
   * Utilize pull requests para sugerir correções e melhorias.
   * Adicione comentários com dúvidas ou sugestões.

## **Passos para Criar um Pull Request**

1. Faça um fork do repositório do colega.
2. Clone o repositório forkado para sua máquina:

git clone https://github.com/seu-usuario/nome-do-repositorio.git

1. Crie uma nova branch para suas alterações:

git checkout -b nome-da-branch

1. Faça as alterações necessárias e realize commits:

git add .

git commit -m "Descrição das alterações"

1. Envie as alterações para o seu fork:

git push origin nome-da-branch

1. No GitHub, abra um pull request do seu fork para o repositório original.

## **Passos para Revisar um Pull Request**

1. Acesse o repositório no GitHub.
2. Vá para a aba "Pull requests".
3. Selecione o pull request que deseja revisar.
4. Adicione comentários com dúvidas ou sugestões.
5. Aprove ou solicite mudanças no pull request.
6. **Consolidação dos Exercícios:**
   * A líder da squad deve clonar os repositórios individuais para sua máquina.
   * Consolidar todos os exercícios em um único repositório.
   * Realizar commits e pushes para manter o histórico de alterações.

## **Passos para Consolidar os Exercícios**

1. Clone os repositórios individuais:

git clone https://github.com/usuario1/exercicio1.git

git clone https://github.com/usuario2/exercicio2.git

...

1. Copie os arquivos para um único repositório:

mkdir consolidado

cp -r exercicio1/\* consolidado/

cp -r exercicio2/\* consolidado/

...

1. Inicialize um repositório Git no diretório consolidado:

cd consolidado

git init

1. Adicione todos os arquivos ao índice e realize um commit:

git add .

git commit -m "Consolidando exercícios da squad"

1. Adicione o repositório remoto e envie as alterações:

git remote add origin https://github.com/lider-da-squad/consolidado.git

git push -u origin master

1. **Postagem no Portal:**
   * A líder da squad deve postar o link do repositório consolidado no portal para a professora corrigir.

# Dicas e Boas Práticas

* **Mensagens de Commit:** Escreva mensagens de commit claras e significativas. A mensagem deve descrever o que foi alterado e por quê.
* **Commits Frequentes:** Realize commits frequentes para registrar o progresso. Commits pequenos e frequentes são mais fáceis de revisar e integrar.
* **Organização de Branches:** Utilize uma convenção de nomes para branches, como feature/nome-da-feature ou bugfix/nome-do-bug. Mantenha a branch main ou master sempre estável.
* **Revisão de Código:** Forneça feedback construtivo e detalhado durante as revisões de código. Utilize pull requests para integrar as contribuições dos colegas.
* **Documentação:** Documente o código e as decisões de design. Utilize o arquivo README.md para fornecer informações sobre o projeto.

# Recursos Adicionais

Para aprofundar seu conhecimento em Git e GitHub, aqui estão alguns recursos adicionais que podem ser úteis:

1. **Tutoriais de Git:**

* [[Git e Github] Guia rápido e Comandos básicos para iniciantes - DEV Community](https://dev.to/womakerscode/git-e-github-guia-rapido-e-comandos-basicos-para-iniciantes-4ile)
* [[Tutorial Git] git branch: O que são branches (ramos) no Git? - DEV Community](https://dev.to/womakerscode/tutorial-git-o-que-sao-branches-ramos-no-git-57pn)
* [Conceitos básicos do Git e do GitHub para a documentação do Microsoft Learn - Contributor guide | Microsoft Learn](https://learn.microsoft.com/pt-br/contribute/content/git-github-fundamentals)
* [**https://dev.to/womakerscode/git-e-github-guia-rapido-e-comandos-basicos-para-iniciantes-4ile**](https://dev.to/womakerscode/git-e-github-guia-rapido-e-comandos-basicos-para-iniciantes-4ile)
* [**https://dev.to/womakerscode/tutorial-git-adicionando-um-arquivo-modificado-no-git-116c**](https://dev.to/womakerscode/tutorial-git-adicionando-um-arquivo-modificado-no-git-116c)
* [**https://dev.to/womakerscode/tutorial-git-enviando-arquivos-para-o-repositorio-git-1k91**](https://dev.to/womakerscode/tutorial-git-enviando-arquivos-para-o-repositorio-git-1k91)
* [**https://dev.to/womakerscode/tutorial-git-adicionando-arquivos-no-git-3a0o**](https://dev.to/womakerscode/tutorial-git-adicionando-arquivos-no-git-3a0o)
* [**https://dev.to/womakerscode/tutorial-git-enviando-um-projeto-para-o-repositorio-remoto-no-github-2p36**](https://dev.to/womakerscode/tutorial-git-enviando-um-projeto-para-o-repositorio-remoto-no-github-2p36)
* [**https://dev.to/womakerscode/tutorial-git-puxando-commits-remotos-44an**](https://dev.to/womakerscode/tutorial-git-puxando-commits-remotos-44an)
* [**https://dev.to/womakerscode/tutorial-git-ignorando-arquivos-no-git-32m9**](https://dev.to/womakerscode/tutorial-git-ignorando-arquivos-no-git-32m9)
* [[Tutorial Git] git fetch: Verificando commits remotos - DEV Community](https://dev.to/womakerscode/tutorial-git-verificando-commits-remotos-56f3)
* [[Tutorial Git] Copiando um repositório existente (git clone) - DEV Community](https://dev.to/womakerscode/tutorial-git-copiando-um-repositorio-existente-git-clone-1bfe)
* [[Tutorial Git] fork: Como colaborar com projetos de código aberto - DEV Community](https://dev.to/womakerscode/tutorial-git-fork-como-colaborar-com-projetos-de-codigo-aberto-1lkm)

1. **Documentação Oficial:**
   * [Recursos de aprendizagem Git e GitHub - GitHub Docs](https://docs.github.com/pt/get-started/start-your-journey/git-and-github-learning-resources)
   * [Git - git Documentation](https://git-scm.com/docs/git/pt_BR)
   * [GitHub Docs](https://docs.github.com/pt)
2. **Cursos Online:**
   * [Codificar com o GitHub Codespaces - Training | Microsoft Learn](https://learn.microsoft.com/pt-br/training/modules/code-with-github-codespaces/?WT.mc_id=academic-116050-cyzanon)
   * [Tutorial de GitHub grátis - Git e Github para iniciantes | Udemy](https://www.udemy.com/course/git-e-github-para-iniciantes/?srsltid=AfmBOorXK1t8PNWhDzzyPBJUNOi1kQzT0orauJvqAIx7N0xA8HiwO7ye)
   * [GitHub Apps - GitHub Learning Lab](https://github.com/apps/github-learning-lab)
   * [Introdução ao Git e GitHub | FGV Educação Executiva](https://educacao-executiva.fgv.br/cursos/online/curta-media-duracao-online/introducao-ao-git-e-github)
   * [Treinamento para o GitHub | Microsoft Learn](https://learn.microsoft.com/pt-br/training/github/)
3. **Comunidades e Fóruns:**
   * Stack Overflow: [Stack Overflow em Português](https://pt.stackoverflow.com/)
   * GitHub Community Forum [community/community: Public feedback discussions for: GitHub Mobile, GitHub Discussions, GitHub Codespaces, GitHub Sponsors, GitHub Issues and more!](https://github.com/community/community)

# Perguntas Frequentes (FAQ)

1. **O que é Git?** Git é um sistema de controle de versão distribuído que permite rastrear mudanças no código-fonte durante o desenvolvimento de software. Ele ajuda a gerenciar o histórico de alterações e facilita a colaboração entre desenvolvedores.
2. **O que é GitHub?** GitHub é uma plataforma de hospedagem de código-fonte que utiliza Git para controle de versão. Ele permite que desenvolvedores armazenem, compartilhem e colaborem em projetos de software.
3. **Qual a diferença entre Git e GitHub?** Git é uma ferramenta de controle de versão, enquanto GitHub é uma plataforma de hospedagem de código que utiliza Git. Git pode ser usado localmente, enquanto GitHub permite colaboração online.
4. **Como faço para clonar um repositório?** Use o comando git clone https://github.com/usuario/nome-do-repositorio.git para clonar um repositório remoto para o seu computador.
5. Como eu envio as alterações que eu comitei localmente para mim origem remota? Com o comando git push podemos enviar nossas alterações locais para o nosso repositório remoto. Dessa forma o seu repositório remoto estará atualizado com a sua versão local, e outras pessoas que forem acessar ele poderá ter uma versão mais atualizada.
6. Como enviar uma contribuição para o repositório de outra pessoa? Fazendo uma pull request. após forkar o repositório de outra pessoa, podemos resolver issues dele e fazer pull requests enviando soluções para erros e adicionando novos conteúdos. a pessoa dona do repositório vai avaliar sua pull request e se estiver tudo ok, sera mergeada no repositório principal.
7. **Qual o comando usado para adicionar todas as mudanças feitas nos arquivos do seu diretório? Git add**
8. **O que é um fork?** Um fork é uma cópia de um repositório que é criada na sua própria conta GitHub. Ele permite que você faça mudanças no projeto original sem afetar o repositório original.
9. **Como faço um pull request?** Faça um fork do repositório, clone-o para sua máquina, crie uma nova branch, faça as alterações, envie as alterações para o seu fork e abra um pull request no GitHub.
10. **O que é um merge?** Merge é o processo de unir duas branches. Ele combina as alterações de uma branch com outra.
11. **Como resolver conflitos de merge?** Identifique os conflitos, edite os arquivos conflitantes para resolver os conflitos, adicione os arquivos ao índice e finalize o merge com um commit.
12. **Como faço para criar uma nova branch?** Use o comando git branch nome-da-branch para criar uma nova branch e git checkout nome-da-branch para alternar para a nova branch.
13. **Como faço para enviar minhas alterações para o GitHub?** Use os comandos git add ., git commit -m "mensagem do commit" e git push origin nome-da-branch para enviar suas alterações para o repositório remoto no GitHub.
14. **O que é um commit?** Um commit é um registro de alterações no repositório. Ele inclui uma mensagem descritiva que explica as mudanças feitas.
15. **Como faço para atualizar meu repositório local com as mudanças do repositório remoto?** Use o comando git pull para atualizar seu repositório local com as mudanças do repositório remoto.
16. **O que é um rebase?** Rebase é o processo de reaplicar commits de uma branch em outra. Ele é usado para manter um histórico de commits mais limpo e linear.
17. **Como faço para ver o histórico de commits?** Use o comando git log para ver o histórico de commits do repositório.
18. **Como faço para desfazer um commit?** Use o comando git revert <commit-hash> para desfazer um commit específico. Para desfazer o commit mais recente, use git reset --soft HEAD~1.
19. **O que é um repositório remoto?** Um repositório remoto é uma versão do seu repositório hospedada em um servidor, como o GitHub. Ele permite que você colabore com outros desenvolvedores e mantenha uma cópia segura do seu código.
20. **Como faço para adicionar um repositório remoto?** Use o comando git remote add origin https://github.com/usuario/nome-do-repositorio.git para adicionar um repositório remoto.
21. **Como faço para listar todas as branches do meu repositório?** Use o comando git branch para listar todas as branches locais e git branch -r para listar todas as branches remotas.
22. **O que é um pull?** Pull é o processo de atualizar seu repositório local com as mudanças do repositório remoto. Ele combina os comandos git fetch e git merge.
23. **Como faço para configurar meu nome de usuário e email no Git?** Use os comandos git config --global user.name "Seu Nome" e git config --global user.email "seu-email@example.com" para configurar seu nome de usuário e email no Git.
24. **Como faço para ignorar arquivos no Git?** Crie um arquivo .gitignore no diretório raiz do seu repositório e adicione os nomes dos arquivos ou diretórios que deseja ignorar.
25. **O que é um stash no Git?** Stash é uma funcionalidade do Git que permite salvar temporariamente mudanças não commitadas para que você possa trabalhar em outra coisa e depois aplicar essas mudanças novamente.
26. **Como faço para aplicar um stash?** Use o comando git stash apply para aplicar as mudanças salvas no stash.
27. **Como faço para ver as mudanças entre commits?** Use o comando git diff para ver as mudanças entre commits. Você pode especificar os commits que deseja comparar.
28. **Como faço para renomear uma branch?** Use o comando git branch -m nome-antigo nome-novo para renomear uma branch.
29. **Como faço para excluir uma branch?** Use o comando git branch -d nome-da-branch para excluir uma branch localmente e git push origin --delete nome-da-branch para excluir uma branch remotamente.
30. **O que é um submódulo no Git?** Um submódulo é um repositório Git dentro de outro repositório Git. Ele permite que você inclua e gerencie dependências de outros projetos Git.
31. **Como faço para adicionar um submódulo?** Use o comando git submodule add https://github.com/usuario/nome-do-repositorio.git caminho/do/submodulo para adicionar um submódulo.
32. **Como faço para atualizar um submódulo?** Use o comando git submodule update --remote para atualizar um submódulo para a última versão do branch configurado.
33. **Como faço para resolver conflitos de rebase?** Identifique os conflitos, edite os arquivos conflitantes para resolver os conflitos, adicione os arquivos ao índice e continue o rebase com git rebase --continue.

# Exercícios Adicionais

**Exercício 1: Criar um Repositório e Realizar Commits**

1. Crie um repositório local.
2. Adicione arquivos e realize commits frequentes.
3. Publique o repositório no GitHub.

**Exercício 2: Trabalhar com Branches**

1. Crie uma nova branch para uma funcionalidade.
2. Realize commits na nova branch.
3. Faça o merge da branch com a branch main.

**Exercício 3: Colaboração com Pull Requests**

1. Faça um fork de um repositório.
2. Faça alterações no fork e crie um pull request.
3. Revise e aceite pull requests de colegas.

**Exercício 4: Resolver Conflitos de Merge**

1. Crie duas branches e faça alterações conflitantes no mesmo arquivo.
2. Tente fazer o merge das branches e resolva os conflitos.
3. Realize o commit para finalizar o merge.

**Exercício 5: Praticar a adição, commit e push para o GitHub**

1. Crie uma pasta com o nome do projeto (a sua escolha) e um arquivochamado index.html2.
2. Adicione o seguinte conteúdo ao arquivo:<!DOCTYPE html><html lang="pt-br"><head><title>Título da página</title><meta charset="utf-8"></head><body><h1>Aqui vai um título</h1></body></html>
3. Subindo seu projeto para o GitHub1.
4. Crie um repositório em seu GitHub2.
5. Abra o seu terminal de comando e adicione a origem remota e conecte seu projeto local com o GitHub
6. Verifique as alterações do seu projeto e adicione ao fluxo de versionamento
7. Faça um commit com uma mensagem útil e na sequência, um push para o repositório remoto (GitHub)
8. Lidando com alterações
9. Abra a pasta do seu projeto, crie uma pasta chamada imagens, procure no Google por uma imagem de gato e salve dentro dessa nova pasta.
10. Feito isso, abra seu arquivo index.html e adicione as seguintes modificações e salve o arquivo:<!DOCTYPE html><html lang="pt-br"><head>
11. <title>Fanpage de Gatinhos</title><meta charset="utf-8"></head><body><h1>Perfil #catsoninstagram</h1><img src="images/nome\_da\_sua\_imagem.jpg" /></body></html>

# Resolução de Problemas

**Problema 1: Conflitos de Merge**

* **Solução:** Identifique os conflitos, edite os arquivos conflitantes para resolver os conflitos, adicione os arquivos ao índice e finalize o merge com um commit.

**Problema 2: Erro ao Clonar Repositório**

* **Solução:** Verifique se o URL do repositório está correto. Certifique-se de que você tem permissão para acessar o repositório.

**Problema 3: Não Consigo Fazer Push para o Repositório Remoto**

* **Solução:** Verifique se você configurou corretamente o repositório remoto. Certifique-se de que você tem permissão para fazer push.

**Problema 4: Pull Request Não Aceito**

* **Solução:** Revise o feedback fornecido no pull request. Faça as alterações necessárias e envie novamente.

**Problema 5: Branch Não Encontrada**

* **Solução:** Verifique se o nome da branch está correto. Use o comando git branch para listar todas as branches disponíveis.

# Conclusão

Nesta apostila, você aprendeu diversos conceitos e comandos essenciais do Git e GitHub, incluindo:

* Commit: Como salvar as mudanças locais no repositório com uma mensagem.
* Git push: Enviar commits locais para um repositório remoto para que outros possam vê-los.
* Git checkout: Mudar de branch (ramificação).
* Git pull: Atualizar o repositório local com as mudanças do repositório remoto para trabalhar com a versão mais recente do código.
* Criação de branches: Criar branches para não afetar o código principal.
* Revisão de código com pull requests: Fazer revisão de código no GitHub e integrar mudanças feitas por outros.
* Git clone: Criar uma cópia local de um repositório remoto.
* Git status: Ver o estado atual do repositório, incluindo mudanças não comitadas e arquivos não rastreados.
* Git merge: Combinar mudanças de diferentes branches.
* Fork: Criar uma cópia de um repositório em sua própria conta no GitHub para fazer mudanças sem afetar o repositório original.

**Pontos Importantes sobre Forks:**

* Criação: Você pode criar um fork de qualquer repositório público clicando no botão "Fork" no GitHub.
* Independência: O fork é independente do repositório original, permitindo que você faça mudanças livremente.
* Contribuição: Para contribuir de volta ao projeto original, faça mudanças no seu fork e crie um pull request.
* Colaboração: Forks são úteis para colaborar em projetos open source, permitindo trabalhar em melhorias ou correções e propor essas mudanças aos mantenedores do projeto original.

**Sugestões de Estudo Futuro**

Embora esta apostila tenha coberto muitos tópicos importantes, há alguns conceitos que não foram abordados, mas que são igualmente importantes para o seu desenvolvimento como usuário de Git e GitHub:

* Resolução de Conflitos: Aprender a resolver conflitos que podem surgir durante o merge.
* Git rebase: Reaplicar commits de uma branch em outra.
* GitHub Actions: Automação de fluxos de trabalho, como testes e deploys.
* GitHub Pages: Hospedar sites diretamente dos repositórios.

Recomendo fortemente que você estude esses tópicos para aprofundar ainda mais seu conhecimento e habilidades com Git e GitHub. Boa sorte e continue aprendendo! 🚀