📘 Aula 1 – Parte 1: Fundamentos do Git e Primeiros Passos

**🧩 Objetivos da aula (Parte 1)**

Nesta primeira parte da aula, o objetivo é apresentar os conceitos fundamentais do Git, entender o que é um sistema de controle de versão, e dar os primeiros passos práticos com Git no ambiente local.

**🔹 1. O que é Git?**

O **Git** é um sistema de controle de versão distribuído, criado por Linus Torvalds em 2005 para gerenciar o desenvolvimento do kernel do Linux. Com o Git, é possível rastrear mudanças em arquivos ao longo do tempo, reverter modificações, trabalhar em paralelo com outras pessoas e manter o histórico completo de um projeto.

**Por que usar o Git?**

* Rastrear o histórico de alterações
* Trabalhar em equipe com controle
* Reverter arquivos ou versões do projeto
* Criar múltiplas versões do código em paralelo (branches)
* Facilitar a colaboração com plataformas como o GitHub

**🔹 2. Git vs GitHub**

É comum confundir os dois, mas são coisas diferentes:

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA poderão estar incorretos.

**🔹 3. Repositórios Git**

Um **repositório Git** é um diretório que armazena os arquivos do projeto e o histórico de alterações desses arquivos. Há dois tipos:

* **Repositório local**: armazenado na sua máquina
* **Repositório remoto**: hospedado em um servidor como o GitHub, GitLab ou Bitbucket

**🔹 4. Instalação e Configuração Inicial**

**Verificando se o Git está instalado:**

git --version

**Configurando nome e e-mail:**

Essas informações aparecem nos commits.

git config --global user.name "Seu Nome"

git config --global user.email "seu@email.com"

Conferindo as configurações:

git config --list

**🔹 5. Iniciando um projeto com Git**

**Criar uma pasta para o projeto:**

mkdir meu-projeto

cd meu-projeto

Inicializar o Git na pasta:

git init

Isso cria uma pasta oculta .git/, que guarda todo o histórico do projeto.

**🔹 6. Entendendo os três estados principais**

Ao trabalhar com Git, os arquivos podem estar em três estados:

1. **Modificado (Modified)** – Foi alterado, mas ainda não preparado para commit.
2. **Preparado (Staged)** – Marcado para ser incluído no próximo commit.
3. **Confirmado (Committed)** – Gravado no histórico do repositório.

**🔹 7. Criando o primeiro commit**

**Criar um arquivo:**

echo "# Meu Projeto" > README.md

Verificar o status:

git status

Adicionar o arquivo ao staging:

git add README.md

Ou para adicionar todos os arquivos:

git add .

Criar o commit:

git commit -m "Primeiro commit"

🔹 8. Visualizando o histórico de commits

git log

Formas resumidas:

git log --oneline

git log --graph --oneline --all

**🔹 9. Arquivo .gitignore**

Serve para dizer ao Git quais arquivos ou pastas **não** devem ser versionados (por exemplo, arquivos temporários, binários, senhas).

**Exemplo de .gitignore:**

\*.log

node\_modules/

.env

**🔹 Atividade prática 1: Criando e versionando um projeto local**

1. Criar uma pasta de projeto
2. Inicializar com git init
3. Criar arquivos como README.md, index.html, script.js
4. Usar git add, git commit
5. Criar e testar o .gitignore

📘 Aula 1 – Parte 2: GitHub e Sincronização com Repositórios Remotos

**🧩 Objetivos da aula (Parte 2)**

* Entender a diferença entre repositórios locais e remotos
* Aprender a conectar o Git local ao GitHub
* Criar e clonar repositórios online
* Realizar os comandos push, pull e clone
* Testar cenários reais de trabalho com o GitHub

**🔹 1. Criando um repositório no GitHub**

1. Acesse <https://github.com>
2. Clique em **New repository**
3. Escolha:
   * Nome do repositório
   * Descrição (opcional)
   * Público ou privado
   * Desmarque a opção de criar README (faremos isso localmente)
4. Clique em **Create repository**

**🔹 2. Subindo um projeto local para o GitHub**

**Passos:**

1. Inicialize um repositório local (caso ainda não tenha feito):

git init

1. Adicione e comite os arquivos:

git add .

git commit -m "Primeiro commit"

1. Copie a URL do repositório remoto no GitHub (HTTPS ou SSH)
2. Criea branch main e adicione o repositório remoto:

git branch -M main

git remote add origin https://github.com/seuusuario/nome-do-repo.git

1. Envie os arquivos locais para o GitHub:

git push -u origin main

**🔹 3. Clonando um repositório do GitHub para a sua máquina**

**Quando você quer começar com um projeto que já está no GitHub:**

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

Isso já cria a pasta, baixa o conteúdo e já vem com o repositório conectado ao remoto.

4. Salvando alterações no GitHub (push)

Depois de modificar um arquivo:

git add .

git commit -m "mensagem do commit"

git push

**🔹 5. Atualizando o projeto local com mudanças do GitHub (pull)**

**Se alguém fez alterações no GitHub e você quer trazer para sua máquina:**

git pull

Esse comando sincroniza o que está no GitHub com a sua máquina. Pode haver conflitos se você tiver modificado os mesmos arquivos localmente.

**🔹 6. Cenários práticos com repositórios remotos**

**▶️ Cenário 1: Criar projeto no GitHub e clonar**

* Criar repositório no GitHub
* Clonar com git clone
* Adicionar arquivos, comitar e fazer push

**▶️ Cenário 2: Criar projeto local e subir para GitHub**

* Criar projeto na máquina
* Inicializar com git init
* Criar repositório no GitHub
* Conectar com git remote add origin
* Enviar com git push

**▶️ Cenário 3: Clonar repositório de outra pessoa**

* Usar git clone https://github.com/fulano/projeto.git
* Você não poderá dar push, pois não é dono do repositório

**🔹 7. Remover a origem remota e trocar por outra**

Quando você clona o repositório de alguém mas quer criar o seu próprio repositório e usá-lo:

git remote remove origin

git remote add origin https://github.com/seuusuario/novorepo.git

git push -u origin main

**🔹 8. Atividade prática 2: GitHub e repositórios remotos**

**Atividade A:**

1. Criar repositório no GitHub
2. Clonar na máquina local
3. Criar index.html, comitar e dar push

**Atividade B:**

1. Criar repositório local na máquina
2. Criar repositório no GitHub
3. Conectar com git remote
4. Enviar projeto com git push

**Atividade C:**

1. Clonar repositório de um colega
2. Criar uma nova origem no seu próprio GitHub
3. Subir o projeto modificado para o seu repositório

🔹 9. Dica bônus: Comando git remote -v

Para verificar quais repositórios remotos estão conectados:

📘 Aula 2 – Parte 1: Trabalhando com Branches (Ramificações)

**🧩 Objetivos da aula (Parte 1)**

* Compreender o conceito de ramificações (branches)
* Criar, renomear, mudar e excluir branches
* Trabalhar com múltiplas linhas de desenvolvimento
* Fazer merge de branches simples (sem conflito)
* Entender o papel dos branches no trabalho em equipe

**🔹 1. O que são branches?**

Branches (ou ramificações) são **linhas paralelas de desenvolvimento** dentro de um projeto Git. A branch principal geralmente se chama main, mas podemos criar outras para desenvolver funcionalidades, corrigir bugs ou testar novas ideias sem interferir no código principal.

**Analogia**: Imagine o código como uma estrada. Branches são desvios que você cria para testar caminhos alternativos sem bloquear o tráfego principal.

**🔹 2. Criando e listando branches**

**Criar uma nova branch:**

git branch nome-da-branch

Listar todas as branches:

git branch

O asterisco \* indica a branch atual em que você está

**🔹 3. Mudando de branch**

**Alternar entre branches:**

git checkout nome-da-branch

**🔹 4. Criando e mudando para uma nova branch (atalho)**

**Você pode fazer os dois passos anteriores com um único comando:**

git checkout -b nome-da-branch

**🔹 5. Renomeando uma branch**

**Se você estiver na branch:**

git branch -m novo-nome

**Para renomear outra branch:**

git branch -m nome-antigo novo-nome

**🔹 6. Excluindo uma branch**

git branch -d nome-da-branch

**O -d só funciona se a branch já foi mesclada (merge). Use -D para forçar.**

**🔹 7. Subindo uma branch para o GitHub**

**Criou uma branch local e quer subir:**

git push -u origin nome-da-branch

**O -u faz com que o Git "lembre" que a branch local está conectada à remota.**

**🔹 8. Cenário prático: desenvolvimento com branches**

**Situação:**

**Você está trabalhando no site de um cliente. A branch principal (main) está estável. Você quer desenvolver uma nova funcionalidade chamada “formulário de contato” sem afetar o site online.**

**Passo a passo:**

git checkout -b formulario-contato

# modifica os arquivos do projeto

git add .

git commit -m "Adiciona estrutura do formulário"

git push -u origin formulario-contato

**Depois você pode abrir um Pull Request (na parte 2 da aula).**

**🔹 9. Fazendo merge (mesclar) de branches**

**Voltar para a main e aplicar as mudanças da outra branch:**

git checkout main

git merge formulario-contato

**🔹 10. Visualizando o histórico de branches**

git log --oneline --graph --all

**Mostra o histórico com ramificações e merges de forma visual.**

**🔹 11. Atividades práticas com branches**

**Atividade A: Branch básica**

1. **Crie um projeto simples com um index.html**
2. **Crie uma branch chamada layout-inicial**
3. **Adicione um título à página e comite**
4. **Volte para main, veja que a mudança não está lá**
5. **Faça o merge da layout-inicial para main**

**Atividade B: Desenvolvimento paralelo**

1. **Crie uma branch chamada menu**
2. **Crie outra chamada rodape**
3. **Em cada uma, modifique index.html com conteúdo diferente**
4. **Faça merge de ambas para main, verificando os resultados**

**🧩 Objetivos da aula (Parte 2)**

* **Entender o conceito e uso de fork**
* **Enviar alterações via pull request**
* **Simular o fluxo de contribuição em repositórios públicos**
* **Resolver conflitos de merge**
* **Compreender o papel das branches nesse fluxo**

**1. O que é fork?**

**Fork é uma cópia de um repositório no seu próprio GitHub. Ele permite que você faça alterações sem afetar o projeto original.**

**Usado especialmente para contribuir com repositórios públicos nos quais você não tem permissão de escrita.**

**Como fazer um fork:**

1. **Vá até o repositório original no GitHub.**
2. **Clique em Fork (no canto superior direito).**
3. **Escolha o seu perfil.**
4. **O GitHub criará uma cópia desse repositório na sua conta.**

**🔹 2. Clonando o fork para sua máquina**

**Depois de fazer o fork, clone o repositório para trabalhar localmente:**

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

cd nome-do-repositorio

**🔹 3. Criar branch para contribuição**

**Sempre crie uma nova branch para sua contribuição:**

git checkout -b minha-contribuicao

**Faça suas alterações, adicione e comite:**

git add .

git commit -m "Minha contribuição"

**Suba a branch para o seu GitHub:**

git push origin minha-contribuicao

**🔹 4. Criando um Pull Request (PR)**

1. **No GitHub, vá até o seu repositório (forkado).**
2. **Clique em Compare & Pull Request.**
3. **Escreva um título e uma descrição claros.**
4. **Clique em Create Pull Request.**

**Isso envia sua proposta para o repositório original. O dono pode revisar e aceitar.**

**🔹 5. Atualizando seu fork (mantendo sincronizado)**

**Adicione o repositório original como upstream:**

git remote add upstream https://github.com/original-autor/repositorio.git

**Para buscar as atualizações do original:**

git fetch upstream

**Para atualizar sua branch main com as mudanças do original:**

git checkout main

git merge upstream/main

**🔹 6. Resolvendo conflitos de merge**

**Conflitos acontecem quando duas branches modificam a mesma parte do mesmo arquivo. Exemplo:**

git merge nome-da-branch

**Se houver conflito:**

* **O Git vai mostrar os arquivos em conflito.**
* **Abra o arquivo, e você verá marcações assim:**

<<<<<<< HEAD

versão da sua branch atual

=======

versão da outra branch

>>>>>>> nome-da-branch

**Resolva o conflito manualmente, remova as marcações <<<<<<<, =======, >>>>>>>, e salve o arquivo.**

**Depois:**

git add arquivo-com-conflito

git commit

**🔀 Quem deve fazer o merge?**

**O merge é o momento em que as alterações de uma branch (geralmente de uma feature ou correção) são integradas à branch principal, como a main ou develop. Mas quem deve fazer esse merge?**

**👤 Em projetos pessoais ou individuais:**

**Se você estiver trabalhando sozinho em um projeto, você pode criar uma branch para organizar melhor seu trabalho (ex: nova-funcionalidade), mas o merge geralmente será feito por você mesmo. Nesse caso, você pode:**

* **Fazer o merge manualmente via terminal (git merge)**
* **Ou usar o GitHub para abrir um Pull Request e depois aceitá-lo sozinho (o que te dá a chance de revisar antes de fundir)**

**Mesmo sendo só uma pessoa, abrir Pull Requests pode ajudar a manter um histórico organizado e revisar melhor o que está sendo alterado.**

**👥 Em projetos em equipe ou em empresas:**

**Em projetos com mais pessoas, o fluxo costuma ser mais controlado. Você cria uma branch, desenvolve nela, e depois abre um Pull Request pedindo que seu código seja avaliado antes de ser integrado à branch principal.**

**Nesse cenário:**

* **O merge normalmente é feito por outra pessoa, como um revisor ou líder técnico, após revisar seu código.**
* **Algumas empresas exigem que ao menos 1 ou 2 pessoas aprovem o PR antes do merge.**
* **O objetivo é garantir a qualidade, evitar bugs e manter o código consistente.**

**Em resumo: você pode fazer o merge sozinho, mas em equipes é ideal (e muitas vezes obrigatório) que outra pessoa revise e aprove antes do merge.**

**🔹 7. Simulação de contribuição (atividade prática)**

**Cenário:**

**Você quer contribuir com um projeto open source:**

1. **Faça fork do repositório de um colega.**
2. **Clone seu fork localmente.**
3. **Crie uma nova branch.**
4. **Adicione um novo componente ou linha no README.**
5. **Comite e suba a branch.**
6. **Crie um Pull Request.**
7. **O colega revisa e aceita (ou pede ajustes).**
8. **Atualize seu fork com as mudanças do repositório original.**

**🔹 8. Remover a origem original (caso clone de outra pessoa)**

**Se você clonou o repositório de outra pessoa e deseja "desvincular" da origem e colocar a sua:**

git remote remove origin

git remote add origin https://github.com/seu-usuario/seu-repo.git

git push -u origin main

**🧭 Aula 3 – Parte 1: Voltando no Tempo com Git – Histórico, Revisões e Recuperações**

**🎯 Objetivos da aula (Parte 1)**

* **Entender o funcionamento do histórico de commits**
* **Navegar entre versões anteriores do projeto**
* **Visualizar, comparar, restaurar e desfazer alterações**
* **Saber escolher o comando certo para cada tipo de necessidade:**
  + **checkout, restore, revert, reset**

**🕰️ 1. Entendendo o histórico de commits**

**O Git mantém um histórico completo de tudo que foi feito no repositório.**

**Visualizando o histórico:**

git log

**Algumas opções úteis:**

git log --oneline

git log --graph --all

git log --stat

**Esses comandos ajudam a identificar commits específicos por sua hash e entender o que foi alterado em cada um.**

**🔍 2. Visualizar uma versão anterior (sem alterar nada)**

**Útil para ver como o projeto estava em um commit passado sem afetar o projeto atual.**

**Passos:**

git checkout <hash-do-commit>

**📌 Você estará em modo de leitura (detached HEAD). Isso significa que você pode navegar pelo projeto, mas não deve fazer alterações permanentes aqui.**

**Para voltar ao estado mais recente:**

git checkout main

**🧪 3. Modificar uma versão antiga e reaproveitar as mudanças**

**Cenário: você volta para um commit antigo, faz modificações, e quer trazer essas mudanças para o projeto atual.**

**Passos:**

1. **Volte para o commit:**

git checkout <hash>

1. **Crie uma nova branch a partir dali:**

git checkout -b ajuste-no-commit-antigo

1. **Faça as modificações, commit e push normalmente:**

git add .

git commit -m "Ajustes em versão antiga"

git push origin ajuste-no-commit-antigo

1. **Volte para a branch principal e faça o merge:**

git checkout main

git merge ajuste-no-commit-antigo

**✅ Agora as alterações feitas naquela versão antiga foram integradas ao projeto atual, de forma controlada.**

**🧹 4. Voltar o projeto para um commit anterior (de forma permanente)**

**Use isso com cuidado. Ideal em projetos pessoais. Pode sobrescrever o histórico.**

**Usando reset:**

git reset --hard <hash>

**O revert não apaga o histórico, mas cria um commit inverso. Ideal para uso em equipe.**

**🧯 5. Recuperar arquivos deletados ou sobrescritos**

**Se você deletou um arquivo e ainda não comitou:**

git restore nome-do-arquivo

**Se você já comitou a exclusão, use:**

git restore --source=<hash> nome-do-arquivo

**🧩 6. Comparar versões**

**Compare duas versões diferentes:**

git diff <hash1> <hash2>

**Compare o que mudou desde o último commit:**

**Compare com a branch principal:**

git diff main

**💡 Conclusão da Parte 1**

**Nesta parte, vimos:**

**✅ Como navegar no histórico do projeto  
✅ Como visualizar versões antigas com segurança  
✅ Como modificar versões antigas e aplicar as mudanças  
✅ Como voltar ou desfazer commits  
✅ Como recuperar arquivos perdidos  
✅ Como comparar diferentes versõe**

**🤝 Aula 3 – Parte 2: Cenários Reais de Colaboração com Git e GitHub**

**🎯 Objetivos da aula (Parte 2)**

* **Compreender como funciona o trabalho em equipe com Git**
* **Simular situações comuns em projetos colaborativos**
* **Praticar estratégias para evitar conflitos e melhorar a organização**
* **Aplicar boas práticas de versionamento**

**🧩 1. Organização de branches em equipe**

**📌 Boas práticas:**

* **Evite trabalhar diretamente na main ou master**
* **Crie branches nomeadas com clareza:**
  + **feature/login, bugfix/erro-login, hotfix/ajuste-layout**
* **Mantenha a branch principal estável**

**🛠 Exemplo prático:**

1. **Um desenvolvedor cria a branch feature/cadastro:**

git checkout -b feature/cadastro

1. **Faz o trabalho, comita e sobe para o GitHub**

git push -u origin feature/cadastro

1. **Abre um Pull Request para a main**

**🗂️ 2. Workflow GitHub comum em equipe**

main

│

├── feature/funcionalidade-x (desenvolvedor A)

├── feature/ajuste-y         (desenvolvedor B)

│

└── pull requests → revisão → merge na main

**🔁 Revisão e aprovação:**

* **Antes do merge, os membros da equipe revisam o código**
* **Comentários podem ser deixados diretamente no GitHub**
* **Após a aprovação, o responsável (ou a própria pessoa) faz o merge**

**🧪 3. Simulação: dois desenvolvedores trabalhando no mesmo projeto**

**Cenário:**

* **Aluno A cria uma funcionalidade em feature/login**
* **Aluno B corrige um bug em bugfix/email**
* **Ambos enviam Pull Requests para a branch main**
* **O professor atua como revisor (merge manual ou via GitHub)**

**💡 Mostre na prática:**

* **Como abrir Pull Request**
* **Como revisar e comentar um PR**
* **Como fazer o merge com "Create a merge commit", "Squash and merge", etc.**

**⚔️ 4. Conflitos de merge e como resolvê-los**

**O que são conflitos?**

**Quando dois commits alteram a mesma linha de um mesmo arquivo, o Git não sabe qual manter.**

**Simulação prática:**

1. **Aluno A edita App.js e comita**
2. **Aluno B também edita App.js na mesma linha e comita**
3. **Ao fazer o merge, surge o conflito**

**🛠 Como resolver:**

* **Git vai marcar os conflitos assim:**

<<<<<<< HEAD

Linha da branch atual

=======

Linha da branch que está sendo mesclada

>>>>>>> nome-da-branch

**O desenvolvedor deve editar o arquivo e escolher a versão correta**

**Depois disso:**

git add arquivo-com-conflito

git commit

**🔄 5. Sincronizando seu repositório com a branch principal**

**Para manter sua branch atualizada com a main:**

git checkout main

git pull origin main

git checkout sua-branch

git merge main

**Ou usando rebase (para histórico mais limpo):**

git checkout sua-branch

git fetch origin

git rebase origin/main

**🔍 Linha por linha:**

**1. git checkout sua-branch**

**Você está dizendo:  
➡️ *"Quero trabalhar na minha branch de funcionalidade."***

**2. git fetch origin**

**Você está buscando (mas sem aplicar ainda) todas as mudanças mais recentes do repositório remoto.  
➡️ *"Atualiza meu Git local com o que tem no GitHub, mas não aplica em nenhuma branch ainda."***

**Esse passo é importante porque garante que você está rebasing contra a versão mais atual da origin/main.**

**3. git rebase origin/main**

**Aqui está o truque:  
➡️ *"Pegue todos os commits da origin/main e coloque-os antes dos meus commits. Depois, aplique os meus commits por cima, como se eles tivessem sido feitos agora, em cima da versão mais recente do projeto."***

**⚠️ Cuidados:**

* **Nunca use rebase em commits que já foram enviados para o GitHub e compartilhados com outras pessoas, porque isso reescreve o histórico.**
* **Ideal para quando você ainda está trabalhando localmente ou numa branch só sua.**

**🧼 6. Boas práticas de trabalho em equipe com Git**

* **Use commits pequenos e descritivos**
* **Nunca suba senhas ou arquivos grandes/sensíveis**
* **Faça pull com frequência para evitar conflitos**
* **Crie Pull Requests organizados e bem documentados**
* **Atualize a branch principal sempre antes de iniciar algo novo**