



# Git

## DO ZERO AO AVANÇADO

Guia Prático e Visual para Desenvolvedores

**DeValeCode Technology**

Diego Nascimento

# Aula 1 – O que é Git e como começar

## Objetivo da aula:

Ao final desta aula, você entenderá o que é o Git, sua importância no desenvolvimento de software, como instalá-lo e usá-lo pela primeira vez.

---

## 1. Introdução: O que é Git?

### Definição:

Git é um **sistema de controle de versão distribuído**, criado por Linus Torvalds em 2005. Ele permite que você:

- Acompanhe o histórico de alterações dos arquivos.
- Trabalhe em equipe sem sobrescrever o trabalho de outros.
- Restaure versões anteriores do seu projeto facilmente.

### Por que usar Git?

- Segurança: Armazena todas as versões do código.
  - Colaboração: Várias pessoas podem trabalhar no mesmo projeto simultaneamente.
  - Reversibilidade: Errou algo? Pode voltar a uma versão anterior.
  - Popularidade: É o sistema mais usado no mundo do desenvolvimento.
- 

## 2. Instalando o Git

### Passo 1: Acesse o site oficial

Vá para: <https://git-scm.com>

### Passo 2: Baixe conforme seu sistema operacional

- **Windows:** Baixe o instalador e siga o assistente padrão.

**macOS:** Use o Homebrew:

bash

```
brew install git
```

- 

**Linux (Ubuntu):**

bash

```
sudo apt install git
```

- 

---

### 3. Abrindo o Git Bash

 **No Windows:**

- Após a instalação, abra o menu iniciar e digite "**Git Bash**".
- Também é possível abrir clicando com o botão direito em qualquer pasta no Windows Explorer e escolhendo "**Git Bash Here**".

### **Navegando até seu projeto**

No terminal do Git Bash, use o comando `cd` para acessar sua pasta de trabalho. Exemplo:

bash

```
cd Desktop/meu-projeto
```

---

### 4. Verificando a instalação

Digite no terminal:

bash

```
git --version
```

Se tudo estiver certo, você verá a versão do Git instalada, como:

nginx

```
git version 2.44.0.windows.1
```

---



### Atividade prática (5 a 10 minutos)

1. Instale o Git em sua máquina.
  2. Abra o Git Bash.
  3. Navegue até uma pasta de teste no seu computador.
  4. Execute `git --version` e veja o resultado.
- 



### Dica para iniciantes:

- Evite espaços em nomes de pastas e arquivos no começo.
  - Use sempre nomes simples: `meu-projeto`, `site-portfolio`, etc.
- 

### ? Perguntas frequentes

#### 1. Git é o mesmo que GitHub?

➡ Não. Git é a ferramenta de versionamento. GitHub é uma plataforma online que usa Git para hospedar e compartilhar projetos.

#### 2. Preciso saber programação para usar Git?

➡ Não necessariamente. Git pode ser usado por qualquer pessoa que queira versionar arquivos de texto, como documentos ou scripts.

---

## Aula 2 – Criando seu primeiro repositório Git

### Objetivo da aula:

Aprender a criar uma pasta de projeto em seu computador, abrir o Git Bash na pasta correta e iniciar um repositório Git local com `git init`.

---

### 1. Onde criar seu projeto?

Você pode escolher qualquer local no seu computador para criar seus projetos. Os dois locais mais comuns são:

- Área de Trabalho (Desktop)
  - Pasta Documentos (Documents)
- 

### 2. Exemplo 1 – Criar projeto na Área de Trabalho (Desktop)

#### Passo a passo:

1. Vá até sua **Área de Trabalho**.
2. Crie uma nova pasta chamada `meu-projeto`.
  - Clique com o botão direito > **Novo > Pasta**
3. Clique com o botão direito **dentro da pasta** e selecione **"Git Bash Here"**.
  - Isso abrirá o terminal Git já apontando para essa pasta.

#### Alternativa: Criar via terminal Git Bash

Abra o Git Bash e digite:

bash

```
cd ~/Desktop      # Vai para a Área de Trabalho
mkdir meu-projeto # Cria a pasta do projeto
cd meu-projeto    # Entra na pasta
```

---

### 3. Exemplo 2 – Criar projeto na pasta Documentos (Documents)

Abra o Git Bash e digite:

bash

```
cd ~/Documents      # Vai para a pasta Documentos
mkdir meu-projeto    # Cria a pasta do projeto
cd meu-projeto       # Entra na pasta
```


---

### 4. Inicializando o repositório Git

Depois de estar **dentro da pasta do seu projeto**, rode o comando:

bash

```
git init
```

 O que isso faz?

Esse comando transforma a pasta comum em um **repositório Git**. Ele cria uma subpasta oculta chamada `.git`, onde todo o histórico e configurações do Git são armazenados.

Você verá a mensagem:

swift

```
Initialized empty Git repository in
/c/Users/SeuNome/Desktop/meu-projeto/.git/
```

---

### Atividade prática (5 a 10 minutos)

1. Escolha se quer criar seu projeto no Desktop ou em Documentos.
  2. Crie a pasta com o nome `meu-projeto`.
  3. Abra o Git Bash dentro da pasta (clcando com botão direito ou usando `cd`).
  4. Execute `git init` e verifique se a mensagem apareceu.
-

## ? Dica rápida: Como saber que o Git está ativo?

Depois de rodar `git init`, o terminal pode mostrar o nome da branch, assim:

```
bash
```

```
meu-projeto (main) $
```

Isso é um sinal de que o Git já está funcionando dentro daquela pasta!

## 🎓 Aula 3 – `git init`: Inicializando um Repositório + Usando o `git help`

---

### 🎯 Objetivos da Aula

1. Aprender como transformar uma pasta comum em um repositório Git usando `git init`.
  2. Utilizar o comando `git help` para acessar a documentação oficial do Git diretamente pelo terminal.
- 

## 🧩 Parte 1 – `git init`: Inicializando um repositório

---

### 📖 Conceito

O comando `git init` transforma uma pasta comum em um **repositório Git local**.

🔍 O que acontece ao rodar `git init`?

- É criada uma **pasta oculta chamada `.git`**.

- Essa pasta contém **todo o histórico de alterações, configurações, branches**, e muito mais.
  - A partir desse momento, o Git começará a rastrear o conteúdo da pasta.
- 

## Exemplo prático

### Passos:

1. Crie uma pasta chamada **meu-projeto** no seu Desktop (ou Documentos).
2. Abra o **Git Bash dentro dessa pasta**.
  - Dica: Clique com o botão direito e escolha "Git Bash Here".
3. No terminal, digite:

```
bash
```

```
git init
```

### Resultado esperado:

```
bash
```

```
Initialized empty Git repository in  
/c/Users/seu-nome/Desktop/meu-projeto/.git/
```

Agora, sua pasta está "Gitificada"! 🎉

---

## Dica Importante

Você **só precisa rodar `git init` uma vez por projeto**. Depois disso, o Git já estará pronto para acompanhar tudo o que acontece nessa pasta.

---



## Parte 2 – **git help**: Aprendendo com o próprio Git

---

### Objetivo

Aprender a usar a **documentação interna do Git** para se tornar independente na hora de aprender comandos novos.

---

### Conceito

O Git possui um sistema de ajuda embutido que mostra explicações detalhadas de cada comando. Isso é ótimo para tirar dúvidas rapidamente!

---

### Exemplos práticos

Digite no terminal:


bash

```
git help init
```

ou

bash

```
git help commit
```

 Isso abrirá a documentação oficial do comando, com:

- Descrição
  - Sintaxe
  - Exemplos
  - Opções avançadas
-

## ✓ Dicas alternativas

Você também pode usar:

- `git status --help`
  - `git add --help`
  - `man git-init` (no **Linux/macOS**, usa o manual do sistema)
- 



## Atividade prática (10 minutos)

1. Crie uma nova pasta chamada `exemplo-git`.
  2. Abra o Git Bash dentro dela.
  3. Rode o comando `git init`.
  4. Rode o comando `git help init` e explore as informações mostradas.
  5. Como desafio, execute também `git help status` e anote algo novo que aprendeu.
- 



## Extra: O que tem dentro da pasta `.git`?

Se quiser ver, rode no terminal:

```
bash
```

```
ls -a
```

Você verá `.git` lá! Mas **nunca edite os arquivos dentro dela manualmente**.

## Aula 4 – Configurando Git com **git config** + Verificando mudanças com **git status**

---

### Objetivos da Aula

1. Aprender a configurar o nome e o e-mail que o Git usará para registrar seus commits.
  2. Usar o comando **git status** para acompanhar as mudanças no repositório.
- 

## Parte 1 – **git config**: Configurando nome e e-mail

---

### Conceito

O Git precisa saber **quem está fazendo cada alteração** no projeto. Para isso, você deve configurar:

- Seu **nome** (como será exibido nos commits)
- Seu **e-mail** (associado ao commit)

Essas informações aparecem sempre no histórico do Git.

---

### Comandos principais

**Configuração global (para todos os projetos da máquina):**

bash

```
git config --global user.name "Seu Nome"
git config --global user.email "seu@email.com"
```

Exemplo:

bash

```
git config --global user.name "Ana Silva"
```

```
git config --global user.email "ana.silva@gmail.com"
```

### Verificando a configuração:

bash

```
git config --list
```

📌 Isso mostra uma lista com todas as configurações atuais do Git no seu sistema.

---

### ✅ Dica

Se você quiser definir **configurações diferentes para um projeto específico**, não use o **--global**:

bash

```
git config user.name "Nome Local"
git config user.email "emaillocal@exemplo.com"
```

Essas configurações ficarão salvas **apenas no repositório atual**.

---

## 🔍 Parte 2 – **git status**: Verificando o estado do repositório

---

### 📘 Conceito

O comando **git status** mostra o **que está acontecendo dentro do repositório**:

- Quais arquivos foram **modificados**;
  - Quais estão **prontos para commit**;
  - Quais ainda **não estão sendo rastreados** pelo Git.
-

## Exemplo de uso

### Passos práticos:

1. Crie (ou entre em) uma pasta de projeto com Git inicializado.
2. Crie um novo arquivo com:

bash

```
echo "conteúdo" > arquivo.txt
```

3. Digite:

bash

```
git status
```

### Exemplo de saída:

text

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
    arquivo.txt
```

```
nothing added to commit but untracked files present (use "git add"
to track)
```

---

## Dica importante

Use `git status` antes e depois de qualquer comando importante, como:

- `git add`
- `git commit`
- `git reset`

Isso ajuda a entender exatamente o que está acontecendo no repositório.



## Atividade prática (10–15 minutos)

1. Abra um repositório Git local (use `git init` se ainda não tiver feito isso).
  2. Configure seu nome e e-mail com `git config`.
  3. Use `git config --list` para verificar.
  4. Crie um arquivo novo com `echo "Hello World" > hello.txt`.
  5. Use `git status` para ver o estado atual do repositório.
  6. Tente alterar o arquivo e rode `git status` novamente.
- 



## Perguntas frequentes

### 1. Preciso configurar nome e e-mail toda vez?

➡ Não, se você usar `--global`, a configuração vale para todos os seus projetos futuros.

### 2. O que significa "untracked"?

➡ São arquivos que o Git ainda não está monitorando. Você pode adicioná-los com `git add`.

## Aula 5 – Preparando e Salvando Alterações com `git add` e `git commit`

---

### Objetivos da Aula

1. Entender como adicionar arquivos modificados à área de preparação (staging) com `git add`.
  2. Aprender a salvar oficialmente essas alterações com `git commit`.
- 

## Parte 1 – `git add`: Preparando arquivos para o commit


---

### Conceito

O Git não registra todas as mudanças automaticamente. Primeiro, você precisa **especificar quais arquivos serão salvos** no próximo commit. Isso é feito com o comando:

```
bash
```

```
git add
```

 **Staging area** (área de preparação) é onde você coloca os arquivos que quer incluir no próximo commit.

---

### Exemplo prático

1. Crie ou modifique um arquivo:

```
bash
```

```
echo "linha nova" >> arquivo.txt
```

## 2. Verifique o status atual:

bash

```
git status
```

📌 Saída esperada:

text

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    modified:   arquivo.txt
```

## 3. Adicione o arquivo ao staging:

bash

```
git add arquivo.txt
```

## 4. Verifique o novo status:

bash

```
git status
```

📌 Agora o arquivo aparece em verde, indicando que está pronto para o commit:

text

```
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
    modified:   arquivo.txt
```

---

## ✅ Dicas úteis com **git add**

- Adicionar **todos os arquivos modificados** de uma vez:

bash

```
git add .
```



- Adicionar arquivos **específicos**:

bash

```
git add nome-do-arquivo.txt outro-arquivo.md
```

- Para **retirar um arquivo do staging** (caso tenha adicionado por engano):

bash

```
git reset nome-do-arquivo.txt
```

---

## Parte 2 – **git commit**: Salvando as alterações

---

### Conceito

Um **commit** é como um ponto de salvamento. Ele grava permanentemente no histórico do projeto tudo o que estava na staging area.

Cada commit tem:

- Um **identificador único**
- Uma **mensagem descritiva**
- O **autor (você)**
- Um **registro das alterações feitas**

---

### Exemplo prático

Depois de usar **git add**, salve a mudança com:

bash

```
git commit -m "Adiciona arquivo.txt com conteúdo inicial"
```

📌 A flag `-m` permite escrever a mensagem direto no terminal.

---

### ✅ Dicas importantes

- Sempre escreva mensagens **claras e objetivas**, por exemplo:
  - `"Cria arquivo de configuração inicial"`
  - `"Corrige erro de digitação no README"`
- O commit só registra arquivos que foram **adicionados com `git add`**. Arquivos fora da staging area não serão incluídos.
- Para arquivos **já rastreados**, você pode pular o `add` e usar:

bash

```
git commit -am "Mensagem do commit"
```

---



### Atividade prática (10 a 15 minutos)

1. Crie ou edite um arquivo no seu projeto.
  2. Verifique o status com `git status`.
  3. Adicione o arquivo com `git add`.
  4. Verifique novamente com `git status`.
  5. Faça um commit com `git commit -m "Mensagem descritiva"`.
  6. Teste também o comando `git commit -am` em arquivos já existentes.
-

## Perguntas frequentes

### 1. Posso fazer um commit sem usar `git add`?

➡ Só se os arquivos já forem rastreados e você usar `-am`. Para arquivos novos, `git add` é obrigatório.

### 2. Como saber o que foi incluído no commit?

➡ Use `git status` antes do commit para ver exatamente o que está na staging area.

## Aula 6 – Visualizando e Comparando o Histórico com Git

---

### Objetivos da Aula

1. Usar `git log` para ver o histórico completo de commits.
  2. Utilizar `git show` para examinar os detalhes de um commit específico.
  3. Usar `git diff` para comparar modificações feitas nos arquivos.
- 

## Parte 1 – `git log`: Visualizando o histórico de commits

---

### Conceito

O comando `git log` mostra todos os **commits realizados**, incluindo:

- O **hash** único do commit
- O **autor**
- A **data e hora**

- A mensagem descritiva

Isso ajuda você a acompanhar todas as mudanças feitas no projeto desde o início.

---

### Exemplo prático

bash

```
git log
```

 Saída típica:

text

```
commit e43a48d5f9e3c77d6e1d4fa83...
Author: Seu Nome <email@example.com>
Date:   Sat May 4 10:00 2025
```

```
    Adiciona arquivo.txt com conteúdo inicial
```

---

### Dicas úteis com **git log**

- Log em uma linha por commit:

bash

```
git log --oneline
```

♦ Exemplo de saída:

text

```
e43a48d (HEAD -> main) Adiciona arquivo.txt com conteúdo inicial
```

- Log com gráfico visual das branches:

bash

```
git log --oneline --graph --all
```

🧠 Muito útil para ver ramificações e merges!

---

## 🧩 Parte 2 – `git show`: Exibindo detalhes de um commit

---

### 📖 Conceito

O comando `git show` exibe o conteúdo completo de um **commit específico**, incluindo:

- Diferenças de código (diff)
  - Mensagem do commit
  - Autor e data
- 

### 🔧 Exemplo de uso

**Ver o commit mais recente:**

bash

```
git show
```

**Ver um commit específico (use o hash do `git log`):**

bash

```
git show e43a48d
```

📌 Mostra exatamente o que foi adicionado ou removido no commit.

---

## Dica

Você pode copiar o **início do hash** (por exemplo, os 7 primeiros caracteres) para identificar um commit:

bash

```
git show abc1234
```

---

## Parte 3 – **git diff**: Comparando mudanças

---

### Conceito

**git diff** mostra as **diferenças entre versões de arquivos**:

- Antes de adicionar ao staging
  - Após adicionar
  - Entre commits
  - Entre branches
- 

### Exemplos práticos

#### 1. Ver alterações não adicionadas (working directory vs. staging):

bash

```
git diff
```

#### 2. Ver mudanças desde o último commit (HEAD):

bash

```
git diff HEAD
```

### 3. Comparar duas branches:

bash

```
git diff main nova-feature
```

---

### ✓ Como ler o `git diff`

- Linhas com `-` (vermelho) foram **removidas**
  - Linhas com `+` (verde) foram **adicionadas**
- 



## Atividade prática

1. Faça uma alteração em algum arquivo do repositório.
  2. Execute `git diff` para ver o que mudou.
  3. Use `git add` e depois `git commit -m "Mensagem do commit"`.
  4. Use `git log` para ver o histórico.
  5. Copie o hash de um commit e execute `git show <hash>`.
- 

## 💬 Perguntas frequentes

### 1. Posso desfazer um commit ao ver um erro com `git log`?

➡ Sim. Você pode usar comandos como `git revert` ou `git reset` (veremos em aulas futuras).

### 2. Posso comparar dois commits diretamente?

➡ Sim, com:

bash

```
git diff <hash1> <hash2>
```

## Aula 7 – Trabalhando com Branches no Git

---

### Objetivos da Aula

1. Entender como criar, listar, mudar e excluir branches.
  2. Aprender a usar `git switch`, `git checkout` e `git merge` corretamente.
  3. Trabalhar com diferentes versões do projeto de forma segura e organizada.
- 

## Parte 1 – `git branch`: Gerenciando ramificações

---

### Conceito

**Branches** (ramificações) permitem trabalhar em **funcionalidades isoladas**, sem afetar o código principal (geralmente a branch `main` ou `master`). Isso facilita o desenvolvimento paralelo e colaborações.

---

### Exemplos práticos

#### Listar todas as branches:

bash

```
git branch
```

- ♦ A branch atual será marcada com `*`.

#### Criar uma nova branch:

bash

```
git branch nova-feature
```

Isso cria a branch, mas **não muda para ela**.

#### Deletar uma branch:

bash



```
git branch -d nova-feature
```

Use com **-D** (letra maiúscula) se quiser forçar a exclusão de uma branch com mudanças não mescladas:

bash

```
git branch -D nova-feature
```

---

### Dica

Branches são **leves e rápidas** de criar. Não tenha medo de usar várias!

---

## Parte 2 – **git switch**: Mudando de branch (forma moderna)

---

### Conceito

**git switch** é o comando moderno e mais seguro para **trocar de branch** ou **criar uma nova já trocando para ela**.

---

### Exemplos práticos

 Mudar para uma branch existente:

bash

```
git switch nova-feature
```

 Criar e trocar para uma nova branch:

bash

```
git switch -c nova-feature
```

---

## ✓ Dica

Prefira `git switch` em vez de `git checkout` para **evitar erros acidentais**.

---

## 🧩 Parte 3 – `git checkout`: Alternar ou restaurar arquivos

---

### 📖 Conceito

Antes de `git switch`, o comando usado para tudo era `git checkout`. Ele ainda é válido, mas tem usos mais específicos hoje.

---

### 🔧 Exemplos

#### 🔄 Trocar de branch (forma antiga):

bash

```
git checkout main
```

#### 🔧 Restaurar um arquivo para o último commit:

bash

```
git checkout -- arquivo.txt
```

---

### ⚠ Cuidado!

`git checkout` pode sobrescrever alterações. Use com atenção ou prefira `git switch` para mudar de branch.

---

## Parte 4 – **git merge**: Juntando branches

---

### Conceito

Quando uma branch com uma nova funcionalidade estiver pronta, você pode **mesclar** (**merge**) com a branch principal (**main**, por exemplo).

---

### Exemplo prático

#### 1. Certifique-se de estar na **main**:


bash

```
git switch main
```

#### 2. Faça a mesclagem com a branch **nova-feature**:

bash

```
git merge nova-feature
```

 O Git tentará unir as mudanças automaticamente.

---

### Dicas sobre merge

- Use **git status** antes e depois de um **merge** para entender o que está acontecendo.
- Se houver **conflito**, o Git avisará. Você precisará editar os arquivos manualmente, depois:

bash

```
git add arquivo-com-conflito.txt  
git commit
```

---



## Atividade prática (15 minutos)

1. Crie uma branch nova chamada `melhoria-ui`.
  2. Altere um arquivo (ex: `index.html`) e faça um commit.
  3. Volte para `main` com `git switch main`.
  4. Mescle as mudanças com `git merge melhoria-ui`.
  5. Use `git log --oneline --graph` para ver o histórico visual.
- 



## Perguntas frequentes

### 1. O que acontece se eu mesclar uma branch duas vezes?

➡ O Git ignora commits já mesclados. Nada será duplicado.

### 2. Posso deletar a branch depois de mesclar?

➡ Sim! Se a branch já foi integrada, você pode excluí-la com segurança usando `git branch -d`.

### 3. `git merge` pode causar perda de dados?

➡ Não. Se houver conflitos, o Git pausará o processo até que você resolva.

# Aula 8 – Trabalhando com Repositórios Remotos no Git

---

## Objetivos da Aula

- Conectar seu repositório local com um repositório remoto (como o GitHub).
  - Aprender a clonar, enviar e receber atualizações.
  - Entender a diferença entre `push`, `pull`, `fetch` e `remote`.
- 

## Parte 1 – `git remote`: Conectando com repositórios remotos

---

### Conceito

O comando `git remote` serve para **adicionar, visualizar ou remover** conexões com repositórios hospedados online.

---

### Exemplos práticos

 Ver os repositórios remotos configurados:

bash

```
git remote -v
```

 Adicionar um repositório remoto chamado `origin`:

bash

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

 Remover um repositório remoto:

bash

```
git remote remove origin
```

---

### ✓ Dica

O nome padrão mais usado é `origin`, mas você pode usar outro nome se quiser.

---

## ✖ Parte 2 – `git clone`: Copiando um repositório remoto

---

### 📘 Conceito

O comando `git clone` serve para **baixar um repositório remoto** completo, incluindo o histórico de versões.

---

### 🔧 Exemplo prático

bash

```
git clone https://github.com/seu-usuario/seu-repo.git
```

Isso cria uma **nova pasta** com todo o conteúdo do repositório.

---

### ✓ Dica

Depois de clonar, você já pode usar `git status`, criar branches e trabalhar normalmente.

---

## 🧩 Parte 3 – **git push**: Enviando mudanças para o repositório remoto

---

### 📘 Conceito

O comando **git push** envia seus **commits locais** para o repositório remoto.

---

### 🔧 Exemplo prático

bash

```
git push origin main
```

Isso envia as alterações da sua branch **main** local para o remoto chamado **origin**.

---

### ✅ Dicas

- No **primeiro push**, defina o upstream:

bash

```
git push -u origin main
```

- Esse comando conecta sua branch local com a remota, facilitando os próximos envios.
-

## 🧩 Parte 4 – `git pull`: Baixando e aplicando mudanças do remoto

---

### 📖 Conceito

O `git pull` é um atalho para `git fetch` + `git merge`. Ele **baixa e aplica automaticamente** as atualizações do repositório remoto.

---

### 🔧 Exemplo prático

bash

Copiar código

```
git pull origin main
```

Esse comando sincroniza sua cópia local com as últimas alterações da branch `main` no repositório remoto.

---

### ✅ Dica

Sempre use `git pull` **antes de começar a trabalhar**, para garantir que você está com o código mais recente.

---

## 🧩 Parte 5 – `git fetch`: Baixando sem aplicar

---

### 📖 Conceito

`git fetch` apenas **baixa as atualizações** do repositório remoto, mas **não as aplica automaticamente**. Você decide quando integrá-las.

---

### 🔧 Exemplo prático

bash

Copiar código

```
git fetch origin
```



Depois, você pode ver as diferenças entre sua branch local e a remota com:

bash

Copiar código

```
git diff main origin/main
```

---

### Dica

Use `git fetch` para **verificar mudanças no repositório remoto sem afetar seu código local imediatamente**.

---

## Resumo visual

Comando	Função
<code>git remote -v</code>	Ver repositórios remotos conectados
<code>git remote add</code>	Adicionar um novo repositório remoto
<code>git clone</code>	Baixar uma cópia de um repositório remoto
<code>git push</code>	Enviar commits locais para o repositório remoto
<code>git pull</code>	Baixar e aplicar alterações do repositório remoto
<code>git fetch</code>	Baixar alterações do repositório remoto (sem aplicar)

---

## Atividade prática

1. Crie um repositório no GitHub chamado `projeto-exemplo`.

No terminal:

bash

Copiar código

```
git init
```

```
git remote add origin
```

```
https://github.com/seu-usuario/projeto-exemplo.git
```

2.

Crie um arquivo, faça commit e envie com:

```
bash
Copiar código
git add .
git commit -m "Commit inicial"
git push -u origin main
```

3.

Em outro computador, tente clonar com:

```
bash
Copiar código
git clone https://github.com/seu-usuario/projeto-exemplo.git
```

4.

## Aula 9 – Desfazendo Mudanças com Git

---

### Objetivo da Aula

Aprender a desfazer ou guardar alterações com segurança usando:

- `git reset` (redefinir mudanças e commits)
  - `git revert` (desfazer commits já enviados)
  - `git stash` (guardar mudanças temporárias)
- 

## Parte 1 – `git reset`: Desfazendo Commits ou Alterações

---

### Conceito

O `git reset` redefine o estado do seu repositório para um ponto anterior.

- Pode tirar arquivos da **área de staging**.
  - Pode apagar **commits recentes**.
  - É **perigoso** se você já enviou os commits ao repositório remoto!
- 

## Exemplos práticos

### Tirar um arquivo do staging:

bash

Copiar código

```
git reset arquivo.txt
```

O arquivo volta ao estado de "modificado", mas não preparado para commit.

---

### Voltar um commit sem apagar arquivos:

bash

Copiar código

```
git reset --soft HEAD~1
```

Remove o último commit, mas mantém os arquivos na staging area.

---

### Voltar e apagar completamente mudanças:

bash

Copiar código

```
git reset --hard HEAD~1
```

 Cuidado! Isso apaga o commit e todas as mudanças nos arquivos.

---

## Atenção

- **Não use `--hard`** se o commit já foi enviado para o GitHub (pode causar perda de dados).

- Para visualizar antes de resetar: `git log` ou `git reflog`.
- 

## 🧩 Parte 2 – `git revert`: Desfazendo um commit com segurança

---

### 📘 Conceito

O `git revert` desfaz um commit **sem apagar o histórico**.

Ideal para projetos colaborativos ou quando o commit **já foi enviado para o remoto**.

---

### 🔧 Exemplo prático

bash

Copiar código

```
git revert a1b2c3d
```

Substitua `a1b2c3d` pelo hash do commit (pegue com `git log`).

---

### ✅ Dicas

- Cria um novo commit que "anula" as mudanças anteriores.
  - Pode ser usado várias vezes para reverter commits específicos.
- 

## 🧩 Parte 3 – `git stash`: Guardando mudanças temporárias

---

### 📘 Conceito

O `git stash` salva suas alterações atuais **sem precisar fazer commit**.

Ideal para:

- Trocar de branch rapidamente.
  - Evitar conflito sem perder seu progresso atual.
- 

## Exemplos práticos

### Guardar as mudanças atuais:

bash

Copiar código

```
git stash
```

### Ver a lista de mudanças guardadas:

bash

Copiar código

```
git stash list
```

### Recuperar as mudanças:

bash

Copiar código

```
git stash apply
```

### Remover o stash recuperado:

bash

Copiar código

```
git stash drop
```

---

## Dica útil

Você pode dar nome aos stashes:

bash

Copiar código

```
git stash save "Mudanças no formulário de login"
```

E aplicar uma stash específica:

bash

Copiar código

```
git stash apply stash@{1}
```

---

## Resumo visual

Comando	O que faz
<code>git reset arquivo.txt</code>	Tira arquivo da staging area
<code>git reset --soft HEAD~1</code>	Apaga commit, mas mantém as mudanças
<code>git reset --hard HEAD~1</code>	Apaga commit e as alterações nos arquivos
<code>git revert &lt;hash&gt;</code>	Cria um novo commit que desfaz um commit anterior
<code>git stash</code>	Guarda alterações temporárias
<code>git stash apply</code>	Restaura as alterações guardadas
<code>git stash drop</code>	Remove alterações guardadas

---

## Desafio Prático

1. Faça um commit com um erro de digitação.
2. Use `git revert` para desfazer.
3. Em seguida, adicione algo novo ao código, mas antes de mudar de branch, use `git stash`.
4. Volte depois e recupere com `git stash apply`.

## Aula 10 – Avançado: Reorganizando e Investigando o Histórico com Git

---

## Objetivo da Aula

Explorar comandos que:

- Reorganizam o histórico (`git rebase`, `git cherry-pick`)
- Marcam versões (`git tag`)
- Investigam bugs (`git bisect`)
- Rastreiam autores de mudanças (`git blame`)

---

## Parte 1 – `git rebase`: Organizando o histórico de commits

---

### Conceito

O `git rebase` move os commits de uma branch para “cima” de outra, criando um histórico **linear e mais limpo**.

É uma **alternativa ao merge**, mas **reescreve o histórico**.

---

### Exemplo prático

bash

Copiar código

```
git checkout nova-feature
```

```
git rebase main
```

Isso reaplica os commits da branch `nova-feature` como se tivessem sido criados a partir da `main`.

---

### Em caso de conflito

Resolva manualmente os arquivos afetados e siga com:

```
bash
Copiar código
git add arquivo.txt
git rebase --continue
```

Cancelar o rebase:

```
bash
Copiar código
git rebase --abort
```

---

### ✓ Dicas

- Use apenas em commits **ainda não enviados ao remoto**.
  - Deixe o histórico linear para revisões de código mais fáceis.
- 

## 🧩 Parte 2 – git cherry-pick: Copiando um commit específico

---

### 📘 Conceito

O `git cherry-pick` permite copiar um commit **isolado** de uma branch para outra, sem trazer todos os commits intermediários.

---

### 🖋️ Exemplo prático

```
bash
Copiar código
git cherry-pick 1a2b3c4
```

Isso aplica as mudanças do commit com hash `1a2b3c4` na branch atual.

---

### ✓ Dica



Use com atenção: se o código mudou muito, pode gerar conflitos.

---

## Parte 3 – **git tag**: Marcando versões importantes

---

### Conceito

Tags são **etiquetas permanentes** em commits importantes, como versões de produção (**v1.0**, **v2.0**, etc).

---

### Exemplo prático

Criar uma tag:

bash

Copiar código

```
git tag v1.0
```

Enviar a tag para o remoto:

bash

Copiar código

```
git push origin v1.0
```

Listar tags:

bash

Copiar código

```
git tag
```

Ver o conteúdo de uma tag:

bash

Copiar código

```
git show v1.0
```

---

### Dica

Você pode voltar no tempo com:

bash

Copiar código

```
git checkout v1.0
```

---

## Parte 4 – **git bisect**: Encontrando onde surgiu um bug

---

### Conceito

O **git bisect** ajuda a encontrar o commit exato onde um bug foi introduzido, por meio de **busca binária**.

---

### Exemplo prático

bash

Copiar código

```
git bisect start
git bisect bad           # commit atual com o bug
git bisect good v1.0     # commit onde tudo funcionava
```

O Git alternará entre commits intermediários. Teste o projeto a cada passo e informe se está bom ou ruim:

bash

Copiar código

```
git bisect good
# ou
git bisect bad
```

Finalize:

bash

Copiar código

```
git bisect reset
```

---

## ✓ Dica

Perfeito para times que trabalham em projetos grandes e precisam rastrear bugs antigos.

---

## 🧩 Parte 5 – `git blame`: Descobrimos quem escreveu o quê

---

### 📘 Conceito

Com `git blame`, você pode ver **quem escreveu cada linha de um arquivo**, quando e em qual commit.

---

### 🔧 Exemplo prático

bash

Copiar código

```
git blame index.js
```

Cada linha exibirá o **hash do commit**, **autor**, **data** e o conteúdo.

---

## ✓ Dica

Você pode usar junto com `git show` para examinar o commit responsável:

bash

Copiar código

```
git show 1a2b3c4
```

---

## 📌 Resumo Visual

Comando	Função
<code>git rebase main</code>	Reorganiza commits sobre outra base
<code>git cherry-pick &lt;hash&gt;</code>	Copia um commit específico de outra branch

<code>git tag v1.0</code>	Marca um ponto importante no histórico
<code>git bisect</code>	Encontra o commit onde um bug começou
<code>git blame arquivo</code>	Mostra quem alterou cada linha de um arquivo

---

## Desafio Prático

1. Crie uma branch nova a partir da main.
2. Faça dois commits.
3. Volte para main e use `git cherry-pick` para trazer apenas o segundo.
4. Use `git tag` para marcar uma versão.
5. Introduza um bug, e use `git bisect` para descobrir o commit responsável.

## Aula Final – Encerramento e Cheatsheet Visual de Git

---

### Objetivo

Revisar todos os comandos aprendidos em um **cheatsheet visual**, facilitando:

- A memorização
  - A consulta rápida no dia a dia
  - A fixação do fluxo de trabalho com Git
- 

### Conceito

Um **cheatsheet visual** é um resumo gráfico com os principais comandos do Git e suas finalidades. Ele ajuda você a entender **quando** e **como** usar cada comando, tornando seu uso mais natural e eficiente.

---

## Cheatsheet Visual de Git

---

### Inicialização do Repositório

Comando	Função
<code>git init</code>	Cria um repositório local
<code>git clone &lt;url&gt;</code>	Copia um repositório remoto

---

### Adição e Commit

Comando	Função
<code>git add .</code>	Adiciona todas as mudanças
<code>git commit -m "msg"</code>	Faz o commit com mensagem
<code>git status</code>	Mostra o estado atual dos arquivos
<code>git log</code>	Lista o histórico de commits
<code>git show &lt;hash&gt;</code>	Mostra os detalhes de um commit

---

### Branches (Ramificações)

Comando	Função
<code>git branch</code>	Lista ou cria branches
<code>git branch -d &lt;nome&gt;</code>	Apaga uma branch
<code>git switch &lt;nome&gt;</code>	Troca de branch

<code>git switch -c &lt;nome&gt;</code>	Cria e troca de branch
<code>git merge &lt;branch&gt;</code>	Mescla branch no ponto atual

---

## Repositórios Remotos

Comando	Função
<code>git remote -v</code>	Lista os repositórios remotos
<code>git remote add origin &lt;url&gt;</code>	Adiciona repositório remoto chamado origin
<code>git push origin main</code>	Envia as alterações da branch main
<code>git pull origin main</code>	Baixa e aplica as alterações do remoto
<code>git fetch</code>	Baixa dados do remoto (sem aplicar ainda)

---

## Controle de Erros e Alterações

Comando	Função
<code>git reset arquivo.txt</code>	Retira o arquivo da área de staging
<code>git reset --soft HEAD~1</code>	Desfaz último commit, mantém alterações
<code>git reset --hard HEAD~1</code>	Desfaz tudo, inclusive alterações
<code>git revert &lt;hash&gt;</code>	Cria novo commit que desfaz um commit anterior
<code>git stash</code>	Guarda mudanças temporárias
<code>git stash apply</code>	Restaura o stash
<code>git stash drop</code>	Remove stash da lista

---

## Comparação e Histórico

Comando	Função
<code>git diff</code>	Compara mudanças não comitadas
<code>git diff HEAD</code>	Compara com o último commit
<code>git diff main nova-feature</code>	Compara duas branches
<code>git blame arquivo.txt</code>	Mostra quem modificou cada linha

---

## Recursos Avançados

Comando	Função
<code>git rebase main</code>	Reaplica commits sobre a branch <code>main</code>
<code>git cherry-pick &lt;hash&gt;</code>	Aplica um commit específico na branch atual
<code>git tag v1.0</code>	Marca uma versão importante
<code>git push origin v1.0</code>	Envia a tag para o repositório remoto
<code>git bisect</code>	Localiza o commit que causou um bug

---


## Dica Final

Imprima esse cheatsheet ou mantenha como arquivo PDF por perto. Ele será seu melhor aliado até que os comandos virem instinto!

Se quiser, posso gerar esse cheatsheet como **PDF ou imagem visual** com ícones, cores e layout otimizado.

---

## Encerramento

 Parabéns por chegar até aqui!

Você agora tem:

- ✓ Um domínio completo do fluxo de trabalho com Git
- ✓ Ferramentas para colaborar de forma profissional
- ✓ Conhecimento para reverter erros, investigar problemas e organizar o histórico com clareza