Git e GitHub

Material de Autoestudo para Programadores Iniciantes

Miguel Morales Hova

30 de dezembro de 2025

Sumário

1 Introdução

Este material foi desenvolvido para pessoas que **nunca tiveram contato com Git ou GitHub** e desejam aprender, de forma autodidata, a utilizar versionamento de código com segurança em projetos reais.

O foco está em:

- Clareza conceitual
- Progressão gradual de dificuldade
- Situações reais do dia a dia de um desenvolvedor
- Boas práticas utilizadas no mercado

2 Fundamentos

2.1 O que é versionamento de código

Objetivo da sessão: entender o problema que o Git resolve.

Versionamento de código é o controle do histórico de alterações de um projeto. Ele permite saber:

- O que foi alterado
- Quem alterou
- Quando alterou
- Por que alterou

Git vs GitHub

- **Git**: ferramenta local de versionamento
- **GitHub**: plataforma online para hospedar repositórios Git

Dica: Você pode usar Git sem GitHub, mas não GitHub sem Git.

2.2 Instalação e configuração inicial

Objetivo da sessão: instalar o Git e configurar sua identidade.

Configuração básica

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

# Verifica as configs
git config --list
```

Listing 1: Configuração inicial do Git

Erro comum: Não configurar nome e email faz com que seus commits fiquem sem identificação.

2.3 Criando o primeiro repositório

Objetivo da sessão: criar um repositório e realizar o primeiro commit.

```
mkdir projeto-git
cd projeto-git
git init
```

Listing 2: Criando um repositório Git

O Git cria uma pasta oculta chamada `.git`, onde todo o histórico é armazenado.

Primeiro commit

```
echo "Ol  Git" > README.md

git status          # Mostra o estado do reposit rio
git add README.md # Adiciona o arquivo     rea  de stage
git commit -m "Primeiro commit"
```

Listing 3: Primeiro commit

Boa prática: Commits representam unidades lógicas de trabalho, não apenas salvar arquivos.

2.4 Ciclo básico do Git

O fluxo básico do Git é:

1. Alterar arquivos

2. Adicionar ao stage

3. Criar commit

```
git status
git add arquivo.txt
git commit -m "Descricao clara da alteracao"
```

Listing 4: Ciclo básico do Git

Erro comum: Usar mensagens genéricas como "update"ou "alterações".

3 Nível Médio — Trabalho em Equipe

3.1 Histórico e comparação de versões

Objetivo da sessão: entender o histórico do projeto.

```
git log
git log --oneline
```

Listing 5: Visualizando histórico

```
git diff
```

Listing 6: Comparando alterações

Dica: O comando `git diff` é essencial para entender o que mudou antes de um commit.

3.2 Branches

Objetivo da sessão: trabalhar em paralelo sem quebrar o projeto principal.

```
git checkout -b nova-feature
```

Listing 7: Criando e acessando branches

```
git checkout main
```

Listing 8: Voltando para a branch principal

Erro comum: Desenvolver novas funcionalidades diretamente na branch `main`.

3.3 Merge e conflitos

Objetivo da sessão: unir branches corretamente.

```
git checkout main
git merge nova-feature
```

Listing 9: Realizando merge

Conflitos ocorrem quando duas alterações afetam a mesma linha de código.

Boa prática: Commits pequenos reduzem drasticamente conflitos.

3.4 GitHub e repositórios remotos

Objetivo da sessão: sincronizar repositório local com o GitHub.

```
git remote add origin https://github.com/usuario/repositorio.git
git push -u origin main
```

Listing 10: Conectando ao GitHub

```
git pull origin main
```

Listing 11: Atualizando o projeto local

3.5 Pull Requests

Pull Requests permitem:

- Revisão de código
- Discussão técnica
- Controle de qualidade

Fluxo comum:

1. Criar branch
2. Commitar
3. Push
4. Abrir Pull Request
5. Revisar e fazer merge

4 Nível Avançado — Segurança e Boas Práticas

4.1 Desfazendo erros

Objetivo da sessão: corrigir erros com segurança.

```
git restore arquivo.txt
```

Listing 12: Desfazendo alterações locais

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

Listing 13: Voltando commits

Erro comum: Usar `-hard` sem entender que ele apaga alterações permanentemente.

4.2 Arquivo `.gitignore`

Objetivo da sessão: evitar subir arquivos desnecessários ou sensíveis.

```
node_modules/
.env
*.log
```

Listing 14: Exemplo de .gitignore

Boa prática: Nunca versionar senhas, tokens ou arquivos de build.

4.3 Fluxo profissional simplificado

Estrutura comum:

- `main` — produção
- `develop` — desenvolvimento
- `feature/nome`
- `hotfix/erro`

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

Listing 15: Criando branch de feature

5 Conclusão

Ao concluir este material, o aluno será capaz de:

- Versionar projetos com Git
- Trabalhar com branches e merges
- Utilizar GitHub profissionalmente
- Resolver conflitos com segurança
- Aplicar boas práticas de mercado

> *Git não é sobre comandos, é sobre controle, colaboração e confiança.*