Manual de Git: Controle de Versão para Iniciantes

Prof. Cláudia Jacy Barenco Abbas

Março/2025

1. O que é Git?

Git é um sistema de controle de versão distribuído usado para rastrear mudanças em arquivos e coordenar trabalho colaborativo em projetos. É amplamente utilizado no desenvolvimento de software, mas pode ser aplicado a qualquer tipo de arquivo.

Principais Vantagens:

- Histórico completo de alterações.
- Facilita colaboração em equipe.
- Permite voltar a versões anteriores.
- Trabalho offline (cada usuário tem uma cópia local do repositório).

2. Instalação do Git

Windows

1. Acesse git-scm.com.
2. Baixe o instalador e siga as instruções.

MacOS

- Via Homebrew:

bash

```
brew install git
```

Linux (Debian/Ubuntu)

bash

```
sudo apt-get update && sudo apt-get install git
```

3. Configuração Inicial

Configure seu nome e e-mail (obrigatório para commits):

bash

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

Outras configurações úteis:

bash

```
git config --global core.editor "code --wait"  # Usar VS Code como editor
git config --global init.defaultBranch main    # Define branch principal como "main"
```

4. Comandos Básicos

Iniciar um Repositório

```
git init            # Cria um repositório Git vazio na pasta atual
```

Clonar um Repositório Existente

```
git clone https://github.com/usuario/repositorio.git
```

Verificar Status

```
git status          # Mostra arquivos modificados/rastreados

Adicionar Arquivos ao Stage
git add arquivo.txt   # Adiciona um arquivo específico
git add .             # Adiciona todos os arquivos modificados

Criar um Commit
git commit -m "Mensagem descritiva das alterações"

Ver Histórico de Commits
git log               # Exibe o histórico completo
git log --oneline     # Histórico resumido

Comparar Mudanças
git diff              # Mostra diferenças entre arquivos não commitados
git diff HEAD         # Compara com o commit anterior
```

5. Trabalhando com Branches (Ramificações)

```
Criar e Trocar de Branch
git branch nova-feature   # Cria uma nova branch
git checkout nova-feature # Troca para a branch

Ou, em uma linha:
git checkout -b nova-feature

Listar Branches
git branch            # Lista todas as branches locais
git branch -a         # Lista branches locais e remotas

Mesclar Branches
git checkout main         # Volta para a branch principal
git merge nova-feature  # Mescla "nova-feature" em "main"


Deletar Branch
git branch -d nova-feature  # Deleta a branch local
```

6. Trabalhando com Repositórios Remotos (GitHub, GitLab, etc.)

```
Adicionar um Repositorio Remoto
git remote add origin https://github.com/usuario/repositorio.git

Enviar Commits para o Remoto
git push -u origin main   # Primeiro push (define upstream)
git push                  # Push subsequente

Atualizar Repositório Local
git pull origin main     # Puxa mudanças e mescla localmente

Buscar Mudanças sem Mesclar
```

git fetch # Atualiza referências do remoto

7. Boas Práticas
1. Commits Atômicos: Cada commit deve ter uma única finalidade.
2. Mensagens Claras: Use mensagens no imperativo (ex: "Corrige bug de login").
3. Branch Estratégico: Use branches para features/fixes isolados.
4. gitignore: Exclua arquivos desnecessários (logs, binários).
5. Sincronização Frequente: Faça `pull` regularmente para evitar conflitos.

8. Workflows Comuns
Git Flow
- main: Versões estáveis.
- develop: Desenvolvimento contínuo.
- feature: Branches para novas funcionalidades.

9. Recuperação de Erros
Recuperar Branch Deletada
git reflog # Encontre o hash do último commit da branch
git checkout -b branch-name <hash>

Alterar Último Commit
git commit --amend # Edita mensagem ou adiciona arquivos esquecidos

10. Recursos Adicionais
- Livro Oficial: https://git-scm.com/book/pt-br/v2 (gratuito).
- GitHub Learning Lab: Cursos interativos.
- Try Git: https://try.github.io.