LUIZA OLIVEIRA DE SOUZA GARCIA

MARCO ANTONIO MAIA

MANUAL DE USO DO GITHUB

Departamento de Ciência da Computação

GCC267 - Projeto Integrador

Prof. Rafael Serapilha Durelli

LAVRAS - MG

2025

O que é o Git e o GitHub?

O Git é um sistema de controle de versão distribuído que permite registrar o histórico de alterações em arquivos e possibilita o trabalho colaborativo entre desenvolvedores. Já o GitHub é uma plataforma que hospeda repositórios Git na nuvem e oferece recursos adicionais, como colaboração em equipe, revisão de código, integração contínua e organização de projetos.

Os principais objetivos do uso dessas ferramentas são rastrear as mudanças feitas no código, possibilitar o trabalho em equipe sem que um desenvolvedor sobrescreva o trabalho do outro, permitir a reversão para versões anteriores em caso de erros, facilitar processos de integração e deploy automatizado e, por fim, organizar o fluxo de desenvolvimento por meio do uso de branches e pull requests.

Conceitos Importantes

- **Repositório**: local onde o código e histórico ficam armazenados.
- **Commit**: registro de uma alteração feita no código.
- **Branch**: ramificação para desenvolver funcionalidades isoladas.
- **Merge**: junção de branches.
- **Pull Request (PR)**: solicitação de revisão/mesclagem de código.
- **Fork**: cópia de um repositório para sua conta.
- **Clone**: baixar o repositório do GitHub para sua máquina.
- **Remote**: conexão entre repositório local e o GitHub.

Principais comandos Git

Criar e conectar repositórios

`git init` → inicia um repositório local.

`git clone <url>` → clona do GitHub para a máquina.

`git remote add origin <url>` → conecta local ao GitHub.

Status e histórico

`git status` → mostra arquivos modificados.

`git log` → mostra histórico de commits.

`git diff` → exibe diferenças antes de commitar.

Controle de arquivos

`git add <arquivo>` → adiciona arquivo específico.

`git add .` → adiciona tudo.

`git commit -m "mensagem"` → cria um commit.

`git restore <arquivo>` → descarta alterações.

`git rm <arquivo>` → remove arquivo.

Trabalhando com branches

`git branch` → lista branches.

`git branch nova-branch` → cria branch.

`git checkout nova-branch` → troca de branch.

`git checkout -b nova-branch` → cria e muda para ela.

`git merge nome-branch` → mescla branch na atual.

Sincronização com o GitHub

`git push origin <branch>` → envia commits para o GitHub.

`git pull origin <branch>` → atualiza local com remoto.

`git fetch` → baixa alterações sem aplicar.

Correções

`git revert <id-commit>` → desfaz commit criando outro.

`git reset --hard <id-commit>` → volta a um commit anterior

`git stash` → guarda alterações sem commit.

`git stash pop` → recupera alterações guardadas.

Fluxo de Trabalho Recomendado

1. **Clonar ou criar repositório**

```
Shell
git clone <url>
```

2. **Criar branch para nova funcionalidade**

```
Shell
git checkout -b minha-feature
```

3. **Editar arquivos e salvar**

4. **Adicionar alterações**

```
Shell
git add .
```

5. **Fazer commit**

```
Shell
git commit -m "Descrição clara da mudança"
```

6. **Enviar para GitHub**

```shell
git push origin minha-feature
```

7. **Abrir Pull Request no GitHub** → solicitar revisão.
8. **Merge** após aprovação.
9. **Atualizar branch principal**

```shell
git checkout main

git pull origin main
```

Boas Práticas

- Commits pequenos e bem descritos.
- Sempre criar branches para novas features.
- Atualizar a branch `main` com frequência.
- Revisar código via Pull Request antes do merge.
- Usar `.gitignore` para evitar arquivos desnecessários no repositório.

Padrões de Commit

Manter commits organizados facilita o entendimento do histórico e a colaboração entre desenvolvedores.

Boas práticas:

- Mensagens **claras e objetivas**.
- Usar **tempo verbal no imperativo**: *"adiciona"*, *"corrige"*, *"remove"*.
- Commits pequenos e com **uma única responsabilidade**.
- Evitar mensagens genéricas como *“update”* ou *“fix”*.

Estrutura recomendada (Conventional Commits):

```
None
<tipo>(escopo opcional): descrição breve
```

Tipos comuns:

- *feat*: → nova funcionalidade.
- *fix*: → correção de bug.
- *docs*: → documentação.
- *style*: → mudanças de formatação (espaços, vírgulas, indentação).
- *refactor*: → refatoração de código sem mudar comportamento.
- *test*: → adição ou alteração de testes.
- *chore*: → tarefas diversas (configs, dependências, build).

Exemplos:

feat(login): adiciona validação de senha

fix(api): corrige erro de autenticação

docs(readme): atualiza instruções de instalação

refactor(user-service): simplifica método de busca