

Documento Explicativo: Git

Parte 1: Git

O que é o Git?

O **Git** é um sistema de controle de versão distribuído. Isso significa que ele permite que você acompanhe as mudanças no código ao longo do tempo e colabore com outras pessoas de forma segura. Com o Git, é possível experimentar novas ideias sem comprometer a versão estável do seu projeto, além de ser uma ferramenta essencial no mundo do desenvolvimento de software.

1. Download e Instalação do Git

Antes de qualquer coisa, é necessário instalar o Git em sua máquina.

Por que instalar o Git?

Para usar os comandos de versionamento, salvar versões do seu código e se comunicar com repositórios remotos, o Git precisa estar instalado localmente.

Como instalar?

Windows:


1. Acesse: <https://git-scm.com>
2. Baixe a versão para Windows.
3. Siga as etapas da instalação (recomenda-se deixar as opções padrão).

Linux (Ubuntu/Debian):

```
1 sudo apt update
2 sudo apt install git
3
```

macOS:

```
1 brew install git
2
```

 **Dica:** Após instalar, abra um terminal e digite `git --version` para verificar se tudo correu bem.

2. Configuração Inicial do Git

Depois de instalado, precisamos configurar o nome e e-mail que serão usados nos commits:

```
1 git config --global user.name "Seu Nome"
2 git config --global user.email "seu@email.com"
3
```

Por que isso é importante?

Essas informações identificam o autor de cada mudança feita no projeto.

Para verificar se a configuração deu certo:

```
1 git config --list
2
```

💡 **Dica:** Use a opção `--global` apenas uma vez para configurações padrão. Você também pode fazer configurações específicas para cada projeto, omitindo `--global`.

3. Inicializando o Primeiro Repositório [↗](#)

O repositório é onde o Git armazena todas as informações sobre o histórico do seu projeto.

```
1 cd pasta-do-projeto
2 git init
3
```

Esse comando cria uma pasta oculta chamada `.git`, responsável por monitorar tudo que acontece dentro do projeto.

🔴 **Importante:** A partir desse momento, o Git começa a observar seu projeto, mas ainda não está rastreando arquivos.

4. Rastreando Arquivos (Tracking) [↗](#)

Para que o Git acompanhe um arquivo, ele precisa ser adicionado ao *staging area*:

```
1 git add nome-do-arquivo.extensao
2
```

Para adicionar todos os arquivos:

```
1 git add .
2
```

Use `git status` para visualizar quais arquivos foram modificados ou adicionados.

💡 **Dica:** Use `git status` com frequência para entender o que está acontecendo no seu projeto.

5. Realizando o Primeiro Commit [↗](#)

O commit salva as alterações da área de staging no histórico do projeto:

```
1 git commit -m "mensagem explicando o que foi feito"
2
```

Por que fazer commits? [↗](#)

Os commits permitem que você salve versões do seu projeto e volte a elas caso algo dê errado.

💡 **Dica:** Escreva mensagens claras, como "adiciona cabeçalho no HTML" ao invés de "alterações gerais".

6. Alterações, Renomeações e Restaurações [↗](#)

Alterar um arquivo:

Após modificar um arquivo, use `git add` novamente para registrar a nova versão e `git commit` para salvar.

Renomear arquivos:

```
1 git mv antigo.txt novo.txt
2
```

Remover arquivos:

```
1 git rm arquivo.txt
2
```

Restaurar arquivos modificados:

```
1 git restore arquivo.txt
2
```

Restaurar arquivos do staging:

```
1 git restore --staged arquivo.txt
2
```

⚠ **Atenção:** Restaurar arquivos pode sobrescrever alterações feitas localmente.

7. Pulando o Staging (Commit Direto) [🔗](#)

Se quiser pular a etapa de `git add`, use:

```
1 git commit -am "mensagem"
2
```

🔴 Funciona apenas com arquivos que já foram adicionados anteriormente ao Git.

8. Corrigindo Mensagem com Amend [🔗](#)

Errou na mensagem do último commit?

```
1 git commit --amend -m "nova mensagem"
2
```

⚠ **Cuidado:** Não use `amend` se o commit já foi enviado para o repositório remoto (pode causar conflitos).

9. Visualizando o Histórico (Log) [🔗](#)

Para ver os commits feitos:

```
1 git log
2
```

Formatos alternativos:

```
1 git log --oneline
2
```

```
1 git log --stat
2
```

💡 Use `--oneline` para um resumo rápido e `--graph` para visualizar o fluxo de branches.

10. Git Reset (Desfazendo Commits) [🔗](#)

Desfaz o último commit:

```
1 git reset HEAD~1
2
```

Esse comando volta o commit, mas mantém as alterações feitas nos arquivos.

🔥 Ideal para quando você esqueceu de incluir um arquivo ou cometeu cedo demais.

11. Reset Hard (Desfaz Totalmente) [🔗](#)

Apaga o commit e as mudanças nos arquivos:

```
1 git reset --hard HEAD~1
2
```

⚠️ **Irreversível!** Use com extremo cuidado. Ideal apenas em ambientes locais.

12. Git Alias (Atalhos para Comandos) [🔗](#)

Facilite sua vida criando comandos personalizados:

```
1 git config --global alias.st status
2
```

Agora `git st` executa `git status`.

💡 Crie atalhos para comandos que você mais usa: `co` para `checkout`, `ci` para `commit`, `br` para `branch`, etc.

13. O que são Branches? [🔗](#)

Uma **branch** (ramo) representa uma linha paralela de desenvolvimento. Usar branches permite testar novas funcionalidades sem afetar o código principal (geralmente na branch `main` ou `master`).

💡 **Dica:** Use branches para desenvolver features, corrigir bugs e revisar códigos.

14. Criando a Branch AddMenu [🔗](#)

```
1 git checkout -b AddMenu
2
```

Esse comando **cria e já muda para** a nova branch.

🔗 Nomeie suas branches com algo descritivo, como `feature/login` ou `bugfix/menu`.

15. Removendo a Branch `AddMenu` [🔗](#)

Volte para a branch principal antes de remover:

```
1 git checkout main
2
```

Remova a branch:

```
1 git branch -d AddMenu
2
```

Se a branch não tiver sido mesclada:

```
1 git branch -D AddMenu
2
```

💡 Sempre verifique se a branch foi integrada antes de deletá-la para não perder trabalho.
