Git

📖 Sumário

O que é Git?

O **Git** é um **sistema de controle de versão**.

Pense nele como um "Salvar" superpoderoso para seu código. Em vez de criar arquivos como `main_v1.cpp`, `main_v2.cpp` ou `main_final_agora_vai.cpp`, o Git gerencia o histórico para você.

Ele permite que você salve "instantâneos" (chamados **commits**) do seu projeto. Se algo quebrar, você pode facilmente "voltar no tempo" para uma versão que funcionava.

Por que usar?

- **Histórico:** Você tem um registro completo de *quem* mudou *o quê* e *por quê*.
- **Colaboração:** É a ferramenta que permite que dezenas de pessoas trabalhem no mesmo projeto sem sobrescrever o trabalho umas das outras.
- **Segurança:** Ao usá-lo com um serviço como **GitHub** ou **GitLab**, você tem um backup do seu código na nuvem.

Instalando o Git

Antes de usar qualquer comando, você precisa ter o Git instalado no seu computador.

Em Windows

1. Baixe o instalador oficial do **Git for Windows** no site: https://git-scm.com/download/win
2. Execute o instalador. Durante a instalação, você pode deixar as opções padrão, elas são seguras e funcionais.
3. **Importante:** A instalação incluirá o **Git Bash**, um terminal que recomendamos usar, pois ele entende tanto os comandos Git quanto os comandos Linux (como `ls`, `cd`, `rm`).

Em macOS

A forma mais fácil é instalar as Ferramentas de Linha de Comando do Xcode, que já incluem o Git.

1. Abra o Terminal (em `Aplicativos/Utilitários`).
2. Digite o comando:

```
xcode-select --install
```

- Como alternativa, se você usa o Homebrew, você pode simplesmente rodar:

```
brew install git
```

Em Linux (Debian/Ubuntu)

Abra seu terminal e use o gerenciador de pacotes:

```
sudo apt update
sudo apt install git
```

Verificando a Instalação

Após instalar, abra um **novo** terminal e digite:

```
git --version
```

Se o terminal responder com um número de versão (ex: `git version 2.40.1`), significa que o Git foi instalado com sucesso.

⚙️ Os Comandos Essenciais para Iniciar

Vamos ver o fluxo de trabalho mais básico.

1. Configuração Inicial (Faça isso só uma vez)

Antes de tudo, você precisa dizer ao Git quem você é. Isso será usado para assinar todos os "saves" (commits) que você fizer.

Abra seu terminal e digite:

```
# Configura seu nome de usuário
git config --global user.name "Seu Nome"

# Configura seu e-mail (o mesmo que você usa no GitHub/GitLab)
git config --global user.email "seu@email.com"
```

2. Iniciando um Repositório (Duas Formas)

Você pode começar um projeto de duas maneiras: criando um novo do zero ou copiando um existente.

Opção A: Começar um projeto novo (`git init`)

Se você já tem uma pasta de projeto (ex: `C:\Projetos\MeuApp`) e quer começar a rastreá-la com o Git:

```
# Navegue até a pasta do seu projeto
cd C:\Projetos\MeuApp

# Diga ao Git para começar a rastrear esta pasta
git init
```

Isso criará uma subpasta oculta chamada `.git` que guardará todo o histórico.

Opção B: Copiar um projeto existente (`git clone`)

Se o projeto já existe (ex: no GitHub), você pode cloná-lo (baixar uma cópia).

```
# Vá para a pasta onde você guarda seus projetos
cd C:\Projetos

# Clone o projeto (a URL você pega no site do GitHub/GitLab)
git clone https://github.com/usuario/nome-do-projeto.git

# Isso já cria a pasta "nome-do-projeto" para você
cd nome-do-projeto
```

3. O Fluxo de Trabalho Básico (O loop diário)

Este é o ciclo que você fará dezenas de vezes por dia. O Git tem **três "áreas"**:

1. **Working Directory:** A sua pasta, onde você edita os arquivos.
2. **Staging Area (Área de Preparação):** Onde você coloca os arquivos que *quer* salvar no próximo commit.
3. **Repository (Histórico):** Onde os "snapshots" salvos (commits) ficam guardados.

O fluxo é sempre: **Modificar -> Adicionar (Stage) -> Commitar (Salvar)**.

Passo 1: `git status`

Sempre comece com este comando. Ele é seu melhor amigo e lhe diz o que está acontecendo.

```
# Mostra quais arquivos foram modificados, quais estão na Staging Area, etc.
git status
```

- Arquivos em **vermelho** (`Untracked` ou `Modified`) estão no seu Working Directory.
- Arquivos em **verde** (`Staged`) estão na Staging Area, prontos para o commit.

Passo 2: `git add`

Quando você terminar de editar um arquivo e estiver pronto para salvá-lo no histórico, você o move para a Staging Area.

```
# Adiciona um arquivo específico
git add main.cpp

# OU (mais comum)
# Adiciona TODOS os arquivos modificados na pasta atual
git add .
```

> Se você rodar `git status` de novo, verá que os arquivos agora estão em verde.

Passo 3: `git commit`

Agora que os arquivos estão preparados, você "tira a foto" e salva o snapshot no histórico.

```
# O -m significa "message" (mensagem)
git commit -m "O que eu fiz neste commit (ex: Adicionei a função main)"
```

> **Boas práticas:** Suas mensagens de commit devem ser curtas e claras. Elas explicam *o que* você mudou.

4. Sincronizando com a Nuvem (GitHub/GitLab)

Até agora, seus commits (saves) estão **apenas no seu computador local**. Para colaborar ou fazer backup, você envia (push) seus commits para um servidor remoto (como o GitHub).

Passo 1: `git remote` (Só precisa fazer uma vez por projeto)

Se você usou `git init`, você precisa "conectar" seu repositório local a um repositório na nuvem.

```
# Crie um repositório vazio no GitHub primeiro
# O GitHub lhe dará esta URL. 'origin' é o apelido padrão para o servidor.
git remote add origin https://github.com/usuario/nome-do-projeto.git
```

(Se você usou `git clone`, isso já foi feito para você.)

Passo 2: `git push`

Isso envia seus commits locais para o servidor remoto (`origin`).

```
# A primeira vez que você envia, você precisa usar -u
# 'main' é o nome da branch (ramo) principal. Pode ser 'master' em projetos
antigos.
```

```
git push -u origin main

# Nas próximas vezes, você pode usar apenas:
git push
```

---

5. O Fluxo de Trabalho em Equipe (O Essencial)

Quando você está trabalhando em equipe, duas regras são mais importantes:

1. **Sempre** atualize seu repositório local antes de começar a trabalhar.
2. **Nunca** trabalhe diretamente na branch `main`.

Passo 1: `git pull` (Atualizando-se)

Antes de começar a programar, você deve "puxar" (pull) quaisquer mudanças que seus colegas de equipe enviaram (push) para o servidor.

```
# Baixa as atualizações do servidor (origin) e mescla
# com o seu código local.
git pull origin main
```

> **Dica:** Faça disso um hábito. Sempre rode `git pull` antes de começar a trabalhar.

Passo 2: `git branch` (Trabalhando em Isolamento)

A branch `main` é a versão "oficial" e estável do código. Você nunca deve fazer commits diretamente nela. Em vez disso, você cria uma "ramificação" (branch) para trabalhar na *sua* funcionalidade.

```
# Cria uma nova branch e já muda para ela
# (ex: 'feature/add-strobe-light')
git checkout -b nome-da-sua-branch

# Exemplo:
git checkout -b treinamento/joao-teste-mosfet
```

> Agora você está em uma "cópia" segura do código. Você pode fazer `git add` e `git commit` à vontade aqui, sem quebrar o `main`.

Passo 3: `git push` (Enviando sua Branch)

Quando você terminar de trabalhar na sua branch, você faz o `push` dela para o servidor.

```
# Envia a SUA branch para o servidor (origin)
git push origin nome-da-sua-branch
```

> Depois de fazer isso, você pode ir ao site do GitHub e abrir um **"Pull Request" (PR)**, que é o pedido formal para "mesclar" o seu código da sua branch de volta para a `main`.

Tabela Rápida: Resumo dos Comandos

Comando	O que faz (em poucas palavras)
`git config`	Configura seu nome/email (só uma vez).
`git init`	Inicia um repositório novo na pasta atual.
`git clone [url]`	Baixa/copia um repositório da internet.
`git status`	Mostra o estado dos seus arquivos (o mais usado).
`git add [arquivo]`	Prepara um arquivo para o "save" (commit).
`git commit -m "msg"`	Salva o "snapshot" (o save) no histórico local.
`git push`	Envia seus commits locais para o servidor (GitHub).
`git pull`	Baixa as atualizações do servidor (o oposto de `push`).
