



GIT E GITHUB



Guia completo para
desenvolvedores







Introdução

No mundo do desenvolvimento, saber programar é o ponto de partida — não o diferencial. O verdadeiro destaque vem da capacidade de colaborar, organizar seu código e evoluir em equipe. E é exatamente aqui que **Git** e **GitHub** entram como protagonistas.



Sobre o autor

Minha jornada com a tecnologia começou aos 13 anos, quando tive minha primeira oportunidade em uma Lan House. Desde então, entendi que resolver problemas reais com soluções digitais não é só uma habilidade — é um propósito que transformou minha vida e guia minha carreira até hoje.

Sou formado em Ciências da Computação e pós-graduado em Desenvolvimento Web e Inteligência Artificial. Atuo como Analista de Desenvolvimento de Software e como Professor de Desenvolvimento de Sistemas.

Hoje, foco no desenvolvimento Full Stack, criando aplicações escaláveis e intuitivas com React, Next.js e Node.js, sempre buscando entregar soluções eficientes, modernas e com excelente experiência de uso.

Além de desenvolver, também crio conteúdo para ajudar outros devs a evoluírem de forma estruturada. Ofereço cursos, ebooks, mentorias e dicas práticas que facilitam o aprendizado, o direcionamento e a entrada no mercado de tecnologia.

Sumário

- Por que dominar Git e GitHub transforma sua carreira?
- Git e GitHub
- Instalação e configuração: Git + GitHub + VSCode
- Comandos essenciais
- Branches, merge e resolução de conflitos
- GitHub na prática: forks, pull requests e issues
- Boas práticas, padrões e fluxos de trabalho
- Casos de uso reais e dicas avançadas
- Glossário de termos técnicos

Por que dominar Git e GitHub transforma sua carreira?

Ao longo da sua **jornada como dev**, você vai escrever milhares de linhas de código. Sem **versionamento**, esse processo se torna arriscado, caótico e limitado. Com **Git** e **GitHub**, você ganha:

-  **Histórico completo do seu projeto — cada passo registrado**
-  **Segurança para experimentar sem medo**
-  **Colaboração profissional com revisões, sugestões e integração**
-  **Autonomia para contribuir com projetos open source do mundo**

Git e GitHub

O que é Git?



Git é um sistema de **controle de versão distribuído**, criado por Linus Torvalds em 2005. Ele permite que desenvolvedores acompanhem, compartilhem e revertam qualquer mudança no código com segurança e agilidade.

Característica	O que significa na prática
Distribuído	Cada dev tem uma cópia completa do repositório
Rápido	Operações como commit, log e branch são instantâneas
Seguro	Cada alteração gera um hash único (SHA-1)
Flexível	Suporta múltiplos fluxos e estratégias de trabalho

O que é GitHub?



GitHub é uma [plataforma de hospedagem](#) e [colaboração baseada em Git](#). Nele você pode:

- Criar e manter repositórios online
- Trabalhar em equipe com Pull Requests
- Organizar tarefas com Issues
- Automatizar fluxos com Actions
- Conectar com devs e empresas



Dica Prática:

Seu GitHub é seu [portfólio vivo](#). Cada [repositório](#) bem [documentado](#) e [organizado](#) pode abrir portas no mercado.

Instalação e configuração: Git + GitHub + VSCode

Git instalação



Antes de dominar os comandos, é preciso garantir que seu ambiente esteja configurado corretamente. Aqui está um guia direto e confiável para você começar com o pé direito.

Windows

- Acesse: <https://git-scm.com>
- Baixe a versão para Windows
- Siga os passos padrão de instalação

Linux (Ubuntu/Debian)



```
sudo apt update  
sudo apt install git  
git --version
```

MacOS (via Homebrew)



```
brew install git  
git --version
```

GitHub configuração



Criando e configurando sua conta no GitHub:

- Acesse o site: <https://github.com/signup>
- Crie sua conta
- Personalize seu perfil
- Use o mesmo e-mail do GitHub na sua configuração global do Git

Git configuração

No terminal configure seu nome e e-mail globalmente para identificar seus commits:

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

Visualize suas configurações com:

```
git config --list
```

Integração com o VSCode

O VSCode é uma excelente [IDE](#) para usar com [Git](#).

Faça assim:

- Instale a extensão [GitLens](#) para visualização poderosa do histórico
- Use o [terminal integrado do VSCode](#) para executar comandos Git
- No menu lateral esquerdo, use os ícones do [GitLens](#) ou do [Source Control](#)



Dica do Profissional:

Configure atalhos no VSCode para abrir o terminal, alternar entre branches e visualizar o diff de arquivos.

Comandos essenciais

Comandos iniciais

Comando	O que faz	Quando usar
git init	Inicia um repositório Git vazio	Ao começar um novo projeto local
git clone URL	Clona repositório remoto	Para trabalhar com um projeto existente
git config --global	Define configurações globais	Primeira vez usando Git

Controle de versão

Comando	O que faz	Quando usar
git add .	Move arquivos para a staging area	Antes de salvar mudanças
git commit -m "mensagem"	Registra as mudanças com uma mensagem	Sempre que concluir uma etapa relevante
git status	Mostra status dos arquivos	Para ver arquivos modificados ou pendentes
git log	Lista o histórico de commits	Para revisar o progresso do projeto

Branches e navegação

Comando	O que faz	Quando usar
git branch nome	Cria uma nova branch	Para desenvolver uma nova funcionalidade
git checkout nome	Troca para outra branch	Para alternar entre contextos
git switch nome	Alternativa moderna ao checkout	Recomendado para maior clareza
git merge nome	Une branch atual com outra	Ao finalizar o desenvolvimento e integrar

Repositórios remotos

Comando	O que faz	Quando usar
<code>git remote add origin URL</code>	Define o repositório remoto	Depois de iniciar um repositório local
<code>git push origin branch</code>	Envia suas alterações para o GitHub	Após commit, para sincronizar com a equipe
<code>git pull origin branch</code>	Atualiza seu projeto local com mudanças remotas	Antes de começar a trabalhar no código
<code>git fetch</code>	Busca mudanças do remoto sem aplicar	Para revisão manual de mudanças

Correções e ajustes

Comando	O que faz	Quando usar
git reset	Desfaz commits ou staging	Para reverter rapidamente uma ação
git revert hash	Cria um commit "inverso" para desfazer	Para desfazer sem apagar o histórico
git stash	Salva alterações temporárias	Para trocar de branch sem perder trabalho

Branches, merge e **resolução de conflitos**

O Que é uma branch?

Uma **branch** (ramificação) permite que você desenvolva uma nova funcionalidade ou experimente mudanças sem afetar o código principal.

Imagine uma **cópia paralela do seu projeto**, onde você pode criar livremente, e depois — se tudo estiver certo — unir com o trabalho principal.

Criando e alternando branches

```
# Cria uma nova branch  
git branch nome-da-branch  
# Alterna para essa branch  
git checkout nome-da-branch  
# Cria e já muda (mais moderno)  
git switch -c nome-da-branch
```



Dica Prática:

Nomeie suas branches com clareza:
`feature/login`, `hotfix/erro-login`,
`refactor/menu` etc.

Fazendo merge: unindo as branches

Após desenvolver, o **fluxo** típico é:

```
# Volta para a branch principal  
git checkout main  
# Une a branch ao main  
git merge nome-da-branch
```

O **Git** tentará **unir automaticamente as alterações**.
Mas quando dois devs alteram a mesma linha do
mesmo arquivo, acontece um **conflito**...



Dica Prática:

Evite conflitos grandes com
commits pequenos e frequentes.

Conflito! (como resolver)

Quando há **conflito**, o **Git** marca assim:

```
...  
<<<<< HEAD  
Seu código aqui  
=====  
Código da outra branch  
>>>>> feature/nova-funcionalidade
```

Passo a passo para **resolver**:

- Edite o arquivo, decidindo qual versão manter (ou combine ambas)
- Salve e use:

```
...  
git add nome-do-arquivo  
git commit -m "Conflito resolvido"
```

GitHub na prática: **forks, pull requests e issues**

Fork: o seu clone autorizado

O [fork](#) cria uma [cópia](#) pública de um repositório no seu GitHub. Isso permite:

- Estudar um projeto [sem afetar o original](#)
- Criar [alterações](#) e propor [melhorias](#)
- Contribuir com projetos open source bash



```
# Depois de dar fork via GitHub:  
git clone  
https://github.com/seuperfil/repositorio-forkado.git
```

Pull request: propondo mudanças com qualidade

Após alterar seu fork, você pode pedir para o projeto original aceitar suas mudanças com um Pull Request (PR).

Fluxo Clássico para PR:

- Faça fork
- Clone o repositório
- Crie uma nova branch
- Faça commits com clareza
- Dê push para o seu GitHub
- Vá até o repositório original e clique em “Compare & Pull Request”

Checklist para um bom PR

- Nome claro da branch
- Commits limpos e sem lixo
- Mensagem explicando o “porquê” das mudanças
- Relacione Issues: Resolves #42

Issues: gestão de bugs, tarefas e ideias

Issues são tickets usados para:

- Reportar bugs
- Sugerir funcionalidades
- Acompanhar progresso de tarefas

Você pode atribuir, comentar, vincular PRs e usar labels como bug, enhancement, documentation.

Boas práticas, padrões e fluxos de trabalho

Commits semânticos: nomeando como um profissional

A forma como você escreve seus commits comunica mais que o código. Commits semânticos tornam o histórico claro, rastreável e fácil de revisar.

Padrão recomendado:



<tipo>: descrição clara da mudança

Tipos comuns

Tipo	Uso
feat	Adição de nova funcionalidade
fix	Correção de bug
docs	Alterações na documentação
style	Ajustes de formatação, sem impacto no código
refactor	Refatorações sem mudança de funcionalidade
test	Adição ou alteração de testes
chore	Tarefas menores, sem relação com produção (ex: atualização de dependência)



Dica Prática:

Exemplo:

feat(login): adiciona validação de email no formulário

Organização de branches

Manter uma estratégia de nomes padronizada facilita a colaboração.

Sugestão:

- `main`: versão de produção
- `dev`: ambiente de desenvolvimento
- `feature/nome`: nova funcionalidade
- `bugfix/nome`: correções específicas
- `hotfix/nome`: correções urgentes em produção

Casos de uso reais e dicas avançadas

Atualizando forks sem **perder suas alterações**

```
git remote add upstream  
https://github.com/original/repo.git
```

```
git fetch upstream
```

```
git merge upstream/main
```

Conflitos em equipe: simulação prática

Você e outro dev editam a mesma linha em
`branches` diferentes.

Quando fizer `merge`, o `Git` sinaliza o
conflito.

Como evitar?

- Faça pull regularmente antes de começar
- Comunique mudanças significativas
- Prefira branches pequenas e focadas

Hooks: automatizando tarefas com Git

Git permite automatizar scripts que executam antes ou depois de ações como `commit`, `merge` ou `push`.

Exemplo: `pre-commit`

```
● ● ●  
# .git/hooks/pre-commit (arquivo bash)  
#!/bin/sh  
npm run lint
```



Dica Prática:

Os hooks são locais e não são enviados via push por padrão.

Glossário de termos técnicos

Termo	Definição Rápida
Branch	Ramificação do código, usada para trabalhar em paralelo
Commit	Registro de uma alteração no projeto
Merge	União entre duas branches
Fork	Cópia pública de um repositório no seu GitHub pessoal
Pull Request	Solicitação para integrar mudanças de um branch/fork ao projeto principal
Upstream	Repositório original de onde um fork foi feito
Origin	Repositório remoto principal (o seu GitHub)
HEAD	Referência ao commit atual
Rebase	Reorganização dos commits para um histórico linear
Tag	Marcador para sinalizar versões específicas
Hook	Script automático executado por eventos do Git (ex: antes de um commit)
Stash	Armazena temporariamente alterações não commitadas
CI/CD	Integração Contínua / Entrega Contínua — automação de testes, builds e deploys

Fim da linha?

Chegamos ao fim — pelo menos por aqui. [Agora é com você.](#)

Quero te parabenizar por ter chegado até aqui. [Investir no seu aprendizado](#) já é um grande passo, e você deu esse passo com consistência.

Escrevi este [Ebook](#) com atenção e dedicação, pensando em como facilitar sua jornada com [Git](#) e [GitHub](#).

Dominar essas ferramentas vai além dos comandos: é adotar uma [mentalidade de versionamento](#), colaboração e evolução contínua no código.

Se quiser trocar ideia, tirar dúvidas ou acompanhar novos conteúdos, estou nas redes sociais: [@eeymatheusdev](#). Nos vemos por aí.

Matheus Willian Bento



