# GIT GUIA PARA INICIANTES >_

Commits, branches e fluxo de trabalho — passo a passo, com exemplos

**Prof. Gustavo Franz (Science/Biology)**
*Python Developer in Progress*
*Building Educational Solutions with Python*

**GitHub:** github.com/GustaFranz

## O que é Git?

Git é um **sistema de controle de versão**. Ele registra o histórico do seu projeto, permite trabalhar em equipe, experimentar sem medo e voltar atrás quando necessário.

- ✓ Salva o histórico do projeto (commits).
- ✓ Permite trabalhar em várias linhas (branches).
- ✓ Facilita colaboração e revisão de código.
- ✓ Protege contra perda de código.
- ✓ Dá liberdade para testar sem medo de errar.

## O fluxo básico do Git

**Arquivo alterado** -> (git add) -> **Stage** -> (git commit) -> **Repositório local** -> (git push) -> **GitHub (remoto)**.

- ✓ Você altera arquivos no projeto.
- ✓ **git add** coloca no stage (você escolhe o que será salvo).
- ✓ **git commit** salva no histórico.
- ✓ **git push** envia para o GitHub.

## 1. Stage (área de preparação)

O stage é uma área intermediária entre suas alterações e o commit. Você escolhe exatamente o que entra no próximo commit.

| Comando | O que faz | Exemplo |
|---|---|---|
| `git add arquivo.py` | Coloca um arquivo específico no stage. | `git add app.py` |
| `git add .` | Coloca todos os arquivos no stage. | `git add .` |
| `git status` | Mostra o que mudou e o que está no stage. | `git status` |
| `git restore --staged x` | Tira do stage sem apagar as alterações. | `git restore --staged app.py` |

**Dica**

Sempre use git status antes de commitar. Ele mostra exatamente o que está pronto para ser salvo.

## 2. Commits (salvando alterações)

O commit é um 'ponto de salvamento' no histórico. Cada commit deve ter uma mensagem clara sobre o que foi feito.

| Comando | O que faz | Exemplo |
|---|---|---|
| `git commit -m "msg"` | Salva as mudanças do stage no histórico. | `git commit -m "cria login"` |
| `git log` | Mostra o histórico de commits. | `git log` |
| `git log --oneline` | Mostra o histórico resumido (1 linha). | `git log --oneline` |
| `git diff` | Mostra o que foi alterado e ainda não adicionado. | `git diff` |

**Exemplo de histórico (git log --oneline)**

```
1ab32cd  cria sistema de login
f4d9ab6  corrige botão de acesso
98f736e  ajusta layout da tela inicial
```

## 3. Branches (linhas de trabalho)

Branch é uma linha paralela de desenvolvimento. Permite criar funcionalidades sem mexer na branch principal (geralmente main).

| Comando | O que faz | Exemplo |
|---|---|---|
| `git branch` | Lista todas as branches. | `git branch` |
| `git branch nome` | Cria uma nova branch. | `git branch tela-login` |
| `git checkout nome` | Muda para outra branch. | `git checkout tela-login` |
| `git checkout -b nome` | Cria e já muda para a nova branch. | `git checkout -b pagamento` |
| `git switch nome` | Muda de branch (alternativa ao checkout). | `git switch main` |
| `git switch -c nome` | Cria e já muda (alternativa). | `git switch -c nova-feature` |

## 4. Mesclando (merge)

Quando a funcionalidade da branch estiver pronta, juntamos (merge) na branch principal.

| Comando | O que faz | Exemplo |
|---|---|---|
| `git checkout main` | Vai para a branch principal. | `git checkout main` |
| `git merge nome` | Mescla a branch informada na atual. | `git merge tela-login` |
| `git push` | Envia as mudanças para o GitHub. | `git push` |

**Dica**

Sempre teste a branch antes do merge. Depois, você pode excluir a branch que não precisa mais.

## 5. Envio e atualização (GitHub)

No comando **git push -u origin NOME_DA_BRANCH**, o último termo (*feature*, *main*, etc.) é o **nome da branch** que vai receber o push — não é um comando fixo. Antes de enviar, rode **git status**: a linha *On branch ...* mostra em qual branch você está. No **primeiro push** de um repo novo, essa branch costuma ser **main** (não feature).

| Comando | O que faz | Exemplo |
|---|---|---|
| `git push -u origin NOME` | Envia a branch pela 1ª vez e vincula ao remoto. | `git push -u origin main` |
| `git push -u origin NOME` | Mesmo comando em branch de funcionalidade. | `git push -u origin feature-login` |
| `git push` | Envia as mudanças da branch atual (já vinculada). | `git push` |
| `git pull` | Baixa e aplica as alterações do GitHub. | `git pull` |
| `git fetch` | Baixa as alterações sem aplicar (só informa). | `git fetch` |

**Conferir a branch antes do primeiro push**

```
git status  # On branch main <- use este nome no push
git push -u origin main  # primeiro push (repo novo)
# git push -u origin feature-login  # só se você ESTIVER nessa branch
```

**Dica**

Sempre dê git pull antes de começar a trabalhar, para evitar conflitos. E use git status antes do push para não enviar para a branch errada.

## 6. Fluxo de trabalho recomendado

Siga esta ordem sempre que for criar uma nova funcionalidade:

**Ordem dos comandos**

```
git pull  # atualiza a branch principal
git checkout -b nova-funcionalidade  # cria e entra na branch
# ... faça suas alterações no código ...
git add .  # coloca tudo no stage
git commit -m "mensagem clara"  # salva no histórico
git push -u origin nova-funcionalidade  # envia para o GitHub
# abra um Pull Request, faça o merge na main e exclua a branch
```

## Conceitos importantes

| Termo | Significado |
|---|---|
| `Repositório` | Projeto onde o código é gerenciado. |
| `Commit` | Salvamento de um ponto da alteração. |
| `Branch` | Linha paralela de desenvolvimento. |

| Termo | Significado |
|---|---|
| `Merge` | Junção de uma branch em outra. |
| `Stage` | Área de preparação do commit (git add). |
| `Push` | Enviar alterações para o GitHub. |
| `Pull` | Baixar alterações do GitHub. |
| `Remote / Remoto` | Repositório na nuvem (ex.: GitHub). |
| `Origin` | Apelido padrão da conexão com o remoto. |
| `CLI` | Interface de linha de comando (digitar no terminal). |
| `gh (GitHub CLI)` | Programa para usar o GitHub pelo terminal. |

## Comandos de emergência (use com cuidado)

| Comando | O que faz | Exemplo |
|---|---|---|
| `git restore arquivo` | Descarta alterações (volta ao último commit). | `git restore app.py` |
| `git reset --soft HEAD~1` | Desfaz o último commit, mantendo as alterações. | `git reset --soft HEAD~1` |
| `git reset --hard HEAD~1` | Desfaz o commit e APAGA as alterações (perigoso). | `git reset --hard HEAD~1` |

## Como Criar e Publicar um Repositório no GitHub pelo Terminal

**Git e GitHub na Prática: Criação de Repositório e Primeiro Push**

Até aqui você viu comandos do **Git** (controle de versão na sua máquina). Agora vamos publicar o projeto no **GitHub** (site que hospeda repositórios na nuvem) usando o **terminal**, sem abrir o navegador para criar o repo.

Para isso usamos o **GitHub CLI** — programa chamado **gh** que conversa com o GitHub por comandos. **CLI** significa *Command Line Interface*: interface de linha de comando, ou seja, digitar comandos em vez de clicar em botões.

## 1. Verificar a instalação do GitHub CLI

O **gh** é um programa separado do Git. Você precisa instalá-lo uma vez (site: cli.github.com). Depois, confira se está disponível:

```
gh --version
```

Se aparecer a versão (ex.: *gh version 2.x*), está instalado. Se der erro *command not found*, instale o GitHub CLI antes de continuar.

## 2. Verificar a autenticação no GitHub

Autenticação é o processo de provar ao GitHub quem você é (login). Sem isso, o **gh** não consegue criar repositórios em seu nome.

```
gh auth status
```

Se não estiver logado, rode **gh auth login** e siga as perguntas (GitHub.com, HTTPS, login pelo navegador). Ao final, repita **gh auth status** — deve mostrar sua conta conectada.

## 3. Inicializar o repositório Git local

**Repositório local** é a pasta do seu projeto com histórico Git apenas no computador. Entre na pasta do projeto e inicie o Git:

```
cd caminho/do/meu-projeto
git init  # cria a pasta oculta .git (histórico local)
```

Na primeira vez na máquina, configure nome e e-mail (Git usa isso nos commits):

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

## 4. Criar o repositório remoto pelo terminal

**Repositório remoto** é a cópia do projeto na nuvem (GitHub). Com o **gh**, você cria o repo no site sem sair do terminal:

```
gh repo create meu-projeto --public --source=. --remote=origin
```

- ✓ **meu-projeto** — nome do repositório no GitHub (pode ser outro).
- ✓ **--public** — repo visível para todos (use --private se quiser fechado).
- ✓ **--source=.** — usa a pasta atual como origem.
- ✓ **--remote=origin** — cadastra o GitHub com o apelido **origin**.

**O que é origin?**

**origin** é o nome padrão da conexão com o repositório remoto. Pense nele como um 'apelido' para a URL do GitHub. Depois você usa *git push origin ...* em vez de digitar a URL completa toda vez.

## 5. Verificar a conexão com o repositório remoto

Confirme se o Git 'enxerga' o GitHub corretamente:

```
git remote -v
```

A saída deve listar **origin** com URL do tipo *https://github.com/seu-usuario/meu-projeto.git* (fetch e push). Também confira a branch atual:

```
git status  # On branch main (ou master em repos antigos)
git branch  # lista branches locais; * marca a atual
```

## 6. Adicionar arquivos ao stage

Lembre: o **stage** é a área de preparação — você escolhe o que entrará no próximo commit. Para o primeiro envio, costuma-se adicionar tudo:

```
git status  # vê arquivos novos ou alterados (vermelho = fora do stage)
git add .  # coloca todos os arquivos da pasta no stage
git status  # verde = prontos para commit
```

**Dica**

Crie um arquivo .gitignore para não enviar pastas como .venv ou senhas acidentalmente. O Git só commita o que você adicionou com git add.

## 7. Criar o primeiro commit

O **commit** grava um 'checkpoint' no histórico local. Use uma mensagem curta e clara sobre o que este primeiro salvamento representa:

```
git commit -m "primeiro commit: estrutura inicial do projeto"
```

Se aparecer erro pedindo identidade, volte ao passo 3 e configure user.name e user.email.

## 8. Realizar o primeiro push

**Push** envia seus commits locais para o repositório remoto (GitHub). No primeiro envio, use **-u** (upstream) para vincular sua branch local à branch remota. O nome depois de **origin** deve ser o da sua branch atual — confira com **git status**. Em repos novos, quase sempre é **main**:

**Sequência completa (repo novo, branch main)**

```
git status  # confirme: On branch main
git push -u origin main
```

Depois do primeiro push com **-u**, basta **git push** na mesma branch. Se você estiver em outra branch (ex.: *feature-login*), o comando seria **git push -u origin feature-login** — sempre o nome que aparece no git status.

## 9. Erros comuns e como corrigir

| Erro / situação | Causa provável | Como corrigir |
|---|---|---|
| gh: command not found | GitHub CLI não instalado. | Instale em cli.github.com e teste gh --version. |
| not logged in | Conta GitHub não autenticada. | Rode gh auth login e depois gh auth status. |
| fatal: not a git repository | Pasta sem git init. | Entre na pasta certa e rode git init. |
| Please tell me who you are | Nome/e-mail não configurados. | git config --global user.name e user.email. |
| src refspec main does not match | Branch main não existe localmente. | git branch — use o nome real (main ou master). |
| failed to push some refs | Remoto tem commits que você não tem. | git pull --rebase origin main, depois git push. |
| repository already exists | Repo com mesmo nome no GitHub. | Escolha outro nome ou use gh repo create ... --push. |
| nothing to commit | Nenhum arquivo no stage. | git add . e git status antes do commit. |

**Resumo: do zero ao GitHub (copie e adapte)**

```
cd meu-projeto
git init
gh auth login  # se ainda não estiver logado
gh repo create meu-projeto --public --source=. --remote=origin
git add .
git commit -m "primeiro commit"
git status  # veja o nome da branch (geralmente main)
git push -u origin main
```

### Resumo rápido + boas práticas

- ✓ Comandos do dia a dia: status, add ., commit -m, push, pull.
- ✓ Faça commits pequenos e com mensagens claras e objetivas.
- ✓ Use uma branch para cada funcionalidade ou correção.
- ✓ Sempre teste antes do merge e mantenha a main estável.
- ✓ Pratique e erre sem medo: com o tempo, vira natural.

**Prof. Gustavo Franz (Science/Biology)**
*Python Developer in Progress*
*Building Educational Solutions with Python*

Professor de Ciencias e Biologia desde 2013 — atualmente estudando Programacao.

Eu crio estes resumos para organizar os assuntos e estudar de forma clara e objetiva. Estou compartilhando porque eles tambem podem ajudar outros desenvolvedores que estao comecando. Bons estudos — vamos aprender juntos!

**GitHub:** github.com/GustaFranz

**Exercicios Python:** github.com/GustaFranz/python_exercises
Para quem quiser ver a resolucao dos meus exercicios em Python.

**EasyAnsi:** github.com/GustaFranz/easyansi
Para quem quiser codificar personalizando com cores e estilos de forma mais facil, pratica e intuitiva.