Git Descomplicado

Do "Init" a ao "commit" sem dor de cabeça

Sumário

Capítulo 1: Introdução ao Git (A sua Máquina do Tempo Pessoal)

Olá, futuro mestre do Git!

Se você já passou pela situação de ter arquivos como `projeto_final.js`, `projeto_final_v2.js`, `projeto_final_AGORA_VAI.js`... você precisa do Git.

O que raios é o Git?

Pense no Git como um videogame que permite "salvar o jogo" (criar *checkpoints*) a qualquer momento no seu código. Fez algo que quebrou tudo? Sem problemas! Você pode simplesmente "carregar o jogo" de um ponto anterior onde tudo funcionava.

Em termos técnicos, o **Git é um Sistema de Controle de Versão (VCS)**. Ele tira "fotos" (snapshots) do seu projeto ao longo do tempo.

Por que isso é tão importante?

1. **Segurança:** Você nunca mais vai perder trabalho.
2. **Histórico:** Você sabe *quem* mudou *o quê* e *por quê*.
3. **Colaboração:** (Este é o pulo do gato para o futuro) Permite que várias pessoas trabalhem no mesmo projeto sem pisar no código umas das outras.

> *Nota rápida: Git vs. GitHub Muita gente confunde!*
>
> - *Git: É a ferramenta, o software que você instala no seu computador para controlar as versões.*
> - *GitHub (ou GitLab/Bitbucket): É um serviço online (um "hotel") para guardar seus repositórios Git na nuvem, facilitando o backup e a colaboração.*
>
> *Neste ebook, vamos focar 100% no Git, a ferramenta local.*

Capítulo 2: Comandos Básicos e Fluxo de Trabalho (A "Trindade Sagrada")

Para começar a usar o Git, você só precisa entender três estágios e três comandos principais.

Os 3 Estágios do Git (A Analogia da Mudança)

Imagine que você está se mudando. Seus arquivos passam por três fases:

1. **Working Directory (Seus arquivos bagunçados):** É a sua pasta de projeto. Você está editando, criando e deletando arquivos. É a "bagunça" no chão do seu quarto.
2. **Staging Area (A caixa aberta):** Quando você acha que um arquivo está pronto, você o coloca na "área de preparação" (Staging Area). É como colocar um item dentro da caixa de mudança.
3. **Repository (.git) (A caixa fechada e etiquetada):** Quando você está satisfeito com tudo que está na caixa (Staging Area), você "fecha a caixa", coloca uma etiqueta (uma mensagem) e a envia para o caminhão. Esse é o *commit*.

A Trindade Sagrada: `init`, `add`, `commit`

Vamos usar esses comandos para controlar nosso primeiro projeto.

Antes de Começar: Apresente-se ao Git (`git config`)

Antes de fazer seu primeiro "save" (o *commit*), você precisa dizer ao Git quem você é. Afinal, ele precisa etiquetar quem fez cada mudança no histórico.

Você só precisa fazer isso **uma vez** no seu computador. Usamos o comando `git config --global` para salvar seu nome e e-mail permanentemente.

Abra seu terminal e digite os dois comandos abaixo, substituindo pelos seus dados:

```
# Define o nome que aparecerá nos seus commits
git config --global user.name "Seu Nome Completo"

# Define o e-mail que aparecerá nos seus commits
git config --global user.email "seuemail@exemplo.com"
```

Importante: Use o mesmo e-mail que você pretende usar em plataformas como o GitHub ou GitLab. Isso ajuda a vincular seus commits ao seu perfil online.

Pronto! Agora o Git sabe quem é o autor de todas as mudanças que você fizer.

1. `git init` (Ligando a máquina)

Este comando "ativa" o Git em uma pasta. Ele cria um diretório oculto chamado `.git` que guardará todo o histórico. Você só faz isso **uma vez** por projeto.

```
# Navegue até a pasta do seu projeto
cd meu-projeto

# Ligue o Git
git init
```

O terminal dirá algo como: `Initialized empty Git repository in /caminho/para/meu-projeto/.git/`

2. `git add` (Colocando na caixa)

Este comando move os arquivos do *Working Directory* para a *Staging Area*. Você está dizendo ao Git: "Preste atenção neste arquivo; ele deve entrar no próximo 'save'".

```
# Para adicionar um arquivo específico:
git add index.html

# Para adicionar TODOS os arquivos modificados na pasta atual (o mais comum):
git add .
```

3. `git commit` (Etiquetando a caixa)

Este é o comando que "salva o jogo". Ele pega tudo que está na *Staging Area* e cria um *snapshot* permanente no seu histórico.

O `-m` significa "mensagem". **Sempre** escreva mensagens claras sobre *o quê* você fez.

```
git commit -m "Adiciona a página inicial HTML básica"
```

O Comando Bônus: `git status`

Esse é seu melhor amigo. A qualquer momento, se você estiver perdido, apenas digite:

```
git status
```

O Git dirá exatamente o que está acontecendo: quais arquivos estão modificados, quais estão na *Staging Area* e quais não estão sendo rastreados.

Capítulo 3: Exemplo Prático: Criando um Site

Vamos ver como isso funciona na prática.

1. Crie a pasta do projeto e entre nela:

```
mkdir meu-site
cd meu-site
```

2. Inicie o Git:

```
git init
```

3. Crie seu primeiro arquivo (ex: `index.html`):

```
<!DOCTYPE html>
<html>
  <head>
    <title>Meu Site</title>
  </head>
  <body>
    <h1>Bem-vindo!</h1>
  </body>
</html>
```

4. Verifique o status: (O Git dirá que existe um "Untracked file" chamado `index.html`)

```
git status
```

5. Adicione o arquivo à Staging Area:

```
git add index.html
```

6. Verifique o status novamente: (Agora o Git dirá "Changes to be committed")

```
git status
```

7. Faça seu primeiro commit (salve o snapshot):

```
git commit -m "Commit inicial: Cria o index.html básico"
```

Parabéns! Você acabou de salvar sua primeira versão.

8. Vamos fazer uma mudança. Crie um `style.css`:

```
/* style.css */
body {
  background-color: #f0f0f0;
  font-family: sans-serif;
}
```

9. ...e não se esqueça de linkar no seu `index.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Meu Site</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1>Bem-vindo!</h1>
  </body>
</html>
```

10. Verifique o status: (O Git mostrará que o `index.html` foi modificado e o `style.css` é novo)

```
git status
```

11. Adicione TUDO de uma vez:

```
git add .
```

12. Faça o segundo commit:

```
git commit -m "Adiciona arquivo CSS e aplica estilos básicos"
```

Pronto! Agora você tem duas "fotos" do seu projeto salvas.

Capítulo 4: Trabalhando com o Histórico (O Diário de Bordo)

Agora que salvamos as coisas, como as vemos?

`git log` (Lendo o diário)

O `git log` mostra todo o histórico de commits, do mais novo para o mais antigo.

```
git log
```

A saída será algo assim (cheia de detalhes):

```
commit a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0 (HEAD -> master)
Author: Seu Nome <seuemail@email.com>
Date:   Fri Nov 14 16:30:00 2025 -0300

    Adiciona arquivo CSS e aplica estilos básicos

commit f0e1d2c3b4a5f6e7d8c9b0a1f2e3d4c5b6a7f8e9
Author: Seu Nome <seuemail@email.com>
Date:   Fri Nov 14 16:25:00 2025 -0300

    Commit inicial: Cria o index.html básico
```

Aquela sequência longa de letras e números (ex: `a1b2c3d4`) é o **hash** – o ID único de cada commit.

> *Dica Pro:* *O `git log` normal é muito verboso. Tente este comando para uma visão limpa e bonita:*
>
> ```
> git log --oneline --graph
> ```

"O que eu mudei mesmo?"

Às vezes, a mensagem do commit não é suficiente. Se você quiser ver *exatamente* o que mudou em um commit específico, use o `git show`:

```
# Use as primeiras 7 letras do hash que você viu no 'git log'
git show a1b2c3d
```

O Git mostrará as linhas exatas que foram adicionadas ou removidas.

O "Botão de Pânico" (Desfazendo mudanças)

O Git oferece muitas formas de desfazer coisas, mas vamos focar em duas situações comuns:

Pânico 1: "Modifiquei um arquivo, mas ainda não 'commitei', e quero descartar tudo."

Você mexeu no `index.html`, quebrou tudo e só quer voltar para a versão do *último commit*.

```
# Isso descarta TODAS as mudanças no arquivo 'index.html'
# CUIDADO: Essas mudanças são perdidas para sempre.
git checkout -- index.html
```

Pânico 2: "Eu 'commitei', mas me arrependi."

Você fez um commit (`git commit -m "Adiciona recurso X"`) e percebeu que ele quebrou o site. Você quer *desfazer* esse commit.

A forma mais segura de fazer isso é com `git revert`. Ele não apaga o commit antigo; ele cria um **novo commit** que faz exatamente o *oposto* do commit errado.

```
# Isso cria um NOVO commit que desfaz o que o ÚLTIMO commit (HEAD) fez.
git revert HEAD
```

Isso mantém o histórico limpo e seguro, pois você ainda pode ver o commit errado e o commit que o consertou.

Capítulo 5: Conclusão e Próximos Passos

Ufa! Se você chegou até aqui e acompanhou o exemplo, você já sabe o essencial para usar o Git no seu dia a dia.

O que você aprendeu:

- **O que é Git:** Um sistema de controle de versão (sua máquina do tempo).
- **O Fluxo Básico:** O ciclo de `git add` (preparar) e `git commit` (salvar).
- **Comandos Essenciais:** `init`, `add`, `commit` e o importantíssimo `status`.
- **Histórico:** Como usar `git log` para ver o passado e `git revert` para consertar erros de forma segura.

O Próximo Grande Passo: Branches

Até agora, trabalhamos em uma única "linha do tempo", chamada de `master` (ou `main`).

O verdadeiro poder do Git aparece quando você aprende a usar **Branches (Ramos)**.

Pense neles como universos paralelos. Você pode criar um *branch* novo (ex: "feature-formulario-contato") para trabalhar em algo novo sem bagunçar a linha do tempo principal (`master`). Quando terminar, você "funde" (merge) esse universo paralelo de volta ao principal.

Isso é o que permite que equipes inteiras trabalhem juntas, e será o foco do nosso próximo guia!

Continue praticando. Use `git commit` com frequência. Salve seu progresso. Você agradecerá a si mesmo no futuro!