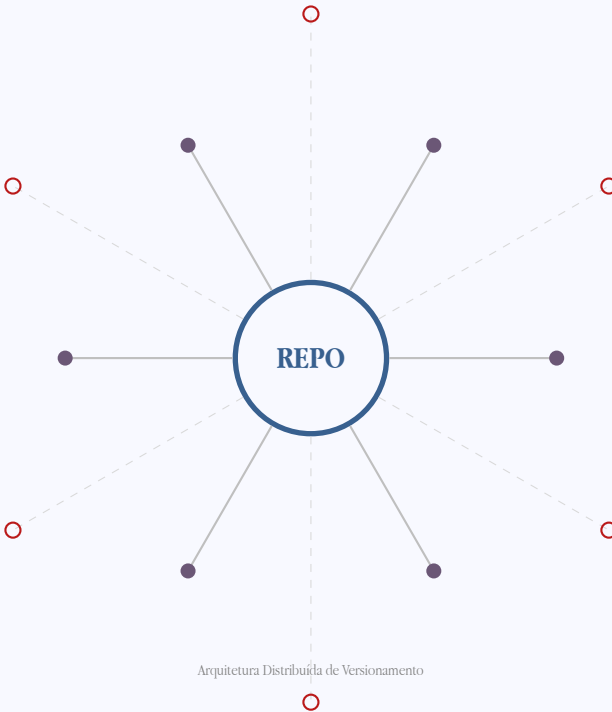


DOCUMENTAÇÃO OFICIAL - XXX

# GIT E GITHUB

*Guia de Referência e Comandos Essenciais*



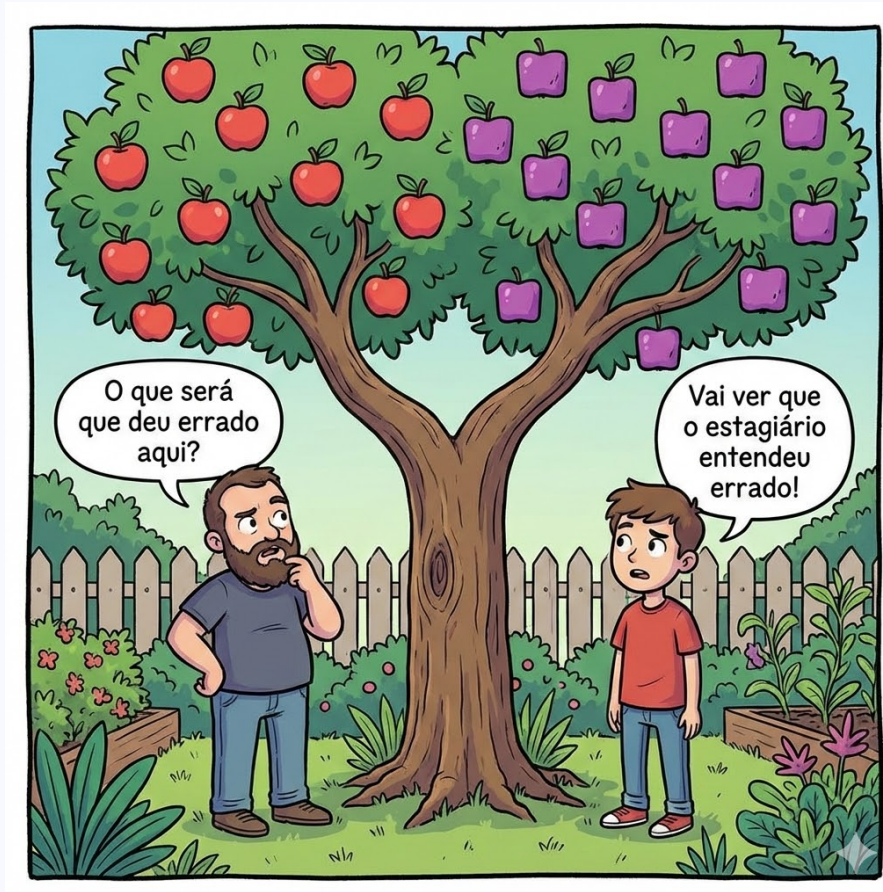
**Destinatário:** Equipe de Desenvolvimento

**Atualização:** 3 de dezembro de 2025

**Status:** Revisão 1.0

# GIT ESSENCIAL

*Sua Jornada no Controle de Versão Começa Aqui*



---

Para mais recursos e suporte, visite:  
[github.com/manual-git-iniciante](https://github.com/manual-git-iniciante)

Imagem gerada por IA: Gemini AI • 3 de dezembro de 2025

# Sumário

<b>1</b>	<b>Ciclo de Vida e Estados do Arquivo</b>	<b>8</b>
1.1	A Filosofia dos Três Estados . . . . .	8
1.2	Visualização do Fluxo de Dados . . . . .	8
1.3	O Painel de Controle: <code>git status</code> . . . . .	9
1.4	Preparando o Terreno (Staging) . . . . .	10
1.5	Consolidando a História (Commit) . . . . .	10
1.6	Visualizando o Histórico ( <code>git log</code> ) . . . . .	10
<b>2</b>	<b>Gerenciamento de Ramificações: Paralelismo e Isolamento</b>	<b>12</b>
2.1	Conceito de Branches . . . . .	12

Primary

	mdprimary5	mdprimary10	mdprimary15
mdprimary20	mdprimary25	mdprimary30	mdprimary35
mdprimary40	mdprimary50	mdprimary60	mdprimary70
mdprimary80	mdprimary90	mdprimary95	mdprimary98
mdprimary99	mdprimary100		

Secondary

	mdsecondary5	mdsecondary10	mdsecondary15
mdsecondary20	mdsecondary25	mdsecondary30	mdsecondary35
mdsecondary40	mdsecondary50	mdsecondary60	mdsecondary70
mdsecondary80	mdsecondary90	mdsecondary95	mdsecondary98
mdsecondary99	mdsecondary100		

Tertiary

	mdtertiary5	mdtertiary10	mdtertiary15
mdtertiary20	mdtertiary25	mdtertiary30	mdtertiary35
mdtertiary40	mdtertiary50	mdtertiary60	mdtertiary70
mdtertiary80	mdtertiary90	mdtertiary95	mdtertiary98
mdtertiary99	mdtertiary100		

# Neutral

	mdneutral5	mdneutral10	mdneutral15
mdneutral20	mdneutral25	mdneutral30	mdneutral35
mdneutral40	mdneutral50	mdneutral60	mdneutral70
mdneutral80	mdneutral90	mdneutral95	mdneutral98
mdneutral99	mdneutral100		

# Neutral Variant

	mdneutralvariant5	mdneutralvariant10	mdneutralvariant15
mdneutralvariant20	mdneutralvariant25	mdneutralvariant30	mdneutralvariant35
mdneutralvariant40	mdneutralvariant50	mdneutralvariant60	mdneutralvariant70
mdneutralvariant80	mdneutralvariant90	mdneutralvariant95	mdneutralvariant98
mdneutralvariant99	mdneutralvariant100		

# Light Scheme

lightprimary	lightsurfacetint	lightonprimary	lightprimarycontainer
lightonprimarycontainer	lightsecondary	lightonsecondary	lightsecondarycontainer
lightonsecondarycontainer	lighttertiary	lightontertiary	lighttertiarycontainer
lightontertiarycontainer	lighterror	lightonerror	lighterrorcontainer
lightonerrorcontainer	lightbackground	lightonbackground	lightsurface
lightonsurface	lightsurfacevariant	lightonsurfacevariant	lightoutline
lightoutlinevariant			lightinversesurface
lightinverseonsurface	lightinverseprimary		

# Dark Scheme

darkprimary	darksurfacetint	darkonprimary	darkprimarycontainer
darkonprimarycontainer	darksecondary	darkonsecondary	darksecondarycontainer
darkonsecondarycontainer	darktertiary	darkontertiary	darktertiarycontainer
darkontertiarycontainer	darkerror	darkonerror	darkerrorcontainer
darkonerrorcontainer	darkbackground	darkonbackground	darksurface
darkonsurface	darksurfacevariant	darkonsurfacevariant	darkoutline
darkoutlinevariant			darkinversesurface
darkinverseonsurface	darkinverseprimary		

# Git Colors

gitmain	gitmaincontainer	ongitmain	ongitmaincontainer
gitdev	gitdevcontainer	ongitdev	ongitdevcontainer
gitfeat	gitfeatcontainer	ongitfeat	ongitfeatcontainer
githot	githotcontainer	ongithot	ongithotcontainer
gitrelease	gitreleasecontainer	ongitrelease	ongitreleasecontainer

# Syntax Highlighting

codegreen	codegray	codepurple	codestring
codefunction	codevariable		

# Backgrounds

pagebg	backcolour	surfacelight	surfacedark
--------	------------	--------------	-------------

# Semantic Colors

success	successlight	successdark	warning
warninglight	warningdark	error	errorlight
errordark	info	infolight	infodark

## Aliases

primary	onprimary	primarycontainer	onprimarycontainer
secondary	onsecondary	secondarycontainer	onsecondarycontainer
tertiary	ontertiary	tertiarycontainer	ontertiarycontainer
surface	onsurface	surfacevariant	onsurfacevariant
background	onbackground	outline	outlinevariant



## Capítulo 1

# Ciclo de Vida e Estados do Arquivo

### 1.1 A Filosofia dos Três Estados

Muitos iniciantes acreditam erroneamente que, ao salvar um arquivo no editor de texto (Ctrl+S), o Git já o "salvou". **Isso não é verdade.** O Git é passivo; ele não rastreia nada a menos que você ordene explicitamente.

Para dominar o Git, você deve entender que seu arquivo transita por três zonas distintas. Imagine uma fotografia profissional:

1. **Working Directory (A Cena):** É o mundo real. Onde as coisas mudam caoticamente. O Git vê as mudanças, mas as ignora.
2. **Staging Area (O Visor da Câmera):** É o enquadramento. Você escolhe, dentre a bagunça da cena, o que quer que apareça na foto. É uma área de preparação.
3. **Repository / .git (A Fotografia Revelada):** É o registro histórico permanente e imutável.

### 1.2 Visualização do Fluxo de Dados

Observe o diagrama abaixo. Seu objetivo diário é mover seu código da esquerda (Inseguro/Volátil) para a direita (Seguro/Histórico).

#### ⚠ Atenção

Meu novo aviso

#### i Nota Importante

Minha nota informativa

#### </> Selecionando arquivos cirurgicamente

```
1 # Opcao 1: Adiciona APENAS o arquivo de estilos (Recomendado)
2 git add css/style.css
3
4 # Opcao 2: Adiciona TUDO que foi modificado (Use com cautela)
5 git add .
6
```

Código 1.1: Selecionando arquivos cirurgicamente



O Git só salva o que está no quadrado amarelo!

Figura 1.1: O fluxo mandatório: Nada vai para o Repositório sem passar pelo Staging.

## 1.3 O Painel de Controle: `git status`

Se você estivesse pilotando um avião, o `git status` seria seu altímetro. Você **deve** executar este comando constantemente. Ele lhe diz exatamente onde seus arquivos estão no diagrama acima.

### 💡 Dica

*Sempre configure seu nome e email antes de fazer o primeiro commit. Isso garante que suas contribuições sejam corretamente atribuídas no histórico do projeto.*

Nunca execute um comando destrutivo sem antes checar o status.

### 💡 Dica Pro

*Use `git config --global` para configurações que valem para todos os repositórios, e `git config --local` para configurações específicas de um projeto.*

```
1 # O comando mais importante do dia a dia
2 git status
```

Código 1.2: Consultando o estado do projeto

O retorno do terminal lhe dirá:

- **Untracked files:** Arquivos novos que o Git nunca viu.
- **Changes not staged for commit:** Arquivos modificados, mas que não entrarão no próximo pacote.
- **Changes to be committed:** Arquivos na Staging Area, prontos para a "foto".

## 1.4 Preparando o Terreno (Staging)

A Staging Area permite que você faça **commits atômicos**. Em vez de salvar todo o seu trabalho de 5 horas em um único pacote confuso, você pode dividir em partes lógicas.

```
1 # Opcao 1: Adiciona APENAS o arquivo de estilos (Recomendado)
2 git add css/style.css
3
4 # Opcao 2: Adiciona TUDO que foi modificado (Use com cautela)
5 git add .
```

Código 1.3: Seleccionando arquivos cirurgicamente

## 1.5 Consolidando a História (Commit)

O *commit* não é um "Save", é um ponto na história. Se o projeto quebrar amanhã, é para este ponto que você voltará. Portanto, a mensagem do commit deve ser tratada como documentação técnica.

**Regra de Ouro:** A mensagem deve completar a frase: *"Se eu aplicar este commit, ele irá..."*

- **Ruim:** "ajustes", "fix", "mudando coisas".
- **Bom:** "adiciona validação de CPF no formulário", "corrige erro de cor no header".

```
1 # O flag -m define a mensagem
2 git commit -m "feat: implementa layout responsivo na home"
```

Código 1.4: O registro definitivo

## 1.6 Visualizando o Histórico (git log)

Para ver a lista de fotos (snapshots) que você tirou até agora:

```
1 # Mostra autor, data e mensagem de cada commit
2 git log
3
4 # Versao resumida (uma linha por commit)
5 git log --oneline
```

Código 1.5: Auditoria do projeto

### Desafio Avançado

Implemente um Git Hook que impeça commits na branch **main** e force o desenvolvedor a criar uma Pull Request. O hook deve:

- Detectar se o commit está sendo feito na main
- Exibir uma mensagem de erro explicativa
- Sugerir o fluxo correto (criar feature branch)

**Arquivo:** `.git/hooks/pre-commit`

### Exercícios

**Exercício 1.6.1:** Configure seu nome de usuário no Git usando o comando apropriado.

**Exercício 1.6.2:** Crie um novo repositório local e adicione um arquivo README.md com uma breve descrição do projeto.

**Exercício 1.6.3:** Faça um commit com a mensagem "feat: adiciona README inicial".

**Exercício 1.6.4:** Verifique o histórico de commits usando `git log`.

## Capítulo 2

# Gerenciamento de Ramificações: Paralelismo e Isolamento

## 2.1 Conceito de Branches

As `main`, `develop`, `feature/*` e `hotfix/*` branches são fundamentais no Git.

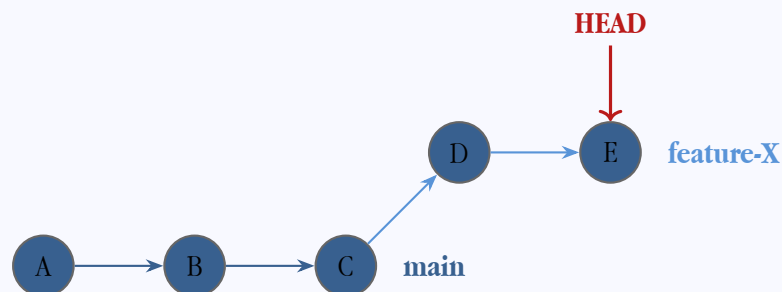


Figura 2.1: Exemplo usando a nova paleta Material Design 3

### Desafio

**Cenário:** Você está trabalhando em uma branch `feature/login` e descobriu que a branch `main` recebeu atualizações críticas.

**Seu desafio:**

1. Faça o rebase da sua branch `feature/login` com a `main`
2. Resolva os conflitos que aparecerem
3. Garanta que todos os testes passem após o rebase
4. Faça o push forçado (force push) da branch rebased

**Dica:** Use `git rebase -i main` para ter controle interativo do processo.

Lembre-se: sempre crie branches a partir da develop!

Atenção: Nunca faça commit direto na main!

Merge realizado com sucesso!

Conflito detectado! Resolução manual necessária.

### Exercícios

**Exercício 2.1.1:** Crie uma nova branch chamada `feature/menu`.

**Exercício 2.1.2:** Adicione um arquivo `menu.html` e faça o commit.