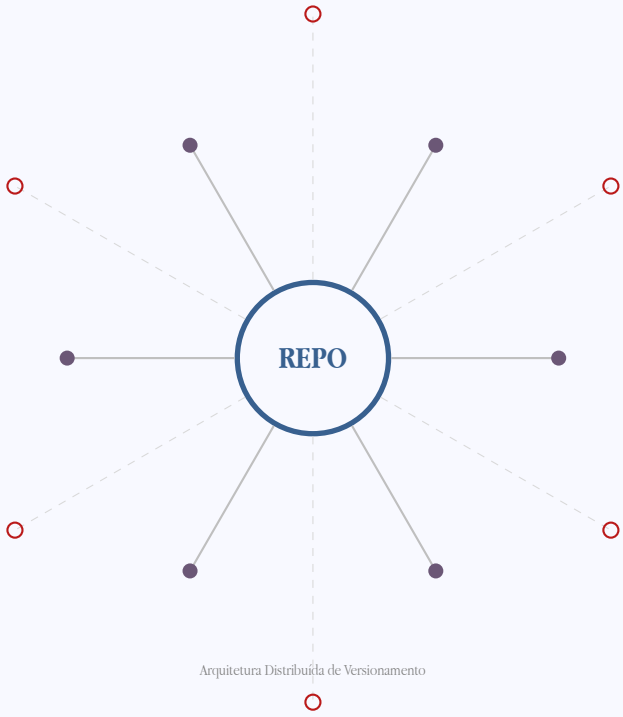
MANUAL TÉCNICO DEV OPS

DOCUMENTAÇÃO OFICIAL - XXX

GIT E GITHUB

Guia de Referência e Comandos Essenciais

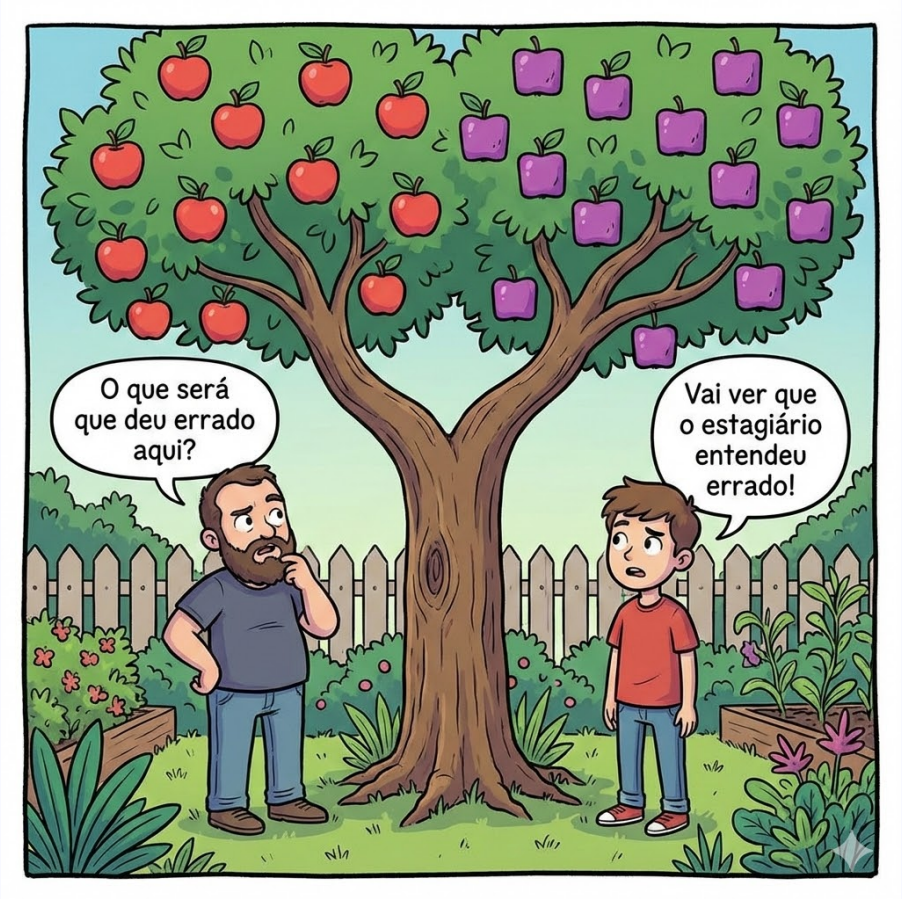


Arquitetura Distribuída de Versionamento

Destinatário: Equipe de Desenvolvimento
Atualização: 3 de dezembro de 2025
Status: Revisão 1.0

GIT ESSENCIAL

Sua Jornada no Controle de Versão Começa Aqui



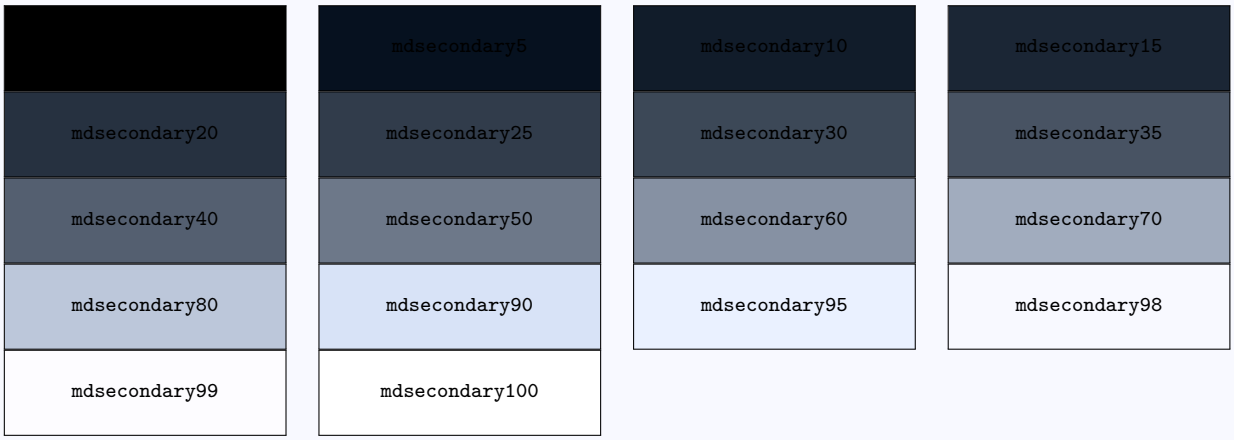
Para mais recursos e suporte, visite:
github.com/manual-git-iniciante

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

Sumário

Primary

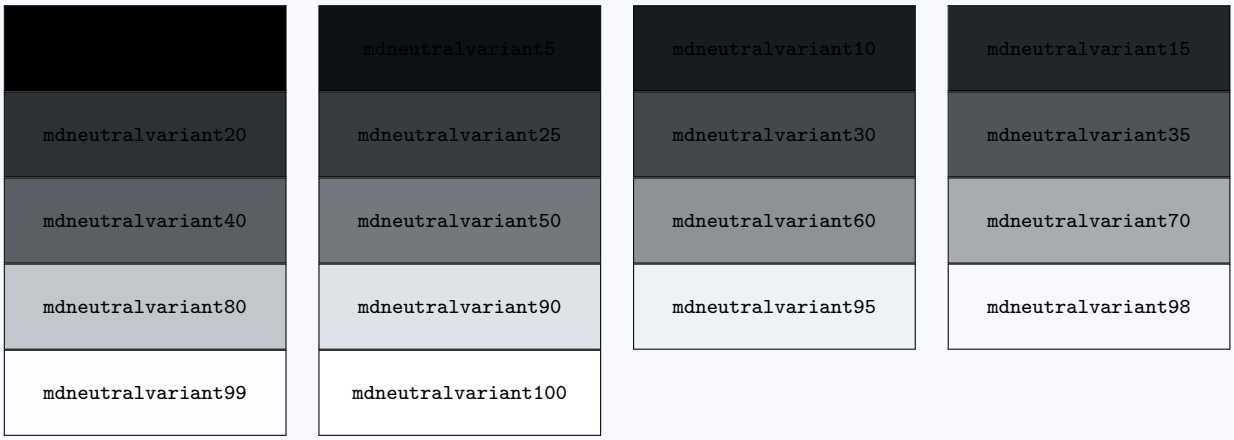
Secondary



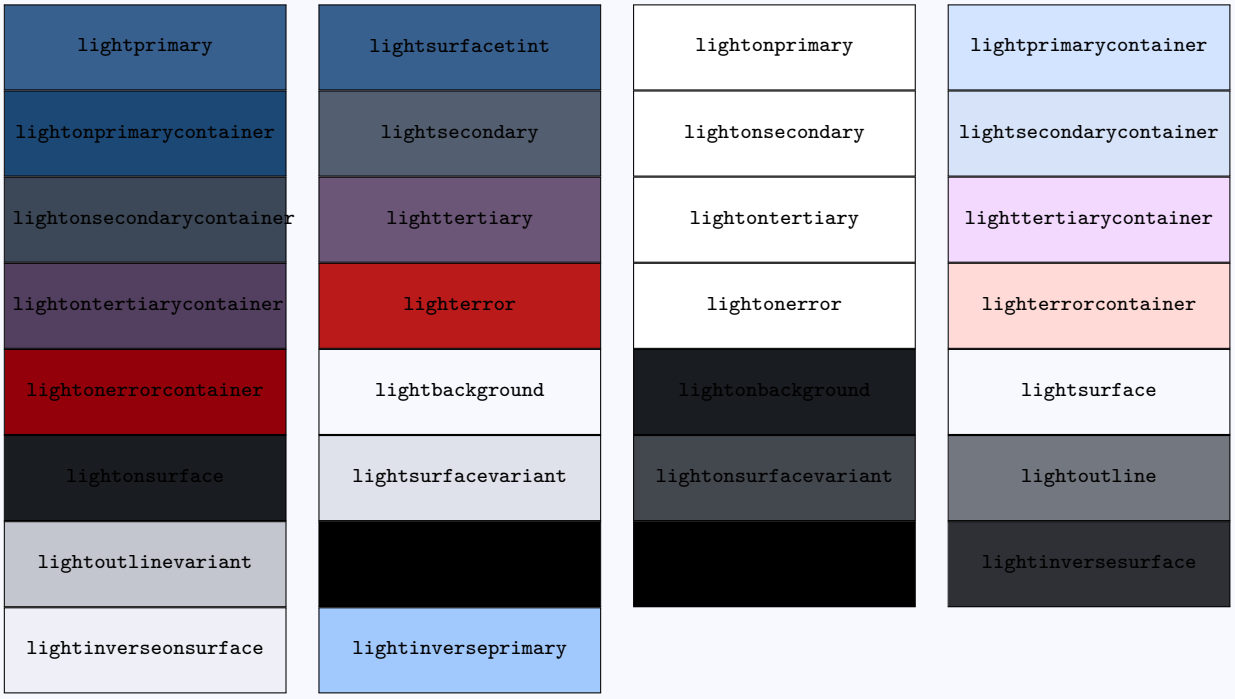
Tertiary

Neutral

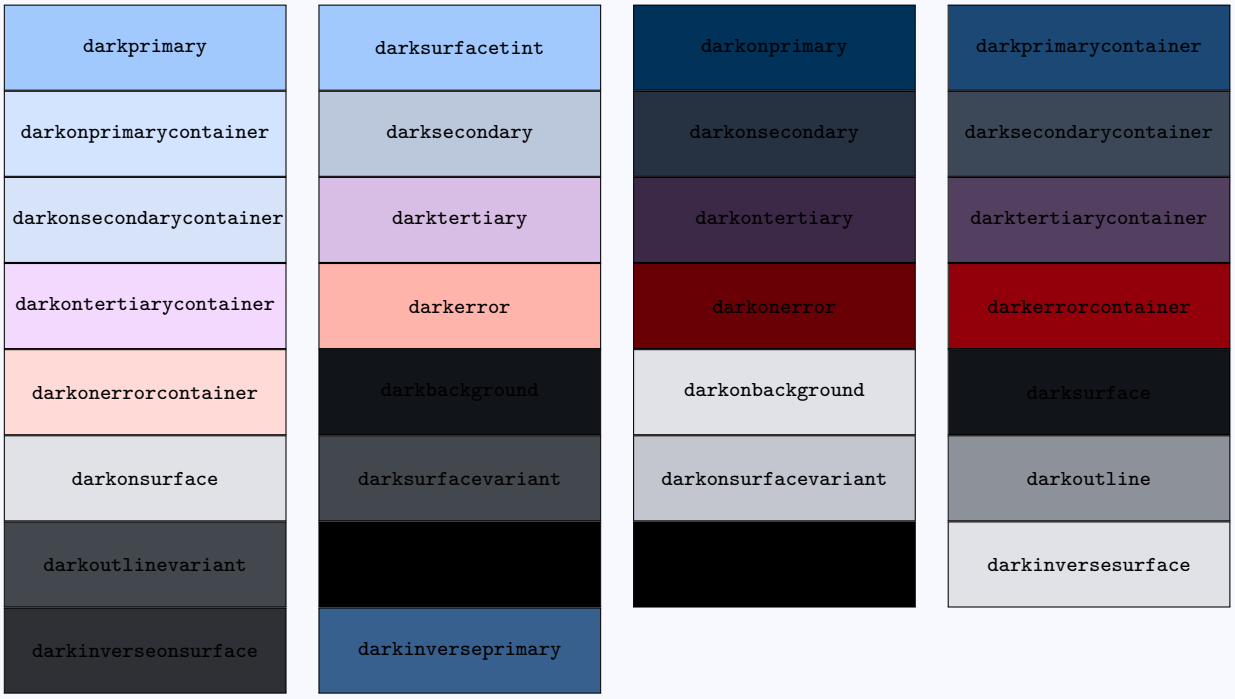
Neutral Variant



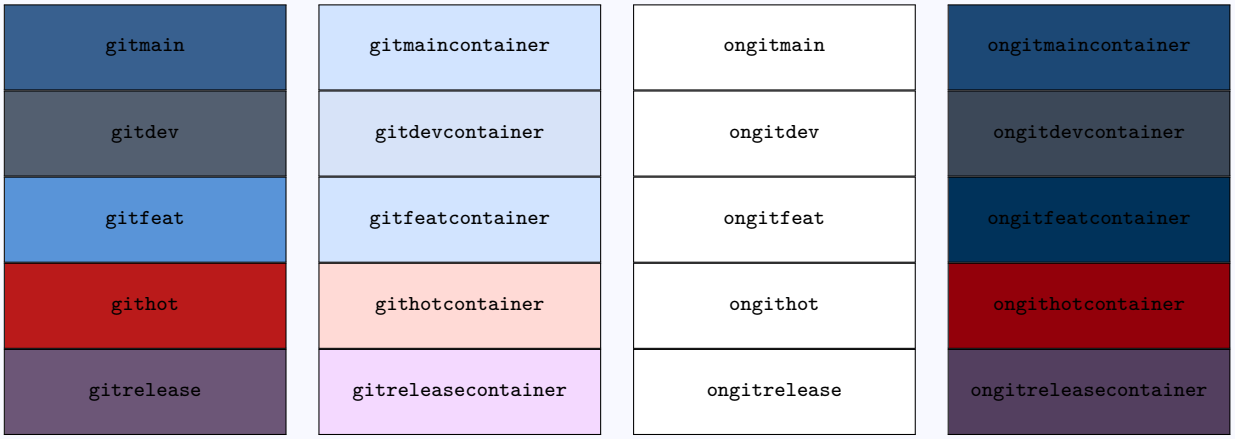
Light Scheme



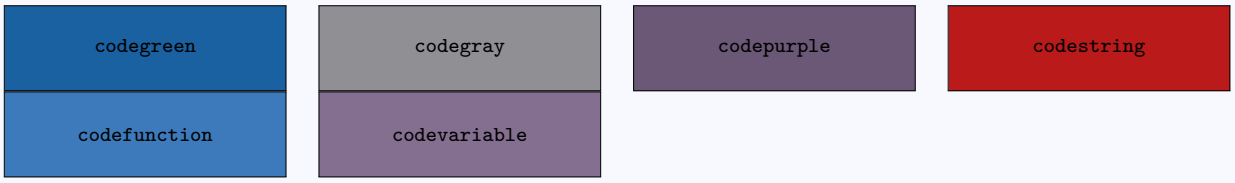
Dark Scheme



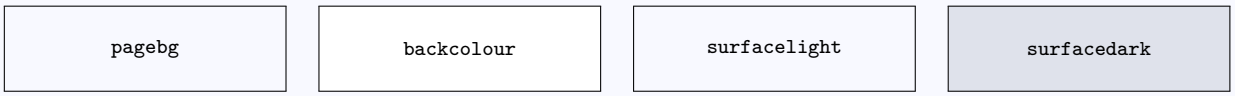
Git Colors



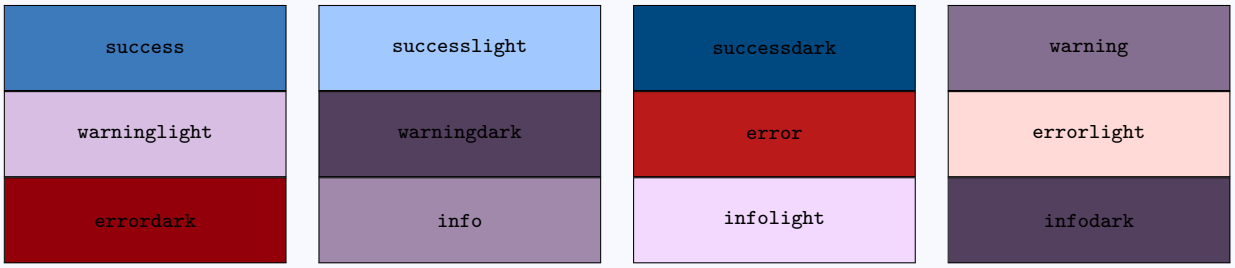
Syntax Highlighting



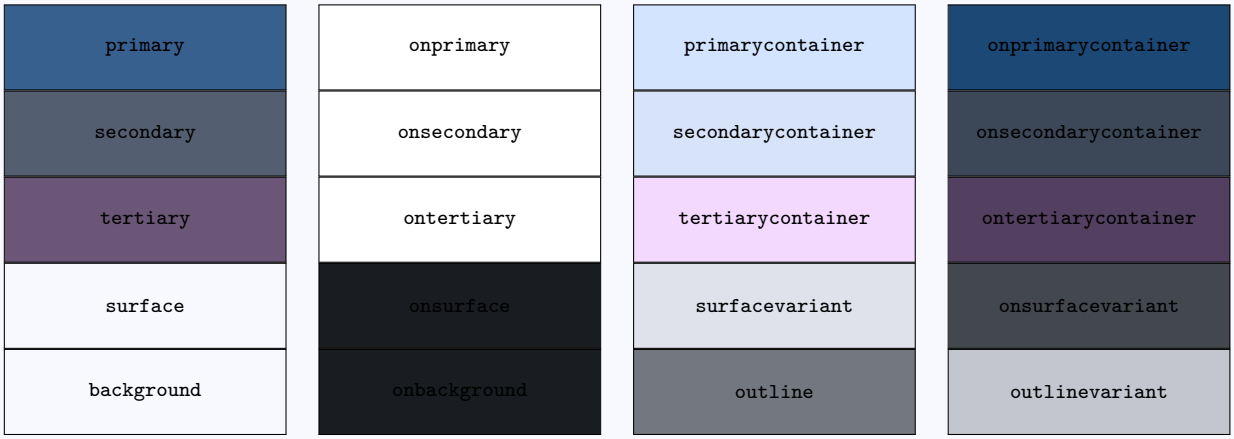
Backgrounds



Semantic Colors



Aliases


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

Nota Importante

Minha nota informativa

Selecionando arquivos cirurgicamente

```
# Opcao 1: Adiciona APENAS o arquivo de estilos (Recomendado)
git add css/style.css

# Opcao 2: Adiciona TUDO que foi modificado (Use com cautela)
git add .

```

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.

```
# O comando mais importante do dia a dia
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.

```
# Opcao 1: Adiciona APENAS o arquivo de estilos (Recomendado)
git add css/style.css

# Opcao 2: Adiciona TUDO que foi modificado (Use com cautela)
git add .
```

Código 1.3: Selecionando 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".

```
# O flag -m define a mensagem
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:

```
# Mostra autor, data e mensagem de cada commit
git log

# Versao resumida (uma linha por commit)
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.

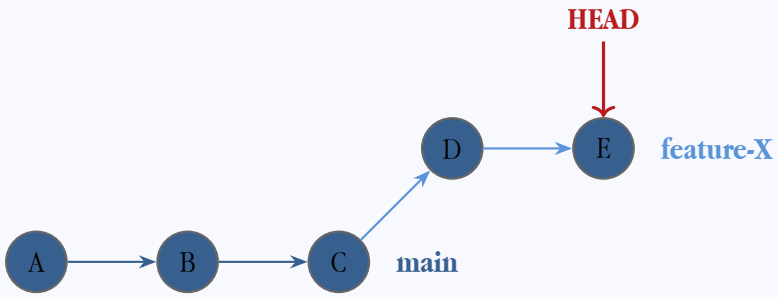


Figura 2.1: Exemplo usando a nova paleta Material Design 3

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.