

Guia Completo de Claude Code

Índice

- Estrutura de arquivos
- Como configurar
 - Globalmente (recomendado)
 - Por projeto
 - Como desabilitar menções nos commits
 - Habilitando skill-creator
- Como economizar tokens
 - Use assinatura, não API tokens
- Como funciona
 - Claude Code
 - Continue
 - Resume
 - Aceitar alterações automaticamente
 - CLAUDE.md
 - Customizando o seu CLAUDE.md (opcional)
 - Skills
 - Commands
 - Como gosto de usar
 - Hooks
- Trocando de Modelo
 - Usando /model
- Ligando os pontos
 - Fluxo sem planejamento
 - Devagar a gente chegar onde a gente quer chegar
 - Staged vs unstaged
- Economizando contexto
 - Não use modo terminal
- Paralelizando serviço
 - Git Worktrees
 - Tmux
- Hooks
 - Notification
- Comandos
 - /investigate
 - /investigate-batch
 - /create-feature
 - /review-staged
 - /open-pr
 - /trim
- MCP Servers

- o [Context7](#)
 - [Instalação](#)
 - o [Playwright](#)
-

Estrutura de arquivos

```
./claudie/
├-- CLAUDE.md
└-- commands/
    ├── investigate.md
    ├── investigate-batch.md
    ├── create-feature.md
    ├── review-staged.md
    ├── open-pr.md
    └── trim.md
└-- skills/
    ├── software-engineering/
    ├── reviewing-code/
    └── writing/
└-- hooks/
    └── notification.sh
```

Como configurar

Globalmente (recomendado)

Mova a pasta `.claudie` para a localização global do Claude Code: `~/ .claudie` ou `$HOME/.claudie`.

Isso faz com que as regras, comandos e skills estejam disponíveis para qualquer conversa com o Claude Code.

```
cd $HOME
mkdir .claudie
cp -r $HOME/Downloads/.claudie .claudie/
```

Por projeto

Ao copiar a pasta `.claudie` para a raiz do seu projeto, as regras, comandos e skill estarão disponíveis apenas para aquele projeto.

```
cd seu-projeto
mkdir .claudie
cp -r ~/Downloads/.claudie .claudie/
```

Como desabilitar menções nos commits

É controlado pela propriedade `includeCoAuthoredBy: false` no arquivo `settings.json`.

```
{  
  "includeCoAuthoredBy": false  
}
```

Antes:

```
Add user authentication  
 Generated with Claude Code
```

Depois:

```
Add user authentication
```

OBS: Essa configuração já está aplicada no `settings.json` do zip

Como funciona

Claude Code

Para abrir a aplicação, basta digitar `claude` no terminal.

Continue

Executa do ponto onde parou a última conversa.

```
claude --continue
```

Resume

Executa com uma lista de conversas passadas para você escolher de qual quer retomar.

```
claude --resume
```

Aceitar alterações automaticamente (perigoso!)

Executa o `claude` com permissão para editar arquivos e rodar comandos.

```
claude --dangerously-skip-permissions
```

ATENÇÃO: Tome cuidado, use por sua conta e risco, mas é útil para não ter que ficar aceitando edições toda hora.

CLAUDE.md

É carregado em todo início conversa. Pode ser criado através do comando `/init` dentro do Claude Code (ele analisa o código do seu projeto e monta um resumo com: tech stack, estrutura de pastas e padrões de implementação).

Customizando o seu CLAUDE{.}md (opcional)

Uma boa forma de fornecer mais contexto individual do seu projeto é rodar o comando `init` e mesclar o resultado com o Claude.md deste guia, assim você usufrui das boas práticas de engenharia de software, acessibilidade, testes, nomenclatura e escrita, mas de forma adaptada à sua stack.

Skills

São regras/atributos disponíveis ao Claude Code que são carregados apenas quando necessários.

Economiza tokens.

ATENÇÃO: Por padrão, o arquivo CLAUDE.md da pasta é o mínimo (que utiliza Skills para economizar token). Embora útil e funcional, caso tenha problemas com o Claude não seguindo as regras vale a pena tentar renomear o CLAUDE.md → CLAUDE-min.md e o CLAUDE-verbose.md para CLAUDE.md.

Comparação:

Abordagem	Tamanho	Descrição
Sem skills	CLAUDE-verbose.md: 1.003 tokens	Todas as regras no arquivo principal
Com skills	CLAUDE.md: 119 tokens	Regras básicas + skills carregadas sob demanda

Commands

Automatizam tarefas, podem se comunicar com MCPs e rodar comandos no seu terminal.

Como gosto de usar

```
/investigate {tema}
```

Faz perguntas focadas antes de planejar. Entender o problema primeiro.

```
/create-feature {descrição}
```

Cria branch, planeja, implementa e deixa as mudanças staged.

```
/review-staged
```

Revisa mudanças staged contra os padrões de código.

```
/open-pr {título}
```

Comita, dá push e abre um PR com resumo e plano de testes.

```
/trim
```

Reduz a descrição do PR em 70% mantendo info essencial.

Hooks

Scripts executados ao concluir tarefas.

Eu só uso o hook de *Notification*, para tocar o som do duolingo de sucesso ao finalizar uma tarefa.

Isso ajuda a não ficar preso olhando para o terminal, esperando a IA fazer o meu trabalho.

notification.sh:

```
#!/bin/bash

on_tool_complete() {
    afplay ~/duolingo-success.mp3
}
```

Resultado: Som quando Claude termina de escrever/alterar arquivos.

Trocando de Modelo

Usando /model

O Claude Code permite trocar de modelo durante uma conversa usando o comando **/model**.

```
/model
```

Isso abre um seletor de modelo onde você pode escolher entre os modelos disponíveis.

Quando usar Haiku:

- Tarefas simples (renomear, pequenos fixes)
- Perguntas rápidas sobre o codebase
- Tarefas que não exigem raciocínio profundo
- Economizar tokens em operações diretas

Quando usar Sonnet/Opus:

- Implementações complexas
- Decisões de arquitetura
- Refatoração multi-arquivo
- Tarefas que exigem raciocínio cuidadoso

Dica: Comece com Haiku para exploração e investigação, troque para Sonnet/Opus para implementação.

Ligando os pontos

Pela manhã, anote 1-3 tarefas pendentes em um bloco de notas.

Para cada uma delas, separe 5 minutos para pensar no que deve ser feito e +5 para escrever tudo que você sabe sobre a tarefa, detalhes da regra de negócio, particularidades do sistema, requisitos mínimos, tecnologias que serão utilizadas, design pattern a ser aplicado (caso exista).

Escreva tudo de forma contínua no bloco de notas, abaixo do título, dê uma pausa de 2 minutos, releia, ajuste e abra o Claude Code.

Fluxo recomendado:

1. Aperte *Shift + Tab* até ativar o modo *Plan*
2. Cole o texto e adicione referências de arquivos/pastas relacionadas à funcionalidade usando o `@ + caminho do arquivo`
3. Espere o plano ficar pronto → revise-o (altere-o com `ctrl + g` caso necessário)
4. Uma vez finalizado o plano, deixe que o Claude execute-o com `bypass permissions` (por sua conta e risco)
5. Ao finalizar, teste manualmente para ver se funciona
6. Caso dê erro, copie e cole a mensagem de erro no chat, itere até ficar aceitável
7. Finalizadas as alterações, utilize o comando `/review-staged` para garantir que nenhum conceito importante foi deixado de lado na implementação
8. Esse comando gerará algumas sugestões, leia-as e aplique as que fizer sentido
9. Uma vez aplicada as sugestões, é hora de rodar o `/open-pr` (vai commitar as alterações e abrir um pull request)

Fluxo sem planejamento

Caso não queira utilizar o modo de planejamento, basta utilizar o comando `/create-feature` + descrição. Ele criará uma nova *feature branch* e implementará as alterações.

Devagar a gente chegar onde a gente quer chegar

Depois da primeira "pernada" do dia, recomendo que façam iterações pequenas ao invés de longas funcionalidades. Recomendo também fechar e iniciar conversas frequentemente, a fim de economizar contexto.

Staged vs unstaged

Uma boa estratégia *pré-review* é deixar as alterações em *staged* (*git*) e pedir para o Claude aplicar melhorias, mas deixá-las em *unstaged*, assim você consegue comparar os 2.

Economizando contexto

Não use modo terminal

EVITE:

```
Você: Rode npm test  
Claude: [Executa e mostra 200 linhas]
```

Custo: 500 tokens por execução.

PREFIRA:

```
# Em outro terminal  
npm test  
  
# Se der erro, copie apenas a linha relevante  
# x Expected 'user' to be defined
```

Custo: 20 tokens.

Paralelizando serviço

Git Worktrees

Trabalhe em múltiplas features simultaneamente.

Com Git Worktrees, você consegue manter múltiplas versões do seu repositório (cada uma em uma branch), com isso, é possível executar uma instância de Claude Code em cada pasta.

Exemplo:

```
# Feature 1
cd projeto-main
git worktree add ../projeto-oauth -b feat/oauth
cd ../projeto-oauth
claude
# Claude trabalha aqui

# Em outro terminal/window/session
cd projeto-main
git worktree add ../projeto-pagamento -b feat/payment
cd ../projeto-pagamento
claude
# Claude trabalha aqui
```

Duas pastas, duas branches, zero conflito.

Tmux

```
my-website git:(main) > claudie --continue
my-website git:(main) > claudie --dangerously-skip-permissions
my-website git:(main) > claudie [ ] --dangerously-skip-permissions
my-website git:(main) > claudie --resume
```

É um multiplexador de terminal, permite abrir múltiplas janelas em 1 único *shell*.

É uma excelente ferramenta, mudou minha vida, mas meio complexa de ensinar a configurar e utilizar por texto, recomendo assistir tutoriais visuais sobre como configurá-lo no seu sistema operacional.

Hooks

Notification

Som do Duolingo:

```
#!/bin/bash
# ~/.claude/hooks/notification.sh

on_tool_complete() {
    afplay ~/Downloads/duolingo-success.mp3
}
```

Comandos

/investigate

Descubra antes de planejar.

Uso:

```
/investigate fluxo de autenticação
```

O que faz:

1. Faz perguntas focadas sobre o tema
2. Explora o codebase por padrões
3. Identifica lacunas de conhecimento
4. Fornece resumo estruturado

Quando usar: Antes de iniciar qualquer tarefa complexa.

⚠ ATENÇÃO: Este comando pode consumir tokens mais rápido que o normal, pois adiciona várias mensagens ao contexto durante a investigação. É um tradeoff entre consumo de tokens e qualidade da engenharia de prompt — quanto mais contexto o Claude tem, melhores são as respostas. Ideal para tarefas complexas onde o investimento vale a pena.

/investigate-batch

Versão econômica do /investigate — agrupa perguntas para economizar tokens.

Uso:

```
/investigate-batch fluxo de autenticação
```

O que faz:

1. Faz até 5 perguntas por rodada (ao invés de uma por vez)
2. Explora o codebase por padrões
3. Identifica pontos de integração
4. Fornece resumo estruturado com próximos passos

Quando usar: Quando você quer investigar mas precisa economizar tokens.

/create-feature

Cria branch + planeja + implementa.

Uso:

```
/create-feature Add user profile page
```

O que faz:

1. Cria **feat/add-user-profile-page**
2. Planeja implementação
3. Implementa com type-safety
4. Deixa mudanças staged para review

Quando usar: Features novas do zero.

/review-staged

Revisa código staged contra os padrões.

Uso:

```
git add src/auth.ts  
/review-staged
```

Verifica:

- Type safety (sem **any**)
- Clareza (nomes descritivos)
- Segurança (OWASP)
- Acessibilidade
- Testes

Quando usar: Antes de comitar.

/open-pr

Cria pull request.

Uso:

```
/open-pr Add OAuth authentication
```

Gera:

```
## Summary
- Integrates Google OAuth
- Adds session management
- Implements refresh tokens

## Test Plan
- [ ] Login with Google works
- [ ] Session persists
- [ ] Logout clears session
```

Quando usar: Feature pronta para review.

/trim

Reduz descrição do PR em 70%.

Uso:

```
/trim
```

O que faz:

1. Lê a descrição atual do PR
2. Identifica informação essencial
3. Reescreve de forma concisa
4. Retorna versão enxuta

Quando usar: Quando descrições de PR ficam longas demais.

MCP Servers

Model Context Protocol Servers é um assunto extenso, mas funcionam como "braços" para o Claude Code, tornando possível com que ele se comunique com ferramentas externas.

Existem MCP's de acesso a banco de dados, controle de navegador, github, terraform, figma, etc. Vale a pena pesquisar por MCPs das ferramentas que vocês usam, caso queiram integrá-la ao Claude Code.

Eu gosto de usar poucos.

Context7

Busca documentação atualizada das bibliotecas/linguagens e retorna em forma de contexto para IA.

CUIDADO: Pode consumir seus limites muito rápido, porque algumas documentações são bem extensas.

Acesse o site: context7.com para ver o tamanho de cada documentação.

Instalação

```
claude mcp add context7 -- npx -y @upstash/context7-mcp --api-key {API_KEY}
```

Exemplo:

Você: Use o mcp context7 para aprender mais sobre App Router do Next.js 14?

Claude: [Busca docs oficiais via Context7]

Claude: No Next.js 14, use 'use server' para...

Informação atualizada. Sem respostas defasadas.

Playwright

Testes E2E automatizados / Automação de navegador.

Exemplo:

Você: Crie teste E2E para login
Claude: [Gera via Playwright MCP]

```
test('user login', async ({ page }) => {
  await page.goto('/login')
  await page.fill('[name=email]', 'test@example.com')
  await page.fill('[name=password]', 'password123')
  await page.click('button[type=submit]')
  await expect(page).toHaveURL('/dashboard')
})
```

Claude roda o teste e valida automaticamente.

Por hoje é só, espero que tenha gostado!

Qualquer dúvida, minha DM no Twitter está sempre aberta: @ocodista