

# Guia Completo de Claude Code

Por FelipeNess | 13 de Dezembro de 2025

---

## Indice

- [Estrutura de arquivos](#)
- [Como configurar](#)
  - [Globalmente \(recomendado\)](#)
  - [Por projeto](#)
  - [Como desabilitar mencoes nos commits](#)
- [Como economizar tokens](#)
  - [Use assinatura, nao API tokens](#)
- [Como funciona](#)
  - [Claude Code](#)
  - [CLAUDE.md](#)
  - [Skills](#)
  - [Commands](#)
  - [Hooks](#)
  - [Trocando de Modelo](#)
- [Ligando os pontos](#)
  - [Fluxo sem planejamento](#)
  - [Staged vs unstaged](#)
- [Economizando contexto](#)
- [Paralelizando servico](#)
  - [Git Worktrees](#)
  - [Tmux](#)
- [Comandos](#)
  - [/investigate](#)
  - [/create-feature](#)
  - [/review](#)
  - [/review-staged](#)
  - [/open-pr](#)
  - [/ultrathink-review](#)
  - [/trim](#)
- [MCP Servers](#)
  - [Context7](#)
  - [Playwright](#)

---

## Estrutura de arquivos

```
/.claude/
├── CLAUDE.md
├── settings.json
└── commands/
    ├── investigate.md
    ├── investigate-batch.md
    ├── create-feature.md
    ├── new-feat.md
    ├── review.md
    └── review-staged.md
```

```
|   └── open-pr.md
|   └── ultrathink-review.md
|       └── trim.md
├── skills/
|   ├── coding-guidelines/
|   ├── typescript/
|   ├── react/
|   ├── software-engineering/
|   ├── planning/
|   ├── review-changes/
|   ├── reviewing-code/
|   ├── writing/
|   ├── copywriting/
|   └── ultrathink-review/
└── hooks/
    └── notification.sh
```

## Como configurar

### Globalmente (recomendado)

Mova a pasta `.claude` para `~/claude` ou `$HOME/.claude`.

Assim as regras, comandos e skills ficam disponíveis em qualquer conversa.

```
# Linux/Mac
cd $HOME
mkdir .claude
cp -r ~/Downloads/.claude .claude/

# Windows PowerShell
Copy-Item -Recurse ".claude" "$HOME\claude"
```

### Por projeto

Copie a pasta `.claude` para a raiz do projeto. As regras valem só pra aquele projeto.

```
cd seu-projeto
cp -r ~/Downloads/.claude .claude/
```

### Como desabilitar menções nos commits

Edite `settings.json`:

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

**Antes:**

```
Add user authentication
```

```
Generated with Claude Code
```

**Depois:**

```
Add user authentication
```

*Ja vem configurado no pacote.*

## Como economizar tokens

### Use assinatura, nao API tokens

Plano	Custo	Quando usar
<b>API</b>	\$3-15/milhao tokens	Automacao, CI/CD, uso esporadico
<b>Pro</b>	\$20/mes	<b>Desenvolvimento diario</b>
<b>Max</b>	\$100/mes	Uso muito intenso

**Recomendacao:** Pro pra 99% dos casos.

## Como funciona

### Claude Code

Abra o terminal e digite:

```
claudie
```

**Continue** - Retoma a ultima conversa:

```
claudie --continue
```

**Resume** - Escolha qual conversa retomar:

```
claudie --resume
```

**Aceitar tudo automaticamente** (perigoso):

```
claudie --dangerously-skip-permissions
```

*Use por sua conta e risco. Util pra nao ficar aceitando edição toda hora.*

## CLAUDE.md

Carregado em toda conversa. Crie com `/init` - o Claude analisa seu projeto e monta um resumo com tech stack, estrutura e padroes.

**Dica:** Rode `/init` e mescle com este guia. Assim voce pega as boas praticas adaptadas a sua stack.

## Skills

Regras carregadas sob demanda. **Economiza tokens.**

Abordagem	Tokens	Descricao
Sem skills	~1.000	Tudo no CLAUDE.md

Com skills	~120	Carrega só quando precisa
------------	------	---------------------------

## Economia: 88%

Skills disponíveis:

- `coding-guidelines` - Padrões de código
- `typescript` - Standards TS/JS
- `react` - Best practices React/Next.js
- `software-engineering` - Princípios core
- `planning` - Arquitetura e decisões
- `review-changes` - Code review completo
- `reviewing-code` - Review rápido
- `writing` - Docs e commits
- `copywriting` - Marketing
- `ultrathink-review` - Review profundo (SOLID, DRY, KISS, YAGNI, CUPID)

---

## Commands

Automatizam tarefas. Podem rodar comandos no terminal e se comunicar com MCPs.

### Como uso:

```
/investigate {tema}
```

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

```
/create-feature {descrição}
```

Cria branch, planeja, implementa e deixa staged.

```
/review-staged
```

Revisa mudanças staged contra os padrões.

```
/ultrathink-review
```

Review profundo com SOLID, DRY, KISS, YAGNI, CUPID. Analisa defensive programming, early return, short-circuit.

```
/open-pr {título}
```

Comita, da push e abre PR com resumo e test plan.

```
/trim
```

Reduz descrição do PR em 70%.

---

## Hooks

Scripts que rodam ao concluir tarefas.

Uso o hook de **Notification** - toca o som do Duolingo quando termina uma tarefa. Assim não fico preso olhando o terminal.

### notification.sh:

```
#!/bin/bash

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

## Trocando de Modelo

Use `/model` pra trocar durante a conversa.

### Quando usar Haiku:

- Tarefas simples (renomear, pequenos fixes)
- Perguntas rápidas
- Economizar tokens

### Quando usar Sonnet/Opus:

- Implementações complexas
- Decisões de arquitetura
- Refatoração multi-arquivo

*Comece com Haiku pra explorar, troque pra Sonnet/Opus pra implementar.*

## Ligando os pontos

Pela manhã, anote 1-3 tarefas no bloco de notas.

Pra cada uma:

1. 5 min pensando no que fazer
2. 5 min escrevendo tudo que sabe (regras de negócio, requisitos, tecnologias, patterns)
3. Pausa de 2 min, releia, ajuste
4. Abra o Claude Code

### Fluxo recomendado:

1. Aperte `Shift + Tab` ate ativar modo Plan
2. Cole o texto e adicione referencias com `@` + caminho
3. Espere o plano ficar pronto, revise (altere com `Ctrl + G` se precisar)
4. Deixe o Claude executar com bypass permissions (por sua conta e risco)
5. Teste manualmente
6. Deu erro? Cole a mensagem no chat, itere ate funcionar
7. Rode `/review-staged` pra garantir que nada ficou de fora
8. Aplique as sugestões que fizerem sentido
9. Rode `/open-pr` pra commitar e abrir o PR

## Fluxo sem planejamento

Não quer planejar? Use direto:

```
/create-feature Add user authentication
```

Cria branch e implementa.

## Staged vs unstaged

**Dica pre-review:** Deixe as alterações em staged e peça pro Claude aplicar melhorias em unstaged. Assim você compara os dois.

---

## Economizando contexto

### Nao use modo terminal

**EVITE:**

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

Custo: 500 tokens.

**PREFIRA:**

```
# Em outro terminal  
npm test  
  
# Deu erro? Copie so a linha relevante  
# x Expected 'user' to be defined
```

Custo: 20 tokens.

---

## Paralelizando servico

### Git Worktrees

Trabalhe em multiphas features ao mesmo tempo.

```
# Feature 1  
cd projeto-main  
git worktree add ../projeto-oauth -b feat/oauth  
cd ../projeto-oauth  
claudie  
# Claude trabalha aqui  
  
# Em outro terminal  
cd projeto-main  
git worktree add ../projeto-pagamento -b feat/payment  
cd ../projeto-pagamento  
claudie  
# Claude trabalha aqui
```

Duas pastas, duas branches, zero conflito.

### Tmux

Multiplexador de terminal - multiphas janelas em 1 shell.

Ferramenta excelente, mudou minha vida. Mas complexa de explicar por texto. Assista tutoriais visuais pra configurar no seu SO.

---

## Comandos

### /investigate

Descubra antes de planejar.

```
/investigate fluxo de autenticacao
```

#### O que faz:

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

**Quando usar:** Antes de qualquer tarefa complexa.

**ATENCAO:** Pode consumir tokens rapido. E um tradeoff - mais contexto = respostas melhores.

---

#### /create-feature

Cria branch + planeja + implementa.

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

#### O que faz:

1. Cria `feat/add-user-profile-page`
2. Planeja implementacao
3. Implementa com type-safety
4. Deixa mudancas staged

**Quando usar:** Features novas do zero.

---

#### /review

Revisa todas as alteracoes contra os padroes.

```
/review
```

#### Verifica:

- Type safety (sem `any`)
- Clareza (nomes descritivos)
- Seguranca (OWASP)
- Acessibilidade (WCAG)
- Performance
- Testes

**Quando usar:** Após implementar, antes de commitar.

---

#### /review-staged

Revisa só o que tá em staged.

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

**Quando usar:** Antes de commitar.

---

#### /open-pr

Cria pull request.

```
/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 pra review.

## /ultrathink-review

Review profundo. Analisa codigo com:

#### Principios:

- **SOLID** - Single Responsibility, Open/Closed, Liskov, Interface Segregation, Dependency Inversion
- **DRY** - Don't Repeat Yourself
- **KISS** - Keep It Simple
- **YAGNI** - You Aren't Gonna Need It
- **CUPID** - Composable, Unix Philosophy, Predictable, Idiomatic, Domain-based

#### Verifica tambem:

- Defensive Programming (validacao de inputs, edge cases)
- Early Return (codigo flat, sem if-else chains)
- Short-Circuit Evaluation (operacoes caras por ultimo)
- Compatibilidade Node.js
- Comentarios desnecessarios e codigo morto

#### Output:

- 1.Codigo revisado e otimizado
- 2.Resumo tecnico das melhorias
- 3.Impacto em performance e manutencao
- 4.Perguntas ao autor

```
/ultrathink-review
```

**Quando usar:** Reviews completos, auditorias de codigo, PRs importantes.

## /trim

Reduz descricao do PR em 70%.

```
/trim
```

**Quando usar:** Descricoes longas demais.

## MCP Servers

Model Context Protocol - "bracos" pro Claude se comunicar com ferramentas externas.

Existem MCPs de banco de dados, navegador, GitHub, Terraform, Figma...

Uso poucos.

## Context7

Busca documentacao atualizada das libs/linguagens.

**CUIDADO:** Pode consumir limites rapido. Algumas docs sao extensas.

### Instalacao:

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

### Exemplo:

Voce: Use context7 pra aprender sobre App Router do Next.js 14  
Claude: [Busca docs oficiais]  
Claude: No Next.js 14, use 'use server' para...

Informacao atualizada. Sem respostas defasadas.

## Playwright

Testes E2E / automacao de navegador.

Voce: Crie teste E2E pro 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')
})
```

## Principios de Código

### Type Safety

- Nunca use `any`
- Quase nunca use `as`
- E2E type-safety (API -> Database -> UI)
- Use query-builders, nao SQL puro

### Clareza

- Nomes descritivos, sem abreviacoes
- Early returns, codigo flat
- Sem magic strings/numbers
- `camelCase` funcoes, `kebab-case` arquivos, `SNAKE_CAPS` constantes

## React

- Componentes puros
- React Query pra data fetching (nao useEffect)
- Suspense + Error Boundaries
- Cache keys com enums

## Testes

- Teste comportamento, nao implementacao
- Teste pra cada bug corrigido
- Verbos em 3a pessoa (nao "should")

---

Por hoje e so.

Qualquer duvida, me encontre no GitHub.

---

## FelipeNess 13 de Dezembro de 2025

GitHub: [github.com/Felipeness](https://github.com/Felipeness) Dotfiles: [github.com/Felipeness/dotfiles](https://github.com/Felipeness/dotfiles) Config Claude: [github.com/Felipeness/clade-config-felipeness](https://github.com/Felipeness/clade-config-felipeness)

"Voce pode ter muito conhecimento, mas sabedoria voce nao aprende em livros." — Sean (Robin Williams), Genio Indomavel