

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
- 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
 - /new-feat
 - /review
 - /open-pr
- MCP Servers
 - Context7
 - Instalação
 - Playwright

Estrutura de arquivos

```
./.claude/
├─ CLAUME.md
└─ commands/
    ├─ new-feat.md
    ├─ review.md
    └─ open-pr.md
└─ skills/
    ├─ coding-guidelines/
    ├─ copywriting/
    ├─ planning/
    ├─ review-changes/
    └─ writing/
└─ hooks/
    └─ notification.sh
```

Como configurar

Globalmente (recomendado)

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

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

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

Por projeto

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

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

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

Habilitando `skill-creator`

Permite criar skills sob demanda durante conversas.

Passo 1: Acesse as configurações de capacidades

The screenshot shows the 'Configurações' (Settings) page of the Claude AI web interface. On the left, there's a sidebar with icons for Home, Plus, Question, Print, and Refresh, followed by a list of sections: Geral, Conta, Privacidade, Cobrança, Uso, Capacidades (which is highlighted with a dark background), Conectores, and Claude Code.

The main content area is titled 'Capacidades' and contains the following sections:

- Capacidades**: Describes controlling what capabilities Claude uses in conversations.
- Artefatos**: Allows generating code snippets, documents, or website designs. A toggle switch is turned on.
- Artefatos com IA**: Allows creating interactive applications and documents using Claude. A toggle switch is turned off.
- Execução de código e criação de arquivos**: Allows executing code and creating/editing documents, spreadsheets, presentations, PDFs, and data reports. A toggle switch is turned on.
- Permitir saída de rede**: Allows access to the network for installing packages and libraries. A toggle switch is turned on.
- Lista de domínios permitidos**: A dropdown menu set to 'Apenas gerenciadores de pacotes'.
 - A tooltip explains that Claude can access common package managers plus specific ones listed below.
- Domínios adicionais permitidos**: A text input field containing 'example.com ou *.example.com' and a 'Adicionar' button.
- 0 domínio adicional**: Displays the count of additional domains.

At the bottom left, there's a note: 'Open <https://claude.ai/settings/claude-code> in a new tab and focus it' with a 'Memória' button.

Passo 2: Arraste até "Habilidades" e habilite "skill-creator"

Habilidades Ver prévia Fazer upload de habilidade

algorithmsic-art Creating algorithmic art using p5.js with seeded randomness and interactive parameter exploration. Use this when users request creating art using code, generative art, flow fields, or particle systems....

artifacts-builder Suite of tools for creating elaborate, multi-component claude.ai HTML artifacts using modern frontend web technologies (React, Tailwind CSS, shadcn/ui). Use for complex artifacts requiring state management, routing, ...

brand-guidelines Applies Anthropic's official brand colors and typography to any sort of artifact that may benefit from having Anthropic's look-and-feel. Use it when brand colors or style guidelines, visual formatting, or company design...

canvas-design Create beautiful visual art in .png and .pdf documents using design philosophy. You should use this skill when the user asks to create a poster, piece of art, design, or other static piece. Create original visual designs, never...

internal-comm A set of resources to help me write all kinds of internal communications, using the formats that my company likes to use. Claude should use this skill whenever asked to write some sort of internal communications (status...)

mcp-builder Guide for creating high-quality MCP (Model Context Protocol) servers that enable LLMs to interact with external services through well-designed tools. Use when building MCP servers to integrate external APIs or services...

skill-creator Guide for creating effective skills. This skill should be used when users want to create a new skill (or update an existing skill) that extends Claude's capabilities with specialized knowledge, workflows, or tool integrations.

slack-gif-creator Toolkit for creating animated GIFs optimized for Slack, with validators for size constraints and composable animation primitives. This skill applies when users request animated GIFs or emoji animations for Slack from...

theme-factory

Como usar:

Você: Crie uma skill para programação em {{ linguagem que você usa }}, seguindo {{ regras do time, equipe, empresa ou individuais }},
 Claude: [Cria automaticamente skill em ~/.claude/skills/]

Como economizar tokens

Use assinatura, não API tokens

Plano	Custo	Características	Quando usar
API	\$3/milhão tokens entrada	Custo médio: \$80-200/mês	Automação em produção CI/CD pipelines
	\$15/milhão tokens saída	(R\$400 a R\$1.000)	Uso esporádico (<10h/mês)
Pro	\$20/mês	Prioridade em horários de pico Economia: 75-90% vs API Limite de uso diário/semanal	RECOMENDADO: Desenvolvimento diário

Plano	Custo	Características	Quando usar
Max	\$100/mês	5x mais limite que Pro Dificilmente atinge limite	Uso muito intenso (considere 2 contas Pro antes)

Recomendação: Desenvolvimento diário = Assinatura Pro

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 críticas 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

```
/new-feat {descrição da tarefa}
```

Gera uma nova branch partindo da *main*, planeja a tarefa utilizando as diretrizes de código, executa e deixa mudanças *staged*.

```
/review
```

Revisa as alterações feitas e sugere melhorias.

Aplico as melhorias, pedindo para deixá-las *unstaged*, assim consigo comparar as duas versões (implementação + sugestão de melhoria)

```
/pr
```

Commita as alterações atômicamente, utilizando *conventional commits*, dá *push* e abre um PR utilizando o seguinte template:

```
# What and why?  
{1 Line: What's the most important information about this change?}  
  
{3 Lines: How was implemented and why is it necessary?}  
  
# Changes  
{Describe the changes concisely}  
  
# Security  
{Are there any security concerns or edge cases that deserves attention?}  
  
# References  
{Include documentation, articles or book references to enrich the context of this PR.}
```

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.

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` 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 `/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 `/new-feat` + 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) ✘ claude --continue
+ my-website git:(main) ✘ claude --dangerously-skip-permissions
+ my-website git:(main) ✘ claude --dangerously-skip-permissions
+ my-website git:(main) ✘ claude --resume
[0] 0:node~ 1:website* "Caio's MacBook Pro loc" 21:47 29-Oct-29
```

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

/new-feat

Cria branch + desenvolve feature.

Uso:

```
/new-feat Add user profile page
```

O que faz:

1. Cria [feat/add-user-profile-page](#)
2. Planeja arquitetura
3. Implementa código
4. Adiciona testes
5. Comita com mensagem descritiva

Quando usar: Features novas do zero.

/review

Revisa código staged.

Uso:

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

Analisa:

- Type safety
- Padrões do CLAUDE.md
- Segurança (OWASP)
- Performance
- Acessibilidade

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.

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