AI コーディング コストカットチートシート

チームの大元に仕込むものと、個人で仕込むもの。Claude Code を中心に Codex と Copilot も ／ 2026-09 時点

Givery

チームの大元に仕込む（Claude Code、効く順）

施策	仕込む場所	効き方	注意
使えるモデルを絞る	managed の `availableModels` と `enforceAvailableModels`、個別に止めるなら `deniedModels`	Opus や Fable を既定のまま使い続ける事故を止める	managed の `model` は初期値だけ。/model で変えられる
既定を Sonnet 5.5 に	managed の `model: "sonnet"`。v2.1.284 から Sonnet 5.5 を指す	既定の Opus 5.5 より単価が半分。Sonnet 5 より1タスク最大30%安い	Bedrock などでは sonnet が 4.5 を指す。ID で書く
effort の上限	managed の `maxEffortLevel`	max と xhigh の常用を止める。複数あれば最低値が勝つ	思考のトークンは出力単価で課金される
サブエージェントを安く	env の `CLAUDE_CODE_SUBAGENT_MODEL=haiku` と `CLAUDE_CODE_SUBAGENT_MODEL_FORCE=1`	調べ役、チームメイト、ワークフローも同じモデルに	FORCE がないとエージェント定義の model が勝つ
Fast mode を封じる	Team と Enterprise は既定で無効。managed で `fastMode: false`	Opus 5.5 の $8 / $40 の割増を防ぐ	プランの残量があっても usage credits から引かれる
支出の上限	Team と Enterprise は Admin settings > Usage で組織、グループ、個人。Console は workspace ごと	枠を超えた追加課金を止める	超えると翌請求期間まで Claude、Cowork、Code が停止
見える化	managed の env で OpenTelemetry を配り、`OTEL_RESOURCE_ATTRIBUTES` に部署タグ	`claude_code.cost.usage` を人と部署で集計できる	リポジトリの settings からは有効にできない
契約単価で表示	managed の `modelPricing`	/usage、ステータスライン、OTel の金額を契約単価に	表示が変わるだけで、請求額は変わらない
リポジトリの共通設定	CLAUDE.md は200行未満にし、手順はスキルへ。`# Compact instructions` を書く	毎ターン読み直す量が減る	型付き言語は code intelligence プラグインも効く
テスト出力を絞る	`.claude/settings.json` の PreToolUse フック	失敗行だけを渡す。数万トークンが数百に	公式の costs ページにそのまま使える例がある

managed-settings.json の例

送り先の `OTEL_EXPORTER_OTLP_ENDPOINT` も env に足す

```
{
  "model": "sonnet",
  "availableModels": ["sonnet", "haiku", "opus"],
  "enforceAvailableModels": true,
  "maxEffortLevel": "high",
  "fastMode": false,
  "env": {
    "CLAUDE_CODE_SUBAGENT_MODEL": "haiku",
    "CLAUDE_CODE_SUBAGENT_MODEL_FORCE": "1",
    "CLAUDE_CODE_ENABLE_TELEMETRY": "1",
    "OTEL_METRICS_EXPORTER": "otlp",
    "OTEL_RESOURCE_ATTRIBUTES": "team.id=web"
  }
}
```

管理設定の配り方

ファイル Mac は `/Library/Application Support/ClaudeCode/`、Linux は `/etc/claude-code/` に managed-settings.json。Windows は `C:\Program Files\ClaudeCode\`

管理画面 Team と Enterprise は Owner が Admin Settings > Claude Code > Managed settings から配る。起動時と毎時に取りに行く

効かない例 利用者が独自の `ANTHROPIC_BASE_URL` などを export していると、管理画面の設定を取りに行かない

Codex と Copilot の大元

Codex の requirements.toml 利用者が上書きできない制約。承認、サンドボックス、Web 検索、MCP の許可リストを縛る。配布は Codex Policies 画面か MDM

Codex の managed_config.toml config.toml と同じキーの既定値を配る。利用者は上書きできる

ChatGPT Business Workspace settings > Billing で、シートの種類ごと、個人ごとに月のクレジット上限をかける

Copilot の料金 6/1 から従量課金。Business は1人月 $19、Enterprise は $39 で、同額の AI クレジット込み。コード補完は消費しない

Copilot の予算 組織やコストセンターの予算は「上限で利用を止める」が既定でオフ。作るたびに ON にする。個人の予算は常に止まる

Copilot のモデル 組織の Settings > Copilot > Models で、追加料金のモデルを個別に無効にする

CI、ゲートウェイ、社内アプリ

CI の `claude -p` `--max-turns` と `--max-budget-usd`、ジョブの timeout。設定を読ませないなら `--setting-sources ""` と `--strict-mcp-config`

ゲートウェイ LiteLLM でキーごとに支出を追い、予算をかける。公式は利用例として挙げつつ、Anthropic とは無関係で監査も受けていないと明記

ゲートウェイの落とし穴 cache_control を落とす作りだと、毎ターン全履歴を定価で払う

API のキャッシュ 読み込みは入力の0.1倍（Opus 5.5 は 0.05倍）。5分の書き込み（1.25倍）は1回、1時間（2倍）は2回読めば元が取れる

Batch 半額。多くは1時間以内に返り、24時間で期限切れ。急がない一括処理に

レート制限の目安 1人あたり TPM は、20〜50人で 50k〜75k、100〜500人で 15k〜20k（公式の推奨）

個人で仕込む（設定と習慣）

`/clear` 無関係な作業に移るたび。費用はゼロ。先に `/rename` しておけば `/resume` で戻れる

`/compact` 残す観点 要約するために会話全体を読むので、それ自体が大きなリクエスト。続きが要らないなら `/clear`

`/usage` スキル、MCP、サブエージェント別の割合とキャッシュ命中率。無駄が1割を超えると警告が出る

`/context` 何が文脈を食っているかを見る。使わない MCP は `/mcp` で切り、`gh` などの CLI で済ませる

モデルと effort 普段は Sonnet 5.5、単純なサブエージェントは `model: haiku`。浅い作業は effort を1段下げる

休憩明け キャッシュが切れて全履歴を読み直す。寿命はサブスク1時間、usage credits を使っている間は5分

定期タスク `/loop` などは放置中も毎回全文脈を送る。`/usage` の Loops 行で重いものを探す

依頼の書き方 「auth.ts の login に入力検証」のように場所を名指す。大きい作業は Plan mode から

Codex `model_verbosity = "low"`、`project_doc_max_bytes` で AGENTS.md の読込に上限。軽い作業は Luna のプロファイルで

OSS（チーム向け）

BerriAI/litellm ★59.8k。キーごとの予算、支出ログ、管理画面を持つゲートウェイ

ccusage/ccusage ★18.8k。Claude Code の記録から日別、月別、セッション別の費用を出す `npx ccusage`

getagentseal/codeburn ★11.3k。41ツールの記録を読み、費用をツール、モデル、プロジェクト、タスク別に `npx codeburn`

OSS（個人向け）

mksglu/context-mode ★24.2k。ツールの出力を文脈の外に置く。315KB が 5.4KB（98%減）

colbymchenry/codegraph ★72.3k。コードのグラフで探索を減らし、トークン62%減、費用44%減。ただし終了時の文脈は約8割多く残る

MinishLab/semble ★6.2k。grep と読み込みの代わりのコード検索。トークンは約99%少ない

DietrichGebert/ponytail ★147k。作るコードを減らすスキル。実測でトークン22%減、費用20%減

steipete/CodexBar ★22.0k。各社の利用上限とリセットまでの時間をメニューバーに出す

sirmalloc/ccstatusline ★13.1k。ステータスラインに文脈の使用量とトークンを出す

ollaya-dev/ollaya 判定だけの処理を LLM から外し、手元の判定モデルで返す

Tips・注意

企業導入の平均は1人・稼働日あたり約$13、月$150〜250で、9割が1日$30未満（公式）。まず少人数で基準を取る。

出力を短くする caveman は、ponytail の比較ではトークン7%増、費用3%増だった。費用の大半は毎ターン読み直す入力なので、削るなら文脈から。

設定を読ませない `claude -p` で、1件約8万トークンが約1.2万トークンになった（この Mac の夜間採点で実測）。

Haiku 4.5 の退役は早くて 10/15（公式）。Haiku 5.5 は数週間のうちに出る。haiku 固定の設定は差し替えに備える。

Sources（公式・一次情報 / クリックで開く）

1. Manage costs
2. Model configuration
3. Sonnet 5.5 の発表
4. モデルの一覧と退役予定
5. Managed settings
6. Server-managed settings
7. Subagents
8. Fast mode
9. Monitoring usage
10. LLM gateway
11. Prompt caching（API）
12. Batch processing
13. Team・Enterprise の支出上限
14. Codex の管理設定
15. ChatGPT Business のクレジット
16. Copilot の従量課金
17. Copilot の予算
18. Copilot のモデル管理
19. ponytail のベンチマーク