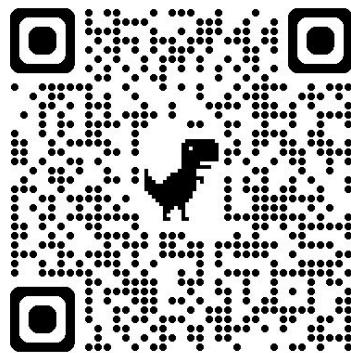




--dangerously-skip-permissions

の裏側で何が起きているのか



Claude Code パーミッションシステム Deep Dive
@yujiteshima

「許可疲れ」という問題



Claude Code を使っていると許可プロンプトが頻繁に出る



--dangerously-skip-permissions で全てスキップできる



でも本当にそれでいいのか？



このフラグの裏側を理解し、より良い選択肢を考える

ツール(Tools)とは

Claude Code が外部操作を行うための機能

ツール	機能	許可
Read	ファイル読み取り	不要
Glob/Grep	検索	不要
Edit	ファイル編集	必要
Write	ファイル作成	必要
Bash	コマンド実行	必要
WebFetch	URL取得	必要

パーミッション設定 =「どのツールの、どの操作を許可 /拒否するか」

設定ファイルの種類と場所

Enterprise Managed

/etc/clause-code/managed-settings.json

組織ポリシー(上書き不可)

User Settings

~/.clause/settings.json

ユーザー全体の設定

Project Shared

.clause/settings.json

チーム共有(git管理)

Project Local

.clause/settings.local.json

個人用(gitignore)

Runtime State

~/.clause.json

「Yes, always」の記録

設定の優先順位 (Precedence)

上位が下位を上書きする

1 Enterprise Managed

最高・上書き不可

2 CLI Arguments

--allowedTools など

3 Local Project

.claude/settings.local.json

4 Shared Project

.claude/settings.json

5 User Settings

~/.claude/settings.json

優先順位で起きうる問題



プロジェクト設定が User 設定を上書きする

シナリオ:

自分の User 設定で Bash(rm -rf:*) を deny している

クローンした OSS の .claude/settings.json に allow: ["Bash(rm:*)"] がある

→ プロジェクト設定が優先され、自分の deny が無効化される

```
~/.claude/settings.json → deny: ["Bash(rm -rf:*)"]
.claude/settings.json   → allow: ["Bash(rm:*)"] ← こちらが優先
```

対策:

- 信頼できないリポジトリの .claude/ を確認する
- Hooks で独自のチェックを追加する

パーミッション評価フロー

Claude がツールを使おうとしたとき、以下の順序で評価される

1

Hooks (PreToolUse)

カスタムスクリプト

2

Deny Rules

拒否ルール(最優先)

3

Allow Rules

許可ルール

4

Ask Rules

確認要求ルール

5

Permission Mode

動作モード

6

ユーザーに確認

プロンプト表示

重要: Deny は Allow より優先 / Hooks は Deny より先に評価

Hooks とは

ツール実行の前後にユーザーが設定するカスタムスクリプト(デフォルトでは未設定)

主な Hook の種類

PreToolUse

ツール実行前(許可/拒否を判定)



deny ルールより先に評価される

PostToolUse

ツール実行後(結果の検証)



bypassPermissions でも動作する

SessionStart

セッション開始時



任意のロジックを実装可能

```
{  
  "hooks": {  
    "PreToolUse": [{  
      "matcher": "Bash",  
      "hooks": [{ "type": "command", "command": "check.sh" }]  
    }]  
  }  
}
```

Permission Modes

モード	動作	リスク
<code>default</code>	初回使用時に許可を求める	低
<code>acceptEdits</code>	ファイル編集は自動承認、Bash は確認	中
<code>plan</code>	実行なし、分析のみ	低
<code>bypassPermissions</code>	全ての許可をスキップ	高

切り替え方法：

- CLI: --permission-mode acceptEdits
- セッション中: Shift+Tab でトグル
- 設定: "defaultMode": "acceptEdits" in settings.json

--dangerously-skip-permissions の正体

このフラグは bypassPermissions モードを有効化するだけ

何が起きるか:

-  全てのツール使用が自動承認
-  Bash コマンドも確認なしで実行
-  ファイルの読み書きも自動許可
-  サブエージェントもこのモードを継承
-  ただし Hooks は引き続き実行される

推奨: Docker コンテナ(ネットワーク分離)内でのみ使用

deny ルールの落とし穴 (1/2)

⚡️ プレフィックスマッチの限界

```
{ "deny": [ "Bash(git commit:*)" ] }
```

回避されるパターン：

```
git commit -m "msg"      # スペース2つ  
/usr/bin/git commit     # フルパス  
cd repo && git commit  # チェイン  
git -c user.name=x commit # オプション先
```

⚡️ 別コマンドという抜け穴

git commit を deny しても：

```
gh pr create --fill      # gh 経由  
gh api repos/.../commits # API 直叩き
```

deny ルールの落とし穴 (2/2)

curl で git 操作をバイパス

```
{ "deny": [ "Bash(git commit:*)", "Bash(git push:*)", "Bash(gh:*)" ] }
```

curl で GitHub API を直接叩けばコミットできる：

```
# コミット作成
curl -X POST https://api.github.com/repos/OWNER/REPO/git/commits \
  -H "Authorization: token $GITHUB_TOKEN" \
  -d '{"message": "commit via API", ...}'

# ref 更新(push 相当)
curl -X PATCH .../git/refs/heads/main -d '{"sha": "...."}'
```

攻撃チェーン: git deny → gh deny → curl API → コミット成功

curl も deny したら？ → まだ回避できる：

```
wget --post-data='...' https://api.github.com/...    # wget
python -c "import requests; requests.post(...)"      # Python
node -e "fetch('https://...', { method: 'POST' })"  # Node.js
```

deny ルールの限界と対策

公式の警告：

"Bash permission patterns that try to constrain command arguments are fragile. Do not rely on them as a security boundary."

なぜこうなるのか：

- Bash の deny ルールはプレフィックスマッチ
- シェルは柔軟で、同じ目的を達成する方法が多数ある
- Claude は賢いので「別の方法」を思いつく

本当に防ぐには：

サンドボックス

api.github.com へのネットワーク自体を禁止

Hooks

入力を解析して GitHub API アクセスを検出・ブロック

環境変数を渡さない

GITHUB_TOKEN がなければ API 認証できない

Git hooks

リポジトリ側の pre-commit / pre-push で制御

コンテキスト圧縮とパーミッションの関係

Q: 長い会話でコンテキストが圧縮されたら、パーミッション設定も消える？

A: 消えない。パーミッションはコンテキストとは別管理。

コンテキストウィンドウ

- ・会話履歴
- ・CLAUDE.md の内容
- ・コードの内容

← 圧縮の対象

プロセスマモリ

- ・settings.json の設定
- ・パーミッションルール

← 圧縮されない

- ・settings.json は起動時にファイルから読み込まれる
- ・パーミッションルールはプロセスのメモリに保持
- ・LLM のコンテキストウィンドウには入らない
- ・ツール実行のたびにメモリ上のルールでチェック

CLAUDE.md と settings.json の違い

項目	CLAUDE.md	settings.json
目的	Claude への指示・コンテキスト	ツール許可・動作設定
保存場所	コンテキストに注入	プロセスマモリに保持
圧縮の影響	受けける可能性あり	受けない
強制力	お願い(破られる可能性)	技術的な制約

重要:

- CLAUDE.md に「rm -rf しないで」→ 長い会話で忘れる可能性
- settings.json の deny: ["Bash(rm -rf:*)"] → 技術的にブロック
- セキュリティ上重要なルールは settings.json に書く

許可疲れへの処方箋: 設定を育てる



YOLO(全許可)と 毎回確認 の間を目指す

Step 1

まず観察

default モードで使い、よく許可するコマンドを把握

Step 2

安全なものから allow に追加

cat, ls, grep, git status, git diff など

Step 3

プロジェクト固有を追加

npm run lint, npm run test など

Step 4

「Yes, always」を活用

~/.claude.json に蓄積 → 自然と設定が育つ

おすすめ設定テンプレート

~/.claude/settings.json(ユーザー全体)

```
{ "permissions": {  
  "allow": [  
    "Bash(cat:*)", "Bash(ls:*)",  
    "Bash(git status)",  
    "Bash(git diff:*)"  
  ],  
  "deny": [  
    "Bash(rm -rf:*)",  
    "Read(../.env)"  
  ]  
}
```

.claude/settings.json(プロジェクト)

```
{ "permissions": {  
  "allow": [  
    "Bash(npm run lint:*)",  
    "Bash(npm run test:*)",  
    "Bash(npm run build)"  
  ]  
}
```

ポイント:

- ・読み取り系(cat, ls, git status)は安全なので allow
- ・危険な操作(rm -rf, sudo)と機密ファイル(.env)は deny
- ・プロジェクト固有のスクリプトは別ファイルで管理

より安全な自動化: サンドボックス



OS レベルでコマンドの実行環境を隔離する機能

ファイルシステム : 許可されたディレクトリのみアクセス可能

ネットワーク : 許可されたドメインのみ接続可能

実装 : macOS: Seatbelt / Linux: bubblewrap

```
{ "sandbox": {  
    "enabled": true,  
    "autoAllowBashIfSandboxed": true  
} }
```

メリット

- ✓ サンドボックス内なら Bash 自動許可
- ✓ 許可疲れを軽減
- ✓ セキュリティ維持

YOLO の安全版 = サンドボックス + autoAllowBashIfSandboxed

まとめ (Key Takeaways)

- ツールとは Claude が外部操作を行う機能
- dangerously-skip-permissions = bypassPermissions モード
- 設定は5層の階層構造、プロジェクトが User より優先
- 評価順序は Hooks → deny → allow → ask → mode
- deny ルールには限界がある(イタチごっこ)
- パーミッションはコンテキスト圧縮の影響を受けない
- CLAUDE.md と settings.json は別物
- 許可疲れには「設定を育てる」で対処
- より安全な選択肢: サンドボックス、Hooks

References



Settings

code.claude.com/docs/en/settings



IAM

code.claude.com/docs/en/iam



CLI Reference

code.claude.com/docs/en/cli-reference



SDK Permissions

code.claude.com/docs/en/sdk/sdk-permissions



Hooks

code.claude.com/docs/en/hooks



Sandboxing

code.claude.com/docs/en/sandboxing



Memory (CLAUDE.md)

code.claude.com/docs/en/memory



Security

code.claude.com/docs/en/security