



Hookイベント一覧

イベント	発火タイミングと制御
SessionStart	startup・resume・clear・compact・forkの時。stdoutは文脈へ入る
Setup	--init-onlyなどで起動した時。CIの準備用
UserPromptSubmit	送信直後、処理の前。exit 2 で入力を消して止める
UserPromptExpansion	コマンドがプロンプトに展開される時。exit 2 で止める
PreToolUse	実行の直前。exit 2 か deny で止め、入力も書き換えられる

イベント	発火タイミングと制御
PermissionRequest	許可を求める時。decision で allow・deny、exit 2 は無視
PermissionDenied	auto mode が拒否した時。retry: true で再試行を許す
PostToolUse・PostToolUseFailure	成功後・失敗後。実行済みなので exit 2 は Claude への指摘
PostToolBatch	並列の呼び出しの終了後。exit 2 で次の呼び出し前に止める
Notification	許可待ち・入力待ちなどの通知時。ブロック不可
MessageDisplay	応答の文を表示する間。置換文を返して表示だけ変える
SubagentStart・SubagentStop	子の起動時・終了時。Stop 側は exit 2 で続けさせる

イベント	発火タイミングと制御
TaskCreated・TaskCompleted	タスクの作成時・完了時。exit 2 で作成の取消、完了の阻止
Stop	応答を終える時。exit 2 で止めずに続けさせる
StopFailure	API エラーでターンが終わった時。出力も exit code も無視
TeammateIdle	agent team の仲間が待機に入る直前。exit 2 で止める
InstructionsLoaded	CLAUDE.md や rules を読み込んだ時。ブロック不可
ConfigChange	設定ファイルが変わった時。exit 2 で反映を止める
CwdChanged・DirectoryAdded	cd や /add-dir で作業場所が変わった時。env を読み直せる

イベント	発火タイミングと制御
FileChanged	matcher に書いたファイルが変わった時。ブロック不可
WorktreeCreate・WorktreeRemove	worktree の作成時・削除時。非0の exit で失敗させる
PreCompact・PostCompact	圧縮の直前・直後。Pre は exit 2 で圧縮を止める
PreModelSwitch・PostModelSwitch	モデル切替の直前・直後。Pre は exit 2 で切替を止める
Elicitation・ElicitationResult	MCP サーバーが入力を求めた時・応答の後。exit 2 で拒否
SessionEnd	セッション終了時。ブロック不可、持ち時間は1.5秒

settings.json 最小例

```
{  "hooks": {    "PreToolUse": [      {        "matcher": "Bash",        "hooks": [          {            "type": "command",            "if": "Bash(rm *)",            "command": "${CLAUDE_PROJECT_DIR}/.claude/hooks/guard.sh",            "args": [],            "timeout": 30          }        ]      }    ]  }
```

hook の type

command シェルを実行。stdin の JSON を読み、exit code と stdout で返す

http 入力 JSON を URL へ POST し、応答の本文を stdout と同じ形で読む

mcp_tool 接続済み MCP サーバーのツールを呼ぶ。起動直後の SessionStart と Setup では動かない

prompt モデル（既定は Haiku）が1回で判定し、ok と reason を JSON で返す。入力は \$ARGUMENTS で差し込む

agent Read や Grep を使うサブエージェントが確かめて判定する。実験的

matcher記法

"*"・"'"・省略 全件にマッチし、毎回発火する

完全一致 英数字・_・-・空白・,・| だけなら完全一致。| か、で複数並べる
Edit|Write Edit, Write

正規表現 ほかの記号を含むと JavaScript 正規表現の部分一致。全体一致なら ^ と \$ で囲む
^Notebook mcp_memory_.*

if フィールド handler ごとに権限ルールの構文で絞る。ツール系イベントだけで効く

"if": "Bash(git *)" "if": "Edit(*.ts)"

ツール以外の対象 SessionStart は開始の種類、Notification は通知の種類、FileChanged はファイル名
startup|resume|clear|compact|fork permission_prompt

出力制御（exit code ・ JSON）

exit 0 成功。stdout を JSON として読む。素の文が文脈に入るのは UserPromptSubmit ・ SessionStart など4種だけ

exit 2 ブロック。stderr が理由になり、JSON の allow でも覆らない

その他の非0 ブロックしないエラー。stderr の先頭行を出して続行する。JSON は読まれる

"continue": false Claude の処理を止める。stopReason にユーザー向けの理由を書く

"decision": "block" reason と組で使う。UserPromptSubmit ・ Post 系 ・ Stop など

hookSpecificOutput PreToolUse は permissionDecision (allow ・ deny ・ ask ・ defer) と updatedInput

additionalContext 文脈に文を足す。SessionStart ・ UserPromptSubmit ・ PreToolUse ・ Stop など

systemMessage ユーザーへの警告文。suppressOutput は今は効果なし

{ "hookSpecificOutput": { "hookEventName": "PreToolUse", "permissionDecision": "deny", "permissionDecisionReason": "理由" } }

ユースケース例

PreToolUse で危険コマンド遮断 matcher "Bash"。rm -rf を含むなら stderr に理由を書いて exit 2
CMD=\$(jq -r '.tool_input.command')
[[\$CMD == *"rm -rf"*]] && { echo NG >&2; exit 2; }

PostToolUse で整形 matcher "Edit|Write"。編集したファイルに prettier をかける
jq -r '.tool_input.file_path' |
xargs npx prettier --write

Notification で通知 入力待ちで macOS の通知を出す
osascript -e 'display notification "入力待ち" with title "Claude Code"'

SessionStart と CwdChanged で env CLAUDE_ENV_FILE に書いた export が以後の Bash に効く
direnv export bash >> "\$CLAUDE_ENV_FILE"

注意・Tips

無限ループ防止 Stop は8回続けてブロックすると打ち切り。stop_hook_active が true なら exit 0

権限との関係 PreToolUse の deny は bypassPermissions でも効く。allow を返しても deny と ask のルールは残る

\${CLAUDE_PROJECT_DIR} 起動時のプロジェクト root。worktree でも変わらず、今の場所は入力の cwd

timeout 既定 command ・ http ・ mcp_tool は600秒、prompt は30秒、agent は60秒。UserPromptSubmit は30秒

stdin JSON session_id ・ cwd ・ hook_event_name ・ permission_mode ・ effort ・ tool_name ・ tool_input など。jq で読む

登録場所 user ・ project ・ local の設定ファイル、managed、plugin の hooks/hooks.json、skill と subagent の frontmatter

実行のされ方 一致した hook は並列に走る。async: true で背景実行、ただし決定は返せない。disableAllHooks で一時停止、/hooks は読み取り専用の一覧

引数のクォート パスを渡すなら args を使う exec 形式が安全。シェル形式ならブレースホルダを二重引用符で囲む

任意コード実行の自覚 hook は自分の権限でコマンドを動かす。信頼できないものは登録しない