T03 # Permission Modes

Technical Beginner

Tool access control, from safest to most permissive

Available modes

Mode	Canonical name	Activation	Recommended use
Default	`default`	`--permission-mode default`	Daily development
Auto-accept edits	`acceptEdits`	`Shift+Tab`	Code reviews
Plan	`plan`	`Shift+Tab x2` or `/plan`	Analysis without modification
Don't Ask	`dontAsk`	`/permissions` or `permissions.allow` rules	Restrictive workflows, silent auto-deny if not pre-approved
Auto (AI classifier)	`auto`	`permissions` → `defaultMode: "auto"`	Long tasks, fewer interruptions
Full bypass	`bypassPermissions`	`--dangerously-skip-permissions`	Headless CI/CD, sandboxed
Fewer prompts	(none)	`/fewer-permission-prompts`	Bundled skill: adds an allowlist built from common read-only calls in your transcripts

CLI activation: `claude --permission-mode <mode>` accepts `default`, `plan`, `acceptEdits`, `dontAsk`, `auto`, `bypassPermissions`. **Persistent activation:** `permissions.defaultMode` key in `settings.json`. Since v2.1.283, interactive terminal and VS Code sessions start in `auto` unless `permissions.defaultMode` or `--permission-mode` overrides it.

Note: `--dangerously-skip-permissions` no longer prompts for writes to `.claude/skills/`, `.claude/agents/`, and `.claude/commands/` (v2.1.121); the rest of `.claude/` (settings, hooks) and `.git/` stay protected even under full bypass. The `auto` mode relies on a classifier (server-side by default; `CLAUDE_CODE_AUTO_MODE_MODEL` sets the client-side model), less friction, not a security boundary.

Tool whitelist

```
# Allow only specific tools
claude --tools "Read,Grep,Glob" --allowedTools "Read,Grep,Glob"

# Block specific tools
claude --disallowedTools "Bash,Write"

# Useful combinations
claude --allowedTools "Read,Edit,Bash(git *)"
```

Configuration in settings.json

```
{
  "permissions": {
    "allow": [
      "Bash(git log*)",
      "Bash(npm test*)",
      "Read",
      "Edit"
    ],
    "deny": [
      "Bash(rm*)",
      "Bash(sudo*)"
    ]
  }
}
```

Permission hierarchy

Permissions accumulate and are inherited in this order:

1. `~/.claude/settings.json`: global user
2. `.claude/settings.json`: project (shared)
3. `.claude/settings.local.json`: project (local, gitignored)
4. CLI flags: session only
5. Managed policy: organization constraints

Deny rules from any applicable scope take priority over allow rules.

Glob patterns for bash

```
# Allow git only
"Bash(git *)"

# Allow npm test and build
"Bash(npm test*)", "Bash(npm run build*)"

# Allow file reading
"Bash(cat *)", "Bash(ls *)"
```

Best practices

CI/CD: Pre-approve the tools needed for a headless task and restrict availability with `--tools`. If bypass is necessary, use an isolated environment with restricted credentials and network access.

Sensitive projects: Restrict Bash tools with precise globs in `.claude/settings.json`. Commit shared defaults; use managed policy for constraints developers must not override.

Audit: Inspect the local session transcripts under `~/.claude/projects/`. Custom hook logs need their own configured directory; transcripts are not a tamper-proof audit trail.