How to Claude Code Like a Pro

A comprehensive guide distilled from the Claude Code team's own workflows and official documentation.

---

1. Run Multiple Sessions in Parallel

The single biggest productivity unlock. Spin up 3-5 git worktrees, each running its own Claude session.

Setup with Git Worktrees

```
# Create a worktree with a new branch
git worktree add ../project-feature-a -b feature-a

# Create a worktree from an existing branch
git worktree add ../project-bugfix bugfix-123

# Run Claude in each worktree
cd ../project-feature-a && claude
cd ../project-bugfix && claude

# List and clean up worktrees
git worktree list
git worktree remove ../project-feature-a
```

Pro tips

- Name worktrees descriptively so you know which task each one is for.
- Set up shell aliases (`za`, `zb`, `zc`) to hop between worktrees in one keystroke.
- Keep a dedicated "analysis" worktree that's only for reading logs and running queries.
- Remember to install dependencies (`npm install`, etc.) in each new worktree.
- All worktrees share the same Git history and remote connections.

---

2. Master Plan Mode

Start every complex task in plan mode. Pour your energy into the plan so Claude can one-shot the implementation.

How to enter Plan Mode

- **During a session**: Press `Shift+Tab` to cycle through permission modes until you see `plan mode on`.
- **Start in plan mode**: `claude --permission-mode plan`
- **Headless plan mode**: `claude --permission-mode plan -p "Analyze the auth system and suggest improvements"`

Advanced planning strategies

- Have one Claude session write the plan, then spin up a second Claude to review it as a staff engineer.
- The moment something goes sideways, switch back to plan mode and re-plan. Don't keep pushing.
- Tell Claude to enter plan mode for verification steps too, not just for the build.
- Press `Ctrl+G` to open the plan in your default text editor for direct editing.

Make plan mode the default

```
// .claude/settings.json
{
  "permissions": {
    "defaultMode": "plan"
  }
}
```

---

3. Invest in Your CLAUDE.md

Your `CLAUDE.md` is persistent memory. After every correction, end with: **"Update your CLAUDE.md so you don't make that mistake again."** Claude is very good at writing rules for itself.

Best practices

- Ruthlessly edit your `CLAUDE.md` over time. Keep iterating until Claude's mistake rate drops.
- Maintain a notes directory for every task/project, updated after every PR. Point `CLAUDE.md` at it.
- Include coding conventions, project-specific terms, and patterns Claude should follow.
- Use `@` references in prompts to include `CLAUDE.md` from specific directories.

CLAUDE.md locations

Location	Scope
`~/.claude/CLAUDE.md`	All your projects (personal)
`CLAUDE.md` in project root	This project only
`CLAUDE.md` in subdirectories	Auto-discovered when working in those dirs

---

4. Create Custom Skills

If you do something more than once a day, turn it into a skill or slash command.

Create a skill

```
mkdir -p ~/.claude/skills/my-skill
```

Create `~/.claude/skills/my-skill/SKILL.md`:

```
---
name: my-skill
description: What this skill does and when to use it
---

Your instructions here...
```

Skill locations

Location	Scope
`~/.claude/skills/<name>/SKILL.md`	All your projects (personal)
`.claude/skills/<name>/SKILL.md`	This project only

Skill ideas from the team

- `/techdebt`: Run at the end of every session to find and kill duplicated code.
- `/context-dump`: Sync 7 days of Slack, GDrive, Asana, and GitHub into one context dump.
- `/fix-issue`: Takes a GitHub issue number and fixes it end-to-end.
- `/deploy`: Deploy with `disable-model-invocation: true` so only you can trigger it.
- `/pr-summary`: Use `!`gh pr diff`` to inject live PR data.

Key frontmatter options

Field	Purpose
`description`	Helps Claude decide when to auto-load the skill
`disable-model-invocation: true`	Only you can trigger it (e.g., deploy, commit)
`user-invocable: false`	Only Claude can trigger it (background knowledge)
`context: fork`	Run in an isolated subagent
`allowed-tools`	Restrict which tools the skill can use

String substitutions

- `$ARGUMENTS` - All arguments passed to the skill
- `$ARGUMENTS[0]`, `$1` - Positional arguments
- `${CLAUDE_SESSION_ID}` - Current session ID
- `!`command`` - Inject shell command output before Claude sees the prompt

---

5. Let Claude Fix Bugs Autonomously

Claude fixes most bugs by itself. Minimize context switching.

- **From Slack**: Enable the Slack MCP, paste a bug thread, and say "fix."
- **From CI**: Say "Go fix the failing CI tests." Don't micromanage how.
- **From logs**: Point Claude at docker logs to troubleshoot distributed systems.
- **From errors**: Just paste the error or say `I'm seeing an error when I run npm test`.

---

6. Level Up Your Prompting

Challenge Claude

- "Grill me on these changes and don't make a PR until I pass your test."
- "Prove to me this works" -- have Claude diff behavior between main and your feature branch.

Push for quality

- After a mediocre fix: "Knowing everything you know now, scrap this and implement the elegant solution."
- Write detailed specs and reduce ambiguity before handing work off.

Use `@` references

```
> Explain the logic in @src/utils/auth.js
> What's the structure of @src/components?
> Compare @file1.js and @file2.js
```

---

7. Optimize Your Terminal and Environment

Recommended terminal

The team loves Ghostty for its synchronized rendering, 24-bit color, and proper unicode support.

Status line

Use `/statusline` to customize your status bar -- always show context usage and current git branch.

Line breaks in prompts

- Type `\` then Enter for a newline.
- `Shift+Enter` works natively in iTerm2, WezTerm, Ghostty, and Kitty.
- Run `/terminal-setup` in other terminals (VS Code, Alacritty).

Voice dictation

Hit `fn` twice on macOS. You speak 3x faster than you type, and prompts get way more detailed.

Color-code your sessions

Name and color-code terminal tabs (or use tmux) -- one tab per task/worktree.

Vim mode

Enable with `/vim` or via `/config`. Supports mode switching, navigation, editing, yank/paste, text objects, and more.

Notifications

Set up iTerm2 notifications (Preferences > Profiles > Terminal > "Silence bell") or use custom notification hooks so you never miss when Claude finishes.

---

8. Use Subagents

Quick usage

- Append `"use subagents"` to any request where you want Claude to throw more compute at the problem.

- Offload individual tasks to subagents to keep your main agent's context window clean and focused.

Auto-approve safe actions with hooks

Route permission requests to a model via a hook -- let it scan for attacks and auto-approve the safe ones:

```
{
  "hooks": {
    "PermissionRequest": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Evaluate if this tool use is safe: $ARGUMENTS. Check
for destructive commands, sensitive file access, or network calls to
untrusted domains.",
            "timeout": 30
          }
        ]
      }
    ]
  }
}
```

Built-in subagent types

- **Explore**: Read-only codebase exploration (Glob, Grep, Read)
- **Plan**: Architecture and planning
- **Bash**: Command execution
- **general-purpose**: Full tool access

Custom subagents

Create in `.claude/agents/` for team sharing. Define when Claude should use them, which tools they access, and their system prompt.

---

9. Use Claude for Data and Analytics

- Ask Claude to use the `bq` CLI (BigQuery), `psql`, or any database CLI to pull and analyze metrics.
- Build a BigQuery/database skill and commit it so the whole team can query from Claude.

- Works for any database with a CLI, MCP, or API.

---

10. Learn with Claude

- **Explanatory mode**: Enable "Explanatory" or "Learning" output style in `/config` to have Claude explain the *why* behind changes.
- **Visual presentations**: Have Claude generate interactive HTML presentations explaining unfamiliar code.
- **ASCII diagrams**: Ask Claude to draw diagrams of protocols and codebases.
- **Spaced-repetition skill**: Build a skill where you explain your understanding, Claude asks follow-ups to fill gaps, and stores the result.

---

11. Hooks: Automate Your Workflow

Hooks are shell commands or LLM prompts that execute at specific lifecycle points.

Hook events

Event	When it fires
`SessionStart`	Session begins or resumes
`UserPromptSubmit`	Before Claude processes your prompt
`PreToolUse`	Before a tool call (can block it)
`PermissionRequest`	When a permission dialog appears
`PostToolUse`	After a tool call succeeds
`Stop`	When Claude finishes responding
`SessionEnd`	When session terminates

Example: Block destructive commands

```
{
  "hooks": {
    "PreToolUse": [
      {
```

```
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": ".claude/hooks/block-rm.sh"
          }
        ]
      }
    ]
  }
}
```

Example: Auto-run tests after file changes

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": ".claude/hooks/run-tests.sh",
            "async": true,
            "timeout": 300
          }
        ]
      }
    ]
  }
}
```

Hook types

- `command`: Run a shell script. Receives JSON on stdin, returns decisions via exit codes and stdout.
- `prompt`: Single-turn LLM evaluation (yes/no decisions).
- `agent`: Multi-turn subagent that can read files and search code before deciding.

Where to define hooks

Location	Scope
`~/.claude/settings.json`	All your projects
`.claude/settings.json`	This project (committable)

Location	Scope
`.claude/settings.local.json`	This project (gitignored)

Manage interactively with `/hooks`.

---

12. Session Management

Resume sessions

- `claude --continue` -- Resume the most recent conversation.
- `claude --resume` -- Open the session picker.
- `claude --resume auth-refactor` -- Resume by name.
- `claude --from-pr 123` -- Resume a session linked to a PR.

Name your sessions

```
> /rename auth-refactor
```

Then resume later with `claude --resume auth-refactor`.

Session picker shortcuts

Key	Action
`P`	Preview session
`R`	Rename session
`/`	Search/filter
`A`	Toggle all projects
`B`	Filter by current branch

---

13. Use Claude as a Unix Utility

Pipe data through Claude

```
cat build-error.txt | claude -p 'concisely explain the root cause' >
```

```
output.txt
```

Add Claude to your build scripts

```
{
  "scripts": {
    "lint:claude": "claude -p 'you are a linter. look at the changes vs. main and report any typo issues. report filename and line number on one line, description on the second.'"
  }
}
```

Output formats

- `--output-format text` -- Plain text (default)
- `--output-format json` -- Full conversation log with metadata
- `--output-format stream-json` -- Real-time streaming JSON

---

14. Extended Thinking

Enabled by default, reserves up to 31,999 tokens for Claude to reason through complex problems.

When to use

- Complex architectural decisions
- Challenging bugs
- Multi-step implementation planning
- Evaluating tradeoffs

Configure

- **Toggle**: `Option+T` (macOS) / `Alt+T`
- **Global default**: `/config`
- **Limit budget**: `export MAX_THINKING_TOKENS=10000`
- **View thinking**: `Ctrl+O` to toggle verbose mode

Quick Reference Card

Action	How
New worktree	`git worktree add ../name -b branch`
Plan mode	`Shift+Tab` or `claude --permission-mode plan`
Name session	`/rename my-session`
Resume session	`claude --resume my-session`
Toggle thinking	`Option+T`
Verbose mode	`Ctrl+O`
Vim mode	`/vim`
View skills	"What skills are available?"
Manage hooks	`/hooks`
Status line	`/statusline`
Compact context	`/compact`
Voice input	`fn` twice (macOS)