A RESOURCE FOR TEAMS WORKING WITH CODING AGENTS

Context management decision tree.

Fourteen questions between you and a cluttered context window.

Clear by default. Compact on purpose. Delegate the reading.

24	14	18
decisions, each with its evidence	questions in the tree	actions at its leaves

The context window is the resource to manage.

Claude Code's own best practices start from one constraint: the context window fills up fast, and performance degrades as it fills [S01]. Anthropic calls the effect context rot, after research that found recall worsening as input grows, without a fixed point where it starts [S06, S07]. The practical question is never "how full is too full" but "what should this context hold for the next step".

Claude Code gives you a dozen ways to answer that: clear, compact, rewind, a subagent, a fork, a side question, a notes file, a new session, and four places to keep knowledge out of the conversation. This guide puts them in one tree, with the reason and the source for each leaf. The kit ships the tree as data, so you can walk it in a terminal, check it in CI and regenerate the diagram from the same file.

Version 1.0 / Sources checked 24 September 2026

Field Guide 31 of the Design × AI series, with 28 (CLAUDE.md), 29 (skills) and 30 (verification). Commands checked against the Claude Code docs (behavior up to v2.1.281); `/subtask` needs v2.1.212 or later. Diagram rendered with Mermaid 11.15.0; CLI tested on Node.js 22.22.

Practical guidance, not a standard. Command behavior is from the Claude Code docs; the order of the questions is this guide's recommendation, and your own sessions will sometimes justify letting context accumulate. Prepared with AI assistance and edited by hand.

START HERE

Know the tools before the tree.

Every leaf of the tree is one of these. Each costs something different, and some cost nothing.

Tool	What it does	Cost
`/clear`	Empties the conversation	Nothing
`/compact [focus]`	Replaces it with a summary	One request that reads it all
`/rewind`	Restores or summarizes from a checkpoint	Nothing
`/btw`	Answers outside the history	Nothing added
Subagent	Works in its own window	Only the summary returns
Fork (`/subtask`)	A subagent with the whole conversation	Shares the prompt cache
Notes file	State on disk for the next session	What you read back
CLAUDE.md, rules, skills, hooks	Knowledge outside the conversation	Always, by path, on use, zero

From the Claude Code commands, costs, context window and subagent docs [S01 to S05, S09].

In the middle of a session

Run `node scripts/walk-tree.mjs` and answer by number, or read Appendix A.

Setting up a team

Section 05, then Field Guides 28, 29 and 30.

Long-running work

Section 04, with `NOTES.md` and `task-brief.md` from the kit.

Adapting the tree

Edit `decision-tree.json`, run `--check`, regenerate with `--mermaid`.

Label	Meaning
`DOCS`	Behavior stated in the Claude Code or Anthropic documentation.
`KIT CHECK`	Backed by a kit file or checked by `walk-tree.mjs`.
`PRACTICE`	A working method; the evidence line says what to look for.

SECTION 01

Reset between tasks.

The cheapest context management is starting clean. The tree asks first whether the next task belongs in this conversation at all.

- [ ] **Clear between unrelated tasks**

  D01 DOCS

  Unrelated history costs tokens on every message and crowds out the files the next task needs. `/clear` costs nothing; CLAUDE.md reloads.

  Evidence: A new task starts in a conversation that contains only that task.

  Manage context aggressively; clear between tasks. S01, S03.

- [ ] **Name it before you clear it**

  D02 DOCS

  If you will come back, `/rename` the session first and return with `/resume`. A named session is a branch; you do not have to drag it into the next task.

  Evidence: Sessions carry workstream names in the resume picker.

  Resume conversations; rename before clearing. S01, S03.

- [ ] **After two failed corrections, start over**

  D03 DOCS

  Failed approaches stay in context and pull the next attempt toward them. Write down what you learned, `/clear`, and send a better first prompt.

  Evidence: No session has three corrections on the same problem.

  Course-correct early; the correcting-over-and-over failure pattern. S01.

- [ ] **Rewind rather than argue**

  D04 DOCS

  If there was a good point before it went wrong, `/rewind` to it and restore the conversation, the code or both. Checkpoints track only Claude's own file edits; use git for the rest.

  Evidence: Rewinds are followed by a sharper prompt, not a repeat.

  Rewind with checkpoints. S01.

- [ ] **Side questions go to /btw**

  D05 DOCS

  A quick question about something already in context does not need to become part of it. `/btw` answers without adding to the history.

  Evidence: Lookups and clarifications do not appear in the main transcript.

  Side questions with /btw. S01, S05.

Clearing is free. Carrying stale context is not.

SECTION 02

Compact on purpose.

Automatic compaction keeps a session alive when the window fills. Compacting yourself, with a focus, keeps what you choose instead of what the summary guesses.

- [ ] **Compact with a focus before a new phase**

  D06 DOCS

  `/compact keep the contract decisions and the failing tests` before moving from exploration to implementation. The focus decides what the summary keeps.

  Evidence: Every manual compact carries instructions.

  When your context fills up. S02.

- [ ] **Summarize only part of the conversation**

  D07 DOCS

  `/rewind`, select a message, then Summarize from here or Summarize up to here. The part you still need stays in full.

  Evidence: Long exploratory stretches are summarized; the current work is not.

  Compact part of the conversation. S01, S02.

- [ ] **Know what survives compaction**

  D08 DOCS

  Project-root CLAUDE.md, unscoped rules and auto memory are re-read from disk. Path-scoped rules and nested CLAUDE.md files come back only when a matching file is read. Up to five recent files are re-read; skills keep their first 5,000 tokens.

  Evidence: After a compact, you check that rules you rely on have reloaded.

  What survives compaction. S02, S08.

- [ ] **Tell compaction what to keep**

  D09 DOCS KIT CHECK

  A Compaction section in CLAUDE.md (Field Guide 28) survives every compact. A SessionStart hook on the `compact` matcher can print `NOTES.md` back into context.

  Evidence: `settings.compaction.example.json` passes Field Guide 30's hook validator and the SchemaStore schema.

  Re-inject context after compaction. S02, S11.

- [ ] **Prefer clearing when continuity is not needed**

  D10 DOCS

  `/compact` is itself a large request: it reads the conversation it summarizes. If the next step needs state, not history, write notes and clear instead.

  Evidence: Compacts happen for continuity; resets happen through notes.

  Why usage climbs in a long session. S03.

A summary keeps what you ask for. Ask.

SECTION 03

Delegate the reading.

Research, logs and full test runs fill context with text you read once. Send them somewhere else and keep only the answer.

- [ ] **Reads you will not reference again go to a subagent**

  D11 DOCS

  A subagent explores in its own window and returns a summary. Use it for codebase research, log analysis and verbose test runs.

  Evidence: The main transcript shows the summary, not the files read.

  Use subagents for investigation; delegate verbose operations. S01, S03, S04.

- [ ] **Brief a subagent like a new colleague**

  D12 DOCS KIT CHECK

  It starts fresh: it loads CLAUDE.md but not this conversation, and Explore and Plan skip even CLAUDE.md. Restate the rules it must follow; the `task-brief.md` template has the fields.

  Evidence: Every delegation names the goal, the files, the constraints and what to return.

  What loads at startup. S04.

- [ ] **Fork when the side task needs the history**

  D13 DOCS

  `/subtask <task>` starts a fork that inherits the whole conversation, so it needs no brief, and its tool calls stay out of your window. It reuses the parent's prompt cache.

  Evidence: Forks are used where a brief would repeat the conversation.

  Fork the current conversation; v2.1.212 and later. S04.

- [ ] **Ask for a short answer back**

  D14 DOCS

  The value of a subagent is the distillation. Anthropic describes subagent summaries of often 1,000 to 2,000 tokens; say the length and shape you want.

  Evidence: Delegation prompts state the return format.

  Sub-agent architectures. S06.

- [ ] **Parallel work gets separate checkouts**

  D15 DOCS PRACTICE

  Tasks that cannot touch the same files can run in parallel in git worktrees, or subagents with `isolation: worktree`. Tasks that might touch the same files run in sequence; the essay's author found that slower but more reliable.

  Evidence: No two concurrent sessions edit the same checkout.

  Run multiple sessions; worktrees. S01, S04. The essay's experience.

Keep the answer, not the search.

SECTION 04

Keep state in files, not in the conversation.

A context window is working memory. Anything that has to outlast it belongs on disk, where the next session can read exactly what it needs.

- [ ] **The plan is a file, and a fresh session executes it**

  D16 DOCS

  For larger features, write the spec to a file, with the files involved, what is out of scope and an end-to-end check. Then start a new session to implement it.

  Evidence: A spec or plan file exists before implementation starts.

  Let Claude interview you; start a fresh session to execute. S01.

- [ ] **Notes hold state, not a transcript**

  D17 KIT CHECK PRACTICE

  Goal, decisions with reasons, what is done and verified, what is next, what failed. The code and git history hold the rest. Keep it under a screen.

  Evidence: The kit's `templates/NOTES.md`; the next session starts from it.

  Structured note-taking. S06.

- [ ] **Each verified task is a commit**

  D18 PRACTICE

  Commit when the checks pass (Field Guide 30). The history is the durable log of what was done and why, and it survives every reset.

  Evidence: One commit per task, each with a passing verification report.

  The essay's atomic commits; Field Guide 30, item V28.

- [ ] **One named session per workstream**

  D19 DOCS

  Resume a workstream with `claude --continue` or `/resume` instead of re-explaining it. Keep unrelated work out of it.

  Evidence: Session names map to workstreams.

  Resume conversations. S01.

What must survive the session goes to disk.

SECTION 05

Put repeated knowledge in the right layer.

If you explain the same thing every session, the conversation is the wrong place for it. Pick the layer by when it is needed [S09].

- [ ] **Every session: CLAUDE.md**

  D20 DOCS

  Loaded every session and re-read after compaction, so every line costs every request. Keep it under 200 lines (Field Guide 28).

  Evidence: Field Guide 28's linter passes.

  CLAUDE.md files. S08, S09.

- [ ] **Certain files: a path-scoped rule**

  D21 DOCS

  `.claude/rules/<topic>.md` with `paths` loads when Claude reads a matching file. It is summarized away on compaction and reloads on the next matching read.

  Evidence: File-specific guidance lives in rules, not in CLAUDE.md.

  Path-specific rules; what survives compaction. S02, S08.

- [ ] **Certain tasks: a skill**

  D22 DOCS

  Only the description is in context until the skill runs, so a long procedure costs almost nothing until needed (Field Guide 29).

  Evidence: Field Guide 29's validator passes on the skills folder.

  Skills load on demand. S09.

- [ ] **Every time, whatever Claude decides: a hook**

  D23 DOCS

  A hook runs at its lifecycle event and costs no context unless it returns output (Field Guide 30).

  Evidence: Guarantees are hooks; instructions are not relied on for them.

  Context cost by feature. S09.

- [ ] **Watch symptoms, not a percentage**

  D24 DOCS

  Neither the docs nor the research give a fill level where quality drops, and auto-compaction thresholds differ by model. Check `/context`, show usage in the status line, and act on symptoms: forgotten instructions, repeated mistakes.

  Evidence: The tree's context question asks about symptoms and points to `/context`.

  S01, S06, S07, S10.

The best context is the one that holds only the next step.

APPENDIX A

The tree, in Mermaid.

Generated by `node scripts/walk-tree.mjs --mermaid` from `decision-tree.json`, one line per answer. Paste it into any Mermaid renderer; the kit also ships `decision-tree.svg`.

decision-tree.mmd

```
flowchart TD
  related{"Related to this conversation?"} -->|no| come_back{"Come back to it later?"}
  related -->|yes| corrections{"Corrected twice or more?"}
  come_back -->|yes| rename_clear_do("/rename, then /clear")
  come_back -->|no| clear_do("/clear")
  corrections -->|yes| checkpoint{"Good point before it went wrong?"}
  corrections -->|no| heavy_read{"Many files or long output next?"}
  checkpoint -->|yes| rewind_do("/rewind")
  checkpoint -->|no| restart_do("/clear, better prompt")
  heavy_read -->|yes| needs_history{"Side task needs this conversation?"}
  heavy_read -->|no| filling{"Context filling up?"}
  needs_history -->|yes| fork_do("fork: /subtask")
  needs_history -->|no| subagent_do("subagent")
  filling -->|yes| keep{"What must the next step keep?"}
  filling -->|no| outlast{"Outlasts this session?"}
  keep -->|all| compact_do("/compact with a focus")
  keep -->|part| rewind_summarize_do("/rewind, summarize part")
  keep -->|state only| notes_clear_do("notes file, then /clear")
  outlast -->|yes| state_files_do("plan + notes + commits")
  outlast -->|no| repeat{"Re-explaining the same thing?"}
  repeat -->|yes| when_needed{"When is it needed?"}
  repeat -->|no| parallel{"Another task at the same time?"}
  when_needed -->|always| claude_md_do("CLAUDE.md")
  when_needed -->|certain files| rule_do(".claude/rules + paths")
  when_needed -->|certain tasks| skill_do("skill")
  when_needed -->|every time| hook_do("hook")
  parallel -->|yes| same_files{"Could they touch the same files?"}
  parallel -->|no| side_question{"A quick side question?"}
  same_files -->|yes| sequential_do("run them in sequence")
  same_files -->|no| worktrees_do("worktrees")
  side_question -->|yes| btw_do("/btw")
  side_question -->|no| continue_do("carry on, then verify")
```

Diamonds are questions, rounded boxes are actions (ids ending in `_do`), edge labels are answers. Parsed and rendered with Mermaid 11.15.0.

APPENDIX B

The tree as data, and the CLI that walks it.

Each node is a question with options or an action with steps, a reason and sources. The CLI walks it interactively, from a list of answers, or checks it. Output below is from the kit.

terminal

```
$ node scripts/walk-tree.mjs --check
14 questions, 18 actions, 18 routes, at most 8 questions per route. 0 error(s).

$ node scripts/walk-tree.mjs --answers 2,2,2,1,3
Is the next task related to what is already in this conversation?
  Yes, it continues this task
...
How much of the history does the next step need?
  Only the decisions and current state

=> Write the state to a file, then clear
   1. Ask Claude to update NOTES.md from the template: goal, decisions with
      reasons, what is done and verified, what is next.
   2. Commit what is finished.
   3. Run /clear and start with: read NOTES.md, then continue with the next step.
```

decision-tree.json (one question, one action)

```
"q-needs-history": {
  "type": "question",
  "short": "Side task needs this conversation?",
  "text": "Does that side task need what this conversation already knows?",
  "options": [
    { "label": "Yes, it would take too long to brief", "short": "yes", "next": "a-fork" },
    { "label": "No, a short brief is enough", "short": "no", "next": "a-subagent" }
  ]
},
"a-fork": {
  "type": "action",
  "short": "fork: /subtask",
  "title": "Fork the conversation for the side task",
  "do": ["Run /subtask followed by the side task.", "..."],
  "why": "A fork inherits the whole conversation, so it needs no briefing ...",
  "sources": ["https://code.claude.com/docs/en/sub-agents"]
}
```

`--check` enforces what the JSON Schema cannot: every `next` resolves, no cycles, every node reachable, every action sourced.

KEEP WITH THE TEAM'S WORKING AGREEMENT

Leave a context review record.

One record per retrospective on agent sessions. It shows which decisions the team makes by habit and which it should make on purpose.

Team / period	**Sessions reviewed**
Repeated corrections (count)	**Clears, compacts, rewinds**
Subagents and forks used	**Notes files in use**
Knowledge moved to CLAUDE.md or rules	**Procedures moved to skills**
Guarantees moved to hooks	**Tree changes (decision-tree.json diff)**

> **Change the tree when the team disagrees with it.**
> The JSON is yours. Edit the questions, run `--check` and regenerate the diagram.

SOURCES / MAINTENANCE

Keep the guide current.

Sources checked 24 September 2026. Command names change between releases (the fork command was `/fork` from v2.1.161 to v2.1.211); check `/help` in your version.

S01 / Claude Code docs, best practices
Context as the constraint; clear, rewind, compact, subagents, specs, parallel sessions, failure patterns.
https://code.claude.com/docs/en/best-practices

S02 / Claude Code docs, explore the context window
What survives compaction; acting before auto-compaction.
https://code.claude.com/docs/en/context-window

S03 / Claude Code docs, manage costs
Clear between tasks, compaction instructions, delegate verbose operations, why usage climbs.
https://code.claude.com/docs/en/costs

S04 / Claude Code docs, subagents
What loads at startup, forks and /subtask, worktree isolation.
https://code.claude.com/docs/en/sub-agents

S05 / Claude Code docs, commands
/clear, /compact, /rewind, /btw, /context, /rename, /resume, /subtask.
https://code.claude.com/docs/en/commands

S06 / Anthropic, effective context engineering for AI agents
Context rot, compaction, structured note-taking, subagent summaries (September 2025).
https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents

S07 / Chroma, Context Rot
18 models degrade non-uniformly as input grows; no single threshold (July 2025).
https://www.trychroma.com/research/context-rot

S08 / Claude Code docs, how Claude remembers your project
CLAUDE.md and rules loading; what reloads after /compact.
https://code.claude.com/docs/en/memory

S09 / Claude Code docs, extend Claude Code
Context cost by feature: CLAUDE.md, skills, subagents, hooks.
https://code.claude.com/docs/en/features-overview

S10 / Claude Code docs, model configuration
Default auto-compact thresholds per model and the /autocompact window.
https://code.claude.com/docs/en/model-config

S11 / Claude Code docs, hooks guide
Re-inject context after compaction with a SessionStart hook.
https://code.claude.com/docs/en/hooks-guide

Maintenance: recheck the command names and the compaction table at each Claude Code minor release; edit `decision-tree.json`, run `--check`, and regenerate `decision-tree.mmd` and the SVG. Update the PDF, HTML, Markdown and JSON together.