Phases of Using Claude Code

A structured workflow for getting the most out of Claude Code across a development task.

TL;DR

Phase	Name	One-liner
0	Setup	Run `/init`, install plugins, configure hooks and MCP servers.
1	Research	Gather context — fetch tickets, search the web, read project docs.
2	Plan	Define goals, success criteria, must-haves, and out of scope.
3	Execute	Implement the plan using skills, subagents, and hooks.
4	Review	Verify with manual inspection and review plugins.
5	Post-Execution	Update README, CLAUDE.md, and other project docs.
6	Commit & PR	Commit, push, and create a pull request.

Phase 0: Setup

One-time project initialization before any development work begins:

- **Run `/init`** — Bootstraps the project for Claude Code by generating a `CLAUDE.md` file with project conventions, build commands, test patterns, and coding standards.
- **Install plugins** — Browse the Claude Code marketplace and install plugins that match your workflow:
  - **Review plugins** (e.g., `pr-review-toolkit`) — Automated code review, silent failure detection, test coverage analysis.
  - **Development plugins** (e.g., `feature-dev`, `superpowers`) — Structured workflows for planning, TDD, debugging, and parallel execution.
  - **Commit plugins** (e.g., `commit-commands`) — Streamlined commit, push, and PR creation.
  - **Custom plugins** — Build your own for team-specific workflows using `plugin-dev`.
- **Configure hooks** — Set up `PreToolUse`, `PostToolUse`, or other event hooks to enforce guardrails (e.g., auto-lint on save, block dangerous commands).
- **Add MCP servers** — Connect external tools and services (e.g., Jira, Confluence, databases) via Model Context Protocol for richer context during sessions.

Goal: Equip the project so every future Claude Code session starts with the right context, tools, and guardrails in place.

Phase 1: Research

Gather all relevant context before writing any code. Claude Code can assist with:

- **Fetch ticket/issue** — Pull in the GitHub issue, Jira ticket, or task description so the full requirements are in context.
- **Web search** — Look up documentation, API references, or prior art relevant to the task.
- **Document search** — Read existing project docs, `CLAUDE.md`, `README.md`, and related source files to understand current state.

Goal: Ensure you and Claude Code share a complete understanding of the problem before planning begins.

Phase 2: Plan

Enter a prompt that clearly defines:

- **Goals** — What the task should accomplish.
- **Success criteria** — How you'll know it's done correctly.
- **Must-haves** — Non-negotiable requirements or constraints.
- **Out of scope** — What explicitly should *not* be done (prevents over-engineering).

Claude Code's **Plan Mode** (`EnterPlanMode`) is designed for this — it explores the codebase, designs an approach, and presents it for your approval before any code is written.

Phase 3: Execute Plan

Carry out the implementation with the help of plugins and tools:

- Use **skills** (e.g., `test-driven-development` , `feature-dev`) to follow structured workflows.
- Use **subagents** for parallelizable work (e.g., `dispatching-parallel-agents`).
- Use **hooks** for automated guardrails during execution (e.g., linting, formatting).
- Let Claude Code write the code, but contribute where meaningful decisions and trade-offs exist.

Goal: Translate the approved plan into working code.

Phase 4: Review

Verify the work through both manual inspection and automated tooling:

- **Manual review** — Read through the changes, test locally, confirm behavior matches expectations.
- **Plugin-assisted review** — Use agents like:
  - `code-reviewer` — Checks for bugs, logic errors, security vulnerabilities, and adherence to project conventions.
  - `silent-failure-hunter` — Identifies inadequate error handling and suppressed failures.
  - `pr-test-analyzer` — Evaluates test coverage quality and completeness.
  - `type-design-analyzer` — Reviews type design for encapsulation and invariant expression.
  - `comment-analyzer` — Checks that comments are accurate and maintainable.

Goal: Catch issues before they reach the main branch.

Phase 5: Post-Execution

Capture what was learned and update project documentation:

- **Update** `README.md` — Reflect any new features, setup changes, or usage instructions.
- **Update** `CLAUDE.md` — Record learnings, conventions, and context that will help future Claude Code sessions.
- **Update other docs** — Changelogs, API docs, architecture decision records, etc.

Goal: Ensure institutional knowledge is preserved for the next session (human or AI).

Phase 6: Commit, Push & Create PR

Finalize the work and prepare it for integration:

- **Commit** — Use structured commit messages that explain the *why*, not just the *what*.
- **Push** — Push the branch to the remote repository.
- **Create PR** — Generate a pull request with a clear summary, test plan, and context.

Claude Code's `/commit` skill handles all three steps, and the PR description is auto-generated from the commit history and changes.

Goal: Deliver the work in a reviewable, mergeable state.