Prompt Patterns for Agent Mode

A cheat sheet. *Six reusable shapes for the most common Agent jobs. Copy, fill in the blanks, run.*

01 Spec-first

When you know the behavior but not the code.

Shape: behavior + signature + invariants → "now build it."

```
> Build a function `parsePagination(req)`
that:
  - returns { page, limit } as integers
  - defaults: page=1, limit=20
  - throws on negative values
  - never returns limit > 100
  No tests yet. Just the function.
```

Watch out: leaving invariants implicit. "Reasonable defaults" is not a spec.

02 Failing test → fix

When the test exists but the code is wrong.

Shape: paste the failing test + error → "make this pass without changing the test."

```
> The test below fails. Make it pass.
  Do NOT modify the test.
  Do NOT add a new test.
  Touch only @src/auth.ts.
  [paste test + error output]
```

Watch out: letting Agent change the test to match the code. Always pin the test.

03 Refactor by intent

When you want shape changes without behavior changes.

Shape: the rule + the file + the boundary → "preserve behavior; show me the diff."

```
> Refactor @userService.ts: extract HTTP
calls into a separate
  `userClient.ts` module. Public exports must
not change.
  All existing tests must pass without
modification.
  Show me the diff before applying.
```

Watch out: overshoot. Without "preserve behavior," refactors become rewrites.

04 Migration

When the codebase has to move from A to B systematically.

Shape: before/after example + scope + acceptance → "do all of them."

```
> Migrate every `useEffect` in
@src/components/ that fetches data
  to `useQuery` from @tanstack/react-query.
  Pattern (before/after) below: [paste].
  Skip files in @src/components/legacy/. Run
`pnpm test` after.
```

Watch out: no example = no consistency. Always show the before/after pair.

05 Audit and report

When you want findings, not edits.

Shape: scope + criteria + format → "don't fix, just list."

```
> Read @src/api/. Find every endpoint that:
  - lacks input validation, OR
  - logs PII, OR
  - returns 500 on user error.
  Output: a table with file, line, issue,
severity. Don't edit.
```

Watch out: Agent will start fixing if you don't say "don't."

06 Debug with hypothesis

When you've already looked and have a guess.

Shape: symptom + your hypothesis + ask to confirm or refute.

```
> Symptom: requests time out under load.
  Hypothesis: connection pool starvation in
@src/db.ts.
  Confirm or refute by reading the code. If
confirmed, propose the
  smallest change. Don't apply yet.
```

Watch out: asking with no hypothesis turns into a long fishing expedition.

The four levers, in every prompt

Move them, the output moves.

1. **Scope.** Which files? @-mention them. Without scope, Agent reads too much or too little.
2. **Constraints.** What NOT to touch. Tests, public APIs, dependency files. Capitalised NOTs work.
3. **Format.** Diff, file, table, plan? Asking for the wrong shape wastes tokens.
4. **Stop condition.** "Run X and stop." "Apply once." Without it, Agent will keep going.

Meta-prompts that work

When stuck, ask the model to write the prompt.

`"Before answering, ask me three clarifying questions."`
Forces a Plan-mode-like pause without switching modes.

`"Walk me through how you'd approach this. Don't write code yet."`
Surfaces wrong assumptions before they become wrong code.

`"Write the prompt I should send next, given this conversation."`
When your own framing isn't landing, let the model rewrite it.

Re-use, don't re-invent. The patterns above stay the same; only the blanks change. Save your best ones in `.cursor/rules/` as Manual rules and pull them in with @-mention.