AY AUTOMATE — PLAYBOOK

Playbook of Tips For AI Coding #1

playbook of tips for ai coding

this is my simple guide. how i code with ai. how i keep it clear, fast, and real.

---

pattern 1 write things down first

ai is only as good as the context you give it.

- make one doc for each project goal, architecture, naming rules, priorities
- keep it short. update after big changes
- think of it like onboarding a new teammate

```
flowchart LR
  A[raw idea] --> B[questions]
  B --> C[tiny doc]
  C --> D[update after changes]
```

```
here’s the project. ask me simple questions. help me write a tiny doc with goals
```

---

pattern 2 plan before code

start with a chat, not code.

- agree on requirements and user flow
- pick a simple architecture
- split mvp vs nice to have

prompts

```
i want to build [idea]. ask me 5 questions about users, flows, and constraints.
```

```
based on this, suggest a simple architecture. db tables, api routes, ui pieces.
```

```
split features into mvp vs later. what's the smallest useful version?
```

flowchart LR

R[requirements chat] --> A[arch snapshot]

A --> M[mvp vs later]

```
```

---

pattern 3 build in small chunks

```
flowchart LR
  A[one chat = one feature] --> B[brief each chat]
  B --> C[start fresh when long]
```

one chat = one feature.

- short conversations stay focused
- start a new chat when the topic changes
- if a chat gets long, start fresh with a short brief

prompt to brief a new chat

```
i'm working on [feature]. current stack [stack]. goal [short goal]. relevant fi
```

---

pattern 4 use git like a seatbelt

```
flowchart LR
  A[new branch] --> B[test locally]
  B --> C[clear commit]
  C --> D[tag]
```

- one branch per feature or fix
- test locally before commit
- write clear commit messages

commit message example

```
implement password reset
- add endpoint /api/reset
- validate tokens
- update password flow
- generated with ai, reviewed manually
- tests added for expired + invalid tokens
```

---

pattern 5 review ai code fast

```
flowchart LR
  A[functionality] --> B[integration]
  B --> C[security]
  C --> D[performance]
```

before you accept code, check

- does it solve the exact problem
- will it break something else
- inputs validated, secrets safe
- any obvious performance issues

prompt to ask for clarity

```
explain the approach in plain words. why this pattern, what trade-offs, and what
```

---

pattern 6 use multiple ai “lanes” when needed

```
flowchart LR
  A[feature] --> B[ui lane]
  A --> C[api lane]
  A --> D[tests lane]
  B & C & D --> E[merge]
```

for bigger work, split lanes

- lane 1 frontend ui
- lane 2 backend api + db
- lane 3 tests + debugging

keep shared docs so names and patterns match.

pattern 7 debug like a pro

```
flowchart LR
  A[exact error] --> B[expected]
  B --> C[relevant code]
  C --> D[smallest fix + test]
```

be specific.

- show the exact error
- say what you expected
- share only the relevant code

prompt

```
error [paste]. it happens when [action]. i expected [result]. here are the 2 fi
```

add logs, test a hypothesis, repeat.

pattern 8 scale without stress

```
flowchart LR
  A[decision log] --> B[weekly refactor]
  B --> C[share templates]
```

- keep a short decision log (why we chose things)
- refactor weekly remove duplicates, tidy files
- share templates and prompts with the team

pattern 9 mindset

```
flowchart LR
  A[be specific] --> B[iterate]
  B --> C[ship]
```

ai accelerates. you direct.

- be the editor and the director
- be specific
- iterate quickly ask, test, adjust

pattern 10 context that fits

```
flowchart LR
  A[1-page brief] --> B[link every chat]
  B --> C[refresh]
```

- keep a 1-page project brief and link it every chat
- use surgical snippets, not whole files
- after changes, refresh the brief

prompt

```
make a 1-page brief from this repo map + notes. include only what speeds coding
```

pattern 11 code contracts first

```
flowchart LR
  A[types/contracts] --> B[api shapes]
  B --> C[logic follows]
```

- write types/interfaces and api shapes before logic
- your ai follows the contract and stays consistent

prompt

```
define types and request/response contracts for [feature]. add zod validation.
```

pattern 12 observable by default

```
flowchart LR
  A[inputs/outputs] --> B[duration]
  B --> C[errors tagged]
```

- add logs, metrics, and simple traces while building
- makes debugging 10x easier later

prompt

```
add minimal logging for [area]. include inputs, outputs, duration, and error tag
```

pattern 13 feature flags and safe rollout

```
flowchart LR
  A[flag off] --> B[self test]
  B --> C[gradual on]
```

- ship behind a flag, test on yourself first
- flip on gradually

prompt

```
wrap [feature] with a feature flag. add a config guard and a fallback path.
```

pattern 14 rollback ready

```
flowchart LR
  A[tag] --> B[deploy]
  B --> C{issue?}
  C -- yes --> D[rollback]
  C -- no --> E[keep]
```

- tiny commits + tags mean easy undo

prompt

```
create a rollback plan for this change. list the tag, the files, and the single
```

pattern 15 cost and quota awareness

```
flowchart LR
  A[calls] --> B[cache]
  A --> C[batch]
  A --> D[rate limit]
```

- cache, batch, and rate limit external calls

prompt

```
audit external calls. suggest caching + rate limits. add exponential backoff.
```

pattern 16 dependency hygiene

```
flowchart LR
  A[audit deps] --> B[flag heavy]
  B --> C[swap lighter]
```

- pin versions, avoid heavy libs for simple needs

prompt

```
review dependencies. flag heavy or risky ones and propose lighter swaps.
```

pattern 17 env and secrets sanity

```
flowchart LR
  A[.env.example] --> B[no secrets in code]
  B --> C[scan + fix]
```

- .env.example always, no secrets in code

prompt

```
generate .env.example with descriptions. scan code for hard-coded secrets and fi
```

pattern 18 accessibility by habit

```
flowchart LR
  A[labels] --> B[focus]
  B --> C[roles]
  C --> D[contrast]
```

- keyboard nav, labels, contrast, focus states

prompt

```
audit [screen] for a11y basics. fix labels, focus, roles, and contrast with code
```

pattern 19 i18n from day one (lightweight)

```
flowchart LR
  A[extract copy] --> B[messages file]
  B --> C[use in ui]
```

- keep copy in a simple messages file

prompt

```
extract user-facing strings from [component] to a messages file. show usage.
```

pattern 20 analytics that matter

```
flowchart LR
  A[feature] --> B[success event]
  A --> C[failure event]
```

- log one success metric per feature

prompt

```
add a minimal analytics event for [feature] success + failure. show payload shap
```

pattern 21 cache the boring stuff

```
flowchart LR
  A[memoize ui] --> B[cache api]
  B --> C[invalidate]
```

- memoize expensive ui bits, cache hot api reads

prompt

```
identify 3 places to cache/memoize. add code with invalidation notes.
```

pattern 22 happy paths + sharp edges

```
flowchart LR
  A[loading] --> B[empty]
  B --> C[error]
  C --> D[retry]
```

- always ship loading, empty, error states

prompt

```
add loading/empty/error states to [component] with clear copy and retry.
```

pattern 23 smoke tests before deploy

```
flowchart LR
  A[smoke list] --> B[run]
  B --> C{green?}
  C -- yes --> D[deploy]
  C -- no --> E[fix]
```

- 5-minute checklist catches most issues

prompt

```
write a 5-minute smoke test checklist for this app. link commands.
```

pattern 24 refactor rhythm

```
flowchart LR
  A[weekly slot] --> B[dedupe]
  B --> C[simplify]
  C --> D[update docs]
```

- schedule a weekly 30-minute clean-up

prompt

```
propose a refactor plan: top 5 duplicates/long functions to simplify this week.
```

pattern 25 team brief, not a novel

```
flowchart LR
  A[what shipped] --> B[what’s next]
  B --> C[blockers]
  C --> D[asks]
```

- update the team in bullets, not essays

prompt

```
write a team update: what shipped, what's next, blockers, asks. 8 bullets max.
```

pattern 26 agent-friendly tasks

```
flowchart LR
  A[inputs] --> B[steps]
  B --> C[outputs]
  C --> D[done signal]
```

- describe tasks with inputs, outputs, done signal

prompt

```
turn [work] into agent tasks with inputs, outputs, steps, and done criteria.
```

pattern 27 prompt once, reuse everywhere

```
flowchart LR
  A[collect prompts] --> B[categorize]
  B --> C[reuse]
```

- keep a prompts.md and reuse winners

prompt

```
organize our best prompts into prompts.md with categories and examples.
```

pattern 28 prod parity locally

```
flowchart LR
  A[seed data] --> B[local env]
  B --> C[parity checks]
```

- use seed data and env parity to avoid surprises

prompt

```
create a tiny seed script and a local .env that mirrors prod safely.
```

pattern 29 golden path docs

```
flowchart LR
  A[clone] --> B[install]
  B --> C[run]
  C --> D[test]
  D --> E[ship]
```

- one readme section that tells a new dev how to ship in 15 minutes

prompt

```
write a golden path: clone, install, run, test, ship. 12 steps max.
```

pattern 30 keep copy human

```
flowchart LR
  A[plain copy] --> B[helpful errors]
  B --> C[no jargon]
```

- plain language in ui. helpful errors. no jargon

prompt

```
rewrite user messages in [component] in simple, friendly language.
```

---

ultra-focused “one-liner” prompts (speed mode)

```
smallest mvp for [feature] in 6 bullets.
```

```
1 schema, 2 routes, 3 components for [idea].
```

```
tests first for [feature]. then code to pass.
```

```
find the slowest thing on this page and speed it up with 1 change.
```

```
explain this file like i’m a new teammate. 10 bullets.
```

```
turn this bug into a failing test, then fix.
```

```
make this commit message tell the story in 8 lines.
```

wrap-up

these prompts and patterns are how i work every day simple, direct, and fast. copy, paste, ship. then improve.

start small. keep it simple. ship. then improve. that's how i work every day.

how i (walid) actually do it

- i run discovery, scope clearly, and keep docs tiny and current
- i build mvp first. nice to have goes later
- i prefer next.js + typescript + tailwind + supabase + vercel for speed
- i keep chats short. one feature at a time
- i branch for every change. tiny commits. clear messages
- i review ai output fast. security, inputs, and performance first
- i share what works on linkedin with real examples

---

copy-paste starters (useful prompts)

```
kickoff
help me write a tiny project doc with goals, architecture, conventions, and curr
```

```
feature plan
for [feature], write user story, tech breakdown, tests, and integration points.
```

```
coding briefing
stack next.js + ts + tailwind + supabase. goal build [component]. inputs [x]. ou
```

```
debug
error [paste]. happens on [action]. expected [result]. relevant files [list]. gi
```

---

final thought

visuals mermaid for key patterns

```
flowchart LR
  P1[pattern 1: write it down] --> P2[pattern 2: plan]
  P2 --> P3[pattern 3: small chunks]
  P3 --> P4[pattern 4: git seatbelt]
  P4 --> P5[pattern 5: review fast]
  P5 --> P7[pattern 7: debug]
  P7 --> P23[pattern 23: smoke tests]
  P23 --> DEPLOY[ship]
```

```
flowchart LR
  subgraph pattern 1
    A[raw idea] --> B[questions]
    B --> C[tiny doc]
    C --> D[update after changes]
  end
```

```
flowchart LR
  subgraph pattern 2
    A[requirements chat] --> B[architecture snap]
    B --> C[mvp vs later]
  end
```

```
flowchart LR
  subgraph pattern 3
    A[one chat = one feature] --> B[brief each chat]
    B --> C[start fresh when long]
  end
```

```
flowchart LR
  subgraph pattern 4
    A[new branch] --> B[test locally]
    B --> C[clear commit]
    C --> D[tag]
  end
```

```
flowchart LR
  subgraph pattern 5
    A[functionality ok?] --> B[integration ok?]
    B --> C[security ok?]
```

```
    C --> D[performance ok?]
  end
```

```
flowchart LR
  subgraph pattern 7
    A[exact error] --> B[expected result]
    B --> C[relevant code only]
    C --> D[smallest fix + test]
  end
```

```
flowchart LR
  subgraph pattern 10
    A[1-page brief] --> B[link every chat]
    B --> C[refresh after changes]
  end
```

```
flowchart LR
  subgraph pattern 11
    A[types/contracts] --> B[api shapes]
    B --> C[logic follows]
  end
```

```
flowchart LR
  subgraph pattern 12
    A[inputs/outputs logs] --> B[duration]
    B --> C[errors tagged]
  end
```

```
flowchart LR
  subgraph pattern 13
    A[feature behind flag] --> B[self test]
    B --> C[gradual rollout]
  end
```

```
flowchart LR
  subgraph pattern 21
    A[memoize ui] --> B[cache api reads]
    B --> C[invalidation notes]
  end
```

```
flowchart LR
  subgraph pattern 23
    A[5-min checklist] --> B[run locally]
    B --> C[green? deploy]
  end
```

```
flowchart LR
  subgraph pattern 24
    A[weekly 30-min] --> B[dedupe]
    B --> C[shorten long funcs]
  end
```

big picture everything at a glance

```
flowchart TD
  IDEA[idea] --> DOC[pattern 1: tiny doc]
  DOC --> PLAN[pattern 2: plan]
  PLAN --> CHUNKS[pattern 3: small chunks]
  CHUNKS --> BRANCH[pattern 4: git branch]
  BRANCH --> REVIEW[pattern 5: review]
  REVIEW --> LANES[pattern 6: lanes]
  LANES --> DEBUG[pattern 7: debug]
  DEBUG --> CONTEXTFIT[pattern 10: context fit]
  CONTEXTFIT --> CONTRACTS[pattern 11: contracts]
  CONTRACTS --> OBS[pattern 12: observable]
  OBS --> FLAGS[pattern 13: flags]
  FLAGS --> COST[pattern 15: cost]
  COST --> DEPS[pattern 16: deps]
  DEPS --> ENV[pattern 17: env]
  ENV --> A11Y[pattern 18: a11y]
  A11Y --> I18N[pattern 19: i18n]
  I18N --> ANALYTICS[pattern 20: analytics]
  ANALYTICS --> CACHE[pattern 21: cache]
  CACHE --> STATES[pattern 22: states]
  STATES --> SMOKE[pattern 23: smoke test]
  SMOKE --> REFACTOR[pattern 24: refactor]
  REFACTOR --> TEAM[pattern 25: team brief]
  TEAM --> AGENTS[pattern 26: agent tasks]
  AGENTS --> PROMPTS[pattern 27: prompt lib]
  PROMPTS --> PARITY[pattern 28: prod parity]
  PARITY --> GOLDEN[pattern 29: golden path]
  GOLDEN --> COPY[pattern 30: human copy]
  COPY --> DEPLOY[ship]
```

power prompts 3.0 stronger, clearer, faster

use these as drop-ins. each one is terse, specific, and built for speed.

```
context condenser
here’s messy context: [paste]. compress into a clean brief with: goal, users, co
```

```
scope slicer
feature [name]. propose the smallest shippable slice that proves value in under
```

```
arch snap
for [idea], propose db tables, api routes, and ui components. 1 line per item. r
```

```
test-first stub
write tests first for [feature]. cover happy path, 2 edge cases, 1 failure. jest
```

```
impl after tests
now implement to pass the tests. touch only the files you listed. keep functions
```

```
error surgeon
here’s the error: [paste].
happens when: [action].
i expected: [result].
show smallest change that fixes it. give patch diff and 1 regression test.
```

```
perf poke
this feels slow in [component/endpoint]. propose 3 quick wins with estimated imp
```

```
security sweep
scan this change for input validation, auth, secrets, and data leaks. return a c
```

```
commit storyteller
generate a clear commit message: what, why, how, risks, follow-ups. max 8 lines
```

```
docs sync
update tiny docs: goals, architecture, conventions, priorities. include decisior
```
