MILES HIGH CLUB · CLAUDE

PAPER TRADING BOT PROMPTS

Six Claude prompts that take a paper-trading bot from broker connection through an honest no-memory baseline, a memory system with a live raw-vs-memory comparison, and a final experimentation-ready build — in the right order, with nothing skipped.

TypeScript + Node.js · Real market data · No real trades

Run them in order — each one gates the next

These six prompts are designed to be pasted in sequence into Claude. Earlier prompts produce a reviewable artifact — a connection report, a backtest, an instructions file — and pause for your confirmation. Later prompts build, run, and compare in the same step, because by that point the groundwork has already been verified.

BEFORE YOU START

- These prompts work in Claude Desktop, Claude Code, Claude in the browser, Cursor, or any MCP-capable client.
- Paper/test mode only, throughout. No prompt in this sequence enables live trading.
- Prompts 01 through 03 explicitly pause for your input or confirmation — don't skip ahead even if Claude offers to continue.
- Prompts 04 through 06 build real files and run real commands — each one reports its own results honestly, including blockers.

01 Connect MCP To Claude Before Build

Interviews you on venue, asset type, and Claude client, then verifies the MCP connection with read-only smoke tests. No bot code is written here.

02 Backtest Strategy With TradingView

Validates a strategy hypothesis in TradingView — live if a TradingView MCP is available, otherwise via a generated Pine Script — before it's trusted as the bot's strategy.

03 Create The Instructions File

Claude interviews you on goals and trading style, writes trading_bot_instructions.md, then tells you exactly where to place it before any build prompt runs.

04 First Build Without Memory

Builds the complete bot end to end. No memory, no staged failure, no fake data — an honest first version that reports whatever the real market data shows.

05 Add Memory System And Run Comparison

Implements ledger.csv, learnings.md, and the adaptive filter, then actually runs replay:raw and replay:memory side by side and reports the real difference between them.

06 Finalize Bot For Experimentation

Writes the README and env config, adds guardrails against accidental live orders, and runs a full verification pass so the bot is safe to experiment with.

PROMPT 01

Connect MCP To Claude Before Build

Verify a broker or exchange MCP connection safely — before a single line of bot code exists.

What This Prompt Does

This prompt asks Claude to guide you through connecting a broker or exchange MCP — not to build anything. It interviews you on venue, asset type, and Claude client, confirms paper/test mode before proceeding, then runs read-only smoke tests to verify the connection actually works. No code is written and no order of any kind is placed during this prompt.

The Interview

Claude asks these questions before doing anything else, and stops if any answer points toward live trading:

– Which venue: Alpaca, Pionex, Binance, Bybit, another crypto exchange, or another brokerage?

– Trading stocks, crypto, or both?

– Which Claude environment: Claude Desktop, Claude Code, Claude in the browser, Cursor with Claude, or another client?

– Confirmed paper/test mode? If not, Claude stops and tells you to create a paper/test setup first.

– Market-data-only permissions first, or paper-trading permissions?

CONNECTION RULES

- Paper/test mode only — live trading is never enabled in this prompt.
- No order is placed, previewed, or cancelled during the smoke test.
- API keys are never pasted into chat or source code.
- Credentials stay in the MCP client config, an approved secrets store, or server-side environment variables.
- If the venue supports restricted or sub-account API keys, Claude recommends that restricted setup first.

The Three Steps

Step 1. Identify the Correct Setup Path

If the venue has an official MCP server, use that path. If not, Claude explains the safest API fallback without building it yet. If MCP tools aren't visible in the current session, Claude stops and gives the exact client setup checklist.

Step 2. Verify the MCP Connection

Check whether the MCP tools are visible, confirm paper/test mode before any trading-related action, read account status and balances, read open positions and open orders, and fetch recent market data for one default symbol. No order action is taken unless explicitly requested later.

Step 3. Return a Connection Report

Venue selected, Claude client selected, MCP/API status, account mode, tools verified, market data verified, whether credentials stayed out of the codebase, and the next step before building begins.

IF ANYTHING LOOKS WRONG

- If the account mode comes back as live, unknown, or unsafe, Claude stops immediately.
- Claude states exactly what needs to be fixed before continuing — it does not proceed cautiously anyway.

The Prompt

PASTE INTO CLAUDE

```
Help me connect my broker or exchange MCP/API to Claude before we
build the trading bot.
Do not write bot code yet. First guide me through the connection
safely.
Start by asking me these questions if the answers are not obvious:
- Which venue: Alpaca, Pionex, Binance, Bybit, another crypto
exchange, or another brokerage?
- Trading stocks, crypto, or both?
- Which Claude environment: Claude Desktop, Claude Code, Claude
in the browser, Cursor with Claude, or another client?
- Am I using paper/test mode? If not, stop and tell me to create
a paper/test setup first.
- Market-data-only permissions first, or paper-trading
permissions?
Connection rules: paper/test mode only, no live trading, no
order placed, no API keys pasted into chat or source code, no
secrets logged, no credentials exposed to frontend code. If an
exchange supports sub-accounts or restricted API keys,
recommend that restricted paper/test setup first.
Step 1: Identify the correct setup path — official MCP server
if available, safest API fallback if not, stop and give setup
checklist if MCP tools aren't visible.
Step 2: Verify the connection — account status, balances, open
positions, open orders, market data. Do not submit, preview-
submit, or cancel an order during this smoke test unless I
explicitly ask later.
Step 3: Give me a connection report — venue, client, MCP
status, account mode, tools verified, credentials kept out of
codebase, next step before building.
If anything is live, unknown, or unsafe, stop and tell me
exactly what to fix before continuing.
```

PROMPT 02

Backtest Strategy With TradingView

Validate the strategy hypothesis in TradingView before it becomes the bot's source of truth.

What This Prompt Does

Before any strategy rule gets baked into the bot's instructions file, this prompt tests it in TradingView. If a TradingView MCP is available, Claude runs the backtest directly and reports real results. If not, Claude generates a Pine Script v5 strategy and gives exact steps to run it manually — it never pretends to have run a backtest it didn't actually run.

Inputs Claude Asks For

– Market or ticker — e.g. BTCUSDT, ETHUSDT, AAPL, NVDA, or SPY

– Timeframe — e.g. 5m, 15m, 1h, or 1d

– Test window — e.g. last 90 days, last 12 months, or a specific date range

– Strategy idea, indicators, and entry/exit rules

– Risk rules — position size, stop loss, take profit, max drawdown

– Fees, slippage, and whether the strategy is long-only or long/short

Default Strategy If None Provided

BTCUSDT, 5m timeframe, 9-period fast MA, 21-period slow MA. Buy when fast MA crosses above slow MA, sell or exit when fast MA crosses below slow MA. Paper/backtest only.

BACKTEST QUALITY RULES

- No future data and no repainting signals.
- Commission and slippage assumptions are included explicitly.
- Entry and exit rules are made fully explicit — no implied logic.
- In-sample observations are separated from out-of-sample validation where possible.
- The strategy is never claimed profitable unless the actual backtest output supports it.

What Gets Returned

– The strategy hypothesis and the exact rules tested

– Pine Script v5 code, if a live TradingView MCP wasn't available

– Backtest metrics: net profit, win rate, max drawdown, profit factor, total trades, average trade, best trade, worst trade

– Weaknesses or market conditions where the strategy fails

– Whether the rule set is solid enough to copy into the instructions file

– The final, concise rule set ready to paste into trading_bot_instructions.md

The Prompt

PASTE INTO CLAUDE

```
Help me backtest a trading strategy in TradingView before we
build the bot.
Start by asking me for the missing inputs: market/ticker,
timeframe, test window, strategy idea and rules, risk rules,
fees/slippage, long-only or long/short.
Default strategy if I don't provide one: BTCUSDT, 5m, 9-period
fast MA, 21-period slow MA, buy on bullish crossover, sell/exit
on bearish crossover, paper/backtest only.
Use TradingView if the TradingView MCP or browser workflow is
available.
If a TradingView MCP is available: open or prepare the chart,
add the strategy as a backtestable strategy (not just an
indicator), run it over the date range, report results clearly.
If a TradingView MCP is not available: do not pretend you ran
it. Generate a Pine Script v5 strategy instead and give exact
steps to paste it into Pine Editor, add it to the chart, and
read the Strategy Tester results.
Backtest quality rules: no future data, no repainting signals,
include commission and slippage assumptions, make entry/exit
rules explicit, separate in-sample from out-of-sample where
possible, do not claim profitability the backtest doesn't
support.
Return: the strategy hypothesis, the exact rules tested, Pine
Script if needed, full backtest metrics, weaknesses or market
conditions where it fails, whether the rule set is good enough
for the instructions file, and the final concise rule set for
the bot instructions file.
```

PROMPT 03

Create The Instructions File

Claude interviews you first, writes trading_bot_instructions.md from your actual answers, then tells you exactly where to put it.

What This Prompt Does

Instead of generating an instructions file from a fixed template, this prompt has Claude interview you first — one to three questions at a time, with follow-ups on vague answers — and only writes trading_bot_instructions.md once your actual goals and trading style are understood. It then tells you exactly where the file needs to live before any build prompt can use it. No code, no bot, and no broker connection happen in this prompt.

Interview Rules

– Ask 1 to 3 questions at a time — never an overwhelming list at once.

– If an answer is vague, ask a follow-up rather than guessing.

– Never give financial advice — only translate preferences into bot rules.

– If you don't know an answer, Claude suggests a safe paper-trading default.

– Everything stays paper/test mode first, regardless of your answers.

What Claude Asks About

– Asset class (crypto, stocks, or both) and venue

– Default symbols and timeframes

– Strategy style — trend, momentum, mean reversion, breakout, scalping, or simple MA crossover

– Backtest result or strategy source (this is where Prompt 02's output feeds in)

– Position sizing, max position, stop loss, take profit

– Max daily loss or max drawdown rule

– Whether memory should start as two local files

– What the bot should never do, and what terminal output you want to see

The Seven Sections of the Output File

1. Project Goal	What the bot does, starting market and timeframe, why paper/test mode comes first.
2. Safety Rules	Paper trading by default, no live trading path, no secrets in source, no frontend credential exposure, no action unless risk passes.
3. Strategy Rules	Indicators, entry rules, exit rules, hold/skip conditions, and any backtest notes from Prompt 02.
4. Risk Rules	Position sizing, max position, stop loss or invalidation, max loss/drawdown limits, and what triggers SKIP.
5. Broker/MCP Rules	Selected venue, paper/test mode requirement, connection verification, and account/position/order/market-data checks.

6. Memory Rules	ledger.csv logs trades and skips, learnings.md stores plain-English lessons, both are read before every future trade, known failed setups become SKIP.
7. Definition of Done	Commands that should work, logs the bot should print, confirmation that no real trades are ever placed.

OUTPUT OF THIS PROMPT

- trading_bot_instructions.md, written from your actual interview answers.
- Claude shows you the full file contents.
- Claude tells you to add or drag the file into your Claude Project / project knowledge, or keep it at the root of your local coding project.
- Claude then waits for your confirmation before any build prompt runs.

The Prompt

PASTE INTO CLAUDE

```
I want to create the instructions file for a paper-trading bot,
but before writing it, interview me so the bot matches my goals
and trading style.
Do not write code yet. Do not build the bot yet. Do not connect
to a broker yet.
Your job is to ask me questions, go back and forth with me, and
then create a one-page file called trading_bot_instructions.md.
Interview rules: ask 1-3 questions at a time, follow up on
vague answers, do not give financial advice, translate my
preferences into clear bot rules, suggest a safe paper-trading
default if I don't know, keep everything paper/test mode first.
Ask about: asset class, venue, default symbols, strategy style,
timeframes, backtest result/source, position sizing, max
position, stop loss/take profit, max drawdown, whether memory
should start as two local files, what the bot should never do,
and desired terminal outputs.
After the interview, create trading_bot_instructions.md with:
1. Project Goal 2. Safety Rules 3. Strategy Rules
4. Risk Rules 5. Broker/MCP Rules 6. Memory Rules
7. Definition Of Done
When done: show me the full file contents, tell me to add or
drag trading_bot_instructions.md into my Claude Project/project
knowledge, or keep it at the root of my local coding project,
and wait for my confirmation before any build prompt.
```

PROMPT 04

First Build Without Memory

Build the complete bot end to end — an honest, unstaged baseline using real market data, with no memory yet.

What This Prompt Does

This prompt builds the entire first version of the bot from trading_bot_instructions.md. The defining rule of this step is honesty: the bot must not be made to fail on purpose, must not fake a losing trade, and must not use generated or fixture candle data. Whatever the real market data shows — win, loss, or neither — is what gets reported.

Files Created

package.json	tsconfig.json	src/types.ts	src/market.ts	src/strategy.ts
src/risk.ts	src/execution.ts	src/replay.ts	src/bot.ts	src/index.ts

Default Setup

Market: BTCUSDT. Interval: 5m. Data source: Binance public klines endpoint. Strategy: moving-average crossover — 9-period fast MA, 21-period slow MA. Buy on bullish crossover, sell on bearish crossover, hold when there is no fresh crossover.

NON-NEGOTIABLE HONESTY RULES

- This first version must not have memory — no ledger.csv, no learnings.md, no adaptive filtering yet.
- Do not intentionally make the bot fail. Do not fake losing trades.
- Do not use generated candle data or fixture candles, ever.
- Use real market candles where possible and report the actual results — whatever they are.
- Paper/local execution only — no live broker order endpoints, no API keys requested.

Architecture Rule

Keep broker/MCP integration as a future adapter unless it already exists and is verified in paper/test mode. The strategy should return a signal; the risk module should approve or reject it; the execution module should only simulate a paper order.

Risk Requirements

– Quantity must be configurable.

– Max position must be configurable.

– If quantity exceeds max position, the final action must be SKIP.

– Every decision must include a plain-English reason.

CLI Behaviour

npm run scan — fetches recent real candles, calculates the strategy signal, runs the risk check, simulates paper execution only when BUY or SELL passes risk, and prints timestamped logs for market data, signal, risk, and final decision.

npm run replay:raw — uses real historical BTCUSDT candles, detects actual moving-average crossover events, calculates forward outcome after a configurable number of candles, and prints a trade-by-trade table plus summary metrics: total setups, wins, losses, win rate, average PnL, best trade, worst trade, and max drawdown if practical. It flags repeated weak setups only if they actually appear in the data, and says clearly if there aren't enough setups or if the raw strategy doesn't show repeated losses in the lookback window.

Acceptance Criteria

– npm install works, npm run scan works, npm run replay:raw works.

– The bot uses real public market data, or fails clearly if data is unavailable.

– No memory system exists in this first version.

– No real broker execution path exists.

– The replay is an honest baseline — not a staged failure.

The Prompt

PASTE INTO CLAUDE

```
Build the first working version of the paper-trading bot
described in trading_bot_instructions.md.
Use TypeScript and Node.js. If the project is empty, create
the project from scratch.
Important: this first version should not have memory yet. Do
not create ledger.csv. Do not create learnings.md. Do not add
adaptive filtering yet. Do not intentionally make the bot fail.
Do not fake losing trades. Do not use generated candle data or
fixture candles. Use real market candles where possible and
report the actual results.
Create: package.json, tsconfig.json, src/types.ts,
src/market.ts, src/strategy.ts, src/risk.ts, src/execution.ts,
src/replay.ts, src/bot.ts, src/index.ts
Default: BTCUSDT, 5m interval, Binance public klines endpoint,
9/21 MA crossover. BUY when fast MA crosses above slow MA,
SELL when it crosses below, HOLD when there's no fresh
crossover.
Safety: paper/local execution only, no real trades, no live
broker order endpoints, no API keys requested. Keep broker/MCP
integration as a future adapter unless already verified in
paper/test mode. Strategy returns a signal; risk approves or
rejects; execution only simulates a paper order.
Risk: configurable quantity and max position. SKIP if quantity
exceeds max position. Every decision needs a plain-English
reason.
CLI: npm run scan, npm run replay:raw. replay:raw must print a
trade-by-trade table and summary metrics, flag repeated weak
setups only if they appear, and say clearly if there aren't
enough setups or no repeated losses in the lookback window.
After implementing, show me: final file structure, exact
commands to run, sample scan output, sample replay:raw output
with summary metrics, and a note confirming this version has
no memory system yet.
```

PROMPT 05

Add Memory System And Run Comparison

Implement the two-file memory layer, then actually run replay:raw and replay:memory side by side and report the real difference.

What This Prompt Does

This prompt builds the memory system and runs the comparison in the same step. Claude writes ledger.csv, learnings.md, the memory module, and the adaptive filter, keeps the original honest replay:raw path working, then runs both replay:raw and replay:memory and reports the real difference between the two — including the latest ledger row and the matching learnings note.

Files Created or Updated

data/ledger.csv	data/learnings.md	src/memory.ts	src/adaptiveFilter.ts
src/replay.ts	src/bot.ts	src/index.ts	package.json

ledger.csv Header & Fields

CSV HEADER

```
timestamp,symbol,action,price,quantity,reason,mode,outcome,pnl
```

Logs completed paper/replay trades and skipped decisions — symbol, action, price, quantity, plain-English reason, mode, outcome, and PnL for each entry. learnings.md stores plain-English lessons distilled from closed trades.

Before Any Future BUY or SELL, the Bot Must Ask

– Has this symbol lost on a similar setup before?

– Does the learnings file warn about this setup?

– Is this signal repeating a known bad trade?

If the setup matches a previous losing pattern: change the final action to SKIP, do not create a paper order, append a SKIP row to ledger.csv when the memory path runs, and print the reason clearly.

MEMORY QUALITY RULES

- Do not seed fake BTCUSDT losses.
- Do not invent candles.
- Do not force the strategy to fail.
- Learn only from real paper-trade or replay outcomes.
- If no prior loss exists, replay:memory should say to run replay:raw first.

How replay:raw and replay:memory Now Interact

– replay:raw must still calculate the raw baseline without using memory.

– Once the memory system exists, replay:raw may append real replay outcomes to ledger.csv.

– If replay:raw finds a losing crossover setup in real historical data, it writes a plain-English lesson to learnings.md.
– If replay:raw finds no losing setup, no lesson gets seeded — nothing is invented.
– replay:memory only skips when ledger.csv or learnings.md contains a real prior warning.
– If there isn't enough real memory yet, replay:memory should HOLD or tell you to keep paper testing.

This Prompt Runs Both Commands

TERMINAL

```
npm run replay:raw
npm run replay:memory
```

Expected replay:memory Log

TERMINAL OUTPUT

```
Scanner started for BTCUSDT on 5m
Loaded real historical candles
Detected a moving-average crossover candidate
Risk check passed
Loaded data/ledger.csv
Loaded data/learnings.md
Found prior BTCUSDT crossover loss, or reported none exists yet
Found matching lesson requiring confirmation, or reported
no matching lesson exists yet
Decision: SKIP if memory blocks it, otherwise HOLD until
enough memory exists
No paper order was sent
```

Acceptance Criteria

– npm run replay:raw still works.
– npm run memory:reset exists and npm run replay:memory exists.
– replay:memory uses the ledger and learnings file before any BUY or SELL.
– replay:memory skips only when prior real paper/replay losses support the skip.
– If no matching prior loss exists, replay:memory returns HOLD or tells you to run replay:raw first.
– data/ledger.csv logs the memory decision after replay:memory runs, and learnings.md stays readable and useful.
– No live broker execution, no API keys or secrets, and no fake seeded losses are added.

What Claude Shows You After Running

– Files changed
– The exact commands that were run
– A short sample of replay:raw output and a short sample of replay:memory output
– The latest ledger.csv row
– The relevant learnings.md note
– A short explanation of what changed between the raw bot and the memory-enabled bot

The Prompt

PASTE INTO CLAUDE

```
Now add the two-file memory system to the existing TypeScript
paper-trading bot.
Important: keep the original raw strategy path working, add
memory as a separate improved path, run the improved memory
path after implementation, compare the raw strategy output
against the memory-enabled output. Do not fake losses, invent
candles, or force the bot to fail.
Keep replay:raw as an honest baseline that still ignores memory.
Add: npm run replay:memory, npm run memory:reset.
Create or update: data/ledger.csv, data/learnings.md,
src/memory.ts, src/adaptiveFilter.ts, src/replay.ts, src/bot.ts,
src/index.ts, package.json
ledger.csv header:
timestamp,symbol,action,price,quantity,reason,mode,outcome,pnl
Before any future BUY or SELL, read both files and ask: has
this symbol lost on a similar setup before, does learnings.md
warn about this setup, is this signal repeating a known bad
trade? If matched: SKIP, no paper order, append a SKIP row when
the memory path runs, print the reason clearly.
Memory quality rules: no seeded fake losses, no invented
candles, no forced failure, learn only from real outcomes. If
no prior loss exists, replay:memory should say to run
replay:raw first.
Replay learning behavior: replay:raw still calculates the raw
baseline without memory. After the memory system exists,
replay:raw may append real replay outcomes to ledger.csv. If
replay:raw finds a real losing crossover setup, write a plain-
English lesson to learnings.md. If it finds none, don't seed
one. replay:memory should only skip with a real prior warning
— otherwise HOLD or tell me to keep paper testing.
After implementing, run: npm run replay:raw and npm run
replay:memory.
Then show me: files changed, exact commands run, sample
replay:raw output, sample replay:memory output, the latest
ledger.csv row, the relevant learnings.md note, and a short
explanation of what changed between the raw bot and the
memory-enabled bot.
```

PROMPT 06

Finalize Bot For Experimentation

README, env config, guardrails, and a full verification pass — so the bot is safe to experiment with after the build.

What This Prompt Does

This is the closing prompt. It takes everything built across Prompts 01 through 05 — the instructions file, the first bot implementation, any broker/MCP guardrails, and the two-file memory system — and packages it into something documented, configurable, and safe to keep experimenting with. It finishes by actually running a full verification pass and reporting honestly what passed and what didn't.

Non-Negotiable Rules

– Do not add live trading.

– Do not expose secrets to frontend code.

– Do not commit API keys.

– Do not fake market data or fake performance.

What the Finalized Project Must Have

– Clear setup instructions and safe paper/local defaults

– Optional paper MCP/API mode — only if the connection is already verified

– Working commands for one scan, raw replay, memory-enabled replay, and memory reset

– A clear place to change symbols, intervals, strategy settings, quantity, and max position

– Clear terminal logs

– Guardrails that prevent accidental live orders

Files Created or Updated

README.md	.env.example	.gitignore	package.json

Plus any config file needed to make the bot easy to run, and any small cleanup in src files needed to make the commands reliable.

What README.md Must Explain

– What the bot does and how to install dependencies

– How to run the first scan, the raw replay, and the memory replay

– How to reset memory

– How paper/local execution works, and how optional paper MCP/API mode should be configured safely

– Where ledger.csv and learnings.md live

– How to experiment with a new strategy or symbol

– The safety rules and limitations

Package Scripts

PACKAGE.JSON SCRIPTS

```
scan — always present
replay:raw — always present
replay:memory — always present
memory:reset — always present
broker:check — only if a broker/MCP adapter exists
broker:preview — only if a broker/MCP adapter exists
```

Final Verification Run

This prompt actually executes the full sequence as part of finishing the build:

TERMINAL

```
npm install # if dependencies changed
npm run scan
npm run replay:raw
npm run replay:memory
npm run memory:reset
npm run replay:memory # run again to confirm empty-memory
# behavior is clear
```

IF SOMETHING CAN'T RUN

- If any command can't run because of missing credentials, missing MCP tools, or unavailable public market data, Claude does not fake the result.
- The blocker gets printed clearly.
- The fallback behavior is made safe rather than silently skipped.

What Claude Shows You At The End

– Final file structure

– Commands that passed

– Commands that were blocked and why

– Any environment variables required

– A short safety checklist

– The next three experiments you can try in paper mode

The Prompt

PASTE INTO CLAUDE

```
Finalize this paper-trading bot so I can safely experiment
with it after the build.
Use the existing project, trading_bot_instructions.md, first
bot implementation, broker/MCP guardrails, and two-file memory
system.
Do not add live trading. Do not expose secrets to frontend
code. Do not commit API keys. Do not fake market data or fake
performance.
Finalize the project so it has: clear setup instructions, safe
paper/local defaults, optional paper MCP/API mode only if the
connection is already verified, working commands for scan,
raw replay, memory replay, and memory reset, a clear place to
change symbols/intervals/strategy settings/quantity/max
position, clear terminal logs, and guardrails that prevent
accidental live orders.
Create or update: README.md, .env.example, .gitignore,
package.json scripts, any config file needed, and any small
src cleanup needed to make commands reliable.
README.md should explain: what the bot does, how to install,
how to run scan/raw replay/memory replay/memory reset, how
paper/local execution works, how to configure optional paper
MCP/API mode safely, where ledger.csv and learnings.md live,
how to experiment with a new strategy or symbol, and the
safety rules and limitations.
Package scripts should include: scan, replay:raw,
replay:memory, memory:reset, and broker:check/broker:preview
only if a broker/MCP adapter exists.
Run final verification: npm install if needed, npm run scan,
npm run replay:raw, npm run replay:memory, npm run
memory:reset, then npm run replay:memory again to confirm the
empty-memory behavior is clear.
If any command can't run due to missing credentials, missing
MCP tools, or unavailable public market data, do not fake the
result — print the blocker clearly and make the fallback
behavior safe.
At the end, show me: final file structure, commands that
passed, commands that were blocked and why, any environment
variables required, a short safety checklist, and the next
three experiments I can try in paper mode.
```