Opus 4.7 Trading Bot — Setup Guide

A complete blueprint for building an autonomous, cloud-scheduled trading agent on top of Claude Code.

Designed to be self-contained: paste this document into your own Claude Code session and it should have everything needed to replicate the system end-to-end.

The agent places real trades on Alpaca, writes its own daily research, executes a disciplined strategy with hard rules, and notifies you via chat. It is stateless between runs — all memory lives in your Git repo.

Prerequisites: a GitHub account, an Alpaca brokerage account (paper is fine to start), a Perplexity API account, a ClickUp account, and access to Claude Code cloud routines.

By: Nate Herk

Table of Contents

Part 1 — What You're Building

A fully autonomous trading agent that runs on a daily schedule. Five cron jobs fire throughout each weekday, each one spinning up a fresh Claude Code cloud container that clones your repo, reads memory, pulls live account state, decides on action, places real orders if warranted, writes new memory, commits everything back to Git, and sends you a chat notification.

There is no separate Python bot process. **Claude is the bot.** Every scheduled run is a fresh LLM invocation reading a well-defined prompt.

The five daily jobs at a glance

- **Pre-market** (early morning): research catalysts, write today's trade ideas to the research log.
- **Market-open** (shortly after the bell): execute planned trades, set trailing stops on every new position.
- **Midday**: scan open positions, cut losers, tighten stops on winners.
- **Daily summary** (late afternoon): snapshot portfolio state, send chat recap.
- **Weekly review** (Friday afternoon): compute weekly stats, grade performance, update the strategy if needed.

Why this design

Three properties drove every decision:

- **Stateless runs** — each firing is independent, which means failures self-heal on the next tick.
- **Git as memory** — every piece of state is a markdown file committed to main. Free versioning, diffs, rollback, and human-readable audit trail.
- **Hard rules as gates** — strategy discipline is enforced programmatically before every order, not left to interpretation.

Part 2 — The Trading Strategy

This is a swing trading strategy for stocks only. The logic is simple; the discipline is the hard part. These rules exist because they were learned the hard way in a previous 30-day live challenge where tight stops shook the bot out of winners repeatedly, overtrading cost several percent in one bad week, and a single options trade wiped out a full month of gains.

Hard rules (non-negotiable)

- No options. Ever. Stocks only.
- Maximum 5-6 open positions at a time.
- Maximum 20% of equity per position (about $2,000 on a $10,000 account).
- Maximum 3 new trades per week.
- Target 75-85% of capital deployed.

- Every position gets a 10% trailing stop placed as a real GTC order on Alpaca. Never mental.
- Cut any losing position at -7% from entry. Manual sell. No hoping, no averaging down.
- Tighten the trailing stop to 7% when a position is up +15%. Tighten to 5% when up +20%.
- Never tighten a stop to within 3% of current price. Never move a stop down.
- Exit an entire sector after 2 consecutive failed trades in that sector.
- Follow sector momentum. Don't force a thesis if the whole sector is rolling over.
- Patience beats activity. A week with zero trades can be the right answer.

The buy-side gate

Before placing any buy order, every single one of these checks must pass. If any fail, the trade is skipped and the reason is logged.

- Total positions after this fill will be no more than 6.
- Total trades placed this week (including this one) is no more than 3.
- Position cost is no more than 20% of account equity.
- Position cost is no more than available cash.
- Pattern day trader day-trade count leaves room (under 3 on a sub-$25k account).
- A specific catalyst is documented in today's research log entry.
- The instrument is a stock (not an option, not anything else).

The sell-side rules

Evaluated at the midday scan and opportunistically:

- If unrealized loss is -7% or worse, close immediately.
- If the thesis has broken (catalyst invalidated, sector rolling over, news event), close, even if not yet at -7%.
- If position is up +20% or more, tighten trailing stop to 5%.
- If position is up +15% or more, tighten trailing stop to 7%.
- If a sector has two consecutive failed trades, exit all positions in that sector.

Entry checklist (agent documents all of these before placing)

- What is the specific catalyst today?
- Is the sector in momentum?
- What is the stop level (7-10% below entry)?
- What is the target (minimum 2:1 risk/reward)?

Part 3 — Repository Layout

Create a new GitHub repository. Inside, you will have this structure:

```
trading-bot/
├── CLAUDE.md              # Agent rulebook (auto-loaded every session)
├── README.md              # Human-facing quickstart
├── env.template           # Template for local .env file
├── .gitignore             # Must exclude .env
├── .claude/
│   └── commands/          # Ad-hoc slash commands for local use
│       ├── portfolio.md
│       ├── trade.md
│       ├── pre-market.md
│       ├── market-open.md
│       ├── midday.md
│       ├── daily-summary.md
│       └── weekly-review.md
├── routines/              # Cloud routine prompts (the prod path)
│   ├── README.md
│   ├── pre-market.md
│   ├── market-open.md
│   ├── midday.md
│   ├── daily-summary.md
│   └── weekly-review.md
├── scripts/               # API wrappers (the only way to touch the outside world)
│   ├── alpaca.sh
│   ├── perplexity.sh
│   └── clickup.sh
└── memory/                # Agent's persistent state (committed to main)
    ├── TRADING-STRATEGY.md
    ├── TRADE-LOG.md
    ├── RESEARCH-LOG.md
    ├── WEEKLY-REVIEW.md
    └── PROJECT-CONTEXT.md
```

Two parallel execution modes share this codebase:

- **Local mode**: you invoke slash commands like /pre-market manually inside Claude Code. Credentials come from a local .env file. Good for testing and ad-hoc runs.
- **Cloud mode**: Claude's cloud routines fire each routines/*.md prompt on a cron. Credentials come from the routine's environment variables. **No `.env` file.** This is the production path.

Part 4 — The Three Wrapper Scripts

All external API calls flow through three bash scripts in the scripts/ directory. The agent never calls curl directly. This keeps auth handling in one place, standardizes error messages, and makes the prompts much shorter.

scripts/alpaca.sh — trading

Wraps the Alpaca v2 API. Reads ALPACA_API_KEY and ALPACA_SECRET_KEY from the environment (or .env if present). Subcommands:

```
bash scripts/alpaca.sh account          # equity, cash, buying_power, daytrade_count
bash scripts/alpaca.sh positions        # all open positions w/ unrealized P&L
bash scripts/alpaca.sh position SYM     # single position
bash scripts/alpaca.sh quote SYM        # latest bid/ask (uses data.alpaca.markets)
bash scripts/alpaca.sh orders [status]  # default status=open
bash scripts/alpaca.sh order '<json>'   # POST new order
bash scripts/alpaca.sh cancel ORDER_ID
bash scripts/alpaca.sh cancel-all
bash scripts/alpaca.sh close SYM        # market-sell entire position
bash scripts/alpaca.sh close-all
```

Three canonical order shapes cover everything the strategy does:

```
// 1. Market buy
{"symbol":"XOM","qty":"12","side":"buy","type":"market","time_in_force":"day"}

// 2. 10% trailing stop (the default stop for every new position)
{"symbol":"XOM","qty":"12","side":"sell","type":"trailing_stop","trail_percent":"10","time_in_force":"gtc"}

// 3. Fixed stop (fallback when pattern-day-trader rules block a trailing stop)
{"symbol":"XOM","qty":"12","side":"sell","type":"stop","stop_price":"140.00","time_in_force":"gtc"}
```

scripts/perplexity.sh — research

Wraps the Perplexity chat completions API. All market research routes through this — not the agent's native WebSearch tool — so every claim in the research log gets Perplexity citations. Usage:

```
bash scripts/perplexity.sh "<query>"
```

Exits with code 3 if PERPLEXITY_API_KEY is unset, letting the agent gracefully fall back to native WebSearch and flag the fallback in the log.

scripts/clickup.sh — notifications

Wraps the ClickUp Chat v3 message API. Posts to a chat channel (not a task). Usage:

```
bash scripts/clickup.sh "<markdown message>"
```

Graceful fallback: if any of CLICKUP_API_KEY, CLICKUP_WORKSPACE_ID, or CLICKUP_CHANNEL_ID is missing, the script appends the message to a local fallback file and exits 0. The agent never crashes on missing notification credentials.

Alpaca gotchas (bake these into the agent's priors)

- **Pattern day trader rule**: 3 day trades per 5 rolling business days on accounts under $25k. Check daytrade_count before buying. Same-day stops on same-day buys sometimes get rejected. Fallback ladder: trailing_stop → fixed stop → queue for tomorrow morning.

- **`trail_percent` is a string** in the JSON, not a number. Use "10", not 10. qty is also a string.
- **Market data has a different base URL**: data.alpaca.markets for quotes, api.alpaca.markets for everything else.
- **Quote response shape**: quote.ap is ask, quote.bp is bid. Wide spread or zero means halted or illiquid — skip.
- **Trailing stops only work during market hours.** Overnight gaps can blow right through them.
- **Env-var name ≠ HTTP header name.** Env var is ALPACA_API_KEY. Header is APCA-API-KEY-ID. The wrapper handles translation.
- **Alpaca timestamps are UTC.** Your crons are whatever timezone you set. Convert carefully.

Part 5 — The Five Workflows in Detail

Every workflow follows the same 8-step scaffold. The differences are what each one reads and writes. All times are in your local timezone, on weekdays only.

Pre-market workflow (early morning, before the open)

1. Read the strategy doc and the tail of the trade log and research log for context.
2. Pull live account state: account, positions, open orders.
3. Run a handful of Perplexity queries: oil prices, S&P futures, VIX, top catalysts today, pre-market earnings, economic calendar, sector momentum, news on each currently-held ticker.
4. Write a dated entry to memory/RESEARCH-LOG.md with: account snapshot, market context, 2-3 actionable trade ideas (each with catalyst, entry, stop, target), risk factors, and a trade/hold decision (default hold).
5. Notification: silent unless something urgent (a held position is already below -7% in pre-market, a thesis broke overnight, a major geopolitical event).
6. Commit memory/RESEARCH-LOG.md and push to main.

Market-open workflow (shortly after the open)

1. Read today's research log entry. If missing, run the pre-market research steps inline first. Never trade without documented research.
2. Re-validate each planned trade with fresh quotes. Check bid/ask spread, make sure nothing is halted.
3. Run the buy-side gate from Part 2 on each planned order. Skip any that fail, log the reason.
4. For each approved trade: place a market buy (day time-in-force), wait for fill, then immediately place a 10% trailing_stop as a GTC order. If Alpaca rejects the trailing stop due to pattern-day-trader rules, fall back to a fixed stop. If that is also blocked, queue the stop for tomorrow morning in the trade log.

5. Append every trade to memory/TRADE-LOG.md with full thesis, entry price, stop level, target, and risk/reward ratio.
6. Notification: ClickUp message **only if a trade was actually placed.**
7. Commit memory/TRADE-LOG.md and push. Skip commit if no trades fired.

Midday workflow (middle of the trading day)

1. Read the strategy doc, the tail of the trade log, and today's research log.
2. Pull positions and open orders.
3. For any position with unrealized P&L percentage ≤ -7%: close the position, cancel its trailing stop order, log the exit with realized P&L and the reason.
4. For winners: if up +20% or more, cancel old trailing stop and place a new one with trail_percent "5". If up +15% or more, trail_percent "7". Respect the 3%-of-current-price guardrail.
5. Thesis check: for each remaining position, review price action and any midday news. If a thesis broke intraday, cut the position even if not at -7% yet.
6. Optional intraday research via Perplexity if something is moving sharply with no obvious cause.
7. Notification: only if action was taken.
8. Commit any memory changes and push. Skip commit if nothing changed.

Daily-summary workflow (after the close)

1. Find yesterday's closing equity in the trade log (needed for day-over-day P&L math).
2. Pull today's final account state, positions, open orders.
3. Compute: day P&L in dollars and percent, phase-to-date cumulative P&L, trades today, running trade count for the week.
4. Append a dated EOD snapshot section to memory/TRADE-LOG.md with a positions table and a plain-english notes paragraph.
5. Send one ClickUp message, **always, even on no-trade days**. Keep it under 15 lines.
6. Commit the trade log and push. This commit is **mandatory** — tomorrow's day P&L calculation depends on it persisting.

Weekly-review workflow (Friday, end of day)

1. Read the full week of trade log and research log entries, the existing weekly review template, and the strategy doc.
2. Pull Friday close account state and positions.
3. Compute: starting portfolio (Monday open), ending portfolio, week return in dollars and percent, S&P 500 week return (via Perplexity), W/L/open trade counts, win rate, best trade, worst trade, profit factor.
4. Append a full review section to memory/WEEKLY-REVIEW.md: stats table, closed trades table, open positions at week end, 3-5 "what worked" bullets, 3-5 "what didn't work" bullets, key lessons, adjustments for next week, and an overall letter grade A-F.
5. If a rule has proven itself for 2+ weeks or failed badly, also update memory/TRADING-STRATEGY.md in the same commit and call out the change in the review.

6. Send one ClickUp message with headline numbers, always.
7. Commit the weekly review (and strategy doc if changed) and push.

Ad-hoc: portfolio

Read-only snapshot. Calls account, positions, orders. Prints a clean summary. No state changes, no orders, no file writes. The only commentary allowed: flag if a position has no stop, or a stop is below current price.

Ad-hoc: trade

Manual trade helper. Takes SYMBOL SHARES SIDE as arguments. Runs the full buy-side gate from Part 2. Prints the order JSON and validation results. Asks for y/n confirmation before placing. On confirm, executes the buy and immediately places the 10% trailing stop. Logs to the trade log. Sends a ClickUp message. Refuses any trade that fails a rule check.

Part 6 — Memory Model

Five markdown files, all committed to main, are the agent's only state between runs.

File	Purpose	Write Cadence
TRADING-STRATEGY.md	The rulebook. Every workflow reads this first.	Only updated Friday if a rule proves out/fails
TRADE-LOG.md	Every trade + daily EOD snapshot	Every trade, every EOD
RESEARCH-LOG.md	One dated entry per day	Every pre-market, optional midday addendum
WEEKLY-REVIEW.md	Friday recaps with letter grade	Weekly
PROJECT-CONTEXT.md	Static background, mission, platform	Rarely updated

Why memory-in-git actually works

- Schedules are hours apart. No race conditions between routines.
- Memory writes are append-only dated sections. Merge conflicts are effectively impossible.
- Each run reads from committed main. Nothing held in-memory across firings.
- Rollback is git revert. Audit is git log. Diff is git diff. Free observability.

Where memory-in-git would break

- Two routines scheduled seconds apart (don't do that).
- Someone editing memory files manually during a scheduled run (don't do that).
- Partial mid-run failure. An Alpaca order could go through with no trade log entry. Mitigation: the next run reads live positions from Alpaca and reconciles.

Part 7 — Setting Up Cloud Routines

This is the most common sticking point for first-time setup. A Claude Code cloud routine is a scheduled agent run. Each firing is an ephemeral container: clone, run, destroy.

What happens on each scheduled run

1. The cron fires in the timezone you set on the routine.
2. Claude's cloud spins up a new container.
3. It clones your GitHub repo at main, so it sees the latest memory.
4. It injects the environment variables you configured on the routine into the shell.
5. It starts Claude with the prompt you pasted into the routine.
6. Claude does the work: reads memory, calls wrappers, writes memory.
7. Claude **must run `git commit` and `git push origin main` before exiting.** Otherwise everything it did evaporates.
8. The container is destroyed.

The mental model: the cloud runner is stateless. Git is the memory. If it's not in main, it didn't happen.

One-time prerequisites

Do these once before creating any routine.

Prereq 1: Install the Claude GitHub App

Visit the Claude GitHub App install page, select only your trading bot repo (least privilege), and grant access. This gives the cloud container permission to both clone and push to your repo. Alternative: run /web-setup inside Claude Code to sync your gh CLI token — same effect.

Prereq 2: Enable unrestricted branch pushes on the routine's environment

In the routine's environment settings, there is a toggle labeled **"Allow unrestricted branch pushes"**. Without this toggle enabled, git push origin main silently fails with a proxy error. **This is the number-one reason first-time setups break.**

Prereq 3: Set environment variables on the routine (NOT in a .env file)

In the routine's environment config, add these credentials:

```
ALPACA_API_KEY            (required)
ALPACA_SECRET_KEY         (required)
ALPACA_ENDPOINT           (optional; defaults to live trading URL)
ALPACA_DATA_ENDPOINT      (optional; defaults to data URL)
PERPLEXITY_API_KEY        (required for research workflows)
PERPLEXITY_MODEL          (optional; defaults to 'sonar')
CLICKUP_API_KEY           (required for notifications)
CLICKUP_WORKSPACE_ID      (required; numeric)
CLICKUP_CHANNEL_ID        (required; format 4-XXXXXXX-X)
```

Why the "no .env file" rule matters

The wrapper scripts read .env first, falling back to process environment variables. In local mode you want the .env. In the cloud:

- A .env with real credentials committed to the repo would leak secrets immediately.
- A .env created at runtime is either a secret leak (if pushed) or wasted work (if not).
- The worst-case failure mode: agent sees "key not set," helpfully creates a .env as a workaround, and leaks credentials. **Every cloud routine prompt contains an explicit "do not create a .env file" block to prevent this.**

Step-by-step: creating your first routine

Walk-through using pre-market as the example. Repeat for each of the five.

1. In Claude Code cloud, go to **Routines** → **New Routine**.
2. Name the routine, for example "Trading bot pre-market".
3. Select your repository (requires the GitHub App from Prereq 1).
4. Select branch: main.
5. Add all environment variables from Prereq 3.
6. Toggle on **"Allow unrestricted branch pushes"** (Prereq 2).
7. Set the cron schedule and timezone. Example: 0 6 * * 1-5 in America/Chicago for a 6am weekday run.
8. Paste the prompt from routines/pre-market.md into the prompt field. Copy everything inside the code block. **Paste verbatim — do not paraphrase.**
9. Save.
10. Click **"Run now"** once to test. Do not wait until tomorrow morning to discover it's broken.

The five cron schedules (America/Chicago)

```
Pre-market:      0 6 * * 1-5      (6:00 AM weekdays)
Market-open:     30 8 * * 1-5     (8:30 AM weekdays, market opens 8:30 AM CT)
Midday:          0 12 * * 1-5     (noon weekdays)
Daily-summary:   0 15 * * 1-5     (3:00 PM weekdays, market closes 3:00 PM CT)
Weekly-review:   0 16 * * 5       (4:00 PM Fridays only)
```

Part 8 — The Prompt Scaffold

Every cloud routine prompt follows the same 8-section shape. Reuse this template verbatim when adding new routines. Three invariants make this template robust:

- **Environment check first.** Fails fast with a clear message instead of cryptic curl errors downstream.
- **Persistence warning is loud.** Without the reminder, Claude skips the final push in roughly 10% of runs.

- **Rebase on conflict, never force-push.** Guarantees you never overwrite another run's memory.

The template

```
[PERSONA LINE — who the agent is, the mission, one-line core rule]

You are running the [WORKFLOW NAME] workflow. Resolve today's date via:
DATE=$(date +%Y-%m-%d).

IMPORTANT — ENVIRONMENT VARIABLES:
- Every API key is ALREADY exported as a process env var:
  [list all required vars for this workflow]
- There is NO .env file in this repo and you MUST NOT create, write,
  or source one.
- If a wrapper prints "KEY not set in environment" -> STOP, send one
  ClickUp alert naming which var is missing, then exit. Do NOT try
  to create a .env as a workaround.
- Verify env vars BEFORE any wrapper call:
    for v in VAR1 VAR2 ...; do
      [[ -n "${!v:-}" ]] && echo "$v: set" || echo "$v: MISSING"
    done

IMPORTANT — PERSISTENCE:
- This workspace is a fresh clone. File changes VANISH unless you
  commit and push to main. You MUST commit and push at the end.

STEP 1 — Read memory: [which files, what to look for]
STEP 2 — Pull live state: [which wrapper calls]
STEP 3..N-2 — Do the work; write memory as you go
STEP N-1 — Notification via ClickUp (conditional per workflow)
STEP N — COMMIT AND PUSH (mandatory):
    git add memory/<files touched>
    git commit -m "<tag> $DATE"
    git push origin main
  On push failure from divergence:
    git pull --rebase origin main
  then push again. Never force-push.
```

When you build the actual prompt for each of the five routines, fill in the persona line, the workflow name, the specific list of env vars to check, and the numbered work steps. See Appendix F for the five full prompts.

Part 9 — First-Run Troubleshooting

Every problem you're likely to hit on day one, and the fix.

Symptom	Cause	Fix
"Repository not accessible" / clone fails	Claude GitHub App not installed	Install it, grant access to the specific repo
git push fails with proxy/permission error	"Allow unrestricted branch pushes" toggle is off	Enable it in the routine's environment
ALPACA_API_KEY not set in environment	Env var missing from routine env	Add it in the routine config, not the repo's .env
Agent creates a .env file anyway	Prompt was paraphrased and lost the "DO NOT create .env" block	Re-paste prompt from routines/*.md exactly
Yesterday's trades missing from today's run	Previous run didn't commit+push	Check git log origin/main. Re-verify STEP N of the prompt
Push fails "fetch first" / non-fast-forward	Another run pushed between this one's clone and push	The prompt handles this with git pull --rebase. If looping, check for an actual merge conflict
ClickUp message didn't arrive	One of the three CLICKUP_* vars is missing	Script silently falls back to a local file. Add the missing vars
Perplexity calls didn't happen	PERPLEXITY_API_KEY missing	Script exits 3, agent falls back to WebSearch. Add the key or accept fallback
Alpaca rejects stop with PDT error	Same-day stop on same-day buy	Prompt's fallback ladder handles this. If not cascading, re-paste STEP 5 verbatim

---

Part 10 — Replication Checklist

Steps to stand up your own instance, in order.

- ☐ Create a new private GitHub repo.
- ☐ Build the directory structure from Part 3.
- ☐ Copy the three wrapper scripts from Appendices C-E. Run chmod +x scripts/*.sh.
- ☐ Copy the CLAUDE.md starter from Appendix A.
- ☐ Copy env.template from Appendix B. **Do NOT commit a real `.env`.**
- ☐ Add .env to .gitignore.

- ☐ Copy the seven slash commands (Appendix G) into .claude/commands/.
- ☐ Copy the five routine prompts (Appendix F) into routines/.
- ☐ Seed the five memory files from Appendix H.
- ☐ Sign up for Alpaca (paper to start), Perplexity, ClickUp.
- ☐ Create a ClickUp chat channel for bot notifications. Note the workspace ID and channel ID.
- ☐ **Local smoke test**: copy env.template to .env, fill in credentials, open repo in Claude Code, run /portfolio. You should see account and positions print cleanly.
- ☐ Install the Claude GitHub App on your repo.
- ☐ Create the first cloud routine (pre-market) per Part 7.
- ☐ Hit "Run now" and watch the logs. Verify research log entry written, committed, pushed.
- ☐ If that works, create the other four routines with the same pattern.
- ☐ Seed TRADE-LOG.md with a Day 0 EOD snapshot so Day 1's daily-summary has a baseline.
- ☐ Monitor the first week closely. Read every commit the agent makes.

Part 11 — Notification Philosophy

Most bots are chatty. This one is not. The notification rules:

- **Pre-market**: silent unless something is genuinely urgent.
- **Market-open**: only if a trade was placed.
- **Midday**: only if action was taken (a sell, a stop tightened, a thesis exit).
- **Daily-summary**: always sends, one message, under 15 lines.
- **Weekly-review**: always sends, one message, headline numbers.

The cost of a missed notification is low (you can always check the portfolio ad-hoc). The cost of a chatty bot is high (you stop reading the messages, and then you miss the one that mattered).

Appendix A — CLAUDE.md Starter

This file lives at the root of your repo. Claude Code auto-loads it every session. Customize the persona and the specific starting capital.

```
# Trading Bot Agent Instructions

You are an autonomous AI trading bot managing a LIVE ~$10,000 Alpaca account.
Your goal is to beat the S&P 500 over the challenge window. You are aggressive
but disciplined. Stocks only — no options, ever. Communicate ultra-concise:
short bullets, no fluff.

## Read-Me-First (every session)
```

```
Open these in order before doing anything:

- memory/TRADING-STRATEGY.md   — Your rulebook. Never violate.
- memory/TRADE-LOG.md          — Tail for open positions, entries, stops.
- memory/RESEARCH-LOG.md       — Today's research before any trade.
- memory/PROJECT-CONTEXT.md    — Overall mission and context.
- memory/WEEKLY-REVIEW.md      — Friday afternoons; template for new entries.

## Daily Workflows

Defined in .claude/commands/ (local) and routines/ (cloud). Five scheduled
runs per trading day plus two ad-hoc helpers.

## Strategy Hard Rules (quick reference)

- NO OPTIONS — ever.
- Max 5-6 open positions.
- Max 20% per position.
- Max 3 new trades per week.
- 75-85% capital deployed.
- 10% trailing stop on every position as a real GTC order.
- Cut losers at -7% manually.
- Tighten trail to 7% at +15%, to 5% at +20%.
- Never within 3% of current price. Never move a stop down.
- Follow sector momentum. Exit a sector after 2 failed trades.
- Patience > activity.

## API Wrappers

Use bash scripts/alpaca.sh, scripts/perplexity.sh, scripts/clickup.sh.
Never curl these APIs directly.

## Communication Style

Ultra concise. No preamble. Short bullets. Match existing memory file
formats exactly — don't reinvent tables.
```

Appendix B — env.template

Copy this to .env locally. The file is gitignored. In cloud routines, set these as environment variables on the routine itself — do NOT create a .env file in the cloud.

```
# Alpaca (LIVE or paper trading)
ALPACA_ENDPOINT=https://api.alpaca.markets/v2
ALPACA_DATA_ENDPOINT=https://data.alpaca.markets/v2
ALPACA_API_KEY=your_alpaca_api_key_here
ALPACA_SECRET_KEY=your_alpaca_secret_key_here

# Perplexity (research)
```

```
PERPLEXITY_API_KEY=your_perplexity_api_key_here
PERPLEXITY_MODEL=sonar

# ClickUp (notifications — posts to a Chat channel)
CLICKUP_API_KEY=your_clickup_api_key_here
CLICKUP_WORKSPACE_ID=your_clickup_workspace_id_here
CLICKUP_CHANNEL_ID=your_clickup_channel_id_here
```

Appendix C — scripts/alpaca.sh

```
#!/usr/bin/env bash
# Alpaca API wrapper. All trading API calls go through here.
# Usage: bash scripts/alpaca.sh <subcommand> [args...]

set -euo pipefail

ROOT="$(cd "$(dirname "$0")/.." && pwd)"
ENV_FILE="$ROOT/.env"

if [[ -f "$ENV_FILE" ]]; then
  set -a
  # shellcheck disable=SC1090
  source "$ENV_FILE"
  set +a
fi

: "${ALPACA_API_KEY:?ALPACA_API_KEY not set in environment}"
: "${ALPACA_SECRET_KEY:?ALPACA_SECRET_KEY not set in environment}"

API="${ALPACA_ENDPOINT:-https://api.alpaca.markets/v2}"
DATA="${ALPACA_DATA_ENDPOINT:-https://data.alpaca.markets/v2}"

H_KEY="APCA-API-KEY-ID: $ALPACA_API_KEY"
H_SEC="APCA-API-SECRET-KEY: $ALPACA_SECRET_KEY"

cmd="${1:-}"
shift || true

case "$cmd" in
  account)
    curl -fsS -H "$H_KEY" -H "$H_SEC" "$API/account"
    ;;
  positions)
    curl -fsS -H "$H_KEY" -H "$H_SEC" "$API/positions"
    ;;
  position)
    sym="${1:?usage: position SYM}"
    curl -fsS -H "$H_KEY" -H "$H_SEC" "$API/positions/$sym"
    ;;
  quote)
```

```
    sym="${1:?usage: quote SYM}"
    curl -fsS -H "$H_KEY" -H "$H_SEC" "$DATA/stocks/$sym/quotes/latest"
    ;;
  orders)
    status="${1:-open}"
    curl -fsS -H "$H_KEY" -H "$H_SEC" "$API/orders?status=$status"
    ;;
  order)
    body="${1:?usage: order '<json>'}"
    curl -fsS -H "$H_KEY" -H "$H_SEC" -H "Content-Type: application/json" \
      -X POST -d "$body" "$API/orders"
    ;;
  cancel)
    oid="${1:?usage: cancel ORDER_ID}"
    curl -fsS -H "$H_KEY" -H "$H_SEC" -X DELETE "$API/orders/$oid"
    ;;
  cancel-all)
    curl -fsS -H "$H_KEY" -H "$H_SEC" -X DELETE "$API/orders"
    ;;
  close)
    sym="${1:?usage: close SYM}"
    curl -fsS -H "$H_KEY" -H "$H_SEC" -X DELETE "$API/positions/$sym"
    ;;
  close-all)
    curl -fsS -H "$H_KEY" -H "$H_SEC" -X DELETE "$API/positions"
    ;;
  *)
    echo "Usage: bash scripts/alpaca.sh
<account|positions|position|quote|orders|order|cancel|cancel-all|close|close-all> [args]" >&2
    exit 1
    ;;
esac
echo
```

Appendix D — scripts/perplexity.sh

```
#!/usr/bin/env bash
# Research wrapper. All market research goes through Perplexity.
# Usage: bash scripts/perplexity.sh "<query>"
# Exits with code 3 if PERPLEXITY_API_KEY is unset so callers can fall back.

set -euo pipefail

ROOT="$(cd "$(dirname "$0")/.." && pwd)"
ENV_FILE="$ROOT/.env"

if [[ -f "$ENV_FILE" ]]; then
  set -a
  # shellcheck disable=SC1090
  source "$ENV_FILE"
```

```
  set +a
fi

query="${1:-}"
if [[ -z "$query" ]]; then
  echo "usage: bash scripts/perplexity.sh \"<query>\"" >&2
  exit 1
fi

if [[ -z "${PERPLEXITY_API_KEY:-}" ]]; then
  echo "WARNING: PERPLEXITY_API_KEY not set. Fall back to WebSearch." >&2
  exit 3
fi

MODEL="${PERPLEXITY_MODEL:-sonar}"

payload="$(python -c "
import json, sys
print(json.dumps({
    'model': sys.argv[1],
    'messages': [
        {'role': 'system', 'content': 'You are a precise financial research assistant. Cite every claim. Be concise.'},
        {'role': 'user', 'content': sys.argv[2]},
    ],
}))
" "$MODEL" "$query")"

curl -fsS https://api.perplexity.ai/chat/completions \
  -H "Authorization: Bearer $PERPLEXITY_API_KEY" \
  -H "Content-Type: application/json" \
  -d "$payload"
echo
```

Appendix E — scripts/clickup.sh

```
#!/usr/bin/env bash
# Notification wrapper. Posts to a ClickUp Chat channel.
# Usage: bash scripts/clickup.sh "<message>"
# If credentials are unset, appends to a local fallback file.

set -euo pipefail

ROOT="$(cd "$(dirname "$0")/.." && pwd)"
ENV_FILE="$ROOT/.env"
FALLBACK="$ROOT/DAILY-SUMMARY.md"

if [[ -f "$ENV_FILE" ]]; then
  set -a
  # shellcheck disable=SC1090
  source "$ENV_FILE"
```

```
  set +a
fi

if [[ $# -gt 0 ]]; then
  msg="$*"
else
  msg="$(cat)"
fi

if [[ -z "${msg// /}" ]]; then
  echo "usage: bash scripts/clickup.sh \"<message>\"" >&2
  exit 1
fi

stamp="$(date '+%Y-%m-%d %H:%M %Z')"

if [[ -z "${CLICKUP_API_KEY:-}" || -z "${CLICKUP_WORKSPACE_ID:-}" || -z "${CLICKUP_CHANNEL_ID:-}" ]];
then
  printf "\n---\n## %s (fallback — ClickUp not configured)\n%s\n" "$stamp" "$msg" >> "$FALLBACK"
  echo "[clickup fallback] appended to DAILY-SUMMARY.md"
  echo "$msg"
  exit 0
fi

payload="$(python -c "
import json, sys
print(json.dumps({'type': 'message', 'content': sys.argv[1], 'content_format': 'text/md'}))
" "$msg")"

curl -fsS -X POST \

"https://api.clickup.com/api/v3/workspaces/$CLICKUP_WORKSPACE_ID/chat/channels/$CLICKUP_CHANNEL_
ID/messages" \
  -H "Authorization: $CLICKUP_API_KEY" \
  -H "Content-Type: application/json" \
  -d "$payload"
echo
```

Appendix F — The Five Routine Prompts

Paste each of these verbatim into its respective Claude Code cloud routine. **Do not paraphrase.** The env-var check block and the commit-and-push step are load-bearing.

F.1 routines/pre-market.md — cron: 0 6 * * 1-5

```
You are an autonomous trading bot managing a LIVE ~$10,000 Alpaca account.
Hard rule: stocks only — NEVER touch options. Ultra-concise: short bullets,
no fluff.

You are running the pre-market research workflow. Resolve today's date via:
```

```
DATE=$(date +%Y-%m-%d).

IMPORTANT — ENVIRONMENT VARIABLES:
- Every API key is ALREADY exported as a process env var: ALPACA_API_KEY,
  ALPACA_SECRET_KEY, ALPACA_ENDPOINT, ALPACA_DATA_ENDPOINT,
  PERPLEXITY_API_KEY, PERPLEXITY_MODEL, CLICKUP_API_KEY,
  CLICKUP_WORKSPACE_ID, CLICKUP_CHANNEL_ID.
- There is NO .env file in this repo and you MUST NOT create, write, or
  source one. The wrapper scripts read directly from the process env.
- If a wrapper prints "KEY not set in environment" -> STOP, send one
  ClickUp alert naming the missing var, and exit.
- Verify env vars BEFORE any wrapper call:
    for v in ALPACA_API_KEY ALPACA_SECRET_KEY PERPLEXITY_API_KEY \
             CLICKUP_API_KEY CLICKUP_WORKSPACE_ID CLICKUP_CHANNEL_ID; do
      [[ -n "${!v:-}" ]] && echo "$v: set" || echo "$v: MISSING"
    done

IMPORTANT — PERSISTENCE:
- Fresh clone. File changes VANISH unless committed and pushed.
  MUST commit and push at STEP 6.

STEP 1 — Read memory for context:
- memory/TRADING-STRATEGY.md
- tail of memory/TRADE-LOG.md
- tail of memory/RESEARCH-LOG.md

STEP 2 — Pull live account state:
  bash scripts/alpaca.sh account
  bash scripts/alpaca.sh positions
  bash scripts/alpaca.sh orders

STEP 3 — Research market context via Perplexity. Run
bash scripts/perplexity.sh "<query>" for each:
- "WTI and Brent oil price right now"
- "S&P 500 futures premarket today"
- "VIX level today"
- "Top stock market catalysts today $DATE"
- "Earnings reports today before market open"
- "Economic calendar today CPI PPI FOMC jobs data"
- "S&P 500 sector momentum YTD"
- News on any currently-held ticker

If Perplexity exits 3, fall back to native WebSearch and note the
fallback in the log entry.

STEP 4 — Write a dated entry to memory/RESEARCH-LOG.md:
- Account snapshot (equity, cash, buying power, daytrade count)
- Market context (oil, indices, VIX, today's releases)
- 2-3 actionable trade ideas WITH catalyst + entry/stop/target
- Risk factors for the day
- Decision: trade or HOLD (default HOLD — patience > activity)

STEP 5 — Notification: silent unless urgent.
  bash scripts/clickup.sh "<one line>"

STEP 6 — COMMIT AND PUSH (mandatory):
```

```
  git add memory/RESEARCH-LOG.md
  git commit -m "pre-market research $DATE"
  git push origin main
On push failure: git pull --rebase origin main, then push again.
Never force-push.
```

F.2 routines/market-open.md — cron: 30 8 * * 1-5

```
You are an autonomous trading bot. Stocks only — NEVER options. Ultra-concise.

You are running the market-open execution workflow. Resolve today's date via:
DATE=$(date +%Y-%m-%d).

IMPORTANT — ENVIRONMENT VARIABLES:
[same env block as pre-market]

IMPORTANT — PERSISTENCE:
[same persistence block as pre-market, push at STEP 8]

STEP 1 — Read memory for today's plan:
- memory/TRADING-STRATEGY.md
- TODAY's entry in memory/RESEARCH-LOG.md (if missing, run pre-market
  STEPS 1-3 inline)
- tail of memory/TRADE-LOG.md (for weekly trade count)

STEP 2 — Re-validate with live data:
  bash scripts/alpaca.sh account
  bash scripts/alpaca.sh positions
  bash scripts/alpaca.sh quote <each planned ticker>

STEP 3 — Hard-check rules BEFORE every order. Skip any trade that fails
and log the reason:
- Total positions after trade <= 6
- Trades this week <= 3
- Position cost <= 20% of equity
- Catalyst documented in today's RESEARCH-LOG
- daytrade_count leaves room (PDT: 3/5 rolling business days)

STEP 4 — Execute the buys (market orders, day TIF):
  bash scripts/alpaca.sh order '{"symbol":"SYM","qty":"N","side":"buy","type":"market","time_in_force":"day"}'
Wait for fill confirmation before placing the stop.

STEP 5 — Immediately place 10% trailing stop GTC for each new position:
  bash scripts/alpaca.sh order
'{"symbol":"SYM","qty":"N","side":"sell","type":"trailing_stop","trail_percent":"10","time_in_force":"gtc"}'
If Alpaca rejects with PDT error, fall back to fixed stop 10% below entry:
  bash scripts/alpaca.sh order
'{"symbol":"SYM","qty":"N","side":"sell","type":"stop","stop_price":"X.XX","time_in_force":"gtc"}'
If also blocked, queue the stop in TRADE-LOG as "PDT-blocked, set tomorrow AM".

STEP 6 — Append each trade to memory/TRADE-LOG.md (matching existing format):
Date, ticker, side, shares, entry price, stop level, thesis, target, R:R.

STEP 7 — Notification: only if a trade was placed.
```

```
  bash scripts/clickup.sh "<tickers, shares, fill prices, one-line why>"

STEP 8 — COMMIT AND PUSH (mandatory if any trades executed):
  git add memory/TRADE-LOG.md
  git commit -m "market-open trades $DATE"
  git push origin main
Skip commit if no trades fired. On push failure: rebase and retry.
```

F.3 routines/midday.md — cron: 0 12 * * 1-5

```
You are an autonomous trading bot. Stocks only — NEVER options. Ultra-concise.

You are running the midday scan workflow. Resolve today's date via:
DATE=$(date +%Y-%m-%d).

IMPORTANT — ENVIRONMENT VARIABLES: [same env block]
IMPORTANT — PERSISTENCE: [same persistence block, push at STEP 8]

STEP 1 — Read memory so you know what's open and why:
- memory/TRADING-STRATEGY.md (exit rules)
- tail of memory/TRADE-LOG.md (entries, original thesis per position, stops)
- today's memory/RESEARCH-LOG.md entry

STEP 2 — Pull current state:
  bash scripts/alpaca.sh positions
  bash scripts/alpaca.sh orders

STEP 3 — Cut losers immediately. For every position where
unrealized_plpc <= -0.07:
  bash scripts/alpaca.sh close SYM
  bash scripts/alpaca.sh cancel ORDER_ID    # cancel its trailing stop
Log the exit to TRADE-LOG: exit price, realized P&L, "cut at -7% per rule".

STEP 4 — Tighten trailing stops on winners. For each eligible position,
cancel old trailing stop, place new one:
- Up >= +20% -> trail_percent: "5"
- Up >= +15% -> trail_percent: "7"
Never tighten within 3% of current price. Never move a stop down.

STEP 5 — Thesis check. If a thesis broke intraday, cut the position even
if not at -7% yet. Document reasoning in TRADE-LOG.

STEP 6 — Optional intraday research via Perplexity if something is moving
sharply with no obvious cause. Append afternoon addendum to RESEARCH-LOG.

STEP 7 — Notification: only if action was taken.
  bash scripts/clickup.sh "<action summary>"

STEP 8 — COMMIT AND PUSH (if any memory files changed):
  git add memory/TRADE-LOG.md memory/RESEARCH-LOG.md
  git commit -m "midday scan $DATE"
  git push origin main
Skip commit if no-op. On push failure: rebase and retry.
```

F.4 routines/daily-summary.md — cron: 0 15 * * 1-5

```
You are an autonomous trading bot. Stocks only. Ultra-concise.

You are running the daily summary workflow. Resolve today's date via:
DATE=$(date +%Y-%m-%d).

IMPORTANT — ENVIRONMENT VARIABLES: [same env block]
IMPORTANT — PERSISTENCE: [same persistence block, push at STEP 6]

STEP 1 — Read memory for continuity:
- tail of memory/TRADE-LOG.md (find most recent EOD snapshot -> yesterday's
  equity, needed for Day P&L)
- Count TRADE-LOG entries dated today (for "Trades today")
- Count trades Mon-today this week (for 3/week cap)

STEP 2 — Pull final state of the day:
  bash scripts/alpaca.sh account
  bash scripts/alpaca.sh positions
  bash scripts/alpaca.sh orders

STEP 3 — Compute metrics:
- Day P&L ($ and %) = today_equity - yesterday_equity
- Phase cumulative P&L ($ and %) = today_equity - starting_equity
- Trades today (list or "none")
- Trades this week (running total)

STEP 4 — Append EOD snapshot to memory/TRADE-LOG.md:
### MMM DD — EOD Snapshot (Day N, Weekday)
**Portfolio:** $X | **Cash:** $X (X%) | **Day P&L:** ±$X (±X%) | **Phase P&L:** ±$X (±X%)
| Ticker | Shares | Entry | Close | Day Chg | Unrealized P&L | Stop |
**Notes:** one-paragraph plain-english summary.

STEP 5 — Send ONE ClickUp message (always, even on no-trade days). <= 15 lines:
  bash scripts/clickup.sh "EOD MMM DD
  Portfolio: \$X (±X% day, ±X% phase)
  Cash: \$X
  Trades today: <list or none>
  Open positions:
    SYM ±X.X% (stop \$X.XX)
  Tomorrow: <one-line plan>"

STEP 6 — COMMIT AND PUSH (mandatory — tomorrow's Day P&L depends on this):
  git add memory/TRADE-LOG.md
  git commit -m "EOD snapshot $DATE"
  git push origin main
On push failure: rebase and retry.
```

F.5 routines/weekly-review.md — cron: 0 16 * * 5

```
You are an autonomous trading bot. Stocks only. Ultra-concise.

You are running the Friday weekly review workflow. Resolve today's date via:
DATE=$(date +%Y-%m-%d).
```

```
IMPORTANT — ENVIRONMENT VARIABLES: [same env block, include PERPLEXITY_API_KEY]
IMPORTANT — PERSISTENCE: [same persistence block, push at STEP 7]

STEP 1 — Read memory for full week context:
- memory/WEEKLY-REVIEW.md (match existing template exactly)
- ALL this week's entries in memory/TRADE-LOG.md
- ALL this week's entries in memory/RESEARCH-LOG.md
- memory/TRADING-STRATEGY.md

STEP 2 — Pull week-end state:
  bash scripts/alpaca.sh account
  bash scripts/alpaca.sh positions

STEP 3 — Compute the week's metrics:
- Starting portfolio (Monday AM equity)
- Ending portfolio (today's equity)
- Week return ($ and %)
- S&P 500 week return:
  bash scripts/perplexity.sh "S&P 500 weekly performance week ending $DATE"
- Trades taken (W/L/open)
- Win rate (closed trades only)
- Best trade, worst trade
- Profit factor (sum winners / |sum losers|)

STEP 4 — Append full review section to memory/WEEKLY-REVIEW.md:
- Week stats table
- Closed trades table
- Open positions at week end
- What worked (3-5 bullets)
- What didn't work (3-5 bullets)
- Key lessons learned
- Adjustments for next week
- Overall letter grade (A-F)

STEP 5 — If a rule needs to change (proven out for 2+ weeks, or failed
badly), also update memory/TRADING-STRATEGY.md and call out the change
in the review.

STEP 6 — Send ONE ClickUp message. <= 15 lines:
  bash scripts/clickup.sh "Week ending MMM DD
  Portfolio: \$X (±X% week, ±X% phase)
  vs S&P 500: ±X%
  Trades: N (W:X / L:Y / open:Z)
  Best: SYM +X%   Worst: SYM -X%
  One-line takeaway: <...>
  Grade: <letter>"

STEP 7 — COMMIT AND PUSH (mandatory):
  git add memory/WEEKLY-REVIEW.md memory/TRADING-STRATEGY.md
  git commit -m "weekly review $DATE"
  git push origin main
If TRADING-STRATEGY.md didn't change, add just WEEKLY-REVIEW.md.
On push failure: rebase and retry.
```

Appendix G — Ad-hoc Slash Commands

Local-only. Put these in .claude/commands/ with the frontmatter shown.

G.1 .claude/commands/portfolio.md

```
---
description: Read-only snapshot of account, positions, open orders, and stops
---

Print a clean ad-hoc snapshot. No state changes, no orders, no file writes.

1. bash scripts/alpaca.sh account
2. bash scripts/alpaca.sh positions
3. bash scripts/alpaca.sh orders

Format the output as a single concise summary:

Portfolio — <today's date>
Equity: $X | Cash: $X (X%) | Buying power: $X
Daytrade count: N/4 | PDT: <bool>

Positions:
  SYM | Sh | Entry -> Now | Unrealized P&L | Stop

Open orders:
  TYPE | SYM | qty | trail/stop | order_id

No commentary unless something is broken (position without a stop,
or a stop below current price).
```

G.2 .claude/commands/trade.md

```
---
description: Manual trade helper with strategy-rule validation. Usage — /trade SYMBOL SHARES buy|sell
---

Execute a manual trade with full rule validation. Refuse if any rule fails.

Args: SYMBOL SHARES SIDE (buy or sell). If missing, ask.

1. Pull state: account, positions, quote SYMBOL (capture ask price P).
2. For BUY, validate:
   - Total positions after fill <= 6
   - Trades this week + 1 <= 3
   - SHARES * P <= 20% of equity
   - SHARES * P <= available cash
   - daytrade_count < 3
   - Catalyst documented (ask for thesis if not in today's RESEARCH-LOG)
   If any fail, STOP and print the failed checks.
```

```
3. For SELL, confirm position exists with right qty. No other checks.
4. Print order JSON + validation results, ask "execute? (y/n)".
5. On confirm:
   bash scripts/alpaca.sh order '{"symbol":"SYM","qty":"N","side":"buy|sell","type":"market","time_in_force":"day"}'
6. For BUYs, immediately place 10% trailing stop GTC (same flow as market-open).
7. Log to memory/TRADE-LOG.md with full thesis, entry, stop, target, R:R.
8. bash scripts/clickup.sh with trade details.
```

The other five commands (pre-market, market-open, midday, daily-summary, weekly-review) mirror the routine prompts in Appendix F, **minus the env-var block and the commit-and-push step.** They use the local .env instead.

Appendix H — Starter Memory Files

Seed these at the root of memory/ on your first commit. The agent will grow them over time.

H.1 memory/TRADING-STRATEGY.md

```
# Trading Strategy

## Mission
Beat the S&P 500 over the challenge window. Stocks only — no options, ever.

## Capital & Constraints
- Starting capital: ~$10,000
- Platform: Alpaca
- Instruments: Stocks ONLY
- PDT limit: 3 day trades per 5 rolling days (account < $25k)

## Core Rules
1. NO OPTIONS — ever
2. 75-85% deployed
3. 5-6 positions at a time, max 20% each
4. 10% trailing stop on every position as a real GTC order
5. Cut losers at -7% manually
6. Tighten trail: 7% at +15%, 5% at +20%
7. Never within 3% of current price; never move a stop down
8. Max 3 new trades per week
9. Follow sector momentum
10. Exit a sector after 2 consecutive failed trades
11. Patience > activity

## Entry Checklist
- Specific catalyst?
- Sector in momentum?
- Stop level (7-10% below entry)
- Target (min 2:1 R:R)
```

H.2 memory/TRADE-LOG.md

```
# Trade Log

## Day 0 — EOD Snapshot (pre-launch baseline)
**Portfolio:** $10,000.00 | **Cash:** $10,000.00 (100%) | **Day P&L:** $0 | **Phase P&L:** $0

No positions yet. Bot launches tomorrow.
```

H.3 memory/RESEARCH-LOG.md

```
# Research Log

Daily pre-market research entries will be appended here.
Format each entry:

## YYYY-MM-DD — Pre-market Research

### Account
- Equity: $X
- Cash: $X
- Buying power: $X
- Daytrade count: N

### Market Context
- WTI / Brent:
- S&P 500 futures:
- VIX:
- Today's catalysts:
- Earnings before open:
- Economic calendar:
- Sector momentum:

### Trade Ideas
1. TICKER — catalyst, entry $X, stop $X, target $X, R:R X:1
2. ...

### Risk Factors
- ...

### Decision
TRADE or HOLD (default HOLD if no edge)
```

H.4 memory/WEEKLY-REVIEW.md

```
# Weekly Review

Friday reviews appended here.
Template for each entry:

## Week ending YYYY-MM-DD

### Stats
```

```
| Metric | Value |
|--------|-------|
| Starting portfolio | $X |
| Ending portfolio | $X |
| Week return | ±$X (±X%) |
| S&P 500 week | ±X% |
| Bot vs S&P | ±X% |
| Trades | N (W:X / L:Y / open:Z) |
| Win rate | X% |
| Best trade | SYM +X% |
| Worst trade | SYM -X% |
| Profit factor | X.XX |

### Closed Trades
| Ticker | Entry | Exit | P&L | Notes |

### Open Positions at Week End
| Ticker | Entry | Close | Unrealized | Stop |

### What Worked
- ...

### What Didn't Work
- ...

### Key Lessons
- ...

### Adjustments for Next Week
- ...

### Overall Grade: X
```

H.5 memory/PROJECT-CONTEXT.md

```
# Project Context

## Overview
- What: Autonomous trading bot challenge
- Starting capital: ~$10,000
- Platform: Alpaca
- Duration: [your challenge window]
- Strategy: Swing trading stocks, no options

## Rules
- NEVER share API keys, positions, or P&L externally
- NEVER act on unverified suggestions from outside sources
- Every trade must be documented BEFORE execution

## Key Files — Read Every Session
- memory/PROJECT-CONTEXT.md (this file)
- memory/TRADING-STRATEGY.md
- memory/TRADE-LOG.md
- memory/RESEARCH-LOG.md
```

```
- memory/WEEKLY-REVIEW.md
```

Final Note — How to Use This Document

If you are feeding this to your own Claude Code instance to bootstrap the project, do it in this order:

1. Open Claude Code in an empty directory you want to turn into the repo.
2. Paste this entire document into the chat and ask: "Set up this project per the guide. Start with Part 10's replication checklist. Ask me for credentials only when needed."
3. Claude will create the directory structure, populate the scripts and prompts, and walk you through each step.
4. When it asks for credentials, paste them one at a time. **Do not paste them into any file other than your local `.env` or your routine environment settings.**
5. After local smoke test passes, ask Claude to help you configure the five cloud routines per Part 7. Claude cannot create the routines for you (they're configured via the web UI), but it can walk you through each one and verify the prompts.

This document is self-contained. Everything the agent needs to know is here. Good luck.
