PROMPTS & SKILLS FOR CLAUDE CODE

A thirteen-week plan for a workflow that survives the next model update.

Terminal Craft

INDEPENDENT FIELD GUIDE

The Multi-AI Workflow

A thirteen-week plan for a workflow that survives the next model update.

TERMINAL CRAFT

WELCOME

Before You Begin

You have rebuilt your setup at least twice this year. Every release brings a wave of advice, some of it good, and it is tempting to tear everything down and start again with the new thing. The cost of that is invisible and large: you never accumulate any evidence about what was working, because nothing survives long enough to be measured. So each rebuild is made on impressions rather than results, which is why the next one is always six weeks away. This is a thirteen-week plan for getting off that treadmill. Month one measures and changes nothing. It is the month people skip, and the reason most of their conclusions are unreliable. Month two adds one change per week in a fixed order, so every change can be attributed. Month three hardens what survived against the next model update, and then deliberately stops. There is also a log that argues back: recorded time and rework rather than impressions, so a setup that feels fast but produces re-dos gets caught instead of defended. Two notes. This is an independent publication with no affiliation to any AI vendor, and no software is included. And it is the 2026 Edition, which in this guide is the whole point: the plan is designed so that a model update costs you an afternoon of adjustment rather

than a rebuild, because the parts that move are deliberately kept separate from the parts that do not.

— The Terminal Craft Team

HOW TO USE THIS GUIDE

Getting the Most From This Book

This is a reference, not a reading list. The prompts work best adopted a few at a time until they are automatic. Here's how to get the most from the pages ahead.

1. **Use it as a reference.**
   Every chapter ends with a *Try This* exercise and a set of *Key Takeaways*. Read the chapter once, then come back to the prompts when you need them.

2. **Adopt a few at a time.**
   Take four prompts and use them until they are automatic. Reaching for forty at once is the fastest way to use none of them.

3. **Read the 'not for' line.**
   A structured prompt on a trivial task is pure overhead. Knowing when not to use one matters as much as knowing the wording.

4 **Adapt, don't discard.**

When a model version changes and a prompt stops landing, rebuild it from the four-part frame rather than hunting for a new download.

Whenever you're ready, turn the page and let's begin.

CONTENTS

What's Inside

CHAPTER 1

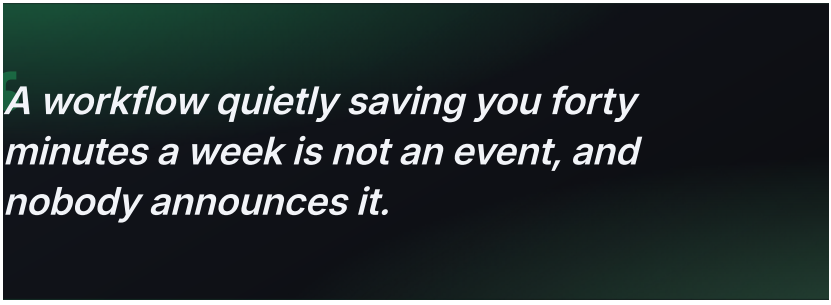
Why Setups Keep Collapsing

> Three forces push toward the rebuild, and none of them are about your setup being bad. Recognising them is what makes it possible to resist at the right moments and give in at the right ones.

Novelty is louder than results

A new release is an event. A workflow quietly saving you forty minutes a week is not an event, and nobody writes an announcement about it. So attention flows to the new thing regardless of whether the current thing is working, and the comparison is between a demo and your ordinary Tuesday.

That comparison is unfair in a specific way: the demo shows the new tool's best case against your setup's average case. Almost anything wins that.



Without a log, everything is impressions

If you have no record, the only evidence available is how things felt, and feelings about tools are dominated by the most recent frustration and the most recent delight. A single bad session can convince you a setup is failing when the log would show it saving time on nineteen of twenty tasks.

This is the whole reason month one exists. Not because measuring is virtuous, but because without it you will be arguing with yourself using anecdotes.

Coupling makes updates expensive

The other collapse is structural. If your workflow depends on a specific version's quirks, every update breaks something: an exact file path, a particular selection behaviour, a prompt that only works with one model's phrasing. After two or three of those a rebuild feels easier than repair.

Keeping the durable parts separate from the mechanism is what turns an update into an afternoon. Month three is about identifying exactly which parts those are.

UNDERSTANDING YOUR OWN PATTERN

The Rebuild Audit

Ask me what I changed about my AI setup in the last twelve months and why. Then tell me which changes were driven by a measured problem and which by a new release.

WHY IT WORKS

Separating measured problems from novelty makes the treadmill visible, and it is invisible from inside.

USE WHEN Before starting the plan **PAIRS WITH** The Baseline Question

A first setup with no history.

KNOWING WHAT YOU ARE COMPARING AGAINST

The Baseline Question

Before I change anything: what does my current setup actually do well? Name three things, and how I would notice if they stopped.

WHY IT WORKS

Naming what works gives you something to protect, which is what stops a rebuild throwing out the good parts by accident.

USE WHEN Day one **PAIRS WITH** The Rebuild Audit

Setups you have already decided to abandon entirely.

Try This: Count Your Rebuilds

The pattern is easier to break once you have seen it written down.

1. List every significant change you made to your AI workflow this year.
2. Mark each as: solved a measured problem, or followed a release.
3. Count the second group.
4. That number is what this plan is designed to reduce.

KEY TAKEAWAYS

- A demo shows the new tool's best case against your setup's average case.
- Without a log, you argue with yourself using your two most recent sessions.
- Coupling to a version's quirks is what makes updates feel like rebuilds.

CHAPTER 2

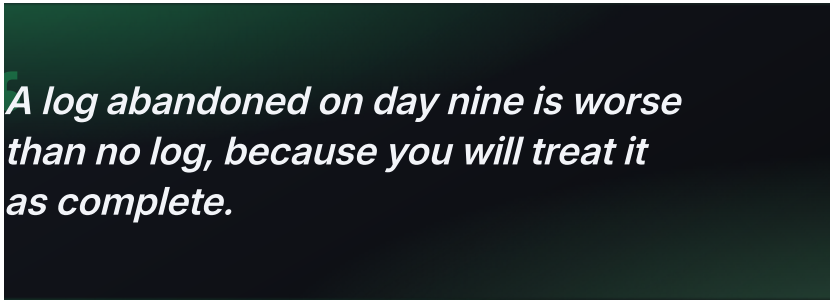
Month One: Measure

> Weeks one to four have one job and it is not improvement. You change nothing and record everything, and the discipline of that is harder than any other part of the plan.

The log is four columns

Task type, tool used, minutes taken, and whether you had to redo it. Nothing else. A richer log gets abandoned around day nine, and a log abandoned on day nine is worse than no log at all because it produces a partial picture you will treat as complete.

Ten seconds per task. If it takes longer than that, cut a column.



Change nothing, including the obvious things

You will spot something worth fixing in week one. Write it down and do not fix it. The point of the baseline is that it describes your actual normal, and a baseline you improved halfway through describes neither the old state nor the new one.

Keep a separate list of everything you wanted to change. That list becomes month two's queue, ordered by evidence rather than by whichever annoyance was loudest.

What the four weeks will tell you

Where the time actually goes, which is almost never where people expect. Which tasks generate rework, which is the real cost nobody tracks. And whether the frustration you feel is concentrated in one type of work or spread evenly.

That third one matters most. Concentrated frustration is fixable with one change; evenly spread frustration usually means the process rather than the tool.

WEEK ONE SETUP

The Four-Column Log

Give me a four-column log format (type, tool, minutes, redone) and a worked example row. Nothing else, and no additional columns however useful they sound.

WHY IT WORKS

Constraint is what makes a log survive four weeks, and survival is worth more than richness.

USE WHEN Day one **PAIRS WITH** The Wish List

Anyone unwilling to record for four full weeks.

PARKING IMPROVEMENTS WITHOUT MAKING THEM

The Wish List

I want to change something but I am in the measurement month. Add it to my queue with the specific problem it would solve and how I would know it worked.

WHY IT WORKS

Writing the change down with a success condition converts an urge into a testable item for month two.

USE WHEN Whenever the urge appears **PAIRS WITH** The Four-Column Log

Genuine blockers that stop you working at all.

END OF WEEK FOUR

The Baseline Read

Here are four weeks of logs. Where does my time actually go, which type produces the most rework, and is my frustration concentrated or spread?

WHY IT WORKS

Concentrated frustration is one change away from fixed; spread frustration is a process problem, and the responses differ entirely.

USE WHEN Day 29 **PAIRS WITH** The Four-Column Log

Partial logs, which mislead more than they inform.

Try This: Park Everything

The urge to fix things during measurement is the main failure mode of month one.

1. Start the four-column log today.
2. Keep a second list titled 'month two queue'.
3. Every time you want to change something, write it there with the problem it solves.
4. Change nothing until day 29.

KEY TAKEAWAYS

- Four columns, ten seconds a task. Anything richer gets abandoned by day nine.
- Park every improvement. A baseline you improved halfway describes nothing.
- Concentrated frustration is one change away. Spread frustration is the process.

CHAPTER 3

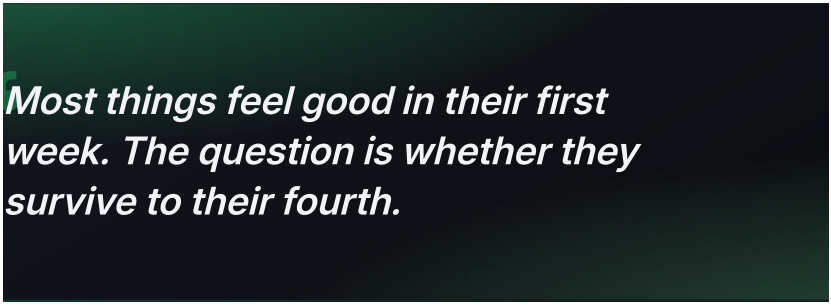
Month Two: Build

> Weeks five to eight. One change per week, in a fixed order, with the log still running, so that every change can be judged on its own rather than as part of a bundle nobody can untangle.

The order is deliberate

Week five: install your project standards, because they are the cheapest change and they affect every session afterwards. Week six: add the review handoff, since it is the control most people lack. Week seven: add or change one tool, guided by the bench test. Week eight: nothing new, just consolidate.

Standards before tools is the important part. Adding a tool to an unstandardised workflow means you cannot tell whether an improvement came from the tool or from finally writing down what you wanted.



One change, one week, no exceptions

Two changes in a week means neither can be attributed, and attribution is the entire reason this plan takes thirteen weeks instead of a weekend. The temptation is strongest in week five, when the first change works and momentum feels available.

Write the change and its date clearly in the log. Future you reading the log needs to know exactly when each thing started.

Week eight is not a wasted week

A week with no changes is what lets the previous three settle into ordinary use. Most things feel good in their first week. The interesting question is whether they are still being used in their fourth, and only an unchanged week can answer it.

It is also when you find out which changes quietly stopped being followed, which is information you would never get

while still adding things.

THE CHEAPEST CHANGE, MADE FIRST

The Week Five Standards

From my month-one log, which repeated instructions or recurring rework would a project standard have prevented? Draft the two most valuable ones.

WHY IT WORKS

Deriving standards from logged rework grounds them in evidence rather than in what feels tidy.

USE WHEN Week five **PAIRS WITH** The Change Marker

Projects that already have well-tested standards.

KEEPING ATTRIBUTION POSSIBLE

The Change Marker

Record this in my log as a dated change with: what changed, what problem it addresses, and what I expect to see in the numbers if it works.

WHY IT WORKS

A stated expectation is what makes the following week's data interpretable rather than just present.

USE WHEN Every change in month two **PAIRS WITH** The Week Five Standards

Trivial adjustments with no expected effect.

WEEK EIGHT

The Consolidation Read

Three changes are now in. For each, compare the two weeks before and after it. Which produced a measurable difference and which did not?

WHY IT WORKS

Before-and-after per change is only possible because they were made one at a time, and it is the payoff for that patience.

USE WHEN End of week eight **PAIRS WITH** The Change Marker

Weeks where several changes overlapped.

Try This: Resist Week Five

The first change working is exactly when the plan is most at risk.

1. Make one change in week five and mark the date in the log.
2. When it works and you want to add three more, write them in the queue.
3. Wait for week six.
4. At week eight, compare each change against the two weeks before it.

KEY TAKEAWAYS

- Standards before tools, or you cannot tell which produced the improvement.
- One change per week, dated in the log, with a stated expectation.
- The no-change week is what reveals which changes stopped being followed.

CHAPTER 4

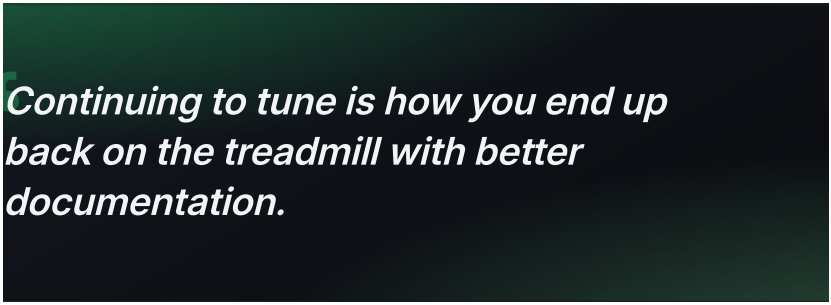
Month Three: Harden

> Weeks nine to thirteen. The setup works; now make it survive the next update. This is the chapter that decides whether you are back here in six months.

List what breaks on an update

Write down every part of your workflow that depends on something outside your control: file locations, selection behaviour, a specific model's phrasing, a tool's output format. That list is your fragility surface, and most people have never written it down.

Then fix the top two. Not all of them. The top two, by how much would break and how likely the change is.



Separate content from mechanism

Your standards, your review questions, your prompt structures are content: they express decisions you made and they do not expire. Where files live and how selection works are mechanism: they belong to a version and will move.

Keep them visibly separate in different files, clearly labelled, so that after an update you know which half to check. That single organisational choice is most of what makes an update cost an afternoon.

Then stop

Week thirteen has one instruction that is harder than it sounds: stop adding. Let the setup run. The purpose of the whole exercise was a workflow you can stop thinking about, and continuing to tune is how you end up back on the treadmill with better documentation.

Put a date three months out for the next review. Until then, changes require a logged problem, not an announcement.

FINDING WHAT AN UPDATE WOULD BREAK

The Fragility List

List every part of my workflow that depends on something outside my control: file locations, selection behaviour, model-specific phrasing, output formats. Rank by how much would break.

WHY IT WORKS

You cannot harden what you have not enumerated, and almost nobody has this written down.

USE WHEN Week nine **PAIRS WITH** The Separation Pass

Workflows with no external dependencies at all.

CONTENT ON ONE SIDE, MECHANISM ON THE OTHER

The Separation Pass

Sort my setup into content (decisions that do not expire) and mechanism (whatever belongs to a version). Propose a file layout that keeps them visibly apart.

WHY IT WORKS

After an update you need to know which half to check, and the layout is what tells you without thinking.

USE WHEN Week ten **PAIRS WITH** The Fragility List

Very small setups of one or two files.

WEEK THIRTEEN

The Stop Rule

Write my stopping rule: what must be true for me to change this setup before the next quarterly review? Make it specific enough that a release announcement does not qualify.

WHY IT WORKS

Naming what does not qualify is the operative part, because that is where every rebuild starts.

USE WHEN Week thirteen **PAIRS WITH** The Separation Pass

Setups still failing on measured problems.

Try This: Write the Fragility List

Fifteen minutes that decide how the next update goes.

1. List everything depending on a file path, a version behaviour or a model's phrasing.
2. Rank by how much would break if it changed.
3. Fix the top two only.
4. Reorganise so content and mechanism live in clearly separate files.

KEY TAKEAWAYS

- Enumerate your fragility surface, then fix only the top two.
- Content does not expire. Mechanism belongs to a version. Keep them visibly apart.
- Week thirteen's instruction is stop. A release announcement is not a logged problem.

CHAPTER 5

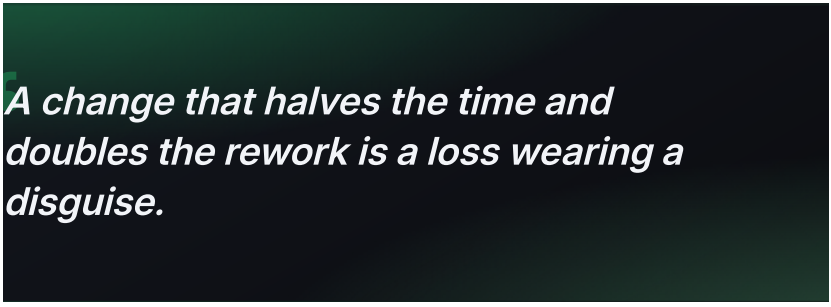
Reading The Log

> Thirteen weeks of data, read in one sitting. This hour is the most valuable in the plan, and it is the one people skip because the answer feels like it is already known.

Time saved and rework caused

Two numbers, not one. A change that halves the time and doubles the rework is a loss wearing a disguise, and it is the most common way a setup feels fast while producing worse work.

Read them together, per change, comparing the two weeks either side. Anything that improved one at the expense of the other needs a decision rather than a celebration.



Look for the things you stopped doing

The log records what you did. The interesting entries are the ones that disappear: a standard that stopped being followed, a review step that quietly lapsed, a tool that has not appeared in three weeks.

Those are decisions you made without noticing, and they are usually right. You dropped the thing because it was not earning its place. Confirm that, then remove it deliberately rather than leaving it installed and ignored.

Be fair in the review

Thirteen weeks of consistency deserves an honest read rather than a critical one. If the numbers show a modest improvement and a workflow you no longer think about, that is the goal achieved. It is not supposed to feel dramatic.

The dramatic feeling belongs to rebuilds, which is exactly why they are so appealing and so unproductive.

THE FULL REVIEW

The Thirteen-Week Read

Here is the whole log. For each dated change: time before and after, rework before and after. Then name the one change that most clearly paid for itself.

WHY IT WORKS

Per-change before-and-after is the payoff for the whole one-at-a-time discipline, and it takes an hour once.

USE WHEN Week thirteen **PAIRS WITH** The Disappearance Check

Logs with overlapping changes that cannot be separated.

DECISIONS YOU MADE WITHOUT NOTICING

The Disappearance Check

What appears in my early logs and not in the recent ones? For each, tell me whether it was dropped because it was not helping or because I simply forgot.

WHY IT WORKS

Silent drops are real decisions, and turning them into explicit ones prevents a set full of ignored files.

USE WHEN During the thirteen-week read **PAIRS WITH** The Thirteen-Week Read

Short logs with no drift yet.

JUDGING THE WHOLE QUARTER

The Honest Verdict

Given these numbers, was this quarter's work worth it? Argue both sides and then commit to one, using only what the log shows.

WHY IT WORKS

Arguing both sides before committing keeps the verdict from being decided by the most recent session.

USE WHEN End of the read **PAIRS WITH** The Thirteen-Week Read

Reviews you have already decided the outcome of.

Try This: One Sitting

The read only works if it is done all at once.

1. Block an hour and read the whole log end to end.
2. For each dated change, write the before and after numbers.
3. List everything that disappeared from the log along the way.
4. Write a one-paragraph verdict before you look at any announcements.

KEY TAKEAWAYS

- Time saved and rework caused are two numbers, and one without the other misleads.
- What disappeared from the log is a decision you already made. Make it explicit.
- A modest improvement and a workflow you stopped thinking about is the goal, not a disappointment.

CHAPTER 6

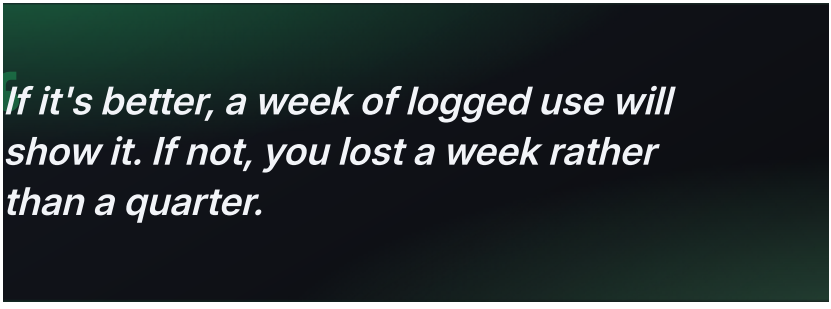
When The Landscape Shifts

> Something significant will be released. This chapter is about responding to that without restarting, which is the difference between a workflow and a hobby.

Test it, do not adopt it

When something new appears, run your three bench tasks. That is fifteen minutes and it produces a comparison against your actual work rather than against a demo. If it wins clearly on a job type you do a lot of, it earns a week-two slot in the next cycle.

If it wins narrowly or on a type you barely use, it goes in the queue. Nothing gets adopted on the strength of an announcement.



Adopting is one change, like any other

A new tool enters the same way everything else did: one change, one week, logged, with a stated expectation. The fact that it is exciting is not a reason to skip the process. It is the reason the process exists, since excitement is what makes attribution sloppy.

If it is better, a week of logged use will show it clearly. If it is not, you have lost a week rather than a quarter.

Re-run the plan when things really change

Occasionally something shifts enough that the whole shape is wrong, such as a new capability that changes what the four job types even mean. That is a genuine reason to re-run the full thirteen weeks, and it happens rarely.

The second time is much easier. The log format exists, the bench tasks exist, the standards exist, and you are adjusting a system rather than assembling one. That is what all of this was for.

SOMETHING NEW APPEARED

The Announcement Filter

Before I get interested: does this address a problem in my log? Quote the log entry. If there is no entry, tell me to put it in the queue.

WHY IT WORKS

Requiring a logged problem is the single rule that keeps a workflow from being driven by other people's release schedules.

USE WHEN Every release that tempts you

PAIRS WITH The Fifteen-Minute Bench

Genuine blockers you are already logging weekly.

TESTING A CANDIDATE QUICKLY

The Fifteen-Minute Bench

Run my three bench tasks against this new option. Report only correctness and rework versus my current tool, and whether the ranking changed.

WHY IT WORKS

Restricting the output to the ranking prevents interesting-but-irrelevant differences from triggering a rebuild.

USE WHEN Any credible new candidate

PAIRS WITH The Announcement Filter

Tools you have no access to yet.

WHEN THE SHAPE ITSELF CHANGED

The Re-Run Decision

Has something changed enough that my four job types or my standards no longer describe the work? Argue for re-running the full plan, then against it.

WHY IT WORKS

Forcing both arguments prevents both the reflexive rebuild and the stubborn refusal to notice a real shift.

USE WHEN Rarely, and only on a genuine capability shift

PAIRS WITH The Announcement Filter

Ordinary version updates.

Try This: The Filter Question

One question, asked before you get interested.

1. Next time something is released, open your log first.
2. Find the entry describing a problem it would solve.
3. If there is no entry, put it in the queue and close the tab.
4. If there is one, run the fifteen-minute bench before anything else.

KEY TAKEAWAYS

- Test with your three bench tasks; never adopt on an announcement.
- A new tool enters as one change, one week, logged. Excitement is why the process exists.
- Re-running the full plan is rare, and much easier the second time.

PUT IT INTO PRACTICE

Your Action Plan

Thirteen weeks on one page. Measure, build, harden, then stop.

- [ ] List this year's changes and mark which followed a measured problem versus a release.
- [ ] Name three things your current setup does well before changing anything.
- [ ] Start the four-column log: type, tool, minutes, redone.
- [ ] Log for four full weeks and change nothing, parking every urge in a queue.
- [ ] Day 29: read where the time goes and where the rework concentrates.
- [ ] Week five: install the two standards your log most justifies.
- [ ] Week six: add the review handoff. Week seven: one tool change, from bench evidence.
- [ ] Week eight: change nothing and see which changes are still being followed.
- [ ] Week nine: write the fragility list and fix only the top two items.
- [ ] Week ten: separate content from mechanism into clearly different files.

- [ ] Week thirteen: read the whole log in one sitting, then write your stop rule.
- [ ] After that, require a logged problem before any change. Announcements do not qualify.

FINAL WORDS

Where You Go From Here

Thirteen weeks from now you will have something most people never get, because most people rebuild before the evidence arrives: a workflow you can defend with numbers. Whatever those numbers say will be worth having. If they show a setup that saves real time and produces less rework, keep it and stop touching it. The goal was always a workflow you no longer think about, not one you enjoy tuning. If they show that a change you were sure about did nothing, that is equally valuable and much cheaper to learn from a log than from another quarter of the same. Be fair in the week-thirteen review. A modest, durable improvement is what success actually looks like here; the dramatic feeling belongs to rebuilds, which is why they are so appealing and so expensive. And when the next big release lands, probably before you finish this plan, open your log before you open the announcement. If there is no entry describing a problem it solves, it goes in the queue. That single habit is worth more than every tool decision in this book.

Breathe. Begin. Return.

between versions and runs, so nothing here guarantees a specific result. Payments are processed securely by Digistore24.