
A PRACTICAL FIRST GUIDE

Beyond "It Works"

A practical workflow for building
features with AI coding agents

A small feature. A clear boundary.
Evidence you can review.

George Hanu

George Hanu | Coding with AI

[@georgehanu](#)

FIRST EDITION / 21 SEPTEMBER 2026

The message says done. Is the feature done?

The agent adds an Export button, the application builds, and the completion message sounds certain. Then you filter the table and download the file. Half the matching issues are missing. The code runs. The request is still unfinished.

This guide is for developers and solo builders who already use a coding agent, understand a repository and a diff, and can run an application. You will take one small feature from a vague request to a reviewable change: a brief, relevant context, a bounded plan, implementation, checks, and a useful handoff.

The agent can do the implementation and run the checks. Your work is to define the outcome, settle consequential choices, assess the evidence, and decide what is acceptable. You do not need to approve every file read.

Use the quick start on the next page during your next feature. Read the chapters when a step needs more care. The eight prompts are tools to reach an outcome, not a ceremony to complete.

Find your next step

- [Quick start: choose the right amount of process](#) · 3
- [1. Define what done means](#) · 4
- [2. Give the agent useful context](#) · 7
- [3. Bound the plan, then build](#) · 10
- [4. Recover without random rewrites](#) · 13
- [5. Review evidence, then the change](#) · 16
- [6. Leave a usable handoff](#) · 20
- [Toolkit and exercise](#) · 22
- [References and edition notes](#) · 25

Quick start: match the process to the risk

Lightweight. A clear, low-risk edit needs a small request and a proportionate check. For example: change the table's button label to "Export filtered issues," preserve its behavior, and inspect the rendered label and diff. A separate plan, five documents, and all eight prompts would add little.

Ordinary feature. CSV export affects row selection, data formatting, and browser behavior. Write a short brief, let the agent inspect the relevant code, agree on a bounded approach, and request evidence against the brief. The worked example follows this path.

Higher risk. Payments, authentication, sensitive exports, and production migrations need controls beyond this guide: appropriate expertise, permission and privacy review, realistic testing, rollout decisions, and recovery procedures. A small diff can still carry substantial risk. Classify the consequences, not the line count.

One feature, six moves

1. Describe the outcome with examples that could pass or fail.
2. Find the code, conventions, versions, and real check commands.
3. Bound the change and let the agent implement it.
4. Investigate any mismatch using observations and focused checks.
5. Compare evidence with the acceptance criteria and inspect the diff.
6. Record what changed, what was checked, and what remains.

The guide's CSV application is fictional, local, and synthetic. Its implementation checks were run during guide production by the AI assistant, not personally by George. Tool documentation supports the adaptation notes. The workflow and examples are editorial recommendations, not a measured prompt success rate.

1. Define what done means

Start with the missing decision

"Add CSV export to this table."

That request names an interface action. It leaves the meaning of the file open. Should the download contain the two visible rows, all filtered rows, or every issue? A technically tidy implementation can choose the wrong answer.

Weak brief: "Add a CSV button. Make it work with filters and handle edge cases."

Improved brief: "Add an Export filtered issues button. Export all issues matching the current status and title/ID search, including matches on other pages. Keep ID ascending order. Include ID, Title, Status, and Assignee, in that order. When no issues match, disable export and explain why. Keep the existing table, filters, and pagination. Use local synthetic data; no backend, new production dependency, or unrelated redesign."

The improved brief names a set of rows, a stable order, a schema, an empty state, and boundaries. "Handle edge cases" has become decisions you can evaluate. Clear scope and acceptance criteria also appear in GitHub's guidance for assigning agent tasks. [\[5\]](#)

Decide product behavior yourself or agree on a recommendation from the agent. Export scope, included fields, and what users see when nothing matches are product choices. Helper names and where to place a small function are usually implementation choices the agent can make from repository patterns.

Make the examples disagree with a bad implementation

An acceptance criterion is an observable condition for accepting the change. Give it a stable label so the report can refer to it without rewriting the brief.

A1 • Selection and order. Six synthetic issues exist. With status Open and an empty search, four match: ISS-101, ISS-103, ISS-104, ISS-106. Page size is two. Exporting from either page produces those four IDs, once each, in that order. Closed issues stay out. Searching for ISS-104 while Open is selected exports only ISS-104.

A2 • Shape. The header is ID, Title, Status, Assignee. Every record has four fields. An empty assignee becomes an empty field. No internal or extra fields appear. The filename is issues.csv.

A3 • Text. Commas, embedded double quotes, line breaks, and non-ASCII names survive the chosen CSV representation. Fixtures include a comma in one title, quotes in another, a two-line title, and an assignee named Chloé. Check parsed values, not just the appearance of a text editor.

A4 • Formula policy. Formula-like text receives the documented apostrophe prefix described on page 12. This intentionally changes exported text. The test verifies that transformation; it does not establish safety in every spreadsheet.

A5 • Empty and existing behavior. A search with zero results shows an explanation, disables export, and produces no download. Filtering and moving between pages still work. The button can be reached and activated with a keyboard when enabled.

These examples expose ambiguity cheaply. A file with two correct-looking rows fails A1. A screenshot of a button cannot satisfy A3. Before implementation, ask whether each important criterion has at least one observation that could reject the work.

Prompt 1 • Turn the request into a brief

Use when: the request leaves meaningful behavior undecided. **Supply:** the feature, user situation, constraints, and any known examples. **Expect:** a concise brief with decisions and checkable criteria. **Watch for:** the agent quietly treating its assumptions as your decisions.

```
Help turn [FEATURE] into a small, testable task brief.  
Context: [USER_SITUATION]. Constraints: [CONSTRAINTS].  
Known examples: [EXAMPLES].  
Identify missing product decisions that affect behavior,  
risk, or scope. Ask only consequential questions and label  
your proposed defaults. Do not implement yet.  
Draft the agreed outcome, acceptance criteria, exclusions,  
and unresolved decisions. Leave routine implementation  
choices to existing repository patterns.
```

For the CSV example, the brief also records the encoding choice: UTF-8 without a byte-order mark, comma separators, and CRLF record endings. These are explicit demonstration choices. If your intended recipient requires a different format, that changes the brief before it changes the code.

Keep exclusions visible. This task does not add bulk selection, a background export job, new sorting controls, server pagination, or a generic reporting system. A useful exclusion prevents an attractive adjacent feature from becoming an accidental requirement.

Evaluate the brief: hand it to someone who has not seen the conversation. Could they tell whether a download containing only the visible page is wrong? Could they identify what must remain unchanged? If not, revise the brief before requesting code.

2. Give the agent useful context

Ask for a map, not a repository dump

The agent needs to find where the table's rows come from and where pagination happens. Sending every file adds noise without answering either question. Start with the screen, a relevant path if you know one, and the task brief. Let the agent trace the behavior.

Ask it to inspect the table entry point, filtering and sorting logic, shared types, download utilities, tests, package scripts, and lockfile. A lockfile records resolved dependency versions. A package manifest's version range alone does not establish what is installed.

In the demo, `src/table.ts` filters, orders, and pages the issues; `src/app.ts` connects the controls. Export uses the filtered, ordered set before the display slice. The new `src/csv.ts` handles formatting. `package.json` defines `npm run typecheck`, `npm run build`, and `npm test`. That small map explains where the change belongs and how to check it.

Have the agent report the actual commands it found and where it found them. "Run the tests" is incomplete if the project has separate unit and browser commands. A command can exist and still fail because dependencies or a local service are missing. Report discovery and execution separately.

Look for an existing solution before adding a dependency. A production application may already have a reviewed CSV library or export service. Reuse it when it fits. The small standalone demonstration uses a focused serializer so its behavior is inspectable.

Evaluate the context: can the agent point to the source of the filtered rows, the display slice, and the applicable checks? If its map lists files without explaining these relationships, ask for the missing trace.

Prompt 2 · Inspect the relevant context

Use when: the repository or affected area is unfamiliar. **Supply:** the brief and known entry points. **Expect:** a small evidence-backed map and executable check commands. **Watch for:** invented paths, assumed scripts, or exploration that drifts into unrelated cleanup.

```
Inspect the repository for [TASK_BRIEF]. Start at [KNOWN_ENTRY_POINTS], then follow relevant references. Do not change files yet. Identify the data flow, reusable patterns, applicable guidance, installed versions, and actual run/build/test commands with their source files. Separate facts from assumptions and note existing local edits. Return the smallest useful context map, likely change locations, and unanswered questions.
```

Keep lasting guidance separate

Durable repository guidance explains how work is normally done: real commands, architectural boundaries, conventions, and recurring traps. Temporary task context explains this feature: export scope, fixtures, current symptoms, and accepted exclusions. Avoid copying the entire brief into permanent instructions.

A useful lasting rule might be: "Table actions that operate on all matches must use the filtered set before pagination; check with more matches than one page." It names a concrete trap and a way to detect it. "Always write perfect code" does neither.

Before editing, preserve unrelated local work. Record which changes already exist. Use a separate folder or checkout when isolation helps. Do not authorize broad cleanup just because the working tree is untidy.

Adapt the guidance to your agent

These are documented behaviors checked on 21 September 2026, not a claim that the two tools load instructions identically. Check your installed version and configuration when behavior differs.

Codex. The documented discovery chain starts with global guidance, then follows the project root to the current working directory. At each project directory it prefers `AGENTS.override.md`, then `AGENTS.md`, then configured fallback names, using at most one file. Later, closer guidance takes precedence. Discovery has a configurable size cap. Use a concise root `AGENTS.md` for shared conventions and inspect the applicable chain. [2]

Claude Code. Use a project `CLAUDE.md` or `.claude/CLAUDE.md`. Ancestor guidance loads at launch; subdirectory guidance loads as relevant files are read. Instructions accumulate, so remove conflicts instead of assuming Codex-style override behavior. `/context` lists `CLAUDE.md` memory files, but not directly loaded `AGENTS.md`. Current docs describe conditional direct `AGENTS.md` support from v2.1.277, with configuration and session exceptions. `@AGENTS.md` in `CLAUDE.md` imports shared content. [4]

Instructions guide model behavior. They are not an enforced security boundary. Use the tool's actual permission and isolation controls for access limits. A sentence saying "do not read secrets" does not make secrets inaccessible.

Tool documentation recommends useful context and verification, but cannot choose your acceptance criteria for you. OpenAI's guidance emphasizes the goal, context, constraints, and completion conditions. [1] Your brief makes those specific to the current feature.

3. Bound the plan, then build

A plan earns its space by reducing uncertainty

For the CSV feature, a short plan helps because row selection, serialization, and downloading can each be correct alone while the assembled feature is wrong. The plan should explain how they connect and how the criteria will be checked.

An adequate plan has four moves: reuse the filtered, ordered issue set; serialize the agreed columns and text policy; connect the button and empty state; run focused data checks and inspect browser behavior. Identify likely files, but allow a justified helper or test file within scope.

Do not require a plan for every typo. Claude Code's official guidance also distinguishes useful planning for uncertain work from unnecessary overhead on small, clear changes. [3] Here, the plan exists because it can catch "export the visible rows" before that choice spreads through the code.

Agree on approval boundaries once. In this example, local edits and checks within the demo are authorized. Changing the export meaning, adding a production dependency, touching an external service, or expanding to unrelated components requires a fresh decision. The agent can choose helper names without stopping.

Prompt 3 · Produce a bounded plan

Use after the brief and context are clear. Supply both plus approval boundaries. Expect a short sequence with checks. Watch for a plan that repeats the feature name without describing data flow.

```
Plan [TASK_BRIEF] using [REPOSITORY_CONTEXT].
Name the smallest coherent changes, likely files, and
checks for [ACCEPTANCE_CRITERIA]. Respect [EXCLUSIONS].
Work within [APPROVAL_BOUNDARIES]; identify anything
outside them. Surface unresolved decisions before coding.
Keep the plan proportionate and explain the data flow.
```

Prompt 4 · Implement the agreed change

Use when: the outcome and boundaries are settled. **Supply:** the brief, plan, criteria, and available commands. **Expect:** a focused change with evidence. **Watch for:** helpful-looking refactors that make the feature harder to review.

```
Implement [AGREED_PLAN] for [TASK_BRIEF]. Reuse existing
patterns and preserve unrelated local edits. You may make
routine implementation decisions within [BOUNDARIES].
If a discovery changes product behavior or expands scope,
explain it before proceeding with that expansion.
Run [DISCOVERED_CHECK_COMMANDS] and focused checks for
[ACCEPTANCE_CRITERIA]. Report actual results, failures,
and anything not run. Leave a reviewable diff and a concise
completion report; do not publish or deploy.
```

The essential connection in this example is short enough to review directly:

```
const matching = selectIssues(issues, filters);
const visible = matching.slice(start, start + pageSize);
// Display visible; serialize matching for export.
```

This is an explanatory excerpt, not a replacement for the runnable companion source. The important decision is which collection crosses the export boundary.

Keep the new serializer separate from browser actions so text behavior can be exercised without clicking a button. Then test the button too: a correct helper does not prove the application calls it with the right rows.

The browser download uses a Blob and an object URL; the implementation releases that URL after initiating the download. These APIs manage in-memory data and its URL lifetime. [8][9][10] The download filename is a browser hint; actual handling can vary with browser and user settings. [11]

CSV has two different problems

Preserving structure. The example quotes every field and doubles embedded double quotes. Records end with CRLF; quoted fields retain internal line breaks. RFC 4180 describes these conventions and the `text/csv` media type. It is an informational RFC, not a guarantee that every importer behaves identically. [6]

The chosen encoding is UTF-8 without a byte-order mark. Tests cover non-ASCII text in bytes and parsed values. That does not verify how a particular spreadsheet guesses encoding when someone double-clicks the file. Test the intended import path before promising compatibility.

Handling formula-like text. Quoting a CSV field does not by itself prevent a spreadsheet from treating its value as a formula. OWASP documents formula injection and warns that no CSV sanitization policy works for every application and downstream consumer; saving and reopening can also undermine mitigations. [7]

The demonstration prefixes an apostrophe when the first non-whitespace/control character is `=`, `+`, `-`, `@`, or a fullwidth counterpart, or when the value begins with a tab, carriage return, or line feed. Here whitespace means JavaScript `\s` and controls mean Unicode Cc; format characters such as U+200B are outside this policy. Original text is retained after the prefix, including benign values such as `-1`.

For the harmless fixture `=1+1`, the parsed exported value is `'=1+1`. The prefix changes the data. It may remain visible to a programmatic consumer. Do not market this serializer as universally safe or lossless.

If real users need spreadsheet imports, agree on the consumer, locale, encoding, and acceptable transformations. Test those exact combinations or choose a format with explicit cell types. This local example does not establish production security.

4. Recover without random rewrites

Use the mismatch as the starting point

The following wrong turn is constructed for teaching. It is not a transcript of a named model making a mistake.

Illustrative completion: "CSV export is implemented. The build passes."

Observed symptom: select Open with an empty search. The table reports four matches, but the download contains only ISS-101 and ISS-103. Move to the second page and the file changes to ISS-104 and ISS-106.

Missing information: the initial request did not settle whether "this table" meant the visible page or all matches. The improved brief resolves that choice, but the implementation must still use it.

Focused correction: "A1 requires all four filtered issues, regardless of page. Reproduce the missing-row case. Trace the collection passed to the serializer and distinguish filtering from pagination. Correct the selection boundary, keep the existing table behavior, and show the same check rejecting the faulty version and passing after the fix."

There are competing hypotheses: the export receives the page slice; filtering drops two rows; or the serializer loses records. Check the selected IDs before formatting. If all four reach the serializer, inspect serialization. If only two arrive, follow the earlier data flow. That observation eliminates a branch of the investigation.

In the companion demo, the faulty variant is intentionally introduced and isolated from the final application. Its expected failure is preserved with the corrected run. This shows that the check can detect this particular mistake. It does not measure how often any agent would make it.

Prompt 5 · Diagnose before rewriting

Use when: a criterion fails or a reported fix does not match observation.

Supply: reproduction steps, expected and actual behavior, the actual error, and recent changes. **Expect:** a discriminating check and a targeted correction.

Watch for: changing several unrelated things until the symptom disappears.

```
Diagnose [FAILURE] against [ACCEPTANCE_CRITERION].  
Reproduce with [STEPS_AND_INPUTS]. Expected: [EXPECTED].  
Actual: [ACTUAL]. Evidence: [ERROR_OR_OUTPUT].  
Inspect relevant code and propose competing explanations.  
Run the smallest check that distinguishes them before  
editing. Fix the supported cause within [SCOPE].  
Preserve useful tests and local changes. Show the failing  
evidence, the correction, and the same check afterward.  
Do not weaken the criterion to make the result pass.
```

An error message is an observation; "the library is broken" is a hypothesis. Keep them separate in your request. Include the relevant error lines and command rather than paraphrasing away the detail that explains the failure.

If a command cannot run, investigate that as an environment problem first. A missing compiler does not establish that the feature is defective. Equally, installing a compiler does not prove the feature works. Report each step with its own outcome.

After a correction, rerun the focused check and nearby checks affected by the change. A fix to selection could alter filtering or ordering. A change to escaping could break quotes or line breaks. Choose regression checks from those relationships rather than rerunning everything indefinitely.

Continue, reduce, checkpoint, or hand off

Continue the session when the brief is still clear and the next check can resolve a specific uncertainty. The agent already knows the relevant files; use that context.

Reduce the task when too many changes obscure the cause. For this feature, test selection separately from serialization, then reconnect them. Reduction should produce a smaller question, not quietly remove a requirement.

Return to a safe checkpoint when an experiment has made the diff difficult to understand. First inspect and preserve useful edits, including work that predates the task. Save a patch, a copy, or a local commit as appropriate. Reverse only identified task changes after review. A destructive reset can discard uncommitted work, so it is a poor default recovery instruction.

Prepare a fresh-session handoff when the conversation contains conflicting decisions or the next person would struggle to find the current state. Include the brief, files, actual checks, rejected hypotheses, remaining failure, and next discriminating action. The number of messages alone is not a reliable trigger.

For example: "All-filtered selection is fixed. Quoted line breaks pass the parser check. Browser download behavior has not been checked in this session. Next, start the local app and download Open results from page two. Expect all four IDs."

Evaluate recovery: can you identify what observation changed your explanation, which edit addressed it, and which check now passes? If the answer is only "we rewrote it and it seems fine," the causal story is still missing.

5. Review evidence, then the change

Different checks answer different questions

A successful build shows that the chosen compilation or bundling step completed. A type check can reject incompatible types. Neither tells you whether export includes all filtered rows. Valid code can implement the wrong product decision.

Automated tests can exercise selection, field ordering, escaping, and the empty-result function. Their strength depends on the inputs and expected outcomes. A test that copies the implementation's own output into its expectation may only preserve the current mistake.

Browser checks exercise the connected interface: filters, pagination, keyboard activation, the download action, and the empty state. They can reveal that a correct helper is wired incorrectly. One browser run does not establish support across browsers, assistive technologies, or spreadsheet applications.

Diff review asks a different question: is the change coherent and within scope? Read the route from the click handler to the filtered set and serializer. Look for extra fields, changed filter behavior, accidental dependency changes, and unrelated edits. A screenshot cannot answer those questions.

Build an evidence map from the brief. A1 needs the exact exported IDs from a multi-page filtered set. A2 needs parsed headers and fields. A3 needs difficult text fixtures. A4 needs transformation checks plus an explicit limitation. A5 needs interaction evidence.

The person or agent running a check should report the input, expected result, observed result, command or procedure, and status. Ask for enough detail to reproduce the important claim, not a transcript of every tool call. "All tests passed" is useful only when you know what those tests cover.

Prompt 6 · Check against the criteria

Use when: implementation is ready for evaluation. **Supply:** the agreed criteria, example data, and available environment. **Expect:** a criterion-by-criterion evidence report. **Watch for:** only testing the happy path or relabeling an unrun check as a pass.

```
Check [CHANGE] against [ACCEPTANCE_CRITERIA] using
[Test_INPUTS] in [AVAILABLE_ENVIRONMENT]. Derive expected
results from the criteria before inspecting actual output.
Run the relevant discovered commands and behavior checks.
For each criterion, report input, expected result, actual
result, evidence location, and status: passed, failed,
blocked, or not run. Explain coverage limits. Report any
new issue without silently changing the agreed behavior.
```

Give the check an independent expectation

For A1, write down the four expected IDs from the synthetic dataset. Do not ask the export helper for the expected list. Compare the download with that list, including order and duplicates. Use a second page because a one-page fixture cannot distinguish the correct implementation from the visible-page mistake.

For A3, parse generated bytes with a CSV parser rather than splitting on newline or comma. Embedded line breaks belong inside fields. The companion checks include an independent Python CSV reader as well as focused TypeScript/JavaScript checks. A parser round trip helps verify structure, but it does not simulate spreadsheet formula interpretation.

A useful negative control is an intentionally bad variant that the same expectation rejects. Keep its role obvious. A red teaching run is expected evidence, not a hidden failure of the delivered application. Preserve both results so the reader can see that the test was capable of saying no.

Read the demonstration evidence honestly

The companion preserves the synthetic starting table, task artifacts, change, and logs. Recorded versions: Node 22.14.0, TypeScript 5.9.3, and Python 3.12.14. From the demo folder, the recorded runs include:

```
npm run typecheck      passed, exit 0
npm run build          passed, exit 0
npm test               9 passed, 0 failed
npm run teaching:fail  expected failure, exit 1
npm run teaching:pass  passed, exit 0
```

The deliberately faulty run produced ISS-101 and ISS-103; the unchanged expectation also required ISS-104 and ISS-106. The corrected run passed. This distinguishes those implementations on this data.

The serializer checks cover commas, quotes, line breaks, empty fields, non-ASCII text, and the chosen formula-prefix policy. The independent parser check reads generated CSV. The local browser review covers filter and page interactions, download initiation, and the empty state. Detailed results and any limits are in `verification-report.md` and the demo evidence folder.

Use evidence according to its reach. A parser accepting a CSV file says something about that parser and those bytes. It does not certify Excel, LibreOffice, every locale, huge datasets, or hostile production inputs. The example has six rows and no database, network pagination, or access-control layer.

The prompts were not evaluated across multiple agents or a repeated task set. A working implementation cannot establish a prompt success rate, a time saving, or demand for this guide. No such measurement is claimed.

Prompt 7 • Review concrete issues in the diff

Use when: you have a bounded change and check results. **Supply:** the diff, criteria, surrounding context, and evidence. **Expect:** actionable findings tied to behavior. **Watch for:** speculative style complaints or an approval that cites no inspected path.

```
Review [DIFF] with [ACCEPTANCE_CRITERIA] and [CHECK_RESULTS].  
Inspect surrounding code where needed. Look for concrete  
correctness, regression, data-exposure, and scope issues.  
For each finding, give the location, triggering input or  
action, likely effect, and a focused way to confirm it.  
Separate verified issues from hypotheses. Do not rewrite  
the change or invent findings. If none are found, state  
what you inspected and what remains unverified.
```

Start your own reading at the changed behavior. For CSV export, follow the button to the current filter state, then to the selected rows and field mapping. Confirm that pagination is not applied to export and no internal properties sneak into the file through generic object serialization.

Then read the tests. Do their expectations come from the brief? Is a meaningful failing case now missing? Did the change simply rename a test or bypass a check? Review additions and deletions, including configuration and lockfiles.

A second agent can offer another reading, especially when it receives the criteria and evidence without a persuasive implementation narrative. Its approval is still an opinion, not proof. Ask for concrete findings and reproduce consequential ones.

Accept the feature when the evidence is adequate for its scope and risk. If a required browser or target-import check is unavailable, leave that limitation visible and assign the next action. Do not turn "no issue found" into "no issue possible."

6. Leave a usable handoff

Make the next decision easy

A completion report should let someone review or resume the work without reconstructing the conversation. Record the behavior change, changed files, criteria covered, commands and outputs, failures, blocked or unrun checks, remaining risks, and the next action.

Use three distinct statements. **Implemented** means the code is present. **Checked** means named observations support named criteria. **Ready for release** means the relevant release conditions have been met for the intended environment. The first two do not automatically imply the third.

The feature adds `src/csv.ts`, updates `src/app.ts` and `index.html`, and adds focused tests. Here is a compact summary; the companion report contains the full file list and logs:

Implemented CSV export of all filtered issues before pagination. Header order, text handling, and empty-state behavior follow A1-A5. Build, automated checks, parser checks, and the documented local browser review support the covered cases. The deliberately faulty visible-page variant was rejected; the corrected variant passed the same selection expectation. Spreadsheet application imports, broader browser coverage, and production-scale behavior are not verified. Next: review the evidence and choose target-consumer checks before adapting this code for real data.

Do not bury an important failed check below a celebratory headline. If a failure is unrelated, describe the evidence for that conclusion and its impact. If a check was not run, say so instead of guessing what would happen.

Prompt 8 · Prepare a fresh-session handoff

Use when: someone else or a fresh session must continue the work. **Supply:** the brief, current state, decisions, and evidence locations. **Expect:** a self-contained continuation note. **Watch for:** retelling the conversation while omitting the failing command or next step.

Prepare a handoff for [TASK] from [CURRENT_STATE].
Include the agreed brief, criteria, boundaries, relevant files, existing local changes, decisions, and actual check commands with results and evidence paths. Preserve useful failed attempts and rejected hypotheses briefly.
Separate completed work from failed, blocked, not-run, and not-verified items. State the next specific action and unresolved decisions. Do not imply release readiness or invent evidence. Make this usable without chat history.

Use `completion-report.md` when the task is ready for a decision. Use `session-handoff.md` when work must continue. They overlap deliberately, but you usually do not need two lengthy documents for a small feature.

Improve one recurring weakness

If exports repeatedly use the current page by mistake, add a focused regression test with more matches than one page. A concise guidance note can point to that test and the selection pattern. Do not paste the entire debugging conversation into permanent instructions.

Evaluate the handoff: could a colleague run the next check, tell which changes are yours, and distinguish a demonstrated fact from a recommendation? If yes, the note is doing its job. If not, add the missing evidence or decision, not another paragraph of reassurance.

Toolkit · One-page workflow checklist

Use the relevant items. A lightweight change may fit in one request and one review.

Before implementation

- Set the outcome, risk level, and product decisions.
- Write acceptance examples, including a rejecting case.
- Record constraints, exclusions, and approval boundaries.
- Find the data flow, patterns, guidance, and real commands.
- Identify and preserve existing local edits.

While the agent works

- Plan when uncertainty warrants it; allow routine decisions.
- Keep changes connected to the requested behavior.
- Diagnose from observations; preserve evidence and tests.

Before accepting the change

- Compare outputs with criteria, including boundary cases.
- Distinguish build, test, browser, and diff evidence.
- Inspect data fields, dependencies, and unrelated edits.
- Record failures, blockers, unrun checks, and uncertainties.
- Separate implemented, checked, and ready for release.
- Leave a report or handoff with a specific next action.

For the CSV example, the decisive boundary case is page two of a four-match result set. If your feature has a different boundary, choose an example that makes two plausible implementations produce different outcomes.

Find the prompt or template you need

Eight prompts, chosen by situation

1. **Clarify the feature** · p. 6 · Resolve meaningful ambiguity.
2. **Inspect context** · p. 8 · Find the relevant code and checks.
3. **Bound the plan** · p. 10 · Reduce implementation uncertainty.
4. **Implement** · p. 11 · Build within agreed boundaries.
5. **Diagnose** · p. 14 · Explain a mismatch before rewriting.
6. **Check criteria** · p. 17 · Produce observable evidence.
7. **Review the diff** · p. 19 · Find concrete defects or drift.
8. **Hand off** · p. 21 · Preserve state for the next session.

The companion `prompts.md` contains the same copy-ready text and usage notes. Replace the descriptive placeholders. Exact wording cannot guarantee an outcome; clarity and evidence matter more than preserving a phrase.

Five reusable companion templates

task-brief.md captures the outcome, decisions, criteria, exclusions, and boundaries. Use it when the task needs to survive beyond a conversation.

repository-guidance-template.md holds durable conventions and verified command locations. Its separate Codex and Claude Code notes explain how to adapt it; the template filename itself has no special loading behavior.

acceptance-checklist.md maps each criterion to an input, expectation, procedure, evidence, and status. Fill expected results before accepting actual output.

completion-report.md records changed files, check outcomes, remaining risks, and the release decision still needed.

session-handoff.md preserves the current state, useful failed attempts, local edits, unresolved choices, and the next focused action.

Try it on one small feature

Choose a change you can explain without describing an entire product: add an empty state, persist a table filter, or export a small report. Keep the environment local or otherwise appropriate to the data and risk.

Write your first request exactly as you would normally send it. Then identify one decision it leaves open. For example, does a saved filter apply to one browser, one account, or everyone? Settle that decision and write two acceptance examples, including one where a plausible shortcut would fail.

Let the agent inspect the relevant context and implement within your boundaries. Ask it to run the available checks and report evidence. You can direct this work without watching every tool call.

Compare the result with the examples you wrote before implementation. Mark each passed, failed, blocked, or not run. Inspect the diff for changes you did not request. If something fails, use the symptom and a focused check to guide the correction.

Finish with a short report: what changed, what you observed, what remains uncertain, and whether another check is needed before release. Keep one reusable lesson, such as a missing fixture or a concise repository note. Avoid turning a single task into a new bureaucracy.

The useful result is a change you can explain and evaluate. The prompts help you get there; the acceptance criteria and evidence tell you whether you arrived.

Keep the conversation practical

Share which step helped or where the workflow failed with [@georgehanu](#).

George Hanu | Coding with AI

First edition · 21 September 2026

References • Agent workflows

Documentation accessed 21 September 2026. Numbered citations distinguish sourced tool behavior from the original teaching examples and editorial workflow. The companion sources.md records claims, exact URLs, dates, and caveats.

[1] OpenAI • Best practices. Goal, context, constraints, completion conditions, and durable guidance. The original Codex URL redirects to ChatGPT Learn.

[Read the official guide](#)

[2] OpenAI • Custom instructions with AGENTS.md. Instruction discovery, per-directory file selection, precedence, and size limits. The original Codex URL redirects to ChatGPT Learn.

[Read instruction discovery](#)

[3] Anthropic • Best practices for Claude Code. Exploration, proportional planning, specific context, and verification.

[Read Claude Code best practices](#)

[4] Anthropic • How Claude remembers your project. CLAUDE.md loading and imports; conditional AGENTS.md support; guidance versus enforcement. Direct AGENTS.md support is documented for v2.1.277 or later with exceptions. These CLI behaviors were documentation-checked, not executed in this demonstration.

[Read project memory documentation](#)

[5] GitHub • Best practices for using GitHub Copilot to work on tasks. Clear scope, acceptance criteria, and context for delegated tasks.

[Read GitHub's task guidance](#)

These sources inform the advice; the underlying practices are not claimed as inventions by George. No community anecdotes, private client stories, or tool leaderboards are used.

References • CSV and browser behavior

[6] **Y. Shafranovich • RFC 4180, October 2005.** Informational description of CSV records, quoting, headers, and media type. Used for format conventions, not spreadsheet-security claims.

[Read RFC 4180](#)

[7] **OWASP • CSV Injection.** Formula interpretation, mitigation tradeoffs, and application-dependent limitations. The demo's selective prefix policy is a local design choice, not an OWASP certification.

[Read CSV injection guidance](#)

[8] **MDN • Blob() constructor.** Creating a Blob from string data.

[Read Blob documentation](#)

[9] **MDN • URL.createObjectURL().** Creating an object URL for a Blob.

[Read object URL creation](#)

[10] **MDN • URL.revokeObjectURL().** Releasing an object URL.

[Read object URL cleanup](#)

[11] **MDN • The anchor element.** Download attribute and browser-dependent behavior.

[Read download behavior](#)

About this edition

This is a proposed first free guide for followers, prepared with AI research, writing, implementation, and production assistance. George's authorship credit is supplied for the proposed guide; personal testing or endorsement of every example is not implied. No demand, conversion, productivity gain, or prompt reliability has been measured. The evidence describes the local demonstration only.