

The FOCUS toolkit

One task. Clear boundaries. A record you can review.

C. Shane Rivers

Companion to The Focused Agent

Author-review edition | September 2026

Ten editable templates, a complete worked cycle, and a runnable application-code demonstration.

The example uses synthetic data and a deliberately seeded defect. Its before-and-after command results are actual recorded executions.

Copyright 2026 C. Shane Rivers. Public distribution and repository reuse terms remain for the author to approve.

The FOCUS toolkit

One task, one usable record

C. Shane Rivers | Companion to The Focused Agent | September 2026

You don't need to fill out ten forms before an agent can fix a small bug. Start with a task brief, keep the evidence with it, and use the other forms when the work gives you a reason.

This download includes editable Markdown files, a printable handbook, and a runnable application-code demonstration. It doesn't connect to an agent, install a framework, or grant permission to change your systems.

Start here

1. Choose a small task you already understand. Copy `templates/03-task-brief.md` into your project and give it a task name.
2. Write the result in terms somebody can check. Name the allowed area, what must stay unchanged, and the checks needed for review.
3. Ask for inspection before editing. Give the agent the relevant project guide, not every document you can find. Resolve business questions before they become code.
4. Set the permission boundary in your tools as well as in the brief. A sentence saying “don't deploy” is not a substitute for restricted credentials.
5. Run the task. Save the actual commands, outputs, and changed source version. Keep failures and checks that couldn't run.
6. Review the diff against the brief. Use generated, executed, observed, and approved to describe the result. Make the next decision explicit.

For a first pass, read `worked-example.md`. It uses the included application-code demonstration and follows the same task from request through review. The execution is recorded; the application and its initial defect were created for teaching. The example doesn't borrow credibility from a customer system.

Choose the form that does the job

Need	File
Give a new worker a project map	templates/01-project-guide.md
Keep recurring behavior rules	templates/02-agent-operating-rules.md
Assign one bounded task	templates/03-task-brief.md
Hand off a distinct role or result	templates/04-delegation-handoff.md
Compare the change with the request	templates/05-review-checklist.md
Learn from a mistake	templates/06-incident-record.md
Preserve an out-of-scope finding	templates/07-follow-up-ledger.md
Track effort without guessing	templates/08-usage-and-rework.md
Record what actually ran	templates/09-command-history.md
Give the reviewer an evidence index	templates/10-evidence-packet-index.md

The handbook prints these forms and the worked example. The Markdown versions are the ones to adapt in your editor. Keep the dated download intact as a reference and make working copies for tasks.

FOCUS on one page

Frame the work. State the requested behavior and acceptance criteria. “Fix the report” needs a date field, a period rule, and an example of the wrong result.

Orient the agent. Identify the relevant code, tests, conventions, and unknowns. An unresolved business rule is a finding, not permission to guess.

Constrain the action. Name what may be read, edited, executed, or sent. Define the point at which the agent must return for a decision.

Use the right worker. Give inspection, implementation, testing, and review distinct jobs. One worker can perform several passes; describe that honestly. Add delegation when it helps the task, not because the form has a handoff section.

Show the evidence. Link each important completion claim to a source version, command result, observation, or approval. Carry useful lessons into the next task's rules.

A completion report you can reuse

State what changed and where. Then answer these four questions:

- Generated: what exists now that didn't exist before?
- Executed: what actually ran, against which environment and source version?
- Observed: what result was inspected, and what does it establish?

- Approved: who or what authorized the next action, and how far does that approval go?

Name checks not run and why. If acceptance is pending, say “pending.” If time or usage wasn't measured, say “not measured.” Neither field improves when somebody guesses at it.

Keep the record safe

Use synthetic or permitted, sanitized data. Don't put secrets, personal data, private customer names, live credentials, or sensitive service addresses into the record you share. Preserve the full record privately only where you are authorized to retain it. Explain any redaction that affects interpretation.

Recorded, reconstructed, and illustrative describe where evidence came from. Sanitized describes what private detail was removed. Keep both distinctions visible.

Try the application example

Open `application-code-example/README.md`. It explains the input contract, the saved failure, the correction, the test command, and what the example doesn't test. No packages or external services are needed for the included Node tests. The recorded runtime was Node v24.19.0; another runtime should be checked, not assumed identical.

Before a public release

This is an author-review download, not a hosted campaign. Copyright remains with C. Shane Rivers. No public open-source license or public repository has been declared. Decide distribution and reuse terms before publishing a repository or a public download. Keep customer evidence out of that release unless its use has been cleared.

9. Evidence states

Use these labels in task records and final reports:

State	Meaning
Generated	The agent produced a proposal, code, or command
Executed	The proposal or command actually ran
Observed	Someone inspected the result or output
Approved	A human or explicitly authorized policy accepted it

The labels keep a report from collapsing several different claims into one word: complete.

The FOCUS cycle

7. Worked example

Recorded execution on an illustrative application. September 15, 2026.

This is the application-code demonstration from Chapter 15, carried through the toolkit. The assistant created the small report function, synthetic records, and deliberately faulty starting version for teaching. The command results below came from actual runs. Nothing here describes a customer incident.

The editable companion includes the files under `application-code-example/`. The completed record below has no invented approval or time saving. An explicit “not measured” is a completed field when the value isn't known.

Frame

Task: APP-DEMO-01

Request: Exclude the next period's first instant from a report summary.

Affected behavior:

`report.cjs` returns a count and IDs for completed records.

Contract:

- Use `completedAt`, not creation time.
- Start: 2026-08-01T00:00:00.000Z, included.
- End: 2026-09-01T00:00:00.000Z, excluded.
- Count only records with status "completed".
- Caller supplies validated ISO timestamps and a valid range.

Acceptance:

Ordinary records, exact start, exact end, outside records, canceled records, empty input, and input preservation have explicit tests.

Open business decision:

None within this teaching contract. UTC boundaries are fixture choices, not approval of a real organization's calendar rule.

Completion condition:

Preserve a baseline; change only `report.cjs`; rerun unchanged tests; report the result and limits for author review.

Orient

The inspected files were `report.cjs` and `report.test.cjs`. The function filtered records by status and completion timestamp, then returned their count and IDs. Inspection found the upper bound used `<= end`.

The baseline command was run from the example folder:

```
node --test --test-reporter=tap --test-concurrency=1 report.test.cjs
```

Runtime: Node v24.19.0

Result: 8 tests; 7 passed; 1 failed; none skipped.

Exit status: 1

Failure: excludes the exact end instant

Expected: `completedCount 0; completedIds []`

Actual: `completedCount 1; completedIds ["at-end"]`

This was the deliberately seeded defect showing up in execution. The baseline source is preserved as `recorded/report.before.cjs`; the command record is `recorded/baseline.json`.

Constrain

May read:

`report.cjs`, `report.test.cjs`, `task-brief.md`, and the recorded results.

May modify for the repair:

`report.cjs` only. The tests must stay unchanged.

May execute:

The local Node test command, with synthetic data.

Outside scope:

Network access, package installs, production data, deployment, database integration, authentication, UI, and time-zone conversion.

Stop condition:

If the fix requires a different contract or broader application changes, return the dependency for a scope decision before editing.

The setup created the source, tests, and task brief. The repair boundary began after that setup and the baseline run. Counting those as one undifferentiated change would hide what the implementation actually touched.

Use the right worker

One assistant performed the role passes. The inspection identified the comparison. Implementation changed it from `<= end` to `< end`. The test pass reran the same eight checks. The review pass compared the source versions, unchanged test hashes, outputs, and task brief.

No independent reviewer or parallel worker is claimed. This task didn't need a larger agent arrangement to demonstrate the handoff between an assignment and its evidence.

Show the evidence

Status: Ready for author review of the demonstration.

Generated:
Task brief, synthetic fixtures, report function, eight tests, one-line correction, command records, and review summary.

Executed:
The same Node test command before and after the repair.
Baseline: 7 passed, 1 failed, exit 1.
Corrected: 8 passed, 0 failed, exit 0.
No tests skipped in either run.

Observed:
The upper-bound test rejected the original function.
The corrected function excluded the "at-end" record.
Only the comparison changed between preserved source versions.
The baseline and corrected test-file SHA-256 values match.

Not run:
Database, UI, authorization, time-zone conversion, and production checks: those systems are not part of this local teaching fixture.

Approved:
The user requested preparation of the editorial improvements.
Author acceptance of this demonstration remains pending.
No production or public-release approval is implied.

Measurement:
Human review time and model usage were not measured.
Test durations are present in the raw output; no savings claim follows.

The corrected command record is recorded/corrected.json. Both JSON records include the runtime version, relative invocation, timestamps, output, exit status, and source fingerprints. Their matching test hashes let a reader check that a weaker test wasn't substituted to obtain the pass.

Follow-up and review

Item	Disposition
Business-zone conversion	Outside this fixture; specify and test it before adapting the function to a real report
Invalid input	Caller validation is assumed here; inspect the actual input boundary in a real application
Access control and integration	Not exercised; require separate evidence when those layers exist
Broader test claim	Eight checks support this contract, not the correctness of every possible report
Author acceptance	Pending; no signature or acceptance date supplied

These are adaptation limits, not evidence that an existing customer system has those defects.

Strengthen the system

The proposed reusable rule is small:

When a report uses a time interval, write down the date field and both endpoint rules. Include a test at each exact endpoint, not just a record comfortably inside the range.

That rule is a recommendation from this demonstration, not a claimed change to a customer's operating policy. The task is useful because another developer can inspect it, run it, and disagree with the size of the claim without having to trust the completion summary.

Project Guide

Purpose

What does this application do?

Structure

Where are the main entry points, services, data access layers, tests, and configuration files?

Run and test

What commands install, start, build, lint, and test the project?

Conventions

Which patterns should be followed for naming, errors, data access, UI, and logging?

Sensitive areas

Which files, directories, data, commands, or environments require extra care?

Definition of a safe change

What must be true before a change can be considered ready for review?

Agent Operating Rules

1. Inspect the relevant project structure before editing.
2. State important assumptions before relying on them.
3. Modify only files required by the task.
4. Stop and ask when a business rule or high-risk action is ambiguous.
5. Never expose, request, or commit secrets.
6. Reuse established project patterns unless there is a documented reason not to.
7. Run the relevant checks after making changes.
8. Report exact commands, results, changed files, and checks not run.
9. Put useful but unauthorized discoveries in the follow-up ledger.
10. Don't claim approval merely because code was generated or executed.

Task Brief

Request

User or system affected

Desired result

Acceptance criteria

Approved files or area

Must not change

Known constraints

Open questions

Required checks

Definition of done

Delegated Assignment

Role

Objective

What single result is needed?

Context

Only the files, decisions, and background needed for this assignment.

Permissions

- May read:
- May modify:
- May execute:
- Must not:

Expected output

Return findings, changed files, test results, risks, and open questions.

Completion criteria

The assignment is complete when: -

Review Checklist

- [] Does the change solve the stated problem?
- [] Does it follow existing project patterns?
- [] Did it stay within scope?
- [] Did it preserve unrelated behavior?
- [] Are permissions and security requirements handled?
- [] Are edge cases covered?
- [] Did the stated checks actually run?
- [] Do the results support the completion claim?
- [] Is anything still waiting on human judgment?

Agent Incident Record

What happened

What the agent was asked to do

What it actually did

Impact

Why the framework allowed it

Rule, test, boundary, or routing decision to add

How we will know the fix works

Follow-up ledger

Task: Date: Owner:

Finding and location	Why it matters	Needed for this task?	Decision	Next owner or action

Record useful discoveries here when they fall outside the assignment. A ledger entry does not authorize implementation. If the current task cannot be completed without a broader change, explain the dependency and obtain a scope decision.

Workflow Measurement

Task

Date

Worker roles used

Initial estimate

Attempts or retries

Files inspected

Files changed

Checks run

Review findings

Human time spent

What caused rework?

What should change in the next framework version?

Command history

Task identifier: Artifact label: Recorded only after execution Source version or file hashes:

Command entry

Executable and arguments: Working directory: Environment or fixture used, without secrets: Started at: Ended at: Exit status: Output file or retained output: Behavior this check supports: What it does not establish:

Checks not run

Check: Reason: Effect on the completion claim: Who must resolve it:

Measurement limits

Command elapsed time: Human setup time, if measured: Human review time, if measured: Retries and their causes:

Keep automated command duration separate from human effort. Record failed checks as well as successful ones. A later passing run should not replace the failure record.

Evidence packet index

Task: Owner: Current status:

Record	Location	Provenance	Open decision
Original request			
Initial source state			
Task brief			
Planner findings			
Approved scope			
Exact worker assignments			
Changed-file list and diff			
Command history and outputs			
Reviewer findings			
Follow-up ledger			
Approval decision			
Framework change			

Final evidence states

Generated: Executed: Observed: Approved by whom and for what action:

Limits

Checks or evidence missing: Human review time if measured: Comparison or baseline if a savings claim is made: Next action:

A report boundary, recorded

This example is deliberately small: a JavaScript function returns the count and IDs of completed records between two supplied instants. Its first version included the end instant when the contract said to exclude it.

What is real here

The application, data, and original defect were constructed by the assistant for the September 15, 2026 editorial revision. The before-and-after executions and their output are real. This is not a client incident, a production integration test, or historical evidence for Chapter 10.

One assistant performed the passes. Human review time and model usage were not measured. Author acceptance of the material remains pending.

Read or run

1. Read `task-brief.md` for the contract and boundary.
2. Compare `recorded/report.before.cjs` with `report.cjs`. The correction is `<= end` to `< end`.
3. Read `report.test.cjs` and both recorded JSON files. The test hashes match.
4. From this directory, run:

```
node --test --test-reporter=tap --test-concurrency=1 report.test.cjs
```

The saved run used Node v24.19.0. The corrected checkpoint contains eight passes, no failures, no skipped tests, and exit status 0. The baseline has seven passes, one failure, and exit status 1. The failing assertion names the extra record, `at-end`.

To reproduce the original failure without overwriting this corrected copy, create a separate disposable directory. Copy `recorded/report.before.cjs` into it as `report.cjs`, copy `report.test.cjs` alongside it, and run the same command there. An exit status of 1 is expected for that deliberately faulty version.

The native test runner and TAP reporting are documented in the [Node 24 test-runner reference](#). No third-party test package is used.

What the records contain

`recorded/baseline.json` and `recorded/corrected.json` preserve UTC timestamps, runtime version, relative working directory and arguments, source fingerprints, stdout, stderr, and exit status. The `.tap.txt` files are reading copies of those outputs. Absolute local directory details were removed from distributed diagnostics. No test names or outcomes were changed.

An initial baseline capture exposed a path-sanitization gap in the recording helper. That attempt was kept in the author's private revision record. The helper was corrected and the distributed baseline was rerun before the application fix; the function and test inputs were unchanged between those two baseline runs.

Boundary of the claim

Inputs are assumed to be validated ISO timestamps and a valid interval. UTC endpoints are a fixture choice. A real report must establish its business date field and operating time zone before translating the period into instants.

No database, UI, network service, authorization rule, malformed-input behavior, or production environment was exercised. Those omissions are not missing results to fill with a confident summary. They are the boundary of this demonstration.

The complete FOCUS record appears in `../worked-example.md`. The internal recording helper is retained with the editable book sources; ordinary readers only need the function, tests, and saved checkpoints.