A VISUAL REFERENCE · ROOT&SHAPE

# ROOT&SHAPE

**Explain the program. Rebuild the result.**

ROOT the tools. SHAPE the workflow.

Follow one fictional research question from a raw TSV to checked means, a native figure and a readable report. Eight chapters connect 32 executable stations. The literary source owns the programs; the raw data and independent expectations have separate, explicit roles.

> **Last changes: 2026-11-01.** First complete edition: 32 real experiments, native Org tangling, AWK checks, gnuplot figures, parallel GNU Make, provenance and HTML/PDF reports.

---

**32**
worked examples

**8**
topic families

**1**
canonical Org source

READING THE TOUR

# Four views of one explained program

### Read

Follow the question, assumptions and named literary fragment. A learning goal and a concrete decision accompany every station.

### Source and Tangled

Open the complete Org source used for that experiment. Inspect or download the actual extracted programs; helper blocks are not duplicate files.

### Result

Compare captured native output, tables and figures. Download the source-only project or repeat a station from its complete experiment bundle.

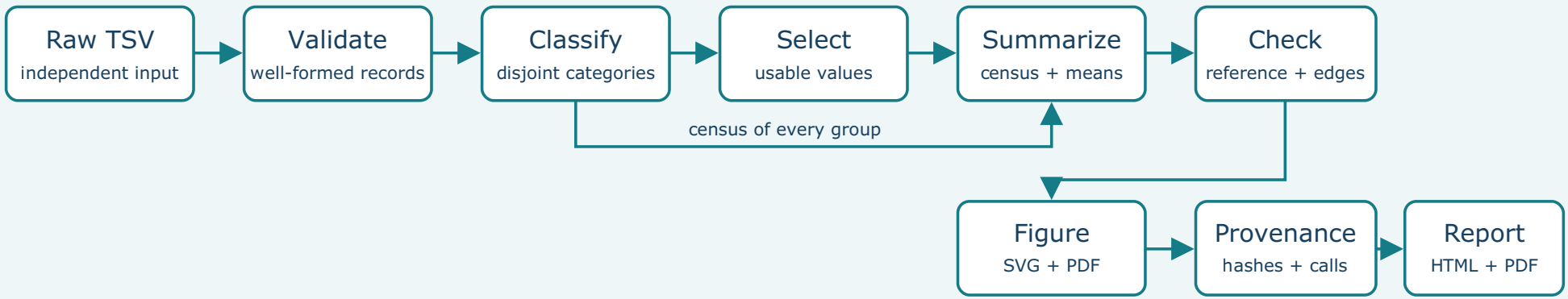
Each station runs in a fresh private project copy. Baseline inputs are never edited by the build. Source views in change experiments intentionally show that experiment’s changed source; each repeatable station bundle begins from the canonical baseline. Setup commands appear in the complete runnable example.

# ROOT the tools. SHAPE the workflow.

- **R — Robust:** validate before accepting a handover; retain NA and real zero distinctly; fail with a useful diagnostic.
- **O — Open:** readable Org, TSV, AWK, Make and gnuplot sources, plus independently usable programs and open output formats.
- **O — Ongoing:** change a question, input or explanation together with the justified expectations and rebuild its consequences.
- **T — Time-tested:** use established tools consciously and document the actual versions and prerequisites.
- **S — Structure stages:** give the scientific processing steps clear tasks.
- **H — Harmonize handovers:** agree on formats, columns, units, missing values and interfaces.
- **A — Automate actions:** make the agreed workflow executable with explicit dependencies.
- **P — Propagate provenance:** connect sources, transformations and results with evidence.
- **E — Evaluate explicitly:** formulate expected behavior and check concrete cases.

SHAPE describes five design perspectives across the workflow. These letters do not prescribe five execution steps. The actual dependency graph is shown below.

## One question. Explicit handovers. Checked evidence.




SHAPE is a set of design perspectives across this graph, not five execution steps.

# Build and download the study

Download the standalone source-only project ZIP or sources plus captured reference results. Open the actual reference HTML report, reference PDF report, provenance JSON, or native SVG figure.

With the documented prerequisites installed, extract the project, enter its folder, and run `python3 build.py all --jobs 4`. Then run `python3 build.py verify`. Inside this tour checkout, `make example` selects the existing publisher Python. `make full` prepares the reviewed 32 stations and publishes this tour locally.

Requirements: Python 3.10+ with the supplied requirements.txt installed; batch Emacs with Org; AWK with /dev/stderr support; gnuplot with svg and pdfcairo; GNU Make 4.3+ (gmake is preferred, otherwise make is verified). No build uses a network connection, account, Doom configuration or personal Emacs init file. Installing prerequisites is a separate setup step, as in the rest of this collection.

Generated files stay below build/. The maintained bootstrap owns first-time tangling; the generated workflow never needs to create itself. Make targets separate results, check, figure, provenance and report. A working Org report copy receives the table; the maintained research.org is unchanged. Unchanged rebuilds run no scientific stage. Verification checks recorded identities rather than claiming universal cross-platform PDF or SVG byte identity.

# Foundations and primary references

The Org tour's Babel chapter, tangling example, named-fragment example and tangle-only noweb example cover the mechanics. This tour concentrates on the design of a complete explained program. Continue with the AWK, GNU Make and gnuplot tours for individual tools.

- Donald E. Knuth: Literate Programming — explain the program in a useful reading order, then extract its computational structure.
- Org manual: Extracting Source Code — tangling named program text.
- Org manual: Noweb Reference Syntax — textual references, evaluation calls and expansion contexts.
- Org manual: Exporting Code Blocks — controlling the published representation.
- Schulte et al. (2012), A Multi-Language Computing Environment for Literate Programming and Reproducible Research, DOI 10.18637/jss.v046.i03 — explanation and executable research in Org.
- Norman Ramsey: noweb guide — the separate noweb system. Org's noweb reference syntax is not a dependency on the standalone noweb executable; this project requires no noweb installation.

Explanations, programs, expectations and fictional data are original teaching material. The references support the method and tool semantics; no research measurements or quoted passages are reused.

## 01 - Start with the question

Name the scientific transformations before choosing commands. ROOT — Open is evidenced by a separate, inspectable raw TSV; SHAPE — Structure stages is evidenced by the six explicit handovers.

Chapter focus: **Structure stages**.

RSH-01-001

# Define the denominator

Learning goal: distinguish all collected records from measurements that can estimate a mean. The question is “What is the mean usable value in each fictional measurement group?” Values use arbitrary units (a.u.); this is a teaching dataset, not an empirical claim. Count the raw rows first: each group has five records, but five will not be the denominator of either mean.

Decide the stages explicitly: raw input → validation → classification/selection → group summary → figure → explanatory report. Validation establishes whether a record is well formed; selection decides whether a well-formed record is scientifically usable. A rejected row still contributes to the census. The observed A/B totals below are the first evidence for that distinction.

### Code

```
awk -F '\t' 'NR>1 {n[$2]++} END {print "A",n["A"]; print "B",n["B"]}' data/
measurements.tsv
```

### Actual output

Captured run · exit 0

```
A 5
B 5
```

Download: Driver only · RSH-01-001.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** console/source above · **Result:** console/source above

### Read · summary-program

```
#+begin_src awk :tangle programs/summarize.awk :mkdirp yes :noweb tangle
<<summarize-by-group>>
<<ordered-groups>>
#+end_src
```

RSH-01-002

# Keep the raw input independent

Learning goal: identify a file's ownership before changing it. The complete input below is a maintained TSV, with real tab separators and one exact header. It is never generated from the explanation or overwritten by the build. The ten identifiers let us trace an observation through the handovers.

A03 and B04 are missing; A04 is a measured zero. A05 and B05 have invalid quality status. Do not delete those rows to obtain a convenient mean: retaining them allows the final table to explain exclusions. Every station runs in a fresh private copy of this project, so later edits remain experiments. Exercise: find the row whose accidental truth-value test would lose a legitimate observation; it is A04.

Compare with RSH-01-001 · Define the denominator.

### Code

```
cat data/measurements.tsv
```

### Actual output

Captured run · exit 0

```
sample_id       group   value   status
A01     A       10      valid
A02     A       20      valid
A03     A       NA      valid
A04     A       0       valid
A05     A       100     invalid
B01     B       5       valid
B02     B       15      valid
B03     B       25      valid
B04     B       NA      valid
B05     B       40      invalid
```

Download: Driver only · RSH-01-002.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** console/source above · **Result:** console/source above

## Read · analysis-contract

```
#+begin_src json :tangle no :noweb tangle
{
  "units": "a.u.",
  "missing": "NA",
  "status": [
    "valid",
    "invalid"
  ],
  "exclusion_priority": [
    "invalid",
    "NA"
  ],
  "locale": "C",
  "output_order": "ASCII group ascending",
  "number_format": "finite decimal/scientific; abs(value)<=1000000",
  "mean_format": "two decimal places; NA when n_used=0",
  "empty_input": "zero bytes: error 2; header only: no groups"
}
#+end_src
```

RSH-01-003

# Work out the reference by hand

Learning goal: author an expectation before trusting a larger pipeline. For A, the usable values are 10, 20 and 0: sum 30, denominator 3, mean 10.00. For B, 5, 15 and 25 give sum 45, denominator 3, mean 15.00. The small independent AWK calculation below makes that arithmetic visible; the maintained expected-summary.tsv records the full expected table separately from the generated programs.

This first calculation assumes the supplied fixture is valid. It is not yet a general input validator. Its intentionally narrow role is to establish an oracle that the complete program must match. Exercise: counting A03 as zero would change the denominator to four and the mean to 7.50; that is a changed scientific assumption, not a formatting change.

Compare with RSH-01-002 · Keep the raw input independent.

### Code

```
awk -F '\t' 'NR>1 && $4=="valid" && $3!="NA" {s[$2]+=$3; n[$2]++} END {split("A B",g,"
"); for(i=1;i<=2;i++) printf "%s: %.0f / %d = %.2f\n",g[i],s[g[i]],n[g[i]],s[g[i]]/
n[g[i]]}' data/measurements.tsv
```

### Actual output

Captured run · exit 0

```
A: 30 / 3 = 10.00
B: 45 / 3 = 15.00
```

Download: Driver only · RSH-01-003.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** console/source above · **Result:** expected-summary.tsv

### Read · summarize-by-group

```
#+begin_src awk :tangle no :noweb tangle
BEGIN { FS=OFS="\t" }
FILENAME==ARGV[1] {
 if (FNR==1) next
 g=$2
 if (!(g in total)) groups[++ng]=g
 total[g]++
 if ($4=="missing") missing[g]++
 if ($4=="rejected") rejected[g]++
 next
}
FILENAME==ARGV[2] {
 if (FNR==1) next
 g=$2
 used[g]++; sum[g]+=$3
}
#+end_src
```

RSH-01-004

# Observe the complete data path

Learning goal: connect the reference arithmetic to explicit intermediate files. The results target runs validation, classification, selection and summary, leaving all four TSV handovers available. The final census is five per group, with three used, one missing and one rejected. The six-column table is an interface used unchanged by checks, gnuplot and the report.

The program enforces n_total = n_used + n_missing + n_rejected. Classification happens before selection so exclusions remain countable after unusable rows have disappeared from the three-column usable file. The full build will later add checks, figure, provenance and report; this station deliberately stops at the scientific result. Exercise: inspect usable.tsv and confirm that A04 survives while A03 does not.

Compare with RSH-01-003 · Work out the reference by hand.

## Code

```
python3 build.py results --jobs 4
cat build/results/summary.tsv
```

## Actual output

Captured run · exit 0

```
build: results; actions: validate,classify,select,summarize
group   n_total n_used  n_missing       n_rejected      mean
A       5       3       1       1       10.00
B       5       3       1       1       15.00
```

Download: Driver only · RSH-01-004.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** summarize.awk · **Result:** usable.tsv · summary.tsv

## Read · workflow

```
#+begin_src makefile :tangle no :noweb tangle
.DEFAULT_GOAL := all
.RECIPEPREFIX := >
.DELETE_ON_ERROR:
.SUFFIXES:
PYTHON ?= python3
AWK ?= awk
RUN = $(PYTHON) build/tools/pipeline.py
.PHONY: all results check figure provenance report
all: report
results: build/results/summary.tsv
check: build/checks.json
figure: build/results/figure.svg build/results/figure.pdf
provenance: build/provenance.json
report: build/site/index.html build/site/report.pdf
build/results/validated.tsv: data/measurements.tsv build/raw.sha256 build/environment.json build/programs/validate.awk build/tools/pipeline.py
>@$(RUN) stage validate $(AWK)
build/results/classified.tsv: build/results/validated.tsv build/programs/classify.awk build/tools/pipeline.py
>@$(RUN) stage classify $(AWK)
build/results/usable.tsv: build/results/classified.tsv build/programs/select.awk build/tools/pipeline.py
>@$(RUN) stage select $(AWK)
build/results/summary.tsv: build/results/classified.tsv build/results/usable.tsv build/programs/summarize.awk build/tools/pipeline.py
>@$(RUN) stage summarize $(AWK)
build/checks.json: build/results/summary.tsv tests/expected-summary.tsv tests/cases.json $(wildcard tests/fixtures/*.tsv) $(wildcard build/programs/*.awk) build/tools/pipeline.py
>@$(RUN) check $(AWK)
build/results/figure.svg build/results/figure.pdf &: build/results/summary.tsv build/checks.json build/programs/plot.gp build/tools/pipeline.py
>@$(RUN) plot
build/provenance.json: build/results/figure.svg build/results/figure.pdf build/checks.json build/source.json build/environment.json build/invocation.json build/tangle.json $(wildcard
build/programs/*) build/workflow.mk build/tools/pipeline.py
>@$(RUN) provenance
build/site/index.html build/site/report.pdf &: build/provenance.json build/programs/report-template.org build/programs/report.css build/tools/pipeline.py bootstrap.el
>@$(RUN) report
#+end_src
```

## 02 - Give ideas a name

Name ideas so that explanation can follow the reader rather than the interpreter. ROOT — Ongoing is evidenced by a section-reordering experiment with equal program hashes. SHAPE — Structure stages is evidenced by separately named validation, classification, selection, summary and reporting fragments.

Chapter focus: **Structure stages and Ongoing**.

RSH-02-001

# Start from a small working calculation

Learning goal: recognize the limit of a short, functional script before splitting it into named ideas. The initial calculation combines eligibility, accumulation and presentation in one expression. It already includes measured zero because it tests the missing marker explicitly. It gives the reference means, but says little about malformed input, exclusions or the provenance of those numbers.

Literate programming adds an explained program structure rather than replacing working code with prose. We keep this tiny script as a comparison, then give validation, selection, summary and reporting their own fragments. Do not extend this shorthand into the final validator: a duplicate ID or unknown status must be an error. Exercise: identify which three responsibilities share one line here, then locate their names in research.org.

### Code

```
awk -F '\t' 'NR>1 && $4=="valid" && $3!="NA" {sum[$2]+=$3; used[$2]++} END {split("A
B",g," "); for(i=1;i<=2;i++) printf "%s\t%d\t%.2f\n",g[i],used[g[i]],sum[g[i]]/
used[g[i]]}' data/measurements.tsv
```

### Actual output

Captured run · exit 0

```
A       3       10.00
B       3       15.00
```

Download: Driver only · RSH-02-001.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** console/source above · **Result:** console/source above

### Read · summary-program

```
#+begin_src awk :tangle programs/summarize.awk :mkdirp yes :noweb tangle
<<summarize-by-group>>
<<ordered-groups>>
#+end_src
```

RSH-02-002

# Give the summary a readable skeleton

Learning goal: distinguish a textual reference from a computation. This master block says what the summary program contains: summarize-by-group followed by ordered-groups. Both expand other named fragments. report-exclusions is explained early in the document because the output contract helps the reader, although its code runs inside END after every row has been read.

The master uses :noweb tangle, so extraction expands <<fragment>> while publication can show the reference. Helpers use :tangle no to avoid a second top-level copy of their code. Org's <<fragment()>> form evaluates a block and inserts its result; this project uses no evaluated noweb calls. Follow the linked Org-tour foundations for Babel mechanics; here the important decision is how names connect the scientific explanation to an actual standalone program.

Compare with RSH-02-001 · Start from a small working calculation.

### Code

```
python3 ../tools/stations.py fragment summary-program
```

### Actual output

Captured run · exit 0

```
<<summarize-by-group>>
<<ordered-groups>>
```

Download: Driver only · RSH-02-002.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** console/source above · **Result:** console/source above

### Read · summary-program

```
#+begin_src awk :tangle programs/summarize.awk :mkdirp yes :noweb tangle
<<summarize-by-group>>
<<ordered-groups>>
#+end_src
```

RSH-02-003

# Reorder the explanation without changing the program

Learning goal: demonstrate that reader order and program order are independent. The experiment moves the complete Count every exclusion and Validate before publishing sections, including their named blocks. It changes the literary source, retangles it and compares SHA-256 values for all ten generated files. The references and their order inside master blocks stay unchanged.

Equal hashes are a concrete, local claim about this experiment. Moving text inside a fragment, renaming a referenced fragment or changing master-reference order could change the program. Exercise: put validation first for a new audience, then perform the same comparison. This is Ongoing in practice: reorganize an explanation without accidentally changing the computation. The Source view for this station contains the reordered source actually used, while the download bundle starts from the canonical baseline and repeats the experiment.

Compare with RSH-02-002 · Give the summary a readable skeleton.

### Code

```
python3 ../tools/stations.py reorder
```

### Actual output

Captured run · exit 0

```
Reader sections reordered: Count every exclusion / Validate before publishing
Tangled file hashes unchanged: 10/10
```

Download: Driver only · RSH-02-003.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** summarize.awk · **Result:** tangle.json

### Read · summary-program

```
#+begin_src awk :tangle programs/summarize.awk :mkdirp yes :noweb tangle
<<summarize-by-group>>
<<ordered-groups>>
#+end_src
```

RSH-02-004

# Check expansion and single ownership

Learning goal: verify what tangling actually produced. Ten master targets own ten output files: four AWK programs, a gnuplot program, the contract, report template, stylesheet, Makefile and Python execution/provenance helper. The helper fragments produce no separate files. The generated AWK files must contain no unresolved textual references, and the validator's fail function must appear exactly once.

The assertions below inspect the generated bytes, not a parallel handwritten listing. This catches a common mistake: giving both a helper and its containing master the same tangle target, which appends duplicate code. Exercise: compare the validator master with its two helpers in the full source. Keep the helper's :tangle no even when moving its explanatory section. Open means the resulting AWK program remains readable and usable independently of Org.

Compare with RSH-02-003 · Reorder the explanation without changing the program.

### Code

```
python3 ../tools/stations.py identities
```

### Actual output

Captured run · exit 0

```
10 master targets; helper fragments: :tangle no
Unexpanded references: 0; validation function definitions: 1
```

Download: Driver only · RSH-02-004.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** validate.awk · **Result:** tangle.json

### Read · validate-program

```
#+begin_src awk :tangle programs/validate.awk :mkdirp yes :noweb tangle
<<validation-setup>>
<<validate-input>>
#+end_src
```

## 03 - Agree on the handovers

Write the contract at each boundary: field names, types, units, order and missing values. SHAPE — Harmonize handovers is evidenced by contract.json and two deliberately different treatments of the same invalid/NA record. These perspectives apply across stages, not as a five-step execution sequence.

Chapter focus: **Harmonize handovers**.

RSH-03-001

# Publish the data contract

Learning goal: make implicit conventions inspectable. contract.json is tangled from the same explanation as the programs. Raw TSV has exactly sample_id, group, value and status; IDs are unique ASCII identifiers and groups are uppercase ASCII codes. Values are finite decimal or scientific notation in a.u., within ±1000000. NA alone denotes missingness; valid and invalid are the only quality states.

The build fixes LC_ALL=C, decimal points and ASCII group order; means use two decimals. A zero-byte input is a status-2 error because even the schema is absent. A header-only file is valid, contains no groups and yields only a summary header. Groups with no usable measurements retain their census and mean NA. Exercise: explain why a blank value or the word missing is rejected rather than silently converted to zero.

### Code

```
cat build/programs/contract.json
```

### Actual output

Captured run · exit 0

```
{
  "units": "a.u.",
  "missing": "NA",
  "status": [
    "valid",
    "invalid"
  ],
  "exclusion_priority": [
    "invalid",
    "NA"
  ],
  "locale": "C",
  "output_order": "ASCII group ascending",
  "number_format": "finite decimal/scientific; abs(value)<=1000000",
  "mean_format": "two decimal places; NA when n_used=0",
  "empty_input": "zero bytes: error 2; header only: no groups"
}
```

Download: Driver only · RSH-03-001.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** contract.json · **Result:** console/source above

## Read · analysis-contract

```
#+begin_src json :tangle no :noweb tangle
{
  "units": "a.u.",
  "missing": "NA",
  "status": [
    "valid",
    "invalid"
  ],
  "exclusion_priority": [
    "invalid",
    "NA"
  ],
  "locale": "C",
  "output_order": "ASCII group ascending",
  "number_format": "finite decimal/scientific; abs(value)<=1000000",
  "mean_format": "two decimal places; NA when n_used=0",
  "empty_input": "zero bytes: error 2; header only: no groups"
}
#+end_src
```

RSH-03-002

# Replace missingness with an observed zero

Learning goal: observe the consequence of a changed input meaning. In this isolated copy, A03 changes from NA to the real value 0. The raw baseline in the downloadable canonical project stays unchanged. Rebuilding through results gives A four usable observations, zero missing, one rejected and mean 7.50. B is unchanged.

This edit does not change the exclusion rule: A04 was already a real zero and remains used. The test expectations for the original fixture would now fail, so this experiment stops before the publication checks. A real correction to the dataset would require reviewing the independent expected table and the evidence for the correction. Exercise: predict the effect of replacing B04 with zero: four used values sum to 45, giving 11.25. Distinguishing data correction from rule change keeps the handover meaningful.

Compare with RSH-03-001 · Publish the data contract.

### Code

```
python3 ../tools/stations.py zero-change
```

### Actual output

Captured run · exit 0

```
group   n_total n_used  n_missing       n_rejected      mean
A       5       4       0       1       7.50
B       5       3       1       1       15.00
```

Download: Driver only · RSH-03-002.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** select.awk · **Result:** summary.tsv

### Read · select-usable-measurements

```
#+begin_src awk :tangle no :noweb tangle
BEGIN { FS=OFS="\t" }
NR==1 { print "sample_id","group","value"; next }
$4=="used" { print $1,$2,$3 }
#+end_src
```

RSH-03-003

# Make exclusion priority a scientific decision

Learning goal: distinguish two plausible classification policies with one counterexample. The fixture contains a row whose value is NA and whose status is invalid. The first run uses the agreed invalid-first rule and assigns rejected. The second changes only the conditional's precedence and assigns missing. The experiment executes both real AWK variants on that same row.

The final project uses invalid first, so missing and rejected do not overlap. n_total can therefore be partitioned exactly. Reporting both categories for one row would inflate the exclusion count even if the computed mean happened to stay the same. Exercise: justify the alternative policy in words, then update its expected category before adopting it. Do not interpret this local comparison as permission to change the canonical rule: the reference project retains the required invalid-first semantics.

Compare with RSH-03-002 · Replace missingness with an observed zero.

### Code

```
python3 ../tools/stations.py priority-change
```

### Actual output

Captured run · exit 0

```
invalid first: rejected
NA first: missing
```

Download: Driver only · RSH-03-003.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** classify.awk · **Result:** priority.tsv

### Read · classify-measurements

```
#+begin_src awk :tangle no :noweb tangle
BEGIN { FS=OFS="\t" }
NR==1 { print "sample_id","group","value","category"; next }
{
 category=($4=="invalid" ? "rejected" : ($3=="NA" ? "missing" : "used"))
 print $1,$2,$3,category
}
#+end_src
```

RSH-03-004

# Hand over values without losing the census

Learning goal: choose interfaces for two different consumers. classified.tsv keeps all ten validated records and assigns exactly one category to each. usable.tsv contains only sample_id, group and value for the six usable records, including A04 with value zero. The summary consumes both files: classified supplies total/missing/rejected counts, usable supplies values and their denominator.

That dual input is intentional: a group with no usable values would vanish if the summary discovered groups only from usable.tsv. No unit conversion happens here; every numeric handover remains in a.u. Exercise: follow A03 through raw, validated and classified, then explain its absence from usable without losing its contribution to n_total and n_missing. The Result download includes both handovers, so this decision can be checked row by row.

Compare with RSH-03-003 · Make exclusion priority a scientific decision.

### Code

```
cat build/results/usable.tsv
```

### Actual output

Captured run · exit 0

```
sample_id	group	value
A01	A	10
A02	A	20
A04	A	0
B01	B	5
B02	B	15
B03	B	25
```

Download: Driver only · RSH-03-004.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** classify.awk · select.awk · **Result:** classified.tsv · usable.tsv

### Read · select-usable-measurements

```
#+begin_src awk :tangle no :noweb tangle
BEGIN { FS=OFS="\t" }
NR==1 { print "sample_id","group","value"; next }
$4=="used" { print $1,$2,$3 }
#+end_src
```

## 04 - Tangle and run

Separate extraction from execution and publication. SHAPE — Automate actions is evidenced by a maintained bootstrap that tangles before invoking the generated Makefile. ROOT — Time-tested requires real prerequisites: batch Emacs/Org, AWK, GNU Make 4.3+ grouped targets and gnuplot svg/pdfcairo terminals.

Chapter focus: **Automate actions**.

RSH-04-001

# Bootstrap before the generated Makefile exists

Learning goal: solve the first-build dependency explicitly. A maintained build.py and bootstrap.el exist before any generated program. Batch Emacs tangles research.org into a staging directory; the bootstrap installs changed files under build/ and only then calls GNU Make on build/workflow.mk. The generated Makefile does not need to create itself.

This station starts without build/. Its output counts ten installed targets. Emacs runs with -Q and Babel execution disabled; no Doom configuration, account or network is used. GNU Make 4.3+ is a real requirement because grouped targets prevent duplicate writers for output pairs. Exercise: remove build/ and run the command again. The same bootstrap must work without copying a previous Makefile from a successful run.

### Code

```
python3 build.py tangle
```

### Actual output

Captured run · exit 0

```
tangle: 10 files; changed: 10
```

Download: Driver only · RSH-04-001.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** workflow.mk · **Result:** tangle.json

### Read · make-program

```
#+begin_src makefile :tangle workflow.mk :mkdirp yes :noweb tangle
<<workflow>>
#+end_src
```

RSH-04-002

# Run a tangled program without Org

Learning goal: prove that extraction leaves an independently executable program. AWK reads the generated validate.awk and the maintained raw TSV; its successful output must match the original input exactly. cmp checks those bytes before the success message is printed. No Org session is involved in this execution.

The validator passes well-formed records through, including invalid quality status and NA: those are valid data states, not syntax errors. Classification makes the scientific inclusion decision later. Direct AWK may emit an initial valid prefix before a subsequent error; the workflow runner captures that output and installs a handover only on success, so a partial stream never becomes a published result. Exercise: run the same program on tests/fixtures/duplicate.tsv and inspect its nonzero status and useful line number.

Compare with RSH-04-001 · Bootstrap before the generated Makefile exists.

### Code

```
awk -f build/programs/validate.awk data/measurements.tsv > build/validated-direct.tsv
cmp data/measurements.tsv build/validated-direct.tsv
printf 'Standalone AWK validation: exact TSV passthrough\n'
```

Download: Driver only · RSH-04-002.sh

### Actual output

Captured run · exit 0

```
Standalone AWK validation: exact TSV passthrough
```

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** validate.awk · **Result:** validated-direct.tsv

### Read · validate-input

```
#+begin_src awk :tangle no :noweb tangle
NR==1 {
 if ($0!="sample_id\tgroup\tvalue\tstatus") fail("line 1: expected sample_id, group, value, status TSV header")
 print; next
}
{
 if (NF!=4) fail("line " NR ": expected 4 tab-separated fields, got " NF)
 if ($1 !~ /^[A-Za-z][A-Za-z0-9_-]*$/) fail("line " NR ": invalid sample_id")
 if (seen[$1]++) fail("line " NR ": duplicate sample_id " $1)
 if ($2 !~ /^[A-Z][A-Z0-9_]*$/) fail("line " NR ": group must be an uppercase ASCII identifier")
 if ($4!="valid" && $4!="invalid") fail("line " NR ": unexpected status " $4)
 if ($3!="NA" && $3 !~ number) fail("line " NR ": value must be a finite decimal number or NA")
 if ($3!="NA" && (($3+0)>1000000 || ($3+0)<-1000000)) fail("line " NR ": absolute value exceeds 1000000 a.u.")
 print
}
END { if (!failed && NR==0) fail("empty input: expected TSV header") }
#+end_src
```

RSH-04-003

# Execute explicit dependencies

Learning goal: see what the generated workflow actually runs. The results target has four file targets with explicit raw-data and program prerequisites. Each native command has a receipt recording its argv, successful status and output hash. The bootstrap fixes the locale and chooses the active Python environment, while AWK does the scientific calculation.

The observed action list is validate, classify, select, summarize. The generated Makefile, shown in Tangled, names both summary inputs and does not rely on shell-session state. A checksum side file also notices a changed raw input even if its mtime was preserved. Exercise: inspect summary.tsv and the summarize receipt, then identify which prerequisite would change if the usable-selection rule changed. Make tracks this real graph; SHAPE's five perspectives describe its design, not its execution order.

Compare with RSH-04-002 · Run a tangled program without Org.

### Code

```
python3 build.py results --jobs 4
cat build/results/summary.tsv
```

### Actual output

Captured run · exit 0

```
build: results; actions: validate,classify,select,summarize
group   n_total n_used  n_missing       n_rejected      mean
A       5       3       1       1       10.00
B       5       3       1       1       15.00
```

Download: Driver only · RSH-04-003.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** workflow.mk · **Result:** summary.tsv · summarize.json

### Read · workflow

```
#+begin_src makefile :tangle no :noweb tangle
.DEFAULT_GOAL := all
.RECIPEPREFIX := >
.DELETE_ON_ERROR:
.SUFFIXES:
PYTHON ?= python3
AWK ?= awk
RUN = $(PYTHON) build/tools/pipeline.py
.PHONY: all results check figure provenance report
all: report
results: build/results/summary.tsv
check: build/checks.json
figure: build/results/figure.svg build/results/figure.pdf
provenance: build/provenance.json
report: build/site/index.html build/site/report.pdf
build/results/validated.tsv: data/measurements.tsv build/raw.sha256 build/environment.json build/programs/validate.awk build/tools/pipeline.py
>@$(RUN) stage validate $(AWK)
build/results/classified.tsv: build/results/validated.tsv build/programs/classify.awk build/tools/pipeline.py
>@$(RUN) stage classify $(AWK)
build/results/usable.tsv: build/results/classified.tsv build/programs/select.awk build/tools/pipeline.py
>@$(RUN) stage select $(AWK)
build/results/summary.tsv: build/results/classified.tsv build/results/usable.tsv build/programs/summarize.awk build/tools/pipeline.py
>@$(RUN) stage summarize $(AWK)
build/checks.json: build/results/summary.tsv tests/expected-summary.tsv tests/cases.json $(wildcard tests/fixtures/*.tsv) $(wildcard build/programs/*.awk) build/tools/pipeline.py
>@$(RUN) check $(AWK)
build/results/figure.svg build/results/figure.pdf &: build/results/summary.tsv build/checks.json build/programs/plot.gp build/tools/pipeline.py
>@$(RUN) plot
build/provenance.json: build/results/figure.svg build/results/figure.pdf build/checks.json build/source.json build/environment.json build/invocation.json build/tangle.json $(wildcard
build/programs/*) build/workflow.mk build/tools/pipeline.py
>@$(RUN) provenance
build/site/index.html build/site/report.pdf &: build/provenance.json build/programs/report-template.org build/programs/report.css build/tools/pipeline.py bootstrap.el
>@$(RUN) report
#+end_src
```

RSH-04-004

# Separate tangle, execute and weave

Learning goal: identify three operations with different effects. Tangle creates standalone program text. Execute applies those programs to inputs and captures result files. Weave/export publishes an explanation using those already produced results. The three commands below make the boundaries observable: first ten program files, then four scientific actions, then checks, native plot, manifest and report.

The maintained research.org is never used as a scratch buffer for result insertion. The report is filled into build/site/report.org, then batch Org exports that working copy to HTML; the existing PDF libraries produce its PDF. Exercise: after building, compare research.org with its original checksum. Changing export styling should not change a mean, while changing a scientific fragment should invalidate the appropriate calculation. The Result receipt records the real export command and PDF API operation.

Compare with RSH-04-003 · Execute explicit dependencies.

### Code

```
python3 build.py tangle
python3 build.py results --jobs 4
python3 build.py all --jobs 4
```

### Actual output

Captured run · exit 0

```
tangle: 10 files; changed: 10
build: results; actions: validate,classify,select,summarize
build: all; actions: check,plot,provenance,report
```

Download: Driver only · RSH-04-004.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** pipeline.py · **Result:** report.json

### Read · pipeline-program

```
#+begin_src python :tangle tools/pipeline.py :mkdirp yes :noweb tangle
<<pipeline-utilities>>
<<execute-stages>>
<<evaluate-explicitly>>
<<render-native-figure>>
<<propagate-provenance>>
<<weave-explanation>>
<<verify-handover>>
<<pipeline-entry>>
#+end_src
```

## 05 - Make assumptions testable

State an expected result independently of the program, then try to disprove it. ROOT — Robust is evidenced by useful status-2 input errors and a status-3 conservation failure. SHAPE — Evaluate explicitly is evidenced by the authored reference and sixteen edge/error fixtures.

Chapter focus: **Evaluate explicitly and Robust**.

RSH-05-001

# Check the independently authored reference

Learning goal: test meaning as well as successful execution. The check target compares the complete summary bytes with tests/expected-summary.tsv, written from the hand calculation rather than copied from program output. It also runs sixteen fixture cases through native AWK and checks their expected rows, statuses and diagnostics. The short display below verifies the reference and the conservation equation again.

The expected A and B means are 10.00 and 15.00, each with counts 5 = 3 + 1 + 1. These are scientific expectations for the recorded input and environment, not a claim that every platform will emit byte-identical figures or PDFs. Exercise: intentionally change A's expected mean to 10.01 in a private copy. The build must stop at check rather than quietly publish a new figure.

### Code

```
python3 ../tools/stations.py reference
```

### Actual output

Captured run · exit 0

```
A: 5 = 3 + 1 + 1; mean=10.00
B: 5 = 3 + 1 + 1; mean=15.00
Native tests: reference + 16 edge/error cases passed
```

Download: Driver only · RSH-05-001.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** pipeline.py · **Result:** checks.json · expected-summary.tsv

## Read · evaluate-explicitly

```
#+begin_src python :tangle no :noweb tangle
def check(awk):
 expected=Path('tests/expected-summary.tsv').read_bytes()
 if Path('build/results/summary.tsv').read_bytes()!=expected:raise RuntimeError('reference summary differs from authored tests/expected-summary.tsv')
 cases=json.loads(Path('tests/cases.json').read_text());outcomes=[];commands=[]
 for case in cases:
  if not re.fullmatch('[a-z0-9-]+',case['name']):raise RuntimeError('invalid test case name')
  directory=Path('build/tests')/case['name'];directory.mkdir(parents=True,exist_ok=True)
  p,paths=awk_pipeline(case['file'],directory,awk,commands)
  if p.returncode!=case['exit']:raise RuntimeError(case['name']+': wrong error status')
  if case['error'] and p.stderr.strip()!='input error: '+case['error']:raise RuntimeError(case['name']+': wrong diagnostic')
  if not case['exit']:
   actual=[{k:r[k] for k in KEYS} for r in read(paths[-1])]
   if actual!=case['rows']:raise RuntimeError(case['name']+': incorrect scientific result')
   for row in actual:
    if int(row['n_total'])!=sum(int(row[k]) for k in ('n_used','n_missing','n_rejected')):raise RuntimeError('count conservation failed')
  outcomes.append({'case':case['name'],'status':p.returncode,'passed':True})
 js('build/checks.json',{'reference':'passed','cases':outcomes,'count_conservation':'passed','scope':'native AWK reference and specified edge cases; no cross-platform byte claim'})
 receipt('check',commands,['build/checks.json'],[{'api':'Python bytes equality','expected':'tests/expected-summary.tsv','actual':'build/results/summary.tsv'}])
#+end_src
```

RSH-05-002

# Turn malformed input into useful errors

Learning goal: distinguish invalid quality from invalid syntax. A quality status of invalid is an allowed state and excludes a row later. An unknown status is a schema error. The native runs below demonstrate duplicate IDs, an unexpected status, three instead of four fields, and a nonnumeric value. Each stops with status 2 and identifies the offending line.

Numeric syntax is checked even for rejected rows, so junk values cannot hide behind quality exclusions. Other authored cases test the header, zero-byte input, out-of-range exponent and empty group. The runner captures native stderr and refuses to install an unsuccessful handover. Exercise: repair the duplicate fixture by changing only the second ID; predict that both rows then contribute to the group census. A useful diagnostic explains what the user can fix without fabricating a scientific result.

Compare with RSH-05-001 · Check the independently authored reference.

## Code

```
for name in duplicate status fields number; do
  python3 ../tools/stations.py case "$name"
done
```

## Actual output

Captured run · exit 0

```
duplicate: exit 2
input error: line 3: duplicate sample_id X01
status: exit 2
input error: line 2: unexpected status unknown
fields: exit 2
input error: line 2: expected 4 tab-separated fields, got 3
number: exit 2
input error: line 2: value must be a finite decimal number or NA
```

Download: Driver only · RSH-05-002.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** validate.awk · **Result:** duplicate.tsv · status.tsv

### Read · validate-input

```
#+begin_src awk :tangle no :noweb tangle
NR==1 {
 if ($0!="sample_id\tgroup\tvalue\tstatus") fail("line 1: expected sample_id, group, value, status TSV header")
 print; next
}
{
 if (NF!=4) fail("line " NR ": expected 4 tab-separated fields, got " NF)
 if ($1 !~ /^[A-Za-z][A-Za-z0-9_-]*$/) fail("line " NR ": invalid sample_id")
 if (seen[$1]++) fail("line " NR ": duplicate sample_id " $1)
 if ($2 !~ /^[A-Z][A-Z0-9_]*$/) fail("line " NR ": group must be an uppercase ASCII identifier")
 if ($4!="valid" && $4!="invalid") fail("line " NR ": unexpected status " $4)
 if ($3!="NA" && $3 !~ number) fail("line " NR ": value must be a finite decimal number or NA")
 if ($3!="NA" && (($3+0)>1000000 || ($3+0)<-1000000)) fail("line " NR ": absolute value exceeds 1000000 a.u.")
 print
}
END { if (!failed && NR==0) fail("empty input: expected TSV header") }
#+end_src
```

RSH-05-003

# Keep absent results absent

Learning goal: distinguish an actual zero, an all-missing group and no groups at all. The zero fixture produces mean 0.00 with n_used=1. A missing-only group retains n_total=1 but has n_used=0 and mean NA. A header-only input yields the summary header and no rows; a zero-byte input is an error because the required header is missing.

The missing-only and no-group tables are also passed through native gnuplot here. One figure labels n=0 and omits the missing bar; the other displays No groups. No invented zero is plotted to make an empty dataset look complete. Exercise: add a valid zero to the missing-only group: n_total becomes 2, n_used becomes 1 and the mean becomes 0.00. The data meaning, rather than plotting convenience, determines that change.

Compare with RSH-05-002 · Turn malformed input into useful errors.

## Code

```
python3 ../tools/stations.py case zero
python3 ../tools/stations.py case empty
python3 ../tools/stations.py empty-figures
```

## Actual output

Captured run · exit 0

```
zero: exit 0
group   n_total n_used  n_missing       n_rejected      mean
X       1       1       0       0       0.00
empty: exit 2
input error: empty input: expected TSV header
missing: exit 0
group   n_total n_used  n_missing       n_rejected      mean
X       1       0       1       0       NA
missing: SVG and PDF rendered; n=0
header-only: exit 0
group   n_total n_used  n_missing       n_rejected      mean
header-only: SVG and PDF rendered; No groups
```

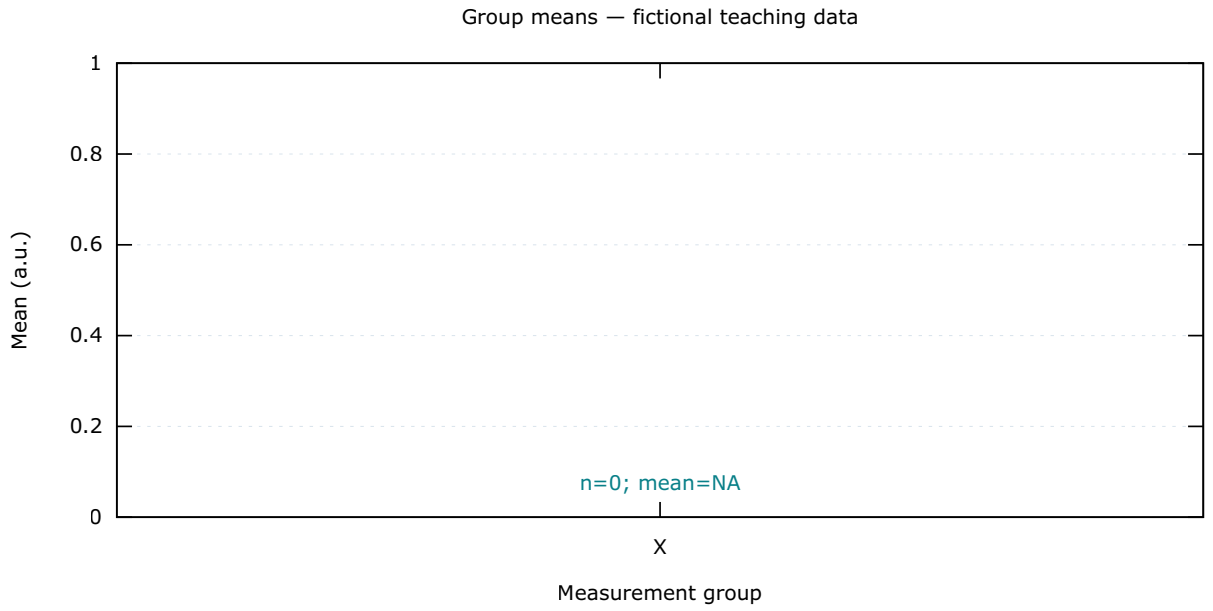
Download: Driver only · RSH-05-003.sh

---

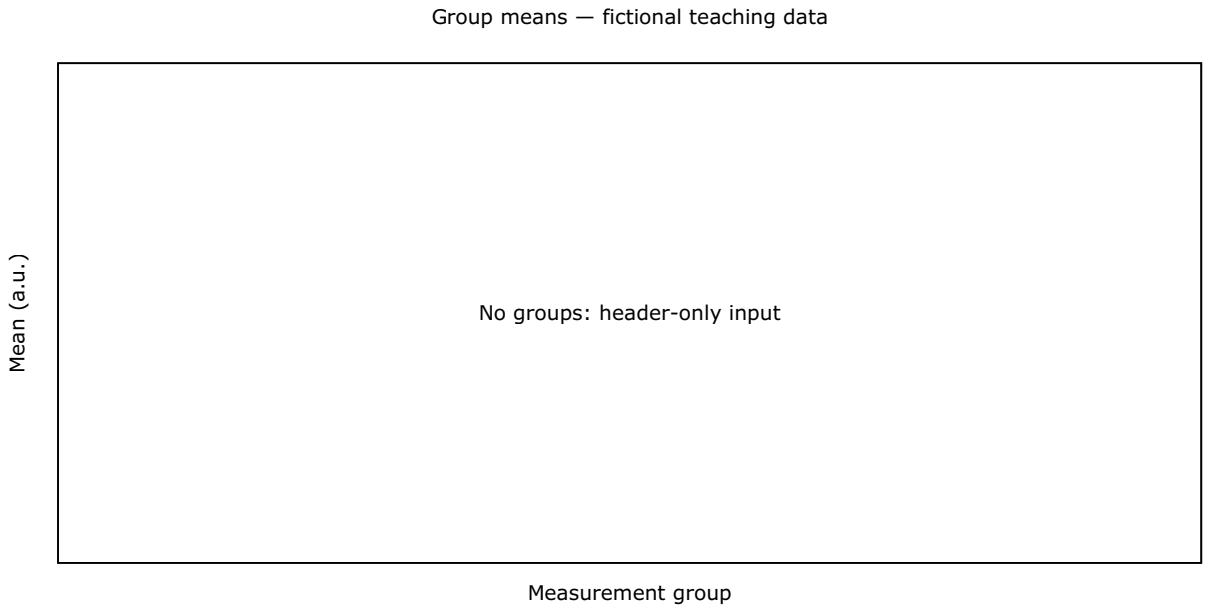
**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** plot.gp · **Result:** summary.tsv · figure.svg · figure.svg

### Read · plot-results

```
#+begin_src gnuplot :tangle no :noweb tangle
set datafile separator "\t"
set datafile missing "NA"
set terminal svg size 740,380 font "DejaVu Sans,12" enhanced
set output SVG
set title "Group means — fictional teaching data"
set xlabel "Measurement group"
set ylabel "Mean (a.u.)"
set boxwidth 0.55
set style fill solid 0.8 border rgb "#00549f"
set grid ytics lc rgb "#d8e3ec"
set key off
if (GROUPS>0) {
 stats SUMMARY using (column("n_used")>0 ? column("mean") : 0) nooutput
 low=(STATS_min<0 ? STATS_min*1.25 : 0)
 high=(STATS_max>0 ? STATS_max*1.3 : 1)
 if (low==high) { low=low-1; high=high+1 }
 set yrange [low:high]
 set xrange [-0.6:GROUPS-0.4]
 plot SUMMARY using 0:(column("mean")):xticlabels(1) with boxes lc rgb "#00549f", \
      SUMMARY using 0:(column("n_used")>0 ? column("mean") : 0):(column("n_used")>0 ? sprintf("n=%d",column("n_used")) : "n=0; mean=NA") with labels offset 0,1.1 tc rgb "#087f88"
} else {
 set xrange [0:1]; set yrange [0:1]; unset xtics; unset ytics
 set label "No groups: header-only input" at graph 0.5,0.5 center
 plot NaN
}
set terminal pdfcairo size 16cm,8.5cm font "DejaVu Sans,11"
set output PDF
replot
unset output
#+end_src
```




Result · build/experiment/missing/figure.svg · fictional teaching data




Result · build/experiment/header-only/figure.svg · fictional teaching data

RSH-05-004

# Break a handover to test the invariant

Learning goal: check an agreement between stages, not just isolated functions. This private experiment removes the last usable row after selection, then calls the generated summarizer with the unchanged classified census. B would have three classified used records but only two handed-over values. The summarizer detects that the categories no longer conserve the total and exits with status 3.

This is a deliberately corrupted internal handover, not an accepted missing-value convention. The normal Make graph regenerates usable.tsv from its producer; consumers should not edit generated results. Exercise: explain why silently computing a mean from the shortened file would conceal data loss. The observed handover error is concrete evidence for both Robust and Evaluate explicitly. In the canonical project, the untouched usable file still yields all required reference results.

Compare with RSH-05-003 · Keep absent results absent.

### Code

```
python3 ../tools/stations.py conservation
```

### Actual output

Captured run · exit 0

```
Dropped one usable row: exit 3
handover error: count conservation failed for B
```

Download: Driver only · RSH-05-004.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** summarize.awk · **Result:** usable.tsv

### Read · report-exclusions

```
#+begin_src awk :tangle no :noweb tangle
if (total[g]!=used[g]+missing[g]+rejected[g]) {
 print "handover error: count conservation failed for " g > "/dev/stderr"; exit 3
}
printf "%s\t%d\t%d\t%d\t%d\t%s\n",g,total[g],used[g],missing[g],rejected[g],(used[g] ? sprintf("%.2f",sum[g]/used[g]) : "NA")
#+end_src
```

## 06 - Connect analysis and explanation

Use the same summary in a table, a native plot and a readable report. SHAPE — Harmonize handovers is evidenced by named TSV columns; Automate actions by the grouped figure and report targets. ROOT — Open is evidenced by portable program downloads and a report with source/evidence attachments.

Chapter focus: **Harmonize handovers and Automate actions**.

RSH-06-001

# Use one summary table in the explanation

Learning goal: connect computed evidence to a readable claim. The six-column TSV is the sole numeric input for the report table and figure. The template explains the eligibility policy and identifies the data as fictional teaching measurements. A reader can audit the denominator and both exclusion counts without reverse-engineering the plot.

Means describe the supplied measurements; no population inference or uncertainty estimate is claimed for this tiny invented dataset. The table retains NA where a mean is unavailable, and it sorts group names explicitly rather than trusting AWK's associative-array iteration order. Exercise: add a third group in a private copy and predict that its position follows ASCII group order. A report that hard-codes A and B would fail that handover; this template reads every summary row and its actual header.

### Code

```
cat build/results/summary.tsv
```

### Actual output

Captured run · exit 0

```
group   n_total n_used  n_missing       n_rejected      mean
A       5       3       1       1       10.00
B       5       3       1       1       15.00
```

Download: Driver only · RSH-06-001.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** report-template.org · **Result:** summary.tsv

## Read · report-template

```
#+begin_src org :tangle no :noweb tangle
,#+TITLE: Fictional measurement study
,#+SUBTITLE: Explain the program. Rebuild the result.
,#+LANGUAGE: en
,#+OPTIONS: toc:nil num:nil author:nil date:nil creator:nil html-postamble:nil ^:nil
,#+HTML_HEAD: <link rel="stylesheet" href="report.css">

,* Question and decision
What is the mean of each group's usable measurements, in arbitrary units?
These are fictional teaching data, not observations supporting a scientific claim.
A recorded zero is usable. NA is missing. Invalid status takes precedence over
missingness. No usable observations means NA, never an invented zero mean.

,* Results
{{TABLE}}

,#+ATTR_HTML: :alt Group means from fictional data with used sample counts :width 740
[[file:results/figure.svg]]

The n labels count used measurements. A group without usable values has no bar
and is explicitly labeled n=0; mean=NA. Exclusion categories do not overlap.

,* Evidence and handover
:PROPERTIES:
:CUSTOM_ID: evidence
:END:
ROOT the tools. SHAPE the workflow.
The reference and edge cases passed: {{CHECKS}}. Data conservation is checked
for every group: n_total = n_used + n_missing + n_rejected.
The source revision is {{REVISION}}; its recorded dirty status is {{DIRTY}}.

- [[file:research.org][Read the complete literate source]]
- [[file:data/measurements.tsv][Read the original TSV input]]
- [[file:results/summary.tsv][Download the summary TSV]]
- [[file:provenance.json][Inspect the machine-readable provenance]]
- [[file:programs/validate.awk][Reuse input validation]]
- [[file:programs/summarize.awk][Reuse the group statistic program]]

The manifest hashes inputs, source, generated programs and checked scientific
results. It does not hash itself or this report: manifest precedes report.
Identical scientific tables are tested in this recorded environment. PDF bytes
can differ with fonts, tool versions and timestamps; no cross-platform byte
identity is claimed. Source, raw TSV and manifest are attached to the PDF.
#+end_src
```

RSH-06-002

# Plot group means with visible denominators

Learning goal: make a plot communicate the inclusion decision. Native gnuplot reads columns by their header names and emits both SVG and vector PDF in one action. Each bar carries n_used; A and B both have n=3. The axes name the group and mean in a.u., and the caption labels the measurements as fictional teaching data.

Zero means remain legitimate plotted values. Missing means do not become zero-height observations; the missing-only and no-group cases were rendered in the preceding chapter. No uncertainty bars are invented. Exercise: remove the n labels temporarily and explain what important difference between five collected and three used measurements becomes invisible. The assertions below inspect the real SVG text and native PDF; the Result panel shows the actual figure produced by that run.

Compare with RSH-06-001 · Use one summary table in the explanation.

## Code

```
python3 ../tools/stations.py figure
```

## Actual output

Captured run · exit 0

```
Native SVG: n=3 labels, A/B groups, fictional teaching data
Native figure PDF: one vector page
```

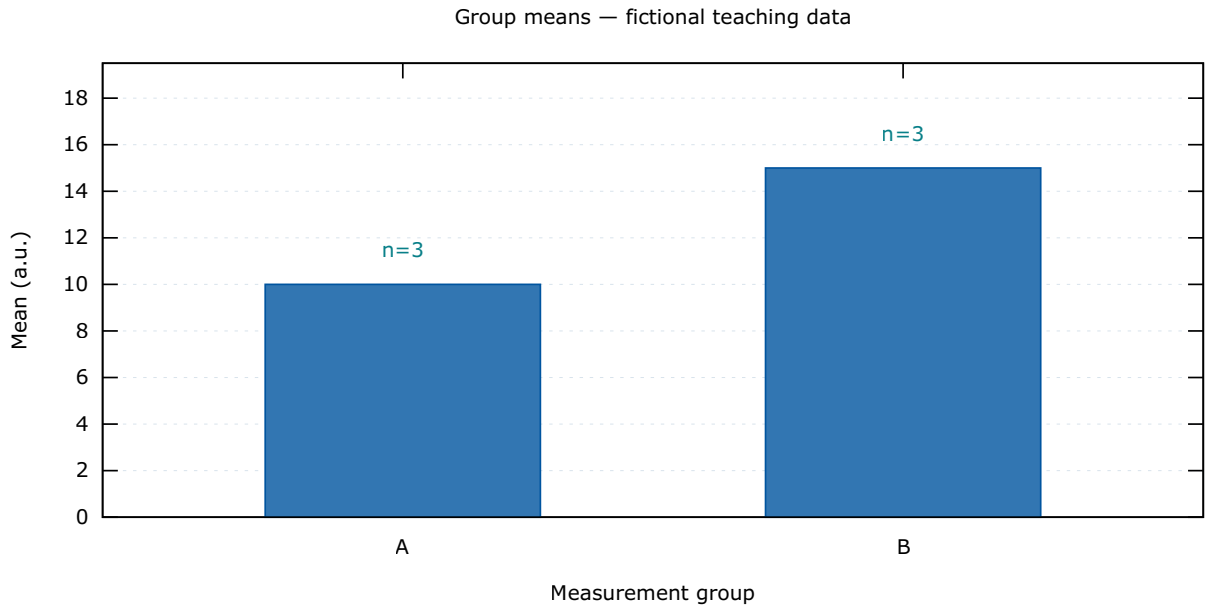
Download: Driver only · RSH-06-002.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** plot.gp · **Result:** figure.svg · figure.pdf

### Read · plot-results

```
#+begin_src gnuplot :tangle no :noweb tangle
set datafile separator "\t"
set datafile missing "NA"
set terminal svg size 740,380 font "DejaVu Sans,12" enhanced
set output SVG
set title "Group means — fictional teaching data"
set xlabel "Measurement group"
set ylabel "Mean (a.u.)"
set boxwidth 0.55
set style fill solid 0.8 border rgb "#00549f"
set grid ytics lc rgb "#d8e3ec"
set key off
if (GROUPS>0) {
 stats SUMMARY using (column("n_used")>0 ? column("mean") : 0) nooutput
 low=(STATS_min<0 ? STATS_min*1.25 : 0)
 high=(STATS_max>0 ? STATS_max*1.3 : 1)
 if (low==high) { low=low-1; high=high+1 }
 set yrange [low:high]
 set xrange [-0.6:GROUPS-0.4]
 plot SUMMARY using 0:(column("mean")):xticlabels(1) with boxes lc rgb "#00549f", \
      SUMMARY using 0:(column("n_used")>0 ? column("mean") : 0):(column("n_used")>0 ? sprintf("n=%d",column("n_used")) : "n=0; mean=NA") with labels offset 0,1.1 tc rgb "#087f88"
} else {
 set xrange [0:1]; set yrange [0:1]; unset xtics; unset ytics
 set label "No groups: header-only input" at graph 0.5,0.5 center
 plot NaN
}
set terminal pdfcairo size 16cm,8.5cm font "DejaVu Sans,11"
set output PDF
replot
unset output
#+end_src
```




Result · build/results/figure.svg · fictional teaching data

RSH-06-003

# Build a readable HTML and PDF report

Learning goal: hand over evidence in formats a colleague can read. The generated report combines the actual table, native figure, inclusion policy, test summary, source revision and relative links to source, input, programs and manifest. Batch Org exports the working report; the publisher's existing WeasyPrint/PyMuPDF dependencies create the PDF and embed the source, raw data and manifest.

The Org template protects its own heading and keyword lines with a leading comma; extraction removes that protection. The checks below open the actual HTML and PDF, look for both reference means and the fictional-data label, and verify three embedded evidence files. The full standalone project download includes sources and prerequisites; the reference-output archive includes the produced reports. Exercise: inspect the HTML table and PDF figure together and confirm that both disclose n=3. Styling changes belong in the literate report-style fragment so that the source remains authoritative.

Compare with RSH-06-002 · Plot group means with visible denominators.

## Code

```
python3 ../tools/stations.py report
```

## Actual output

Captured run · exit 0

```
Report HTML: real table, figure and relative downloads
Report PDF: 2 page(s), 3 embedded source/evidence files
Maintained Org source: unchanged
```

Download: Driver only · RSH-06-003.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** report-template.org · report.css · **Result:** index.html · report.pdf

## Read · report-template

```
#+begin_src org :tangle no :noweb tangle
,#+TITLE: Fictional measurement study
,#+SUBTITLE: Explain the program. Rebuild the result.
,#+LANGUAGE: en
,#+OPTIONS: toc:nil num:nil author:nil date:nil creator:nil html-postamble:nil ^:nil
,#+HTML_HEAD: <link rel="stylesheet" href="report.css">

,* Question and decision
What is the mean of each group's usable measurements, in arbitrary units?
These are fictional teaching data, not observations supporting a scientific claim.
A recorded zero is usable. NA is missing. Invalid status takes precedence over
missingness. No usable observations means NA, never an invented zero mean.

,* Results
{{TABLE}}

,#+ATTR_HTML: :alt Group means from fictional data with used sample counts :width 740
[[file:results/figure.svg]]

The n labels count used measurements. A group without usable values has no bar
and is explicitly labeled n=0; mean=NA. Exclusion categories do not overlap.

,* Evidence and handover
:PROPERTIES:
:CUSTOM_ID: evidence
:END:
ROOT the tools. SHAPE the workflow.
The reference and edge cases passed: {{CHECKS}}. Data conservation is checked
for every group: n_total = n_used + n_missing + n_rejected.
The source revision is {{REVISION}}; its recorded dirty status is {{DIRTY}}.

- [[file:research.org][Read the complete literate source]]
- [[file:data/measurements.tsv][Read the original TSV input]]
- [[file:results/summary.tsv][Download the summary TSV]]
- [[file:provenance.json][Inspect the machine-readable provenance]]
- [[file:programs/validate.awk][Reuse input validation]]
- [[file:programs/summarize.awk][Reuse the group statistic program]]

The manifest hashes inputs, source, generated programs and checked scientific
results. It does not hash itself or this report: manifest precedes report.
Identical scientific tables are tested in this recorded environment. PDF bytes
can differ with fonts, tool versions and timestamps; no cross-platform byte
identity is claimed. Source, raw TSV and manifest are attached to the PDF.
#+end_src
```

RSH-06-004

# Export without executing the explanation

Learning goal: keep publication separate from calculation. This station exports the full research.org itself, in addition to the compact working report. Because :noweb tangle expands only on extraction and Babel evaluation is disabled for export, the published source still displays <<validate-input>> as a reference. It does not rerun AWK or insert results into the maintained document.

The actual export is checked for that textual reference and the source checksum is compared before and after. This distinguishes weave/export from both tangle and execute. Exercise: reorder a prose section and export again; the presentation changes while the same references still tangle to the same program. Org's evaluated noweb-call form is not used here. The source download and Tangled files let a reader inspect both representations without installing an editor configuration.

Compare with RSH-06-003 · Build a readable HTML and PDF report.

### Code

```
python3 ../tools/stations.py weave
```

### Actual output

Captured run · exit 0

```
Weave/export: textual <<validate-input>> remains visible
Babel evaluation: disabled; maintained source: unchanged
```

Download: Driver only · RSH-06-004.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** validate.awk · **Result:** literate.html

### Read · validate-program

```
#+begin_src awk :tangle programs/validate.awk :mkdirp yes :noweb tangle
<<validation-setup>>
<<validate-input>>
#+end_src
```

## 07 - Carry provenance forward

Connect the actual inputs, programs, commands, environment and results. SHAPE — Propagate provenance is evidenced by a machine-readable manifest and a tampering experiment. ROOT — Time-tested means recording the implementations actually used, not assuming all installations behave identically.

Chapter focus: **Propagate provenance**.

RSH-07-001

# Join source, input, program and result identities

Learning goal: follow a result through a machine-readable evidence chain. provenance.json hashes the raw TSV, independent expectations and fixtures; the maintained literary/bootstrap sources; all ten generated programs/templates; and the scientific results, checks and native figures. Every published path is relative. The assertions below recompute every listed SHA-256 value.

The manifest excludes itself and the downstream HTML/PDF reports, avoiding a hash cycle. A later report receipt records those report hashes separately, and the PDF embeds the manifest it describes. No run timestamp forces a new build. Exercise: locate the source hash, the summarize.awk hash and the summary.tsv hash and explain their different roles. Propagating provenance means connecting the actual transformation, not merely stamping a final document with a date.

### Code

```
python3 ../tools/stations.py provenance
```

### Actual output

Captured run · exit 0

```
inputs: 19 checked SHA-256 entries
sources: 3 checked SHA-256 entries
programs: 10 checked SHA-256 entries
results: 7 checked SHA-256 entries
Manifest and downstream reports excluded: no hash cycle
```

Download: Driver only · RSH-07-001.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** pipeline.py · **Result:** provenance.json

## Read · propagate-provenance

```
#+begin_src python :tangle no :noweb tangle
def provenance():
 tangle=json.loads(Path('build/tangle.json').read_text());source=json.loads(Path('build/source.json').read_text())
 inputs=['data/measurements.tsv','tests/expected-summary.tsv','tests/cases.json']+sorted(str(p) for p in Path('tests/fixtures').glob('*.tsv'))
 programs=['build/'+p for p in sorted(tangle['files'])]
 results=sorted(str(p) for p in Path('build/results').glob('*'))+['build/checks.json']
 receipts={p.stem:json.loads(p.read_text()) for p in sorted(Path('build/receipts').glob('*.json')) if p.stem in {'validate','classify','select','summarize','check','plot'}}
 record={'schema':1,'question':'Group mean of usable fictional measurements','parameters':json.loads(Path('build/programs/contract.json').read_text()),
  'inputs':{p:sha(p) for p in inputs},'sources':{p:sha(p) for p in ['research.org','build.py','bootstrap.el']},
  'programs':{p:sha(p) for p in programs},'results':{p:sha(p) for p in results},
  'versions':json.loads(Path('build/environment.json').read_text()),'git':source['git'],
  'execution':receipts,'tangle':tangle['command'],'invocation':json.loads(Path('build/invocation.json').read_text()),
  'scope':'Same scientific tables for the specified input and recorded environment. Manifest excludes itself, receipts generated after it, and the downstream reports.'}
 js('build/provenance.json',record);receipt('provenance',[],['build/provenance.json'])
#+end_src
```

RSH-07-002

# Record the environment and real calls

Learning goal: distinguish documented prerequisites from the implementations actually used. The manifest records Python, Emacs/Org, AWK, GNU Make, gnuplot and PDF-library versions, plus the AWK executable's SHA-256 identity. It includes actual native argv from successful stages and fixture runs, the textual tangle invocation and the bootstrap/Make invocation with its parameters.

The displayed version strings are captured evidence from this build. They are not promises that every version or AWK implementation has been tested. ROOT — Time-tested means choosing established tools with conscious requirements: /dev/stderr support, GNU Make grouped targets, gnuplot svg/pdfcairo and Python's available PDF libraries. Exercise: inspect a different environment's manifest before comparing its result table. Reproduction here means the same scientific result for the recorded input and environment; portable PDF byte identity has not been claimed.

Compare with RSH-07-001 · Join source, input, program and result identities.

## Code

```
python3 ../tools/stations.py versions
```

## Actual output

Captured run · exit 0

```
PyMuPDF: 1.28.2
awk: awk version 20200816
awk_binary_sha256: 310e75b917c4d346cecb6ffab9ca88507870242347a6cf0110eef4aa73b6f809
emacs: 31.1 / Org 9.8.7
gnuplot: gnuplot 6.0 patchlevel 6
make: GNU Make 4.4.1
python: 3.14.10
weasyprint: 70.0
Locale: C
Recorded native calls: 42
```

Download: Driver only · RSH-07-002.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** pipeline.py · **Result:** environment.json · summarize.json

### Read · propagate-provenance

```
#+begin_src python :tangle no :noweb tangle
def provenance():
 tangle=json.loads(Path('build/tangle.json').read_text());source=json.loads(Path('build/source.json').read_text())
 inputs=['data/measurements.tsv','tests/expected-summary.tsv','tests/cases.json']+sorted(str(p) for p in Path('tests/fixtures').glob('*.tsv'))
 programs=['build/'+p for p in sorted(tangle['files'])]
 results=sorted(str(p) for p in Path('build/results').glob('*'))+['build/checks.json']
 receipts={p.stem:json.loads(p.read_text()) for p in sorted(Path('build/receipts').glob('*.json')) if p.stem in {'validate','classify','select','summarize','check','plot'}}
 record={'schema':1,'question':'Group mean of usable fictional measurements','parameters':json.loads(Path('build/programs/contract.json').read_text()),
  'inputs':{p:sha(p) for p in inputs},'sources':{p:sha(p) for p in ['research.org','build.py','bootstrap.el']},
  'programs':{p:sha(p) for p in programs},'results':{p:sha(p) for p in results},
  'versions':json.loads(Path('build/environment.json').read_text()),'git':source['git'],
  'execution':receipts,'tangle':tangle['command'],'invocation':json.loads(Path('build/invocation.json').read_text()),
  'scope':'Same scientific tables for the specified input and recorded environment. Manifest excludes itself, receipts generated after it, and the downstream reports.'}
 js('build/provenance.json',record);receipt('provenance',[],['build/provenance.json'])
#+end_src
```

RSH-07-003

# Detect tampering with a result

Learning goal: test whether the evidence chain can reveal an unauthorized change. The experiment replaces A's mean with 99.00 in the generated summary without changing its manifest. Verification must report a summary.tsv hash mismatch. It then restores the exact original bytes and checks the complete manifest and report receipt successfully.

A checksum mismatch identifies inconsistency, not its cause: it could be an edit, corruption or the wrong file. The proper response is to inspect the source and rebuild through the checks, not rewrite a hash to hide the change. Exercise: repeat with the raw TSV in a private copy and identify which role fails first. Verification is deliberately separate from regeneration so that evidence can be checked without overwriting the object being examined.

Compare with RSH-07-002 · Record the environment and real calls.

### Code

```
python3 ../tools/stations.py tamper
```

### Actual output

Captured run · exit 0

```
Tampered table: study: provenance mismatch: build/results/summary.tsv
verify: input, source, program, result and report hashes passed
```

Download: Driver only · RSH-07-003.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** pipeline.py · **Result:** provenance.json

### Read · verify-handover

```
#+begin_src python :tangle no :noweb tangle
def verify():
 record=json.loads(Path('build/provenance.json').read_text())
 for role in ['inputs','sources','programs','results']:
  for file,expected in record[role].items():
   if not Path(file).is_file() or sha(file)!=expected:raise RuntimeError('provenance mismatch: '+file)
 if 'build/provenance.json' in record['results']:raise RuntimeError('manifest self-hash is forbidden')
 for file,expected in json.loads(Path('build/receipts/report.json').read_text())['outputs'].items():
  if sha(file)!=expected:raise RuntimeError('report receipt mismatch: '+file)
 print('verify: input, source, program, result and report hashes passed')
#+end_src
```

RSH-07-004

# Build the handover outside Git

Learning goal: prove that the example is self-contained. The experiment copies only maintained sources, data, bootstrap, requirements and fixtures into a temporary folder outside the checkout, then builds and verifies the entire project. No previous build output or parent repository is available.

The manifest records commit=null and dirty=null when Git metadata is unavailable. Inside a checkout it records the actual revision and working-tree state, without inventing a clean revision. Neither case introduces an absolute machine-specific path into the manifest. Exercise: extract the project ZIP into a new folder, activate an environment with the documented prerequisites, and use python3 build.py all --jobs 4. The absence of Git is a supported handover condition; the absence of a required executable should instead produce a concrete prerequisite error.

Compare with RSH-07-003 · Detect tampering with a result.

### Code

```
python3 ../tools/stations.py outside-git
```

### Actual output

Captured run · exit 0

```
build: all; actions: validate,classify,select,summarize,check,plot,provenance,report
Outside Git: commit=null, dirty=null
verify: input, source, program, result and report hashes passed
```

Download: Driver only · RSH-07-004.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** console/source above · **Result:** console/source above

### Read · propagate-provenance

```
#+begin_src python :tangle no :noweb tangle
def provenance():
 tangle=json.loads(Path('build/tangle.json').read_text());source=json.loads(Path('build/source.json').read_text())
 inputs=['data/measurements.tsv','tests/expected-summary.tsv','tests/cases.json']+sorted(str(p) for p in Path('tests/fixtures').glob('*.tsv'))
 programs=['build/'+p for p in sorted(tangle['files'])]
 results=sorted(str(p) for p in Path('build/results').glob('*'))+['build/checks.json']
 receipts={p.stem:json.loads(p.read_text()) for p in sorted(Path('build/receipts').glob('*.json')) if p.stem in {'validate','classify','select','summarize','check','plot'}}
 record={'schema':1,'question':'Group mean of usable fictional measurements','parameters':json.loads(Path('build/programs/contract.json').read_text()),
  'inputs':{p:sha(p) for p in inputs},'sources':{p:sha(p) for p in ['research.org','build.py','bootstrap.el']},
  'programs':{p:sha(p) for p in programs},'results':{p:sha(p) for p in results},
  'versions':json.loads(Path('build/environment.json').read_text()),'git':source['git'],
  'execution':receipts,'tangle':tangle['command'],'invocation':json.loads(Path('build/invocation.json').read_text()),
  'scope':'Same scientific tables for the specified input and recorded environment. Manifest excludes itself, receipts generated after it, and the downstream reports.'}
 js('build/provenance.json',record);receipt('provenance',[],['build/provenance.json'])
#+end_src
```

## 08 - Change, rebuild and hand over

Treat a changed requirement as a change to the explanation, interfaces, program, expectations and handover together. ROOT — Ongoing is evidenced by three isolated edits and regenerated results. Structure stages, Harmonize handovers, Automate actions, Propagate provenance and Evaluate explicitly remain design perspectives on this one dependency graph.

Chapter focus: **All five SHAPE perspectives and Ongoing**.

RSH-08-001

# Correct one measurement and rebuild

Learning goal: distinguish an input correction from a changed algorithm. In a private copy, A01 changes from 10 to 13. Reviewing that correction changes A's independent expected mean from 10.00 to 11.00; all counts and B's mean remain unchanged. The experiment rebuilds the complete graph and compares before/after hashes for table, SVG, manifest, HTML and PDF.

All five outputs must change, and the immediately repeated build must run no stage. The changed expectation is justified by (13+20+0)/3=11, not copied from the program to silence a failure. Exercise: change only the raw value first and observe check stopping publication before updating the reviewed expectation. This is Ongoing supported by all SHAPE perspectives: the existing stages, handovers, automation, evidence and explicit evaluation carry a correction through the project.

### Code

```
python3 ../tools/stations.py mutate data
```

### Actual output

Captured run · exit 0

```
build: all; actions: validate,classify,select,summarize,check,plot,provenance,report
group   n_total n_used  n_missing       n_rejected      mean
A       5       3       1       1       11.00
B       5       3       1       1       15.00
Updated: table, SVG, manifest, HTML and PDF
build: all; actions: none
```

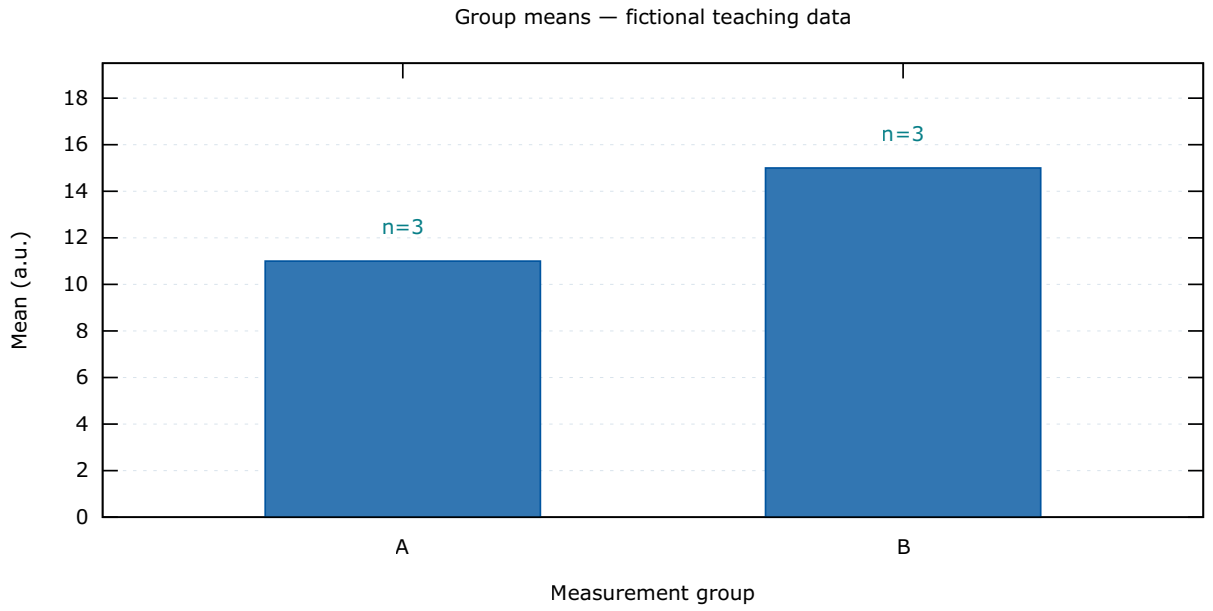
Download: Driver only · RSH-08-001.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** workflow.mk · **Result:** summary.tsv · figure.svg · provenance.json

## Read · workflow

```
#+begin_src makefile :tangle no :noweb tangle
.DEFAULT_GOAL := all
.RECIPEPREFIX := >
.DELETE_ON_ERROR:
.SUFFIXES:
PYTHON ?= python3
AWK ?= awk
RUN = $(PYTHON) build/tools/pipeline.py
.PHONY: all results check figure provenance report
all: report
results: build/results/summary.tsv
check: build/checks.json
figure: build/results/figure.svg build/results/figure.pdf
provenance: build/provenance.json
report: build/site/index.html build/site/report.pdf
build/results/validated.tsv: data/measurements.tsv build/raw.sha256 build/environment.json build/programs/validate.awk build/tools/pipeline.py
>@$(RUN) stage validate $(AWK)
build/results/classified.tsv: build/results/validated.tsv build/programs/classify.awk build/tools/pipeline.py
>@$(RUN) stage classify $(AWK)
build/results/usable.tsv: build/results/classified.tsv build/programs/select.awk build/tools/pipeline.py
>@$(RUN) stage select $(AWK)
build/results/summary.tsv: build/results/classified.tsv build/results/usable.tsv build/programs/summarize.awk build/tools/pipeline.py
>@$(RUN) stage summarize $(AWK)
build/checks.json: build/results/summary.tsv tests/expected-summary.tsv tests/cases.json $(wildcard tests/fixtures/*.tsv) $(wildcard build/programs/*.awk) build/tools/pipeline.py
>@$(RUN) check $(AWK)
build/results/figure.svg build/results/figure.pdf &: build/results/summary.tsv build/checks.json build/programs/plot.gp build/tools/pipeline.py
>@$(RUN) plot
build/provenance.json: build/results/figure.svg build/results/figure.pdf build/checks.json build/source.json build/environment.json build/invocation.json build/tangle.json $(wildcard
build/programs/*) build/workflow.mk build/tools/pipeline.py
>@$(RUN) provenance
build/site/index.html build/site/report.pdf &: build/provenance.json build/programs/report-template.org build/programs/report.css build/tools/pipeline.py bootstrap.el
>@$(RUN) report
#+end_src
```




Result · build/results/figure.svg · fictional teaching data

RSH-08-002

# Add a new group through the same interfaces

Learning goal: test an extension without hard-coded group lists. The private input gains C01, group C, value 8 and valid status. Its independently authored expected row is C, 1 total, 1 used, 0 missing, 0 rejected, mean 8.00. The complete build must add C to the table, figure, provenance and report.

The program discovers groups from the census and sorts them explicitly; the report uses the actual table header and rows; gnuplot reads column names and the real group count. No program edit is needed for this requirement. Exercise: instead add a group containing only NA and invalid rows, then predict an NA mean and n=0 label. That extension tests the distinction between a discovered group and a group with an estimable mean, keeping missing results visible.

Compare with RSH-08-001 · Correct one measurement and rebuild.

### Code

```
python3 ../tools/stations.py mutate group
```

### Actual output

Captured run · exit 0

```
build: all; actions: validate,classify,select,summarize,check,plot,provenance,report
group   n_total n_used  n_missing       n_rejected      mean
A       5       3       1       1       10.00
B       5       3       1       1       15.00
C       1       1       0       0       8.00
Updated: table, SVG, manifest, HTML and PDF
build: all; actions: none
```

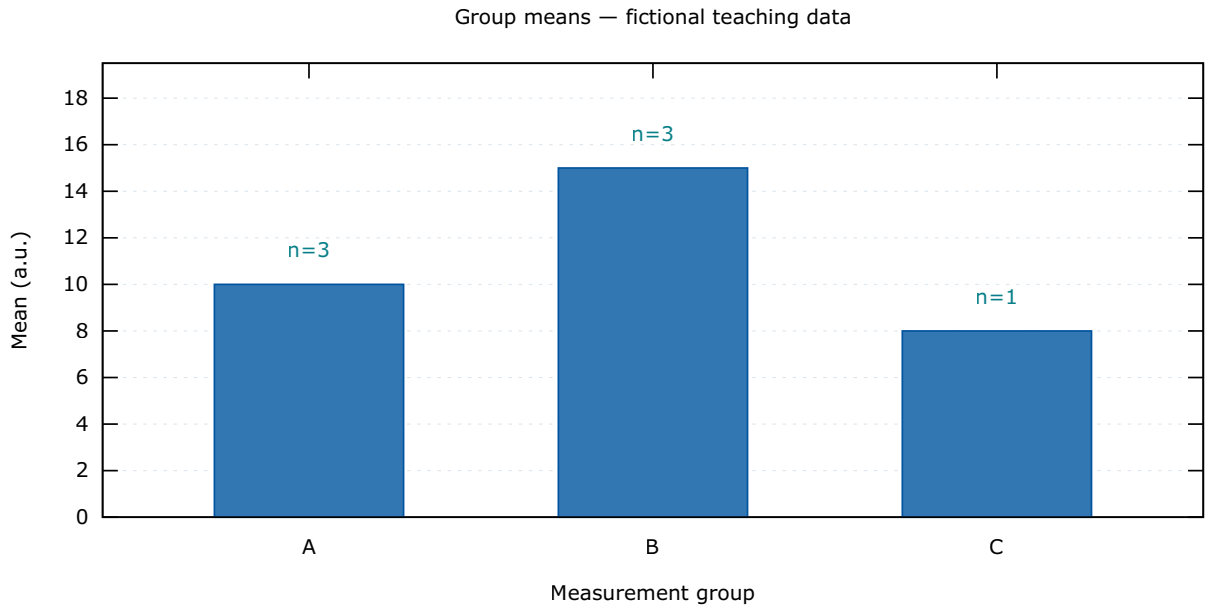
Download: Driver only · RSH-08-002.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** summarize.awk · plot.gp · **Result:** summary.tsv · figure.svg

### Read · ordered-groups

```
#+begin_src awk :tangle no :noweb tangle
END {
 <<summary-header>>
 for (i=1;i<=ng;i++) for (j=i+1;j<=ng;j++) if (groups[j]<groups[i]) { t=groups[i];groups[i]=groups[j];groups[j]=t }
 for (i=1;i<=ng;i++) {
  g=groups[i]
  <<report-exclusions>>
 }
}
#+end_src
```




Result · build/results/figure.svg · fictional teaching data

RSH-08-003

# Extend the requirement with a maximum

Learning goal: change the explanation, implementation and expectation together. The experiment adds a maximum usable value in a.u. while retaining every required reference column. It edits the literary question, the summary accumulation fragment, output header and report-exclusions fragment, then independently adds expected maxima A=20.00 and B=25.00. Retangling produces the new program and the full build checks the new seven-column table.

The maximum is initialized from the first used value, so negative-only groups would not acquire a false zero maximum. With no used values it is NA, consistent with the mean contract. Header-based plotting still uses mean and n_used, while the report includes the new column automatically. Exercise: add a negative-only fixture before adopting this extension. The canonical final project remains the requested six-column reference; this isolated source variant is available in this station's Source/Tangled views.

Compare with RSH-08-002 · Add a new group through the same interfaces.

### Code

```
python3 ../tools/stations.py max
```

### Actual output

Captured run · exit 0

```
build: all; actions: summarize,check,plot,provenance,report
group   n_total n_used  n_missing       n_rejected      mean    max
A       5       3       1       1       10.00   20.00
B       5       3       1       1       15.00   25.00
Changed together: explanation, three named fragments, authored expectations
Additional checks: negative maximum=-2.00; absent maximum=NA; zero maximum=0.00
```

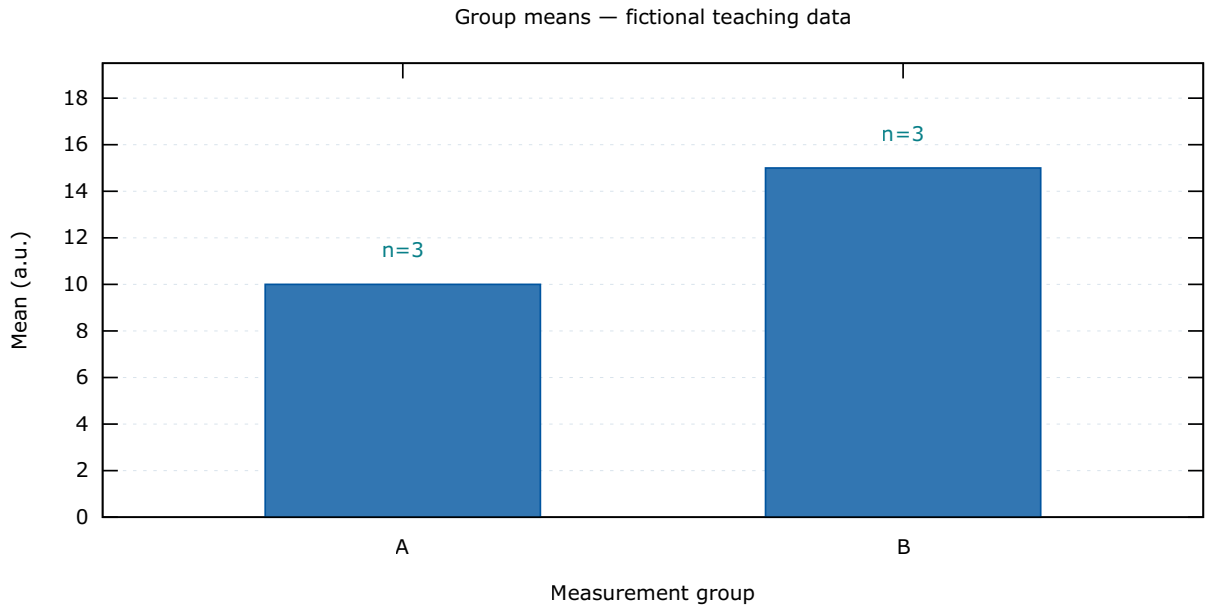
Download: Driver only · RSH-08-003.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** summarize.awk · **Result:** summary.tsv · figure.svg

### Read · report-exclusions

```
#+begin_src awk :tangle no :noweb tangle
if (total[g]!=used[g]+missing[g]+rejected[g]) {
 print "handover error: count conservation failed for " g > "/dev/stderr"; exit 3
}
printf "%s\t%d\t%d\t%d\t%d\t%s\t%s\n",g,total[g],used[g],missing[g],rejected[g],(used[g] ? sprintf("%.2f",sum[g]/used[g]) : "NA"),(used[g] ? sprintf("%.2f",maximum[g]) : "NA")
#+end_src
```




Result · build/results/figure.svg · fictional teaching data

RSH-08-004

# Hand over a clean and incremental build

Learning goal: verify the delivery rather than describing an intended workflow. Starting without generated files, this experiment runs the complete build with four Make jobs, repeats it unchanged and checks the real action log. Validation, classification, selection, summary, checks, plot, provenance and report must each run once; the second build must report actions: none.

It then launches two bootstrap processes concurrently. The advisory lock serializes their access, and GNU Make grouped targets give each SVG/PDF and HTML/PDF pair one writer. Both see current results and add no actions. Exercise: remove a generated program and rebuild; tangling must restore it before Make evaluates its prerequisites. The handover includes a clean source-only ZIP, a reference-output archive and the precise prerequisites. This closes the question with an executable, explained and explicitly checked result.

Compare with RSH-08-003 · Extend the requirement with a maximum.

## Code

```
python3 ../tools/stations.py incremental
```

## Actual output

Captured run · exit 0

```
build: all; actions: validate,classify,select,summarize,check,plot,provenance,report
build: all; actions: none
Parallel Make: plot=1, report=1; unchanged second build: no stage ran
Two concurrent bootstraps: serialized; no duplicate actions
```

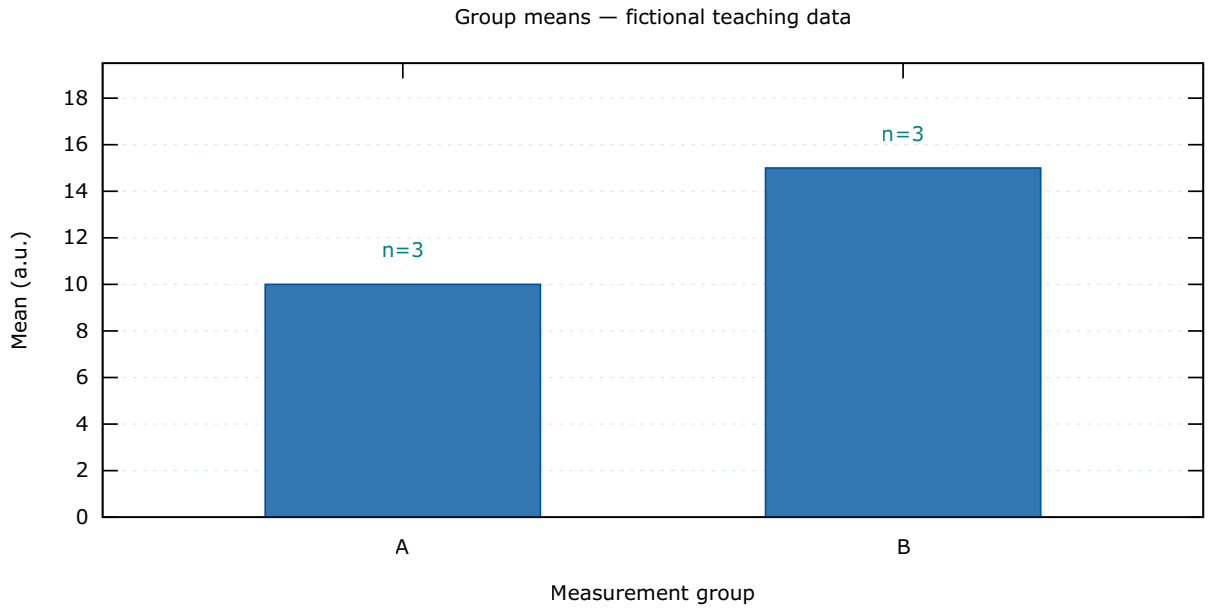
Download: Driver only · RSH-08-004.sh

---

**Source:** Read Org · Download Org · Experiment ZIP · **Tangled:** workflow.mk · **Result:** events.log.txt · summary.tsv · figure.svg · provenance.json

## Read · workflow

```
#+begin_src makefile :tangle no :noweb tangle
.DEFAULT_GOAL := all
.RECIPEPREFIX := >
.DELETE_ON_ERROR:
.SUFFIXES:
PYTHON ?= python3
AWK ?= awk
RUN = $(PYTHON) build/tools/pipeline.py
.PHONY: all results check figure provenance report
all: report
results: build/results/summary.tsv
check: build/checks.json
figure: build/results/figure.svg build/results/figure.pdf
provenance: build/provenance.json
report: build/site/index.html build/site/report.pdf
build/results/validated.tsv: data/measurements.tsv build/raw.sha256 build/environment.json build/programs/validate.awk build/tools/pipeline.py
>@$(RUN) stage validate $(AWK)
build/results/classified.tsv: build/results/validated.tsv build/programs/classify.awk build/tools/pipeline.py
>@$(RUN) stage classify $(AWK)
build/results/usable.tsv: build/results/classified.tsv build/programs/select.awk build/tools/pipeline.py
>@$(RUN) stage select $(AWK)
build/results/summary.tsv: build/results/classified.tsv build/results/usable.tsv build/programs/summarize.awk build/tools/pipeline.py
>@$(RUN) stage summarize $(AWK)
build/checks.json: build/results/summary.tsv tests/expected-summary.tsv tests/cases.json $(wildcard tests/fixtures/*.tsv) $(wildcard build/programs/*.awk) build/tools/pipeline.py
>@$(RUN) check $(AWK)
build/results/figure.svg build/results/figure.pdf &: build/results/summary.tsv build/checks.json build/programs/plot.gp build/tools/pipeline.py
>@$(RUN) plot
build/provenance.json: build/results/figure.svg build/results/figure.pdf build/checks.json build/source.json build/environment.json build/invocation.json build/tangle.json $(wildcard
build/programs/*) build/workflow.mk build/tools/pipeline.py
>@$(RUN) provenance
build/site/index.html build/site/report.pdf &: build/provenance.json build/programs/report-template.org build/programs/report.css build/tools/pipeline.py bootstrap.el
>@$(RUN) report
#+end_src
```




Result · build/results/figure.svg · fictional teaching data

# Example and command index


- RSH-01-001 · Define the denominator
- RSH-01-002 · Keep the raw input independent
- RSH-01-003 · Work out the reference by hand
- RSH-01-004 · Observe the complete data path
- RSH-02-001 · Start from a small working calculation
- RSH-02-002 · Give the summary a readable skeleton
- RSH-02-003 · Reorder the explanation without changing the program
- RSH-02-004 · Check expansion and single ownership
- RSH-03-001 · Publish the data contract
- RSH-03-002 · Replace missingness with an observed zero
- RSH-03-003 · Make exclusion priority a scientific decision
- RSH-03-004 · Hand over values without losing the census
- RSH-04-001 · Bootstrap before the generated Makefile exists
- RSH-04-002 · Run a tangled program without Org
- RSH-04-003 · Execute explicit dependencies
- RSH-04-004 · Separate tangle, execute and weave
- RSH-05-001 · Check the independently authored reference
- RSH-05-002 · Turn malformed input into useful errors
- RSH-05-003 · Keep absent results absent
- RSH-05-004 · Break a handover to test the invariant
- RSH-06-001 · Use one summary table in the explanation
- RSH-06-002 · Plot group means with visible denominators
- RSH-06-003 · Build a readable HTML and PDF report
- RSH-06-004 · Export without executing the explanation
- RSH-07-001 · Join source, input, program and result identities
- RSH-07-002 · Record the environment and real calls
- RSH-07-003 · Detect tampering with a result
- RSH-07-004 · Build the handover outside Git
- RSH-08-001 · Correct one measurement and rebuild
- RSH-08-002 · Add a new group through the same interfaces
- RSH-08-003 · Extend the requirement with a maximum
- RSH-08-004 · Hand over a clean and incremental build



Content: CC0 1.0. Structure and publishing framework: CC BY 4.0 by Lukas C. Bossert. Repository source · Publishing framework source · License scope and third-party notices.