

GUIDE · Differentiable Docs

# User guide

## Writing and building controlled documents

Everything needed to write documents with Differentiable Docs: getting a toolchain, writing the front matter, the shapes a document can take, the writing conventions, building and publishing, and what to do when the build says no.

|                    |                        |
|--------------------|------------------------|
| <b>Document ID</b> | <b>DOC-USER-GD-011</b> |
| Version            | 1.9 <b>APPROVED</b>    |
| Owner              | Documentation owner    |
| Date               | 2026-09-29             |

Public: may be shared without restriction.

Built from commit 4f80011744 on 2026-10-20.

# Document control

|                    |                                 |
|--------------------|---------------------------------|
| <b>Document ID</b> | <b>DOC-USER-GD-011</b>          |
| Type               | Guide                           |
| Product            | Differentiable Docs             |
| Classification     | <b>Public</b>                   |
| Owner              | Documentation owner             |
| Authors            | Differentiable Solutions S.r.l. |
| Reviewers          | —                               |
| Approved by        | Documentation owner             |

## Revision history

| Version | Date       | Author                          | Change                                                                                                                                                                                                                                      |
|---------|------------|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.0     | 2026-09-16 | Differentiable Solutions S.r.l. | First release of the user documentation.                                                                                                                                                                                                    |
| 1.1     | 2026-09-17 | Differentiable Solutions S.r.l. | Corrected what a build produces by default; described the refusal of unfilled skeleton placeholders, inline code breaking in tables, and the build's acceptance of a document with no revision history; removed forced chapter page breaks. |
| 1.2     | 2026-09-17 | Differentiable Solutions S.r.l. | Added how to adopt the build in a repository that already has a docs or Docs folder.                                                                                                                                                        |
| 1.3     | 2026-09-17 | Differentiable Solutions S.r.l. | Added diagrams written as Mermaid blocks.                                                                                                                                                                                                   |
| 1.4     | 2026-09-18 | Differentiable Solutions S.r.l. | Version and date come from the revision history; status defaults to draft; PDF/A-2u by default.                                                                                                                                             |
| 1.5     | 2026-09-18 | Differentiable Solutions S.r.l. | The build's output is now under build/differentiable-docs/.                                                                                                                                                                                 |
| 1.6     | 2026-09-18 | Differentiable Solutions S.r.l. | Writing a document: what the identifier and Git already say is not written again.                                                                                                                                                           |
| 1.7     | 2026-09-22 | Differentiable Solutions S.r.l. | Conventions: a semicolon ends a Mermaid statement, captions included.                                                                                                                                                                       |
| 1.8     | 2026-09-25 | Differentiable Solutions S.r.l. | GitLab Flavored Markdown is the reference: callouts are GitLab alerts, titles included, and the conventions name what GitLab draws that the gate refuses; fenced code that names its language is highlighted.                               |
| 1.9     | 2026-09-29 | Differentiable Solutions S.r.l. | Building: +dirty counts a change to what the documents are built from, and not line endings or file modes.                                                                                                                                  |

## Contents

|     |                           |   |
|-----|---------------------------|---|
| 1   | Getting started .....     | 3 |
| 1.1 | What you need .....       | 3 |
| 1.2 | Your first build .....    | 3 |
| 1.3 | Your first document ..... | 3 |
| 1.4 | Where things live .....   | 4 |
| 2   | Writing a document .....  | 4 |

|     |                                                                       |    |
|-----|-----------------------------------------------------------------------|----|
| 2.1 | The smallest document the build accepts .....                         | 4  |
| 2.2 | What each field is for .....                                          | 5  |
| 2.3 | Identifiers .....                                                     | 5  |
| 2.4 | Saying more .....                                                     | 5  |
| 2.5 | The life of a document .....                                          | 6  |
| 3   | How a project is arranged .....                                       | 7  |
| 3.1 | Areas .....                                                           | 7  |
| 3.2 | The three shapes a document can take .....                            | 7  |
| 3.3 | Settings an area shares .....                                         | 8  |
| 3.4 | Adding the build to a repository that already has a docs folder ..... | 8  |
| 3.5 | Leaving something out .....                                           | 9  |
| 4   | Writing conventions .....                                             | 9  |
| 4.1 | Callouts .....                                                        | 9  |
| 4.2 | Tables .....                                                          | 10 |
| 4.3 | Figures .....                                                         | 10 |
| 4.4 | Diagrams .....                                                        | 10 |
| 4.5 | Page breaks .....                                                     | 11 |
| 4.6 | Evidence tags .....                                                   | 12 |
| 4.7 | Code and identifiers .....                                            | 12 |
| 5   | Building and publishing .....                                         | 12 |
| 5.1 | The commands .....                                                    | 12 |
| 5.2 | What comes out .....                                                  | 13 |
| 5.3 | Reproducibility .....                                                 | 13 |
| 5.4 | Publishing from GitLab .....                                          | 13 |
| 5.5 | Conformance, when documents must outlive the software .....           | 13 |
| 6   | When the build says no .....                                          | 13 |
| 6.1 | Two that look like bugs and are not .....                             | 15 |
| 6.2 | Looking at what the build did .....                                   | 15 |

# 1 Getting started

## 1.1 What you need

One of these, and nothing else:

- **Docker**, if you build from the image. Nothing is installed on your machine.
- **Linux or WSL** with GNU make 4 and bash, if your project holds its own copy of the build. Run `make toolchain` once; it downloads pandoc and Typst, verifies their checksums and puts them in `~/.local/bin`. It needs no administrator rights and touches nothing outside your home directory.

### Note

On Windows, work inside WSL. The build is a Unix toolchain, and the Windows filesystem is reachable from WSL under `/mnt/`.

## 1.2 Your first build

Build the worked examples and look at what comes out:

```
make examples          # or: docker compose run --rm docs make examples
```

Three documents appear under `build/differentiable-docs/examples/pdf/`. Each shows a different shape:

| Example                                       | Shows                                                                  |
|-----------------------------------------------|------------------------------------------------------------------------|
| <code>examples/conventions/document.md</code> | every writing convention, in one single-file document                  |
| <code>examples/security/</code>               | a directory document assembled from parts, with its own area manifest  |
| <code>examples/user/guide/</code>             | a nested directory document, approved and public, with no control page |

Open `build/differentiable-docs/examples/pdf/conventions/document.pdf` beside its source. Everything on the title page and the control page came from the front matter; nothing was typed twice.

## 1.3 Your first document

Copy a skeleton into an area:

```
cp skeletons/report.md docs/quality/first-report.md
```

Every placeholder in it is written `TODO ...`, the identifier included. Replace each one in the front matter, then run the gate and build:

```
make check
make
```

`make check` names the file and the field of anything missing, malformed or still a `TODO`, so a skeleton nobody filled in cannot reach a PDF. `make` then writes `build/differentiable-docs/pdf/quality/first-report.pdf`; `make html` adds a single-file web page beside it.

**Tip**

Run `make check` while you write. It is fast, it catches the whole class of mistakes that would otherwise reach a reviewer, and the pipeline runs exactly the same command.

## 1.4 Where things live

Nothing here is configurable, so it is the same in every project:

|                                         |                                                               |
|-----------------------------------------|---------------------------------------------------------------|
| <code>docs/</code>                      | your documents, one folder per area                           |
| <code>examples/</code>                  | worked examples (built only by <code>`make examples`</code> ) |
| <code>skeletons/</code>                 | one starting point per document type                          |
| <code>build/differentiable-docs/</code> | everything the build produces                                 |

`build/differentiable-docs/` is disposable: `make clean` removes it and nothing else, so a project's own `build/`, where a compiler may keep its output, is left alone. Nothing in it should be committed.

## 2 Writing a document

A document is a Markdown file that begins with a block of front matter between two lines of three dashes. The front matter is what makes it a controlled document rather than a page of text.

### 2.1 The smallest document the build accepts

```
---
title: "Backup procedure"
doc-id: DOC-OPERATIONS-PR-004
classification: Internal
owner: "Platform lead"
revision-history:
  - version: "1.0"
    change: "First issue."
---

# Purpose

Why this procedure exists.
```

The build refuses anything less, and says which field is missing and in which file. You write what only a person can say; everything else is worked out, and nothing is written twice:

- **The type is in the identifier.** PR in DOC-OPERATIONS-PR-004 is the code of a procedure, so the document needs no `doc-type`. The codes are listed in the reference.
- **The version and the date are the revision history's.** The last entry is the current version and its date of issue, so neither is written again above it.
- **Each revision's date and author come from Git.** They are those of the commit that added the entry's `version:` line, the date as its author's calendar showed it. An entry not committed yet is dated today and credited to the Git user who is building.

- **The authors come from Git too.** They are everyone who has committed to the document, in the order they first did, with bots and merge commits left out. For a document made of parts, that means the whole folder.
- **The status can be left out.** A document that does not say it is approved is a draft: it prints the watermark until someone sets `status: approved`.
- **Classification and owner can come from the manifest.** A project or an area that sets them in defaults leaves every document there with only its title, identifier and history.

Anything you write yourself wins. That covers a project without Git, an entry that has to record a date other than its commit's, such as a version issued before the document was moved into the repository, and a document whose byline is an organisation rather than the people who typed it.

## 2.2 What each field is for

| Field          | What it means                                                                                                                                                    |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| title          | printed on the title page and in the header                                                                                                                      |
| doc-id         | the identifier a reader quotes; unique across the project                                                                                                        |
| doc-type       | what kind of document it is; sets the label above the title; read from the identifier's type code unless written                                                 |
| classification | who may see it; sets the band colour and the notice                                                                                                              |
| owner          | the person or role accountable for it                                                                                                                            |
| version        | MAJOR.MINOR, <b>quoted</b> , so YAML does not read <code>1.10</code> as <code>1.1</code> ; written in each history entry, and the last entry's is the document's |
| date           | its date of issue, YYYY-MM-DD, which becomes the PDF's creation date; the last history entry's, from Git unless written                                          |
| status         | optional, draft unless written; anything but approved prints a watermark                                                                                         |

### Quote the version

`version: 1.10` is the number 1.1 in YAML, and your document would say so. Write `version: "1.10"`. The build rejects anything that is not MAJOR.MINOR or MAJOR.MINOR.PATCH.

## 2.3 Identifiers

The default shape is PREFIX-AREA-TYPE-NNN, for example DOC-OPERATIONS-PR-004. The area segment conventionally matches the folder the document sits in, in upper case, and the type segment is the code of the document type: PR procedure, SPEC specification, GD guide, RPT report, and so on for every type in the vocabulary. The build reads the type from that code. An identifier whose code is no type's is refused, unless the document writes `doc-type` itself, and a written `doc-type` that contradicts the code is reported. Your project can tighten the pattern, and give its own types their own codes, in its manifest.

Identifiers must be unique. `make check` compares every document in the project and names both files if two collide.

## 2.4 Saying more

Everything below is optional, and each field does one thing.

| Field                           | Effect                                                                                                                  |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| subtitle                        | a line under the title                                                                                                  |
| abstract                        | a paragraph on the title page; also the document's subject in the metadata                                              |
| author                          | a list, shown on the control page; the document's committers unless written                                             |
| reviewer                        | a list, shown on the control page                                                                                       |
| approved-by                     | shown on the control page; expected once the status is approved                                                         |
| revision-history                | a table on the control page; a list of {version, change}, each with a date and author of its own only to override Git's |
| product                         | shown above the title, for documents belonging to a product                                                             |
| caveat                          | a short warning carried in the footer of every page, in the caveat colour                                               |
| keywords, subject               | metadata, for searching and indexing                                                                                    |
| control-page                    | false drops the control page, for short records such as minutes                                                         |
| toc, toc-depth                  | contents on or off, and how deep                                                                                        |
| h1-pagebreak                    | true starts every top-level section on a new page                                                                       |
| paper, lang, logo, organisation | override the project's defaults for this document                                                                       |

## 2.5 The life of a document

status moves from draft to review to approved, and eventually to superseded. Only approved prints without a watermark, so an unapproved document can never be mistaken for a finished one — including in a printout, where the watermark is the only thing a reader has to go on.

### Decision required: Who sets approved-by

The approver sets status: approved and approved-by, not the author. If the status says approved and the approver is blank, the build warns.

Add a revision-history entry, with a new version and a line on what changed, for every change a reader could notice, and commit it: the commit dates and signs the entry. A document that writes a version or date of its own above the history, and disagrees with the last entry, is warned rather than refused: it is a mistake, but not one worth stopping a build over.

### Approve what is committed

Approval applies to a version, and a version is a commit. An approved document built from changes that are not committed is reported: what is being built is not what was approved.

### In a pipeline

Dates and authors come from the history, so the pipeline needs all of it. GitLab clones 20 commits by default. The CI/CD component sets GIT\_DEPTH: "0" for its jobs; a pipeline of your own sets it too. A shallow clone that lacks a commit the build needs is refused with that advice, not built with a wrong date.

## 3 How a project is arranged

### 3.1 Areas

Documents live in areas under `docs/`, one folder each. The standard set comes with the project and covers what most products need:

| Area         | Holds                                                 |
|--------------|-------------------------------------------------------|
| overview     | what the product is, for whom, the shared vocabulary  |
| requirements | what the system shall do                              |
| architecture | how it is built: context, building blocks, interfaces |
| decisions    | decision records, one per significant choice          |
| security     | threat models, risk registers, compliance, advisories |
| quality      | verification plans, test and assessment reports       |
| operations   | installation, configuration, procedures, runbooks     |
| user         | tutorials, how-to guides, reference, explanation      |
| releases     | release notes and per-release records                 |
| governance   | policies, plans, roles, minutes                       |

Each ships with a README stating its purpose. `make scaffold` recreates any that are missing and never overwrites. Rename, remove or add areas in the manifest to fit your organisation; a document in an area the manifest does not declare is reported.

### 3.2 The three shapes a document can take

| Shape              | Source                                                              | Output                                                                   |
|--------------------|---------------------------------------------------------------------|--------------------------------------------------------------------------|
| Single file        | <code>docs/&lt;area&gt;/&lt;name&gt;.md</code>                      | <code>build/differentiable-docs/pdf/&lt;area&gt;/&lt;name&gt;.pdf</code> |
| Directory          | <code>docs/&lt;area&gt;/&lt;name&gt;/index.md</code> plus its parts | <code>build/differentiable-docs/pdf/&lt;area&gt;/&lt;name&gt;.pdf</code> |
| Area as a document | <code>docs/&lt;area&gt;/index.md</code> plus its parts              | <code>build/differentiable-docs/pdf/&lt;area&gt;.pdf</code>              |

A directory document is one document written as several files. `index.md` carries the front matter and includes its parts in reading order, one GitLab include directive per line, such as `::include{file=01-scope.md}`, the lines together in a paragraph of their own; the parts are plain Markdown with no front matter of their own. The set stays one artefact, with one identifier, one version and one approval, while each part can be written and reviewed on its own. GitLab shows the document assembled, as the PDF does, so the build keeps to GitLab's rules: a part cannot include another, a document has at most 32 includes, and a part's headings are written at the level they take in the document. A directive inside a code block puts the file's text in the block. The older spelling, a code block of class `include` listing the parts, still builds, with a warning.

Symlinks are refused anywhere under the roots. A symlink reaches outside the repository without any path that looks as though it does, and it is the first thing to break on a Windows checkout, so a shared fragment is included rather than linked.

Files whose names begin with an underscore, and everything inside a directory whose name begins with one, are never documents. That is where shared fragments belong — `docs/_shared/` for anything used by several documents, `_parts/` beside a single-file document for its own sections.

**Note**

A part sitting beside an `index.md` that no include names is in the repository and in no document. `make check` reports it, because that is nearly always an oversight.

### 3.3 Settings an area shares

A `differentiable-docs.yaml` in any directory applies to every document in it and below, nearest first. It is where settings shared by an area belong, rather than repeated in each front matter:

```
look:
  accent: "#8A1538"
defaults:
  classification: Confidential
  owner: "Security owner"
```

Documents in that area then state neither their classification nor their owner. `defaults` can satisfy a required field, and a document's own front matter always wins.

An area manifest may set `project`, `look`, `strings`, `defaults` and `meta`, and nothing else. The rules, the build settings and the layout stay in the project manifest, so the neighbourhood of a document cannot weaken the gate that document has to pass.

### 3.4 Adding the build to a repository that already has a docs folder

Most software repositories already keep something in `docs/`: pages for MkDocs, a wiki export, notes. They can stay where they are. A Markdown file is a controlled document when its first line opens a front-matter block; a file with no front matter is left alone — not validated, not built — and `make check` names it, so nothing is skipped without being said:

```
check-docs: not controlled (no front matter; neither validated nor built)
  skip docs/fw_build.md
```

So adopting the build takes one step: create an area folder, for example `docs/security/`, copy a skeleton into it and fill it in. The other tool keeps reading the same folder; if it should not publish your controlled documents, exclude the area in that tool's configuration.

Three things to know:

- A file with *any* front matter is validated in full. A page of another tool that carries its own front matter therefore has to be listed under `build.exclude`.
- The folder may be `docs/`, `Docs/` or `DOCS/`: it is read under the name it already has, so nothing has to be renamed and no existing link breaks. Output paths do not contain it.
- Once everything under `docs/` is controlled, set `build.uncontrolled: refuse`, so that a document whose front matter was forgotten fails the gate instead of being skipped.

### 3.5 Leaving something out

Every controlled document under `docs/` is built. To keep drafts or archives out, list them in the project manifest by path, with globs if you like:

```
build:
  exclude: [drafts, "*/archive", platform/old-spec.md]
```

## 4 Writing conventions

Documents are written in GitLab Flavored Markdown, so what GitLab draws in a merge request is what the PDF shows: pipe tables, task lists, footnotes, strikethrough, fenced code, mathematics, Mermaid diagrams and alerts. The build adds four things for what GitLab has no syntax for, and each renders in both PDF and HTML.

| Added            | What it is for                                            |
|------------------|-----------------------------------------------------------|
| attributes       | {width=80%} on an image                                   |
| fenced divs      | landscape sections and page breaks                        |
| implicit figures | an image alone in a paragraph becomes a numbered figure   |
| smart typography | quotes and dashes set for print rather than left as ASCII |

Nothing else is added, which is the point: a document reads correctly in a merge request or a repository browser before anyone builds it, and there is no dialect to learn. A few things GitLab draws have no equal in a PDF; the gate refuses them and names what to write instead (the reference lists them).

### 4.1 Callouts

A callout is a GitLab alert, so it looks the same in a merge request as in the PDF: the same heading, the same colour. Text after the marker is the title; without it, the callout is headed by its kind.

```
> [!warning] Check the rollback step first
> Applying this on a live system without it will lose the current configuration.
```

GitLab draws five kinds. The project's other three are titled alerts of the kind closest to them, and the title says which one it is:

| Written as                            | For                                                          |
|---------------------------------------|--------------------------------------------------------------|
| > [!note]                             | context the reader needs that is not itself a conclusion     |
| > [!tip]                              | a shortcut or a recommended practice                         |
| > [!warning]                          | a consequence worth stopping for                             |
| > [!caution]                          | a consequence that is worse than a warning                   |
| > [!important]                        | what must not be missed                                      |
| > [!important] Decision required: ... | a choice only the owner can make, and what each option costs |
| > [!warning] Gap: ...                 | something that should exist and does not, with an owner      |
| > [!caution] Finding: ...             | one verified issue: evidence, precondition, consequence      |

Any other marker, [!FINDING] among them, is refused: GitLab would print it as text, and the build names the alert to use instead. The kind may be written in either case. A title needs GitLab 17.10 or later; GitHub shows a titled alert as an ordinary quote. The older spelling, a fenced div such as ::: {.finding title="..."}, still builds, with a warning, because GitLab shows its colons and braces as text.

#### Tip

A decision is the block a reviewer looks for first. Use it for the questions you need answered, not for opinions.

## 4.2 Tables

Write ordinary pipe tables. Two things happen automatically.

A table of six columns or more is given a **landscape page of its own**, so a register or a compliance matrix stays readable without anyone thinking about page geometry. Wrap a narrower table in ::: {.landscape} ... ::: to force the same treatment; a table that is both explicit and wide is not wrapped twice.

**Column widths follow the content.** A pipe table's separator row says how a column is aligned, not how wide it is, which is how a one-word Status column ends up as wide as a Description one. The build sizes each column from its longest cell instead, blended towards equal by tables.width-blend so a narrow column still fits its heading.

## 4.3 Figures

```
![[The build: each step is one pinned tool.](images/pipeline.svg){width=80%}
```

The path is relative to the document, and figures conventionally live in an images/ folder beside it. The caption becomes the figure's caption and its alternative text. A path that is absolute, or that climbs out of the project, is refused.

## 4.4 Diagrams

Write a diagram as a Mermaid block. GitLab and GitHub draw it in the merge request, and the build draws it into the document:

```
```mermaid
flowchart LR
    accTitle: How a change reaches a reader.
    A[Author] --> B[Review]
    B -- approved --> C[Published]
    B -- changes asked --> A
```
```

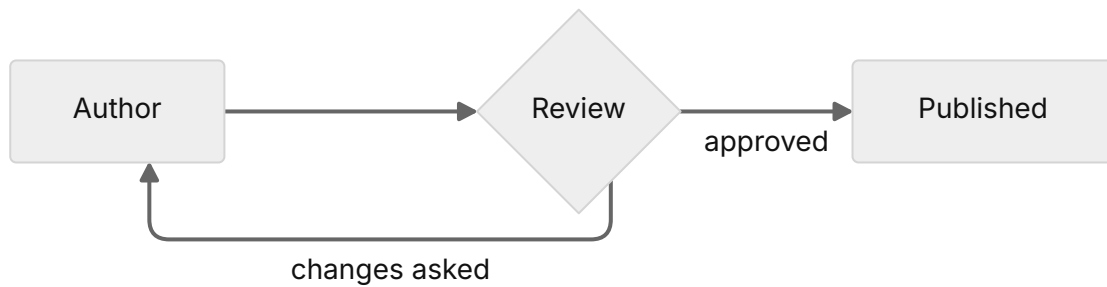


Figure 1: How a change reaches a reader.

accTitle is Mermaid's own accessibility title. Give one and the diagram becomes a numbered figure with that caption; leave it out and the diagram is placed without one.

A semicolon ends a Mermaid statement wherever it stands, the accTitle line included: a caption reading listens; every other path is closed draws a stray node named every. Write captions and labels without one.

Put what an arrow carries in the node it points at, not on the arrow: the renderer routes a labelled edge into its target along the target's border, and the arrowhead follows that last segment sideways. PRX[nginx proxy<br/>TCP 443] with a plain --> draws a straight arrow with its head where it belongs; -- "TCP 443" --> does not. The same holds for an edge that skips a rank.

The picture is an intermediate file under build/differentiable-docs/diagrams/, drawn when the block changes. Nothing is committed but the text.

Every kind below has a test of its own, which draws it and checks that its labels are readable in the result, so this list is what the build is known to draw rather than what it hopes to:

|                              |                     |              |
|------------------------------|---------------------|--------------|
| flowchart (flowchart, graph) | sequence            | class        |
| state                        | entity relationship | user journey |
| gantt                        | pie                 | mindmap      |
| timeline                     | git graph           | quadrant     |
| requirement                  | sankey              | xy chart     |
| block                        | C4 context          | packet       |
| kanban                       | architecture        | radar        |
| treemap                      |                     |              |

Two things to expect. The PDF is drawn by a different program from the one GitLab uses, so the structure matches and the spacing does not. And a diagram the build cannot draw stops it with the renderer's own message, in which case include an SVG image instead.

## 4.5 Page breaks

```

::: {.pagebreak}
:::

```

Or set h1-pagebreak: true in the front matter to start every top-level section on a new page. A LaTeX \newpage is not Markdown and would print as text, so the build refuses it and names the spelling above.

## 4.6 Evidence tags

Optional, and useful wherever a reader needs to know how firm a claim is: [C] confirmed against a named artefact, [I] inferred, [A] an assumption that a named check would settle. They render as small coloured badges.  This sentence is an example of one.

The rule that goes with them: absence of evidence is a gap, not a pass.

## 4.7 Code and identifiers

Commands and configuration go in fenced blocks, never in running prose; inline identifiers use backticks. Inline code never overflows a table: a long identifier breaks at its separators when a cell cannot hold it, and a list such as `.tip`, `.warning` wraps like any other. Nothing is inserted into the text to make that happen, so what a reader copies out of the PDF is what you wrote.

Name the language after the opening fence (````powershell`, ````bash`, ````yaml`) and the block is highlighted, in the PDF and the HTML alike and in the same colours, each legible on the block's background. `pandoc --list-highlight-languages` prints the names the build knows; a block in any other language, or with none, is printed plain.

### Never paste a secret

Not even one a report handed you. Name the file and the line and say what kind of secret it is. `make check` rejects private-key headers, cloud and platform tokens, and assignments that look like credentials. A line that legitimately needs such a pattern carries the marker `allow-secret-pattern`.

# 5 Building and publishing

## 5.1 The commands

| Command                                                | Does                                                                            |
|--------------------------------------------------------|---------------------------------------------------------------------------------|
| <code>make</code>                                      | every document, in the formats the manifest asks for (the PDF, by default)      |
| <code>make pdf</code> , <code>make html</code>         | one format only                                                                 |
| <code>make check</code>                                | the gate: names, front matter, unique identifiers, orphan parts, fonts, secrets |
| <code>make test</code>                                 | the build's own tests                                                           |
| <code>make list</code>                                 | the document keys the build would produce                                       |
| <code>make examples</code>                             | the worked examples                                                             |
| <code>make scaffold</code>                             | create any missing area under <code>docs/</code>                                |
| <code>make manifest</code>                             | print the effective manifest                                                    |
| <code>make clean</code>                                | remove <code>build/differentiable-docs/</code> , and nothing else               |
| <code>./build.sh docs/user/guide -o ~/guide.pdf</code> | one document, copied where you want it                                          |

From the container image the command is `differentiable-docs`, and takes the same arguments: `differentiable-docs check`, `differentiable-docs list`.

## 5.2 What comes out

```
build/differentiable-docs/
pdf/<area>/<name>.pdf      the document
typ/<area>/<name>.typ       the intermediate Typst, kept on purpose
html/<area>/<name>.html    the same content as one self-contained file,
                           from `make html` or from `formats: [pdf, html]`
```

The intermediate file is there for the day a page looks wrong: it is readable, and the conventions map onto named functions in it that are easy to find.

## 5.3 Reproducibility

The PDF's timestamp comes from the commit, not from the moment you ran the build, so two builds of the same commit are byte-identical. The title page states the commit it was built from, with +dirty when what the documents are built from has changes not yet committed: the documents roots, the manifest, the project's fonts, and the tool when it lives in the same repository. So a printed page can always be traced back to a source you can check out. A change elsewhere in the repository, such as the code of a product whose documents sit beside it, does not count, and nor do line endings or file modes: a Windows checkout built in the container is not dirty.

### Note

Build a document that has never been committed and the title page says unversioned, and the build says so too. That is not a failure, but it does mean the PDF cannot be traced to anything.

## 5.4 Publishing from GitLab

The pipeline runs the same gate you run, then builds. On the default branch it can publish the documents to Pages; on a tag it creates a Release with every PDF in the package registry. The operations area has the procedure, including the three-line pipeline a project needs when the build comes from the container image.

## 5.5 Conformance, when documents must outlive the software

Every PDF is built as PDF/A-2u, an archival file whose text can always be searched and copied: a controlled document is kept for as long as the product it describes, and has to open long after the software that made it. A project with an accessibility obligation asks for more:

```
pdf:
  standard: a-2a,ua-1 # archival and accessible
```

Both require alternative text on every image: the text between the brackets of an image, or an accTitle: line in a diagram. make check names every image that has none, so turn them on early and let the build tell you.

## 6 When the build says no

Most messages name the file and the field. This is what the less obvious ones mean.

| Message                                                           | What happened                                                                                        | What to do                                                                                  |
|-------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| required metadata 'owner' is missing                              | a required field is absent                                                                           | add it, or let an area manifest supply it through defaults                                  |
| classification 'Secret' is not one of: ...                        | a value outside the vocabulary                                                                       | use a listed value, or add yours to the manifest                                            |
| version 'v1' must be MAJOR.MINOR                                  | not a version number                                                                                 | write "1.0", quoted                                                                         |
| date '2026-13-45' is not a calendar date                          | a real-looking date that does not exist                                                              | correct it                                                                                  |
| doc-id '...' does not match the configured pattern                | the identifier is the wrong shape                                                                    | follow PREFIX-AREA-TYPE-NNN, or relax the pattern                                           |
| duplicate doc-id ...                                              | two documents claim one identifier                                                                   | both files are named; change one                                                            |
| skip docs/page.md under <i>not controlled</i> , and no PDF for it | the file has no front matter, so it is not a controlled document                                     | start it from a skeleton; if no file should ever be skipped, set build.uncontrolled: refuse |
| documents are read from docs/; Docs/ also exists and is not read  | two spellings of the folder on a case-sensitive system                                               | keep one; they cannot coexist in a Windows checkout                                         |
| a Mermaid diagram could not be drawn                              | a syntax error, or a kind of diagram the renderer does not support; its own message is printed above | fix the block (GitLab's preview shows syntax errors too), or include an SVG image           |
| mmdr, which draws it, is not installed                            | the toolchain predates diagrams                                                                      | make toolchain; the image already carries it                                                |
| front matter must not define 'vocabulary'                         | a document tried to change the rules                                                                 | rules live in the manifest, not in documents                                                |
| 'owner' still holds a skeleton placeholder                        | a TODO copied from a skeleton was never replaced                                                     | fill it in; the message names the field                                                     |
| cannot read '...' from an include                                 | a listed part does not exist                                                                         | check the path, which is relative to the including file                                     |
| escapes the project root                                          | an include or image points outside the project                                                       | keep everything inside; this is refused by design                                           |
| Could not fetch resource                                          | an image path is wrong                                                                               | paths are relative to the document                                                          |
| \newpage is LaTeX, not Markdown                                   | a LaTeX page break, which would print as text                                                        | write ::: {.pagebreak} / :::                                                                |
| this looks like a grid table                                      | a table in a dialect GitLab does not draw                                                            | write it as a pipe table                                                                    |
| document paths must not contain spaces                            | a file or folder name has a space                                                                    | rename with hyphens                                                                         |
| symlinks are not followed                                         | a symlink under one of the roots                                                                     | include the file instead, or put a shared fragment in docs/_shared/                         |
| area '...' is not in the manifest layout                          | a document sits outside the declared areas                                                           | move it, or declare the area                                                                |
| font '...' is not available                                       | the manifest names a face that is not installed                                                      | install it, or accept the substitute                                                        |

**Warnings are not failures**

A missing revision history, a version that disagrees with the last history entry, an approved document with no approver, an orphan part, an unavailable font: these are reported and the build continues. Everything else stops it.

## 6.1 Two that look like bugs and are not

**"It says unversioned and the build is not reproducible."** The project has no commits yet, or git will not read the checkout because it is owned by another user. The message tells you the command that fixes the second case.

**"My long identifier breaks across a line in a paragraph, but not in a table."** Tables always let inline code break at its separators when a cell cannot hold it. In running text that is off, because justified text would also take such a break just to even out a line. Set `look.break-long-code: true` if a paragraph has an identifier too long for the line.

## 6.2 Looking at what the build did

```
make list          # what it thinks the documents are
make manifest      # every setting in effect
cat build/differentiable-docs/typ/<area>/<name>.typ # the intermediate, when a page
looks wrong
```

If a change to a part does not seem to rebuild a document, `make list` and the generated plan under `build/differentiable-docs/plan.mk` show exactly which files each document depends on.