Authoring Essentials

Session 2

Mickaël CANOUIL, Ph.D.[1*]

Saturday, the 7th of February, 2026

Affiliations

[1] Independent Contractor, Lille, France

ORCID

Mickaël CANOUIL, Ph.D.: 0000-0002-3396-4549

[*]Corresponding author: pro@mickael.canouil.dev

Table of Contents

Table of Contents

1. Session Overview

1.1. Session 2: Authoring Essentials

- Markdown Fundamentals
  - Creating structured documents: headings, lists, links, and text formatting.
  - Best practices for writing clean, reproducible content in Quarto.
  - Working with code blocks, tables, and mathematical expressions.
- Quarto Markdown
  - Callout blocks, shortcodes, and Mermaid diagrams.
  - Cross-references for figures, tables, and custom elements.
  - Code annotations and advanced formatting features.
- YAML Configuration
  - Document headers and multi-format output.
  - Metadata inheritance and project configuration.
  - Extensions and customisation options.

1.2. Session Objectives

This session focuses on building foundational skills in Quarto authoring, covering enhanced markdown capabilities, YAML configuration, and document structuring best practices.

1.3. Learning Objectives

By the end of this session, participants will be able to:

- Create well-structured documents using enhanced Quarto markdown features.
- Configure documents using YAML headers for different output formats.
- Use advanced formatting features including callouts, cross-references, and tabsets.

2. Markdown Basics

2.1. Text Formatting Essentials

Basic text formatting you'll use every day:

- Bold text with `**bold**` or `__bold__`.
- Italic text with `*italic*` or `_italic_`.
- Bold and italic with `***text***`.
- Superscript^2 with `superscript^2^`.
- Subscript$_2$ with `subscript~2~`.
- `Verbatim code` with backticks.

2.2. Headers Structure Your Content

Headers create your document structure and enable automatic table of contents generation.

MARKDOWN

```
# Heading 1 - Document Title
## Heading 2 - Major Sections
### Heading 3 - Subsections
#### Heading 4 - Minor Sections
##### Heading 5 - Rare Usage
###### Heading 6 - Very Rare
```

Tip

Add a empty lines before and after!

2.3. Lists

Important

Quarto requires empty lines before and after lists to ensure proper rendering!

Unordered Lists:

MARKDOWN

```
- First item
- Second item
- Third item
```

Ordered Lists:

MARKDOWN

```
1. First step
2. Second step
3. Third step
```

2.4. Lists - Nested

> **!** Important
>
> The number of spaces for indentation does not matter, but the list marker must be aligned with the first character of the parent item. Items marked bad below show incorrect alignment of nested content.

MARKDOWN

````
- First item

  ```r
  print("Hello, World!")
  ```

-    Second item (**bad**)

   ::: {.callout-tip}
  Here is a tip!
   :::

        - Sub item

-    Third item

      ::: {.callout-tip}
     Here is a tip!
      :::

   - Sub item
````

MARKDOWN

````
- First item

  ```r
  print("Hello, World!")
  ```

-  Second item (**bad**)

   ::: {.callout-tip}
  Here is a tip!
   :::

   - Sub item

-    Third item

     ::: {.callout-tip}
    Here is a tip!
     :::

     - Sub item
````

2.5. Working with Code

Inline code: Use single backticks

MARKDOWN

```
The `print()` function outputs text.
```


> **Tip**
>
> Code highlighting for inline code:
>
> ```markdown
> The `print()`{.python} function outputs text.
> ```

Code blocks: Use triple backticks with optional language

````markdown
```python
def hello_world():
    print("Hello, Quarto!")
```
````

> **Note**
>
> The syntax highlighting is provided by `skylighting`.

2.6. Tables Made Simple

```markdown
| Default | Left | Right | Center |
|---------|:-----|------:|:------:|
| 12      | 12   |    12 |   12   |
| 123     | 123  |   123 |  123   |
| 1       | 1    |     1 |   1    |

: Demonstration of pipe table
syntax
```

Default	Left	Right	Center
12	12	12	12
123	123	123	123
1	1	1	1

Use colons (`:`) to control alignment:

- `:---` for left alignment.
- `---:` for right alignment.
- `:---:` for centre alignment.

See Table basics for more.

2.7. Mathematical Expressions

Inline maths: `$E = mc^2$` renders as $E = mc^2$.

Display maths: Use double dollar signs (on separate lines).

```markdown
$$
E = mc^2
$$
```

Define custom LaTeX macros in hidden blocks for reuse across your document, see Markdown Basics - Equations

2.8. Footnotes and References

Standard footnotes:

```markdown
Here is a footnote reference[^1].

[^1]: Here is the footnote content.
```

Inline footnotes:

```markdown
Here is an inline note^[Much easier to write!].
```

> **!** Important
>
> Footnote identifiers must be unique within the document.

2.9. Figure Basics

- Basic

```markdown
![Elephant](elephant.png)
```

- Size and alignment

```markdown
![Elephant](elephant.png){width=300}
![Elephant](elephant.png){width=80%}
![Elephant](elephant.png){width=4in}

![Elephant](elephant.png){fig-align="left"}
```

- Alternative text and titles


```markdown
![](elephant.png){fig-alt="A drawing of an elephant."}
![Elephant](elephant.png "Title: An elephant"){fig-alt="A drawing of an elephant."}
```

See Figure basics for more.

2.10. Advanced Formatting: Divs and Spans

Divs for block-level content:

```markdown
::: {#ID .class attribute=value}
This is a "fenced div".
:::
```

Spans for inline formatting:

```markdown
[This is a "span"]{#ID .class attribute=value}
```

See Divs and Spans for more.

3. Quarto Markdown

3.1. Enhanced Markdown Capabilities

Quarto extends standard markdown with powerful features designed specifically for computational documents.

3.2. Shortcodes

3.2.1. What is a Shortcode?

Shortcodes are special markdown directives that generate various types of content.

They provide a simple, standardised way to insert dynamic or complex content into your Quarto documents without requiring complex markdown syntax or HTML knowledge.

3.2.2. Built-in Shortcodes

Table 3.1: Shortcodes

Shortcode	Description
version	Print Quarto CLI version.
var	Print value from `_variables.yml` file.
meta	Print value from document metadata.
env	Print system environment variable.
pagebreak	Insert a native page-break.
kbd	Describe keyboard shortcuts.
video	Embed a video in a document.
include	Include contents of another qmd.
embed	Embed cells from a Jupyter Notebook.
placeholder	Add placeholder images to your document.
lipsum	Add placeholder text to your document.
contents	Rearrange content in your document.

3.3. The `include` Shortcode

Reuse content across multiple documents effortlessly:

MARKDOWN

```
{{< include _content.qmd >}}
```


3.3.1. Key Benefits

- DRY Principle: Write once, use everywhere.
- Consistency: Shared content stays synchronised.
- Maintenance: Update in one place, changes propagate automatically.
- Modularity: Break complex documents into manageable pieces.

3.3.2. Common Use Cases

- Reusable code.
- Common configuration snippets.

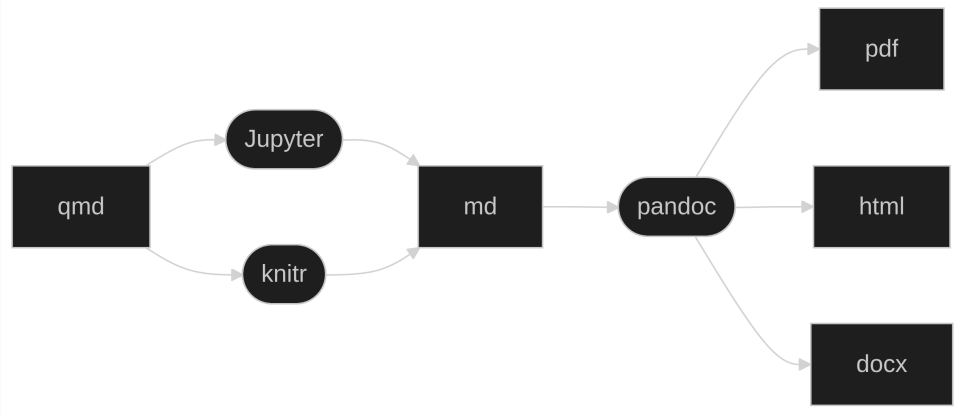
> **Tip**
>
> Prefix your file or directory with an underscore (`_`) to hide from Quarto's rendering process.

> **!Important**
>
> The `include` shortcode is a plain copy-paste of the file content. Any YAML header in the included file will replace the current document's YAML header.

3.4. Mermaid Diagrams

{{MERMAID}}

```
{{mermaid}}
%%| label: quarto-mermaid-example-1
%%| fig-width: 5
flowchart LR
  qmd ⟶ J([Jupyter])
  qmd ⟶ K([knitr])
  J ⟶ md
  K ⟶ md
  md ⟶ P([pandoc])
  P ⟶ typst
  P ⟶ html
  P ⟶ docx
```



See Diagrams for more.

3.5. Callout Blocks

Five types of callouts for better communication: `note`, `tip`, `warning`, `caution`, `important`.

MARKDOWN

```
::: {.callout-note}
## Note

Important information for readers.
:::

::: {.callout-warning}
## Warning

Something to be careful about.
:::
```

> **Note**
>
> Important information for readers.

> **Warning**
>
> Something to be careful about.

See Callout Blocks for more.

3.6. Code Annotations

- Source

MARKDOWN

````
```r
library(tidyverse)
library(palmerpenguins)
penguins |>                                                  # <1>
  mutate(                                                    # <2>
    bill_ratio = bill_depth_mm / bill_length_mm,             # <2>
    bill_area  = bill_depth_mm * bill_length_mm              # <2>
  )                                                          # <2>
```

1. Take `penguins`, and then,
2. add new columns for the bill ratio and bill area.
````


- Output

```r
library(tidyverse)
library(palmerpenguins)
penguins |>                                    (1)
  mutate(
    bill_ratio = bill_depth_mm /
bill_length_mm,
    bill_area  = bill_depth_mm *
bill_length_mm
  )                                            (2)
```

(1) Take `penguins`, and then,
(2) add new columns for the bill ratio and bill area.

See Code Annotation for more.

3.7. Cross-references: “figures”

```markdown
::: {#fig-example}
This is a figure

This is a caption for the figure.
:::

@fig-example is the reference to a
figure.
Or is it?
```

This is a figure

Figure 3.1: This is a caption for the figure.

Figure 3.1 is the reference to a figure.

Any label/ID starting with `fig-` will be treated as a figure cross-reference.

See Basic Cross-Reference - Figures for more.

3.8. Cross-references: “tables”

```markdown
::: {#tbl-example}
This is a table

This is a caption for the table.
:::

@tbl-example is the reference to a
table.
Or is it?
```

Table 3.2: This is a caption for the table.
This is a table

Table 3.2 is the reference to a table.

Any label/ID starting with `tbl-` will be treated as a table cross-reference.

See Basic Cross-Reference - Tables for more.

3.9. Cross-references: “anything”

Define a new custom cross-reference type:

```yaml
crossref:
  custom:
    - kind: float                    (1)
      key: txt                       (2)
      reference-prefix: Text         (3)
```

(1) Cross-referenceable elements with captions are `float`.
(2) The identifier prefix used in labels (e.g., `@txt-example`).
(3) The text prefix shown in output (e.g., “Text 1”).

```markdown
::: {#txt-example}
This is text

This is a caption for the text.
:::

@txt-example is the reference to a
text.
```

This is text
Text 3.1: This is a caption for the text.

Text 3.1 is the reference to a text.

See Custom Float Cross-Reference Types for more.

3.10. Citations

3.10.0.1. Setup Required

- Bibliography file (`.bib`, `.bibtex`, etc.).
- Optional: CSL file for custom styling.

3.10.0.2. YAML Configuration

YAML

```yaml
bibliography: references.bib
csl: nature.csl  # Optional custom style
```

3.10.0.3. Bibliography Generation

- Automatic placement at document end.
- Use `## References {#refs}` for custom placement.
- `nocite: "@*"` includes all references.

3.10.0.4. Citation Syntax

- `[@author2023]` → (Author 2023)
- `[@author2023, p. 15]` → (Author 2023, 15)
- `[@smith2020; @jones2021]` → (Smith 2020; Jones 2021)
- `@author2023 says ...` → Author (2023) says...
- `[-@author2023]` → (2023) - suppress author

4. YAML Configuration

4.1. Overview

The YAML header serves as the configuration hub for your documents, controlling output formats, execution behaviour, and visual presentation.

Metadata Inheritance:

- Project level: `_quarto.yml` (lowest priority).
- Directory level: `_metadata.yml` (medium priority).
- Document level: YAML header (highest priority).

Multiple Metadata Files:

```yaml
metadata-files:
  - _website.yml
  - _common.yml
```

4.2. Document Header

```yaml
---
title: "My Analysis Report"
author: "Your Name"
date: today                    (1)
format:
  html:
    toc: true
    code-fold: true
    theme: cosmo               (2)
execute:
  echo: true
  warning: false
  message: false               (3)
---
```

(1) Common metadata fields. Option shared by all formats.
(2) Option specifically set for HTML output.
(3) Global options controlling code execution behaviour.

4.3. Multi-Format Output

Configure multiple output formats simultaneously and share options:


```yaml
---
title: "Multi-Format Document"
toc: true                                    (1)
format:
  html:
    theme: flatly
    code-fold: true
    respect-user-color-scheme: true
  pdf:
    documentclass: article
    geometry: margin=1in                     (2)
  docx:
    reference-doc: template.docx
---
```

(1) Option shared by all formats.
(2) Option specifically set for PDF (LaTeX) output.

4.4. Extensions

Installing the `highlight-text` extension

```bash
quarto add mcanouil/quarto-highlight-text@2.0.1
```

```yaml
filters:
  - highlight-text
```

```markdown
[White text, Blue background]{
  colour="#FFFFFF" bg-
colour="#0000FF"
}
```

White text, Blue background

Discover more extensions at:

- https://quarto.org/docs/extensions/.
- http://m.canouil.dev/quarto-extensions/.

https://github.com/mcanouil/quarto-highlight-text

5. Hands-On Exercise: Building Your Quarto Portfolio Document

5.1. Exercise Overview

Objective: Enhance your Session 1 project with advanced Quarto markdown features, demonstrating authoring essentials and creating a comprehensive personal portfolio document.

Example Code: Exercises

```bash
tar -xzf "02-exercises.tar.gz" -C "02-authoring-essentials"
```

5.2. Part 1: Document Enhancement

Building on your Session 1 project, enhance your document with these new authoring features:

1. Create a new document with multi-format support:

```yaml
---
title: "My Quarto Learning Portfolio"
author: "Your Name"
date: today
format:
  html: default
  typst: default
---
```

2. Add a numbered table of contents structured by using:

```markdown
### Learning Progress
[New section - see Part 2]

### Skills Showcase
[New section - see Part 3]

### Reflection & Next Steps
[Your existing learning goals and next steps]
```

5.3. Part 2: Advanced Formatting Features

Add these enhanced markdown elements to your “Learning Progress” section:

1. Create a skills tracking table:

```markdown
## Learning Progress

| Feature | Confidence Level | Notes |
|---------|:----------------:|-------|
| Basic Markdown | ☆☆☆ | Comfortable with headers, lists |
| YAML Configuration | ☆☆ | Learning multi-format setup |
| Cross-references | ☆ | Just discovered this! |

: My Quarto Skills Progress {#tbl-skills}

As shown in @tbl-skills, I'm making steady progress!
```

2. Add callout blocks for key insights:

```markdown
::: {.callout-tip}
## Today's Biggest Discovery

I learned that Quarto can render to multiple formats  simultaneously -
this will revolutionise my workflow!
:::

::: {.callout-note collapse="true"}
## Session Notes

- Enhanced markdown features are powerful
- Cross-references work across formats
- YAML controls everything!
:::
```

5.4. Part 3: Skills Showcase Section

Create a “Skills Showcase” section demonstrating your new capabilities:

- Mathematical expressions:

```markdown
## Skills Showcase

### Mathematical Typesetting

I can now write inline maths like
$E = mc^2$ and display equations:

$$
\text{Learning Rate} =
\frac{\text{New Concepts}}
{\text{Practice Time}}
$$
```

- Code blocks with annotations:

````markdown
### Code Documentation

```r
# My first annotated code block
library(ggplot2)
# <1>
data(mtcars)
# <2>
ggplot(mtcars, aes(x = mpg)) +
# <3>
  geom_histogram()
# <3>
```

1. Load the plotting library
2. Use built-in dataset
3. Create a simple histogram
````

- Cross-referenced figure:

````markdown
### Workflow Diagram

::: {#fig-workflow}

```
Learn → Practice → Apply →
Teach
```

My Quarto learning workflow
:::

@fig-workflow shows my approach to
mastering new tools.
````

- Footnotes for additional context:

```markdown
This exercise builds on Session
1^[Where we learned project setup
and IDE integration]
and adds the authoring essentials
from Session 2.
```

5.5. Part 4: Multi-Format Testing

Test your enhanced document:

- Render to HTML

```bash
quarto render your-document.qmd --to html
```

- Try PDF output

```bash
quarto render your-document.qmd --to typst
```

- Compare the outputs

5.6. Success Criteria

☑ You've successfully completed the exercise if you can:

- Customise your existing project with YAML configuration.
- Implement cross-references with tables and figures.
- Use callout blocks effectively for highlighting information.
- Create annotated code blocks.
- Integrate mathematical expressions appropriately.
- Successfully render to multiple formats.