Contents

Dev. Guide

This guide explains how this resource is organized, how it is built and deployed, and how to maintain this resource. It is intended to be comprehensive, but should most likely be read only after having read our contributing[1] and UCA[2] guides.

Resources Organization Overview

Folders and Files

The source code repository[3]'s main branch is organized as follows:

[1] https:/princomp.github.io/docs/about/contributing
[2] https:/princomp.github.io/docs/academic_life/uca_guide#editing-the-resources
[3] https://github.com/princomp/princomp.github.io

path	description
`.github/`	github templates and configuration for github actions
`misc/`	resources that need to be either integrated into the resource, or discarded
`source/`	source for the material
`licence.md`	license file
`readme.md`	presentation of the repository

The `source/` folder contains the following:

path	description
`code/`	code examples (snippets and projects)
`docs/`	additional helpful documentation
`fonts/`	the fonts (redistributed with permission) used by this resource
`img/`	images, sometimes with their LaTeX source code
`labs/`	lab exercises
`lectures/`	lecture notes
`slides/`	slides
`templates/`	templates and filters used for building this resource
`vid/`	video files
`Makefile`	makefile used to compile this resource
`index.md`	website index page
`order`	file used to specify the order on the website's menu and the book

Building and Deploying

The content is built and deployed in two phases:

- Running `make all` in the `source/` folder will create a `content/` folder at root level containing:
  - one `.md` file per `.md` file in the `source/` folder (in the same location: `source/labs/If.md` is compiled to `content/labs/If.md`),

resulting from pandoc[4]'s conversion,
  - one `.pdf`, `.odt` and `.docx` file per `.md` file (with the exception of the `index.md` files) in the `source/` folder (in the same location: `source/labs/If.md` is compiled to `content/labs/If.pdf`), resulting from pandoc[5]'s conversion,
  - some files from the `img/`, `slides/` and `vid/` folders, copied selectively (for example, only the `.jpeg`, `.png`, `.pdf`, `.svg` and `.gif` files are copied from the `img/` folder),
  - the `.woff` and `.woff2` files copied from the `fonts/` folders,
  - a `code/projects/` folder containing, for each `Program.cs` file contained in a `source/code/projects/x/y`, a `x.zip` archive containing a C# project including `Program.cs` along with some (optional) class file,
  - a `web-order.ts` file, compiled from the `source/order` file, that fixes the order used by the website in the menu,
  - a `book.html`, a `book.pdf`, a `book.html` and a `book.docx` file resulting from pandoc[6]'s conversion of the `.md` files contained in the `SOURCE_BOOK`'s makefile variable (containing all the `.md` files in the `source/docs/` and `source/lectures/`, in the order fixed by the `order` file).
- Then, using the files in the generated `content/` folder, a website is built using quartz[7] and deployed to `https://princomp.github.io/`. This is achieved mainly thanks to the `.github/workflows/build_and_deploy.yaml` file and github's actions[8].

Tools, Briefly

This resource is mainly developed and powered using

- git[9]
- pandoc[10]
- make[11]
- python[12]
- quartz[13],

[4] https://pandoc.org/
[5] https://pandoc.org/
[6] https://pandoc.org/
[7] https://quartz.jzhao.xyz/
[8] https://docs.github.com/en/actions
[9] https://git-scm.com/
[10] https://pandoc.org/installing.html
[11] https://www.gnu.org/software/make/
[12] https://www.python.org
[13] https://quartz.jzhao.xyz/

- github's actions[14].

But note that knowing git and markdown are enough to contribute online through the github repository[15].

While most of those tools are standard (with the exception of quartz, but it relies itself on the standard Node[16] and `npm` technologies), we acknowledge that

1. It is challenging to understand that many different technologies,
2. We should strive to welcome contributions from collaborators not familiar with them,
3. Our set-up is unique in some respects.

This guide tries to alleviate some challenges resulting from this overall unique and diverse resource organization. For more details about our tools, please refer to the Installing dependencies and Repository Maintenance sections.

Locating Resources

To obtain the latest version of this resource, you can either

- visit the accompanying website princomp.github.io[17],
- download the latest version of the built resource[18],
- clone our repository[19].

This resource is an extension of csci-1301.github.io/[20], please refer to their user guide[21] for more information about it.

Editing Resources

If you are new to this project, first read through Contributing Guidelines[22] to learn how you can contribute to the improvement of this resource, and if applicable, how to join a contributing team.

[14] https://docs.github.com/en/actions
[15] https://github.com/princomp/princomp.github.io
[16] https://nodejs.org/
[17] https://princomp.github.io
[18] https://github.com/princomp/princomp.github.io/releases/download/latest/release.zip
[19] https://github.com/princomp/princomp.github.io/
[20] https://csci-1301.github.io/
[21] https://csci-1301.github.io/user_guide.html#locating-course-resources
[22] /contributing

Best practices for all forms of content

Inclusivity Follow the IT Inclusive Language Guide[23] from the University of Washington:

> use gender-neutral terms; avoid ableist language; focus on people not disabilities or circumstances; avoid generalizations about people, regions, cultures and countries; and avoid slang, idioms, metaphors and other words with layers of meaning and a negative history.

Typically, we recommend using

- "unethical hacker" instead of "black hat",
- "main" instead of "master",
- "blank space" instead of "white space",
- "display on the screen" instead of "printing", -etc.

In doubt, please start by referring to this list of problematic words and phrases[24].

Structure for accessibility

- All resources are titled
  - title each markdown document by having one (and only one) title at top level (that is, using #),
  - use subtitles when appropriate,
  - title all images with a descriptive title and add an alt-tag,
  - title all code blocks in labs and lecture notes.
- All resources are labelled when applicable, see content labelling for more details

Resources to assess accessibility:

- Affordable Learning Georgia's guide[25]
- Specific Review Standards from the QM Higher Education Rubric[26]
- UWG Accessibility Services's guide[27]
- Penn State's recommendations for alternative text and complex images.[28]

---

[23] https://itconnect.uw.edu/guides-by-topic/identity-diversity-inclusion/inclusive-language-guide/

[24] https://itconnect.uw.edu/guides-by-topic/identity-diversity-inclusion/inclusive-language-guide/#list

[25] https://alg.manifoldapp.org/projects/oer-accessibility-series-and-rubric

[26] https://www.qualitymatters.org/sites/default/files/PDFs/StandardsfromtheQMHigherEducationRubric.pdf

[27] https://docs.google.com/document/d/16Ri1XgaXiGx28ooO-zRvYPraV3Aq3F5ZNJYbVDGVnEA/edit?ts=57b4c82d#

[28] https://accessibility.psu.edu/images/

- WebAim Color Contrast Checker[29]
- WebAIM (Web Accessibility In Mind)[30]

Markdown Text documents are written using standard markdown syntax[31]. More precisely,

- in the `markdown+emoji` format, that is, in pandoc's markdown[32], using the emoji[33] extension[34]),
- using the pandoc-include[35] filter,
- and a custom[36] filter that sets all the code blocks[37], or all the code block and inline code[38]'s syntax highlighting to C# by default.

Because of the way the markdown is processed, please refrain from using the “ and ” characters: pandoc will automatically convert " into language-appropriate quotes for us.

Images

- Images belong in `source/img/` directory.
- Explain the image in written form.
- Title each image, this will create a URL for the image and enables linking to it.
- Always include a descriptive alt tag for accessibility.
- Do not rely on everyone seeing colors the same way[39].
- Prefer scalable vector images.
- When referring to images in markdown, use path from root, see example below

Syntax example. The quoted text is the alt tag and in parentheses is path to file

```
![“image of visual studio IDE”](./img/vs_ide.jpg){
 ↪  width=80% }
```

[29] https://webaim.org/resources/contrastchecker/
[30] https://webaim.org/
[31] https://commonmark.org/
[32] https://pandoc.org/MANUAL.html#pandocs-markdown
[33] https://pandoc.org/MANUAL.html#extension-emoji
[34] https://pandoc.org/MANUAL.html#extensions
[35] https://github.com/DCsunset/pandoc-include
[36] https://github.com/princomp/princomp.github.io/tree/main/source/templates/filters
[37] https://github.com/princomp/princomp.github.io/blob/main/source/templates/filters/default-code-class-block.lua
[38] https://github.com/princomp/princomp.github.io/blob/main/source/templates/filters/default-code-class-block-inline.lua
[39] https://www.wikiwand.com/en/Color_blindness

The `{ width=80% }` attribute is optional.

Images generated by LaTeX Some images are generated by LaTeX: the `.tex` file is what is used to generate the `.pdf` file, and then pdf2svg converts the `.pdf` into a `.svg` file. The `.svg` files are used in the `.html`, `.odt` and `.docx` documents, while the `.pdf` is used in the `.pdf` documents. The resulting images are added to the repository so that there is no need to re-compile them every time, or to set-up LaTeX and latexmk on each system.

UML class diagrams The UML class diagrams are created using Mermaid[40] and located in `source/uml`. To create a new class diagram, say for a `Documentation` class, follow those steps:

1. Create a `Documentation.txt` file in `source/uml` that follows the syntax for class diagrams[41] (note that there is no need to add `classDiagram` at the beginning, it will be done automatically),
2. Run (from the `source/` folder) `make uml/Documentation.md`,
3. Integrate the resulting drawing, properly captioned and with a link to your `Documentation.txt` file (for visually impaired readers, or to facilitate automatic processing) using `!include uml/Documentation.md`.

Source code

- Source code programs belong in `source/code/` directory.
- The code included in this directory should either be:
  - Placed in the `snippets/` sub-folder, and be a complete program.
  - Placed in the `projects/<solution>/<project>/` sub-folder, and contains a `Program.cs` file:
    * Go to `source/code/projets/`,
    * Create a subdirectory with the name of the solution you would like to use,
    * Create a subdirectory with the name of the project you would like to use,
    * Create a file called `Program.cs` in `source/code/projects/<solution>/<project>/Program.cs`
    * If you want to add additional classes, add them in `code/projects/<solution>/<project>/<Class>.cs` files.

    Do **not** add solution (`sln`) or project (`csproj`) files: they will be created automatically using the project and solution's name you specified (and a makefile rule similar

[40] https://mermaid.js.org/
[41] https://mermaid.js.org/syntax/classDiagram.html

to this one[42]), if multiple classes are present they will all be linked, and the resulting archive will be hosted at `content/code/projects/<solution>.zip`.
- Source code that is faulty, partial, or does not terminate can be included in markdown as inline code block.

Code snippets can be included in markdown documents using pandoc-include[43] filter:

```
```text
!include code/sample.cs
```
```

Note that for an unknown reason[44], no special characters (such as _) should be used in the filenames.

- Title each source code block included in markdown, this will create a URL for the code block and enables linking to it.
- code blocks are by default annotated as `csharp`
  - syntax highlighting is applied automatically at build time based on the code block language
  - to use a language other than C#, specify the language locally in the specific code block:
- only include code in text form such that it can be copy-pasted for reuse
- make sure to include blank lines before and after code blocks, since the absence of these can cause the code block to display incorrectly.

Tidying Source code CSharpier[45] is used to tidy the source code and make it uniform. Use

```
make tidy
```

to tidy all the source code present in the `source/code/` folders. The configuration file[46] is at `source/code/csharpierrc.yaml`.

Creating new lectures

Lecture notes belong to the `source/lectures/` directory.

To create a new lecture, for instance on exception handling:

[42] https://github.com/csci-1301/C-Sharp-project-maker
[43] https://github.com/DCsunset/pandoc-include
[44] https://github.com/DCsunset/pandoc-include/issues/45
[45] https://csharpier.com/
[46] https://csharpier.com/docs/Configuration

1. Create a directory corresponding to the theme if it does not exist already (say, `exceptions`), under `source/lectures/` directory
   - Follow the existing pattern for naming convention which is lowercase and separation by underscores.
   - At the root of this folder, create an `index.md` file (so, at `source/lectures/exceptions/index.md`) containing

     ```
     ---
     title: Desired Title for Theme
     ---
     ```

     so that your theme will be labeled "Desired Title for Theme" on the website's menu (see content labelling on how to further label it).
2. Under the directory corresponding to your theme, create a file named after the lecture's title (e.g., `exception-handling.md`) in lowercase. Write lecture notes in this file using markdown.
3. Edit the `source/order` file and insert where appropriate
   - `./lectures/exception/` (if you created a folder called `exception`),
   - `./lectures/exception/exception-handling.md` (which *must be* between `./lectures/exception/` and the next `./lectures/xyz/` folder).

   This last step will insure that your lecture is 1. included in the book, 2. sorted correctly on the website's menu (the default ordering is alphabetical).

If the lecture does not appear, here are the steps for troubleshooting the issue:

1. Check that after committing changes, the automated build has completed successfully, by checking the workflows[47],
2. The newly created lecture is under the subdirectory you picked in the `source/lectures/` directory[48],
3. The `.md` file exists,
4. Hard refresh the browser page if viewing the resources website

Known issues: When concatenating files pandoc may or may not include empty spaces between individual files. This may cause the subsequent lecture title to not appear in the generated book. For this reason, each lecture file should end with a newline.

[47] https://github.com/princomp/princomp.github.io/actions

[48] https://github.com/princomp/princomp.github.io/tree/main/source/lectures

Creating new labs

The process is very close to the process to create a new lecture, with the following exceptions:

- All lab resources are located under `source/labs/` directory, at root level (there is no "theme" sub-folder).
- You do not need to edit the `source/order` file, since labs are not included in the book nor sorted on the website.

Additionally, remember to:

1. Choose a short and unique name that describes the lab (say, `StringMethods.md`)
   - follow the existing convention for naming,
   - do not number labs or make assumptions about numbering because another instructor may not follow the exact same lab order,
   - make the lab standalone to support alternative ordering (avoid assumptions about what was done "last time"),
   - do not make assumptions about student using specific OS, include instructions for all supported options (Windows, MacOS, Linux),
   - do not make assumptions about student using Visual Studio, refer to IDE instead.
2. (optional) You can add a downloadable project (use a link of the form `[the Rectangle project](./code/projects/Rectangle.zip)`) or include snippets of code by following our instructions to add source code.

Using this established build system generates labs that are cross-platform (Windows, MacOS, Linux) and work on different IDEs (this process is documented in the corresponding repository[49]). Do not attempt to create labs locally as that approach does not have the same cross-platform guarantee.

Content Labelling

Technically Quartz[50] support a powerful tagging system[51] which should be leveraged. Markdown files can contain at their very top a YAML metadata block[52] containing, e.g.

```
---
tags:
```

[49] https://github.com/csci-1301/C-Sharp-project-maker
[50] https://quartz.jzhao.xyz/authoring-content#syntax
[51] https://quartz.jzhao.xyz/features/folder-and-tag-listings#tag-listings
[52] https://pandoc.org/MANUAL.html#extension-yaml_metadata_block

```
- Resource
---
```

to "tag" this resource with "Resource" so that it will appears in the tag listing[53]. To include multiple tags, simply make a list:

```
---
tags:
- Resource
- Guide
---
```

Conceptually We will follow the guidance provided on this page[54]:

- Use as Few Tags as Possible
- Limit Yourself to a Self-Defined Set of Tags
- Tags Within Your Set Must Not Overlap
- By Convention, Tags Are in Plural
- Tags Lower Case
- Tags Are Single Words
- Keep Tags on a General Level
- Omit Tags That Are Obvious
- Use One Tag Language
- Explain Your Tags

Styling and Templating

Templating files are under `source/templates/` directory. Templates directory contain layout files that are applied by pandoc when resources are built: note that the website's style uses a completely different mechanism.

For maintainability reasons it is preferable to apply templates during build time. This strategy makes it easy to edit templates later and apply those changes across all resources. Avoid applying templating to individual resource files whenever possible.

Currently templates directory contains the following:

- `docx/` - contains template used to produce `.docx` files (this template is not used yet, for size issues[55]).
- `filters/` - contains pandoc filters for annotating code blocks, configured to default to C#, which then allows applying syntax highlighting to all code block.

[53] https://princomp.github.io/tags/Guide
[54] https://karl-voit.at/2022/01/29/How-to-Use-Tags/
[55] https://github.com/csci-1301/csci-1301.github.io/issues/156

- `html/` - contains template used to produce *only the book.html file* (to edit the style of the website, refer to editing website)
- `latex` - contains templates used to produce `.pdf` files,
- `docx/` - contains template used to produce `.odt` files.

Updating docx template

Note that this template is not used yet, for (among other) size issues[56].

To edit this template, start by obtaining the default template file:

```
pandoc -o custom-reference.docx --print-default-data-file
↪  reference.docx
```

Then, open `reference.docx`, and, following loosely this tutorial[57], do:

- Click pretty much anywhere, and right-click on the highlighted style (displayed if you are under "Home", you may need to scroll down the styles),
- Change the font for everything but the source code,
- Click on the "Block code", then right-click on the highlighted style, and select the font for the source code,
- The font for "Verbatim Char" was also changed, but I am not sure if this has an impact,
- Make sure the fonts are embedded[58],
- Save and close the document.

This was inspired by this post[59] but does not seem to work properly.

Updating odt template

First, output the default template file:

```
pandoc -o custom-reference.odt --print-default-data-file
↪  reference.odt
```

Then, open `reference.odt`, and, following loosely this tutorial[60], do:

- Click on View, then Styles.
- Right-click on "Preformatted Text", click on "Modify…", and then select the desired font family for source code.

[56] https://github.com/csci-1301/csci-1301.github.io/issues/156

[57] https://support.microsoft.com/en-us/office/customize-or-create-new-styles-d38d6e47-f6fc-48eb-a607-1eb120dec563?ui=en-us&rs=en-us&ad=us

[58] https://support.microsoft.com/en-us/office/benefits-of-embedding-custom-fonts-cb3982aa-ea76-4323-b008-86670f222dbc

[59] https://stackoverflow.com/a/70513063

[60] https://github.com/jgm/pandoc/wiki/Defining-custom-DOCX-styles-in-LibreOffice-(and-Word)#libreoffice

- In the dialog or sidebar which opens make sure the button in the top panel marked with ¶ is highlighted (it is very subtle).
- In the menu at the bottom of the dialog/sidebar choose Applied Styles. Only "Default Paragraph Style" and "Footer" should appear.
- Right-click on "Default Paragraph Style", click on "Modify…", and then select the desired font family for the rest of the text.
- Then, highlight the A next to ¶.
- Right-click on "Source_Text", click on "Modify…", and then select the desired font family for source code.
- Click on File, then Properties, then on the Font tab, click on "Embed fonts in the document".
- Save and close the document.

Building locally

It is generally not necessary to build this resource locally unless the intent is to preview templating changes or to make changes to build scripts. For the purposes of editing content, it is sufficient to make edits to markdown files and commit those changes.

Installing dependencies To find the current list of dependencies needed to build this resource, refer to the build and deploy script install section[61]. The exact installation steps vary depending on your local operating system.

In general the following dependencies are needed:

- pandoc[62]
- texlive[63]
- make[64] and other standard unix utilities (such as sed or wget, all included in the Windows Subsystem for Linux[65]),
- python 3.+[66]
- packages and filters: Pygments[67], pandoc-include[68], texlive-xetex[69], texlive-latex-extra, lmodern, librsvg2-bin[70]
- symbola font

[61] https://github.com/princomp/princomp.github.io/blob/main/.github/workflows/build_and_deploy.yaml
[62] https://pandoc.org/installing.html
[63] https://www.tug.org/texlive/
[64] https://www.gnu.org/software/make/
[65] https://learn.microsoft.com/en-us/windows/wsl/install
[66] https://www.python.org/
[67] https://pygments.org/download/
[68] https://github.com/DCsunset/pandoc-include#installation
[69] https://tug.org/xetex/
[70] https://askubuntu.com/a/31446

For this later, note that starting with version 11[71], the licence is too restrictive for non-personal use. As a consequence, users are asked to make sure they do not use a version greater than v.10.24, which is "free for any use" and archived on-line[72] (curious users can also refer to the related webpage[73]). Note that installing this dependency using a unix-like package manager will result in installing a version of the font that is free to use in any context[74].

You can make sure you are currently using the latest version of panflute by running

```
pip install -U panflute
```

This is needed if running a recent version of pandoc (as of pandoc 3.1.6.1 at least).

Running the build

⚠ Warning

Running `make all` can be *very resource-incentive* and may render your system unstable. Read this section entirely before running any command.

Testing the installation After installing all dependencies, from the `source/` folder, run:

```
make
```

to display a list of useful rules.

It is recommended to first run a command building simple documents or copying files to test your installation, such as

```
make ../content/docs/about/credits.md
make ../content/docs/about/credits.pdf
make ../content/docs/about/credits.odt
make ../content/docs/about/credits.docx
make ../content/code/projects/Rectangle.zip
make ../content/web-order.ts
```

[71] http://web.archive.org/web/20181228102842/http://users.teilar.gr/%7Eg1951d/Symbola.pdf

[72] http://web.archive.org/web/20180307012615/http://users.teilar.gr/~g1951d/Symbola.zip

[73] http://web.archive.org/web/20180307012615/http://users.teilar.gr/~g1951d/

[74] https://metadata.ftp-master.debian.org/changelogs//main/t/ttf-ancient-fonts/ttf-ancient-fonts_2.60-1.1_copyright

```
make ../content/img/create_project_monodevelop.png
make ../content/fonts/hack/hack-italic-subset.woff
```

If this was successful, you can compile the resources needed for the website using

```
make build-light
```

Building all resources You can run

```
make -l 2.5 -j$(nproc --ignore=2) all
```

to create and populate the `content/` folder at root level with all the resources compiled. Note that this command limits the number of jobs in parallel and the number of CPU used (using this trick[75]), but that tweaking those values[76] may be needed to find the sweet spot on your own machine.

If you want to speed-up the compilation time, you can run

```
make fetch
```

which will fetch the latest build output, extract it and populate the `content/` folder using its content. Due to make's unique feature[77] only the files whose source was edited will be re-created when executing `make all` the next time, hence saving *a lot* of time. However, please not that files moved or deleted will still be present in the build.

Website

Editing the website

The website https://princomp.github.io/ is built from the `.md` files contained in the `content/` folder using a dedicated branch[78] of quartz[79]. To edit the layout, style, or other features such as the footer, please *start by checking out the quartz branch* (using `git checkout quartz`), and then

- Refer to quartz's website[80], repository[81] and general community,
- Knowing that multiple edits already tweaked its style.

A couple of indications about the edits made to quartz:

[75] https://stackoverflow.com/a/56607839
[76] https://stackoverflow.com/a/32487943
[77] https://makefiletutorial.com/
[78] https://github.com/princomp/princomp.github.io/tree/quartz
[79] https://quartz.jzhao.xyz/
[80] https://quartz.jzhao.xyz/
[81] https://github.com/jackyzha0/quartz

- The favicon at `quartz/static/`, and have been generated using https://realfavicongenerator.net/.
- The order in the menu is constructed using the `content/web-order.ts` file, itself generated from the `source/order` file in the main branch: refer to the makefile (again, in the main branch) for explanations on how this file is created, to the quartz documentation[82] for the main inspiration, and to the `quartz.layout.ts` and `sortFn.ts` files for the concrete implementation. If you change the order, setting

```
useSavedState: true, // To debug the explorer, change to
↪  "false" (this way, the menu is not cached /
↪  permanent),
```

to `false` in the `quartz/components/Explorer.tsx` file *may* help in refreshing the menu more easily.

- Other files edited or created include:
  - The files
    `quartz/components/AlternativeFormats.tsx`
    `quartz/components/styles/alternativeFormats.scss`
    list alternative formats at the top of the page,
  - The files
    `quartz/components/Comments.tsx`
    `quartz/components/scripts/darkmode.inline.ts`
    `quartz/components/Footer.tsx`
    `quartz/components/styles/listPage.scss`
    customize the footer and add a link to our repository feedback (while following the selected style[83]),
  - `quartz/styles/base.scss` loads a different set of fonts,
  - The files
    `quartz/components/Explorer.tsx`
    `quartz.layout.ts`
    tweak the menu and layout,
  - `quartz.config.ts` sets meta-data about the website,
  - `quartz/components/pages/404.tsx` customizes the 404 error message,
  - `quartz/plugins/emitters/assets.ts` emits the `.md` files (they are not available by default),
  - `quartz/components/index.ts` ties it all together.

Refer to Generate the git patch for instruction on how to generate a patch containing all the edits performed to our local copy of quartz.

[82] https://quartz.jzhao.xyz/features/explorer#use-sort-with-pre-defined-sort-order
[83] https://github.com/jackyzha0/quartz/issues/1161

Deploying locally the website

Follow closely those steps:

- Build the resource locally (note that running `make build-light` is enough to deploy the website).
- Move to the `quartz` branch by running

  ```
  git checkout quartz
  ```

  Note that the `content/` folder is still here, but that the source is absent from this branch: only files related to quartz are committed in this branch.
- Rename the `content/index.md` file (this is due to an annoying bug[84]) by running

```
mv content/index.md content/index_b.md
```

- Follow quartz's instructions[85]:
  - If you don't have at least Node v18.14 and npm v9.3.1, install node[86] and npm[87] (npm is probably installed automatically when you install node),
  - Run the following commands *at root level* (do *not* enter the `quartz/` folder):

  ```
  npm i
  npx quartz create
  ```

  for this last command, select

  | • Empty Quartz

  then,

  | • Treat links as shortest path ((default))

  - If the previous command succeeded, run

  ```
  mv content/index_b.md content/index.md
  ```

  to restore our index file, then

  ```
  npx quartz build --serve
  ```

  to start the server. Then, navigate to `localhost:8080/` to see the website deployed locally.

[84] https://github.com/jackyzha0/quartz/issues/1175
[85] https://quartz.jzhao.xyz/#-get-started
[86] https://nodejs.org/en/download/package-manager
[87] https://github.com/npm/cli?tab=readme-ov-file#installation

Updating quartz

Our local copy of quartz, in the `quartz` branch[88], is "frozen" in the sense that it corresponds to the development of quartz at a point of time. It is possible to

1. Save the edits made to our local copy (as a git patch[89]),
2. Pull the current version of quartz in a different branch (called `quartz-update`),
3. Apply our edits to this updated version of quartz,
4. Replace the `quartz` branch with the `quartz-update` branch to deploy the updated version of quartz with our edits.

This process is not without risks and requires to be able to deploy locally the website to test it before deploying it. The following guide was inspired by this discussion[90].

Generate the git patch The first step is to save as a git patch all the edits that have been made on our local copy of quartz since it was last updated.

- Make sure you are

  1. At root level in your repository's copy,
  2. In the `quartz` branch,
  3. That your branch is up-to-date.

  by running a command such as

  ```
  pwd && git checkout quartz && git pull
  ```

- Locate the commit (short) `id` of the last commit performed by quartz maintainer. A way of achieving this is to look for "PCP" in the commit messages, using

  ```
  git rev-parse --short :/PCP
  ```

  and then to look for the commit id of the commit that came *before* it. For instance, if the previous command returns `847e3356`, then the command

  ```
  git rev-parse --short 847e3356^1
  ```

  will return information about the commit that came before that last commit: we will assume its (short) id is `3b74453f` in the following.

[88] https://github.com/princomp/princomp.github.io/tree/quartz
[89] https://git-scm.com/docs/git-apply
[90] https://github.com/jackyzha0/quartz/issues/1145

Visual inspection using github's interface[91] or a program such as gitk[92] can facilitate this process. Note that removing the `--short` option will give the *long* version of the id, which may be harder to compare.

- Use the (short) id previously obtained to generate a patch containing all the changes made since that commit:

  ```
  git diff-index 3b74453f --binary > pcp_quartz_patch
  ```

  The `--binary` option insures that any file created will be included in the patch: as a result, this file can be heavy.

- Make sure this `pcp_quartz_patch` file is located at the root level in your repository's copy but do not commit it to the repository.

Clone the latest version of quartz Execute the following commands:

```
git remote add quartz
 ↪  https://github.com/jackyzha0/quartz.git
git fetch quartz
git checkout -b quartz-update quartz/v4
```

where `quartz-update` is the name we use for our branch, and `quartz/v4` is the name of the branch in the quartz repository we want to copy.

Apply the git patch There are two ways of applying the patch. First, make sure you are in the `quartz-update` branch by executing

```
git rev-parse --abbrev-ref HEAD
```

Then follows the first method if possible.

Using apply First, check if the `pcp_quartz_patch` patch is applicable, by executing

```
git apply --ignore-space-change --ignore-whitespace
 ↪  --check --reject pcp_quartz_patch
```

Some sections of the patch may be rejected: make sure you take note of which file will need to be merged by hand. Then, apply the patch, using

```
git apply --ignore-space-change --ignore-whitespace
 ↪  --reject pcp_quartz_patch
```

[91] https://github.com/princomp/princomp.github.io/commits/quartz/
[92] https://git-scm.com/docs/gitk

Then look for the `.rej` files: they will contain the edited version of a file that you will need to merge manually with the updated version of the same file from quartz's update.

Using patch If `git apply` gave an error starting with

```
Checking patch quartz.layout.ts...
error: while searching for:
```

then, instead, do

```
patch -p1 < pcp_quartz_patch
```

And look for the `.rej` files as described above. Note that using this technique *requires to copy the binary files by hand*. Indeed, you should receive warning messages like

```
File quartz/static/android-chrome-192x192.png: git binary
↪  diffs are not supported.
```

and those files will have to be copied by hand from another branch, and / or re-added to the repository.

Testing Once you are done manually merging, **test** your updated version by deploying locally the website and making sure that quartz does not return any error. If everything looks ok, add all the new files and commit the edits using a message containing the "PCP" string (to facilitate future generation of git patch), and push, using for example:

- First, use `bash git add --all -n .` to list all the files you are about to add: make sure you are not adding files from the `content/` folder, for instance. If everything looks fine, proceed to the next step.
- Then, actually add the files, commit, and push, using:

  ```
  git add --all
  git commit -a -m "Applying previous PCP patch."
  git push origin quartz-update
  ```

Update the branch **If you were able to fix all the conflicts and to check that the website could still be deployed locally**, then overwrite the `quartz` branch with the `quartz-update` branch, by executing[93]:

```
# Make sure your working tree is in a clean state
git status
```

[93] https://www.reddit.com/r/git/comments/bqx85v/comment/eo8j4zh

```
# Check out the branch you want to change, e.g.
 ↪  some-branch
git checkout quartz

# Reset that branch to some other branch/commit, e.g.
 ↪  target-branch
git reset --hard quartz-update
```

If the deployment was successful and everything seems to be working, you can delete the quartz-update branch, locally then remotely, by executing

```
git branch -D quartz-update
git push -d origin quartz-update
```

Repository Maintenance

This repository uses following tools and technologies:

- git - version control
- Github - to make source code available on the web
- markdown, LaTeX - for writing the resources
- pandoc - for converting documents to various output formats
- make - for specifying how to build this resource
- github actions - to automatically build the resource
- github pages - to serve the accompanying website
- additional packages for specific tasks: texlive, Pygments, pandoc filters, lua filter[94], etc.
- fonts-symbola - to produce the emoji and other symbols in the pdf document.
- utteranc.es[95] - for feedback through website
- csharpier[96] - to tidy the C# source code

Build outputs

The resource material is organized into specific directories inside the `source/` folder. These resources are then compiled into templated documents in various formats using pandoc[97]. The makefile explains the exact steps applied to each type of resource.

[94] https://github.com/jgm/pandoc/issues/2104
[95] https://utteranc.es/
[96] https://github.com/belav/csharpier
[97] https://pandoc.org/MANUAL.html

Github actions

This resource is built automatically every time changes concerning files in the `source/` folder are committed to the main branch of the repository. This is configured to run on Github actions[98]. The workflow[99] that is automatically triggered has two jobs: one to build the resource, and one to deploy it.

Currently Github actions offers unlimited free build minutes for public repositories (and 2000 min/mo. for *private* repositories, should we ever need them), which hopefully continues in perpetuity (if it does not there are other alternative services). Going with one specific CI service over another is simply a matter of preference.

Following a successful build, the build script will automatically deploy the generated resources to an accompanying website hosted on github pages[100].

Fetch and No Fetch Versions There is a second workflow[101] that is identical to the first one with one important exception: to speed up compilation, `build_and_deploy.yaml` uses `make fetch` to speed up compilation time by re-downloading the latest build output, and then compiling only the required files. This can sometimes complicate the propagation of changes, typically if a template is modified (as this does not triggers a re-compilation of the files using it currently) or if a file is renamed (as the previous version will not be deleted).

The build_and_deploy_no_fetch.yaml[102] can be triggered manually[103] to force a "fresh" remote compilation.

Creating releases

Currently a github action is setup to do the following: whenever a new commit is made to the main branch, the action will build the resource and add the generated resources as a pre-release[104] and tag them as

[98] https://github.com/features/actions

[99] https://github.com/princomp/princomp.github.io/blob/main/.github/workflows/build_and_deploy.yaml

[100] https://pages.github.com/

[101] https://github.com/princomp/princomp.github.io/blob/main/.github/workflows/build_and_deploy_no_fetch.yaml

[102] https://github.com/princomp/princomp.github.io/blob/main/.github/workflows/build_and_deploy_no_fetch.yaml

[103] https://github.com/princomp/princomp.github.io/actions/workflows/build_and_deploy_no_fetch.yaml

[104] https://github.com/princomp/princomp.github.io/releases

"latest"[105]. If a subsequent commit occurs it will overwrite the previous latest files and become the new latest version. This cycle continues until maintainers are ready to make a versioned release (or "package").

Making a versioned release is done as follows:

1. Go to repository releases[106]
2. Choose latest, which contains the files of the latest build
3. Edit this release, giving it a semantic name and a version, such as v1.0.0. Name and version can be the same. (cf. semantic versioning[107])
4. Enter release notes to explain what changed since last release
5. Uncheck "This is a pre-release"
6. Check "Set as the latest release"
7. Update release

Following these steps will generate a new, versioned release. The versioned releases will be manually uploaded to and archived on galileo.

Once this is done, remember to create the next pre-release:

1. Go to the repository releases[108].
2. Click on "Draft a new release".
3. Pick the tag "Latest".
4. Click on "Generate release notes"
5. Check "This is a pre-release"
6. Click on "Publish release"

Maintaining repository feedback

Resource users can submit feedback about the resource through various means, one of which is leaving comments on the website. This feature is enabled by utteranc.es[109], using repositories hosted by the `princomp` github organization[110].

To manage user feedback over time, a semester-specific repository is created for issues only. This must be a public repository and located under the same organization as the resources repository. utteranc.es widget is configured to point to this repository. After a semester is over, this feedback repository will be archived, and a new one created for the next semester. This will simultaneously archive all older issues and reset the feedback across website pages.

[105] https://github.com/princomp/princomp.github.io/releases/tag/latest
[106] https://github.com/princomp/princomp.github.io/releases
[107] https://semver.org/
[108] https://github.com/princomp/princomp.github.io/releases
[109] https://utteranc.es/
[110] https://github.com/princomp

Migrating feedback repository The steps for migrating feedback target repository are as follows:

1. Create a new **public** repository under `princomp` github organization[111]. Follow the established naming convention (`feedback-<fall|spring|summer>-<YYYY>`), and leave all the options except for visibility (which needs to be set to public) by default.
2. Go to repository Issues (make sure issues is enabled in repository settings).
3. Create a new label whose *label name* is `comment` (to match widget configuration as indicated in `quartz/components/Footer.tsx`, in the `quartz` branch).
4. Go to `Organization Settings > Installed GitHub Apps`[112].
5. Choose "utterances" > "configure"
6. Under "Repository access" > "Only select repositories"
   - Select the repository created in step 1.
   - Remove the previous semester feedback repository.
   - Save.
7. In `princomp/princomp.github.io/` repository, in the `quartz` branch[113], open `quartz/components/Footer.tsx`
8. Update utteranc.es widget code to point to the new feedback repository created in step 1.

   ```
   <script data-external="1"
           src="https://utteranc.es/client.js"
           repo="princomp/{REPOSITORY_NAME}"
           label="comment" …>
   </script>
   ```
9. Commit change to `quartz/components/Footer.tsx`
10. Make sure the feedback works after migration. If it does not, retrace your steps.
11. Archive the earlier feedback repository in its settings.

[111] https://github.com/organizations/princomp/repositories/new
[112] https://github.com/organizations/princomp/settings/installations
[113] https://github.com/princomp/princomp.github.io/blob/quartz/quartz/components/Footer.tsx

Maintaining Instructors / G/UCA rights

This is handled by the `csci-1301` github organization[114] and documented at https://csci-1301.github.io/user_guide.html#maintaining-instructors-guca-rights.

[114] https://github.com/csci-1301