Creating a web manual using MkDocs

some intro about system...

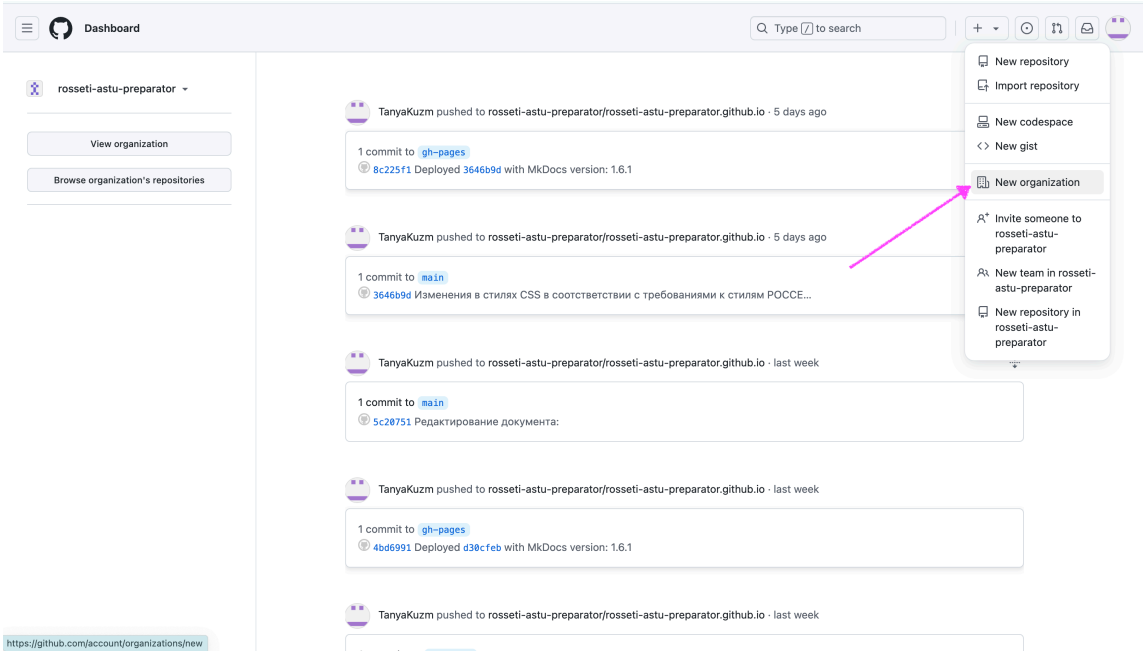
Links

- This guide is based on the official MkDocs documentation – visit the https://www.mkdocs.org/.
- If you have any questions, feel free to contact me via email: tedkuzmina@gmail.com.
- Please leave your feedback or suggestions about this document My LinkedInn.
- You can view the result of our work Our site.
- The source files are available My GitHub.
- You can view the instructions for the Material Material for MkDocs.

Step 1: Creating an organization in GitHub

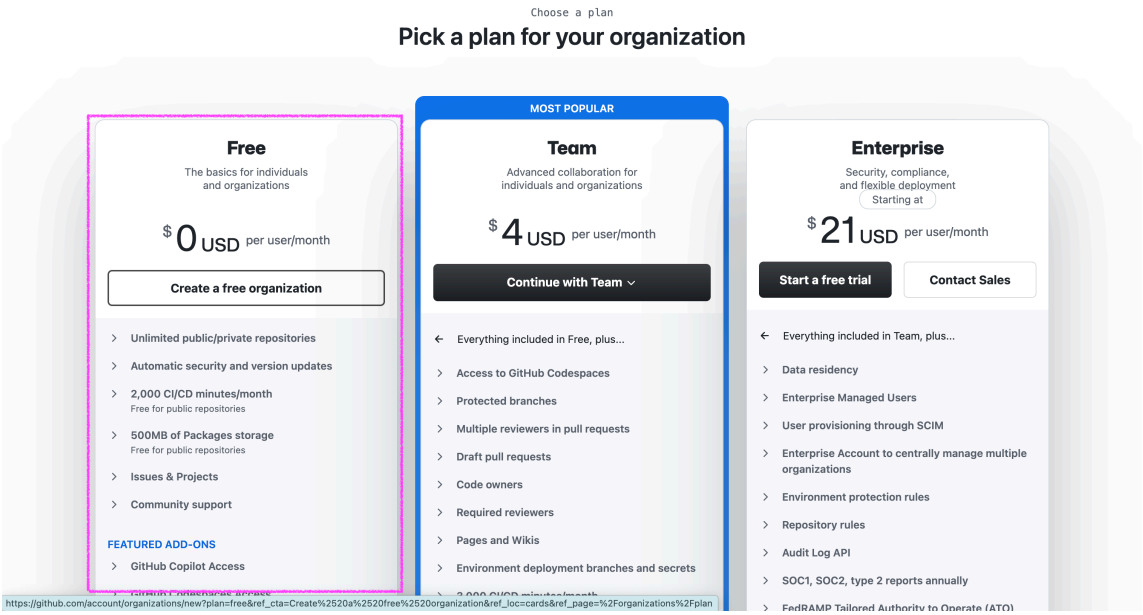
Create an organization:

- Log in to GitHub.
- Select the "Create an organization" menu.



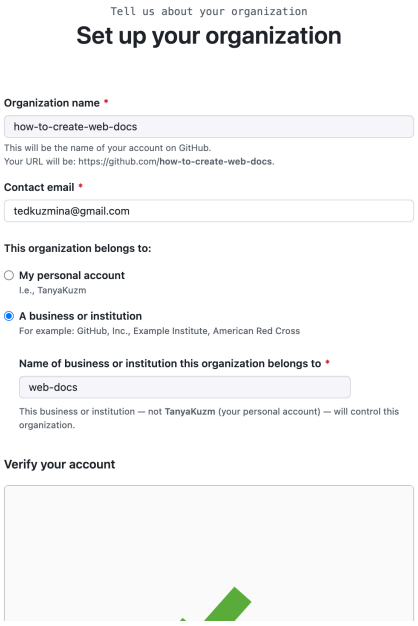
Select a pricing plan:

- Choose an appropriate pricing plan for your organization.



Set up the organization:

- Fill in the necessary settings.

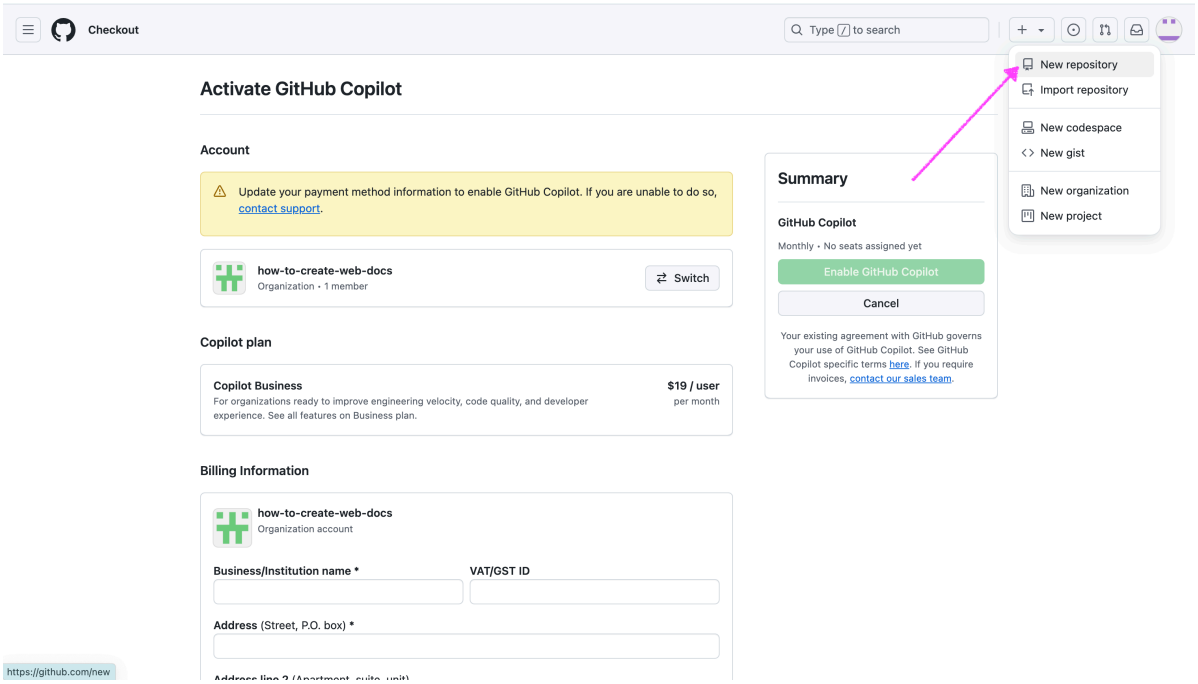


- Confirm the creation.

➤ The URL of your future site will be generated from the "Organization name" field and will follow this structure: https://[site-name].github.io/.

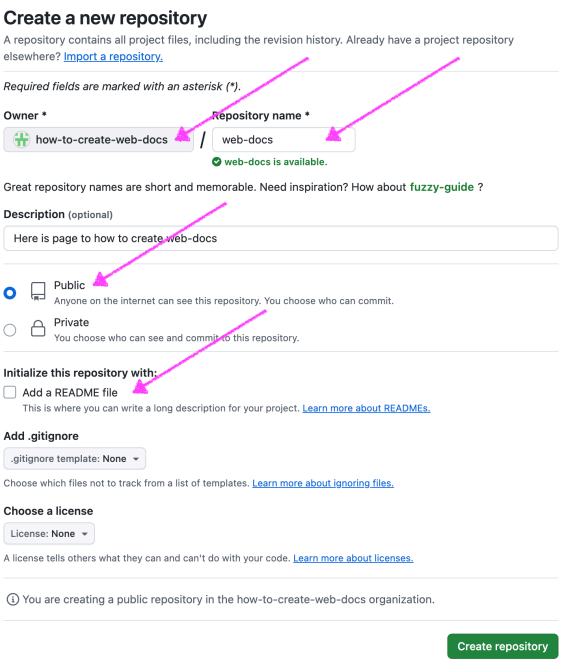
Step 2: Create a new repository

Create a repository:



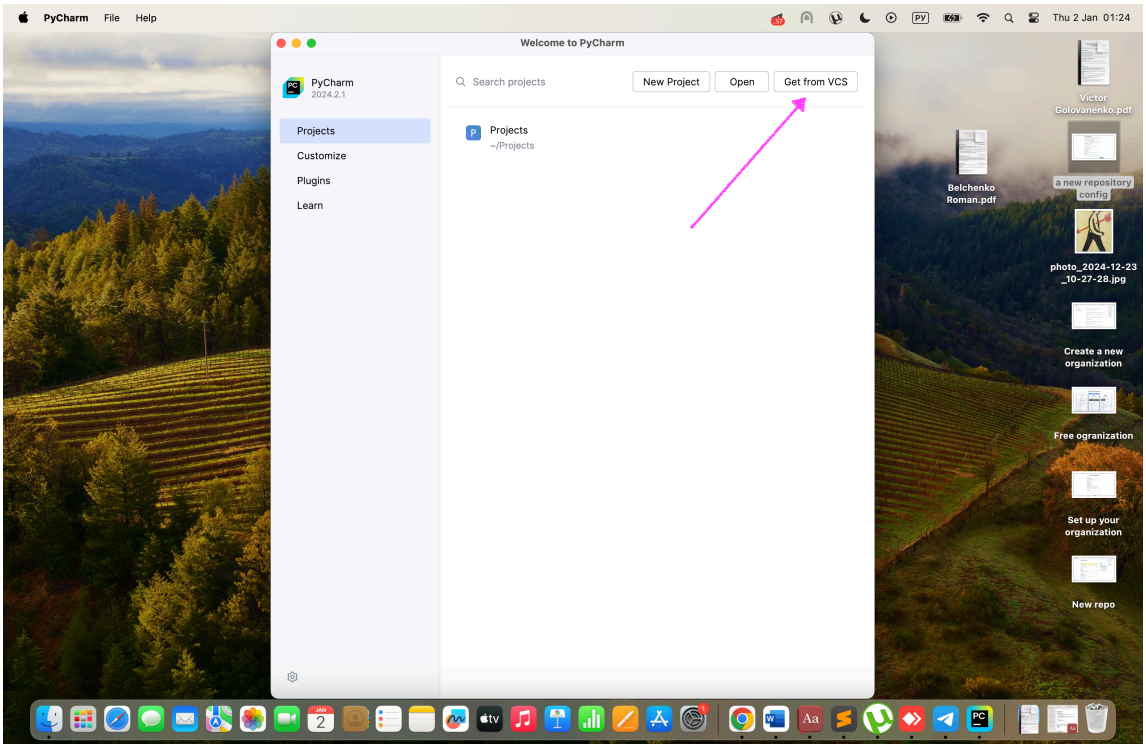
➤ Create a repository on behalf of the organization.

➤ Select public/empty (the repository should not contain, for example, README.md).

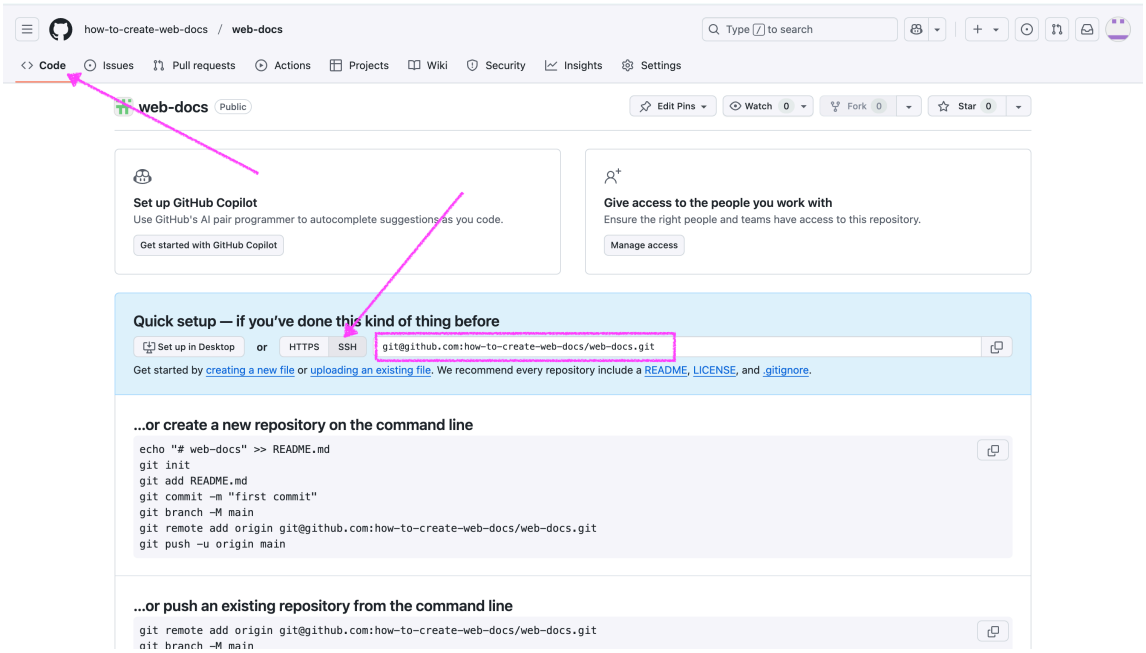


Clone the repository in PyCharm:

- Copy the repository URL (either HTTPS or SSH).
- In PyCharm, select "Clone Repository" or "Get from VCS".



- Copy the URL from your organization's page in GitHub (I choose SSH).



- Paste the URL and click "Clone".

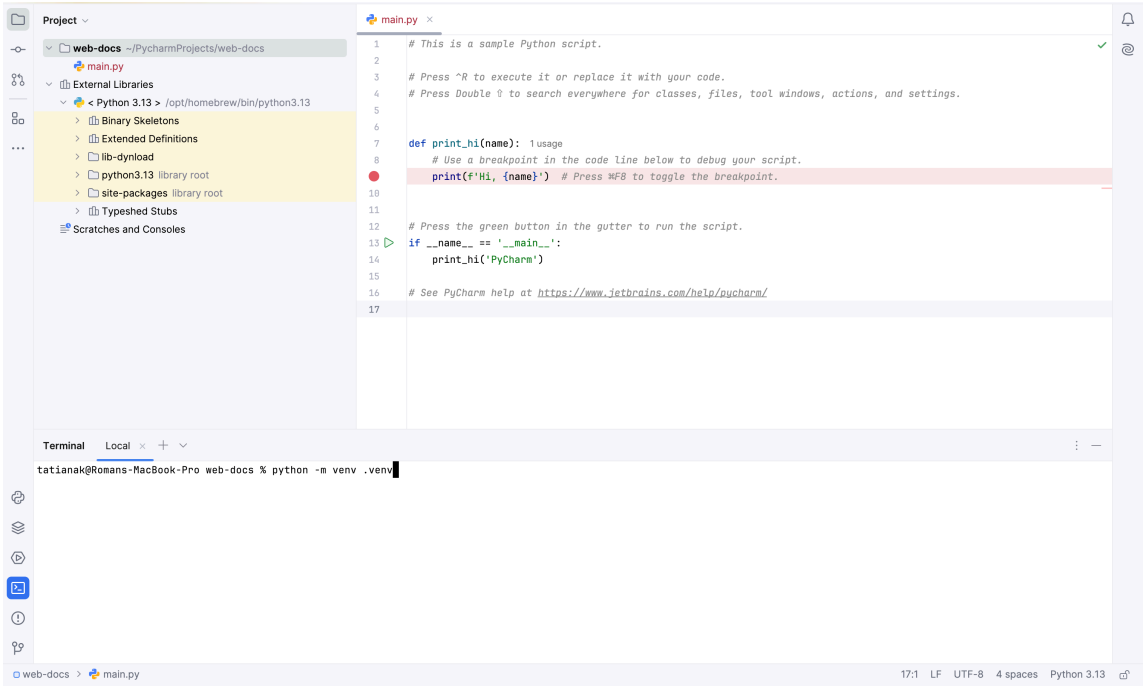
➤ If you choose SSH, make sure you have SSH settings configured. We'll discuss why this is more convenient in later posts.

- Click “Trust Project” in the window that opens.

Step 3: Setting up a virtual environment

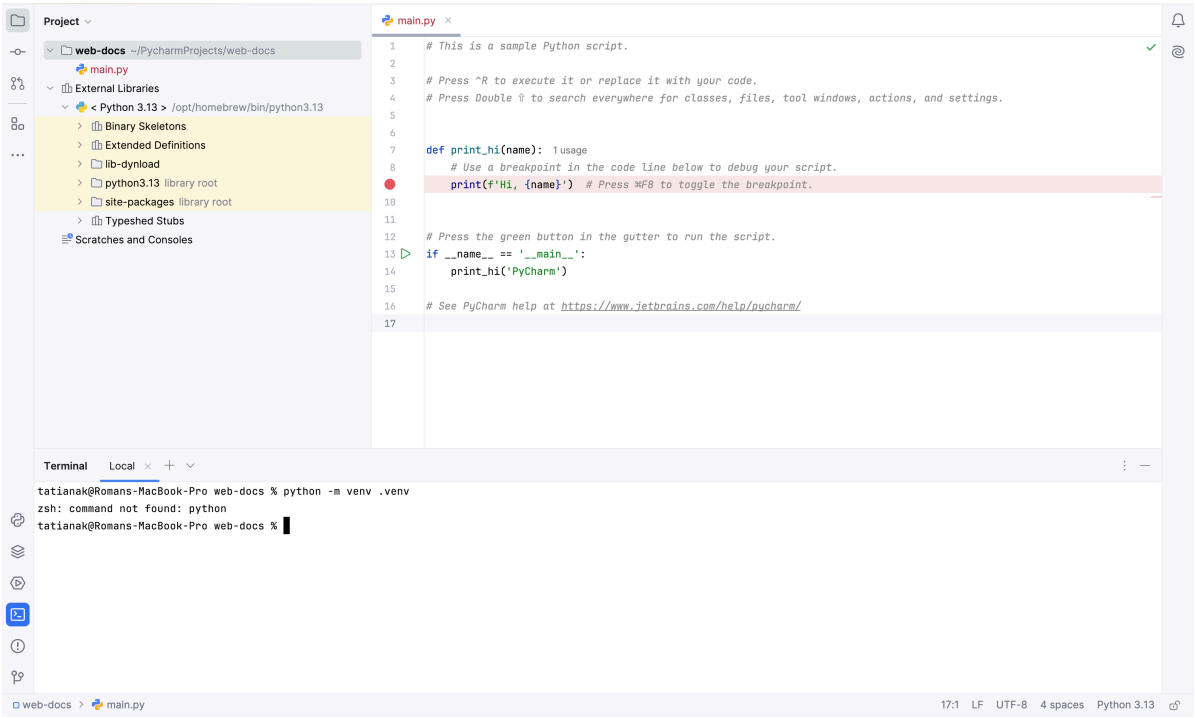
Create a virtual environment:

- Run the following command in the PyCharm terminal:

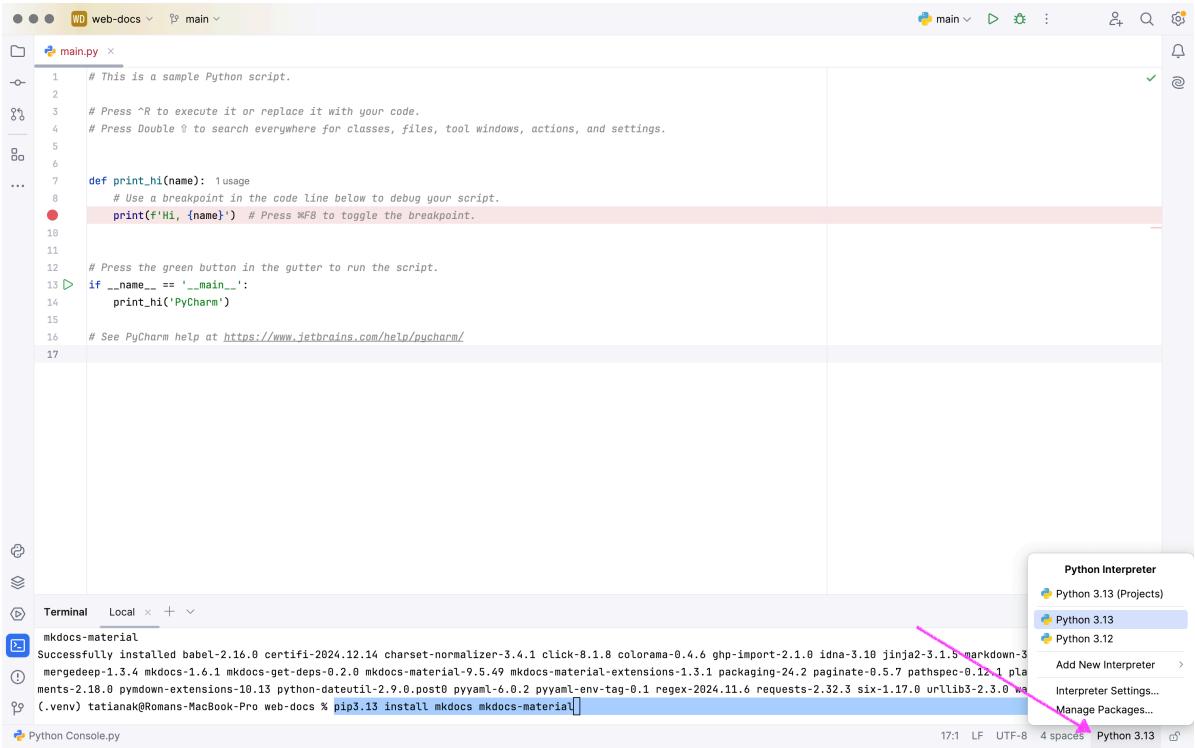



```
python -m venv .venv
```

➤ If you get a "command not found" error, check your installed Python version and adjust the command accordingly.



Your Python version:



For example, I used: `python3.13 -m venv .venv`

➤ You can see the PyCharm version in the lower right corner of the screen.

➤ Setting up a virtual environment helps avoid conflicts between packages in different projects.

➤ It's not possible to install two or more different versions of the same package in one environment. The solution is to create another environment.

Activate the environment:

- In the PyCharm terminal, run: `source .venv/bin/activate`

Step 4: Install MkDocs and the necessary packages

Install via pip:

- Run the following in the terminal: `pip install mkdocs mkdocs-material`

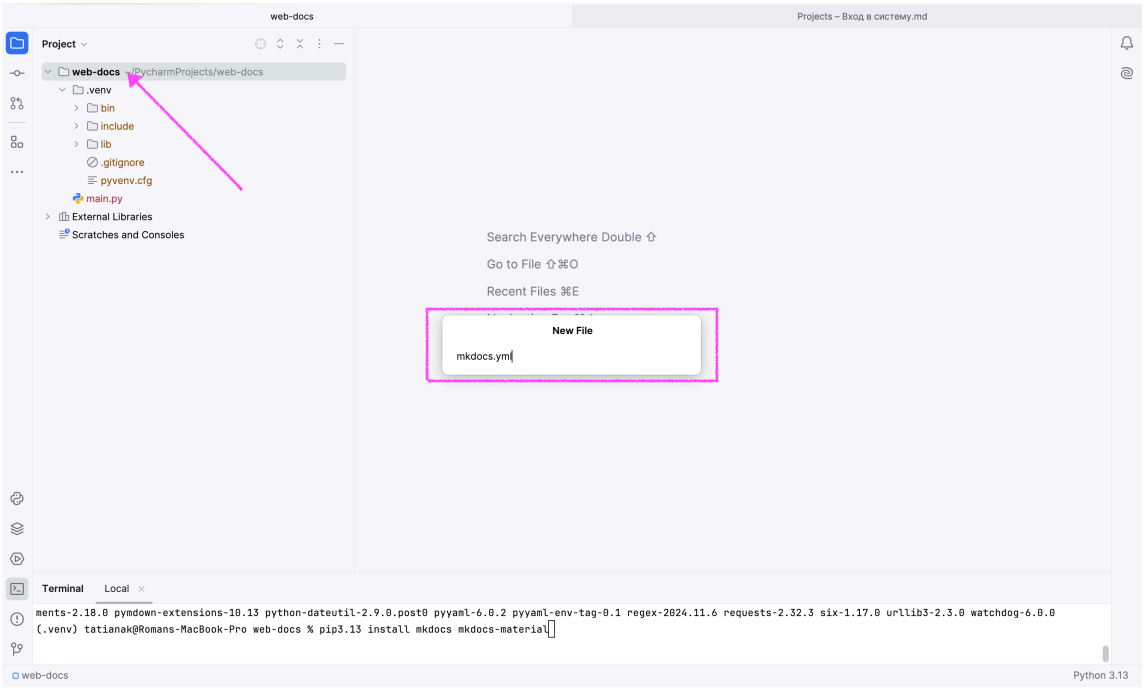
➤ If needed, specify the version with: `pip3.13 install mkdocs mkdocs-material`

- Alternatively, use the "Packages" toolbar in PyCharm.

Step 5: Create MkDocs Configuration (mkdocs.yml)

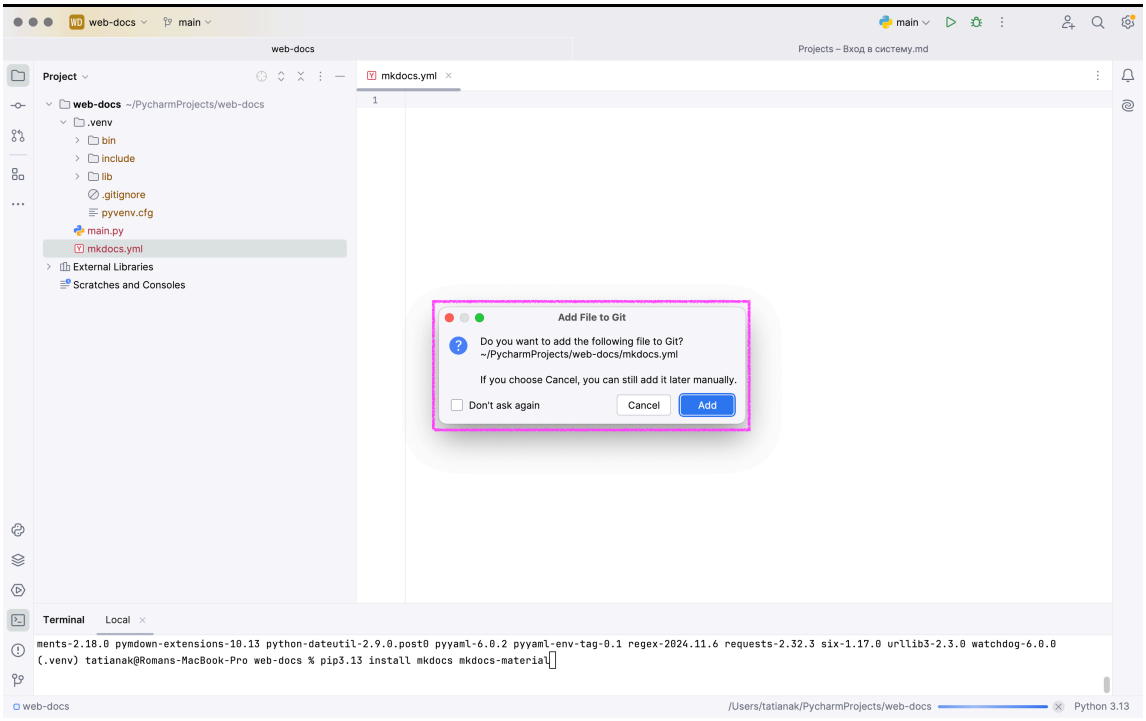
Create the `mkdocs.yml` file:

- In the root of the project, right-click the folder and choose "Create File".
- In the window that appears, enter the filename `mkdocs.yml`.



➤ To create a Yaml file, just add the `.yml` extension to the file name.

- Add the file to Git.



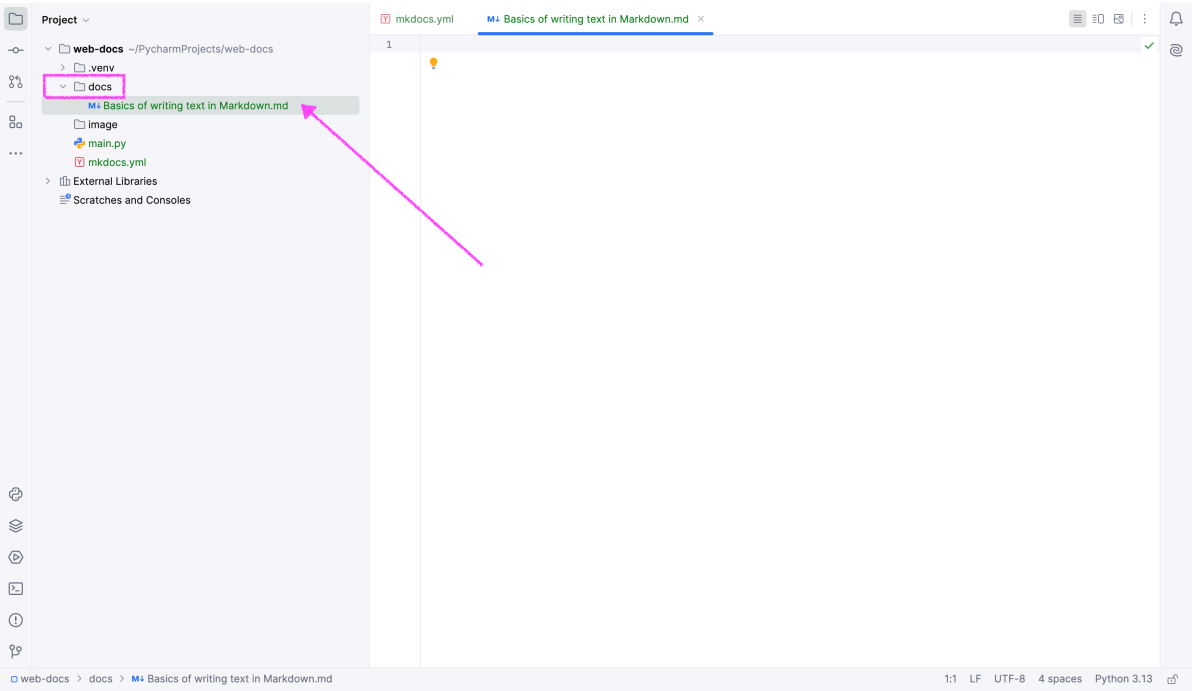
  ➤ After this step, you will have a `mkdocs.yml` file in your project tree.

- **Create a docs folder** for storing the source text files.
- **Create an `image` folder** for storing images.

As a result, your project structure will be scalable and look like this:

```
Web-docs/ # Parent directory
├── .venv/ # Virtual environment of the project
├── docs/ # Documentation source files
├── image/ # Documentation images
├── mkdocs.yml # MkDocs configuration file
```

Next, within the docs folder, create your first Markdown file.



➤ To create a Markdown file, just add the `.md` extension to the filename and start adding content according to the rules described below.

Customize your config (mkdocs.yml)

As a result of editing, your config may have the following structure.

```
# The name of your site, displayed in the page header
site_name: Web-docs using MkDocs

# The URL of your site after deployment (essential for correct link
functionality)
site_url: https://MkDocs-web-docs.github.io/how-to-create-web-docs/

# Specify an additional [CSS](# "Cascading Style Sheets: a style language for
describing the appearance of HTML documents, including colors, fonts, indents,
and other visual aspects") file for custom styles
extra_css:
  - extra.css

# Navigation (site page structure)
nav:
  - Creating a web manual using MkDocs: index.md    # The main page of the site
  - Links: Links.md                                 # Link to the Links page
  - Project Setup: Project Setup.md                 # Link to the Project Setup
page
  - Documentation Setup: Documentation Setup.md     # Link to the Documentation
Setup page
  - Deploy and Customize: Deploy and Customize MkDocs.md # Link to the Deploy and
Customize page

# Site theme and localization
theme:
  name: mkdocs       # The theme being used (in this case, the standard MkDocs
theme)
  locale: ru         # Localization (Russian language)


# Step 6: Create Documentation in Markdown Format
```

> You need to include index.md in your navigation.

Basics of writing text in Markdown

Markdown is a simple markup language for formatting text. It allows you to create structured text using simple symbols.
Here are the main features:

1. **Headings**
   Use the `#` symbol to create headings of different levels. The more `#` symbols, the lower the heading level.
   Example: ```markdown # Heading Level 1 ## Heading Level 2 ### Heading Level 3
2. **Paragraphs and Line Breaks**

3. To separate paragraphs, leave a blank line.
4. To force a line break, add two spaces at the end of a line.

Example:

```
This is the first paragraph.

This is the second paragraph.
And this is the second line without a paragraph.
```

1. **Text Formatting**
2. Bold: Wrap text in double asterisks (**) or double underscores (__).** Example:**

```
**Bold text**
__Bold text__
```

- Italics: Wrap text in single asterisks (*) or single underscores (_). Example:*

```
*Italics*
_Italics_
```

- Strikethrough: Use double tildes (~~). **Example:**

```
~~Strikethrough text~~
```

1. **Lists**
2. Bulleted list: Use -, , or +. Example:*

```
- Item 1
- Item 2
  - Sub-item
  ```
* Numbered list: Use numbers with a period.
Example:
```markdown
1. First
2. Second
```

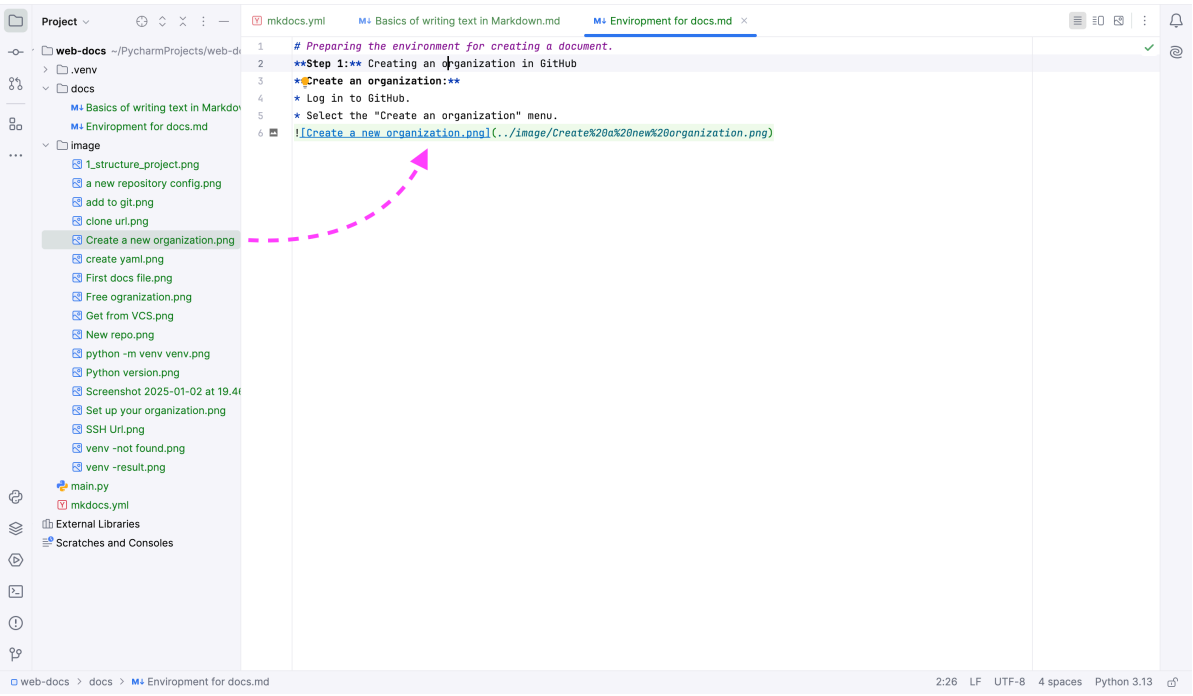
1. **Links**

```
[Link text](URL)
```

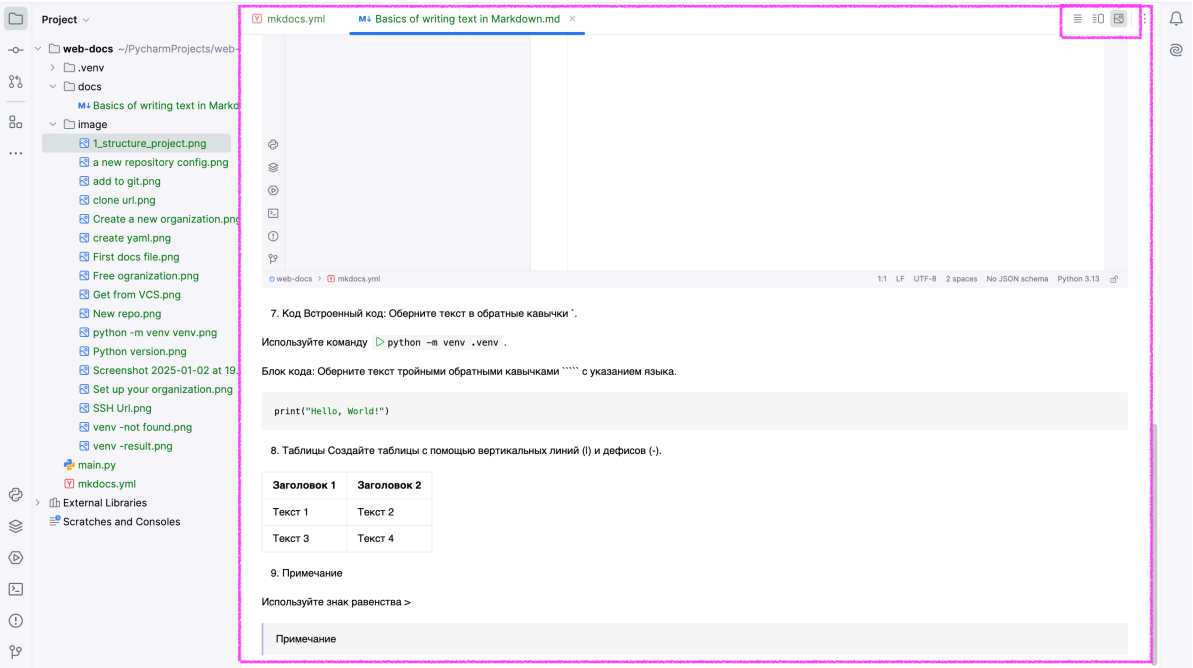
1. **Images**

```
![Image name](../image/1_structure_project.png)
```

➤ To add an image, you don't need to specify the path manually. Just drag the image from the left menu to the desired location, and PyCharm will automatically generate the path.



➤ PyCharm has three document display modes, including a preview with images.



7. **Code** * Inline code: Wrap the text in backticks (`). **Example:**

```
`python -m venv .venv`
```

- Code block: Wrap the text in triple backticks (`````) followed by the language. **Example:**

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

1. **Tables** Create tables using vertical bars (|) and hyphens (-). **Example:**

```
| Heading 1 | Heading 2 |
|-----------|-----------|
| Text 1    | Text 2    |
```

1. **Note** Use the `>` symbol to create notes. **Example:**

```
> Note: Always test your code before deploying.
```

More about numbering

To ensure that mixed numbering is displayed correctly in MkDocs (or any Markdown document), it is important to use proper indentation and formatting. Here is an example of how to do it:

> !Indentation must be a multiple of 4 spaces.

1. First item
    - Unnumbered item inside a sub-item
        - Another unnumbered item
2. Second item
    - Unnumbered item in main list
        - Nested unnumbered item
            - Deeper nested unnumbered item

> In **MkDocs**, as in **Markdown**, the correct display of nested numbering depends on the theme used and the way Markdown is interpreted. For standard Markdown, automatic support for "sub-items with nested numbers" (e.g. 2.1.1) may not be available, since Markdown does not support multi-part numbering by default.

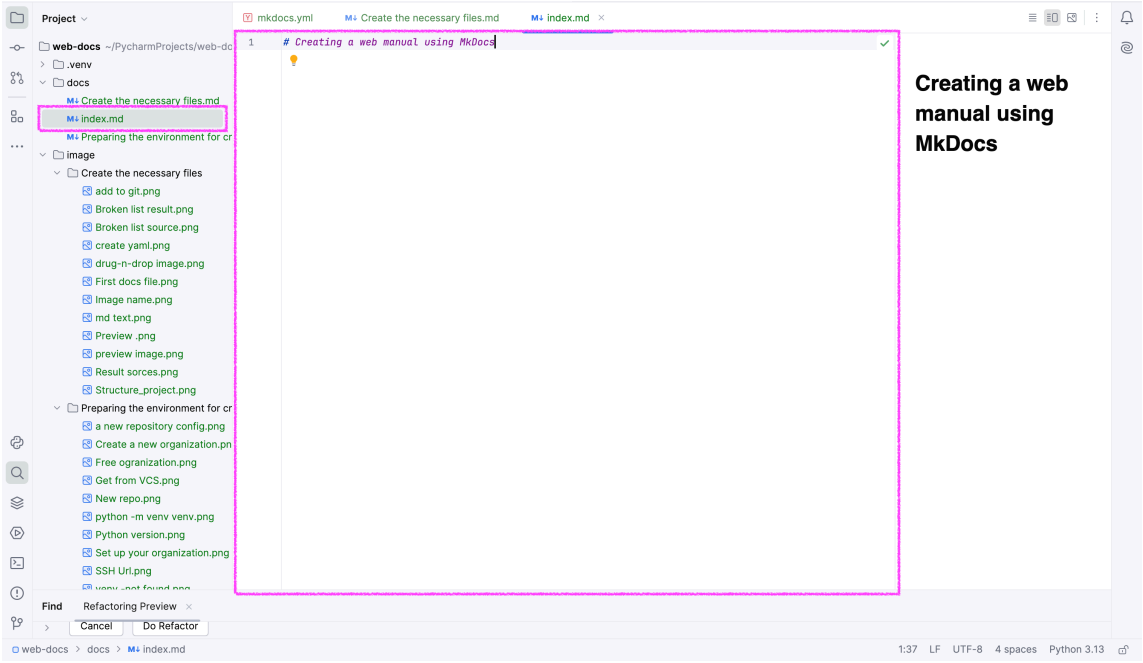
Later we will look at working with other, more flexible, themes.
```

> In some cases, the numbering may be broken. To fix this, I use HTML to continue the numbering:

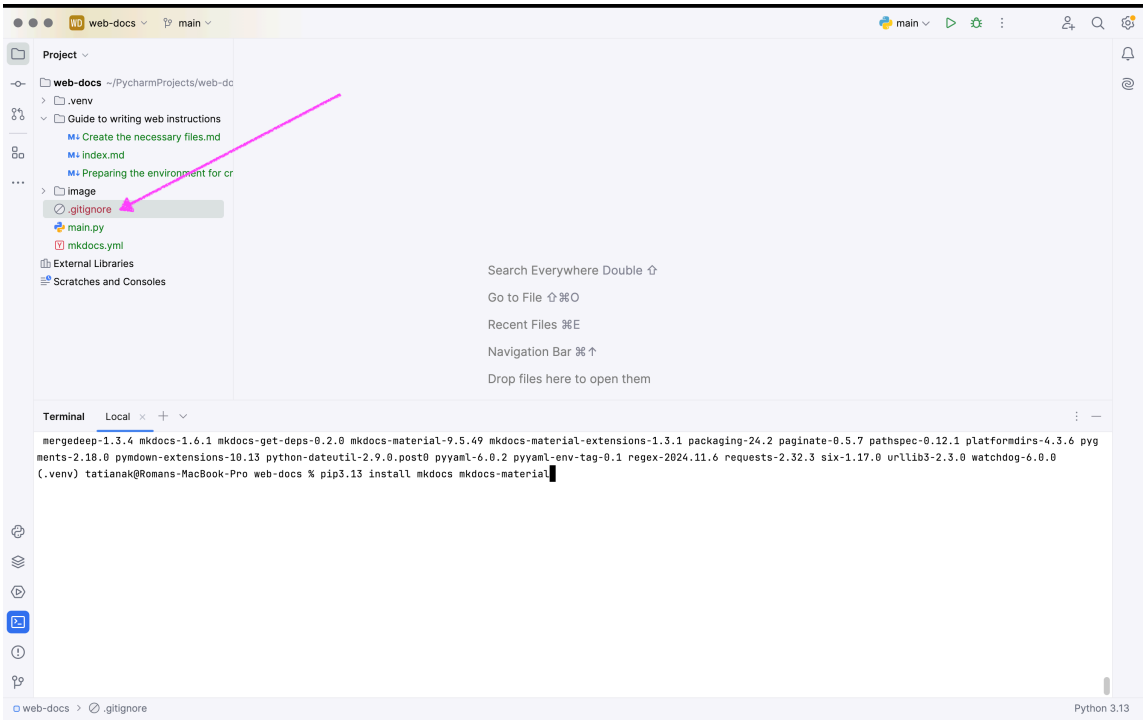
```
<ol start="5">
  <li>Links:</li>
  <a href="URL">Link text</a>
</ol>
```

Step 7: Complete the Structure of Your Project

1. In the source folder, create the file `index.md` — the text of this document will be displayed on the home page of the site.



2. Create the `.gitignore` file in the root of the project and write `site` in it — this will prevent Git from tracking the compiled files.



➤ The "site" folder will be automatically generated during deployment.

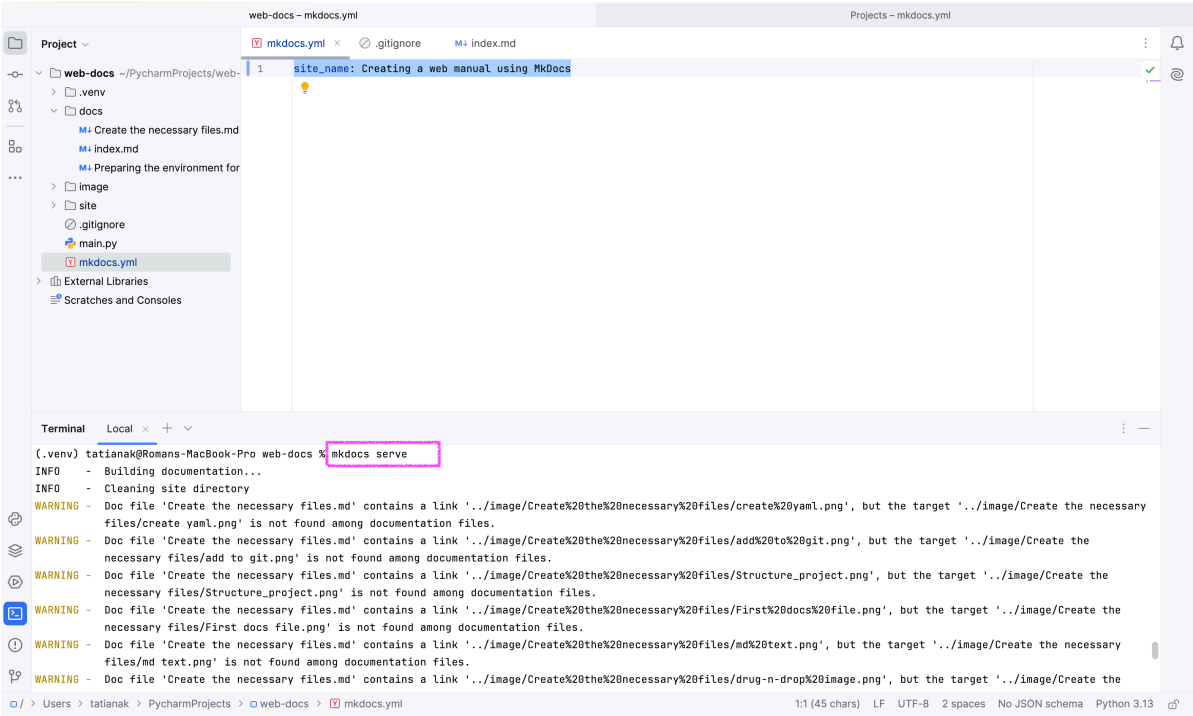
Now, your project structure looks like this:

```
Web-docs/  # Parent directory
├── .venv/  # Virtual environment
├── docs/  # Documentation source files
    ├── index.md/  # Home page
├── image/  # Documentation images
├── mkdocs.yml  # MkDocs configuration file
├── .gitignore  # Ignored files
```

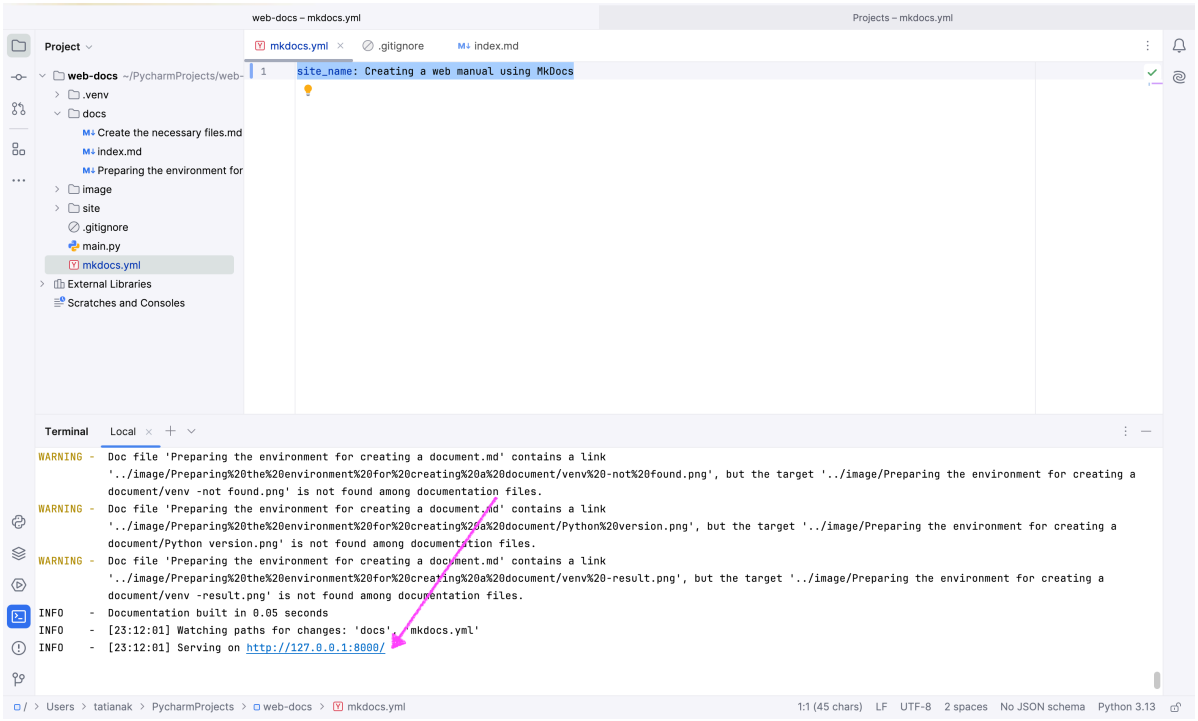
Step 8: Deploy Documentation to GitHub Pages

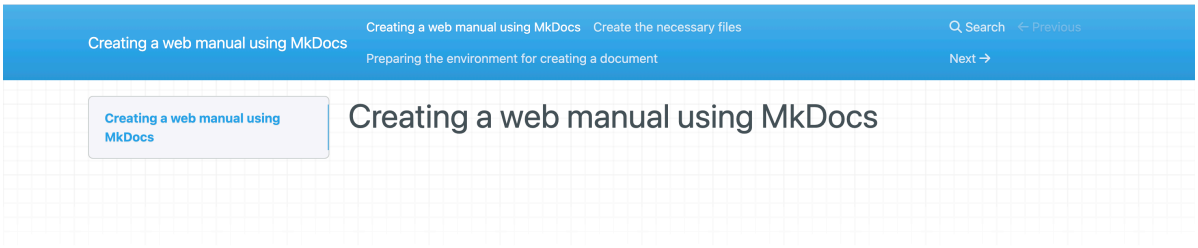
Build and preview the documentation using the MkDocs built-in web server:

- Run the following command in the terminal: `mkdocs serve`



The site will be available locally for testing.





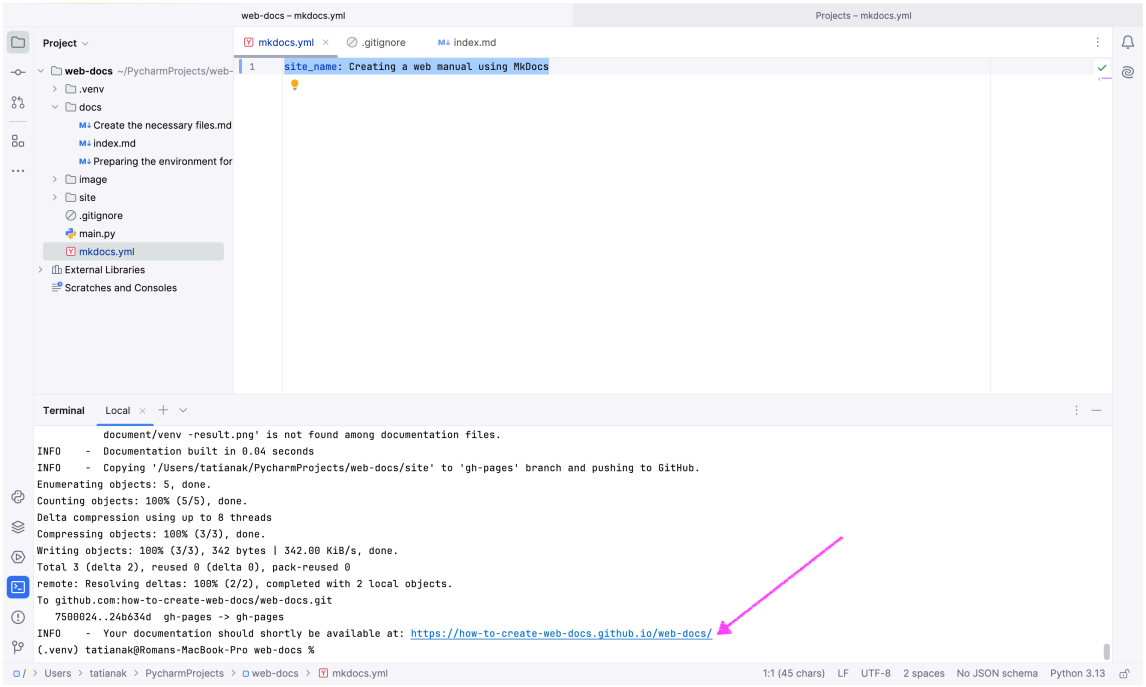
Documentation built with MkDocs.

➤ The site will automatically reload when you make changes in PyCharm.

➤ To stop the server, press Ctrl+C on your keyboard.

Deploy to GitHub Pages:

- Run the `mkdocs gh-deploy` command in the terminal.

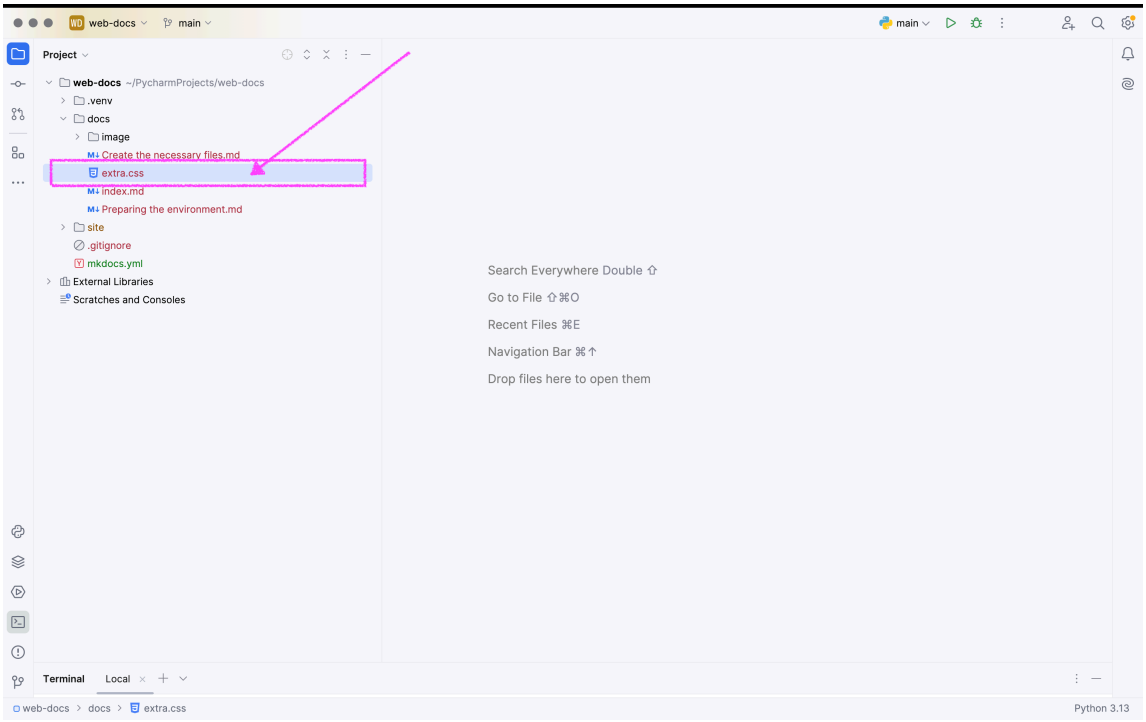


➤ This command creates a gh-pages branch, adds the compiled site to it, and pushes it to GitHub.

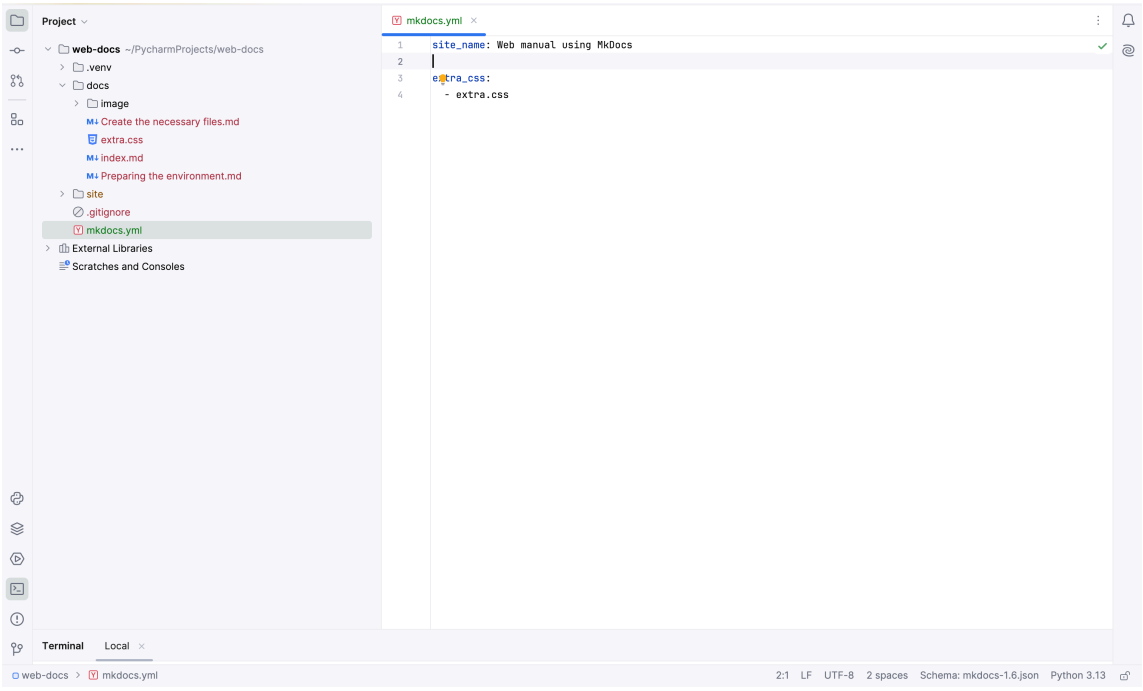
➤ The generated URL will look like this: https://[username].github.io/[repository-name]

Step 9: Edit the Appearance of Your Site

1. **Create the `extra.css` file** in the `docs` folder.



2. **Register your custom CSS** in the `mkdocs.yml` configuration file.



3. **Add your custom styles** to the `extra.css` file.

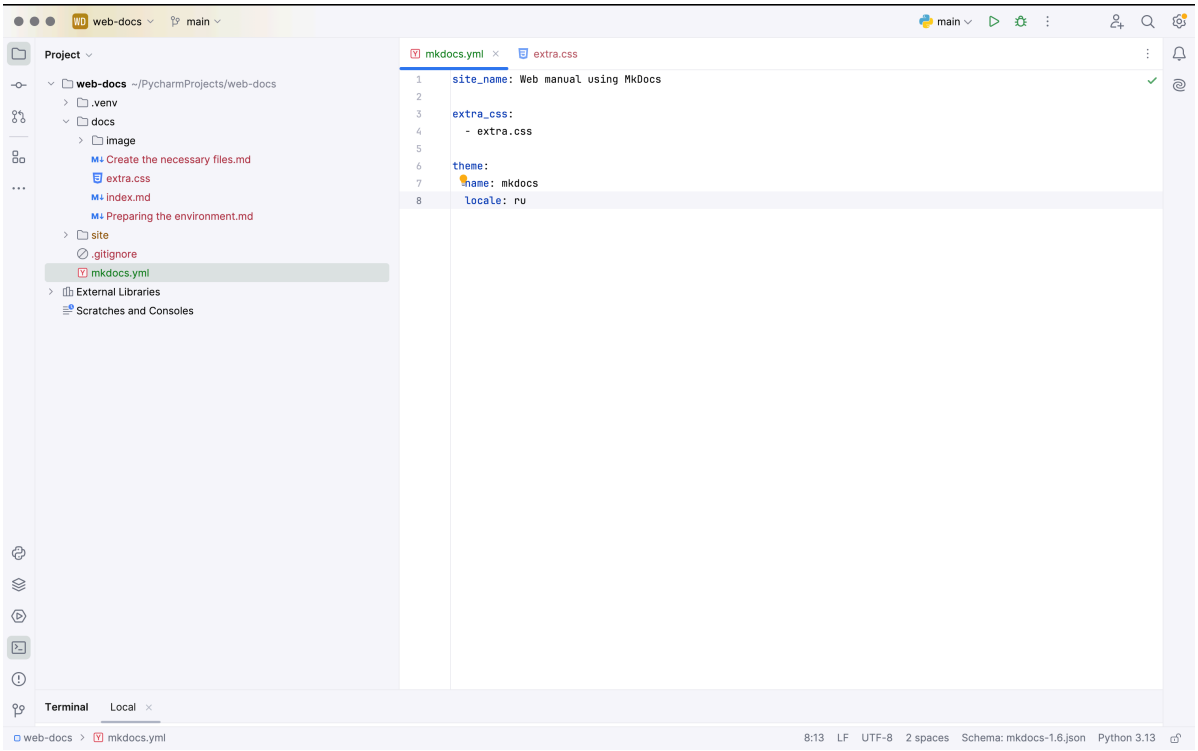
   For example, in my case:

```
/* Header styles */
.navbar.bg-primary {
    background-image: none !important;
    background-color: #2d6da7 !important;
}

/* Removing the default footer */
footer {
    display: none !important;
}

/* Styles for the navigation bar */
.navbar .nav-link {
    color: #F8F9FA !important; /* Text color for the navigation bar */
}

.navbar .nav-link:hover {
    color: #d3d9dA !important; /* Hover text color for the navigation bar */
}
```



4. Write in our configuration (mkdocs.yml) the styling theme (in my case, it is “mkdocs”) and localization (for example, ru).

➤ Localization is written to translate default elements, for example, Pagination.

Step 10: Committing Sources and Sending to the Git Repository

Follow these commands in sequence:

Adding files to the staging area: `git add .`

- This command adds all files and changes in the current directory (indicated by the `.`) to the staging area. It means you tell Git which files should be included in the next commit.
- If you need to add files from a specific folder (e.g., `docs`), use: `git add docs/` or `git add ./docs`

The choice depends on your preference, but both are valid.

Creating a commit with a message: `git commit -m "Description of changes"`

- This command saves the changes from the staging area to the repository with a message.
- If you don't use the -m flag, Git will open a text editor for you to write the description manually.

Example: `git commit -m "Added documentation files"`

Pushing changes to the remote repository: `git push origin main`

- This command sends (pushes) changes from your local repository to the remote repository (e.g., GitHub).
- `origin` is the name of the remote repository (default).
- `main` is the name of the branch where the changes are pushed. If your branch is named differently (e.g., `master`), replace `main` with the name of your branch.

Now you can view the committed changes by visiting the following link: GitHub Repository

Adjustments Made:

1. Explained the meaning of the dot (.) as a reference to the current directory.
2. Added alternative commands for adding files from specific folders (git add ./docs and git add docs/).
3. Emphasized the logic behind each command for better understanding.

Convert Markdown to Word with Pandoc

Let's look at how to convert Markdown (.md) files to Word (.docx) using Pandoc.

> Pandoc is a command-line tool and does not have a graphical interface.

At first check if Pandoc is installed on your computer:

- Open a terminal and run the command: `brew info pandoc`

If the response says `not installed`, then Pandoc is not installed.

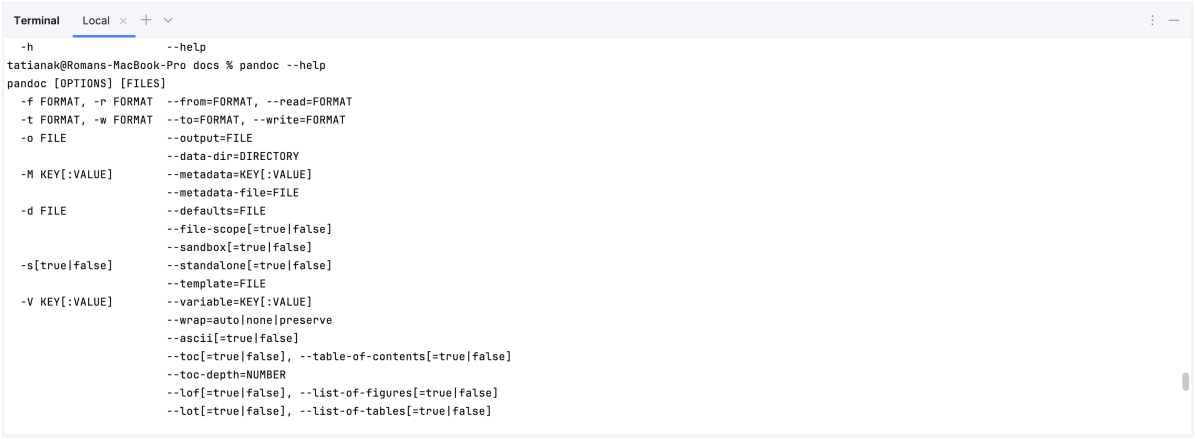
Install Pandoc

To install Pandoc, run the following command in the terminal: `brew install pandoc`

> **Pandoc** is written in **Haskell** and is installed via **brew**, not **pip**.
> There is a **Python** wrapper called **pypandoc**, but it wraps **Pandoc**.

Basics of working with Pandoc

After installation, check Pandoc help to get acquainted with its capabilities: `pandoc --help`



Command structure:

- Enter the command pandoc.
- Specify the required options (the list can be found in the help).

- Upload files for conversion.

Convert Markdown files to Word

For example, we will convert Markdown files to Word.

To perform the conversion, use the following command:

```
pandoc -f markdown -t docx -o <path_to_result.docx> <path_to_source.md>
```

Explanation of command parameters:

`-f markdown` – source file format (Markdown). `-t docx` – result format (Word). `-o` – path and name of the resulting file.

Example command: `pandoc -f markdown -t docx -o 'docs/Project Setup.docx' docs/Project\ Setup.md`

> Note that in the shell we need to specify a backslash for the md file name so that the terminal will read the command as one argument. You can also wrap the entire document in quotes.

If the file name contains spaces, note:

- Use a backslash \ to indicate a space (e.g. `Project\ Setup.md`).
- Or wrap the file name in quotes (e.g. `Project Setup.md`).

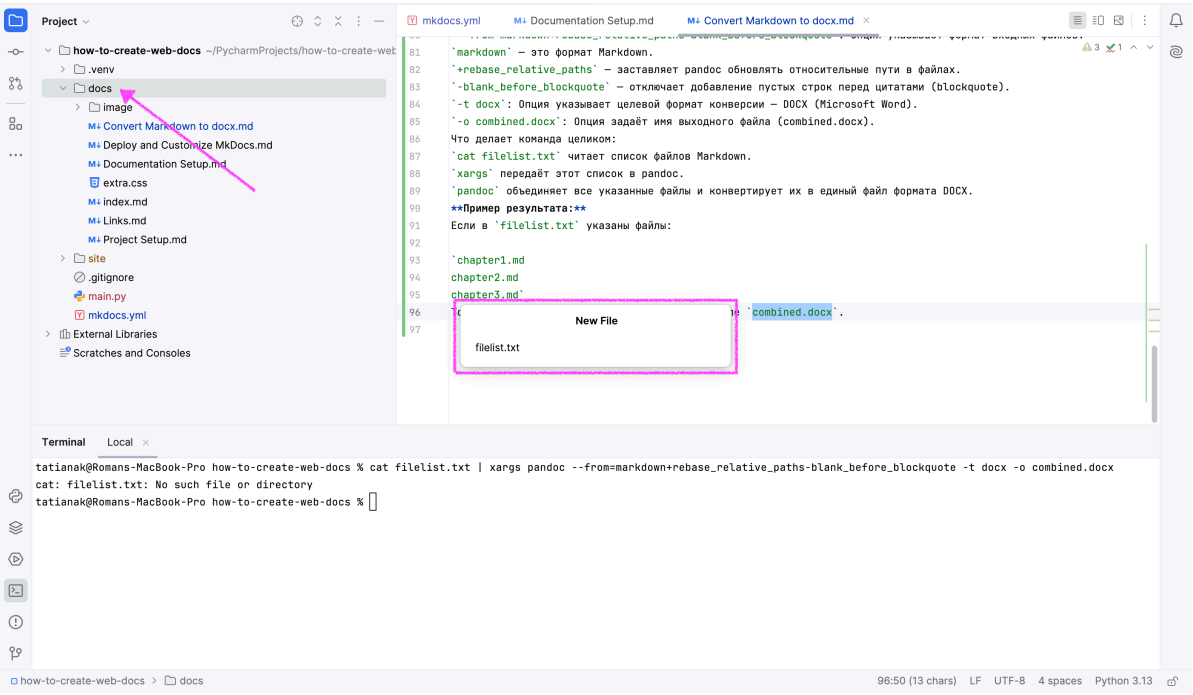
> Use Tab to autocomplete file names in the terminal - this will also automatically add the necessary slashes.

Converting multiple Markdown-files into a single DOCX file

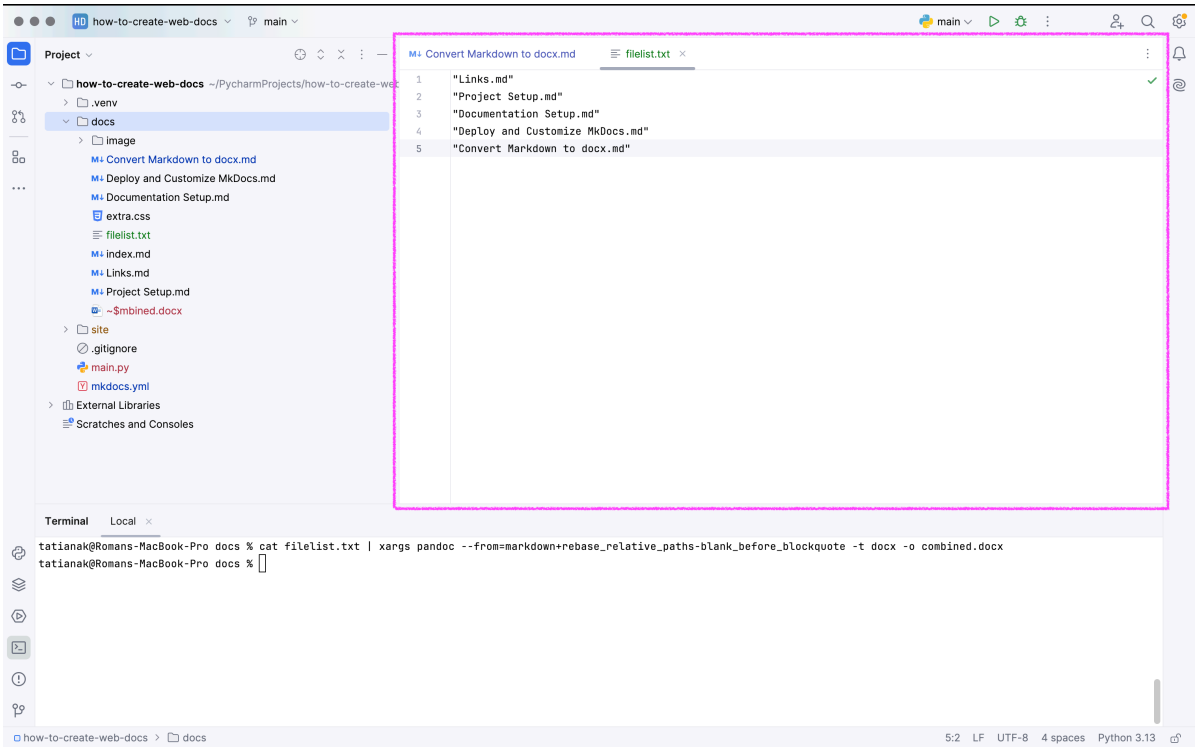
In the previous section, we learned how to convert Markdown files to docx, but we encountered the fact that each file needs to be converted separately. This is inconvenient, since you will have to manually assemble the document later. In order to convert multiple Markdown files to a DOCX file in the specified order, you need to create an additional file and run just one command.

CREATING A FILELIST.TXT FILE

Setting the order of Markdown documents is necessary, since PyCharm automatically sorts documents alphabetically. To do this, I create a document `filelist.txt`.



Then I fill the document with paths in the order in which they should be written in DOCX.



> I use relative paths and escape the paths in quotes.

RUNNING THE COMMAND

Once the `filelist.txt` document is created, I use the following command: `cat filelist.txt | xargs pandoc --from=markdown+rebase_relative_paths-blank_before_blockquote -t docx -o combined.docx`

This will generate the `combined.docx` file.

> If you get `No such file or directory` as a result of running the command, check which directory you are running the command from. Run 'pwd' and make sure you are running the document conversion from the same directory where the Markdown files and images are located. You are probably running the convert command from the top directory, so change to the correct directory (use `cd` command)

My project structure is shown below:

```
how-to-create-web-docs/
├── .venv/ # Directory for storing the Python virtual environment
├── docs/ # Folder with the main project documentation
│ ├──image/ # Document images
│ ├── Convert Markdown to docx.md # Markdown file
│ ├── Deploy and Customize MkDocs.md # Markdown file
│ ├── Documentation Setup.md # Markdown file
│ ├── Links.md # Markdown file
│ ├── Project Setup.md # Markdown file
│ ├── index.md # Main documentation page
│ ├── extra.css # Additional styles for formatting the documentation
│ ├── filelist.txt # File containing a list of Markdown documents
├── site/ # Directory where MkDocs generates a static site
├── .gitignore # Git configuration file that specifies which files or folders to exclude from the repository
├── mkdocs.yml # Main MkDocs configuration file
```

COMMAND EXPLANATION

This command converts multiple Markdown files specified in filelist.txt into a single DOCX document using the pandoc utility. Here's a step-by-step explanation:

1. `cat filelist.txt` The `cat` command outputs the contents of `filelist.txt`. This file is assumed to list the paths to the Markdown files to be processed. Each path is on a new line.

   Example contents of `filelist.txt`:

   `chapter1.md`
   `chapter2.md`

`chapter3.md`

2. `|` (pipe) The `|` character routes the output of the cat command as input to the command xargs.
3. `xargs` The `xargs` command takes the strings passed through the pipe and substitutes them as arguments to the pandoc command. This allows each of the specified files to be processed.
4. pandoc `--from=markdown+rebase_relative_paths-blank_before_blockquote -t docx -o combined.docx` This command uses `pandoc` to convert Markdown files into a single DOCX document. Let's look at pandoc's arguments:
    - `--from=markdown+rebase_relative_paths-blank_before_blockquote`: The option specifies the format of the input files.
    - `markdown` is the Markdown format.
    - `+rebase_relative_paths` makes pandoc update relative paths in files.
    - `-blank_before_blockquote` disables adding blank lines before blockquotes.
    - `-t docx`: The option specifies the target conversion format - DOCX (Microsoft Word).
    - `-o combined.docx`: The option specifies the name of the output file (combined.docx).

What the whole command does:

`cat filelist.txt` reads a list of Markdown files.

`xargs` passes this list to pandoc.

`pandoc` combines all the specified files and converts them into a single DOCX file.

Example output:

If `filelist.txt` specifies the files:

`chapter1.md`
`chapter2.md`
`chapter3.md` Then pandoc will combine their contents into a single file `combined.docx`.

Setting up the Material theme for MkDocs

> Enhance your document with the Material theme.

In this tutorial, we'll cover how to:

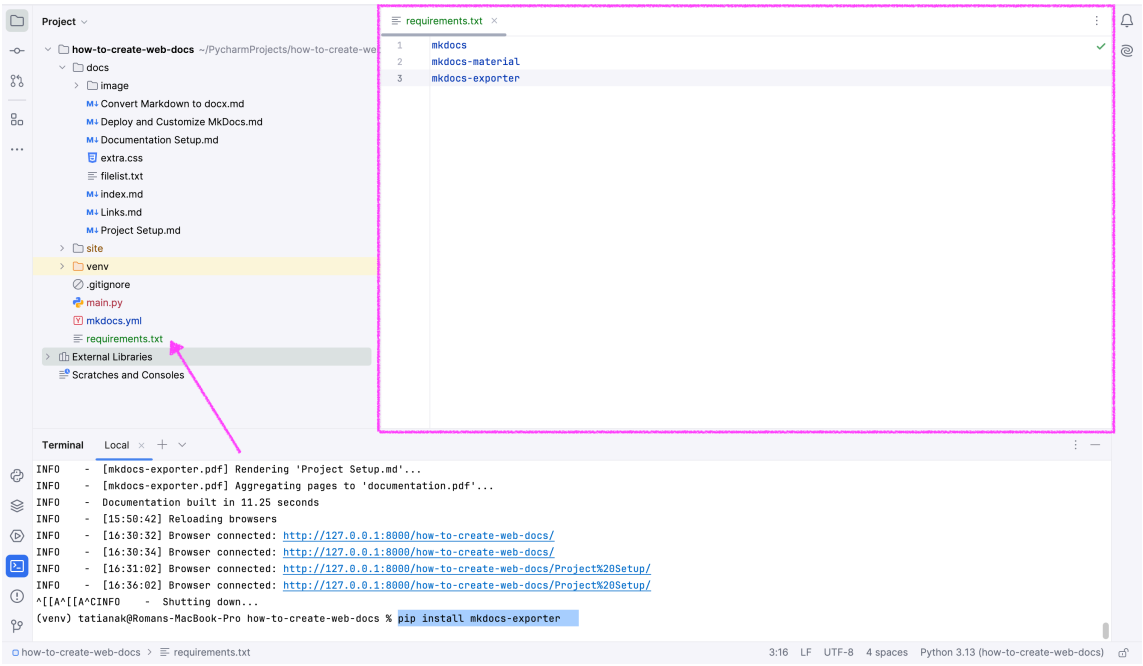
1. Set up tooltips with definitions for concepts.
2. Add a footer.
3. Add a PDF export button.
4. Set up a site footer.
5. Change the site color scheme and add styles.

Reconfiguring a virtual environment

If you encounter errors while setting up a theme, follow these steps:

1. Delete the old virtual environment if you created one.
2. Create a new one:
   - In the IDE panel select `Add New Interpreter` > `Add Local Interpreter`.
   - Specify the settings and confirm.

3. Activate the environment: `source venv/bin/activate`
4. Add a `requirements.txt` file to the root of the project with the following content:
   - mkdocs
   - mkdocs-material
   - mkdocs-exporter



> Why this file? It simplifies dependency management, allowing you to quickly install all the necessary libraries with a single command.

5. Install dependencies: `pip install -r requirements.txt`
6. If the `mkdocs-exporter` library is not installed, run the command `pip install mkdocs-exporter`.

MkDocs Configuration

This config is suitable for a basic project with an emphasis on interface customization (color scheme, custom footer) and support for exporting documentation to PDF. Here is the `mkdocs.yml` configuration file with explanations:

Basic settings

```
site_name: Web-docs using MkDocs
site_url: https://MkDocs-web-docs.github.io/how-to-create-web-docs/
```

- **site_name:** The name of your documentation site. It appears in the browser title and site header.
- **site_url:** The base URL for generating links when building a static site.

Navigation

```
nav:
  - Creating a web manual using MkDocs: index.md
  - Links: Links.md
  - Project Setup: Project Setup.md
  - Documentation Setup: Documentation Setup.md
  - Deploy and Customize: Deploy and Customize MkDocs.md
  - Convert Markdown to docx: Convert Markdown to docx.md
  - Glossary: Glossary.md
```

> Navigation defines the menu structure. Each line links the section title to a Markdown file.

Theme settings

```
theme:
  palette:
    primary: pink
    accent: lime
  name: material
  custom_dir: overrides
  language: en
  features:
    - navigation.instant
    - header.autohide
    - content.tooltips
  hide:
    - footer
```

- **palette:** sets the color scheme.
  - **primary:** the main color (e.g. header and buttons).
  - **accent:** accent color (e.g. links).
- **custom_dir:** specifies the folder with custom templates (e.g. for changing the footer).
- **features:**
  - **navigation.instant:** instant navigation without reloading the page.
  - **header.autohide:** hide the header when scrolling down.
  - **content.tooltips:** improved tooltips for the title attribute.
- **hide.footer:** hides the standard footer.

Plugins

```
plugins:
  - exporter:
      formats:
        pdf:
          enabled: true
          aggregator:
            enabled: true
            output: documentation.pdf
            covers: all
      buttons:
        - title: Сохранить в PDF
          icon: material-file-download-outline
  - search
```

- **exporter:** adds PDF export functionality. The button with the text "Save to PDF" will be available on the site.
- **search:** enables built-in search.

Markdown Extensions

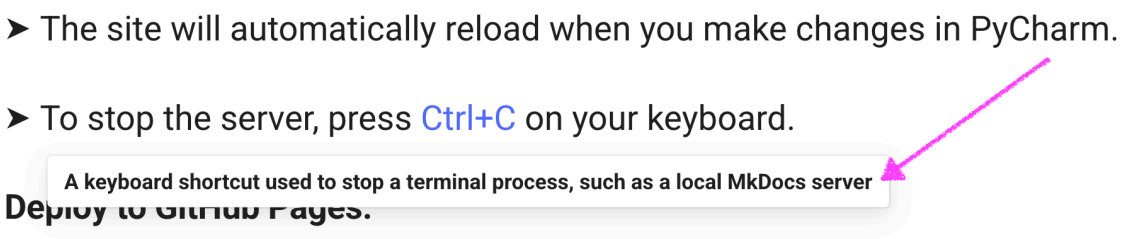
```
markdown_extensions:
  - abbr
  - attr_list
  - pymdownx.snippets
```

- **abbr:** allows you to specify abbreviations.
- **attr_list:** adds attribute support for Markdown elements.
- **pymdownx.snippets:** allows you to reuse certain Markdown blocks.

Adding tooltips

1. Add the following configuration to your mkdocs.yml file to enable tooltips: `yaml`
   `theme:`
   `features:`
   `- content.tooltips`
2. Add the title attribute to the Markdown text:
   Example:
   `markdown`
   `[Ctrl+C](# "A keyboard shortcut used to stop a terminal process")`

This code will add a tooltip when you hover over a concept, and it will display its definition.



> The title attribute will not work inside a code block.

3. Check the result.
   When you run the site with mkdocs serve, you will be able to see the tooltip that appears when you hover over `Ctrl+C`.

> Tooltip display is not shown in PyCharm preview

Removing the footer

Next, remove the footer if necessary. The instructions indicate a simple method, which is described here.

If the standard method did not work, use the Material theme customization:

1. Create an overrides folder: `mkdir overrides`
2. Create a footer.html file: `mkdir -p overrides/partials` `echo "" > overrides/partials/footer.html`
3. In the config, specify the custom_dir directory:

```
theme:
  name: material
  custom_dir: overrides
```

Now the footer will be completely removed.

Changing the color scheme

To change the colors, use the instructions Changing the colors.

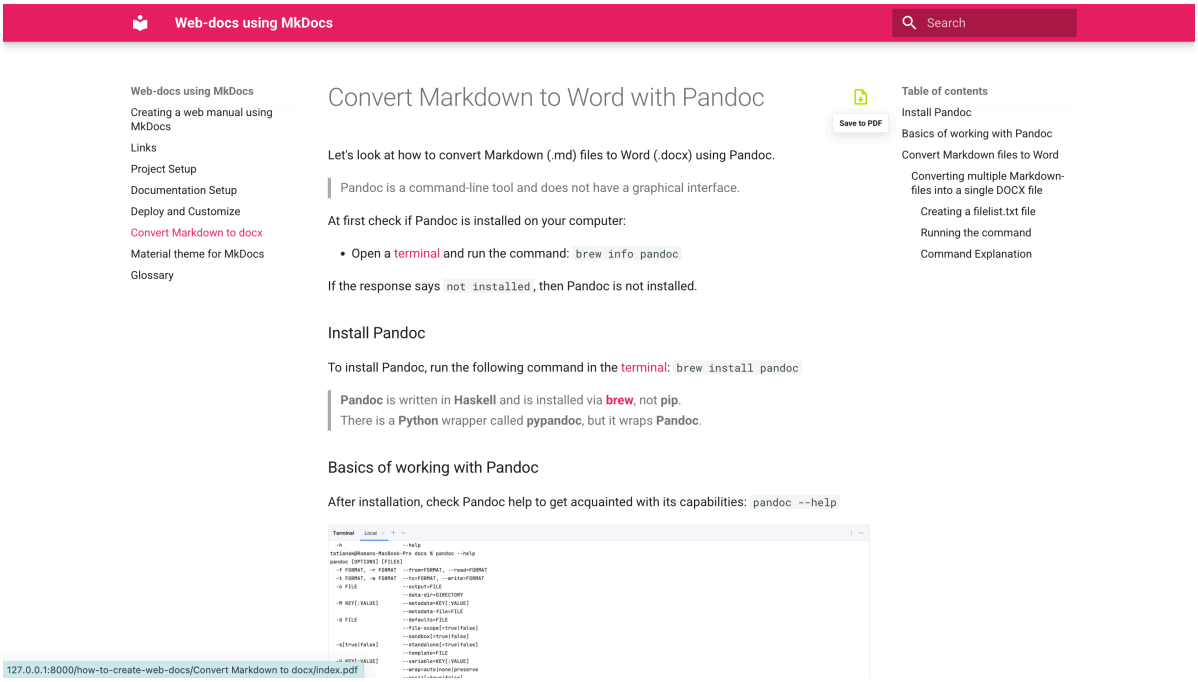
In the config, change the parameters palette:

> My example is pink)

```
palette:
  primary: pink
  accent: lime
```

> The material is an amazing theme with very clear and concise instructions. Imagine that from boring Word you can now create beautiful web documents simply and easily.

For take this example, you can just copy this config:



```
site_name: Web-docs using MkDocs
site_url: https://MkDocs-web-docs.github.io/how-to-create-web-docs/

nav:
  - Creating a web manual using MkDocs: index.md
  - Links: Links.md
  - Project Setup: Project Setup.md
  - Documentation Setup: Documentation Setup.md
  - Deploy and Customize: Deploy and Customize MkDocs.md
  - Convert Markdown to docx: Convert Markdown to docx.md
  - Material theme for MkDocs: Material theme for MkDocs.md
  - Glossary: Glossary.md

theme:
  palette:
    primary: pink
    accent: lime
  name: material
  custom_dir: overrides
  language: en
  features:
    - navigation.instant
    - header.autohide
    - content.tooltips
  hide:
    - footer

plugins:
  - exporter:
      formats:
        pdf:
          enabled: true
          aggregator:
            enabled: true
            output: documentation.pdf
            covers: all
      buttons:
        - title: Save to PDF
          icon: material-file-download-outline
          enabled:
!!python/name:mkdocs_exporter.formats.pdf.buttons.download.enabled
          attributes:
!!python/name:mkdocs_exporter.formats.pdf.buttons.download.attributes
  - search

markdown_extensions:
  - abbr
  - attr_list
  - pymdownx.snippets

extra: {}
```

Now our project structure looks this:

```
how-to-create-web-docs/
├── docs/ # Directory with Markdown documentation files
│ ├── Convert Markdown to docx.md
│ ├── Deploy and Customize MkDocs.md
│ ├── Documentation Setup.md
│ ├── extra.css # Custom styles (if applicable)
│ ├── filelist.txt # Auxiliary file (e.g. file list)
│ ├── Glossary.md
│ ├── index.md
│ ├── Links.md
│ ├── Material theme for MkDocs.md
│ ├── Project Setup.md
├── image/ # Image folder
├── overrides/ # Template customization
│ ├── partials/ # Template parts
│ │ ├── footer.html # Custom footer
├── site/ # Generated site (created automatically)
├── venv/ # Python virtual environment
├── .gitignore # File to exclude from repository
├── mkdocs.yml # Main MkDocs configuration file
├── requirements.txt # Project dependencies file
```

Glossary

.venv: python virtual environment folder. Contains project dependencies isolated from the system.

Brew (Homebrew): a package manager for macOS and Linux that allows you to install and manage various command line tools.

cat: a terminal command that prints the contents of a file to the terminal.

cd: a command to change the working directory in the terminal.

Clone Repository / Get from VCS: the process of copying a remote repository from GitHub to a local computer for working with it.

Command-line tool: a program controlled via a text-based command-line interface, without a graphical interface.

Combined.docx: the resulting Word file created by combining and converting several Markdown documents.

CSS (Cascading Style Sheets): a style language for describing the appearance of HTML documents, including colors, fonts, indents, and other visual aspects.

Ctrl+C: a keyboard shortcut used to stop a terminal process, such as a local MkDocs server.

docs/: a folder containing the source documentation files in Markdown format.

DOCX: a Microsoft Word document format that supports complex markup, images, tables, and other formatting.

Escape characters: special characters used to represent spaces or other characters in terminal commands. For example, Project\ Setup.md uses a backslash () to represent a space.

extra.css: a file containing custom CSS styles for changing the appearance of the documentation generated by MkDocs.

Filelist.txt: a text file containing a list of paths to Markdown files to be merged or processed.

gh-pages: a special branch in the GitHub repository that contains the generated site files for GitHub Pages.

GitHub: a platform for collaborative work on projects, managing repositories, and hosting code.

GitHub Pages: a service for hosting static sites directly from GitHub repositories. Often used to publish documentation.

GitHub Repository: a storage space for project files used for version control and collaborative work on code.

Haskell: the programming language in which Pandoc is written. It features a functional style and strong typing.

HTTPS: a protocol for secure connection and data transfer between a computer and a server.

image/: a folder containing images used in documentation.

index.md: a Markdown file that appears on the main page of a site created with MkDocs.

Localization: the process of configuring a site to support a specific language, such as Russian (ru), with translation of interface elements.

Markdown: a markup language that allows you to format text using simple characters. It is used to write structured text, such as documentation.

mkdocs-material: a skin for MkDocs that provides modern and convenient styles for documentation.

mkdocs.yml: MkDocs configuration file. Used to configure the site, including the theme, localization, and custom styles.

mkdocs gh-deploy: a command that generates a static site, creates a gh-pages branch, and uploads the site's content to a GitHub repository.

mkdocs serve: a command to start a local MkDocs web server. Allows you to preview the documentation before deployment.

nav: a section in mkdocs.yml that describes the site's navigation structure. Determines which files are displayed as pages.

Pandoc: a feature-rich command-line tool for converting documents between different formats, such as Markdown, DOCX, PDF.

Pagination: splitting content into pages with buttons for navigating between them. MkDocs supports and translates this when localizing.

Pipe (|): a symbol used to pass the output of one command as input to another command.

pip: a tool for installing and managing Python libraries and packages.

Project structure: a hierarchical organization of files and folders in a project. For example, the docs/ folder contains Markdown documents.

Python: a programming language for its simple syntax and wide range of libraries. It is used to work with virtual environments and install packages.

Python wrapper: an interface written in Python that provides access to the functionality of another program or library, such as pypandoc.

pwd: a terminal command that shows the current working directory.

PyCharm: integrated development environment (IDE) with support for Python, Markdown files, and documentation preview functions.

Relative paths: references to files or folders relative to the current working directory, instead of specifying the full path.

SSH (Secure Shell): a protocol for securely connecting to remote servers, often used to work with repositories. Requires SSH keys to be configured.

SSH keys: cryptographic keys used for secure authentication without entering a password.

Staging area: a time zone where changes ready for commit are stored.

Static site: a website consisting of pre-generated HTML pages.

Terminal: a program for executing commands in the text interface of the operating system.

theme: a parameter in mkdocs.yml that specifies the theme of the site.

xargs: a command-line utility that allows you to pass input (such as a list of files) to another command as arguments.

YAML: data serialization format that uses indentation for structuring. Used in configuration files (such as mkdocs.yml).