Jupytext

The Jupytext Team

Jul 29, 2022

CONTENTS

Have you always wished Jupyter notebooks were plain text documents? Wished you could edit them in your favorite IDE? And get clear and meaningful diffs when doing version control? Then… Jupytext may well be the tool you're looking for!

Jupytext is a plugin for Jupyter that can save Jupyter notebooks as either

- Markdown files (or *MyST Markdown* files, or *R Markdown* or *Quarto* text notebooks)
- Scripts in *many languages*.

CHAPTER

ONE

USE CASES

Common *use cases* for Jupytext are:

- Doing version control on Jupyter Notebooks
- Editing, merging or refactoring notebooks in your favorite text editor
- Applying Q&A checks on notebooks.

CHAPTER

TWO

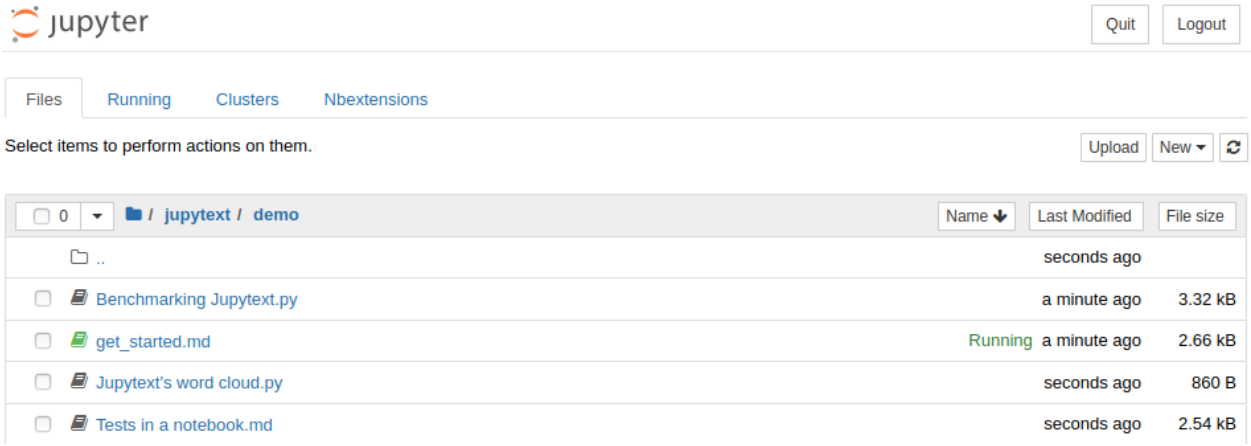
INSTALL

You can install Jupytext with

- `pip install jupytext`
- or `conda install jupytext -c conda-forge`.

Then, restart your Jupyter server (for more installation details, see the *install section* in the documentation).

When Jupytext is installed, `.py` and `.md` files have a notebook icon. And you can really open and run these files as notebooks



(click on the image above to try this on)

```
{
  "defaultViewers": {
    "markdown": "Jupytext Notebook",
    "myst": "Jupytext Notebook",
    "r-markdown": "Jupytext Notebook",
    "quarto": "Jupytext Notebook",
    "julia": "Jupytext Notebook",
    "python": "Jupytext Notebook",
    "r": "Jupytext Notebook"
  }
}
```

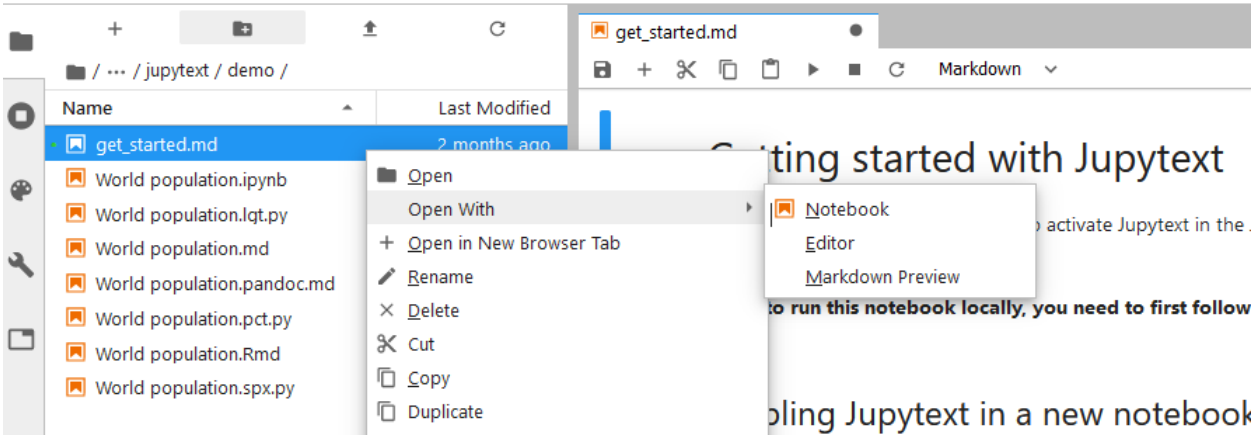
Here is a screencast of the steps to follow:

(click on the image above to try this on)

Another possibility is to activate this with a default_setting_overrides.json file in the `.jupyter/labconfig` folder with e.g.

```
wget https://raw.githubusercontent.com/mwouts/jupytext/main/binder/labconfig/default_
↪setting_overrides.json -P  ~/.jupyter/labconfig/
```

Note: to open links to `.md` files in notebooks with the Notebook editor, use `jupyterlab>=4.0.0a16`.



(click on the image above to try this on)

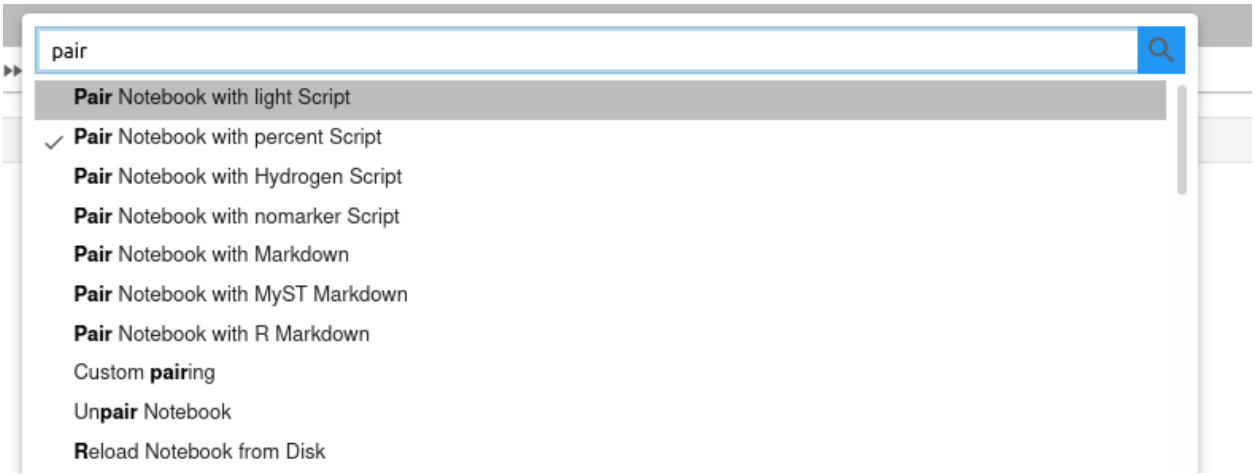
CHAPTER

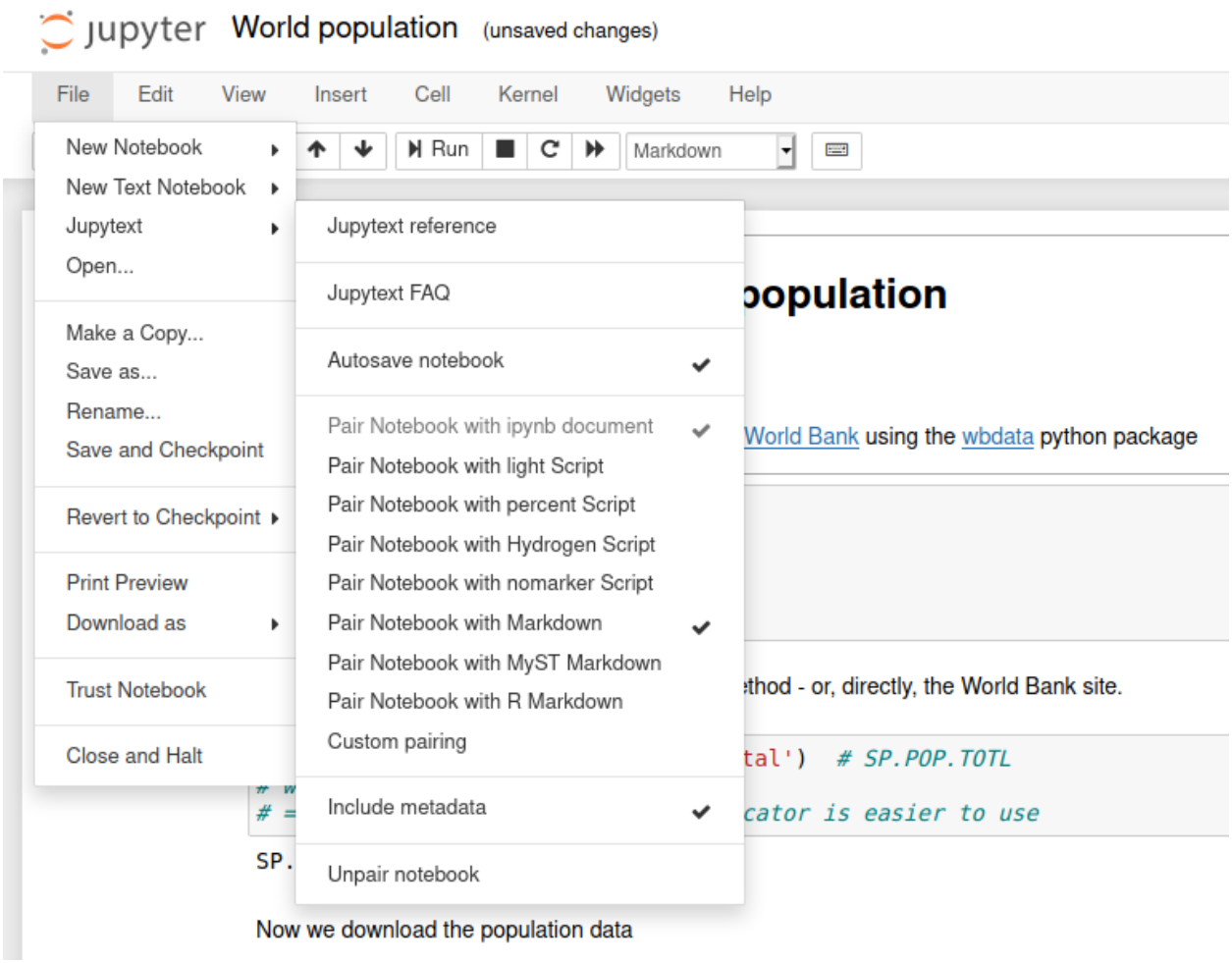
THREE

PAIRED NOTEBOOKS

The most convenient way to use Jupytext is probably through *paired notebooks*.

To pair a given `.ipynb` or text notebook to an additional notebook format, use either





with e.g.

```
jupytext --set-formats ipynb,py:percent notebook.ipynb
```

see the *documentation*.

with e.g. the following content:

```
formats = "ipynb,py:percent"
```

see the *documentation*.

When you save a paired notebook in Jupyter, both the `.ipynb` file and the text version are updated on disk.

When a paired notebook is opened or *reloaded* in Jupyter, the input cells are loaded from the text file, and combined with the output cells from the `.ipynb` file.

You can edit the text representation of the notebook in your favorite editor, and get the changes back in Jupyter by simply *reloading* the notebook (Ctrl+R in Jupyter Notebook, "reload notebook" in Jupyter Lab). And the changes are propagated to the `.ipynb` file when you *save* the notebook.

Alternatively, you can synchronise the two representations by running `jupytext --sync notebook.ipynb` at the command line.

CHAPTER

FOUR

WHICH TEXT FORMAT?

Jupytext implements many text *formats* for Jupyter Notebooks. If your notebook is mostly made of code, you will probably prefer to save it as a script:

- Use the *percent format*, a format with explicit cell delimiters (`# %%`), supported by many IDE (Spyder, Hydrogen, VS Code, PyCharm and PTVS)
- Or use the *light format*, if you prefer to see fewer cell markers.

If your notebook contains more text than code, if you are writing a documentation or a book, you probably want to save your notebook as a Markdown document

- Use the *Jupytext Markdown format* if you wish to render your notebook as a `.md` file (without its outputs) on GitHub
- Use the *MyST Markdown format*, a markdown flavor that "implements the best parts of reStructuredText", if you wish to render your notebooks using Sphinx or Jupyter Book.
- Use the *R Markdown format* or the *Quarto format* if you want to open your Jupyter Notebooks in RStudio.

CHAPTER

FIVE

MORE RESOURCES?

If you're new to Jupytext, you may want to start with the *FAQ* or with the *Tutorials*.

CHAPTER

SIX

TABLE OF CONTENTS

6.1 Installation

Jupytext is available on pypi and on conda-forge. Run either of

```
pip install jupytext --upgrade
```

or

```
conda install jupytext -c conda-forge
```

If you want to use Jupytext within Jupyter Notebook or JupyterLab, make sure you install Jupytext in the Python environment where the Jupyter server runs. If that environment is read-only, for instance if your server is started using JupyterHub, install Jupytext in user mode with:

```
/path_to_your_jupyter_environment/python -m pip install jupytext --upgrade --user
```

6.1.1 Jupytext's contents manager

Jupytext provides a contents manager for Jupyter that allows Jupyter to open and save notebooks as text files. When Jupytext's content manager is active in Jupyter, scripts and Markdown documents have a notebook icon.

In most cases, Jupytext's contents manager is activated automatically by Jupytext's server extension. When you restart either `jupyter lab` or `jupyter notebook`, you should see a line that looks like:

```
[I 10:28:31.646 LabApp] [Jupytext Server Extension] Changing NotebookApp.contents_
↪manager_class from LargeFileManager to jupytext.TextFileContentsManager
```

If you don't have the notebook icon on text documents after a fresh restart of your Jupyter server, you can either enable our server extension explicitly (with `jupyter serverextension enable jupytext`), or install the contents manager manually. Append

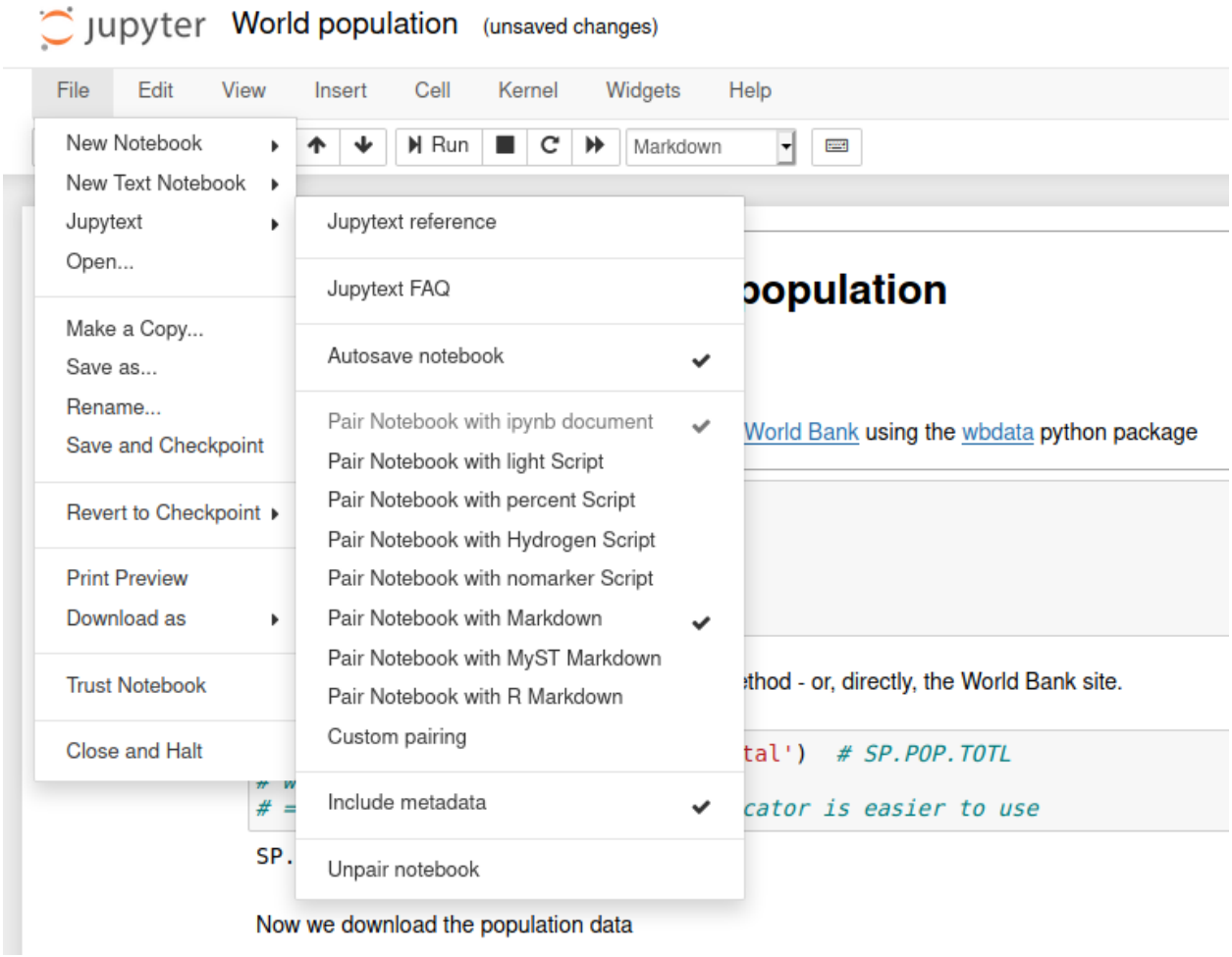
```
c.NotebookApp.contents_manager_class = "jupytext.TextFileContentsManager"
```

to your `.jupyter/jupyter_notebook_config.py` file (generate a Jupyter config, if you don't have one yet, with `jupyter notebook --generate-config`). Our contents manager accepts a few options: default formats, default metadata filter, etc. Then, restart Jupyter Notebook or JupyterLab, either from the JupyterHub interface or from the command line with

```
jupyter notebook # or lab
```

6.1.2 Jupytext menu in Jupyter Notebook

Jupytext includes an extensions for Jupyter Notebook that adds a Jupytext section in the File menu.

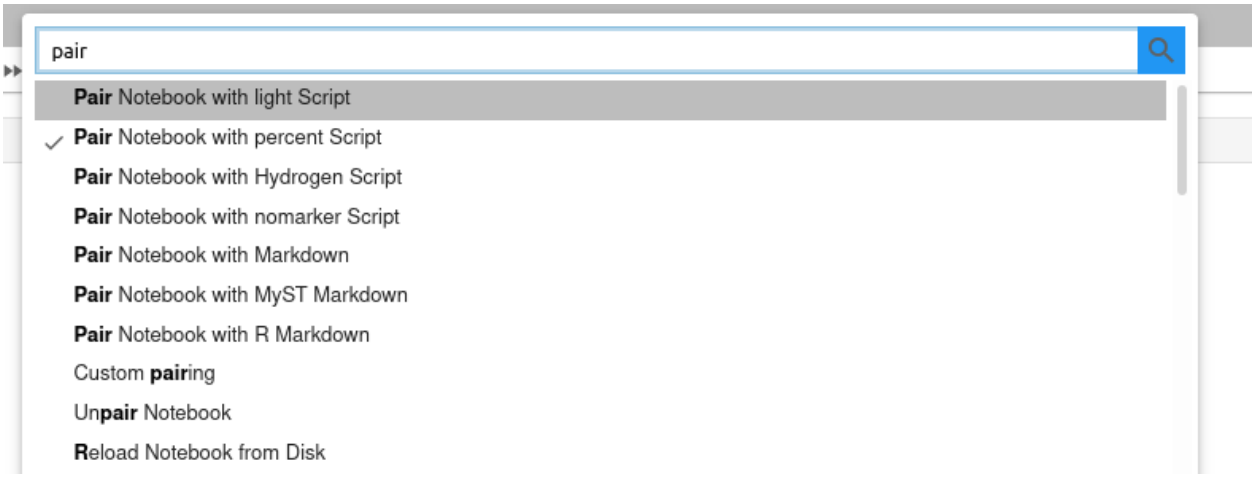


If the extension was not automatically installed, install and activate it with

```
jupyter nbextension install --py jupytext [--user]
jupyter nbextension enable --py jupytext [--user]
```

6.1.3 Jupytext commands in JupyterLab

In JupyterLab, Jupytext adds a set of commands to the command palette (View / Activate Command Palette, or Ctrl+Shift+C):



The Jupytext extension for JupyterLab is bundled with Jupytext. In most cases you don't need to install it explicitly.

From Jupytext 1.9.0 on, the version of the extension is compatible with JupyterLab 3.x only. If you wish to use Jupytext with JupyterLab 2.x or 1.x, please

- install the `jupytext` package using `pip` or `conda`
- and then, install the last version of the `jupyterlab-jupytext` extension that is compatible with your version of JupyterLab, i.e.

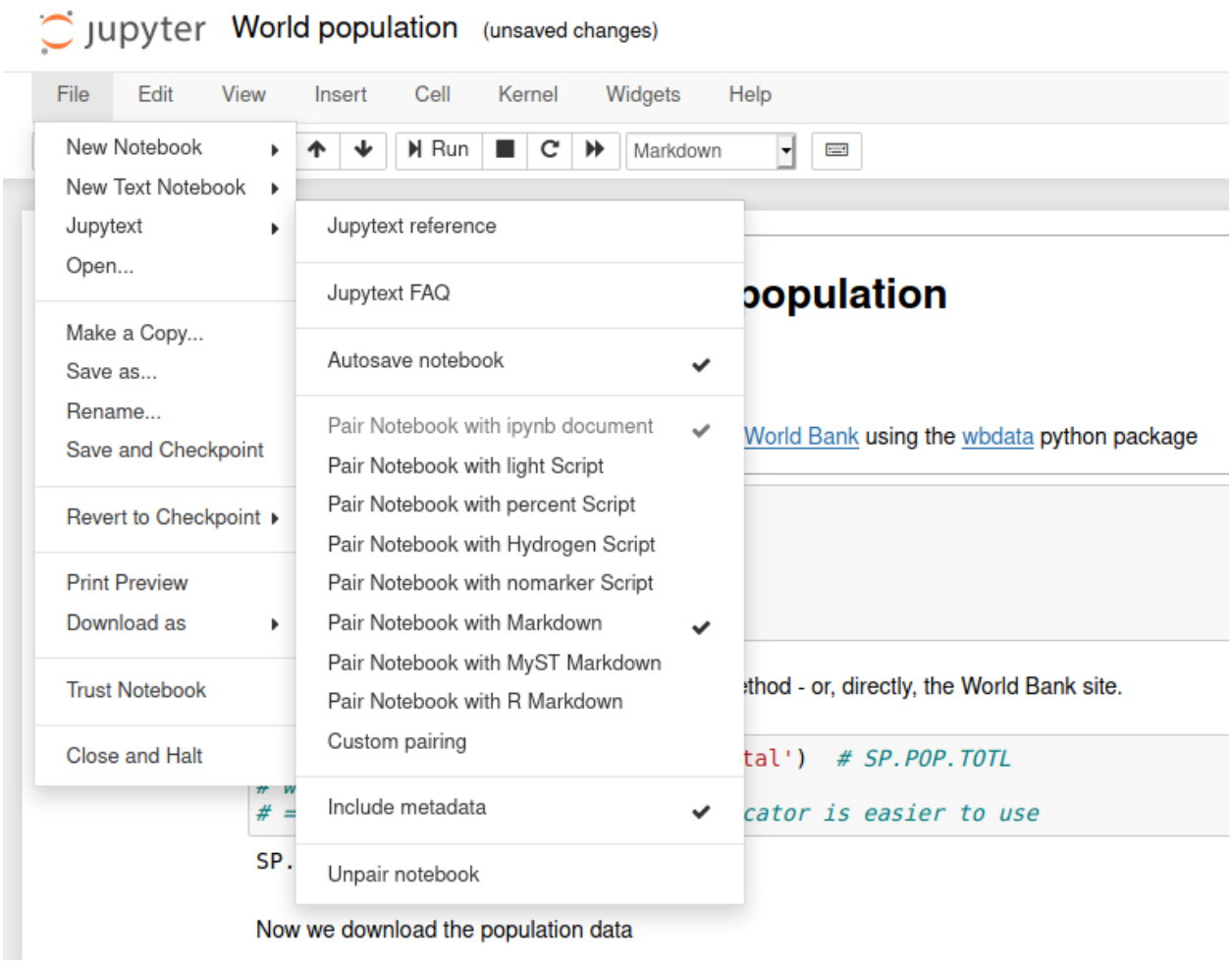
```
jupyter labextension install jupyterlab-jupytext@1.2.2  # for JupyterLab 2.x
jupyter labextension install jupyterlab-jupytext@1.1.1  # for JupyterLab 1.x
```

6.2 Paired notebooks

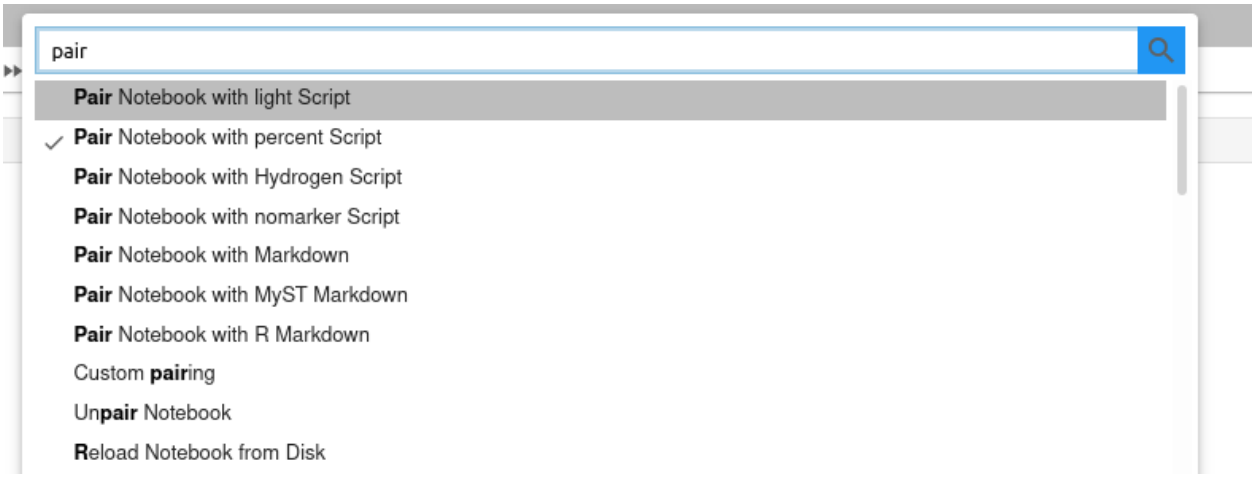
Jupytext can write a given notebook to multiple files. In addition to the original notebook file, Jupytext can save the input cells to a text file — either a script or a Markdown document. Put the text file under version control for a clear commit history. Or refactor the paired script, and reimport the updated input cells by simply refreshing the notebook in Jupyter.

6.2.1 How to pair a notebook to multiple formats

In Jupyter Notebook, pair your notebook to one or more text formats with the *Jupytext menu*:



In JupyterLab, use the *Jupytext commands*:



These command simply add a `"jupytext": {"formats": "ipynb,md"}`-like entry in the notebook metadata.

You can also configure the notebook pairing on the command line, and set a default pairing for all the notebooks either globally or in a subfolder - see *here*.

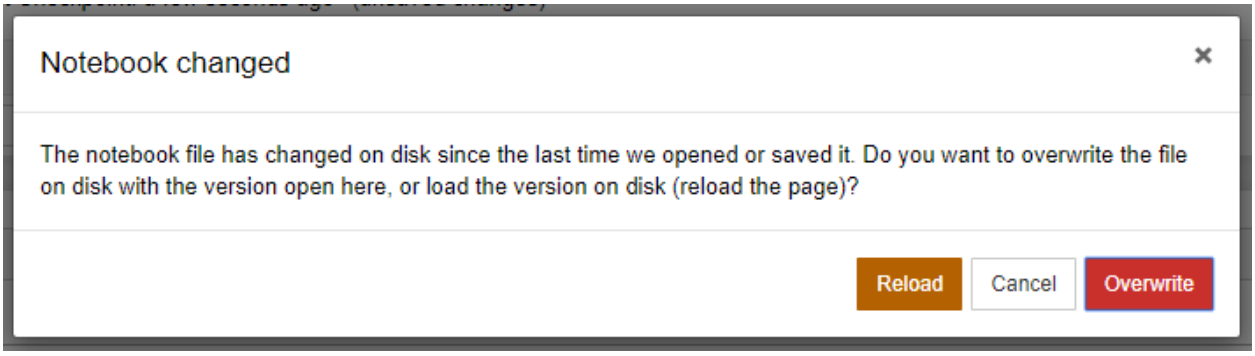
6.2.2 Can I edit a notebook simultaneously in Jupyter and in a text editor?

When saving a paired notebook using Jupytext's contents manager, Jupyter updates both the `.ipynb` and its text representation. The text representation can be edited outside of Jupyter. When the notebook is refreshed in Jupyter, the input cells are read from the text file, and the output cells from the `.ipynb` file.

It is possible (and convenient) to leave the notebook open in Jupyter while you edit its text representation. However, you don't want the two editors to save the notebook simultaneously. To avoid this:

- deactivate Jupyter's autosave, by either toggling the `"Autosave notebook"` menu entry or run `%autosave 0` in a cell of the notebook (see in the faq how to deactivate autosave permanently)
- and refresh the notebook when you switch back from the editor to Jupyter.

In case you forgot to refresh, and saved the Jupyter notebook while the text representation had changed, no worries: Jupyter will ask you which version you want to keep:

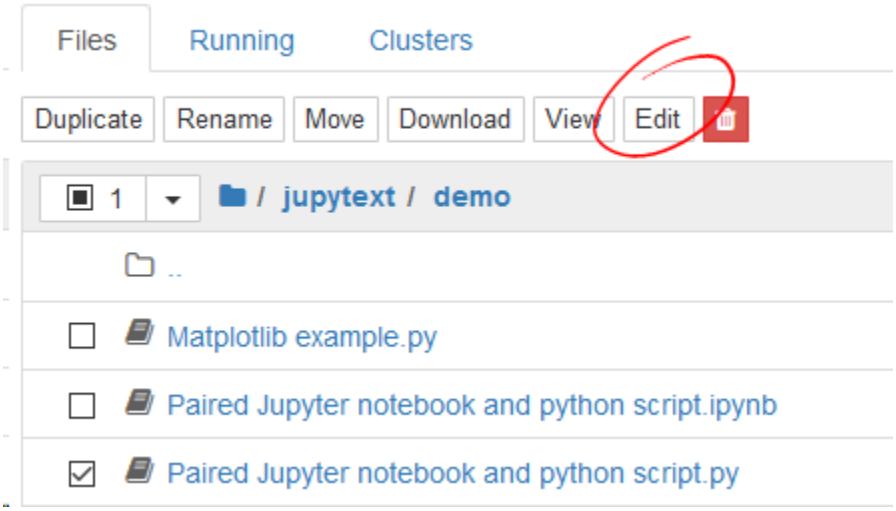


When that occurs, please choose the version in which you made the latest changes. And give a second look to our advice to deactivate the autosaving of notebooks in Jupyter.

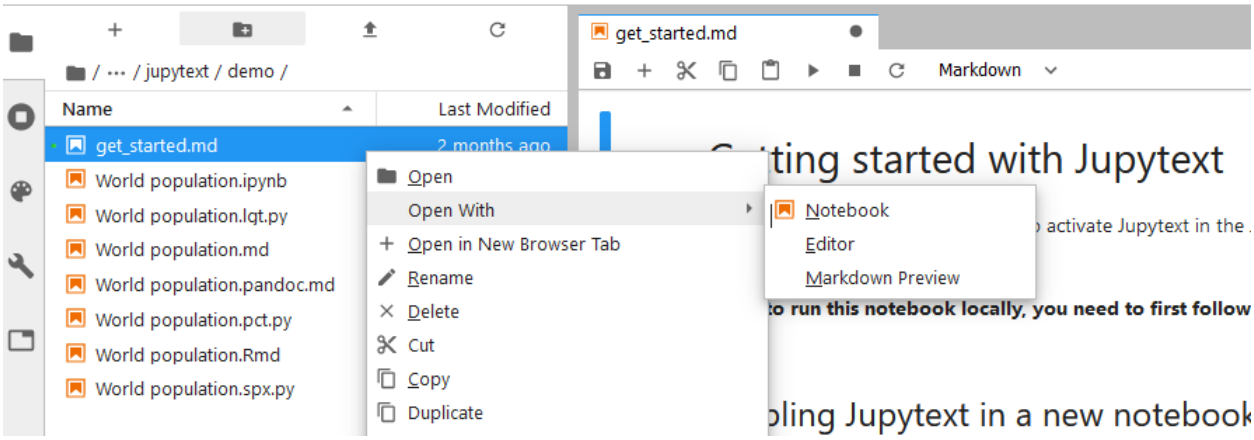
6.2.3 How to open scripts with either the text or notebook view in Jupyter?

With Jupytext's contents manager for Jupyter, scripts and Markdown documents gain a notebook icon. If you don't see the notebook icon, double check the *contents manager configuration*.

By default, Jupyter Notebook open scripts and Markdown documents as notebooks. If you want to open them with the text editor, select the document and click on *edit*:



In JupyterLab this is slightly different. Scripts and Markdown document also have a notebook icon. But they open as text by default. Open them as notebooks with the *Open With -> Notebook* context menu (available in JupyterLab 0.35 and above):



If do not want to classify scripts or Markdown documents as notebooks, please use the `notebook_extension` option. For instance, if you want to get the notebook icon only for `.ipynb` and `.Rmd` files, set

```
notebook_extensions = "ipynb,Rmd"
```

in your local or global *jupytext.toml* file.

Please note that, with the above setting, Jupyter will not let you open scripts as notebooks. If you still want to do so, use the Jupytext command line (see below) to first convert or pair the script to an `.ipynb` notebook.

6.3 Configuration

6.3.1 Per-notebook configuration

The pairing information for one or multiple notebooks can be set on the command line:

```
jupytext --set-formats ipynb,py [--sync] notebook.ipynb
```

You can pair a notebook to as many text representations as you want (see our *World population* notebook in the demo folder). Format specifications are of the form

```
[[root_folder//][path/][prefix]/][suffix.]ext[:format_name]
```

where

- `ext` is one of `ipynb`, `md`, `Rmd`, `jl`, `py`, `R`, `sh`, `cpp`, `q`. Use the `auto` extension to have the script extension chosen according to the Jupyter kernel.
- `format_name` (optional) is either `light` (default for scripts), `nomarker`, `percent`, `hydrogen`, `sphinx` (Python only), `spin` (R only) — see the *format specifications*.
- `root_folder`, `path`, `prefix` and `suffix` allow to save the text representation to files with different names, or in different folders (see the *configuration files examples*).

Jupytext accepts a few additional options. These options should be added to the `"jupytext"` section in the metadata — use either the metadata editor or the `--opt/--format-options` argument on the command line.

- `comment_magics`: By default, Jupyter magics are commented when notebooks are exported to any other format than markdown. If you prefer otherwise, use this boolean option, or its global counterpart (see below).
- `notebook_metadata_filter`: By default, Jupytext only exports the `kernelspec` and `jupytext` metadata to the text files. Set `"jupytext": {"notebook_metadata_filter": "-all"}` if you want that the script has no notebook metadata at all. The value for `notebook_metadata_filter` is a comma separated list of additional/excluded (negated) entries, with `all` a keyword that allows to exclude all entries. Use dots to filter recursively the metadata. For instance, use `notebook_metadata_filter="-jupytext.text_representation.jupytext_version"` to remove the `jupytext_version` field in the `jupytext.text_representation` metadata.
- `cell_metadata_filter`: By default, cell metadata `autoscroll`, `collapsed`, `scrolled`, `trusted` and `ExecuteTime` are not included in the text representation. Add or exclude more cell metadata with this option.

6.3.2 Jupytext configuration file

Possible locations and formats

Jupytext's contents manager, and the command line interface, can load some configuration options from a configuration file.

The configuration file should be either in the local or a parent directory, or in any directory listed in

```
from jupytext.config import global_jupytext_configuration_directories
list(global_jupytext_configuration_directories())
```

which include `XDG_CONFIG_HOME` (defaults to `$HOME/.config`) and `XDG_CONFIG_DIR`.

The name for the configuration file can be any of `jupytext.config.JUPYTEXT_CONFIG_FILES`, i.e. `.jupytext` (in TOML), `jupytext.toml`, `jupytext.yml`, `jupytext.yaml`, `jupytext.json` or `jupytext.py` (dot-files like `.jupytext.toml` are accepted by the CLI version of Jupytext, but are not effective in Jupyter). Alternatively, if you are using it, you can also use your Python project's `pyproject.toml` file by adding configuration to a `[tool.jupytext]` table within it.

If you want to know, for a given directory, which configuration file Jupytext is using, please execute:

```
from jupytext.config import find_jupytext_configuration_file
find_jupytext_configuration_file('.')
```

If you want to limit the search for a configuration file to a given parent directory, you can create an empty `.jupytext` configuration file in that directory. Alternatively, you can set the search boundaries with an environment variable `JUPYTEXT_CEILING_DIRECTORIES` - a colon-separated list of absolute paths.

If `JUPYTEXT_CEILING_DIRECTORIES` is defined, Jupytext will stop searching for configuration files when it meets one of these path. This can be helpful to avoid searching for configuration files on slow filesystems. It can also be useful if you don't want to use a global configuration - for instance, when running `pytest` on Jupytext, we use `JUPYTEXT_CEILING_DIRECTORIES="/tmp"`.

Configuring paired notebooks globally

The examples below assume that you use a `.jupytext`, `jupytext.toml` or `.jupytext.toml` Jupyter configuration file in TOML format. If you use another extension, please adapt the examples. For instance, if you want to use `jupytext.yml` in YAML format, replace the = sign with `:` and remove the double quotes. See also `test_config.py` for short examples in all the supported formats.

Also, the examples are for Jupytext 1.11.0 or later. If you are using an older version, you should consult the previous version of this documentation.

Say you want to always associate every `.ipynb` notebook with a `.md` file (and reciprocally). This is done by adding the following to your `jupytext.toml` or `.jupytext.toml` Jupyter configuration file:

```
# Always pair ipynb notebooks to md files
formats = "ipynb,md"
```

If you prefer to use a default `ipynb` - `py:percent` pairing, then that would be:

```
# Always pair ipynb notebooks to py:percent files
formats = "ipynb,py:percent"
```

or alternatively, using an explicit format list:

```
# Always pair ipynb notebooks to py:percent files
formats = ["ipynb", "py:percent"]
```

If you wish to use the `pyproject.toml` config file rather than `jupytext.toml`, you just need to create a `[tool.jupytext]` section in the `pyproject.toml` file, like here:

```
[tool.jupytext]
formats = "ipynb,py:percent"
```

You can pair notebooks in trees with a `root_prefix` separated with three slashes, e.g.

```
# Pair notebooks in subfolders of 'notebooks' to scripts in subfolders of 'scripts'
formats = "notebooks///ipynb,scripts///py:percent"
```

or alternatively, using a dict to map the prefix path to the format name:

```
# Pair notebooks in subfolders of 'notebooks' to scripts in subfolders of 'scripts'
[formats]
"notebooks/" = "ipynb"
"scripts/" = "py:percent"
```

Note that if you are using a `pyproject.toml` file with this dict format, you should make sure the table header is instead `[tool.jupytext.formats]`.

The `root_prefix` is matched with the top-most parent folder of the matching name, not above the Jupytext configuration file.

For instance, with the pairing above, a notebook with path `/home/user/jupyter/notebooks/project1/example.ipynb` will be paired with the Python file `/home/user/jupyter/scripts/project1/example.py`.

In addition to the `root_prefix`, you can use symbolic links if you wish to distribute your notebook folders at different places. Be sure to use symbolic links on folders, not files (#696).

To disable the default pairing for an individual notebook, set formats to a single format, with e.g.:

```
jupytext --set-formats ipynb notebook.ipynb
```

Please note that, while Jupytext is Jupyter acts accordingly to both local or global Jupytext configuration files, the Jupytext menu in Jupyter, and the Jupyter commands in JupyterLab, only display the pairing information set in the notebooks itself and are not aware of the global configuration (#177).

Metadata filtering

You can specify which metadata to include or exclude in the text files created by Jupytext by setting `notebook_metadata_filter` (notebook metadata) and `cell_metadata_filter` (cell metadata) in the configuration file. They accept a string of comma separated keywords. A minus sign `-` in front of a keyword means exclusion.

Suppose you want to keep all the notebook metadata but `widgets` and `varInspector` in the YAML header. For cell metadata, you want to allow `ExecuteTime` and `autoscroll`, but not `hide_output`. You can set

```
notebook_metadata_filter = "all,-widgets,-varInspector"
cell_metadata_filter = "ExecuteTime,autoscroll,-hide_output"
```

If you want that the text files created by Jupytext have no metadata, you may use the global metadata filters below. Please note that with this setting, the metadata is only preserved in the `.ipynb` file.

```
notebook_metadata_filter = "-all"
cell_metadata_filter = "-all"
```

It is possible to filter nested metadata. For example, if you want to preserve the Jupytext metadata, but not the Jupytext version number, you can use:

```
notebook_metadata_filter = "-jupytext.text_representation.jupytext_version"
```

Finally, to hide the notebook metadata in an HTML comment in Markdown files, use the option `hide_notebook_metadata`.

More options

There are a couple more options available - please have a look at the `JupytextConfiguration` class in config.py.

6.4 Notebook formats

Jupytext supports conversion between the `.ipynb` format and many different formats. This page describes each format, as well as some considerations for each.

6.4.1 Markdown formats

Jupytext Markdown

Jupytext can save notebooks as Markdown documents. This format is well adapted to tutorials, books, or more generally notebooks that contain more text than code. Notebooks represented in this form are well rendered by most Markdown editors or renderers, including GitHub.

Like all the Jupytext formats, Jupytext Markdown notebooks start with an (optional) YAML header. This header is used to store selected notebook metadata like the kernel information, together with Jupytext's format and version information.

```
---
jupyter:
  jupytext:
    text_representation:
      extension: .md
      format_name: markdown
      format_version: '1.1'
      jupytext_version: 1.1.0
  kernelspec:
    display_name: Python 3
    language: python
    name: python3
---
```

You can add custom notebook metadata like `author` or `title` under the `jupyter:` section, it will be synchronized with the notebook metadata. And if you wish to export more metadata from the notebook, have a look at the paragraph on *metadata filtering*.

In the Markdown format, markdown cells are inserted verbatim and separated with two blank lines.

If you'd like that cell breaks also occurs on Markdown headers, add a `split_at_heading: true` entry in the `jupytext` section in the YAML header, or if you want that option to be the default for all Markdown documents in Jupyter, activate the option in the *`jupytext.toml` configuration file*:

```
split_at_heading = true
```

Code cells are encoded using the classical triple backticks, followed by the notebook language. Cell metadata are appended after the language information, with a `key=value` syntax, where `value` is encoded in JSON format. For instance, in a Python notebook, a simple code cell with a `parameters` tag is represented as:

````
```python tags=["parameters"]
param = 5
```
````

Code snippets are turned into code cells in Jupyter as soon as they have an explicit language, when that language is supported in Jupyter. Thus, you have a code snippet that you don't want to execute in Jupyter, you can either

- remove the language information,
- or, start the code snippet with a triple tilde, e.g. `~~~python`, instead of ```` ```python ````
- or, add an `active="md"` cell metadata, or a `.noeval` attribute after the language information, e.g. ```` ```python .noeval ````
- or, surround the code snippet with explicit Markdown cell markers (see below).

Raw cells are delimited with HTML comments, and accept cell metadata in the same key=value format:

```
<!-- #raw -->
raw text
<!-- #endraw -->

<!-- #raw key="value"-->
raw cell with metadata
<!-- #endraw -->
```

Markdown cells can also have explicit markers: use one of `<!-- #md -->` or `<!-- #markdown -->` or `<!-- #region -->` and the corresponding `<!-- #end... -->` counterpart. Note that the `<!-- #region -->` and

`<!-- #endregion -->` cells markers are foldable in VS Code, and that you can also insert a title there, e.g. `<!-- #region This is a title for my protected cell -->`. Cell metadata are accepted in the format `key="value"` (`"value"` being encoded in JSON) as for the other cell types.

For a concrete example, see how our `World population.ipynb` notebook in the demo folder is represented in Markdown.

R Markdown

R Markdown is RStudio's format for notebooks, with support for R, Python, and many other languages.

Jupytext's implementation of R Markdown is very similar to that of the Markdown format. The major difference is on code cells, which use R Markdown's convention, i.e. the language and options are surrounded by curly brackets, and the cell metadata are encoded as R objects. For instance our cell with the `parameters` tags would be represented as:

````
```{python tags=c("parameters")}
param = 5
```
````

Python and R notebooks represented in the R Markdown format can run both in Jupyter and RStudio. Note that you can change the default Python environment in RStudio with `RETICULATE_PYTHON` in a `.Renviron` file, see here.

See how our `World population.ipynb` notebook in the demo folder is represented in R Markdown.

MyST Markdown

MyST (Markedly Structured Text) is a markdown flavor that "implements the best parts of reStructuredText". It provides a way to call Sphinx directives and roles from within Markdown, using a *slight* extension of CommonMark markdown. MyST-NB and Jupyter Book builds on this markdown flavor, to offer direct conversion of Jupyter Notebooks into Sphinx documents.

Similar to the jupytext Markdown format, MyST Markdown uses code blocks to contain code cells. The difference though, is that the metadata is contained in a YAML block:

````
```{code-cell} ipython3
---
other:
  more: true
tags: [hide-output, show-input]
---

print("Hallo!")
```
````

The `ipython3` here is purely optional, as an aide for syntax highlighting. In the round-trip, it is copied from `notebook.metadata.language_info.pygments_lexer`.

Also, where possible the conversion will use the short-hand metadata format (see the MyST guide):

````
```{code-cell} ipython3
:tags: [hide-output, show-input]

print("Hallo!")
```
````

Raw cells are also represented in a similar fashion:

````
```{raw-cell}
:raw_mimetype: text/html

<b>Bold text<b>
```
````

Markdown cells are not wrapped. If a markdown cell has metadata, or directly proceeds another markdown cell, then a block break will be inserted above it, with an (optional) single line JSON representation of the metadata:

```
+++ {"slide": true}

This is a markdown cell with metadata

+++

This is a new markdown cell with no metadata
```

See for instance how our `World population.ipynb` notebook is represented in the `myst` format.

Note: The `myst` format requires Python >= 3.6

Tip: You can use the myst-highlight VS Code extension to provide better syntax highlighting for this format.

Pandoc Markdown

Pandoc, the *Universal document converter*, can read and write Jupyter notebooks - see Pandoc's documentation.

In Pandoc Markdown, all cells are marked with pandoc divs (`:::`). The format is therefore slightly more verbose than the Jupytext Markdown format.

See for instance how our `World population.ipynb` notebook is represented in the `md:pandoc` format.

If you wish to use that format, please install `pandoc` in version 2.7.2 or above, with e.g. `conda install pandoc -c conda-forge`.

Quarto

Quarto is a scientific and technical publishing system built on Pandoc. If you have `quarto` installed, Jupytext lets you edit `.qmd` documents as notebooks in Jupyter, and pair `.ipynb` notebooks with `.qmd` notebooks.

The conversion from `.ipynb` to `.qmd` and back directly calls `quarto convert`, and consequently requires an installation of Quarto v0.2.134 or higher.

Note that the round trip of `.ipynb` to `.qmd` to `.ipynb` has the effect of concatenating consecutive Markdown cells and turning raw cells into Markdown cells (since `.qmd` files represent all content as either Markdown or code cells).

6.4.2 Notebooks as scripts

The `light` format

The `light` format was created for this project. That format can read any script in one of these languages as a Jupyter notebook, even scripts which were never prepared to become a notebook.

When a script in the `light` format is converted to a notebook, Jupytext code paragraphs are turned into code cells, and comments that are not adjacent to code are converted to Markdown cells. Cell breaks occurs on blank lines outside of functions, classes or multiline comments.

For instance, in this example we have three cells:

```
# This is a multiline
# Markdown cell

# Another Markdown cell


# This is a code cell
class A():
    def one():
        return 1

    def two():
        return 2
```

Code cells can contain multiple code paragraphs. In that case Jupytext uses an explicit start-of-cell delimiter that is, by default, `# +` (`// +` in C++, etc). The default end of cell delimiter is `# -`, and can be omitted when followed by another explicit start of cell marker, or the end of the file:

```
# +
# A single code cell made of two paragraphs
a = 1


def f(x):
    return x+a
```

Metadata can be associated to a given cell using a key/value representation:

```
# + key="value"
# A code cell with metadata

# + [markdown] key="value"
# A Markdown cell with metadata
```

The `light` format can use custom cell markers instead of `# +` or `# -`. If you prefer to mark cells with VS Code/PyCharm (resp. Vim) folding markers, set `"cell_markers": "region,endregion"` (resp. `"{{{,}}}"`) in the jupytext section of the notebook metadata. If you want to configure this as a global default, add either

```
cell_markers = "region,endregion"  # Use VS Code/PyCharm region folding delimiters
```

or

```
cell_markers = "{{{,}}}"          # Use Vim region folding delimiters
```

to your *jupytext.toml configuration file*.

See how our `World population.ipynb` notebook is represented in that format.

The `nomarker` format

The `nomarker` format is a variation of the `light` format with no cell marker at all. Please note that this format does not provide round-trip consistency - code cells are split on code paragraphs. By default, the `nomarker` format still includes a YAML header - if you prefer to also remove the header, set `"notebook_metadata_filter": "-all"` in the jupytext section of your notebook metadata.

The `percent` format

The `percent` format is a representation of Jupyter notebooks as scripts, in which all cells are explicitly delimited with a commented double percent sign `# %%`. The `percent` format is currently available for these languages.

The format was introduced by Spyder in 2013, and is now supported by many editors, including

- Spyder IDE,
- Hydrogen, a package for Atom,
- VS Code,
- Python Tools for Visual Studio,
- and PyCharm Professional.

Our implementation of the `percent` format is as follows: cells can have

- a title
- a cell type (`markdown`, `md` or `raw`, omitted for code cells)
- and cell metadata like in this example:

```
# %% Optional title [cell type] key="value"
```

In the `percent` format, our previous example becomes:

```
# %% [markdown]
# This is a multiline
# Markdown cell

# %% [markdown]
# Another Markdown cell


# %%
# This is a code cell
class A():
    def one():
        return 1

    def two():
        return 2
```

In the case of Python scripts, Markdown cells do accept multiline comments:

```
# %% [markdown]
"""
This is a Markdown cell
that uses multiline comments
"""
```

By default Jupytext will use line comments when it converts your Jupyter notebooks for `percent` scripts. If you prefer to use multiline comments for all text cells, add a `{"jupytext": {"cell_markers": "\"\"\""}}` metadata to your notebook, either with the notebook metadata editor in Jupyter, or at the command line:

```
jupytext --update-metadata '{"jupytext": {"cell_markers": "\"\"\""}}' notebook.ipynb --
↪to py:percent
```

If you want to use multiline comments for all your paired notebooks, you could also add

```
cell_markers = """"
```

to your *jupytext.toml configuration file*.

See how our `World population.ipynb` notebook is represented in the `percent` format.

The hydrogen format

By default, *Jupyter magics* are commented in the `percent` representation. If you run the percent scripts in Hydrogen, use the `hydrogen` format, a variant of the `percent` format that does not comment Jupyter magic commands.

Sphinx-gallery scripts

Another popular notebook-like format for Python scripts is the Sphinx-gallery format. Scripts that contain at least two lines with more than twenty hash signs are classified as Sphinx-Gallery notebooks by Jupytext.

Comments in Sphinx-Gallery scripts are formatted using reStructuredText rather than markdown. They can be converted to markdown for a nicer display in Jupyter by adding a `sphinx_convert_rst2md = True` line to your Jupytext configuration file. Please note that this is a non-reversible transformation—use this only with Binder. Revert to the default value `sphinx_convert_rst2md = False` when you edit Sphinx-Gallery files with Jupytext.

Turn a GitHub repository containing Sphinx-Gallery scripts into a live notebook repository with Binder and Jupytext by adding only two files to the repo:

- `binder/requirements.txt`, a list of the required packages (including `jupytext`)
- a *jupytext.toml configuration file* with the following contents:

```
preferred_jupytext_formats_read = "py:sphinx"
sphinx_convert_rst2md = true
```

Our sample notebook is also represented in `sphinx` format here.

6.4.3 Jupytext format options

Metadata filtering

All the Jupytext formats (except Sphinx Gallery scripts) store a selection of the notebook metadata in a YAML header at the top of the text file. By default, Jupytext only includes the `kernelspec` and `jupytext` metadata (the remaining notebook metadata are preserved in the `.ipynb` document when you use paired notebook).

If you want to include more (or less) jupyter metadata here, add a `notebook_metadata_filter` option to the `jupytext` metadata. The additional metadata will be added to the `jupyter:` section in the YAML header (or, at the root of the YAML header for the `md:pandoc` and `md:myst` formats). The value for `notebook_metadata_filter` is a comma separated list of additional/excluded (negated) entries, with `all` a keyword that allows to exclude all entries. For instance, if you don't want to store any notebook metadata in the text file, use `notebook_metadata_filter: -all`. If you want to store the whole, unfiltered notebook metadata then use `notebook_metadata_filter: all`. And if you want the default, plus a few specific section, use e.g. `notebook_metadata_filter: section_one, section_two`.

Similarly, cell metadata can be filtered with the `cell_metadata_filter` option. To minimize the differences when a notebook is edited, Jupytext's default cell metadata filter does not include the `autoscroll`, `collapsed`, `scrolled`, `trusted` and `ExecuteTime` cell metadata in the text representation.

Magic commands

Jupyter notebooks often include *magic commands* like `%load_ext` or `%matplotlib inline`. These commands are Jupyter specific and cannot be executed by the classical interpreter.

By default, magic commands are commented out in all the Jupytext formats, with the exception of the Markdown format (not meant to be executed) and the Hydrogen format. You can change this by setting a `comment_magics` option (`true` or `false`) in the Jupytext section.

Active and inactive cells

Sometimes you want a specific cell to be executable only in the `.ipynb` files, or only in the `.py` or `.Rmd` representation. To this end Jupytext introduces the notion of *active* cells.

Mark a code cell with an `"active": "ipynb"` metadata or with an `active-ipynb` tag if you want it to be commented out in the paired script.

Mark a raw cell with an `"active": "py"` metadata or with an `active-py` tag if you want it to be inactive in the notebook but active in the script.

6.5 Supported Languages

Jupytext works with notebooks in any of the following languages:

- Bash
- C#
- C++
- Clojure
- Coconut
- F#

- Groovy
- IDL
- Java
- Javascript
- Julia
- Matlab
- OCaml
- Octave
- PowerShell
- Python
- q/kdb+
- R
- Robot Framework
- Rust/Evxcr
- Sage
- Scala
- Scheme
- Script of Script
- TypeScript
- Haskell
- Tcl

Extending Jupytext to more languages should be easy, see the sections on *contributing to* and *developing* Jupytext.

6.6 Using at the Command Line

6.6.1 Command line conversion

The package provides a `jupytext` script for command line conversion between the various notebook extensions:

```
jupytext --to py notebook.ipynb                 # convert notebook.ipynb to a .py file
jupytext --to py:percent notebook.ipynb         # convert notebook.ipynb to a .py file␣
↪in the double percent format
jupytext --to py:percent --opt comment_magics=false notebook.ipynb   # same as above +␣
↪do not comment magic commands
jupytext --to markdown notebook.ipynb           # convert notebook.ipynb to a .md file
jupytext --output script.py notebook.ipynb      # convert notebook.ipynb to a script.py␣
↪file

jupytext --to notebook notebook.py              # convert notebook.py to an .ipynb file␣
↪with no outputs
jupytext --update --to notebook notebook.py     # update the input cells in the .ipynb␣
↪file and preserve outputs and metadata
```

(continues on next page)

(continued from previous page)

```
jupytext --to md --test notebook.ipynb          # Test round trip conversion

jupytext --to md --output - notebook.ipynb      # display the markdown version on screen
jupytext --from ipynb --to py:percent           # read ipynb from stdin and write double␣
↪percent script on stdout
```

Jupytext has a `--sync` mode that updates all the paired representations of a notebook based on timestamps:

```
jupytext --set-formats ipynb,py notebook.ipynb  # Turn notebook.ipynb into a paired␣
↪ipynb/py notebook
jupytext --sync notebook.ipynb                  # Update whichever of notebook.ipynb/
↪notebook.py is outdated
```

You may also find useful to `--pipe` the text representation of a notebook into tools like `black`:

```
jupytext --sync --pipe black notebook.ipynb    # read most recent version of notebook,␣
↪reformat with black, save
```

To reorder the imports in your notebook, use

```
jupytext --pipe 'isort - --treat-comment-as-code "# %%" --float-to-top' notebook.ipynb
```

(remove the `--float-to-top` argument if you prefer to run `isort` per cell).

For programs that don't accept pipes, use `{}` as a placeholder for the name of a temporary file that will contain the text representation of the notebook. For instance, run `pytest` on your notebook with:

```
jupytext --check 'pytest {}' notebook.ipynb    # export the notebook in format␣
↪py:percent in a temp file, run pytest
```

Read more about running `pytest` on notebooks in our example `Tests in a notebook.md`. Note also that on Windows you need to use double quotes instead of single quotes and type e.g. `jupytext --check "pytest {}" notebook.ipynb`.

Execute `jupytext --help` to access the full documentation.

Execute notebook cells

For convenience, when creating a notebook from text you can execute it:

```
jupytext --set-kernel - notebook.md             # create a YAML header with kernel␣
↪metadata matching the current python executable
jupytext --set-formats md:myst notebook.md      # create a YAML header with an explicit␣
↪jupytext format
jupytext --to notebook --execute notebook.md    # convert notebook.md to an .ipynb file␣
↪and run it
```

If you wanted to convert a collection of Markdown files to paired notebooks, and execute them in the current Python environment, you could run:

```
jupytext --set-formats ipynb,md --execute *.md
```

Advanced usage: error tolerance

If any notebook cell errors, execution will terminate and `jupytext` will not save the notebook. This can cause headaches as the details of any error would be encoded in the notebook, which would not have been saved. But there's an error-tolerant way to execute a notebook: `jupyter nbconvert` has a mode which will still save a notebook if a cell errors, producing something akin to what would happen if you ran all cells manually in Jupyter's notebook UI.

```
# First, convert script (py/sh/R/jl etc) -> notebook. May need additional args to define␣
↪input format etc as above.
jupytext --to ipynb script.py
# Then, execute notebook in place and allowing cells to produce errors
jupyter nbconvert --to ipynb --inplace --execute --allow-errors script.ipynb
# One can also combine these to a single command using jupytext --pipe
jupytext --to ipynb --pipe-fmt ipynb \
  --pipe 'jupyter nbconvert --to ipynb --execute --allow-errors --stdin --stdout' \
  script.py
```

In each of the above, `jupyter nbconvert` could be replaced with any alternative tool to execute a jupyter notebook non-interactively, including papermill which would allow notebook parameterisation (see @mwouts' post on the topic here).

6.6.2 Notebook and cell metadata filters

If you want to preserve (or filter out) certain notebook or cell metadata, change the value of either `notebook_metadata_filter` or `cell_metadata_filter` with the `--update-metadata` option. For instance, if you wish to convert an `.ipynb` document to a `.md` file and preserve all the notebook metadata in that document, run

```
jupytext --to md --update-metadata '{"jupytext": {"notebook_metadata_filter":"all"}}'␣
↪notebook.ipynb
```

Read more on the default and possible values for the metadata filters in *this section*.

6.6.3 Testing the round-trip conversion

Representing Jupyter notebooks as scripts requires a solid round trip conversion. You don't want your notebooks (nor your scripts) to be modified because you are converting them to the other form. Our test suite includes a few hundred tests to ensure that round trip conversion is safe.

You can easily test that the round trip conversion preserves your Jupyter notebooks and scripts. Run for instance:

```
# Test the ipynb -> py:percent -> ipynb round trip conversion
jupytext --test notebook.ipynb --to py:percent

# Test the ipynb -> (py:percent + ipynb) -> ipynb (à la paired notebook) conversion
jupytext --test --update notebook.ipynb --to py:percent
```

Note that `jupytext --test` compares the resulting notebooks according to its expectations. If you wish to proceed to a strict comparison of the two notebooks, use `jupytext --test-strict`, and use the flag `-x` to report with more details on the first difference, if any.

Please note that

- Scripts opened with Jupyter have a default *metadata filter* that prevents additional notebook or cell metadata to be added back to the script. Remove the filter if you want to store Jupytext's settings, or the kernel information, in the text file.
- Cell metadata are available in the `light` and `percent` formats, as well as in the Markdown and R Markdown formats. R scripts in `spin` format support cell metadata for code cells only. Sphinx Gallery scripts in `sphinx` format do not support cell metadata.
- By default, a few cell metadata are not included in the text representation of the notebook. And only the most standard notebook metadata are exported. Learn more on this in the sections for *notebook specific* and *global settings* for metadata filtering.

6.7 Using as a pre-commit hook

Jupytext works well with the pre-commit framework. You can add the following to your `.pre-commit-config.yaml` file to sync all staged notebooks:

```
repos:
-   repo: https://github.com/mwouts/jupytext
    rev: v1.14.0  # CURRENT_TAG/COMMIT_HASH
    hooks:
    - id: jupytext
      args: [--sync]
```

You can provide almost all command line arguments to Jupytext in pre-commit, for example to produce several kinds of output files:

```
repos:
-   repo: https://github.com/mwouts/jupytext
    rev: v1.14.0  # CURRENT_TAG/COMMIT_HASH
    hooks:
    - id: jupytext
      args: [--from, ipynb, --to, "py:percent"]
```

If you are combining Jupytext with other pre-commit hooks, you must ensure that all hooks will pass on any files you generate. For example, if you have a hook for using `black` to format all your python code, then you should use Jupytext's `--pipe` option to also format newly generated Python scripts before writing them:

```
repos:
-   repo: https://github.com/mwouts/jupytext
    rev: v1.14.0  # CURRENT_TAG/COMMIT_HASH
    hooks:
    - id: jupytext
      args: [--sync, --pipe, black]
      additional_dependencies:
        - black==22.3.0 # Matches hook

-   repo: https://github.com/psf/black
    rev: 22.3.0
    hooks:
    - id: black
      language_version: python3
```

Tested examples of how to use the pre-commit hook are available in our tests - see for instance test_pre_commit_1_sync_with_config.py.

6.8 Using as a Python library

Jupytext provides the same `read`, `write`, `reads` and `writes` functions as `nbformat`. You can use `jupytext`'s functions as drop-in replacements for `nbformat`'s ones.

6.8.1 Reading notebooks from many text formats

To read text files as notebooks, simply provide the path to a Jupytext-supported format.

```
import jupytext

# Read a notebook from a file
ntbk = jupytext.read('notebook.md')

# Read a notebook from a string
jupytext.reads(text, fmt='md')
```

Jupytext will read in the content and infer metadata about the file from the YAML header (if there is one). If there is no Jupytext header, then Jupytext will make some assumptions about the format based on the file extension.

This function returns an instance of an `nbformat` `NotebookNode`. You can find more information for working with this notebook representation in the nbformat documentation.

6.8.2 Writing notebooks to many text files

You can also write in-memory notebooks to a variety of text formats by using `jupytext.write`.

Jupytext's implementation provides an additional `fmt` argument, which can be any of the accepted Jupytext extensions (e.g., `py`, `md`, `jl:percent`) If not explicitly provided, the argument is inferred from the file extension.

```
# Return the text representation of a notebook
jupytext.writes(notebook, fmt='py:percent')

# Write a notebook to a file in the desired format
jupytext.write(notebook, 'notebook.py')
jupytext.write(notebook, 'notebook.py', fmt='py:percent')
```

6.9 Sample Use Cases

6.9.1 Writing notebooks as plain text

You like to work with scripts? The good news is that plain scripts, which you can draft and test in your favorite IDE, open transparently as notebooks in Jupyter when using Jupytext. Run the notebook in Jupyter to generate the outputs, *associate* an `.ipynb` representation, save and share your research as either a plain script or as a traditional Jupyter notebook with outputs.

6.9.2 Collaborating on Jupyter Notebooks

With Jupytext, collaborating on Jupyter notebooks with Git becomes as easy as collaborating on text files.

The setup is straightforward:

- Open your favorite notebook in Jupyter notebook
- *Associate* a `.py` representation (for instance) to that notebook
- Save the notebook, and put the Python script under Git control. Sharing the `.ipynb` file is possible, but not required.

Collaborating then works as follows:

- Your collaborator pulls your script.
- The script opens as a notebook in Jupyter, with no outputs (in JupyterLab right-click the script and use the open-with context menu).
- They run the notebook and save it. Outputs are regenerated, and a local `.ipynb` file is created.
- Note that, alternatively, the `.ipynb` file could have been regenerated with `jupytext --sync notebook.py`.
- They change the notebook, and push their updated script. The diff is nothing else than a standard diff on a Python script.
- You pull the changed script, and refresh your browser. Input cells are updated. The outputs from cells that were changed are removed. Your variables are untouched, so you have the option to run only the modified cells to get the new outputs.

6.9.3 Code refactoring

In the animation below we propose a quick demo of Jupytext. While the example remains simple, it shows how your favorite text editor or IDE can be used to edit your Jupyter notebooks. IDEs are more convenient than Jupyter for navigating through code, editing and executing cells or fractions of cells, and debugging.

- We start with a Jupyter notebook.
- The notebook includes a plot of the world population. The plot legend is not in order of decreasing population, we'll fix this.
- We want the notebook to be saved as both a `.ipynb` and a `.py` file: we select *Pair Notebook with a light Script* in either the *Jupytext menu* in Jupyter Notebook, or in the *Jupytext commands* in JupyterLab. This has the effect of adding a `"jupytext": {"formats": "ipynb,py:light"},` entry to the notebook metadata.
- The Python script can be opened with PyCharm:
  - Navigating in the code and documentation is easier than in Jupyter.
  - The console is convenient for quick tests. We don't need to create cells for this.
  - We find out that the columns of the data frame were not in the correct order. We update the corresponding cell, and get the correct plot.
- The Jupyter notebook is refreshed in the browser. Modified inputs are loaded from the Python script. Outputs and variables are preserved. We finally rerun the code and get the correct plot.

6.9.4 Importing Jupyter Notebooks as modules

Jupytext allows to import code from other Jupyter notebooks in a very simple manner. Indeed, all you need to do is to pair the notebook that you wish to import with a script, and import the resulting script.

If the notebook contains demos and plots that you don't want to import, mark those cells as either

- *active* only in the `ipynb` format, with the `{"active": "ipynb"}` cell metadata, or with an `active-ipynb` tag (you may use the jupyterlab-celltags extension for this). Use a `{"active": "ipynb,py"}` metadata or a an `active-ipynb-py` tag if you want the cell to be active only in the `ipynb` and `py` formats, but not in the R Markdown format.
- *frozen*, using the freeze extension for Jupyter notebook.

6.10 Frequently Asked Questions

6.10.1 What is Jupytext?

Jupytext is a Python package that provides *two-way* conversion between Jupyter notebooks and several other text-based formats like Markdown documents or scripts.

6.10.2 Why would I want to convert my notebooks to text?

The text representation only contains the part of the notebook that you wrote (not the outputs). You get a cleaner diff history. Thanks to the *two-way* conversion, you can also act on the text file and then propagate the changes to the original `.ipynb` file. Refactor your code or merge multiple contributions easily!

6.10.3 How do I use Jupytext?

Open the notebook that you want to version control. *Pair* the notebook to a script or a Markdown file using either the *Jupytext Menu* in Jupyter Notebook or the *Jupytext Commands* in JupyterLab.

Save the notebook, and you get two copies of the notebook: the original `*.ipynb` file, together with its paired text representation.

Read more about how to use Jupytext in the *documentation*.

6.10.4 Which Jupytext format do you recommend?

Notebooks that contain more text than code are best represented as Markdown documents. These are conveniently edited in IDEs and are also well rendered on GitHub.

Saving notebooks as scripts is an appropriate choice when you want to act on the code (refactor the code, import it in another script or notebook, etc). Use the `percent` format if you prefer to get explicit cell markers (compatible with VScode, PyCharm, Spyder, Hydrogen…). And if you prefer to get the minimal amount of cell markers, go for the `light` format.

6.10.5 Can I see a sample of each format?

Go to our demo folder and see how our sample `World population` notebook is represented in each format.

6.10.6 Can I edit the paired text file?

Yes! When you're done, reload the notebook in Jupyter. There, you will see the updated input cells combined with the matching output cells from the `.ipynb` file.

6.10.7 Do I need to close my notebook in Jupyter?

No, you don't (*). You can edit the paired text file and simply refresh your navigator to reload the updated input cells. When you refresh the notebook, the kernel variables are preserved, so you can continue your work where you left it.

(*) Please read about Jupyter's autosave below.

6.10.8 How do paired notebooks work?

The `.ipynb` file contains the full notebook. The paired text file only contains the input cells and selected metadata. When the notebook is loaded by Jupyter, input cells are loaded from the text file, while the output cells and the filtered metadata are restored using the `.ipynb` file. When the notebook is saved in Jupyter, the two files are updated to match the current content of the notebook.

6.10.9 How do I remove pairing?

Paired Jupyter notebooks contains specific `jupytext` metadata that you may want to remove. You may want to keep the pairing only while editing the files, and when it comes the time to distribute them, it may make sense to remove the pairing. To do so, you can update the metadata in the `.ipynb` files as follows:

```
jupytext --update-metadata '{"jupytext": null}' path/to/notebooks/*.ipynb
```

6.10.10 Can I create a notebook from a text file?

Certainly. Open your pre-existing scripts or Markdown files as notebooks with a click in Jupyter Notebook, and with the *Open as Notebook* menu in JupyterLab.

In Jupyter Notebook you can also create text notebooks with the *New Text Notebook* menu.

Output cells appear in the browser when you execute the notebook, but they are not written to the disk when you save the notebook.

The output cells are lost when you reload the notebook - if you want to avoid this, just *pair* the text file to an `.ipynb` file.

If you want to convert text formats to notebooks programmatically, use one of

```
jupytext --to ipynb *.md                          # convert all .md files to notebooks␣
↪with no outputs
jupytext --to ipynb --execute *.md                # convert all .md files to notebooks and␣
↪execute them
jupytext --set-formats ipynb,md --execute *.md  # convert all .md files to paired␣
↪notebooks and execute them
```

Conversions the other way use a similar format

```
jupytext --to md *.ipynb                    # convert all .ipynb files to .md files
```

6.10.11 I want a specific cell to be commented out in the paired script

That's possible! See how to *activate or deactivate cells*.

6.10.12 Which files should I version control?

Unless you want to version the outputs, you should version *only the text representation*. The paired `.ipynb` file can safely be deleted. It will be recreated locally the next time you open the notebook (from the text file) and save it.

Note that if you version both the `.md` and `.ipynb` files, you can configure `git diff` to ignore the diffs on the `.ipynb` files.

6.10.13 I have modified a text file, but git reports no diff for the paired `.ipynb` file

The synchronization between the two files happens when you reload and *save* the notebook in Jupyter, or when you explicitly run `jupytext --sync`. If you want to force the synchronization on every commit, create a file `.git/hooks/pre-commit` with the following content:

```
#!/bin/sh
jupytext --sync --pre-commit
```

and make it executable:

```
chmod u+x .git/hooks/pre-commit
```

Alternatively, Vim users can give a try to the jupytext.vim plugin.

6.10.14 Jupyter warns me that the file has changed on disk

By default, Jupyter saves your notebook every 2 minutes. Fortunately, it is also aware that you have edited the text file, yielding this message.

You should simply click on *Reload*.

Note you can deactivate Jupyter's autosave function with the Jupytext Menu in Jupyter Notebook, and with the *Autosave Document* setting in JupyterLab. If you want to permanently deactivate autosave in Jupyter Notebook, use a `custom.js` file:

```
mkdir -p ~/.jupyter/custom
echo "Jupyter.notebook.set_autosave_interval(0);" >> ~/.jupyter/custom/custom.js
```

6.10.15 When I reload, Jupyter warns me that my notebook has unsaved changes

Oh - you have edited both the notebook and the paired text file at the same time? If you know which version you want to keep, save it and reload the other. If you want to compare and merge both versions, backup the text file (with e.g. `git stash`), save the notebook, and merge the updated paired file with the backup (with e.g. `git stash pop`). Then, refresh the notebook in Jupyter.

If your IDE has the ability to compare the changes in memory versus on disk (like PyCharm), you can simply save the notebook and let your IDE do the merge.

6.10.16 Jupyter complains that the `.ipynb` file is more recent than the text representation

This happens if you have edited the `.ipynb` file outside of Jupyter. It is a safeguard to avoid overwriting the input cells of the notebook with an outdated text file.

Manual action is requested as the paired text representation may be outdated. Please edit (`touch`) the paired `.md` or `.py` file if it is not outdated, or if it is, delete it, or update it with

```
jupytext --sync notebook.ipynb
```

6.10.17 Can I use Jupytext with JupyterHub, Binder, Nteract, Colab, Saturn or Azure?

Jupytext is compatible with JupyterHub (execute `pip install jupytext --user` to install it in user mode) and with Binder (add `jupytext` to the project requirements and `jupyter lab build` to `postBuild`).

If you use another editor than Jupyter Notebook, Lab or Hub, you probably can't get Jupytext there. However you can still use Jupytext at the command line to manually sync the two representations of the notebook:

```
jupytext --set-formats ipynb,py:light notebook.ipynb   # Pair a notebook to a light␣
↪script
jupytext --sync notebook.ipynb                         # Sync the two representations
```

6.10.18 Can I re-write my git history to use text files instead of notebooks?

Indeed, you could substitute every `.ipynb` file in the project history with its Jupytext Markdown representation.

Technically this is available in just one command, which results in a complete rewrite of the history. Please experiment that in a branch, and think twice before pushing the result...

```
git filter-branch --tree-filter 'jupytext --to md */*.ipynb && rm -f */*.ipynb' HEAD
```

See the result and the cleaner diff history in the case of the Python Data Science Handbook.

6.11 Demos and Tutorials

Looking for a demo?

- Read the original announcement in *Towards Data Science* (Sept. 2018),
- Watch the PyParis talk (Nov. 2018),
- Read our article on Jupytext and Papermill in *CFM Insights* (Sept. 2019)
- See how you can edit Jupyter Notebooks in VS Code or PyCharm with (or without!) Jupytext (Jan. 2020)
- Watch the JupyterCon 2020 talk on Jupytext (Oct. 2020),
- or, try Jupytext online with binder!

6.12 Contributing

Thanks for reading this. Contributions to this project are welcome. And there are many ways you can contribute…

6.12.1 Spread the word

You like Jupytext? Probably your friends and colleagues will like it too. Show them what you've been able to do with: version control, collaboration on notebooks, refactoring of notebooks, notebooks integrated in library, notebook generated from Markdown documents…

By the way, we're also interested to know how you use Jupytext! There may well be applications we've not thought of!

6.12.2 Improve the documentation

You think the documentation could be improved? You've spotted a typo, or you think you can rephrase a paragraph to make is easier to follow? Please follow the *Edit on Github* link, edit the document, and submit a pull request.

6.12.3 Report an issue

You have seen an issue with Jupytext, or you can't find your way in the documentation? Please let us know, and provide enough information so that we can reproduce the problem.

6.12.4 Propose enhancements

You want to submit an enhancement on Jupytext? Unless this is a small change, we usually prefer that you let us know beforehand: open an issue that describe the problem you want to solve.

6.12.5 Add support for another language

A pull request for which you do not need to contact us in advance is the addition of a new language to Jupytext. In principle that should be easy - you would only have to:

- document the language extension and comment by adding one line to `_SCRIPT_EXTENSIONS` in `languages.py`.
- add the language to `docs/languages.md`
- contribute a sample notebook in `tests/notebooks/ipynb_[language]`.
- run the tests suite (with just `pytest`). The mirror tests will generate various text representations corresponding to your notebook under `tests/notebooks/mirror/`. Please verify that these files are valid scripts, and commit them.

6.13 Developing Jupytext

6.13.1 How to test development versions from GitHub

If you want to test a feature that has been integrated in `main` but not delivered yet to `pip` or `conda`, use

```
pip install git+https://github.com/mwouts/jupytext.git
```

If you want to test Jupytext in JupyterLab 3 then you will have to build the extension for JupyterLab. To do so, make sure that you have a recent version of `node`, and prefix the command above with `BUILD_JUPYTERLAB_EXTENSION=1`.

Finally, if you want to test a development branch, use

```
pip install git+https://github.com/mwouts/jupytext.git@branch
```

where `branch` is the name of the branch you want to test (again, prefix the command above with `BUILD_JUPYTERLAB_EXTENSION=1` if you want to use Jupytext within JupyterLab 3).

6.13.2 Install and develop Jupytext locally

Most of Jupytext's code is written in Python. To develop the Python part of Jupytext, you should clone Jupytext, then create a dedicated Python env:

```
cd jupytext
conda env create --file environment.yml  # or conda env update --file ...
conda activate jupytext-dev
python -m ipykernel install --name jupytext-dev --user
pip install -e .
```

We use the pre-commit package to run pre-commit scripts like `black` and `flake8` on the code. Install it with

```
pre-commit install
```

Tests are executed with `pytest`. You can run them in parallel with for instance

```
pytest -n 5
```

We also have a `tox.ini` file available if you wish to test your contribution on multiple version of Python before making a PR - just run `tox`.

Build the `jupytext` package and install it with

```
BUILD_JUPYTERLAB_EXTENSION=1 python setup.py sdist bdist_wheel
pip install dist/jupytext-x.x.x-py3-none-any.whl
```

or with

```
BUILD_JUPYTERLAB_EXTENSION=1 pip install .
```

Finally, note that you can remove `BUILD_JUPYTERLAB_EXTENSION=1` if you don't need the lab extension, and if you don't want to install `nodejs` or if you want a faster build.

6.13.3 Jupytext's extension for Jupyter Notebook

Our extension for Jupyter Notebook adds a Jupytext entry to Jupyter Notebook Menu. The code is found at `jupytext/nbextension/index.js`. Instructions to develop that extension are at `jupytext/nbextension/README.md`.

6.13.4 Jupytext's extension for JupyterLab

Our extension for JupyterLab adds a series of Jupytext commands to JupyterLab. The code is in `packages/labextension`. See the `README.md` there for instructions on how to develop that extension.

6.13.5 Jupytext's documentation

Install the documentation tools with

```
conda env create --file docs/environment.yml
conda activate jupytext-docs
cd docs
```

and build the HTML documentation locally with

```
rm -rf _build
make html
```

6.14 Jupytext ChangeLog

6.14.1 1.14.1 (2022-07-29)

Fixed

- The timestamp of a paired notebook is the timestamp of the most recent paired file. This fixes the warning "File Changed" after reloading the notebook in Jupyter (#978).

6.14.2 1.14.0 (2022-07-03)

Changed

- The Jupytext configuration file has a new option `cm_config_log_level` that defaults to `info_if_changed`. With that value, the contents manager will log a line regarding the configuration file used only when the config file is not the same as the one previously used (#959) - many thanks to R.C. Thomas for suggesting this and thoughtfully testing the patch.
- Hidden configuration files like `.jupytext.toml` or `.jupytext.py` are now ignored by Jupytext's contents manager when `allow_hidden=False` (that option was introduced in `jupyter_server==2.0.0a1`) (#964).
- We have changed `jupytext --set-formats` to make it more similar to `jupytext --sync`. Now `--set-formats` will not override existing paired files anymore (#969).

Added

- We have added a test `test_pre_commit_hook_sync_with_no_config` that documents how to use the pre-commit hook without a configuration file (#967)

6.14.3 1.13.8 (2022-04-04)

Fixed

- Text-only notebooks are always trusted (as they don't include any output cells) (#941)
- We made sure that our tests also work in absence of a Python kernel (#906)
- The coverage of the `tests` folder has been restored at 100%
- Bash commands like `!{cmd}` are now correctly escaped in the `py:percent` format (#938)

Added

- Added Tcl as a supported language (#930) - thanks to shishitao for this contribution
- Added Maxima as a supported language (#927) - thanks to Alberto Lusiani for contributing a sample Maxima notebook.

Changed

- The Jupytext contents manager is derived from the `LargeFileManager` imported from `jupyter_server` rathen than `notebook` (#933)
- Allow for markdown-it-py v2 (#924)
- We have updated the hooks used in the test pre-commits, to fix an issue on the CI (#940, #942)
- We updated the `yarn.lock` file for the jupyter lab extension to address security vulnerabilities (#904, #925, #935, #939)

6.14.4 1.13.7 (2022-02-09)

Fixed

- The Jupytext CLI only suggest `--update` when the target is an .ipynb file (#905) - thanks to st– for this contribution
- We made sure that commands like `cat notebook.md | jupytext --execute` work (#908)

Added

- Added Haskell as supported language (#909) - thanks to codeweber for this contribution

Changed

- We have updated the pre-commit hooks and in particular we switched to the first stable version of `black==22.1.0`.
- We require `pandoc==2.16.2` for testing. The representation for code cells changed from ```` ``` {.python} ```` to ```` ``` python ```` in that version of Pandoc (#906). We don't use `pandoc>=2.17` in tests at the moment because of the introduction of cell ids that cannot be filtered.
- Jupytext will not add anymore a UTF-8 encoding on Python scripts when the notebook contains non-ascii characters (#907)
- We have added `pyupgrade` to the pre-commit hooks used for developing Jupytext (#907)

6.14.5 1.13.6 (2022-01-11)

Fixed

- The `text_representation` metadata of text notebooks is filtered from `.ipynb` files both in `jupytext.write` and in the contents manager for Jupyter (#900)

Changed

- Jupytext will not issue a warning when a format suffix starting with '.', '-' or '_' is passed to the `--to` option (#901)

6.14.6 1.13.5 (2021-12-27)

Fixed

- Jupytext will not open a text notebook that is not UTF-8 (#896)

6.14.7 1.13.4 (2021-12-12)

Changed

- The test suite filters the warnings that don't belong to Jupytext (#823)

Fixed

- The parsing of notebooks that don't have a YAML header (like `docs/formats.md`) was improved.
- The test suite works with `pytest-randomly` (#838)

6.14.8 1.13.3 (2021-12-04)

Changed

- The "Jupytext Notebook" factory that lets the user configure the Notebook viewer as the default for text notebooks accepts more filetypes: "myst", "r-markdown" and "quarto" (#803)
- Empty MyST Markdown files are valid notebooks (#883)
- Jupytext also works with `markdown-it-py` v2.0 (#885)

6.14.9 1.13.2 (2021-11-30)

Changed

- The extension for Jupyter Lab benefited from a series of improvements contributed by Frédéric Collonval:
  - A new "Jupytext Notebook" factory offers the option to open text notebooks directly with the notebook view (#803). To use it, follow the instructions in the documentation.
  - The ICommandPalette is optional, for compatibility with RISE within JupyterLab [RISE#605](https://github.com/damianavila/RISE/pull/605)
  - Added support for translation
- Branch `master` was renamed to `main` (links in the documentation were updated)

6.14.10 1.13.1 (2021-10-07)

Fixed

- The magic commands in `py:percent` scripts with no explicit format information remain commented over a round trip (#848)

6.14.11 1.13.0 (2021-09-25)

Added

- The Jupytext CLI has a new `--diff` command to show the differences between two notebooks (and if you want to see the changes in a file being updated by Jupytext, use `--show-changes`) (#799)
- Jupyter will show the diff between text and `ipynb` paired notebooks when it cannot open a paired notebook because the `ipynb` version is more recent. Also, if the inputs in the two files are identical then the notebook will open with no error (#799)
- The `py:percent` format will use raw strings when encoding Markdown cells as string, if they contain backslash characters (#836)

Fixed

- We have upgraded the jupyterlab extension dependencies and especially `ansi-regex` to fix a security vulnerability (#857)

Changed

- The Jupytext configuration file is reloaded only when a notebook is opened, saved, or when a different folder is explored (#797)

6.14.12 1.12.0 (2021-09-09)

Added

- Jupytext supports Quarto notebooks (with `.qmd` extension) (#837)
- Jupytext can be configured through the `pyproject.toml` file. Thanks to Robin Brown for this contribution! (#828)
- Jupytext now supports OCaml files with `.ml` extension. Thanks to Quentin Fortier for getting this started (#832)

Fixed

- Added more test to make sure that notebooks can be trusted. In practice, notebooks could not be trusted in JupyterLab<3.0.13 because of the absence of cell ids (#826)

6.14.13 1.11.5 (2021-08-31)

Fixed

- Fixed typos revealed by `codespell` - thanks to @hectormz for this contribution (#829)
- We updated the dependencies of the `jupyterlab-jupytext` extension to address several security issues (#842) (#843)
- The Jupytext dev environment (`requirements-dev.txt`) now uses `jupyterlab==3.0.17` rather than `3.0.0` because of another security issue (#839)

6.14.14 1.11.4 (2021-07-14)

Changed

- The documentation illustrates how the `cell_markers` option (and the other ones) can be set directly in the `jupytext.toml` config file (#809).
- The dependency on `mdit-py-plugins` through `markdown-it-py[plugins]` was made explicit (#814)

Fixed

- System assigns of the form `var = !cmd` are commented out (#816)
- Fixed an `InconsistentPath` issue with notebooks paired with scripts in a folder. The prefix in the Jupytext formats always use /, while paths might use either / or \ (#806)
- Tests that cannot succeed are skipped when either the Jupytext folder is not a git repository, when `sphinx-gallery` is too recent, or when `pandoc` is not up-to-date (#814)
- Removed the mention of '–update' in 'jupytext –pipe' since outputs are preserved already

6.14.15 1.11.3 (2021-06-10)

Changed

- Jupytext CLI has a new option `--use-source-timestamp` that sets the last modification time of the output file equal to that of the source file (this avoids having to change the timestamp of the source file) (#784)
- In the pre-commit mode, Jupytext now uses the commit timestamp to determine which file in the pair is the most recent (#780)

Fixed

- Dependencies of the JupyterLab extension have been upgraded to fix a security vulnerability (#798)
- The `--warn-only` option also applies to pipes. Use this if the pipe may fail, e.g. if you apply `black` on a possibly invalid script (#781)
- Variables assigned from a magic command are commented out in `py` scripts (#781)
- Fixed a round-trip issue on notebooks that have None/null in their metadata (#792)

6.14.16 1.11.2 (2021-05-02)

Changed

- Jupytext's dependency markdown-it-py is now in v1 (#769)
- The optional argument `fmt` in `jupytext.reads` now has the default value `None` - thanks to Yuvi Panda (#763)

Fixed

- All text files are opened with an explicit `utf-8` encoding (#770)
- Previously `--pipe black` was not always putting two blank lines between functions. To fix that we load the internal Jupytext cell metadata like `lines_to_next_cell` from the text file rather than ipynb (#761)
- The timestamp of the source file is not updated any more when the destination file is not in the pair (#765, #767)

Added

- A new test documents when the `ipython3` pygment lexer appears in MyST Markdown files (#759)

6.14.17 1.11.1 (2021-03-26)

Fixed

- Format options stored in the notebook itself are now taken into account (Fixes #757)

6.14.18 1.11.0 (2021-03-18)

Fixed

- The `jupytext.toml` config file can now be used together with the `jupytext` pre-commit hook (#752)
- The `notebook_extensions` option of the `jupytext.toml` file now works (#746)

Changed

- The options in `jupytext.toml` where renamed to match the `jupytext` metadata in the text notebooks. One should now use `formats` rather than `default_jupytext_formats` and `notebook_metadata_filter` rather than `default_notebook_metadata_filter` (#753)

6.14.19 1.10.3 (2021-03-07)

Fixed

- We have updated `marked`, an indirect dependency of the `jupyterlab-jupytext` extension, to fix a moderate vulnerability (#750).
- We use non-random cell ids in the tests to avoid test failures due to duplicate cell ids (#747)

6.14.20 1.10.2 (2021-02-17)

Fixed

- We have adjusted the `MANIFEST.in` file to exclude the `node_modules` but still include the JupyterLab extension that was missing in the `.tar.gz` (and conda) package in v1.10.1. Many thanks to Martin Renou for providing the fix at (#741)

6.14.21 1.10.1 (2021-02-11)

Added

- The recursive glob pattern `**/*.ipynb` is now supported by Jupytext - Thanks to Banst for this contribution (#731)
- Sage notebooks are supported. They can be converted to `.sage` and `.md` files and back. Thanks to Lars Franke for suggesting this! (#727)
- Jupytext is also accessible with `python -m jupytext`. Thanks to Matthew Brett for his PR! (#739)

Changed

- We have tested Jupytext with the new cell ids introduced in `nbformat>=5.1.0`. Cell ids are preserved by the `--sync` and `--update` command. So we removed the constraint on the version of `nbformat` (#735).

Fixed

- We filtered out the `node_modules` folder from the `.tar.gz` package for Jupytext (#730)

6.14.22 1.10.0 (2021-02-04)

Added

- Jupytext has a pre-commit hook! Many thanks to John Paton and Aaron Gokaslan for making this happen (#698)
- Jupytext CLI will not rewrite files that don't change (#698).
- If you want to see the diff for changed files, use the new `--diff` option (#722)
- We have added `isort` and `autoflake8` to the `pre-commit` configuration file used for developing the Jupytext project (#709)
- We made sure that `py:percent` scripts end with exactly one blank line (#682)
- We checked that Jupytext works well with symbolic links to folders (not files!) (#696)

Changed

- Jupytext does not work properly with the new cell ids of the version 4.5 of `nbformat>=5.1.0` yet, so we added the requirement `nbformat<=5.0.8` (#715)

- Jupytext will issue an informative error or warning on notebooks in a version of nbformat that is not known to be supported (#681, #715)

Fixed

- Code cells that contain triple backticks (or more) are now encapsulated with four backticks (or more) in the Markdown and MyST Markdown formats. The version number for the Markdown format was increased to 1.3, and the version number for the MyST Markdown format was increased to 0.13 (#712)
- Indented magic commands are supported (#694)

6.14.23 1.9.1 (2021-01-06)

Fixed

- Include the lab extension that was missing in the conda package (#703).

6.14.24 1.9.0 (2021-01-05)

Changed

- The Jupytext extension for JupyterLab is compatible with Jupyter Lab 3.0, thanks to Martin Renou's awesome contribution (#683).

6.14.25 1.8.2 (2021-01-04)

Changed

- Jupytext 1.8.2 depends on `python>=3.6`. The last version of Jupytext explicitly tested with Python 2.7 and 3.5 was Jupytext 1.7.1 (#697).

6.14.26 1.8.1 (2021-01-03)

Changed

- The dependency on `markdown-it-py` is conditional on `python>=3.6` (#697)

6.14.27 1.8.0 (2020-12-22)

Changed

- Removed support for Python 2.7 and 3.5, a preliminary step towards a JupyterLab 3.0-compatible extension (#683)
- The MyST Markdown format uses `markdown-it-py~=0.6.0` (#692)

6.14.28 1.7.1 (2020-11-16)

Fixed

- Text notebooks have the same format and mimetype as ipynb notebooks. This fixes the *File Load Error - content.indexOf is not a function* error on text notebooks (#659)

6.14.29 1.7.0 (2020-11-14)

Changed

- Jupytext's contents manager uses the parent CM's `get` and `save` methods to read and save text files, and explicitly calls `jupytext.reads` and `jupytext.writes` to do the conversion. We don't use `mock` nor internal parent methods any more. Thanks to Max Klein for helping making this work! (#634, #635)
- Thanks to the above, Jupytext can work on top of contents manager that don't derive from `FileContentsManager`, and in particular it works with `jupyterfs` (#618)
- The documentation was reorganized. `README.md` was simplified and now includes many links to the documentation.
- The documentation now uses `myst_parser` rather than `recommonmark`. And we use `conda` on RTD (#650, #652)
- The `readf` and `writef` functions were dropped (they had been deprecated in favor of `read` and `write` in June 2019, v1.2.0)
- The description & dependencies of the JupyterLab extension were updated (#654)
- The `--set-kernel -` command, on a Python notebook, gives an explicit error when no kernel is not found that matches the current Python executable.
- All the GitHub workflow files were concatenated into a unique file, and we have added an `pypi-publish` step to automatically publish the package on PyPi when new releases are created.
- The `CHANGELOG.md` file was moved under `docs` to better expose the history of changes.

Added

- Configuration errors are reported in the console and/or in Jupyter (#613)
- Jupytext's Contents Manager internal errors are logged on the console, and trigger an HTTP Error 500 (#638)
- The GitHub actions run on both push events and pull requests, and duplicate jobs are skipped (#605)
- Jupytext has a `tox.ini` file, thanks to Chris Sewell (#605)
- Jupytext is tested against Python 3.9
- The `execution` cell metadata is now filtered by default (#656)

Fixed

- Optional dependency on `sphinx-gallery` frozen to version `~=0.7.0` (#614)
- Codecov/patch reports should be OK now (#639)
- Jupytext tests work on non-English locales (#636)
- Cell metadata that are already present in text notebook can be filtered out using a config file (#656)
- Optional cell attributes like attachments are preserved (#671)

6.14.30 1.6.0 (2020-09-01)

Added

- New option `hide_notebook_metadata` to encapsulate the notebook metadata in an HTML comment (#527)
- New option `root_level_metadata_as_raw_cell`. Set it to `False` if you don't want to see root level metadata of R Markdown notebooks as a raw cell in Jupyter (#415)
- New option `doxygen_equation_markers` to translate Markdown equations into Doxygen equations (#517)
- New option `custom_cell_magics` to comment out cells starting with user-specific cell magics (#513)
- Documented how to use `isort` on notebooks (#553)
- `jupytext notebook.ipynb --to filename.py` will warn that `--to` is used in place of `--output`.
- `jupytext --set-formats filename.py` will suggest to use `--sync` instead of `--set-formats` (#544)
- Warn if 'Include Metadata' is off when saving text files in Jupyter (#561)
- Test that notebooks paired through a configuration file are left unmodified (#598)
- Test that metadata filters in the configuration files are taken into account when using `jupytext --to` (#543)
- New argument `--run-path` to execute the notebooks at the desired location (#595)
- Activated GitHub code scanning alerts

Changed

- Jupytext now depends on `markdown-it-py` (Python 3.6 and above) and always features the MyST-Markdown format, thanks to Chris Sewell (#591)
- The `md:myst` and `md:pandoc` are always included in the Jupytext formats, and an informative runtime error will occur if the required dependencies, resp. `markdown-it-py` and `pandoc`, are not installed. (#556)
- The `# %%` cell marker has the same indentation as the first line in the cell (#562)
- Jupytext is now installed from source on MyBinder to avoid cache issues (#567)
- The tests that execute a notebook are now skipped on Windows to avoid timeout issues (#489)

Fixed

- Only scripts can have an encoding comment, not Markdown or R Markdown files (#576)
- Spaces in `--pipe` commands are supported (#562)
- Bash commands starting with special characters are now correctly detected, thanks to Aaron Gokaslan (#587)
- MyST Markdown files are recognized as such even if MyST-Markdown is not available (#556)
- Build JupyterLab with `dev-build=False` and `minimize=False` on mybinder to avoid build errors
- Configured coverage targets in `codecov.yml`

6.14.31 1.5.2 (2020-07-21)

Changed

- The documentation uses the Alabaster theme

Fixed

- Preserve indentation when commenting out magic commands (#437)
- Fixed MyBinder - `jupytext.py` is not a configuration file (#559, #567)

6.14.32 1.5.1 (2020-07-05)

Fixed

- Added `toml` as a dependency (#552).
- Filtered out `__pycache__` and `.pyc` files from the pip package.
- Fixed coverage upload by adding `coverage` as a dependency to the conda CI workflow.
- Fixed the conda CI / Python 2.7 job by explicitly installing `backports.functools_lru_cache` (#554).

6.14.33 1.5.0 (2020-06-07)

Added

- Jupytext can use a local or global configuration file (#508)
- Jupytext can pair notebooks in trees. Use e.g. `notebooks///ipynb,scripts///py:percent` if you want to replicate the tree of notebooks under `notebooks` in a folder named `scripts` (#424)
- The extension for Jupyter Notebook has a *New Text Notebook* menu that creates text-only notebooks (#443)
- Groovy and Java are now supported, thanks to Przemek Wesołek (#500)
- The Coconut language is also supported, thanks to Thurston Sexton (#532)
- Resource files with `.resource` extension from the Robot Framework are supported, thanks to Hiski Valli (#535)
- Jupytext is tested in `pip` and `conda` environments, on Linux, Mac OS and Windows, using Github actions (#487)
- Jupytext uses pre-commit checks and automatic reformatting with `pre-commit`, `black` and `flake8` (#483)
- Documentation improvements:
  - Mention that the YAML header can be created with either `--set-kernel`, `--set-formats`, or both (#485)
  - Mention that one should use double quotes, not single quotes, around `jupytext --check` commands like `"pytest {}"` on Windows (#475)
  - Improved error message when a file is in a version that can't be read by Jupytext (#531)

Fixed

- Skip the `jupytext --execute` tests when the warning *Timeout waiting for IOPub output* occurs, which is the case intermittently on Windows (#489)
- Fixed wrong paired paths when syncing with the –pre-commit flag (#506)

6.14.34 1.4.2 (2020-04-05)

Added

- Added an example with custom notebook metadata (#469)
- Added an example with custom cell tags (#478)

Changed

- The outputs from the `.ipynb` file are matched with the input cells from the text file with less strict rules. In this version, a search and replace on the text file will not remove the outputs any more (#464).
- Update parsing of myst notebooks to new (markdown-it based) parser (please upgrade to `myst-parser` to version `~0.8`) (#473)
- Use `os.path.samefile` when searching for the kernel that corresponds to the current environment (`--set-kernel -`)

Fixed

- Fixed the CLI example for not commenting out magic commands: `--opt comment_magics=false`. In addition, most of the `jupytext` commands in `using-cli.md` are now tested! (#465)
- `jupytext.read` and `jupytext.write` now give more meaningful errors when the format information is incorrect (#462)
- Multiline comments starting or ending with quadruple quotes should not cause issues anymore (#460)
- Fixed active cells in the py:percent format (#477)

6.14.35 1.4.1 (2020-03-19)

Added

- Script of script (SoS) notebooks are now supported. Thanks to Thomas Pernet-coudrier for contributing the sample notebook (#453).
- New MyST Markdown format (`md:myst`), developed by the ExecutableBookProject team. Read more about the MyST Markdown format in the documentation. And many thanks to Chris Sewell for contributing the actual implementation! (#447 #456 #458)

Fixed

- When using `jupytext --pipe cmd`, the output of `cmd` should not appear in the terminal (#432)

6.14.36 1.4.0 (2020-03-09)

Changed

- The new jupyterlab extension (in version 1.2.0) is compatible with JupyterLab 2.0. Many thanks to Jean Helie! (#449)
- It is not compatible with JupyterLab 1.x anymore. If you wish, you can install manually the previous version of the extension with `jupyter labextension install jupyterlab-jupytext@1.1.1`.

Fixed

- Display the output/errors of command executed with `jupytext --pipe` or `jupytext --check` (#432)

6.14.37 1.3.5 (2020-03-08)

Fixed

- Removed leading slash in notebook paths (#444)
- Fixed `jupytext --set-formats` when using formats with prefix and/or suffix (#450)

6.14.38 1.3.4 (2020-02-18)

Added

- C# and F# Jupyter notebooks are now supported (#427, #429)

Fixed

- `jupytext --to script *.ipynb` now computes the script extension for each notebook (#428)
- Fix shebang handling for languages with non-# comments, by Jonas Bushart (#434)
- Indented bash commands are now commented out (#437)
- The main formats are documented in `jupytext --help` (#426, #433)

6.14.39 1.3.3 (2020-01-27)

Added

- Jupytext has a logo! Many thanks to Kyle Kelley for contributing the actual logo (#423), and to Chris Holdgraf for suggesting this (#260).
- Nested metadata filtering is now supported! You can use this to rid of `jupytext_version` if you wish (#416).

Fixed

- Code cells in the Markdown format can contain triple backticks inside multiline strings (#419).
- Changes in the YAML header when running `jupytext --test` on text files are ignored (#414).

6.14.40 1.3.2 (2020-01-08)

Fixed

- The `--pre-commit` mode now ignores non-notebook files in the index (#338).
- `nbformat_minor` is taken from the notebook with outputs When inputs and outputs are merged.

6.14.41 1.3.1 (2019-12-26)

Added

- New `nomarker` format in the Jupyter Notebook and JupyterLab extensions (#397)

Changed

- The `normarker` format replaces the one previously named `bare`.

Fixed

- Multiline strings are now accepted in the YAML header (#404). Fix contributed by ZHUO Qiang (#405).

- Fixed the instructions on how to use multiline comments for all Markdown cells in the py:percent format, by ZHUO Qiang (#403).
- Added `cd` to the list of magic commands.
- Do not read paired files when `--set-formats` is being used (fixes #399).

6.14.42 1.3.0 (2019-11-23)

See also What's new in Jupytext 1.3?

Added

- Pairing a notebook to both `.md` and `.py` is now supported. Input cells are loaded from the most recent text representation (#290)
- Both the Jupyter Notebook and the Jupyter Lab extension for Jupytext were updated (#290)
- Raw cells are now encoded using HTML comments (`<!-- #raw -->` and `<!-- #endraw -->`) in Markdown files (#321)
- Markdown cells can be delimited with any of `<!-- #region -->`, `<!-- #markdown -->` or `<!-- #md -->` (#344)
- Code blocks from Markdown files, when they don't have an explicit language, appear in Markdown cells in Jupyter (#321)
- Code blocks with an explicit language and a `.noeval` attribute are inactive in Jupyter (#347)
- Markdown and raw cells can be quoted with triple quotes in the `py:percent` format. And Markdown cells can start with just `# %% [md]` (#305)
- The light format uses `# + [markdown]` rather than the previous `cell_type` metadata to identify markdown cells with metadata (#356)
- Explicit Markdown cells in the light format `# + [markdown]` can use triple quotes (#356)
- IPython magic help commands like `float?` are classified as magics, and thus commented in Python scripts (#297)
- Cell metadata can be encoded as either `key=value` (the new default) or in JSON. An automatic option `cell_metadata_json` should help minimize the impact on existing files (#344)
- R Markdown hidden inputs, outputs, or cells are now mapped to the corresponding Jupyter Book tags by default (#337)
- The notebook metadata filter is automatically updated when extra keys are added to the YAML header (#376)
- The Jupyter Notebook extension for Jupytext is compatible with Jupyter Notebook 6.0 (#346)
- `jupytext notebook.py --to ipynb` updates the timestamp of `notebook.py` so that the paired notebook still works in Jupyter (#335, #254)
- `jupytext --check pytest notebook.ipynb` can be used to run test functions in a notebook (#286)
- `jupytext --check` and `jupytext --pipe` can run commands that only operate on files: when `{}` is found in the text of the command, `jupytext` saves the text representation of the notebook in a temp file, and replaces `{}` with the name of that file before executing the command. (#286)
- Documented how to sync notebooks in a pre-commit hook (#338)
- Added support for Rust/Evxcr, by Jonas Bushart (#351)
- Added support for Robot Framework, by Asko Soukka (#378)

- Added support for PowerShell (#349)

Changed

- Use `CHANGELOG.md` rather than `HISTORY.rst`

Fixed

- Commands like `cat = x` are not magic commands, so they are not commented any more (#339)
- Fixed an inconsistent round trip (code cell with `"cat"` being converted to a markdown cell) in the `py:light` format (#339)
- `jupytext --test textfile.ext` now really compares the text file to its round trip (rather than the corresponding notebook) (#339)
- Markdown cells that contain code are now preserved in a round trip through the Markdown and R Markdown formats (#361)
- Code cells with a `%%python3` cell magic are now preserved in a round trip through the Markdown format (#365)
- `jupytext --execute` runs the notebook in its folder (#382)
- Strings in the metadata of code cells are quoted in the Rmd representation. And we escape R code in chunk options with `#R_CODE#` (#383)

6.14.43 1.2.4 (2019-09-19)

Added

- The documentation includes a mention on how to set metadata filters at the command line (#330)
- Jupytext will not catch any error when the flag `--warn-only` is not set (#327)

Fixed

- Now the flag `--warn-only` catches every possible error (#263)
- `.md` and `.markdown` files are treated identically (#325)
- Fixed `--set-kernel` when using pipes (#326)
- Fixed utf-8 encoding on stdout on Python 2 (#331)

6.14.44 1.2.3 (2019-09-02)

Fixed

- Dependency on `setuptools` in `pandoc.py` made optional to fix the build of the conda package (#310, #323)

6.14.45 1.2.2 (2019-09-01)

Added

- Documentation includes a section on how to use Jupytext as a Python library (#317)
- Mention of the server extension in the documentation (#304)
- Text notebooks can be tested with `jupytext --execute notebook.md` (#303)
- The default value of `as_version` in `jupytext.read` is `nbformat.NO_CONVERT`, as for `nbformat.read`
- Jupytext tests are now included in sdist (#310)

Fixed

- Fixed the usability of the `fmt` argument in `jupytext.read` (#312)
- Fixed the download notebook error when `c.notebook_extensions` has a custom value (#318)
- String delimiters in commented text are now ignored (#307)

6.14.46 1.2.1 (2019-07-20)

Added

- Added documentation on how to use Jupytext with JupyterLab 0.35 (#299)
- and on using Jupytext with the pre-commit package manager (#292)
- The `read` and `write` functions are easier to use (#292)

Fixed

- Jupytext now ships the `jupyterlab-jupytext` extension in version 1.0.2. The version 1.0.1 erroneously introduces a `target_formats` metadata in the jupytext section, instead of `formats`, and works only after two clicks.

6.14.47 1.2.0 (2019-07-18)

Added

- New `--execute` option in Jupytext CLI (#231)
- The `--set-formats` option in Jupytext CLI also triggers `--sync`, allowing shorter commands.
- `jupytext`'s `read` and `write` functions can be used as drop-in replacements for `nbformat`'s ones (#262).
- `jupytext --sync` will now skip unpaired notebooks (#281).
- The JupyterLab extension was updated. It now works on on text files (#213) and has a new option to include (or not) the metadata in the text representation of the notebook.
- Jupytext's contents manager class is derived dynamically from the default CM class for compatibility with `jupyter_server` (#270)
- Removed dependency on `mock` and `testfixtures`, thanks to Jean-Sebastien Dieu (#279)
- Added support for `.markdown` extension (#288)

Fixed

- The `jupyterlab-jupytext` extension shipped with the python package is in version 1.0.1, and is compatible only with JupyterLab >= 1.0. If you use an earlier version of JupyterLab, please install the version 0.19 of the extension with `jupyter labextension install jupyterlab-jupytext@0.19` (#276, #278)
- Text files can be unpaired (#289)

6.14.48 1.1.7 (2019-06-23)

Added

- Added support for Scala notebook, by Tobias Frischholz (#253)
- Added a getting started notebook for jupytext (and Binder), by Chris Holdgraf (#257)
- The Markdown and R Markdown representations are now tested for all the languages
- The Jupytext notebook extension also works when the notebook is a text file (#213)

Fixed

- The Jupytext Menu in Jupyter Notebook is now compatible with `jupyter_nbextensions_configurator` (#178)
- Entries in the Jupytext menu are updated when the menu is hovered on (#248)
- Fixed link to `.md` files in the documentation (#255)

6.14.49 1.1.6 (2019-06-11)

Added

- Jupytext now supports Javascript and Typescript, thanks to Hatem Hosny (#250)
- Jupytext works with Python 3.8 as well

Fixed

- Fix global `auto` pairing (#249)

6.14.50 1.1.5 (2019-06-06)

Fixed

- Fixed implicit dependency on `jupyter_client` (#245)

6.14.51 1.1.4 (2019-06-05)

Added

- New argument `--set-kernel` in Jupytext command line (#230)
- Jupytext now accepts `--to script` or `--to auto` (#240)
- Jupytext now has a real Sphinx documentation on readthedocs, thanks to Chris Holdgraf (#237)

Fixed

- Invalid notebooks may cause a warning, but not a fatal error (#234)
- Jupyter server extension leaves the contents manager unchanged if it is a sub-class of Jupytext's CM (#236)
- Fixed format inference when metadata is present but not format information (#239)
- Preserve executable and encoding information in scripts with metadata (#241)

6.14.52 1.1.3 (2019-05-22)

Added

- Support for IDL notebooks and .pro scripts (#232)

6.14.53 1.1.2 (2019-05-16)

Added

- Jupytext's content manager has a new `notebook_extensions` option (#224, #183)
- Cells can be made inactive in scripts with the `active-ipynb` cell tag (#226)

Fixed

- Directories ending in .jl (or .ipynb) are not notebooks (#228)
- Empty notebooks have no language (#227)

6.14.54 1.1.1 (2019-04-16)

Added

- Jupytext server extension leaves the contents manager unchanged when it is already a subclass of TextFileContentsManager (#218)
- The base class for TextFileContentsManager defaults to FileContentsManager when LargeFileManager is not available (#217)

6.14.55 1.1.0 (2019-04-14)

Added

- Markdown and R Markdown formats now support metadata (#66, #111, #188)
- The `light` format for Scripts can use custom cell markers, e.g. Vim or VScode/PyCharm folding markers (#199)
- Pandoc's Markdown format for Jupyter notebooks is available in Jupytext as `md:pandoc` (#208)

Fixed

- Jupytext's contents manager is now based on `LargeFileManager` to allow large file uploads (#210)
- YAML header parsed with yaml.safe_load rather than yaml.load (#215)
- IPython line magic can be split across lines (#209)
- `jupytext --to py` rather than `--to python` in the README (#216)

6.14.56 1.0.5 (2019-03-26)

Fixed

- Fix the error 'notebook file has changed on disk' when saving large notebooks (#207)

6.14.57 1.0.4 (2019-03-20)

Added

- Wildcard are now supported on Windows (#202)
- Trusted notebooks remain trusted when inputs cells are modified (#203)

Fixed

- Pre-commit mode adds the result of conversion to the commit (#200)

6.14.58 1.0.3 (2019-03-13)

Added

- Matlab and Octave notebooks and scripts are now supported (#197)

Fixed

- `notebook_metadata_filter = "all"` now works (#196)
- Default pairing in subfolders fixed in Jupyter Lab (#180)

6.14.59 1.0.2 (2019-02-27)

Added

- Rename notebooks in pairs in the tree view (#190)
- Associate `.scm` file extension with Scheme scripts (#192)
- Added support for Clojure, by bzinberg (#193)

Fixed

- Allow spaces between `?` or `!` and python or bash command (#189)

6.14.60 1.0.1 (2019-02-23)

Fixed

- Exclude tests in package deployment (#184)
- Jupytext's serverextension only runs selected init steps (#185)
- Added an additional test for magic arguments (#111)

6.14.61 1.0.0 (2019-02-19)

Added

- Jupytext now includes a Jupyter Notebook extension and a JupyterLab extension (#86).
- Jupytext command line has more arguments: `--paired-paths` to list the paths for the paired representations of the notebook, and `--sync` to synchronise the content of all paired paths based on the most recent file (#146). In addition, the `--from` argument is optional even when the notebook is read from stdin (#148).
- The pairing information, and more generally the notebook metadata can be edited with the CLL, see the `--set-formats` and the `--update-metadata` arguments (#141).
- Jupytext can `--pipe` the text representation of a notebook to external programs like `black` or `flake8` (#154, #142)
- The Python representation of notebooks containing PEP8 cells is now expected to be PEP8 compliant (#154).
- Format specification allow prefix and suffix for path and file name (#138, #142). Use `ipynb, prefix/suffix.py:percent` to pair the current notebook named `notebook.ipynb` to a script named `prefixnotebooksuffix.py`. Suffix and prefix can also be configured on the `ipynb` file, with the same syntax.
- Introducing a new `hydrogen` format for scripts, which derives from `percent`. In that format Jupyter magic commands are not commented (#59, #126, #132).
- Introducing a new `bare` format for scripts, which derives from `light`. That format has no cell marker. Use a notebook metadata filter `{"jupytext": {"notebook_metadata_filter":"-all"}}` if you want no YAML header (#152).
- The default format for R script is now `light`, as for the other languages.
- Added support for q/kdb+ notebooks (#161).
- Python scripts or Markdown documents that have no Jupyter metadata receive a metadata filter that ensures that metadata is not exported back to the text representation (#124).
- Metadata filters are represented as strings rather than dictionaries to make YAML headers shorter. Previous syntax from #105 is still supported. They were also renamed to `notebook_metadata_filter` and `cell_metadata_filter`.
- Markdown and RMarkdown formats have a new option `split_at_heading` to split Markdown cells at heading (#130)

Fixed

- Main language of scripts is inferred from script extension. Fixes a round trip conversion issue for Python notebooks with a Javascript cell.
- Non-Python scripts opened as notebooks in Jupyter are now correctly saved even when no matching kernel is found.
- Jupyter magic commands like `ls` are commented in the light and R markdown format (#149).
- Cell starting with `%%html`, `%%latex` are now commented out in the `light`, `percent` and `Rmd` formats (#179).

6.14.62 0.8.6 (2018-11-29)

Added

- The `language_info` section is not part of the default header any more. Language information is now taken from metadata `kernelspec.language`. (#105).
- When opening a paired notebook, the active file is now the file that was originally opened (#118). When saving a notebook, timestamps of all the alternative representations are tested to ensure that Jupyter's autosave does not override manual modifications.
- Jupyter magic commands are now commented per default in the `percent` format (#126, #132). Version for the `percent` format increases from '1.1' to '1.2'. Set an option `comment_magics` to `false` either per notebook, or globally on Jupytext's contents manager, or on `jupytext`'s command line, if you prefer not to comment Jupyter magics.
- Jupytext command line has a pre-commit mode (#121).

6.14.63 0.8.5 (2018-11-13)

Added

- `bash` scripts as notebooks (#127)
- R scripts with `.r` extension are supported (#122)
- Jupytext selects the first kernel that matches the language (#120)

6.14.64 0.8.4 (2018-10-29)

Added

- Notebook metadata is filtered - only the most common metadata are stored in the text representation (#105)
- New config option `freeze_metadata` on the content manager and on the command line interface (defaults to `False`). Use this option to avoid creating a YAML header or cell metadata if there was none initially. (#110)
- Language magic arguments are preserved in R Markdown, and also supported in `light` and `percent` scripts (#111, #114, #115)
- First markdown cell exported as a docstring when using the Sphinx format (#107)

6.14.65 0.8.3 (2018-10-19)

Added

- Frozen cells are supported in R Markdown, light and percent scripts (#101)
- Inactive cells extended to percent scripts (#108)
- `jupytext` gains a `--version` argument (#103)
- "ExecuteTime" cell metadata is not included in the text representation anymore (#106)

6.14.66 0.8.2 (2018-10-15)

Added

- Round trip conversion testing with `jupytext --test` was improved (#99)
- Round trip conversion tested on Jake Vanderplas' Python for Data Science Handbook.

Fixed

- Nested lists and dictionaries are now supported in notebook metadata
- Final empty code cell supported in Sphinx representation

6.14.67 0.8.1 (2018-10-11)

Fixed

- Sphinx format tested on `World population` notebook (#97)
- Mirror test made stronger on this occasion!
- Markdown representation recognize Julia, Scheme and C++ code cells as such
- Light representation of Scheme and C++ notebooks fixed (#61)

6.14.68 0.8.0 (2018-10-10)

Added

- All `jupytext` related metadata goes to a `jupytext` section (#91). Please make sure your collaborators use the same version of Jupytext, as the new version can read previous metadata, but not the opposite.
- Notebooks extensions can be prefixed with any prefix of at most three chars (#87).
- Export of the same notebook to multiple formats is now supported. To export to all python formats, plus `.ipynb` and `.md`, use `"jupytext": {"formats": "ipynb,pct.py:percent,lgt.py:light,spx.py:sphinx,md"},`.
- README includes a short section on how to extend `light` and `percent` formats to more languages (#61).
- Jupytext's contents manager accepts the `auto` extension in `default_jupytext_formats` (#93).
- All Jupyter magics are escaped in `light` scripts and R markdown documents. Escape magics in other formats with a `comment_magics` metadata (true or false), or with the contents manager `comment_magics` global flag (#94).

Fixed

- Trusting notebooks made functional again.
- Command line `jupytext` returns a meaningful error when no argument is given.
- Fixed global pairing configuration (#95).

6.14.69 0.7.2 (2018-10-01)

Added

- `light` and `percent` formats made available for scheme and cpp notebooks. Adding more formats is straightforward - just add a new entry to _SCRIPT_EXTENSIONS in languages.py, a sample notebook and a mirror test (#61)
- Format name is automatically appended to extension in `jupytext_formats` when notebook is loaded/saved.

Fixed

- Notebooks extensions can only be prefixed with `.nb` (#87)

6.14.70 0.7.1 (2018-09-24)

Fixed

- Markdown cells header in sphinx gallery format may have a space between first # and following.

6.14.71 0.7.0 (2018-09-23)

Added

- Header for cells in `percent` format is more robust: use `[markdown]` and `[raw]` to identify cell types. Cell type comes after the cell title. (#59)
- Jupytext can read and write notebooks as Hydrogen/VScode/Spyder/PyCharm compatible scripts (cells starting with `# %%`) (#59)
- Jupytext can read and write notebooks as Sphinx-gallery compatible scripts (#80)
- Metadata are supported for all cell types in light python and percent formats (#66). Due to this, light python format version is now 1.3. Light python notebooks in versions 1.1 and 1.2 are still readable.
- Command line `jupytext` has a `from` argument, and now accepts notebook from the standard input.

Fixed

- Fix merging of input and output notebooks (#83)
- Removed extra new line on stdout in command line `jupytext` (#84)

6.14.72 0.6.5 (2018-09-13)

Added

- Code lines that start with a quotation mark in Jupyter are commented in the corresponding Python and Julia scripts (#73)
- Update pypy, add flake8 tests on Travis CI (#74)

Fixed

- Import notebook.transutils before notebook.services.contents.filemanager (#75)

6.14.73 0.6.4 (2018-09-12)

Added

- Jupytext will not load paired notebook when text representation is out of date (#63)
- Package tested against Python 3.7 (#68)

Fixed

- Allow unicode characters in notebook path (#70)
- Read README.md as unicode in `setup.py` (#71)

6.14.74 0.6.3 (2018-09-07)

Added

- Lighter cell markers for Python and Julia scripts (#57). Corresponding file format version at 1.2. Scripts in previous version 1.1 can still be opened.
- New screenshots for the README.

Fixed

- Command line conversion tool `jupytext` fixed on Python 2.7 (#46)

6.14.75 0.6.2 (2018-09-05)

Added

- Initial support for Jupyter notebooks as Julia scripts (#56)
- Command line conversion tool `jupytext` has explicit `to` and `output` options (#46)
- Round trip test with `jupytext --test` improved (#54)
- Improved README (#51)

Fixed

- testfixtures now in requirements (#55)
- Empty code cells are now preserved (#53)

6.14.76 0.6.1 (2018-08-31)

Added

- Package and conversion script renamed from `nbrmd` to `jupytext`.

6.14.77 0.6.0 (2018-08-31)

Added

- Cell parsing and exporting done in two specialized classes. This is way easier to read. Pylint score at 9.9 !
- Python file format updated to 1.1: default end of cell for python scripts is one blank space. Two blank spaces are allowed as well. Now you can reformat code in Python IDE without breaking notebook cells (#38).
- Added support for plain markdown files (#40, #44).
- Demonstration notebooks more user friendly (#45).
- Command line tool simpler to use (#46).
- Start code patterns present in Jupyter cells are escaped.
- Default `nbrmd_format` is empty (mwouts/nbsrc/#5): no Jupyter notebook is created on disk when the user opens a Python or R file and saves it from Jupyter, unless the users asks for it by setting `nbrmd_format`.

Fixed

- Fixed message in the `nbsrc` script (#43)
- Technical metadata don't appear any more in scripts unless required (#42)
- Code cells that are fully commented remain code cells after round trip (#41)

6.14.78 0.5.4 (2018-08-24)

Added

- R to Rmd conversion compares well to knitr::spin (#26)
- Increased coverage to 98%

6.14.79 0.5.3 (2018-08-22)

Fixed

- Read and write version to the same metadata (#36)

6.14.80 0.5.2 (2018-08-22)

Added

- Classical jupyter extensions (autoreload, rmagics) are also escaped (#35)
- Explicit file format version, set at 1.0, to avoid overriding ipynb files by accident (#36)

6.14.81 0.5.1 (2018-08-21)

Fixed

- Source only notebooks can be trusted.

6.14.82 0.5.0 (2018-08-21)

Added

- Jupyter magic commands escaped when exported (#29)
- 'endofcell' option for explicit (optional) end-of-cell marker (#31)
- 'active' cell option now supported for .py and .R export (#30)
- Raw cells now preserved when exported to .py or .R (#32)
- Extensions can be prefixed, like `.nb.py`, (mwouts/nbsrc#5)
- When a file with an extension not associated to 'ipynb' is opened and saved, no 'ipynb' file is created (mwouts/nbsrc#5)
- Extensions can now be a sequence of groups. For instance, `nbrmd_formats="ipynb,nb.py;script.ipynb,py"` will create an `ipynb` file when a `nb.py` is opened (and conversely), and a `script.ipynb` file when a `py` file is opened (mwouts/nbsrc#5)
- `nbsrc` script was moved to the `nbrmd` package. The `nbsrc` package now only contains the documentation (mwouts/nbsrc#3)

6.14.83 0.4.6 (2018-07-26)

- Ping pypi - previous version still not available

6.14.84 0.4.5 (2018-07-26)

Fixed

- Removed dependency of `setup.py` on `yaml`

6.14.85 0.4.4 (2018-07-26)

Fixed

- Package republished with `python setup.py sdist bdist_wheel` to fix missing dependencies

6.14.86 0.4.3 (2018-07-26)

Added

- Multiline comments now supported #25
- Readme refactored, notebook demos available on binder #23

Fixed

- ContentsManager can be imported even if `notebook.transutils` is not available, for compatibility with older python distributions.
- Fixed missing cell metadata #27
- Documentation tells how to avoid creating `.ipynb` files #16

6.14.87 0.4.2 (2018-07-23)

Added

- Added test for R notebooks
- Added pylint badge, imports now in correct order
- New `active` cell metadata that allows cell activation only for desired extensions (currently available for Rmd and ipynb extensions only)

6.14.88 0.4.1 (2018-07-20)

Fixed

- Indented python code will not start a new cell #20
- Fixed parsing of Rmd cell metadata #21

6.14.89 0.4.0 (2018-07-18)

Added

- `.py` format for notebooks is lighter and pep8 compliant

Fixed

- Default nbrmd config not added to notebooks (#17)
- `nbrmd_formats` becomes a configurable traits (#16)
- Removed `nbrmd_sourceonly_format` metadata. Source notebook is current notebook when not `.ipynb`, otherwise the first notebook format in `nbrmd_formats` (not `.ipynb`) that is found on disk

6.14.90 0.3.0 (2018-07-17)

Added

- Introducing support for notebooks as python `.py` or R scripts `.R`

6.14.91 0.2.6 (2018-07-13)

Added

- Introduced `nbrmd_sourceonly_format` metadata
- Inputs are loaded from `.Rmd` file when a matching `.ipynb` file is opened.

Fixed

- Trusted notebooks remain trusted (#12)

6.14.92 0.2.5 (2018-07-11)

Added

- Outputs of existing `.ipynb` versions are combined with matching inputs of R markdown version, as suggested by @grst (#12)

Fixed

- Support for unicode text in python 2.7 (#11)

6.14.93 0.2.4 (2018-07-05)

Added

- nbrmd will always open notebooks, even if header of code cells are not terminated. Merge conflicts can thus be solved in Jupyter directly.
- New metadata 'main language' that preserves the notebook language.

Fixed

- dependencies included in `setup.py`
- pre_save_hook work with non-empty `notebook_dir` (#9)

6.14.94 0.2.3 (2018-06-28)

Added

- Screenshots in README

Fixed

- RMarkdown exporter for nbconvert fixed on non-recent python
- Tests compatible with other revisions of nbformat >= 4.0
- Tests compatible with older pytest versions

6.14.95 0.2.2 (2018-06-28)

Added

- RMarkdown exporter for nbconvert
- Parsing of R options robust to parenthesis
- Jupyter cell tags are preserved

Fixed

- requirements.txt now included in pypi packages

6.14.96 0.2.1 (2018-06-24)

Added

- Support for editing markdown files in Jupyter
- New pre-save hook `update_selected_formats` that saves to formats in metadata 'nbrmd_formats'
- Rmd cell options directly mapped to cell metadata

Fixed

- ContentManager compatible with Python 2.7

6.14.97 0.2.0 (2018-06-21)

Added

- The package provides a `RmdFileContentsManager` for direct edit of R markdown files in Jupyter
- Notebook metadata and cell options are preserved

6.14.98 0.1.1 (2018-06-19)

Added

- `nbrmd` prints the result of conversion to stdout, unless flag `-i` is provided
- Notebooks with R code chunks are supported

6.14.99 0.1 (2018-06-18)

- Initial version with the `nbrmd` converter and Jupyter `pre_save_hook`