Jupyter Notebook Documentation

Release 6.5.0.dev0

https://jupyter.org

Feb 28, 2022

USER DOCUMENTATION

1 The Jupyter Notebook 3

2 User interface components 11

3 Notebook Examples 15

4 What to do when things go wrong 57

5 Changelog 63

6 Comms 103

7 Configuration Overview 105

8 Config file and command line options 109

9 Running a notebook server 129

10 Security in the Jupyter notebook server 137

11 Security in notebook documents 139

12 Distributing Jupyter Extensions as Python Packages 143

13 Extending the Notebook 151

14 Contributing to the Jupyter Notebook 169

15 Developer FAQ 173

16 My Notebook 175

17 Other notebook 177

- Installation
- Starting the Notebook

CHAPTER

ONE

THE JUPYTER NOTEBOOK

1.1 Introduction

The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results. The Jupyter notebook combines two components:

A web application: a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output.

Notebook documents: a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects.

See also:

See the installation guide on how to install the notebook and its dependencies.

1.1.1 Main features of the web application

- In-browser editing for code, with automatic syntax highlighting, indentation, and tab completion/introspection.
- The ability to execute code from the browser, with the results of computations attached to the code which generated them.
- Displaying the result of computation using rich media representations, such as HTML, LaTeX, PNG, SVG, etc. For example, publication-quality figures rendered by the matplotlib library, can be included inline.
- In-browser editing for rich text using the Markdown markup language, which can provide commentary for the code, is not limited to plain text.
- The ability to easily include mathematical notation within markdown cells using LaTeX, and rendered natively by MathJax.

1.1.2 Notebook documents

Notebook documents contains the inputs and outputs of a interactive session as well as additional text that accompanies the code but is not meant for execution. In this way, notebook files can serve as a complete computational record of a session, interleaving executable code with explanatory text, mathematics, and rich representations of resulting objects. These documents are internally JSON files and are saved with the `.ipynb` extension. Since JSON is a plain text format, they can be version-controlled and shared with colleagues.

Notebooks may be exported to a range of static formats, including HTML (for example, for blog posts), reStructuredText, LaTeX, PDF, and slide shows, via the nbconvert command.

Furthermore, any `.ipynb` notebook document available from a public URL can be shared via the Jupyter Notebook Viewer <nbviewer>. This service loads the notebook document from the URL and renders it as a static web page. The results may thus be shared with a colleague, or as a public blog post, without other users needing to install the Jupyter notebook themselves. In effect, nbviewer is simply nbconvert as a web service, so you can do your own static conversions with nbconvert, without relying on nbviewer.

See also:

Details on the notebook JSON file format

1.1.3 Notebooks and privacy

Because you use Jupyter in a web browser, some people are understandably concerned about using it with sensitive data. However, if you followed the standard install instructions, Jupyter is actually running on your own computer. If the URL in the address bar starts with `http://localhost:` or `http://127.0.0.1:`, it's your computer acting as the server. Jupyter doesn't send your data anywhere else—and as it's open source, other people can check that we're being honest about this.

You can also use Jupyter remotely: your company or university might run the server for you, for instance. If you want to work with sensitive data in those cases, talk to your IT or data protection staff about it.

We aim to ensure that other pages in your browser or other users on the same computer can't access your notebook server. See *Security in the Jupyter notebook server* for more about this.

1.2 Starting the notebook server

You can start running a notebook server from the command line using the following command:

```
jupyter notebook
```

This will print some information about the notebook server in your console, and open a web browser to the URL of the web application (by default, `http://127.0.0.1:8888`).

The landing page of the Jupyter notebook web application, the **dashboard**, shows the notebooks currently available in the notebook directory (by default, the directory from which the notebook server was started).

You can create new notebooks from the dashboard with the `New Notebook` button, or open existing ones by clicking on their name. You can also drag and drop `.ipynb` notebooks and standard `.py` Python source code files into the notebook list area.

When starting a notebook server from the command line, you can also open a particular notebook directly, bypassing the dashboard, with `jupyter notebook my_notebook.ipynb`. The `.ipynb` extension is assumed if no extension is given.

When you are inside an open notebook, the *File* | *Open...* menu option will open the dashboard in a new browser tab, to allow you to open another notebook from the notebook directory or to create a new notebook.

Note: You can start more than one notebook server at the same time, if you want to work on notebooks in different directories. By default the first notebook server starts on port 8888, and later notebook servers search for ports near that one. You can also manually specify the port with the `--port` option.

1.2.1 Creating a new notebook document

A new notebook may be created at any time, either from the dashboard, or using the *File* → *New* menu option from within an active notebook. The new notebook is created within the same directory and will open in a new browser tab. It will also be reflected as a new entry in the notebook list on the dashboard.

1.2.2 Opening notebooks

An open notebook has **exactly one** interactive session connected to a kernel, which will execute code sent by the user and communicate back results. This kernel remains active if the web browser window is closed, and reopening the same notebook from the dashboard will reconnect the web application to the same kernel. In the dashboard, notebooks with an active kernel have a `Shutdown` button next to them, whereas notebooks without an active kernel have a `Delete` button in its place.

Other clients may connect to the same kernel. When each kernel is started, the notebook server prints to the terminal a message like this:

```
[NotebookApp] Kernel started: 87f7d2c0-13e3-43df-8bb8-1bd37aaf3373
```

This long string is the kernel's ID which is sufficient for getting the information necessary to connect to the kernel. If the notebook uses the IPython kernel, you can also see this connection data by running the `%connect_info` magic, which will print the same ID information along with other details.

You can then, for example, manually start a Qt console connected to the *same* kernel from the command line, by passing a portion of the ID:

```
$ jupyter qtconsole --existing 87f7d2c0
```

Without an ID, `--existing` will connect to the most recently started kernel.

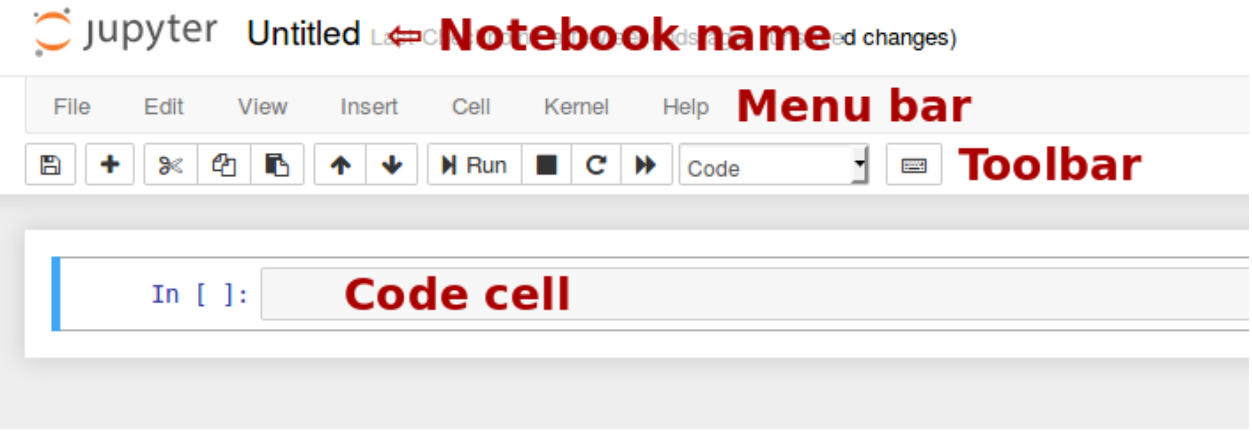
With the IPython kernel, you can also run the `%qtconsole` magic in the notebook to open a Qt console connected to the same kernel.

See also:

Decoupled two-process model

1.3 Notebook user interface

When you create a new notebook document, you will be presented with the **notebook name**, a **menu bar**, a **toolbar** and an empty **code cell**.



Notebook name: The name displayed at the top of the page, next to the Jupyter logo, reflects the name of the `.ipynb` file. Clicking on the notebook name brings up a dialog which allows you to rename it. Thus, renaming a notebook from "Untitled0" to "My first notebook" in the browser, renames the `Untitled0.ipynb` file to `My first notebook.ipynb`.

Menu bar: The menu bar presents different options that may be used to manipulate the way the notebook functions.

Toolbar: The tool bar gives a quick way of performing the most-used operations within the notebook, by clicking on an icon.

Code cell: the default type of cell; read on for an explanation of cells.

1.4 Structure of a notebook document

The notebook consists of a sequence of cells. A cell is a multiline text input field, and its contents can be executed by using `Shift-Enter`, or by clicking either the "Play" button the toolbar, or *Cell*, *Run* in the menu bar. The execution behavior of a cell is determined by the cell's type. There are three types of cells: **code cells**, **markdown cells**, and **raw cells**. Every cell starts off being a **code cell**, but its type can be changed by using a drop-down on the toolbar (which will be "Code", initially), or via *keyboard shortcuts*.

For more information on the different things you can do in a notebook, see the collection of examples.

1.4.1 Code cells

A *code cell* allows you to edit and write new code, with full syntax highlighting and tab completion. The programming language you use depends on the *kernel*, and the default kernel (IPython) runs Python code.

When a code cell is executed, code that it contains is sent to the kernel associated with the notebook. The results that are returned from this computation are then displayed in the notebook as the cell's *output*. The output is not limited to text, with many other possible forms of output are also possible, including `matplotlib` figures and HTML tables (as used, for example, in the `pandas` data analysis package). This is known as IPython's *rich display* capability.

See also:

Rich Output example notebook

1.4.2 Markdown cells

You can document the computational process in a literate way, alternating descriptive text with code, using *rich text*. In IPython this is accomplished by marking up text with the Markdown language. The corresponding cells are called *Markdown cells*. The Markdown language provides a simple way to perform this text markup, that is, to specify which parts of the text should be emphasized (italics), bold, form lists, etc.

If you want to provide structure for your document, you can use markdown headings. Markdown headings consist of 1 to 6 hash # signs # followed by a space and the title of your section. The markdown heading will be converted to a clickable link for a section of the notebook. It is also used as a hint when exporting to other document formats, like PDF.

When a Markdown cell is executed, the Markdown code is converted into the corresponding formatted rich text. Markdown allows arbitrary HTML code for formatting.

Within Markdown cells, you can also include *mathematics* in a straightforward way, using standard LaTeX notation: `$...$` for inline mathematics and `$$...$$` for displayed mathematics. When the Markdown cell is executed, the LaTeX portions are automatically rendered in the HTML output as equations with high quality typography. This is made possible by MathJax, which supports a large subset of LaTeX functionality

Standard mathematics environments defined by LaTeX and AMS-LaTeX (the `amsmath` package) also work, such as `\begin{equation}...\end{equation}`, and `\begin{align}...\end{align}`. New LaTeX macros may be defined using standard methods, such as `\newcommand`, by placing them anywhere *between math delimiters* in a Markdown cell. These definitions are then available throughout the rest of the IPython session.

See also:

Working with Markdown Cells example notebook

1.4.3 Raw cells

Raw cells provide a place in which you can write *output* directly. Raw cells are not evaluated by the notebook. When passed through nbconvert, raw cells arrive in the destination format unmodified. For example, you can type full LaTeX into a raw cell, which will only be rendered by LaTeX after conversion by nbconvert.

1.5 Basic workflow

The normal workflow in a notebook is, then, quite similar to a standard IPython session, with the difference that you can edit cells in-place multiple times until you obtain the desired results, rather than having to rerun separate scripts with the `%run` magic command.

Typically, you will work on a computational problem in pieces, organizing related ideas into cells and moving forward once previous parts work correctly. This is much more convenient for interactive exploration than breaking up a computation into scripts that must be executed together, as was previously necessary, especially if parts of them take a long time to run.

To interrupt a calculation which is taking too long, use the *Kernel*, *Interrupt* menu option, or the `i,i` keyboard shortcut. Similarly, to restart the whole computational process, use the *Kernel*, *Restart* menu option or `0,0` shortcut.

A notebook may be downloaded as a `.ipynb` file or converted to a number of other formats using the menu option *File*, *Download as*.

See also:

Running Code in the Jupyter Notebook example notebook

Notebook Basics example notebook

1.5.1 Keyboard shortcuts

All actions in the notebook can be performed with the mouse, but keyboard shortcuts are also available for the most common ones. The essential shortcuts to remember are the following:

- **`Shift-Enter`: run cell** Execute the current cell, show any output, and jump to the next cell below. If `Shift-Enter` is invoked on the last cell, it makes a new cell below. This is equivalent to clicking the *Cell*, *Run* menu item, or the Play button in the toolbar.
- **`Esc`: Command mode** In command mode, you can navigate around the notebook using keyboard shortcuts.
- **`Enter`: Edit mode** In edit mode, you can edit text in cells.

For the full list of available shortcuts, click *Help*, *Keyboard Shortcuts* in the notebook menus.

1.6 Plotting

One major feature of the Jupyter notebook is the ability to display plots that are the output of running code cells. The IPython kernel is designed to work seamlessly with the matplotlib plotting library to provide this functionality. Specific plotting library integration is a feature of the kernel.

1.7 Installing kernels

For information on how to install a Python kernel, refer to the IPython install page.

The Jupyter wiki has a long list of Kernels for other languages. They usually come with instructions on how to make the kernel available in the notebook.

1.8 Trusting Notebooks

To prevent untrusted code from executing on users' behalf when notebooks open, we store a signature of each trusted notebook. The notebook server verifies this signature when a notebook is opened. If no matching signature is found, Javascript and HTML output will not be displayed until they are regenerated by re-executing the cells.

Any notebook that you have fully executed yourself will be considered trusted, and its HTML and Javascript output will be displayed on load.

If you need to see HTML or Javascript output without re-executing, and you are sure the notebook is not malicious, you can tell Jupyter to trust it at the command-line with:

```
$ jupyter trust mynotebook.ipynb
```

See *Security in notebook documents* for more details about the trust mechanism.

1.9 Browser Compatibility

The Jupyter Notebook aims to support the latest versions of these browsers:

- Chrome
- Safari
- Firefox

Up to date versions of Opera and Edge may also work, but if they don't, please use one of the supported browsers.

Using Safari with HTTPS and an untrusted certificate is known to not work (websockets will fail).

CHAPTER

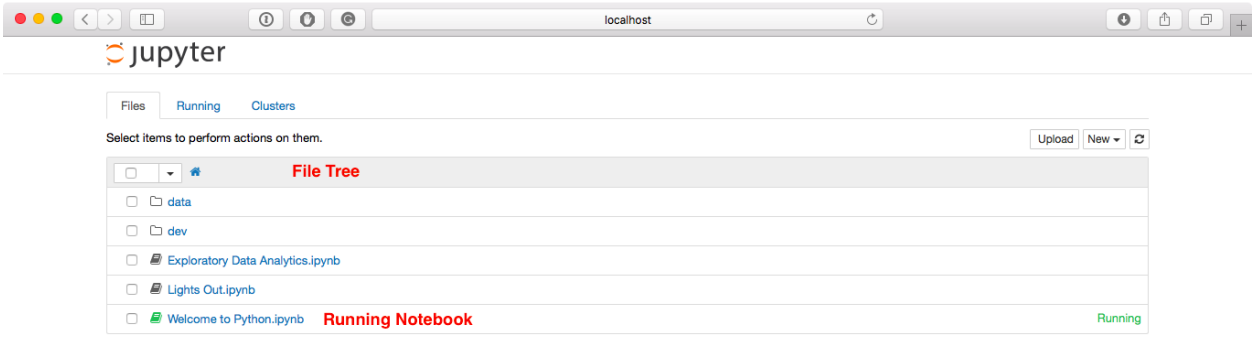
TWO

USER INTERFACE COMPONENTS

When opening bug reports or sending emails to the Jupyter mailing list, it is useful to know the names of different UI components so that other developers and users have an easier time helping you diagnose your problems. This section will familiarize you with the names of UI elements within the Notebook and the different Notebook modes.

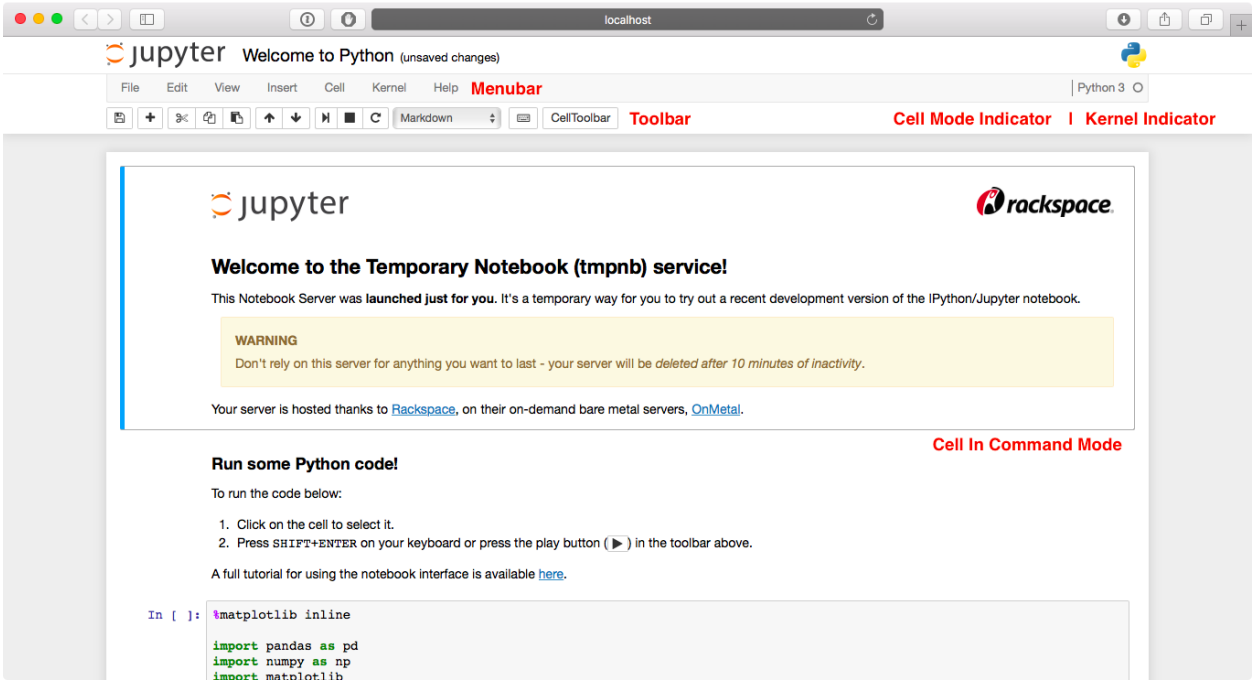
2.1 Notebook Dashboard

When you launch `jupyter notebook` the first page that you encounter is the Notebook Dashboard.



2.2 Notebook Editor

Once you've selected a Notebook to edit, the Notebook will open in the Notebook Editor.

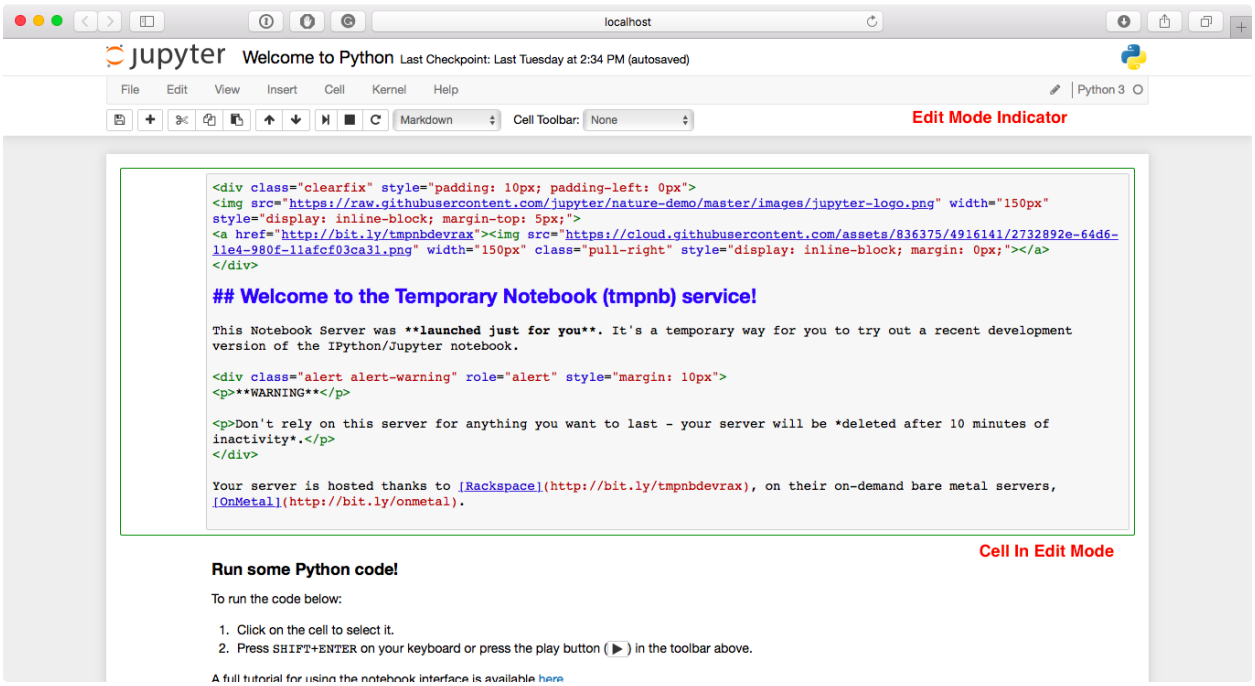


2.3 Interactive User Interface Tour of the Notebook

If you would like to learn more about the specific elements within the Notebook Editor, you can go through the user interface tour by selecting *Help* in the menubar then selecting *User Interface Tour*.

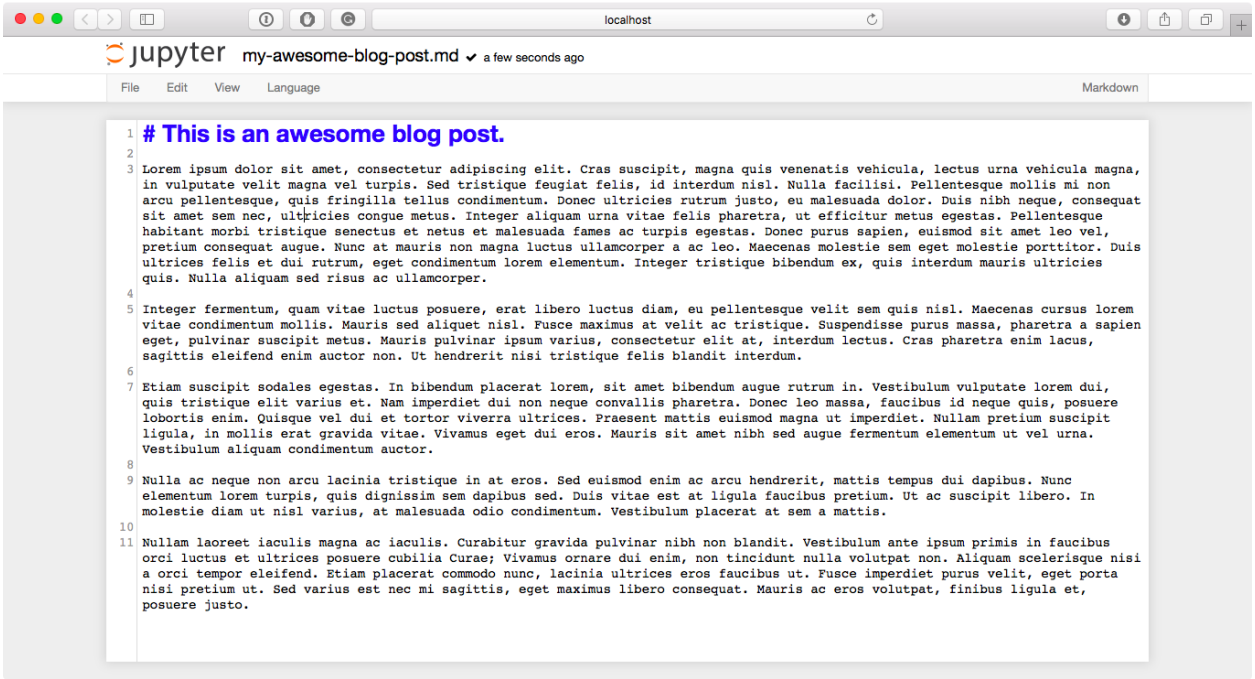
2.3.1 Edit Mode and Notebook Editor

When a cell is in edit mode, the Cell Mode Indicator will change to reflect the cell's state. This state is indicated by a small pencil icon on the top right of the interface. When the cell is in command mode, there is no icon in that location.



2.4 File Editor

Now let's say that you've chosen to open a Markdown file instead of a Notebook file whilst in the Notebook Dashboard. If so, the file will be opened in the File Editor.



CHAPTER

THREE

NOTEBOOK EXAMPLES

The pages in this section are all converted notebook files. You can also view these notebooks on nbviewer.

3.1 What is the Jupyter Notebook?

3.1.1 Introduction

The Jupyter Notebook is an **interactive computing environment** that enables users to author notebook documents that include: - Live code - Interactive widgets - Plots - Narrative text - Equations - Images - Video

These documents provide a **complete and self-contained record of a computation** that can be converted to various formats and shared with others using email, Dropbox, version control systems (like git/GitHub) or nbviewer.jupyter.org.

Components

The Jupyter Notebook combines three components:

- **The notebook web application**: An interactive web application for writing and running code interactively and authoring notebook documents.
- **Kernels**: Separate processes started by the notebook web application that runs users' code in a given language and returns output back to the notebook web application. The kernel also handles things like computations for interactive widgets, tab completion and introspection.
- **Notebook documents**: Self-contained documents that contain a representation of all content visible in the notebook web application, including inputs and outputs of the computations, narrative text, equations, images, and rich media representations of objects. Each notebook document has its own kernel.

3.1.2 Notebook web application

The notebook web application enables users to:

- **Edit code in the browser**, with automatic syntax highlighting, indentation, and tab completion/introspection.
- **Run code from the browser**, with the results of computations attached to the code which generated them.
- See the results of computations with **rich media representations**, such as HTML, LaTeX, PNG, SVG, PDF, etc.
- Create and use **interactive JavaScript widgets**, which bind interactive user interface controls and visualizations to reactive kernel side computations.
- Author **narrative text** using the Markdown markup language.

- Include mathematical equations using **LaTeX syntax in Markdown**, which are rendered in-browser by MathJax.

3.1.3 Kernels

Through Jupyter's kernel and messaging architecture, the Notebook allows code to be run in a range of different programming languages. For each notebook document that a user opens, the web application starts a kernel that runs the code for that notebook. Each kernel is capable of running code in a single programming language and there are kernels available in the following languages:

- Python(https://github.com/ipython/ipython)
- Julia (https://github.com/JuliaLang/IJulia.jl)
- R (https://github.com/IRkernel/IRkernel)
- Ruby (https://github.com/minrk/iruby)
- Haskell (https://github.com/gibiansky/IHaskell)
- Scala (https://github.com/Bridgewater/scala-notebook)
- node.js (https://gist.github.com/Carreau/4279371)
- Go (https://github.com/takluyver/igo)

The default kernel runs Python code. The notebook provides a simple way for users to pick which of these kernels is used for a given notebook.

Each of these kernels communicate with the notebook web application and web browser using a JSON over ZeroMQ/WebSockets message protocol that is described here. Most users don't need to know about these details, but it helps to understand that "kernels run code."

3.1.4 Notebook documents

Notebook documents contain the **inputs and outputs** of an interactive session as well as **narrative text** that accompanies the code but is not meant for execution. **Rich output** generated by running code, including HTML, images, video, and plots, is embeddeed in the notebook, which makes it a complete and self-contained record of a computation.

When you run the notebook web application on your computer, notebook documents are just **files on your local filesystem with a `.ipynb` extension**. This allows you to use familiar workflows for organizing your notebooks into folders and sharing them with others.

Notebooks consist of a **linear sequence of cells**. There are three basic cell types:

- **Code cells:** Input and output of live code that is run in the kernel
- **Markdown cells:** Narrative text with embedded LaTeX equations
- **Raw cells:** Unformatted text that is included, without modification, when notebooks are converted to different formats using nbconvert

Internally, notebook documents are JSON **data** with **binary values** base64 encoded. This allows them to be **read and manipulated programmatically** by any programming language. Because JSON is a text format, notebook documents are version control friendly.

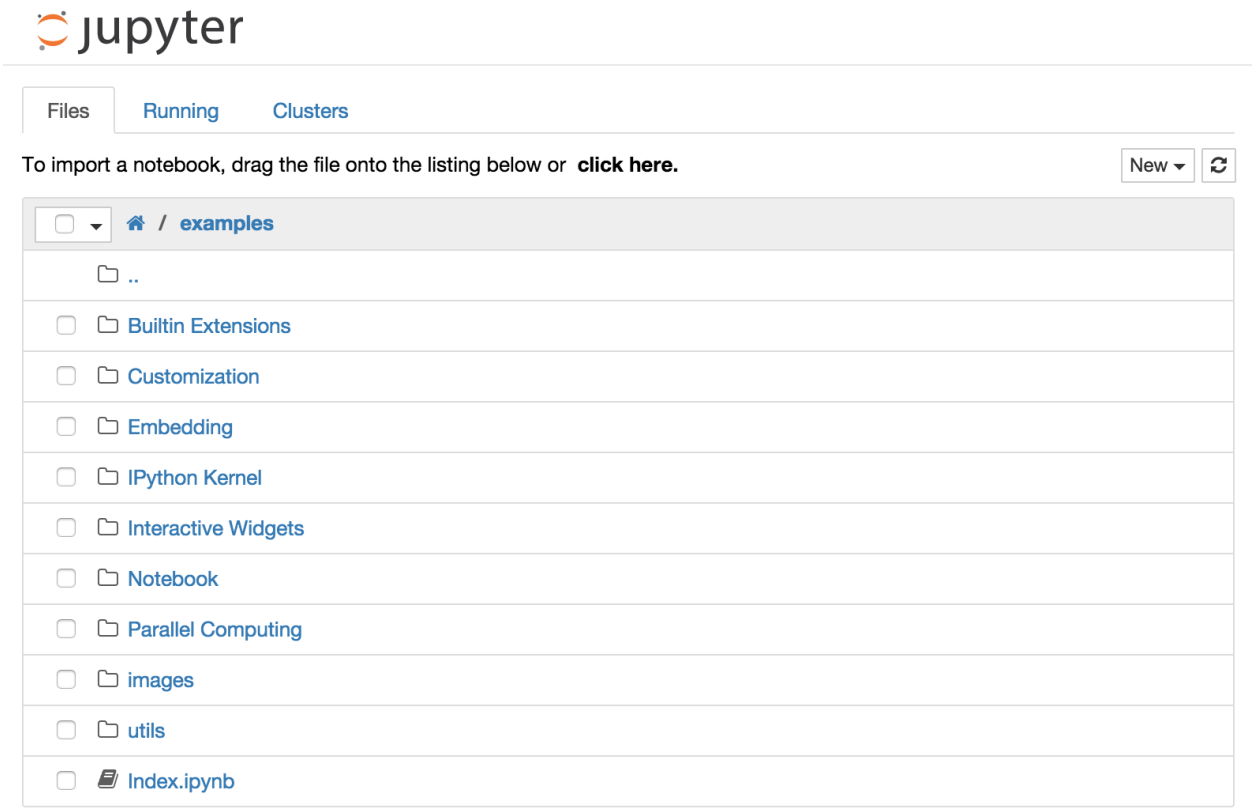
Notebooks can be exported to different static formats including HTML, reStructeredText, LaTeX, PDF, and slide shows (reveal.js) using Jupyter's `nbconvert` utility.

Furthermore, any notebook document available from a **public URL or on GitHub can be shared** via nbviewer. This service loads the notebook document from the URL and renders it as a static web page. The resulting web page may thus be shared with others **without their needing to install the Jupyter Notebook**.

3.2 Notebook Basics

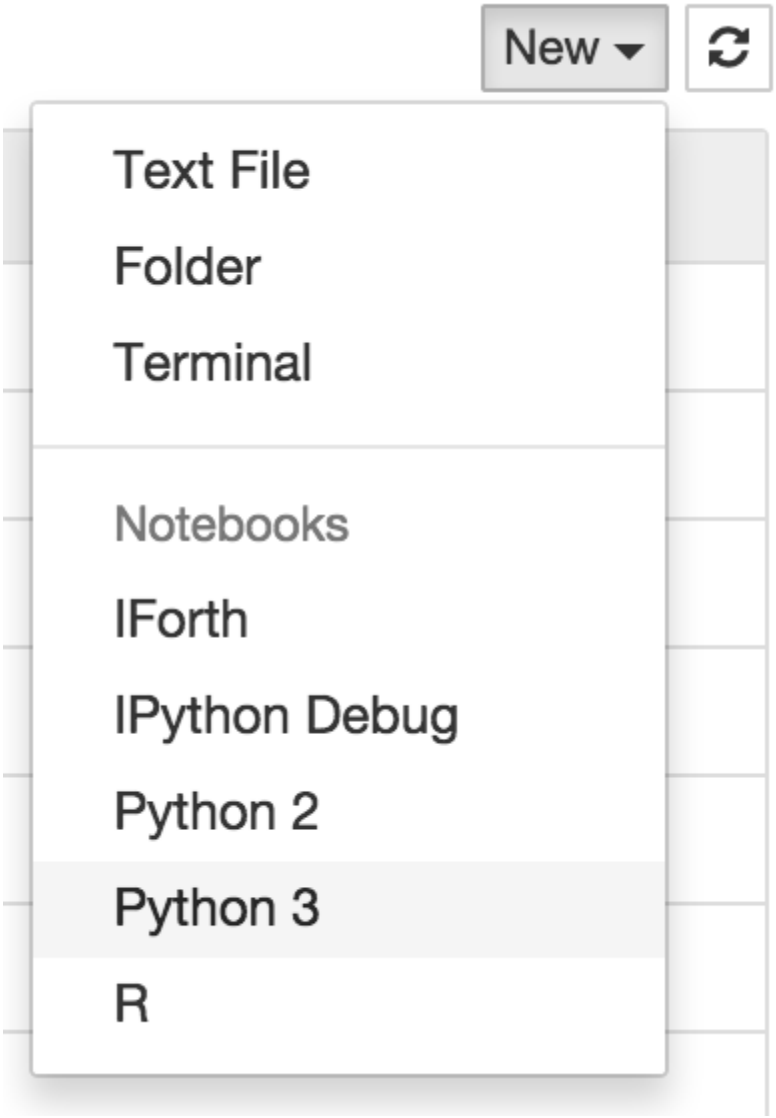
3.2.1 The Notebook dashboard

When you first start the notebook server, your browser will open to the notebook dashboard. The dashboard serves as a home page for the notebook. Its main purpose is to display the notebooks and files in the current directory. For example, here is a screenshot of the dashboard page for the `examples` directory in the Jupyter repository:



The top of the notebook list displays clickable breadcrumbs of the current directory. By clicking on these breadcrumbs or on sub-directories in the notebook list, you can navigate your file system.

To create a new notebook, click on the "New" button at the top of the list and select a kernel from the dropdown (as seen below). Which kernels are listed depend on what's installed on the server. Some of the kernels in the screenshot below may not exist as an option to you.



Notebooks and files can be uploaded to the current directory by dragging a notebook file onto the notebook list or by the "click here" text above the list.

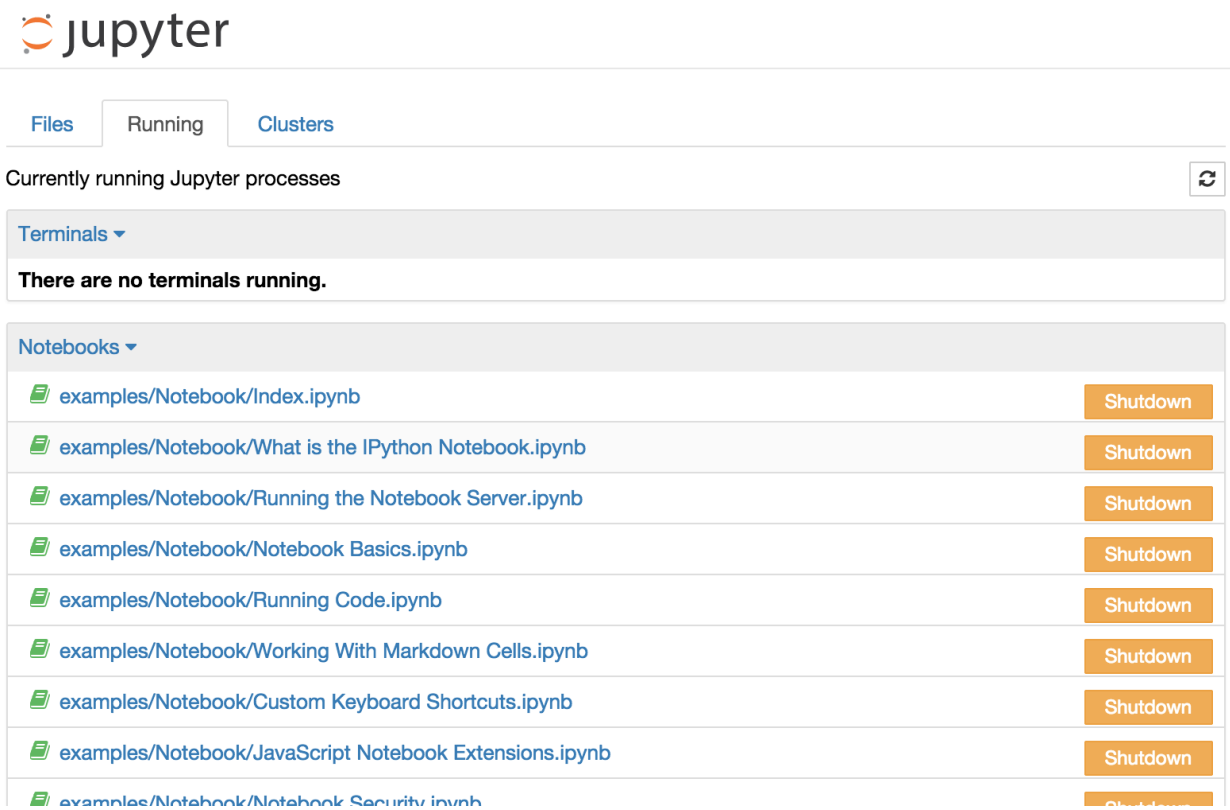
The notebook list shows green "Running" text and a green notebook icon next to running notebooks (as seen below). Notebooks remain running until you explicitly shut them down; closing the notebook's page is not sufficient.



To shutdown, delete, duplicate, or rename a notebook check the checkbox next to it and an array of controls will appear at the top of the notebook list (as seen below). You can also use the same operations on directories and files when applicable.



To see all of your running notebooks along with their directories, click on the "Running" tab:



This view provides a convenient way to track notebooks that you start as you navigate the file system in a long running notebook server.

3.2.2 Overview of the Notebook UI

If you create a new notebook or open an existing one, you will be taken to the notebook user interface (UI). This UI allows you to run code and author notebook documents interactively. The notebook UI has the following main areas:

- Menu
- Toolbar
- Notebook area and cells

The notebook has an interactive tour of these elements that can be started in the "Help:User Interface Tour" menu item.

3.2.3 Modal editor

Starting with IPython 2.0, the Jupyter Notebook has a modal user interface. This means that the keyboard does different things depending on which mode the Notebook is in. There are two modes: edit mode and command mode.

Edit mode

Edit mode is indicated by a green cell border and a prompt showing in the editor area:



When a cell is in edit mode, you can type into the cell, like a normal text editor.

Enter edit mode by pressing `Enter` or using the mouse to click on a cell's editor area.

Command mode

Command mode is indicated by a grey cell border with a blue left margin:



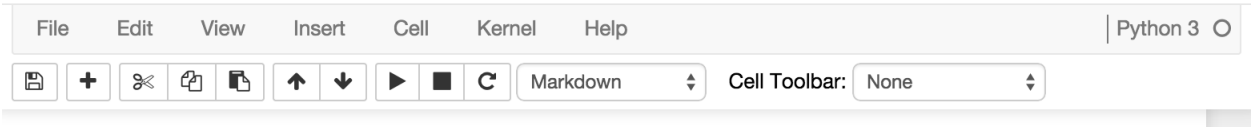
When you are in command mode, you are able to edit the notebook as a whole, but not type into individual cells. Most importantly, in command mode, the keyboard is mapped to a set of shortcuts that let you perform notebook and cell actions efficiently. For example, if you are in command mode and you press `c`, you will copy the current cell - no modifier is needed.

Don't try to type into a cell in command mode; unexpected things will happen!

Enter command mode by pressing `Esc` or using the mouse to click *outside* a cell's editor area.

3.2.4 Mouse navigation

All navigation and actions in the Notebook are available using the mouse through the menubar and toolbar, which are both above the main Notebook area:



The first idea of mouse based navigation is that **cells can be selected by clicking on them.** The currently selected cell gets a grey or green border depending on whether the notebook is in edit or command mode. If you click inside a cell's editor area, you will enter edit mode. If you click on the prompt or output area of a cell you will enter command mode.

If you are running this notebook in a live session (not on http://nbviewer.jupyter.org) try selecting different cells and going between edit and command mode. Try typing into a cell.

The second idea of mouse based navigation is that **cell actions usually apply to the currently selected cell**. Thus if you want to run the code in a cell, you would select it and click the

button in the toolbar or the "Cell:Run" menu item. Similarly, to copy a cell you would select it and click the

button in the toolbar or the "Edit:Copy" menu item. With this simple pattern, you should be able to do most everything you need with the mouse.

Markdown cells have one other state that can be modified with the mouse. These cells can either be rendered or unrendered. When they are rendered, you will see a nice formatted representation of the cell's contents. When they are unrendered, you will see the raw text source of the cell. To render the selected cell with the mouse, click the

button in the toolbar or the "Cell:Run" menu item. To unrender the selected cell, double click on the cell.

3.2.5 Keyboard Navigation

The modal user interface of the Jupyter Notebook has been optimized for efficient keyboard usage. This is made possible by having two different sets of keyboard shortcuts: one set that is active in edit mode and another in command mode.

The most important keyboard shortcuts are `Enter`, which enters edit mode, and `Esc`, which enters command mode.

In edit mode, most of the keyboard is dedicated to typing into the cell's editor. Thus, in edit mode there are relatively few shortcuts. In command mode, the entire keyboard is available for shortcuts, so there are many more. The `Help->``Keyboard Shortcuts`` dialog lists the available shortcuts.

We recommend learning the command mode shortcuts in the following rough order:

1. Basic navigation: `enter`, `shift-enter`, `up/k`, `down/j`
2. Saving the notebook: `s`
3. Change Cell types: `y`, `m`, `1-6`, `t`
4. Cell creation: `a`, `b`
5. Cell editing: `x`, `c`, `v`, `d`, `z`
6. Kernel operations: `i`, `0` (press twice)

3.3 Running Code

First and foremost, the Jupyter Notebook is an interactive environment for writing and running code. The notebook is capable of running code in a wide range of languages. However, each notebook is associated with a single kernel. This notebook is associated with the IPython kernel, therefore runs Python code.

3.3.1 Code cells allow you to enter and run code

Run a code cell using `Shift-Enter` or pressing the

button in the toolbar above:

[2]:
```
a = 10
```

[3]:
```
print(a)
```

```
10
```

There are two other keyboard shortcuts for running code:

- `Alt-Enter` runs the current cell and inserts a new one below.
- `Ctrl-Enter` run the current cell and enters command mode.

3.3.2 Managing the Kernel

Code is run in a separate process called the Kernel. The Kernel can be interrupted or restarted. Try running the following cell and then hit the

button in the toolbar above.

[4]:
```
import time
time.sleep(10)
```

If the Kernel dies you will be prompted to restart it. Here we call the low-level system libc.time routine with the wrong argument via ctypes to segfault the Python interpreter:

[5]:
```
import sys
from ctypes import CDLL
# This will crash a Linux or Mac system
# equivalent calls can be made on Windows

# Uncomment these lines if you would like to see the segfault

# dll = 'dylib' if sys.platform == 'darwin' else 'so.6'
# libc = CDLL("libc.%s" % dll)
# libc.time(-1)  # BOOM!!
```

3.3.3 Cell menu

The "Cell" menu has a number of menu items for running code in different ways. These includes:

- Run and Select Below
- Run and Insert Below
- Run All
- Run All Above
- Run All Below

3.3.4 Restarting the kernels

The kernel maintains the state of a notebook's computations. You can reset this state by restarting the kernel. This is done by clicking on the

in the toolbar above.

3.3.5 sys.stdout and sys.stderr

The stdout and stderr streams are displayed as text in the output area.

[6]:
```
print("hi, stdout")
```

```
hi, stdout
```

[7]:
```
from __future__ import print_function
print('hi, stderr', file=sys.stderr)
```

```
hi, stderr
```

3.3.6 Output is asynchronous

All output is displayed asynchronously as it is generated in the Kernel. If you execute the next cell, you will see the output one piece at a time, not all at the end.

[8]:
```
import time, sys
for i in range(8):
    print(i)
    time.sleep(0.5)
```

```
0
1
2
3
4
5
6
7
```

3.3.7 Large outputs

To better handle large outputs, the output area can be collapsed. Run the following cell and then single- or double-click on the active area to the left of the output:

[9]:
```
for i in range(50):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

(continues on next page)

(continued from previous page)

```
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
```

Beyond a certain point, output will scroll automatically:

[10]:
```
for i in range(500):
    print(2**i - 1)
```

```
0
1
3
7
15
31
63
127
255
511
1023
2047
4095
8191
16383
32767
65535
131071
262143
```

(continues on next page)

(continued from previous page)

```
524287
1048575
2097151
4194303
8388607
16777215
33554431
67108863
134217727
268435455
536870911
1073741823
2147483647
4294967295
8589934591
17179869183
34359738367
68719476735
137438953471
274877906943
549755813887
1099511627775
2199023255551
4398046511103
8796093022207
17592186044415
35184372088831
70368744177663
140737488355327
281474976710655
562949953421311
1125899906842623
2251799813685247
4503599627370495
9007199254740991
18014398509481983
36028797018963967
72057594037927935
144115188075855871
288230376151711743
576460752303423487
1152921504606846975
2305843009213693951
4611686018427387903
9223372036854775807
18446744073709551615
36893488147419103231
73786976294838206463
147573952589676412927
295147905179352825855
590295810358705651711
1180591620717411303423
```

(continues on next page)

(continued from previous page)

```
2361183241434822606847
472236648286964521369 5
9444732965739290427391
18889465931478580854783
37778931862957161709567
75557863725914323419135
15111572745182864683827 1
30223145490365729367654 3
60446290980731458735308 7
12089258196146291747061 75
24178516392292583494123 51
48357032784585166988247 03
96714065569170333976494 07
19342813113834066795298815
38685626227668133590597631
77371252455336267181195263
15474250491067253436239 0527
30948500982134506872478 1055
61897001964269013744956 2111
12379400392853802748991 24223
24758800785707605497982 48447
49517601571415210995964 96895
99035203142830421991929 93791
19807040628566084398385987583
39614081257132168796771975167
79228162514264337593543950335
15845632502852867518708 7900671
31691265005705735037417 5801343
63382530011411470074835 1602687
12676506002282294014967 03205375
25353012004564588029934 06410751
50706024009129176059868 12821503
10141204801825835211973625643007
20282409603651670423947251286015
40564819207303340847894502572031
81129638414606681695789005144063
16225927682921336339157 8010288127
32451855365842672678315 6020576255
64903710731685345356631 2041152511
12980742146337069071326 24082305023
25961484292674138142652 48164610047
51922968585348276285304 96329220095
10384593717069655257060 992658440191
20769187434139310514121 985316880383
41538374868278621028243 970633760767
83076749736557242056487 941267521535
16615349947311448411297 5882535043071
33230699894622896822595 1765070086143
66461399789245793645190 3530140172287
13292279957849158729038 07060280344575
26584559915698317458076 14120560689151
53169119831396634916152 28241121378303
```

(continues on next page)

(continued from previous page)

```
106338239662793269832304564822427566O7
212676479325586539664609129644855132l5
425352958651173079329218259289710264Il
850705917302346158658436518579420528b3
170141183460469231731687303715884105727
340282366920938463463374607431768211455
680564733841876926926749214863536422911
136112946768375385385349842972707284582З
272225893536750770770699685945414569164Z
544451787073501541541399371890829138329Ь
108890357414700308308279874378165827665Эl
217780714829400616616559748756331655331Ь3
435561429658801233233119497512663310663б7
871122859317602466466238995025326621327З5
174224571863520493293247799005065324265471
348449143727040986586495598010130648530943
696898287454081973172991196020261297061887
139379657490816394634598239204052259412377Б
278759314981632789269196478408104518824755l
557518629963265578538392956816209037649510З
111503725992653115707678591363241807529902О7
223007451985306231415357182726483615059804l5
446014903970612462830714365452967230119608З1
892029807941224925661428730905934460239216б3
178405961588244985132285746181186892047843327
356811923176489970264571492362373784095686655
713623846352979940529142984724747568191373311
142724769270595988105828596944949513638274662З
285449538541191976211657193889899027276549324Z
570899077082383952423314387779798054553098649Б
114179815416476790484662877555959610910619729Эl
228359630832953580969325755111919221821239459Эl
456719261665907161938651510223838443642478919б7
913438523331814323877303020447676887284957839З5
182687704666362864775460604089535377456991567871
365375409332725729550921208179070754913983135743
730750818665451459101842416358141509827966271487
146150163733090291820368483271628301965593254297Б
292300327466180583640736966543256603931186508595l
584600654932361167281473933086513207862373017190З
116920130986472233456294786617302641572474603438О7
233840261972944466912589573234605283144949206876l5
467680523945888933825179146469210566289898413752З1
935361047891777867650358292938421132579796827504б3
187072209578355573530071658587684226515959365500927
374144419156711147060143317175368453031918731001855
748288838313422294120286634350736906063837462003711
149657767662684458824057326870147381212767492400742З
299315535325368917648114653740294762425534984801485Т
598631070650737835296229307480589524851069969602969Б
119726214130147567059245861496117904970213993920593Эl
239452428260295134118491722992235809940427987841187Эl
```

(continues on next page)

(continued from previous page)

```
4789048565205902682369834459844716198808559756823756 7
9578097130411805364739668919689432397617119513647513 5
1915619426082361072947933783937886479523423902729502 71
3831238852164722145895867567875772959046847805459005 43
7662477704329444291791735135751545918093695610918010 87
1532495540865888858358347027150309183618739122183602 175
3064991081731777716716694054300618367237478244367204 351
6129982163463555433433388108601236734474956488734408703
1225996432692711086686677621720247346894991297746881740 7
2451992865385422173373355243440494693789982595493763481 5
4903985730770844346746710486880989387579965190987526963 1
9807971461541688693493420973761978775159930381975053926 3
1961594292308337738698684194752395755031986076395010785 27
3923188584616675477397368389504791510063972152790021570 55
7846377169233350954794736779009583020127944305580043141 11
1569275433846670190958947355801916604025588861116008628223
3138550867693340381917894711603833208051177722232017256447
6277101735386680763835789423207666416102355444464034512895
1255420347077336152767157884641533283220471088892806902 5791
2510840694154672305534315769283066566440942177785613805 1583
5021681388309344611068631538566133132881884355571227610 3167
1004336277661868922213726307713226626576376871114245522 06335
2008672555323737844427452615426453253152753742228491044 12671
4017345110647475688854905230852906506305507484456982088 25343
8034690221294951377709810461705813012611014968913964176 50687
1606938044258990275541962092341162602522202993782792835 301375
3213876088517980551083924184682325205044405987565585670 602751
6427752177035961102167848369364650410088811975131171341 205503
1285550435407192220433569673872930082017762395026234268 2411007
2571100870814384440867139347745860164035524790052468536 4822015
5142201741628768881734278695491720328071049580104937072 9644031
1028440348325753776346855739098344065614209916020987414 59288063
2056880696651507552693711478196688131228419832041974829 18576127
4113761393303015105387422956393376262456839664083949658 37152255
8227522786606030210774845912786752524913679328167899316 74304511
1645504557321206042154969182557350504982735865633579863 348609023
3291009114642412084309938365114701009965471731267159726 697218047
6582018229284824168619876730229402019930943462534319453 394436095
1316403645856964833723975346045880403986188692506863890 6788872191
2632807291713929667447950692091760807972377385013727781 3577744383
5265614583427859334895901384183521615944754770027455562 7155488767
1053122916685571866979180276836704323188950954005491112 54310977535
2106245833371143733958360553673408646377901908010982225 08621955071
4212491666742287467916721107346817292755803816021964450 17243910143
8424983333484574935833442214693634585511607632043928900 34487820287
1684996666696914987166688442938726917102321526408785780 068975640575
3369993333393829974333376885877453834204643052817571560 137951281151
6739986666787659948666753771754907668409286105635143120 275902562303
1347997333357531989733350754350981533681857221127028624 0551805124607
2695994666715063979466701508701963067363714442254057248 1103610249215
5391989333430127958933403017403926134727428884508114496 2207220498431
1078397866686025591786680603480785226945485776901622899 24414440996863
```

(continues on next page)

(continued from previous page)

```
215679573337205118357336120696157045389097155380324579848828881993727
431359146674410236714672241392314090778194310760649159697657763987455
862718293348820473429344482784628181556388621521298319395315527974911
172543658669764094685868896556925636311277724304259663879063105594982 3
345087317339528189371737793113851272622555448608519327758126211189964 7
690174634679056378743475586227702545245110897217038655516252422379929 5
138034926935811275748695117245540509049022179443407731103250484475985 91
276069853871622551497390234491081018098044358886815462206500968951971 83
552139707743245102994780468982162036196088717773630924413001937903943 67
110427941548649020598956093796432407239217743554726184882600387580788735
220855883097298041197912187592864814478435487109452369765200775161577471
441711766194596082395824375185729628956870974218904739530401550323154943
883423532389192164791648750371459257913741948437809479060803100646309887
176684706477838432958329750074291851582748389687561895812160620129261977 5
353369412955676865916659500148583703165496779375123791624321240258523955 1
706738825911353731833319000297167406330993558750247583248642480517047910 3
141347765182270746366663800059433481266198711750049516649728496103409582 07
282695530364541492733327600118866962532397423500099033299456992206819164 15
565391060729082985466655200237733925064794847000198066598913984413638328 31
113078212145816597093331040047546785012958969400039613319782796882727665663
226156424291633194186662080095093570025917938800079226639565593765455331327
452312848583266388373324160190187140051835877600158453279131187530910662655
904625697166532776746648320380374280103671755200316906558262375061821325311
180925139433306555349329664076074856020734351040063381311652475012364265062 3
361850278866613110698659328152149712041468702080126762623304950024728530124 7
723700557733226221397318656304299424082937404160253525246609900049457060249 5
144740111546645244279463731260859884816587480832050705049321980009891412049 91
289480223093290488558927462521719769633174961664101410098643960019782824099 83
578960446186580977117854925043439539266349923328202820197287920039565648199 67
115792089237316195423570985008687907853269984665640564039457584007913129639935
231584178474632390847141970017375815706539969331281128078915168015826259279871
463168356949264781694283940034751631413079938662562256157830336031652518559743
926336713898529563388567880069503262826159877325124512315660672063305037119487
185267342779705912677713576013900652565231975465024902463132134412661007423897 5
370534685559411825355427152027801305130463950930049804926264268825322014847795 1
741069371118823650710854304055602610260927901860099609852528537650644029695590 3
148213874223764730142170860811120522052185580372019921970505707530128805939118 07
296427748447529460284341721622241044104371160744039843941011415060257611878236 15
592855496895058920568683443244482088208742321488079687882022830120515223756472 31
118571099379011784113736688648896417641748464297615937576404566024103044751294463
237142198758023568227473377297792835283496285923187515280913204820608950258892 7
474284397516047136454946754595585670566993857190463750305618264096412179005177855
948568795032094272909893509191171341133987714380927500611236528192824358010355711
189713759006418854581978701838234268226797542876185500122247305638564871602071 1423
379427518012837709163957403676468536453595085752371000244494611277129743204142 2847
758855036025675418327914807352937072907190171504742000488989222554259486408284 5695
151771007205135083665582961470587414581438034300948400097797844510851897281656913 91
303542014410270167331165922941174829162876068601896800195595689021703794563313827 83
607084028820540334662331845882349658325752137203793600391191378043407589126627655 67
121416805764108066932466369176469931665150427440758720078238275608681517825325531135
242833611528216133864932738352939863330300854881517440156476551217363035650651062271
485667223056432267729865476705879726660601709763034880312953102434726071301302124543
```

(continues on next page)

(continued from previous page)

```
971334446112864535459730953411759453321203419526069760625906204869452142602604249087
194266889222572907091946190682351890664240683905213952125181240973890428520520849817 5
388533778445145814183892381364703781328481367810427904250362481947780857041041699635 1
777067556890291628367784762729407562656962735620855808500724963895561714082083399270 3
155413511378058325673556952545881512531392547124171161700144992779112342816416679854 07
310827022756116651347113905091763025062785094248342323400289985558224685632833359708 15
621654045512233302694227810183526050125570188496684646800579971116449371265666719416 31
124330809102446660538845562036705210025114037699336929360115994223289874253133343883263
248661618204893321077691124073410420050228075398673858720231988446579748506266687766527
497323236409786642155382248146820840100456150797347717440463976893159497012533375533055
994646472819573284310764496293641680200912301594695434880927953786318994025066751066111
198929294563914656862152899258728336040182460318939086976185590757263798805013350213222 3
397858589127829313724305798517456672080364920637878173952371181514527597610026700426444 7
795717178255658627448611597034913344160729841275756347904742363029055195220053400852889 5
159143435651131725489722319406982668832145968255151269580948472605811039044010680170577 91
318286871302263450979444638813965337664291936510302539161896945211622078088021360341155 83
636573742604526901958889277627930675328583873020605078323793890423244156176042720682311 67
127314748520905380391777855525586135065716774604121015664758778084648831235208544136462335
254629497041810760783555711051172270131433549208242031329517556169297662470417088272924671
509258994083621521567111422102344540262867098416484062659035112338595324940834176545849343
101851798816724304313422284420468908052573419683296812531807022467719064988166835309169868 7
203703597633448608626844568840937816105146839366593625063614044935438129976333670618339737 5
407407195266897217253689137681875632210293678733187250127228089870876259952667341236679475 1
814814390533794434507378275363751264420587357466374500254456179741752519905334682473358950 3
162962878106758886901475655072750252884117471493274900050891235948350503981066936494671790 07
325925756213517773802951310145500505768234942986549800101782471896701007962133872989343580 15
651851512427035547605902620291001011536469885973099600203564943793402015924267745978687160 31
130370302485407109521180524058200202307293977194619920040712988758680403184853549195737432063
260740604970814219042361048116400404614587954389239840081425977517360806369707098391474864127
521481209941628438084722096232800809229175908778479680162851955034721612739414196782949728255
104296241988325687616944419246560161845835181755695936032570391006944322547882839356589945651 1
208592483976651375233888838493120323691670363511391872065140782013888645095765678713179891302 3
417184967953302750467777676986240647383340727022783744130281564027777290191531357426359782604 7
834369935906605500935555353972481294766681454045567488260563128055554580383062714852719565209 5
166873987181321100187111070794496258953336290809113497652112625611110916076612542970543913041 91
333747974362642200374222141588992517906672581618226995304225251222221832153225085941087826083 83
667495948725284400748444283177985035813345163236453990608450502444443664306450171882175652167 67
133499189745056880149688856635597007162669032647290798121690100488888732861290034376435130433535
266998379490113760299377713271194014325338065294581596243380200977777465722580068752870260867071
533996758980227520598755426542388028650676130589163192486760401955554931445160137505740521734143
106799351796045504119751085308477605730135226117832638497352080391110986289032027501148104346828 7
213598703592091008239502170616955211460270452235665276994704160782221972578064055002296208693657 5
427197407184182016479004341233910422920540904471330553989408321564443945156128110004592417387315 1
854394814368364032958008682467820845841081808942661107978816643128887890312256220009184834774630 3
170878962873672806591601736493564169168216361788532221595763328625777578062451244001836966954926 07
341757925747345613183203472987128338336432723577064443191526657251555156124902488003673933909852 15
683515851494691226366406945974256676672865447154128886383053314503110312249804976007347867819704 31
136703170298938245273281389194851335334573089430825777276610662900622062449960995201469573563940863
273406340597876490546562778389702670669146178861651554553221325801244124899921990402939147127881727
546812681195752981093125556779405341338292357723303109106442651602488249799843980805878294255763455
109362536239150596218625111355881068267658471544660621821288530320497649959968796161175658851152691 1
218725072478301192437250222711762136535316943089321243642577060640995299919937592322351317702305382 3
```

(continues on next page)

(continued from previous page)

```
437450144956602384874500445423524273070633886178642487285154121281990599839875184644702635404610764
874900289913204769749000890847048546141267772357284974570308242563981199679750369289405270809221529
174980057982640953949800178169409709228253554471456994914061648512796239935950073857881054161844305
349960115965281907899600356338819418456507108942913989828123297025592479871900147715762108323688611
699920231930563815799200712677638836913014217885827979656246594051184959743800295431524216647377223
139984046386112763159840142535527767382602843577165595931249318810236991948760059086304843329475444
279968092772225526319680285071055534765205687154331191862498637620473983897520118172609686658950889
559936185544451052639360570142111069530411374308662383724997275240947967795040236345219373317901778
111987237108890210527872114028422213906082274861732476744999455048189593559000804726904387466358035
223974474217780421055744228056844427812164549723464953489998910096379187118016094538087749327160711
447948948435560842111488456113688855624329099446929906979997820192758374236032189076175498654321423
895897896871121684222976912227377711248658198893859813959995640385516748472064378152350997308642846
179179579374224336844595382445475542249731639778771962791999128077103349694412875630470199461728569
358359158748448673689190764890951084499463279557543925583998256154206699388825751260940398923457138
716718317496897347378381529781902168998926559115087851167996512308413398777651502521880797846914277
143343663499379469475676305956380433799785311823017570233599302461682679755530300504376159569382855
286687326998758938951352611912760867599570623646035140467198604923365359511060601008752319138765710
573374653997517877902705223825521735199141247292070280934397209846730719022121202017504638277531421
114674930799503575580541044765104347039828249458414056186879441969346143804424240403500927655506284
229349861599007151161082089530208694079656498916828112373758883938692287608848480807001855311012568
458699723198014302322164179060417388159312997833656224747517767877384575217696961614003710622025137
917399446396028604644328358120834776318625995667312449495035535754769150435393923228007421244050274
183479889279205720928865671624166955263725199133462489899007107150953830087078784645601484248810054
366959778558411441857731343248333910527450398266924979798014214301907660174157569291202968497620109
733919557116822883715462686496667821054900796533849959596028428603815320348315138582405936995240219
146783911423364576743092537299333564210980159306769991919205685720763064069663027716481187399048043
293567822846729153486185074598667128421960318613539983838411371441526128139326055432962374798096078
587135645693458306972370149197334256843920637227079967676822742883052256278652110865924749596192175
117427129138691661394474029839466851368784127445415993535364548576610451255730422173184949919238435
234854258277383322788948059678933702737568254890831987070729097153220902511460844346369899838476870
469708516554766645577896119357867405475136509781663974141458194306441805022921688692739799676953740
939417033109533291155792238715734810950273019563327948282916388612883610045843377385479599353907481
187883406621906658231158447743146962190054603912665589656583277722576722009168675477095919870781496
375766813243813316462316895486293924380109207825331179313166555445153444018337350954191839741562992
751533626487626632924633790972587848760218415650662358626333110890306888036674701908383679483125984
150306725297525326584926758194517569752043683130132471725266622178061377607334940381676735896625196
300613450595050653169853516389035139504087366260264943450533244356122755214669880763353471793250393
601226901190101306339707032778070279008174732520529886901066488712245510429339761526706943586500787
120245380238020261267941406555614055801634946504105977380213297742449102085867952305341388717300157
240490760476040522535882813111228111603269893008211954760426595484898204171735904610682777434600315
480981520952081045071765626222456223206539786016423909520853190969796408343471809221365554869200630
961963041904162090143531252444912446413079572032847819041706381939592816686943618442731109738401260
192392608380832418028706250488982489282615914406569563808341276387918563337388723688546221947680252
384785216761664836057412500977964978565231828813139127616682552775837126674777447377092443895360504
769570433523329672114825001955929957130463657626278255233365105551674253349554894754184887790721008
153914086704665934422965000391185991426092731525255651046673021110334850669910978950836977558144201
307828173409331868845930000782371982852185463050511302093346042220669701339821957901673955116288403
615656346818663737691860001564743965704370926101022604186692084441339402679643915803347910232576806
123131269363732747538372000312948793140874185220204520837338416888267880535928783160669582046515361
246262538727465495076744000625897586281748370440409041674676833776535761071857566321339164093030722
492525077454930990153488001251795172563496740880818083349353667553071522143715132642678328186061445
985050154909861980306976002503590345126993481761636166698707335106143044287430265285356656372122891
```

(continues on next page)

(continued from previous page)

```
19701003098197239606139520050071806902539869635232723333974146702122860885748605305707133127442457820401
39402006196394479212279040100143613805079739270465446667948293404245721771497210611414266254884915640800
78804012392788958424558080200287227610159478540930893335896586808491443542994421222828532509769831281611
15760802478557791684911616040057445522031895708186178667179317361698288708598884244565706501953966256322
31521604957115583369823232080114891044063791416372357334358634723396577417197768489131413003907932512645
63043209914231166739646464160229782088127582832744714668717269446793154834395536978262826007815865025290
12608641982846233347929292832045956417625516566548942933743453889358630966879107395652565201563173005058
25217283965692466695858585664091912835251033133097885867486907778717261933758214791305130403126346010110
50434567931384933391717171328183825670502066266195771734973815557434523867516429582610260806252692020232
10086913586276986678343434265636765134100413253239154346994763111486904773503285916522052161250538404040
20173827172553973356686868531273530268200826506478308693989526222973809547006571833044104322501076808092
40347654345107946713373737062547060536401653012956617387979052445947619094013143666088208645002153616185
80695308690215893426747474125094121072803306025913234775958104891895238188026287332176417290004307232371
16139061738043178685349494825018824214560661205182646955191620978379047637605257466435283458000861446474
32278123476086357370698989650037648429121322410365293910383241956758095275210514932870566916001722892948
64556246952172714741397979300075296858242644820730587820766483913516190550421029865741133832003445785891
12911249390434542948279595860015059371648528964146117564153296782703238110084205973148226766400689157179
25822498780869085896559191720030118743297057928292235128306593565406476220168411946296453532801378314359
51644997561738171793118383440060237486594115856584470256613187130812952440336823892592907065602756628718
10328999512347634358623676688012047497318823171316894051322637426162590488067364778518581413120551325741
20657999024695268717247353376024094994637646342633788102645274852325180976134729557037162826241102651487
41315998049390537434494706752048189989275292685267576205290549704650361952269459114074325652482205302974
82631996098781074868989413504096379978550585370535152410581099409300723904538918228148651304964410605948
16526399219756214973797882700819275995710117074107030482116219881860144780907783645629730260992882121189
33052798439512429947595765401638551991420234148214060964232439763720289561815567291259460521985764242379
66105596879024859895191530803277103982840468296428121928464879527440579123631134582518921043971528484759
13221119375804971979038306160655420796568093659285624385692975905488115824726226916503784208794305696951
26442238751609943958076612321310841593136187318571248771385951810976231649452453833007568417588611393901
52884477503219887916153224642621683186272374637142497542771903621952463298904907666015136835177222787807
10576895500643977583230644928524336637254474927428499508554380724390492659780981533203027367035444557561
21153791001287955166461289857048673274508949854856999017108761448780985319561963066406054734070889115122
42307582002575910332922579714097346549017899709713998034217522897561970639123926132812109468141778230245
84615164005151820665845159428194693098035799419427996068435045795123941278247852265624218936283556460491
16923032801030364133169031885638938619607159883885599213687009159024788255649570453124843787256711292098
33846065602060728266338063771277877239214319767771198427374018318049576511299140906249687574513422584190
67692131204121456532676127542555754478428639535542396854748036636099153022598281812499375149026845168391
13538426240824291306535225508511150895685727907108479370949607327219830604519656362499875029805369033678
27076852481648582613070451017022301791371455814216958741899214654439661209039312724999750059610738067351
54153704963297165226140902034044603582742911628433917483798429308879322418078625449999500119221476134714
10830740992659433045228180406808920716548582325686783496759685861775864483615725089999900023844295226942
21661481985318866090456360813617841433097164651373566993519371723551728967231450179999800047688590453885
43322963970637732180912721627235682866194329302747133987038743447103457934462900359999600095377180907771
86645927941275464361825443254471365732388658605494267974077486894206915868925800719999200190754361815541
17329185588255092872365088650894273146477731721098853594815497378841383173785160143999840038150872363108
34658371176510185744730177301788546292955463442197707189630994757682766347570320287999680076301744726217
69316742353020371489460354603577092585910926884395414379261989515365532695140640575999360152603489452434
13863348470604074297892070920715418517182185376879082875852397903073106539028128115199872030520697890480
27726696941208148595784141841430837034364370753758165751704795806146213078056256230399744061041395780971
55453393882416297191568283682861674068728741507516331503409591612292426156112512460799488122082791561941
11090678776483259438313656736572334813745748301503266300681918322458485231222502492159897624416558312389
22181357552966518876627313473144669627491496603006532601363836644916970462445004984319795248833116624779
44362715105933037753254626946289339254982993206013065202727673289833940924890009968639590497666233249558
```

(continues on next page)

(continued from previous page)

```
887254302118660755065092538925786785099659864120261304054553465796678818497800199372791809953324664991
177450860423732151013018507785157357019931972824052260810910693159335763699560039874558361990664932998
354901720847464302026037015570314714039863945648104521621821386318671527399120079749116723981329865996
709803441694928604052074031140629428079727891296209043243642772637343054798240159498233447962659731992
141960688338985720810414806228125885615945578259241808648728554527468610959648031899646689592531946398
283921376677971441620829612456251771231891156518483617297457109054937221919296063799293379185063892797
567842753355942883241659224912503542463782313036967234594914218109874443838592127598586758370127785594
113568550671188576648331844982500708492756462607393446918982843621974888767718425519717351674025557118
227137101342377153296663689965001416985512925214786893837965687243949777535436851039434703348051114237
454274202684754306593327379930002833971025850429573787675931374487899555070873702078869406696102228475
908548405369508613186654759860005667942051700859147575351862748975799110141747404157738813392204456950
181709681073901722637330951972001133588410340171829515070372549795159822028349480831547762678440891390
363419362147803445274661903944002267176820680343659030140745099590319644056698961663095525356881782780
726838724295606890549323807888004534353641360687318060281490199180639288113397923326191050713763565560
145367744859121378109864761577600906870728272137463612056298039836127857622679584665238210142752713112
290735489718242756219729523155201813741456544274927224112596079672255715245359169330476420285505426224
581470979436485512439459046310403627482913088549854448225192159344511430490718338660952840571010852448
116294195887297102487891809262080725496582617709970889645038431868902286098143667732190568114202170489
232588391774594204975783618524161450993165235419941779290076863737804572196287335464381136228404340979
465176783549188409951567237048322901986330470839883558580153727475609144392574670928762272456808681958
930353567098376819903134474096645803972660941679767117160307454951218288785149341857524544913617363917
186070713419675363980626894819329160794532188335953423432061490990243657757029868371504908982723472783
372141426839350727961253789638658321589064376671906846864122981980487315514059736743009817965446945567
744282853678701455922507579277316643178128753343813693728245963960974631028119473486019635930893891134
148856570735740291184501515855463328635625750668762738745649192792194926205623894697203927186178778226
297713141471480582369003031710926657271251501337525477491298385584389852411247789394407854372357556453
595426282942961164738006063421853314542503002675050954982596771168779704822495578788815708744715112907
119085256588592232947601212684370662908500600535010190996519354233755940964499115757763141748943022581
238170513177184465895202425368741325817001201070020381993038708467511881928998231515526283497886045162
476341026354368931790404850737482651634002402140040763986077416935023763857996463031052566995772090325
952682052708737863580809701474965303268004804280081527972154833870047527715992926062105133991544180651
190536410541747572716161940294993060653600960856016305594430966774009505543198585212421026798308836130
381072821083495145432323880589986121307201921712032611188861933548019011086397170424842053596617672260
762145642166990290864647761179972242614403843424065222377723867096038022172794340849684107193235344521
152429128433398058172929552235994448522880768684813044475544773419207604434558868169936821438647068904
304858256866796116345859104471988897045761537369626088951089546384152088691177363398736428772941378085
609716513733592232691718208943977794091523074739252177902179093676830417738235472679747285754588275617
121943302746718446538343641788795558818304614947850435580435818735366083547647094535949457150917655123
243886605493436893076687283577591117636609229895700871160871637470732167095294189071898914301835310246
487773210986873786153374567155182235273218459791401742321743274941464334190588378143797828603670620493
975546421973747572306749134310364470546436919582803484643486549882926683811767562875956572073412409874
195109284394749514461349826862072894109287383916560696928697309976585733676235351257519131441468248197
390218568789499028922699653724145788218574767833121393857394619953171467352470702515038262882936496394
780437137578998057845399307448291576437149535666242787714789239906342934704941405030076525765872992789
156087427515799611569079861489658315287429907133248557542957847981268586940988281006015305153174598557
312174855031599223138159722979316630574859814266497115085915695962537173881976562012030610306349197115
624349710063198446276319445958633261149719628532994230171831391925074347763953124024061220612698394231
124869942012639689255263889191726652229943925706598846034366278385014869552790624804812244122539678846
249739884025279378510527778383453304459887851413197692068732556770029739105581249609624488245079357692
499479768050558757021055556766906606891977570282639538413746511354005947821116249921924897649015871538
998959536101117514042111113533813217839551405652790768274930227080118956422324998438497952980317430771
199791907220223502808422222706762643567910281130558153654986045416023791284464999687699590596063486154
```

(continues on next page)

(continued from previous page)

```
399583814440447005616844445413525287135820562261116307309972090832047582568929999375399181192126972308
799167628880894011233688890827050574271641124522326146199441816640951651378599987507983623842539446169
159833525776178802246737778165410114854328224904446522923988836332819033027571999750159672476850788923
319667051552357604493475556330820229708656449808893045847977672665638066055143999500319344953701577846
639334103104715208986951112661640459417312899617786091695955345331276132110287999000638689907403155693
127866820620943041797390222532328091883462579923557218339191069066255226422057599800127737981480631138
255733641241886083594780445064656183766925159847114436678382138132510452844115199600255475962961262274
511467282483772167189560890129312367533850319694228873356764276265020905688230399200510951925922524554
102293456496754433437912178025862473506770063938845774671352855253004181137646079840102190385184504910
204586912993508866875824356051724947013540127877691549342705710506008362275292159680204380770369009821
409173825987017733751648712103449894027080255755383098685411421012016724550584319360408761540738019643
818347651974035467503297424206899788054160511510766197370822842024033449101168638720817523081476039287
163669530394807093500659484841379957610832102302153239474164568404806689820233727744163504616295207857
```

3.4 Markdown Cells

Text can be added to Jupyter Notebooks using Markdown cells. You can change the cell type to Markdown by using the `Cell` menu, the toolbar, or the key shortcut `m`. Markdown is a popular markup language that is a superset of HTML. Its specification can be found here:

https://daringfireball.net/projects/markdown/

3.4.1 Markdown basics

You can make text *italic* or **bold** by surrounding a block of text with a single or double * respectively

You can build nested itemized or enumerated lists:

- One
  - Sublist
    - This
- Sublist - That - The other thing
- Two
- Sublist
- Three
- Sublist

Now another list:

1. Here we go
   1. Sublist
   2. Sublist
2. There we go
3. Now this

You can add horizontal rules:

Here is a blockquote:

> Beautiful is better than ugly. Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts. Special cases aren't special enough to break the rules. Although practicality beats purity. Errors should never pass silently. Unless explicitly silenced. In the face of ambiguity, refuse the temptation to guess. There should be one– and preferably only one –obvious way to do it. Although that way may not be obvious at first unless you're Dutch. Now is better than never. Although never is often better than *right* now. If the implementation is hard to explain, it's a bad idea. If the implementation is easy to explain, it may be a good idea. Namespaces are one honking great idea – let's do more of those!

And shorthand for links:

Jupyter's website

You can use backslash to generate literal characters which would otherwise have special meaning in the Markdown syntax.

```
\*literal asterisks\*
 *literal asterisks*
```

Use double backslash to generate the literal $ symbol.

3.4.2 Headings

You can add headings by starting a line with one (or multiple) # followed by a space, as in the following example:

```
# Heading 1
# Heading 2
## Heading 2.1
## Heading 2.2
```

3.4.3 Embedded code

You can embed code meant for illustration instead of execution in Python:

```
def f(x):
    """a docstring"""
    return x**2
```

or other languages:

```
for (i=0; i<n; i++) {
  printf("hello %d\n", i);
  x += 4;
}
```

3.4.4 LaTeX equations

Courtesy of MathJax, you can include mathematical expressions both inline: $e^{i\pi} + 1 = 0$ and displayed:

$$e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i \tag{3.1}$$

Inline expressions can be added by surrounding the latex code with `$`:

```
$e^{i\pi} + 1 = 0$
```

Expressions on their own line are surrounded by `\begin{equation}` and `\end{equation}`:

```
\begin{equation}
e^x=\sum_{i=0}^\infty \frac{1}{i!}x^i
\end{equation}
```

3.4.5 GitHub flavored markdown

The Notebook webapp supports Github flavored markdown meaning that you can use triple backticks for code blocks:

````
```python
print "Hello World"
```

```javascript
console.log("Hello World")
```
````

Gives:

```
print "Hello World"
```

```
console.log("Hello World")
```

And a table like this:

```
| This | is   |
|------|------|
|   a  | table|
```

A nice HTML Table:

This	is
a	table

3.4.6 General HTML

Because Markdown is a superset of HTML you can even add things like HTML tables:

Header 1

Header 2

row 1, cell 1

row 1, cell 2

row 2, cell 1

row 2, cell 2

3.4.7 Local files

If you have local files in your Notebook directory, you can refer to these files in Markdown cells directly:

```
[subdirectory/]<filename>
```

For example, in the images folder, we have the Python logo:

```
<img src="../images/python_logo.svg" />
```

and a video with the HTML5 video tag:

```
<video controls src="../images/animation.m4v">animation</video>
```

animation

These do not embed the data into the notebook file, and require that the files exist when you are viewing the notebook.

Security of local files

Note that this means that the Jupyter notebook server also acts as a generic file server for files inside the same tree as your notebooks. Access is not granted outside the notebook folder so you have strict control over what files are visible, but for this reason it is highly recommended that you do not run the notebook server with a notebook directory at a high level in your filesystem (e.g. your home directory).

When you run the notebook in a password-protected manner, local file access is restricted to authenticated users unless read-only views are active.

Markdown attachments

Since Jupyter notebook version 5.0, in addition to referencing external file you can attach a file to a markdown cell. To do so drag the file from in a markdown cell while editing it:



Files are stored in cell metadata and will be automatically scrubbed at save-time if not referenced. You can recognized attached images from other files by their url that starts with `attachment:`. For the image above:

```
![pycon-logo.jpg](attachment:pycon-logo.jpg)
```

Keep in mind that attached files will increase the size of your notebook.

You can manually edit the attachment by using the `View > Cell Toolbar > Attachment` menu, but you should not need to.

3.5 Keyboard Shortcut Customization

Starting with Jupyter Notebook 5.0, you can customize the `command` mode shortcuts from within the Notebook Application itself.

Head to the ``**Help**`` menu and select the ``**Edit keyboard Shortcuts**`` item. A dialog will guide you through the process of adding custom keyboard shortcuts.

Keyboard shortcut set from within the Notebook Application will be persisted to your configuration file. A single action may have several shortcuts attached to it.

3.6 Keyboard Shortcut Customization (Pre Notebook 5.0)

Starting with IPython 2.0 keyboard shortcuts in command and edit mode are fully customizable. These customizations are made using the Jupyter JavaScript API. Here is an example that makes the `r` key available for running a cell:

[ ]:
```
%%javascript

Jupyter.keyboard_manager.command_shortcuts.add_shortcut('r', {
    help : 'run cell',
    help_index : 'zz',
    handler : function (event) {
        IPython.notebook.execute_cell();
        return false;
    }}
);
```

"By default the keypress `r`, while in command mode, changes the type of the selected cell to `raw`. This shortcut is overridden by the code in the previous cell, and thus the action no longer be available via the keypress `r`."

There are a couple of points to mention about this API:

- The `help_index` field is used to sort the shortcuts in the Keyboard Shortcuts help dialog. It defaults to `zz`.
- When a handler returns `false` it indicates that the event should stop propagating and the default action should not be performed. For further details about the `event` object or event handling, see the jQuery docs.
- If you don't need a `help` or `help_index` field, you can simply pass a function as the second argument to `add_shortcut`.

[ ]:
```
%%javascript

Jupyter.keyboard_manager.command_shortcuts.add_shortcut('r', function (event) {
    IPython.notebook.execute_cell();
    return false;
});
```

Likewise, to remove a shortcut, use `remove_shortcut`:

[ ]:
```
%%javascript

Jupyter.keyboard_manager.command_shortcuts.remove_shortcut('r');
```

If you want your keyboard shortcuts to be active for all of your notebooks, put the above API calls into your `custom.js` file.

Of course we provide name for majority of existing action so that you do not have to re-write everything, here is for example how to bind `r` back to it's initial behavior:

[ ]:
```
%%javascript

Jupyter.keyboard_manager.command_shortcuts.add_shortcut('r', 'jupyter-notebook:change-
↪cell-to-raw');
```

3.7 Embracing web standards

One of the main reasons why we developed the current notebook web application was to embrace the web technology.

By being a pure web application using HTML, JavaScript, and CSS, the Notebook can get all the web technology improvement for free. Thus, as browser support for different media extend, the notebook web app should be able to be compatible without modification.

This is also true with performance of the User Interface as the speed of JavaScript VM increases.

The other advantage of using only web technology is that the code of the interface is fully accessible to the end user and is modifiable live. Even if this task is not always easy, we strive to keep our code as accessible and reusable as possible. This should allow us - with minimum effort - development of small extensions that customize the behavior of the web interface.

3.7.1 Tampering with the Notebook application

The first tool that is available to you and that you should be aware of are browser "developers tool". The exact naming can change across browser and might require the installation of extensions. But basically they can allow you to inspect/modify the DOM, and interact with the JavaScript code that runs the frontend.

- In Chrome and Safari, Developer tools are in the menu `View > Developer > JavaScript Console`
- In Firefox you might need to install Firebug

Those will be your best friends to debug and try different approaches for your extensions.

Injecting JS

Using magics

The above tools can be tedious for editing edit long JavaScript files. Therefore we provide the `%%javascript` magic. This allows you to quickly inject JavaScript into the notebook. Still the JavaScript injected this way will not survive reloading. Hence, it is a good tool for testing and refining a script.

You might see here and there people modifying css and injecting js into the notebook by reading file(s) and publishing them into the notebook. Not only does this often break the flow of the notebook and make the re-execution of the notebook broken, but it also means that you need to execute those cells in the entire notebook every time you need to update the code.

This can still be useful in some cases, like the `%autosave` magic that allows you to control the time between each save. But this can be replaced by a JavaScript dropdown menu to select the save interval.

[ ]:
```
## you can inspect the autosave code to see what it does.
%autosave??
```

custom.js

To inject JavaScript we provide an entry point: `custom.js` that allows the user to execute and load other resources into the notebook. JavaScript code in `custom.js` will be executed when the notebook app starts and can then be used to customize almost anything in the UI and in the behavior of the notebook.

`custom.js` can be found in the `~/.jupyter/custom/custom.js`. You can share your custom.js with others.

Back to theory

```
[ ]: from jupyter_core.paths import jupyter_config_dir
     jupyter_dir = jupyter_config_dir()
     jupyter_dir
```

and custom js is in

```
[ ]: import os.path
     custom_js_path = os.path.join(jupyter_dir, 'custom', 'custom.js')
```

```
[ ]: #  my custom js
     if os.path.isfile(custom_js_path):
         with open(custom_js_path) as f:
             print(f.read())
     else:
         print("You don't have a custom.js file")
```

Note that `custom.js` is meant to be modified by user. When writing a script, you can define it in a separate file and add a line of configuration into `custom.js` that will fetch and execute the file.

Warning : even if modification of `custom.js` takes effect immediately after browser refresh (except if browser cache is aggressive), *creating* a file in `static/` directory needs a **server restart**.

3.7.2 Exercise :

- Create a `custom.js` in the right location with the following content:

```
alert("hello world from custom.js")
```

- Restart your server and open any notebook.
- Be greeted by custom.js

Have a look at default custom.js, to see it's content and for more explanation.

For the quick ones :

We've seen above that you can change the autosave rate by using a magic. This is typically something I don't want to type every time, and that I don't like to embed into my workflow and documents. (readers don't care what my autosave time is). Let's build an extension that allows us to do it.

Create a dropdown element in the toolbar (DOM `Jupyter.toolbar.element`), you will need

- `Jupyter.notebook.set_autosave_interval(milliseconds)`
- know that 1 min = 60 sec, and 1 sec = 1000 ms

```
var label = jQuery('<label/>').text('AutoScroll Limit:');
var select = jQuery('<select/>')
     //.append(jQuery('<option/>').attr('value', '2').text('2min (default)'))
     .append(jQuery('<option/>').attr('value', undefined).text('disabled'))

     // TODO:
     //the_toolbar_element.append(label)
     //the_toolbar_element.append(select);

select.change(function() {
     var val = jQuery(this).val() // val will be the value in [2]
     // TODO
     // this will be called when dropdown changes

});

var time_m = [1,5,10,15,30];
for (var i=0; i < time_m.length; i++) {
     var ts = time_m[i];
                                          //[2]   ____ this will be `val` on [1]
                                          //     |
                                          //     v
     select.append($('<option/>').attr('value', ts).text(thr+'min'));
     // this will fill up the dropdown `select` with
     // 1 min
     // 5 min
     // 10 min
     // 10 min
     // ...
}
```

A non-interactive example first

I like my cython to be nicely highlighted

```
Jupyter.config.cell_magic_highlight['magic_text/x-cython'] = {}
Jupyter.config.cell_magic_highlight['magic_text/x-cython'].reg = [/^%%cython/]
```

`text/x-cython` is the name of CodeMirror mode name, `magic_` prefix will just patch the mode so that the first line that contains a magic does not screw up the highlighting. `reg`is a list or regular expression that will trigger the change of mode.

Get more documentation

Sadly, you will have to read the js source file (but there are lots of comments) and/or build the JavaScript documentation using yuidoc. If you have `node` and `yui-doc` installed:

```
$ cd ~/jupyter/notebook/notebook/static/notebook/js/
$ yuidoc . --server
warn: (yuidoc): Failed to extract port, setting to the default :3000
info: (yuidoc): Starting YUIDoc@0.3.45 using YUI@3.9.1 with NodeJS@0.10.15
info: (yuidoc): Scanning for yuidoc.json file.
info: (yuidoc): Starting YUIDoc with the following options:
info: (yuidoc):
{ port: 3000,
  nocode: false,
  paths: [ '.' ],
  server: true,
  outdir: './out' }
info: (yuidoc): Scanning for yuidoc.json file.
info: (server): Starting server: http://127.0.0.1:3000
```

and browse http://127.0.0.1:3000 to get documentation

Some convenience methods

By browsing the documentation you will see that we have some convenience methods that allows us to avoid re-inventing the UI every time :

```
Jupyter.toolbar.add_buttons_group([
        {
             'label'   : 'run qtconsole',
             'icon'    : 'fa-terminal', // select your icon from
                                        // http://fontawesome.io/icons/
             'callback': function(){Jupyter.notebook.kernel.execute('%qtconsole')}
        }
        // add more button here if needed.
        ]);
```

with a lot of icons you can select from.

3.7.3 Cell Metadata

The most requested feature is generally to be able to distinguish an individual cell in the notebook, or run a specific action with them. To do so, you can either use `Jupyter.notebook.get_selected_cell()`, or rely on `CellToolbar`. This allows you to register a set of actions and graphical elements that will be attached to individual cells.

Cell Toolbar

You can see some example of what can be done by toggling the `Cell Toolbar` selector in the toolbar on top of the notebook. It provides two default `presets` that are `Default` and `slideshow`. Default allows the user to edit the metadata attached to each cell manually.

First we define a function that takes at first parameter an element on the DOM in which to inject UI element. The second element is the cell this element wis registered with. Then we will need to register that function and give it a name.

Register a callback

[ ]:
```
%%javascript
var CellToolbar = Jupyter.CellToolbar
var toggle =  function(div, cell) {
     var button_container = $(div)

     // let's create a button that shows the current value of the metadata
     var button = $('<button/>').addClass('btn btn-mini').text(String(cell.metadata.
↪foo));

     // On click, change the metadata value and update the button label
     button.click(function(){
                 var v = cell.metadata.foo;
                 cell.metadata.foo = !v;
                 button.text(String(!v));
             })

     // add the button to the DOM div.
     button_container.append(button);
}

 // now we register the callback under the name foo to give the
 // user the ability to use it later
 CellToolbar.register_callback('tuto.foo', toggle);
```

Registering a preset

This function can now be part of many `preset` of the CellToolBar.

[ ]:
```
%%javascript
Jupyter.CellToolbar.register_preset('Tutorial 1',['tuto.foo','default.rawedit'])
Jupyter.CellToolbar.register_preset('Tutorial 2',['slideshow.select','tuto.foo'])
```

You should now have access to two presets :

- Tutorial 1
- Tutorial 2

And check that the buttons you defined share state when you toggle preset. Also check that the metadata of the cell is modified when you click the button, and that when saved on reloaded the metadata is still available.

Exercise:

Try to wrap the all code in a file, put this file in `{jupyter_dir}/custom/<a-name>.js`, and add

```
require(['custom/<a-name>']);
```

in `custom.js` to have this script automatically loaded in all your notebooks.

`require` is provided by a JavaScript library that allow you to express dependency. For simple extension like the previous one we directly mute the global namespace, but for more complex extension you could pass a callback to `require([...], <callback>)` call, to allow the user to pass configuration information to your plugin.

In Python language,

```
require(['a/b', 'c/d'], function( e, f){
    e.something()
    f.something()
})
```

could be read as

```
import a.b as e
import c.d as f
e.something()
f.something()
```

See for example @damianavila "ZenMode" plugin :

```
// read that as
// import custom.zenmode.main as zenmode
require(['custom/zenmode/main'],function(zenmode){
    zenmode.background('images/back12.jpg');
})
```

For the quickest

Try to use the following to bind a dropdown list to `cell.metadata.difficulty.select`.

It should be able to take the 4 following values :

- `<None>`
- `Easy`
- `Medium`
- `Hard`

We will use it to customize the output of the converted notebook depending on the tag on each cell

[1]:
```
# %load soln/celldiff.js
```

[ ]:
```
```

3.8 Importing Jupyter Notebooks as Modules

It is a common problem that people want to import code from Jupyter Notebooks. This is made difficult by the fact that Notebooks are not plain Python files, and thus cannot be imported by the regular Python machinery.

Fortunately, Python provides some fairly sophisticated hooks into the import machinery, so we can actually make Jupyter notebooks importable without much difficulty, and only using public APIs.

[ ]:
```
import io, os, sys, types
```

[ ]:
```
from IPython import get_ipython
from nbformat import read
from IPython.core.interactiveshell import InteractiveShell
```

Import hooks typically take the form of two objects:

1. a Module **Loader**, which takes a module name (e.g. `'IPython.display'`), and returns a Module
2. a Module **Finder**, which figures out whether a module might exist, and tells Python what **Loader** to use

[ ]:
```
def find_notebook(fullname, path=None):
    """find a notebook, given its fully qualified name and an optional path

    This turns "foo.bar" into "foo/bar.ipynb"
    and tries turning "Foo_Bar" into "Foo Bar" if Foo_Bar
    does not exist.
    """
    name = fullname.rsplit('.', 1)[-1]
    if not path:
        path = ['']
    for d in path:
        nb_path = os.path.join(d, name + ".ipynb")
        if os.path.isfile(nb_path):
            return nb_path
        # let import Notebook_Name find "Notebook Name.ipynb"
        nb_path = nb_path.replace("_", " ")
        if os.path.isfile(nb_path):
            return nb_path
```

3.8.1 Notebook Loader

Here we have our Notebook Loader. It's actually quite simple - once we figure out the filename of the module, all it does is:

1. load the notebook document into memory
2. create an empty Module
3. execute every cell in the Module namespace

Since IPython cells can have extended syntax, the IPython transform is applied to turn each of these cells into their pure-Python counterparts before executing them. If all of your notebook cells are pure-Python, this step is unnecessary.

[ ]:
```
class NotebookLoader(object):
    """Module Loader for Jupyter Notebooks"""
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, path=None):
        self.shell = InteractiveShell.instance()
        self.path = path

    def load_module(self, fullname):
        """import a notebook as a module"""
        path = find_notebook(fullname, self.path)

        print ("importing Jupyter notebook from %s" % path)

        # load the notebook object
        with io.open(path, 'r', encoding='utf-8') as f:
            nb = read(f, 4)


        # create the module and add it to sys.modules
        # if name in sys.modules:
        #    return sys.modules[name]
        mod = types.ModuleType(fullname)
        mod.__file__ = path
        mod.__loader__ = self
        mod.__dict__['get_ipython'] = get_ipython
        sys.modules[fullname] = mod

        # extra work to ensure that magics that would affect the user_ns
        # actually affect the notebook module's ns
        save_user_ns = self.shell.user_ns
        self.shell.user_ns = mod.__dict__

        try:
          for cell in nb.cells:
            if cell.cell_type == 'code':
                # transform the input to executable Python
                code = self.shell.input_transformer_manager.transform_cell(cell.source)
                # run the code in themodule
                exec(code, mod.__dict__)
        finally:
            self.shell.user_ns = save_user_ns
        return mod
```

3.8.2 The Module Finder

The finder is a simple object that tells you whether a name can be imported, and returns the appropriate loader. All this one does is check, when you do:

```
import mynotebook
```

it checks whether `mynotebook.ipynb` exists. If a notebook is found, then it returns a NotebookLoader.

Any extra logic is just for resolving paths within packages.

[ ]:
```
class NotebookFinder(object):
    """Module finder that locates Jupyter Notebooks"""
    def __init__(self):
        self.loaders = {}

    def find_module(self, fullname, path=None):
        nb_path = find_notebook(fullname, path)
        if not nb_path:
            return

        key = path
        if path:
            # lists aren't hashable
            key = os.path.sep.join(path)

        if key not in self.loaders:
            self.loaders[key] = NotebookLoader(path)
        return self.loaders[key]
```

3.8.3 Register the hook

Now we register the `NotebookFinder` with `sys.meta_path`

[ ]:
```
sys.meta_path.append(NotebookFinder())
```

After this point, my notebooks should be importable.

Let's look at what we have in the CWD:

[ ]:
```
ls nbpackage
```

So I should be able to `import nbpackage.mynotebook`.

[ ]:
```
import nbpackage.mynotebook
```

Aside: displaying notebooks

Here is some simple code to display the contents of a notebook with syntax highlighting, etc.

[ ]:
```
from pygments import highlight
from pygments.lexers import PythonLexer
from pygments.formatters import HtmlFormatter

from IPython.display import display, HTML

formatter = HtmlFormatter()
lexer = PythonLexer()

# publish the CSS for pygments highlighting
display(HTML("""
```

(continues on next page)

(continued from previous page)

```
<style type='text/css'>
%s
</style>
""" % formatter.get_style_defs()
))
```

[ ]:
```
def show_notebook(fname):
    """display a short summary of the cells of a notebook"""
    with io.open(fname, 'r', encoding='utf-8') as f:
        nb = read(f, 4)
    html = []
    for cell in nb.cells:
        html.append("<h4>%s cell</h4>" % cell.cell_type)
        if cell.cell_type == 'code':
            html.append(highlight(cell.source, lexer, formatter))
        else:
            html.append("<pre>%s</pre>" % cell.source)
    display(HTML('\n'.join(html)))

show_notebook(os.path.join("nbpackage", "mynotebook.ipynb"))
```

So my notebook has some code cells, one of which contains some IPython syntax.

Let's see what happens when we import it

[ ]:
```
from nbpackage import mynotebook
```

Hooray, it imported! Does it work?

[ ]:
```
mynotebook.foo()
```

Hooray again!

Even the function that contains IPython syntax works:

[ ]:
```
mynotebook.has_ip_syntax()
```

3.8.4 Notebooks in packages

We also have a notebook inside the `nb` package, so let's make sure that works as well.

[ ]:
```
ls nbpackage/nbs
```

Note that the `__init__.py` is necessary for `nb` to be considered a package, just like usual.

[ ]:
```
show_notebook(os.path.join("nbpackage", "nbs", "other.ipynb"))
```

[ ]:
```
from nbpackage.nbs import other
other.bar(5)
```

So now we have importable notebooks, from both the local directory and inside packages.

I can even put a notebook inside IPython, to further demonstrate that this is working properly:

[ ]:
```
import shutil
from IPython.paths import get_ipython_package_dir

utils = os.path.join(get_ipython_package_dir(), 'utils')
shutil.copy(os.path.join("nbpackage", "mynotebook.ipynb"),
            os.path.join(utils, "inside_ipython.ipynb")
)
```

and import the notebook from `IPython.utils`

[ ]:
```
from IPython.utils import inside_ipython
inside_ipython.whatsmyname()
```

This approach can even import functions and classes that are defined in a notebook using the `%%cython` magic.

3.9 Connecting to an existing IPython kernel using the Qt Console

3.9.1 The Frontend/Kernel Model

The traditional IPython (`ipython`) consists of a single process that combines a terminal based UI with the process that runs the users code.

While this traditional application still exists, the modern Jupyter consists of two processes:

- Kernel: this is the process that runs the users code.
- Frontend: this is the process that provides the user interface where the user types code and sees results.

Jupyter currently has 3 frontends:

- Terminal Console (`jupyter console`)
- Qt Console (`jupyter qtconsole`)
- Notebook (`jupyter notebook`)

The Kernel and Frontend communicate over a ZeroMQ/JSON based messaging protocol, which allows multiple Frontends (even of different types) to communicate with a single Kernel. This opens the door for all sorts of interesting things, such as connecting a Console or Qt Console to a Notebook's Kernel. For example, you may want to connect a Qt console to your Notebook's Kernel and use it as a help browser, calling `??` on objects in the Qt console (whose pager is more flexible than the one in the notebook).

This Notebook describes how you would connect another Frontend to an IPython Kernel that is associated with a Notebook. The commands currently given here are specific to the IPython kernel.

3.9.2 Manual connection

To connect another Frontend to a Kernel manually, you first need to find out the connection information for the Kernel using the `%connect_info` magic:

[ ]:
```
%connect_info
```

You can see that this magic displays everything you need to connect to this Notebook's Kernel.

3.9.3 Automatic connection using a new Qt Console

You can also start a new Qt Console connected to your current Kernel by using the `%qtconsole` magic. This will detect the necessary connection information and start the Qt Console for you automatically.

[ ]:
```
a = 10
```

[ ]:
```
%qtconsole
```

The Markdown parser included in the Jupyter Notebook is MathJax-aware. This means that you can freely mix in mathematical expressions using the MathJax subset of Tex and LaTeX. Some examples from the MathJax demos site are reproduced below, as well as the Markdown+TeX source.

3.10 Motivating Examples

3.10.1 The Lorenz Equations

Source

```
\begin{align}
\dot{x} & = \sigma(y-x) \\
\dot{y} & = \rho x - y - xz \\
\dot{z} & = -\beta z + xy
\end{align}
```

Display

3.10.2 The Cauchy-Schwarz Inequality

Source

```
\begin{equation*}
\left( \sum_{k=1}^n a_k b_k \right)^2 \leq \left( \sum_{k=1}^n a_k^2 \right) \left( \sum_
↪{k=1}^n b_k^2 \right)
\end{equation*}
```

Display

$$\left(\sum_{k=1}^{n} a_k b_k\right)^2 \leq \left(\sum_{k=1}^{n} a_k^2\right) \left(\sum_{k=1}^{n} b_k^2\right)$$

3.10.3 A Cross Product Formula

Source

```
\begin{equation*}
\mathbf{V}_1 \times \mathbf{V}_2 =  \begin{vmatrix}
\mathbf{i} & \mathbf{j} & \mathbf{k} \\
\frac{\partial X}{\partial u} &  \frac{\partial Y}{\partial u} & 0 \\
\frac{\partial X}{\partial v} &  \frac{\partial Y}{\partial v} & 0
\end{vmatrix}
\end{equation*}
```

Display

$$\mathbf{V}_1 \times \mathbf{V}_2 = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ \frac{\partial X}{\partial u} & \frac{\partial Y}{\partial u} & 0 \\ \frac{\partial X}{\partial v} & \frac{\partial Y}{\partial v} & 0 \end{vmatrix}$$

3.10.4 The probability of getting (k) heads when flipping (n) coins is

Source

```
\begin{equation*}
P(E)   = {n \choose k} p^k (1-p)^{ n-k}
\end{equation*}
```

Display

$$P(E) = {n \choose k} p^k (1-p)^{n-k}$$

3.10.5 An Identity of Ramanujan

Source

```
\begin{equation*}
\frac{1}{\Bigl(\sqrt{\phi \sqrt{5}}-\phi\Bigr) e^{\frac25 \pi}} =
1+\frac{e^{-2\pi}} {1+\frac{e^{-4\pi}} {1+\frac{e^{-6\pi}}
{1+\frac{e^{-8\pi}} {1+\ldots} } } }
\end{equation*}
```

Display

$$\frac{1}{\left(\sqrt{\phi\sqrt{5}}-\phi\right)e^{\frac{2}{5}\pi}} = 1+\cfrac{e^{-2\pi}}{1+\cfrac{e^{-4\pi}}{1+\cfrac{e^{-6\pi}}{1+\cfrac{e^{-8\pi}}{1+\cdots}}}}$$

3.10.6 A Rogers-Ramanujan Identity

Source

```
\begin{equation*}
1 +  \frac{q^2}{(1-q)}+\frac{q^6}{(1-q)(1-q^2)}+\cdots =
\prod_{j=0}^{\infty}\frac{1}{(1-q^{5j+2})(1-q^{5j+3})},
\quad\quad \text{for $|q|<1$}.
\end{equation*}
```

Display

$$1 + \frac{q^2}{(1-q)} + \frac{q^6}{(1-q)(1-q^2)} + \cdots = \prod_{j=0}^{\infty} \frac{1}{(1-q^{5j+2})(1-q^{5j+3})}, \qquad \text{for } |q| < 1.$$

3.10.7 Maxwell's Equations

Source

```
\begin{align}
\nabla \times \vec{\mathbf{B}} -\, \frac1c\, \frac{\partial\vec{\mathbf{E}}}{\partial t}␣
↪& = \frac{4\pi}{c}\vec{\mathbf{j}} \\   \nabla \cdot \vec{\mathbf{E}} & = 4 \pi \rho \\
\nabla \times \vec{\mathbf{E}}\, +\, \frac1c\, \frac{\partial\vec{\mathbf{B}}}{\partial␣
↪t} & = \vec{\mathbf{0}} \\
\nabla \cdot \vec{\mathbf{B}} & = 0
\end{align}
```

Display

3.10.8 Equation Numbering and References

Equation numbering and referencing will be available in a future version of the Jupyter notebook.

3.10.9 Inline Typesetting (Mixing Markdown and TeX)

While display equations look good for a page of samples, the ability to mix math and *formatted* **text** in a paragraph is also important.

Source

```
This expression $\sqrt{3x-1}+(1+x)^2$ is an example of a TeX inline equation in a␣
↪[Markdown-formatted](https://daringfireball.net/projects/markdown/) sentence.
```

Display

This expression $\sqrt{3x-1}+(1+x)^2$ is an example of a TeX inline equation in a Markdown-formatted sentence.

3.10.10 Other Syntax

You will notice in other places on the web that $$ are needed explicitly to begin and end MathJax typesetting. This is **not** required if you will be using TeX environments, but the Jupyter notebook will accept this syntax on legacy notebooks.

3.10.11 Source

```
$$
\begin{array}{c}
y_1 \\\
y_2 \mathtt{t}_i \\\
z_{3,4}
\end{array}
$$
```

```
$$
\begin{array}{c}
y_1 \cr
y_2 \mathtt{t}_i \cr
y_{3}
\end{array}
$$
```

```
$$\begin{eqnarray}
x' &=& &x \sin\phi &+& z \cos\phi \\
z' &=& - &x \cos\phi &+& z \sin\phi \\
\end{eqnarray}$$
```

```
$$
x=4
$$
```

3.10.12 Display

$$y_1$$
$$y_2\texttt{t}_i$$
$$z_{3,4}$$

$$y_1$$
$$y_2\texttt{t}_i$$
$$y_3$$

to

$$x' = x\sin\phi + z\cos\phi$$
$$z' = -x\cos\phi + z\sin\phi$$
(3.2)

=

+
- =
+$x\cos\phi$
$z\sin\phi$

$$x = 4$$

CHAPTER

FOUR

WHAT TO DO WHEN THINGS GO WRONG

First, have a look at the common problems listed below. If you can figure it out from these notes, it will be quicker than asking for help.

Check that you have the latest version of any packages that look relevant. Unfortunately it's not always easy to figure out what packages are relevant, but if there was a bug that's already been fixed, it's easy to upgrade and get on with what you wanted to do.

4.1 Jupyter fails to start

- Have you installed it? ;-)
- If you're using a menu shortcut or Anaconda launcher to start it, try opening a terminal or command prompt and running the command `jupyter notebook`.
- If it can't find `jupyter`, you may need to configure your `PATH` environment variable. If you don't know what that means, and don't want to find out, just (re)install Anaconda with the default settings, and it should set up PATH correctly.
- If Jupyter gives an error that it can't find `notebook`, check with pip or conda that the `notebook` package is installed.
- Try running `jupyter-notebook` (with a hyphen). This should normally be the same as `jupyter notebook` (with a space), but if there's any difference, the version with the hyphen is the 'real' launcher, and the other one wraps that.

4.2 Jupyter doesn't load or doesn't work in the browser

- Try in another browser (e.g. if you normally use Firefox, try with Chrome). This helps pin down where the problem is.
- Try disabling any browser extensions and/or any Jupyter extensions you have installed.
- Some internet security software can interfere with Jupyter. If you have security software, try turning it off temporarily, and look in the settings for a more long-term solution.
- In the address bar, try changing between `localhost` and `127.0.0.1`. They should be the same, but in some cases it makes a difference.

4.3 Jupyter can't start a kernel

Files called *kernel specs* tell Jupyter how to start different kinds of kernels. To see where these are on your system, run `jupyter kernelspec list`:

```
$ jupyter kernelspec list
Available kernels:
  python3      /home/takluyver/.local/lib/python3.6/site-packages/ipykernel/resources
  bash         /home/takluyver/.local/share/jupyter/kernels/bash
  ir           /home/takluyver/.local/share/jupyter/kernels/ir
```

There's a special fallback for the Python kernel: if it doesn't find a real kernelspec, but it can import the `ipykernel` package, it provides a kernel which will run in the same Python environment as the notebook server. A path ending in `ipykernel/resources`, like in the example above, is this default kernel. The default often does what you want, so if the `python3` kernelspec points somewhere else and you can't start a Python kernel, try deleting or renaming that kernelspec folder to expose the default.

If your problem is with another kernel, not the Python one we maintain, you may need to look for support about that kernel.

4.4 Python Environments

Multiple python environments, whether based on Anaconda or Python Virtual environments, are often the source of reported issues. In many cases, these issues stem from the Notebook server running in one environment, while the kernel and/or its resources, derive from another environment. Indicators of this scenario include:

- `import` statements within code cells producing `ImportError` or `ModuleNotFound` exceptions.
- General kernel startup failures exhibited by nothing happening when attempting to execute a cell.

In these situations, take a close look at your environment structure and ensure all packages required by your notebook's code are installed in the correct environment. If you need to run the kernel from different environments than your Notebook server, check out IPython's documentation for using kernels from different environments as this is the recommended approach. Anaconda's nb_conda_kernels package might also be an option for you in these scenarios.

Another thing to check is the `kernel.json` file that will be located in the aforementioned *kernel specs* directory identified by running `jupyter kernelspec list`. This file will contain an `argv` stanza that includes the actual command to run when launching the kernel. Oftentimes, when reinstalling python environments, a previous `kernel.json` will reference an python executable from an old or non-existent location. As a result, it's always a good idea when encountering kernel startup issues to validate the `argv` stanza to ensure all file references exist and are appropriate.

4.5 Windows Systems

Although Jupyter Notebook is primarily developed on the various flavors of the Unix operating system it also supports Microsoft Windows - which introduces its own set of commonly encountered issues, particularly in the areas of security, process management and lower-level libraries.

4.5.1 pywin32 Issues

The primary package for interacting with Windows' primitives is `pywin32`.

- Issues surrounding the creation of the kernel's communication file utilize `jupyter_core`'s `secure_write()` function. This function ensures a file is created in which only the owner of the file has access. If libraries like `pywin32` are not properly installed, issues can arise when it's necessary to use the native Windows libraries.

  Here's a portion of such a traceback:

```
File "c:\users\jovyan\python\myenv.venv\lib\site-packages\jupyter_core\paths.py",␣
↪line 424, in secure_write
win32_restrict_file_to_user(fname)
File "c:\users\jovyan\python\myenv.venv\lib\site-packages\jupyter_core\paths.py",␣
↪line 359, in win32_restrict_file_to_user
import win32api
ImportError: DLL load failed: The specified module could not be found.
```

- As noted earlier, the installation of `pywin32` can be problematic on Windows configurations. When such an issue occurs, you may need to revisit how the environment was setup. Pay careful attention to whether you're running the 32 or 64 bit versions of Windows and be sure to install appropriate packages for that environment.

  Here's a portion of such a traceback:

```
File "C:\Users\jovyan\AppData\Roaming\Python\Python37\site-packages\jupyter_core\
↪paths.py", line 435, in secure_write
win32_restrict_file_to_user(fname)
File "C:\Users\jovyan\AppData\Roaming\Python\Python37\site-packages\jupyter_core\
↪paths.py", line 361, in win32_restrict_file_to_user
import win32api
ImportError: DLL load failed: %1 is not a valid Win32 application
```

Resolving pywin32 Issues

In this case, your `pywin32` module may not be installed correctly and the following should be attempted:

```
pip install --upgrade pywin32
```

or:

```
conda install --force-reinstall pywin32
```

followed by:

```
python.exe Scripts/pywin32_postinstall.py -install
```

where `Scripts` is located in the active Python's installation location.

- Another common failure specific to Windows environments is the location of various python commands. On *`nix` systems, these typically reside in the `bin` directory of the active Python environment. However, on Windows, these tend to reside in the `Scripts` folder - which is a sibling to `bin`. As a result, when encountering kernel startup issues, again, check the `argv` stanza and verify it's pointing to a valid file. You may find that it's pointing in `bin` when `Scripts` is correct, or the referenced file does not include its `.exe` extension - typically resulting in `FileNotFoundError` exceptions.

4.6 This Worked An Hour Ago

The Jupyter stack is very complex and rightfully so, there's a lot going on. On occassion you might find the system working perfectly well, then, suddenly, you can't get past a certain cell due to `import` failures. In these situations, it's best to ask yourself if any new python files were added to your notebook development area.

These issues are usually evident by carefully analyzing the traceback produced in the notebook error or the Notebook server's command window. In these cases, you'll typically find the Python kernel code (from `IPython` and `ipykernel`) performing *its* imports and notice a file from your Notebook development error included in that traceback followed by an `AttributeError`:

```
File "C:\Users\jovyan\anaconda3\lib\site-packages\ipykernel\connect.py", line 13, in
from IPython.core.profiledir import ProfileDir
File "C:\Users\jovyan\anaconda3\lib\site-packages\IPython_init.py", line 55, in
from .core.application import Application
...
File "C:\Users\jovyan\anaconda3\lib\site-packages\ipython_genutils\path.py", line 13, in
import random
File "C:\Users\jovyan\Desktop\Notebooks\random.py", line 4, in
rand_set = random.sample(english_words_lower_set, 12)
AttributeError: module 'random' has no attribute 'sample'
```

What has happened is that you have named a file that conflicts with an installed package that is used by the kernel software and now introduces a conflict preventing the kernel's startup.

Resolution: You'll need to rename your file. A best practice would be to prefix or *namespace* your files so as not to conflict with any python package.

4.7 Asking for help

As with any problem, try searching to see if someone has already found an answer. If you can't find an existing answer, you can ask questions at:

- The Jupyter Discourse Forum
- The jupyter-notebook tag on Stackoverflow
- Peruse the jupyter/help repository on Github (read-only)
- Or in an issue on another repository, if it's clear which component is responsible. Typical repositories include:
  - jupyter_core - `secure_write()` and file path issues
  - jupyter_client - kernel management issues found in Notebook server's command window.
  - IPython and ipykernel - kernel runtime issues typically found in Notebook server's command window and/or Notebook cell execution.

4.7.1 Gathering Information

Should you find that your problem warrants that an issue be opened in notebook please don't forget to provide details like the following:

- What error messages do you see (within your notebook and, more importantly, in the Notebook server's command window)?
- What platform are you on?
- How did you install Jupyter?
- What have you tried already?

The `jupyter troubleshoot` command collects a lot of information about your installation, which can also be useful.

When providing textual information, it's most helpful if you can *scrape* the contents into the issue rather than providing a screenshot. This enables others to select pieces of that content so they can search more efficiently and try to help.

Remember that it's not anyone's job to help you. We want Jupyter to work for you, but we can't always help everyone individually.

CHAPTER

FIVE

CHANGELOG

A summary of changes in the Jupyter notebook. For more detailed information, see GitHub.

Use `pip install notebook --upgrade` or `conda upgrade notebook` to upgrade to the latest release.

We strongly recommend that you upgrade pip to version 9+ of pip before upgrading `notebook`.

Use `pip install pip --upgrade` to upgrade pip. Check pip version with `pip --version`.

5.1 6.4.8

(Full Changelog)

5.1.1 Bugs fixed

- Fix to remove potential memory leak on Jupyter Notebooks ZMQChannelHandler code #6251 (@Vishwajeet0510)

5.1.2 Contributors to this release

(GitHub contributors page for this release)

@Vishwajeet0510

5.2 6.4.7

(Full Changelog)

5.2.1 Bugs fixed

- Fix Chinese punctuation #6268 (@LiHua-Official)
- Add date field to kernel message header #6265 (@kevin-bates)
- Fix deprecation warning #6253 (@tornaria)

5.2.2 Maintenance and upkeep improvements

- Enforce labels on PRs #6235 (@blink1073)
- Fix: CI error for python 3.6 & macOS #6215 (@penguinolog)

5.2.3 Other merged PRs

- handle KeyError when get session #6245 (@ccw630)
- Updated doc for passwd #6209 (@antoinecarme)

5.2.4 Contributors to this release

(GitHub contributors page for this release)

@antoinecarme | @blink1073 | @ccw630 | @kevin-bates | @LiHua-Official | @penguinolog | @tornaria

5.3 6.4.6

(Full Changelog)

5.3.1 Bugs fixed

- Fix `asyncio` error when opening notebooks #6221 (@dleen)
- Change to use a universal Chinese translation on certain words #6218 (@jackexu)
- Fix Chinese translation typo #6211 (@maliubiao
- Fix `send2trash` tests failing on Windows #6127 (@dolfinus)

5.3.2 Maintenance and upkeep improvements

- TST: don't look in user site for serverextensions #6233 (@bnavigator)
- Enable terminal tests as `pywinpty` is ported for python 3.9 #6228 (@nsait-linaro)

5.3.3 Contributors to this release

(GitHub contributors page for this release)

@bnavigator | @dleen | @dolfinus | @jackexu | @kevin-bates | @maliubiao | @nsait-linaro | @takluyver | @Zsailer

5.4 6.4.5

(Full Changelog)

5.4.1 Bug fixes

- Recover from failure to render mimetype #6181 (@martinRenou)

5.4.2 Maintenance and upkeep improvements

- Fix crypto handling #6197 (@blink1073)
- Fix `jupyter_client` warning #6178 (@martinRenou)

5.4.3 Documentation improvements

- Fix nbsphinx settings #6200 (@mgeier)
- Fully revert the pinning of `nbsphinx` to 0.8.6 #6201 (@kevin-bates)
- Pin `nbsphinx` to 0.8.6, clean up orphaned resources #6194 (@kevin-bates)
- Fix typo in docstring #6188 (@jgarte)

5.4.4 Contributors to this release

(GitHub contributors page for this release)

@blink1073 | @jgarte | @kevin-bates | @martinRenou | @mgeier

5.5 6.4.4

(Full Changelog)

5.5.1 Documentation improvements

- Update Manual Release Instructions #6152 (@blink1073)

5.5.2 Other merged PRs

- Use default JupyterLab CSS sanitizer options for Markdown #6160 (@krassowski)
- Fix syntax highlight #6128 (@massongit)

5.5.3 Contributors to this release

(GitHub contributors page for this release)

@blink1073 | @kevin-bates | @krassowski | @massongit | @minrk | @Zsailer

5.6 6.4.3

(Full Changelog)

5.6.1 Bugs fixed

- Add @babel/core dependency #6133 (@afshin)
- Switch webpack to production mode #6131 (@afshin)

5.6.2 Maintenance and upkeep improvements

- Clean up link checking #6130 (@blink1073)

5.6.3 Contributors to this release

(GitHub contributors page for this release)

@afshin | @blink1073 | @Zsailer

5.7 6.4.2

(Full Changelog)

5.7.1 Bugs fixed

- Add missing file to manifest #6122 (@afshin)
- Fix issue #3218 #6108 (@Nazeeh21)
- Fix version of jupyter-packaging in pyproject.toml #6101 (@frenzymadness)
- "#element".tooltip is not a function on home page fixed. #6070 (@ilayh123)

5.7.2 Maintenance and upkeep improvements

- Enhancements to the desktop entry #6099 (@Amr-Ibra)
- Add missing spaces to help messages in config file #6085 (@saiwing-yeung)

5.7.3 Contributors to this release

(GitHub contributors page for this release)

@afshin | @Amr-Ibra | @frenzymadness | @ilayh123 | @kevin-bates | @Nazeeh21 | @saiwing-yeung

5.8 6.4.0

(Full Changelog)

5.8.1 Bugs fixed

- Fix Handling of Encoded Paths in Save As Dialog #6030 (@afshin)
- Fix: split_cell doesn't always split cell #6017 (@gamestrRUS)
- Correct 'Content-Type' headers #6026 (@faucct)
- Fix skipped tests & remove deprecation warnings #6018 (@befeleme)
- [Gateway] Track only this server's kernels #5980 (@kevin-bates)
- Bind the HTTPServer in start #6061

5.8.2 Maintenance and upkeep improvements

- Revert "do not apply asyncio patch for tornado >=6.1" #6052 (@minrk)
- Use Jupyter Releaser #6048 (@afshin)
- Add Workflow Permissions for Lock Bot #6042 (@jtpio)
- Fixes related to the recent changes in the documentation #6021 (@frenzymadness)
- Add maths checks in CSS reference test #6035 (@stef4k)
- Add Issue Lock and Answered Bots #6019 (@afshin)

5.8.3 Documentation improvements

- Spelling correction #6045 (@wggillen)
- Minor typographical and comment changes #6025 (@misterhay)
- Fixes related to the recent changes in the documentation #6021 (@frenzymadness)
- Fix readthedocs environment #6020 (@blink1073)

5.8.4 Contributors to this release

(GitHub contributors page for this release)

@afshin | @befeleme | @blink1073 | @faucct | @frenzymadness | @gamestrRUS | @jtpio | @kevin-bates | @minrk | @misterhay | @stef4k | @wggillen

5.9 6.3.0

5.9.1 Merged PRs

- Add square logo and desktop entry files #6010 (@befeleme)
- Modernize Changelog #6008 (@afshin)
- Add missing "import inspect" #5999 (@mgeier)
- Add Codecov badge to README #5989 (@thomasrockhu)
- Remove configuration for nosetests from setup.cfg #5986 (@frenzymadness)
- Update security.rst #5978 (@dlrice)
- Docs-Translations: Updated Hindi and Chinese Readme.md #5976 (@rjn01)
- Allow /metrics by default if auth is off #5974 (@blairdrummond)
- Skip terminal tests on Windows 3.9+ (temporary) #5968 (@kevin-bates)
- Update GatewayKernelManager to derive from AsyncMappingKernelManager #5966 (@kevin-bates)
- Drop use of deprecated pyzmq.ioloop #5965 (@kevin-bates)
- Drop support for Python 3.5 #5962 (@kevin-bates)
- Allow jupyter_server-based contents managers in notebook #5957 (@afshin)
- Russian translation fixes #5954 (@insolor)
- Increase culling test idle timeout #5952 (@kevin-bates)
- Re-enable support for answer_yes flag #5941 (@afshin)
- Replace Travis and Appveyor with Github Actions #5938 (@kevin-bates)
- DOC: Server extension, extra docs on configuration/authentication. #5937 (@Carreau)

5.9.2 Contributors to this release

(GitHub contributors page for this release)

@abielhammonds | @afshin | @ajharry | @Alokrar | @befeleme | @blairdrummond | @blink1073 | @bollwyvl | @Carreau | @ChenChenDS | @cosmoscalibur | @dlrice | @dwanneruchi | @ElisonSherton | @FazeelUsmani | @frenzymadness | @goerz | @insolor | @jasongrout | @JianghuiDu | @JuzerShakir | @kevin-bates | @Khalilsqu | @meeseeksdev | @mgeier | @michaelpedota | @mjbright | @MSeal | @ncoughlin | @NTimmons | @ProsperousHeart | @rjn01 | @slw07g | @stenivan | @takluyver | @thomasrockhu | @wgilpin | @wxtt522 | @yuvipanda | @Zsailer

5.10 6.2.0

5.11 Merged PRs

- Increase minimum tornado version (5933)
- Adjust skip decorators to avoid remaining dependency on nose (5932)
- Ensure that cell ids persist after save (5928)
- Add reconnection to Gateway (form nb2kg) (5924)
- Fix some typos (5917)
- Handle TrashPermissionError, now that it exist (5894)

Thank you to all the contributors:

- @kevin-bates
- @mishaschwartz
- @oyvsyo
- @user202729
- @stefanor

5.12 6.1.6

5.13 Merged PRs

- do not require nose for testing (5826)
- [docs] Update Chinese and Hindi readme.md (5823)
- Add support for creating terminals via GET (5813)
- Made doc translations in Hindi and Chinese (5787)

Thank you to all the contributors:

- @pgajdos
- @rjn01
- @kevin-bates
- @virejdasani

5.14 6.1.5

6.1.5 is a security release, fixing one vulnerability:

- Fix open redirect vulnerability GHSA-c7vm-f5p4-8fqh (CVE to be assigned)

5.15 6.1.4

- Fix broken links to jupyter documentation (5686)
- Add additional entries to troubleshooting section (5695)
- Revert change in page alignment (5703)
- Bug fix: remove double encoding in download files (5720)
- Fix typo for Check in zh_CN (5730)
- Require a file name in the "Save As" dialog (5733)

Thank you to all the contributors:

- bdbai
- Jaipreet Singh
- Kevin Bates
- Pavel Panchekha
- Zach Sailer

5.16 6.1.3

- Title new buttons with label if action undefined (5676)

Thank you to all the contributors:

- Kyle Kelley

5.17 6.1.2

- Fix russian message format for delete/duplicate actions (5662)
- Remove unnecessary import of bind_unix_socket (5666)
- Tooltip style scope fix (5672)

Thank you to all the contributors:

- Dmitry Akatov
- Kevin Bates
- Magda Stenius

5.18 6.1.1

- Prevent inclusion of requests_unixsocket on Windows (5650)

Thank you to all the contributors:

- Kevin Bates

5.19 6.1.0

Please note that this repository is currently maintained by a skeleton crew of maintainers from the Jupyter community. For our approach moving forward, please see this notice from the README. Thank you.

Here is an enumeration of changes made since the last release and included in 6.1.0.

- Remove deprecated encoding parameter for Python 3.9 compatibility. (5174)
- Add support for async kernel management (4479)
- Fix typo in password_required help message (5320)
- Gateway only: Ensure launch and request timeouts are in sync (5317)
- Update Markdown Cells example to HTML5 video tag (5411)
- Integrated LoginWidget into edit to enable users to logout from the t… (5406)
- Update message about minimum Tornado version (5222)
- Logged notebook type (5425)
- Added nl language (5354)
- Add UNIX socket support to notebook server. (4835)
- Update CodeMirror dependency (5198)
- Tree added download multiple files (5351)
- Toolbar buttons tooltip: show help instead of label (5107)
- Remove unnecessary import of requests_unixsocket (5451)
- Add ability to cull terminals and track last activity (5372)
- Code refactoring notebook.js (5352)
- Install terminado for docs build (5462)
- Convert notifications JS test to selenium (5455)
- Add cell attachments to markdown example (5412)
- Add Japanese document (5231)
- Migrate Move multiselection test to selenium (5158)
- Use `cmdtrl-enter` to run a cell (5120)
- Fix broken “Raw cell MIME type” dialog (5385)
- Make a notebook writable after successful save-as (5296)
- Add actual watch script (4738)
- Added `--autoreload` flag to `NotebookApp` (4795)

- Enable check_origin on gateway websocket communication (5471)
- Restore detection of missing terminado package (5465)
- Culling: ensure `last_activity` attr exists before use (5355)
- Added functionality to allow filter kernels by Jupyter Enterprise Gat… (5484)
- 'Play' icon for run-cell toolbar button (2922)
- Bump minimum version of jQuery to 3.5.0 (5491)
- Remove old JS markdown tests, add a new one in selenium (5497)
- Add support for more RTL languages (5036)
- Make markdown cells stay RTL in edit mode (5037)
- Unforce RTL output display (5039)
- Fixed multicursor backspacing (4880)
- Implemented Split Cell for multicursor (4824)
- Alignment issue [FIXED] (3173)
- MathJax: Support for `\gdef` (4407)
- Another (Minor) Duplicate Code Reduction (5316)
- Update readme regarding maintenance (5500)
- Document contents chunks (5508)
- Backspace deletes empty line (5516)
- The dropdown submenu at notebook page is not keyboard accessible (4732)
- Tooltips visible through keyboard navigation for specified buttons (4729)
- Fix for recursive symlink (4670)
- Fix for the terminal shutdown issue (4180)
- Add japanese translation files (4490)
- Workaround for socket permission errors on Cygwin (4584)
- Implement optional markdown header and footer files (4043)
- Remove double link when using `custom_display_url` (5544)
- Respect `cell.is_editable` during find-and-replace (5545)
- Fix exception causes all over the codebase (5556
- Improve login shell heuristics (5588)
- Added support for `JUPYTER_TOKEN_FILE` (5587)
- Kill notebook itself when server cull idle kernel (5593)
- Implement password hashing with bcrypt (3793)
- Fix broken links (5600)
- Russian internationalization support (5571)
- Add a metadata tag to override notebook direction (ltr/rtl) (5052)
- Paste two images from clipboard in markdown cell (5598)

- Add keyboard shortcuts to menu dropdowns (5525)
- Update codemirror to `5.56.0+components1` (5637)

Thank you to all the contributors:

- Aaron Myatt
- Adam Blake
- Afshin Taylor Darian
- Aman Bansal
- Ben Thayer
- berendjan
- Bruno P. Kinoshita
- bzinberg
- Christophe Cadilhac
- Daiki Katsuragawa
- David Lukes
- Dmitriy Q
- dmpe
- dylanzjy
- dSchurch
- E. M. Bray
- ErwinRussel
- Felix Mönckemeyer
- Grant Nestor
- Jarrad Whitaker
- Jesus Panales Castillo
- Joshua Zeltser
- Karthikeyan Singaravelan
- Kenichi Ito
- Kevin Bates
- Koki Nishihara
- Kris Wilson
- Kyle Kelley
- Laura Merlo
- levinxo
- Luciano Resende
- Luis Cabezon Manchado
- Madhusudhan Srinivasa

- Matthias Geier
- mattn
- Max Klein
- Min RK
- Mingxuan Lin
- Mohammad Mostafa Farzan
- Niko Felger
- Norah Abanumay
- Onno Broekmans
- PierreMB
- pinarkavak
- Ram Rachum
- Reece Hart
- Remi Rampin
- Rohit Sanjay
- Shane Canon
- Simon Li
- Steinar Sturlaugsson
- Steven Silvester
- taohan16
- Thew Dhanat
- Thomas Kluyver
- Toon Baeyens
- Vidar Tonaas Fauske
- Zachary Sailer

5.20 6.0.3

- Dependency updates to fix startup issues on Windows platform
- Add support for nbconvert 6.x
- Creation of recent tab

Thanks for all the contributors:

- Luciano Resende
- Kevin Bates
- ahangsleben
- Zachary Sailer

- Pallavi Bharadwaj
- Thomas Kluyver
- Min RK
- forest0
- Bibo Hao
- Michal Charemza
- Sergey Shevelev
- Shuichiro MAKIGAKI
- krinsman
- TPartida
- Landen McDonald
- Tres DuBiel

5.21 6.0.2

- Update JQuery dependency to version 3.4.1 to fix security vulnerability (CVE-2019-11358)
- Update CodeMirror to version 5.48.4 to fix Python formatting issues
- Continue removing obsolete Python 2.x code/dependencies
- Multiple documentation updates

Thanks for all the contributors:

- David Robles
- Jason Grout
- Kerwin Sun
- Kevin Bates
- Kyle Kelley
- Luciano Resende
- Marcus D Sherman
- Sasaki Takeru
- Tom Jarosz
- Vidar Tonaas Fauske
- Wes Turner
- Zachary Sailer

5.22 6.0.1

- Attempt to re-establish websocket connection to Gateway (4777)
- Add missing react-dom js to package data (4772)

Thanks for all the contributors:

- Eunsoo Park
- Min RK

5.23 6.0

This is the first major release of the Jupyter Notebook since version 5.0 (March 2017).

We encourage users to start trying JupyterLab, which has just announced it's 1.0 release in preparation for a future transition.

- Remove Python 2.x support in favor of Python 3.5 and higher.
- Multiple accessibility enhancements and bug-fixes.
- Multiple translation enhancements and bug-fixes.
- Remove deprecated ANSI CSS styles.
- Native support to forward requests to Jupyter Gateway(s) (Embedded NB2KG).
- Use JavaScript to redirect users to notebook homepage.
- Enhanced SSL/TLS security by using PROTOCOL_TLS which selects the highest ssl/tls protocol version available that both the client and server support. When PROTOCOL_TLS is not available use PROTOCOL_SSLv23.
- Add `?no_track_activity=1` argument to allow API requests. to not be registered as activity (e.g. API calls by external activity monitors).
- Kernels shutting down due to an idle timeout is no longer considered an activity-updating event.
- Further improve compatibility with tornado 6 with improved checks for when websockets are closed.
- Launch the browser with a local file which redirects to the server address including the authentication token. This prevents another logged-in user from stealing the token from command line arguments and authenticating to the server. The single-use token previously used to mitigate this has been removed. Thanks to Dr. Owain Kenway for suggesting the local file approach.
- Respect nbconvert entrypoints as sources for exporters
- Update to CodeMirror to 5.37, which includes f-string syntax for Python 3.6.
- Update jquery-ui to 1.12
- Execute cells by clicking icon in input prompt.
- New "Save as" menu option.
- When serving on a loopback interface, protect against DNS rebinding by checking the `Host` header from the browser. This check can be disabled if necessary by setting `NotebookApp.allow_remote_access`. (Disabled by default while we work out some Mac issues in 3754).
- Add kernel_info_timeout traitlet to enable restarting slow kernels.
- Add `custom_display_host` config option to override displayed URL.

- Add /metrics endpoint for Prometheus Metrics.
- Optimize large file uploads.
- Allow access control headers to be overriden in jupyter_notebook_config.py to support greater CORS and proxy configuration flexibility.
- Add support for terminals on windows.
- Add a "restart and run all" button to the toolbar.
- Frontend/extension-config: allow default json files in a .d directory.
- Allow setting token via jupyter_token env.
- Cull idle kernels using `--MappingKernelManager.cull_idle_timeout`.
- Allow read-only notebooks to be trusted.
- Convert JS tests to Selenium.

Security Fixes included in previous minor releases of Jupyter Notebook and also included in version 6.0.

- Fix Open Redirect vulnerability (CVE-2019-10255) where certain malicious URLs could redirect from the Jupyter login page to a malicious site after a successful login.
- Contains a security fix for a cross-site inclusion (XSSI) vulnerability (CVE-2019–9644), where files at a known URL could be included in a page from an unauthorized website if the user is logged into a Jupyter server. The fix involves setting the `X-Content-Type-Options: nosniff` header, and applying CSRF checks previously on all non-GET API requests to GET requests to API endpoints and the /files/ endpoint.
- Check Host header to more securely protect localhost deployments from DNS rebinding. This is a pre-emptive measure, not fixing a known vulnerability. Use `.NotebookApp.allow_remote_access` and `.NotebookApp.local_hostnames` to configure access.
- Upgrade bootstrap to 3.4, fixing an XSS vulnerability, which has been assigned CVE-2018-14041.
- Contains a security fix preventing malicious directory names from being able to execute javascript.
- Contains a security fix preventing nbconvert endpoints from executing javascript with access to the server API. CVE request pending.

Thanks for all the contributors:

- AAYUSH SINHA
- Aaron Hall, MBA
- Abhinav Sagar
- Adam Rule
- Adeel Ahmad
- Alex Rothberg
- Amy Skerry-Ryan
- Anastasis Germanidis
- Andrés Sánchez
- Arjun Radhakrishna
- Arovit Narula
- Benda Xu
- Björn Grüning

- Brian E. Granger
- Carol Willing
- Celina Kilcrease
- Chris Holdgraf
- Chris Miller
- Ciaran Langton
- Damian Avila
- Dana Lee
- Daniel Farrell
- Daniel Nicolai
- Darío Hereñú
- Dave Aitken
- Dave Foster
- Dave Hirschfeld
- Denis Ledoux
- Dmitry Mikushin
- Dominic Kuang
- Douglas Hanley
- Elliott Sales de Andrade
- Emilio Talamante Lugo
- Eric Perry
- Ethan T. Hendrix
- Evan Van Dam
- Francesco Franchina
- Frédéric Chapoton
- Félix-Antoine Fortin
- Gabriel
- Gabriel Nützi
- Gabriel Ruiz
- Gestalt LUR
- Grant Nestor
- Gustavo Efeiche
- Harsh Vardhan
- Heng GAO
- Hisham Elsheshtawy
- Hong Xu

- Ian Rose
- Ivan Ogasawara
- J Forde
- Jason Grout
- Jessica B. Hamrick
- Jiaqi Liu
- John Emmons
- Josh Barnes
- Karthik Balakrishnan
- Kevin Bates
- Kirit Thadaka
- Kristian Gregorius Hustad
- Kyle Kelley
- Leo Gallucci
- Lilian Besson
- Lucas Seiki Oshiro
- Luciano Resende
- Luis Angel Rodriguez Guerrero
- M Pacer
- Maarten Breddels
- Mac Knight
- Madicken Munk
- Maitiú Ó Ciaráin
- Marc Udoff
- Mathis HAMMEL
- Mathis Rosenhauer
- Matthias Bussonnier
- Matthias Geier
- Max Vovshin
- Maxime Mouchet
- Michael Chirico
- Michael Droettboom
- Michael Heilman
- Michael Scott Cuthbert
- Michal Charemza
- Mike Boyle

- Milos Miljkovic
- Min RK
- Miro Hrončok
- Nicholas Bollweg
- Nitesh Sawant
- Ondrej Jariabka
- Park Hae Jin
- Paul Ivanov
- Paul Masson
- Peter Parente
- Pierre Tholoniat
- Remco Verhoef
- Roland Weber
- Roman Kornev
- Rosa Swaby
- Roy Hyunjin Han
- Sally
- Sam Lau
- Samar Sultan
- Shiti Saxena
- Simon Biggs
- Spencer Park
- Stephen Ward
- Steve (Gadget) Barnes
- Steven Silvester
- Surya Prakash Susarla
- Syed Shah
- Sylvain Corlay
- Thomas Aarholt
- Thomas Kluyver
- Tim
- Tim Head
- Tim Klever
- Tim Metzler
- Todd
- Tom Jorquera

- Tyler Makaro
- Vaibhav Sagar
- Victor
- Vidar Tonaas Fauske
- Vu Minh Tam
- Vít Tuček
- Will Costello
- Will Starms
- William Hosford
- Xiaohan Li
- Yuvi Panda
- ashley teoh
- nullptr

5.24 5.7.8

- Fix regression in restarting kernels in 5.7.5. The restart handler would return before restart was completed.
- Further improve compatibility with tornado 6 with improved checks for when websockets are closed.
- Fix regression in 5.7.6 on Windows where .js files could have the wrong mime-type.
- Fix Open Redirect vulnerability (CVE-2019-10255) where certain malicious URLs could redirect from the Jupyter login page to a malicious site after a successful login. 5.7.7 contained only a partial fix for this issue.

5.25 5.7.6

5.7.6 contains a security fix for a cross-site inclusion (XSSI) vulnerability (CVE-2019–9644), where files at a known URL could be included in a page from an unauthorized website if the user is logged into a Jupyter server. The fix involves setting the `X-Content-Type-Options: nosniff` header, and applying CSRF checks previously on all non-GET API requests to GET requests to API endpoints and the /files/ endpoint.

The attacking page is able to access some contents of files when using Internet Explorer through script errors, but this has not been demonstrated with other browsers.

5.26 5.7.5

- Fix compatibility with tornado 6 (4392, 4449).
- Fix opening integer filedescriptor during startup on Python 2 (4349)
- Fix compatibility with asynchronous [KernelManager.restart_kernel]{.title-ref} methods (4412)

5.27 5.7.4

5.7.4 fixes a bug introduced in 5.7.3, in which the `list_running_servers()` function attempts to parse HTML files as JSON, and consequently crashes (4284).

5.28 5.7.3

5.7.3 contains one security improvement and one security fix:

- Launch the browser with a local file which redirects to the server address including the authentication token (4260). This prevents another logged-in user from stealing the token from command line arguments and authenticating to the server. The single-use token previously used to mitigate this has been removed. Thanks to Dr. Owain Kenway for suggesting the local file approach.
- Upgrade bootstrap to 3.4, fixing an XSS vulnerability, which has been assigned CVE-2018-14041 (4271).

5.29 5.7.2

5.7.2 contains a security fix preventing malicious directory names from being able to execute javascript. CVE request pending.

5.30 5.7.1

5.7.1 contains a security fix preventing nbconvert endpoints from executing javascript with access to the server API. CVE request pending.

5.31 5.7.0

New features:

- Update to CodeMirror to 5.37, which includes f-string syntax for Python 3.6 (3816)
- Update jquery-ui to 1.12 (3836)
- Check Host header to more securely protect localhost deployments from DNS rebinding. This is a preemptive measure, not fixing a known vulnerability (3766). Use `.NotebookApp.allow_remote_access` and `.NotebookApp.local_hostnames` to configure access.
- Allow access-control-allow-headers to be overridden (3886)
- Allow configuring max_body_size and max_buffer_size (3829)
- Allow configuring get_secure_cookie keyword-args (3778)
- Respect nbconvert entrypoints as sources for exporters (3879)
- Include translation sources in source distributions (3925, 3931)
- Various improvements to documentation (3799, 3800, 3806, 3883, 3908)

Fixing problems:

- Fix breadcrumb link when running with a base url (3905)

- Fix possible type error when closing activity stream (3907)
- Disable metadata editing for non-editable cells (3744)
- Fix some styling and alignment of prompts caused by regressions in 5.6.0.
- Enter causing page reload in shortcuts editor (3871)
- Fix uploading to the same file twice (3712)

See the 5.7 milestone on GitHub for a complete list of pull requests involved in this release.

Thanks to the following contributors:

- Aaron Hall
- Benjamin Ragan-Kelley
- Bill Major
- bxy007
- Dave Aitken
- Denis Ledoux
- Félix-Antoine Fortin
- Gabriel
- Grant Nestor
- Kevin Bates
- Kristian Gregorius Hustad
- M Pacer
- Madicken Munk
- Maitiu O Ciarain
- Matthias Bussonnier
- Michael Boyle
- Michael Chirico
- Mokkapati, Praneet(ES)
- Peter Parente
- Sally Wilsak
- Steven Silvester
- Thomas Kluyver
- Walter Martin

5.32 5.6.0

New features:

- Execute cells by clicking icon in input prompt (3535, 3687)
- New "Save as" menu option (3289)
- When serving on a loopback interface, protect against DNS rebinding by checking the `Host` header from the browser (3714). This check can be disabled if necessary by setting `NotebookApp.allow_remote_access`. (Disabled by default while we work out some Mac issues in 3754).
- Add kernel_info_timeout traitlet to enable restarting slow kernels (3665)
- Add `custom_display_host` config option to override displayed URL (3668)
- Add /metrics endpoint for Prometheus Metrics (3490)
- Update to MathJax 2.7.4 (3751)
- Update to jQuery 3.3 (3655)
- Update marked to 0.4 (3686)

Fixing problems:

- Don't duplicate token in displayed URL (3656)
- Clarify displayed URL when listening on all interfaces (3703)
- Don't trash non-empty directories on Windows (3673)
- Include LICENSE file in wheels (3671)
- Don't show "0 active kernels" when starting the notebook (3696)

Testing:

- Add find replace test (3630)
- Selenium test for deleting all cells (3601)
- Make creating a new notebook more robust (3726)

Thanks to the following contributors:

- Arovit Narula (arovit)
- lucasoshiro (lucasoshiro)
- M Pacer (mpacer)
- Thomas Kluyver (takluyver)
- Todd (toddrme2178)
- Yuvi Panda (yuvipanda)

See the 5.6 milestone on GitHub for a complete list of pull requests involved in this release.

5.33 5.5.0

New features:

- The files list now shows file sizes (3539)
- Add a quit button in the dashboard (3004)
- Display hostname in the terminal when running remotely (3356, 3593)
- Add slides exportation/download to the menu (3287)
- Add any extra installed nbconvert exporters to the "Download as" menu (3323)
- Editor: warning when overwriting a file that is modified on disk (2783)
- Display a warning message if cookies are not enabled (3511)
- Basic `__version__` reporting for extensions (3541)
- Add `NotebookApp.terminals_enabled` config option (3478)
- Make buffer time between last modified on disk and last modified on last save configurable (3273)
- Allow binding custom shortcuts for 'close and halt' (3314)
- Add description for 'Trusted' notification (3386)
- Add `settings['activity_sources']` (3401)
- Add an `output_updated.OutputArea` event (3560)

Fixing problems:

- Fixes to improve web accessibility (3507)
- Fixed color contrast issue in tree.less (3336)
- Allow cancelling upload of large files (3373)
- Don't clear login cookie on requests without cookie (3380)
- Don't trash files on different device to home dir on Linux (3304)
- Clear waiting asterisks when restarting kernel (3494)
- Fix output prompt when `execution_count` missing (3236)
- Make the 'changed on disk' dialog work when displayed twice (3589)
- Fix going back to root directory with history in notebook list (3411)
- Allow defining keyboard shortcuts for missing actions (3561)
- Prevent default on pageup/pagedown when completer is active (3500)
- Prevent default event handling on new terminal (3497)
- ConfigManager should not write out default values found in the .d directory (3485)
- Fix leak of iopub object in activity monitoring (3424)
- Javascript lint in notebooklist.js (3409)
- Some Javascript syntax fixes (3294)
- Convert native for loop to `Array.forEach()` (3477)
- Disable cache when downloading nbconvert output (3484)

- Add missing digestmod arg to HMAC (3399)
- Log OSErrors failing to create less-critical files during startup (3384)
- Use powershell on Windows (3379)
- API spec improvements, API handler improvements (3368)
- Set notebook to dirty state after change to kernel metadata (3350)
- Use CSP header to treat served files as belonging to a separate origin (3341)
- Don't install gettext into builtins (3330)
- Add missing `import _` (3316, 3326)
- Write `notebook.json` file atomically (3305)
- Fix clicking with modifiers, page title updates (3282)
- Upgrade jQuery to version 2.2 (3428)
- Upgrade xterm.js to 3.1.0 (3189)
- Upgrade moment.js to 2.19.3 (3562)
- Upgrade CodeMirror to 5.35 (3372)
- "Require" pyzmq>=17 (3586)

Documentation:

- Documentation updates and organisation (3584)
- Add section in docs about privacy (3571)
- Add explanation on how to change the type of a cell to Markdown (3377)
- Update docs with confd implementation details (3520)
- Add more information for where `jupyter_notebook_config.py` is located (3346)
- Document options to enable nbextensions in specific sections (3525)
- jQuery attribute selector value MUST be surrounded by quotes (3527)
- Do not execute special notebooks with nbsphinx (3360)
- Other minor fixes in 3288, 3528, 3293, 3367

Testing:

- Testing with Selenium & Sauce labs (3321)
- Selenium utils + markdown rendering tests (3458)
- Convert insert cell tests to Selenium (3508)
- Convert prompt numbers tests to Selenium (3554)
- Convert delete cells tests to Selenium (3465)
- Convert undelete cell tests to Selenium (3475)
- More selenium testing utilities (3412)
- Only check links when build is trigger by Travis Cron job (3493)
- Fix Appveyor build errors (3430)
- Undo patches in teardown before attempting to delete files (3459)

- Get tests running with tornado 5 (3398)
- Unpin ipykernel version on Travis (3223)

Thanks to the following contributors:

- Arovit Narula (arovit)
- Ashley Teoh (ashleytqy)
- Nicholas Bollweg (bollwyvl)
- Alex Rothberg (cancan101)
- Celina Kilcrease (ckilcrease)
- dabuside (dabuside)
- Damian Avila (damianavila)
- Dana Lee (danagilliann)
- Dave Hirschfeld (dhirschfeld)
- Heng GAO (ehengao)
- Leo Gallucci (elgalu)
- Evan Van Dam (evandam)
- forbxy (forbxy)
- Grant Nestor (gnestor)
- Ethan T. Hendrix (hendrixet)
- Miro Hrončok (hroncok)
- Paul Ivanov (ivanov)
- Darío Hereñú (kant)
- Kevin Bates (kevin-bates)
- Maarten Breddels (maartenbreddels)
- Michael Droettboom (mdboom)
- Min RK (minrk)
- M Pacer (mpacer)
- Peter Parente (parente)
- Paul Masson (paulmasson)
- Philipp Rudiger (philippjfr)
- Mac Knight (Shels1909)
- Hisham Elsheshtawy (Sheshtawy)
- Simon Biggs (SimonBiggs)
- Sunil Hari (`@sunilhari`)
- Thomas Kluyver (takluyver)
- Tim Klever (tklever)
- Gabriel Ruiz (unnamedplay-r)

- Vaibhav Sagar (vaibhavsagar)
- William Hosford (whosford)
- Hong (xuhdev)

See the 5.5 milestone on GitHub for a complete list of pull requests involved in this release.

5.34 5.4.1

A security release to fix CVE-2018-8768.

Thanks to Alex for identifying this bug, and Jonathan Kamens and Scott Sanderson at Quantopian for verifying it and bringing it to our attention.

5.35 5.4.0

- Fix creating files and folders after navigating directories in the dashboard (3264).
- Enable printing notebooks in colour, removing the CSS that made everything black and white (3212).
- Limit the completion options displayed in the notebook to 1000, to avoid performance issues with very long lists (3195).
- Accessibility improvements in `tree.html` (3271).
- Added alt-text to the kernel logo image in the notebook UI (3228).
- Added a test on Travis CI to flag if symlinks are accidentally introduced in the future. This should prevent the issue that necessitated `release-5.3.1`{.interpreted-text role="ref"} (3227).
- Use lowercase letters for random IDs generated in our Javascript (3264).
- Removed duplicate code setting `TextCell.notebook` (3256).

Thanks to the following contributors:

- Alex Soderman (asoderman)
- Matthias Bussonnier (Carreau)
- Min RK (minrk)
- Nitesh Sawant (ns23)
- Thomas Kluyver (takluyver)
- Yuvi Panda (yuvipanda)

See the 5.4 milestone on GitHub for a complete list of pull requests involved in this release.

5.36 5.3.1

Replaced a symlink in the repository with a copy, to fix issues installing on Windows (3220).

5.37 5.3.0

This release introduces a couple noteable improvements, such as terminal support for Windows and support for OS trash (files deleted from the notebook dashboard are moved to the OS trash vs. deleted permanently).

- Add support for terminals on windows (3087).
- Add a "restart and run all" button to the toolbar (2965).
- Send files to os trash mechanism on delete (1968).
- Allow programmatic copy to clipboard (3088).
- Use DOM History API for navigating between directories in the file browser (3115).
- Add translated files to folder(docs-translations) (3065).
- Allow non empty dirs to be deleted (3108).
- Set cookie on base_url (2959).
- Allow token-authenticated requests cross-origin by default (2920).
- Change cull_idle_timeout_minimum to 1 from 300 (2910).
- Config option to shut down server after n seconds with no kernels (2963).
- Display a "close" button on load notebook error (3176).
- Add action to command pallette to run CodeMirror's "indentAuto" on selection (3175).
- Add option to specify extra services (3158).
- Warn_bad_name should not use global name (3160).
- Avoid overflow of hidden form (3148).
- Fix shutdown trans loss (3147).
- Find available kernelspecs more efficiently (3136).
- Don't try to translate missing help strings (3122).
- Frontend/extension-config: allow default json files in a .d directory (3116).
- Use [requirejs]{.title-ref} vs. [require]{.title-ref} (3097).
- Fixes some ui bugs in firefox #3044 (3058).
- Compare non-specific language code when choosing to use arabic numerals (3055).
- Fix save-script deprecation (3053).
- Include moment locales in package_data (3051).
- Fix moment locale loading in bidi support (3048).
- Tornado 5: periodiccallback loop arg will be removed (3034).
- Use [/files]{.title-ref} prefix for pdf-like files (3031).
- Add folder for document translation (3022).

- When login-in via token, let a chance for user to set the password (3008).
- Switch to jupyter_core implementation of ensure_dir_exists (3002).
- Send http shutdown request on 'stop' subcommand (3000).
- Work on loading ui translations (2969).
- Fix ansi inverse (2967).
- Add send2trash to requirements for building docs (2964).
- I18n readme.md improvement (2962).
- Add 'reason' field to json error responses (2958).
- Add some padding for stream outputs (3194).
- Always use setuptools in `setup.py` (3206).
- Fix clearing cookies on logout when `base_url` is configured (3207).

Thanks to the following contributors:

- bacboc (bacboc)
- Steven Silvester (blink1073)
- Matthias Bussonnier (Carreau)
- ChungJooHo (ChungJooHo)
- edida (edida)
- Francesco Franchina (`ferdas`)
- forbxy (forbxy)
- Grant Nestor (gnestor)
- Josh Barnes (jcb91)
- JocelynDelalande (JocelynDelalande)
- Karthik Balakrishnan (karthikb351)
- Kevin Bates (kevin-bates)
- Kirit Thadaka (kirit93)
- Lilian Besson (Naereen)
- Maarten Breddels (maartenbreddels)
- Madhu94 (Madhu94)
- Matthias Geier (mgeier)
- Michael Heilman (mheilman)
- Min RK (minrk)
- PHaeJin (PHaeJin)
- Sukneet (Sukneet)
- Thomas Kluyver (takluyver)

See the 5.3 milestone on GitHub for a complete list of pull requests involved in this release.

5.38 5.2.1

- Fix invisible CodeMirror cursor at specific browser zoom levels (2983).
- Fix nbconvert handler causing broken export to PDF (2981).
- Fix the prompt_area argument of the output area constructor. (2961).
- Handle a compound extension in new_untitled (2949).
- Allow disabling offline message buffering (2916).

Thanks to the following contributors:

- Steven Silvester (blink1073)
- Grant Nestor (gnestor)
- Jason Grout (jasongrout)
- Min RK (minrk)
- M Pacer (mpacer)

See the 5.2.1 milestone on GitHub for a complete list of pull requests involved in this release.

5.39 5.2.0

- Allow setting token via jupyter_token env (2921).
- Fix some errors caused by raising 403 in get_current_user (2919).
- Register contents_manager.files_handler_class directly (2917).
- Update viewable_extensions (2913).
- Show edit shortcuts modal after shortcuts modal is hidden (2912).
- Improve edit/view behavior (2911).
- The root directory of the notebook server should never be hidden (2907).
- Fix notebook require config to match tools/build-main (2888).
- Give page constructor default arguments (2887).
- Fix codemirror.less to match codemirror's expected padding layout (2880).
- Add x-xsrftoken to access-control-allow-headers (2876).
- Buffer messages when websocket connection is interrupted (2871).
- Load locale dynamically only when not en-us (2866).
- Changed key strength to 2048 bits (2861).
- Resync jsversion with python version (2860).
- Allow copy operation on modified, read-only notebook (2854).
- Update error handling on apihandlers (2853).
- Test python 3.6 on travis, drop 3.3 (2852).
- Avoid base64-literals in image tests (2851).
- Upgrade xterm.js to 2.9.2 (2849).

- Changed all python variables named file to file_name to not override built_in file (2830).
- Add more doc tests (2823).
- Typos fix (2815).
- Rename and update license [ci skip] (2810).
- Travis builds doc (2808).
- Pull request i18n (2804).
- Factor out output_prompt_function, as is done with input prompt (2774).
- Use rfc5987 encoding for filenames (2767).
- Added path to the resources metadata, the same as in from_filename(...) in nbconvert.exporters.py (2753).
- Make "extrakeys" consistent for notebook and editor (2745).
- Bidi support (2357).

Special thanks to samarsultan and the Arabic Competence and Globalization Center Team at IBM Egypt for adding RTL (right-to-left) support to the notebook!

See the 5.2 milestone on GitHub for a complete list of issues and pull requests involved in this release.

5.40 5.1.0

- Preliminary i18n implementation (2140).
- Expose URL with auth token in notebook UI (2666).
- Fix search background style (2387).
- List running notebooks without requiring `--allow-root` (2421).
- Allow session of type other than notebook (2559).
- Fix search background style (2387).
- Fix some Markdown styling issues (2571), (2691) and (2534).
- Remove keymaps that conflict with non-English keyboards (2535).
- Add session-specific favicons (notebook, terminal, file) (2452).
- Add /api/shutdown handler (2507).
- Include metadata when copying a cell (2349).
- Stop notebook server from command line (2388).
- Improve "View" and "Edit" file handling in dashboard (2449) and (2402).
- Provide a promise to replace use of the `app_initialized.NotebookApp` event (2710).
- Fix disabled collapse/expand output button (2681).
- Cull idle kernels using `--MappingKernelManager.cull_idle_timeout` (2215).
- Allow read-only notebooks to be trusted (2718).

See the 5.1 milestone on GitHub for a complete list of issues and pull requests involved in this release.

5.41 5.0.0

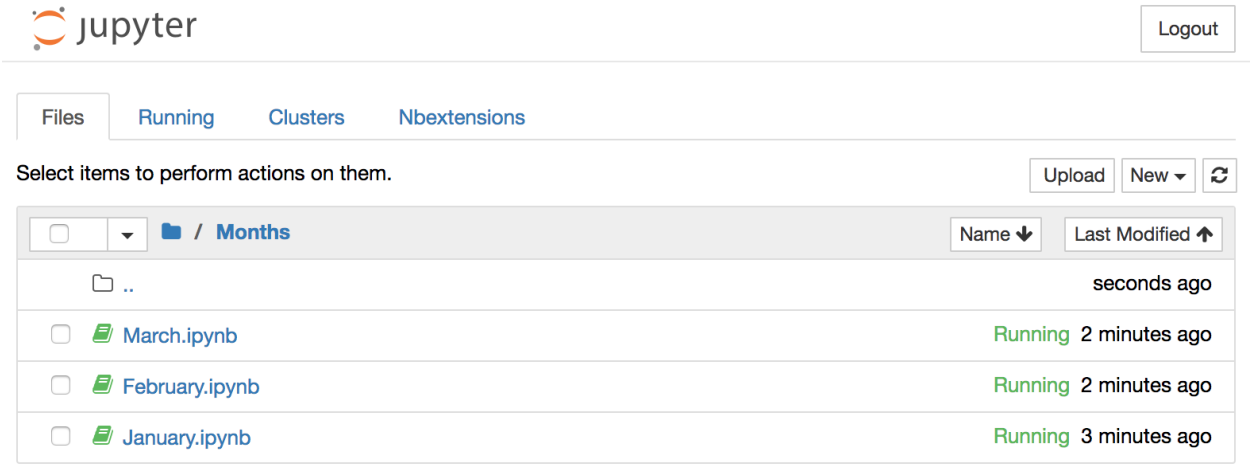
This is the first major release of the Jupyter Notebook since version 4.0 was created by the "Big Split" of IPython and Jupyter.

We encourage users to start trying JupyterLab in preparation for a future transition.

We have merged more than 300 pull requests since 4.0. Some of the major user-facing changes are described here.

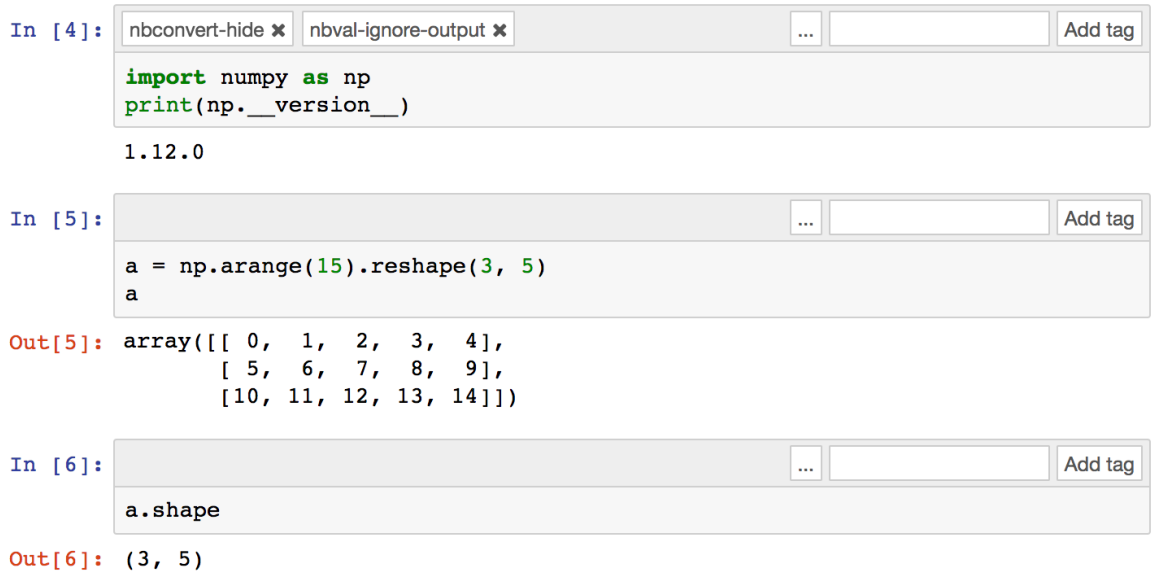
5.41.1 File sorting in the dashboard

Files in the dashboard may now be sorted by last modified date or name (943):



5.41.2 Cell tags

There is a new cell toolbar for adding *cell tags* (2048):



Cell tags are a lightweight way to customise the behaviour of tools working with notebooks; we're working on building support for them into tools like nbconvert and nbval. To start using tags, select `Tags` in the `View > Cell Toolbar` menu in a notebook.

The UI for editing cell tags is basic for now; we hope to improve it in future releases.

5.41.3 Table style

The default styling for tables in the notebook has been updated (1776).

Before:

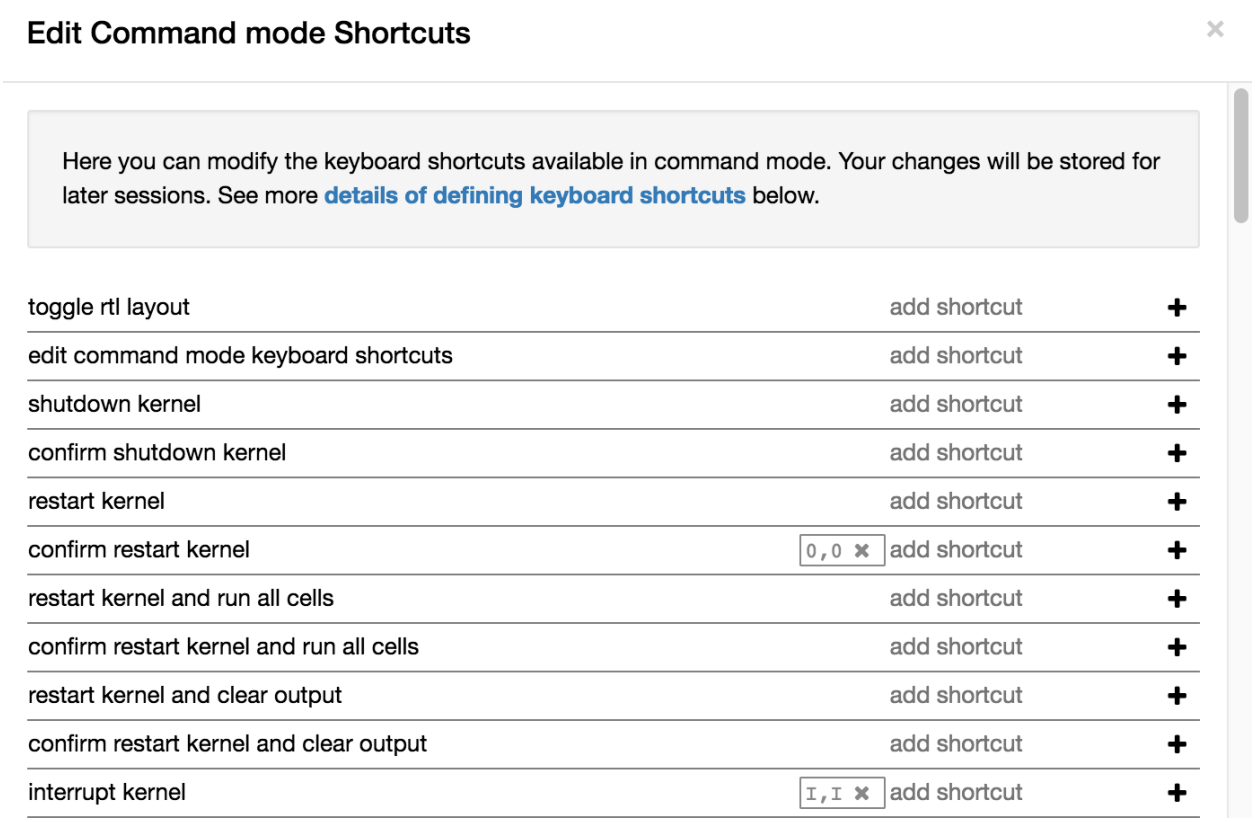
	Acceleration	Cylinders	Displacement	Horsepower	Miles_per_Gallon	Name	Origin	Weight_in_lbs	Year
0	12.0	8	307.0	130.0	18.0	chevrolet chevelle malibu	USA	3504	1970-01-01
1	11.5	8	350.0	165.0	15.0	buick skylark 320	USA	3693	1970-01-01
2	11.0	8	318.0	150.0	18.0	plymouth satellite	USA	3436	1970-01-01
3	12.0	8	304.0	150.0	16.0	amc rebel sst	USA	3433	1970-01-01
4	10.5	8	302.0	140.0	17.0	ford torino	USA	3449	1970-01-01
5	10.0	8	429.0	198.0	15.0	ford galaxie 500	USA	4341	1970-01-01
6	9.0	8	454.0	220.0	14.0	chevrolet impala	USA	4354	1970-01-01
7	8.5	8	440.0	215.0	14.0	plymouth fury iii	USA	4312	1970-01-01
8	10.0	8	455.0	225.0	14.0	pontiac catalina	USA	4425	1970-01-01
9	8.5	8	390.0	190.0	15.0	amc ambassador dpl	USA	3850	1970-01-01

After:

	Acceleration	Cylinders	Displacement	Horsepower	Miles_per_Gallon	Name	Origin	Weight_in_lbs	Year
0	12.0	8	307.0	130.0	18.0	chevrolet chevelle malibu	USA	3504	1970-01-01
1	11.5	8	350.0	165.0	15.0	buick skylark 320	USA	3693	1970-01-01
2	11.0	8	318.0	150.0	18.0	plymouth satellite	USA	3436	1970-01-01
3	12.0	8	304.0	150.0	16.0	amc rebel sst	USA	3433	1970-01-01
4	10.5	8	302.0	140.0	17.0	ford torino	USA	3449	1970-01-01
5	10.0	8	429.0	198.0	15.0	ford galaxie 500	USA	4341	1970-01-01
6	9.0	8	454.0	220.0	14.0	chevrolet impala	USA	4354	1970-01-01
7	8.5	8	440.0	215.0	14.0	plymouth fury iii	USA	4312	1970-01-01
8	10.0	8	455.0	225.0	14.0	pontiac catalina	USA	4425	1970-01-01
9	8.5	8	390.0	190.0	15.0	amc ambassador dpl	USA	3850	1970-01-01

5.41.4 Customise keyboard shortcuts

You can now edit keyboard shortcuts for *Command Mode* within the UI (1347):



See the `Help > Edit Keyboard Shortcuts` menu item and follow the instructions.

5.41.5 Other additions

- You can copy and paste cells between notebooks, using `Ctrl-C`{.interpreted-text role="kbd"} and `Ctrl-V`{.interpreted-text role="kbd"} (`Cmd-C`{.interpreted-text role="kbd"} and `Cmd-V`{.interpreted-text role="kbd"} on Mac).
- It's easier to configure a password for the notebook with the new `jupyter notebook password` command (2007).
- The file list can now be ordered by *last modified* or by *name* (943).
- Markdown cells now support attachments. Simply drag and drop an image from your desktop to a markdown cell to add it. Unlike relative links that you enter manually, attachments are embedded in the notebook itself. An unreferenced attachment will be automatically scrubbed from the notebook on save (621).
- Undoing cell deletion now supports undeleting multiple cells. Cells may not be in the same order as before their deletion, depending on the actions you did on the meantime, but this should should help reduce the impact of accidentally deleting code.
- The file browser now has *Edit* and *View* buttons.
- The file browser now supports moving multiple files at once (1088).
- The Notebook will refuse to run as root unless the `--allow-root` flag is given (1115).
- Keyboard shortcuts are now declarative (1234).
- Toggling line numbers can now affect all cells (1312).

- Add more visible *Trusted* and *Untrusted* notifications (1658).
- The favicon (browser shortcut icon) now changes to indicate when the kernel is busy (1837).
- Header and toolbar visibility is now persisted in nbconfig and across sessions (1769).
- Load server extensions with ConfigManager so that merge happens recursively, unlike normal config values, to make it load more consistently with frontend extensions(2108).
- The notebook server now supports the bundler API from the jupyter_cms incubator project (1579).
- The notebook server now provides information about kernel activity in its kernel resource API (1827).

Remember that upgrading `notebook` only affects the user interface. Upgrading kernels and libraries may also provide new features, better stability and integration with the notebook interface.

5.42 4.4.0

- Allow override of output callbacks to redirect output messages. This is used to implement the ipywidgets Output widget, for example.
- Fix an async bug in message handling by allowing comm message handlers to return a promise which halts message processing until the promise resolves.

See the 4.4 milestone on GitHub for a complete list of issues and pull requests involved in this release.

5.43 4.3.2

4.3.2 is a patch release with a bug fix for CodeMirror and improved handling of the "editable" cell metadata field.

- Monkey-patch for CodeMirror that resolves #2037 without breaking #1967
- Read-only (`"editable": false`) cells can be executed but cannot be split, merged, or deleted

See the 4.3.2 milestone on GitHub for a complete list of issues and pull requests involved in this release.

5.44 4.3.1

4.3.1 is a patch release with a security patch, a couple bug fixes, and improvements to the newly-released token authentication.

Security fix:

- CVE-2016-9971. Fix CSRF vulnerability, where malicious forms could create untitled files and start kernels (no remote execution or modification of existing files) for users of certain browsers (Firefox, Internet Explorer / Edge). All previous notebook releases are affected.

Bug fixes:

- Fix carriage return handling
- Make the font size more robust against fickle browsers
- Ignore resize events that bubbled up and didn't come from window
- Add Authorization to allowed CORS headers
- Downgrade CodeMirror to 5.16 while we figure out issues in Safari

Other improvements:

- Better docs for token-based authentication
- Further highlight token info in log output when autogenerated

See the 4.3.1 milestone on GitHub for a complete list of issues and pull requests involved in this release.

5.45 4.3.0

4.3 is a minor release with many bug fixes and improvements. The biggest user-facing change is the addition of token authentication, which is enabled by default. A token is generated and used when your browser is opened automatically, so you shouldn't have to enter anything in the default circumstances. If you see a login page (e.g. by switching browsers, or launching on a new port with `--no-browser`), you get a login URL with the token from the command `jupyter notebook list`, which you can paste into your browser.

Highlights:

- API for creating mime-type based renderer extensions using `OutputArea.register_mime_type` and `Notebook.render_cell_output` methods. See mimerender-cookiecutter for reference implementations and cookiecutter.
- Enable token authentication by default. See `server_security`{.interpreted-text role="ref"} for more details.
- Update security docs to reflect new signature system
- Switched from term.js to xterm.js

Bug fixes:

- Ensure variable is set if exc_info is falsey
- Catch and log handler exceptions in `events.trigger`
- Add debug log for static file paths
- Don't check origin on token-authenticated requests
- Remove leftover print statement
- Fix highlighting of Python code blocks
- `json_errors` should be outermost decorator on API handlers
- Fix remove old nbserver info files
- Fix notebook mime type on download links
- Fix carriage symbol behavior
- Fix terminal styles
- Update dead links in docs
- If kernel is broken, start a new session
- Include cross-origin check when allowing login URL redirects

Other improvements:

- Allow JSON output data with mime type `application/*+json`
- Allow kernelspecs to have spaces in them for backward compat
- Allow websocket connections from scripts

- Allow `None` for post_save_hook
- Upgrade CodeMirror to 5.21
- Upgrade xterm to 2.1.0
- Docs for using comms
- Set `dirty` flag when output arrives
- Set `ws-url` data attribute when accessing a notebook terminal
- Add base aliases for nbextensions
- Include `@` operator in CodeMirror IPython mode
- Extend mathjax_url docstring
- Load nbextension in predictable order
- Improve the error messages for nbextensions
- Include cross-origin check when allowing login URL redirects

See the 4.3 milestone on GitHub for a complete list of issues and pull requests involved in this release.

5.46 4.2.3

4.2.3 is a small bugfix release on 4.2.

Highlights:

- Fix regression in 4.2.2 that delayed loading custom.js until after `notebook_loaded` and `app_initialized` events have fired.
- Fix some outdated docs and links.

5.47 4.2.2

4.2.2 is a small bugfix release on 4.2, with an important security fix. All users are strongly encouraged to upgrade to 4.2.2.

Highlights:

- **Security fix**: CVE-2016-6524, where untrusted latex output could be added to the page in a way that could execute javascript.
- Fix missing POST in OPTIONS responses.
- Fix for downloading non-ascii filenames.
- Avoid clobbering ssl_options, so that users can specify more detailed SSL configuration.
- Fix inverted load order in nbconfig, so user config has highest priority.
- Improved error messages here and there.

5.48 4.2.1

4.2.1 is a small bugfix release on 4.2. Highlights:

- Compatibility fixes for some versions of ipywidgets
- Fix for ignored CSS on Windows
- Fix specifying destination when installing nbextensions

5.49 4.2.0

Release 4.2 adds a new API for enabling and installing extensions. Extensions can now be enabled at the system-level, rather than just per-user. An API is defined for installing directly from a Python package, as well.

Highlighted changes:

- Upgrade MathJax to 2.6 to fix vertical-bar appearing on some equations.
- Restore ability for notebook directory to be root (4.1 regression)
- Large outputs are now throttled, reducing the ability of output floods to kill the browser.
- Fix the notebook ignoring cell executions while a kernel is starting by queueing the messages.
- Fix handling of url prefixes (e.g. JupyterHub) in terminal and edit pages.
- Support nested SVGs in output.

And various other fixes and improvements.

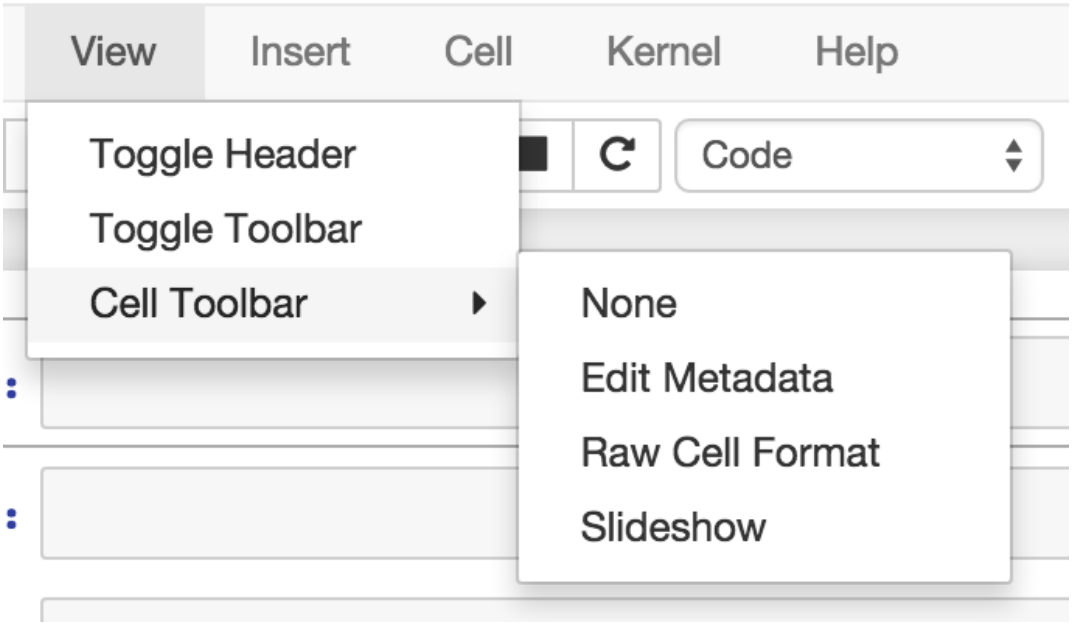
5.50 4.1.0

Bug fixes:

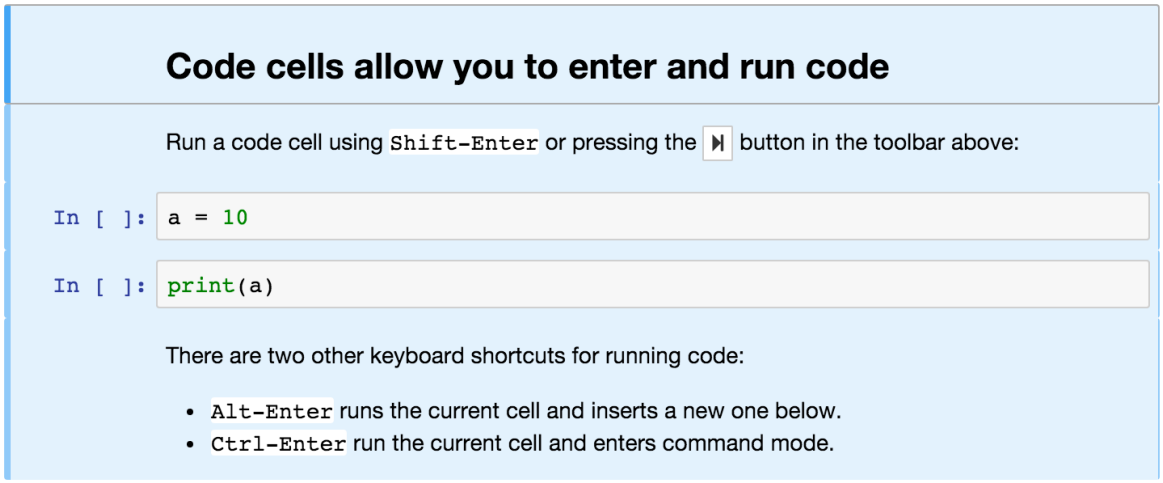
- Properly reap zombie subprocesses
- Fix cross-origin problems
- Fix double-escaping of the base URL prefix
- Handle invalid unicode filenames more gracefully
- Fix ANSI color-processing
- Send keepalive messages for web terminals
- Fix bugs in the notebook tour

UI changes:

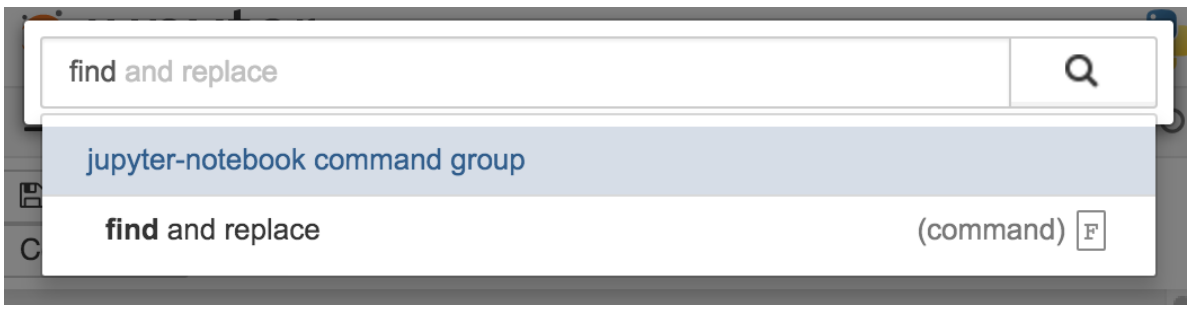
- Moved the cell toolbar selector into the *View* menu. Added a button that triggers a "hint" animation to the main toolbar so users can find the new location. (Click here to see a screencast)



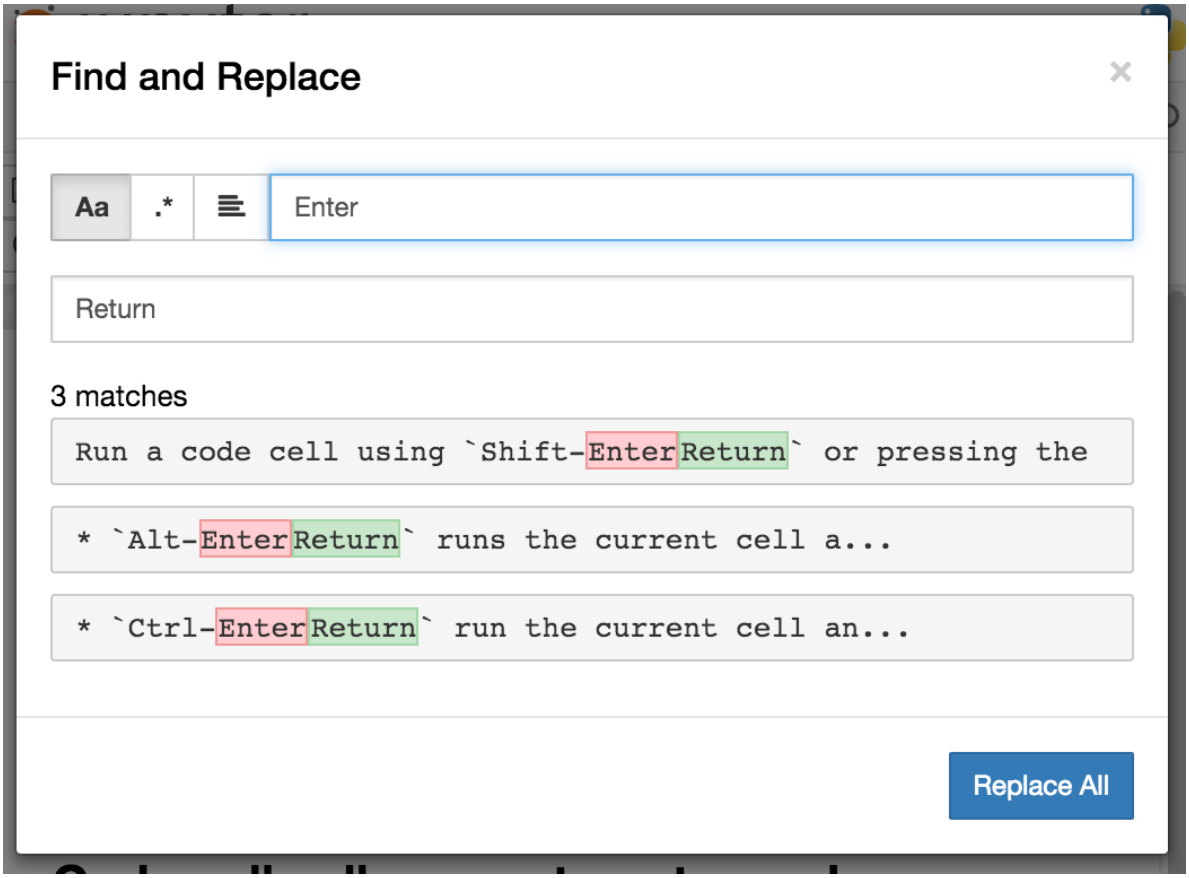
- Added *Restart & Run All* to the *Kernel* menu. Users can also bind it to a keyboard shortcut on action `restart-kernel-and-run-all-cells`.
- Added multiple-cell selection. Users press `Shift-Up/Down` or `Shift-K/J` to extend selection in command mode. Various actions such as cut/copy/paste, execute, and cell type conversions apply to all selected cells.



- Added a command palette for executing Jupyter actions by name. Users press `Cmd/Ctrl-Shift-P` or click the new command palette icon on the toolbar.



- Added a *Find and Replace* dialog to the *Edit* menu. Users can also press `F` in command mode to show the dialog.



Other improvements:

- Custom KernelManager methods can be Tornado coroutines, allowing async operations.
- Make clearing output optional when rewriting input with `set_next_input(replace=True)`.
- Added support for TLS client authentication via `--NotebookApp.client-ca`.
- Added tags to `jupyter/notebook` releases on DockerHub. `latest` continues to track the master branch.

See the 4.1 milestone on GitHub for a complete list of issues and pull requests handled.

5.51 4.0.x

5.51.1 4.0.6

- fix installation of mathjax support files
- fix some double-escape regressions in 4.0.5
- fix a couple of cases where errors could prevent opening a notebook

5.51.2 4.0.5

Security fixes for maliciously crafted files.

- CVE-2015-6938: malicious filenames
- CVE-2015-7337: malicious binary files in text editor.

Thanks to Jonathan Kamens at Quantopian and Juan Broullón for the reports.

5.51.3 4.0.4

- Fix inclusion of mathjax-safe extension

5.51.4 4.0.2

- Fix launching the notebook on Windows
- Fix the path searched for frontend config

5.51.5 4.0.0

First release of the notebook as a standalone package.

CHAPTER

SIX

COMMS

Comms allow custom messages between the frontend and the kernel. They are used, for instance, in ipywidgets to update widget state.

A comm consists of a pair of objects, in the kernel and the frontend, with an automatically assigned unique ID. When one side sends a message, a callback on the other side is triggered with that message data. Either side, the frontend or kernel, can open or close the comm.

See also:

Custom Messages The messaging specification section on comms

6.1 Opening a comm from the kernel

First, the function to accept the comm must be available on the frontend. This can either be specified in a *requirejs* module, or registered in a registry, for example when an *extension* is loaded. This example shows a frontend comm target registered in a registry:

```
Jupyter.notebook.kernel.comm_manager.register_target('my_comm_target',
    function(comm, msg) {
        // comm is the frontend comm instance
        // msg is the comm_open message, which can carry data

        // Register handlers for later messages:
        comm.on_msg(function(msg) {...});
        comm.on_close(function(msg) {...});
        comm.send({'foo': 0});
    });
```

Now that the frontend comm is registered, you can open the comm from the kernel:

```
from ipykernel.comm import Comm

# Use comm to send a message from the kernel
my_comm = Comm(target_name='my_comm_target', data={'foo': 1})
my_comm.send({'foo': 2})

# Add a callback for received messages.
@my_comm.on_msg
def _recv(msg):
    # Use msg['content']['data'] for the data in the message
```

This example uses the IPython kernel; it's up to each language kernel what API, if any, it offers for using comms.

6.2 Opening a comm from the frontend

This is very similar to above, but in reverse. First, a comm target must be registered in the kernel. For instance, this may be done by code displaying output: it will register a target in the kernel, and then display output containing Javascript to connect to it.

```
def target_func(comm, open_msg):
    # comm is the kernel Comm instance
    # msg is the comm_open message

    # Register handler for later messages
    @comm.on_msg
    def _recv(msg):
        # Use msg['content']['data'] for the data in the message
        comm.send({'echo': msg['content']['data']})

    # Send data to the frontend on creation
    comm.send({'foo': 5})

get_ipython().kernel.comm_manager.register_target('my_comm_target', target_func)
```

This example uses the IPython kernel again; this example will be different in other kernels that support comms. Refer to the specific language kernel's documentation for comms support.

And then open the comm from the frontend:

```
const comm = Jupyter.notebook.kernel.comm_manager.new_comm('my_comm_target', {'foo': 6})
// Send data
comm.send({'foo': 7})

// Register a handler
comm.on_msg(function(msg) {
    console.log(msg.content.data.foo);
});
```

CHAPTER

SEVEN

CONFIGURATION OVERVIEW

Beyond the default configuration settings, you can configure a rich array of options to suit your workflow. Here are areas that are commonly configured when using Jupyter Notebook:

- *Jupyter's common configuration system*
- *Notebook server*
- *Notebook front-end client*
- *Notebook extensions*

Let's look at highlights of each area.

7.1 Jupyter's Common Configuration system

Jupyter applications, from the Notebook to JupyterHub to nbgrader, share a common configuration system. The process for creating a configuration file and editing settings is similar for all the Jupyter applications.

- Jupyter's Common Configuration Approach
- Common Directories and File Locations
- Language kernels
- traitlets provide a low-level architecture for configuration.

7.2 Notebook server

The Notebook server runs the language kernel and communicates with the front-end Notebook client (i.e. the familiar notebook interface).

- Configuring the Notebook server

  To create a `jupyter_notebook_config.py` file in the `.jupyter` directory, with all the defaults commented out, use the following command:

  ```
      $ jupyter notebook --generate-config

  :ref:`Command line arguments for configuration <config>` settings are
  documented in the configuration file and the user documentation.
  ```

- *Running a Notebook server*

- Related: Configuring a language kernel to run in the Notebook server enables your server to run other languages, like R or Julia.

7.3 Notebook front-end client

7.3.1 Configuring the notebook frontend

Note: The ability to configure the notebook frontend UI and preferences is still a work in progress.

This document is a rough explanation on how you can persist some configuration options for the notebook JavaScript.

There is no exhaustive list of all the configuration options as most options are passed down to other libraries, which means that non valid configuration can be ignored without any error messages.

How front end configuration works

The frontend configuration system works as follows:

- get a handle of a configurable JavaScript object.
- access its configuration attribute.
- update its configuration attribute with a JSON patch.

Example - Changing the notebook's default indentation

This example explains how to change the default setting `indentUnit` for CodeMirror Code Cells:

```
var cell = Jupyter.notebook.get_selected_cell();
var config = cell.config;
var patch = {
      CodeCell:{
        cm_config:{indentUnit:2}
      }
    }
config.update(patch)
```

You can enter the previous snippet in your browser's JavaScript console once. Then reload the notebook page in your browser. Now, the preferred indent unit should be equal to two spaces. The custom setting persists and you do not need to reissue the patch on new notebooks.

`indentUnit`, used in this example, is one of the many CodeMirror options which are available for configuration.

You can similarly change the options of the file editor by entering the following snippet in the browser's Javascript console once (from a file editing page).:

```
var config = Jupyter.editor.config
var patch = {
      Editor: {
        codemirror_options: {
          indentUnit: 2
        }
```

(continues on next page)

(continued from previous page)

```
      }
    }
config.update(patch)
```

Example - Restoring the notebook's default indentation

If you want to restore a notebook frontend preference to its default value, you will enter a JSON patch with a `null` value for the preference setting.

For example, let's restore the indent setting `indentUnit` to its default of four spaces. Enter the following code snippet in your JavaScript console:

```
var cell = Jupyter.notebook.get_selected_cell();
var config = cell.config;
var patch = {
      CodeCell:{
        cm_config:{indentUnit: null} // only change here.
      }
    }
config.update(patch)
```

Reload the notebook in your browser and the default indent should again be two spaces.

Persisting configuration settings

Under the hood, Jupyter will persist the preferred configuration settings in `~/.jupyter/nbconfig/<section>.json`, with `<section>` taking various value depending on the page where the configuration is issued. `<section>` can take various values like `notebook`, `tree`, and `editor`. A `common` section contains configuration settings shared by all pages.

7.4 Notebook extensions

- Distributing Jupyter Extensions as Python Packages
- Extending the Notebook

Security in Jupyter notebooks: Since security policies vary from organization to organization, we encourage you to consult with your security team on settings that would be best for your use cases. Our documentation offers some responsible security practices, and we recommend becoming familiar with the practices.

CHAPTER

EIGHT

CONFIG FILE AND COMMAND LINE OPTIONS

The notebook server can be run with a variety of command line arguments. A list of available options can be found below in the *options section*.

Defaults for these options can also be set by creating a file named `jupyter_notebook_config.py` in your Jupyter folder. The Jupyter folder is in your home directory, `~/.jupyter`.

To create a `jupyter_notebook_config.py` file, with all the defaults commented out, you can use the following command line:

```
$ jupyter notebook --generate-config
```

8.1 Options

This list of options can be generated by running the following and hitting enter:

```
$ jupyter notebook --help
```

Application.log_datefmt [Unicode] Default: `'%Y-%m-%d %H:%M:%S'`

The date format used by logging formatters for %(asctime)s

Application.log_format [Unicode] Default: `'[%(name)s]%(highlevel)s %(message)s'`

The Logging format template

Application.log_level [any of `0``|``10``|``20``|``30``|``40``|``50``|'DEBUG'|'INFO'|'WARN'|'ERROR'|'CRITICAI`
Default: `30`

Set the log level by value or name.

Application.show_config [Bool] Default: `False`

Instead of starting the Application, dump configuration to stdout

Application.show_config_json [Bool] Default: `False`

Instead of starting the Application, dump configuration to stdout (as JSON)

JupyterApp.answer_yes [Bool] Default: `False`

Answer yes to any prompts.

JupyterApp.config_file [Unicode] Default: `''`

Full path of a config file.

JupyterApp.config_file_name [Unicode] Default: `''`

Specify a config file to load.

JupyterApp.generate_config [Bool] Default: `False`

Generate default config file.

JupyterApp.log_datefmt [Unicode] Default: `'%Y-%m-%d %H:%M:%S'`

The date format used by logging formatters for %(asctime)s

JupyterApp.log_format [Unicode] Default: `'[%(name)s]%(highlevel)s %(message)s'`

The Logging format template

JupyterApp.log_level [any of `0``|``10``|``20``|``30``|``40``|``50`|'DEBUG'|'INFO'|'WARN'|'ERROR'|'CRITICA
Default: `30`

Set the log level by value or name.

JupyterApp.show_config [Bool] Default: `False`

Instead of starting the Application, dump configuration to stdout

JupyterApp.show_config_json [Bool] Default: `False`

Instead of starting the Application, dump configuration to stdout (as JSON)

NotebookApp.allow_credentials [Bool] Default: `False`

Set the Access-Control-Allow-Credentials: true header

NotebookApp.allow_origin [Unicode] Default: `''`

Set the Access-Control-Allow-Origin header

Use '*' to allow any origin to access your server.

Takes precedence over allow_origin_pat.

NotebookApp.allow_origin_pat [Unicode] Default: `''`

Use a regular expression for the Access-Control-Allow-Origin header

Requests from an origin matching the expression will get replies with:

Access-Control-Allow-Origin: origin

where *origin* is the origin of the request.

Ignored if allow_origin is set.

NotebookApp.allow_password_change [Bool] Default: `True`

Allow password to be changed at login for the notebook server.

While logging in with a token, the notebook server UI will give the opportunity to the user to enter a new password at the same time that will replace the token login mechanism.

This can be set to false to prevent changing password from the UI/API.

NotebookApp.allow_remote_access [Bool] Default: `False`

Allow requests where the Host header doesn't point to a local server

By default, requests get a 403 forbidden response if the 'Host' header shows that the browser thinks it's on a non-local domain. Setting this option to True disables this check.

This protects against 'DNS rebinding' attacks, where a remote web server serves you a page and then changes its DNS to send later requests to a local IP, bypassing same-origin checks.

Local IP addresses (such as 127.0.0.1 and ::1) are allowed as local, along with hostnames configured in local_hostnames.

NotebookApp.allow_root [Bool] Default: `False`

Whether to allow the user to run the notebook as root.

NotebookApp.answer_yes [Bool] Default: `False`

Answer yes to any prompts.

NotebookApp.authenticate_prometheus [Bool] Default: `True`

" Require authentication to access prometheus metrics.

NotebookApp.autoreload [Bool] Default: `False`

Reload the webapp when changes are made to any Python src files.

NotebookApp.base_project_url [Unicode] Default: `'/'`

DEPRECATED use base_url

NotebookApp.base_url [Unicode] Default: `'/'`

The base URL for the notebook server.

Leading and trailing slashes can be omitted, and will automatically be added.

NotebookApp.browser [Unicode] Default: `''`

Specify what command to use to invoke a web browser when opening the notebook. If not specified, the default browser will be determined by the *webbrowser* standard library module, which allows setting of the BROWSER environment variable to override it.

NotebookApp.certfile [Unicode] Default: `''`

The full path to an SSL/TLS certificate file.

NotebookApp.client_ca [Unicode] Default: `''`

The full path to a certificate authority certificate for SSL/TLS client authentication.

NotebookApp.config_file [Unicode] Default: `''`

Full path of a config file.

NotebookApp.config_file_name [Unicode] Default: `''`

Specify a config file to load.

NotebookApp.config_manager_class [Type] Default: `'notebook.services.config.manager.ConfigManager'`

The config manager class to use

NotebookApp.contents_manager_class [TypeFromClasses] Default: `'notebook.services.contents.largefilemanager.LargeFileManager'`

The notebook manager class to use.

NotebookApp.cookie_options [Dict] Default: `{}`

Extra keyword arguments to pass to *set_secure_cookie*. See tornado's set_secure_cookie docs for details.

NotebookApp.cookie_secret [Bytes] Default: `b''`

The random bytes used to secure cookies. By default this is a new random number every time you start the Notebook. Set it to a value in a config file to enable logins to persist across server sessions.

Note: Cookie secrets should be kept private, do not share config files with cookie_secret stored in plaintext (you can read the value from a file).

NotebookApp.cookie_secret_file [Unicode] Default: `''`

The file where the cookie secret is stored.

NotebookApp.custom_display_url [Unicode] Default: `''`

Override URL shown to users.

Replace actual URL, including protocol, address, port and base URL, with the given value when displaying URL to the users. Do not change the actual connection URL. If authentication token is enabled, the token is added to the custom URL automatically.

This option is intended to be used when the URL to display to the user cannot be determined reliably by the Jupyter notebook server (proxified or containerized setups for example).

NotebookApp.default_url [Unicode] Default: `'/tree'`

The default URL to redirect to from `/`

NotebookApp.disable_check_xsrf [Bool] Default: `False`

Disable cross-site-request-forgery protection

Jupyter notebook 4.3.1 introduces protection from cross-site request forgeries, requiring API requests to either:

- originate from pages served by this server (validated with XSRF cookie and token), or
- authenticate with a token

Some anonymous compute resources still desire the ability to run code, completely without authentication. These services can disable all authentication and security checks, with the full knowledge of what that implies.

NotebookApp.enable_mathjax [Bool] Default: `True`

Whether to enable MathJax for typesetting math/TeX

MathJax is the javascript library Jupyter uses to render math/LaTeX. It is very large, so you may want to disable it if you have a slow internet connection, or for offline use of the notebook.

When disabled, equations etc. will appear as their untransformed TeX source.

NotebookApp.extra_nbextensions_path [List] Default: `[]`

extra paths to look for Javascript notebook extensions

NotebookApp.extra_services [List] Default: `[]`

handlers that should be loaded at higher priority than the default services

NotebookApp.extra_static_paths [List] Default: `[]`

Extra paths to search for serving static files.

This allows adding javascript/css to be available from the notebook server machine, or overriding individual files in the IPython

NotebookApp.extra_template_paths [List] Default: `[]`

Extra paths to search for serving jinja templates.

Can be used to override templates from notebook.templates.

NotebookApp.file_to_run [Unicode] Default: `''`

No description

NotebookApp.generate_config [Bool] Default: `False`

Generate default config file.

NotebookApp.get_secure_cookie_kwargs [Dict] Default: `{}`

Extra keyword arguments to pass to *get_secure_cookie*. See tornado's get_secure_cookie docs for details.

NotebookApp.ignore_minified_js [Bool] Default: `False`

Deprecated: Use minified JS file or not, mainly use during dev to avoid JS recompilation

NotebookApp.iopub_data_rate_limit [Float] Default: `1000000`

(bytes/sec) Maximum rate at which stream output can be sent on iopub before they are limited.

NotebookApp.iopub_msg_rate_limit [Float] Default: `1000`

(msgs/sec) Maximum rate at which messages can be sent on iopub before they are limited.

NotebookApp.ip [Unicode] Default: `'localhost'`

The IP address the notebook server will listen on.

NotebookApp.jinja_environment_options [Dict] Default: `{}`

Supply extra arguments that will be passed to Jinja environment.

NotebookApp.jinja_template_vars [Dict] Default: `{}`

Extra variables to supply to jinja templates when rendering.

NotebookApp.kernel_manager_class [Type] Default: `'notebook.services.kernels.kernelmanager.MappingKernelManager'`

The kernel manager class to use.

NotebookApp.kernel_spec_manager_class [Type] Default: `'jupyter_client.kernelspec.KernelSpecManager'`

The kernel spec manager class to use. Should be a subclass of *jupyter_client.kernelspec.KernelSpecManager*.

The Api of KernelSpecManager is provisional and might change without warning between this version of Jupyter and the next stable one.

NotebookApp.keyfile [Unicode] Default: `''`

The full path to a private key file for usage with SSL/TLS.

NotebookApp.local_hostnames [List] Default: `['localhost']`

Hostnames to allow as local when allow_remote_access is False.

Local IP addresses (such as 127.0.0.1 and ::1) are automatically accepted as local as well.

NotebookApp.log_datefmt [Unicode] Default: `'%Y-%m-%d %H:%M:%S'`

The date format used by logging formatters for %(asctime)s

NotebookApp.log_format [Unicode] Default: `'[%(name)s]%(highlevel)s %(message)s'`

The Logging format template

NotebookApp.log_json [Bool] Default: `False`

Set to True to enable JSON formatted logs. Run "pip install notebook[json-logging]" to install the required dependent packages. Can also be set using the environment variable JUPYTER_ENABLE_JSON_LOGGING=true.

NotebookApp.log_level [any of `0``|``10``|``20``|``30``|``40``|``50``|'DEBUG'|'INFO'|'WARN'|'ERROR'|'CRITIC`
Default: `30`

Set the log level by value or name.

NotebookApp.login_handler_class [Type] Default: `'notebook.auth.login.LoginHandler'`

The login handler class to use.

NotebookApp.logout_handler_class [Type] Default: `'notebook.auth.logout.LogoutHandler'`

The logout handler class to use.

NotebookApp.mathjax_config [Unicode] Default: `'TeX-AMS-MML_HTMLorMML-full,Safe'`

The MathJax.js configuration file that is to be used.

NotebookApp.mathjax_url [Unicode] Default: `''`

A custom url for MathJax.js. Should be in the form of a case-sensitive url to MathJax, for example: /static/components/MathJax/MathJax.js

NotebookApp.max_body_size [Int] Default: `536870912`

Sets the maximum allowed size of the client request body, specified in the Content-Length request header field. If the size in a request exceeds the configured value, a malformed HTTP message is returned to the client.

Note: max_body_size is applied even in streaming mode.

NotebookApp.max_buffer_size [Int] Default: `536870912`

Gets or sets the maximum amount of memory, in bytes, that is allocated for use by the buffer manager.

NotebookApp.min_open_files_limit [Int] Default: `0`

Gets or sets a lower bound on the open file handles process resource limit. This may need to be increased if you run into an OSError: [Errno 24] Too many open files. This is not applicable when running on Windows.

NotebookApp.nbserver_extensions [Dict] Default: `{}`

Dict of Python modules to load as notebook server extensions. Entry values can be used to enable and disable the loading of the extensions. The extensions will be loaded in alphabetical order.

NotebookApp.notebook_dir [Unicode] Default: `''`

The directory to use for notebooks and kernels.

NotebookApp.open_browser [Bool] Default: `True`

Whether to open in a browser after starting. The specific browser used is platform dependent and determined by the python standard library *webbrowser* module, unless it is overridden using the –browser (NotebookApp.browser) configuration option.

NotebookApp.password [Unicode] Default: `''`

Hashed password to use for web authentication.

To generate, type in a python/IPython shell:

```
from notebook.auth import passwd; passwd()
```

The string should be of the form type:salt:hashed-password.

NotebookApp.password_required [Bool] Default: `False`

Forces users to use a password for the Notebook server. This is useful in a multi user environment, for instance when everybody in the LAN can access each other's machine through ssh.

In such a case, serving the notebook server on localhost is not secure since any user can connect to the notebook server via ssh.

NotebookApp.port [Int] Default: `8888`

The port the notebook server will listen on (env: JUPYTER_PORT).

NotebookApp.port_retries [Int] Default: `50`

The number of additional ports to try if the specified port is not available (env: JUPYTER_PORT_RETRIES).

NotebookApp.pylab [Unicode] Default: `'disabled'`

DISABLED: use %pylab or %matplotlib in the notebook to enable matplotlib.

NotebookApp.quit_button [Bool] Default: `True`

If True, display a button in the dashboard to quit (shutdown the notebook server).

NotebookApp.rate_limit_window [Float] Default: `3`

(sec) Time window used to check the message and data rate limits.

NotebookApp.reraise_server_extension_failures [Bool] Default: `False`

Reraise exceptions encountered loading server extensions?

NotebookApp.server_extensions [List] Default: `[]`

DEPRECATED use the nbserver_extensions dict instead

NotebookApp.session_manager_class [Type] Default: `'notebook.services.sessions.sessionmanager.SessionManager'`

The session manager class to use.

NotebookApp.show_config [Bool] Default: `False`

Instead of starting the Application, dump configuration to stdout

NotebookApp.show_config_json [Bool] Default: `False`

Instead of starting the Application, dump configuration to stdout (as JSON)

NotebookApp.shutdown_no_activity_timeout [Int] Default: `0`

Shut down the server after N seconds with no kernels or terminals running and no activity. This can be used together with culling idle kernels (MappingKernelManager.cull_idle_timeout) to shutdown the notebook server when it's not in use. This is not precisely timed: it may shut down up to a minute later. 0 (the default) disables this automatic shutdown.

NotebookApp.sock [Unicode] Default: `''`

The UNIX socket the notebook server will listen on.

NotebookApp.sock_mode [Unicode] Default: `'0600'`

The permissions mode for UNIX socket creation (default: 0600).

NotebookApp.ssl_options [Dict] Default: `{}`

Supply SSL options for the tornado HTTPServer. See the tornado docs for details.

NotebookApp.terminado_settings [Dict] Default: `{}`

Supply overrides for terminado. Currently only supports "shell_command". On Unix, if "shell_command" is not provided, a non-login shell is launched by default when the notebook server is connected to a terminal, a login shell otherwise.

NotebookApp.terminals_enabled [Bool] Default: `True`

Set to False to disable terminals.

This does *not* make the notebook server more secure by itself. Anything the user can in a terminal, they can also do in a notebook.

Terminals may also be automatically disabled if the terminado package is not available.

NotebookApp.token [Unicode] Default: `'<generated>'`

Token used for authenticating first-time connections to the server.

The token can be read from the file referenced by JUPYTER_TOKEN_FILE or set directly with the JUPYTER_TOKEN environment variable.

When no password is enabled, the default is to generate a new, random token.

Setting to an empty string disables authentication altogether, which is NOT RECOMMENDED.

NotebookApp.tornado_settings [Dict] Default: `{}`

Supply overrides for the tornado.web.Application that the Jupyter notebook uses.

NotebookApp.trust_xheaders [Bool] Default: `False`

Whether to trust or not X-Scheme/X-Forwarded-Proto and X-Real-Ip/X-Forwarded-For headers sent by the upstream reverse proxy. Necessary if the proxy handles SSL

NotebookApp.use_redirect_file [Bool] Default: `True`

Disable launching browser by redirect file

For versions of notebook > 5.7.2, a security feature measure was added that prevented the authentication token used to launch the browser from being visible. This feature makes it difficult for other users on a multi-user system from running code in your Jupyter session as you.

However, some environments (like Windows Subsystem for Linux (WSL) and Chromebooks), launching a browser using a redirect file can lead the browser failing to load. This is because of the difference in file structures/paths between the runtime and the browser.

Disabling this setting to False will disable this behavior, allowing the browser to launch by using a URL and visible token (as before).

NotebookApp.webapp_settings [Dict] Default: `{}`

DEPRECATED, use tornado_settings

NotebookApp.webbrowser_open_new [Int] Default: `2`

Specify Where to open the notebook on startup. This is the *new* argument passed to the standard library method *webbrowser.open*. The behaviour is not guaranteed, but depends on browser support. Valid values are:

- 2 opens a new tab,
- 1 opens a new window,
- 0 opens in an existing window.

See the *webbrowser.open* documentation for details.

NotebookApp.websocket_compression_options [Any] Default: `None`

Set the tornado compression options for websocket connections.

This value will be returned from `WebSocketHandler.get_compression_options()`. None (default) will disable compression. A dict (even an empty one) will enable compression.

See the tornado docs for WebSocketHandler.get_compression_options for details.

NotebookApp.websocket_url [Unicode] Default: `''`

The base URL for websockets, if it differs from the HTTP server (hint: it almost certainly doesn't).

Should be in the form of an HTTP origin: ws[s]://hostname[:port]

ConnectionFileMixin.connection_file [Unicode] Default: `''`

JSON file in which to store connection info [default: kernel-<pid>.json]

This file will contain the IP, ports, and authentication key needed to connect clients to this kernel. By default, this file will be created in the security dir of the current profile, but can be specified by absolute path.

ConnectionFileMixin.control_port [Int] Default: `0`

set the control (ROUTER) port [default: random]

ConnectionFileMixin.hb_port [Int] Default: `0`

set the heartbeat port [default: random]

ConnectionFileMixin.iopub_port [Int] Default: `0`

set the iopub (PUB) port [default: random]

ConnectionFileMixin.ip [Unicode] Default: `''`

Set the kernel's IP address [default localhost]. If the IP address is something other than localhost, then Consoles on other machines will be able to connect to the Kernel, so be careful!

ConnectionFileMixin.shell_port [Int] Default: `0`

set the shell (ROUTER) port [default: random]

ConnectionFileMixin.stdin_port [Int] Default: `0`

set the stdin (ROUTER) port [default: random]

ConnectionFileMixin.transport [any of `'tcp'``|'ipc'``` (case-insensitive)] Default: `'tcp'`

No description

KernelManager.autorestart [Bool] Default: `True`

Should we autorestart the kernel if it dies.

KernelManager.connection_file [Unicode] Default: `''`

JSON file in which to store connection info [default: kernel-<pid>.json]

This file will contain the IP, ports, and authentication key needed to connect clients to this kernel. By default, this file will be created in the security dir of the current profile, but can be specified by absolute path.

KernelManager.control_port [Int] Default: `0`

set the control (ROUTER) port [default: random]

KernelManager.hb_port [Int] Default: `0`

set the heartbeat port [default: random]

KernelManager.iopub_port [Int] Default: `0`

set the iopub (PUB) port [default: random]

KernelManager.ip [Unicode] Default: `''`

Set the kernel's IP address [default localhost]. If the IP address is something other than localhost, then Consoles on other machines will be able to connect to the Kernel, so be careful!

KernelManager.shell_port [Int] Default: `0`

set the shell (ROUTER) port [default: random]

KernelManager.shutdown_wait_time [Float] Default: `5.0`

Time to wait for a kernel to terminate before killing it, in seconds. When a shutdown request is initiated, the kernel will be immediately sent an interrupt (SIGINT), followedby a shutdown_request message, after 1/2 of *shutdown_wait_time`it will be sent a terminate (SIGTERM) request, and finally at the end of `shutdown_wait_time* will be killed (SIGKILL). terminate and kill may be equivalent on windows. Note that this value can beoverridden by the in-use kernel provisioner since shutdown times mayvary by provisioned environment.

KernelManager.stdin_port [Int] Default: `0`

set the stdin (ROUTER) port [default: random]

KernelManager.transport [any of `'tcp'``|'ipc'``` (case-insensitive)] Default: `'tcp'`

No description

Session.buffer_threshold [Int] Default: `1024`

Threshold (in bytes) beyond which an object's buffer should be extracted to avoid pickling.

Session.check_pid [Bool] Default: `True`

Whether to check PID to protect against calls after fork.

This check can be disabled if fork-safety is handled elsewhere.

Session.copy_threshold [Int] Default: `65536`

Threshold (in bytes) beyond which a buffer should be sent without copying.

Session.debug [Bool] Default: `False`

Debug output in the Session

Session.digest_history_size [Int] Default: `65536`

The maximum number of digests to remember.

The digest history will be culled when it exceeds this value.

Session.item_threshold [Int] Default: `64`

The maximum number of items for a container to be introspected for custom serialization. Containers larger than this are pickled outright.

Session.key [CBytes] Default: `b''`

execution key, for signing messages.

Session.keyfile [Unicode] Default: `''`

path to file containing execution key.

Session.metadata [Dict] Default: `{}`

Metadata dictionary, which serves as the default top-level metadata dict for each message.

Session.packer [DottedObjectName] Default: `'json'`

The name of the packer for serializing messages. Should be one of 'json', 'pickle', or an import name for a custom callable serializer.

Session.session [CUnicode] Default: `''`

The UUID identifying this session.

Session.signature_scheme [Unicode] Default: `'hmac-sha256'`

The digest scheme used to construct the message signatures. Must have the form 'hmac-HASH'.

Session.unpacker [DottedObjectName] Default: `'json'`

The name of the unpacker for unserializing messages. Only used with custom functions for *packer*.

Session.username [Unicode] Default: `'username'`

Username for the Session. Default is your system username.

MultiKernelManager.default_kernel_name [Unicode] Default: `'python3'`

The name of the default kernel to start

MultiKernelManager.kernel_manager_class [DottedObjectName] Default: `'jupyter_client.ioloop.IOLoopKernelManager'`

The kernel manager class. This is configurable to allow subclassing of the KernelManager for customized behavior.

MultiKernelManager.shared_context [Bool] Default: `True`

Share a single zmq.Context to talk to all my kernels

MappingKernelManager.allowed_message_types [List] Default: `[]`

White list of allowed kernel message types. When the list is empty, all message types are allowed.

MappingKernelManager.buffer_offline_messages [Bool] Default: `True`

Whether messages from kernels whose frontends have disconnected should be buffered in-memory. When True (default), messages are buffered and replayed on reconnect, avoiding lost messages due to interrupted connectivity. Disable if long-running kernels will produce too much output while no frontends are connected.

MappingKernelManager.cull_busy [Bool] Default: `False`

Whether to consider culling kernels which are busy. Only effective if cull_idle_timeout > 0.

MappingKernelManager.cull_connected [Bool] Default: `False`

Whether to consider culling kernels which have one or more connections. Only effective if cull_idle_timeout > 0.

MappingKernelManager.cull_idle_timeout [Int] Default: `0`

Timeout (in seconds) after which a kernel is considered idle and ready to be culled. Values of 0 or lower disable culling. Very short timeouts may result in kernels being culled for users with poor network connections.

MappingKernelManager.cull_interval [Int] Default: `300`

The interval (in seconds) on which to check for idle kernels exceeding the cull timeout value.

MappingKernelManager.default_kernel_name [Unicode] Default: `'python3'`

The name of the default kernel to start

MappingKernelManager.kernel_info_timeout [Float] Default: `60`

Timeout for giving up on a kernel (in seconds). On starting and restarting kernels, we check whether the kernel is running and responsive by sending kernel_info_requests. This sets the timeout in seconds for how long the kernel can take before being presumed dead. This affects the MappingKernelManager (which handles kernel restarts) and the ZMQChannelsHandler (which handles the startup).

MappingKernelManager.kernel_manager_class [DottedObjectName] Default: `'jupyter_client.ioloop.IOLoopKernelManager'`

The kernel manager class. This is configurable to allow subclassing of the KernelManager for customized behavior.

MappingKernelManager.root_dir [Unicode] Default: `''`

No description

MappingKernelManager.shared_context [Bool] Default: `True`

Share a single zmq.Context to talk to all my kernels

KernelSpecManager.allowed_kernelspecs [Set] Default: `set()`

List of allowed kernel names.

By default, all installed kernels are allowed.

KernelSpecManager.ensure_native_kernel [Bool] Default: `True`

If there is no Python kernelspec registered and the IPython kernel is available, ensure it is added to the spec list.

KernelSpecManager.kernel_spec_class [Type] Default: `'jupyter_client.kernelspec.KernelSpec'`

The kernel spec class. This is configurable to allow subclassing of the KernelSpecManager for customized behavior.

KernelSpecManager.whitelist [Set] Default: `set()`

Deprecated, use *KernelSpecManager.allowed_kernelspecs*

ContentsManager.allow_hidden [Bool] Default: `False`

Allow access to hidden files

ContentsManager.checkpoints [Instance] Default: `None`

No description

ContentsManager.checkpoints_class [Type] Default: `'notebook.services.contents.checkpoints.Checkpoints'`

No description

ContentsManager.checkpoints_kwargs [Dict] Default: `{}`

No description

ContentsManager.files_handler_class [Type] Default: `'notebook.files.handlers.FilesHandler'`

handler class to use when serving raw file requests.

Default is a fallback that talks to the ContentsManager API, which may be inefficient, especially for large files.

Local files-based ContentsManagers can use a StaticFileHandler subclass, which will be much more efficient.

Access to these files should be Authenticated.

ContentsManager.files_handler_params [Dict] Default: `{}`

Extra parameters to pass to files_handler_class.

For example, StaticFileHandlers generally expect a *path* argument specifying the root directory from which to serve files.

ContentsManager.hide_globs [List] Default: `['__pycache__', '*.pyc', '*.pyo', '.DS_Store', '*.so', '*.dyl...`

Glob patterns to hide in file and directory listings.

ContentsManager.pre_save_hook [Any] Default: `None`

Python callable or importstring thereof

To be called on a contents model prior to save.

This can be used to process the structure, such as removing notebook outputs or other side effects that should not be saved.

It will be called as (all arguments passed by keyword):

```
hook(path=path, model=model, contents_manager=self)
```

- model: the model to be saved. Includes file contents. Modifying this dict will affect the file that is stored.
- path: the API path of the save destination
- contents_manager: this ContentsManager instance

ContentsManager.root_dir [Unicode] Default: `'/'`

No description

ContentsManager.untitled_directory [Unicode] Default: `'Untitled Folder'`

The base name used when creating untitled directories.

ContentsManager.untitled_file [Unicode] Default: `'untitled'`

The base name used when creating untitled files.

ContentsManager.untitled_notebook [Unicode] Default: `'Untitled'`

The base name used when creating untitled notebooks.

FileManagerMixin.use_atomic_writing [Bool] Default: `True`

By default notebooks are saved on disk on a temporary file and then if successfully written, it replaces the old ones. This procedure, namely 'atomic_writing', causes some bugs on file system without operation order enforcement (like some networked fs). If set to False, the new notebook is written directly on the old one which could fail (eg: full filesystem or quota)

FileContentsManager.allow_hidden [Bool] Default: `False`

Allow access to hidden files

FileContentsManager.checkpoints [Instance] Default: `None`

No description

FileContentsManager.checkpoints_class [Type] Default: `'notebook.services.contents.checkpoints.Checkpoints'`

No description

FileContentsManager.checkpoints_kwargs [Dict] Default: `{}`

No description

FileContentsManager.delete_to_trash [Bool] Default: `True`

If True (default), deleting files will send them to the platform's trash/recycle bin, where they can be recovered. If False, deleting files really deletes them.

FileContentsManager.files_handler_class [Type] Default: `'notebook.files.handlers.FilesHandler'`

handler class to use when serving raw file requests.

Default is a fallback that talks to the ContentsManager API, which may be inefficient, especially for large files.

Local files-based ContentsManagers can use a StaticFileHandler subclass, which will be much more efficient.

Access to these files should be Authenticated.

FileContentsManager.files_handler_params [Dict] Default: `{}`

Extra parameters to pass to files_handler_class.

For example, StaticFileHandlers generally expect a *path* argument specifying the root directory from which to serve files.

FileContentsManager.hide_globs [List] Default: `['__pycache__', '*.pyc', '*.pyo', '.DS_Store', '*.so', '*.dyl...`

Glob patterns to hide in file and directory listings.

FileContentsManager.post_save_hook [Any] Default: `None`

Python callable or importstring thereof

to be called on the path of a file just saved.

This can be used to process the file on disk, such as converting the notebook to a script or HTML via nbconvert.

It will be called as (all arguments passed by keyword):

```
hook(os_path=os_path, model=model, contents_manager=instance)
```

- path: the filesystem path to the file just written
- model: the model representing the file
- contents_manager: this ContentsManager instance

FileContentsManager.pre_save_hook [Any] Default: `None`

Python callable or importstring thereof

To be called on a contents model prior to save.

This can be used to process the structure, such as removing notebook outputs or other side effects that should not be saved.

It will be called as (all arguments passed by keyword):

```
hook(path=path, model=model, contents_manager=self)
```

- model: the model to be saved. Includes file contents. Modifying this dict will affect the file that is stored.
- path: the API path of the save destination
- contents_manager: this ContentsManager instance

FileContentsManager.root_dir [Unicode] Default: `''`

No description

FileContentsManager.save_script [Bool] Default: `False`

DEPRECATED, use post_save_hook. Will be removed in Notebook 5.0

FileContentsManager.untitled_directory [Unicode] Default: `'Untitled Folder'`

The base name used when creating untitled directories.

FileContentsManager.untitled_file [Unicode] Default: `'untitled'`

The base name used when creating untitled files.

FileContentsManager.untitled_notebook [Unicode] Default: `'Untitled'`

The base name used when creating untitled notebooks.

FileContentsManager.use_atomic_writing [Bool] Default: `True`

By default notebooks are saved on disk on a temporary file and then if successfully written, it replaces the old ones. This procedure, namely 'atomic_writing', causes some bugs on file system without operation order enforcement (like some networked fs). If set to False, the new notebook is written directly on the old one which could fail (eg: full filesystem or quota)

NotebookNotary.algorithm [any of `'blake2s'` | 'sha256' | 'sha1' | 'sha512' | 'sha3_224' | 'sha3_384' | 'md5' | 'sha224' | 'sha384' | 'bl
Default: `'sha256'`

The hashing algorithm used to sign notebooks.

NotebookNotary.data_dir [Unicode] Default: `''`

The storage directory for notary secret and database.

NotebookNotary.db_file [Unicode] Default: `''`

The sqlite file in which to store notebook signatures. By default, this will be in your Jupyter data directory. You can set it to ':memory:' to disable sqlite writing to the filesystem.

NotebookNotary.secret [Bytes] Default: `b''`

The secret key with which notebooks are signed.

NotebookNotary.secret_file [Unicode] Default: `''`

The file where the secret key is stored.

NotebookNotary.store_factory [Callable] Default: `traitlets.Undefined`

A callable returning the storage backend for notebook signatures. The default uses an SQLite database.

AsyncMultiKernelManager.default_kernel_name [Unicode] Default: `'python3'`

The name of the default kernel to start

AsyncMultiKernelManager.kernel_manager_class [DottedObjectName] Default: `'jupyter_client.ioloop.AsyncIOLoopKernelManager'`

The kernel manager class. This is configurable to allow subclassing of the AsyncKernelManager for customized behavior.

AsyncMultiKernelManager.shared_context [Bool] Default: `True`

Share a single zmq.Context to talk to all my kernels

AsyncMultiKernelManager.use_pending_kernels [Bool] Default: `False`

Whether to make kernels available before the process has started. The kernel has a *.ready* future which can be awaited before connecting

AsyncMappingKernelManager.allowed_message_types [List] Default: `[]`

White list of allowed kernel message types. When the list is empty, all message types are allowed.

AsyncMappingKernelManager.buffer_offline_messages [Bool] Default: `True`

Whether messages from kernels whose frontends have disconnected should be buffered in-memory. When True (default), messages are buffered and replayed on reconnect, avoiding lost messages due to interrupted connectivity. Disable if long-running kernels will produce too much output while no frontends are connected.

AsyncMappingKernelManager.cull_busy [Bool] Default: `False`

Whether to consider culling kernels which are busy. Only effective if cull_idle_timeout > 0.

AsyncMappingKernelManager.cull_connected [Bool] Default: `False`

Whether to consider culling kernels which have one or more connections. Only effective if cull_idle_timeout > 0.

AsyncMappingKernelManager.cull_idle_timeout [Int] Default: `0`

Timeout (in seconds) after which a kernel is considered idle and ready to be culled. Values of 0 or lower disable culling. Very short timeouts may result in kernels being culled for users with poor network connections.

AsyncMappingKernelManager.cull_interval [Int] Default: `300`

The interval (in seconds) on which to check for idle kernels exceeding the cull timeout value.

AsyncMappingKernelManager.default_kernel_name [Unicode] Default: `'python3'`

The name of the default kernel to start

AsyncMappingKernelManager.kernel_info_timeout [Float] Default: `60`

Timeout for giving up on a kernel (in seconds). On starting and restarting kernels, we check whether the kernel is running and responsive by sending kernel_info_requests. This sets the timeout in seconds for how long the kernel can take before being presumed dead. This affects the MappingKernelManager (which handles kernel restarts) and the ZMQChannelsHandler (which handles the startup).

AsyncMappingKernelManager.kernel_manager_class [DottedObjectName] Default: `'jupyter_client.ioloop.AsyncIOLoopKernelManager'`

The kernel manager class. This is configurable to allow subclassing of the AsyncKernelManager for customized behavior.

AsyncMappingKernelManager.root_dir [Unicode] Default: `''`

No description

AsyncMappingKernelManager.shared_context [Bool] Default: `True`

Share a single zmq.Context to talk to all my kernels

AsyncMappingKernelManager.use_pending_kernels [Bool] Default: `False`

Whether to make kernels available before the process has started. The kernel has a *.ready* future which can be awaited before connecting

GatewayKernelManager.allowed_message_types [List] Default: `[]`

White list of allowed kernel message types. When the list is empty, all message types are allowed.

GatewayKernelManager.buffer_offline_messages [Bool] Default: `True`

Whether messages from kernels whose frontends have disconnected should be buffered in-memory. When True (default), messages are buffered and replayed on reconnect, avoiding lost messages due to interrupted connectivity. Disable if long-running kernels will produce too much output while no frontends are connected.

GatewayKernelManager.cull_busy [Bool] Default: `False`

Whether to consider culling kernels which are busy. Only effective if cull_idle_timeout > 0.

GatewayKernelManager.cull_connected [Bool] Default: `False`

Whether to consider culling kernels which have one or more connections. Only effective if cull_idle_timeout > 0.

GatewayKernelManager.cull_idle_timeout [Int] Default: `0`

Timeout (in seconds) after which a kernel is considered idle and ready to be culled. Values of 0 or lower disable culling. Very short timeouts may result in kernels being culled for users with poor network connections.

GatewayKernelManager.cull_interval [Int] Default: `300`

The interval (in seconds) on which to check for idle kernels exceeding the cull timeout value.

GatewayKernelManager.default_kernel_name [Unicode] Default: `'python3'`

The name of the default kernel to start

GatewayKernelManager.kernel_info_timeout [Float] Default: `60`

Timeout for giving up on a kernel (in seconds). On starting and restarting kernels, we check whether the kernel is running and responsive by sending kernel_info_requests. This sets the timeout in seconds for how long the kernel can take before being presumed dead. This affects the MappingKernelManager (which handles kernel restarts) and the ZMQChannelsHandler (which handles the startup).

GatewayKernelManager.kernel_manager_class [DottedObjectName] Default: `'jupyter_client.ioloop.AsyncIOLoopKernelManager'`

The kernel manager class. This is configurable to allow subclassing of the AsyncKernelManager for customized behavior.

GatewayKernelManager.root_dir [Unicode] Default: `''`

No description

GatewayKernelManager.shared_context [Bool] Default: `True`

Share a single zmq.Context to talk to all my kernels

GatewayKernelManager.use_pending_kernels [Bool] Default: `False`

Whether to make kernels available before the process has started. The kernel has a *.ready* future which can be awaited before connecting

GatewayKernelSpecManager.allowed_kernelspecs [Set] Default: `set()`

List of allowed kernel names.

By default, all installed kernels are allowed.

GatewayKernelSpecManager.ensure_native_kernel [Bool] Default: `True`

If there is no Python kernelspec registered and the IPython kernel is available, ensure it is added to the spec list.

GatewayKernelSpecManager.kernel_spec_class [Type] Default: `'jupyter_client.kernelspec.KernelSpec'`

The kernel spec class. This is configurable to allow subclassing of the KernelSpecManager for customized behavior.

GatewayKernelSpecManager.whitelist [Set] Default: `set()`

Deprecated, use *KernelSpecManager.allowed_kernelspecs*

GatewayClient.auth_token [Unicode] Default: `None`

The authorization token used in the HTTP headers. (JUPYTER_GATEWAY_AUTH_TOKEN env var)

GatewayClient.ca_certs [Unicode] Default: `None`

The filename of CA certificates or None to use defaults. (JUPYTER_GATEWAY_CA_CERTS env var)

GatewayClient.client_cert [Unicode] Default: `None`

The filename for client SSL certificate, if any. (JUPYTER_GATEWAY_CLIENT_CERT env var)

GatewayClient.client_key [Unicode] Default: `None`

The filename for client SSL key, if any. (JUPYTER_GATEWAY_CLIENT_KEY env var)

GatewayClient.connect_timeout [Float] Default: `40.0`

The time allowed for HTTP connection establishment with the Gateway server. (JUPYTER_GATEWAY_CONNECT_TIMEOUT env var)

GatewayClient.env_whitelist [Unicode] Default: `''`

A comma-separated list of environment variable names that will be included, along with their values, in the kernel startup request. The corresponding *env_whitelist* configuration value must also be set on the Gateway server - since that configuration value indicates which environmental values to make available to the kernel. (JUPYTER_GATEWAY_ENV_WHITELIST env var)

GatewayClient.gateway_retry_interval [Float] Default: `1.0`

The time allowed for HTTP reconnection with the Gateway server for the first time. Next will be JUPYTER_GATEWAY_RETRY_INTERVAL multiplied by two in factor of numbers of retries but less than JUPYTER_GATEWAY_RETRY_INTERVAL_MAX. (JUPYTER_GATEWAY_RETRY_INTERVAL env var)

GatewayClient.gateway_retry_interval_max [Float] Default: `30.0`

The maximum time allowed for HTTP reconnection retry with the Gateway server. (JUPYTER_GATEWAY_RETRY_INTERVAL_MAX env var)

GatewayClient.gateway_retry_max [Int] Default: `5`

The maximum retries allowed for HTTP reconnection with the Gateway server. (JUPYTER_GATEWAY_RETRY_MAX env var)

GatewayClient.headers [Unicode] Default: `'{}'`

Additional HTTP headers to pass on the request. This value will be converted to a dict. (JUPYTER_GATEWAY_HEADERS env var)

GatewayClient.http_pwd [Unicode] Default: `None`

The password for HTTP authentication. (JUPYTER_GATEWAY_HTTP_PWD env var)

GatewayClient.http_user [Unicode] Default: `None`

The username for HTTP authentication. (JUPYTER_GATEWAY_HTTP_USER env var)

GatewayClient.kernels_endpoint [Unicode] Default: `'/api/kernels'`

The gateway API endpoint for accessing kernel resources (JUPYTER_GATEWAY_KERNELS_ENDPOINT env var)

GatewayClient.kernelspecs_endpoint [Unicode] Default: `'/api/kernelspecs'`

The gateway API endpoint for accessing kernelspecs (JUPYTER_GATEWAY_KERNELSPECS_ENDPOINT env var)

GatewayClient.kernelspecs_resource_endpoint [Unicode] Default: `'/kernelspecs'`

The gateway endpoint for accessing kernelspecs resources (JUPYTER_GATEWAY_KERNELSPECS_RESOURCE_ENDPO env var)

GatewayClient.request_timeout [Float] Default: `40.0`

The time allowed for HTTP request completion. (JUPYTER_GATEWAY_REQUEST_TIMEOUT env var)

GatewayClient.url [Unicode] Default: `None`

The url of the Kernel or Enterprise Gateway server where kernel specifications are defined and kernel management takes place. If defined, this Notebook server acts as a proxy for all kernel management and kernel specification retrieval. (JUPYTER_GATEWAY_URL env var)

GatewayClient.validate_cert [Bool] Default: `True`

For HTTPS requests, determines if server's certificate should be validated or not. (JUPYTER_GATEWAY_VALIDATE_CERT env var)

GatewayClient.ws_url [Unicode] Default: `None`

The websocket url of the Kernel or Enterprise Gateway server. If not provided, this value will correspond to the value of the Gateway url with 'ws' in place of 'http'. (JUPYTER_GATEWAY_WS_URL env var)

TerminalManager.cull_inactive_timeout [Int] Default: `0`

Timeout (in seconds) in which a terminal has been inactive and ready to be culled. Values of 0 or lower disable culling.

TerminalManager.cull_interval [Int] Default: `300`

The interval (in seconds) on which to check for terminals exceeding the inactive timeout value.

CHAPTER

NINE

RUNNING A NOTEBOOK SERVER

The *Jupyter notebook* web application is based on a server-client structure. The notebook server uses a two-process kernel architecture based on ZeroMQ, as well as Tornado for serving HTTP requests.

Note: By default, a notebook server runs locally at 127.0.0.1:8888 and is accessible only from *localhost*. You may access the notebook server from the browser using *http://127.0.0.1:8888*.

This document describes how you can *secure a notebook server* and how to *run it on a public interface*.

Important: **This is not the multi-user server you are looking for**. This document describes how you can run a public server with a single user. This should only be done by someone who wants remote access to their personal machine. Even so, doing this requires a thorough understanding of the set-ups limitations and security implications. If you allow multiple users to access a notebook server as it is described in this document, their commands may collide, clobber and overwrite each other.

If you want a multi-user server, the official solution is JupyterHub. To use JupyterHub, you need a Unix server (typically Linux) running somewhere that is accessible to your users on a network. This may run over the public internet, but doing so introduces additional security concerns.

9.1 Securing a notebook server

You can protect your notebook server with a simple single password. As of notebook 5.0 this can be done automatically. To set up a password manually you can configure the `NotebookApp.password` setting in `jupyter_notebook_config.py`.

9.1.1 Prerequisite: A notebook configuration file

Check to see if you have a notebook configuration file, `jupyter_notebook_config.py`. The default location for this file is your Jupyter folder located in your home directory:

- Windows: `C:\Users\USERNAME\.jupyter\jupyter_notebook_config.py`
- OS X: `/Users/USERNAME/.jupyter/jupyter_notebook_config.py`
- Linux: `/home/USERNAME/.jupyter/jupyter_notebook_config.py`

If you don't already have a Jupyter folder, or if your Jupyter folder doesn't contain a notebook configuration file, run the following command:

```
$ jupyter notebook --generate-config
```

This command will create the Jupyter folder if necessary, and create notebook configuration file, `jupyter_notebook_config.py`, in this folder.

9.1.2 Automatic Password setup

As of notebook 5.3, the first time you log-in using a token, the notebook server should give you the opportunity to setup a password from the user interface.

You will be presented with a form asking for the current _token_, as well as your _new_ _password_ ; enter both and click on `Login and setup new password`.

Next time you need to log in you'll be able to use the new password instead of the login token, otherwise follow the procedure to set a password from the command line.

The ability to change the password at first login time may be disabled by integrations by setting the `--NotebookApp.allow_password_change=False`

Starting at notebook version 5.0, you can enter and store a password for your notebook server with a single command. **`jupyter notebook password`** will prompt you for your password and record the hashed password in your `jupyter_notebook_config.json`.

```
$ jupyter notebook password
Enter password:  ****
Verify password: ****
[NotebookPasswordApp] Wrote hashed password to /Users/you/.jupyter/jupyter_notebook_
↪config.json
```

This can be used to reset a lost password; or if you believe your credentials have been leaked and desire to change your password. Changing your password will invalidate all logged-in sessions after a server restart.

9.1.3 Preparing a hashed password

You can prepare a hashed password manually, using the function `notebook.auth.security.passwd()`:

```
In [1]: from notebook.auth import passwd
In [2]: passwd()
Enter password:
Verify password:
Out[2]: 'sha1:67c9e60bb8b6:9ffede0825894254b2e042ea597d771089e11aed'
```

> **Caution:** `passwd()` when called with no arguments will prompt you to enter and verify your password such as in the above code snippet. Although the function can also be passed a string as an argument such as `passwd('mypassword')`, please **do not** pass a string as an argument inside an IPython session, as it will be saved in your input history.

9.1.4 Adding hashed password to your notebook configuration file

You can then add the hashed password to your `jupyter_notebook_config.py`. The default location for this file `jupyter_notebook_config.py` is in your Jupyter folder in your home directory, `~/.jupyter`, e.g.:

```
c.NotebookApp.password = u'sha1:67c9e60bb8b6:9ffede0825894254b2e042ea597d771089e11aed'
```

Automatic password setup will store the hash in `jupyter_notebook_config.json` while this method stores the hash in `jupyter_notebook_config.py`. The `.json` configuration options take precedence over the `.py` one, thus the manual password may not take effect if the Json file has a password set.

9.1.5 Using SSL for encrypted communication

When using a password, it is a good idea to also use SSL with a web certificate, so that your hashed password is not sent unencrypted by your browser.

Important: Web security is rapidly changing and evolving. We provide this document as a convenience to the user, and recommend that the user keep current on changes that may impact security, such as new releases of OpenSSL. The Open Web Application Security Project (OWASP) website is a good resource on general security issues and web practices.

You can start the notebook to communicate via a secure protocol mode by setting the `certfile` option to your self-signed certificate, i.e. `mycert.pem`, with the command:

```
$ jupyter notebook --certfile=mycert.pem --keyfile mykey.key
```

Tip: A self-signed certificate can be generated with `openssl`. For example, the following command will create a certificate valid for 365 days with both the key and certificate data written to the same file:

```
$ openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout mykey.key -out mycert.pem
```

When starting the notebook server, your browser may warn that your self-signed certificate is insecure or unrecognized. If you wish to have a fully compliant self-signed certificate that will not raise warnings, it is possible (but rather involved) to create one, as explained in detail in this tutorial. Alternatively, you may use Let's Encrypt to acquire a free SSL certificate and follow the steps in *Using Let's Encrypt* to set up a public server.

9.2 Running a public notebook server

If you want to access your notebook server remotely via a web browser, you can do so by running a public notebook server. For optimal security when running a public notebook server, you should first secure the server with a password and SSL/HTTPS as described in *Securing a notebook server*.

Start by creating a certificate file and a hashed password, as explained in *Securing a notebook server*.

If you don't already have one, create a config file for the notebook using the following command line:

```
$ jupyter notebook --generate-config
```

In the ~/.jupyter directory, edit the notebook config file, jupyter_notebook_config.py. By default, the notebook config file has all fields commented out. The minimum set of configuration options that you should uncomment and edit in jupyter_notebook_config.py is the following:

```
# Set options for certfile, ip, password, and toggle off
# browser auto-opening
c.NotebookApp.certfile = u'/absolute/path/to/your/certificate/mycert.pem'
c.NotebookApp.keyfile = u'/absolute/path/to/your/certificate/mykey.key'
# Set ip to '*' to bind on all interfaces (ips) for the public server
c.NotebookApp.ip = '*'
c.NotebookApp.password = u'sha1:bcd259ccf...<your hashed password here>'
c.NotebookApp.open_browser = False

# It is a good idea to set a known, fixed port for server access
c.NotebookApp.port = 9999
```

You can then start the notebook using the jupyter notebook command.

9.2.1 Using Let's Encrypt

Let's Encrypt provides free SSL/TLS certificates. You can also set up a public server using a Let's Encrypt certificate.

Running a public notebook server will be similar when using a Let's Encrypt certificate with a few configuration changes. Here are the steps:

1. Create a Let's Encrypt certificate.
2. Use *Preparing a hashed password* to create one.
3. If you don't already have config file for the notebook, create one using the following command:

```
$ jupyter notebook --generate-config
```

4. In the ~/.jupyter directory, edit the notebook config file, jupyter_notebook_config.py. By default, the notebook config file has all fields commented out. The minimum set of configuration options that you should to uncomment and edit in jupyter_notebook_config.py is the following:

```
# Set options for certfile, ip, password, and toggle off
# browser auto-opening
c.NotebookApp.certfile = u'/absolute/path/to/your/certificate/fullchain.pem'
c.NotebookApp.keyfile = u'/absolute/path/to/your/certificate/privkey.pem'
# Set ip to '*' to bind on all interfaces (ips) for the public server
c.NotebookApp.ip = '*'
c.NotebookApp.password = u'sha1:bcd259ccf...<your hashed password here>'
c.NotebookApp.open_browser = False

# It is a good idea to set a known, fixed port for server access
c.NotebookApp.port = 9999
```

You can then start the notebook using the jupyter notebook command.

Important: Use 'https'. Keep in mind that when you enable SSL support, you must access the notebook server over https://, not over plain http://. The startup message from the server prints a reminder in the console, but *it is easy to overlook this detail and think the server is for some reason non-responsive*.

When using SSL, always access the notebook server with 'https://'.

You may now access the public server by pointing your browser to `https://your.host.com:9999` where `your.host.com` is your public server's domain.

9.2.2 Firewall Setup

To function correctly, the firewall on the computer running the jupyter notebook server must be configured to allow connections from client machines on the access port `c.NotebookApp.port` set in `jupyter_notebook_config.py` to allow connections to the web interface. The firewall must also allow connections from 127.0.0.1 (localhost) on ports from 49152 to 65535. These ports are used by the server to communicate with the notebook kernels. The kernel communication ports are chosen randomly by ZeroMQ, and may require multiple connections per kernel, so a large range of ports must be accessible.

9.3 Running the notebook with a customized URL prefix

The notebook dashboard, which is the landing page with an overview of the notebooks in your working directory, is typically found and accessed at the default URL `http://localhost:8888/`.

If you prefer to customize the URL prefix for the notebook dashboard, you can do so through modifying `jupyter_notebook_config.py`. For example, if you prefer that the notebook dashboard be located with a subdirectory that contains other ipython files, e.g. `http://localhost:8888/ipython/`, you can do so with configuration options like the following (see above for instructions about modifying `jupyter_notebook_config.py`):

```
c.NotebookApp.base_url = '/ipython/'
```

9.4 Embedding the notebook in another website

Sometimes you may want to embed the notebook somewhere on your website, e.g. in an IFrame. To do this, you may need to override the Content-Security-Policy to allow embedding. Assuming your website is at *https://mywebsite.example.com*, you can embed the notebook on your website with the following configuration setting in `jupyter_notebook_config.py`:

```
c.NotebookApp.tornado_settings = {
    'headers': {
        'Content-Security-Policy': "frame-ancestors https://mywebsite.example.com 'self'
↪"
    }
}
```

When embedding the notebook in a website using an iframe, consider putting the notebook in single-tab mode. Since the notebook opens some links in new tabs by default, single-tab mode keeps the notebook from opening additional tabs. Adding the following to `~/.jupyter/custom/custom.js` will enable single-tab mode:

```
define(['base/js/namespace'], function(Jupyter){
    Jupyter._target = '_self';
});
```

9.5 Using a gateway server for kernel management

You are now able to redirect the management of your kernels to a Gateway Server (i.e., Jupyter Kernel Gateway or Jupyter Enterprise Gateway) simply by specifying a Gateway url via the following command-line option:

```
$ jupyter notebook --gateway-url=http://my-gateway-server:8888
```

the environment:

```
JUPYTER_GATEWAY_URL=http://my-gateway-server:8888
```

or in `jupyter_notebook_config.py`:

```
c.GatewayClient.url = http://my-gateway-server:8888
```

When provided, all kernel specifications will be retrieved from the specified Gateway server and all kernels will be managed by that server. This option enables the ability to target kernel processes against managed clusters while allowing for the notebook's management to remain local to the Notebook server.

9.6 Known issues

9.6.1 Proxies

When behind a proxy, especially if your system or browser is set to autodetect the proxy, the notebook web application might fail to connect to the server's websockets, and present you with a warning at startup. In this case, you need to configure your system not to use the proxy for the server's address.

For example, in Firefox, go to the Preferences panel, Advanced section, Network tab, click 'Settings…', and add the address of the notebook server to the 'No proxy for' field.

9.6.2 Content-Security-Policy (CSP)

Certain security guidelines recommend that servers use a Content-Security-Policy (CSP) header to prevent cross-site scripting vulnerabilities, specifically limiting to `default-src: https:` when possible. This directive causes two problems with Jupyter. First, it disables execution of inline javascript code, which is used extensively by Jupyter. Second, it limits communication to the https scheme, and prevents WebSockets from working because they communicate via the wss scheme (or ws for insecure communication). Jupyter uses WebSockets for interacting with kernels, so when you visit a server with such a CSP, your browser will block attempts to use wss, which will cause you to see "Connection failed" messages from jupyter notebooks, or simply no response from jupyter terminals. By looking in your browser's javascript console, you can see any error messages that will explain what is failing.

To avoid these problem, you need to add `'unsafe-inline'` and `connect-src https: wss:` to your CSP header, at least for pages served by jupyter. (That is, you can leave your CSP unchanged for other parts of your website.) Note that multiple CSP headers are allowed, but successive CSP headers can only restrict the policy; they cannot loosen it. For example, if your server sends both of these headers

> Content-Security-Policy "default-src https: 'unsafe-inline'" Content-Security-Policy "connect-src https: wss:"

the first policy will already eliminate wss connections, so the second has no effect. Therefore, you can't simply add the second header; you have to actually modify your CSP header to look more like this:

> Content-Security-Policy "default-src https: 'unsafe-inline'; connect-src https: wss:"

9.6.3 Docker CMD

Using `jupyter notebook` as a Docker CMD results in kernels repeatedly crashing, likely due to a lack of PID reaping. To avoid this, use the tini `init` as your Dockerfile *ENTRYPOINT*:

```
# Add Tini. Tini operates as a process subreaper for jupyter. This prevents
# kernel crashes.
ENV TINI_VERSION v0.6.0
ADD https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini /usr/bin/tini
RUN chmod +x /usr/bin/tini
ENTRYPOINT ["/usr/bin/tini", "--"]

EXPOSE 8888
CMD ["jupyter", "notebook", "--port=8888", "--no-browser", "--ip=0.0.0.0"]
```

CHAPTER

TEN

SECURITY IN THE JUPYTER NOTEBOOK SERVER

Since access to the Jupyter notebook server means access to running arbitrary code, it is important to restrict access to the notebook server. For this reason, notebook 4.3 introduces token-based authentication that is **on by default**.

Note: If you enable a password for your notebook server, token authentication is not enabled by default, and the behavior of the notebook server is unchanged from versions earlier than 4.3.

When token authentication is enabled, the notebook uses a token to authenticate requests. This token can be provided to login to the notebook server in three ways:

- in the `Authorization` header, e.g.:

```
Authorization: token abcdef...
```

- In a URL parameter, e.g.:

```
https://my-notebook/tree/?token=abcdef...
```

- In the password field of the login form that will be shown to you if you are not logged in.

When you start a notebook server with token authentication enabled (default), a token is generated to use for authentication. This token is logged to the terminal, so that you can copy/paste the URL into your browser:

```
[I 11:59:16.597 NotebookApp] The Jupyter Notebook is running at:
http://localhost:8888/?token=c8de56fa4deed24899803e93c227592aef6538f93025fe01
```

If the notebook server is going to open your browser automatically (the default, unless `--no-browser` has been passed), an *additional* token is generated for launching the browser. This additional token can be used only once, and is used to set a cookie for your browser once it connects. After your browser has made its first request with this one-time-token, the token is discarded and a cookie is set in your browser.

At any later time, you can see the tokens and URLs for all of your running servers with **`jupyter notebook list`**:

```
$ jupyter notebook list
Currently running servers:
http://localhost:8888/?token=abc... :: /home/you/notebooks
https://0.0.0.0:9999/?token=123... :: /tmp/public
http://localhost:8889/ :: /tmp/has-password
```

For servers with token-authentication enabled, the URL in the above listing will include the token, so you can copy and paste that URL into your browser to login. If a server has no token (e.g. it has a password or has authentication disabled), the URL will not include the token argument. Once you have visited this URL, a cookie will be set in your browser and you won't need to use the token again, unless you switch browsers, clear your cookies, or start a notebook server on a new port.

10.1 Alternatives to token authentication

If a generated token doesn't work well for you, you can set a password for your notebook. **jupyter notebook password** will prompt you for a password, and store the hashed password in your `jupyter_notebook_config.json`.

New in version 5.0: **jupyter notebook password** command is added.

It is possible to disable authentication altogether by setting the token and password to empty strings, but this is **NOT RECOMMENDED**, unless authentication or access restrictions are handled at a different layer in your web application:

```
c.NotebookApp.token = ''
c.NotebookApp.password = ''
```

CHAPTER

ELEVEN

SECURITY IN NOTEBOOK DOCUMENTS

As Jupyter notebooks become more popular for sharing and collaboration, the potential for malicious people to attempt to exploit the notebook for their nefarious purposes increases. IPython 2.0 introduced a security model to prevent execution of untrusted code without explicit user input.

11.1 The problem

The whole point of Jupyter is arbitrary code execution. We have no desire to limit what can be done with a notebook, which would negatively impact its utility.

Unlike other programs, a Jupyter notebook document includes output. Unlike other documents, that output exists in a context that can execute code (via Javascript).

The security problem we need to solve is that no code should execute just because a user has **opened** a notebook that **they did not write**. Like any other program, once a user decides to execute code in a notebook, it is considered trusted, and should be allowed to do anything.

11.2 Our security model

- Untrusted HTML is always sanitized
- Untrusted Javascript is never executed
- HTML and Javascript in Markdown cells are never trusted
- **Outputs** generated by the user are trusted
- Any other HTML or Javascript (in Markdown cells, output generated by others) is never trusted
- The central question of trust is "Did the current user do this?"

11.3 The details of trust

When a notebook is executed and saved, a signature is computed from a digest of the notebook's contents plus a secret key. This is stored in a database, writable only by the current user. By default, this is located at:

```
~/.local/share/jupyter/nbsignatures.db  # Linux
~/Library/Jupyter/nbsignatures.db       # OS X
%APPDATA%/jupyter/nbsignatures.db       # Windows
```

Each signature represents a series of outputs which were produced by code the current user executed, and are therefore trusted.

When you open a notebook, the server computes its signature, and checks if it's in the database. If a match is found, HTML and Javascript output in the notebook will be trusted at load, otherwise it will be untrusted.

Any output generated during an interactive session is trusted.

11.3.1 Updating trust

A notebook's trust is updated when the notebook is saved. If there are any untrusted outputs still in the notebook, the notebook will not be trusted, and no signature will be stored. If all untrusted outputs have been removed (either via `Clear Output` or re-execution), then the notebook will become trusted.

While trust is updated per output, this is only for the duration of a single session. A newly loaded notebook file is either trusted or not in its entirety.

11.3.2 Explicit trust

Sometimes re-executing a notebook to generate trusted output is not an option, either because dependencies are unavailable, or it would take a long time. Users can explicitly trust a notebook in two ways:

- At the command-line, with:

```
jupyter trust /path/to/notebook.ipynb
```

- After loading the untrusted notebook, with `File / Trust Notebook`

These two methods simply load the notebook, compute a new signature, and add that signature to the user's database.

11.4 Reporting security issues

If you find a security vulnerability in Jupyter, either a failure of the code to properly implement the model described here, or a failure of the model itself, please report it to security@ipython.org.

If you prefer to encrypt your security reports, you can use `this PGP public key`.

11.5 Affected use cases

Some use cases that work in Jupyter 1.0 became less convenient in 2.0 as a result of the security changes. We do our best to minimize these annoyances, but security is always at odds with convenience.

11.5.1 Javascript and CSS in Markdown cells

While never officially supported, it had become common practice to put hidden Javascript or CSS styling in Markdown cells, so that they would not be visible on the page. Since Markdown cells are now sanitized (by Google Caja), all Javascript (including click event handlers, etc.) and CSS will be stripped.

We plan to provide a mechanism for notebook themes, but in the meantime styling the notebook can only be done via either `custom.css` or CSS in HTML output. The latter only have an effect if the notebook is trusted, because otherwise the output will be sanitized just like Markdown.

11.5.2 Collaboration

When collaborating on a notebook, people probably want to see the outputs produced by their colleagues' most recent executions. Since each collaborator's key will differ, this will result in each share starting in an untrusted state. There are three basic approaches to this:

- re-run notebooks when you get them (not always viable)
- explicitly trust notebooks via `jupyter trust` or the notebook menu (annoying, but easy)
- share a notebook signatures database, and use configuration dedicated to the collaboration while working on the project.

To share a signatures database among users, you can configure:

```
c.NotebookNotary.data_dir = "/path/to/signature_dir"
```

to specify a non-default path to the SQLite database (of notebook hashes, essentially). We are aware that SQLite doesn't work well on NFS and we are working out better ways to do this.

CHAPTER

TWELVE

DISTRIBUTING JUPYTER EXTENSIONS AS PYTHON PACKAGES

12.1 Overview

12.1.1 How can the notebook be extended?

The Jupyter Notebook client and server application are both deeply customizable. Their behavior can be extended by creating, respectively:

- nbextension: a notebook extension
  - a single JS file, or directory of JavaScript, Cascading StyleSheets, etc. that contain at minimum a JavaScript module packaged as an AMD modules that exports a function `load_ipython_extension`
- server extension: an importable Python module
  - that implements `load_jupyter_server_extension`
- bundler extension: an importable Python module with generated File -> Download as / Deploy as menu item trigger
  - that implements `bundle`

12.1.2 Why create a Python package for Jupyter extensions?

Since it is rare to have a server extension that does not have any frontend components (an nbextension), for convenience and consistency, all these client and server extensions with their assets can be packaged and versioned together as a Python package with a few simple commands, or as of Notebook 5.3, handled automatically by your package manager of choice. This makes installing the package of extensions easier and less error-prone for the user.

12.2 Installation of Jupyter Extensions

12.2.1 Install a Python package containing Jupyter Extensions

There are several ways that you may get a Python package containing Jupyter Extensions. Commonly, you will use a package manager for your system:

```
pip install helpful_package
# or
conda install helpful_package
# or
```

(continues on next page)

(continued from previous page)

```
apt-get install helpful_package

# where 'helpful_package' is a Python package containing one or more Jupyter Extensions
```

12.2.2 Automatic installation and Enabling

New in Notebook 5.3

The absolute simplest case requires no user interaction at all! Configured correctly, after installing with their package manager of choice, both server and frontend extensions can be enabled by default in the environment where they were installed, i.e. `--sys-prefix`. See the `setup.py` in the example below.

12.2.3 Enable a Server Extension

The simplest case would be to enable a server extension which has no frontend components.

A `pip` user that wants their configuration stored in their home directory would type the following command:

```
jupyter serverextension enable --py helpful_package
```

Alternatively, a `virtualenv` or `conda` user can pass `--sys-prefix` which keeps their environment isolated and reproducible. For example:

```
# Make sure that your virtualenv or conda environment is activated
[source] activate my-environment

jupyter serverextension enable --py helpful_package --sys-prefix
```

12.2.4 Install the nbextension assets

If a package also has an nbextension with frontend assets that must be available (but not neccessarily enabled by default), install these assets with the following command:

```
jupyter nbextension install --py helpful_package # or --sys-prefix if using virtualenv␣
↪or conda
```

12.2.5 Enable nbextension assets

If a package has assets that should be loaded every time a Jupyter app (e.g. lab, notebook, dashboard, terminal) is loaded in the browser, the following command can be used to enable the nbextension:

```
jupyter nbextension enable --py helpful_package # or --sys-prefix if using virtualenv or␣
↪conda
```

12.3 Did it work? Check by listing Jupyter Extensions.

After running one or more extension installation steps, you can list what is presently known about nbextensions, server extensions, or bundler extensions. The following commands will list which extensions are available, whether they are enabled, and other extension details:

```
jupyter nbextension list
jupyter serverextension list
jupyter bundlerextension list
```

12.4 Additional resources on creating and distributing packages

Of course, in addition to the files listed, there are number of other files one needs to build a proper package. Here are some good resources: - The Hitchhiker's Guide to Packaging - Repository Structure and Python by Kenneth Reitz

How you distribute them, too, is important: - Packaging and Distributing Projects - conda: Building packages

12.5 Example - Server extension

12.5.1 Creating a Python package with a server extension

Here is an example of a python module which contains a server extension directly on itself. It has this directory structure:

```
- setup.py
- MANIFEST.in
- my_module/
  - __init__.py
```

12.5.2 Defining the server extension

This example shows that the server extension and its `load_jupyter_server_extension` function are defined in the `__init__.py` file.

my_module/__init__.py

```
def _jupyter_server_extension_paths():
    return [{
        "module": "my_module"
    }]


def load_jupyter_server_extension(nbapp):
    nbapp.log.info("my module enabled!")
```

12.5.3 Install and enable the server extension

Which a user can install with:

```
jupyter serverextension enable --py my_module [--sys-prefix]
```

12.6 Example - Server extension and nbextension

12.6.1 Creating a Python package with a server extension and nbextension

Here is another server extension, with a front-end module. It assumes this directory structure:

```
- setup.py
- MANIFEST.in
- my_fancy_module/
  - __init__.py
  - static/
    index.js
```

12.6.2 Defining the server extension and nbextension

This example again shows that the server extension and its `load_jupyter_server_extension` function are defined in the `__init__.py` file. This time, there is also a function `_jupyter_nbextension_paths` for the nbextension.

my_fancy_module/__init__.py

```
def _jupyter_server_extension_paths():
    return [{
        "module": "my_fancy_module"
    }]

# Jupyter Extension points
def _jupyter_nbextension_paths():
    return [dict(
        section="notebook",
        # the path is relative to the `my_fancy_module` directory
        src="static",
        # directory in the `nbextension/` namespace
        dest="my_fancy_module",
        # _also_ in the `nbextension/` namespace
        require="my_fancy_module/index")]

def load_jupyter_server_extension(nbapp):
    nbapp.log.info("my module enabled!")
```

12.6.3 Install and enable the server extension and nbextension

The user can install and enable the extensions with the following set of commands:

```
jupyter nbextension install --py my_fancy_module [--sys-prefix|--user]
jupyter nbextension enable --py my_fancy_module [--sys-prefix|--system]
jupyter serverextension enable --py my_fancy_module [--sys-prefix|--system]
```

12.6.4 Automatically enabling a server extension and nbextension

New in Notebook 5.3

Server extensions and nbextensions can be installed and enabled without any user intervention or post-install scripts beyond `<package manager> install <extension package name>`

In addition to the `my_fancy_module` file tree, assume:

```
jupyter-config/
├── jupyter_notebook_config.d/
│   └── my_fancy_module.json
└── nbconfig/
    └── notebook.d/
        └── my_fancy_module.json
```

jupyter-config/jupyter_notebook_config.d/my_fancy_module.json

```
{
  "NotebookApp": {
    "nbserver_extensions": {
      "my_fancy_module": true
    }
  }
}
```

jupyter-config/nbconfig/notebook.d/my_fancy_module.json

```
{
  "load_extensions": {
    "my_fancy_module/index": true
  }
}
```

Put all of them in place via:

setup.py

```
import setuptools

setuptools.setup(
    name="MyFancyModule",
    ...
    include_package_data=True,
    data_files=[
        # like `jupyter nbextension install --sys-prefix`
        ("share/jupyter/nbextensions/my_fancy_module", [
            "my_fancy_module/static/index.js",
        ]),
        # like `jupyter nbextension enable --sys-prefix`
        ("etc/jupyter/nbconfig/notebook.d", [
            "jupyter-config/nbconfig/notebook.d/my_fancy_module.json"
        ]),
        # like `jupyter serverextension enable --sys-prefix`
        ("etc/jupyter/jupyter_notebook_config.d", [
            "jupyter-config/jupyter_notebook_config.d/my_fancy_module.json"
        ])
    ],
    ...
    zip_safe=False
)
```

and last, but not least:

MANIFEST.in

```
recursive-include jupyter-config *.json
recursive-include my_fancy_module/static *.js
```

As most package managers will only modify their environment, the eventual configuration will be as if the user had typed:

```
jupyter nbextension install --py my_fancy_module --sys-prefix
jupyter nbextension enable --py my_fancy_module --sys-prefix
jupyter serverextension enable --py my_fancy_module --sys-prefix
```

If a user manually `disable`s an extension, that configuration will override the bundled package configuration.

When automagical install fails

Note this can still fail in certain situations with `pip`, requiring manual use of `install` and `enable` commands.

Non-python-specific package managers (e.g. `conda`, `apt`) may choose not to implement the above behavior at the `setup.py` level, having more ways to put data files in various places at build time.

12.7 Example - Bundler extension

12.7.1 Creating a Python package with a bundlerextension

Here is a bundler extension that adds a *Download as -> Notebook Tarball (tar.gz)* option to the notebook *File* menu. It assumes this directory structure:

```
- setup.py
- MANIFEST.in
- my_tarball_bundler/
  - __init__.py
```

12.7.2 Defining the bundler extension

This example shows that the bundler extension and its `bundle` function are defined in the `__init__.py` file.

my_tarball_bundler/__init__.py

```
import tarfile
import io
import os
import nbformat

def _jupyter_bundlerextension_paths():
    """Declare bundler extensions provided by this package."""
    return [{
        # unique bundler name
        "name": "tarball_bundler",
        # module containing bundle function
        "module_name": "my_tarball_bundler",
        # human-readable menu item label
        "label" : "Notebook Tarball (tar.gz)",
        # group under 'deploy' or 'download' menu
        "group" : "download",
    }]


def bundle(handler, model):
    """Create a compressed tarball containing the notebook document.

    Parameters
    ----------
    handler : tornado.web.RequestHandler
        Handler that serviced the bundle request
    model : dict
        Notebook model from the configured ContentManager
    """
    notebook_filename = model['name']
    notebook_content = nbformat.writes(model['content']).encode('utf-8')
    notebook_name = os.path.splitext(notebook_filename)[0]
```

(continues on next page)

(continued from previous page)

```
    tar_filename = '{}.tar.gz'.format(notebook_name)

    info = tarfile.TarInfo(notebook_filename)
    info.size = len(notebook_content)

    with io.BytesIO() as tar_buffer:
        with tarfile.open(tar_filename, "w:gz", fileobj=tar_buffer) as tar:
            tar.addfile(info, io.BytesIO(notebook_content))

        # Set headers to trigger browser download
        handler.set_header('Content-Disposition',
                           'attachment; filename="{}"'.format(tar_filename))
        handler.set_header('Content-Type', 'application/gzip')

        # Return the buffer value as the response
        handler.finish(tar_buffer.getvalue())
```

See *Extending the Notebook* for more documentation about writing nbextensions, server extensions, and bundler extensions.

CHAPTER

THIRTEEN

EXTENDING THE NOTEBOOK

Certain subsystems of the notebook server are designed to be extended or overridden by users. These documents explain these systems, and show how to override the notebook's defaults with your own custom behavior.

13.1 Contents API

The Jupyter Notebook web application provides a graphical interface for creating, opening, renaming, and deleting files in a virtual filesystem.

The `ContentsManager` class defines an abstract API for translating these interactions into operations on a particular storage medium. The default implementation, `FileContentsManager`, uses the local filesystem of the server for storage and straightforwardly serializes notebooks into JSON. Users can override these behaviors by supplying custom subclasses of ContentsManager.

This section describes the interface implemented by ContentsManager subclasses. We refer to this interface as the **Contents API**.

13.1.1 Data Model

Filesystem Entities

ContentsManager methods represent virtual filesystem entities as dictionaries, which we refer to as **models**.

Models may contain the following entries:

Key	Type	Info
name	unicode	Basename of the entity.
path	unicode	Full (*API-style*) path to the entity.
type	unicode	The entity type. One of `"notebook"`, `"file"` or `"directory"`.
created	datetime	Creation date of the entity.
last_modified	datetime	Last modified date of the entity.
content	variable	The "content" of the entity. (*See Below*)
mimetype	unicode or `None`	The mimetype of `content`, if any. (*See Below*)
format	unicode or `None`	The format of `content`, if any. (*See Below*)

Certain model fields vary in structure depending on the `type` field of the model. There are three model types: **notebook**, **file**, and **directory**.

- **`notebook` models**
  - The `format` field is always `"json"`.

  - The `mimetype` field is always `None`.
  - The `content` field contains a `nbformat.notebooknode.NotebookNode` representing the .ipynb file represented by the model. See the NBFormat documentation for a full description.

- **`file` models**
  - The `format` field is either `"text"` or `"base64"`.
  - The `mimetype` field can be any mimetype string, but defaults to `text/plain` for text-format models and `application/octet-stream` for base64-format models. For files with unknown mime types (e.g. unknown file extensions), this field may be *None*.
  - The `content` field is always of type `unicode`. For text-format file models, `content` simply contains the file's bytes after decoding as UTF-8. Non-text (`base64`) files are read as bytes, base64 encoded, and then decoded as UTF-8.

- **`directory` models**
  - The `format` field is always `"json"`.
  - The `mimetype` field is always `None`.
  - The `content` field contains a list of *content-free* models representing the entities in the directory.

Note: In certain circumstances, we don't need the full content of an entity to complete a Contents API request. In such cases, we omit the `content`, and `format` keys from the model. The default values for the `mimetype` field will might also not be evaluated, in which case it will be set as *None*. This reduced reply most commonly occurs when listing a directory, in which circumstance we represent files within the directory as content-less models to avoid having to recursively traverse and serialize the entire filesystem.

Sample Models

```
# Notebook Model with Content
{
    'content': {
        'metadata': {},
        'nbformat': 4,
        'nbformat_minor': 0,
        'cells': [
            {
                'cell_type': 'markdown',
                'metadata': {},
                'source': 'Some **Markdown**',
            },
        ],
    },
    'created': datetime(2015, 7, 25, 19, 50, 19, 19865),
    'format': 'json',
    'last_modified': datetime(2015, 7, 25, 19, 50, 19, 19865),
    'mimetype': None,
    'name': 'a.ipynb',
    'path': 'foo/a.ipynb',
    'type': 'notebook',
    'writable': True,
}
```

(continues on next page)

(continued from previous page)

```
# Notebook Model without Content
{
    'content': None,
    'created': datetime.datetime(2015, 7, 25, 20, 17, 33, 271931),
    'format': None,
    'last_modified': datetime.datetime(2015, 7, 25, 20, 17, 33, 271931),
    'mimetype': None,
    'name': 'a.ipynb',
    'path': 'foo/a.ipynb',
    'type': 'notebook',
    'writable': True
}
```

API Paths

ContentsManager methods represent the locations of filesystem resources as **API-style paths**. Such paths are interpreted as relative to the root directory of the notebook server. For compatibility across systems, the following guarantees are made:

- Paths are always `unicode`, not `bytes`.
- Paths are not URL-escaped.
- Paths are always forward-slash (/) delimited, even on Windows.
- Leading and trailing slashes are stripped. For example, `/foo/bar/buzz/` becomes `foo/bar/buzz`.
- The empty string ("") represents the root directory.

13.1.2 Writing a Custom ContentsManager

The default ContentsManager is designed for users running the notebook as an application on a personal computer. It stores notebooks as .ipynb files on the local filesystem, and it maps files and directories in the Notebook UI to files and directories on disk. It is possible to override how notebooks are stored by implementing your own custom subclass of `ContentsManager`. For example, if you deploy the notebook in a context where you don't trust or don't have access to the filesystem of the notebook server, it's possible to write your own ContentsManager that stores notebooks and files in a database.

Required Methods

A minimal complete implementation of a custom `ContentsManager` must implement the following methods:

`ContentsManager.get`(path[, content, type, ...])	Get a file or directory model.
`ContentsManager.save`(model, path)	Save a file or directory model to path.
`ContentsManager.delete_file`(path)	Delete the file or directory at path.
`ContentsManager.rename_file`(old_path, new_path)	Rename a file or directory.
`ContentsManager.file_exists`([path])	Does a file exist at the given path?
`ContentsManager.dir_exists`(path)	Does a directory exist at the given path?
`ContentsManager.is_hidden`(path)	Is path a hidden directory or file?

You may be required to specify a Checkpoints object, as the default one, `FileCheckpoints`, could be incompatible

with your custom ContentsManager.

Chunked Saving

The contents API allows for "chunked" saving of files, i.e. saving/transmitting in partial pieces:

- This can only be used when the `type` of the model is `file`.
- The model should be as otherwise expected for `save()`, with an added field `chunk`.
- The value of `chunk` should be an integer starting at `1`, and incrementing for each subsequent chunk, except for the final chunk, which should be indicated with a value of `-1`.
- The model returned from using `save()` with `chunk` should be treated as unreliable for all chunks except the final one.
- Any interaction with a file being saved in a chunked manner is unreliable until the final chunk has been saved. This includes directory listings.

13.1.3 Customizing Checkpoints

Customized Checkpoint definitions allows behavior to be altered and extended.

The `Checkpoints` and `GenericCheckpointsMixin` classes (from `notebook.services.contents.checkpoints`) have reusable code and are intended to be used together, but require the following methods to be implemented.

`Checkpoints.rename_checkpoint`(checkpoint_id, ...)	Rename a single checkpoint from old_path to new_path.
`Checkpoints.list_checkpoints`(path)	Return a list of checkpoints for a given file
`Checkpoints.delete_checkpoint`(checkpoint_id, ...)	delete a checkpoint for a file
`GenericCheckpointsMixin.create_file_checkpoint`(...)	Create a checkpoint of the current state of a file
`GenericCheckpointsMixin.create_notebook_checkpoint`(nb, ...)	Create a checkpoint of the current state of a file
`GenericCheckpointsMixin.get_file_checkpoint`(...)	Get the content of a checkpoint for a non-notebook file.
`GenericCheckpointsMixin.get_notebook_checkpoint`(...)	Get the content of a checkpoint for a notebook.

No-op example

Here is an example of a no-op checkpoints object - note the mixin comes first. The docstrings indicate what each method should do or return for a more complete implementation.

```
class NoOpCheckpoints(GenericCheckpointsMixin, Checkpoints):
    """requires the following methods:"""
    def create_file_checkpoint(self, content, format, path):
        """ -> checkpoint model"""
    def create_notebook_checkpoint(self, nb, path):
        """ -> checkpoint model"""
    def get_file_checkpoint(self, checkpoint_id, path):
```

(continues on next page)

(continued from previous page)

```
        """ -> {'type': 'file', 'content': <str>, 'format': {'text', 'base64'}}"""
    def get_notebook_checkpoint(self, checkpoint_id, path):
        """ -> {'type': 'notebook', 'content': <output of nbformat.read>}"""
    def delete_checkpoint(self, checkpoint_id, path):
        """deletes a checkpoint for a file"""
    def list_checkpoints(self, path):
        """returns a list of checkpoint models for a given file,
        default just does one per file
        """
        return []
    def rename_checkpoint(self, checkpoint_id, old_path, new_path):
        """renames checkpoint from old path to new path"""
```

See `GenericFileCheckpoints` in `notebook.services.contents.filecheckpoints` for a more complete example.

13.1.4 Testing

`notebook.services.contents.tests` includes several test suites written against the abstract Contents API. This means that an excellent way to test a new ContentsManager subclass is to subclass our tests to make them use your ContentsManager.

Note: PGContents is an example of a complete implementation of a custom `ContentsManager`. It stores notebooks and files in PostgreSQL and encodes directories as SQL relations. PGContents also provides an example of how to re-use the notebook's tests.

13.2 File save hooks

You can configure functions that are run whenever a file is saved. There are two hooks available:

- `ContentsManager.pre_save_hook` runs on the API path and model with content. This can be used for things like stripping output that people don't like adding to VCS noise.
- `FileContentsManager.post_save_hook` runs on the filesystem path and model without content. This could be used to commit changes after every save, for instance.

They are both called with keyword arguments:

```
pre_save_hook(model=model, path=path, contents_manager=cm)
post_save_hook(model=model, os_path=os_path, contents_manager=cm)
```

13.2.1 Examples

These can both be added to `jupyter_notebook_config.py`.

A pre-save hook for stripping output:

```
def scrub_output_pre_save(model, **kwargs):
    """scrub output before saving notebooks"""
    # only run on notebooks
    if model['type'] != 'notebook':
        return
    # only run on nbformat v4
    if model['content']['nbformat'] != 4:
        return

    for cell in model['content']['cells']:
        if cell['cell_type'] != 'code':
            continue
        cell['outputs'] = []
        cell['execution_count'] = None

c.FileContentsManager.pre_save_hook = scrub_output_pre_save
```

A post-save hook to make a script equivalent whenever the notebook is saved (replacing the `--script` option in older versions of the notebook):

```
import io
import os
from notebook.utils import to_api_path

_script_exporter = None

def script_post_save(model, os_path, contents_manager, **kwargs):
    """convert notebooks to Python script after save with nbconvert

    replaces `jupyter notebook --script`
    """
    from nbconvert.exporters.script import ScriptExporter

    if model['type'] != 'notebook':
        return

    global _script_exporter

    if _script_exporter is None:
        _script_exporter = ScriptExporter(parent=contents_manager)

    log = contents_manager.log

    base, ext = os.path.splitext(os_path)
    script, resources = _script_exporter.from_filename(os_path)
    script_fname = base + resources.get('output_extension', '.txt')
    log.info("Saving script /%s", to_api_path(script_fname, contents_manager.root_dir))
```

(continues on next page)

(continued from previous page)

```
    with io.open(script_fname, 'w', encoding='utf-8') as f:
        f.write(script)

c.FileContentsManager.post_save_hook = script_post_save
```

This could be a simple call to `jupyter nbconvert --to script`, but spawning the subprocess every time is quite slow.

13.3 Custom request handlers

The notebook webserver can be interacted with using a well defined RESTful API. You can define custom RESTful API handlers in addition to the ones provided by the notebook. As described below, to define a custom handler you need to first write a notebook server extension. Then, in the extension, you can register the custom handler.

13.3.1 Writing a notebook server extension

The notebook webserver is written in Python, hence your server extension should be written in Python too. Server extensions, like IPython extensions, are Python modules that define a specially named load function, `load_jupyter_server_extension`. This function is called when the extension is loaded.

```
def load_jupyter_server_extension(nb_server_app):
    """
    Called when the extension is loaded.

    Args:
        nb_server_app (NotebookWebApplication): handle to the Notebook webserver␣
↪instance.
    """
    pass
```

To get the notebook server to load your custom extension, you'll need to add it to the list of extensions to be loaded. You can do this using the config system. `NotebookApp.nbserver_extensions` is a config variable which is a dictionary of strings, each a Python module to be imported, mapping to `True` to enable or `False` to disable each extension. Because this variable is notebook config, you can set it two different ways, using config files or via the command line.

For example, to get your extension to load via the command line add a double dash before the variable name, and put the Python dictionary in double quotes. If your package is "mypackage" and module is "mymodule", this would look like `jupyter notebook --NotebookApp.nbserver_extensions="{'mypackage.mymodule':True}"` . Basically the string should be Python importable.

Alternatively, you can have your extension loaded regardless of the command line args by setting the variable in the Jupyter config file. The default location of the Jupyter config file is `~/.jupyter/jupyter_notebook_config.py` (see *Configuration Overview*). Inside the config file, you can use Python to set the variable. For example, the following config does the same as the previous command line example.

```
c = get_config()
c.NotebookApp.nbserver_extensions = {
    'mypackage.mymodule': True,
}
```

Before continuing, it's a good idea to verify that your extension is being loaded. Use a print statement to print something unique. Launch the notebook server and you should see your statement printed to the console.

13.3.2 Registering custom handlers

Once you've defined a server extension, you can register custom handlers because you have a handle to the Notebook server app instance (`nb_server_app` above). However, you first need to define your custom handler. To declare a custom handler, inherit from `notebook.base.handlers.IPythonHandler`. The example below[1] is a Hello World handler:

```
from notebook.base.handlers import IPythonHandler

class HelloWorldHandler(IPythonHandler):
    def get(self):
        self.finish('Hello, world!')
```

The Jupyter Notebook server use Tornado as its web framework. For more information on how to implement request handlers, refer to the Tornado documentation on the matter.

After defining the handler, you need to register the handler with the Notebook server. See the following example:

```
web_app = nb_server_app.web_app
host_pattern = '.*$'
route_pattern = url_path_join(web_app.settings['base_url'], '/hello')
web_app.add_handlers(host_pattern, [(route_pattern, HelloWorldHandler)])
```

Putting this together with the extension code, the example looks like the following:

```
from notebook.utils import url_path_join
from notebook.base.handlers import IPythonHandler

class HelloWorldHandler(IPythonHandler):
    def get(self):
        self.finish('Hello, world!')

def load_jupyter_server_extension(nb_server_app):
    """
    Called when the extension is loaded.

    Args:
        nb_server_app (NotebookWebApplication): handle to the Notebook webserver␣
↪instance.
    """
    web_app = nb_server_app.web_app
    host_pattern = '.*$'
    route_pattern = url_path_join(web_app.settings['base_url'], '/hello')
    web_app.add_handlers(host_pattern, [(route_pattern, HelloWorldHandler)])
```

13.4 Extra Parameters and authentication

Here is a quick rundown of what you need to know to pass extra parameters to the handler and enable authentication:

- extra arguments to the `__init__` constructor are given in a dictionary after the handler class in `add_handlers`:

```
class HelloWorldHandler(IPythonHandler):

    def __init__(self, *args, **kwargs):
        self.extra = kwargs.pop('extra')
        ...

def load_jupyter_server_extension(nb_server_app):

    ...

    web_app.add_handlers(host_pattern,
        [
           (route_pattern, HelloWorldHandler, {"extra": nb_server_app.extra})
        ])
```

All handler methods that require authentication _MUST_ be decorated with `@tornado.web.authenticated`:

```
from tornado import web

class HelloWorldHandler(IPythonHandler):

    ...

    @web.authenticated
    def  get(self, *args, **kwargs):
         ...

    @web.authenticated
    def  post(self, *args, **kwargs):
         ...
```

References:

1. Peter Parente's Mindtrove

13.5 Custom front-end extensions

This describes the basic steps to write a JavaScript extension for the Jupyter notebook front-end. This allows you to customize the behaviour of the various pages like the dashboard, the notebook, or the text editor.

13.5.1 The structure of a front-end extension

Note: The notebook front-end and Javascript API are not stable, and are subject to a lot of changes. Any extension written for the current notebook is almost guaranteed to break in the next release.

A front-end extension is a JavaScript file that defines an AMD module which exposes at least a function called `load_ipython_extension`, which takes no arguments. We will not get into the details of what each of these terms consists of yet, but here is the minimal code needed for a working extension:

```
// file my_extension/main.js

define(function(){

    function load_ipython_extension(){
        console.info('this is my first extension');
    }

    return {
        load_ipython_extension: load_ipython_extension
    };
});
```

Note: Although for historical reasons the function is called `load_ipython_extension`, it does apply to the Jupyter notebook in general, and will work regardless of the kernel in use.

If you are familiar with JavaScript, you can use this template to require any Jupyter module and modify its configuration, or do anything else in client-side Javascript. Your extension will be loaded at the right time during the notebook page initialisation for you to set up a listener for the various events that the page can trigger.

You might want access to the current instances of the various Jupyter notebook components on the page, as opposed to the classes defined in the modules. The current instances are exposed by a module named `base/js/namespace`. If you plan on accessing instances on the page, you should `require` this module rather than accessing the global variable `Jupyter`, which will be removed in future. The following example demonstrates how to access the current notebook instance:

```
// file my_extension/main.js

define([
    'base/js/namespace'
], function(
    Jupyter
) {
    function load_ipython_extension() {
        console.log(
            'This is the current notebook application instance:',
```

(continues on next page)

(continued from previous page)

```
            Jupyter.notebook
        );
    }

    return {
        load_ipython_extension: load_ipython_extension
    };
});
```

13.5.2 Modifying key bindings

One of the abilities of extensions is to modify key bindings, although once again this is an API which is not guaranteed to be stable. However, custom key bindings are frequently requested, and are helpful to increase accessibility, so in the following we show how to access them.

Here is an example of an extension that will unbind the shortcut `0,0` in command mode, which normally restarts the kernel, and bind `0,0,0` in its place:

```
// file my_extension/main.js

define([
    'base/js/namespace'
], function(
    Jupyter
) {

    function load_ipython_extension() {
        Jupyter.keyboard_manager.command_shortcuts.remove_shortcut('0,0');
        Jupyter.keyboard_manager.command_shortcuts.add_shortcut('0,0,0', 'jupyter-
↪notebook:restart-kernel');
    }

    return {
        load_ipython_extension: load_ipython_extension
    };
});
```

Note: The standard keybindings might not work correctly on non-US keyboards. Unfortunately, this is a limitation of browser implementations and the status of keyboard event handling on the web in general. We appreciate your feedback if you have issues binding keys, or have any ideas to help improve the situation.

You can see that I have used the **action name** `jupyter-notebook:restart-kernel` to bind the new shortcut. There is no API yet to access the list of all available *actions*, though the following in the JavaScript console of your browser on a notebook page should give you an idea of what is available:

```
Object.keys(require('base/js/namespace').actions._actions);
```

In this example, we changed a keyboard shortcut in **command mode**; you can also customize keyboard shortcuts in **edit mode**. However, most of the keyboard shortcuts in edit mode are handled by CodeMirror, which supports custom key bindings via a completely different API.

13.5.3 Defining and registering your own actions

As part of your front-end extension, you may wish to define actions, which can be attached to toolbar buttons, or called from the command palette. Here is an example of an extension that defines an (not very useful!) action to show an alert, and adds a toolbar button using the full action name:

```
// file my_extension/main.js

define([
    'base/js/namespace'
], function(
    Jupyter
) {
    function load_ipython_extension() {

        var handler = function () {
            alert('this is an alert from my_extension!');
        };

        var action = {
            icon: 'fa-comment-o', // a font-awesome class used on buttons, etc
            help    : 'Show an alert',
            help_index : 'zz',
            handler : handler
        };
        var prefix = 'my_extension';
        var action_name = 'show-alert';

        var full_action_name = Jupyter.actions.register(action, action_name, prefix); //␣
↪returns 'my_extension:show-alert'
        Jupyter.toolbar.add_buttons_group([full_action_name]);
    }

    return {
        load_ipython_extension: load_ipython_extension
    };
});
```

Every action needs a name, which, when joined with its prefix to make the full action name, should be unique. Built-in actions, like the `jupyter-notebook:restart-kernel` we bound in the earlier *Modifying key bindings* example, use the prefix `jupyter-notebook`. For actions defined in an extension, it makes sense to use the extension name as the prefix. For the action name, the following guidelines should be considered:

- First pick a noun and a verb for the action. For example, if the action is "restart kernel," the verb is "restart" and the noun is "kernel".
- Omit terms like "selected" and "active" by default, so "delete-cell", rather than "delete-selected-cell". Only provide a scope like "-all-" if it is other than the default "selected" or "active" scope.
- If an action has a secondary action, separate the secondary action with "-and-", so "restart-kernel-and-clear-output".
- Use above/below or previous/next to indicate spatial and sequential relationships.
- Don't ever use before/after as they have a temporal connotation that is confusing when used in a spatial context.
- For dialogs, use a verb that indicates what the dialog will accomplish, such as "confirm-restart-kernel".

13.5.4 Installing and enabling extensions

You can install your nbextension with the command:

```
jupyter nbextension install path/to/my_extension/ [--user|--sys-prefix]
```

The default installation is system-wide. You can use `--user` to do a per-user installation, or `--sys-prefix` to install to Python's prefix (e.g. in a virtual or conda environment). Where my_extension is the directory containing the Javascript files. This will copy it to a Jupyter data directory (the exact location is platform dependent - see jupyter_path).

For development, you can use the `--symlink` flag to symlink your extension rather than copying it, so there's no need to reinstall after changes.

To use your extension, you'll also need to **enable** it, which tells the notebook interface to load it. You can do that with another command:

```
jupyter nbextension enable my_extension/main [--sys-prefix][--section='common']
```

The argument refers to the Javascript module containing your `load_ipython_extension` function, which is `my_extension/main.js` in this example. The `--section='common'` argument will affect all pages, by default it will be loaded on the notebook view only. There is a corresponding `disable` command to stop using an extension without uninstalling it.

Changed in version 4.2: Added `--sys-prefix` argument

13.5.5 Kernel Specific extensions

Warning: This feature serves as a stopgap for kernel developers who need specific JavaScript injected onto the page. The availability and API are subject to change at anytime.

It is possible to load some JavaScript on the page on a per kernel basis. Be aware that doing so will make the browser page reload without warning as soon as the user switches the kernel without notice.

If you, a kernel developer, need a particular piece of JavaScript to be loaded on a "per kernel" basis, such as:

- if you are developing a CodeMirror mode for your language
- if you need to enable some specific debugging options

your `kernelspecs` are allowed to contain a `kernel.js` file that defines an AMD module. The AMD module should define an *onload* function that will be called when the kernelspec loads, such as:

- when you load a notebook that uses your kernelspec
- change the active kernelspec of a notebook to your kernelspec.

Note that adding a *kernel.js* to your kernelspec will add an unexpected side effect to changing a kernel in the notebook. As it is impossible to "unload" JavaScript, any attempt to change the kernelspec again will save the current notebook and reload the page without confirmations.

Here is an example of `kernel.js`:

```
define(function(){
  return {onload: function(){
    console.info('Kernel specific javascript loaded');

```

(continues on next page)

(continued from previous page)

```
    // do more things here, like define a codemirror mode

  }}

});
```

13.6 Customize keymaps

Note: Declarative Custom Keymaps is a provisional feature with unstable API which is not guaranteed to be kept in future versions of the notebook, and can be removed or changed without warnings.

The notebook shortcuts that are defined by jupyter both in edit mode and command mode are configurable in the frontend configuration file `~/.jupyter/nbconfig/notebook.json`. The modification of keyboard shortcuts suffers from several limitations, mainly that your Browser and OS might prevent certain shortcuts from working correctly. If this is the case, there is unfortunately not much that can be done. The second issue can arise with keyboards that have a layout different than US English. Again, even if we are aware of the issue, there is not much that can be done.

Shortcuts are also limited by the underlying library that handles code and text editing: CodeMirror. If some keyboard shortcuts are conflicting, the method described below might not work to create new keyboard shortcuts, especially in the `edit` mode of the notebook.

The 4 sections of interest in `~/.jupyter/nbconfig/notebook.json` are the following:

- `keys.command.unbind`
- `keys.edit.unbind`
- `keys.command.bind`
- `keys.edit.bind`

The first two sections describe which default keyboard shortcuts not to register at notebook startup time. These are mostly useful if you need to `unbind` a default keyboard shortcut before binding it to a new `command`.

The first two sections apply respectively to the `command` and `edit` mode of the notebook. They take a list of shortcuts to `unbind`.

For example, to unbind the shortcut to split a cell at the position of the cursor (`Ctrl-Shift-Minus`) use the following:

```
// file ~/.jupyter/nbconfig/notebook.json

{
  "keys": {
    "edit": {
      "unbind": [
        "Ctrl-Shift-Minus"
      ]
    },
  },
}
```

The last two sections describe which new keyboard shortcuts to register at notebook startup time and which actions they trigger.

The last two sections apply respectively to the `command` and `edit` mode of the notebook. They take a dictionary with shortcuts as `keys` and `commands` name as value.

For example, to bind the shortcut `G,G,G` (Press G three time in a row) in command mode to the command that restarts the kernel and runs all cells, use the following:

```
// file ~/.jupyter/nbconfig/notebook.json

{
  "keys": {
    "command": {
        "bind": {
            "G,G,G":"jupyter-notebook:restart-kernel-and-run-all-cells"
        }
    }
  },
}
```

The name of the available `commands` can be find by hovering over the right end of a row in the command palette.

13.7 Custom bundler extensions

The notebook server supports the writing of *bundler extensions* that transform, package, and download/deploy notebook files. As a developer, you need only write a single Python function to implement a bundler. The notebook server automatically generates a *File -> Download as* or *File -> Deploy as* menu item in the notebook front-end to trigger your bundler.

Here are some examples of what you can implement using bundler extensions:

- Convert a notebook file to a HTML document and publish it as a post on a blog site
- Create a snapshot of the current notebook environment and bundle that definition plus notebook into a zip download
- Deploy a notebook as a standalone, interactive dashboard

To implement a bundler extension, you must do all of the following:

- Declare bundler extension metadata in your Python package
- Write a *bundle* function that responds to bundle requests
- Instruct your users on how to enable/disable your bundler extension

The following sections describe these steps in detail.

13.7.1 Declaring bundler metadata

You must provide information about the bundler extension(s) your package provides by implementing a *_jupyter_bundlerextensions_paths* function. This function can reside anywhere in your package so long as it can be imported when enabling the bundler extension. (See *Enabling/disabling bundler extensions*.)

```
# in mypackage.hello_bundler

def _jupyter_bundlerextension_paths():
    """Example "hello world" bundler extension"""
```

(continues on next page)

(continued from previous page)

```
    return [{
        'name': 'hello_bundler',                    # unique bundler name
        'label': 'Hello Bundler',                   # human-readable menu item label
        'module_name': 'mypackage.hello_bundler',   # module containing bundle()
        'group': 'deploy'                           # group under 'deploy' or 'download'␣
↪menu
    }]
```

Note that the return value is a list. By returning multiple dictionaries in the list, you allow users to enable/disable sets of bundlers all at once.

13.7.2 Writing the *bundle* function

At runtime, a menu item with the given label appears either in the *File -> Deploy as* or *File -> Download as* menu depending on the *group* value in your metadata. When a user clicks the menu item, a new browser tab opens and notebook server invokes a *bundle* function in the *module_name* specified in the metadata.

You must implement a *bundle* function that matches the signature of the following example:

```
# in mypackage.hello_bundler

def bundle(handler, model):
    """Transform, convert, bundle, etc. the notebook referenced by the given
    model.

    Then issue a Tornado web response using the `handler` to redirect
    the user's browser, download a file, show a HTML page, etc. This function
    must finish the handler response before returning either explicitly or by
    raising an exception.

    Parameters
    ----------
    handler : tornado.web.RequestHandler
        Handler that serviced the bundle request
    model : dict
        Notebook model from the configured ContentManager
    """
    handler.finish('I bundled {}!'.format(model['path']))
```

Your *bundle* function is free to do whatever it wants with the request and respond in any manner. For example, it may read additional query parameters from the request, issue a redirect to another site, run a local process (e.g., *nbconvert*), make a HTTP request to another service, etc.

The caller of the *bundle* function is *@tornado.gen.coroutine* decorated and wraps its call with *torando.gen.maybe_future*. This behavior means you may handle the web request synchronously, as in the example above, or asynchronously using *@tornado.gen.coroutine* and *yield*, as in the example below.

```
from tornado import gen

@gen.coroutine
def bundle(handler, model):
  # simulate a long running IO op (e.g., deploying to a remote host)
```

(continues on next page)

(continued from previous page)

```
    yield gen.sleep(10)

    # now respond
    handler.finish('I spent 10 seconds bundling {}!'.format(model['path']))
```

You should prefer the second, asynchronous approach when your bundle operation is long-running and would otherwise block the notebook server main loop if handled synchronously.

For more details about the data flow from menu item click to bundle function invocation, see *Bundler invocation details*.

13.7.3 Enabling/disabling bundler extensions

The notebook server includes a command line interface (CLI) for enabling and disabling bundler extensions.

You should document the basic commands for enabling and disabling your bundler. One possible command for enabling the *hello_bundler* example is the following:

```
jupyter bundlerextension enable --py mypackage.hello_bundler --sys-prefix
```

The above updates the notebook configuration file in the current conda/virtualenv environment (*–sys-prefix*) with the metadata returned by the *mypackage.hellow_bundler._jupyter_bundlerextension_paths* function.

The corresponding command to later disable the bundler extension is the following:

```
jupyter bundlerextension disable --py mypackage.hello_bundler --sys-prefix
```

For more help using the *bundlerextension* subcommand, run the following.

```
jupyter bundlerextension --help
```

The output describes options for listing enabled bundlers, configuring bundlers for single users, configuring bundlers system-wide, etc.

13.7.4 Example: IPython Notebook bundle (.zip)

The *hello_bundler* example in this documentation is simplistic in the name of brevity. For more meaningful examples, see *notebook/bundler/zip_bundler.py* and *notebook/bundler/tarball_bundler.py*. You can enable them to try them like so:

```
jupyter bundlerextension enable --py notebook.bundler.zip_bundler --sys-prefix
jupyter bundlerextension enable --py notebook.bundler.tarball_bundler --sys-prefix
```

13.7.5 Bundler invocation details

Support for bundler extensions comes from Python modules in *notebook/bundler* and JavaScript in *notebook/static/notebook/js/menubar.js*. The flow of data between the various components proceeds roughly as follows:

1. User opens a notebook document
2. Notebook front-end JavaScript loads notebook configuration
3. Bundler front-end JS creates menu items for all bundler extensions in the config
4. User clicks a bundler menu item

5. JS click handler opens a new browser window/tab to *<notebook base_url>/bundle/<path/to/notebook>?bundler=<name>* (i.e., a HTTP GET request)
6. Bundle handler validates the notebook path and bundler *name*
7. Bundle handler delegates the request to the *bundle* function in the bundler's *module_name*
8. *bundle* function finishes the HTTP request

CHAPTER

FOURTEEN

CONTRIBUTING TO THE JUPYTER NOTEBOOK

If you're reading this section, you're probably interested in contributing to Jupyter. Welcome and thanks for your interest in contributing!

Please take a look at the Contributor documentation, familiarize yourself with using the Jupyter Notebook, and introduce yourself on the mailing list and share what area of the project you are interested in working on.

14.1 General Guidelines

For general documentation about contributing to Jupyter projects, see the Project Jupyter Contributor Documentation.

14.2 Setting Up a Development Environment

14.2.1 Installing Node.js and npm

Building the Notebook from its GitHub source code requires some tools to create and minify JavaScript components and the CSS, specifically Node.js and Node's package manager, `npm`. It should be node version 6.0.

If you use `conda`, you can get them with:

```
conda install -c conda-forge nodejs
```

If you use Homebrew on Mac OS X:

```
brew install node
```

Installation on Linux may vary, but be aware that the *nodejs* or *npm* packages included in the system package repository may be too old to work properly.

You can also use the installer from the Node.js website.

14.2.2 Installing the Jupyter Notebook

Once you have installed the dependencies mentioned above, use the following steps:

```
pip install --upgrade setuptools pip
git clone https://github.com/jupyter/notebook
cd notebook
pip install -e .
```

If you are using a system-wide Python installation and you only want to install the notebook for you, you can add `--user` to the install commands.

Once you have done this, you can launch the master branch of Jupyter notebook from any directory in your system with:

```
jupyter notebook
```

14.2.3 Verification

While running the notebook, select one of your notebook files (the file will have the extension `.ipynb`). In the top tab you will click on "Help" and then click on "About". In the pop window you will see information about the version of Jupyter that you are running. You will see "The version of the notebook server is:". If you are working in development mode, you will see that your version of Jupyter notebook will include the word "dev". If it does not include the word "dev", you are currently not working in development mode and should follow the steps below to uninstall and reinstall Jupyter.

14.2.4 Troubleshooting the Installation

If you do not see that your Jupyter Notebook is not running on dev mode, it's possible that you are running other instances of Jupyter Notebook. You can try the following steps:

1. Uninstall all instances of the notebook package. These include any installations you made using pip or conda.
2. Run `python3 -m pip install -e .` in the notebook repository to install the notebook from there.
3. Run `npm run build` to make sure the Javascript and CSS are updated and compiled.
4. Launch with `python3 -m notebook --port 8989`, and check that the browser is pointing to `localhost:8989` (rather than the default 8888). You don't necessarily have to launch with port 8989, as long as you use a port that is neither the default nor in use, then it should be fine.
5. Verify the installation with the steps in the previous section.

14.2.5 Rebuilding JavaScript and CSS

There is a build step for the JavaScript and CSS in the notebook. To make sure that you are working with up-to-date code, you will need to run this command whenever there are changes to JavaScript or LESS sources:

```
npm run build
```

IMPORTANT: Don't forget to run `npm run build` after switching branches. When switching between branches of different versions (e.g. `4.x` and `master`), run `pip install -e .` . If you have tried the above and still find that the notebook is not reflecting the current source code, try cleaning the repo with `git clean -xfd` and reinstalling with `pip install -e .`.

Development Tip

When doing development, you can use this command to automatically rebuild JavaScript and LESS sources as they are modified:

```
npm run build:watch
```

Git Hooks

If you want to automatically update dependencies and recompile JavaScript and CSS after checking out a new commit, you can install post-checkout and post-merge hooks which will do it for you:

```
git-hooks/install-hooks.sh
```

See `git-hooks/README.md` for more details.

14.3 Running Tests

14.3.1 Python Tests

Install dependencies:

```
pip install -e '.[test]'
```

To run the Python tests, use:

```
pytest
```

If you want coverage statistics as well, you can run:

```
py.test --cov notebook -v --pyargs notebook
```

14.3.2 JavaScript Tests

To run the JavaScript tests, you will need to have PhantomJS and CasperJS installed:

```
npm install -g casperjs phantomjs-prebuilt
```

Then, to run the JavaScript tests:

```
python -m notebook.jstest [group]
```

where `[group]` is an optional argument that is a path relative to `notebook/tests/`. For example, to run all tests in `notebook/tests/notebook`:

```
python -m notebook.jstest notebook
```

or to run just `notebook/tests/notebook/deletecell.js`:

```
python -m notebook.jstest notebook/deletecell.js
```

14.4 Building the Documentation

To build the documentation you'll need Sphinx, pandoc and a few other packages.

To install (and activate) a conda environment named `notebook_docs` containing all the necessary packages (except pandoc), use:

```
conda create -n notebook_docs pip
conda activate notebook_docs  # Linux and OS X
activate notebook_docs        # Windows
pip install .[docs]
```

If you want to install the necessary packages with `pip`, use the following instead:

```
pip install .[docs]
```

Once you have installed the required packages, you can build the docs with:

```
cd docs
make html
```

After that, the generated HTML files will be available at `build/html/index.html`. You may view the docs in your browser.

You can automatically check if all hyperlinks are still valid:

```
make linkcheck
```

Windows users can find `make.bat` in the `docs` folder.

You should also have a look at the Project Jupyter Documentation Guide.

CHAPTER

FIFTEEN

DEVELOPER FAQ

1. How do I install a prerelease version such as a beta or release candidate?

```
python -m pip install notebook --pre --upgrade
```

CHAPTER

SIXTEEN

MY NOTEBOOK

[1]:
```
def foo():
    return "foo"
```

[2]:
```
def has_ip_syntax():
    listing = !ls
    return listing
```

[4]:
```
def whatsmyname():
    return __name__
```

CHAPTER

SEVENTEEN

OTHER NOTEBOOK

This notebook just defines `bar`

[2]:
```
def bar(x):
    return "bar" * x
```