Directive Dreams

Peter S. Conrad

# Getting started

You can start working with reStructuredText just by opening your favorite text editor and starting to type—but you probably want to do more than that. For example, you might want to share information on a website or wiki, take notes, or publish an eBook. To get the most out of ReStructuredText, you’ll want to be familiar with some of these tools and topics.

## Ingredients

These are the tools and resources you will use for the recipes in this guide. Each recipe lists the tools you’ll need.

**Editor**

A reStructuredText editor is a specialized text editor that makes it easier to get the syntax right. Some code editors and text Editors include tools for working with reStructuredText. Different editors have different features—you’ll want to play with more than one to find out which you like.

**Pandoc**

Pandoc lets you convert between reStructuredText and a few other formats.

**Shared storage**

If you are collaborating with others, you’ll need a shared place to store reStructuredText files. You might also want web hosting or a blogging platform where you can publish your content when it’s complete.

**Git**

Source control can be very important for collaborating without catastrophe. You’ll need an account with a Git host, a Git client, and Git installed on your computer. Some code editors include their own Git client, which makes it easy to manage your files and changes.

**Sphinx**

Sphinx is a documentation generator or a tool that translates a set of plain text source files into various output formats, automatically producing cross-references, indices, etc. That is, if you have a directory containing a bunch of reStructuredText or Markdown documents, Sphinx can generate a series of HTML files, a PDF file (via LaTeX), man pages and much more.

## Getting started

You can install everything up front, or just install what you need as you go along. If you’re not sure, start with the following steps:

1. Install **Pandoc**.
2. Install **Python**.
3. Choose and install a **reStructuredText editor** or a **code editor**.
4. Set up **Git**.
5. Install **Sphinx**.

Once you’ve got these tools installed, you can start trying the recipes.

## Things to know

You’ll find it easiest to follow the recipes in this guide if you are comfortable with the following topics.

**The command line**

Some of the recipes in this guide involve at least some typing on the command line. You'll definitely need to use the command line a little bit when you're working with Pandoc or Sphinx, for example.

**Working with a package manager**

Some tools require installation using one of the following common package managers:

* Linux: [apt-get](https://help.ubuntu.com/community/AptGet/Howto) or [yum](http://yum.baseurl.org/)
* macOS: [Homebrew](https://brew.sh/)
* Windows: [Chocolatey](https://chocolatey.org/)

A package manager can make it easy to install several packages at once.

**Installing Python apps with Pip**

You don't need to be a Python programmer, but having the ability to install python apps with the pip or pip3 command is helpful because some editors and site generators are Python-based.

**Git**

Some of the recipes in this guide use [Git](https://git-scm.com/). Git is very powerful, but doesn’t have to be hard to use. The recipes in this guide emphasize simplicity. If you already know Git, you can choose to do things differently. If you don’t know git, take a look at [Git basics](../getting-started-git-basics).

**Other documentation tools**

You can use reStructuredText to produce websites, blog posts, books, and even a wiki. You should know something about Acrobat, Google Drive, HTML, Microsoft Office, and other tools you might use to work with content originally written in reStructuredText.

# reStructuredText

rest-basics interpreted-text explicit-markup links tables images

Like other markup languages, reStructuredText offers simple syntax for inline markup (emphasis, for example) and headings. *Interpreted text roles* and *explicit markup,* which includes *directives* are general extension mechanisms for marking up phrases and text blocks. In turn, interpreted text and Explicit markup provide the bases for links, tables, images, and a number of useful built-in directives.

This guide shows the basics of some of the available features of reStructuredText, but doesn't provide a full reference. The intention of this document is to give you a good overall understanding of reStructuredText before you try the recipes later in the book.

# Basic syntax

A paragraph, the basic block in a reStructuredText document, is a chunk of text surrounded by blank lines. In reStructuredText, as in Python, indentation is meaningful. All lines of the same paragraph must be indented the same amount, and paragraphs at the same indentation level go together.

You can create a block quote just by indenting three spaces.

Example:

This is a normal paragraph.  
  
 This is a block quote  
  
Back to normal!

To preserve line breaks, you can use line block syntax.

Example:

| J. Q. Worsencraft  
| 123 Example Street  
| Syntax, MA 02134

It's a good idea to separate elements from each other with blank lines. When in doubt, add a blank line. If you use an editor with a live preview, you will see very quickly how blank lines affect the way reStructuredText is interpreted.

## Headings

In reStructuredText, you indicate heading levels with overlining or underlining. The overlining or underlining must be at least as long as the heading text.

There is no specific hierarchy of characters that indicate heading levels, because the reStructuredText parser figures out the headings based on the structure of the document.

However, Python's style guide includes the following suggested convention:

####  
Part  
####  
  
\*\*\*\*\*\*\*  
Chapter  
\*\*\*\*\*\*\*  
  
Section  
=======  
  
Subsection  
----------  
  
Subsubsection  
^^^^^^^^^^^^^  
  
Paragraph  
"""""""""

## Lists

Bullet lists start with an asterisk, hyphen, or plus sign. If a list item requires more than one line, it should be indented.

Example:

\* List item  
\* Another list item, this one with  
 two lines

When using sub-bullets, separate them from the outer list with blank lines.

Example:

\* List item  
\* List item  
  
 \* Sub-bullet  
 \* Sub-bullet  
  
\* List continues

Numbered lists can use numerals, or you can auto-number them with the # sign.

Example:

1. This numbered list uses numerals.  
2. It is one way to do a numbered list.  
  
 a. You can do sub-lists.  
 #. Autonumbering is smart enough.  
  
#. These elements octothorpes.  
#. It is another way to do a numbered list.  
  
 \* Sub-bullets work too.  
 \* They work just like in bullet lists.  
  
7. You can force whatever numbering you like.

## Inline markup

Inline markup lets you format text inline.

Example:

\*emphasis\* (italics)  
\*\*strong emphasis\*\* (bold)  
``inline literal`` (monospace)

A few rules help to make it possible to interpret inline markup while also allowing you to use the special characters in other ways, often without escaping them.

* Inline markup can't be nested.
* The content you're marking up can't start or end with whitespace.
* Non-word characters must separate the inline markup start and end characters from the rest of the content.

Example:

Non-word characters separate \*\*this text\*\* and its  
start and end characters from the rest of the sentence.

# Interpreted text

Interpreted text lets you format a word or phrase, or mark it as a reference, index entry, or other meaningful element. The way that the word or phrase is interpreted is called its *role.* If you don't specify a role, the default role (which is initially set to :title-reference:) is used.

Syntax:

:role:`text`

Example:

:strong:`this text is bold`

Standard roles include:

* :emphasis: - usually displayed with italics, equivalent to \*emphasis\*
* :literal: - usually displayed in monospace, equivalent to ``literal``
* :code: - marks the content as code in a formal language
* :math: - allows the use of mathematical notation to display a formula
* :pep-reference: - an HTTP reference to a Python enhancement proposal (PEP)
* :rfc-reference: - an HTTP reference to an internet request for comments (RFC)
* :strong: - usually displayed with bold, equivalent to \*\*strong\*\*
* :subscript: - displays smaller characters lower than the line of text
* :superscript: - displays smaller characters higher than the line of text
* :title-reference: - describes the titles of books and other materials

The roles :emphasis:, :literal:, and :strong: are functionally the same as their inline markup equivalents.

# Explicit markup

*Explicit markup* lets you specify a purpose for a block of text, such as a footnote, comment, or cross-reference target. An explicit markup block starts with two periods and a space, usually followed by additional syntax depending on the type of block.

Syntax examples:

|  |  |
| --- | --- |
| Block | Syntax |
| Substitution definition | ..|text| replacement |
| Cross-reference target | .. \_label:  Paragraph text |
| Link target | .. \_label: URL |
| Footnote or citation | \.. [label] Footnote or citation text |

## Directives

A type of explicit markup called a *directive* marks a block of text for a particular purpose: a sidebar, topic, admonition, or image, for example. A directive includes a name, arguments, and options.

Syntax:

.. directive-name:: arguments  
 :option-name: option-value  
 :option-name: option-value  
  
 Block body

You can create your own directives with a few lines of Python programming.

### Some useful directives

reStructuredText offers many useful built-in directives. Here are just a few:

* Admonition
* Code
* Comment
* Definition list
* Include
* Literal
* Sidebar
* Substitution
* Table of contents

#### Admonition

An admonition is a specially marked block that calls attention to an important point. An admonition can appear anywhere an ordinary block can, and can contain other formatting.

Example:

.. Note::  
 An admonition looks like this.

The following admonition directives are built in:

* Attention
* Caution
* Danger
* Error
* Hint
* Important
* Note
* Tip
* Warning

#### Code

The code directive is for code snippets. If a language is specified, the syntax is highlighted. You can add line numbering by specifying a starting line number with the :number-lines: option.

Example:

.. code:: python  
 :number-lines: 1  
  
 def my\_function():  
 "Doing some math"  
 print 28 \* 9/5 + 32

#### Comment

A block of explicit markup that doesn't do anything else is treated as a comment.

Example:

.. This is a comment.

You can create longer comments by indenting:

Example:

..  
 Comment text, which can span  
 lines.  
  
 You can add more paragraphs.

#### Definition list

A definition list lets you create a glossary of terms. If you use Sphinx, there is a useful glossary directive that works with definition lists, letting you link from a word in the text body to its definition in the glossary.

A definition list looks like this:

term (up to a line of text)  
 Definition of the term, which must be indented  
  
 and can even consist of multiple paragraphs  
  
next term  
 Definition

#### Include

The include directive brings in content from another file whose path is specified relative to the document containing the directive.

Example:

.. include:: ../path/to/other-file.txt

An included file is normally processed as if it were part of the file it's brought into. That is, if it contains reStructuredText, it is interpreted the same as the syntax in the rest of the document. If the included file contains headings, they must match the conventions used for the headings in the rest of the document.

There are options for including just part of an external file (by line number or by other markers) and for treating the file contents as a code or literal block to prevent it from being parsed as reStructuredText.

#### Sidebar

The sidebar directive creates a short, separate section to add reference, context, or other additional information to a document.

Example:

.. sidebar:: Optional Title  
 :subtitle: Optional Subtitle  
  
 Body text of the sidebar, which  
 can contain body elements (but  
 can't contain other sidebars).

#### Substitution

You can use a substitution to replace a word or phrase with a different piece of text or an image, or to apply a style.

The substitution directive looks like this:

.. |label| replace:: text to replace the label

When you put |label| in the body of your text, the parser replaces it with the text to replace the label.

There are a couple gotchas:

* You can't use a substitution inside another directive
* You will get an error if you create a circular reference such as substituting A for B, B for C, and C for A

Here's an example of replacing one phrase with another:

The syntax of |reST| doesn't have to be difficult.  
  
.. |reST| replace:: reStructuredText

The directive replace tells the parser to swap in the text reStructuredText when it encounters the phrase |reST|.

You can use the image directive inside a substitution for icons or other images that appear frequently in the text:

.. |severe| image:: red.png  
 :alt: Severe fire danger!

Substitutions can be helpful if you need to type something in a way that is not very readable in body text, or takes a long time to type.

Example:

The chemical formula |H2SO4| is not very readable in raw reStructuredText,  
but substitutions make it easy on the eyes.  
  
.. |H2SO4| replace:: H\ :sub:`2`\ SO\ :sub:`4`

As reStructuredText doesn't support nested inline markup, a substitution is the only way to create a reference with styled text:

reStructuredText was created to document |Python|\_.  
  
.. |Python| replace:: \*\*Python\*\*  
.. \_Python: http://www.python.org/

Hint

If there are substitutions you want to use for multiple documents (or chapters), you can put them all into a file together and include it in each document using the include directive. If you're using Sphinx, give the file a different extension from the other files (or put it somewhere else) so that it doesn't get interpreted as a standalone document.

#### Table of contents

The contents directive generates a table of contents.

Example:

.. contents:: Optional title  
 :depth: 2

Below the directive, you can specify a few options. In the above example, depth specifies how many levels of heading to display. Links =====

You can link to websites, files, or places in your documents. There are several ways to create a link:

* A standalone link, which is just a URL:
* http://www.example.com
* A link with the URL embedded inline:
* See `this example <http://www.example.com>`\_
* A reference name in the text body with a corresponding target elsewhere in the document:
* See the example\_ for yourself  
    
  .. \_example: http://www.example.com

The third way is the most useful: it keeps the body text readable by getting the URL out of the way, and lets you link anywhere inside or outside the document. If the reference in the text body is one word, then no backticks are required. With backticks, you can create a *phrase reference:*

See `the above example`\_ for yourself.

Note

You probably already noticed that the reference in the body is interpreted text and the reference target is explicit markup.

A link reference ends with an underscore, and a link target begins with an underscore. You can think of the underscore as an arrow to the right, pointing away from the reference and toward the target.

## Internal links

If you want to link to a place in your document, place a label above the header or paragraph you want to link to. For example:

.. \_example:  
  
This paragraph has the example in it. I want to be   
able to link to it.

You can then create a link to the paragraph like this:

Take a look at the example\_ above.

Note

Notice the difference between the explicit markup that defines a link target, and the syntax of a directive. A directive is followed by two colons (::), while a link target is followed by one.

You can embed a link target in the body of text like this:

Here is an \_`example` target. You can link   
to it like this: `example`\_.

If you use Sphinx, you'll find a useful role called :ref: that lets you link to labels in other files.

## Implicit hyperlinks

Section titles, footnotes, and citations automatically work as link targets. You can use the section title text or the footnote label in your link reference. However, you might prefer to create explicit labels so that you can change section titles without breaking internal links.

## Links to links

You can use link targets to create other links. I am not sure why it is useful, but this works:

The `Example page`\_ is a link to www.example.com, but so is example\_!  
  
.. \_Example page: http://www.example.com  
.. \_example: `Example page`\_

## Footnotes and citations

A footnote looks a bit like an internal reference in square brackets.

Example:

Footnotes are very usful in text [1]\_ because they let you add information  
that doesn't fit in the paragraph elegantly.  
  
.. [1] But does anyone read them?

Notice that the footnote target omits the leading underscore and the trailing colon.

You can use numerals to explicity number footnotes, or you can auto-number them with the octothorpe symbol: [#]\_. You can also use text labels or symbols such as the asterisk.

Citations are similar to footnotes, except that the reference label is a short-form citation and the reference target is a longer-form citation.

Example:

As we learn from [CITATION001]\_, citations are useful.  
  
.. [CITATION001] AuthorLastName, AuthorFirstName. "Article Title." Journal Title,  
 Version, Number, Publication Date, Page Numbers.

# Tables

reStructuredText includes a number of different ways to create tables. There are two standard syntaxes (grid tables and simple tables) and three directives.

## Grid tables

Grid tables are a little cumbersome to create, but offer fairly sophisticated syntax. Cells can span rows or columns. Each cell contains a block, which means you can do lots of things inside a cell.

Example:

+--------------------------+-------------------------------------+  
| Element | Syntax |  
+==========================+=====================================+  
| Substitution definition | ``.. |text| replacement`` |  
+--------------------------+-------------------------------------+  
| Cross-reference target | ``.. \_label:`` |  
| | |  
| | ``Paragraph`` |  
+--------------------------+-------------------------------------+  
| Link target | ``.. \_label: URL`` |  
+--------------------------+-------------------------------------+  
| Footnote or citation | ``.. [label] Footnote or citation``|  
+--------------------------+-------------------------------------+

## Simple tables

Simple tables are much easier to create but offer fewer features.

Example:

=============== ==========  
Markup Effect  
=============== ==========  
``\*emphasis\*`` \*emphasis\*  
``\*\*strong\*\*`` \*\*strong\*\*  
=============== ==========

## Table directives

There are three directives that let you create tables in special ways:

* table
* csv-table
* list-table

### Table directive

The table directive gives a table a title and lets you specify things like cell alignment and column widths.

Example:

.. table:: Inline markup  
 :align: left  
  
 =============== ==========  
 Markup Effect  
 =============== ==========  
 ``\*emphasis\*`` \*emphasis\*  
 ``\*\*strong\*\*`` \*\*strong\*\*  
 =============== ==========

### CSV table directive

The csv-table directive lets you create a table from CSV (comma-separated values) data.

Example:

.. csv-table:: Inline markup  
 :header: "Markup", "Effect"  
  
 "``\*emphasis\*``", "\*emphasis\*"  
 "``\*\*strong\*\*``", "\*\*strong\*\*"

You can use a CSV table with an include directive to display comma-separated values from an external file as a table in a reStructuredText document.

### List table directive

The list-table directive lets you create a table from a two-level bullet list, where each bullet is a row and the sub-bullets are cells in the row. Each sublist must contain the same number of list items.

Example:

.. list-table:: Inline markup  
 :header-rows: 1  
  
 \* - Markup  
 - Effect  
 \* - ``\*emphasis\*``  
 - \*emphasis\*  
 \* - ``\*\*strong\*\*``  
 - \*\*strong\*\*

## Images and figures

Images and figures are directives that display pictures. The simplest version of the syntax looks like this:

.. image:: ../path/to/image.png

The path can be a URL, a file path, or just the image name if the image and the document are in the same directory.

You can specify options about how the picture should be rendered:

.. image:: picture.jpeg  
 :height: 100px  
 :width: 200px  
 :scale: 50%  
 :alt: alternate text  
 :align: right

A figure lets you add a caption, several paragraphs indented the same amount, which can include tables or other markup, including images.

.. figure:: image.png  
 :scale: 50%  
 :alt: Severity of fire danger  
  
 This map shows the relative fire danger of different counties this year.  
  
 +-----------------------+---------+  
 | Symbol | Meaning |  
 +=======================+=========+  
 | .. image:: red.png | Severe |  
 +-----------------------+---------+  
 | .. image:: yellow.png | Moderate|  
 +-----------------------+---------+  
 | .. image:: green.png | Low |  
 +-----------------------+---------+

# Some useful directives

reStructuredText offers many useful built-in directives. Here are just a few:

* Substitution
* Definition list
* Admonition
* Table of contents
* Include
* Sidebar
* Literal
* Code
* Comment

## Substitution

You can use a substitution to replace a word or phrase with a different piece of text or an image, or to apply a style.

The substitution directive looks like this:

.. |label| replace:: text to replace the label

When you put |label| in the body of your text, the parser replaces it with the text to replace the label.

There are a couple gotchas:

* You can't use a substitution inside another directive
* You will get an error if you create a circular reference such as substituting A for B, B for C, and C for A

Here's an example of replacing one phrase with another:

The syntax of |reST| doesn't have to be difficult.  
  
.. |reST| replace:: reStructuredText

The directive replace tells the parser to swap in the text reStructuredText when it encounters the phrase |reST|.

You can use the image directive inside a substitution for icons or other images that appear frequently in the text:

.. |severe| image:: red.png  
 :alt: Severe fire danger!

Substitutions can be helpful if you need to type something in a way that is not very readable in body text, or takes a long time to type.

Example:

The chemical formula |H2SO4| is not very readable in raw reStructuredText,  
but substitutions make it easy on the eyes.  
  
.. |H2SO4| replace:: H\ :sub:`2`\ SO\ :sub:`4`

As reStructuredText doesn't support nested inline markup, a substitution is the only way to create a reference with styled text:

reStructuredText was created to document |Python|\_.  
  
.. |Python| replace:: \*\*Python\*\*  
.. \_Python: http://www.python.org/

Hint

If there are substitutions you want to use for multiple documents (or chapters), you can put them all into a file together and include it in each document using the include directive. If you're using Sphinx, give the file a different extension from the other files (or put it somewhere else) so that it doesn't get interpreted as a standalone document.

## Definition list

A definition list lets you create a glossary of terms. If you use Sphinx, there is a useful glossary directive that works with definition lists, letting you link from a word in the text body to its definition in the glossary.

A definition list looks like this:

term (up to a line of text)  
 Definition of the term, which must be indented  
  
 and can even consist of multiple paragraphs  
  
next term  
 Definition

## Admonition

An admonition is a specially marked block that calls attention to an important point. An admonition can appear anywhere an ordinary block can, and can contain other formatting.

Example:

.. Note::  
 An admonition looks like this.

The following admonition directives are built in:

* Attention
* Caution
* Danger
* Error
* Hint
* Important
* Note
* Tip
* Warning

## Table of contents

The contents directive generates a table of contents.

Example:

.. contents:: Optional title  
 :depth: 2

Below the directive, you can specify a few options. In the above example, depth specifies how many levels of heading to display.

## Include

The include directive brings in content from another file whose path is specified relative to the document containing the directive.

Example:

.. include:: ../path/to/other-file.txt

An included file is normally processed as if it were part of the file it's brought into. That is, if it contains reStructuredText, it is interpreted the same as the syntax in the rest of the document. If the included file contains headings, they must match the conventions used for the headings in the rest of the document.

There are options for including just part of an external file (by line number or by other markers) and for treating the file contents as a code or literal block to prevent it from being parsed as reStructuredText.

## Sidebar

The sidebar directive creates a short, separate section to add reference, context, or other additional information to a document.

Example:

.. sidebar:: Optional Title  
 :subtitle: Optional Subtitle  
  
 Body text of the sidebar, which  
 can contain body elements (but  
 can't contain other sidebars).

## Code

The code directive is for code snippets. If a language is specified, the syntax is highlighted. You can add line numbering by specifying a starting line number with the :number-lines: option.

Example:

.. code:: python  
 :number-lines: 1  
  
 def my\_function():  
 "just a test"  
 print 8/2

## Comment

A block of explicit markup that doesn't do anything else is treated as a comment.

Example:

.. This is a comment.

You can create longer comments by indenting:

Example:

..  
 Comment text, which can span  
 lines.  
  
 You can add more paragraphs.

# Tools

editors storage git git-setup git-basics publishing

You want:

* Editor
* Storage
* Git

- Publishing tools Editors =======

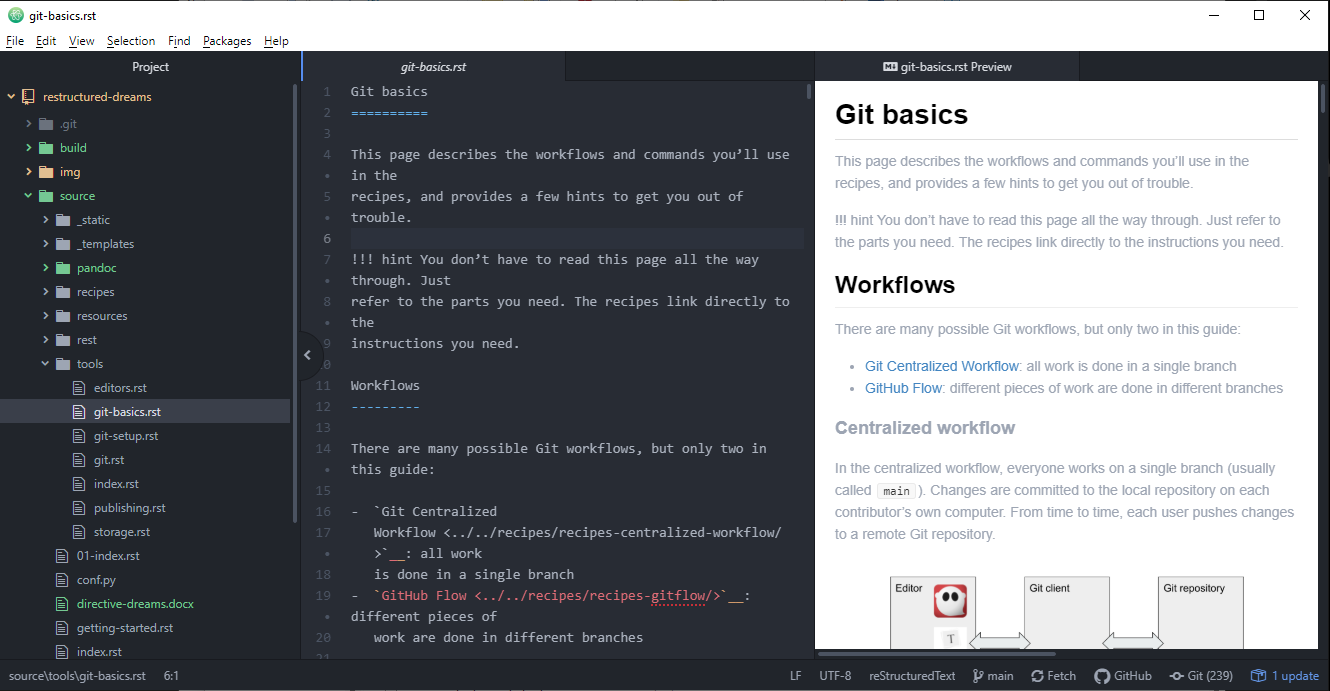
Like other markup languages, reStructuredText works well with text editors and code editors, and a small number of purpose-built editors.

Hint

Atom is easy to install on any platform and there are packages available for reStructuredText (click **Packages > Settings View > Install Packages/Themes** and then search). Atom also provides a Project view (directories and files) and Git integration.

## Code editors

Code editors, while they are not designed specifically for reStructuredText, sometimes offer sophisticated packages to support reStructuredText editing and preview.



An advantage of a code editor is the ability to do things like jump to a specific line number or set text wrapping rules. Some editors let you manage your project in Git directly from the user interface. The following editors, among others, are worth a look:

* [Atom](https://atom.io/)
* [Sublime Text](https://www.sublimetext.com/)
* [Visual Studio Code](https://code.visualstudio.com/)

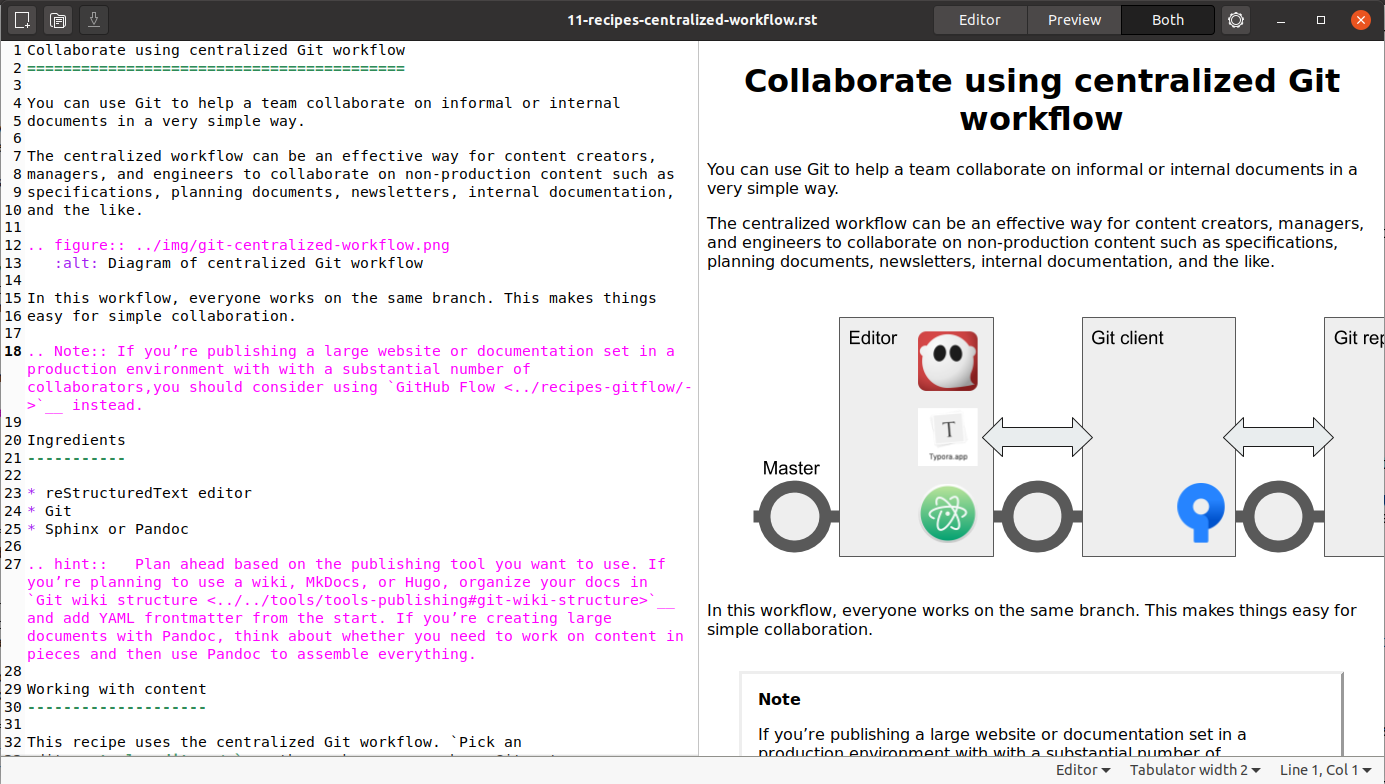
There are also various reStructuredText tools for [Eclipse](https://www.eclipse.org/).

## reStructuredText editors

There are a few dedicated reStructuredText editors out there. Most editors offer a live preview with error messages to help you troubleshoot your reStructuredText as you write. My current favorite is ReText, which is simple but makes tables and images easy. An advantage of a dedicated reStructuredText editor is that some of them include support for editing tables and other syntactical features.

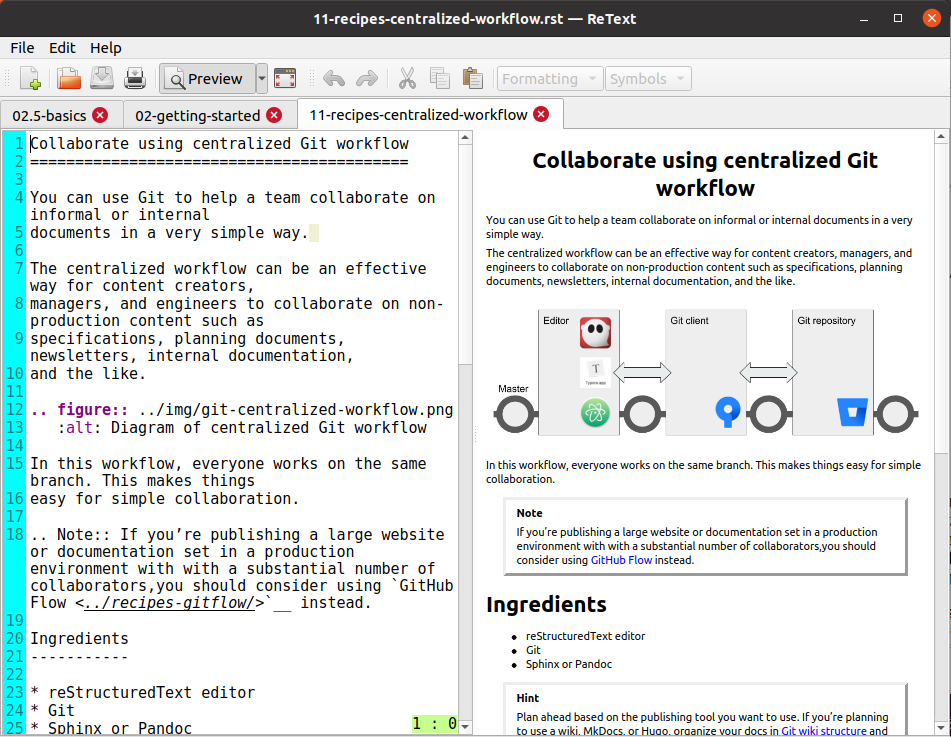
### Formiko

[Formiko](https://github.com/ondratu/formiko#readme) (Python) is a simple, capable editor that works with both reStructuredText and Markdown. It is no-frills, offering a few controls and preferences but not much more.



### ReText

[ReText](https://github.com/retext-project/retext/blob/master/README.md) (Python) is a Markdown, reStructuredText, and Textile editor with a few frills, including menu commands to help with tables and images.



### Online editors

* [Online reStructuredText editor](http://rst.ninjs.org/#)
* [Online Sphinx Editor](https://livesphinx.herokuapp.com/)
* [reStructuredText Viewer](http://rst.aaroniles.net/)

## Text editors

Because reStructuredText is just plain text, even the simplest text editor can be a capable tool. Some well-established text editors offer reStructuredText support:

* [BbEdit](https://www.barebones.com/products/bbedit/bb)
* [Emacs](https://www.gnu.org/software/emacs/)
* [JEdit](http://www.jedit.org/)
* [Notepad++](https://notepad-plus-plus.org/)
* [Vim](https://www.vim.org/)

## Installing a Python-based editor

On Linux, you can sometimes install Python-based programs the same way you install other apps. On macOS or Windows, you'll need to open the terminal and use a program called pip (or sometimes pip3 if you're using Python 3) that knows how to locate and install Python applications.

Here's how it looks on Windows:

C:\Users\pconrad>pip install formiko  
Collecting formiko  
 Downloading formiko-1.4.3-py3-none-any.whl (115 kB)  
 |████████████████████████████████| 115 kB 2.2 MB/s  
Requirement already satisfied: docutils>=0.12 in c:\users\pconrad\appdata\local\programs\python\python38\lib\site-packages (from formiko) (0.16)  
Installing collected packages: formiko  
Successfully installed formiko-1.4.3

Once the installation process is complete, you can run the program (in this case, formiko) from the command line.

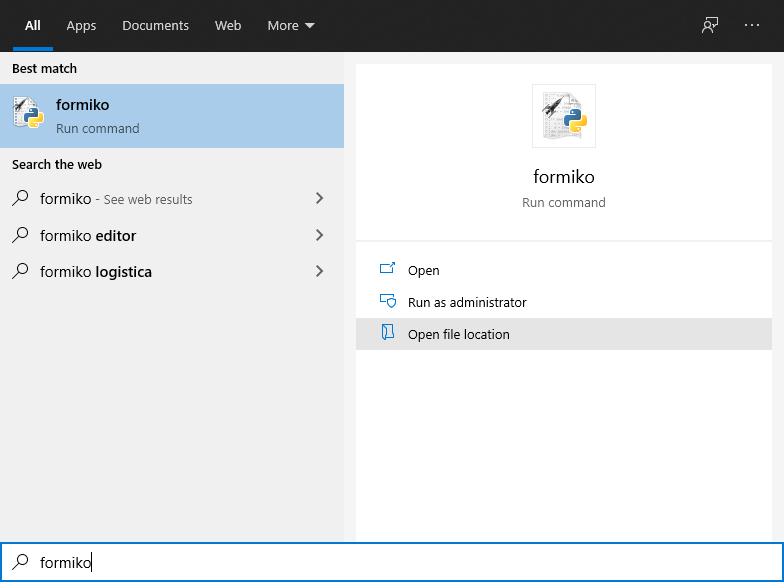
Note

I've had mixed results with Python apps on different platforms. If you have trouble installing and using one Python app, try another.

### Adding a shortcut in Windows

In Windows, it's not difficult to add a taskbar shortcut for a python program.

1. Click the **Start** menu and type the name of the program you installed.

* 

1. Click **Open file location** to view the program (in this case, formiko.exe) in the File Explorer.
2. Drag the file to the taskbar.

# Storage

The simplest place to store ReStructuredText is on your own computer, of course. You can merrily save files to a folder deep in My Documents and go on your way, perhaps emailing someone a Pandoc-generated PDF from time to time. But if you want to enjoy the true power of collaborating and publishing with ReStructuredText, you’ll want to share the files you’re working on or publish them in another form.

## A shared server

If you are collaborating on documents or contributing to documentation at work, you might need access to a shared server such as SharePoint or Samba, WebDAV, or online tools such as a Google Drive account. You can store your source files there or upload HTML, Word, PDF, and other documents you create from your ReStructuredText.

## Cloud storage

Cloud storage services like [Box](https://www.box.com), [DropBox](https://www.dropbox.com/), and [OneDrive](https://onedrive.live.com) are a great place to keep documents you want available from everywhere. You can save ReStructuredText files from one computer and have them available on other devices, even sharing them with colleagues.

## Web hosting

If you’re creating online documentation or a website, you’ll need a place to host it. Web hosting is far too big a subject to cover here, but you might need to investigate whether you need any of the following:

* A blog service such as [Wordpress](https://wordpress.com/), [Blogger](https://www.blogger.com/), or [Medium](https://medium.com/) where you can publish writing directly
* A web host, to present your HTML files as web pages
* FTP software to upload HTML to a web host
* Another integrated online publishing service or platform

# Source control with Git

Any time you collaborate with others, source control is important. Although you could just share a folder in the cloud, eventually there will come a time when someone overwrites or deletes something important. That’s where source control comes in. And by “source control,” I mean Git.

With Git, you synchronize files on your computer with files on a remote repository—usually an online Git host.

## Why Git?

Back in the day, there were all kinds of source control systems. The ones I remember were all pretty easy to use, but were all centralized—meaning that only one person could edit any given file at any given time. Worse, it meant that there was one central source of truth; if that were to be corrupted or lost, everything would be gone. Git solves these problems and [doesn’t have to be hard to use](https://levelup.gitconnected.com/git-doesnt-have-to-be-hard-e1e115be6668).

## Get Git

You’ll need a place to host a Git repository. You might start by signing up with a Git host. Here are a few examples:

* [Bitbucket](https://bitbucket.org)
* [GitHub](https://github.com/)
* [GitLab](https://about.gitlab.com/)

You can use Git on the command line, but it’s even easier if you use a Git client. For example:

* [GitHub Desktop](https://desktop.github.com/)
* [Sourcetree](https://www.sourcetreeapp.com/)

Hint

While any Git client should work with any Git host, it’s not a bad idea to use the client and host that go together. For example:

* Bitbucket and Sourcetree
* GitHub and GitHub Desktop

This guide includes instructions for those two combinations.

## Git concepts

Git is different from older source control systems. A lot of things that used to be intuitive—like the idea that we’re working on *files* and we need to *lock* them so that someone else can’t *check them out* from the *central repository*—have no currency in the Git world:

* There’s no *central repository.* A central repository would represent both a bottleneck and a single point of failure, so why do it?
* No one *checks out* files because there’s no central repository.
* We don’t need to *lock* files because we don’t have to worry about them being *checked out*.

In fact, Git doesn’t care about *files* at all. Git only cares about *changes.*

There’s a lot to know about Git. As a writer, you should be able to decide how much you want to learn. You might just want to learn *exactly* enough to do your writing and keep out of trouble. That’s the goal here. If you want to learn more, check out [The Git Book](https://git-scm.com/book/en/v2).

##### What things mean in Git

Here are a few basic Git terms you’ll see in some of the recipes.

**Commit**

As you make changes to the files you’re working on, saving periodically, you also tell Git from time to time that you want your changes tracked. This is called *committing* the changes, and is a little like “saving changes to Git.” Git makes it easy by noticing the changes you’ve made so you can review them and make sure you’re not accidentally tracking something irrelevant. When you commit, you type a little note describing the changes so that people know what you did.

A group of changes you’ve committed is also called a *commit.*

**Stage**

Before you commit changes, you tell Git which changes to track. This is called *staging.* Since changes go with files, sometimes people think of it as staging the files themselves—but it’s really the changes that Git wants to know about. If you delete a file, that’s a change too.

**Repository**

When you have committed, your changes are stored in the local *repository,* or *repo* for short. It’s just like that old central repository, but it’s on your computer.

**Push**

If you want other people to be able to work on your files, then you need to put them in an online repository (such as Bitbucket, GitHub, or GitLab). This is not *checking in* since the changes are already committed to your local repository. This is called a *push* to a *remote repository*.

**Clone**

Once something is in an online repository, an authorized person can *clone* their own copy of the whole repository and work on the files locally.

**Pull**

As you work on a repository you’ve cloned, you *pull* the latest changes from the remote repo to stay up to date.

**Branch**

Git lets people work in separate work streams called *branches* so that they don’t interfere with each other’s work. A branch is just a series of commits (and a commit is a group of changes). You’re always working in a branch, even if there’s only one branch. When you have several branches to work in, Git remembers the state of everything in each branch so that when you switch between them everything is just how you expect it.

Creating a new branch is called *branching,* of course. The Git command for creating (or switching to) a branch is called, confusingly, *checkout.*

There are different branching strategies. Some are complicated; others are simple. The recipes in this guide use two simple branching strategies, described in [Git basics](../tools-git-basics/).

**Merge**

If there’s more than one branch, there always comes a time to *merge,* which means to add the changes from one branch into another.

##### Why branching is cool

Git keeps track of the whole history of all the changes on all the branches. Not only does that mean you can roll back to any point in time, it also means that when you switch branches all your files magically change to match whatever changes you’ve made in that branch.

For example, I created a branch called test-branch based on the main branch. Working in test-branch, I added a file called new-file.md which you can see in the directory:

$ ls  
getting-started img index.md new-file.md

When I switch to main, it’s not there:

$ git checkout main   
Switched to branch 'main'  
Your branch is up to date with 'origin/main'.  
$ ls  
getting-started img index.md

When I switch back to test-branch, it’s there:

$ git checkout test-branch   
Switched to branch 'test-branch'  
$ ls  
getting-started img index.md new-file.md

Branching is cool.

# Git setup

If you want to be ready for all the recipes in this guide, follow these steps to install Git and a Git client, sign up for a host, and set up your first repository.

## Install Git

On the command line, check whether Git is installed on your computer by typing:

git --version

If Git is not installed, follow these steps.

1. Install Git on your computer:
   * [Windows](https://git-scm.com/download/win)
   * [macOS](https://git-scm.com/download/mac)
   * [Linux](https://git-scm.com/download/linux)
2. Install a git client such as [Sourcetree](https://www.sourcetreeapp.com/) or [GitHub Desktop](https://desktop.github.com/).

Hint

You can use any Git client with any Git host, but some clients work especially well with specific hosts. For example:

* Sourcetree and Bitbucket
* GitHub Desktop and GitHub

## Set up a repository

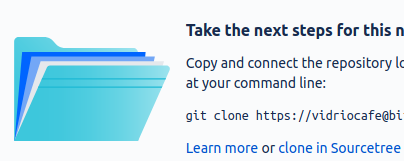
A *repository* is where you keep your work. You’ll need a *local repository* where you edit and save files on your computer, and a *remote repository* online that lets people collaborate on the same project. A straightforward way to create both is to set up a repository with an online host and then *clone* it (create a local copy). Your collaborators can also clone the repository to their own computers, so everyone can keep in sync by pushing and pulling changes.

### Bitbucket and Sourcetree

1. Sign up with Bitbucket (<https://bitbucket.org/>) and log on.
2. Click the new repository button (the plus sign):

* New repository button

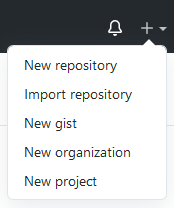
1. Type a repository name, make sure it's set to be a public repository, and
2. click **Create repository**.
3. Choose or create a directory on your computer where you would like to keep your
4. local copy of the project.
5. Click **Clone in Sourcetree**.

* 

1. Choose a folder on your computer for the local copy of the repository
2. and click **Clone**.

### GitHub and GitHub Desktop

1. Sign up with GitHub (<https://bitbucket.org/>) and log on.
2. Click the plus sign and select **New repository**:

* 

1. Type a repository name, make sure it's set to be a public repository, and click **Create repository**.
2. Click **Set up in Desktop** to open the repository in GitHub Desktop:

* Set up in desktop button

1. Choose a folder on your computer for the local copy of the repository and click **Clone**.

It might not look like much has happened, but you now have:

* Git running on your computer
* A repository at an online Git host
* A local copy of the repository on your computer

# Git basics

This page describes the workflows and commands you’ll use in the recipes, and provides a few hints to get you out of trouble.

Hint

You don’t have to read this page all the way through. Just refer to the parts you need. The recipes link directly to the instructions you need.

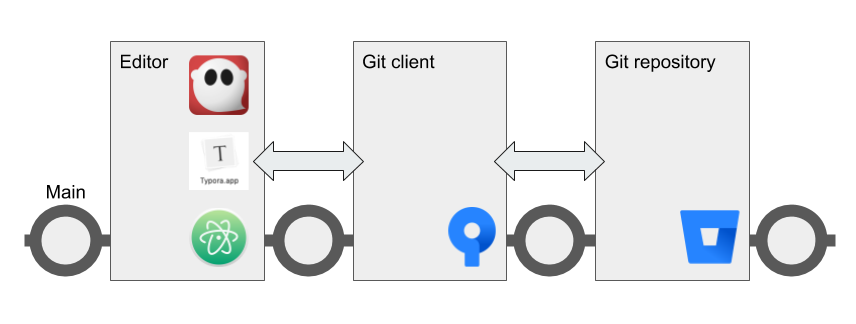
## Workflows

There are many possible Git workflows, but only two in this guide:

* **Centralized Workflow:** all work is done in a single branch
* **GitHub Flow:** different pieces of work are done in different branches

### Centralized workflow

In the centralized workflow, everyone works on a single branch (usually called main). Changes are committed to the local repository on each contributor’s own computer. From time to time, each user pushes changes to a remote Git repository.



Here are the operations a contributor performs when working in the centralized workflow.

1. **Pull** - Fetch the latest changes from the remote repository to the local repository on your computer
2. **Work** - Edit your content in your favorite Markdown editor
3. **Stage and commit** - From time to time, in your Git client, type a short sentence about what you've done and save the changes to Git
4. **Push** - When your work is final, push it up to the remote repository

In the event that two people create conflicting changes, they can be manually resolved and then merged.

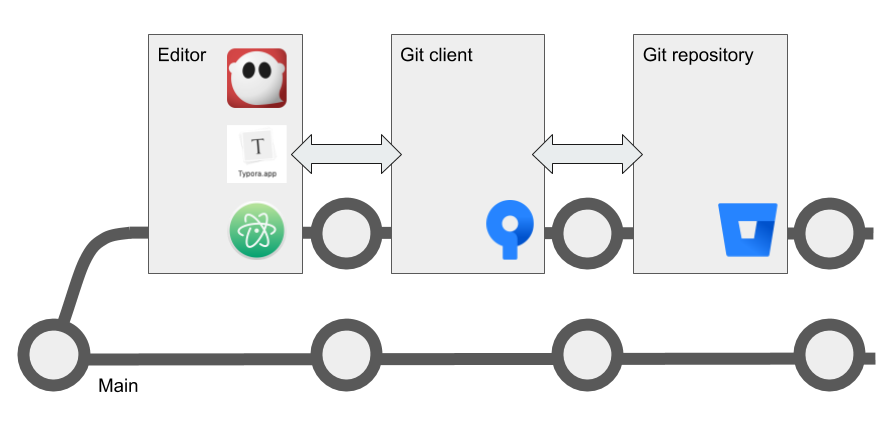
Although this workflow is called “centralized,” it doesn’t really resemble the old centralized source control model. The central remote repository is not a single source of truth, because every contributor has their own copy of the entire repository.

The following recipes use the centralized workflow:

* [Edit a Git wiki](../../recipes/recipes-git-wiki/)
* [Collaborate using centralized Git workflow](../../recipes/recipes-centralized-workflow/)

### GitHub flow

In the GitHub flow, you start a new branch based on main whenever you start a group of related tasks. How you organize those tasks, and how you decide which ones belong in a branch together, is up to you.



Just like in the centralized workflow, you commit changes frequently to your own local repository. When you push to the remote repo, you create a copy of your branch there so that others can review your changes before they are merged back into main.

Here’s how your day looks in the GitHub flow.

1. **Pull** - Make sure you're on the main branch and sync the latest changes from the remote repository.
2. **Create a branch** - Create ("check out") a branch for working on the current part of the content.
3. **Work** - Edit your content in your favorite Markdown editor.
4. **Stage and commit** From time to time, in your Git client, type a short sentence about what you've done and save the changes to Git.
5. **Push** - From time to time, sync your branch up to the remote repository.
6. **Create a pull request** - When your work is ready for review, create a pull request and add reviewers. If there's more work to do before final approval, you can edit your content, stage and commit, an push to the existing pull request.
7. **Merge** - When your work is approved, merge your branch into main on the remote repository.

Merge conflicts in your working branches are less likely, because own your branches and other people don’t necessarily work in them with you. Conflicts are more likely to happen between your working branch and the main branch on the remote repo. Before you push your branch to the remote repo, you can pull from main again and merge any conflicts locally.

Hint

After you’ve pushed, approved, and merged your work, you can delete your working branch or keep it around for further work. When you start work on a different part of the project, remember to switch to main and pull again before creating a new branch.

The following recipe uses the GitHub flow:

* [Managing docs with GitHub flow](/recipes/recipes-gitflow/)

## How to Git

These are the commands that make up the steps in the Git workflows.

### Pull

The Git pull command fetches and downloads content from your remote repository, automatically merges the changes with your local repository, and updates everything so that your repository matches the latest version of everything on the remote. It’s a good idea to pull after making sure you’re on the right branch and before starting to work on the content.

#### Sourcetree

1. Make sure you're on the right branch in the correct repository:
   * The bold text under **Branches** tells you the branch
   * The tab at the top of the screen tells you the repository
2. Select **Repository > Pull** or click the **Pull** button.

* 

#### GitHub Desktop

In GitHub Desktop, you can *fetch* and *pull* separately. Fetch gets the latest updates from origin but doesn't update your local working copy with the changes. After you click **Fetch origin**, the button changes to **Pull Origin**.

1. Make sure you're on the right branch in the correct repository:
   * The bold text under **Current branch** tells you the branch
   * The bold text under **Current repository** tells you the repository
2. Select **Repository > Pull** or:
   1. Click the **Fetch origin** button.
   * Fetch origin button
   1. Click the **Pull origin** button.
   * Pull origin button

#### Linux command line

1. Make sure you’re on the right branch in the correct repository:

* $ git branch   
  \* main   
  $ git remote -v   
  origin https://github.com/pconrad-fb/markdown.git (fetch)   
  origin https://github.com/pconrad-fb/markdown.git (push)

1. Type the git pull command:

* $ git pull

### Stage and commit

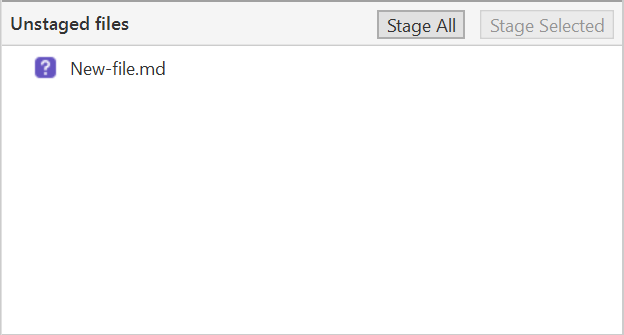
Git knows when you make changes to your files. When you want to save those changes to Git, you must do two things:

* *stage* them, which tells Git which changes you intend to keep
* *commit* them, which saves the changes.

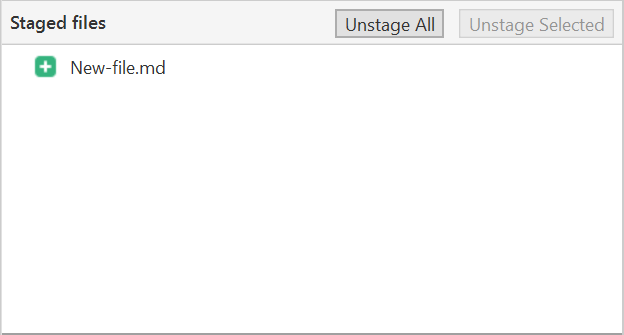
#### Sourcetree

In Sourcetree, you stage and commit your files in two operations.

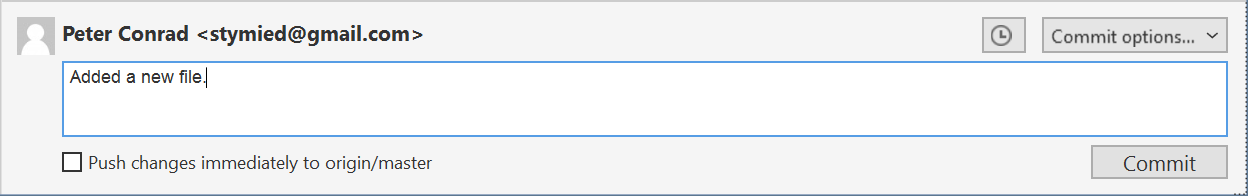
1. Make sure you're on the right branch in the correct repository.
2. Look for the files you changed in the Unstaged files pane. Select the files you want to stage&mdash;in most cases, you can just click **Stage All.**

* 

1. Make sure you see the right files in the Staged files pane.

* 

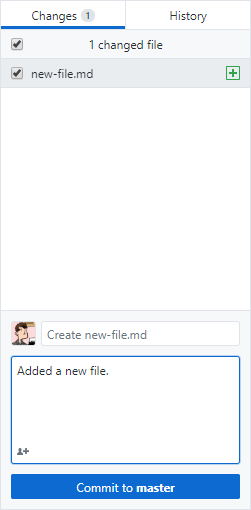
1. Type a short commit message and click **Commit**.

* 

#### GitHub Desktop

In GitHub Desktop, you can stage and commit your files in one step.

1. Make sure you're on the right branch in the correct repository.
2. Look for the files you changed in the Changes tab. Unselect any files you don't want to change&mdash;most of the time, you can leave all the checkboxes checked.

* 

1. Type a short commit message.
2. Make sure the **Commit** button refers to the correct branch ("Commit to main," for example).
3. Click **Commit to [branch]**.

#### Linux command line

1. Make sure you're on the right branch in the correct repository.
2. Use git status to see what changes are not yet staged.
3. Stage any changes you plan to commit. In many cases, you can stage all the changes at once like this:

* $ git add \*

1. Commit the changes, adding a descriptive message:

* $ git commit -m "Type your descriptive message here."

Hint

If you are changing files but not adding or deleting any files, you can often stage and commit all in one line with commit -am like so:

$ git commit -am "Type your descriptive message here."

### Push

#### Sourcetree

1. Make sure you're on the right branch in the correct repository.
2. Select **Repository > Push** or click the **Push** button.

* 

#### GitHub Desktop

1. Make sure you're on the right branch in the correct repository.
2. Select **Repository > Push** or click the **Push origin** button.

* Push origin button

#### Linux command line

1. Make sure you're on the right branch in the correct repository.
2. Push, specifying the remote (usually origin) and the branch. For the recipes where you are working on the main branch, the command looks like this:

* $ git push origin main
* Of course, if git knows what branch you're on and where your remote is, you can sometimes just type git push.

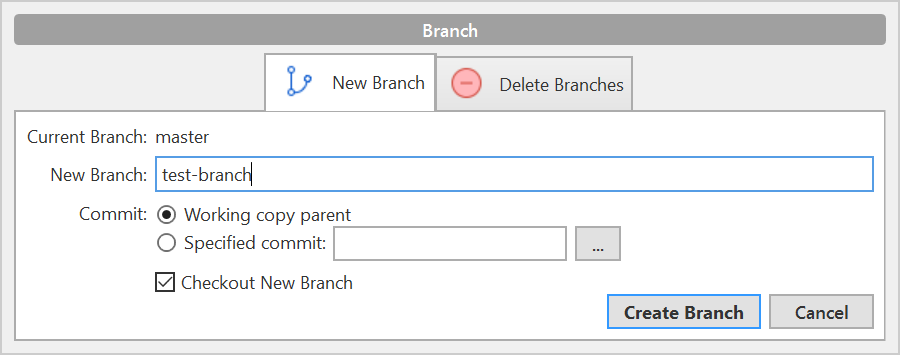
### Create a branch

#### Sourcetree

1. **Pull** from main.
2. Click the **Branch** button:

* 

1. Type a descriptive name and click **Create Branch**.

* 

1. Look under **Branches** to see that you're on the new branch.

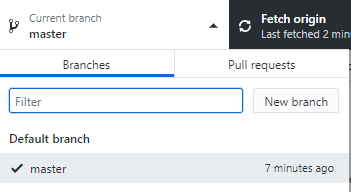
You can switch to a different branch by clicking it in the list of branches.

Hint

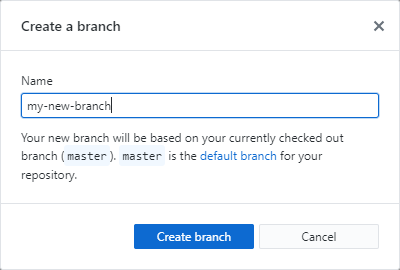
You can’t switch branches with uncommitted changes. You have to commit before switching to a new branch.

#### GitHub Desktop

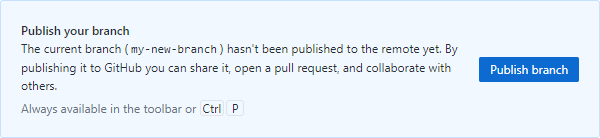
1. **Pull** from main.
2. Click the **Current branch** tab and click **New branch**:

* 

1. Type a descriptive name and click **Create branch**:

* 

1. Click **Publish branch**:

* 

1. Look under **Branches** to see that you're on the new branch.

You can switch to a different branch by clicking it in the list of branches.

Hint

You can’t switch branches with uncommitted changes. You have to commit before switching to a new branch.

#### Linux command line

1. Pull from main to make sure you have the latest changes:

* $ git checkout main  
  Already on 'main'  
  Your branch is up to date with 'origin/main'.  
  $ git pull  
  Already up to date.

1. Create a new branch and switch to it with git checkout -b. Example:

* $ git checkout -b test-branch   
  Switched to a new branch 'test-branch'

You can switch to any existing branch by typing git checkout <branch-name> without the -b. Example:

$ git checkout test-branch   
Switched to branch 'test-branch'

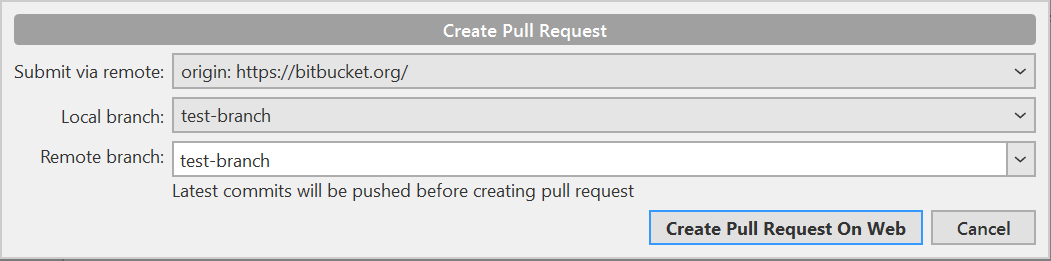
Hint

You can’t switch branches with uncommitted changes. You have to commit before switching to a new branch.

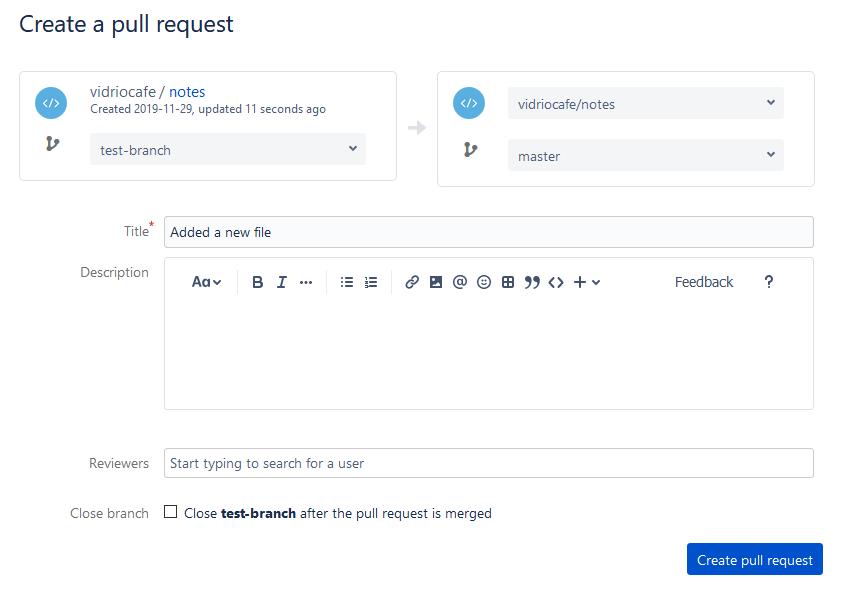
### Create a pull request

#### Bitbucket and Sourcetree

1. Click **Repository > Create pull request**.
2. In the dialog that appears, click **Create Pull Request On Web**:

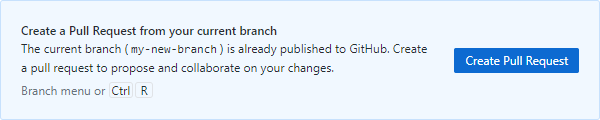
* 

1. Type a description, add reviewers, and click **Create pull request**:

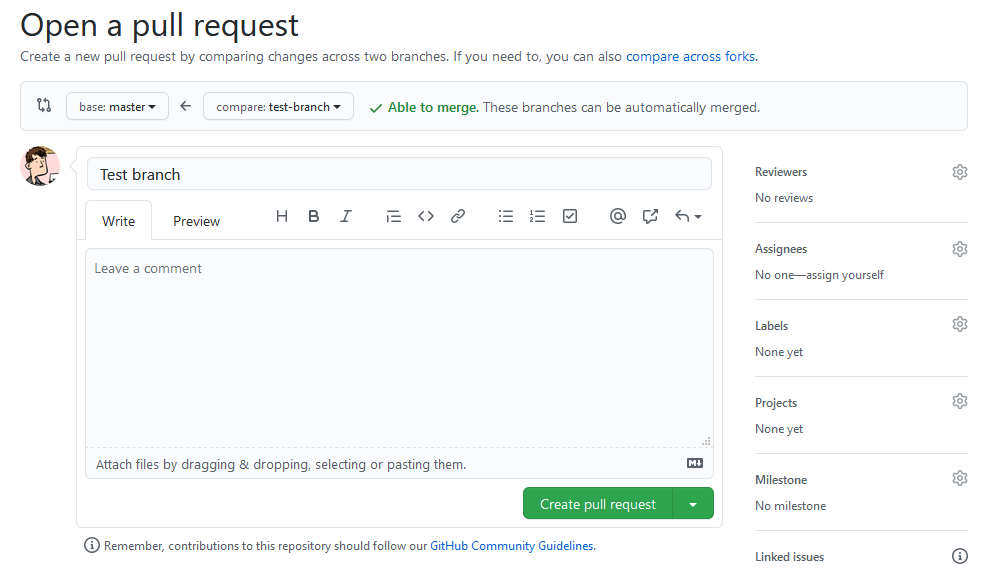
* 

#### GitHub and GitHub Desktop

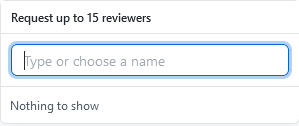
1. When you push, the banner with the Push button changes to read "Create a pull request from your current branch." Click **Create Pull Request**:

* 

1. The browser opens a page with a form for creating a pull request:

* 

1. Click the gear next to **Reviewers** to add reviewers:

* 

1. Click **Create pull request**.

#### Linux command line

1. Take a look at the output from the push command:

* $ git push origin test-branch  
  Enumerating objects: 14, done.  
  Counting objects: 100% (14/14), done.  
  Delta compression using up to 4 threads  
  Compressing objects: 100% (10/10), done.  
  Writing objects: 100% (10/10), 4.39 KiB | 1.10 MiB/s, done.  
  Total 10 (delta 2), reused 0 (delta 0)  
  remote: Resolving deltas: 100% (2/2), completed with 1 local object.  
  remote:   
  remote: Create a pull request for 'test-branch' on GitHub by visiting:  
  remote: https://github.com/pconrad-fb/markdown/pull/new/test-branch  
  remote:   
  To https://github.com/pconrad-fb/markdown.git  
   \* [new branch] test-branch -> test-branch

1. Copy the URL from the line after Create a pull request into a browser.
2. Follow the instructions on the screen. If you get stuck, you can see some hints in the instructions for Bitbucket or GitHub.

### Approve and merge

Merge your branch online in the web interface of your Git host.

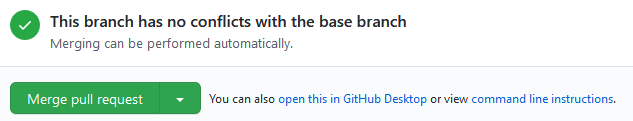
#### Bitbucket

1. When your pull request is sufficiently approved, click **Merge**:

* Merge button

#### GitHub

1. When your pull request is sufficiently approved, click **Merge**:

* 

## Getting out of trouble

If you get out into the woods with Git, there’s usually a way to get back—but for this kind of magic, you have to go to the command line.

### Working in the wrong branch

You’ve edited a file in the wrong branch. What you’d like to be able to do is undo those changes, switch branches, then re-do them. Actually, it would be even better to lift those changes off of the wrong branch, laying them gently on top of the branch you meant to be in. Fortunately, Git provides a command called stash that does exactly that.

1. Make sure you’re in the right directory.
2. Use git status to check what branch you’re on and what changes Git knows about.
3. Make sure you’re in the branch where you were erroneously working. For example:

* $ git checkout the-wrong-branch   
   Switched to branch 'the-wrong-branch'

1. Stash your uncommitted changes:

* $ git stash

1. Switch to the branch you wish you had been working in:

* $ git checkout -b the-wrong-branch   
   Switched to branch 'the-wrong-branch'

1. Use stash to apply the changes there:

* $ git stash apply

### Editing the wrong file

You opened a file to look at it, but then your cat walked across the keyboard. You’re not sure what was added or deleted. You just want to go back to the way things were at the last commit. For this, use checkout—it’s not just for switching branches, but also for fixing changes to files.

1. Make sure you’re in the right directory.
2. Use git status to check what branch you’re on and what changes Git knows about.
3. If necessary, switch to the correct branch. For example:

* $ git checkout the-branch   
   Switched to branch 'the-branch'

1. Use git status to see what files were accidentally modified. For example:

* $ git status  
  On branch main  
  Changes not staged for commit:  
   (use "git add <file>..." to update what will be committed)  
   (use "git checkout -- <file>..." to discard changes in working directory)  
    
   modified: dont-change-this.md

1. Use git checkout -- <file> to undo the changes. For example:

* git checkout -- dont-change-this.md

Hint

The output of the git status command tells you how to use git checkout this way.

### Staged too soon

You edited the right file the right way, but then you added it to the staging area too hastily. You don’t want to undo your changes to the file, but you would like to remove it from the next commit. This is one of the uses of reset. You can also use reset to do more drastic rollbacks—you can undo entire commits if needed.

1. Make sure you’re in the right directory.
2. Use git status to check what branch you’re on and what changes Git knows about.
3. If necessary, switch to the correct branch. For example:

* $ git checkout the-branch   
   Switched to branch 'the-branch'

1. Use git status to see what files were accidentally modified. For example:

* $ git status  
  On branch main  
  Changes to be committed:  
   (use "git reset HEAD <file>..." to unstage)  
    
   renamed: README.md -> README  
   modified: dont-commit-this.md

1. Use git reset to remove the file from the next commit. For example:

* git reset HEAD dont-commit-this.md

Hint

The output of the git status command tells you how to use reset to unstage changes.

### Merge conflicts

When Git is unable to automatically merge, it means that two changes happened in the same place in the same file. That means you need to edit the file and decide which of the two changes to keep.

When you open the file, the merge conflict looks like this:

<<<<<<< HEAD  
Some content that was changed by one person  
=======  
Other content that someone else changed  
>>>>>>> 9af9d3b

HEAD is a pointer to the most recent commit in the branch you’re on. The other label can be another branch name or a number representing another commit.

All you need to do is decide which version of the content you want to keep and then delete the merge conflict markers (<<<<<<<, =======, >>>>>>>).

After you’ve resolved all the changes in that way, just commit again.

### More trouble

For more help and advice, check out [Dangit, Git!?!](https://dangitgit.com/)

# Publishing tools

<https://people.mbi.ucla.edu/leec/docs/gitpublish/intro.html>

## Sphinx

<https://www.sphinx-doc.org/en/master/>

Sphinx is a tool that makes it easy to create intelligent and beautiful documentation, written by Georg Brandl and licensed under the BSD license.

It was originally created for the Python documentation, and it has excellent facilities for the documentation of software projects in a range of languages. Of course, this site is also created from reStructuredText sources using Sphinx! The following features should be highlighted:

Output formats: HTML (including Windows HTML Help), LaTeX (for printable PDF versions), ePub, Texinfo, manual pages, plain text Extensive cross-references: semantic markup and automatic links for functions, classes, citations, glossary terms and similar pieces of information Hierarchical structure: easy definition of a document tree, with automatic links to siblings, parents and children Automatic indices: general index as well as a language-specific module indices Code handling: automatic highlighting using the Pygments highlighter Extensions: automatic testing of code snippets, inclusion of docstrings from Python modules (API docs), and more Contributed extensions: more than 50 extensions contributed by users in a second repository; most of them installable from PyPI

Sphinx uses reStructuredText as its markup language, and many of its strengths come from the power and straightforwardness of reStructuredText and its parsing and translating suite, the Docutils.

Sphinx is a Python documentation generator.

It requires Python, which is installed by default in Linux and Mac OS X systems. For Microsoft Windows systems, see Installing Python on Windows if you need help installing Python and two useful installation utilities (easy\_install and pip).

After you have Python installed, simply use the following command (in a command window):

easy\_install -U Sphinx

Elevated privileges (i.e. administration rights) should not be required.

The Sphinx builder can produce a number of output formats (e.g. HTML, PDF). PDF files can be produced using the LaTeX builder (more complicated) or using the a direct PDF builder called rst2pdf (see below).

## Rst2Pdf

rst2pdf is a tool for transforming reStructuredText to PDF using ReportLab. To install rst2pdf on Windows you also need Python because rst2pdf is coded in python.

Rst2pdf uses ReportLab, which can be installed using:

easy\_install reportlab

Again, in Windows, there may be a problem with the required Microsoft Visual Studio version. While running setup.py for package installations, Python 2.7 searches for an installed Visual Studio 2008. The solution is to define VS90COMNTOOLS variable to point to Tools directory of Visual Studio:

SET VS90COMNTOOLS=%VS100COMNTOOLS%

How to install rst2pdf on Windows?

Download rst2pdf source from <https://code.google.com/p/rst2pdf/downloads/list> Unzip the file to an rst2pdf folder. Goto the the rst2pdf folder which contains setup.py file. Run python setup.py install command and it will be installed. To convert any .rst file to PDF file Run rst2pdf myfile.rst command and you are done.

<http://rst2pdf.ralsina.me/> User Manual: <http://ralsina.me/static/manual.pdf>

## Pandoc

Pandoc is a tool that can convert between Markdown and a number of formats. You can use Pandoc to create Word and Powerpoint documents, PDFs, HTML, and several kinds of presentations. For longer documents, can concatenate Markdown files together.

Pandoc includes formatting tricks that let you do things like create columns or scale images, and you can use stylesheets from existing documents to give your content a specific look and feel.

You can also convert from various formats to Markdown, including doing things like grabbing a web page as a Markdown file. You can even try it online.

## ReST editor for Eclipse

The ReST editor for Eclipse is a plug-in for the Eclipse IDE. If Sphinx is installed, it can also be used to create (and build) Sphinx projects from within Eclipse. The following presentation documents the use of the editor.

This ReST editor has several advantages, namely:

integrated spell-checking using Hunspell4Eclipse contextual ReST syntax help sections outline rearrangement

This document opened in the Eclipse ReST Editor.

# Recipes

recipes-notes recipes-git-wiki recipes-centralized-workflow recipes-gitflow recipes-hugo recipes-sphinx recipes-pandoc-word recipes-pandoc-ebook recipes-pandoc-pdf recipes-pandoc-web recipes-slides recipes-slides-dzslides

You can do so many things with reST! Take notes? Git wiki? Manage docs with centralized and gitflow? Do Hugo? Sphinx? Pandoc word ebook pdf and web? Slides? DZSlides? Take notes ==========

<http://blog.getreu.net/projects/tp-note/tp-note--manual.html>

Tp-Note is a note-taking-tool and a template system - freely available for Windows, MacOS and Linux - that consistently synchronizes the note’s meta-data with its filename. If you like to keep your notes next to your files and you care about expressive filenames, then Tp-Note might be the tool of your choice. Tp-Note collects various information about its environment and the clipboard and stores them in variables. New notes are created by filling these variables in predefined and customizable Tera-templates. TP-Note’s default templates are written in Markdown and can be easily adapted to any other markup language if needed. By default, TP-Note launches the system file-editor (or any other of your choice, e.g. MarkText or Typora) after creating a new note.

But you can also just use ReText or Formiko

Atom, Sublime, VScode might give a better UI with the directory/file nav on the left.

Figure out an atom recipe

* rst-preview-pandoc and language-reStructuredText packages needed

Or there's <https://github.com/audreyfeldroy/sphinx-gui>

# Edit a Git wiki

\*\* Does this work? Try it \*\*

A Git repository comes with a wiki, where people can read and collaboratively edit documentation. You can create a wiki to document projects or code stored in the repository, or you can just use a repository for its wiki capability.

!!! note A Git wiki is a second repository attached to your repository. You clone, pull, and push to the main repository and the wiki separately.

A Git wiki uses [Git wiki structure](../../tools/tools-publishing#git-wiki-structure) and it’s one way to develop content for publishing in

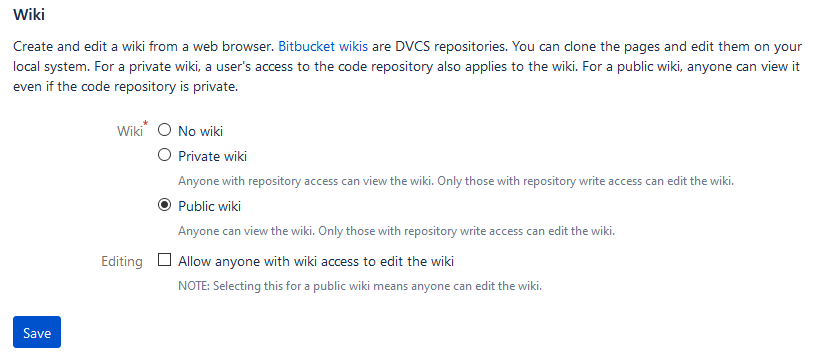
## Ingredients

Markdown editor

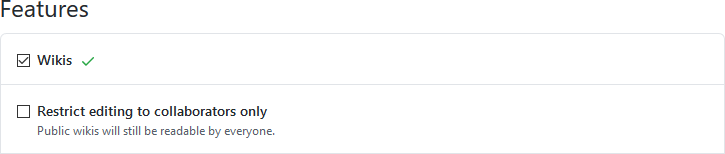
Git

## Set up a wiki on your hosted repository

The easiest way to set up the wiki is by logging onto your Git host and adding it there.

=== “Bitbucket and Sourcetree” 1. In a browser, log on to Bitbucket. 1. Under “Repository settings” look for “Features” and click **Wiki**. 1. Select **Public wiki** and save.  


=== “GitHub and GitHub Desktop” The GitHub documentation on [Wikis](https://docs.gitlab.com/ee/user/project/wiki/) is helpful. Here are the basic steps:

1. In a browser, log on to GitHub.
   1. Click the **Settings** button: Settings button
   2. Scroll down to “Features” and select **Wikis**: 

## Edit content on the host

If you just want to add a few pages to the wiki online, there’s no more setup to do! Just go to your repository, click **Wiki**, and you’ll see buttons for creating and editing pages.

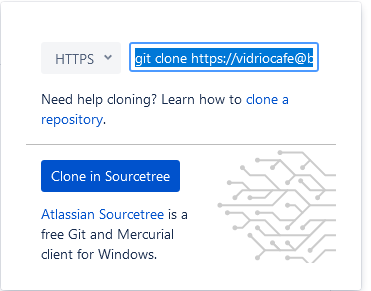
!!! hint To add a page in a new folder, make the folder part of the new filename. For example: morestuff/newpage.md adds newpage.md in a folder called morestuff.

## Clone the wiki to a local repository

There are advantages to working with wiki files locally, on your computer:

* It’s much easier to add folders and move files around
* You can work on it even when you’re offline
* You can use whatever Markdown editor you want
* Others can collaborate with you

To work on the wiki locally, clone the wiki to a local repository:

=== “Bitbucket and Sourcetree” 1. Go to your online repository and click **Wiki**. 2. Click **Clone wiki** then **Clone in Sourcetree**.  
 1. Make sure the local path shows the directory where you want to clone the repository, and click **Clone**.

=== “GitHub and GitHub Desktop” 1. Go to your online repository and click the **Wiki** button:  
Wiki button 1. Copy the **Clone this wiki locally** URL. 1. In GitHub Desktop, click **File > Clone repository**. 1. Paste the URL, make sure the local path shows the directory where you want to clone the repository, and click **Clone**.

=== “Linux command line” 1. Go to your online repository and click **Wiki**. 1. Copy the URL (or command and URL) to clone the wiki repository. 1. On the command line, navigate to the directory where you want to clone the repository. 1. Use git clone and the URL to clone the repository. Example:  
git clone https://my\_name@bitbucket.org/my\_name/markdown-stuff.git/wiki

## Work with the content locally

On your computer, go to the directory where you cloned the wiki. There should be a directory called wiki containing a file called Home.md containing the Markdown source for the welcome page the wiki displayed when you created it online.

You can now work with the wiki using the [Git centralized workflow](../../tools/tools-git-basics#centralized-workflow) or the [GitHub flow](../../tools/tools-git-basics#github-flow), using the familiar cycle of pull, edit, commit, push.

## Tutorial

Here’s a quick tutorial that shows how to organize pages in the wiki.

##### Try creating some content

Try making the following changes:

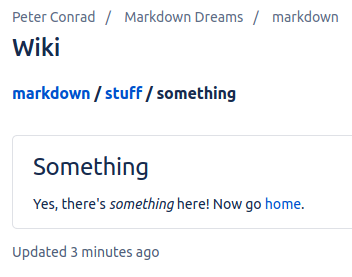
* Add a folder called stuff.
* Using your favorite Markdown editor, make a file called something.md inside stuff, with the following contents:
* :
* # Something  
    
  Yes, there's \*something\* here! Now go [home](../Home).

You should now have a directory structure that looks like this:

Home.md  
stuff/  
 something.md

##### Take a look

After you commit and push the changes, take a look in your online wiki:

1. Go to your online repository and click **Wiki**.
2. View the page tree of the wiki. For example:
   * In Bitbucket, click the name of the wiki.
   * In GitHub, click **Pages**.
3. Navigate to the page you created. 
4. Try the home link.

## Next steps

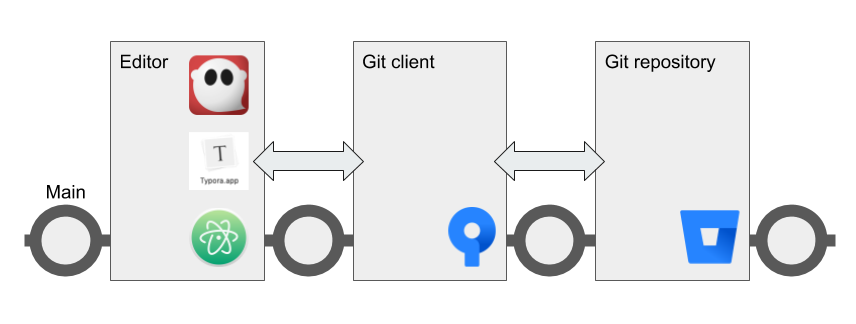
Try the following recipe, which is very similar:

* [Collaborate using the Git centralized workflow](../recipes-centralized-workflow/)

# Collaborate using centralized Git workflow

You can use Git to help a team collaborate on informal or internal documents in a very simple way.

The centralized workflow can be an effective way for content creators, managers, and engineers to collaborate on non-production content such as specifications, planning documents, newsletters, internal documentation, and the like.



In this workflow, everyone works on the same branch. This makes things easy for simple collaboration.

Note

If you’re publishing a large website or documentation set in a production environment with with a substantial number of collaborators,you should consider using [GitHub Flow](../recipes-gitflow/) instead.

## Ingredients

* reStructuredText editor
* Git
* Sphinx or Pandoc

Hint

Plan ahead based on the publishing tool you want to use. If you’re planning to use a wiki, MkDocs, or Hugo, organize your docs in [Git wiki structure](../../tools/tools-publishing#git-wiki-structure) and add YAML frontmatter from the start. If you’re creating large documents with Pandoc, think about whether you need to work on content in pieces and then use Pandoc to assemble everything.

## Working with content

This recipe uses the centralized Git workflow. [Pick an editor](tools-editors/), then make sure you have Git set up and try each step below. The steps work together like a heartbeat that keeps content safe and synchronized for all collaborators.

Hint

If you use a code editor such as Atom, you can keep track of your changes as you edit your reStructuredText files.

1. [Pull](pull)
2. Fetch the latest changes from the remote repository to the local repository on your computer
3. Work
4. Edit your content in your favorite Markdown editor
5. [Stage and commit](stage-and-commit)
6. From time to time, in your Git client, type a short sentence about what you've done and save the changes to Git
7. [Push](push)
8. When your work is final, push it up to the remote repository

If you need to publish, you can use Sphinx to create a website or ePub book, or Pandoc to convert to other formats.

Hint

Remember not to publish content into a Git repository. The repo is only for storing your reStructuredText and other source files.

See also:

* publishing-tools

# Manage docs with GitHub Flow

Once your team or project reaches a certain size, branching is a good way to keep people from accidentally interfering with each other’s work. It’s also a great way to stage and test code or content before migrating to production. There are many branching strategies, some of them quite complicated. The one I like, and that I have used for documentation in the past, is called [GitHub Flow](https://scottchacon.com/2011/08/31/github-flow.html). It’s simple but effective. The **Pull-Work-Commit-Push** steps happen within a working branch.

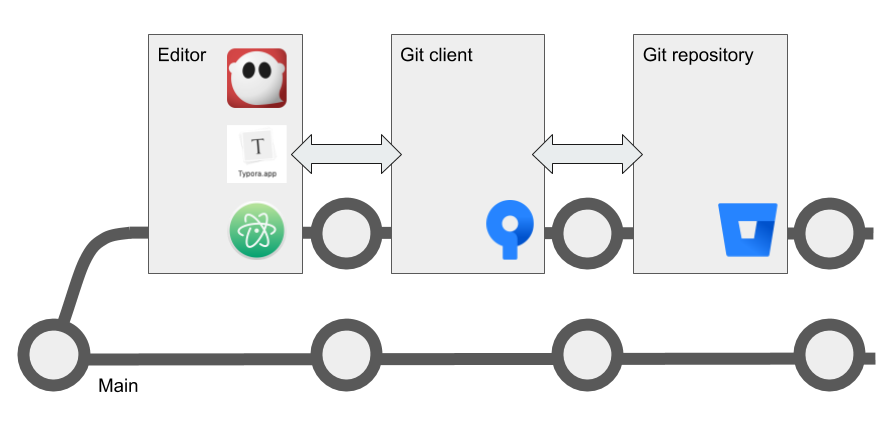


Diagram of GitHub flow

## What you need

reStructuredText editor

Git

Publishing tools

!!! hint Plan ahead based on the publishing tool you want to use. If you’re planning to use a wiki, MkDocs, or Hugo, organize your docs in [Git wiki structure](../../tools/tools-publishing#git-wiki-structure) and add YAML frontmatter from the start. If you’re creating large documents with Pandoc, think about whether you need to work on content in pieces and then use Pandoc to assemble everything.

## Working with content

[Pick an editor](../../tools/tools-editors/), then make sure you have Git set up and try each step below. The steps work together like a heartbeat that keeps content safe and synchronized for all collaborators. –8<– “github-flow-snippet.html”

!!! hint After you’re done with that part of the project, you can delete your working branch or keep it around for further work. Before starting work on a different part of the project, remember to switch to main and pull again.

If you need to [Publish content](../../tools/tools-publishing/), you can use one of these recipes:

* [Publish documentation with MkDocs](../recipes-mkdocs/)
* [Publish a website with Hugo](../recipes-mkdocs/)
* [Create a document with Pandoc](../recipes-pandoc/)

!!! hint Remember not to publish content into a Git repository. The repo is only for storing your reStructuredText and other source files.

## Next steps

Check out the following recipes :

* [Manage docs with Gitflow](../recipes-gitflow/)
* [Create an eBook](../recipes-pandoc-ebook/)
* [Write a Word doc](../recipes-pandoc-word/)
* [Publish a PDF](../recipes-pandoc-pdf/)
* [Present slides](../recipes-slides/)

# Publish a website with Hugo

<https://github.com/fisodd/hugo-restructured>

[Hugo](https://gohugo.io/) is a very powerful open-source static site generator that includes tools for organizing content, adding extensions, and even creating dynamic logic. There is way too much to document here, so the focus of this recipe will be getting started and a few basics.

To install Hugo, use your operating system’s package manager:

* Linux: [apt-get](https://help.ubuntu.com/community/AptGet/Howto) or [yum](http://yum.baseurl.org/)
* macOS: [Homebrew](https://brew.sh/)
* Windows: [Chocolatey](https://chocolatey.org/)

## Ingredients

reStructuredText editor

Hugo

Git (optional)

!!! hint Hugo works best with reStructuredText files that are organized in [Git wiki structure](../../tools/tools-publishing#git-wiki-structure). You can even use [Git wiki](../recipes-git-wiki/) to develop the content, so long as you add YAML frontmatter as you go.

This recipe goes well with:

* [Centralized Git workflow](../recipes-centralized-workflow/)
* [GitHub flow](../recipes-gitflow/)

## Setting up your site

Once Hugo is installed, you can create a new site by typing hugo new site my-project, which provides some instructions when it runs:

$ hugo new site my-project  
Congratulations! Your new Hugo site is created in /home/pconrad/git/my-project.  
  
Just a few more steps and you're ready to go:  
  
1. Download a theme into the same-named folder.  
 Choose a theme from https://themes.gohugo.io/ or  
 create your own with the "hugo new theme <THEMENAME>" command.  
2. Perhaps you want to add some content. You can add single files  
 with "hugo new <SECTIONNAME>/<FILENAME>.<FORMAT>".  
3. Start the built-in live server via "hugo server".  
  
Visit https://gohugo.io/ for quickstart guide and full documentation.

!!! Hint For source control and collaboration, create the project in a Git repo.

## Adding a theme

To add a theme, you use the git submodule add command from within your site directory. Each theme in Hugo’s official [Complete List](https://themes.gohugo.io/) includes instructions for adding the submodule from the correct Git repository. Example:

$ git submodule add https://github.com/budparr/gohugo-theme-ananke.git themes/ananke  
Cloning into '/home/pconrad/git/my-project/themes/ananke'...  
remote: Enumerating objects: 8, done.  
remote: Counting objects: 100% (8/8), done.  
remote: Compressing objects: 100% (8/8), done.  
remote: Total 1839 (delta 2), reused 1 (delta 0), pack-reused 1831  
Receiving objects: 100% (1839/1839), 4.33 MiB | 1.26 MiB/s, done.  
Resolving deltas: 100% (1022/1022), done.

After you’ve installed the theme, add it to the configuration file config.toml. Example:

$ echo 'theme = "ananke"' >> config.toml

Edit the config.toml file to change other things about the site, such as the site title or the base URL.

## Working with content

You can create a new page with hugo new <path>. For example, to create a new post in the posts directory, type:

hugo new posts/my-first-post.md

When it’s created, all this page contains is YAML frontmatter:

---  
title: "My First Post"  
date: 2020-08-28T18:42:02-07:00  
draft: true  
---

You must add reStructuredText content before you can preview the page. A page in Hugo must contain both frontmatter and reStructuredText content. If either is missing, the page shows 404 page not found when you try to preview it.

Frontmatter can signal content status, including publish and expiry dates, and can contain variables. You can use the variables in templates and in content by creating your own Hugo shortcodes. Variables and shortcodes are outside the scope of this recipe, but the Hugo website has a lot of documentation.

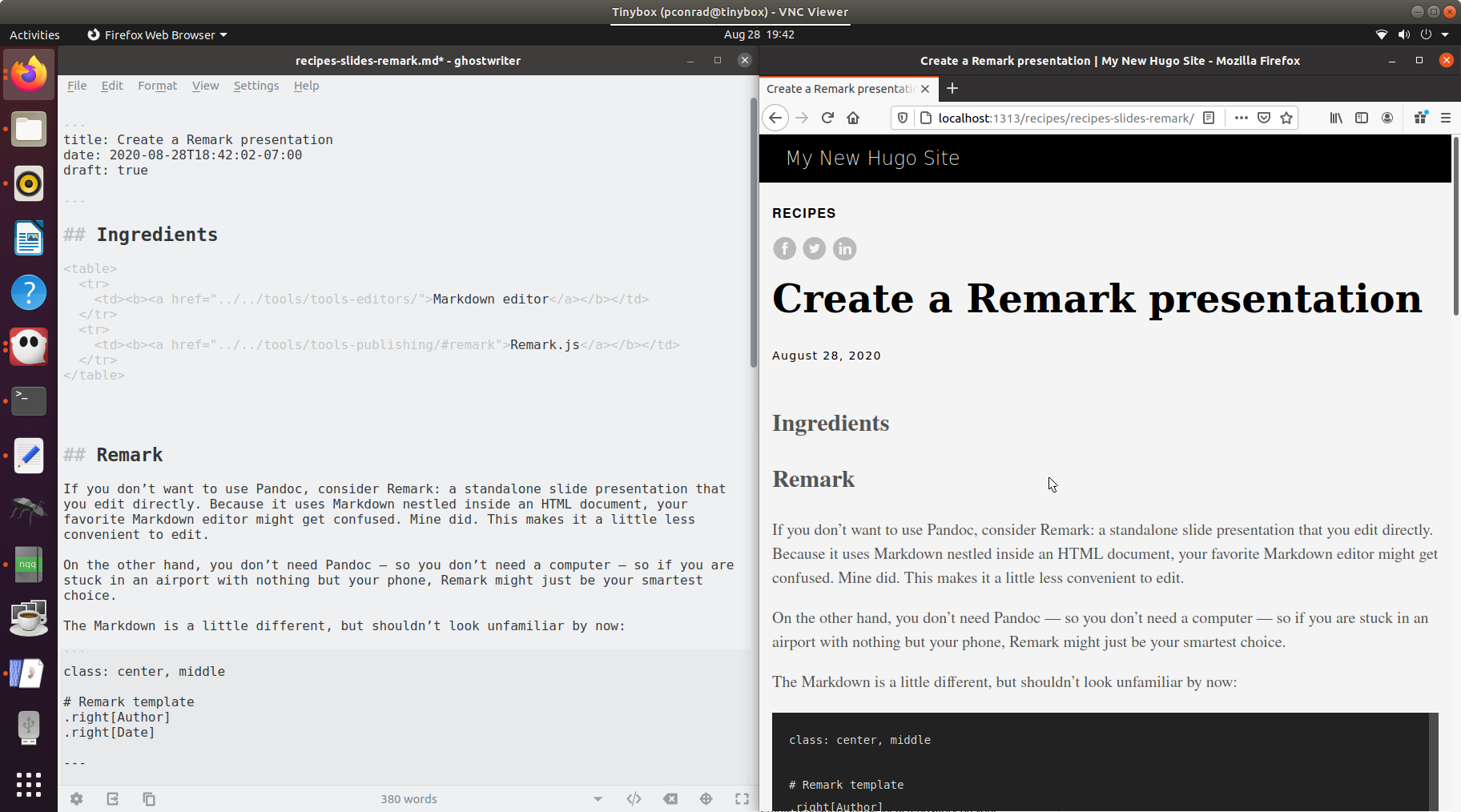
If you are migrating content into Hugo from a Git wiki, MkDocs site, or some other source, you will need to add frontmatter to any pages that don’t already have it.

## Local preview

To preview your content, start the Hugo server:

$ hugo server -D

The -D option tells Hugo to include draft pages in the preview. If you omit this option, any page with draft:true in the frontmatter is ignored.



Screenshot of Hugo live preview

Hugo is a bit finicky about previewing and building content, and there are a few reasons why pages might not show up in the preview. A first troubleshooting step is to make sure each file contains both frontmatter and reStructuredText, and that it doesn’t have a publish date in the future or expiry date in the past.

##### Images

Images in Hugo normally go in the static directory. Here, I’ve added a subdirectory called images and when I add the image in my reStructuredText editor it looks like this:

::

An image

That makes sense, since that is the correct relative path to the file. When the site is built for local preview, however, the result is this:

Unfortunately, that won’t work. The actual relative path to the image is: ../../images/whatever.png instead.

In other words, when you add an image whose relative path is correct with regard to the reStructuredText file, you must remove static/ from the path to make it work in the preview and the built site. If you are using an editor that lets you preview the images as you work, then you have to break all the images to get them to work in Hugo.

One solution might be to upload all your images to the web first, then use absolute paths—but this will make it more difficult to move the website or change the paths later.

## Building and publishing

The hugo command builds the website in a directory called public. To publish the site, use FTP to transfer the contents of that directory to a folder on a webserver.

!!! hint To prevent Git from tracking changes to the public directory, create a file called .gitignore at the top level directory of the Hugo project with the following contents:

public/

If you use git add to add your .gitignore file to change tracking, then it will apply to anyone who clones the repo—meaning that no one will add built HTML pages to Git.

# Build docs with Sphinx

<https://thomas-cokelaer.info/tutorials/sphinx/quickstart.html>

Sphinx focuses on documentation, in particular handwritten documentation, however, Sphinx can also be used to generate blogs, homepages and even books.

The root directory of a Sphinx collection of plain-text document sources is called the source directory. This directory also contains the Sphinx configuration file conf.py, where you can configure all aspects of how Sphinx reads your sources and builds your documentation. 1

Sphinx comes with a script called sphinx-quickstart that sets up a source directory and creates a default conf.py with the most useful configuration values from a few questions it asks you.

Every page needs a title! And must be in a toctree, all or nothing

toctree is a reStructuredText directive, a very versatile piece of markup. Directives can have arguments, options and content.

Arguments are given directly after the double colon following the directive?s name. Each directive decides whether it can have arguments, and how many.

Options are given after the arguments, in form of a ?field list?. The maxdepth is such an option for the toctree directive.

Content follows the options or arguments after a blank line. Each directive decides whether to allow content, and what to do with it.

Originally, Sphinx was conceived for a single project, the documentation of the Python language. Shortly afterwards, it was made available for everyone as a documentation tool, but the documentation of Python modules remained deeply built in ? the most fundamental directives, like function, were designed for Python objects. Since Sphinx has become somewhat popular, interest developed in using it for many different purposes: C/C++ projects, JavaScript, or even reStructuredText markup (like in this documentation).

While this was always possible, it is now much easier to easily support documentation of projects using different programming languages or even ones not supported by the main Sphinx distribution, by providing a domain for every such purpose.

A domain is a collection of markup (reStructuredText directives and roles) to describe and link to objects belonging together, e.g. elements of a programming language. Directive and role names in a domain have names like domain:name, e.g. py:function. Domains can also provide custom indices (like the Python Module Index).

Having domains means that there are no naming problems when one set of documentation wants to refer to e.g. C++ and Python classes. It also means that extensions that support the documentation of whole new languages are much easier to write.

While Docutils provides a number of directives, Sphinx provides many more and uses directives as one of the primary extension mechanisms.

Adding index terms to a paragraph in Sphinx looks like:

.. index::  
 single: Programming languages  
 single: Compiling  
 single: Source code

Since reST does not have facilities to interconnect several documents, or split documents into multiple output files, Sphinx uses a custom directive to add relations between the single files the documentation is made of, as well as tables of contents. The toctree directive is the central element.

* The goal of this document is to give you a quick taste of what Sphinx is and how you might use it. When you?re done here, you can check out the installation guide followed by the intro to the default markup format used by Sphinx, reStucturedText.

<https://www.sphinx-doc.org/en/master/usage/quickstart.html#setting-up-the-documentation-sources>

* Sphinx comes with a script called sphinx-quickstart that sets up a source directory and creates a default conf.py with the most useful configuration values from a few questions it asks you. To use this, run:

$ sphinx-quickstart

Note

Just do quickstart. Then you get the make file.

Let?s assume you?ve run sphinx-quickstart. It created a source directory with conf.py and a master document, index.rst. The main function of the master document is to serve as a welcome page, and to contain the root of the ?table of contents tree? (or toctree). This is one of the main things that Sphinx adds to reStructuredText, a way to connect multiple files to a single hierarchy of documents.

The next thing you should do is set up the folders yo uwant with some index files in them and start figuring out the structure of the content. Then you want to deal with the TOC.

## Dealing with the TOC

Sphinx has this TOC thing. Unline Git Wiki format, if you want a folder, it's a parent page (like Confluence) not just a folder (like Git Wiki). So you have a couple choices:

* Everything is flat, with a TOC on the index page
* Everthing's in folders but the navigation is flat
* Nice hierarchical navigation

The way I found to do that last one is to put things in folders, then every folder gets an index.rst page. Because it's really a page, you need some content there.

On the top level index.rst, you include whatever top-level pages you want, including the indexes in the folders:

.. toctree::  
 :includehidden:  
  
 02-getting-started  
 tools/index  
 recipes/index  
 resources/index

Then on each index in a folder, you do this:

.. toctree::  
 :hidden:  
  
 22-restructuredtext-cheatsheet  
 23-templates  
 24-resources-glossary  
 25-links

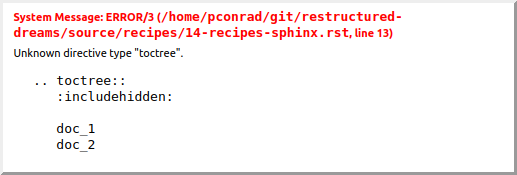
Finally, on those leaf pages, you do this:

.. toctree::  
 :hidden:

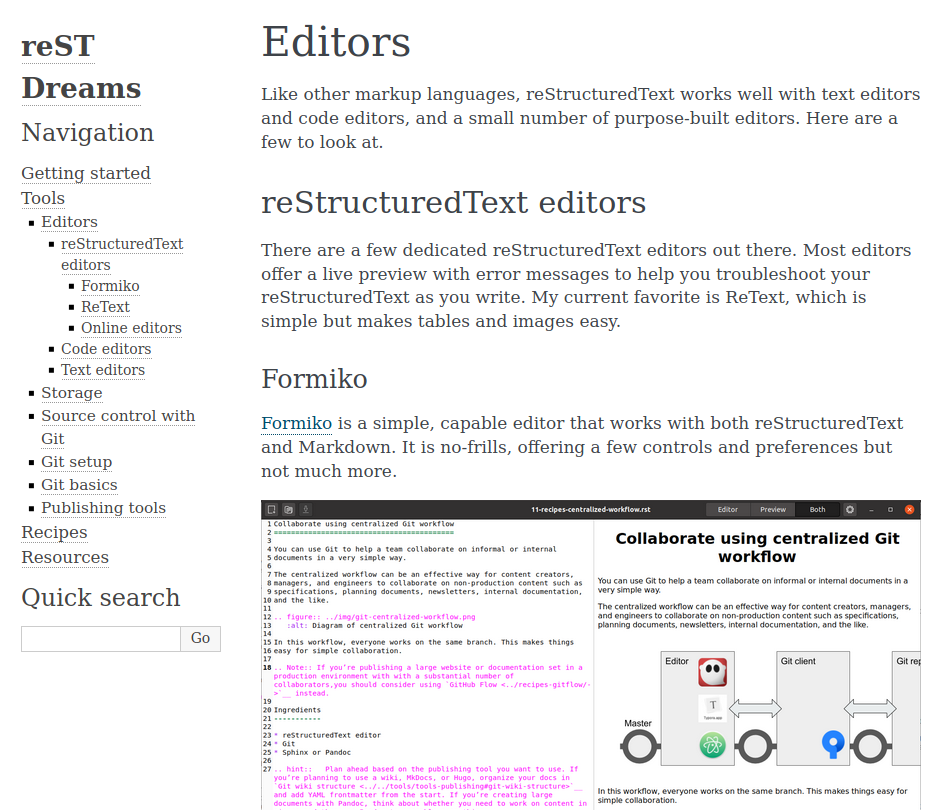
If you don't put a .. toctree:: on every damn page, then your left nav goes bonkers when you hit a page without one. And you get a build warning too:

/home/pconrad/git/restructured-dreams/source/recipes/index.rst:4: WARNING: toctree contains reference to document 'recipes/19-recipes-slides' that doesn't have a title: no link will be generated

Also don't forget that not all editors (based on DocUtils) understand the Sphinx directives, so you'll get this:



But if you do it right, you get this nice expanding/collapsing nav:



## Pick a theme

<https://www.sphinx-doc.org/en/master/usage/theming.html>

Some themes aer built in. Go look at <https://www.sphinx-doc.org> to find a list and then just change the html\_theme parameter in source/conf.py

You can also set theme-specific options using the html\_theme\_options config value. These options are generally used to change the look and feel of the theme.

See <https://www.sphinx-doc.org/en/master/usage/configuration.html#confval-html_theme_options>

You can use non-built-in themes

If the theme does not come with Sphinx, it can be in two static forms or as a Python package. For the static forms, either a directory (containing theme.conf and other needed files), or a zip file with the same contents is supported. The directory or zipfile must be put where Sphinx can find it; for this there is the config value html\_theme\_path. This can be a list of directories, relative to the directory containing conf.py, that can contain theme directories or zip files. For example, if you have a theme in the file blue.zip, you can put it right in the directory containing conf.py and use this configuration:

html\_theme = "blue" html\_theme\_path = ["."]

The third form is a Python package. If a theme you want to use is distributed as a Python package, you can use it after installing

# installing theme package $ pip install sphinxjp.themes.dotted

Once installed, this can be used in the same manner as a directory or zipfile-based theme:

html\_theme = "dotted"

## Doing a build

$ sphinx-build -b html source/ build/

$ make html

## Make targets

$ make  
Sphinx v3.2.1  
Please use `make target' where target is one of  
 html to make standalone HTML files  
 dirhtml to make HTML files named index.html in directories  
 singlehtml to make a single large HTML file  
 pickle to make pickle files  
 json to make JSON files  
 htmlhelp to make HTML files and an HTML help project  
 qthelp to make HTML files and a qthelp project  
 devhelp to make HTML files and a Devhelp project  
 epub to make an epub  
 latex to make LaTeX files, you can set PAPER=a4 or PAPER=letter  
 latexpdf to make LaTeX and PDF files (default pdflatex)  
 latexpdfja to make LaTeX files and run them through platex/dvipdfmx  
 text to make text files  
 man to make manual pages  
 texinfo to make Texinfo files  
 info to make Texinfo files and run them through makeinfo  
 gettext to make PO message catalogs  
 changes to make an overview of all changed/added/deprecated items  
 xml to make Docutils-native XML files  
 pseudoxml to make pseudoxml-XML files for display purposes  
 linkcheck to check all external links for integrity  
 doctest to run all doctests embedded in the documentation (if enabled)  
 coverage to run coverage check of the documentation (if enabled)

If you want to link to other files, use ref there's the general :ref: directive, documented here. They give this example:

.. \_my-reference-label:  
  
Section to cross-reference  
--------------------------  
  
This is the text of the section.  
  
It refers to the section itself, see :ref:`my-reference-label`.

Although the general hyperlinking mechanism offered by RST does work in Sphinx, the documentation recommends against using it when using Sphinx:

Using ref is advised over standard reStructuredText links to sections (like Section title) because it works across files, when section headings are changed, and for all builders that support cross-references.

The Sphinx documentation generator provides a more flexible alternative to definition lists (see Glossaries). Glossaries

The Sphinx ..glossary:: directive contains a reST definition-list-like markup with terms and definitions.

See the following example:

.. glossary::  
  
 environment  
 A structure where information about all documents under the root is  
 saved, and used for cross-referencing. The environment is pickled  
 after the parsing stage, so that successive runs only need to read  
 and parse new and changed documents.  
  
 source directory  
 The directory which, including its subdirectories, contains all  
 source files for one Sphinx project.

The definitions will then be used in cross-references with the :term: role. For example:

The :term:source directory for this project is ...

In contrast to regular definition lists, a glossary supports multiple terms per entry and inline markup is allowed in terms. You can link to all of the terms. For example:

.. glossary::  
  
 term 1  
 term 2  
 Definition of both terms.

When the glossary is sorted, the first term determines the sort order.

To automatically sort a glossary, include the following flag:

.. glossary::  
 :sorted:

## A build

$ sphinx-build -b html source/ build/  
Running Sphinx v3.2.1  
loading pickled environment... done  
building [mo]: targets for 0 po files that are out of date  
building [html]: targets for 2 source files that are out of date  
updating environment: 1 added, 4 changed, 0 removed  
reading sources... [100%] tools/index   
/home/pconrad/git/restructured-dreams/source/recipes/13-recipes-hugo.rst:: WARNING: image file not readable: recipes/../../static/images/whatever.png  
/home/pconrad/git/restructured-dreams/source/resources/22-restructuredtext-cheatsheet.rst:45: WARNING: image file not readable: path/to/image.jpg  
looking for now-outdated files... none found  
pickling environment... done  
checking consistency... /home/pconrad/git/restructured-dreams/source/01-index.rst: WARNING: document isn't included in any toctree  
/home/pconrad/git/restructured-dreams/source/rstest.rst: WARNING: document isn't included in any toctree  
done  
preparing documents... done  
writing output... [100%] tools/index   
generating indices... genindexdone  
writing additional pages... searchdone  
copying images... [100%] recipes/../../img/hugo-preview.png   
copying static files... ... done  
copying extra files... done  
dumping search index in English (code: en)... done  
dumping object inventory... done  
build succeeded, 4 warnings.  
  
The HTML pages are in build

## Another build

$ make html  
Running Sphinx v3.2.1  
loading pickled environment... done  
building [mo]: targets for 0 po files that are out of date  
building [html]: targets for 1 source files that are out of date  
updating environment: 0 added, 9 changed, 0 removed  
reading sources... [100%] tools/07-tools-git-basics   
/home/pconrad/git/restructured-dreams/source/02.5-basics.rst:508: WARNING: Footnote [#] is not referenced.  
/home/pconrad/git/restructured-dreams/source/02.5-basics.rst:509: WARNING: Footnote [#] is not referenced.  
/home/pconrad/git/restructured-dreams/source/02.5-basics.rst:: WARNING: image file not readable: my-image.png  
/home/pconrad/git/restructured-dreams/source/02.5-basics.rst:411: WARNING: image file not readable: gnu.png(options)  
/home/pconrad/git/restructured-dreams/source/02.5-basics.rst:: WARNING: image file not readable: picture.png  
/home/pconrad/git/restructured-dreams/source/02.5-basics.rst:: WARNING: image file not readable: tent.png  
/home/pconrad/git/restructured-dreams/source/02.5-basics.rst:: WARNING: image file not readable: waves.png  
/home/pconrad/git/restructured-dreams/source/02.5-basics.rst:: WARNING: image file not readable: peak.png  
/home/pconrad/git/restructured-dreams/source/recipes/10-recipes-git-wiki.rst:197: WARNING: image file not readable: recipes/../img/recipes-git-wiki-create.png  
/home/pconrad/git/restructured-dreams/source/recipes/10-recipes-git-wiki.rst:198: WARNING: image file not readable: recipes/../img/github-settings.png  
/home/pconrad/git/restructured-dreams/source/recipes/10-recipes-git-wiki.rst:199: WARNING: image file not readable: recipes/../img/git-wikis.png  
/home/pconrad/git/restructured-dreams/source/recipes/10-recipes-git-wiki.rst:200: WARNING: image file not readable: recipes/../img/recipes-git-repo-bb-clone-wiki.png  
/home/pconrad/git/restructured-dreams/source/recipes/10-recipes-git-wiki.rst:201: WARNING: image file not readable: recipes/../img/github-wiki-button.png  
/home/pconrad/git/restructured-dreams/source/recipes/10-recipes-git-wiki.rst:202: WARNING: image file not readable: recipes/../img/recipes-git-wiki-bb-something.png  
/home/pconrad/git/restructured-dreams/source/recipes/11-recipes-centralized-workflow.rst:: WARNING: image file not readable: recipes/../img/git-centralized-workflow.png  
/home/pconrad/git/restructured-dreams/source/recipes/12-recipes-gitflow.rst:17: WARNING: image file not readable: recipes/../img/github-flow.png  
/home/pconrad/git/restructured-dreams/source/recipes/13-recipes-hugo.rst:199: WARNING: image file not readable: recipes/../img/hugo-preview.png  
/home/pconrad/git/restructured-dreams/source/recipes/13-recipes-hugo.rst:217: WARNING: image file not readable: recipes/../../static/images/whatever.png  
/home/pconrad/git/restructured-dreams/source/recipes/20-recipes-slides-dzslides.rst:110: WARNING: image file not readable: recipes/../img/slides-dzslides.png  
/home/pconrad/git/restructured-dreams/source/recipes/20-recipes-slides-dzslides.rst:153: WARNING: image file not readable: recipes/../img/slides-dzslides-images.png  
/home/pconrad/git/restructured-dreams/source/resources/22-restructuredtext-cheatsheet.rst:45: WARNING: image file not readable: path/to/image.jpg  
/home/pconrad/git/restructured-dreams/source/tools/07-tools-git-basics.rst:33: WARNING: image file not readable: tools/../img/git-centralized-workflow.png  
/home/pconrad/git/restructured-dreams/source/tools/07-tools-git-basics.rst:64: WARNING: image file not readable: tools/../img/github-flow.png  
looking for now-outdated files... none found  
pickling environment... done  
checking consistency... /home/pconrad/git/restructured-dreams/source/01-index.rst: WARNING: document isn't included in any toctree  
/home/pconrad/git/restructured-dreams/source/02.5-basics.rst: WARNING: document isn't included in any toctree  
/home/pconrad/git/restructured-dreams/source/rstest.rst: WARNING: document isn't included in any toctree  
done  
preparing documents... done  
writing output... [100%] tools/index   
/home/pconrad/git/restructured-dreams/source/02.5-basics.rst:: WARNING: Could not obtain image size. :scale: option is ignored.  
/home/pconrad/git/restructured-dreams/source/recipes/index.rst:4: WARNING: toctree contains reference to document 'recipes/19-recipes-slides' that doesn't have a title: no link will be generated  
/home/pconrad/git/restructured-dreams/source/recipes/index.rst:4: WARNING: toctree contains reference to document 'recipes/19-recipes-slides' that doesn't have a title: no link will be generated  
/home/pconrad/git/restructured-dreams/source/recipes/index.rst:4: WARNING: toctree contains reference to document 'recipes/19-recipes-slides' that doesn't have a title: no link will be generated  
/home/pconrad/git/restructured-dreams/source/recipes/index.rst:4: WARNING: toctree contains reference to document 'recipes/19-recipes-slides' that doesn't have a title: no link will be generated  
/home/pconrad/git/restructured-dreams/source/recipes/index.rst:4: WARNING: toctree contains reference to document 'recipes/19-recipes-slides' that doesn't have a title: no link will be generated  
/home/pconrad/git/restructured-dreams/source/recipes/index.rst:4: WARNING: toctree contains reference to document 'recipes/19-recipes-slides' that doesn't have a title: no link will be generated  
/home/pconrad/git/restructured-dreams/source/recipes/index.rst:4: WARNING: toctree contains reference to document 'recipes/19-recipes-slides' that doesn't have a title: no link will be generated  
generating indices... genindexdone  
writing additional pages... searchdone  
copying static files... ... done  
copying extra files... done  
dumping search index in English (code: en)... done  
dumping object inventory... done  
build succeeded, 34 warnings.  
  
The HTML pages are in build/html.

# Overview

You can use Pandoc to create a Word document that other people can work on, for upload to Google Drive, or for use with other publishing tools.

## Ingredients

reStructuredText editor

Pandoc

Word, Google Drive, or LibreOffice

## Creating the Word document

The command for creating the document is simple. With a single reStructuredText file, it looks like this:

pandoc -o my\_document.docx reStructuredText.rst

##### Special Pandoc formatting

Pandoc includes a number of formatting tricks that you might find useful. One of the most useful is fenced div syntax, which uses groups of colons.

You can use fenced div syntax to create columns using nested divs without writing <div> tags in HTML. Take a look at this example:

:::::::::::::: {.columns}  
::: {.column width="50%"}  
  
Left column:  
  
- Bullet  
- Bullet  
- Bullet  
  
:::  
::: {.column width="50%"}  
  
![](bench.jpg)  
  
:::  
::::::::::::::

That translates to a <div class="columns"> containing two <div class="column"> tags. Pandoc uses these to create two columns in the Word doc. Each div can be signified with as few as three colons in a row; in the example, the outer div uses more colons for readability.

You can use curly braces to define attributes such as identifiers, classes, and key/value pairs on headers, images, and a few other elements in Pandoc. If you’re using Pandoc to create long Word documents, this is handy because you can set anchors on headings and link to them internally.

You can also use an attribute to scale an image:

![Alt text](bench.jpg){width=25%}

When Pandoc renders the image, it is scaled to a percentage of the container where it resides (a column, for example). The alt text is used for a caption.

##### Concatenating multiple files

You can specify as many files as you like. For example:

pandoc -o my\_document.docx chapter\_1.md chapter\_2.md

##### Title file

You can specify a title, author, licensing, and other information about the book in a file called title.txt at the front of the book, containing YAML that Pandoc uses when it generates the file.

Example:

---  
title: How to do things with reStructuredText and Git  
author: Peter S. Conrad  
language: en-US  
...

Creating a Word doc with a title file and multiple reStructuredText files looks like this:

pandoc -o my\_document.docx title.txt chapter\_1.md chapter\_2.md

##### Images

When your reStructuredText includes images, use relative paths. For example:

![An image](../images/whatever.png)

In the above example, the images directory is at the same level as the file containing the reStructuredText file; the relative path goes up a directory from the reStructuredText file and then down into the images directory to find the image.

When Pandoc follows these relative links, it starts from the directory where you typed the pandoc command. If you want Pandoc to find your images, either run the command from a directory where the relative links to the images make sense, or copy the images to a place where the relative links can find them.

##### Using a reference document

When you use Pandoc to convert reStructuredText to Word, you can apply the theme and styles from another Word document called a *reference document.* For example:

pandoc --reference-doc another.docx -o my\_document.docx title.txt chapter\_1.md chapter\_2.md

# Create an ePub book

You can use Pandoc to create an ePub book suitable for tablets and phones or for self-publishing on [Amazon](https://kdp.amazon.com/en_US/bookshelf).

There's also <https://github.com/mattharrison/rst2epub2>

a binary, rst2epub, to convert rst files into epub2 compliant files (ie that pass epub check, can be loaded into Apple, BN, Kobo, etc. Or converted to mobi and thrown into AMZN) a library, epublib, that has the ability to programatically create epub files. See the test function in epublib/epub.py for more details. There is experimental support for KF8 fixed layout as well.

And Sphinx

<https://pypi.org/project/epubmaker/> Overview ========

You can use Pandoc to create a PDF, which is a pleasant way to share content that you don’t want other people to edit along the way.

# Grab a web page with Pandoc

Converting a web page to reStructuredText on the fly is a small task that can be very helpful if you need to grab content for editing or add information from the web to your notes.

# Slides

rst2html5slides - Presentations from restructuredtext files ... rst2html5slides.readthedocs.io They have new and great visual effects such as slide transitions and positional effects; A web presentation is portable. It runs on any modern browser. On the other ...

Easy Slide Shows With reST & S5 - Docutils - SourceForge docutils.sourceforge.io ? docs ? user ? slide-shows Mar 30, 2006 ? Uses normal reStructuredText as input. One section per slide. Each first-level section is converted into a single slide. XHTML output.

hovercraft/presentations.rst at master · regebro/hovercraft ... github.com ? regebro ? hovercraft ? blob ? master ? docs Calling them "slides" is not really relevant in an impress.js context, as they can overlap and doesn't necessarily slide. The name "steps" is better, but it's also more ...

darkslide · PyPI pypi.org ? project ? darkslide ReStructuredText. Your ReST source files must be suffixed by .rst or .rest (.txt is not supported); Use headings for slide titles; Separate your slides using an ...

hieroglyph: Easy, Beautiful Slides with Restructured Text ... www.yergler.net ? 2012/03/13 ? hieroglyph Mar 13, 2012 ? hieroglyph: Easy, Beautiful Slides with Restructured Text. I was happy to have my talk proposal accepted for PyCon this year, and happy with ...

Using rst for presentations - MorganGoose.com morgangoose.com ? blog ? 2010/09/12 ? using-rst-for-... Sep 12, 2010 ? S5 is a slide show format based entirely on XHTML, CSS, and JavaScript. And rst2s5 takes a reStructuredText document and complies it into the ...

Hieroglyph Documentation - Read the Docs readthedocs.org ? hieroglyph ? downloads ? pdf ? latest PDF Jul 4, 2020 ? In addition to mapping ReStructured Text sections to slides, you can create a slide at any point in your document using the slide directive. The ...

Creating presentations using restructured text | Ralsina.Me ralsina.me ? stories ? BBS52 May 21, 2009 ? So, how do you create slides using rst2pdf? It's very simple. So simple, I will use an example file. This is my first slide -------- ... Create a DZSlides presentation ==============================

You can use Pandoc to create a standalone slide presentation in DZSlides format, which is useful for presenting when you aren’t sure what software will be available. DZSlides creates bold, attractive slides that discourage the use of too much text.

## Ingredients

Markdown editor

Pandoc

A browser

## Markdown for DZSlides

Start your Markdown file with YAML metadata or a simple block like this:

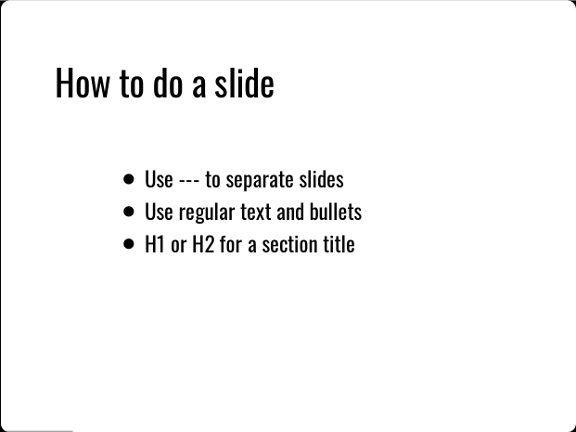
% Title  
% Author Name  
% Date

DZSlides uses the horizontal rule (---) as a separator between slides. A level one or level two heading is a section title.

Here’s some sample Markdown:

---  
  
# H1 or H2 is a Section Title  
  
---  
  
Normal Text or Slide Title  
  
- Bullet  
- Bullet  
  
---

Normal text is big enough for a title or announcement on a slide.



Screenshot of a DZSlides slide

The layout of slides in DZslides is simple; you probably won’t find yourself using tables or columns a lot.

##### Images

When your Markdown includes images, use relative paths. For example:

![An image](../images/whatever.png)

In the above example, the images directory is at the same level as the file containing the Markdown file; the relative path goes up a directory from the Markdown file and then down into the images directory to find the image.

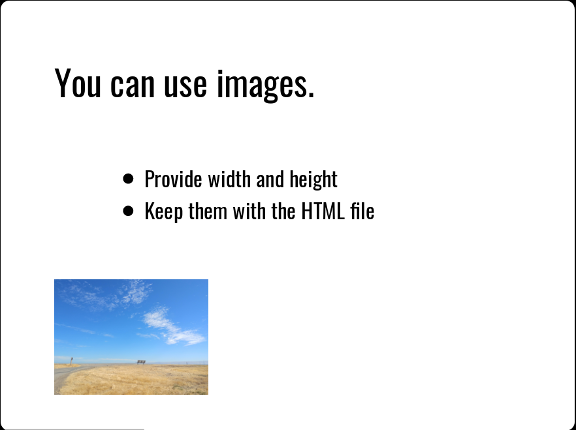
When Pandoc follows these relative links, it starts from the directory where you typed the pandoc command. If you want Pandoc to find your images, either run the command from a directory where the relative links to the images make sense, or copy the images to a place where the relative links can find them.

After you create the presentation, which is an HTML file, you need to keep the images and the presentation together. If you copy your presentation to a thumb drive without the images, the images won’t work.

!!! hint It’s a good idea to create the Markdown presentation in a folder with all the images it needs, then use Pandoc to build the presentation in the same folder. You can copy the entire folder wherever you need it, knowing that all the images for the presentation will work.

Pandoc provides syntax for scaling an image:

![An image](bench.jpg){width=25%}



Screenshot of another DZSlides slide

When Pandoc renders the image, it is scaled to a percentage of the container where it resides (the slide, in most cases). The alt text is used for a caption.

## Creating the presentation

The command for creating the document is simple. With a single Markdown file, it looks like this:

pandoc -t dzslides -s myslides.md -o myslides.htm

The -s option tells Pandoc to create a standalone presentation, including all the CSS, HTML, and JavaScript needed to display it. You can view the presentation by opening the resulting HTML file in a browser.

## Next steps

* Check out the [DZSlides template](../../resources/templates/#dzslides)

# Resources

restructuredtext-cheatsheet templates syntax-comparison glossary links

Examples of reST markup Headers

# Section Header

## Subsection Header

Lists

* A bullet list item
* Second item
  + A sub item
* Spacing between items separates list items
* Different bullet symbols create separate lists
* Third item

1. An enumerated list item
2. Second item
   1. Sub item that goes on at length and thus needs to be wrapped. Note the indentation that must match the beginning of the text, not the enumerator.
      1. List items can even include
      * paragraph breaks.
3. Third item
4. Another enumerated list item
5. Second item

Images

image

Named links

A sentence with links to [Wikipedia](https://www.wikipedia.org/) and the [Linux kernel archive](https://www.kernel.org/).

Anonymous links

Another sentence with an [anonymous link to the Python website](https://www.python.org/).

N.B.: named links and anonymous links are enclosed in grave accents (`), and not in apostrophes ('). Literal blocks

some literal text

This may also be used inline at the end of a paragraph, like so:

some more literal text

print("A literal block directive explicitly marked as python code")

# Templates

Use these templates as starting points for slide presentations.

# Markup comparison

Here's a comparison of three popular markup languages:

* AsciiDoc provides powerful, flexible syntax using blocks and attributes.
* Markdown focuses on simplicity and ease of writing.
* reStructuredText uses an extensible scheme of interpreted text roles, blocks, and directives.

All three are supported in Git repositories.

|  |  |  |  |
| --- | --- | --- | --- |
|  | AsciiDoc | Markdown | reStructuredText |
| **Strengths** | * Easy syntax * Standardized * Designed for tech writing * Converts to DocBook | * Easy syntax * Many tools available * Great for taking notes or writing posts * Can embed HTML | * Extensible * With Sphinx, easy to create docs for the web * Many ways to create tables * Standardized |
| **Weaknesses** | * Few dedicated editors * Toolchains can be complex * Idiosyncratic link syntax | * No code reuse, TOCs, or internal linking * Many incompatible flavors * Lacks admonitions and other features | * Formatting of headings is inconvenient and inconsistent * Link syntax is not intuitive * Directives and roles are clunky * Very easy to create syntax errors * Few tools available * Some features rely on Sphinx |

Here's a quick comparison of the syntax. This is just to show what they look like, not exhaustive, just to give you a feeling. For example, in all 3 languages there are more ways to do links and headings than we're showing.

|  |  |  |  |
| --- | --- | --- | --- |
|  | AsciiDoc | Markdown | reStructuredText |
| **Emphasis** | \_italic | \*italic\* \_italic | \*italic\* |
| **Strong emphasis** | \*bold\* | \*\*bold\*\* \_\_bold\_\_ | \*\*bold\*\* |
| **Inline code** | `backticks` | `backticks` | `double backticks` |
| **Bulleted list** | \* - | \* - + | \* - + |
| **Numbered list** | . | 1. | 1. #. |
| **Headings** | = Document title == H1 === H2 | # H1 ## H2 ### H3 | Underline with a sequence of characters (flexible, figured out by the parser) |
| **Semantics** | Block ID Role Options |  | Interpreted text Explicit markup Directives |
| **Link** | https://{URL}[text] link:{relative-link}[text] | [text](target) | `text <target>`\_ |
| **Image** | image::filename[alt] | ![alt](filename) | .. image::filename    :option: value    :option: value |
| **Admonition** | NOTE: text |  | .. note:: Text |
| **Comment** | // single-line comment | <!-- comment --> [//]: # "comment" | .. comments |
| **Table** | Many formats including from CSV | Grid or HTML | Many formats including from CSV |
| **Content reuse** | include::file[] |  | .. literalinclude:: filename    :option: value    :option: value |

# Glossary

admonition

A note, warning, or other call-out that draws attention to a block of content.

Bash

A Unix/Linux shell and language that lets users execute commands and programs.

branch

A series of commits representing changes to one or more files in a Git repository.

change

In Git, a modification, creation, or deletion of a file.

checkout

In Git, to switch to a different branch or restore a file.

chmod

A shell command that changes file permissions.

client

Hardware or software that accesses a *service*. A web browser is a client to a webserver, and a Git client accesses a service provided by a Git host.

clone

In Git, to make a complete local copy of a *remote repository* so you can work with the files on your computer.

cloud

Someone else’s computer.

commit

In Git, to save your changes to the *local repository*. A group of changes saved together using the commit command.

CSS

Cascading Style Sheets, a style sheet language for defining the look and feel of a document written in HTML or another markup language.

div

A division or section in an HTML document, specified with a <div> tag.

dynamic site

A site that is generated or modified at the time it is displayed. See *static site.*

fenced

Delineated with a series of characters. For example, in Markdown a *fenced code block* is marked with three backticks (```) at the top and bottom.

frontmatter

Metadata at the start of a file, often including information such as the title, author, and date.

FTP

File Transfer Protocol, a way of exchanging files between your computer and a server.

Git

A distributed source control system.

Git wiki

An additional repository, attached to a Git repository, for the purpose of displaying and managing content (often, content about the Git repository).

Git wiki structure

A content structure in which the display paths or URLs to content pages are defined by the directory paths of the files that make up the content.

host

A server, often on the web. A *Git host* provides access to Git repositories, a *web host* provides access to websites, and so on.

HTML

HyperText Markup Language, the standard markup language for creating web pages.

JavaScript

A programming language that enables the creation of interactive features on web pages.

JSON

JavaScript Object Notation, a format for storing and transporting data.

LaTeX

A document preparation system for high-quality typesetting.

Linux

A family of Unix-like operating systems first designed by Linus Torvalds in 1991.

local

On your own computer.

Markdown

A simple *markup language* originally designed as an easy way to write HTML pages.

markup language

A way of indicating display formatting and other information within a document.

merge

In Git, to combine two sets of changes into one branch.

merge conflict

In Git, a merge that cannot be completed automatically because the same parts of the files have been modified in both sets of changes.

metadata

Information about the content in a file, or about the file itself.

package manager

A tool for installing software. See [Getting started](../../getting-started/).

PDF

Portable Document Format, a file format developed by Adobe in 1993 to present documents consistently across software, hardware, and operating systems.

permissions

Settings that specify what actions can be taken and by whom. For example, file permissions can specify who can read, write, or execute the file.

pull

In Git, to fetch and merge changes from a *remote* to your *local repository*.

pull request

In Git, a set of proposed changes to be approved and then merged into a branch.

push

In Git, to upload changes from your local *repository* to a *remote*.

Python

A popular programming language.

Python Markdown extensions

A set of additional features and syntax provided with the Python implementation of Markdown.

recursion

See *recursion*.

remote

A *remote repository*.

remote repository

A version of your project that is hosted on the network or online rather than on your computer.

repo

Repository.

repository

In Git, a collection of files and the entire history of all changes made to them.

reStructuredText

XXX Definition goes here

Samba

Open source software that runs on Unix or Linux to enable communication with Windows clients over a network.

script

A computer program that automates the execution of commands or tasks.

server

A computer or application that provides a service for other programs or devices, which in turn are called *clients*.

Sharepoint

A web-based collaboratoin platform that integrates with Microsoft Office and is often used to manage and store documents.

shell

A program that lets users type commands for the operating system to execute.

source control

A way of tracking and managing changes to code or other content.

stage

In Git, to specify which changes to save in the next *commit.*

stash

In Git, to record the current state of the working directory and revert the working directory to the previous *commit.*

static site

A site composed of HTML pages or other documents that are made available exactly as stored, as opposed to a *dynamic site* whose pages are rendered on the fly when they are requested. A static site often performs better and can be more secure, but lacks some of the capabilities of a *dynamic site.*

static site generator

A tool that builds a *static site.*

TOML

Tom’s Obvious, Minimal Language, a text format for configuration files or metadata.

Unix

A family of operating systems designed at Bell Labs in the 1970s, that Linux is like.

unstage

In Git, to remove previously *staged* changes from the upcoming *commit.*

WebDAV

Web Distributed Authoring and Versioning, an HTTP extension that lets clients perform remote operations on content.

wiki

A structured HTML site, often edited and managed by the readers themselves, that collects information about a particular topic.

working branch

In Git, a temporary branch created for working on a particular set of content or code changes.

working directory

The folder on your local computer where you store the content you are editing.

WYSIWYG

What You See Is What You Get, an editing experience that mimics the appearance of the document in its final form.

YAML

Yaml Ain’t Markup Language, a text format for configuration files or metadata.

# Links

Editors

* [Atom](https://atom.io/)
* [BbEdit](https://www.barebones.com/products/bbedit/bb)
* [Emacs](https://www.gnu.org/software/emacs/)
* [Sublime Text](https://www.sublimetext.com/)
* [Vim](https://www.vim.org/)
* [Visual Studio Code](https://code.visualstudio.com/)

Git

* [Bitbucket](https://bitbucket.org)
* [Centralized Git Workflow](../recipes/recipes-centralized-workflow/)
* [Dangit, Git!?!](https://dangitgit.com/)
* [Git](https://git-scm.com)
* [Git doesn’t have to be hard](https://levelup.gitconnected.com/git-doesnt-have-to-be-hard-e1e115be6668)
* [GitHub](https://bitbucket.org/)
* [GitHub Desktop](https://desktop.github.com/)
* [GitHub Flow](https://scottchacon.com/2011/08/31/github-flow.html)
* [GitLab](https://about.gitlab.com/)
* [Git wiki structure](../../tools/tools-publishing#git-wiki-structure)
* [Git Wikis](https://docs.gitlab.com/ee/user/project/wiki/)
* [Sourcetree](https://www.sourcetreeapp.com/)
* [The Git Book](https://git-scm.com/book/en/v2)