How to do things with Markdown and Git

Peter S. Conrad

# How to do things with Markdown

Markdown is not just a markup language. It is an ecosystem of powerful tools that lets you do everything from quickly writing an HTML page, to managing a full doc set, to taking notes, to writing a novel—to creating a slide presentation. Markdown makes it easy to create content quickly, because its simple syntax covers most of the bases without requiring you to take your fingers off the keys.

With shared storage or source control, Markdown can be a powerful collaboration tool, suitable for both technical and non-technical contributors and available in editors to meet every preference.

This guide presents some basic instructions to get set up with some useful Markdown tools, and some recipes for using them. Each recipe lists what you’ll need. You might want to get a basic set of tools set up first. Read [Getting started](../getting-started/getting-started/) to learn more.

If you want to know more about what tools are available first, you can browse these sections:

* [Markdown editors](../tools/tools-editors/)
* [Publishing tools](../tools/tools-publishing/)
* [Source control with Git](../tools/tools-git/).

You might find it interesting to install more than one Markdown editor to see which features you like.

## About the recipes

Each recipe explains the goals, “ingredients,” and procedures for an interesting way to use Markdown. Just as a lasagna recipe doesn’t contain instructions for preheating (or installing) the oven, the recipes in this guide don’t contain software installation instructions. Git commands are not repeated from recipe to recipe, but documented more centrally to keep the recipes short.

Some recipes go together. You might use [Centralized Git workflow](../recipes/recipes-centralized-workflow/) to manage content you are editing in a [Git wiki](../recipes/recipes-git-wiki/) with the goal of [publishing a website with Hugo](../recipes/recipes-hugo/). Or you might take content that you started by just [taking notes](../recipes/recipes-notes/) and turn it into a [slide presentation](../recipes/recipes-slides-remark/).

# Getting started

You can start working with Markdown just by opening your favorite text editor and starting to type, but you probably want to do more than that. You might want to share information on a website or wiki, take notes, or publish an eBook.

## Ingredients

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

Markdown editor

A Markdown editor is a specialized text editor that works with Markdown. Different editors have different features—you’ll want to play with more than one to find out which ones you like.

Pandoc

Pandoc lets your Markdown editor import and export in several formats and also has a few tricks of its own. I recommend Pandoc for anyone who works with Markdown.

Shared storage

If you are collaborating with others, you might need a shared place to store Markdown 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.

Publishing tools

Documentation management tools, static site generators, and even Pandoc can be useful to convert your content to a final format for general consumption.

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](../../tools/tools-publishing/#pandoc).
2. Choose and install a [Markdown editor](../../tools/tools-editors/).
3. Set up [Git](../../tools/tools-git-setup/).

!!! hint Some editors detect pandoc, so installing it first is a good idea.

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:

**Markdown Syntax**  
Although some editors provide a WYSIWYG experience, knowing how to write [Markdown](https://www.markdownguide.org/basic-syntax/) directly will help you work more quickly and easily with a wider variety of tools.

* See the [Markdown cheatsheet](../resources/markdown-cheatsheet/)

Markdown comes in a bunch of flavors with different capabilities:

* [CommonMark](https://commonmark.org/)
* [GitHub Flavored Markdown (GFM)](https://github.github.com/gfm/)
* [Markdown Extra](https://michelf.ca/projects/php-markdown/extra/)
* [MultiMarkdown (MMD)](https://fletcherpenney.net/multimarkdown/)

Here are a few differences between these common flavors:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CommonMark | GFM | Markdown Extra | MMD |
| Fenced Code Blocks | ✔ | ✔ | ✔ | ✔ |
| Syntax Highlighting |  | ✔ | ✔ | ✔ |
| Tables |  | ✔ | ✔ | ✔ |
| Footnotes |  |  | ✔ | ✔ |
| Auto-linking |  | ✔ |  |  |
| Strikethrough |  | ✔ |  |  |
| Definition Lists |  |  | ✔ | ✔ |
| Abbreviations |  |  | ✔ | ✔ |

Some tools include extensions that provide additional capabilities. [Hugo](../../tools/tools-publishing/#hugo), for example, has an extensible framework of shortcodes that let you do things like admonitions or create your own features. Many tools, including [MkDocs](../tools/tools-publishing/#mkdocs), use the [Python-Markdown Extensions](https://python-markdown.github.io/extensions/), which offer even more capabilities out of the box. There are also more complex systems that work with Markdown, such as [Markdown+Math](https://marketplace.visualstudio.com/items?itemName=goessner.mdmath), which displays LaTeX math, and [Mermaid](https://mermaid-js.github.io/mermaid/#/), which generates diagrams.

**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 [Hugo](../../tools/tools-publishing/#hugo), [MkDocs](../../tools/tools-publishing/#mkdocs), or [Pandoc](../../tools/tools-publishing/#pandoc).

**Working with a package manager**  
Installing some tools requires 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. For example:

sudo apt-get install python3 ghostwriter pandoc mkdocs hugo gftp

**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** The point of Markdown is to produce content that can be published and consumed in another format—originally HTML, and now Word, slides, PDFs, and other formats. You should know something about tools like Acrobat, Google Drive, Microsoft Office, and other places where your Markdown might end up.

# Markdown editors

Markdown editors let you edit your content and often integrate with conversion and publishing tools. There are a few different types available:

* [Dedicated Markdown editors](#dedicated-markdown-editors) usually let you write and edit Markdown code with syntax highlighting, and often have a preview pane or separate preview window.
* [Code editors](#code-editors) can be quite extensible and sophisticated, offering plugins and packages that provide many of the capabilities of a dedicated Markdown editor.
* [Text editors](#text-editors) are often more general-purpose than dedicated Markdown editors, and may or may not offer Markdown syntax highlighing or preview.
* [Browser-based tools](#browser-based-tools) range from simple Markdown preview tools to full-featured writing and publishing tools that integrate with source control, blogging, or collaboration websites

For blogging, you can use a [browser-based tool](#browser-based-tools)—in fact, some blogging sites support Markdown already. If you plan to create and manage documentation or other complex writing projects, it makes sense to install a [dedicated Markdown editor](#dedicated-markdown-editors). If you are adding Markdown to your code development environment, several code editors support Markdown very nicely.

I have used Joplin and iA Writer, but I use Typora and Ghostwriter most often.

## Dedicated Markdown editors

Most dedicated editors are designed so that you edit the Markdown directly, with syntax highlighting to make it easier to see that’s going on. That’s the original intention behind Markdown, after all: to provide formatting tools that are easy to read and don’t require you to lift your fingers from the keys.

For general purposes, many dedicated Markdown editors are fairly similar, differing in only a few features.

Two tools in particular stand out:

* Joplin, a specialized editor for writing notes and to-do lists
* Typora, an elegant WYSIWYG editor for Markdown

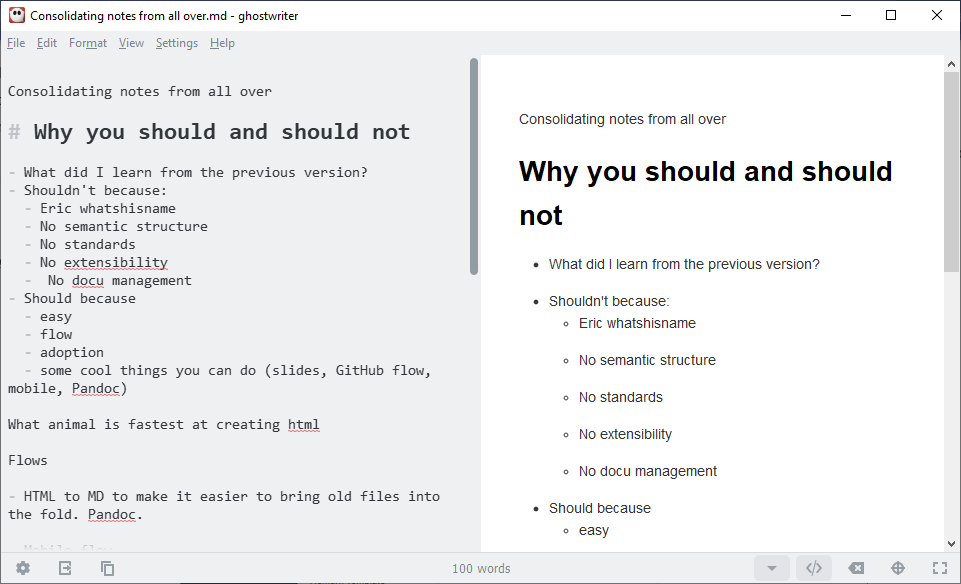
### Byword

[Byword](https://bywordapp.com/) (macOS/iOS) provides syntax highlighting and an in-app preview, letting you exprt to a few formats or publish directly to Medium, WorkPress, Blogger, Tumblr, and Evernote.

### Caret

[Caret](https://caret.io/) (macOS/Windows/Linux) offers interesting features such as a folders/files view, multiple cursors, keyboard-based selection tools, and smart help with tables and other formatting features.

### Ghostwriter

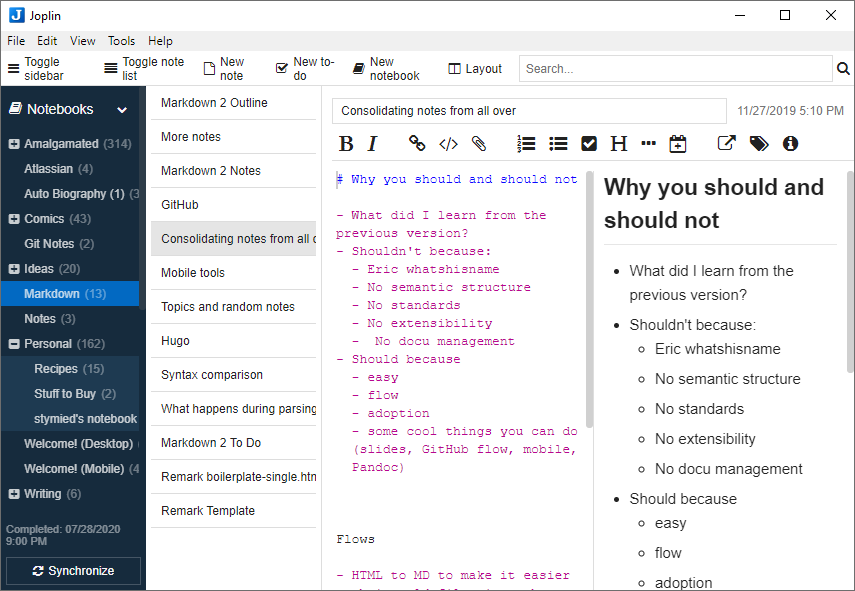


[Ghostwriter](https://wereturtle.github.io/ghostwriter/) (Windows/Linux) is a free and open source split-screen editor that integrates with Pandoc and offers smaller “heads-up display” windows that provide document information and a Markdown cheat sheet.

### iA Writer

[iA Writer](https://ia.net/writer) (macOS/iOS/Android/Windows) is intended for writers who want to focus. It integrates with Dropbox, provides the expected preview capabilities, and imports and exports to Microsoft Word.

### Joplin

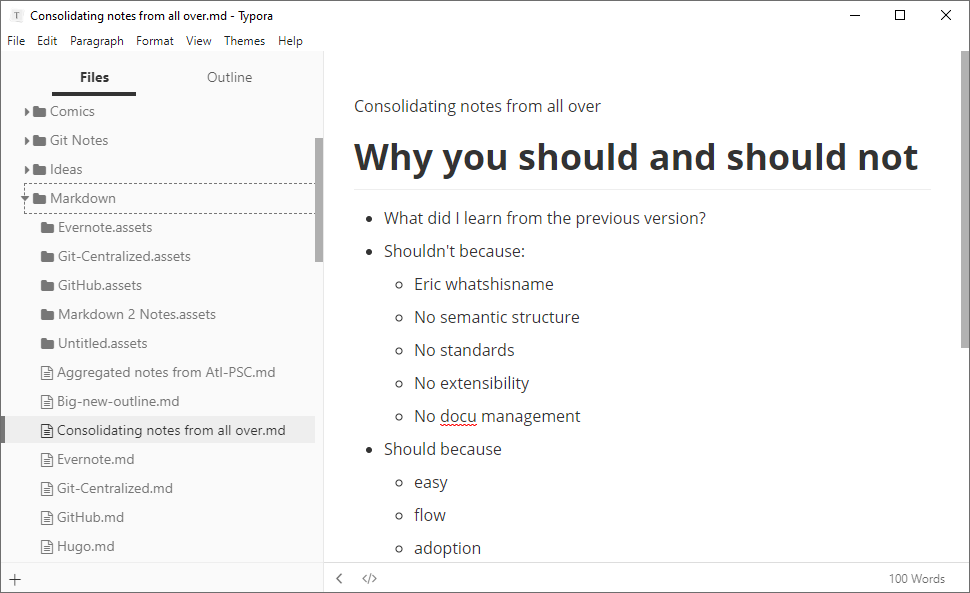


[Joplin](https://joplinapp.org/) (macOS/iOS/Android/Windows/Linux) is an open-source note-taking and to-do application that syncs to DropBox, OneDrive, and other cloud storage. It includes an editor, but also supports the external editor of your choice. It offers end-to-end encryption and can import Evernote notebooks.

### MacDown

[MacDown](https://macdown.uranusjr.com/) (macOS) is an open source Markdown editor with a split screen and live preview, inspired by an early Markdown editor called [Mou](http://25.io/mou/).

### Typora



[Typora](https://typora.io/) (macOS, Windows, Linux) is a Markdown editor that includes not only a syntax editing mode but a fully WYSIWYG editing experience. It integrates with Pandoc, so it can import and export a number of formats. Typora offers a files and folders view, which means it can be used for longer documents or as a note-taking application.

## Code editors

If you are documenting code in a large organization, you might need to write Markdown that meets fairly stringent standards to work with your documentation management and publishing system. Code editors, while they are not designed specifically for Markdown, sometimes offer sophisticated packages to support Markdown 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. 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/)

## Text editors

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

* [BbEdit](https://www.barebones.com/products/bbedit/bb)
* [Emacs](https://www.gnu.org/software/emacs/)
* [Vim](https://www.vim.org/)

## Browser-based tools

Browser-based Markdown tools can be an easy, portable way to write Markdown that you intend to publish to a blog. Some integrate directly with Git hosts, blogging sites, or cloud storage.

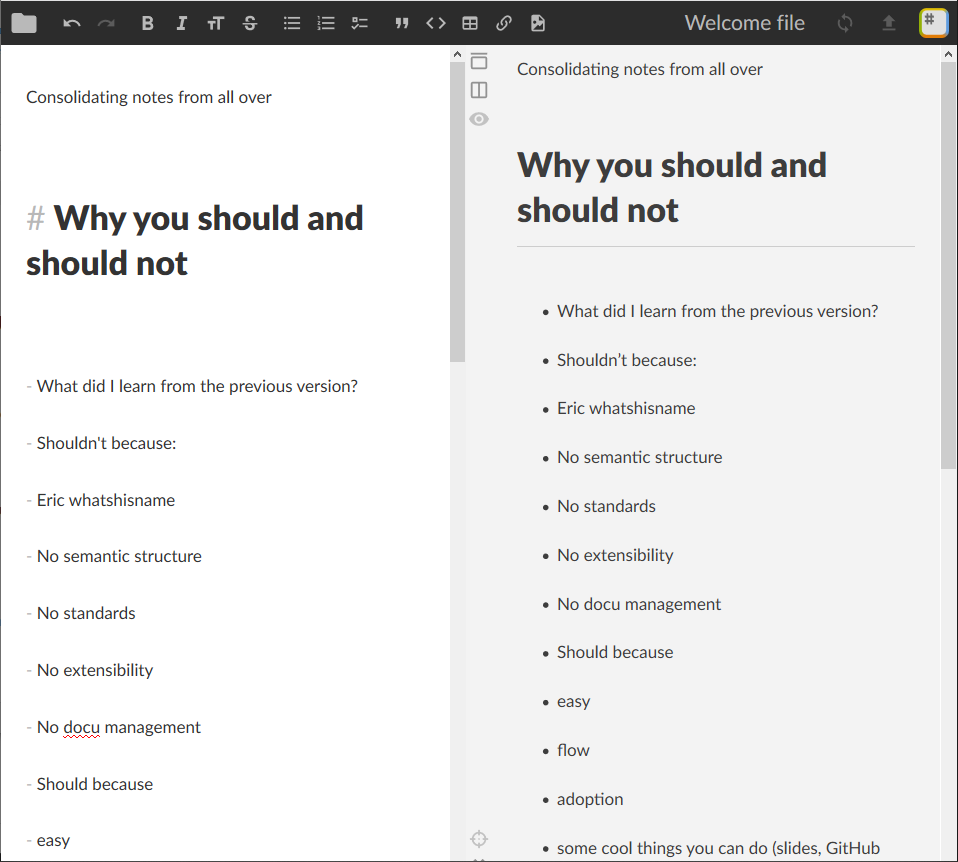
### Dillinger

[Dillinger](https://dillinger.io/) is an open source online Markdown editor that integrates with DropBox, GitHub, Google Drive, and OneDrive. You can also import from BitBucket, or upload Markdown and HTML files from your computer.

### HackMD

[HackMD](https://hackmd.io/) is an online Markdown editor, free for personal use, that integrates with GitHub and includes tools for creating [slides](https://hackmd.io/c/tutorials/%2Fs%2Fhow-to-create-slide-deck).

### StackEdit



[StackEdit](https://stackedit.io/) is a full-featured Markdown editor that syncs with DropBox, GitHub, GitLab, and Google Drive. You can import Markdown and HTML files, export with Pandoc, and publish to Blogger, Wordpress, and Zendesk.

### Markdown Dingus

[Markdown Dingus](https://daringfireball.net/projects/markdown/dingus) shows how John Gruber’s official version of Markdown renders to HTML.

### Babelmark

[Babelmark](https://babelmark.github.io/) is a useful tool for comparing the way different implementations and flavors render Markdown. If you are trying to figure out whether you have a problem in your Markdown or are merely using a feature that’s not supported in the Markdown flavor you’re using, Babelmark can help.

# Storage

The simplest place to store Markdown 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 Markdown, you’ll want to share the files you’re working on or publish them in another form.

!!! hint Some Markdown tools, especially browser-based editors, sync directly with cloud storage and blogging sites.

### 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 Markdown.

### 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 Markdown files from one computer and have them available on other devices, even sharing them with colleagues. Cloud storage is great for [Taking notes](../../recipes/recipes-notes), because it lets you add to your notes from anywhere.

## 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.

!!! note If you’re just using Markdown to take notes or create Word documents, PDFs, and slides, you might not need Git. But if you’re collaborating with others, read on!

With Git, you synchronize files on your computer with files on a remote repository—usually an online Git host. You can work on the files, saving them to the directory you’ve chosen for your local git repository, pushing changes up to the remote repository and pulling down changes others have made.

## 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 wass one central source of truth; if that were 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 one of the best known Git hosts:

* [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. These two are probably the best known:

* [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 install to use the client and host that go together:

* Bitbucket and Sourcetree
  + GitHub and GitHub Desktop
* This guide includes instructions for those two combinations.

## Git concepts

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).

There are a lot of Git terms that are confusing. 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, we don’t care about *files* at all in Git. We only care about *changes.*

### What things mean in Git

Let’s say you have a directory of files you’re working on. yes, Git doesn’t care about files, but you do!

**Commit**  
You open a file in an editor, make some changes, and save the file. Git will notice the *changes* you made to the file. If you meant to make those changes, you must *commit* them. This is like “saving changes to Git.” A commit is a group of changes.

**Stage**  
Sometimes, before you commit changes, you tell Git explicitly 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.

**Repository**  
When you commit, you type a little note describing the changes so that people know what was done. When you have committed, your changes are stored in the *repository,* or *repo* for short. It’s just like that old central repository, but it’s just 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.

**Pull**  
As you work on a repository you’ve cloned, you can *pull* the latest changes 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 it’s the only 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 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 master 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 master, it’s not there:

$ git checkout master   
Switched to branch 'master'  
Your branch is up to date with 'origin/master'.  
$ 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.

## Next steps

* [Set up Git](../tools-git-setup/)

# 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

If you don’t already have Git set up on your computer, 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/).

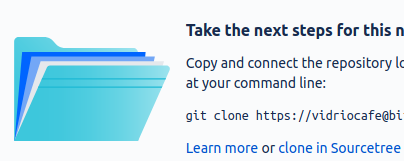
!!! hint You can use any Git client with any Git host, but some clients work especially well with specific hosts. For example:

* Sourcetree and Bitbucket
  + GitHub Desktop and GitHub

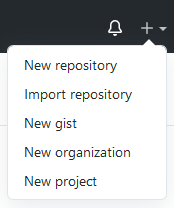
## Set up a repository

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.
   1. Click the new repository button (the plus sign):  
      Add a repository
   2. Type a repository name, make sure it’s set to be a public repository, and click **Create repository**.
   3. Choose or create a directory on your computer where you would like to keep your local copy of the project.
   4. Click **Clone in Sourcetree**.  
      
   5. Choose a folder on your computer for the local copy of the repository and click **Clone**.

=== “GitHub and GitHub Desktop”

1. Sign up with [GitHub](https://bitbucket.org/) and log on.
   1. Click the plus sign and select **New repository**:  
      
   2. Type a repository name, make sure it’s set to be a public repository, and click **Create repository**.
   3. Click **Set up in Desktop** to open the repository in GitHub Desktop:  
      
   4. 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

## Next steps

* Take a look at [Git basics](../getting-started-git-basics/)
* Try setting up a [Git wiki](../../recipes/recipes-git-wiki) # Git basics

This page has the workflows and commands you’ll use in the recipes, and some hints to get you out of trouble.

!!! hint It might not be worth it to read through this page all the way through. Just refer to the parts you need. The recipes link to them.

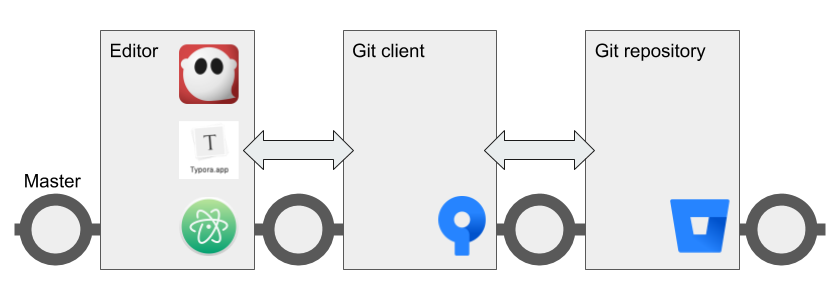
## Branching workflows

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

* [Git Centralized Workflow](../../recipes/recipes-centralized-workflow/): all work is done in a single branch
* [GitHub Flow](../../recipes/recipes-gitflow/): different pieces of work are done in different branches

#### Centralized Git workflow

In the centralized Git workflow, everyone works on a single branch (usually called master). 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 Git workflow.

–8<– “centralized-workflow-snippet.html”

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

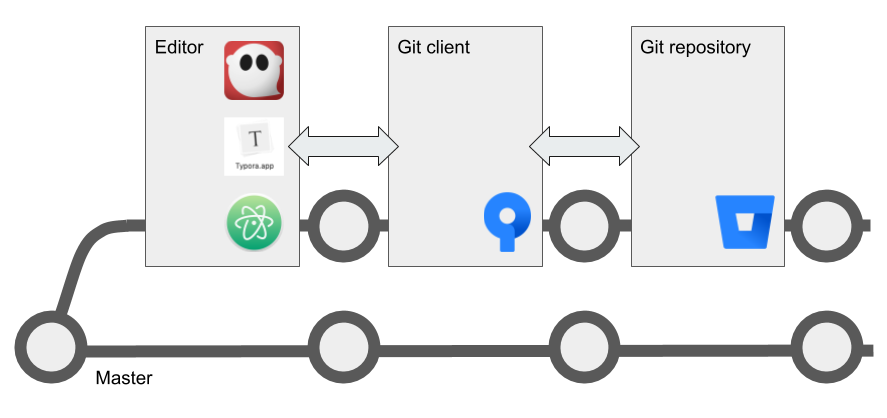
Although this workflow is called “centralized,” it doesn’t really resemble the old centralized source control model. The central remote repository is not a single source of truth, because every contributor has their own copy of the entire repository.

The following recipes use the centralized Git 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 master 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 master.

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

–8<– “github-flow-snippet.html”

Merge conflicts in the same branch are less likely, because you are creating your own branches and other people don’t necessarily work in your branch with you. Before you push your branch to the remote repo, you can [pull](#pull) from master 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 master 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 branching 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
   1. 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
   1. Select **Repository > Pull** or:
      1. Click the **Fetch origin** button.  
         
      2. Click the **Pull origin** button.  
         

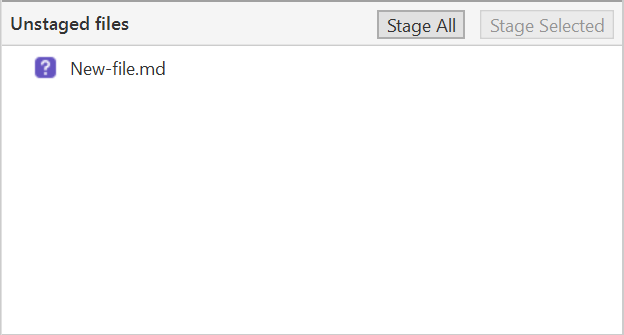
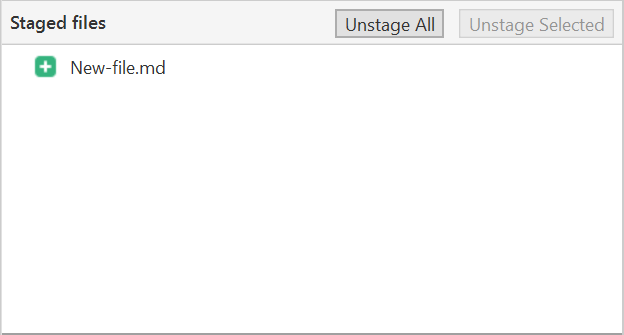
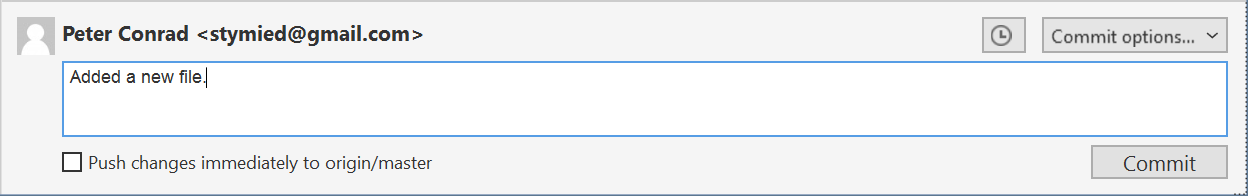
=== “Linux command line” 1. Make sure you’re on the right branch in the correct repository: $ git branch \* master $ git remote -v origin https://github.com/pconrad-fb/markdown.git (fetch) origin https://github.com/pconrad-fb/markdown.git (push) 1. Type the git pull command: $ git pull

### Stage and commit

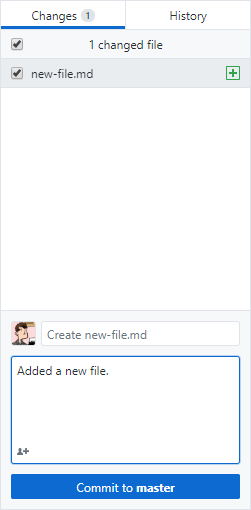
Git knows when you make changes to your files. When you want to save those changes to Git, you must do two things:

* *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.
   1. Look for the files you changed in the Unstaged files pane. Select the files you want to stage—in most cases, you can just click **Stage All.**  
      
   2. Make sure you see the right files in the Staged files pane.  
      
   3. 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.
   1. Look for the files you changed in the Changes tab. Unselect any files you don’t want to change—most of the time, you can leave all the checkboxes checked.  
      
   2. Type a short commit message.
   3. Make sure the **Commit** button refers to the correct branch (“Commit to master,” for example).
   4. Click **Commit to [branch]**.

=== “Linux command line”

1. Make sure you’re on the right branch in the correct repository.
   1. Use git status to see what changes are not yet staged.
   2. 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.
   1. Select **Repository > Push** or click the **Push** button.  
      

=== “GitHub Desktop”

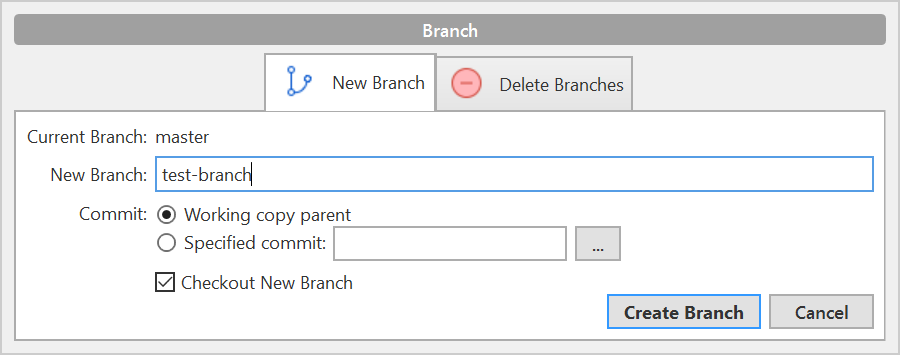
1. Make sure you’re on the right branch in the correct repository.
   1. Select **Repository > Push** or click the **Push origin** button.  
      

=== “Linux command line”

1. Make sure you’re on the right branch in the correct repository.
   1. Push, specifying the remote (usually origin) and the branch. For the recipes where you are working on the master branch, the command looks like this:
   * $ git push origin master
   * Of course, since git knows what branch you’re on and where your remote is, you can sometimes just type git push.

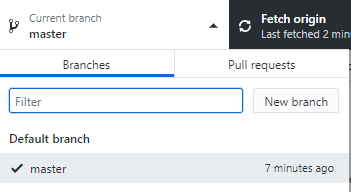
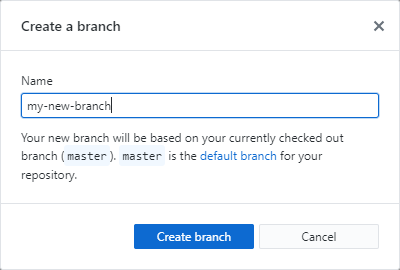
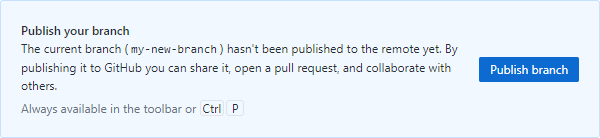
### Create a branch

=== “Sourcetree”

1. [**Pull**](../../getting-started-getting-started-git-basics/#pull) from master.
   1. Click the **Branch** button:  
      
   2. Type a descriptive name and click **Create Branch**.  
      
   3. 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.

=== “GitHub Desktop”

1. [**Pull**](../../getting-started-getting-started-git-basics/#pull) from master.
   1. Click the **Current branch** tab and click **New branch**:  
      
   2. Type a descriptive name and click **Create branch**:  
      
   3. Click **Publish branch**:  
      
   4. 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.

=== “Linux command line”

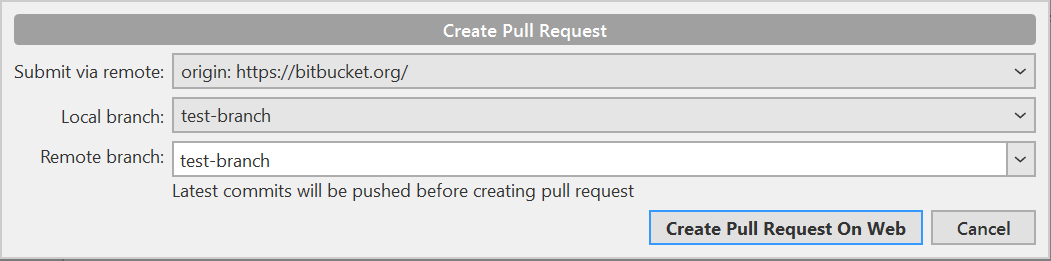
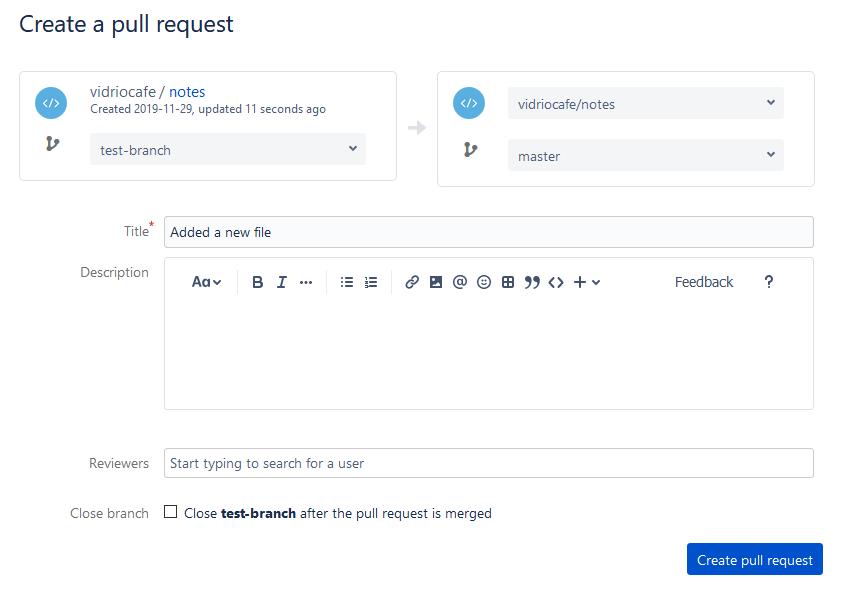
1. Pull from master to make sure you have the latest changes: ``` $ git checkout master Already on ‘master’ Your branch is up to date with ‘origin/master’. $ git pull Already up to date.

* ```
  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 -b test-branch Switched to branch 'test-branch'

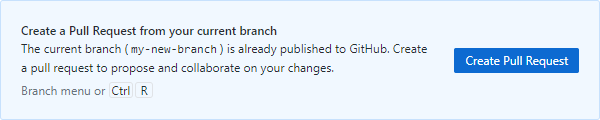
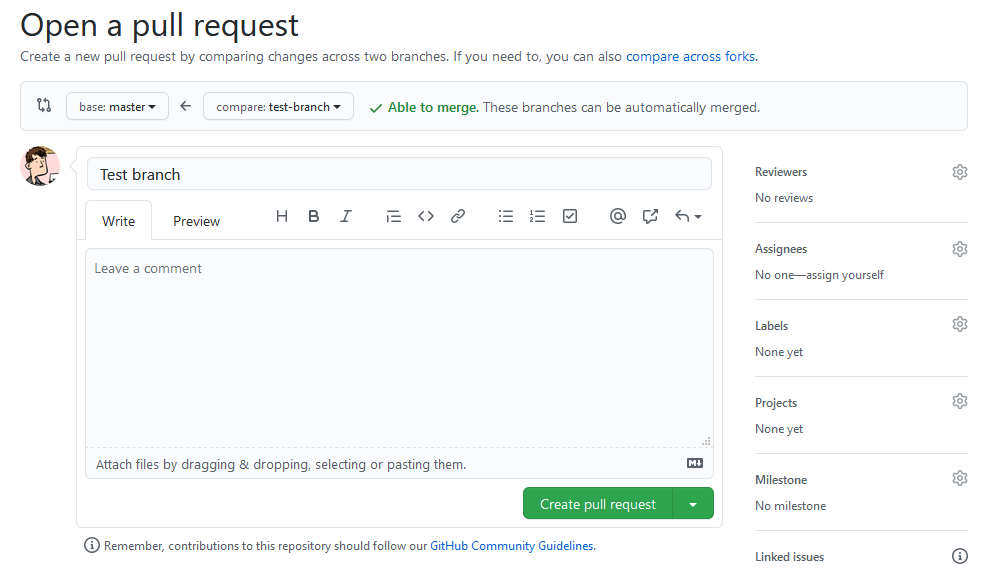
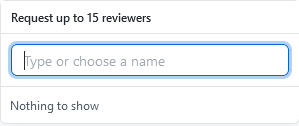
!!! hint You can’t switch branches with uncommitted changes. You have to commit before switching to a new branch.

### Create a pull request

=== “Bitbucket and Sourcetree”

1. Click **Repository > Create pull request**.
   1. In the dialog that appears, click **Create Pull Request On Web**:  
      
   2. 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 “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: 
   2. Click the gear next to **Reviewers** to add reviewers:  
      
   3. 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 screen. If you get stuck, you can see some hints in the instructions for Bitbucket or GitHub.

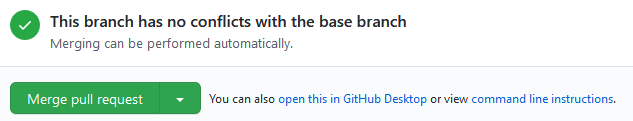
### Approve and merge

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

=== “Bitbucket”

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

=== “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

So 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. If necessary, switch to the correct branch. For example:

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

1. In the branch where you were erroneously working, stash your uncommitted changes:

* $ git stash

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

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 -b the-branch   
   Switched to branch 'the-branch'

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

* $ git status  
  On branch master  
  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 to undo the changes. For example:

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

### 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 you want.

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 -b the-branch   
   Switched to branch 'the-branch'

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

* $ git status  
  On branch master  
  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

### 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 last commit in the branch you’re on. The other label can be another branch name or a number representing another commit.

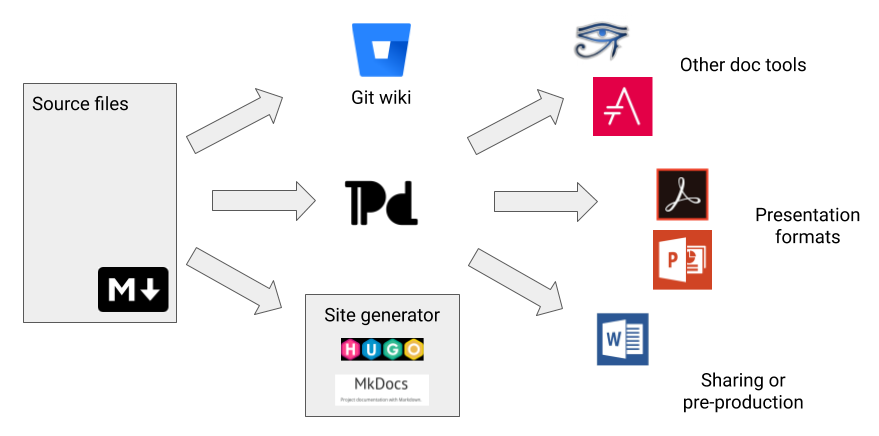
All you need to do is decide which version of the content you want to keep and then delete the merge conflict markers (<<<<<<<, =======, >>>>>>>).

After you’ve resolved all the changes in that way, just [commit](#stage-and-commit) again.

### More trouble

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

There are many tools that can take your Markdown to your audience or just convert them for use in other doc tools. We’ll focus the Git wiki, the static site generators Hugo and MkDocs, and Pandoc. We’ll also touch on tools for creating standalone presentations.



## Git wiki

A Git wiki is a special additional repository attached to your Git code repository. You create and clone a git wiki separately from your main repository.

### Git wiki structure

A Git wiki uses folders to organize files. The path to a file is used to create the URL where the content is displayed.

A simple directory structure might look like this:

Home.md  
stuff/  
 something.md

In that case, the URL to the content in something.md is: **/wiki/stuff/something**

See also:

* [Edit a Git wiki](../../recipes/recipes-git-wiki/)

## Hugo

[Hugo](https://gohugo.io/) is an open-source static site generator that includes templates and provides exensibility in the form of *shortcodes,* which provide additional formatting. You can even create your own shortcodes.

See also:

* [Publish a website with Hugo](../../recipes/recipes-hugo/)

## MkDocs

[MkDocs](https://www.mkdocs.org/) is a static site generator that turns directories of Markdown files into a searchable, presentable website. Customizable themes, many of them provided by the community, provide Markdown extensions for reusing content and for formatting such as tabs and admonitions.

See also:

* [Publish documentation with MkDocs](../../recipes/recipes-mkdocs/)

## Pandoc

[Pandoc](https://pandoc.org/) 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 standalone slide presentations that work with tools like DZslides. For longer documents, Pandoc has options for concatenating Markdown files together.

Pandoc includes formatting tricks that let you do things like create columns or scale images, and 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 dong things like grabbing a web page as a Markdown file. You can even [try it online](https://pandoc.org/try/).

See also:

* [Create a Word document](../../recipes/recipes-pandoc-word/)
* [Create an eBook](../../recipes/recipes-pandoc-ebook/)
* [Create a PDF](../../recipes/recipes-pandoc-pdf/)
* [Grab a web page](../../recipes/recipes-pandoc-web/)
* [Create a Powerpoint presentation](../../recipes/recipes-slides/)
* [Create a DZSlides presentation](../../recipes/recipes-slides-dzslides/)

## FTP

When you create a website with Hugo or MkDocs, you’ll use FTP to upload the files to the web. You can do this from the command line, but it’s much easier to choose an application that gives a nice UI and lets you save bookmarks. Here are a few:

* [FileZilla](https://filezilla-project.org/) - a free, open source FTP tool for macOS and Windows
* [gFTP](https://en.wikipedia.org/wiki/GFTP) - the default Linux option
* [Transmit](https://panic.com/transmit/) - a commercial app for macOS

## Presentation tools

Here are two tools that you can use for in-browser presentations.

### DZSlides

[DZSlides](http://paulrouget.com/dzslides/) is a single page, HTML-based slide presentation template. Pandoc can generate a DZSlides presentation from a Markdown file.

See also:

* [Create a DZSlides presentation](../../recipes/recipes-slides-dzslides/)

### Remark

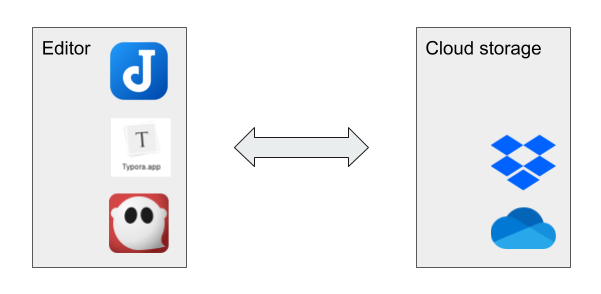
[Remark](https://remarkjs.com/#1) is an in-browser presentation tool made from Javascript, HTML, CSS, and, of course, Markdown. The slideshow lives in a <textarea> tag where you can add and edit Markdown directly.

See also:

* [Create a Remark presentation](../../recipes/recipes-slides-remark/)

# Take notes

Markdown is a great tool for taking notes quickly, because you can accomplish all the basic formatting without taking your hands off the keyboard. You can use any Markdown editor to work with files on your computer, but you can also share notes across your devices using a cloud storage service such as Dropbox plus a note-taking app like Joplin or a Markdown editor like Typora.



## Ingredients

Markdown editor

Cloud storage

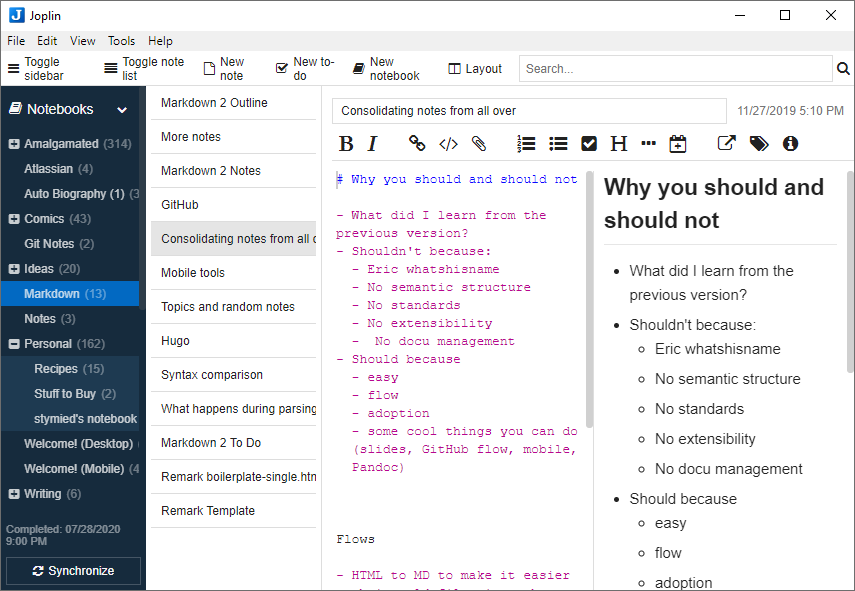
Either Joplin or another editor Either local disk or synced folder/backend

Talk about how to set up Joplin with backend, how it works with external editing, how to do encryption.

Alternative: just point Typora at a Dropbox or other synced folder. You can collaborate with people this way!

## Joplin

[Joplin](https://joplinapp.org/) is designed specifically for taking notes, and shares some features with applications like Evernote. You can store your notes as local files on your computer, or sync them to the cloud with a service like Dropbox or OneDrive. you can even encrypt your notes end-to-end for privacy.

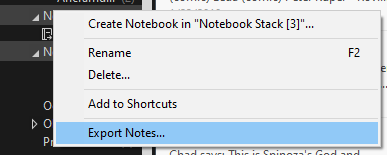
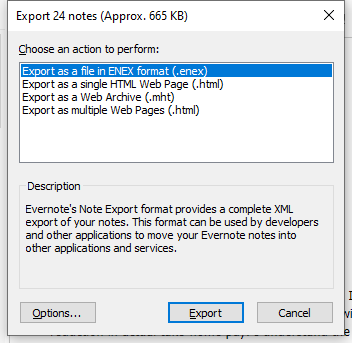
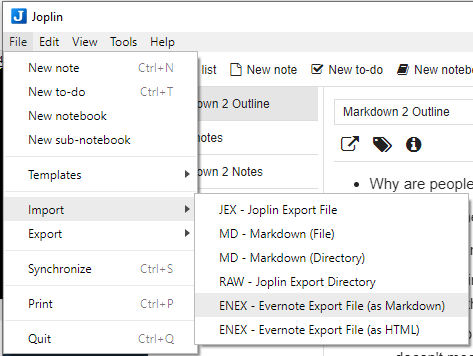
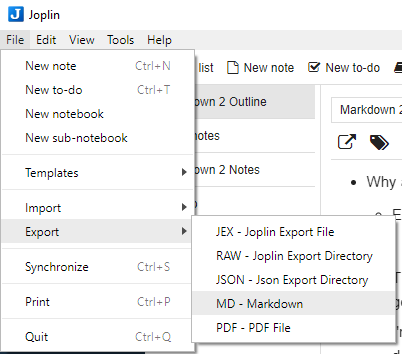


Setting up Joplin is fairly straightforward. In the **Preferences** screen, there is a place to choose your storage: local filesystem, DropBox, and so on. If you want to sync Joplin across devices you should configure all your Joplin instances to use the same cloud storage location.

### Getting your old notes from Evernote

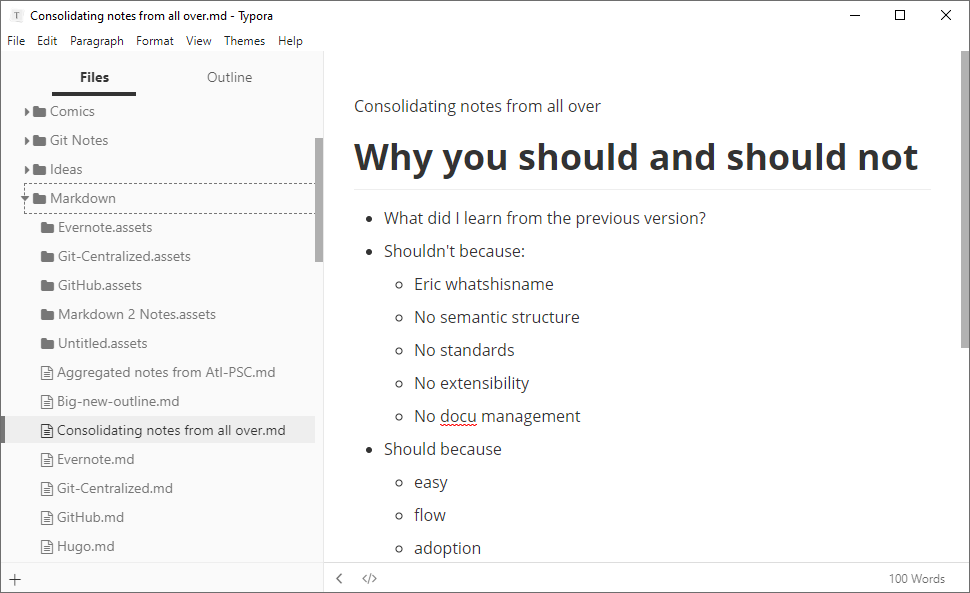
One of the great features of Joplin is that you can import notes from Evernote. The caveat is that the imported Markdown files will not have sensible filenames—but this is still better than manually copying and pasting each note individually.

To can export notebook by notebook from Evernote:

1. Right click a notebook in the left nav and select Export notes  
   
2. Export the notebook as an ENEX file.  
   
3. In Joplin, go to File > Import and import the notebook as Markdown.  
   
4. You can then export the notebook as Markdown from Joplin into the folder where you have decided to store your notes.  
   

## Typora

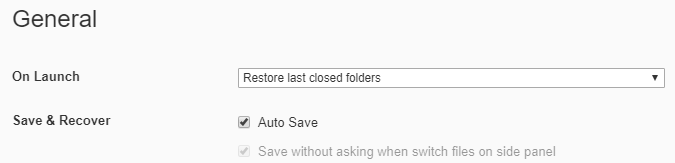
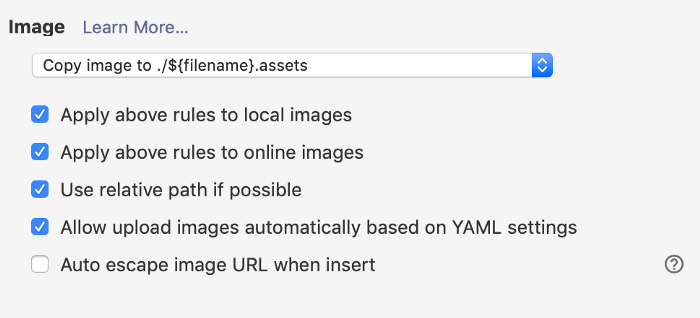
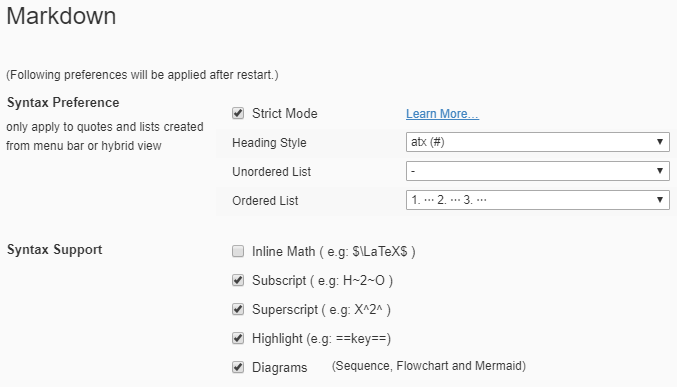
An interesting alternative to Joplin is to use another editor such as Typora. Both Joplin and Typora provide a navigation tree to make it easy to organize and manage “notes” (which are just Markdown files). The advantage of Joplin is that it is a dedicated note-taking app. The advantage of Typora is that it is a Markdown editor, which means you get a lot more tools for integrating into a content management and publishing toolchain. If you plan to do more that note-taking, consider installing a Markdown editor such as Typora or Ghostwriter. Actually, there’s nothing stopping you from using all three!



Installing and using Typora is pretty straightforward. You can store files in multiple folders, which makes it easy to set up a note-taking folder, a drafts folder, a documents folder, and so on. Any of these folders could by synced by Dropbox or another service.

### Setting up Typora to take notes

To fine-tune Typora as a note-taking application, you will want to set a few preferences.

1. Go to **File > Preferences**.
2. Click the **General** tab:
3. For On Launch, select **Restore last closed folders**. When you set up a folder to contain your notes, Typora will open it for you automatically. Of course, you can also open other files and folders.
4. Check the **Auto Save** checkbox and make sure **Save without asking…** is checked. You won’t have to think about saving notes; they’ll just always be saved. 
5. Click the **Image** tab.
6. Select **Copy image to ./${filename}.assets**. Typora will make a folder for each  
   document’s images. This makes it easy to take the images along if you convert the document to a different format or send it to another location.
7. Check all the boxes except for the auto escape one. This makes sure that local files and online images also get added to the .assets folder. By the way, Typora saves copies of pasted images, too! 
8. Click the **Markdown** tab.
9. Check the **Highlight** box so that you can highlight things in your notes. 

There are other interesting syntax features to explore, including diagrams.

# Edit a Git wiki

A Git repository comes with a wiki, which makes it possible to collaboratively read and 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 t the 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 [MkDocs](../recipes-mkdocs/) or [Hugo](../recipes-hugo/)

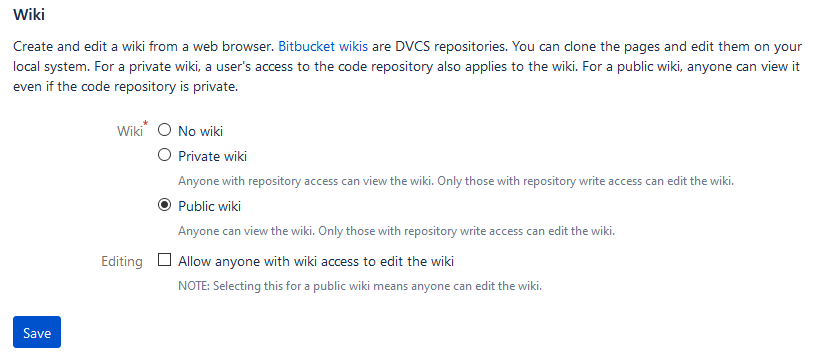
## Ingredients

Markdown editor

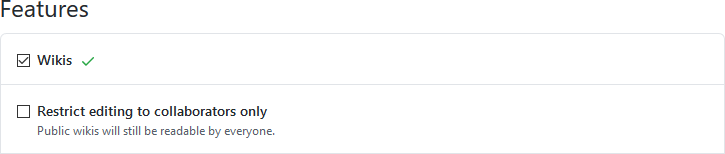
Git

## Set up a wiki on your hosted repository

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

=== “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:  
      
   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.

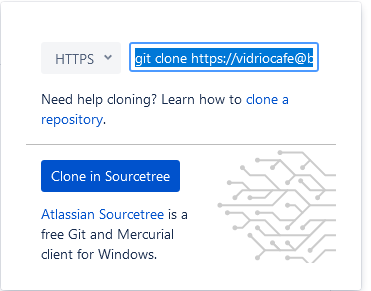
!!! hint To add a page in a new folder, make the folder part of the new filename. For example: morestuff/newpage.md adds newpage.md in a folder called morestuff.

## 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:  
 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—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:

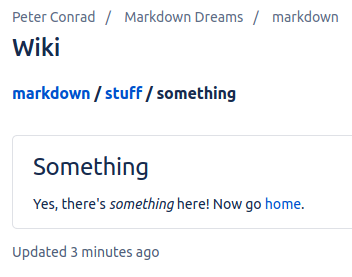
* Add a folder called stuff.
* Using your favorite Markdown editor, make a file called something.md inside stuff, with the following contents:
* # Something  
    
  Yes, there's \*something\* here! Now go [home](../Home).

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:

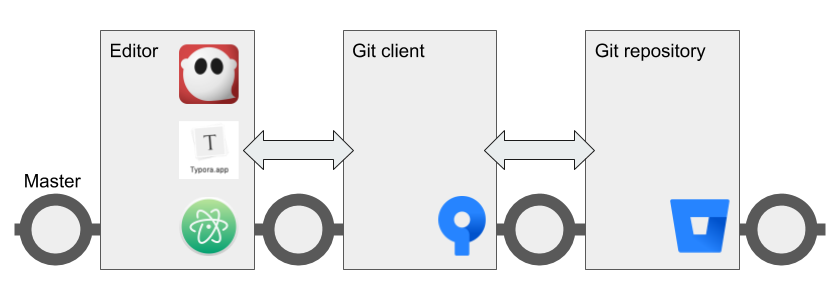
* [Collaborate using the Git centralized workflow](../recipes-centralized-workflow/) # Collaborate using centralized Git workflow

You can use Git to help a team collaborate on informal or internal documents in a very simple way. If you use a graphical Git client and a WYSIWYG Markdown editor, the workflow is easy enough for people who don’t want complicated technical procedures to get in the way of their work. At the same time, Markdown in Git is easy for engineers to work with in their own tools.

This is great for an informal thing of collaborating on a website, doc set, or make an eBook, PDF, Word doc (made up of several files) - can even do Pandoc slides if you break it up and then use Pandoc to concatenate

Although a git repository comes with a separate wiki repository, you might sometimes want to collaborate on content without the constraints of storing it in a wiki. In other words, [Running a Git wiki](../recipes-git-wiki) is a easy way to create, edit, and view a lot of content—but using other tools to work with content on your own computer, using a Git repository only for source control, is a much more robust way to collaborate.

For non-production content such as specifications, planning documents, newsletters, internal documentation, and the like, you can use a simple single-branch strategy called the centralized Git workflow.



If you’re publishing a complex website or documentation set with a team of collaborators, you should consider using [Manage docs with Gitflow](../recipes-gitflow/) instead.

## Ingredients

Markdown editor

Git

Publishing tools (optional)

!!! 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) from the start. If you’re creating 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/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<– “centralized-workflow-snippet.html”

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 Markdown and other source files.

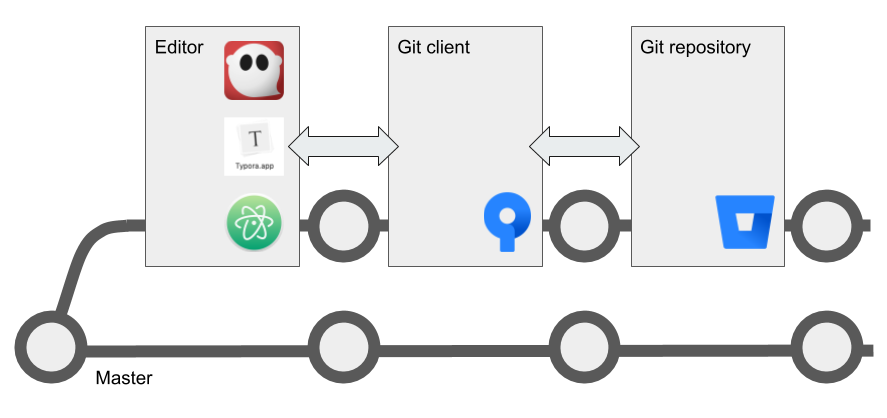
## Next steps

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/)

# 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. Here’s how it changes the single-branch **Pull-Work-Commit-Push** steps used in the [Centralized Git Workflow](../recipes/recipes-centralized-workflow/).



## What you need

Markdown 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) from the start. If you’re creating 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 master 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 Markdown 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

[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/)

You can also download and install the binary, or build it yourself from source.

## Ingredients

Markdown editor

Hugo

Git (optional)

!!! hint Hugo works best with Markdown 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.

This recipe goes well with:

* Working by yourself on your local hard drive
* [Centralized Git workflow](../recipes-centralized-workflow/)
* [GitHub flow](../recipes-gitflow/)

Make sure you have the ingredients for those as well.

## 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/markdown/markdown-hugo.  
  
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.

## 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. Here is how it looks:

$ 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 like so:

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

You can 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, typing hugo new posts/my-first-post.md creates a file in the content directory that will display at the URL /posts/my-first-post. 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  
---

Once you add some Markdown content, you can preview the page. A page in Hugo must contain both frontmatter and Markdown content. If either is missing, the page will show 404 page not found when you try to preview it.

You can use YAML, TOML, or JSON frontmatter in your files. Frontmatter can signal content status inclusing publish and expiry dates, and can contain variables that you can use in templates and in content by creating your own Hugo shortcodes. This is too complex to show here, but the Hugo website has a lot of documentation.

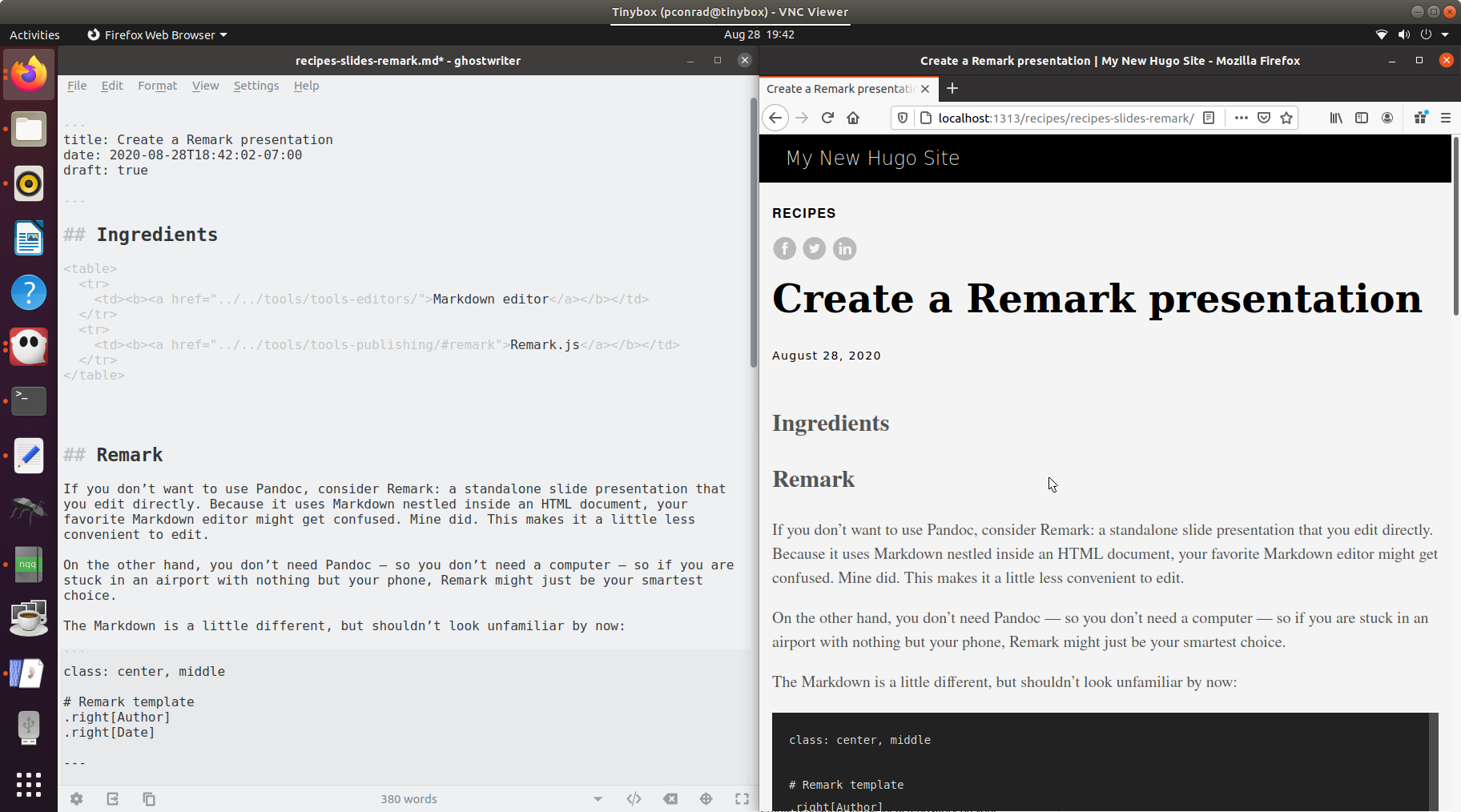
If you are migrating content into Hugo from a Git wiki, MkDocs site, or some other source, you will need to add frontmatter to any pages that don’t already have it.

## Local preview

To preview your content, start the Hugo server:

$ hugo server -D

The -D option includes pages in draft status. If you omit this option, any page with draft:true in the frontmatter is ignored.



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. Make sure each file contains both frontmatter and Markdown, that it doesn’t have a publish date in the future or expiry date in the past, and so on.

### 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 Markdown editor it looks like this:

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 Markdown 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.

You can try another solution: instead of putting images in static, create an images directory inside the content directory. If you do that, you end up with a different problem.

Relative to the Markdown file, the image is now at: ../img/whatever.png

However, relative to the URL, the image needs to be at: ../../img/whatever.png

In other words, if you put an images directory inside content then you must add one more layer of ../ because the page itself is treated as a directory in the browser.

To sum up: images can be a little tricky in Hugo.

## 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&mdash;meaning that no one will   
 add built HTML pages to Git.  
# Publish documentation with MkDocs  
  
[MkDocs](https://www.mkdocs.org/) is a static site generator designed for documentation. It's fairly easy to use, though it does require some work on the command line. Like some other tools, it uses the [Git wiki structure](../../tools/tools-publishing#git-wiki-structure)&mdash;which means you can even use [Git wiki](../recipes-git-wiki/) to develop the content.  
  
To install MkDocs, 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/)  
  
Because MkDocs is based on Python, you can also install manually install it using the `pip` tool.  
  
## Ingredients  
  
<table>  
 <tr>  
 <td><b><a href="../../tools/tools-editors/">Markdown editor</a></b></td>  
 </tr>  
 <tr>  
 <td><b><a href="../../tools/tools-mkdocs/">MkDocs</a></b></td>  
 </tr>  
 <tr>  
 <td><b><a href="../../tools/tools-git-setup/">Git</a></b> (optional)</td>  
 </tr>  
</table>  
  
This recipe goes well with:  
  
- Working by yourself on your local hard drive  
- [Centralized Git workflow](../recipes-centralized-workflow/)  
- [GitHub flow](../recipes-gitflow/)  
  
## Creating a project  
  
You can create a new doc set just by typing `mkdocs new my-project` and starting to add content. Of course, if you plan to use one of the Git workflows, you should run this command inside your local Git repository. The `mkdocs new` command sets up a directory that contains two things:  
  
- A directory called `docs` containing `index.md`  
- A file called `mkdocs.yml` that you use for configuring your project.  
  
As you might have guessed, `index.md` is a congratulatory default first page, and you can change it, add directories and files, and start building the site in the `docs` directory.  
  
## Live preview  
  
The command `mkdocs serve` starts a webserver that lets you preview your content as you create it. Whenever you save a Markdown file, MkDocs does its best to update the preview&mdash;sometimes if you change the site navigation, it can't keep up. When that happens, just use Control-C to stop the server and then type the command again to start it.  
  
![](../img/live-preview.png)  
  
When the webserver starts, it provides information about any broken links in your content, any files that are unused, and where to point your browser to see the content. Here's an abbreviated version of some output I got while working on this website:

$ mkdocs serve INFO - Building documentation… INFO - Cleaning site directory INFO - The following pages exist in the docs directory, but are not included in the “nav” configuration: - getting-started/index.md - recipes/index.md WARNING - Documentation file ‘recipes/recipes-centralized-workflow.md’ contains a link to ‘recipes-slides.md’ which is not found in the documentation files. INFO - Documentation built in 1.82 seconds [I 200728 20:45:18 server:296] Serving on http://127.0.0.1:8000 INFO - Serving on http://127.0.0.1:8000 [I 200728 20:45:18 handlers:62] Start watching changes INFO - Start watching changes [I 200728 20:45:18 handlers:135] Browser Connected: http://127.0.0.1:8000/recipes/recipes-centralized-workflow/ INFO - Browser Connected: http://127.0.0.1:8000/recipes/recipes-centralized-workflow/

## Adding a theme  
  
The default look is fine, but you'll probably want to choose a [theme](https://github.com/mkdocs/mkdocs/wiki/MkDocs-Themes). Themes don't just change the look of the site&mdash;they sometimes add extensions and capabilities (such as the [Python Markdown Extensions](https://python-markdown.github.io/extensions/)).  
  
You can add a theme, activate extensions, and set up site navigation in the site configuration file `mkdocs.yml`.  
  
!!! hint  
 You don't have to set up navigation manually. If you don't, MkDocs provides  
 fairly sensible navigation automatically. But the option is there if you   
 need that level of control.  
  
Here's an abbreviated version of my `mkdocs.yml` file:

site\_name: Markdown Dreams

theme: name: material

markdown\_extensions: - admonition - pymdownx.highlight: use\_pygments: true - pymdownx.snippets base\_path: ‘snippets/’

nav: - ‘How to do things with Markdown’: ‘index.md’ - ‘Getting started’: ‘getting-started/index.md’ - ‘Tools’: - ‘Markdown editors’: ‘tools/tools-editors.md’ - ‘Source control’: ‘tools/tools-git.md’ - ‘Publishing’: ‘tools/tools-publishing.md’ - ‘Recipes’: - ‘Overview’: ‘recipes/index.md’ - ‘Run a Git wiki’: ‘recipes/recipes-git-wiki.md’ - ‘Resources’: ‘resources/index.md’

To save length, I stripped out a lot of the navigation, but you get the idea.  
  
## Working with content  
  
For the most part, working with content is what you would expect. It's just Markdown in Git wiki structure, using the workflow of your choice. Here are a few tips:  
  
- Links take some getting used to. Each link is relative based on the location of the  
 page in which the link appears. Remember, each file is treated as a folder in the  
 browser. To link to the "Source control" page from "Run a Git wiki" in the above  
 navigational structure, you would add the following link:   
 `[Source control](../../tools/tools-git.md)`   
 If you have trouble with a link, look at the output when you run `mkdocs serve`.  
- Because all HTML is valid Markdown, you can use HTML. However, you can't use  
 Markdown inside a block of HTML.  
- If you choose a theme that includes features like snippets and admonitions, then  
 you can do things outside the bounds of normal Markdown. This is very useful, but  
 makes it likely that you can't use your Markdown source files with other toolchains  
 (unless you take the fun stuff out).  
  
### Snippets  
  
If you use snippets, the location of the files you include is relative to the top-level directory of your project (`my-project` for example). It's a good idea to create a directory for snippets and then define it in the `base\_path` variable in your `mkdocs.yml` file.   
  
I created a directory called `snippets` and specified it as shown above. That way, I can include snippets using only the filename and I don't have to think about a relative path from the page where I am using the snipppet:  
  
<pre>&#45;-8<-- "github-flow-snippet.html"</pre>  
  
!!! hint  
 Adding or changing a snippet sometimes requires restarting the webserver before  
 the changes show up in the live preview.  
  
## Building and publishing  
  
The `mkdocs build` command builds the website in a directory called `site`. 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 `site` directory, create a file called  
 `.gitignore` at the top level directory of the MkDocs project with the following  
 contents:  
   
 ```  
 site/  
 ```  
  
 If you use `git add` to add your `.gitignore` file to change tracking, then  
 it will apply to anyone who clones the repo&mdash;meaning that no one will   
 add built HTML pages to Git.  
# Overview  
  
## Ingredients  
  
<table>  
 <tr>  
 <td><b><a href="../../tools/tools-editors/">Markdown editor</a></b></td>  
 </tr>  
 <tr>  
 <td><b><a href="../../tools/tools-pandoc/">Pandoc</a></b></td>  
 </tr>  
</table>  
  
## Hard ones  
  
  
  
 - Documentation with GitFlow  
 - Collaborate with Distributed  
 - Just taking notes - Joplin or Typora/Dropbox  
 - Git Wiki  
 - Slides  
 - Word docs  
 - eBook  
 - PDF  
# Overview  
  
## Ingredients  
  
<table>  
 <tr>  
 <td><b><a href="../../tools/tools-editors/">Markdown editor</a></b></td>  
 </tr>  
 <tr>  
 <td><b><a href="../../tools/tools-pandoc/">Pandoc</a></b></td>  
 </tr>  
</table>  
  
## Hard ones

* Documentation with GitFlow
  + Collaborate with Distributed
  + Just taking notes - Joplin or Typora/Dropbox
  + Git Wiki
  + Slides
  + Word docs
  + eBook
  + PDF

Relative paths to images have to be relative to the script, so you can either copy all the images to a place that makes sense or run the script in a place that's relatively correct. I ran the script from a place where `../img/whatever.png` would work.

pandoc -o markdown-dreams.epub title.txt ../index.md ../getting-started/getting-started.md ../tools/tools-editors.md ../tools/tools-storage.md ../tools/tools-git.md ../tools/tools-git-setup.md ../tools/tools-git-basics.md ../tools/tools-publishing.md ../recipes/recipes-notes.md ../recipes/recipes-git-wiki.md ../recipes/recipes-centralized-workflow.md ../recipes/recipes-gitflow.md ../recipes/recipes-hugo.md ../recipes/recipes-mkdocs.md ../recipes/recipes-pandoc-word.md ../recipes/recipes-pandoc-ebook.md ../recipes/recipes-pandoc-pdf.md ../recipes/recipes-pandoc-web.md ../recipes/recipes-slides.md ../recipes/recipes-slides-dzslides.md ../recipes/recipes-slides-remark.md ../resources/markdown-cheatsheet.md ../resources/resources-glossary.md ../resources/templates.md ../resources/links.md

#!/bin/bash

pandoc -o markdown-dreams.epub  
title.txt  
../index.md  
../getting-started/getting-started.md  
../tools/tools-editors.md  
../tools/tools-storage.md  
../tools/tools-git.md  
../tools/tools-git-setup.md  
../tools/tools-git-basics.md  
../tools/tools-publishing.md  
../recipes/recipes-notes.md  
../recipes/recipes-git-wiki.md  
../recipes/recipes-centralized-workflow.md  
../recipes/recipes-gitflow.md  
../recipes/recipes-hugo.md  
../recipes/recipes-mkdocs.md  
../recipes/recipes-pandoc-word.md  
../recipes/recipes-pandoc-ebook.md  
../recipes/recipes-pandoc-pdf.md  
../recipes/recipes-pandoc-web.md  
../recipes/recipes-slides.md .  
./recipes/recipes-slides-dzslides.md  
../recipes/recipes-slides-remark.md  
../resources/markdown-cheatsheet.md  
../resources/resources-glossary.md  
../resources/templates.md  
../resources/links.md

# Overview  
  
## Ingredients  
  
<table>  
 <tr>  
 <td><b><a href="../../tools/tools-editors/">Markdown editor</a></b></td>  
 </tr>  
 <tr>  
 <td><b><a href="../../tools/tools-pandoc/">Pandoc</a></b></td>  
 </tr>  
</table>  
  
## Hard ones  
  
  
  
 - Documentation with GitFlow  
 - Collaborate with Distributed  
 - Just taking notes - Joplin or Typora/Dropbox  
 - Git Wiki  
 - Slides  
 - Word docs  
 - eBook  
 - PDF  
# Overview  
  
## Ingredients  
  
<table>  
 <tr>  
 <td><b><a href="../../tools/tools-editors/">Markdown editor</a></b></td>  
 </tr>  
 <tr>  
 <td><b><a href="../../tools/tools-pandoc/">Pandoc</a></b></td>  
 </tr>  
</table>  
  
## Hard ones  
  
  
  
 - Documentation with GitFlow  
 - Collaborate with Distributed  
 - Just taking notes - Joplin or Typora/Dropbox  
 - Git Wiki  
 - Slides  
 - Word docs  
 - eBook  
 - PDF  
# Create a standalone presentation  
  
  
  
## Ingredients  
  
<table>  
 <tr>  
 <td><b><a href="../../tools/tools-editors/">Markdown editor</a></b></td>  
 </tr>  
 <tr>  
 <td><b><a href="../../tools/tools-publishing/#pandoc">Pandoc</a></b></td>  
 </tr>  
 <tr>  
 <td><b>Powerpoint, Google Drive, or LibreOffice</b></td>  
 </tr>  
</table>  
  
  
## Slides  
  
Making Slides with Pandoc: the Basics  
  
Pandoc is an everything-to-everything converter, and one of its neatest tricks is turning Markdown into slides. If you can do simple things on the command line, you will have no trouble: Pandoc is easy to install and use.  
  
When using Pandoc, you can start your Markdown file with YAML metadata or a simple block like this:  
  
% Title  
% Author Name  
% Date  
  
Then write some Markdown for your slides. Pandoc uses the following rule to figure out what header level to use for slide titles:  
  
 By default, the slide level is the highest header level in the hierarchy that is followed immediately by content, and not another header, somewhere in the document. In the example above, level 1 headers are always followed by level 2 headers, which are followed by content, so 2 is the slide level. (source)  
  
That seems a little complicated, so I played with the Markdown a bit to see what worked in different presentation formats. In general, an H1 or H2 header works for a section header or slide title. Most other Markdown works as you would expect. Some presentation tools work best if you separate slides with three hyphens (---).  
  
Building a presentation with Pandoc is simple:  
  
pandoc -t <format> -s myslides.md -o myslides.htm  
  
### Powerpoint  
  
With Pandoc, you can go from Markdown directly to Powerpoint. This method offers a great deal of formatting flexibility. Columns, tables, sub-bullets, and images come out just the way you would expect them to look. I had no trouble opening a Pandoc-rendered .pptx file in Powerpoint or importing it into Google Slides.  
  
![](../img/slides-pandoc-powerpoint.png)  
  
Here is some sample Markdown:

# Section Title

## Slide TitleText on a slide:

* Bullet
* Bullet
* Bullet

Speaker notes go here

For Powerpoint slides, Pandoc doesn’t require the hyphen separator. The above Markdown contains two slides: one with an H1 as a section title and another that uses an H2 for the slide title. Notice the ::: — Pandoc’s fenced div syntax, which lets you do a lot of tricks in various formats. Here, it is just used to delineate the speaker notes.  
  
The Pandoc command to build a .pptx file is simple:  
  
pandoc myslides.md -o myslides.pptx  
  
As you would expect, you don’t need the -s flag (because what is the alternative to a standalone Powerpoint presentation?) and you can open the resulting .pptx file in Powerpoint. But there’s one more piece of magic:  
  
pandoc myslides.md -o myslides.pptx --reference-doc another.pptx  
  
When you run the command with the --reference-doc parameter, Pandoc takes the theme from the specified existing Powerpoint file and applies it to the new one you are creating.  
  
That means if you have a theme you like, you can create a whole presentation in that theme — no one will know you started with Markdown. This feature doesn’t always work with heavily modified presentations or corporate templates, but I was able to get it to work with built-in templates and one template that I had customized.  
  
That fenced div notation lets you create columns using nested divs without writing <div> tags in HTML. Take a look at this example:

Left column:

* Bullet
* Bullet
* Bullet

That translates to a <div class="columns"> containing two <div class="column"> tags that Pandoc can understand. It uses these to create a slide with columns. Notice that the opening and closing of the outer div has a whole bunch of : characters instead of just three. This is for readability; you can use as few as three if you like.  
  
  
![](../img/slides-pandoc-powerpoint-columns.png)  
  
The curly braces let you 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 Word or other long text documents, this is handy because you can set anchors on headings and link to them internally.  
  
Above, we are just using the attributes to add the column and columns classes to the fenced divs to create columns. But there is another cool attributes trick from Pandoc that we can use — scaling an image:

Alt text

Alt text

When Pandoc renders the image, it is scaled to a percentage of the container where it resides (a slide or column, for example). The alt text is used for a caption.  
# Create a DZSlides presentation  
  
  
  
## Ingredients  
  
<table>  
 <tr>  
 <td><b><a href="../../tools/tools-editors/">Markdown editor</a></b></td>  
 </tr>  
 <tr>  
 <td><b><a href="../../tools/tools-publishing/#pandoc">Pandoc</a></b></td>  
 </tr>  
</table>  
  
  
## DZslides  
  
If you want to do one of those slick presentations with big images and very few words, DZslides is an easy way to do it.  
  
Heres some sample Markdown:

# H1 or H2 is a Section Title

Normal Text for Slide Title

* Bullet
* Bullet

# Slide title

Normal text:

* Bullet
* Bullet
* Bullet

??? Speaker notes go here

Remark provides some rudimentary CSS to style slides. The first slide uses an H1 header as a section header. The simplified CSS code class: center, middle puts it in the middle of the slide. I added smaller text for the author and date, using .right to right-align it. You can use.right, .left, and .center to align text in different positions on the slide. This does not mean you can easily create columns in Remark. You can do it, but it requires authoring more complicated CSS.  
  
You can use H1 or H2 for slide titles — I prefer H1. Speaker notes are set off with ??? and the usual --- separator denotes slide boundaries.  
  
![](../img/slides-remark.png)  
  
You don’t have to build a Remark presentation; it just is. You can open it in a text editor and a browser. Any changes you save from the text editor show up in the browser when you reload. Like other browser-based slide tools, remember that you need to keep your images where the HTML expects to find them.  
  
There is one important choice to make with Remark: you can work with a standalone presentation containing the entire JavaScript that makes it work; a standalone presentation that references the JavaScript from another local file; or a presentation that references the JavaScript from online. The first option is longest but most “standalone,” and the last is the shortest (and therefore easiest to edit) but requires connectivity.  
  
  
# Markdown cheatsheet  
  
<table>  
 <tr>  
 <td><b>Headers</b></td>  
 <td>  
 <pre>  
# H1  
## H2  
### H3  
#### H4  
##### H5  
###### H6  
 </pre>  
 </td>  
 </tr>  
 <tr>  
 <td><b>Line breaks and paragraphs</b></td>  
 <td>  
 <pre>  
Consecutive lines  
of any length  
run together and become a single paragraph.  
  
An empty line ends the paragraph and starts a new one.  
  
Two spaces at the end of a line   
Forces a line break without   
Starting a new paragraph.  
 </pre>  
 </td>  
 </tr>  
 <tr>  
 <td><b>Text effects</b></td>  
 <td>  
 <pre>  
\*emphasis\* (italics)  
\_emphasis\_ (italics)  
\*\*strong\*\* (bold)  
\_\_strong\_\_ (bold)  
\*\*combined \_emphasis\_\*\*  
`code`  
~~strikethrough~~  
 </pre>  
 </td>  
 </tr>  
 <tr>  
 <td><b>Lists</b></td>  
 <td>  
 <pre>  
Bullet list:  
  
\* Asterisks, plus signs, or minus signs  
  
 Indent two spaces to add another paragraph in a list item  
\* Another list item  
  
Numbered list:  
  
1. The numbers don't matter.  
1. Markdown numbers when it renders  
   
 Indent three spaces to add another paragraph in a list item  
1. Lists inside lists:  
 \* Ordered and unordered lists work  
 </pre>  
 </td>  
 </tr>  
 <tr>  
 <td><b>Links</b></td>  
 <td>  
 You can define links inline or by reference. Reference links have a  
 label&mdash;often, the simplest thing to do us just use a number, like this:  
 <code>[1]</code>  
 <pre>  
This is an [inline-style link](https://www.example.com).  
A link can have a [title](https://www.example.com "Example title").  
Links can be defined later by [reference][label text or number].   
   
...  
  
[label text or number]: https://www.example.com/any/label/text/is/fine  
 </pre>  
 </td>  
 </tr>  
 <tr>  
 <td><b>Images</b></td>  
 <td>  
 You define an image just like a link, but with an exclamation mark.  
 <pre>  
Inline: ![alt text](https://www.example.com/images/some-image.png).  
Reference ![alt text][label text].   
   
...  
  
[label text]: https://www.example.com/images/some-image.png  
 </pre>  
 </td>  
 </tr>  
 <tr>  
 <td><b>Code blocks</b></td>  
 <td>  
 There are two ways to create code blocks, both of which can be embedded in lists.  
 <pre>  
In standard Markdown, indent by 4 spaces:  
  
 10 PRINT "HELLO"  
 20 GOTO 10  
  
Many Markdown flavors provide the capability to do a fenced code block,   
surrounded by three backticks. You can indicate the language for syntax  
highlighting:  
  
&#96;`` basic  
  
10 PRINT "HELLO"  
20 GOTO 10  
  
&#96;``  
 </pre>  
 </td>  
 </tr>  
 <tr>  
 <td><b>Blockquotes</b></td>  
 <td>  
 <pre>  
> Use a greater-than sign plus a space at the beginning of a paragraph. You can add it to the  
start of every line, or just the first line in the paragraph.  
  
> But it looks better to put it at the start of every line. It makes it easier to  
> see which lines are part of the blockquote.  
 </pre>  
 </td>  
 </tr>  
 <tr>  
 <td><b>Tables</b></td>  
 <td>  
 HTML tables are more powerful, more flexible, and more reliable. But some flavors  
 of Markdown include simple table notations. Default alignment is left.  
 <pre>  
| Header | Row is | mandatory |  
| --------------- |:---------------:| ------------------:|  
| Separate header | Colons on both | Colon on the right |  
| from other rows | sides align the | aligns the row to |  
| with hyphens | row to center | the right |  
 </pre>  
 </td>  
 </tr>  
 <tr>  
 <td><b>Horizontal rule</b></td>  
 <td>  
 Three ways to add a horizontal rule&mdash;three or more hyphens, asterisks, or underscores:  
 <pre>  
---  
  
\*\*\*  
  
\_\_\_  
 </pre>  
 </td>  
 </tr>  
 <tr>  
 <td><b>Comments</b></td>  
 <td>  
 There are two ways to write comments: HTML-style comments, which are rendered as  
 part of the HTML source code, and two kinds of link-style comments that are a bit of  
 a hack and don't get rendered at all.   
 <pre>  
&lt;!-- HTML-style comments --&gt;   
 are visible in Markdown and in rendered HTML.   
   
[//]: # "This comment"   
 is only visible in the Markdown and doesn't get rendered.   
   
[comment]: <> "This comment"   
 is only visible in the Markdown and doesn't get rendered.   
 </pre>  
 </td>  
 </tr>  
 <tr>  
 <td><b>HTML and special characters</b></td>  
 <td>  
 HTML is part of Markdown. You can use HTML to insert tables, formatting, special  
 characters, and other features that Markdown doesn't support directly. HTML tables  
 are often better and easier than Markdown tables. You can escape characters  
 with a backslash (\) or use HTML codes:  
 <pre>  
&amp;mdash; em-dash   
&amp;ndash; en-dash   
&amp;#35; octothorpe   
&amp;#42; asterisk   
 </pre>  
 </td>  
 </tr>  
</table>  
# Glossary  
  
admonition  
: A note, warning, or other call-out that draws attention to a block of content.  
  
dynamic site  
: See \*static site.\*  
  
package manager  
: A tool for installing software. See [Getting started](../../getting-started/)  
  
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  
  
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.\*   
  
WYSIWYG  
: "What you see is what you get"&mdash;an editing experience that mimics the appearance of the document in its final form.  
  
  
  
# Templates  
  
## DZSlides  
  
Tell where you put it, link to recipe

% DZslides template for Pandoc % Peter Conrad % 26 November 2019

# Section titles

* H1 or H2
* Centered on slide
* Not much room below

How to do a slide

* Use --- to separate slides
* Use regular text and bullets
* H1 or H2 for a section title

# Images

You can use images.

* Provide width and height
* Keep them with the HTML file

Full-screen image with alt text

Full-screen image with alt text

# Columns

Columns work

**Lists** in columns don’t work

# Syntax

Emphasis

* **bold**
* *italic*
* ***both***
* ~~strike~~

Blockquotes look like this

Lists

* Bullets work
  + Indenting works
* Ordered lists don’t
  + By themselves, they go to the left edge
  + As a sub-list, they outdent to the next level

Incremental “build” slides

* Incremental slides work
* This is how they look
* It’s fine

A slide with a pause can work.

. . .

Or can it?

Tables work but they are not well padded and they are stuck at the top left of the slide.

Don’t use inline code or code blocks. They don’t look quite right.

# Thank you

## Powerpoint  
  
Tell where you put it, link to recipe

% Pandoc Template for Powerpoint % Peter Conrad % 22 November 2019

# Section Title

## How to do a slide

You’ll be happiest if you use:

* H1 for section titles
* H2 for slide titles
* H3 for additional headings in slides

Speaker notes go here

## Two-column slide with image

Left column:

* Looks good
* Stays over here
* Lets the image do its thing

Speaker notes go here

## Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Heading | Heading | Heading | Heading | Heading |
| Cell | Cell | Cell | Cell | Cell |
| Cell | Cell | Cell | Cell | Cell |
| Cell | Cell | Cell | Cell | Cell |
| Cell | Cell | Cell | Cell | Cell |
| Cell | Cell | Cell | Cell | Cell |

Speaker notes go here

## Limitations

Things that don’t work

* HTML in your Markdown
* CSS
* Builds
  + (pauses or incremental lists)
* Inline monospace

Speaker notes go here

## Things that work very well

* Images resize nicely (but don’t fill the screen)
* Tables look nice
* You don’t have to take your images with you

Speaker notes go here

## An image

Alt text looks like this

Alt text looks like this

Speaker notes go here

## Headings

Headings H3 and lower look the same.

### H3

#### H4

##### H5

###### H6

Speaker notes go here

## Emphasis

* **bold**
* *italic*
* ***both***
* ~~strike~~

Speaker notes go here

## Bullet lists

* Bullets work
  + Indenting works
* Don’t try to do multiple lists on a single slide

## Ordered lists

1. Ordered lists work
   1. Sub-lists in ordered lists
2. Next item

Speaker notes go here

# Thank you

## Remark  
  
Tell where you put it, link to recipe

class: center, middle

# Title

# Agenda

1. Introduction
2. Deep-dive
3. …

# Introduction

</textarea>

``` # Links

Here:

–8<– “test.md”