



Thoth Documentation

Table of contents

1 Copyright

2 Feedback

3 Introduction to Thoth

4 What kind of a name is that?!

5 Features

- 5.1 Markdown based editing
- 5.2 Version control and collaboration
- 5.3 Plays nice with formal review processes
- 5.4 Document reuse and aggregation
- 5.5 Content validation
- 5.6 Skinning and templating
- 5.7 Search engine
- 5.8 Meta data
- 5.9 Supports Critic Markup
- 5.10 Configurable and extendable rendering pipeline
- 5.11 Open source, Apache license

6 Concepts

- 6.1 The repository
- 6.2 The Context
- 6.3 The Fragment
- 6.4 The Book
- 6.5 Meta tags
- 6.6 Meta data
- 6.7 Skin

7 Implementing a Review Process with Thoth

8 Using comments

9 Thoth Markdown basics

- 9.1 Table of Contents
- 9.2 Headers
- 9.3 Block quotes
- 9.4 Lists
- 9.5 Code blocks
- 9.6 Horizontal Rules
- 9.7 Links

- 9.8 Embedded images
- 9.9 Emphasis
- 9.10 Code
- 9.11 Backslash escapes
- 9.12 Include processing
 - 9.12.1 Include Markdown
 - 9.12.2 Include Images
 - 9.12.3 Include Source (code block)
- 9.13 Critic markup
- 9.14 Meta data
- 9.15 Tables
- 9.16 Inline HTML

10 Typical setup

- 10.1 Tools in the mix
- 10.2 Workflow

11 The Thoth Toolset

- 11.1 Git Primer
 - 11.1.1 Setting up your local Git repository
 - 11.1.2 Starting SourceTree for the first time
 - 11.1.3 Cloning the central repository
 - 11.1.4 SourceTree main screen
 - 11.1.5 Ready to start writing!
- 11.2 WYSIWYG
 - 11.2.1 Use the Thoth IncludeProcessor as a pre processor
 - 11.2.2 Use Thoth standalone with your browser
 - 11.2.3 Use an editor that does WYSIWYG
- 11.3 Skinning Documents and Screens
 - 11.3.1 Overriding the Skin dynamically
 - 11.3.2 Overriding the Skin based on the path of a document (automatically)
 - 11.3.3 Creating your own Skin
 - Creating the Skin descriptor file
 - Referencing Skin resources from your template
 - JSON support
 - Creating your own templates
 - Generic variables for all templates
 - Types used for variables
 - Book
 - Diff
 - MarkdownDocument
 - DocumentNode
 - Commit
 - ProcessorError
 - LineInfo
 - SearchResult
 - Fragment

- [ContentNode](#)
- [User](#)
- [Group](#)
- [template.index](#)
- [template.contextindex](#)
- [template.login](#)
- [template.html](#)
- [template.diff](#)
- [template.meta](#)
- [template.revisions](#)
- [template.validationreport](#)
- [template.search](#)
- [template.browse](#)
- [template.usermanagement](#)
- [template.userprofile](#)
- [template.error](#)
- [template.log](#)
- [Template helper functions](#)
- [Template example](#)

12 Installing Thoth

- [12.1 Getting hold of the software](#)
- [12.2 Configuration](#)
- [12.3 Running standalone](#)
- [12.4 WAR installation](#)

13 The configuration file

- [13.1 Required settings](#)
 - [13.1.1 defaultgroup](#)
 - [13.1.2 Repository settings](#)
 - [13.1.3 Database settings](#)
 - [13.1.4 Context settings](#)
- [13.2 Optional settings](#)
 - [13.2.1 Custom renderers](#)
 - [13.2.2 Using Pandoc as the HTML rendering engine](#)
 - [13.2.3 Format masks](#)
 - [13.2.4 File classification](#)
 - [13.2.5 Markdown processing options](#)
 - [13.2.6 Miscellaneous](#)
- [13.3 Sample configuration file](#)

14 User Management

- [14.1 Default groups](#)
- [14.2 Permissions](#)
- [14.3 Access control](#)

15 Technicalities

- [15.1 Minimum System Requirements](#)
- [15.2 Acknowledgements](#)

1 Copyright

Copyright (c) 2016 W.T.J. Riezebos

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

2 Feedback

I welcome comments and feedback. Do you need some missing piece of documentation? Is something not clear or do you have a general question? Would you like to discuss a new feature? Then why don't you create an issue at <https://github.com/widoriezebos/Thoth/issues>? Much appreciated!

3 Introduction to Thoth

Before we delve into the details of what Thoth is all about, lets listen to a conversation between Dave (a Documentation Writer) and an Ian (an IT guy). So Dave comes in on a blue Monday morning after a well deserved and kind of relaxing weekend (I'll spare you the details). In the coffee corner he finds Ian enjoying an espresso and while pressing the button for his first regular coffee of the day, Dave starts mumbling something about bloody Microsoft Word and revisions. We're human beings. We all enjoy talking about the moments when tools let us down, especially on Monday mornings. So Ian joins in.

Still using Word for your work then? Well yes, although I am beginning to wonder why. I got a new version of the documentation from Sally and unfortunately she used an older version of the one I had. Merging her changes really made a mess; let alone the fact that for the new version of Product we use a new template now. To make things worse, parts of what she did should also be copied to other documents that mention the concepts. Ian takes a few seconds to respond, pondering whether to put on his consulting hat or start talking about one of his disastrous recovery attempts. No, too painful so the hat it is.

With two releases of product every year I guess that version control in Word simply does not cut it, Ian begins. Dave is about to explain the elaborate folder structure on the shared network drive but then tends to agree there. Nobody ever finds what they need anyways, especially when looking for screenshots of a particular version of the software. What you need is a good version control system; something we in IT have had for decades Ian continues. With that in place you will never have a problem maintaining different versions of a particular manual and working an a document with Sally won't be much of a problem either. And placing all of your content in a single document is not a good idea, you should be able to compose documents of smaller parts so that these can be shared across different documents tailored to different audiences.

In reality Dave does not know what to make of this. Seems like an intelligent thing to do but this also sounds like cocky nerd talk to him. And Dave is not a nerd. Dave is frustrated. And frustration is what drives him to continue the conversation.

Ok, so if I get this fancy merging version control system with these nice composition tricks in place what then? Apart from finally getting to grips with content and collaboration, are there more goodies IT can offer? Ian did not expect Dave to catch on this quickly and now get's the notion of unexpected work coming his way. What Ian meant to achieve was more along the lines of showing off the tricks of his trade, not advertising for more work.

The way forward for Ian in these situations is a proven recipe: just add more to the mix until it all sounds like a very daunting task altogether. And then forget about it. Done.

So on Ian went: Searching and then finding what you need 'Search Engine style' of course. And being able to see who did what-when-to-which document etc. etc. And some smart templating rendering mechanism to publish the documents to HTML, PDF, EPUB and whatever you need. The possibilities are endless!

"Wow, sounds like you have experience there" Dave responds. Well, we use VI ourselves says Ian and he continues: that also does the trick. Kind of. By now Ian is hoping that this all sounds rather expensive and indeed to Dave it does. Mission accomplished.

So Dave presses the coffee button again and goes back to his personal Word hell for the week. Such is life.

But not for you, because you are reading this. A system described by Ian is not expensive at all: actually, it is here for free. It is Thoth.

4 What kind of a name is that?!

Yes, Thoth is not the easiest word to pronounce. (Unless copious amounts of beer are consumed because then almost anything you say automatically sounds like 'Thoth'). When sobered up however it turns out that typing the name Thoth with your keyboard is remarkably easy. Just try it and I think you will agree. Easily writing down something that is not immediately clear when spoken is at the center of what makes a good documentation system. And it turns out that Thoth is also an Egyptian deity that (among a lot of other things) served as a Scribe to the Gods (Ok you got me there).

5 Features

5.1 Markdown based editing

When writing a document you should not be distracted by form and layout, you should be able to focus on content. This is what Markdown enables you to do. In short Markdown is a wiki style of text editing that has very few rules for adding structure to a document. The content is stored as plain text, which makes it easy to process using a version control system for instance. Another benefit of using Markdown that it is kind of an emerging standard for markup. A lot of internet sites are using it, including Github, as their primary vehicle to editing documentation. It makes sense to use Markdown as the basis for a documentation system because there are a lot of editing solutions out there that support Markdown.

5.2 Version control and collaboration

Since the content is stored as 'Markdown fragments', a version control system is very capable of making sense of changes. Thoth uses Git as the basis for it's version control system. With that comes branching, merging, revision control and distributed collaboration out of the box. And when used correctly, maintaining a set of documentation for different versions of a product becomes relatively easy. What you would do is create a branch for every version of your product that is out there. Changes and additions to an older version of a document are then easy to track and merge to any other branch (for one of the newer versions). If you want or need this of course. You could also just use the version control system just as a way of making sure that nobody will ever inadvertently overwrite changes of a colleague.

5.3 Plays nice with formal review processes

The source of your documentation is plain text which plays nice with (code) review tools such as Atlassian Crucible. It is easy to implement a formal review process where changes to your documentation are only merged back to the main branch after a formal review has taken place (and was approved). Basically you use the Git pull request concept to remain in control of your repository and use a tool like Crucible to review and accept any (incoming) changes to the main branch.

5.4 Document reuse and aggregation

When writing a large set of documents there will be several pieces of text that can be shared among them. Typically these are concepts and definitions that have to be part of various documents for different audiences. Screenshots are another example and when isolated they can easily be updated for a new version of product without affecting anything else. Thoth has a robust Include System that enables recursive include processing for Markdown documents. To ease maintenance of interdependencies (the links between document fragments or images) Thoth supports 'Soft Links' that are like

an alias for a link. When you move a document fragment or rename it you will then only have to update the soft link. Any document that uses the Soft Link does not have to be touched.

5.5 Content validation

Thoth validates links and bookmarks and is able to report on problems. This can be per document or for the entire library that you maintain.

5.6 Skinning and templating

You can apply any number of skins to Thoth itself and the documents it can render. A skin can be automatically selected based on the location of a document in your library, so you can tailor the look and feel of a set of documents to your needs. Templates are written in the Velocity template syntax and skins support inheritance. When using multiple skins for your library you do not need to copy a lot of stuff around if you use skin inheritance; just add what you want to change to your new skin and specify from what skin it should inherit.

5.7 Search engine

Thoth has an embedded search engine that automatically updates it's index when changes are submitted (pushed) you the central documentation repository. The search engine supports complex expressions to help you narrow down when a simple query does not cut it.

5.8 Meta data

You can add meta data to your documents that can help you organize (and search) your library. Documents can be categorized based on folder structure or meta tags and meta tags can be used in templates used for rendering your output. The syntax adheres to [MultiMarkdown](#) although currently metadata cannot span multiple lines.

5.9 Supports Critic Markup

You can propose additions, deletions, changes and add comments like you would do in MS Word right in your markdown. You can then choose to have these to be part of the rendered output, or show them for review until either accepted in the final document version. The syntax is based on [Critic Markup](#)

5.10 Configurable and extendable rendering pipeline

Out of the box Thoth can render your Markdown documents to CSS styled HTML pages. By configuring additional outputs like PDF or EPUB you can easily render the same content to any format you want as long as you have a rendering tool that does that for you. Very good options here are [Prince](#), [Wk<html>to*pdf*](#) and [Pandoc](#). As long as you

have a command line tool that can render Markdown, CSS rich or plain HTML then you can plug it into Thoth.

5.11 Open source, Apache license

Thoth is open source, free and available under the [Apache license](#). This basically means you can do whatever you like with Thoth, including use in commercial environments as long as you adhere to the terms of the license.

6 Concepts

In the documentation of Thoth (or some of the screens) there is reference of a few concepts that are defined below.

6.1 The repository

A Repository is the entire set of documents that are related in some way. For unrelated documents it makes sense to set up more than one repository (for instance different sets of documentation for different products would have one repository per repository). A Repository is published with Thoth using a corresponding Context (which basically is just a name for a repository). There are different kinds of repositories that Thoth supports: just a directory on your harddrive, a Git repository or a Zip somewhere on your filesystem.

6.2 The Context

A Context exposes the content of a Repository through the Thoth (user) interface. You can also regard a Context as just a friendly name for your (potentially more technically named) Repository. Note that a Context is by definition unique within Thoth and is associated with exactly one Repository.

6.3 The Fragment

A fragment is just a piece of text. It can be an entire chapter or just a short paragraph. It is up to you to determine the size and contents of your Fragments, basically Fragments are the building blocks of Books (or other fragments). The extension of a Fragment is '.md' for Markdown (unless you changed that in the configuration).

6.4 The Book

A Book is a special kind of Fragment because it is considered to be the root or container of included Fragments. This way Thoth can create indexes of Books automatically and determine whether there are any unused (dead) Fragments in your repository. The extension of a Book is '.book' (unless you changed that in the configuration).

6.5 Meta tags

Any fragment (and book) can contain Meta tags in it's header. For one, any meta tag can be used for searching the repository (just enter a query in the form of 'metatag:value' to search for documents/fragments with that meta tag set). Other use of Meta tags is for instance when rendering the Context Index page (for instance showing the author of the document next to the title).

6.6 Meta data

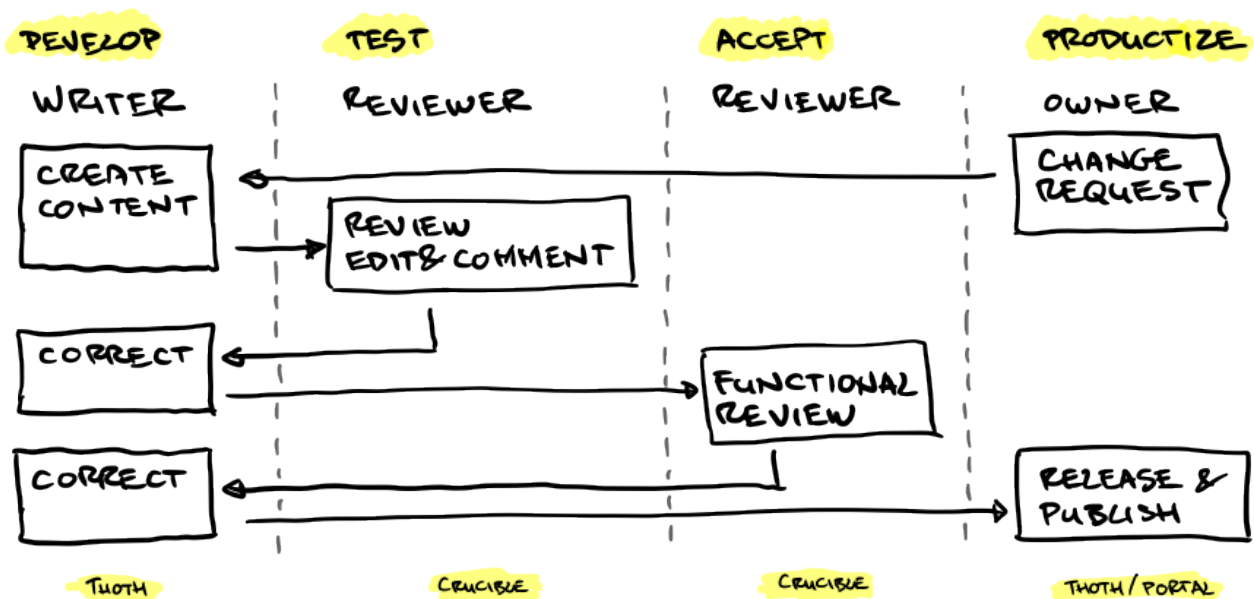
Thoth can also display additional Meta data about any document, showing document structure, the direct and indirect use of a fragment, revision history and any validation errors that occurred during include processing.

6.7 Skin

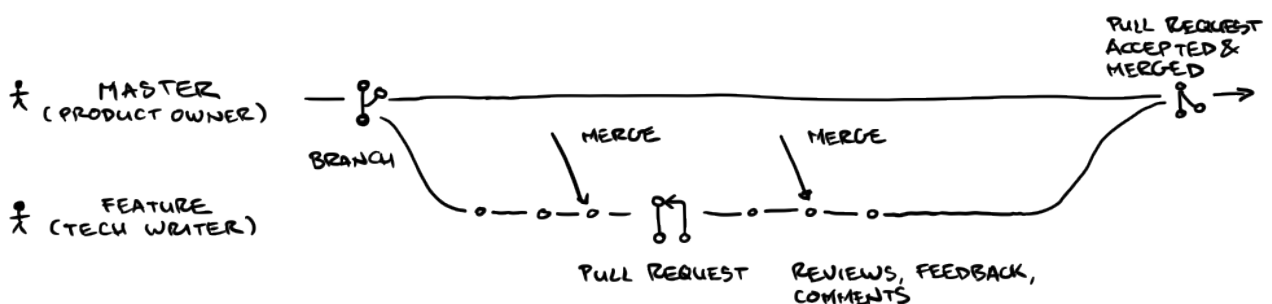
Any screen in Thoth and any document / fragment that is rendered can be completely customized in terms of layout and visual design. The mechanism for this is called a Skin. Technically a Skin consists of a descriptor file, a few Velocity templates and potentially some additional resources (css, images, javascript). A Context can make use of any number of Skins, that can be selected based on the path of a certain Document or Fragment.

7 Implementing a Review Process with Thoth

In a more formal environment working with Thoth might involve a review process that ensures nothing ends up in the final documentation that was not supposed to be there (or be there in a certain way). One way to do this is to use the Git version control system together with a tool like <https://www.atlassian.com/software/crucible>. The basic concept is that you create a new branch of the documentation that is worked on, and that this branch is only merged back when fully reviewed and approved (using a so called pull request, so the owner of the main branch stays in control).



The underlying method of implementing and accepting changes (basically the arrows in the diagram above) is described by the git flow process as described in this tutorial <https://www.atlassian.com/git/tutorials/comparing-workflows> or in this fairly detailed article: <http://nvie.com/posts/a-successful-git-branching-model/> (just look at the concepts; do not get intimidated by the command line code that is mentioned there because you would probably want to use a GUI client anyways)



Basically it comes down to what is depicted in the diagram above, where in a formal process the branches have access control so that only the product owner is allowed to

accept the pull request and merge the feature branch back into the main branch.

8 Using comments

If there is no need for a formal review process you could also use the (light weight) commenting feature. Users who are allowed to comment on a document (not necessarily writers) can then add comments to the fragments of a book directly in Thoth.

9 Thoth Markdown basics

Currently Thoth uses the Markdown syntax as [defined originally by John Gruber](#). On top of that Thoth supports some extra's as defined by [PegDown](#) (depending how you have configured Thoth), [Critic Markup](#) and the [Metadata definition](#) as defined by MultiMarkdown. The include processing is kept close to how you would specify in Latex. Note that a lot of Markdown constructs require an empty line before it. If you see unexpected results with a header, a list or a table then usually this is resolved by placing an empty line directly above your construct.

Whitespace is generally not rendered. If you want an empty line between your paragraphs however you can simply enter two consecutive new-line characters.

9.1 Table of Contents

You can add a generated TOC section to your document by adding a `\tableofcontents` tag to your document. All headers present in the document will be used to create the TOC, using the level of the header to indent appropriately. Note that the `\tableofcontents` construct must appear (without any prefixes) at the beginning of a line.

```
\tableofcontents
```

9.2 Headers

Note: headers will be automatically numbered by Thoth (unless turned off in the configuration by setting `markdown.maxheadernumberlevel = 0`, default is 3). The following two methods of header specification are supported:

SetExt style (either use a '-' or a '=')

```
My header
=====
```

Atx style (1 - 6 hashes to denote the level)

```
#My Level 1 header
##My level 2 header
...
#####My level 6 header
```

9.3 Block quotes

You can use a `>` character to start a block quote.

```
> This is a block quote
```

9.4 Lists

Lists can be unordered or ordered, and can be nested. Use asterisks, pluses, and hyphens – interchangeably – as list markers, indent with a tab. Numbered lists should have a number suffixed with a ‘

- Unordered 1
- Unordered 2
- 1. Ordered 1
- 2. Ordered 2

9.5 Code blocks

Pre-formatted code blocks can be used to include source code and will not be interpreted as Markdown. To create a code block simply indent the text with a Tab or with 4 spaces.

```
Code block  
Taken literally
```

9.6 Horizontal Rules

You can include a horizontal rule in your output by placing three or more hyphens, underscores or asterisks in your text.

```
---
```

9.7 Links

Note that Thoth currently only supports inline links because of complications that arise when files are included (currently this is a restriction). When referencing documents in the library, you can either use an absolute path or a relative path. The absolute path will take the location of your context as the root during resolving.

Note that Thoth supports ‘Soft links’ which means that you can define an alias or substitution pattern as a short-cut in a file called `softlinks.properties` in the root of your context. You reference a soft link by prefixing it’s name (or substitution pattern) with a ‘:’. You might prefer soft links over hard links if you want to avoid having hard links all over the place or want to use them simply as a shorthand. Reorganizing your library without breaking links all over the place can be avoided using soft-links.

```
[Link text showing in the document](http://example.net/ "With a description")  
[A local document](/absolute/path/inlibrary/otherdocument.md)  
[A local document](relativefolder/otherdocument.md)  
[A local document](:mysoftlink)
```

9.8 Embedded images

If you prefix a link with a '!' then it will embed the contents of that link as an image. The link can point to a local or remote resource. Description and title are optional when specifying an image.

```
![Description of the image](images/Setup.png "Title of the image")

```

9.9 Emphasis

You can emphasize pieces of your text by wrapping them inside either two '*' or '_' characters

```
This is an example of a bold piece of text
Although you could also do that this way
```

9.10 Code

You can mark text as code inside a single line enclosing it with a '`' character

```
Use the `printf()` function.
```

9.11 Backslash escapes

With a '\' you can escape any character below that might be interpreted by Markdown otherwise:

```
\    backslash
`    backtick
*    asterisk
_    underscore
{}   curly braces
[]   square brackets
()   parentheses
#    hash mark
+    plus sign
-    minus sign (hyphen)
.    dot
!    exclamation mark
```

9.12 Include processing

There are three different ways of including content in your document; tailored to the type of included content: Markdown, images or raw source code.

9.12.1 Include Markdown

You can include any other markdown file (or any text file for that matter) by using a special include directive. The file specification can be either relative or absolute. Absolute paths will be taken as absolute from the context folder. You can provide an optional second argument that specifies the level adjustment for headers (negative or positive). When given a value of 1 the headers of the included file will be bumped up 1 level when including them in the result.

```
\include{somesubfolder/somedocument.md}
\include{somesubfolder/somedocument.md, 1}
\include{/absolute/path/to/somedocument.md}
```

9.12.2 Include Images

There is also include functionality for images using a wildcard. This enables you to add images from a folder without having to completely specifying their name. The header level for automatically determined headers can be specified as a separate argument. Specify 0 for no headers.

```
\includeimages{images/component/*.png, 1}
```

9.12.3 Include Source (code block)

You can include any text file as a Code Block, Thoth will prefix every line with a tab character making it a Code Block

```
\includecode{sources/SomeFile.java}
```

9.13 Critic markup

You can mark text with the syntax outlined below. Since this kind of markup is meant to work anywhere note that there is a space between the '{' and the directive to keep it from processing in this example. In your own markdown there should be *no space* after the opening '{' character. The Critic markup can be displayed during rendering by adding the request parameter `critics=show`, or displayed as-is with `critics=raw`. By default the critics are processed and the result will then be rendered.

```
Addition { ++My inserted text++}
Deletion { --My deleted text--}
Substitution { ~~Original text~>Changed into something else~~}
Comment { >>This is just a comment<<}
Highlight { ==Highlighted piece of text, probably followed by a comment==}
```

9.14 Meta data

You can add any number of meta data tags to your document. You do this by adding them **at the very top** of your document, separating the key (which must be one word

with no spaces) from the value with a ':' character. The first line that does not follow this rule will start normal Markdown processing. When including files, any meta tag in the included files is added if it is not encountered before. This means that meta tag with a specific key that is first encountered 'wins' and will not be overwritten by subsequent definitions. Meta data is searchable with the search engine and the key/values are also available to any template that wants to use them during rendering.

```
audience: writers, developers
title: Title of the document
author: Wido Riezebos
```

9.15 Tables

You can render a table using a '|' character as a separator between columns (make sure the number of '|' characters per row is always the same). You are required to define a header separator as the second line of your table definition, with at least one '-' as the contents. If you want alignment for a column use a ':' in the separator row to specify where to align (:- for left, -: for right and :: for center)

```
|Code|Description|
|:-:|-|-|
|1   | One       |
|2   | Two       |
```

9.16 Inline HTML

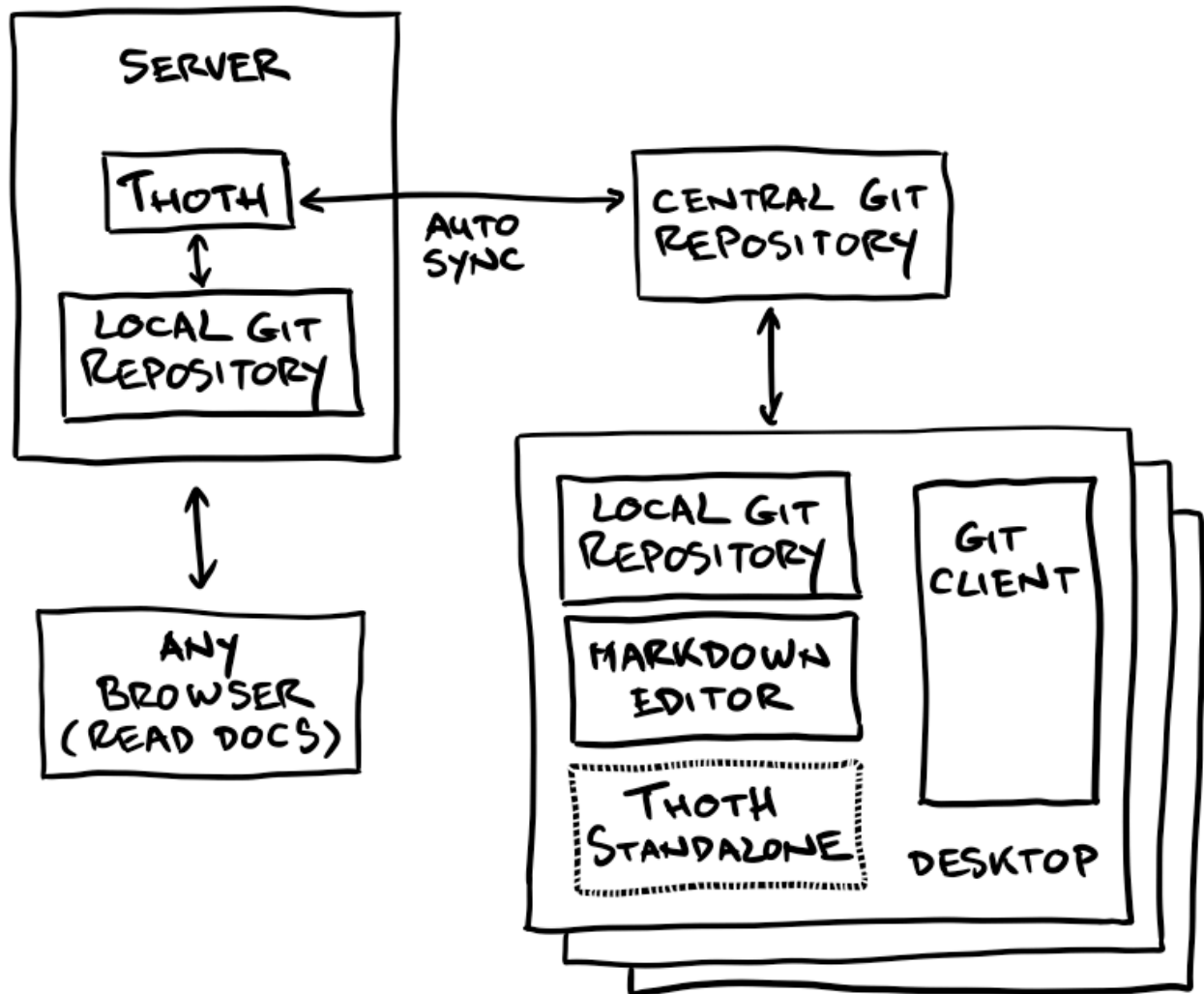
Although not encouraged, you can place HTML fragments directly in your markdown. Since this goes directly against the spirit of Markdown altogether this obviously should be used as the exception of the rule (i.e. title pages etc).

```
<b>Any html</b>
```

10 Typical setup

A typical setup consists of a Thoth server, a Git repository, a Git client and a Markdown editor (or just a plain text editor if you prefer).

To help you understand how this is all connected consider the following diagram:



Note that in this example Thoth standalone can be used on your desktop for WYSIWYG purposes. In this case you would edit your markdown, and after saving it you could simply refresh the corresponding document in your browser (connected to a locally running Thoth). Obviously the use of Thoth on your desktop is optional; if you have a tool like Marked2 in place (with Thoth running as a preprocessor) you might want nor need this.

10.1 Tools in the mix

Depending on your type of desktop there is a good choice of Markdown editors. For OSX there is a great app called [Ulysses](#) that you can use (enable 'external folders', enable 'inline links') and for Windows there are also good Markdown editors available

(have a look at [Haroopad](#)). If you have other suggestions please let me know so I can add them to the list of suggested apps. You might also want to have a look at a WYSIWYG solution; for OSX you could consider [Marked2](#). As a Git client I suggest you use the excellent [SourceTree from Atlassian](#), unless you are a command line tiger of course because then a regular Git client would do as well (and then you would use VI as an editor right :-)).

Note that it is also possible to run Thoth on your desktop, WYSIWYG can then be done by just refreshing your browser. Benefit of this is that it is true WYSIWYG obviously.

10.2 Workflow

Using the above setup you will edit documents with your Markdown editor, commit changes with SourceTree, Pull any changes from the central repository and then Push all your local changes (to the central repository). Thoth will synchronize automatically with the central repository, depending on the configuration your changes will end up on the Thoth server within 60 seconds. Once available on the Thoth server anybody will be able to access them in the various formats and skins that are configured (HTML, PDF etc).

11 The Thoth Toolset

Let's get familiar with the toolset. An important aspect of working with Thoth is version control, currently based on Git. Version control can be a bit daunting if you are new to the concept, but if you stick to the basics and use [a good Git client](#) you will quickly get the hang of it. (Note: For very simple setups you can skip the use of Git altogether and use the File System based content manager)

11.1 Git Primer

It is important to note that Git uses a local version control repository to store all it's information. The local repository is usually created by 'Cloning' the central repository. After cloning the trick is that this local repository can be synchronized with the central repository. This synchronization is done by you using 'Pull' and 'Push'. 'Pull' fetches information from the central repository and 'Push' writes your local changes to the central repository. Before you can 'Push' your changes however there are a few steps you need to do first:

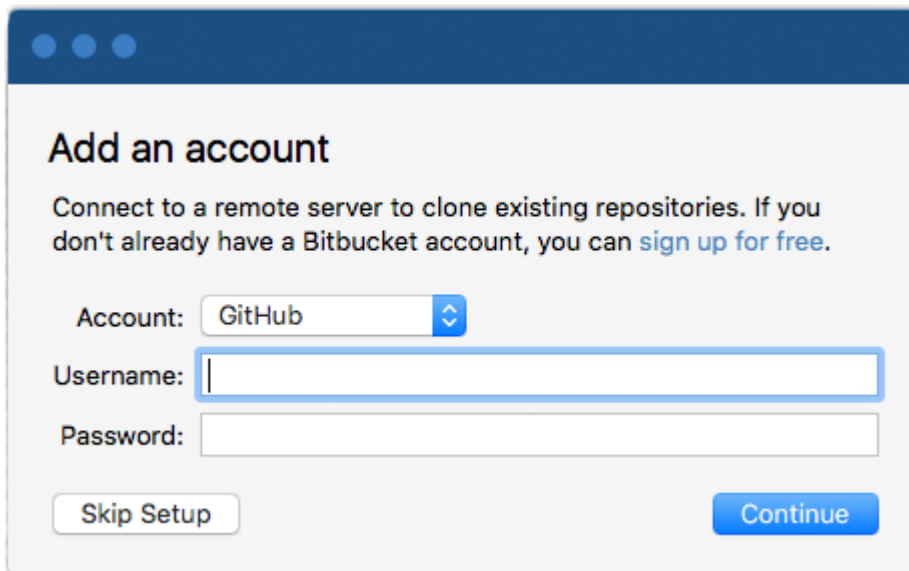
1. You must commit your local changes to your local repository
2. You must then Pull any changes that happened since your last Pull from the central repository
3. If there are conflicting changes (i.e. both you and somebody else changed something on the same line in the same file) you have to resolve them. You do this by editing the file containing the conflict until you are satisfied, and then mark the file 'resolved' in Git. All your changes have to be committed before you continue.
4. You can now Push your local changes to the central repository. In rare cases (on a very busy central repository) somebody might have pushed changes while you were resolving conflicts; in that case you have to retry from step 2.

11.1.1 Setting up your local Git repository

In the examples below we will use SourceTree as a Git client; but there is no pressing reason why you could not use any other Git client. The point is that you create a local Git repository where you will work on your documentation, and that you do this by cloning the central repository.

11.1.2 Starting SourceTree for the first time

When you start SourceTree for the first time you are asked to add an account. If you are familiar with Github then this is where you enter your Github account details. If you are still wandering what Git actually is all about then you might need some help to either set up a central repository (Github would be good) or skip Git altogether and continue with a file based repository. If you go for File Based then you can read the text below just for reference, but you will be missing out on a lot of goodies.



Add an account

Connect to a remote server to clone existing repositories. If you don't already have a Bitbucket account, you can [sign up for free](#).

Account:

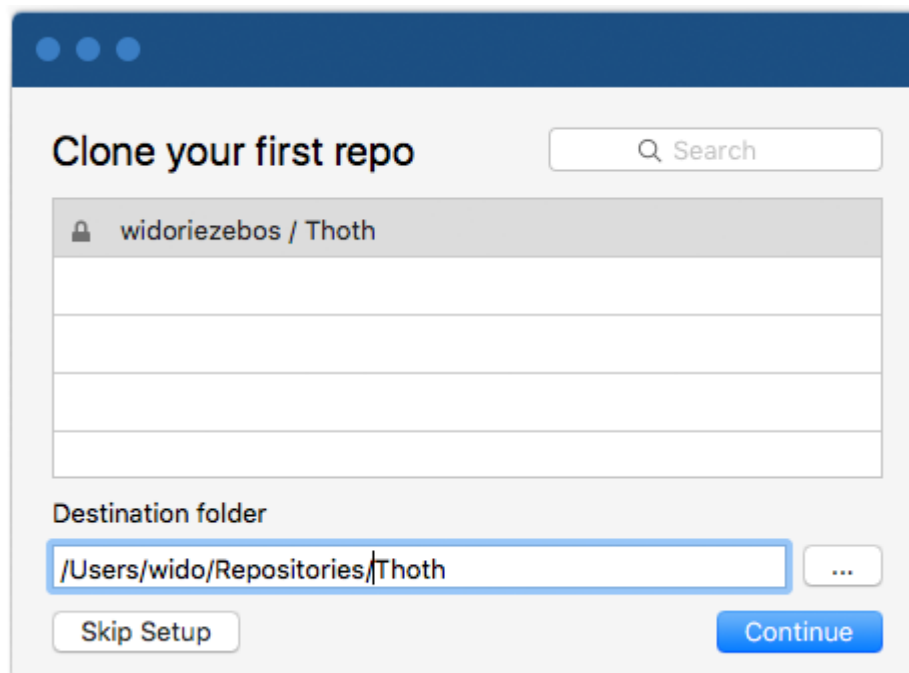
Username:

Password:

Enter your account type and details and click 'continue'.

11.1.3 Cloning the central repository

The next step is about creating a local repository by cloning the central (remote) repository. Determine where you want to place your local repository (I usually create a folder Repositories in my home folder and put all repositories in there)

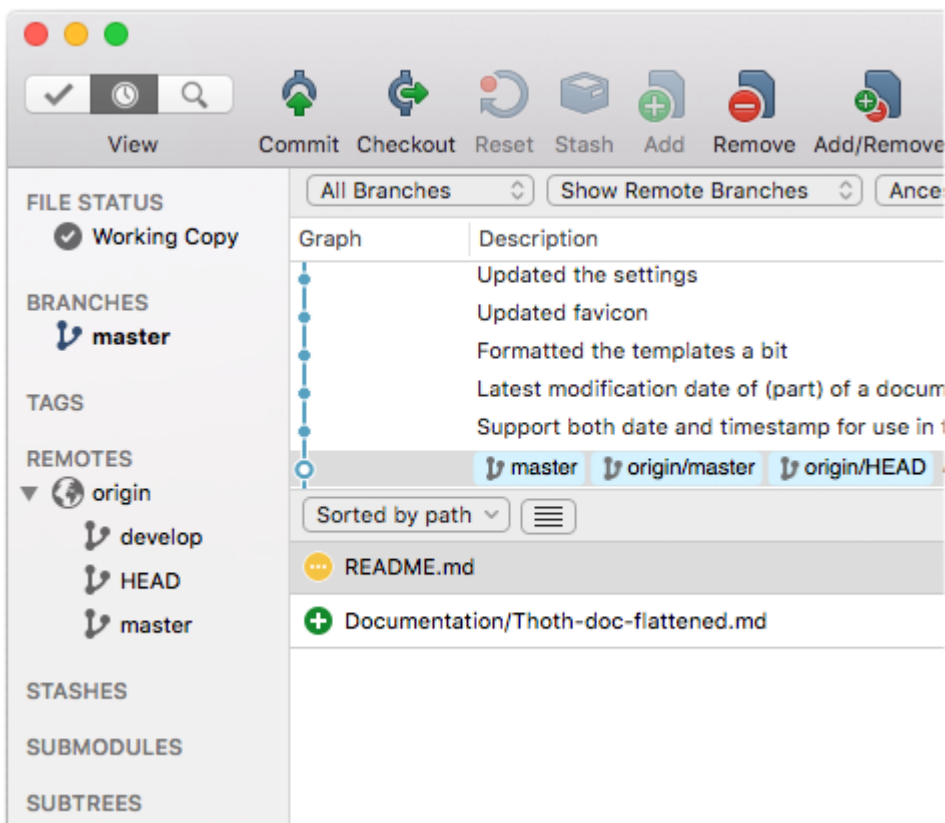


Clone your first repo

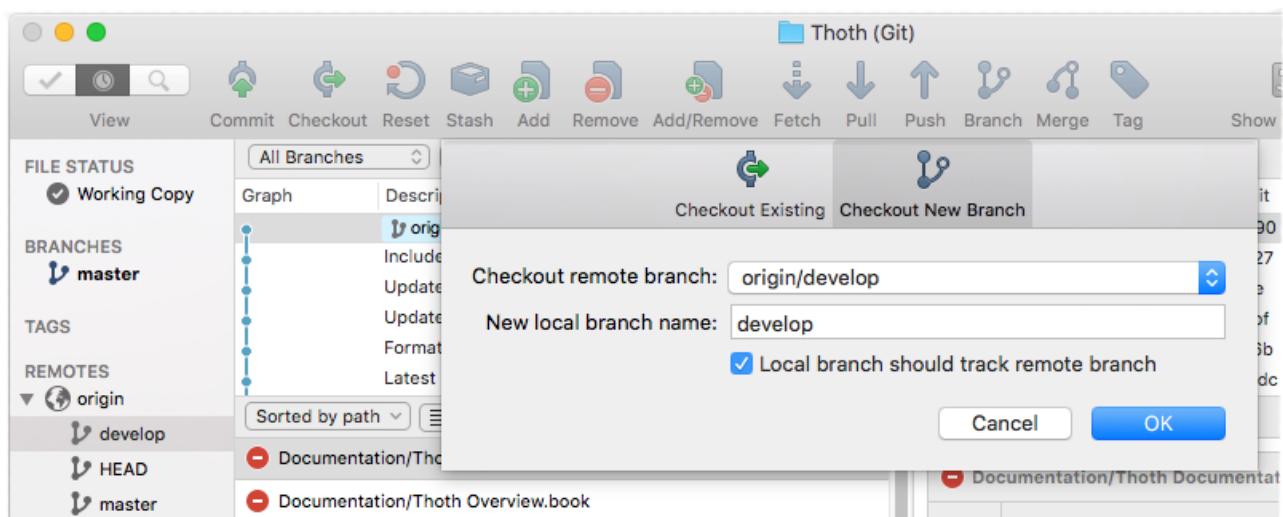
Destination folder

11.1.4 SourceTree main screen

After cloning the repository you should see the main screen of SourceTree. A few things to note here:



Under branches you see only one branch called 'master', and if you open up 'REMOTES' you see more branches that are not yet available locally. You can have one branch at a time in your local repository, it is like a snapshot of a particular version of your files. The master branch is not the one you want to work on. Usually you work on develop or a branch based on develop. To switch to develop, just double click on the REMOTES>Origin develop branch.

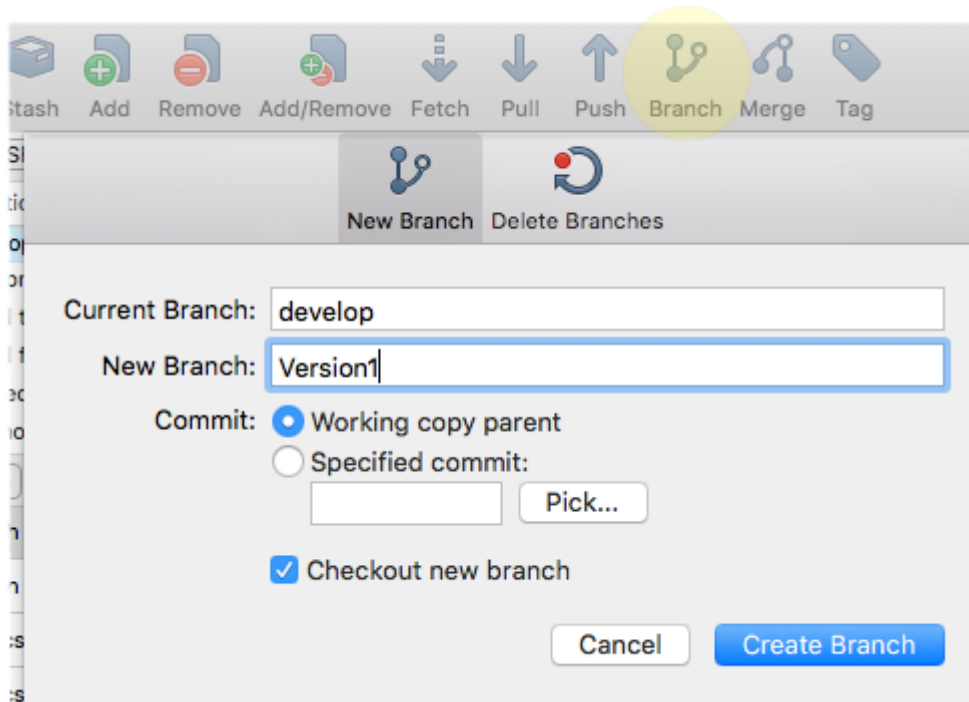


After doing that you have switched to the develop branch. Now you should think about the branches that you want to have.

Typically you would have one repository for a set of documents that logically belong to each other (part of the same product for instance). In this repository you would then have one branch per version of the product. This makes it very easy to promote changes

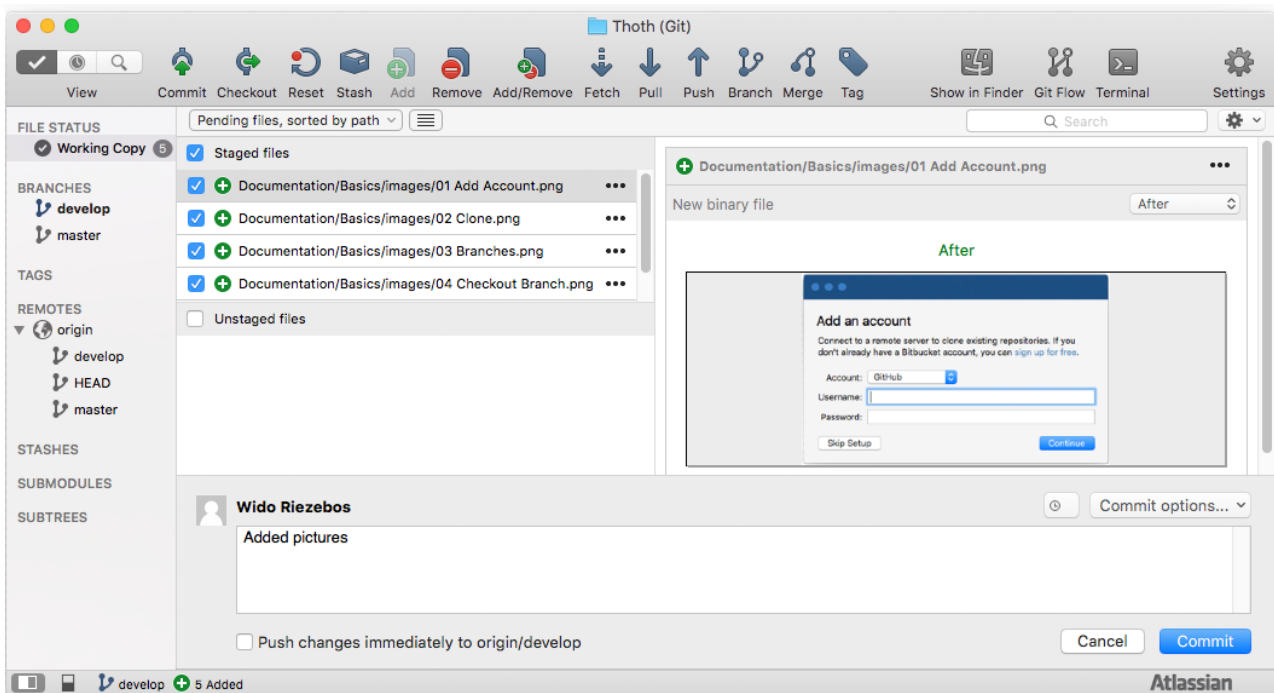
to an older version of the documentation to a new (even after the new branch was created) or back port documentation of a new version to an older version. Git makes it really easy to do by merging one branch into another and selecting the changes you want to have merged. This might sound a bit easier than it is in some cases but you should be able to find enough material on the web to become a Git expert in no time.

In any case, you should create a new branch based on develop and give it the name of a particular version of your (product) documentation. When you start working on the next version of the product documentation, you will create a new branch based on this initial version branch that you are about to create.

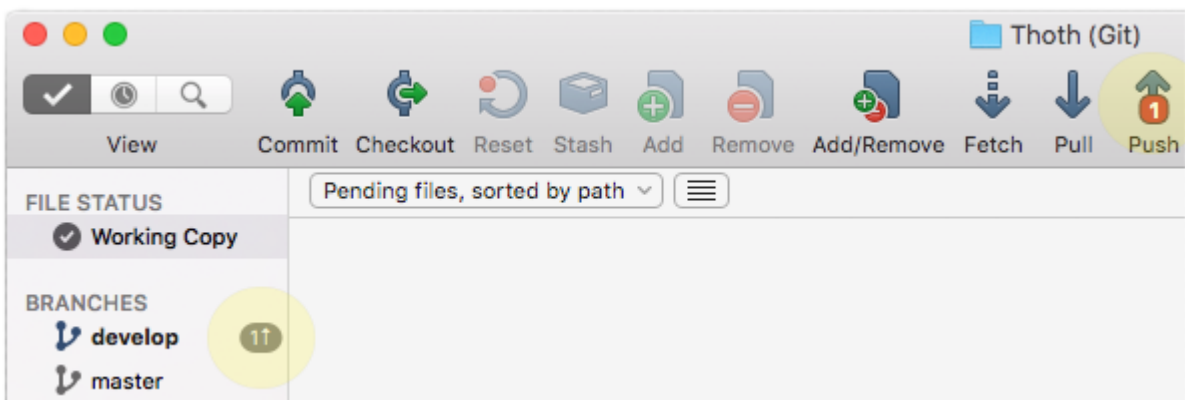


11.1.5 Ready to start writing!

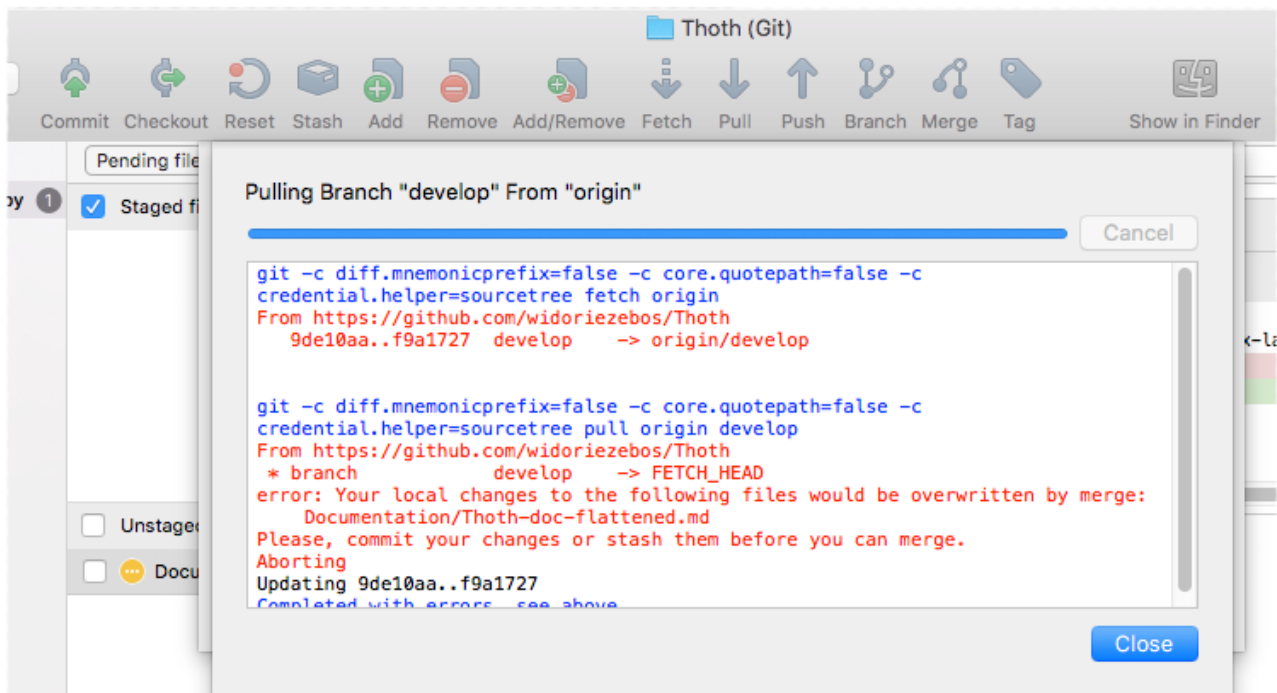
You have now your new branch active in your local repository and you can start creating and editing files (all located inside the repository folder, you remember where you put it right?). After creating files they will show up in SourceTree in the 'Working copy' section. Anything in the working copy is a local change that has not yet been committed to the local repository, so it is just a change in a file and nothing else. To add them to the (local) version control you will have to commit the changes. To do this check all the files listed in the Working Copy section, add a comment for your commit (good practice) and click 'Commit'.



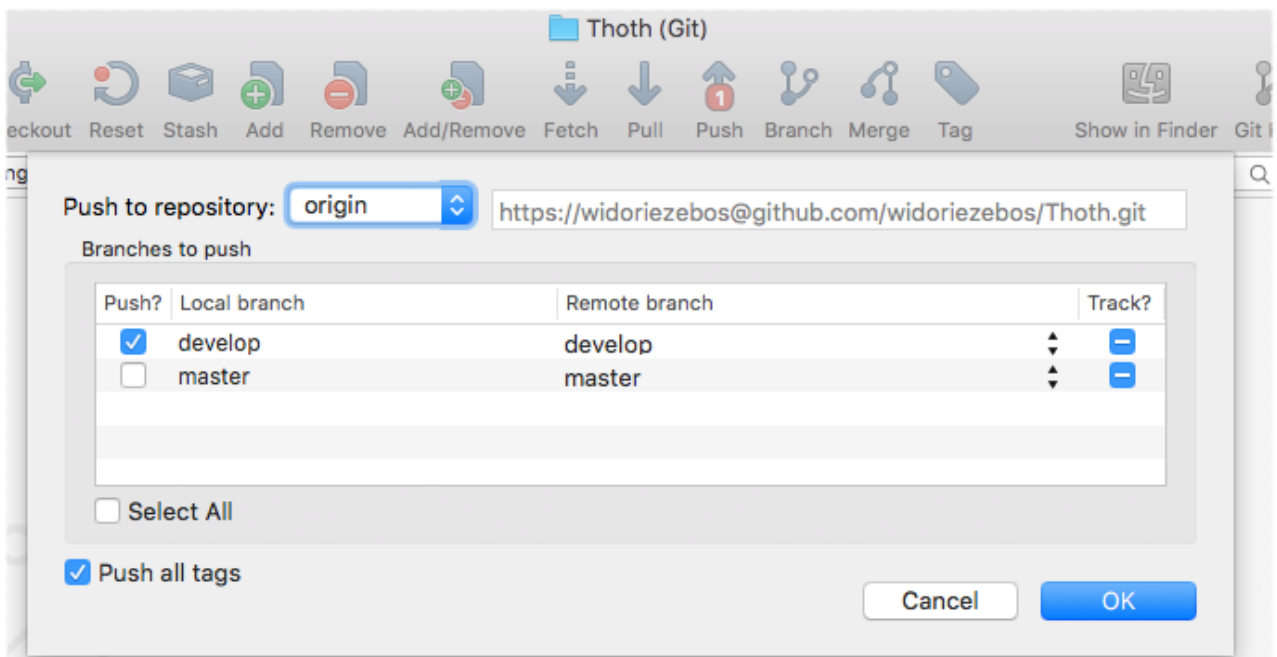
After you have committed your changes they are stored in the local repository and their state is therefore set to unchanged (as compared to what is stored in the local repository). The changes have not left your local machine however, this will only happen when you Push. Your mantra here is 'Commit-Pull-Push'. You should always Pull before you Push. Git needs you to do this to make sure that any changes a colleague has pushed to the repository does not conflict with anything you are about to push.



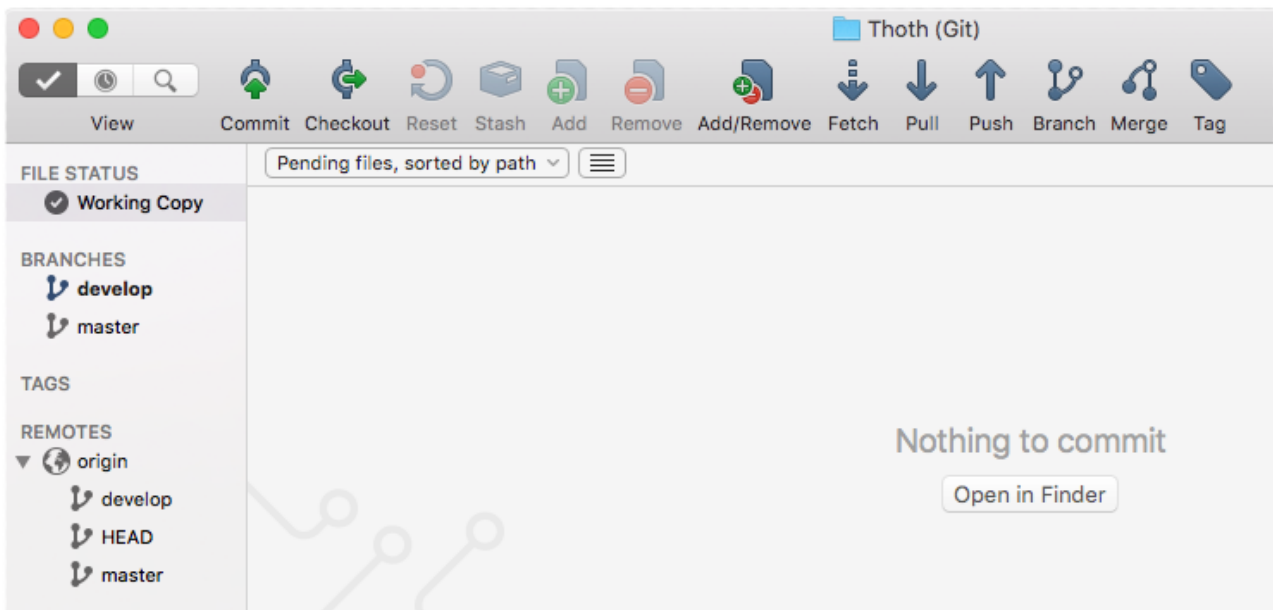
Sidestep: If you forget to Pull before you commit, you might get an ugly error message (if you read carefully you will find 'Please, commit your changes or stash them before you can merge'. To recover from this just close the dialog, commit your changes and then Pull again.



Now click Pull and then Push to push your changes to the central repository. (Note the read badge with the number '1' which shows you the number of local commits that are ready to get pushed)



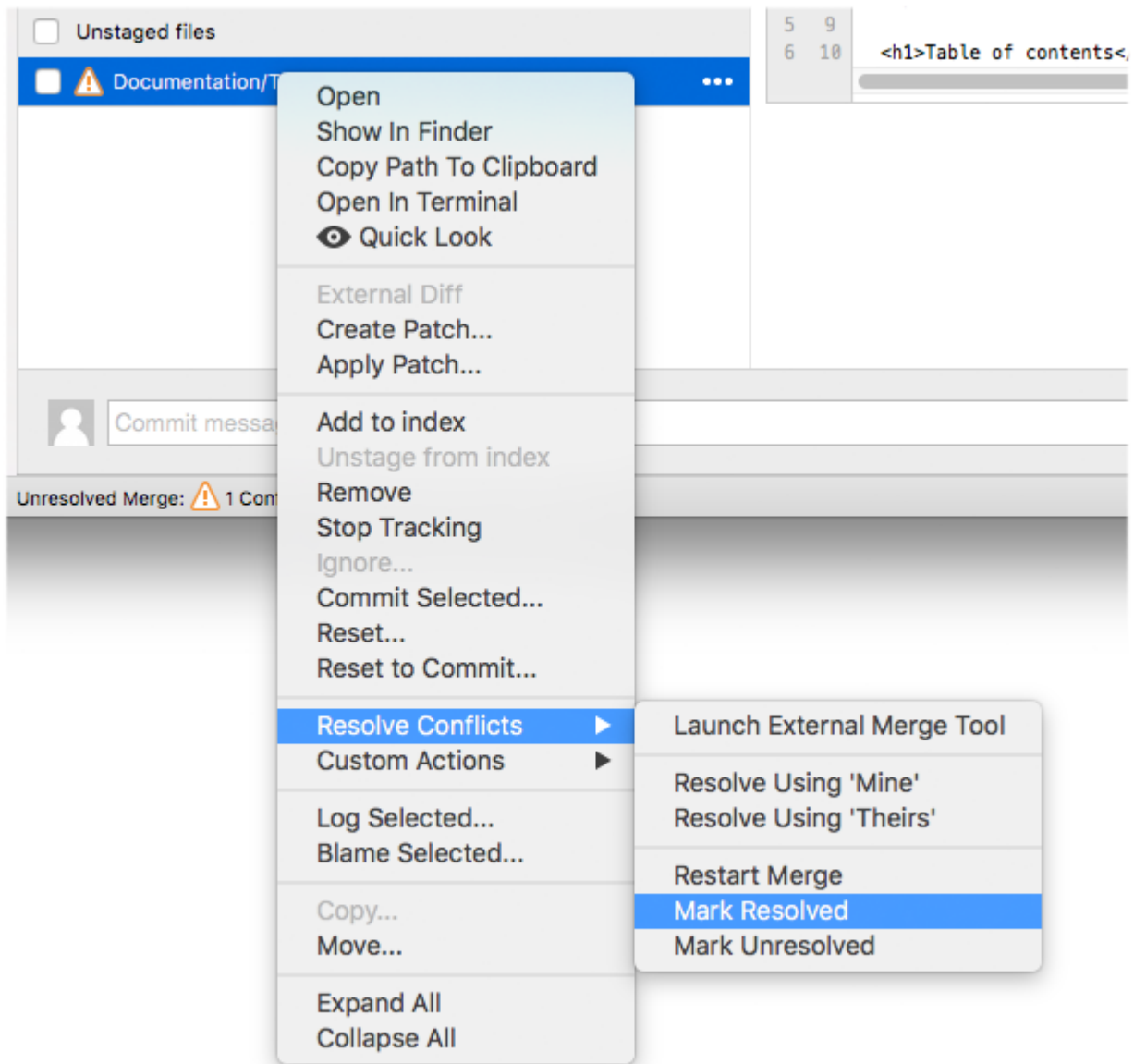
After 'pull/pushing' your Working Copy will show empty again, all your changes are now available to any colleagues sharing the central repository with you (by Pulling in your changes).



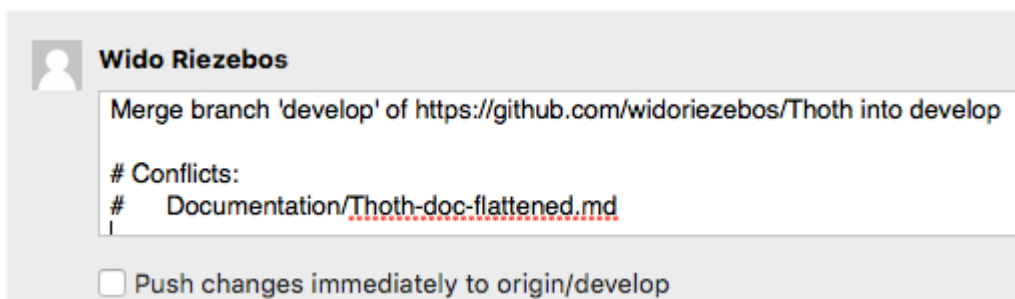
Now there might be a complication when a colleague did indeed push a change since the latest Pull you did, and that this change conflicts with a change of you. A conflict means that there is a change on the same line in the same file, and Git cannot automatically merge both changes. This so called 'Merge Conflict' will have to be resolved by you then.



If you have conflicts you head over to the Working Copy and find out what files have a conflict. There are several ways to resolve a conflict; the easiest one is overwriting either your own or your colleagues' changes. That might be easy but is usually not the way forward because you might overwrite other changes in the file as well. You will either resolve the conflicts with your 'External Merge Tool' (on OSX that would be FileMerge), handpick the segments in SourceTree or you will simply head to your editor and find the conflicts in the source. After making sure everything is in order save the file and switch back to SourceTree. Now right click on the conflicted file and select 'Mark Resolved'.



Once every conflict has been resolved you can finally commit all your changes. You will notice that SourceTree enters a comment automatically about the merge and the conflicts



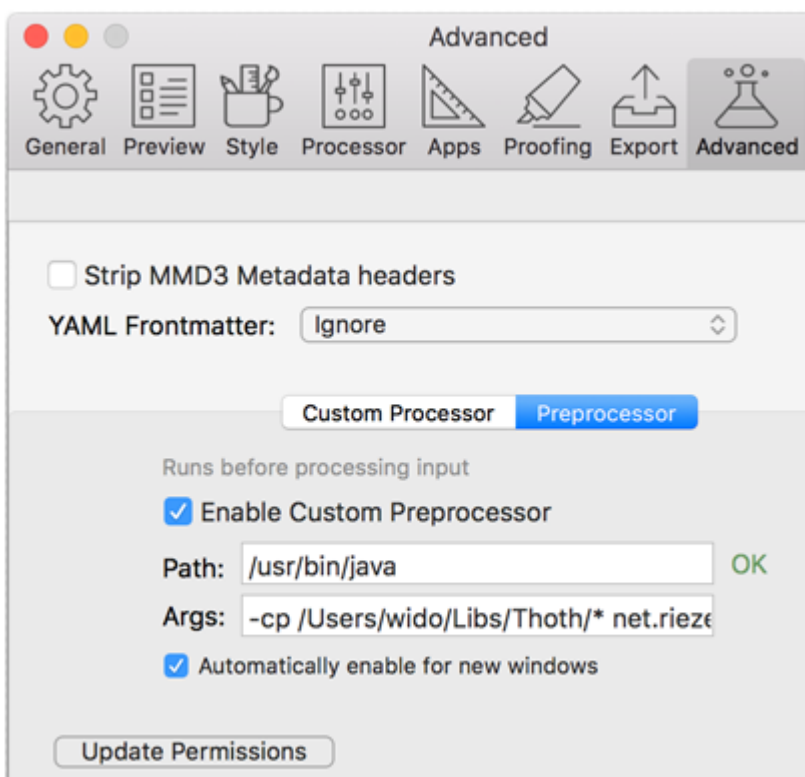
Now Pull again to make sure nobody Pushed in the mean time and then Push yourself.

11.2 WYSIWYG

Although writing in Markdown is all about *not* focussing on how your document is rendered it is sometimes good to have a preview. Depending on the editor you would have some WYSIWYG functionality in place, but it will most certainly not do all that Thoth can do for you. To get around this there are a couple of options.

11.2.1 Use the Thoth IncludeProcessor as a pre processor

For OSX there is the excellent tool called Marked2 that supports a custom pre processor. By making use of this feature you can actually run any Thoth specific functionality (includes, link processing etc) right before it is transformed into HTML. To give you a sense of what a pre processor's role here could be just have a look at the screenshot below. Any editor that supports a pre processor can be set up this way.



The command line that is obscured in the screenshot should be similar to

```
-cp /Users/wido/Libs/thoth-lib.jar  
net.riezebos.thoth.markdown.IncludeProcessor
```

11.2.2 Use Thoth standalone with your browser

You can easily run Thoth as a standalone server on your laptop, using a File System based repository that points straight into your Git documentation repository. Read the section about configuring and installing Thoth on how to do this. Basically you just connect your browser to the locally running Thoth that will show you exactly what your rendered document looks like. Just save your document, switch to your browser and press the refresh-key. To do this; open a terminal and enter:

```
java -jar thoth-standalone-1.0.0.jar /path/to/my/config.properties
```

```
Thoth is firing up. Please hang on...
Setting up content managers...
Thoth server started.
You can now access Thoth at http://localhost:8080
Just enter 'stop' to stop the server
```

11.2.3 Use an editor that does WYSIWYG

If you use an editor that supports WYSIWYG and supports a pre processor as described above then you are set to go. If the editor does not support a pre processor however there will be certain features (include processing being the obvious one) that will not show up in the WYSIWYG (therefore breaking the WYG part). The WYSIWYG will then maybe helpful but it will not show you the whole story.

11.3 Skinning Documents and Screens

Thoth comes equipped with built-in skins that enable you to change the appearance of screens and documents. Basically a Skin is a collection of templates, CSS and any other resource files required to render an HTML page in a certain way. By default the Skin named SimpleSkin will be chosen to render all screens and documents in Thoth. In the configuration you can change the default Skin, but it is also possible to influence the selection of the Skin in other ways

11.3.1 Overriding the Skin dynamically

You can force Thoth to use a skin named 'MySkin' by adding a request parameter

```
skin=MySkin
```

An example forcing the (builtin) Skin named Bootstrap would be

```
http://localhost:8080/skin=Bootstrap
```

11.3.2 Overriding the Skin based on the path of a document (automatically)

In the root of your library you (should) have a plain text document called skins.properties and in this file you associate a Skin with a path. An example of this is

```
*/Datamodel/*=DataModelSkin
*=SimpleSkin
```

In this example any path that matches */Datamodel/* will be rendered using the DataModelSkin. As a fallback any other path will be associated with SimpleSkin. Matching is done from top to bottom and the first match wins.

11.3.3 Creating your own Skin

Of course you can use the builtin Skins (SimpleSkin and Bootstrap) but you can define as

many as you like yourself. You do this by creating a Skin descriptor file and placing this file anywhere in your library. Thoth will find all your Skin descriptors and enable them for use within your library.

Note: if you only intend to change the header or footer of pages rendered by the SimpleSkin template, you just need to change the `header.tpl` or `footer.tpl` templates that are included by all pages (except the markdown template that is).

Creating the Skin descriptor file

The Skin descriptor file is a property file with the name `skin.properties`. Inside this file you must at least have one property that defines the name of the Skin. If you leave out everything else you have effectively defined a new Skin that inherits everything from SimpleSkin. So the minimal skin.properties file looks like

```
# The name of the skin. Used for referencing this skin in the configuration
name=MyOwnSkin
```

Of course it does not make any sense to do this because now there is no difference compared to the SimpleSkin, so let's have a look at a better example:

```
# The name of the skin. Used for referencing this skin in the configuration
name=MyOwnSkin

# From which this Skin inherits anything not defined by this Skin itself
inheritsfrom=

# The velocity template that is used for the Main index (main page)
template.index=index.tpl

# The velocity template that is used for the Context index
template.contextindex=contextindex.tpl

# The velocity template that is used for the login page
template.login=login.tpl

# Template for Markdown rendered pages
template.html=html.tpl

# Template for Diff pages
# Supports keyword replacement for ${title} and ${body}
# for the body of the page
template.diff=diff.tpl

# The velocity template that is used for the meta information page
template.meta=meta.tpl

# The velocity template that is used for the revision information page
template.revisions=revisions.tpl

# The velocity template that is used for the revision information page
template.validationreport=validationreport.tpl

# The velocity template that is used for the revision information page
template.search=search.tpl
```

```
# The velocity template that is used for browsing
template.browse=browse.tpl

# The velocity template that is used for the user management page
template.usermanagement=usermanagement.tpl

# The velocity template that is used for the user profile
template.userprofile=userprofile.tpl

# The velocity template that is used to display any error
# that occurred during execution
template.error=error.tpl
```

Note the *inheritsfrom* setting. If you leave it empty then you inherit from SimpleSkin. If you put a name of another (custom) Skin there, you will inherit from that Skin. This enables you to inherit from a base Skin without having to copy anything that you did not change.

The templates specified in the above example are Velocity templates that render the various pages and documents for Thoth. The paths of the templates are relative to the skin.properties file, it is good practice to place the files directly where the skin.properties file is located, and put all of them in a folder with the name of the Skin.

By changing these templates you can completely alter the way Thoth looks and feels. See the Skin Developers Manual for more information on how to create your own templates.

Referencing Skin resources from your template

You can place any number of resources (css, images etc) inside the folder of your skin and then reference them from your template. To create fully qualified links there is the `${skinbase}` variable to help you. See the example at the end of this chapter for more information (look for `href="${skinbase}..."`)

JSON support

Any command in Thoth supports JSON as an output format; so every variable described below can be returned as part of a JSON encoded result as well (bypassing the template mechanism entirely). This can come in handy when your Skin uses AJAX to render a Single Page Interface for instance. If you want a command to return JSON then add a request parameter ' `mode=json` ' to your request.

Creating your own templates

Templates are written in [Velocity](#) and can use variables provided by Thoth to include specific content. Depending on the command being rendered there will be different variables available (although some are generic).

Generic variables for all templates

The following variables are defined for all templates

- **servername** the name of the server as specified by the configuration. Type: String
- **context** is the name of the context. Type: String
- **skinbase** is the absolute path to the root of the active Skin's resources. Type: String
- **contexturl** is absolute path to the current context. If Thoth is not located in the root of the web container then this path includes the path into the Thoth web context. Type: String
- **path** is the path part of the request inside the context (excluding context URL parts). Type: String
- **title** is the title of the current page or document. Type: String
- **skin** is the name of the active skin. Type: String
- **today** is the current date. Formatted according to the format mask specified by the Configuration. Type: String
- **now** is the current timestamp. Formatted according to the format mask specified by the Configuration. Type: String
- **refresh** is the timestamp of the latest successful refresh (for version control based repositories). Formatted according to the format mask specified by the Configuration. Type: String
- **outputFormats** is the list of supported output formats (html, raw and any custom renderers that are defined in the configuration) Type: List<String>
- **identity** the identifier of the user or group that is currently logged in
- **userfullname** the full name (first, last) of the user that is currently logged in
- **permissions** contains a list of permissions the current user is granted. Valid permissions: ACCESS, READ_BOOKS, READ_FRAGMENTS, READ_RESOURCE, BROWSE, DIFF, META, PULL, REINDEX, REVISION, SEARCH, VALIDATE. Type: Set<String>

Types used for variables

Most types are basic / simple types like String, List and Integer. The exception to the rule are beans Book, Diff etc. The following beans are defined as types and used below:

Book

String **name**
String **path**
String **folder**
String **title**
Map<String, String> **metaTags**

Diff

Operation **operation** (One of INSERT, DELETE or EQUAL)

String **text**

MarkdownDocument

Map<String, String> **metatags**
List<ProcessorError> **errors**
DocumentNode **documentStructure**
String **markdown**
Date **lastModified**

DocumentNode

String **path**
String **description**
String **fileName**
String **folder**
int **includePosition**
int **level**
List<DocumentNode> **children**

Commit

String **id**
String **author**
Date **timestamp**
String **message**
String **shortMessage**
List<Revision> **revisions**

ProcessorError

LineInfo **currentLineInfo**
String **errorMessage**

LineInfo

String **file**
int **line**

SearchResult

String **document**
List<Fragment> **fragments**
List<DocumentNode> **bookReferences**
boolean **isResource**

boolean **isImage**
int **indexNumber**

Fragment

String **text**

ContentNode

String **path**
boolean **isFolder**
Date **dateModified**
long **size**
List<ContentNode> **children**

User

String **identifier**
String **firstname**
String **lastname**
Date **blockedUntil**
List<String> **permissions**
List<String> **memberOf**

Group

String **identifier**
List<String> **permissions**
List<String> **memberOf**

template.index

The main index template is a special one, because the template itself belongs to a specific repository (context) by it is used for the main index that is not specific to a certain context. The following variables are available:

- **contexts** is the list of contexts that are available. Type: String
- **problems** not empty when problems occurred during initializing a context. Type: List<String>

template.contextindex

The index of the context; this is where you would list all the books that are present in

- **classifications** contains the list of names for the classifications specified in the configuration. For every classification name there will be list of books that are marked to be part of that classification using meta data (i.e. classification:

<name>) Type: List<String>

- **classification_<classification>** contains the list of books per classification as defined in the configuration (see the setting context.classifications under [File Classification](#)). Type: List<Book>
- **books** contains a list of all books sorted by path Type: List<Book>
- **versioncontrolled** is a boolean that is true when the underlying repository supports version control. Type: boolean

template.login

The template that provides the login page.

- **message** any error message that applies when login failed. Type: String
- **loggedin** flag to indicate whether currently logged in. Type: boolean

template.html

The template used to render HTML pages.

- **body** is the actual (rendered) contents of the document. In the case of HTML this will be the HTML without any header (just what you would place inside the <body> tag). Type: String

template.diff

The template used to display the revisions of a document (fragment).

- **author** is the author of the revision; based on information from the version control system. Type: String
- **timestamp** is the timestamp of the committed revision. Formatted according to the format mask specified by the Configuration. Type: String
- **commitMessage** is the message that the author entered for the commit. Type: String
- **diffs** is the list of Diff's (revisions) in time. Type: List<Diff>

template.meta

The template used to display meta information of a document.

- **document** is the document concerned. Type: MarkdownDocument
- **usedBy** is the list of document (fragment) paths that directly reference or include the document concerned. Type: List<String>
- **usedByIndirect** is the list of document (fragment) paths that directly OR indirectly reference or include the document concerned. Type: List<String>
- **documentNodes** is the list of DocumentNodes that make up the document. Type: List<DocumentNode>
- **commitMap** is the list of commits per document path. Type: Map<String, List<Commit>>
- **commitList** is the list of commits for the document concerned. Type: List<Commit>

- **metatagKeys** is the list of meta tag names (keys) for the document concerned. Type: List<String>
- **metatags** is the map of for metatag key-value pairs. Use metatagKeys to iterate over the map. Type: Map<String, String>
- **errors** is the list of errors that occurred during (include) processing of the document concerned. Type: List<ProcessorError>
- **versioncontrolled** returns true if the underlying repository supports version control. Type: Boolean

template.revisions

The template that is used to display all the revisions of changes committed to the (version controlled) repository.

- **commitList** is the list of commits. Type: List<Commit>
- **page** is the current page. Note that paging is supported by adding a request parameter called 'page' with the value of the page to be displayed. Type: Integer
- **hasmore** is a boolean that will be true if there are results to be found on the next page. Type: Boolean

template.validationreport

The page that shows all errors that were detected by Thoth in the current repository.

- **documents** is the list of documents (paths) with errors. Type: List<String>
- **errors** is the list of errors that occurred during (include) processing of the document concerned. Type: List<ProcessorError>
- **errorsByDocument** is a map of errors by document. The key of the map is the value found in **documents**. Type: Map<String, ProcessorError>

template.search

The search (engine) page, displaying the results of a query.

- **query** is the query that was entered. Type: String
- **searchResults** contains the search results. Type: List<SearchResult>
- **errorMessage** contains any error message in the case of a search query problem (syntax problem while parsing the Lucene query). Type: String
- **page** is the current page. Note that paging is supported by adding a request parameter called 'page' with the value of the page to be displayed. Type: Integer
- **hasmore** is a boolean that will be true if there are results to be found on the next page. Type: Boolean

template.browse

The template used for browsing the contents of the entire repository.

- **contentNodes** is the list of content nodes for the current folder (path). Type: List<ContentNode>
- **atRoot** is a boolean that will be true if the current page is representing the root

of the repository

template.usermanagement

The template used for user management.

- **users** the list of users. Type: List<User>
- **groups** the list of groups. Type: List<Group>

template.userprofile

The template used for editing the user profile.

- **user** the user currently logged in. Type: User

template.error

The template used to display Thoth system errors.

- **message** is the (technical) error message. Type: String
- **stack** is the full stack trace of the error occurring in Thoth. Type: String

template.log

The template used to display logs (Pull and Reindex commands)

- **title** title of the log. Type: String
- **log** the log to display. Type: String

Template helper functions

There is a special variable called *thothutil* that you can use to access all the functions defined by the ThothUtil helper class. The following functions are available:

String escapeHtml(String html)

Escapes the characters in a String using HTML entities.

String formatTimestamp(Date date)

Formats a Timestamp (a Date actually) according to the configured format mask for timestamps

String formatDate(Date date)

Formats a Date according to the configured format mask for dates.

String getVersion()

Returns the current version of Thoth

String encodeUrl(String url)

Translates a string into application/x-www-form-urlencoded format using a specific encoding scheme. This method uses the supplied encoding scheme to obtain the

bytes for unsafe characters.

String getCanonicalPath(String path)

Returns the canonical path (translating relative constructs like '..' and '.')

String getNameOnly(String imagePath)

Returns just the name (stripping the path and prefix) of a filename

String getFileName(String filespec)

Returns just the filename (stripping any directories from the path)

String getFolder(String filespec)

Returns the folder part of a fully specified filename.

String stripSuffix(String value, String suffix)

Strips the given suffix from the value string (if present)

String getPartBeforeFirst(String value, String prefix)

Returns the part of the string value before the first prefix. If not found the entire string value

String getPartBeforeLast(String value, String prefix)

Returns the part of the string value before the last prefix. If not found the entire string value

String getPartAfterFirst(String value, String prefix)

Returns the part of the string value after the first prefix. If not found the entire string value

String getPartAfterLast(String value, String prefix)

Returns the part of the string value after the last prefix. If not found the entire string value

String getExtension(String path)

Returns the extension part of a given filename (with or without the path)

String prefix(String value, String prefix)

Prefixes value with the given string. If the prefix is already there nothing is added

String suffix(String value, String prefix)

Suffixes value with the given string. If the prefix is already there nothing is added

Template example

The template below is a simple example; it is actually the template used by SimpleSkin to render the main index page. It should give you a feel of what a template looks like though. Especially note the *#foreach*, *#{thothutil.function()}* and *#{variable}* constructs

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="utf-8">
```

```

<title>Documentation Index</title>
<link rel="icon" href="{skinbase}/Webresources/favicon.png"
      type="image/png" />
<link rel="shortcut icon" href="{skinbase}/Webresources/favicon.png"
      type="image/png" />
<link rel="stylesheet" type="text/css"
      href="{skinbase}/Webresources/style.css">
</head>
<body>
  <h3>Please select one of the contexts below</h3>

  #foreach($context in $contexts)
    <a href="{context}">{context}</a><br/>
  #end

  <br/><br/>
  Latest successfull Pull request was at {refresh}
  <br/>
  To update the site to the latest version right now; you can
  <a href="?.cmd=pull">Pull manually</a>
  or click <a href="?.cmd=reindex">here</a> to force a reindex<br/>
  <br/>
  <sub>Powered by Thoth core version {thothutil.getVersion()}</sub>
</body>
</html>

```

12 Installing Thoth

There are two ways you can run Thoth. For server environments there is a WAR that you can deploy in a Web Container like tomcat and for desktop environments there is a standalone version that requires almost no setup.

12.1 Getting hold of the software

You can find the software on Github: <https://github.com/widoriezebos/Thoth>

12.2 Configuration

Create a configuration.properties file as described in the [Configuration](#) section.

12.3 Running standalone

For desktop environments you can run Thoth without a Web Container using the Thoth-standalone version which has an embedded web server. First make sure you have Java8 installed, so [grab a JRE or JDK version 8](#) and install it. Then you can start 'standalone Thoth' if you either have an environment or JVM variable set (as describe above) or simply provide the location of the configuration.properties file as an argument. Not even any of this is required if you have the configuration.properties file in the working folder when starting the Thoth standalone

```
java -jar thoth-standalone-1.0.0.jar
```

By default the Thoth-standalone server is running in interactive mode (listening for commands on stdin). If you want to suppress the interactive mode and run Thoth-standalone purely as a (background process) server then add the `-server` flag:

```
java -jar thoth-standalone-1.0.0.jar -server
```

12.4 WAR installation

Have your Web Container and JDK8 ready and then just drop the Thoth WAR inside the web apps folder of your Web Container. And then the only requirement is that the configuration can be found through an environment variable (or -D provided JVM argument) called `thoth_configuration`. On servers with an init script for Tomcat (/etc/init.d/tomcat) you could add a script line stating

```
export thoth_configuration=/opt/conf/configuration.properties
```

13 The configuration file

In this section you will find the settings Thoth supports. The configuration of Thoth is placed in a property file which is read during startup. Without the minimal configuration (which is about where to store working files and where to find content) Thoth will not be able to start

13.1 Required settings

***workspace*location**

States where the working files of Thoth will be created. This is where Thoth will checkout branches and create search indexes. In principle it is completely safe to delete the entire contents of the workspace (after shutting down Thoth). When Thoth is launched it will automatically recreate the contents of the workspace, including any embedded database. But NOTE: If you delete the 'thoth-database' folder in your workspace you will *also have deleted the embedded Thoth database*. Although the embedded database will be recreated automatically on startup, you will have lost all your users and groups as a result.

13.1.1 defaultgroup

: When not logged in; this group defines the (default) permissions. By default the following groups are defined by Thoth: thoth_administrators, thoth_writers, thoth_readers, thoth_anonymous. Administrators can do anything in Thoth, writers can not pull nor reindex nor do user management but basically anything else. Readers can only access books and anonymous has only access to the main index page (so they can login from there)

13.1.2 Repository settings

Repositories are the source for content. In the case of a Git repository Thoth will pull a branch as the source of the content. In the case of a FileSystem repository nothing will be pulled; the specified location for the repository is then used as-is.

repository.1.name

The uniquely identifying name of a repository. A repository is currently a Git or a local folder; future versions of Thoth might support additional types of repositories. The name is used as a reference when you specify a context (see below). Note the numeric index in the name (1) which enumerates the repository definition. You can define as many repositories as long as you number them sequentially leaving no gaps. Thoth will stop at the first repository.*n*.name that does not have it's value set.

repository.1.type

The type of repository. Choose one of **git**, **filesystem**, **zip**. You can use the alias 'fs' for filesystem and 'jar' for zip.

repository.1.location

The location of the repository. For a Git repository this will be the URL, for a FileSystem repository it is the (absolute) folder name on the filesystem and for a Zip it is the location of the zip (or jar) file. Note that for zips you can suffix the path with '!/pathinzip' to offset the contents of the zip. So if your zip contains a root folder called Documentation (and you want that folder to serve as root inside the zip) you would have something like `location=/path/to/myzip.zip!/Documentation`

repository.1.username

The username for logging in to the repository. In the case of a FS or Zip repository this can be left blank.

repository.1.password

The password for logging in to the repository. In the case of a FS or Zip repository this can be left blank.

13.1.3 Database settings

Thoth uses a database to store it's users and groups (among others). If you use the defaults 'as-is' then the database will be a Derby embedded one that needs no additional setup (will be automatically created on startup, default location will be a subfolder 'thoth-database' in your workspace. If you require a different database, or if you want your derby embedded database files located outside your workspace, then you need to set the properties below. Also note that you need to add the appropriate JDBC driver jar on the classpath. The tables will be automatically created (and filled with initial values) regardless the type of database.

database.type

The type of database (driver) to use. Supported are 'embedded', 'derby', 'oracle' and 'postgresql'. Unsupported but available are also 'h2', 'hsqldb' and 'mysql'. The default is 'embedded'.

database.url

The JDBC url for the database. If you use the embedded database this can be just a folder relative to your workspace or an absolute folder if you want the database files to be stored outside your workspace. In all other cases the JDBC url depends on the driver chosen. For Postgresql you would have something like `jdbc:postgresql://localhost:5432/thoth`

database.user

The username to connect with to the database. Can leave empty for embedded.

database.password

The password to connect with to the database. Can leave empty for embedded.

13.1.4 Context settings

A context is the root of a library, and corresponds to a branch in Git. For FileSystem repositories there is not branch, but the name of the context is used in Thoth to refer to the contents. Note the numeric index in the name (1) which enumerates the context definition. You can define as many contexts as long as you number them sequentially

leaving no gaps. Thoth will stop at the first context.*n*.name that does not have it's value set.

context.1.name

The uniquely identifying name of a context. This name will be part of any path to a document inside the context; and the main index page of Thoth by default displays a list of known contexts.

context.1.repository

The reference (by name) to the repository where the contents of this context resides.

context.1.branch

The name of the branch to pull from Git. In the case of FS repository this can be left blank.

context.1.library

Optional. If you do not have your documentation (your library) located at the root of the repository, you can specify the location of your library as a relative path from the root of your repository. For example, in the case of the Thoth documentation the library is set to 'Documentation'

context.1.refreshseconds

The number of seconds between automatic repository refreshes. When a change is detected an indexer updating the search index is automatically launched. Default = 60

13.2 Optional settings

defaultuser

The name of the default user. Any user of Thoth will fallback to this user since there is no user management yet. In a future version of Thoth there will be support for users, groups and permissions. Right now the only way to restrict access to Thoth is to set the value of this property to one of "reader", "writer", "administrator". Readers can only read books and search books. Writers have full access except for administrative functions like Pull and Reindex. Note: the default value is currently "administrator"; this property will no doubt be removed in a next release of Thoth.

config.autorefresh

When true the configuration will automatically (hot) reload if changes to it are detected (based on comparing file modification timestamps)

config.autorefreshinterval

The interval (in seconds) to check whether if the configuration has changed.

skin.default

The name of the default skin. When no skin is defined for a specific context this is the skin that will be used. Also used when no skin is defined for the main index page. Default value is 'SimpleSkin'

skin.mainindexcontext

Use the skin associated with the given context for the main index page. If left blank

the `skin.default` property will be used as a fallback. Default value is `<no value>`.
Note that if you set this property to a context that is not publicly available you will need to specify access rules that allow public access to (just) your skin(s).

localhost

Custom rendering uses a forked process to render the contents. When this process needs to fetch the contents; this URL is used as the base for the path of the document. Default value is `'http://localhost:8080/'`

13.2.1 Custom renderers

You can define any number of custom renderers. The most basic one would be a PDF or an EPUB renderer.

renderer.1.extension

The extension to recognize this renderer. An example would be `'pdf'`, adding a parameter to a Thoth page URL `'?output=pdf'` would then render the HTML version of that page with this renderer. Must be unique across renderers. Example: `'pdf'`

renderer.1.source

The type of input to produce for the custom renderer. Default is HTML but you can override this to RAW (or any other renderer for that matter, just use it's Extension as the source attribute). You obviously need to avoid setting up recursive loops by pointing to a custom renderer that uses this renderer as a source somewhere in it's chain.

renderer.1.contenttype

The mime contenttype of the content rendered by this custom renderer. For PDF that would be `'application/pdf'`

renderer.1.command

The OS level command to render the output. The following keywords can be used for substitution in the command:

- `input`, the file that contains the source document
- `url`, the URL where to fetch the contents of the source document from. Note that this is an alternative method of specifying the input (without using the input file directly). The URL will contain a token parameter (`ssotoken`) for single sign on, so if you append any parameters to the URL you should use a `'&'` and not a `'?'`. (See the example below, `renderer.1.command`)
- `output`, the name of the file to write the rendered contents to
- `context`, the name of the context
- `path`, the path of the file in the context
- `title`, the title of the document

A few examples of custom renderers (note the use of `${url}` and `${input}`):

```
renderer.1.extension=pdf
renderer.1.source=html
renderer.1.contenttype=application/pdf
renderer.1.command=/usr/local/bin/prince ${url}&suppresserrors=true -o
```

```

${output} --javascript --media=print --page-size=A4 --page-margin=20mm

renderer.2.extension=docx
renderer.2.source=html
renderer.2.contenttype=application/
vnd.openxmlformats-officedocument.wordprocessingml.document
renderer.2.command=/usr/local/bin/pandoc -s -r html -t docx ${input} -o
${output}

renderer.3.extension=epub
renderer.3.source=html
renderer.3.contenttype=application/epub+zip
renderer.3.command=/usr/local/bin/pandoc -s -r html -t epub ${input} -o
${output}

renderer.4.extension=rtf
renderer.4.source=html
renderer.4.contenttype=application/rtf
renderer.4.command=/usr/local/bin/pandoc -s -r html -t rtf ${input} -o
${output}

```

Note: on Windows the command for Prince is different:

```

renderer.1.command=/Program Files/Prince/engine/bin/prince.exe
${url}&suppresserrors=true -o ${output} --javascript --media=print
--page-size=A4 --page-margin=20mm

```

And for Pandoc it will be like (note the user name in the path):

```

renderer.2.command=C:/Users/Wido/AppData/Local/Pandoc/pandoc.exe -s -r html
-t docx ${input} -o ${output} --variable=geometry:a4paper

```

13.2.2 Using Pandoc as the HTML rendering engine

Note that it is possible to replace the default Markdown rendering engine (based on PegDown) with any other. Just define a custom renderer with the extension 'html' to override the built in HTML renderer and make sure you use 'raw' as the source input like so:

```

renderer.1.extension=html
renderer.1.contenttype=text/html
renderer.1.source=raw
renderer.1.command=/usr/local/bin/pandoc -s -r markdown -t html ${input} -o
${output}

```

13.2.3 Format masks

formatmask.timestamp

The format mask to use for timestamps. Note that this is a Java based format mask (i.e. MM is for month, mm for minutes. Check [SimpleDateFormatter](#) for more information about this mask). Default value = dd-MM-yyyy HH:mm:ss

formatmask.date

The format mask to use for dates (without the time part). Note that this is a Java based format mask (i.e. MM is for month, mm for minutes. Check [SimpleDateFormat](#) for more information about this mask). Default value = dd-MM-yyyy

13.2.4 File classification

context.classifications

Comma separated list of classifications. The default branch index page uses the classifications of the documents to create an initial structure. There must be meta tags in the documents matching the classification (i.e. category: primers) for a classification named 'category'. Default value = category,audience,folder

documents

The extensions of the files that will be recognized as a 'Document' and therefore can be rendered to html, pdf etc. Default = marked,book,index,md

books

The extensions of the files that will be recognized as a 'Book' and therefore shown by the auto generated index. Default = marked,book,index

index.extensions

The extensions of file to include in the search index (comma separated; no '.'). Default = md,book,marked,txt

images.extensions

Image recognition. Set (comma separated) the extensions (without the '.') below to determine whether a matching resource will be treated as an image. Default = png,jpeg,jpg,gif,tiff,bmp

13.2.5 Markdown processing options

markdown.newlineheaders

Will add a newline before every header in the source. Avoids some parser issues where strictness is involved. Default = on

markdown.appenderrors

Append any link / include error messages at the bottom of the document. Default = on

markdown.option.SMARTS

Pretty ellipses, dashes and apostrophes. Default = on

markdown.option.QUOTES

Pretty single and double quotes. Default = on

markdown.option.ABBREVIATIONS

PHP Markdown Extra style abbreviations. See <http://michelf.com/projects/php-markdown/extra/#abbr>. Default = on

markdown.option.HARDWRAPS

Enables the parsing of hard wraps as HTML linebreaks. Similar to what github does. See <http://github.github.com/github-flavored-markdown>. Default = on

markdown.option.AUTOLINKS

Enables plain autolinks the way github flavoured markdown implements them. With this extension enabled pegdown will intelligently recognize URLs and email addresses without any further delimiters and mark them as the respective link type. See <http://github.github.com/github-flavored-markdown>. Default = off

markdown.option.TABLES

Table support similar to what Multimarkdown offers. See http://fletcherpenney.net/multimarkdown/users_guide/. Default = on

markdown.option.DEFINITIONS

PHP Markdown Extra style definition lists. Additionally supports the small extension proposed in the article referenced below. See <http://michelf.com/projects/php-markdown/extra/#def-list> and <http://www.justatheory.com/computers/markup/modest-markdown-proposal.html> Default = on

markdown.option.FENCED_CODE_BLOCKS

PHP Markdown Extra style fenced code blocks. See <http://michelf.com/projects/php-markdown/extra/#fenced-code-blocks>. Default = on

markdown.option.WIKILINKS

Support Wiki Style links. See <http://github.github.com/github-flavored-markdown>. Default = on

markdown.option.STRIKETHROUGH

Support strikethroughs as supported in Pandoc and Github. Default = on

markdown.option.ANCHORLINKS

Enables anchor links in headers. Note that Thoth has it's own method of creating anchor links that might interfere with this setting. Default = off

markdown.option.SUPPRESS_HTML_BLOCKS

Suppresses HTML blocks. They will be accepted in the input but not be contained in the output. Default = off

markdown.option.SUPPRESS_INLINE_HTML

Suppresses inline HTML tags. They will be accepted in the input but not be contained in the output. Default = off

markdown.option.ATXHEADERSPACE

Requires a space char after Atx # header prefixes, so that #dasdsdaf is not a header. Default = off

markdown.option.FORCELISTITEMPARA

Force List and Definition Paragraph wrapping if it includes more than just a single paragraph. Default = off

markdown.option.RELAXEDHRULES

Force List and Definition Paragraph wrapping if it includes more than just a single paragraph: Default = off

markdown.option.TASKLISTITEMS

GitHub style task list items. Default = on

markdown.option.EXTANCHORLINKS

Generate anchor links for headers using complete contents of the header. Note that Thoth has it's own method of creating anchor links that might interfere with this setting. Default is off.

13.2.6 Miscellaneous

search.maxresults

The number of search results to show per page of results

versioncontrol.maxfilerevisions

The maximum number of revisions to retrieve for the metadata page. Default = 10

versioncontrol.maxcontextrevisions

The maximum number of revisions to collect / display for latest commits for the entire context (Revisions command). Default = 25

json.prettyprint

When a page is rendered to JSON (by adding the mode=json parameter to a request) it will be pretty printed if this setting is set to 'true'. Default = true

parsetimeout

The maximum time in ms that a Markdown parse by the PegDown parser can last. Default = 4000

13.3 Sample configuration file

```
#####
# Primary options below; these you will have to set to get things going
#####

# The absolute path to the folder that will contain pulled branches
# (repositories) and search indexes
# The workspace is to be used by Thoth alone, never edit content in there
# yourself because that might lead to merge problems when Thoth synchronizes
# your repositories # It is always safe to delete the entire contents of
# the workspace (if Thoth is stopped) because
# Thoth will recreate everything on startup if required.
workspacelocation=/path/to/your/thoth/workspace

# When not logged in; this group defines the (default) permissions. By default
# the following groups are defined by Thoth: thoth_administrators,
# thoth_writers, thoth_readers and thoth_anonymous
defaultgroup=thoth_readers

#####
```

```

# Repository and Context definitions. These specify where to get the
# content from. Can be as many repositories as you like. Make sure you
# number them sequentially and leave no gaps (parsing will stop at the
# first entry that has it's name not set)
#####

# Name (identifier for use by Thoth) of the repository
repository.1.name=Repository1

# Type of version control. Currently only GIT and FS (simple filesystem)
# are supported
repository.1.type=git

# Location (URL or folder) of the repository that contains the
# Documentation
repository.1.location=https://github.com/someuser/MyDocumentation.git

# Username of the repository user
repository.1.username=yourusername

# Password of the repository user
repository.1.password=yourpassword

# The contexts to check out / pull from the repository.

# The name of the context. Will be used as name for the local repository
# folder. Must be unique
context.1.name=Context1

#The name of the branch to check out
context.1.repository=Repository1

# The name of the branch to check out. Must be an existing branch in the
# associated repository
context.1.branch=Branch1

# Optional. If you do not have your documentation (your library) located
# at the root of the repository, you can specify the location of your
# library as a relative path from the root of your repository.
context.1.library=

#####
# Skinning related properties
#####

# The name of the default skin to use. If not set; it will use the builtin
# skin named 'Builtin'. Note that any skin can come from the classpath as
# long as the package remains within net/riezebos/thoth/skins/
skin.default=SimpleSkin

# The skin to use for the main index page (hence not within a context).
# Must be a valid context name as specified by the 'context.name' property.
# If left blank then the default skin is used.
skin.mainindexcontext=

# The URL for the localhost. Will be used for custom renderer processing
localhost=http://localhost:8080/

#####

```

```

# Define any custom renderers below for for additional output formats.
# Make sure you number them sequentially and leave no gaps (parsing will
# stop at the first custom renderer that has it's extension not set.
# You can have any number of custom processors; just keep on numbering them.
#
# Note that you can also override the default html and raw renderers.
# Their extensions are 'raw' and 'html'
#####

renderer.1.extension=pdf
renderer.1.contenttype=application/pdf
renderer.1.source=html
renderer.1.command=/usr/local/bin/prince ${url}&suppresserrors=true -o
${output} --javascript --media=print --page-size=A4 --page-margin=20mm

renderer.2.extension=docx
renderer.2.contenttype=application/
vnd.openxmlformats-officedocument.wordprocessingml.document
renderer.2.source=html
renderer.2.command=/usr/local/bin/pandoc -s -r html -t docx ${input} -o
${output}

renderer.3.extension=epub
renderer.3.contenttype=application/epub+zip
renderer.3.source=html
renderer.3.command=/usr/local/bin/pandoc -s -r html -t epub ${input} -o
${output}

renderer.4.extension=rtf
renderer.4.contenttype=application/rtf
renderer.4.source=html
renderer.4.command=/usr/local/bin/pandoc -s -r html -t rtf ${input} -o
${output}

renderer.5.extension=
renderer.5.contenttype=
renderer.5.source=
renderer.5.command=

#####
# Database options. Simplest is just using the embedded database.
# For any other type of database: make sure you have added the JDBC driver jar
# to the class path of Thoth. (Derby embedded is built in; so no need for
that)
#####

# The type of database to use. Currently supported: embedded, derby,
# oracle and postgres. Not supported but might work: h2, hsqldb and mysql
database.type=embedded

# The JDBC url. For embedded this is a path (either absolute or relative to
# the workspace). When left empty (for embedded) the default folder will be
# the folder 'thoth-database' in the workspace.
database.url=

# Postgresql example:

#database.type=postgres
#database.url=jdbc:postgresql://localhost:5432/thoth
#database.user=thoth

```



```

#database.password=thoth

#####
# Embedded server options. Not applicable when deployed as a WAR
#####

# The port to have the (embedded) server listen to. Default is 8080
embedded.port=8080

# The idle timeout for connections. Note: in specified in seconds
embedded.idletimeout=30

# Hostname of the server
embedded.servername=localhost

#####
# Context index (page) options
#####

# The following comma separated classifications will be available to the
# context index template (grouping of documents by classification) The
# folder classification is built in; and is just listed below for clarity
# In the template (or json) the classification names are available under
# "classification_" + <the name specified below>
context.classifications=category,audience,folder

#####
# Search options
#####

# The number of search results per page; default is 25
search.maxresults=25

#####
# More options below; but you might not necessarily have to change them
#####

# Default timestamp format mask. Note that the month is MM and the minutes
# are mm. HH is 24hr and hh 12hr
formatmask.timestamp=dd-MM-yyyy HH:mm:ss

# Default date format mask. Note that the month is MM and the minutes are
# mm.
formatmask.date=dd-MM-yyyy

# Pretty print JSON responses. You might want to set this to false in a
# production environment; small performance benefit
json.prettyprint=true

# The maximum number of revisions to collect / display for latest commits
# per file (Meta command)
versioncontrol.maxfilerevisions=10

# The maximum number of revisions to collect / display for latest commits
# for the entire context (Revisions command)
versioncontrol.maxcontextrevisions=25

```

```

# Automatic refresh interval (seconds). Minimum is 30 seconds. Set to 0
# for disable
versioncontrol.autorefresh=60

# The maximum time in ms that a Markdown parse can last
parsetimeout=4000

# The extensions of the files that will be recognized as a 'Document' and
# therefore can be rendered to html, pdf etc
documents=marked,book,index,md

# The extensions of the files that will be recognized as a 'Book' and
# therefore shown by the auto generated index
books=marked,book,index

# The extensions of file to include in the search index (comma separated;
# no '.')
index.extensions=md,book,marked,txt

# Append any link / include error messages at the bottom of the document
markdown.appenderrors=true

# Auto number headings up to the specified level. Default is 3, set to 0
# to disable
markdown.maxheadernumberlevel=3

#####
# MARKDOWN OPTIONS BELOW
# See https://github.com/sirthias/pegdown for a full description
#####

#Add a newline before any #header
markdown.newlineheaders=on

# Pretty ellipses, dashes and apostrophes.
markdown.option.SMARTS=on

# Pretty single and double quotes.
markdown.option.QUOTES=on

# PHP Markdown Extra style abbreviations.
# See http://michelf.com/projects/php-markdown/extra/#abbr
markdown.option.ABBREVIATIONS=on

# Enables the parsing of hard wraps as HTML linebreaks. Similar to what
# github does.
# See http://github.github.com/github-flavored-markdown
markdown.option.HARDWRAPS=on

# Enables plain autolinks the way github flavoured markdown implements them.
# With this extension enabled pegdown will intelligently recognize URLs
# and email addresses without any further delimiters and mark them as the
# respective link type.
# See http://github.github.com/github-flavored-markdown
markdown.option.AUTOLINKS=off

#Table support similar to what Multimarkdown offers.
#See http://fletcherpenney.net/multimarkdown/users_guide/
markdown.option.TABLES=on

```

```

# PHP Markdown Extra style definition lists.
# Additionally supports the small extension proposed in the article
# referenced below.
# See http://michelf.com/projects/php-markdown/extra/#def-list
# See http://www.justatheory.com/computers/markup/modest-markdown-proposal.html
markdown.option.DEFINITIONS=on

# PHP Markdown Extra style fenced code blocks.
# See http://michelf.com/projects/php-markdown/extra/#fenced-code-blocks
markdown.option.FENCED_CODE_BLOCKS=on

# Support [[Wiki-style links]]. URL rendering is performed by the active
# {@link LinkRenderer}.
# See http://github.github.com/github-flavored-markdown
markdown.option.WIKILINKS=on

# Support ~strikethroughs~ as supported in Pandoc and Github.
markdown.option.STRIKETHROUGH=on

# Enables anchor links in headers.
markdown.option.ANCHORLINKS=off

# Suppresses HTML blocks. They will be accepted in the input but not be
# contained in the output.
markdown.option.SUPPRESS_HTML_BLOCKS=off

# Suppresses inline HTML tags. They will be accepted in the input but not
# be contained in the output.
markdown.option.SUPPRESS_INLINE_HTML=off

# Requires a space char after Atx # header prefixes, so that #dasdsdaf is
# not a header.
markdown.option.ATXHEADERSPACE=off

# Force List and Definition Paragraph wrapping if it includes more than
# just a single paragraph
markdown.option.FORCELISTITEMPARA=off

# Force List and Definition Paragraph wrapping if it includes more than
# just a single paragraph
markdown.option.RELAXEDHRULES=off

# GitHub style task list items: - [ ] and - [x]
markdown.option.TASKLISTITEMS=on

# Generate anchor links for headers using complete contents of the header
markdown.option.EXTANCHORLINKS=off

```

14 User Management

Thoth comes equipped with built-in user management. Users can be member of (one or more) groups and groups are granted permissions. Permissions determine what functionality is available to a user. By default there is a user 'administrator' with password 'Welcome2Thoth!' who is a member of group 'administrators'. Note that there is a special case for the 'administrator' user: regardless the group administrator is in; all permissions are granted. For any other user the granted permissions depend on the group(s) that the user is a member of.

14.1 Default groups

By default there will be three groups (corresponding permissions in brackets):

thoth_administrators(ACCESS, READ_BOOKS, READ_FRAGMENTS, READ_RESOURCE, BROWSE, DIFF, META, PULL, REINDEX, REVISION, SEARCH, VALIDATE, MANAGE_USERS, MANAGE_CONTEXTS)

thoth_writers(ACCESS, READ_BOOKS, READ_FRAGMENTS, READ_RESOURCE, BROWSE, DIFF, META, REVISION, SEARCH, VALIDATE)

thoth_readers(ACCESS, READ_BOOKS, READ_RESOURCE, SEARCH)

thoth_anonymous()

14.2 Permissions

BASIC_ACCESS

Access to the Index and ContextIndex page. Without this Permission users first need to log in before they can see anything (not even the available contexts). The only exception is that when there are specific access rules that grant access to a certain path to everybody (i.e. /some/path RequireNone)

READ_BOOKS

Access the contents of Books. What a book actually is depends on the configuration, by default it would be any document with the '.book' extension.

READ_FRAGMENTS

Access the contents of fragments (that make up the books)

READ_RESOURCE

Access anything that is neither book nor fragment. This is basically anything in the repository that is not a piece of text (i.e. images, property files, CSS etc)

BROWSE

Browse the entire contents of a context (clicking through folders, clicking on files to access them)

DIFF

Compare revisions of files

META

Access meta information (revisions, structure, usage) of a file

PULL

Manually Pull the contents of a (remote) repository.

REINDEX

Force a reindex of contexts to occur (manually)

REVISION

Access the revision history of a file

SEARCH

Search the entire context using the search engine

VALIDATE

Access the validation information page

MANAGE_USERS

Being able to administer (create, update, delete) groups, users, memberships and permissions

MANAGE_CONTEXTS

Being able to administer (create, update, delete) repositories and contexts

14.3 Access control

Apart from the permissions that drive access to certain content (READ_BOOKS, READ_FRAGMENTS) you can also implement a finer grained access control for the content itself. If you want that you will have to create a file called `access.rules` in the root of your library. In this file you can then place 'require group' statements for path patterns. You can restrict access to files to certain groups based on matching the path of the files. The syntax of a rule in `access.rules` is

```
<path spec> RequireNone
<path spec> Require <groups>
<path spec> RequireAll <groups>
<path spec> RequireAny <groups>
```

where `<path spec>` is a path to a (set of) file(s) using '*' as a wild card.
and `<groups>` is a comma separated list of group names that a user has to be member of (has to be member of all of them) to be able to access the file matching the `<path spec>`. `Require/RequireAll` specifies that the users needs to be member of *all* groups, whereas `RequireAny` relaxes this to match just one (**any**) of the groups specified. With `RequireNone` you can grant *public access* to a certain path. You might want to use this if you use a Skin that is defined in a Context as the Main index skin. (Check configuration property `skin.mainindexcontext`)

Rules are matched from top to bottom; if no rule matches access will be denied.

Example of `access.rules` :

```
/Library/Datamodel/* require modelers  
* requireAny thoth_readers, thoth_writers
```

In the example above part anything in the (sub folders) of /Library/Datamodel will be restricted to users that are member of the 'modelers' group.

Everything else requires users to be member of either the group 'thoth_readers' or 'thoth_writers'. **Note:** without the last line nobody would be able to access anything except the modelers who could access books and fragments under /Library/Datamodel/

The Context Index page make use of these access rules as well to determine whether a user has access to certain books. The main index page determines whether a user has access to a context by checking whether the '/' path is accessible to the user (by applying the rules)

15 Technicalities

Thoth is built on the Java platform and is OS independent. Thoth runs on Linux, OSX and Windows with minimal hardware requirements.

15.1 Minimum System Requirements

The following are considered minimum requirements

- Supported OS: Linux, OSX and Windows
- 4 GB of available RAM
- Disk space depending on the size of your library, say 10 GB
- Java 8 platform
- Apache Tomcat 7

15.2 Acknowledgements

Thoth would not have been possible without the great help of the following projects (in arbitrary order):

- Markdown, <https://daringfireball.net/projects/markdown/>
- Pegdown Markdown parser, <https://github.com/sirthias/pegdown>
- JGit, <http://www.eclipse.org/jgit/>
- Diff-match-patch, <https://code.google.com/archive/p/google-diff-match-patch/>
- Apache Lucene, <https://lucene.apache.org/core/>
- And of course many many other open source projects. Check the pom.xml for a complete list.