Wiser Documentation

Release 0.1

the Nile team

November 27, 2016

The documents in this section (will) provide generic information about the technical documentation generator (**Sphinx_**), the version control system (**Git_**), the project management system and issue tracking system (TaskMan *aka* Redmine) and the helpdesk and support ticket system (OTRS).

It may also contain miscellaneous sections on **How-to...** do things in other tools that we may need to use (e.g. Forum).

It's a work-in-progress and it always will...

CHAPTER 1

Table of Contents

1.1 Using Forum or Projects

1.1.1 Introduction

This section contains tips on how to best use:

- Forum (http://forum.eionet.europa.eu/)
- Projects (http://projects.eionet.europa.eu/)

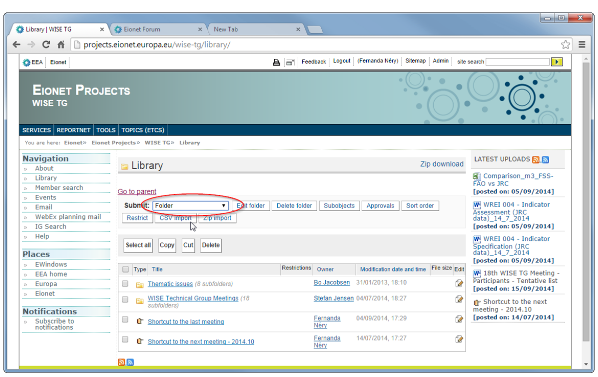
Warning: Don't panic!
Help is available in http://forum.eionet.europa.eu/help

1.1.2 How to...

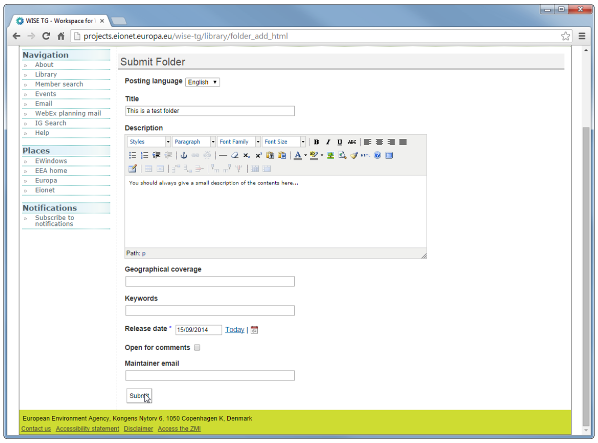
Create a folder

First, read http://forum.eionet.europa.eu/help/content_folder ... Then...

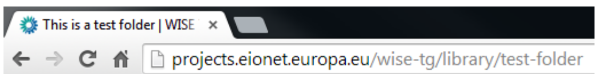
1. Go the library folder
2. Next to the **Submit:** label, select `Folder` in the droplist.



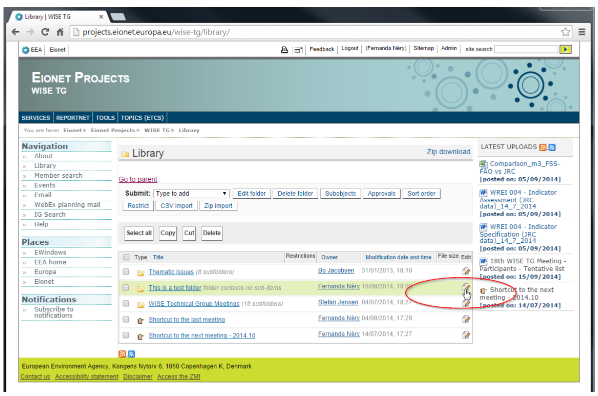
3. Enter the folder's `Title` and `Description`, then press `Submit`



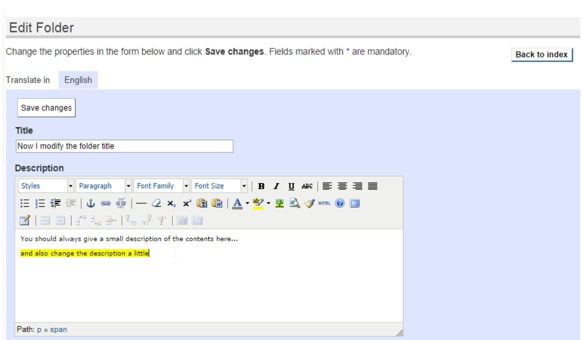
4. Notice the browser's tab title and the URL:



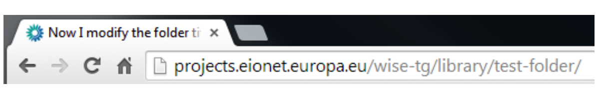
5. You can now edit the `Title` and `Description`, if you want:



6. ...and change the title and description.



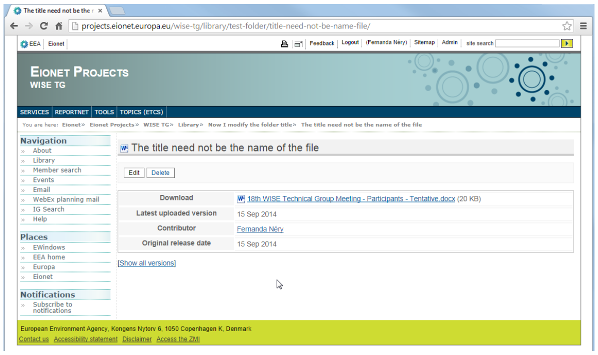
7. Notice that the browser's tab title changed, **but the URL is stable**:



Upload a file (and keep older versions)

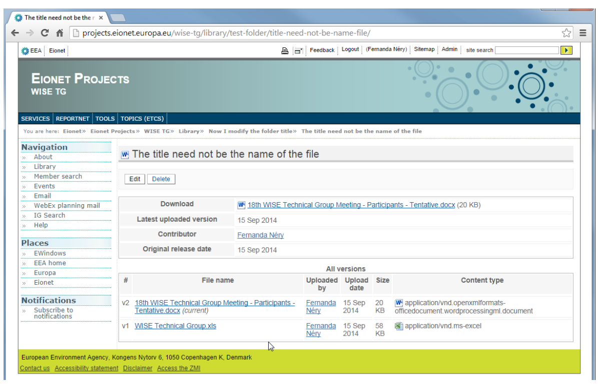
First, read http://forum.eionet.europa.eu/help/content_file ... Then...

1. Go the library folder
2. Next to the **Submit:** label, select `File` in the droplist.
3. Enter the file's `Title`, the `Description` and press `Choose file` to select a file in your computer. Then press `Submit`.
4. Notice that the `Title` need not be the name of the file:

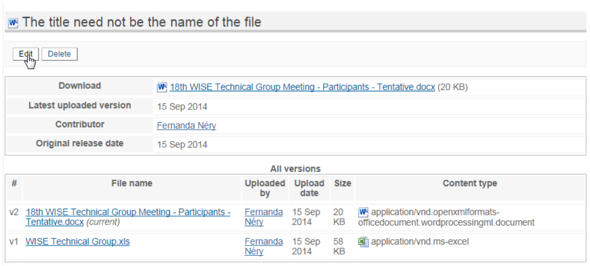


5. Also, notice the URL that s automatically created:

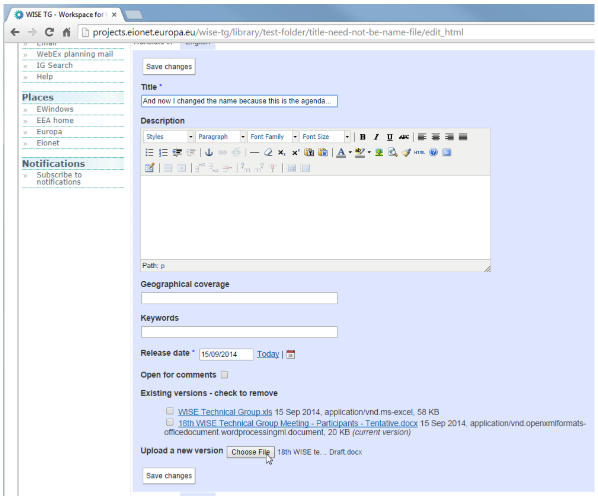
6. You can press `Edit` and choose another file, if you made a mistake or want to add an updated version of the file:



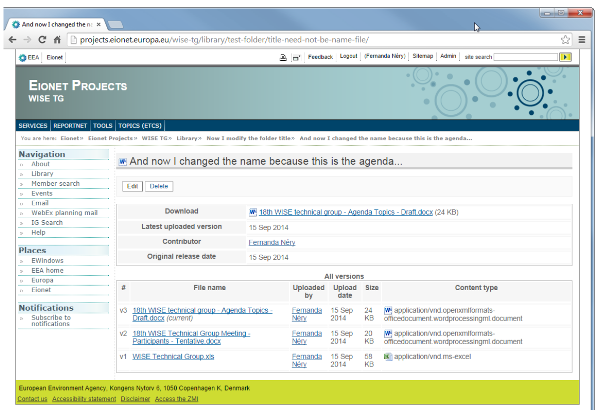
7. And `Edit` again (notice the two previous versions):



8. Now I modified the `Title` and added yet another version of the document...

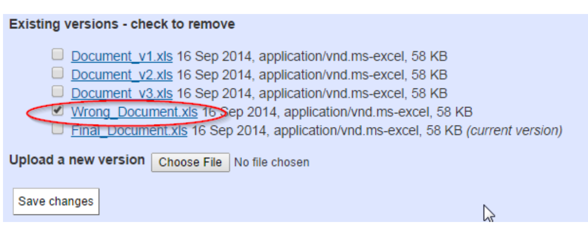


9. Notice that the URL is **always** kept stable:

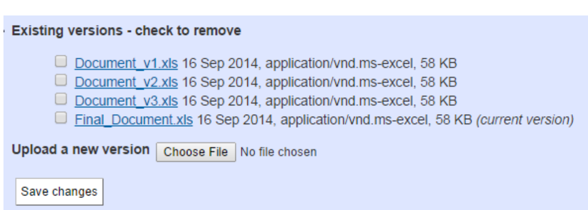


To remove a previous version

1. Press `Edit` to edit to file.
2. Mark the versions you want to remove:

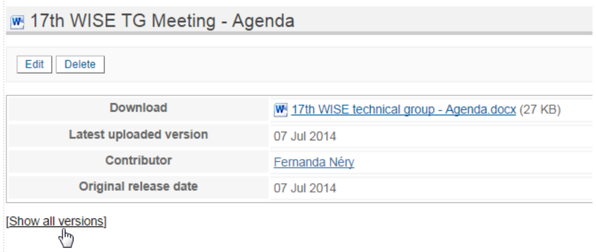


3. Press `Save changes`.
4. The wrong document is gone...

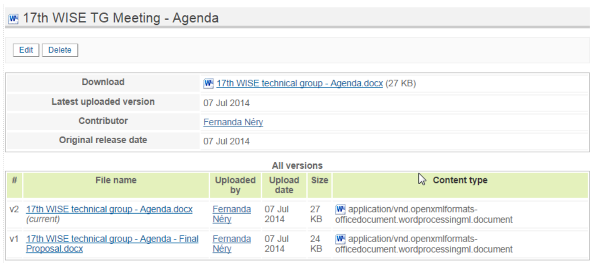


See the previous versions of a file

1. Just press `Show all versions`



2. and there it is:



1.2 Using the technical documentation generator

1.2.1 Introduction

The purpose of this section is to provide ‘quick’ reference information (on syntax, available tools, style conventions, etc) that may be required by technical writers or translators working with the project documentation.

The most important information is about reStructuredText, the markup language used in the technical documentation (user manual, etc.).

There is also some (basic) information about Sphinx, the application that automatically converts the text files and generates the documentation in HTML format or PDF format.

For example, this HTML page was originally written in reStructuredText. Please, press the *Source* link in the menu to see the original text and markup.

1.2.2 References

Source

This document is an abridged and modified version of *Sphinx’s reStructuredText Primer*.

Sphinx's reStructuredText primer

Introduction

A reStructuredText document is simply a plain text file with some markup to specify the format or the semantics of the text.

There are two types of markup:

- inline markup: for example, `*the surrounding asterisks would mark this text as italics*`, like *this*.
- explicit markup: is used for text that need special handling, such as footnotes, tables, or generic directives. Explicit markup blocks always start with `..` followed by whitespace.

Paragraphs

The paragraph is the basic block in a reST document.

Paragraphs are simply chunks of text separated by one or more blank lines.

Indentation is significant in reST, so all lines of the same paragraph must be left-aligned to the same level of indentation.

This is a style convention.

Try to keep each line with a maximum of 78 characters. Remember that changing to next line does not create a paragraph, unless the chunks of text is separated by a blank line.

Try to keep each phrase in a different line. It improves readability and facilitates the translation process.

Remember that consecutive blank lines will be ignored in the HTML output.

Quoted paragraphs Quoted paragraphs are created by just indenting them more than the surrounding paragraphs:

```
Normal paragraph.

   Indented paragraph.
```

This is a style convention.

Each indentation level is created with 3 whitespaces. Do not use tabs.

Line breaks Line blocks are a way of preserving line breaks (the equivalent of using `Shift+Enter` to break a line in Microsoft Word or LibreOffice Writer):

```
| These lines are
| broken exactly like in
| the source file.
```

Sections

Section are created by underlining (and optionally overlining) the section title with a punctuation character:

```
This is a heading
=================
```

Any punctuation character can be used to define a section title. The underlining (and overlining) must be at least as long as the text itself. Sections must be properly nested.

This is a style convention.

Use the following punctuation characters in the section titles:

- `#` for Parts
- `*` for Chapters
- `=` for sections ("Heading 1")
- `-` for subsections ("Heading 2")
- `^` for subsubsections ("Heading 3")
- `"` for paragraphs ("Heading 4")

Please note that, when converting to `HTML` format, sections are automatically converted to an appropriate heading tag (for example: `<h2>Heading text</h2>`).

When converting to `ODT` or `DOCX`, an appropriate Heading style is applied.

Inline markup

Bold, italics, monospace The markup is quite simple:

- use one asterisk for italics: `*text*` (the equivalent of using `Ctrl+i` in Microsoft Word or LibreOffice Writer),
- use two asterisks for strong emphasis (boldface): `**text**`
- use backquotes for text literals: ` ``text`` `

Be aware of some restrictions:

- The markup may not be nested. For example, this markup is wrong: `*italics with **bold** inside*`
- The text content within the markup may not start or end with whitespace. For example, this markup is wrong: `* text*`
- The markup must be separated from surrounding text by non-word characters (whitespace or punctuation). Use a backslash-escaped-space to work around that. For example: `thisis\ *one*\ word` is rendered like thisis*one*word.
- If asterisks or backquotes appear in running text and could be confused with inline markup delimiters, they have to be escaped with a backslash.

Subscript and superscript Subscript is marked with `:sub:`subscript text``. Superscript is marked with `:sup:`superscript text``.

This is a tip.

Whitespace or punctuation is required around interpreted text, but often not desired with subscripts & superscripts. Backslash-escaped whitespace can be used; the whitespace will be removed from the processed document:

```
The chemical formula for molecular oxygen is O\ :sub:`2`.
```

To improve the readability of the text, the use backslash-escapes is discouraged. If possible, use *Substitutions* instead:

```
The chemical formula for pure water is |H2O|.

.. |H2O| replace:: H\ :sub:`2`\ O
```

Keep all substitutions together (e.g. at the end of the file).

Lists

Bulleted lists List markup is natural: just place an asterisk at the start of a paragraph and indent properly:

```
*  This is a bulleted list.
*  It has two items, the second
   item uses two lines.
```

Nested lists are possible, but be aware that they must be separated from the parent list items by blank lines:

```
* this is
* a list

  * with a nested list
  * and some sub-items

* and here the parent list continues
```

Numbered lists The same goes for numbered lists; they can also be auto-numbered using a # sign:

```
1. This is a numbered list.
2. It has two items too.

#. This is a numbered list.
#. It has two items too.
```

Definition lists Definition lists are created as follows:

```
term (up to a line of text)
   Definition of the term, which must be indented

   and can even consist of multiple paragraphs

next term
   Description.
```

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

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

See the following example:

```
.. glossary::

   environment
      A structure where information about all documents under the root is
      saved, and used for cross-referencing.  The environment is pickled
      after the parsing stage, so that successive runs only need to read
      and parse new and changed documents.

   source directory
      The directory which, including its subdirectories, contains all
      source files for one Sphinx project.
```

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

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

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

```
.. glossary::

   term 1
   term 2
      Definition of both terms.
```

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

To automatically sort a glossary, include the following flag:

```
.. glossary::
   :sorted:
```

Field lists Field lists are two-column table-like structures resembling database records (label & data pairs). For example:

```
:Date: 2001-08-16
:Version: 1
:Authors: - Me
          - Myself
          - I
:Indentation: Since the field marker may be quite long, the second
   and subsequent lines of the field body do not have to line up
   with the first line, but they must be indented relative to the
   field name marker, and they must line up with each other.
:Parameter i: integer
```

This is a style convention.

In this project, field lists are used to include metadata in each document.

The basic **'Dublin Core'_** metadata fields should be included in the very beginning of each document (like a header to the text file), like this:

```
.. metadata-placeholder

:DC.Title:
   Document title
:DC.Creator:
   Author
:DC.Date:
   Date yyyy-mm-dd
:DC.Description:
   Abstract
:DC.Language:
   en
:DC.Format:
   text/x-rst
:DC.Rights:
   Access rights
:DC.RightsHolder:
   Copyright.
```

Note that these metadata field names are not automatically recognised by the Sphinx parser, so the text itself will not be visible in the HTML pages (for example). The metadata fields are the equivalent to the *Document properties* fields in a `DOCX` file or an `ODT` file.

docutils recognises a number of Bibliographic Fields (such as `docinfo`, `author`, `authors`, `organization`, `contact`, `version`, `status`, `date`, `copyright`, `field`, `topic`).

This is an advanced topic

Some metadata filed are recognised by Sphinx. For example:

- `:tocdepth:` indicates the maximum number of levels in the Sphinx sidebar table of contents for the file.
- `:orphan:` indicates that, even if the file is not included in any `.. toctree::` directive, no warning should be produced by Sphinx.

Tables

The reStructuredText markup supports two basic types of tables. For *grid tables*, you have to "paint" the cell grid yourself. They look like this:

```
+------------------------+------------+----------+----------+
| Header row, column 1   | Header 2   | Header 3 | Header 4 |
| (header rows optional) |            |          |          |
+========================+============+==========+==========+
| body row 1, column 1   | column 2   | column 3 | column 4 |
+------------------------+------------+----------+----------+
| body row 2             | ...        | ...      |          |
+------------------------+------------+----------+----------+
```

Simple tables are easier to write, but limited: they must contain more than one row, and the first column cannot contain multiple lines. They look like this:

```
=====  =====  =======
A      B      A and B
=====  =====  =======
False  False  False
```

```
True   False  False
False  True   False
True   True   True
=====  =====  =======
```

This is a tip.

These are the basic types of tables, which are rather clumsy. Also available (and easier to use) are *special tables*, namely list-tables and CSV-tables.

An **exceltable_** extension can also be used with Sphinx, which allows the inclusion of `XLS` spreadsheets, or part of them, into a reST document.

Hyperlinks

External links Use `` `link text <http://example.com/>`_ `` for inline web links. If the link text should be the web address, you don't need special markup at all, the parser finds links and mail addresses in ordinary text (with no markup).

You can also separate the link and the target definition, like this:

```
This is a paragraph that contains `a link`_.

.. _a link: http://example.com/
```

This is a tip.

The use of inline web links is discouraged, to improve the readability of the reST text.

Simple links (e.g. to institutional sites, software sites, and so on) should be kept together at the end of the text file (this is merely a way to simplify the editing procedure, and the update and verification of the links).

Internal links To support cross-referencing to arbitrary locations in any document, the standard reST labels are used. For this to work, the label names must be unique throughout the entire documentation. There are two ways in which you can refer to labels:

- If you place a label directly before a section title, you can reference to it with `` :ref:`label-name` ``. Example:

```
.. _my-label-ref:

Section to cross-reference
--------------------------

This is the text of the section.

In the end of this phrase is a reference to the section title, see :ref:`my-label-ref`.
```

  The `:ref:` role would then generate a link to the section, with the link title being "Section to cross-reference". This works just as well when section and reference are in different source files.

  Automatic labels also work with *figures*:

```
.. _my-figure-ref:

.. figure:: my-image.png

   My figure caption
```

A reference like `:ref:`my-figure-ref`` would insert a reference to the figure with link text "My figure caption".

The same works for *tables* that are given an explicit caption using the `table` directive.

- Labels that aren't placed before a section title can still be referenced to, but you must provide the text for the link, using this syntax: `:ref:`Link text <label-name>``.

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

Source Code

Literal code blocks are introduced by ending a paragraph with the special marker `::`. The literal block must be indented (and, like all paragraphs, separated from the surrounding ones by blank lines):

```
This is a normal text paragraph. The next paragraph is a code sample::

   It is not processed in any way, except
   that the indentation is removed.

   It can span multiple lines.

This is a normal text paragraph again.
```

The handling of the `::` marker is smart:

- If it occurs as a paragraph of its own, that paragraph is completely left out of the document.
- If it is preceded by whitespace, the marker is removed.
- If it is preceded by non-whitespace, the marker is replaced by a single colon.

That way, the second sentence in the above example's first paragraph would be rendered as "The next paragraph is a code sample:".

Explicit Markup

Explicit markup is used in reStructuredText for most constructs that need special handling, such as footnotes, specially-highlighted paragraphs, comments, and generic directives.

An explicit markup block begins with a line starting with `..` followed by whitespace and is terminated by the next paragraph at the same level of indentation. (There needs to be a blank line between explicit markup and normal paragraphs. This may all sound a bit complicated, but it is intuitive enough when you write it.)

Directives A directive is a generic block of explicit markup.

The directive content follows after a blank line and is indented relative to the directive start.

Basically, a directive consists of a name, arguments, options and content.

Look at this example:

```
.. contents:: This is my Table of Contents
   :depth: 2
```

The directive starts with `..` followed by one whitespace. The name of the directive is `contents` (it creates a table of contents). This directive takes one argument: the table of contents' title ("This is my Table of Contents"). The option `depth` specifies the number of section levels that are collected in the table of contents.

Options are given in the lines immediately following the arguments and are indicated by the colons. Options must be indented to the same level as the directive content.

Docutils supports the following directives:

- Admonitions: `attention`, `caution`, `danger`, `error`, `hint`, `important`, `note`, `tip`, `warning` and the generic `admonition`. (Most themes style only `note` and `warning` specially.)
- Images:
  - `image` - see the *images* section;
  - `figure` - an image with caption and optional legend.
- Additional body elements:
  - `contents <table-of-contents>` - a local table of contents for the sections in the current file only;
  - `rubric` - a heading without relation to the document's sections that won't be included in any table of contents;
  - `topic` and `sidebar` - special highlighted body elements;
  - `epigraph` - a block quote with optional attribution line;
  - `container` - a container with a custom class, useful to generate an outer ``` ``<div>`` ``` in HTML output.
- *Special tables*:
  - `table` - a table with title;
  - `csv-table` - a table generated from comma-separated values;
  - `list-table` - a table generated from a list of lists.
- Special directives:
  - `include` - include reStructuredText from another file;
  - `raw` - include raw target-format markup, such as LaTeX;
  - `class` - assign a class attribute to the next element.

Table of contents To include a table of contents within a given document, use the directive `contents`. The following example creates a local table of contents with a maximum of two levels (below the level where it is located):

```
Part II
#######

Chapter 1
*********

.. contents::
   :depth: 2
   :local:
```

```
Heading 1
=========
```

The `toctree` directive creates a table of contents that collets information from several files. The following example creates a table of contents from the sections of various documents (up to a depth of 3 levels). The `:glob:` option allows all documents in the 'chapter2' folder to be included (sorted according to their name):

```
.. toctree::
   :glob:
   :maxdepth: 3

   preamble
   chapter1/part1
   chapter1/conclusion
   chapter2/*
   references
```

Note: When building HTML pages from the default template, a `<div class="sphinxsidebar">` is created that holds a 'table of contents' with links to the document sections. The number of levels in the sidebar can be controlled. For example, placing `:tocdepth: 3` in the beggining of the document restricts the number of levels to 3.

Images reST supports an image directive, used like so:

```
.. image:: gnu.png
   (options)
```

The file name given (here `gnu.png`) must either be relative to the source file, or absolute (which means that they are relative to the top source directory).

For example, the file `sketch/spam.rst` could refer to the image `images/spam.png` as `../images/spam.png` or as `/images/spam.png`.

The image size options (`width` and `height`) should be specified in points (`pt`), as that will best support output to different formats (`HTML`, `LaTeX`).

Figures A `figure` consists of image data (including image options), an optional caption (a single paragraph), and an optional legend (arbitrary body elements):

```
.. figure:: picture.png
   :scale: 50 %
   :alt: map to buried treasure

   This is the caption of the figure (a simple paragraph).

   The legend consists of all elements after the caption.  In this
   case, the legend consists of this paragraph and the following
   table:

   +-----------------------+-----------------------+
   | Symbol                | Meaning               |
   +=======================+=======================+
   | .. image:: tent.png   | Campground            |
   +-----------------------+-----------------------+
   | .. image:: waves.png  | Lake                  |
```

```
+-----------------------+-----------------------+
| .. image:: peak.png   | Mountain              |
+-----------------------+-----------------------+
```

There must be blank lines before the caption paragraph and before the legend. To specify a legend without a caption, use an empty comment (`..`) in place of the caption.

Special tables The `table` directive associates a title with the following table:

```
.. table:: User list

   ==========  =========
   First name  Last name
   ==========  =========
   John        Doe
   Jane        Dove
   ==========  =========
```

A `list-table` is created from a uniform two-level bullet list:

```
.. list-table:: User list
   :header-rows:1

   *  - First name
      - Last name
   *  - John
      - Doe
   *  - Jane
      - Dove
```

A `csv-table` is created from comma-separated values (either in the document or in an external file):

```
.. csv-table:: User list
   :header:"First name","Last name"

   "John","Doe"
   "Jane","Dove"
```

Another example of `csv-table`, using and external file:

```
.. csv-table:: Table 1 - Legend of the table goes here...
   :header-rows: 1
   :stub-columns: 1
   :file: ../tables/table1.csv
```

An `exceltable` can also be used:

```
.. exceltable:: Table 1 - Legend of the table goes here...
   :file: ../tables/tables.xls
   :sheet: table1
   :selection: A1:C20
   :header: 1
```

Using Excel tables requires an additional module *sphinxcontrib.exceltable* that is an extension for Sphinx, that adds support for including spreadsheets, or part of them, into Sphinx document. It can be installed using pip:

```
pip install sphinxcontrib-exceltable
```

Then the project `conf.py` file needs to be updated:

```
# Add ``sphinxcontrib.exceltable`` into extension list
extensions = ['sphinxcontrib.exceltable']
```

Another alternative is xmltable (https://pythonhosted.org/rusty/xmltable.html).

Footnotes For footnotes, use `[#name]_` to mark the footnote location, and add the footnote body at the bottom of the document after a "Footnotes" rubric heading, like so:

```
Lorem ipsum [#first-footnote-name]_ dolor sit amet [#second-footnote-name]_

.. rubric:: Footnotes

.. [#first-footnote-name] Text of the first footnote.
.. [#fsecond-footnote-name] Text of the second footnote.
```

You can also explicitly number the footnotes (`[1]_`) or use auto-numbered footnotes without names (`[#]_`).

This is a tip.

To facilitate editing, auto-numbered footnotes should **not** be used. Instead, use short descriptive names (that simplify cross-referencing).

Citations Standard reST citations are supported:

```
Lorem ipsum [Ref]_ dolor sit amet.

.. [Ref] Book or article reference, URL or whatever.
```

Citation usage is similar to footnote usage, but with a label that is not numeric or begins with `#`.

When the documentation is built using the Sphinx document generator, the citations are "global", meaning that every citation can be referenced from any .rst files. In this case, a separate file may be created (e.g. a `references.rst` file).

This is a tip.

See *Managing bibliographic citations in Sphinx* for further information.

Substitutions reST supports "substitutions", which are pieces of text and/or markup referred to in the text by `|name|`. They are defined like footnotes with explicit markup blocks, like this:

```
.. |name| replace:: replacement *text*
```

or this:

```
.. |caution| image:: warning.png
             :alt: Warning!
```

If you want to use some substitutions for all documents, put them into a separate file (e.g. `substitutions.txt`) and include it into all documents you want to use them in, using the `include` directive.

Be sure to use a file name extension which different from that of other source files, to avoid Sphinx finding it as a standalone document. For example, use the `.rst` file extension for the source files, and the `.txt` file extension for the files which are to be included.

This is a tip.

This is useful in technical documentation such as User's Manuals, where a substitution file can be built for each localised version of the interface elements (menus, messages, etc), guaranteeing the consistency of the document translation with the software's human user interface.

Warning.

Substitutions do NOT work inside directives (or inside the options of a directive).

Do not try to google for a solution (...been there). It is a design limitation: RST markup can not be nested. Period.

Comments Every explicit markup block which isn't a valid markup construct is regarded as a comment. For example:

```
.. This is a comment.
```

You can indent text after a comment start to form multiline comments:

```
..
   This whole indented block
   is a comment.

   Still in the comment.
```

This is a style convention.

Comments can also be used as placeholders to mark places within the document. For example:

- the `.. links-placeholder` can mark the place where hyperlinks are kept together at the end of the document;
- the `.. metadata-placeholder` can mark the place where document metadata (author, date, etc) is kept together at the beginning of the document.

Tools for reStructuredText

Introduction

This document presents some of the tools available for working with reStructuredText.

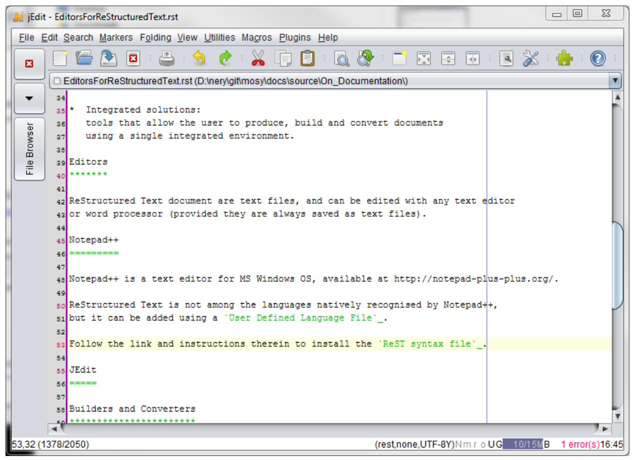
It is divided in the following parts:

- Editors: simple text editors providing syntax highlight for reST documents.
- Builder and Converters: tools that automatically convert reST documents to other formats such as HTML, PDF, DOC, ODT, etc.
- Integrated solutions: tools that allow the user to produce, build and convert documents using a single integrated environment.

Editors

reStructuredText documents are text files, and can be edited with any text editor or word processor (provided they are always saved as text files).

JEdit *jEdit* is a FOSS text editor, written in Java (so it runs in Windows, Mac OS X, Linux, etc.). ReST is among the 211 languages supported natively by jEdit.

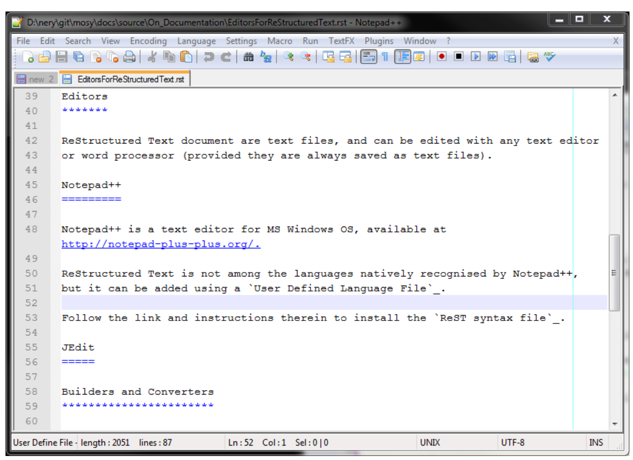


Notepad++ *Notepad++* is a FOSS text editor for **MS Windows OS** only.

reStructuredText is not among the languages natively recognised by Notepad++, but it can be added using a **'User Defined Language File'_** (see install instructions below the list of available language files).

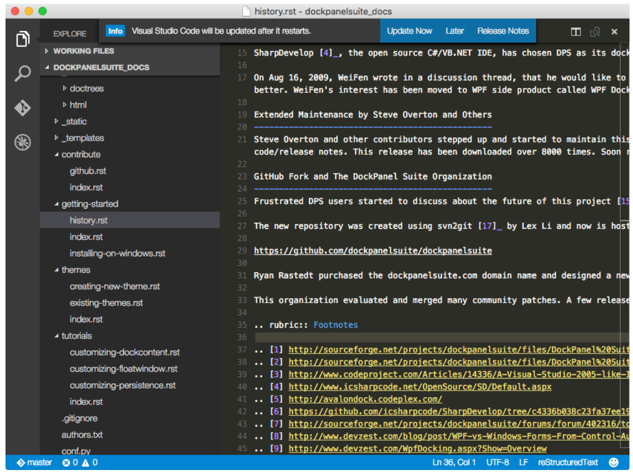
Follow the link to download the **'ReST syntax file'_**.

Notepad++ is simpler and more user friendly than jEdit.



ReText ReText is a simple editor that reads your text with MarkDown or HTML markup and saves it as plain text, HTML or PDF. It is written in Python using Qt libraries.

Visual Studio Code Visual Studio Code is a FOSS text editor, written in TypeScript (so it runs in Windows, Mac OS X, Linux, etc.). ReST is not among the languages natively supported by Visual Studio Code, but it can be added using an extension from LeXtudio.



Builders and converters

Todo

Section on Builders and Converters such as Sphinx and Pandoc.

Sphinx Sphinx is a Python documentation generator.

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

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

```
easy_install -U Sphinx
```

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

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

Rst2Pdf rst2pdf is a tool for transforming reStructuredText to PDF using ReportLab. To install rst2pdf on Windows you also need Python because rst2pdf is coded in python.

Rst2pdf uses ReportLab, which can be installed using:

```
easy_install reportlab
```

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

```
SET VS90COMNTOOLS=%VS100COMNTOOLS%
```

How to install rst2pdf on Windows?

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

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

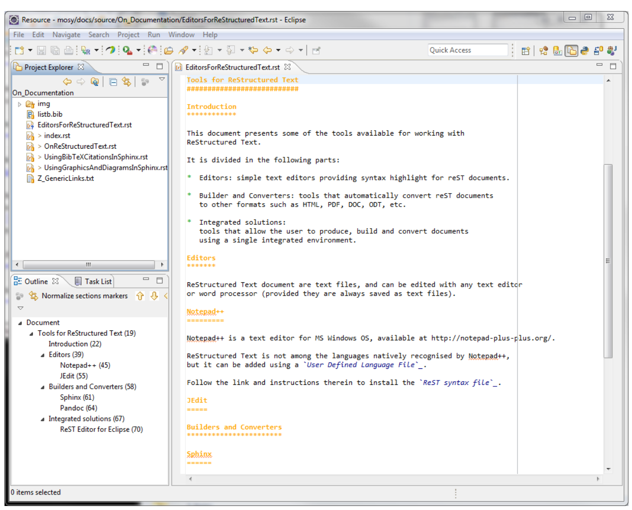
Pandoc

Integrated solutions

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

This ReST editor has several advantages, namely:

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



Publishing the documents online

Introduction

Sphinx documentation can be hosted online.

Dropbox

Google drive

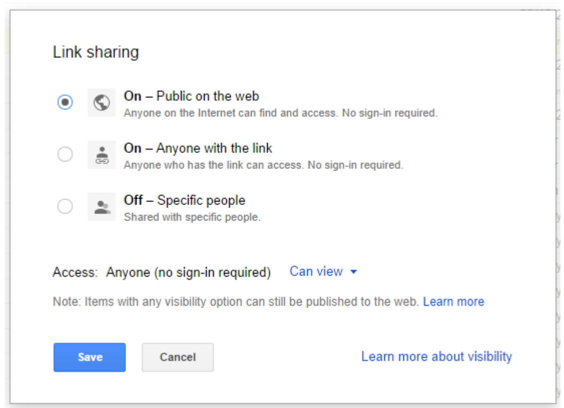
Note: Sources

https://kb.wisc.edu/helpdesk/page.php?id=38083

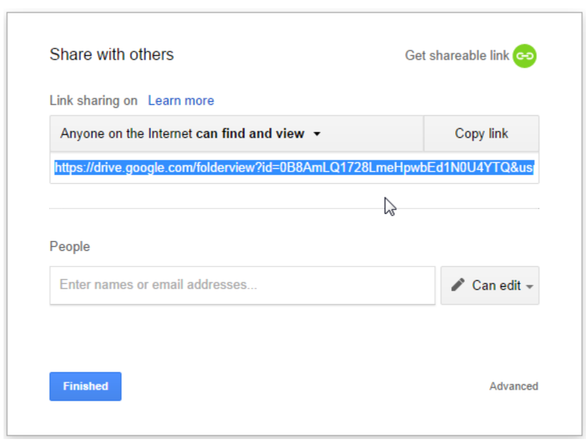
https://support.google.com/drive/answer/2881970?hl=en

https://developers.google.com/drive/web/publish-site

1. Create a folder in Google Drive and share it as `Public on the Web`



2. Upload the folder containing the HTML build, Javascript, and CSS files to this folder.
3. Find the shareable link of the HTML folder and copy the unique identifier



   For example, if the link is `https://drive.google.com/folderview?id=0B8AmLQ1728LmeHpwbEd1N0U4YTQ&u` then the unique id is `0B8AmLQ1728LmeHpwbEd1N0U4YTQ`.

4. So the link to your documentation will be

   www.googledrive.com/host/the-unique-identifier-that-you-just-copied/index.thml

   For example, the link to these pages is:

www.googledrive.com/host/0B8AmLQ1728LmeHpwbEd1N0U4YTQ/index.html

Read The Docs

Github.io

Note: Sources

http://lucasbardella.com/blog/2010/02/hosting-your-sphinx-docs-in-github

http://daler.github.io/sphinxdoc-test/index.html

Turning Jekill off

Note: Source

https://help.github.com/articles/using-jekyll-with-pages#turning-jekyll-off

You can completely opt out of Jekyll processing by creating a file named `.nojekyll` in the root of your Page repository and pushing that file to GitHub.

This should only be necessary if your site uses directories that begin with an underscore, as Jekyll sees these as special directories and does not copy them to the final destination.

Since Sphinx puts all the static files in a `_static` folder, this needs to be done, otherwise the stylesheets, etc, won't be uploaded to the html site.

Showing source code examples in Sphinx

Standard reST literal blocks are started by `::` at the end of the preceding paragraph and delimited by indentation.

Highlight directive

The default highlighting language is Python: it can be be changed using the `highlight` directive within a document:

```
.. highlight:: html

The literal blocks are now highlighted as HTML, until a new directive is found.

::
   <html><head></head>
   <body>This is a text.</body>
   </html>

The following directive changes the hightlight language to SQL.

.. highlight:: sql

::
   SELECT * FROM mytable

.. highlight:: none
```

```
From here on no highlighting will be done.

::
   SELECT * FROM mytable
```

Code-block directive

The `code-block` directive can be used to declare the specific language to be used in a block, regardless of the highlighting language:

```
The following is a SQL statement.

.. code-block:: sql
   :linenos:

   SELECT * FROM mytable
```

Line numbers are useful for long blocks such as this one:

```sql
-- http://www.postgresonline.com/journal/index.php?/archives/97-SQL-Coding-Standards-To-Each-His-Own-
SELECT persons.id, persons.first_name, persons.last_name, forums.category,
   COUNT(DISTINCT posts.id) as num_posts,
   COALESCE(MAX(comments.rating), 0) AS highest_rating,
   COALESCE(MIN(comments.rating), 0) AS lowest_rating
FROM persons JOIN posts ON persons.id = posts.author
   JOIN forums on posts.forum = forums.id
   LEFT OUTER JOIN comments ON posts.id = comments.post
WHERE persons.status > 0
   AND forums.ratings = TRUE
   AND comments.post_date > ( now() - INTERVAL '1 year')
GROUP BY persons.id, persons.first_name, persons.last_name, forums.category
HAVING count(DISTINCT posts.id) > 0
ORDER BY persons.last_name, persons.first_name;
```

Literalinclude directive

Another option is to include part of a given source code file, like this:

```
.. literalinclude:: filename
   :linenos:
   :language:
   :lines:
   :start-after:
   :end-before:
   :emphasize-lines:
```

Just below is a example:

```
Literalinclude directive
************************

Another option is to include part of a given source code file, like this::
```

Instead of using line numbers (which can change), it is possible to use the options `:start-after` and `:end-before:` that search the included file for lines containing the specified text. For example:

```
.. literalinclude:: ShowingCodeExamplesInSphinx.rst
   :language: rst
   :start-after: Instead of using
   :end-before: For example
```

produces this result:

```
use the options ``:start-after`` and ``:end-before:``
that search the included file for lines containing the specified text.
```

Pygments lexers

Syntax highlighting is done by Pygments (if installed): any of the **'Pygments language lexers'_** can be used.

The following table lists some useful lexers (in no particular order).

Lexer	Shortname
Structured Query Language	sql
PostgreSQL dialect of SQL	postgresql
PostgreSQL procedural language	plpgsql
PostgreSQL console sessions	psql
ReStructured Text	rst
TeX and LaTeX	latex
DOS/Windows batch file	bat
Windows PowerShell	powershell
Bash shell scripts	bash
Bash shell sessions	console
Cascading Style Sheets	css
HTML 4 and XHTML 1	html
XML	xml
XSLT	xslt
XQuery	xquery
JavaScript	javascript
JSON data structures	json
PHP source code	php
PHP embedded in HTML	html+php
Python 2	python
Python 2 tracebacks	pytb
Python console	pycon
Java	java
Configuration file in the Java's properties format	jproperties
Configuration file in the Apache config file format	apacheconf
R source code (or S, or S-plus)	r
R console	rout
Matlab	matlab
Matlab sessions	matlabsession
NumPy	numpy

Creating diagrams in Sphinx

Using Graphviz

Besides using raster images (PNG, JPG, etc.), diagrams can be included with the **'sphinx.ext.graphviz'_** extension.

Graphviz_ is an open source graph visualisation software. Graph visualisation is a way of representing structural information as diagrams of abstract graphs and networks.

It must be installed before the extension can be used.

Due to current (2013.10) compatibility issues with PlantUML, it may be preferable to install GraphViz 2.28 instead.

Including the extension in the project configuration file The Graphviz extension is included with Sphinx, but the extension must be enabled in the `conf.py` file:

```
extensions = ['sphinx.ext.graphviz']
```

Changing the configuration file

Extensions local to a project should be put within the project's directory structure. Set Python's module search path, sys.path, accordingly so that Sphinx can find them. E.g., if your extension foo.py lies in the exts subdirectory of the project root, put into conf.py:

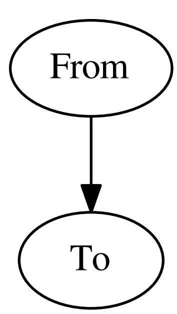
```
import sys, os
sys.path.append(os.path.abspath('exts'))
extensions = ['foo']
```

You can also install extensions anywhere else on sys.path, e.g. in the site-packages directory.

Examples This code:

```
.. graphviz::

   digraph {
      "From" -> "To";
   }
```

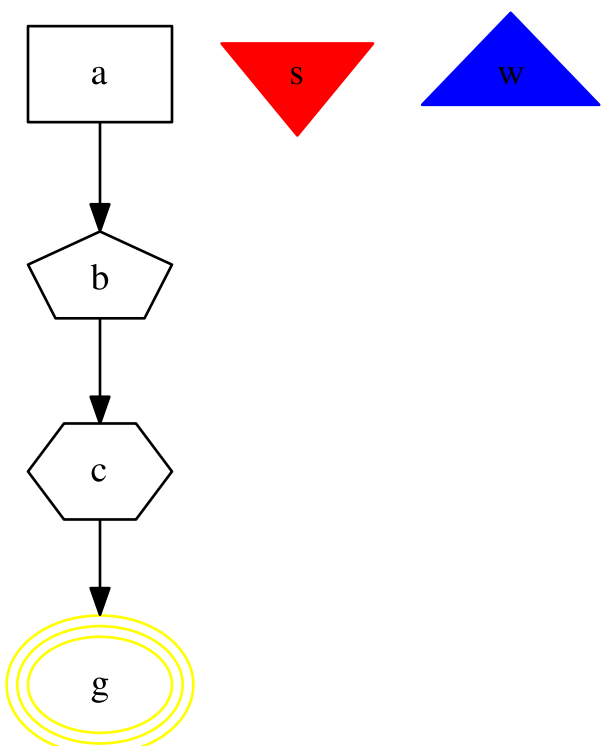
has this result:



This code:

```
.. graphviz::

   digraph Flatland {

      a -> b -> c -> g;
      a  [shape=polygon,sides=4]
      b  [shape=polygon,sides=5]
      c  [shape=polygon,sides=6]

      g [peripheries=3,color=yellow];
      s [shape=invtriangle,peripheries=1,color=red,style=filled];
      w  [shape=triangle,peripheries=1,color=blue,style=filled];

      }
```

has this result:



Numerous examples are available online:

- https://en.wikipedia.org/wiki/DOT_(graph_description_language)
- http://www.graphviz.org/pdf/dotguide.pdf
- http://graphs.grevian.org/example.html

Using PlantUML

The **'Sphinx PlantUML extension'_** (in this case a contributed extension) allows Sphinx to embed UML diagrams by using PlantUML.

PlantUML_ is a Java component that allows to quickly write simple UML diagrams:

- use case diagrams,
- class diagrams,
- activity diagrams,
- state diagrams,
- component diagrams,
- sequence diagrams,
- object diagram.

Diagrams are defined using a simple and intuitive language. This can be used within many other tools. Images can be generated in PNG or SVG format.

Installing the extension The module is installed with the following command:

```
pip install sphinxcontrib-plantuml
```

Including the extension in the project configuration file The extension must be enabled in the `conf.py` file:

```
extensions = ['sphinxcontrib.plantuml']
```

The path to the PlantUML file may have to be specified (assuming that Java itself is already in the search path):

```
plantuml = 'java -jar ../utils/plantum.jar'
```

PlantUML requires Graphviz and an environment variable may have to be defined, pointing to the `dot` executable. For example (in Linux or OS-X):

```
setenv GRAPHVIZ_DOT /usr/local/bin/graphviz/dot
export GRAPHVIZ_DOT
```

Note: For Ubuntu users

Files with the .sh extension in the /etc/profile.d directory get executed whenever a bash login shell is entered (e.g. when logging in from the console or over ssh), as well as by the DisplayManager when the desktop session loads:

```
/etc/profile.d/*.sh
```

You can for instance create the file /etc/profile.d/myenvvars.sh and set variables like this:

```
export GRAPHVIZ_DOT=/usr/bin/dot
```

Note: For Windows users

Regardless of the existence of the GRAPHVIZ_DOT environment variable, the path to the Graphviz bin folder is apparently required to be in the PATH variable as well.

Examples In the Sphinx reST documents, simply begin the PlantUML code with the `uml` directive.

This is the code for the example above:

```
.. uml::

   @startuml
   user -> (use PlantUML)

   note left of user
      Hello!
   end note
   @enduml
```

Another example:

This is the code for the example above:

```
.. uml::

   @startuml
   Alice -> Bob: Hi!
   Alice <- Bob: How are you?
   @enduml
```

Class diagram

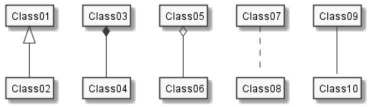
Diagram This is the diagram generated by the Sphinx PlantUML extension.

Code This is the code for example above.

```
.. uml::

      @startuml

      'style options
      skinparam monochrome true
      skinparam circledCharacterRadius 0
      skinparam circledCharacterFontSize 0
      skinparam classAttributeIconSize 0
      hide empty members

      Class01 <|-- Class02
      Class03 *-- Class04
      Class05 o-- Class06
      Class07 .. Class08
      Class09 -- Class10

      @enduml
```

Note that in docutils / Sphinx, `@startuml` and `@enduml` could be omitted. However, it is useful to keep these lines: is necessary, PlantUML can be used (outside Sphinx) to generate the PNG image files with the diagrams directly from the text file; also, if editing the code in Eclipse, the PlantUML diagrams can be previewed without the necessity of building the documentation.

Image The image was exported using the Eclipse PlantUML plug-in. It it static, but can be resized...



Other examples

```
.. uml::

      @startuml

      'style options
      skinparam monochrome true
      skinparam circledCharacterRadius 0
      skinparam circledCharacterFontSize 0
      skinparam classAttributeIconSize 0
      hide empty members

      class Car

      Driver - Car : drives >
      Car *- Wheel : have 4 >
      Car -- Person : < owns

      @enduml
```

To declare fields and methods, you can use the symbol ":" followed by the field's or method's name. The system checks for parenthesis to choose between methods and fields.

```
.. uml::

      @startuml

      'style options
      skinparam monochrome true
      skinparam circledCharacterRadius 9
      skinparam circledCharacterFontSize 8
      skinparam classAttributeIconSize 0
      hide empty members

      abstract class AbstractClass {
        - privateField
        + publicField
        # protectedField
        ~ packagePrivateField
        - privateMethod()
        + publicMethod()
        # protectedMethod()
        ~ packagePrivateMethod()
         }

      class Dummy {
        {static} staticID
        {abstract} void methods()
         }

      class Flight {
         flightNumber : Integer
         departureTime : Date
```

```
    }

package "Classic Collections" {

    abstract class AbstractList
    abstract AbstractCollection
    interface List
    interface Collection

    List <|-- AbstractList
    Collection <|-- AbstractCollection

    Collection <|- List
    AbstractCollection <|- AbstractList
    AbstractList <|-- ArrayList

    class ArrayList {
      Object[] elementData
      size()
       }
}

enum TimeUnit {
  DAYS
  HOURS
  MINUTES
}


class Student {
  Name
}
Student "0..*" -- "1..*" Course
(Student, Course) .. Enrollment

class Enrollment {
  drop()
  cancel()
}

@enduml
```

Use case diagram

Code

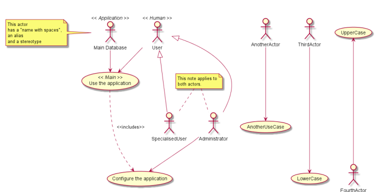
Diagram This is generated by the Sphinx PlantUML extension.

Code

```
.. uml::

   @startuml
   actor "Main Database" as DB << Application >>

   note left of DB
```

```
    This actor
    has a "name with spaces",
    an alias
    and a stereotype
end note

actor User << Human >>
actor SpecialisedUser
actor Administrator

User <|--- SpecialisedUser
User <|--- Administrator

usecase (Use the application) as (Use) << Main >>
usecase (Configure the application) as (Config)
Use ..> Config : <<includes>>

User --> Use
DB --> Use

Administrator --> Config

note "This note applies to\nboth actors." as MyNote
MyNote .. Administrator
MyNote .. SpecialisedUser

'  this is a text comment and won't be displayed
AnotherActor ---> (AnotherUseCase)

'  to increase the length of the edges, just add extras dashes, like this:
ThirdActor ----> (LowerCase)

'  The direction of the edge can also be reversed, like this:
(UpperCase) <---- FourthActor

@enduml
```

Image The image was exported using the Eclipse PlantUML plug-in. It it static, but can be resized...



Activity diagram (new syntax)

Diagram

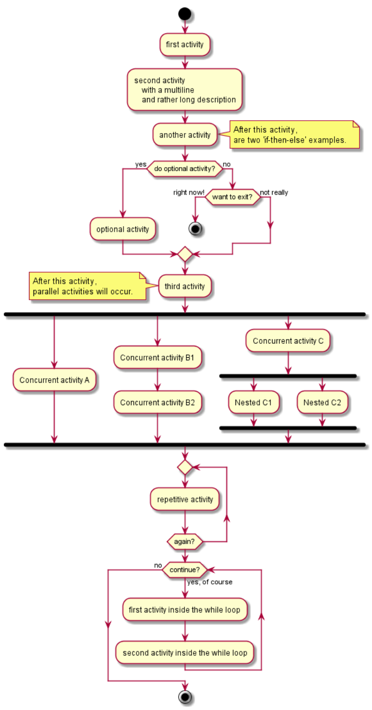
Code There are two syntaxes to create activity diagrams. The example utilises the new syntax (which is still incomplete).

```
.. uml::
```

```
@startuml

start

:first activity;

:second activity
 with a multiline
 and rather long description;

:another activity;

note right
  After this activity,
  are two 'if-then-else' examples.
end note

if (do optional activity?) then (yes)
   :optional activity;
else (no)

   if (want to exit?) then (right now!)
      stop
   else (not really)

   endif

endif

:third activity;

note left
  After this activity,
  parallel activities will occur.
end note

fork
   :Concurrent activity A;
fork again
   :Concurrent activity B1;
   :Concurrent activity B2;
fork again
   :Concurrent activity C;
   fork
   :Nested C1;
   fork again
   :Nested C2;
   end fork
end fork

repeat
   :repetitive activity;
repeat while (again?)

while (continue?) is (yes, of course)
  :first activity inside the while loop;
  :second activity inside the while loop;
endwhile (no)
```

```
   stop

   @enduml
```

Image The image was exported using the Eclipse PlantUML plug-in. It it static, but can be resized...



State diagram

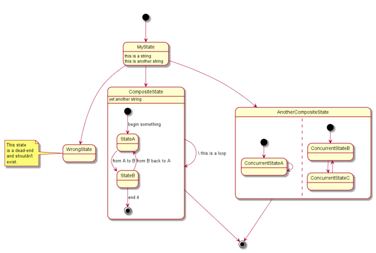
Diagram Generated by the Sphinx PlantUML extension.

Code

```
.. uml::

   @startuml

   [*] --> MyState
   MyState --> CompositeState
   MyState --> AnotherCompositeState
   MyState --> WrongState

   CompositeState --> CompositeState : \ this is a loop
   AnotherCompositeState --> [*]
   CompositeState --> [*]

   MyState : this is a string
   MyState : this is another string

   state CompositeState {
```

```
   [*] --> StateA : begin something
   StateA --> StateB : from A to B
   StateB --> StateA : from B back to A
   StateB --> [*] : end it

   CompositeState : yet another string
   }

   state AnotherCompositeState {

   [*] --> ConcurrentStateA
   ConcurrentStateA --> ConcurrentStateA

   --

   [*] --> ConcurrentStateB
   ConcurrentStateB --> ConcurrentStateC
   ConcurrentStateC --> ConcurrentStateB
   }

   note left of WrongState
      This state
      is a dead-end
      and shouldn't
      exist.
   end note

   @enduml
```

Image The image was exported using the Eclipse PlantUML plug-in. It it static, but can be resized...



GUI mockups

PlantUML can also be used for GUI mockups (see http://plantuml.sourceforge.net/salt.html).

Example

Code

```
.. uml::

   @startuml
   salt
   {
     Just plain text
     [This is my button]
```

```
    ()  Unchecked radio
    (X) Checked radio
    []  Unchecked box
    [X] Checked box
    "Enter text here   "
    ^This is a droplist^
  }
  @enduml
```

Managing bibliographic citations in Sphinx

Introduction

reStructuredText *Citations* are ill-adapted to parenthetical referencing (a.k.a. the **'Harvard System of Referencing'_**).

An alternative is to (manually) use the **'authorship trigraph'_** (common in older computer science texts).

The citation begins with 4 letters:

- one author: first 4 letters of name
- two authors: first 2 letters of author1, first 2 letters of author2
- three authors: first 2 letters of author1, first letter of author2, first letter of author3
- four authors: first letter of each author
- more than four authors: first letter of first four authors

The first letter of a name is always upper case.

After the authors' initials, put the two digits of the year (century-disambiguation is ignored).

If the symbol is exactly the same for two references, a lower case letter is attached.

To facilitate editing, citation text should be kept at the bottom of the document after a "References" rubric heading, like this:

```
.. rubric:: References

.. [BiDB79] Biskup, J.; Dayal, U.; Bernstein, P.A..: Synthesizing independent database schemas. In: A

.. [BeBe79a] Beeri, C.; Bernstein, P.A.: Computational problems related to the design of normal relat

.. [BeBe79b] Beeri, C.; Bernstein, P.A.: Computers are stupid. ACM Trans. Database Syst., No. 4, 1979
```

A similar option is to use the BibTeX `alpha` style:

- one author: first 3 letters of the last name
- two to four authors: first letters of last names concatenated
- more than four authors: first letters of last names of first three authors concatenated and a "+" sign at the end.

For the examples above, the alpha style citation would be: [BDB79], [BB79a] and [BB79b], respectively.

Using Sphinx BibTeX extension

Parenthetical referencing can be produced in Sphinx using the **sphinxcontrib-bibtex_** extension.

The **sphinxcontrib-bibtex_** extension allows BibTeX citations to be inserted into documentation generated by Sphinx.

The extension defines a new `bibliography` directive and a new `cite` role.

These work similarly to the LaTeX's `thebibliography` environment and `\cite` command.

The references are stored in a separate plain text BibTeX format file. Currently, only the `unsrt` and `plain` BibTeX styles are supported.

Please note that the current **sphinxcontrib-bibtex_** is a **beta** version.

Installing the extension The module is installed with:

```
pip install sphinxcontrib-bibtex
```

This is a tip.

For Windows users. To facilitate the installation of 3rd party Python packages, follow the instructions on how to **'add Distribute and Pip to the Python installation'_**.

Including the extension in the project configuration file The Sphinx project `conf.py` file must be altered to include:

```
extensions = ['sphinxcontrib.bibtex']
```

Example In the document, use the following syntax:

```
See :cite:`Strunk1979` for an introduction to stylish blah, blah...
```

And place the directive at the end of the document:

```
.. bibliography:: references.bib
```

The `references.bib` file should contain a BibTex bibliography, including an entry for:

```
@BOOK{Strunk1979,
  title = {The Elements of Style},
  publisher = {Macmillan},
  year = {1979},
  author = {Strunk, Jr., William and E. B. White},
  edition = {Third}
}
```

Using the Sphinx Natbib Extension

A more flexible alternative is to use http://wnielson.bitbucket.org/projects/sphinx-natbib/
This documentation can be completed iff required in this specific project.

Using the Sphinx Thesis Resource

See also http://jterrace.github.io/sphinxtr/html/ch-intro/index.html for various useful adaptations/extensions of Sphinx.

Using LaTeX directly in Sphinx

For advanced users, LaTeX can also be used directly in Sphinx (when only LaTeX output is required):

```
See :raw-tex:`\cite{Strunk1979}` for an introduction to stylish blah, blah...
```

And insert the bibliography at the end of the document:

```
.. raw:: latex

 \bibliographystyle{plain}
 \bibliography{listb.bib}
```

Managing BibTeX bibliographies

The BibTeX files can be easily managed with JabRef.

JabRef is an open source bibliography reference manager. The native file format used by JabRef is BibTeX, the standard LaTeX bibliography format.

JabRef runs on the Java (version 1.6 or newer), and should work equally well on Windows, Linux and Mac OS X.

Creating equations in Sphinx

LaTeX

The syntax for writing equations is LaTeX.

Only brief examples are included here, since LaTeX as a rather steep learning curve, and AMS LaTeX is only concerned with math support.

The following links are useful:

- See ftp://ftp.ams.org/pub/tex/doc/amsmath/short-math-guide.pdf for a not-so-short but clear syntax guide.
- See http://www.ams.org/publications/authors/tex/amslatex for complete references.
- See http://mirrors.fe.up.pt/pub/CTAN/info/examples/mil/mil.pdf for a not-so-gentle introduction to LaTeX.

MathJax

In Sphinx, the rendering (display) of the equations can be done in different ways, that will not be discussed here.

The selected option is to use the `sphinx.ext.mathjax` extension. This extension uses the JavaScript package *MathJax* to transform the LaTeX markup to readable math live in the browser.

The disadvantages are the (large) size and load time of the MathJax library.

The `mathjax_path` in the `conf.py` file indicates where the MathJax library resides. By default, this is the MathJax site, but the path can be changed no cross-site scripting is allowed.

Equation editors or previewers

Given that LaTeX syntax may be daunting, a WYSIWYG math editor can be useful, or at least an interactive previewer:

- If the objective is simply to preview the result, the online **'Interactive LaTeX Editor'_** is very good option and includes numerous equations as examples.
- **LyX_** is a user-friendly LaTeX processor that includes an equation editor.

  In the long run, LyX may be the best choice: it as the same dependencies as EqualX, a larger development and user community, and does not require virtually any LaTeX knowledge.
- **EqualX_** is a LaTeX equation editor (not a document processor as LyX): it can be used to create the equations and then paste the code into the ReST document.

  Like **LyX_**, EqualX requires a LaTeX distribution (in Linux, the dependencies are automatically installed and **TeXLive_** is included in the official repositories of all major distributions; for Windows systems, **MiKTeX_** is a possible alternative).

Examples

See additional examples at http://sphinx-doc.org/ext/math.html.

Code:

```
If :math:`\sigma_{1}` equals :math:`\sigma_{2}` then etc, etc.
```

Output:

If σ_1 equals σ_2 then etc, etc.

Code:

```
:math:`\underline{x}=[  x_{1}, ...,  x_{n}]^{T}`
```

Output:

$\underline{x}=[x_{1},...,x_{n}]^{T}$

Code:

```
\langle \alpha, \beta  \rangle
\in
\Biggl \lbrace
{
M,\text{ if }
   {
    l(\underline{x}) =
      \frac { p(\underline{x}|M ) } { p(\underline{x}|U) }
      \geq
       \frac { p(U) }{ p(M) } }
\atop
U, \text{ otherwise }
}
```

Output:

$$\langle \alpha, \beta \rangle \in \Biggl \lbrace { M,\text{ if } { l(\underline{x}) = \frac { p(\underline{x}|M) } { p(\underline{x}|U) } \geq \frac { p(U) }{ p(M) } } \atop U, \text{ otherwise } }$$

1.3 Using the version control system

1.3.1 Introduction

This section contains information about the main concepts of version control and basic information on how to use Git, the version control system selected for source-code and technical documentation management.

Currently, this section is rather technical and unpalatable. That will change...

1.3.2 References

On Version Control

Introduction

Version Control is the management of of changes to documents, computer programs, large web sites, and other collections of information. The set of files under version control is kept in a *repository*.

The **Version Control System** (VCS) is the application responsible for keeping track of the successive versions of a repository.

The basic workflow is:

1. A user clones the repository and creates a local *working copy* of the files. (The local copy can itself be a local repository under version control).
2. The user works on the local copy of the files. (Note that not every change to every file is registered as a *revision*).
3. When the user wants to (e.g. when a new version of a document is quite complete, or every day at 6:14pm ...), a group of new or modified files (or *changeset*) can be committed as a revision to the local working copy.
4. The user can also choose when to *commit* the revisions back into the original repository.
5. If the local revisions are merged into the original repository, a new revision point is created therein.

   The *merge* can be performed automatically by the version control system:

   (a) if the user has the required permissions to commit (`push`) to the original repository;

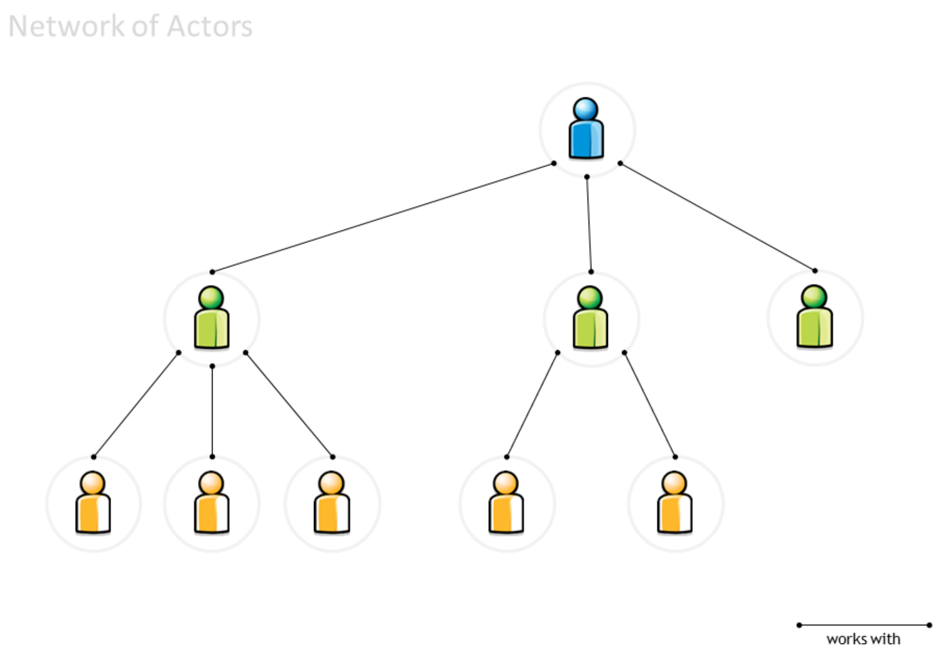
   (b) and if no *conflict* is detected (e.g. while the user was working in the local copy, someone else committed a revision to same files).
6. If the user does not have the required permissions to commit to the original repository, a `pull` request can be sent to the owner of the repository.

   The owner reviews the proposed changes, and accepts or rejects the changeset.

The following storyboard illustrates this steps.

Storyboard #1 - The basic workflow

This storyboard depicts a simplified workflow, using a simplified hierarchy. (In real-world use, links can exist between any two different actors and repositories. The technology allows networks with arbitrary configuration.)



Network of actors Consider a typical hierarchy:

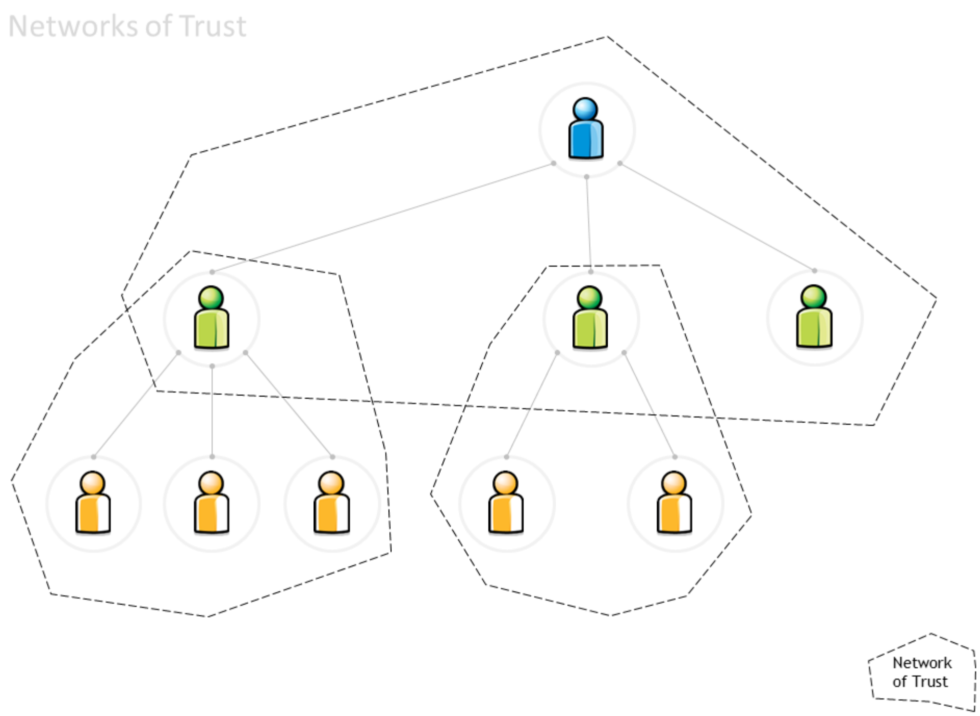
- The 'blue' actor only works and knows the 'green' actors.
- Each 'green' actor works with a distinct group of 'yellow' actors.
- The blue actor's responsibility is to collate the green actors' contributions (changes) and to resolve any conflicts between different changes proposed by distinct green actors.
- Each green actor's responsibility is similar, with regard to the yellow actors.

Networks of trust This hierarchy is a particular type of network (...a directed acyclic network or 'tree').

It can be viewed as 3 distinct "networks of trust".

The concept of "network of trust" simplifies the work:

- The blue actor trusts the green actor to review the work done by his yellow co-workers.
  - From the blue actor's point of view, the specific configuration of each 'green & yellow' network is irrelevant.
  - It is also irrelevant whether the all network is really a tree (or if a given yellow actor participates in two distinct subnetworks).
- Each actor needs only trust (and interact with) his immediate neighbourhood:
  - The green actor accepts any upstream changes approved by his blue neighbour.
  - The green actor approves (or declines) changes made by his yellow neighbours (or resolves conflicts between different changes).
- By definition, the blue actor can directly commit changes to his own blue repository of information. So can the green actors to their own green repositories.



- Each actor can also 'pull' into his repository any changes that his immediate neighbours have made.

Cloning a repository The initial workflow is depicted below:

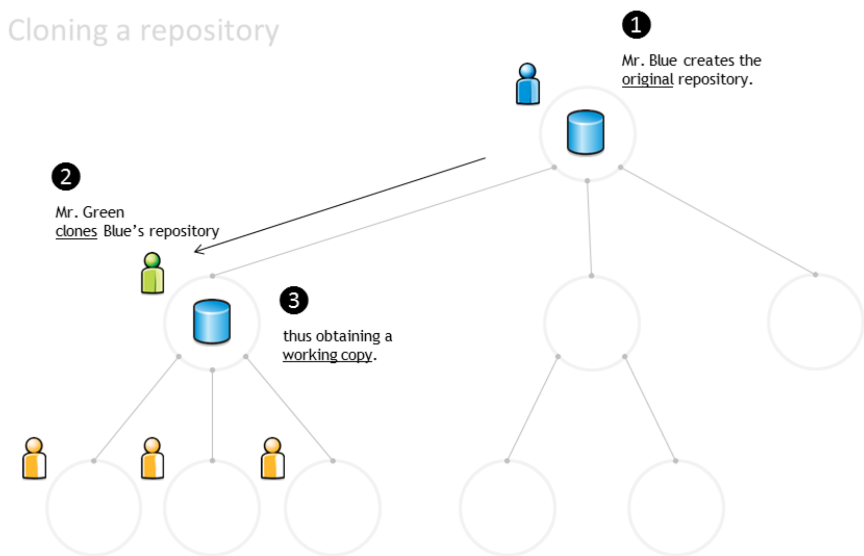
1. Mr Blue creates the **original** repository.
2. Mr Green **clones** Blue's repository
3. thus obtaining a **working** copy.

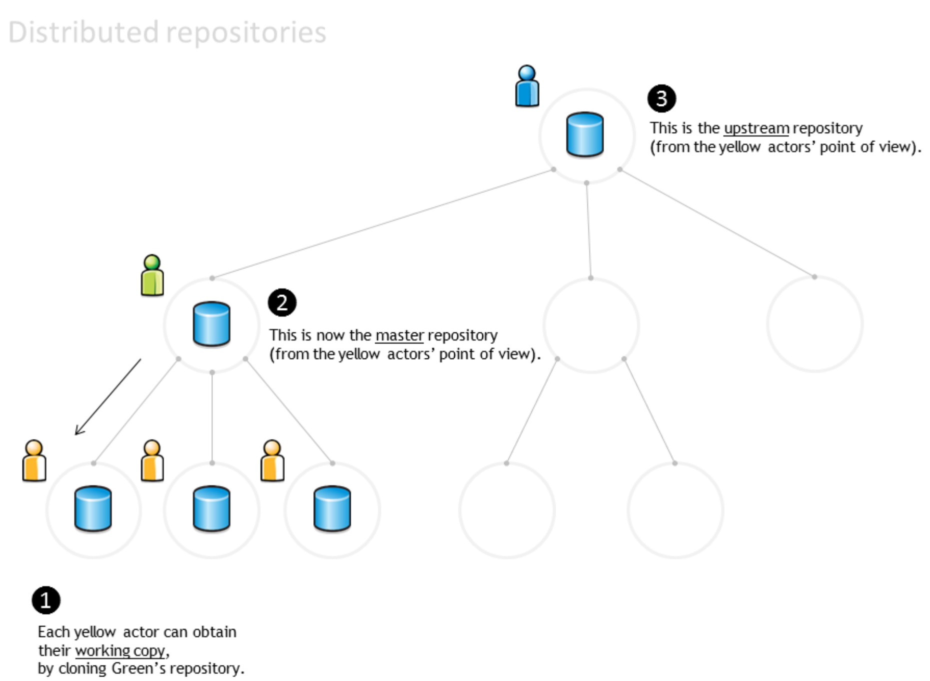
Distributed repositories The repository is then distributed to all the team:

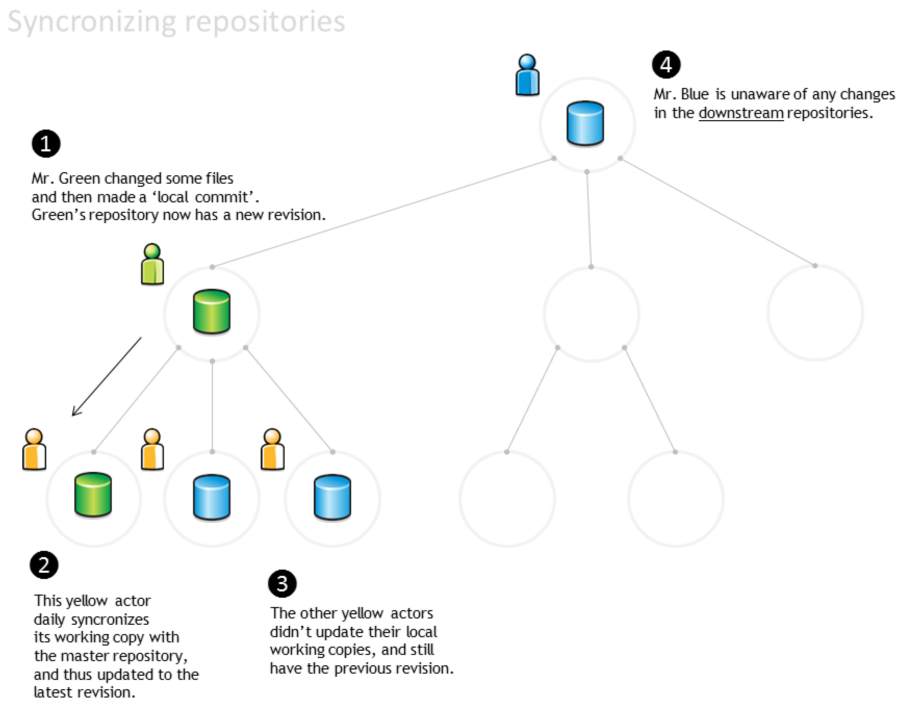
1. Each yellow actor can obtain their working copy, by cloning Green's repository.
2. Mr Green's repository is now the **master** repository for all the yellow actors.
3. Mr Blue's repository is now the **upstream** repository for all the yellow actors.

Synchronising repositories The repositories must be explicitly synced. For example, suppose that:

1. Mr Green changed some files and then made a 'local commit'. Green's repository now has a new revision (which does not exist in any other repository).
2. One of the yellow actors synchronises the local working copy everyday, to ensure that he has the latest files. His local working copy is updated with the latest revision made by Mr Green to the master repository.
3. The other yellow actors didn't update their local working copies, and still have the previous revision.
4. Meanwhile, Mr Blue is unaware of any changes in the downstream repositories.


Cloning a repository
1
Mr. Blue creates the original repository.
2
Mr. Green clones Blue's repository
3
thus obtaining a working copy.


Distributed repositories
3
This is the upstream repository (from the yellow actors' point of view).
2
This is now the master repository (from the yellow actors' point of view).
1
Each yellow actor can obtain their working copy, by cloning Green's repository.



Commit and merge Changes can also by propagated upstream. Suppose that:

1. A yellow actor has changed some files and made a 'local commit'. The local repository has a new revision (jargon: 'the local repository is one commit ahead of the master repository').
2. The yellow actor notifies Mr Green and asks him to merge the changeset into Mr Green's repository (jargon: 'sends Mr Green a pull request').
3. Mr Green reviews the changes, approves them (or not...) and merges the changeset into his own repository.
4. Meanwhile, Mr Blue is still unaware of any changes in the downstream repositories (he has received no pull requests).

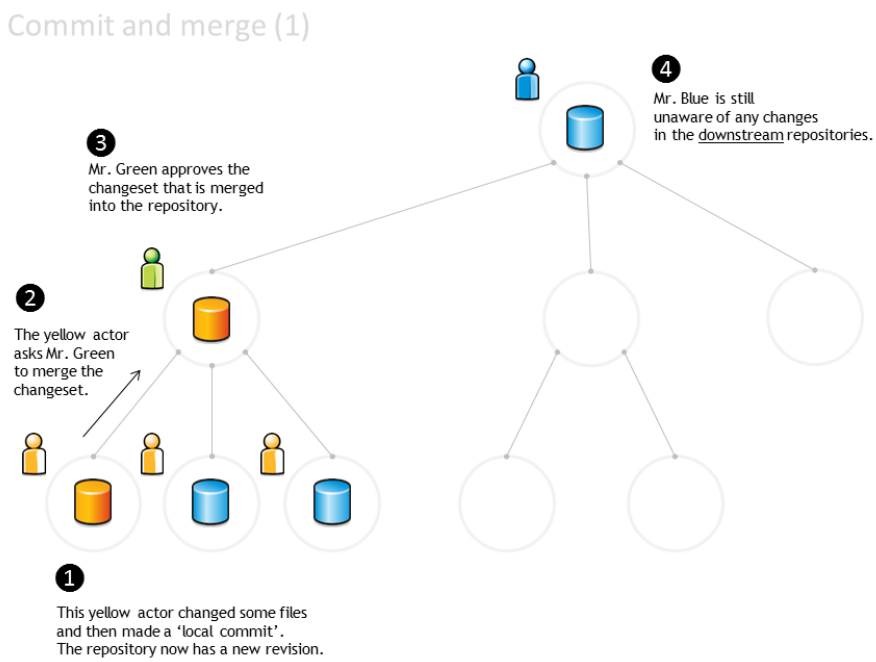
When the work assigned to Mr Green's team is ready:

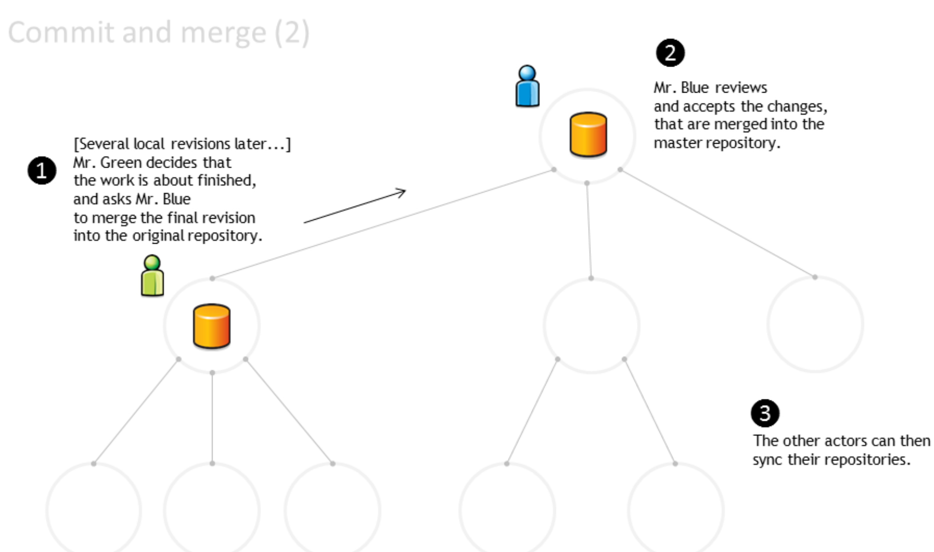
1. Mr Green send a 'pull request' to Mr Blue.
2. Mr. Blue reviews and accepts the changes, and updates his repository.
3. Everyone else can synchronise their repositories to the latest version.

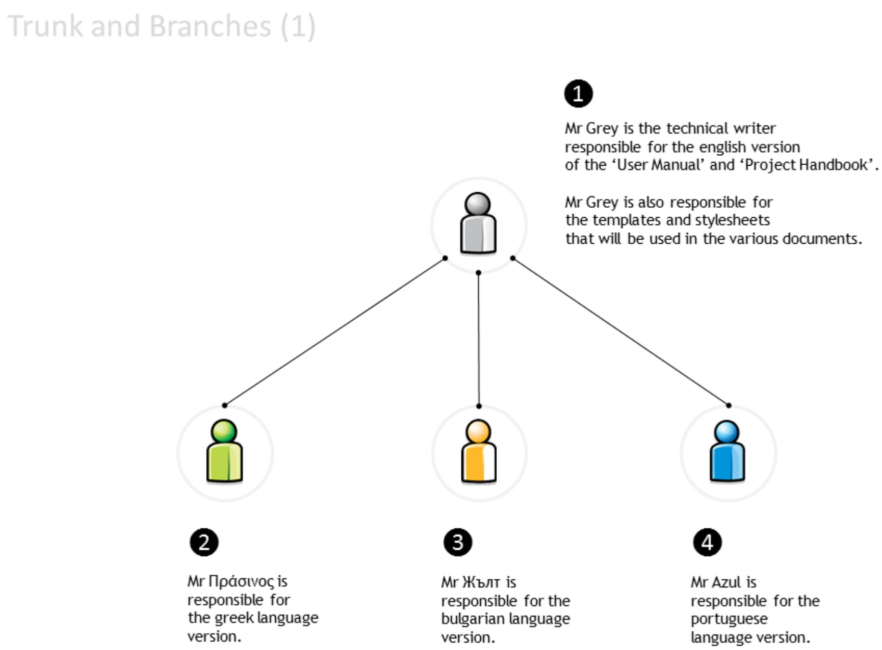
Storyboard #2 - Using branches to manage the document translation process

Trunk and branches Consider the following network:

- Mr Grey is the technical writer responsible for the English version of the 'User Manual' and 'Project Handbook'. Mr Grey is also responsible for the templates and stylesheets that will be used in the various documents.
- Mr $\Pi\rho\sigma\iota\nu$o is responsible for the Greek language version.
- Mr is responsible for the Bulgarian language version.
- Mr Azul is responsible for the Portuguese language version.


Commit and merge (1)
4
Mr. Blue is still unaware of any changes in the downstream repositories.
3
Mr. Green approves the changeset that is merged into the repository.
2
The yellow actor asks Mr. Green to merge the changeset.
1
This yellow actor changed some files and then made a 'local commit'. The repository now has a new revision.


Commit and merge (2)
2
Mr. Blue reviews and accepts the changes, that are merged into the master repository.
1
[Several local revisions later...] Mr. Green decides that the work is about finished, and asks Mr. Blue to merge the final revision into the original repository.
3
The other actors can then sync their repositories.



When the translation process begins:

1. Mr Grey creates a repository with the English documents (e.g. one file per chapter) and with the image files (e.g. the application screenshots), templates and stylesheets required to build the final document.
2. Messrs $\Pi\rho\sigma\iota\nu$o, and Azul all clone the English language repository and create their own working copies.
3. Each of them also creates a local branch: a replica of the files so that each can work on translating the text to their own language while reusing the image files and the stylesheet.

   In this example, the English version is the **trunk**. Each localised version is a **branch**.

When new documents are ready to be translated:

1. Mr Grey completes a new chapter and commits it to the repository.

   Mr Grey also changes some of the images.

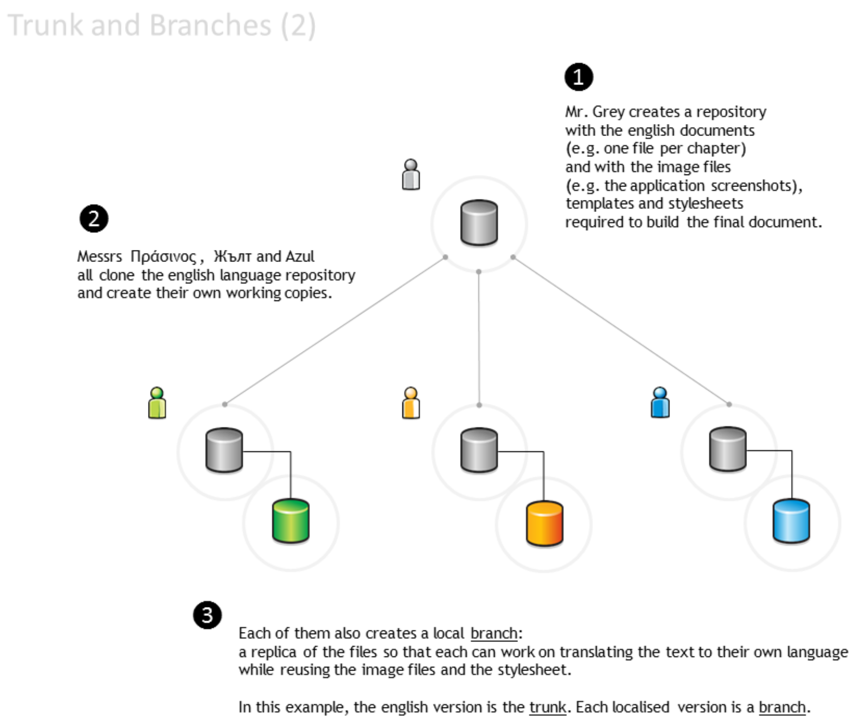
   A new revision is now available.
2. Messrs $\Pi\rho\sigma\iota\nu$o, and Azul synchronise their working copies with the master repository (only the trunk is updated).
3. Each of them also updates the local branch.

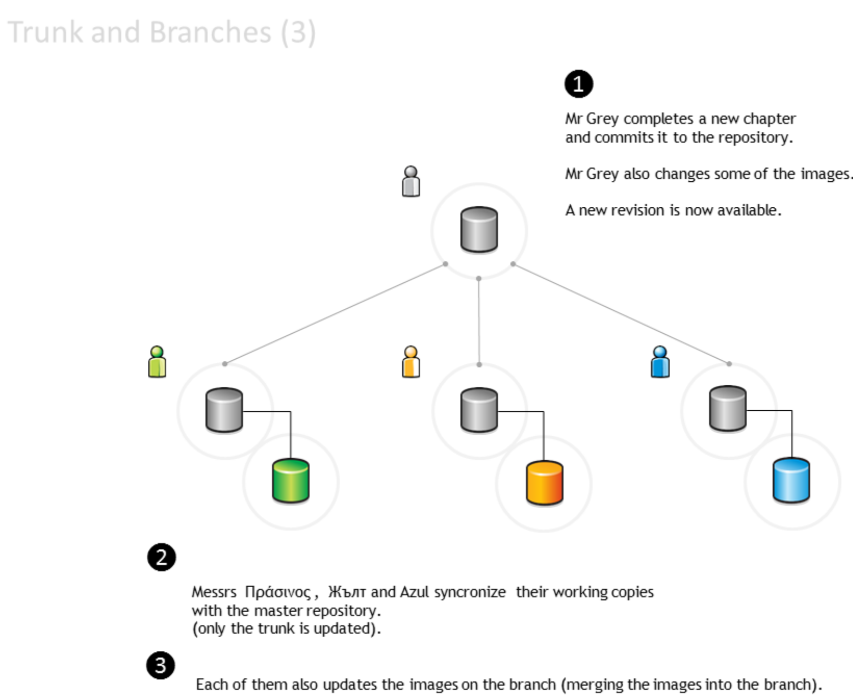
What if Mr $\Pi\rho\sigma\iota\nu$o detects some spelling errors in the English version?

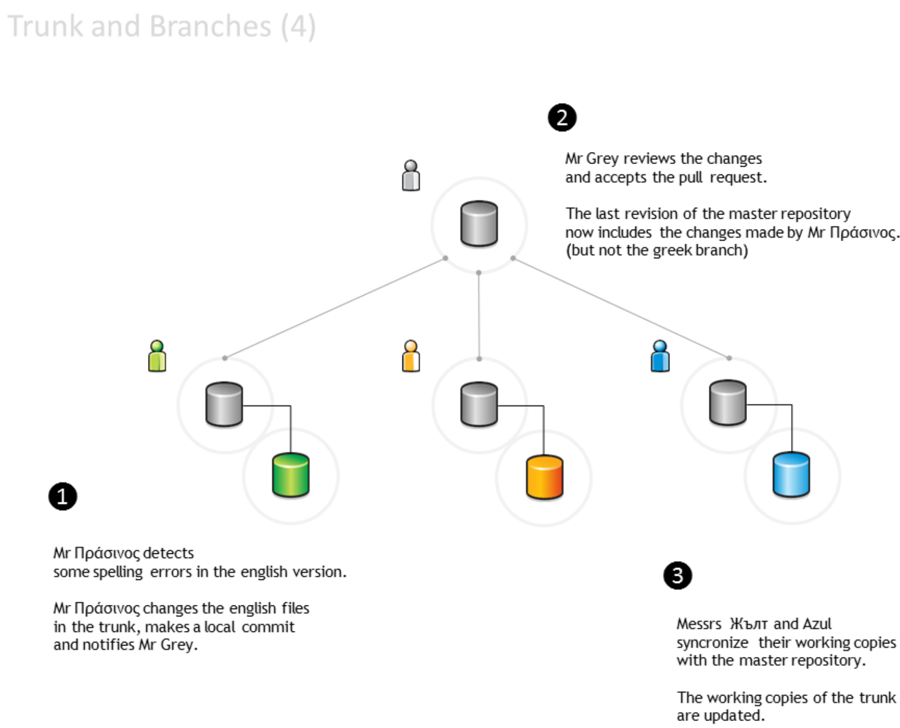
1. Mr $\Pi\rho\sigma\iota\nu$o changes the English files in the trunk, makes a local commit and notifies Mr Grey.
2. Mr Grey reviews the changes and accepts the pull request.

   The last revision of the master repository now includes the changes made by Mr $\Pi\rho\sigma\iota\nu$o (but not the Greek branch).
3. Messrs and Azul synchronise their working copies with the master repository.

   The working copies of the trunk are updated. Each team member must update their specific local branches.


Trunk and Branches (2)
1
Mr. Grey creates a repository
with the english documents
(e.g. one file per chapter)
and with the image files
(e.g. the application screenshots),
templates and stylesheets
required to build the final document.
2
Messrs Πράσινος , Жълт and Azul
all clone the english language repository
and create their own working copies.
3
Each of them also creates a local branch:
a replica of the files so that each can work on translating the text to their own language
while reusing the image files and the stylesheet.
In this example, the english version is the trunk. Each localised version is a branch.


Trunk and Branches (3)
1
Mr Grey completes a new chapter
and commits it to the repository.
Mr Grey also changes some of the images.
A new revision is now available.
2
Messrs Πράσινος , Жълт and Azul syncronize their working copies
with the master repository.
(only the trunk is updated).
3
Each of them also updates the images on the branch (merging the images into the branch).



Branches can function as different versions...

1. Mr $\Pi\rho\sigma\iota\nu$o changes the default stylesheet, including some styles that improve its use with the Greek alphabet.

   Changes are made only in the Greek language branch.

2. Mr makes a similar change, due to the Cyrillic alphabet.

   Changes are made only to the Bulgarian language branch.

3. Mr Azul dislikes the colour of chapter headings and changes the stylesheet in the Portuguese language branch.

   Mr Azul decides to submit the changes to Mr Grey, so that the trunk can also be changed.

4. Mr Grey reviews the changes made by Mr Azul, but does not accept them.

   The trunk stylesheet is not changed.

Glossary

baseline An approved revision of a file from which subsequent changes can be made.

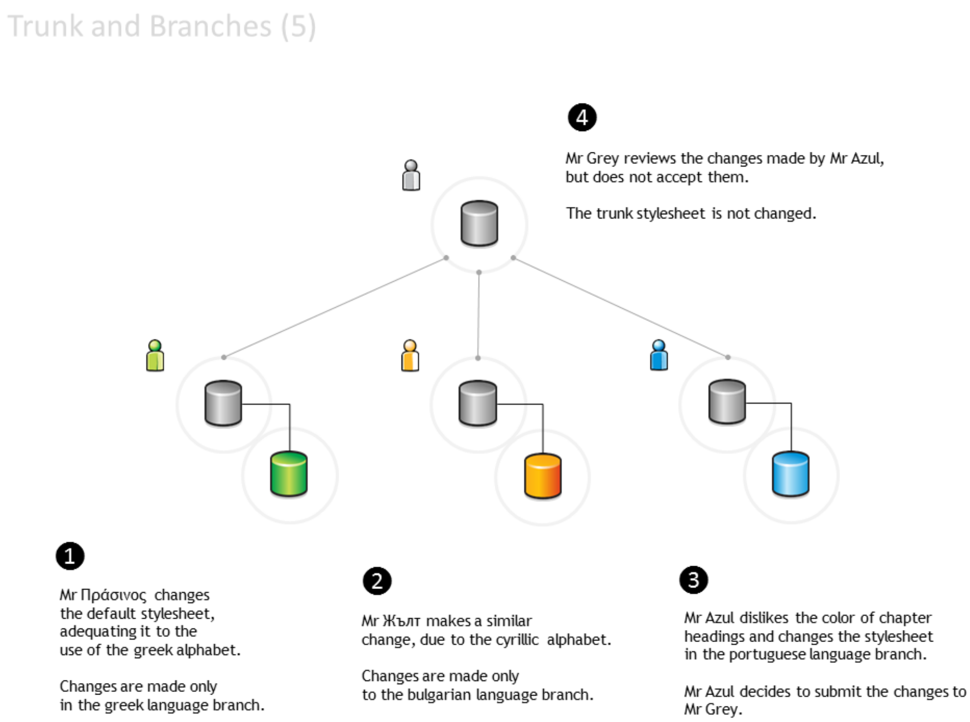
branch A set of files under version control may be branched (forked) at a point in time. From that time forward, the two copies of the files may develop in different ways, independently of each other.

change A change (or diff, or delta) represents a specific modification to a file under version control.

changeset A collection of files that have changes.

checkout See 'clone'.

clone To clone is to create a local working copy from the repository. A user may specify a revision or obtain the latest. In centralised version control systems (with a single central repository), the term 'checkout' is also used. The term 'checkout' can be used as a noun to describe the working copy.



commit To commit is to write or merge the changes made in the working copy back to the repository. The terms 'commit' and 'checkin' can also be used as nouns to describe the revision that is created as a result of committing.

conflict A conflict occurs when different parties make changes to the same file, and the system is unable to reconcile the changes.

A user must resolve the conflict by combining the changes, or by selecting one change in favour of the other.

fork See 'branch'.

head The most recent revision, either to the trunk or to a branch.

The trunk and each branch have their own head. HEAD is sometimes used to refer to the head of the trunk.

merge A merge is an operation in which two sets of changes are applied to a file or set of files under version control.

A user updates their working copy with changes made to the repository by other users.

A user tries to update a repository with changes made to a working copy.

repository The repository is where the files' current and historical data are stored, often on a server.

resolve The act of user intervention to address a conflict between different changes to the same file.

revision A revision (version) is any registered "snapshot" in time of the repository.

sync See 'update'.

trunk The trunk is the "main" line of development to the collection of information under version control, consisting only of 'baseline' (approved) files.

update An update (or sync) merges changes made in the original repository (by other users, for example) into the local working copy.

version See 'revision'.

working copy A working copy is a local copy of files from a repository, made at a specific time (revision).

Version Control using Git

Before you start

The following resources contain useful information on version control systems:

- **A Visual Guide to Version Control:** a simple explanation of version control with Subversion examples.
- A successful Git branching model: a clear and structured workflow.

Git CheatSheet

Source

Git CheatSheet,(c) 2011, salesforce.com, inc., URL: https://na1.salesforce.com/help/doc/en/salesforce_git_developer_cheatsheet.pdf

---

Overview When you first setup Git, set up your user name and email address so your first commits will record them properly:

```
git config --global user.name "My Name"
git config --global user.email "user@email.com"
```

Basic Git Workflow Example Initialise a new git repository, then stage all the files in the directory and finally commit the initial snapshot:

```
$ git init
$ git add .
$ git commit -m 'initial commit'
```

Create a new branch named feature_A, check it out so it is the active branch, then edit and stage some files and finally commit the new snapshot:

```
$ git branch feature_A
$ git checkout feature_A
$ (edit files)
$ git add (files)
$ git commit -m 'add feature A'
```

Switch back to the master branch, reverting the feature_A changes you just made, then edit some files and commit your new changes directly in the master branch context.:

```
$ git checkout master
$ (edit files)
$ git commit -a -m 'change files'
```

Merge the feature_A changes into the master branch context, combining all your work. Finally delete the feature_A branch.:

```
$ git merge feature_A
$ git branch -d feature_A
```

Setup & Init Git configuration, and repository initialisation & cloning.

command	description
`git config [key] [value]`	set a config value in this repository
`git config global [key] [value]`	set a config value globally for this user
`git init`	initialise an existing directory as a Git repository
`git clone [url]`	clone a Git repository from a URL
`git help [command]`	get help on any Git command

Stage & Snapshot Working with snapshots and the Git staging area.

command	description
`git status`	show the status of what is staged for your next commit and what is modified in your working directory
`git add [file]`	add a file as it looks now to your next commit (stage)
`git reset [file]`	reset the staging area for a file so the change is not in your next commit (unstage)
`git diff`	diff of what is changed but not staged
`git diff --staged`	diff of what is staged but not yet committed
`git commit`	commit your staged content as a new commit snapshot
`git rm [file]`	remove a file from your working directory and unstage

Branch & Merge Working with Git branches and with the stash.

command	description
`git branch`	list your branches. a * will appear next to the currently active branch
`git branch [branch-name]`	create a new branch at the current commit
`git checkout [branch]`	switch to another branch and check it out into your working directory
`git checkout -b [branch]`	create a branch and immediately switch to it
`git merge [branch]`	merge another branch into your currently active one and record the merge as a commit
`git log`	show commit logs
`git stash`	stash away the currently uncommitted modifications in your working directory temporarily
`git stash apply`	re-apply the last stashed changes

Share & Update Fetching, merging and working with updates from another repository.

command	description
`git remote add [alias] [url]`	add a git URL as an alias
`git fetch [alias]`	fetch down all the branches from that Git remote
`git merge [alias]/[branch]`	merge a branch on the server into your currently active branch to bring it up to date
`git push [alias] [branch]`	push the work on your branch to update that branch on the remote git repository
`git pull`	fetch from the URL tracked by the current branch and immediately try to merge in the tracked branch

Inspect & Compare Examining logs, diffs and object information.

command	description
`git log`	show the commit history for the currently active branch
`git log branchB..branchA`	show the commits on branchA that are not on branchB
`git log --follow [file]`	show the commits that changed file, even across renames
`git diff branchB...branchA`	show the diff of what is in branchA that is not in branchB
`git show [SHA]`	show any object in Git in human-readable format

Contributing on GitHub To contribute to a project that is hosted on GitHub (or another repository hosting site, such as BitBucket) you can fork the project online, then clone your fork locally, make a change, push back to GitHub and then send a pull request, which will email the maintainer.:

```
fork project on github
$ git clone https://github.com/my-user/project
$ cd project
$ (edit files)
$ git add (files)
$ git commit -m 'Explain what I changed'
$ git push origin master
go to github and click 'pull request' button
```

Visual Git Cheatsheet

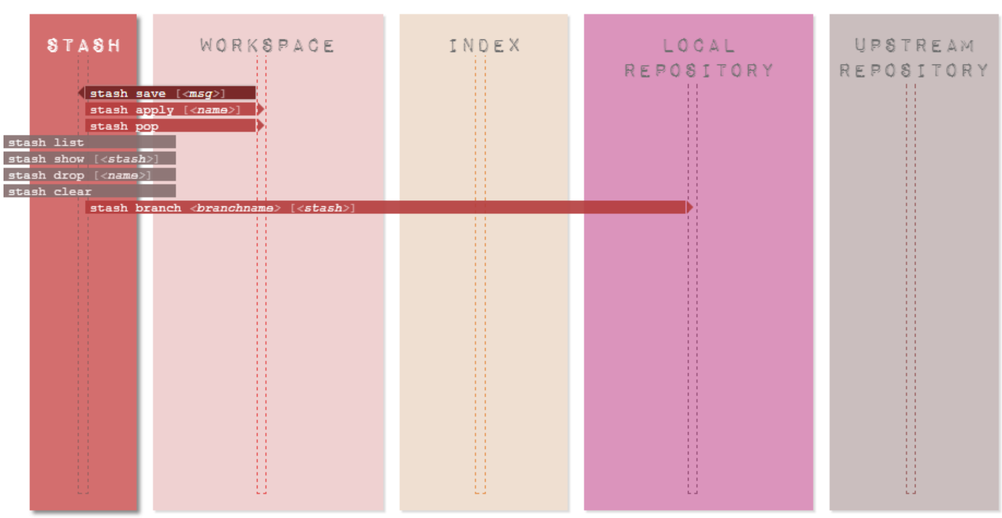
Source

Git Cheatsheet, (c) 2009-2012, Andrew Peterson url: http://ndpsoftware.com/git-cheatsheet.html

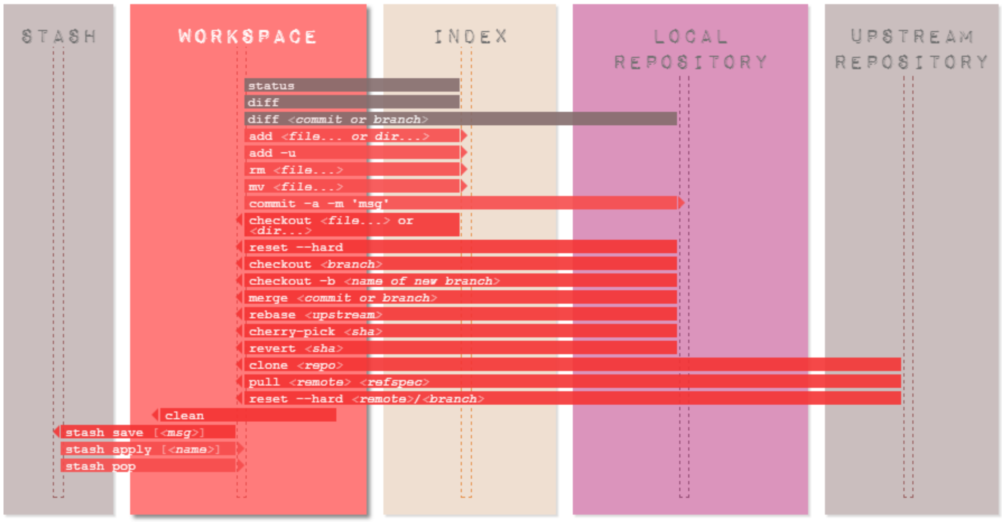
A list of Git commands, categorized on what they affect.

The interactive online version provides a description for each of the commands.

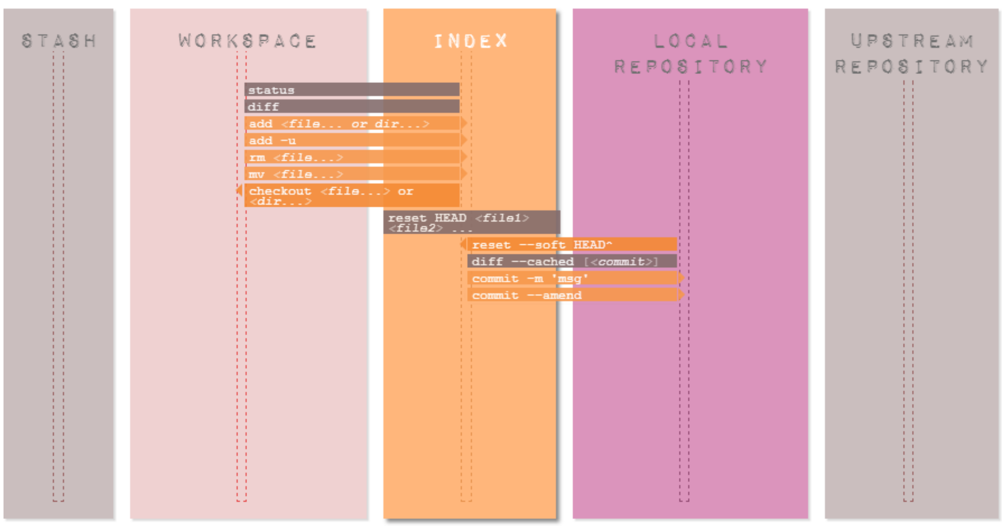
Stash A place to hide modifications made to the workspace, while working on something else. (The stash area is not required in a "normal" workflow.)



Workspace The local working area.

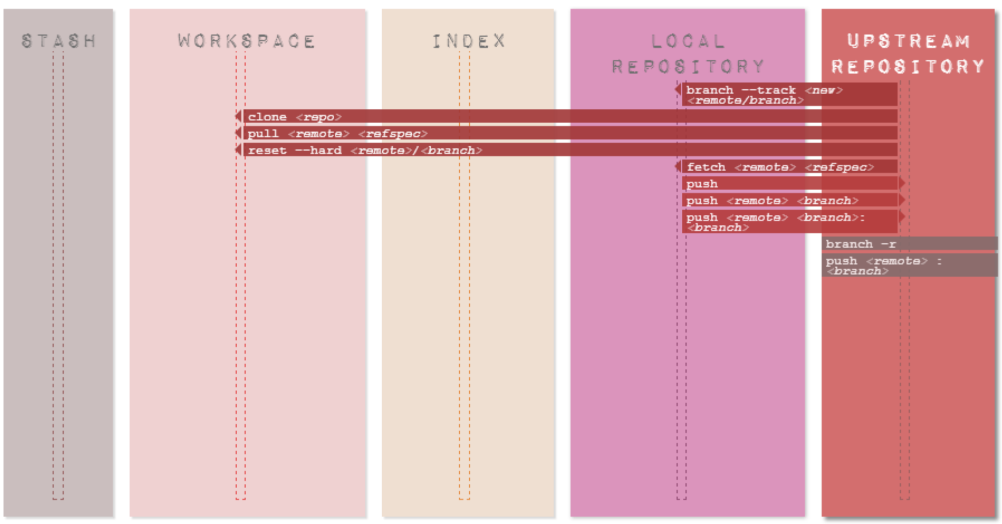


Staging area The "index"– or "staging area" – holds a snapshot of the content of the working area, and it is this snapshot that is taken as the contents of the next commit.



Local repository A local area under version control. Typical branches: master, dev (for local development), feature_x, bugfix_y

Upstream repository Typically a remote area under version control. Default name is 'origin'. Typical branches here: master, shared_feature_x, release_y.



How to...

This section include miscellaneous Git commands to perform different operations.

Set up a merge tool to resolve conflicts Configure kdiff3 as the merge tool (in Windows):

```
$ git config --global mergetool.kdiff3.path 'C:\Program Files (x86)\KDiff3\kdiff3.exe'
$ git config --global merge.tool kdiff3
```

Invoke kdiff3:

```
$  git mergetool <file>
```

Force an update from the upstream repository This operation will discard all changes in the local repository:

```
$ git reset --hard HEAD
$ git pull
```

Add untracked files to the set of files under version control A pattern can be used. For example, this will add any new or untracked `*.rst` file:

```
$ git add $(git ls-files --other *.rst)
```

Remove multiple files from the set of files under version control This will remove multiple files that have already been deleted from disk:

```
$ git rm $(git ls-files --deleted)
```

Alternatively, edit the `.git\config` file, and add the following lines:

```
[alias]
   rma = !git ls-files --deleted -z | xargs -0 git rm
```

Then run the command using the alias:

```
$git rma
```

Disable quoted file names Special character and spaces in file names can be problematic. To disable quotes file names (Windows Unicode Support), use:

```
$  git config [--global] core.quotepath off
```

Setting up an online Git Repository

Three possible alternatives are:

- Using Atlassian Bitbucket: it is free for public and private repositories (up to 5 team members), and also supports Mercurial repositories.
- Using GitHub: it is free for public repositories
- Using Dropbox : is is free up to a 2GB maximum storage.

Using Bitbucket

1. Create a Bitbucket account and a new repository "projectXPTO"
2. Install Git in your computer and set the global configuration (usig Git Bash):

```
$ git config --global user.name "johndoe"
$ git config --global user.email johndoe@example.com
```

3. Create a local git repository:

```
$ mkdir /path/to/your/project
$ cd /path/to/your/project
$ git init
```

4. Link the remote git repository to your local repository:

```
$ git remote add origin https://johndoe@bitbucket.org/johndoe/projectXPTO.git
```

5. Add a ReadMe file:

```
$ echo "# This is my README" >> README.md
$ git add README.md
```

6. Commit and push the first change:

```
$ git commit -m "First commit. Adding a README."
$ git push -u origin master
```

Using GitHub

The steps are similar to the ones when using Bitbucket...

1. Create a GitHub account and a new repository "projectXPTO"

(steps 2 and 3 as above)

4. Link the remote git repository to your local repository:

```
$  git remote add origin https://github.com/johndoe/projectXPTO.git
```

(steps 5 and 6 as above)

Using Dropbox

Please note that this is a more complicated solution, that is only useful if the Bitbucket or Github options cannot be used for some reason...

Dropbox is a cloud storage service provider. A Dropbox client application is available for Windows, Mac OSX, Linux and Android operating systems. The client application synchronises the content of a local Dropbox folder (in the client computer's disk) with the cloud Dropbox storage area.

A git repository is created in the local Dropbox folder and it will work if it were an "remote" upstream git repository.

Another local repository (located somewhere in the local disk, but **not** in the Dropbox folder) can then clone, push or sync with the Dropbox "remote" repository.

The rest is done automatically by the Dropbox application: the "remote" folder will be synced with online storage and will accessible from anywhere.

Setup the "remote" and the local repositories

1. Install both Git and the Dropbox client application on the computer.
2. Go to the local Dropbox folder and create a bare repository. Open a Git Bash window:

```
$ cd ~/Dropbox
$ mkdir -p remoteRepos/ProjectXPTO
$ git init -bare remoteRepos/ProjectXPTO
```

3. Go to the local project folder, and start a local git repository:

```
$ cd ~/localRepos/ProjectXPTO
$ git init .
$ git add .
$ git commit -all -m "Initial commit"
```

4. Link the local repository to the "remote" repository on the Dropbox folder:

```
$ git remote add dropbox /Dropbox/remoteRepos/ProjectXPTO/
```

5. Push all the local changes to the "remote" repository:

```
$ git push dropbox master
```

Clone the "remote" repository to a different machine

1. Again, both Git and the Dropbox application must be installed **and** the Dropbox folders must be synced.
2. Then, clone the "remote" repository with:

```
$ cd ~/otherMachine/ProjectXPTO
$ git clone -o dropbox /Dropbox/remoteRepos/ProjectXPTO/
```

Push changes to the "remote" repository

1. Changes to the local project can be pushed back to the "remote":

```
$ git commit -all -m "Changes made!"
$ git push dropbox master
```

Sync the local copy with the "remote" repository

1. To sync the local copy with the "remote" repository:

```
$ git pull dropbox master
```

Set up SSH for Git

Note: Sources

Mostly from: https://confluence.atlassian.com/display/BITBUCKET/Set+up+SSH+for+Git

Mixed with: https://help.github.com/categories/56/articles https://help.github.com/articles/working-with-ssh-key-passphrases http://nerderati.com/2011/03/17/simplify-your-life-with-an-ssh-config-file/

When you use HTTPS, you need to authenticate (supply a username and password) each time you take an action that communicates with the remote server. This page shows you how to use secure shell (SSH) to communicate with the Bitbucket or Github server and avoid having to manually type a password.

Step 1. Check if you have existing default Identity

The Git Bash shell comes with an SSH client. Do the following to verify your installation:

1. Double-click the Git Bash icon to start a terminal session.
2. Enter the following command to verify the SSH client is available:

```
$ ssh -v
OpenSSH_4.6p1, OpenSSL 0.9.8e 23 Feb 2007
usage: ssh [-1246AaCfgkMNnqsTtVvXxY] [-b bind_address] [-c cipher_spec]
[-D [bind_address:]port] [-e escape_char] [-F configfile]
[-i identity_file] [-L [bind_address:]port:host:hostport]
[-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port]
[-R [bind_address:]port:host:hostport] [-S ctl_path]
[-w local_tun[:remote_tun]] [user@]hostname [command]
```

3. If you have ssh installed, go to the next step.

   If you don't have ssh installed, install it now with your package manager.

4. List the contents of your ~/.ssh directory.

   If you have not used SSH on Bash you might see something like this:

```
$ ls -a ~/.ssh
ls: /c/Users/your-user-name/.ssh: No such file or directory
```

   If you have a default identity already, you'll see two id_* files:

```
$ ls -a ~/.ssh
.    ..    id_rsa    id_rsa.pub  known_hosts
```

   In this case, the default identity used RSA encryption (id_rsa.pub). If you want to use an existing default identity for your Bitbucket account, skip the next section and go to create a config file.

Step 2. Set up your default identity

By default, the system adds keys for all identities to the /Users/your-user-name/.ssh directory. The following procedure creates a default identity.

1. Open a terminal in your local system. Enter ssh-keygen at the command line:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key:
```

   To create a key with a name other than the default, specify the full path to the key. Enter and renter a passphrase when prompted. Unless you need a key for a process such as script, you should always provide a passphrase. The command creates your default identity with its public and private keys.

2. List the contents of ~/.ssh to view the key files. You should see something like the following:

```
$ ls ~/.ssh
id_rsa  id_rsa.pub
```

   The command created two files, one for the public key (for example id_rsa.pub) and one for the private key (for example, id_rsa).

Step 3. Create a SSH config file

1. Using a text editor, edit the ~/.ssh/config file. Add the following entries to the configuration file using the following format:

```
Host bitbucket.org
 IdentityFile ~/.ssh/id_rsa

Host github.com
 IdentityFile ~/.ssh/id_rsa
```

Every second line is indented. That indentation (a single space) is important, so make sure you include it. The second line is the location of your private key file.

2. Save and close the file.
3. Restart the GitBash terminal.

Step 4. Update your .bashrc profile file

It is a good idea to configure your GitBash shell to automatically start the agent when launch the shell. The .bashrc file is the shell initialization file. To start the agent automatically, do the following.

1. Start GitBash.
2. Edit your ~/.bashrc file.

   Add the following lines to the file:

```
SSH_ENV=$HOME/.ssh/environment

# start the ssh-agent
function start_agent {
    echo "Initializing new SSH agent..."
    # spawn ssh-agent
    /usr/bin/ssh-agent | sed 's/^echo/#echo/' > "${SSH_ENV}"
    echo succeeded
    chmod 600 "${SSH_ENV}"
    . "${SSH_ENV}" > /dev/null
    /usr/bin/ssh-add
}

if [ -f "${SSH_ENV}" ]; then
     . "${SSH_ENV}" > /dev/null
     ps -ef | grep ${SSH_AGENT_PID} | grep ssh-agent$ > /dev/null || {
        start_agent;
    }
else
    start_agent;
fi
```

3. Save and close the file.
4. Restart the GitBash terminal.
5. The system prompts you for your passphrase.
6. Enter your passphrase. After accepting your passphrase, the system displays the command shell prompt. Verify that the script identity added your identity successfully by querying the SSH agent:

```
$ ssh-add -l
```

After you install your public key to Bitbucket|Github, having this script should prevent you from having to enter a password each time you push or pull a repository from Bitbucket.

Step 5. Install the public key on your Bitbucket|Github account

In Bitbucket:

1. Open a browser and log into Bitbucket.
2. Choose avatar > Manage Account from the menu bar.
3. The system displays the Account settings page. Click SSH keys. The SSH Keys page displays. It shows a list of any existing keys. Then, below that, a dialog for labeling and entering a new key.

   Copy the contents of the public key file into the SSH Key field. Click the Add key button. The system adds the key to your account.

In Github:

1. Goto to the account settings, everything is pretty much as above.

Return to the GitBash terminal window:

1. Verify your configuration by entering the following commands:

```
ssh -T git@bitbucket.org

ssh -T git@github.com
```

   The command message tells you which Bitbucket account can log in with that key. Verify that the command returns your account name.

Step 6. Configure your repository to use the SSH protocol

The URL you use for a repository depends on which protocol you are using, HTTPS and SSH.

In Bitbucket:

- ssh://git@bitbucket.org/accountname/reponame.git
- https://accountname@bitbucket.org/accountname/reponame.git

The same goes for Github:

```
*  ssh://git@github.com/accountname/reponame.git

*  https://accountname@github.com/accountname/reponame.git
```

So...

1. View your current repository configuration file *.git/config*, that should similar to this:

```
[remote "origin"]
  fetch = +refs/heads/*:refs/remotes/origin/*
  url = https://accountname@domain/accountname/reponame.git
[branch "master"]
  remote = origin
  merge = refs/heads/master
```

2. Change the url:

```
[remote "origin"]
  fetch = +refs/heads/*:refs/remotes/origin/*
  url = ssh://git@domain/accountname/reponame.git
[branch "master"]
```

```
    remote = origin
    merge = refs/heads/master
```

3. Save your edits and close the file.

CHAPTER 2

Notebook

Searchable miscellanea...

2.1 Notebook

2.1.1 Introduction

Just my chaotic notes about tools...

2.1.2 References

Installing Python on Windows

Note: This page was modified from The Hitchhiker's Guide to Python.

Download the latest version of Python 2.7 from the official Website. If you want to be sure you are installing a fully up-to-date version then use the "Windows Installer" link from the home page of the Python.org web site .

The Windows version is provided as an MSI package. To install it manually, just double-click the file. The MSI package format allows Windows administrators to automate installation with their standard tools.

By design, Python installs to a directory with the version number embedded, e.g. Python version 2.7 will install at `C:\Python27\`, so that you can have multiple versions of Python on the same system without conflicts. Of course, only one interpreter can be the default application for Python file types. It also does not automatically modify the `PATH` environment variable, so that you always have control over which copy of Python is run.

Typing the full path name for a Python interpreter each time quickly gets tedious, so add the directories for your default Python version to the `PATH`.

Assuming that your Python installation is in `C:\Python27\`, add this to your `PATH`:

```
C:\Python27\;C:\Python27\Scripts\
```

You can do this easily by running the following in `powershell`:

```
[Environment]::SetEnvironmentVariable("Path", "$env:Path;C:\Python27\;C:\Python27\Scripts\", "User")
```

The second (`Scripts`) directory receives command files when certain packages are installed, so it is a very useful addition.

You do not need to install or configure anything else to use Python. Having said that, I would strongly recommend that you install the tools and libraries described in the next section before you start building Python applications for real-world use. In particular, you should always install Setuptools, as it makes it much easier for you to use other third-party Python libraries.

Setuptools + Pip

The most crucial third-party Python software of all is Setuptools, which extends the packaging and installation facilities provided by the distutils in the standard library. Once you add Setuptools to your Python system you can download and install any compliant Python software product with a single command. It also enables you to add this network installation capability to your own Python software with very little work.

To obtain the latest version of Setuptools for Windows, run the Python script available here: ez_setup.py

You'll now have a new command available to you: **easy_install**. It is considered by many to be deprecated, so we will install its replacement: **pip**. Pip allows for uninstallation of packages, and is actively maintained, unlike easy_install.

To install pip, run the Python script available here: get-pip.py

Virtualenv

After Setuptools & Pip, the next development tool that you should install is virtualenv. Use pip

```
> pip install virtualenv
```

The virtualenv kit provides the ability to create virtual Python environments that do not interfere with either each other, or the main Python installation. If you install virtualenv before you begin coding then you can get into the habit of using it to create completely clean Python environments for each project. This is particularly important for Web development, where each framework and application will have many dependencies.

To set up a new Python environment, change the working directory to wherever you want to store the environment, and run the virtualenv utility in your project's directory

```
> virtualenv venv
```

To use an environment, run the `activate.bat` batch file in the `Scripts` subdirectory of that environment. Your command prompt will change to show the active environment. Once you have finished working in the current virtual environment, run the `deactivate.bat` batch file to restore your settings to normal.

Each new environment automatically includes a copy of `pip` in the `Scripts` subdirectory, so that you can setup the third-party libraries and tools that you want to use in that environment. Put your own code within a subdirectory of the environment, however you wish. When you no longer need a particular environment, simply copy your code out of it, and then delete the main directory for the environment.

Eclipse

> The Eclipse Platform is a generic foundation for an IDE That is, the platform is an IDE without any particular programming language in mind. You can create generic projects, edit files in a generic text editor, and share the projects and files with a version control system. The platform is essentially a glorified version of a file-system browser.
>
> [from http://www.ohloh.net/p/eclipse]

The current version is Eclipse 4.3 Kepler.

The basic platform is indeed "a glorified version of a file-system browser". All functionality is provided through plug-ins.

In Eclipse Kepler, the following two plug-ins are already incorporated in the base product (installation is no longer required):

- Mylyn, the task-focused interface for Eclipse;
- Egit, the Eclipse Team provider for the Git version control system.

Numerous other plug-ins are available. In some cases, packages are provided that bundle together a number of useful plug-ins for a specific purpose.

For example, the Eclipse Java EE IDE for Web Developers also includes the Web Tool Platform (that will be required for XSD and XML creation and validation, CSS editing, etc.). It is the selected option.

The StatET http://www.walware.de/goto/statet

Other plug-ins may be required, and are described below...

Mylyn

Mylyn (http://www.eclipse.org/mylyn/).

Install Mylyn Connectors Mylyn can use a local **task** repository or a remote one.

If the remote task repository is associated with an issue tracking system, a 'connector' is required. By default, a Bugzilla connector is included with Mylyn (and Eclipse).

A long list of different connectors is available at http://wiki.eclipse.org/Mylyn/Extensions. It includes connectors for Trac and Redmine, GitHub and Bitbucket, etc...

Connectors can be installed in different ways:

1. Some (stable) connectors are available through the Mylyn Task List window:

   - Add repository > Install more connectors...

   For example, the Trac connector is available in this list.

2. Other connectors (alpha versions, etc) can be installed using the standard plug-in install procedure inside Eclipse:

   - Goto Help > Install new software...
   - Providing the link to the update site (available in the connector description in the Mylyn/Extensions wiki page).

   This is the case for:

   - The Bitbucket Mylyn Connector, which is alpha status and is available at http://www.mylynbitbucketconnector.xpg.com.br/update
   - The GitHub Connector (egit-github), also in alpha status and available at http://download.eclipse.org/egit/github/updates-nightly

Specifically for my projects, two connectors are required:

- The Trac connector
- The Bitbucket connector (currently 2013.10.21 not working properly due to an identified but not yet fixed bug)

How to... A basic tutorial on Mylyn is available at: http://www.vogella.com/articles/Mylyn/article.html.

A generic introduction is available at: http://www.youtube.com/watch?v=bSYVpjom4pU

Use tags in code comments to generate tasks Tags in source code comments can be used to generate tasks. The following Window > Preferences can be enabled:

- General > Structured Text Editors > Task Tags

  Enable searching for Task Tags

- Java > Compiler > Task Tags
- JavaScript > Validator > Task Tags
- PyDev > Task Tags
- StatET > Task Tags

  The Tasks view includes a helpful customization for Java developers. When a Java project is built, the parser automatically scans for Java task tags in your code comments. You can configure the task tag names and their priorities using the Java > Task Tags preferences. Three tags are provided by default (FIXME, TODO, and XXX), and we added a STORY tag to support our agile development process.

  Eclipse Distilled, by David Carlson

Workarounds

If the task list disappears... The workaround is:

1. Goto the drop-down menu in the Task List pane (or right-click to see the contextual menu).
2. Select Restore tasks from history...
3. Select either a zip file (if you've exported the task list before) or an adequate snapshot.

Tasks can be exported from the Task list pane (right-click to see the contextual menu) using the Import and Export... > Export option.

This apparently can occur when updating and restarting Eclipse (see https://bugs.eclipse.org/bugs/show_bug.cgi?id=403467).

Git integration

How to

Import the content of an existing git repository

1. Open the Git Repository Exploring perspective
2. If the repository isn't already listed, then Add a repository (using the drop-down menu)
3. Select the repository in the list, right-click and select Import projects...
4. Choose Import as a general project... and follow the wizard.

SVN integration

See http://www.eclipse.org/subversive/

PlantUML

PlantUML is a component that allows to quickly write :

- sequence diagram,
- use case diagram,
- class diagram,
- activity diagram,
- component diagram,
- state diagram
- object diagram

Diagrams are defined using a simple and intuitive language. The documentation is available here:

```
http://sourceforge.net/projects/plantuml/files/PlantUML%20Language%20Reference%20Guide.pdf/download
```

Images can be generated in PNG or SVG format.

The Eclipse plug-in is described here:

```
http://plantuml.sourceforge.net/eclipse.html
```

Install It is not clear if the PlantUML plug-in works with the Eclipse Kepler version (4.3).

The update site for Eclipse Juno (4.2) is:

```
http://plantuml.sourceforge.net/updatesitejuno/
```

Let's try it. It's working!

Note that the Graphviz software must be installed.

How to...

1. Goto Window > Show View > Other > PlantUML to open a visualisation tab.
2. Insert the following text into a document (or inside a multiline code comment):

```
@startuml

   user -> (use PlantUML)

   note left of user
      Hello!
   end note

@enduml
```

3. The diagram will be displayed the the PlantUML visualisation pane, where it can be exported to a graphic file.

Papyrus

Papyrus is graphical editing tool for UML2 as defined by OMG.

It can be used as a simple plug-in or as a part of the Eclipse Modelling Tools package. It provides a graphical editor for the Eclipse UML2 project.

> UML2 is an EMF-based implementation of the Unified Modeling Language (UML) 2.x OMG metamodel for the Eclipse platform.
>
> The objectives of the UML2 component are: * to provide a usable implementation of the UML metamodel to support the development of modeling tools * a common XMI schema to facilitate interchange of semantic models * test cases as a means of validating the specification * validation rules as a means of defining and enforcing levels of compliance
>
> Although MDT/UML2 provides the metamodel, it does not provide UML modelling tools themselves. One implementation is Papyrus. An older, no longer supported implementation is UML2Tools (http://wiki.eclipse.org/MDT-UML2Tools).
>
> [http://wiki.eclipse.org/MDT-UML2Tools]

Install

1. Start Eclipse
2. Goto Help > Install New Software
3. Press Add... to add a new resource and specify a name and the URL (the link below is for the Eclipse Kepler version):

```
NAME: Papyrus
URL:  http://download.eclipse.org/modeling/mdt/papyrus/updates/releases/kepler
```

How to Tutorial on the Eclipse Modelling Framework (not on Papyrus, but it will be useful later on): http://www.vogella.com/articles/EclipseEMF/article.html

Python IDE

PyDev is a Python IDE for Eclipse, which may be used in Python, Jython and IronPython development.

Install Python 2.7 is assumed to be installed.

1. Start Eclipse
2. Goto Help > Install New Software
3. Press Add... to add a new resource and specify a name and the URL:

```
NAME: PyDevEnv
URL:  http://pydev.org/updates
```

4. Select PyDev and PyDev Mylyn Integration from the list and press Next.
5. Acept the licence terms when the download ends.
6. Restart Eclipse.

Configure

1. Goto Window > Preferences > PyDev > Editor > Interpreter Python
2. Eclipse can configure the options automatically (press Auto Config) or the location of the Python interpreter can be specified. (in Linux, usually that would be /usr/bin/python)
3. Press OK to finish the configuration.

Start new project

1. Goto File > New > Project and select 'Pydev project'
2. Create a new file (goto File > New > File) HelloWorld.py
3. Add the code
4. Press Run or `Ctrl + F11`

Python projects will be associated with a "Python perspective", i.e. a customised layout of windows and GUI elements.

References: 1 ; 2 .

How to... A generic tutorial covering all the above steps is available at: http://www.vogella.com/articles/Python/article.html

ReST Editor

ReST Editor is an Eclipse plug-in providing support to edit reStructuredText files

reStructuredText is a markup language that can be transformed in various output formats with tools like the Sphinx documentation generator, rst2pdf, rst2beamer, ...

More information here : http://resteditor.sourceforge.net/

Install This plug-in can be installed through the Eclipse Marketplace (Help > Eclipse Marketplace...) or through the standard plug-in installation:

1. Goto Help > Install new software...
2. Add the project update site: http://resteditor.sourceforge.net/eclipse
3. Select the ReST Editor plug-in

Configure The plug-in is configured under Window > Preferences > ReST Editor.

The following options are important:

- The preferred section markers order, that will be used to automatically correct any improper sequence: #*=-^"
- The tab length (3) and the option to insert spaces instead of tabs.
- The spell checking options (see *Hunspell4Eclipse*).

Start a new Sphinx project The ReST Editor plug-in can be used to create a new Sphinx project:

1. Goto File > New > Project > ReST Editor > Sphinx project
2. Follow the wizard's instructions...

If Sphinx is installed, then ReST documents can be built from within Eclipse (using the make.bat or the makefile).

Hunspell4Eclipse

Hunspell4Eclipse is a plug-in that integrates Hunspell into Eclipse's Spell Checking Service. It is useful if Eclipse is used as for general purpose document editing.

Install This plug-in can be installed through the Eclipse Marketplace (Help > Eclipse Marketplace...) or through the standard plug-in installation procedure.

Configure No dictionaries are included. The plug-in uses Hunspell or Myspell dictionaries. These are also used by LibreOffice, and are available in the extensions directory (for example, "C:Program Files (x86)LibreOffice 4.0share-extensions").

Preferences per workspace can be configured in: 1. Preferences - General - Editors > TextEditors > Spelling

1. Select Hunspell4Eclipse
2. Browse... and select a dictionary(.dic) file

Useful links:

- https://code.google.com/p/hunspell4eclipse/
- https://wiki.mozilla.org/L10n:Dictionaries

Build a Sphinx project

- Right-click on the make.bat file and goto Run as > Run Configurations...
- Select the option `Sphinx (via make file)`
- Create a new configuration: * Specify the working directory, for example `${project_loc}/docs` * Specify the type of Sphinx output, for example `html`

The new configuration will be accessible through the button bar (in the Run button drop-down options).

The console pane will show the Sphinx output.

StatET for R

Install Dependencies (for the stable version StatET 3.3 in Eclipse 4.3):

Java	6 or higher
GNU R	2.13 to 3.0
R package RJ	1.1

For Windows users

The path where R is installed should not contain spaces. For example, install in "C:ProgramsR" but not in "C:Programs FilesR".

Otherwise, strange things are likely to happen, such as packages not getting installed properly...

To install the R Packages of RJ 1.1 (StatET 3.0-3.3), use the following command in a common R Term console:

```
install.packages(c("rj", "rj.gd"), repos="http://download.walware.de/rj-1.1")
```

In Eclipse, use the standard plug-in installation procedure:

1. Goto Help > Install New Software
2. Press Add... to add a new resource and specify a name and the URL:

```
NAME: StatET
URL:  http://download.walware.de/eclipse-4.3
```

3. For most users it is recommend to select only StatET (and Add-ons/Utilities, if desired), but no Libraries; the dependencies are resolved automatically.

Tutorials

- Eclipse and StatET – a working environment for R
- A guide to Eclipse and the R plug-in StatET

Toad Extension for Eclipse 1.9.0 Community Edition

Eclipse: how to...

Change the encoding to UTF-8

- For a specific project: File > Properties > Text file encoding
- Goto Window > Preferences > General > Content types and change the *Default encoding* for each type.
- Goto Window > Preferences > General > Workspaces > Text file encoding

Show a print margin

- Goto Window > Preferences > General > Editors > Text Editors > Show print margin (80)

Using GIST

1. Open the Task List
2. Goto pane menu and select Show Task Repositories View
3. Goto pane menu and select Add Task Repository...
4. Select the appropriate connector, which is GitHub Gists
5. and follow the wizard.

2.2 Using the issue tracking system

2.2.1 Introduction

This section contains information about Redmine, the project management and issue tracking system adopted for the current project.

2.2.2 References

Redmine instalation

Preamble

Presently, the BitNami Redmine Stack virtual machine (VM) is being used. The following notes thus apply to the deployment of this VM using Oracle VirtualBox, and not specifically to a Redmine step-by-step installation <http://www.redmine.org/projects/redmine/wiki/RedmineInstall>'_.

The notes below are applicable to a test environment (e.g. in a personal computer) and are not necessarily adequate or recommended for deployment in a production environment.

Note also that the Bitnami Redmine Stack ships with MySql as a database back-end.

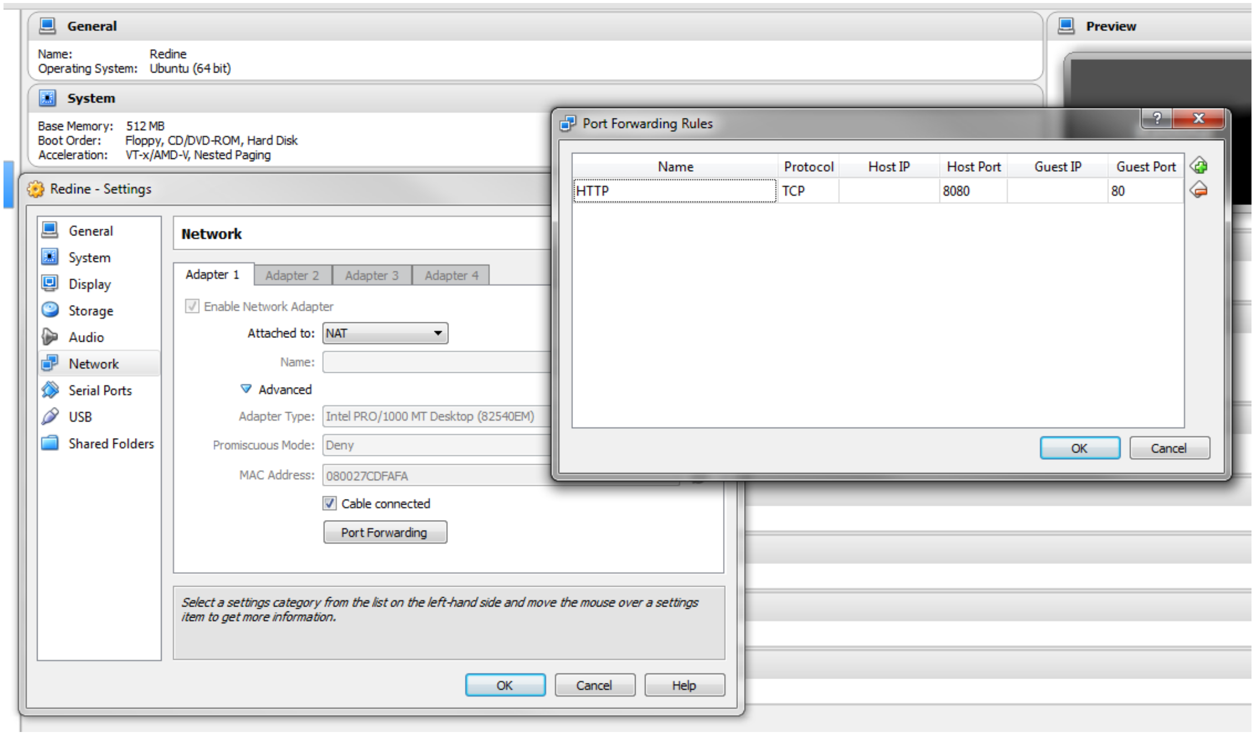
Detailed information can be found at http://wiki.bitnami.com/Applications/BitNami_Redmine.

Deployment

1. Update Oracle VirtualBox if necessary. Install the most recent VirtualBox Extension Pack.
2. Download the VM from http://bitnami.com/stack/limesurvey. Unzip...
3. Create a new VirtualBox VM
   - Guest OS: Ubuntu 64-bit
   - RAM: at least 512 MB RAM
   - Storage: select the VMDK file
   - Network: Enable NAT

   After everything is configured, the network adapter will be changed to Bridged Adaptor (had some problems with the Bridged Adaptor over WiFi, hence this choice).

   NAT can also be used if the Guest is to be accessed only from the Host. Port forwarding was setup as depicted below:



1. Start the VM. Prepare to install the Virtual Box Guest Additions (Devices –> Install Guest Additions)
2. The default Linux login is `bitnami/bitnami`. In real life, the default users and passwords should be changed.
3. Enable root:

```
$  sudo passwd root
```

   In real life, the `root` password should **not** be enabled. (Instead of entering as root, use `sudo` instead).

4. Change keyboard layout if required:

```
$  sudo apt-get update
$  sudo apt-get install console-data
$  sudo dpkg-reconfigure keyboard-configuration
$  sudo dpkg-reconfigure console-setup
```

   If later reconfiguration is required:

```
$  sudo dpkg-reconfigure console-data
```

5. Install the VirtualBox Guest Additions:

```
$  sudo mount /dev/cdrom /mnt
$  cd /mnt
$  sudo ./VBoxLinuxAdditions.run
$  sudo reboot
```

   If the build fails, check the log file:

```
$ nano /var/log/vboxadd-install.log
```

   (Note that there is always be a fail message for windows system if no graphic interface is installed in the server):

```
$  sudo apt-get update
$  sudo apt-get install dkms              #if required
$  sudo apt-get install build-essential   #if required
$  sudo apt-get linux-headers-generic     # and/or
$  sudo apt-get linux-headers-3.2.0-53-virtual # for example, if such is the version required...
```

6. How to access the BitNami Virtual Appliance?

If the network as been set to NAT and port forwarding has been configured as specified above, then the application can be accessed at http://localhost:8080/redmine .

The default application login information is `user/bitnami`

Further information is available here: http://bitnami.com/faq/virtual_machines

Backups

Redmine backups should include:

- data (stored in your redmine database)
- attachments (stored in the files directory of your Redmine install)

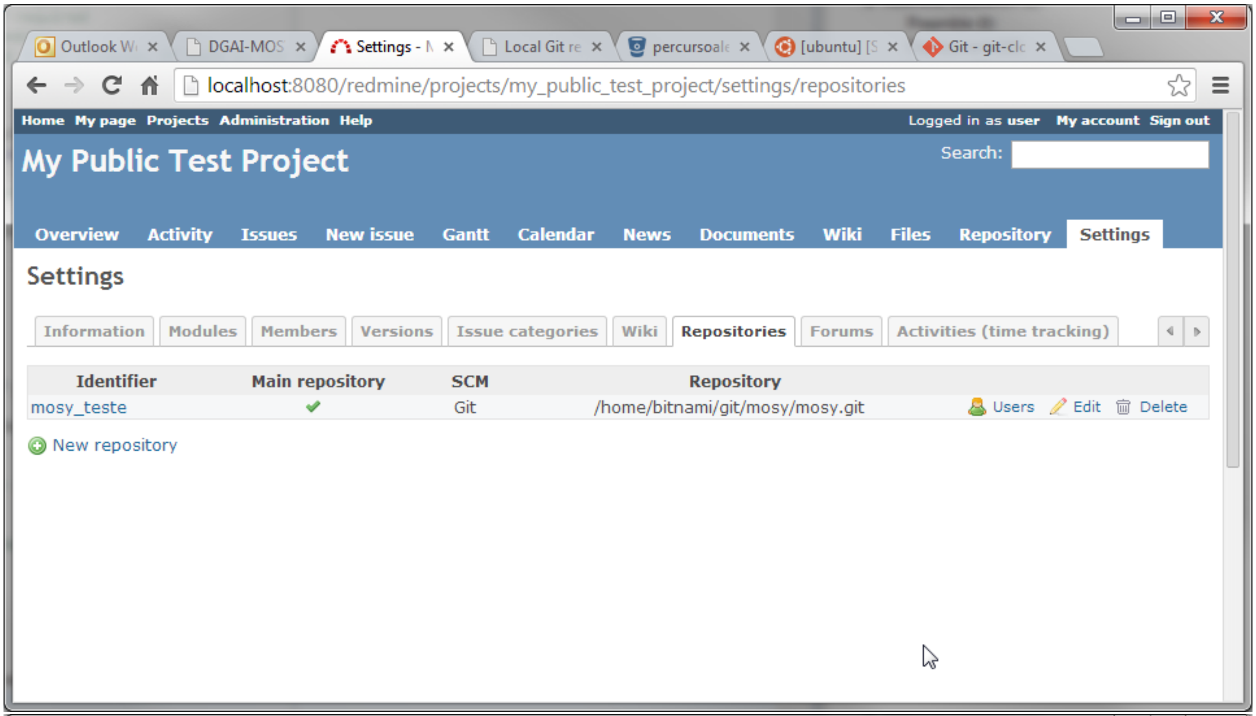
Here is a simple shell script that can be used for daily backups (assuming you're using a mysql database):

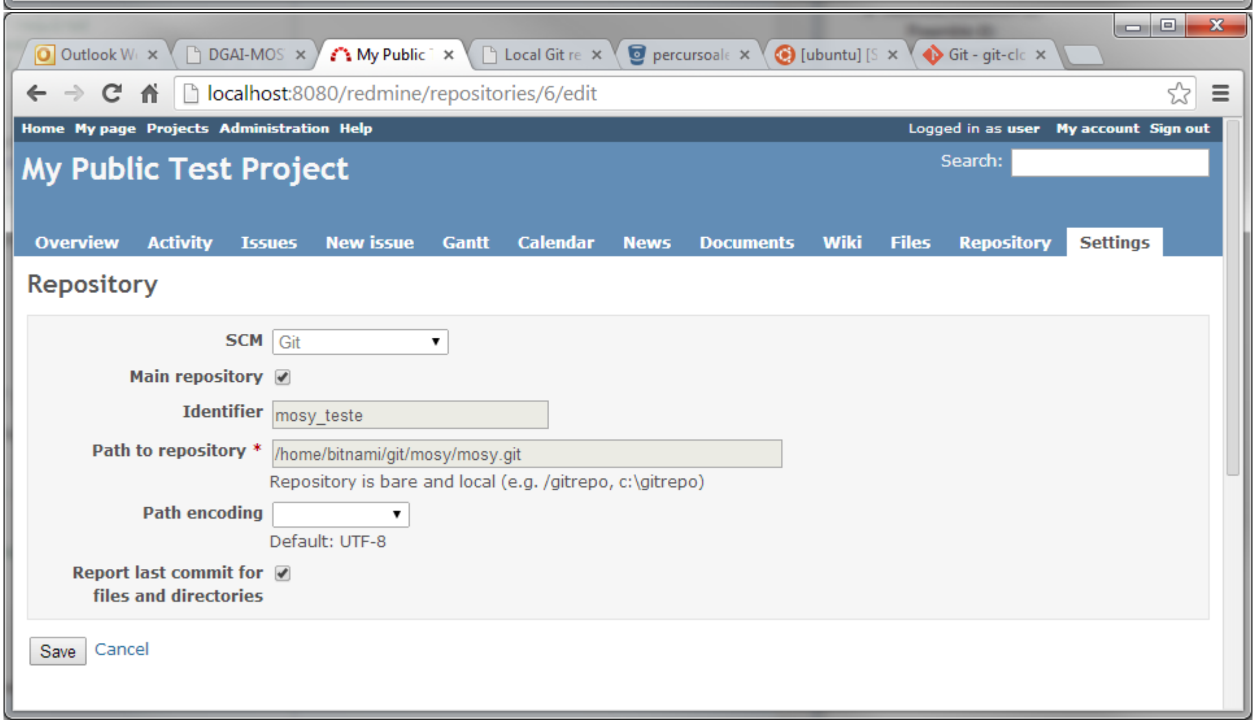
```
# Database
/usr/bin/mysqldump -u <username> -p<password> <redmine_database> | gzip > /path/to/backup/db/redmine_

# Attachments
rsync -a /path/to/redmine/files /path/to/backup/files
```

Repositories

The Bitnami Redmine Stack already includes Git.

Access to a local git repository can be configured by the project's manager at the project settings tab.





Note that the repository should be local and bare. For example, to clone from a remote repository, use:

```
git clone --bare git://yourgitserver.org/project.git
```

and add full path to the repository in Projects > Settings > Repositories e.g. `/var/repositories/project.git`.

Permissions on the repository folder for the redmine user:group should be are 775.

To update the bare repository with changes from its remote working origin do:

```
git fetch -q origin master:master
```

or push the changes into the bare repository:

```
git push --all <url-of-bare-repo>
```

Note: See

http://www.redmine.org/boards/2/topics/34226?r=35533 http://www.saintsjd.com/2011/01/what-is-a-bare-git-repository/