Martin Mundt, Tobias Weis

Pattern Analysis & Machine Intelligence Praktikum: MLPR-19

Week 1: Versioning and documentation

Versioning – what is it and why should we use it?

- Keeping track and documenting changes to your code and / documents

- Reasons to use versioning:
  - Troubleshooting and backtracking
  - Logging
  - Remote backup
  - Simplified addition of features
  - Sharing with a team

Versioning – what is it and why should we use it?

- Works by having a copy of your code on a server as a master-copy, changes can be sent or received.

- There exists a huge ecosystem of different version control systems, most common are:
  - Git
  - Mercurial
  - CVS (concurrent versions systems)
  - SVN (subversion)

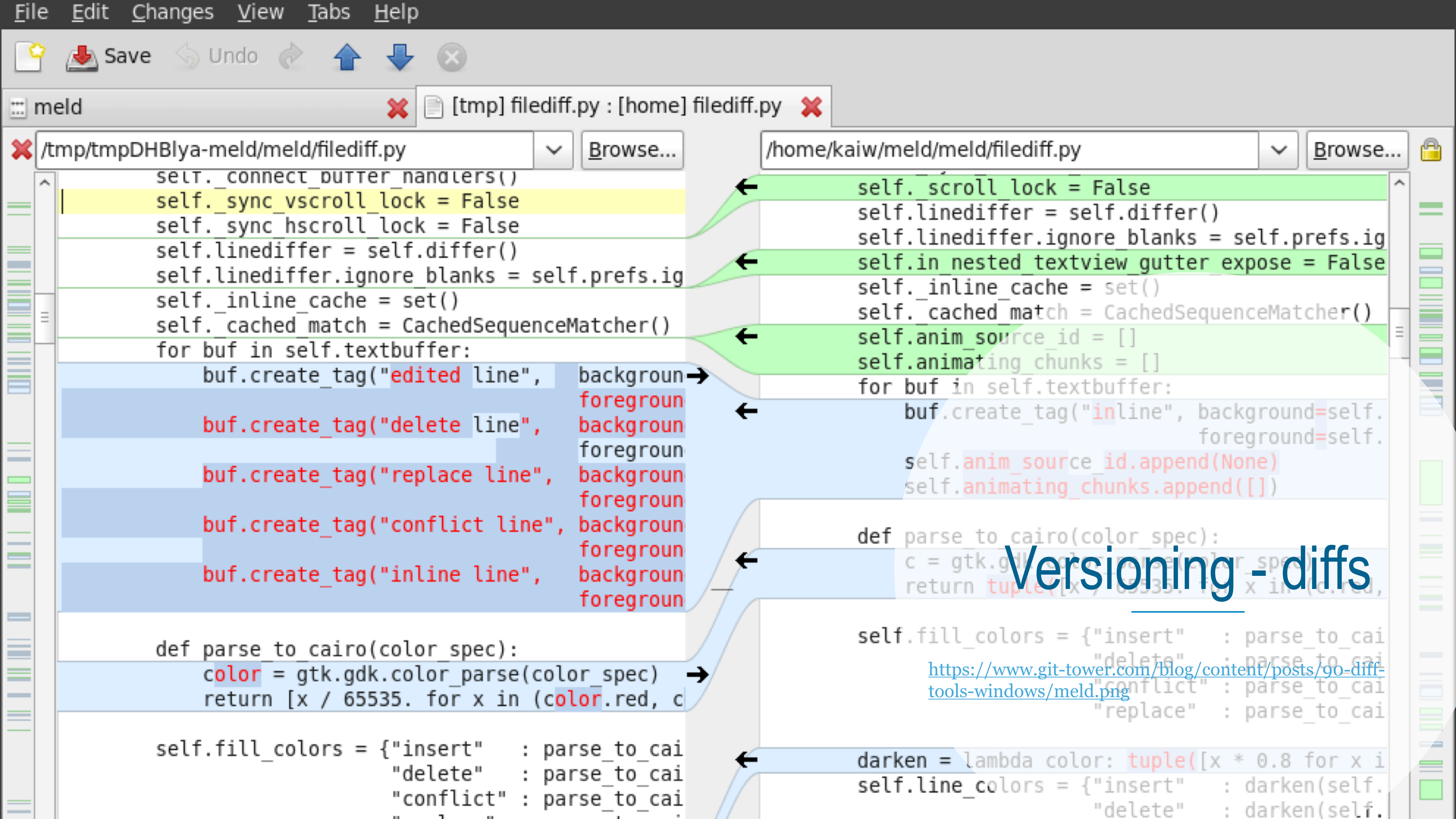
- Git and Mercurial are distributed: the server has a master-repository, but there is a local repository on the user's machine

- SVN and CVS are non-distributed and all commits go to the server

Versioning – what is it and why should we use it?

This enables:

- Keeping track of changes to files and folders
- Teamwork on code with multiple authors in parallel
- A record of who did what and when
- The „why“ is provided by commit messages
- Release of stable versions while working on new features


File Edit Changes View Tabs Help
Save
Undo
meld
[tmp] filediff.py : [home] filediff.py
/tmp/tmpDHBlya-meld/meld/filediff.py
Browse...
/home/kaiw/meld/meld/filediff.py
Browse...
self._connect_buffer_handlers()
self._sync_vscroll_lock = False
self._sync_hscroll_lock = False
self.linediffer = self.differ()
self.linediffer.ignore_blanks = self.prefs.ig
self._inline_cache = set()
self._cached_match = CachedSequenceMatcher()
for buf in self.textbuffer:
buf.create_tag("edited line", backgroun
foregroun
buf.create_tag("delete line", backgroun
foregroun
buf.create_tag("replace line", backgroun
foregroun
buf.create_tag("conflict line", backgroun
foregroun
buf.create_tag("inline line", backgroun
foregroun
def parse_to_cairo(color_spec):
color = gtk.gdk.color_parse(color_spec)
return [x / 65535. for x in (color.red, c
self.fill_colors = {"insert" : parse_to_cai
"delete" : parse_to_cai
"conflict" : parse_to_cai
self._scroll_lock = False
self.linediffer = self.differ()
self.linediffer.ignore_blanks = self.prefs.ig
self.in_nested_textview_gutter_expose = False
self._inline_cache = set()
self._cached_match = CachedSequenceMatcher()
self.anim_source_id = []
self.animating_chunks = []
for buf in self.textbuffer:
buf.create_tag("inline", background=self.
foreground=self.
self.anim_source_id.append(None)
self.animating_chunks.append([])
def parse_to_cairo(color_spec):
return tuple([x / 65535. for x in (c.red,
self.fill_colors = {"insert" : parse_to_cai
"delete" : parse_to_cai
"conflict" : parse_to_cai
"replace" : parse_to_cai
darken = lambda color: tuple([x * 0.8 for x i
self.line_colors = {"insert" : darken(self.
"delete" : darken(self.
Versioning - diffs
https://www.git-tower.com/blog/content/posts/90-diff-tools-windows/meld.png

Git

Open source version control system designed for speed and efficiency

Best insurance against

- Accidental mistakes (like deleting files)
- Remember what has been changed (and why)
- Hard-drive-failures, fire, earthquakes

Created by Linus Torvalds (for managing Linux kernel)

„I‘m an egoistical bastard, and I name all my projects after myself. First ‚Linux‘, now ‚Git‘“
Linus Torvalds

Git

Server

Getting a Git-Server:

- Hosted solutions include github.com (widespread, public repos free), bitbucket.com
- Self-hosted gitserver: gitlab

Client

Getting the Git client:

- Windows: Download from http://git-scm.com/downloads
- Mac: brew install git
- Linux: sudo apt-get install git

Git – why distributed?

Use GIT on your local machine without being connected to the internet

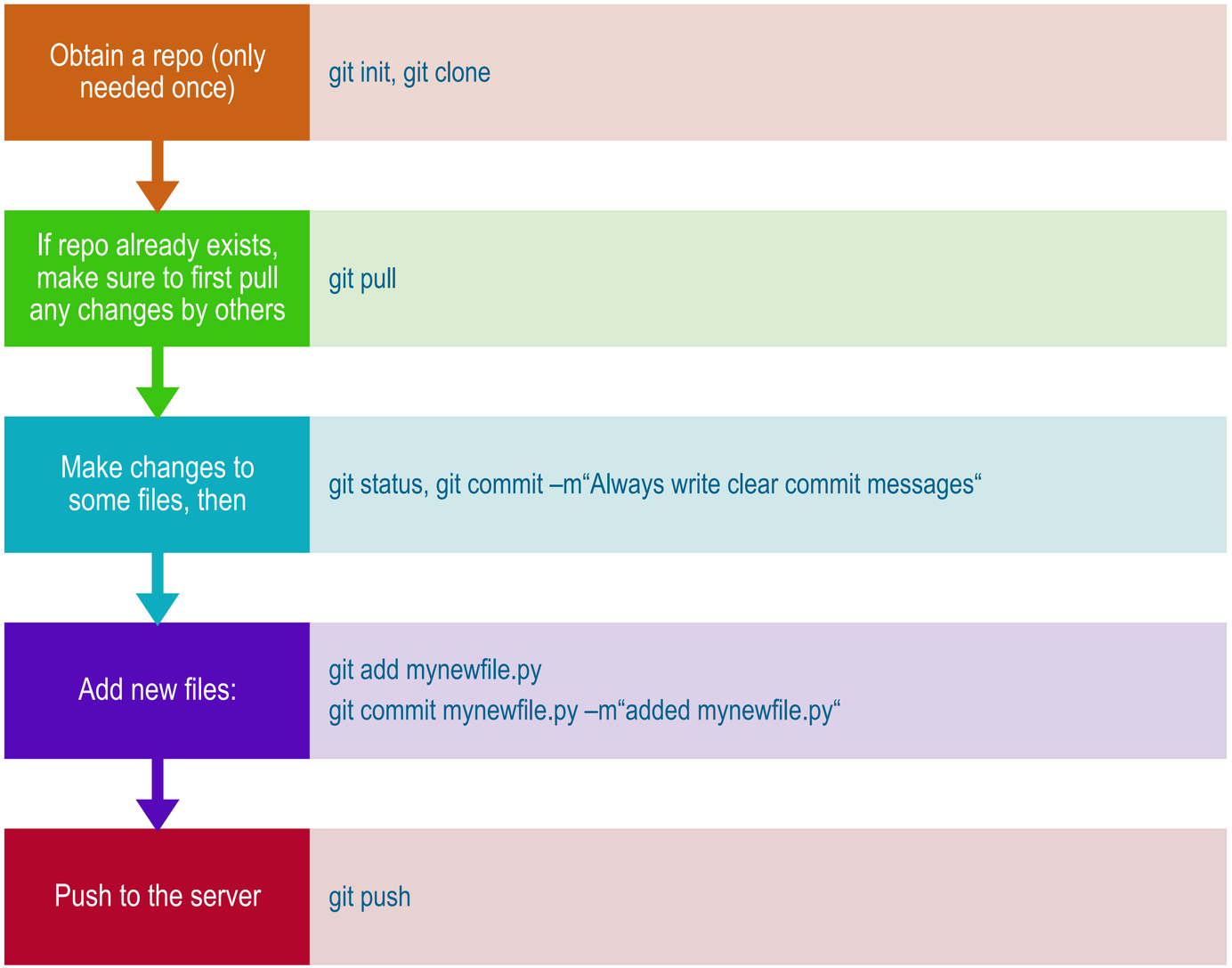
Revisions (commits) you make to your local repo are available to you only

The next time you are online (or you want to), you can push your changes to the server for sharing/backup

The local nature makes it effortless to create branches to isolate your work

A lot of minor changes (local commits) can be pushed as a single commit to remote branch

Git - workflow



Git - workflow

Key Concepts

Work is done here	Things "about to be stored" are put here	History, branches are stored here	*Same as Repository, but remote*
	`$ git add`	`$ git commit`	`$ git push` `$ git pull`
Workspace	Staging Area	Repository	Remote
			

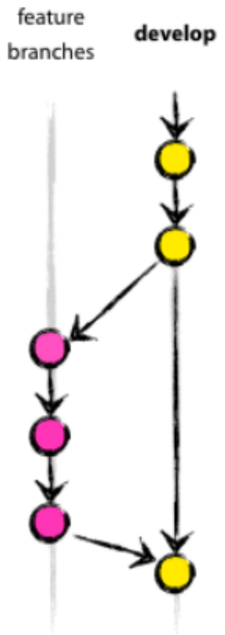
Source: http://de.slideshare.net/eykanal/git-introductory-talk

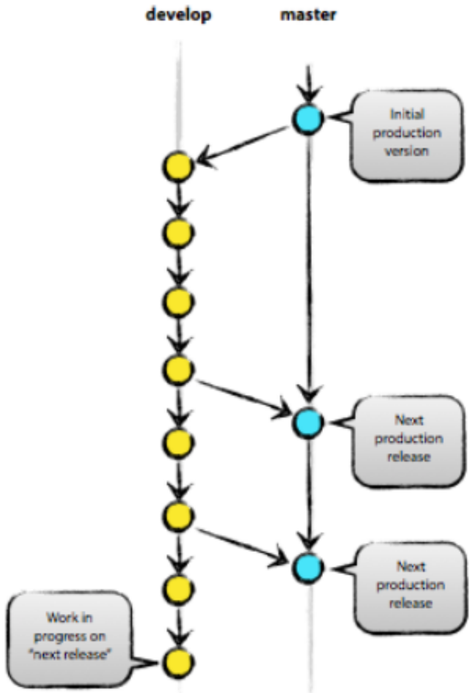
Git

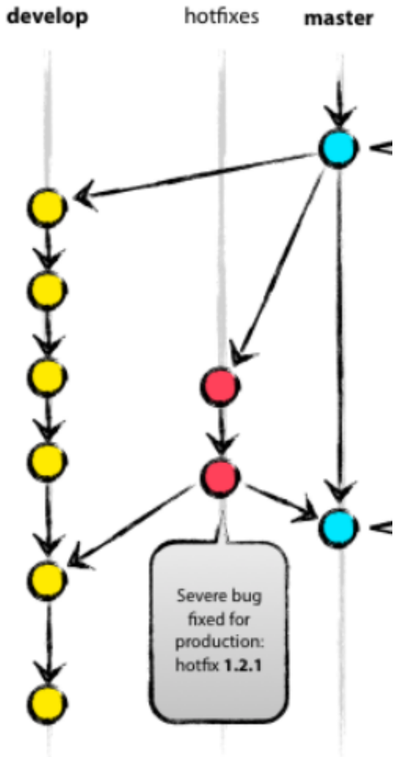
Branches	Allow to try out ideas/experiment isolated from the master-branch of the code
	Enable to work on long-running topics without interferring/breaking the master-branch
	Share work-in-progress with others
	Incorporate finished work into master-branch

Git - branches

- Think of branches as separate folders, each with it's own content and history.



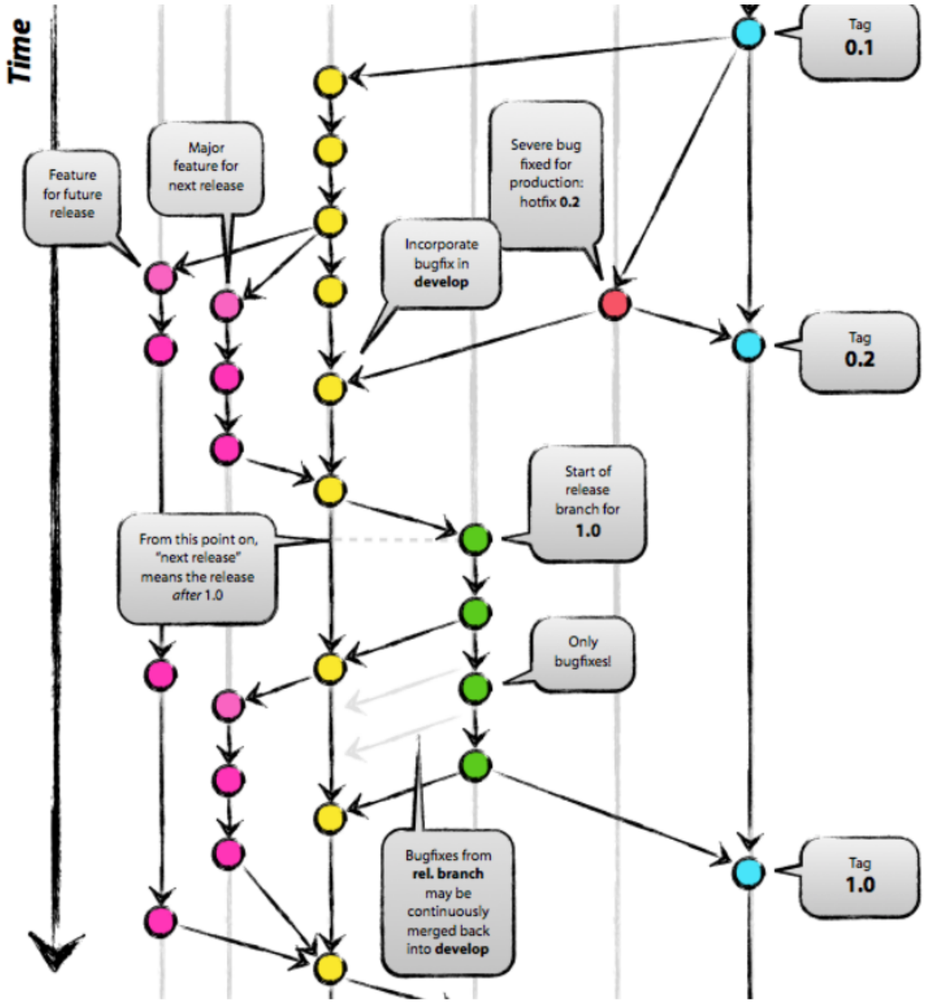




Source: http://de.slideshare.net/eykanal/git-introductory-talk

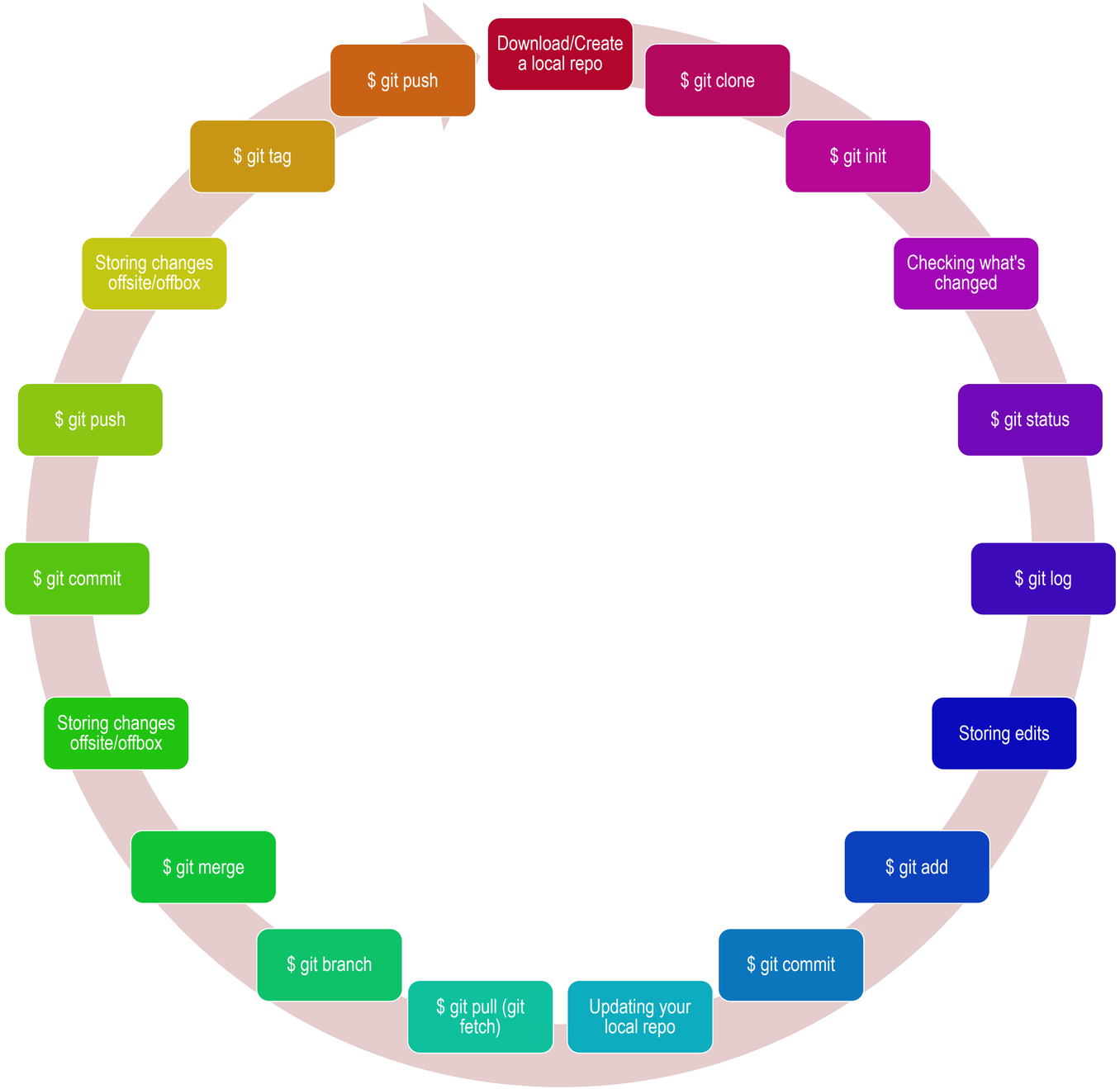
Git - branches

Branches – Go Nuts



Source: http://de.slideshare.net/eykanal/git-introductory-talk

Git – branching workflow



Git

Additional tips/conventions:	Do not put large binary files in plain git (use git-lfs, or a fileserver)
	Always create a README in the toplevel directory describing your work
	Try to only keep runnable code in the master-branch
Github.com:	Public repos are free
	Easy-to-use web-interface
	Has markdown enabled for *.md files (README)
	Has a wiki for every repo

Python style guidelines

Guidelines on formatting code, comments and docstrings for ease of readability

https://www.python.org/dev/peps/pep-0008

PEP8

- Limit all lines to a max of 79 characters
- Separate top-level function and class definitions with two blank lines
- Method definitions inside a class are separated by a single blank line
- Use blank lines in functions (sparingly) to indicate logical sections

PEP8

- Whitespaces
  - NOT immediately inside parentheses, brackets or braces
  - NOT immediately before a comma, semicolon or colon
  - NOT immediately before open parenthesis that starts the argument list
  - NOT immediately before open bracket that starts indexing or slicing
  - NOT more than one space around an operator

Yes:

```
x = 1
y = 2
long_variable = 3
```

No:

```
x             = 1
y             = 2
long_variable = 3
```

PEP8: comments

- Comments
  - Worst: comments that contradict the code (keep them up-to-date!)
  - Should be complete sentences
  - Comments should always be written in English
  - Block comments generally apply to code that follows them, indented to same level as the code.
    Each line starts with a # and a single space
  - Inline comments are on the same line as the code
  - Use inline comments sparingly, and only to add additional explanations

```
x = x + 1                 # Increment x
```

But sometimes, this is useful:

```
x = x + 1                 # Compensate for border
```

Docstrings

Documentation Strings (Docstrings)

- PEP257
- Write them for all public
  - Modules
  - Functions
  - Classes
  - Methods
- Not necessary for non-public methods (but put a comment there (after the def))
- Docstrings start and end with “““

Docstrings

Inside code, for this class: „sphinxy“ docstrings:

```
#!/usr/bin/python
'''
author: Tobias Weis
'''

class A():
    """This class holds some mathematical functions"""
    def __init__(self):
        pass

    def do_something(self,x,y):
        """This function computes the sum of its arguments

        :param x: first number
        :type x: float
        :param y: second number
        :type y: float
        :returns: float -- the sum of both numbers
        """
        return x+y
```

Documentation generation with Sphinx

In source folder; sphinx-quickstart. Set root folder to ./docs

In docs-folder: sphinx-apidoc –o source/ ../

Open index.rst, insert source/modules.rst to toc

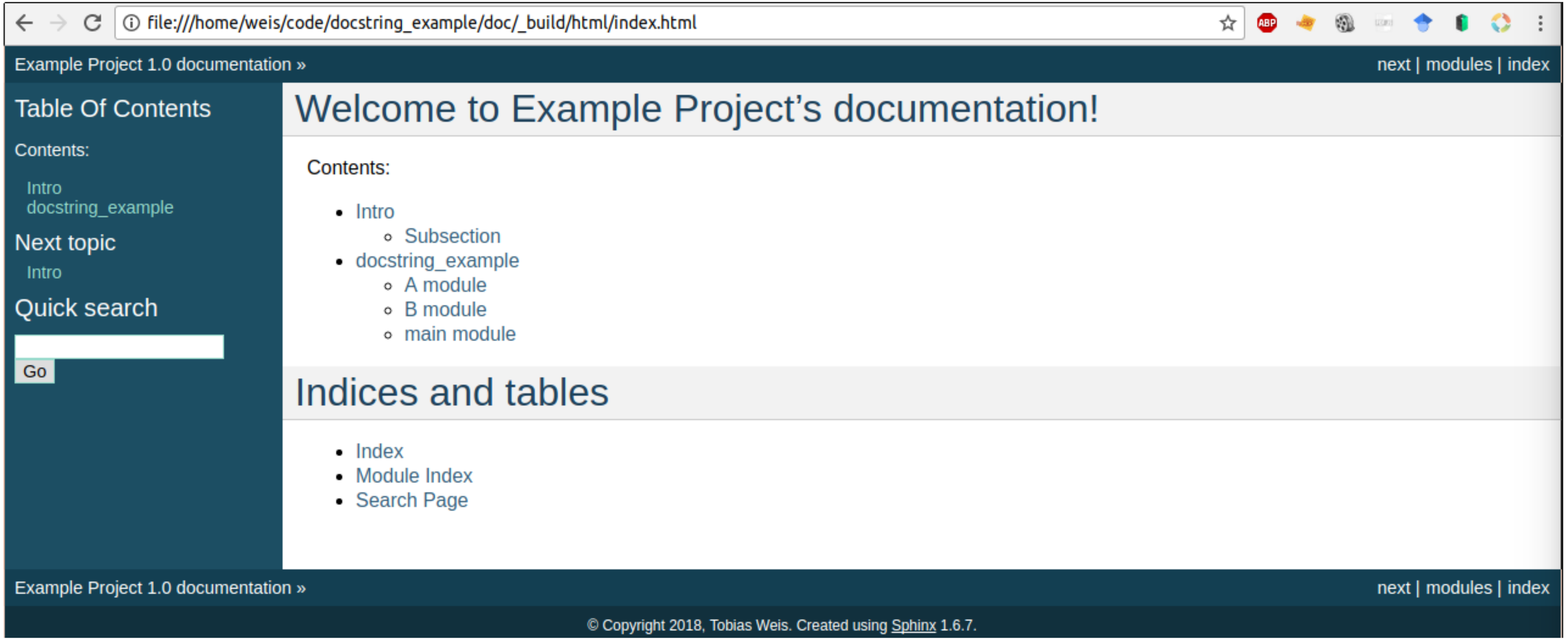
In conf.py, add the source-path to your system path
import os, sys
sys.path.insert(0, os.path.abspath(„..“)

In conf.py, add globaltoc.html to html_sidebars

Make html -> Google-chrome ./_build/html/index.html

Make latexpdf -> evince ./_build/pdf/projectname.pdf

Sphinx


file:///home/weis/code/docstring_example/doc/_build/html/index.html
Example Project 1.0 documentation »
next | modules | index
Table Of Contents
Contents:
Intro
docstring_example
Next topic
Intro
Quick search
Go
Welcome to Example Project's documentation!
Contents:
Intro
Subsection
docstring_example
A module
B module
main module
Indices and tables
Index
Module Index
Search Page
Example Project 1.0 documentation »
next | modules | index
© Copyright 2018, Tobias Weis. Created using Sphinx 1.6.7.

Sphinx: LaTex report

Example Project Documentation

Release 1.0

Tobias Weis

Feb 08, 2018

CONTENTS:

1 Intro 1
1.1 Subsection 1

2 docstring_example 3
2.1 A module 3
2.2 B module 3
2.3 main module 3

3 Indices and tables 5

Python Module Index 7

Index 9

CHAPTER

ONE

INTRO

This is written text (in rst), that is independent of the source code.

You could use these pages for * a description of your project * presenting results * writing tutorials on how to use your code

1.1 Subsection

This is a minor subsection

CHAPTER

TWO

DOCSTRING_EXAMPLE

2.1 A module

author: Tobias Weis

class A.**A**

This class holds some mathematical functions

do_something(*x*, *y*)

This function computes the sum of its arguments

Parameters

- **x** (*float*) – first number
- **y** (*float*) – second number

Returns float – the sum of both numbers

2.2 B module

author: Tobias Weis

class B.**A**

This class holds some string functions

do_something(*x*, *y*)

This function computes the concatenation of its arguments

Parameters

- **x** (*str*) – first number
- **y** (*str*) – second number

Returns str – the concatenation of both numbers

2.3 main module

author: Tobias Weis

main.**main**()

This function does something and uses other classes and their functions