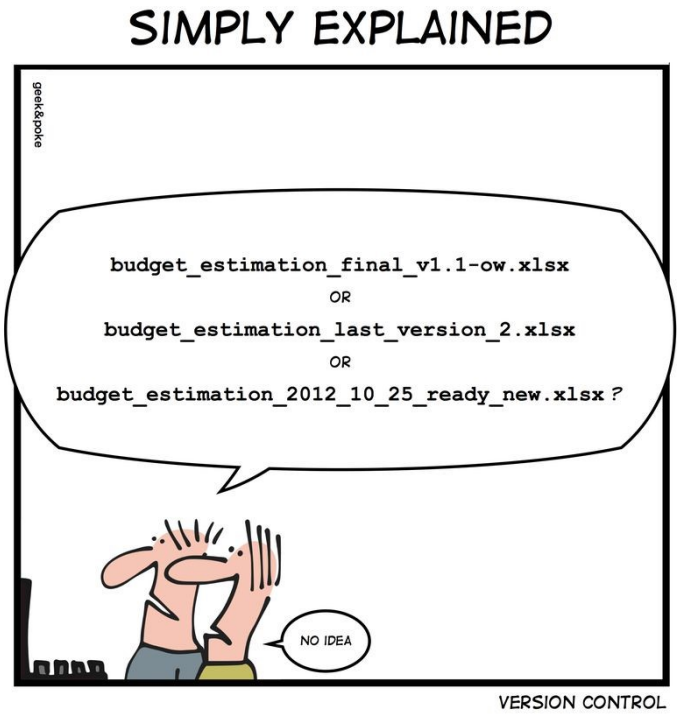
Table of contents (../toc.ipynb)

Why version control?

If you never have heard of version control, you might be familiar with a working mode which is shown in this carton posted by geek and poke.



Idea from Jen Simmons and John Albin Wilkins during episode #40 of "Web Ahead" about Git:
http://5by5.tv/webahead/40

(http://geek-and-poke.com/geekandpoke/2012/11/3/simply-explained.html)

Version control with git

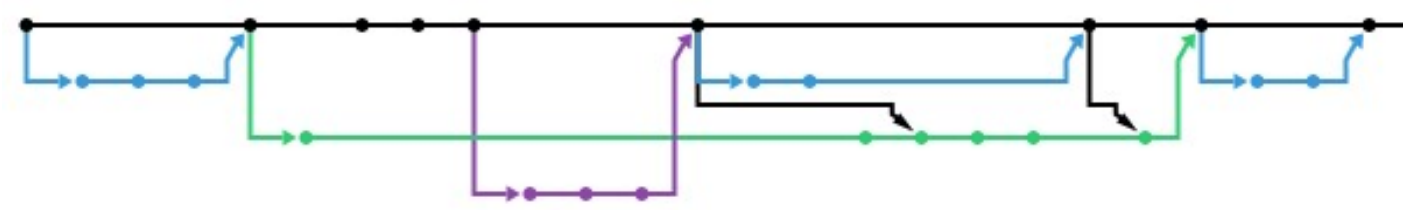
Version control is one fundamental pillar in each software development project. There is no excuse to omit version control!

Next to other version control systems like svn, git (https://git-scm.com/) is the most popular and most prominent tool right now. Its main advantage is the **powerful branch and merge concept**, which **allows to develop software simultaneously and distributed**.

Please find a much deeper feature description of git either on wikipedia https://en.wikipedia.org/wiki/Git (https://en.wikipedia.org/wiki/Git) or in the git docu https://git-scm.com/about (https://git-scm.com/about).

There are different working styles in git controlled projects. These different styles are called branching models and the `master - feature branch` concept is a simple concept you can use for smaller projects. Please find here a picture of a git master feature branch network.

The black solid line is the master and the shorter colored lines are feature branches which provide changed versions of the master. Each bullet is a commit.



Learn git in ten minutes

As pre-note, there is one superior git training web page https://learngitbranching.js.org/ (https://learngitbranching.js.org/) you should check out.

Next we want to try to learn some basics of git in a couple of minutes.

Git can either be used in terminal or through many GUIs like git for windows (https://gitforwindows.org/), sourcetree (https://www.sourcetreeapp.com/), gitkraken (https://www.gitkraken.com/).

As there is so much training material to find in the web (see links in a few slides), we will just briefly take a look at some git commands.

- `git --version` returns the version of your git. If this command fails, install git first (https://git-scm.com/downloads).
- `git init <directory>` initializes a folder as git repository.
- `git clone <repo>` clones a repository for instance from GitHub on your computer.
- `git add <files or A for all>` stages specific files or all for next commit.
- `git commit -m "<message>"` commits all staged files with a commit message.
- `git fetch` fetches changes from remote, for instance new branches.
- `git pull` pulls changes form current branch on your computer.
- `git push` pushes your local changes to remote on same branch.
- `git checkout -b <branch>` checkout `<branch>` from remote on your computer.
- `git status` lists staged or modified files.
- `git log` displays git history.
- `git merge <branch>` merges `<branch>` into you current branch.
- `git revert <commit id>` reverts a specific commit.

Exercise: git (5 minutes)

Please follow these steps to become familiar with some common git commands.

- Create a folder and navigate with your terminal to this folder.
- Type `git init` to initialize the git repository.
- Type `git status` multiple times between next steps to see what is going on.
- Create a file called `file.txt` with the text "Hello git".
- Stage this file with `git add -A`.
- Commit this file with `git commit -m "Add first file"`.
- Create a branch with `git branch develop`.
- Switch to this branch with `git checkout develop` you can see where you are with `git status`.
- Change the text in the file into "Hello git, we are on a branch".
- Commit this change with `git add -A`, `git commit -m "Adds better text"`.
- Switch to master with `git checkout master` and open the file, what text is there?
- Merge the changes from develop with `git merge develop`.
- View the history with `git log` and the graph with `git log --graph`.

Commit messages

During commit, you should add a meaningful commit message with `git commit -m "Your message goes here"` to avoid meaningless git history like in this xdk comic:

(https://xkcd.com/1296/)

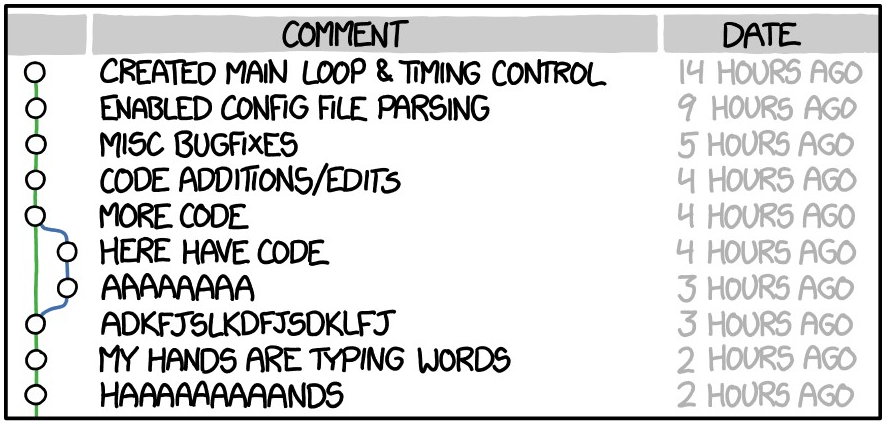
Good commit messages are well explained in numerous blog posts. One great blog post is from Chris Beams and linked in the next code block.

In [3]:
```
IFrame(src='https://chris.beams.io/posts/git-commit/#seven-rules',
       width=1000, height=600)
```

Out[3]:

How to Write a Git Commit Message

Posted 31 Aug 2014 · revision history



Introduction | The Seven Rules | Tips

Introduction: Why good commit messages matter

Pull Requests

If more than one developers works on a repository, which is the common case, merges of branches into master branch are handled through pull requests.

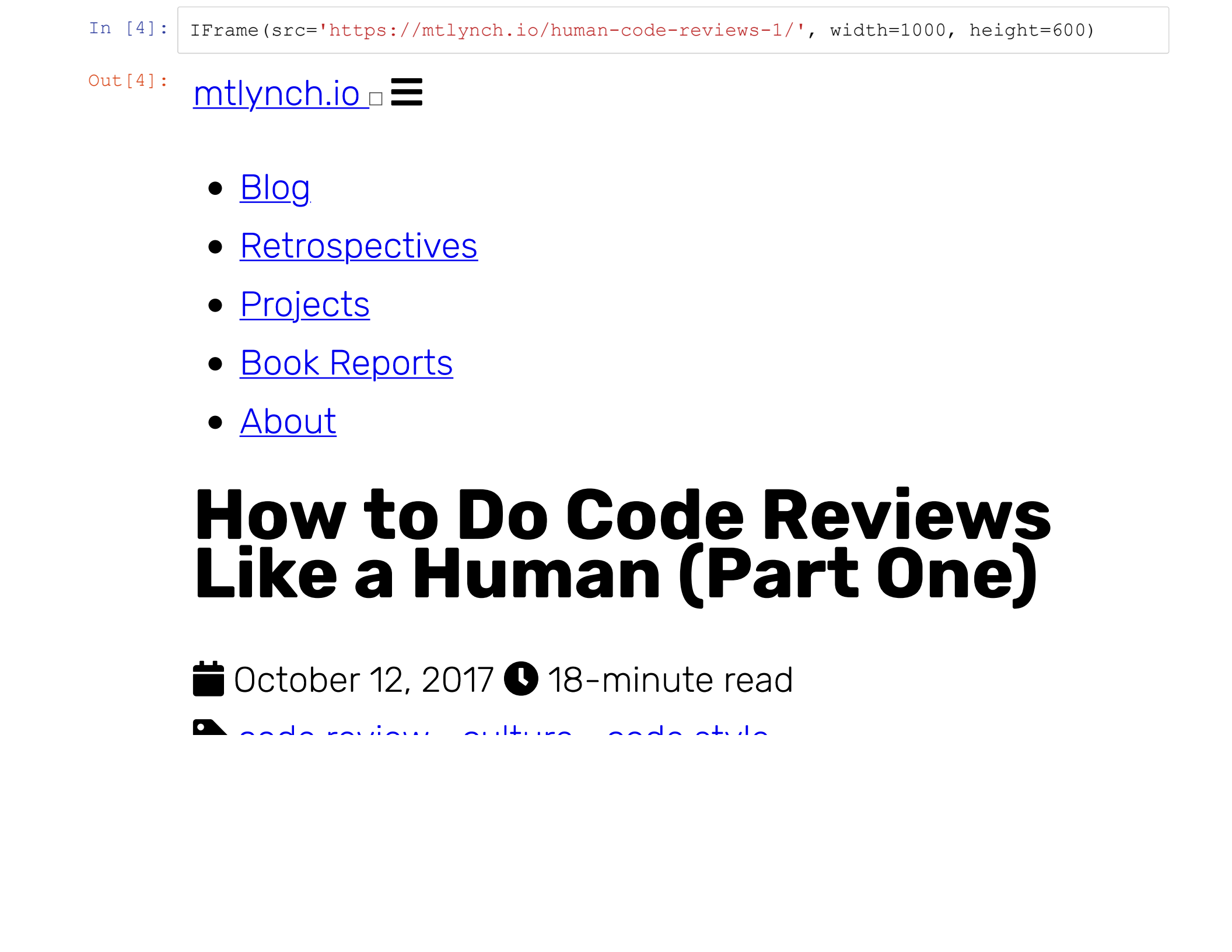
A pull request is a discussion between contributor (developer) and maintainer (usually the most experienced developer in the team) to include the changes of the developer in master version.

A pull request is the practical implementation of code review and ensures code quality. You can find very good explanations how to do pull requests in this blog post [How to Do Code Reviews Like a Human (https://mtlynch.io/human-code-reviews-1/)](https://mtlynch.io/human-code-reviews-1/).

This discussion is done on architecture level down to line of code level and platforms like GitHub provide extensive comment tools as we will see later.

In [4]:
```
IFrame(src='https://mtlynch.io/human-code-reviews-1/', width=1000, height=600)
```

Out[4]:



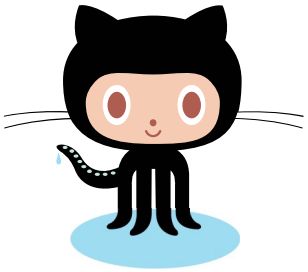
More git training material

Git can be learned from many sources and there is much more possible to do with than we have seen herein.

Here an incomplete list of references to become a git professional.

- A great page to learn git interactively https://learngitbranching.js.org/ (https://learngitbranching.js.org/).
- The main features of git are listed in this git page (https://git-scm.com/about/branching-and-merging).
- There is a free git book available on https://git-scm.com/book/en/v2 (https://git-scm.com/book/en/v2).
- Please find some introduction videos in git's doc (https://git-scm.com/doc).
- Also take a look at this git cheat sheet (https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet).

GitHub



GitHub (https://github.com/) is a development platform built specifically for git and provides beside basic code hosting many features to develop and promote software products open source. Almost all Python libraries are stored on GitHub. The features include: source code management, continuous integration and delivery, code review, project management, team management, and many more services like promotion web pages for your project in GitHub pages.

Other options for source code management in the web are bitbucket (https://bitbucket.org/) and gitlab (https://about.gitlab.com/).

You can find all features on https://github.com/features (https://github.com/features).

Add to this, GitHub is built for developers and promotes open source, and you can learn from the best developers like https://github.com/torvalds (https://github.com/torvalds).

Now let us explore one example project on github, which is scikit-learn (https://github.com/scikit-learn/scikit-learn).

Scikit-learn as example open source project

We will see that a well maintained open source project requires lot effort. Beside the programming code, many additional artifacts, documentation, project management, and quality assurance is to find.

Many projects on GitHub share a common structure of files and we will explore the structure of the scikit-learn repository in the sequel.

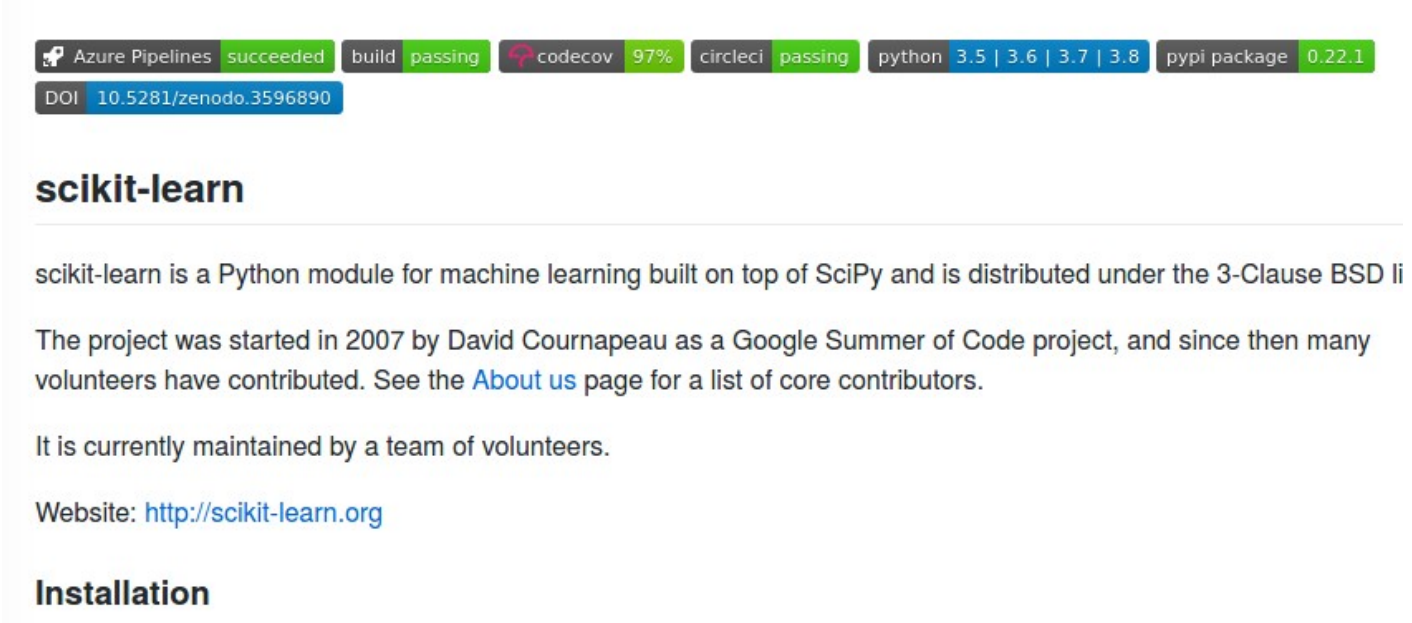
Repository structure

README

This file is the first anchor if you arrive as developer. It should contain information what the software does and what is the aim of the project.

Also information **how to install** the software, **how to contribute**, information about the **software health**, **links to wiki** or other websites and so forth are placed in a good readme.

Here a screen shot of the `README` file, note the colored badges from continuous integration reports.



Additional files for developers

Next to `README`, scikit-learn comes with `CONTRIBUTING`, `CODE_OF_CONDUCT`, and `PULL_REQUEST_TEMPLATE` files to define rules and guidelines for developers who want to contribute and to define some common rules for all project members.

It is actually like an own state with legislation rules that you find there.

`doc`

`doc` is usually a folder which contains the documentation of the software. The documentation is usually divided into code documentation (API), a user manual, tutorial, and additional more theoretical explanation.

Commonly, the **documentation is automatically build through continuous integration** and deployed to a web page. We will explore continuous integration later.

Most Python libraries are documented with Sphinx, which we will discover soon in this course.

Source code folder

Of course the actual source code of the software is stored on GitHub. In sklearn, the code is to find in the `sklearn` folder.

It is very useful to read well written code to become a better programmer. The `load_digits` function, which we used in the scikit-learn part of this course, is for instance here to find (https://github.com/scikit-learn/scikit-learn/blob/d6bb321fc0ddbb5bd91e7c8c90706a2ea8d65c6c/sklearn/datasets/_base.py#L611).

You can see that the developers spent lots of energy in documentation and meaningful code style. This is called *clean code* and we will see some clean code principles later on.

Tests

Code tests are very important to keep a software in a healthy state and to allow changes in the software.

- Hence, tests are vital to keep technical debt low,
- and to keep the software *soft*.

Usually, you can find tests on root folder level in a `tests` folder, here in scikit-learn, they are in the source folder included.

You can find the test coverage here

(https://codecov.io/github/scikit-learn/scikit-learn?branch=master)

Configuration files

There are typically many configuration files on root level of repositories such as

- `.gitignore`, which defines which filed should be ignored by git,
- `.gitattributes`, which defines for instance how merges should be handled,
- `setup.cfg`, which defines options of Python code checkers like `pytest`, and `flake8`,
- `setup.py`, which organizes development version installation.

Issues and pull requests

The project management is handled through issues (https://github.com/scikit-learn/scikit-learn/issues) where bugs and feature requests are discussed and tracked. This is the first place where developers interact through discussions.

The actual change of the master version of scikit-learn is organized, discussed and performed through pull requests (https://github.com/scikit-learn/scikit-learn/pulls).

Take a look at pull requests

A pull request is a discussion between **contributor** and **maintainer** to change the code of the master or development branch.

Here an example with a lean and simple pull request (https://github.com/scikit-learn/scikit-learn/pull/16366).

And here an example with a more elaborate pull request (https://github.com/scikit-learn/scikit-learn/pull/16339).

To sum up, you can also learn a lot if you just follow interesting pull requests.

GitHub in a nutshell

GitHub provides a rich source of information and you can learn from giants in the software development!