

CI Workflow based on GitHub

Table of Contents

[1 Basics 3](#_Toc510697717)

[2 Working with the workflow 4](#_Toc510697718)

[2.1 Getting started with a repository 4](#_Toc510697719)

[2.2 Beginning a new task 5](#_Toc510697720)

[2.3 Development 6](#_Toc510697721)

[2.4 Distribute changes 6](#_Toc510697722)

[2.5 View status 7](#_Toc510697723)

[2.6 Troubleshooting 9](#_Toc510697724)

# Basics

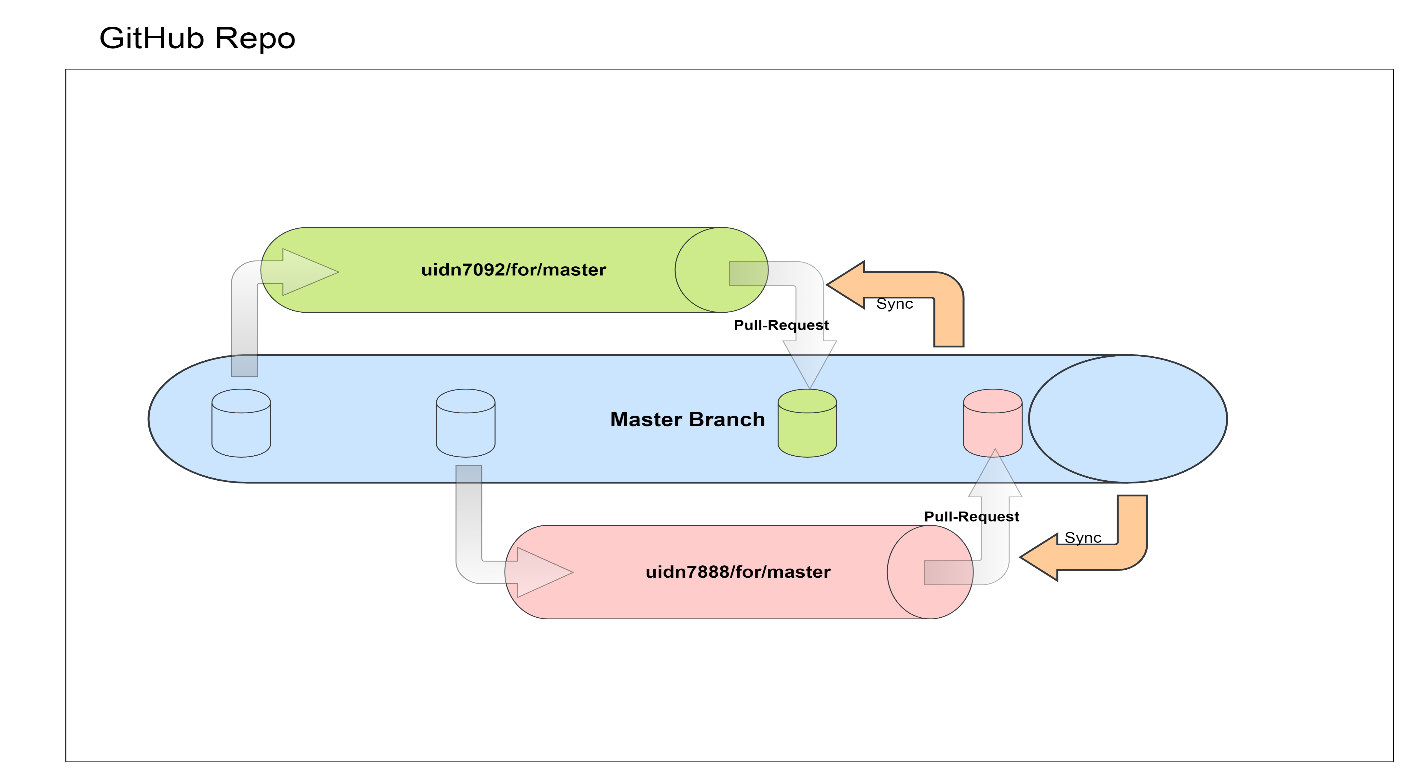
The development takes place on branches that should be as short-lived as possible and are normally personal (shown in green and red in the following illustration). They can also be created for specific bug fixes or features.

Normally, only one person works on a branch. But it is also possible to work in small teams together on a branch. Anyone working on the branch should push the local commits into the branch as often as possible.

As soon as the developed feature is ready for distribution, it is merged into the master via a pull request. The master itself is locked, so the development can actually only take place in branches.

Development versions and features should not be shared "half-finished" via feature branches, but distributed in finished versions via the Master.

The advantage of this approach lies in a very stable growing master and a reduction of merge conflicts.



The workflow supports two approaches:

1. automatic generation of pull requests

The first push into a branch following this approach automatically creates a pull request for the branch on GitHub.

Then, the checks specified for the repository are performed.

The workflow specifies that at least a check is made as to whether the changes delivered with the push could be compiled after merging with the code in the master and the existing unit tests could be carried out successfully.

With each subsequent push, the changes are added to the open pull request and rechecked. Until the pull request is closed.

1. manual creation of pull requests

When pushing into a branch that follows this approach, the workflow starts only the checks specified for the repository.

The pull request must be created manually via GitHub. Everything else is analogous to section 1.

# Working with the workflow

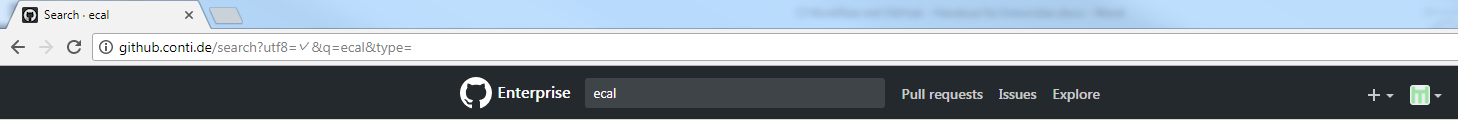
The work steps are explained in this document using Git Bash. It would be beyond the scope of this document to address all GUI clients that exist for Git.

## Getting started with a repository

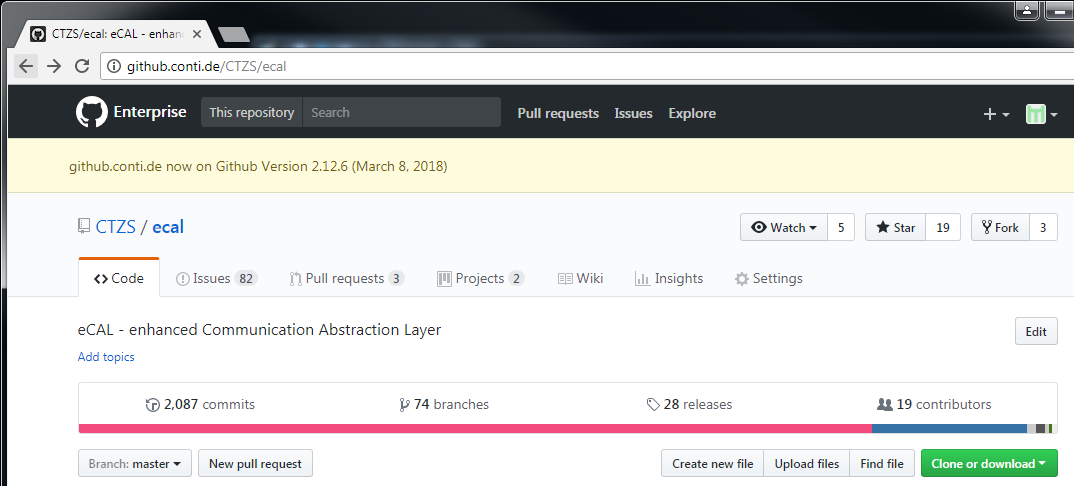
First, clone your remote repository to create a local copy on your computer. In order to do this, follow the steps below:

1. Open GitHub in a browser: <http://github.conti.de/>
2. On GitHub, navigate to the main page of the repository

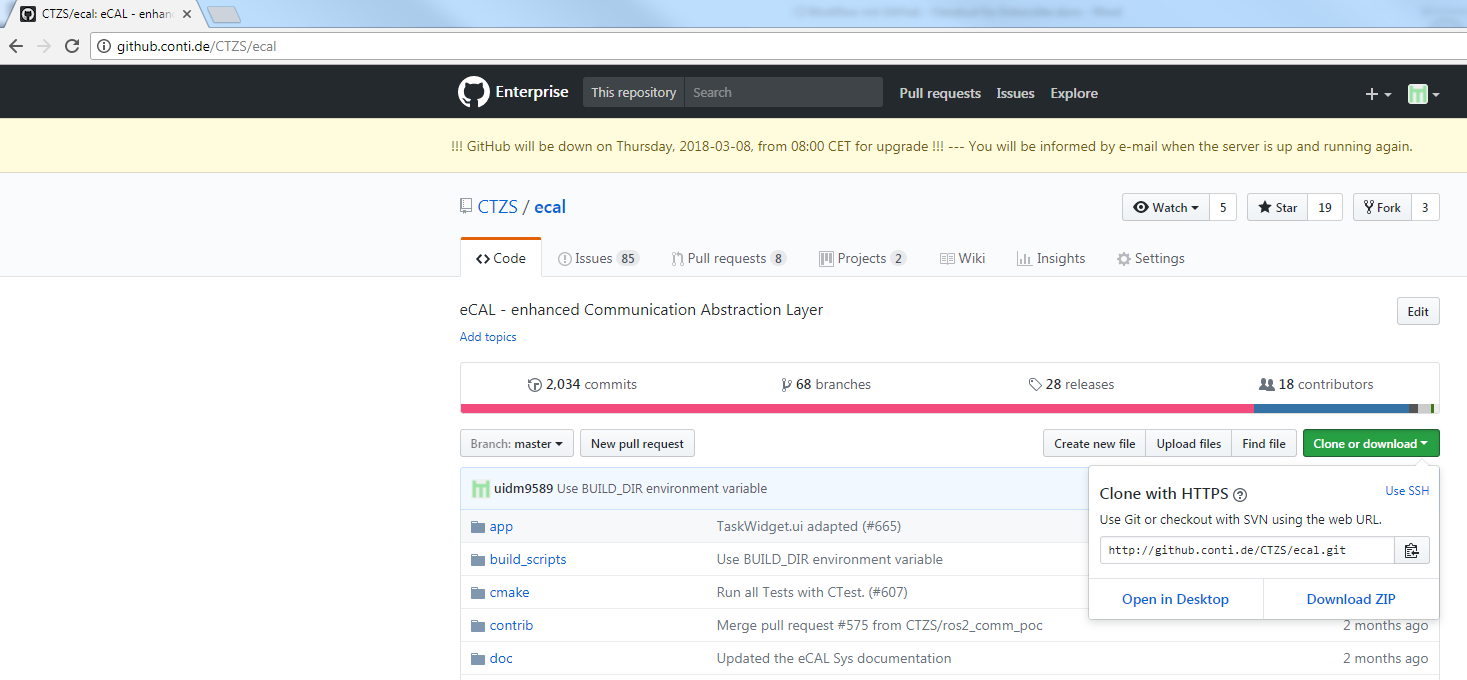
(for example by typing the repository name in the search bar):



1. In the repository view, click “Clone or download” on the right.



1. In the subsequently displayed section, copy the clone URL for the repository:



1. Open Git Bash and change the current working directory to the location where you want the cloned directory to be made.
2. Type „git clone“, and then paste the URL you copied in step 4.

Example: **git clone http://github.conti.de/CTZS/ecal.git**

## Beginning a new task

First, a new branch must be created. In order to do this, follow the steps below:

1. Choose a branch name

The branch name determines which of the two approaches the workflow offers is used.

1. automatic generation of pull requests

In this case, the branch name must match the pattern **{identifier}/for/{base}**

{identifier} stands for any text (for example, the user ID, the issue number or the feature name). But it must not contain /for/.

{base} stands for the target branch of the pull request (e.g. master)

Example: uidm9589/for/master

1. manual creation of pull requests

In this case, the branch name may not contain the text **/for/**.

1. Create a local branch

Open Git Bash and enter:

**git checkout –b <branch-name>**

Example: git checkout –b uidm9589/for/master

1. Add the branch to the remote repository

Open Git Bash and enter:

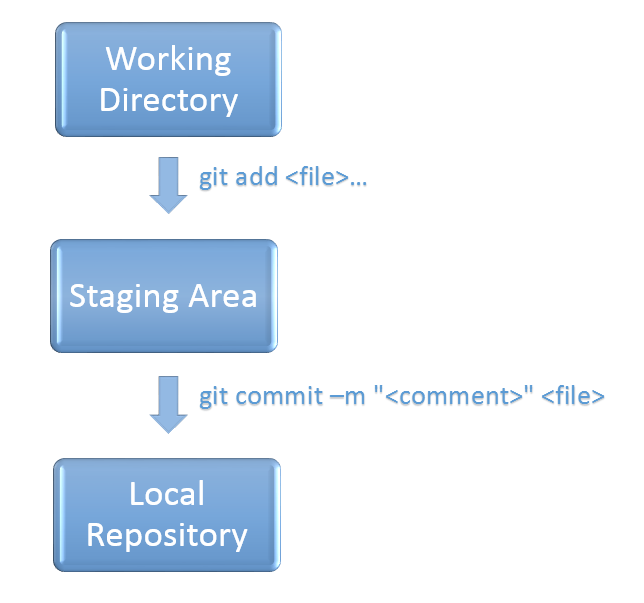
**git push --set-upstream origin <branch-name>**

Example: git push --set-upstream origin uidm9589/for/master

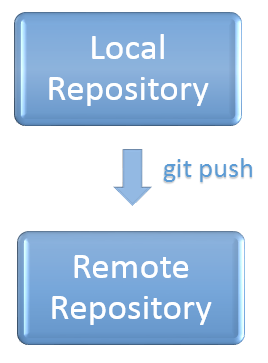
The --set-upstream option ensures that the local branch is associated with the remote branch, so that you can omit the target specification in all subsequent commands.

## Development

Make changes in the local working directory and save as follows:



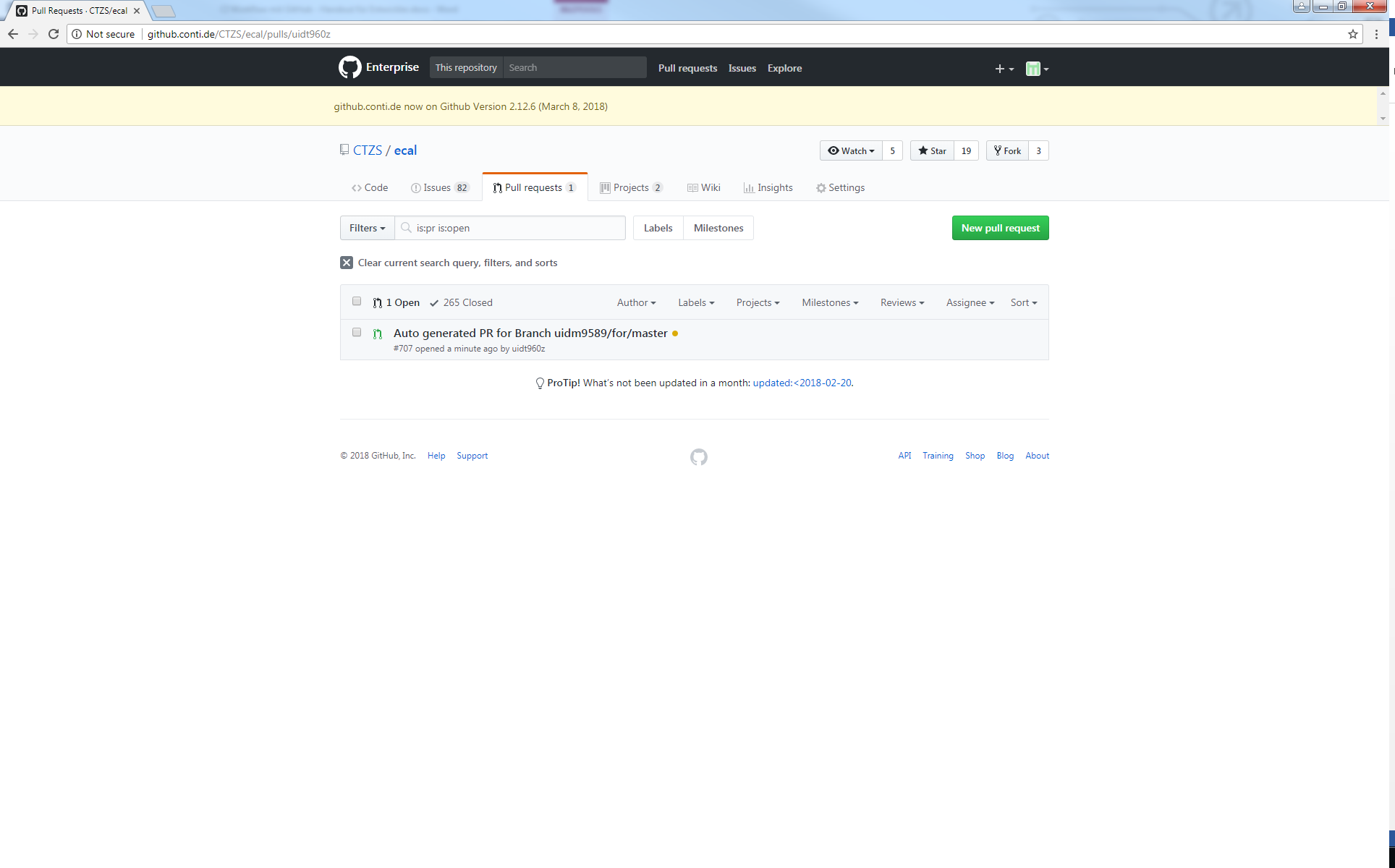
## Distribute changes

Once a stable version has been reached, it can be pushed into the branch in the remote repository:

## View status

Once a pull request has been created (automatically or manually), the status of the checks performed by the workflow can be tracked via GitHub.

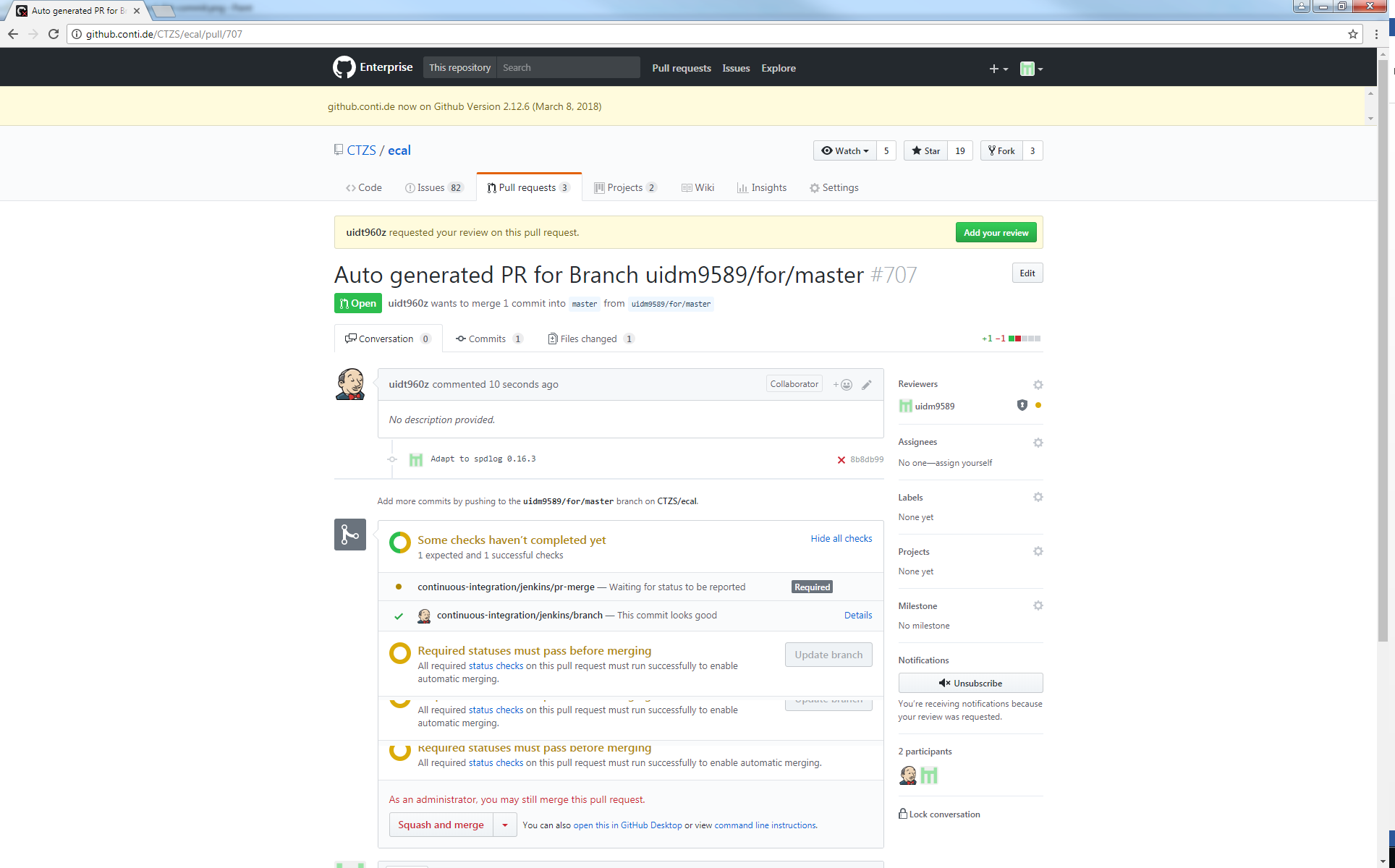
1. Open GitHub: <http://github.conti.de/>.
2. On GitHub, navigate to the main page of the repository.
3. Switch to the „Pull requests“ tab:



1. Open detailed view by clicking on the **name of the pull request**

(in the sample illustration above that is „Auto generated PR for Branch uidm9589/for/master“).

Further down in the view (marked red in the following illustration), the **status of the checks** specified for the repository is displayed.

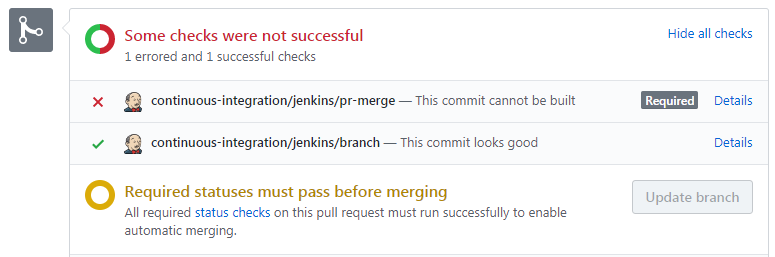


„**continuous-integration/jenkins/pr-merge**“ (marked with a red arrow in the following illustration) stands for the check of whether the changes could be compiled after merging with the code in the master and the existing unit tests could be carried out successfully.

While a check is being performed, a yellow dot is displayed to the left of it.

Once a check is completed, a green tick (if successful) or a red cross (if failed) is displayed instead.

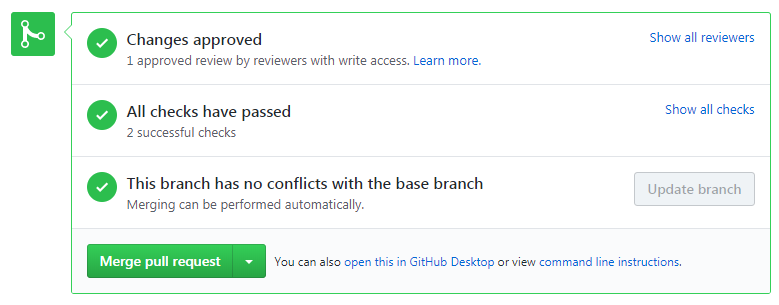
If the compilation or a unit test fails, an e-mail is automatically sent to the developers concerned. In GitHub, the failure is displayed as follows in the pull request view:



To the right of a check, a link called **„Details“** will appear that will allow you to switch to Jenkins for troubleshooting.

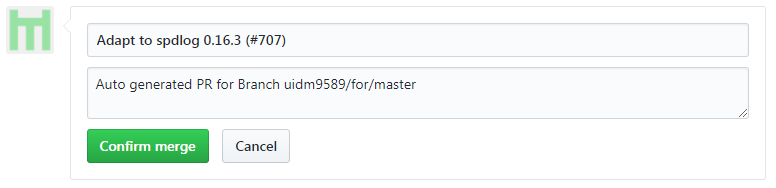
Troubleshooting is done in the same way as the development (see chapter 2.3). Once all changes have been made to fix the error, they can be pushed into the branch in the remote repository. The checks specified for the repository are then carried out again and the status view of the pull request in GitHub is updated accordingly.

When all the checks specified for the repository have been completed successfully, the Merge button will be unlocked:

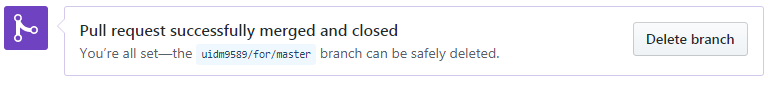


By clicking on the „**Merge pull request**“ button the changes can be prepared for the transfer to the master.

Subsequently, the commit message and the description for the master can be specified before the merge is started by clicking the „**Confirm merge**“ button:



The view in GitHub should look like this afterwards:



Finally, the branch should be deleted to free the disk space in the build environment.

## Troubleshooting

The pull request view in GitHub or the e-mail sent in case of errors can be used to call up the corresponding job in Jenkins to analyze the error.

The workflow checks are performed on both Linux and Windows. For better readability, the corresponding console outputs are indicated at the beginning of each line by [linux] or [windows].

Compiler errors and warnings are collected and can be called up via the menu item „Parsed Console Output“.