1 Introduction

On May 28th 2018, the version history of the QDYN repository was rewritten to address a problem in the history tree itself. All developers that have cloned the QDYN repository before that date should follow the steps below to re-synchronise their local repository with the upstream repository. Users of QDYN that did not make any code changes can simply delete the old repository and clone the latest version to get a clean QDYN repository.

2 Creating and restoring back-ups

Before making any changes, you should always back-up your local QDYN repository. In case things don't turn out as expected (e.g. unpushed work is lost), you can always revert your actions. If you have made any code changes that are not pushed upstream (i.e. are not present in `https://github.com/ydluo/qdyn`), you will first need to ensure that your work is properly stored away.

First, commit all code changes (also stashed changes):

```
git stash apply
git commit -a
```

Once all code changes are committed, you can proceed to create a back-up. The following commands will create and verify a back-up of your entire repository (including branches), all bundled into a single file (`qdyn.bak`, or some other file name of your choice):

```
git bundle create qdyn.bak --all
git bundle verify qdyn.bak
```

You should then move this back-up file to some place safe. In case of emergency, you can restore this back-up with

```
git clone /path/to/qdyn.bak qdyn
```

This will create a directory `qdyn` and restore all source and git files in that directory. You will now need to re-define the `origin` location, which currently points to the back-up file:

```
cd qdyn
git remote set-url origin https://github.com/ydluo/qdyn.git
```

The above procedure should now have fully restored your original repository.

3 Storing unpushed commits

In the case that your local repository contains commits that are not pushed upstream, you will need to store them on a temporary branch so that they do not get overwritten by the upstream changes. I will demonstrate this using a simplified example. If you have no unpushed commits, you can skip this section.

Say we have a repository with a history tree as shown in Fig. 1. In this example, the last common ancestor (i.e. the last commit that is shared between your local and the upstream repository) is commit `227e39f1d4ce`. First we create a temporary branch, and force all commits that have been made since the last common ancestor into the temporary branch:

```
git branch temp
git reset --hard 227e39f1d4ce
```

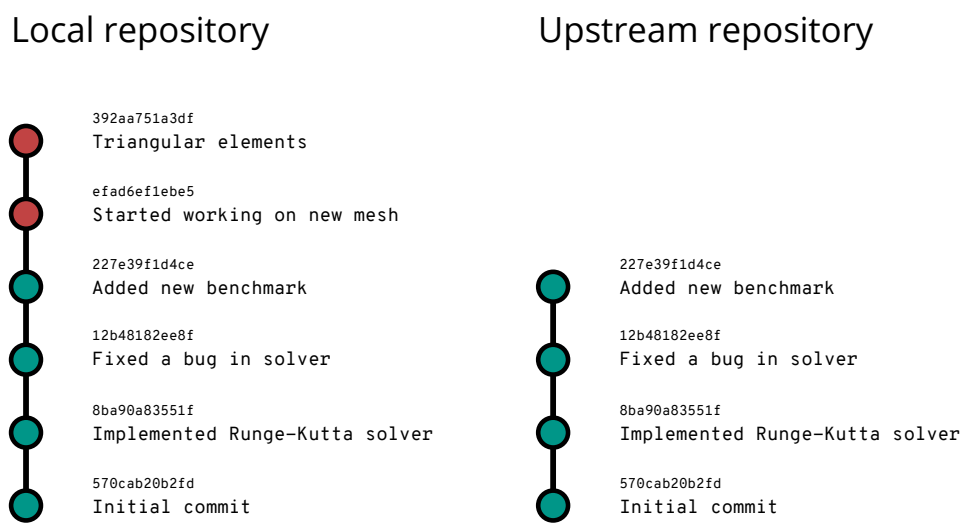

Fig. 1: Local and upstream structure of an example repository. The local repository has unpushed commits which need to be stored away while the upstream fix is applied.

The master branch is now in sync again with the upstream repository, with all of the later commits stored away in `temp`. You can verify this with:

```
git checkout temp
git log
```

Your local commits should be listed in the `log`. Return to the master branch with:

```
git checkout master
```

If you are satisfied with the results proceed to fetch the upstream history fix once it is applied. Note that the commit hashes used in the above example do not correspond to those in the actual QDYN repository, so you need to check for yourself which commit hash you need to use.

4 Fetching the upstream fix

After the upstream history fix is applied, you may fetch the upstream changes with:

```
git fetch origin
git reset --hard origin/master
```

The version history of the `master` branch will now be identical to that upstream. If you had any unpushed commits that you stored on the `temp` branch, you can merge them back onto the `master` branch using the `rebase` command (Fig. 2). Otherwise you are done.

```
git checkout temp
git rebase -i master
```

In the command line a new interface will appear, in which you can specify which commits you wish you include in the `rebase`. In the above example, we only wish to keep the last two commits, so we remove all lines that have a commit hash except for the bottom two (note that the order is reversed in `rebase`, the youngest commits are at the bottom of the list). The terminal should now look something like this:

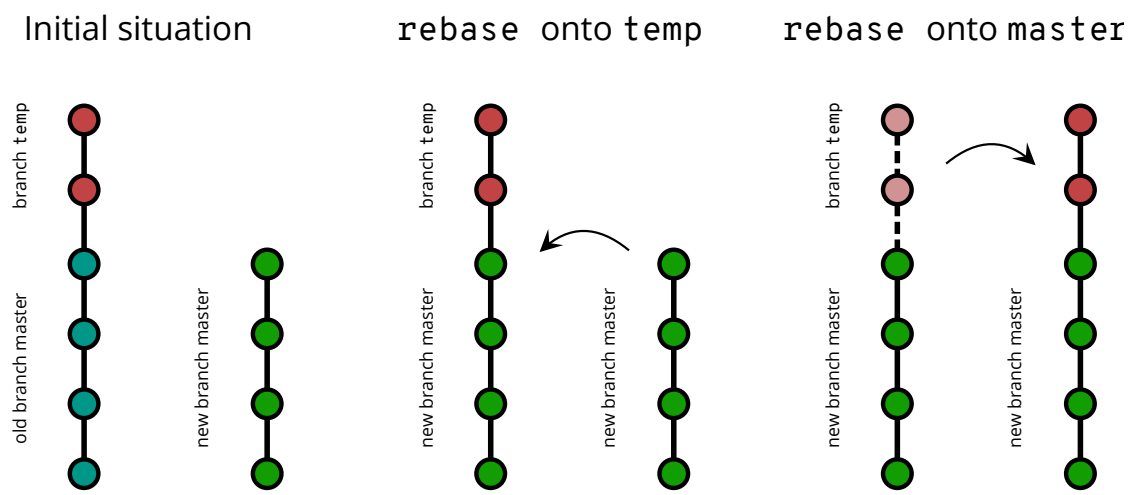

Fig. 2: After the upstream history fix has been fetched, the `temp` is connected to the old `master` branch. First, the new `master` branch is copied onto the `temp` branch, after which the unpushed commits can be safely rebased onto the `master` branch.

```
pick efad6ef Started working on new mesh
pick 392aa75 Triangular elements

# Rebase <hash> onto <hash> (n commands)
#
# Commands:
# etc.
```

By removing all previous commits that are present on the old `master` branch, we override them with those on the new `master` branch. Run a `git log` to verify that the last commits are on top of the new master branch. Then, execute

```
git checkout master
git rebase temp
```

This will copy all unpushed commits on the `temp` branch onto `master`. You are now in sync with the upstream repository, with your unpushed commits on top of the rewritten `master` branch. If you want to keep a clean history tree, you can delete the `temp` branch with (although there is no harm in keeping it)

```
git branch -d temp
```