english
stub

Pull UHSD source code from Github

Option: Download the zippd UHSDR

- The easiest approach is to donload the zipped UHSDR source code from Github UHSDR repository and unpack.
- If using Eclipse: create a work space pointing at the local directory where UHSDR has been unpacked. Check Eclipse settings after that (active build etc.)

Option: Use "Github Desktop"

- Download Github Desktop. The .exe file is the executable (portable) program and does not need installation
- Provide your github username and password, clone the Github UHSDR repository
- Optional: Create a local branch for your source code changes
- If using Eclipse: create a work space pointing at the local directory where UHSDR has been unpacked. Check Eclipse settings after that (active build etc.)

Option: Use "git" on command line level, command line style

Fork UHSDR to your own remote github repo

- Create github account if needed, or sign into github
- Goto https://github.com/df8oe/UHSDR and create a fork - which will be located in your github space
- Note the URL of your github fork, e.g. https://github.com/DF9TS/UHSDR-1

Pull UHSDR from Github

- If git is not intalled on your system:

```
$apt-get install git
```

- Clone UHSDR repository: Go to the directory where you want to store the GIT clone, then create a local clone. Use the URL for your github fork, e.g.

```
$git clone https://github.com/DF9TS/UHSDR-1
```

- You now have a local clone of your fork in the current local directory
- Check the remote repos assigned

```
git remote -v
```

- Add DF8OE UHSD repo as upstream remote repo so that you can esaly synch it with your fork:

```
git remote add upstream https://github.com/df8oe/UHSDR
```

- Check with “git remote -v” that upstream has been added. You now have two remote repos: “origin” which is your fork on github, and “upstream” which is the DF8OE master repo

Rebase: Synchronising your fork with UHSDR DF8OE repo

- You want to synchronise your forked UHSDR with the DF8OE main repo from time to time to include the latest changes of UHSDR into your fork.
- For that use

```
 git pull --rebase upstream active-devel
```

- Your local fork is now synchronised with UHSDR DF8OE repo
- Next you need to push these changes to your remote github repo

```
git push origin active-devel
```

- Your github fork is now synchronised with DF8OE UHSDR main repo

Use local UHSDR clone in Eclipse

See also GNU MCU Eclipse workspace preferences

Option: Create work space

- Good if you need new Eclipse settings specially for this Github clone
- There will be a new Eclipse work space assigned to this import. The new work space can have its own Eclipse settings. Everytime you switch to this work space Eclipse will change all settings accordingly.
- Start Eclipse. File→Switch workspace→Other
- Enter directory of local UHSDR clone.
- When using this new work space Eclipse will use settings of UHSDR clone. So please check Eclipse settings (active build, ..)

Option: Import UHSDR into eclipse

- Good if this Github clone should work under your standard Eclipse settings
- File→Import, then General→Existing Projects into work space, hit “Next”
- “Select root directory”. Use “browse” button and point to local github mchf root (in my case: $HOME/uhdsrd/UHSDR/mchf-eclipse/)
- Hit “Finish”

Contributing, Rebase

The heart of any open source project is the contributions by individuals. Source code for the project is

using git currently and can be found here. There is a good explanation of github workflow here that is worth a read before branching and hacking on the code. And an online book written directly from the git creators you can find here - or locally

here

.

We have specific instructions for contributors collected in guidelines for contributing.

For own developments and experiments in code you should **NOT** use merge/pull to integrate your changes in df8oe's active-devel. Instead use git “rebase” command, as documented on our contribution page. We had very little trouble to follow that approach for the last two years. It also has its issues but in general, once you understand how to operate it, it works quite well. The main benefit of the rebase approach is that it keeps all your “local” changes together on top of the last official release you rebased on. Opposed to the merge approach you have been running, which does mingle your changes with the external changes in the time history.

Further reading

https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-clone