Elettra Sincrotrone Trieste

Working with Git

L. Pivetta

Rev.	Date	By	
1.0.0	2020-04-01	Lorenzo Pivetta	First release
1.1.0	2020-05-21	Lorenzo Pivetta	Add annotated tag details. Update CVS-to-Git section.
1.2.0	2020-05-25	Lorenzo Pivetta	Add Working with submodules
1.2.1	2020-08-10	Lorenzo Pivetta	Fix submodule section to use git@
1.3.0	2021-05-13	Lorenzo Pivetta	Remove unique repository names constraint
1.3.1	2021-09-29	Lorenzo Pivetta	Tag deletion clarification
1.3.2	2022-03-23	Lorenzo Pivetta	Add minor clarifications
1.3.3	2023-05-08	Lorenzo Pivetta	Add .gitignore suggestion, remove redundant cd, update Make-9.3.4.in
1.3.4	2024-02-21	Lorenzo Pivetta	Annotated tag for installation must be on master branch

Elettra Sincrotrone Trieste

Setup your account

Login into https://gitlab.elettra.eu, go to Settings and load your ssh public keys. Elettra ICT servce enables access to gitlab.elettra.eu after the first login attempt.
Configure some essential global configuration variables for your account:

```
$ git config --global user.name "Nome Cognome"
$ git config --global user.email nome.cognome@elettra.eu
```

Please note the above configuration has to be done on each host used for development[1].

Init a repository

Create, or move into, the folder you want to start tracking. We'll assume it's *my-new-git*

```
$ cd /home/<username>/<any-path-here>/my-new-git
$ git init
```

Files are not automatically added. Once you have your source files, README, gitignore populated run *git add* command to move files to staging and then *git commit* to commit changes[2]:

```
$ git add src/*.[hc]
$ git add README.md
$ git add LICENSE
$ git add .gitignore
$ git commit -m 'Initial import'
```

Note that no remote repository is associated. To associate the default remote, conventionally called "origin", run the "git remote add" command.
In the case of a Tango device, hosted on gitlab.elettra.eu in the subgroup "cs/ds", the command is:

```
$ git remote add origin git@gitlab.elettra.eu:cs/ds/my-new-git.git
```

To check the remote setup:

```
$ git remote -v
```

then remember to push the local repository to the remote repository, sticking to the default master branch name:

```
$ git push origin master
```

[1] Unless the hosts share the same home over NFS.
[2] Please note the following lines presume .h and .c source files; your mileage may vary.

The previous command requires write access to the gitlab group.

Cloning an existing repository

Using one common “makefiles” repository in this example, to clone an existing repository run:

```
$ git clone git@gitlab.elettra.eu/cs/ds/makefiles.git
```

Cloning a repository automatically sets up the remote.

Working on an existing repository

Whenever a repository exists in your local hierarchy **first check** the status against the remote. To update the local copy **without** merging:

```
$ git fetch
```

or, to automatically try to merge updates into the local copy:

```
$ git pull
```

Basic git commands

```
$ git clone <repo>
$ git status
$ git fetch
$ git pull
$ git remote -v
$ git add <file>
$ git commit
$ git mv <filefrom> <fileto>
$ git log
```

And a few not so basic

```
$ git rm <file>                # staging a file for removal
$ git reset HEAD <file>        # un-staging a file staged for removal
$ git checkout -- <file>       # un-modifying a modified file
$ git rm --cached <file>       # remove a file from staging but keep it in the harddrive
```

Working on a project

See also the “Branching policy” section in “Software development procedures and installation policy for control systems” document. Each new feature or bugfix deserves a branch. Branch names must be meaningful.

```
$ git branch async-io-thread          # create the branch
$ git checkout async-io-thread        # switch to the new branch
$ git branch -v                       # check status
```

...do some work… **test it's working**[3]. Push your branch to the remote to make it visible to collaborators

```
$ git push origin async-io-thread
```

Make a pull request to merge whenever more than one developer is working on a repository. This is easily done with the web interface. Otherwise, merge the branch into master on your local copy:

```
$ git checkout master
$ git merge async-io-thread
```

Resolve conflicts, if any. Branches shall be short-lived. Merging often into the master it's easier. Once merged, delete the branch, it's not needed anymore:

```
$ git branch -d async-io-thread
```

If you're working alone, e.g. no pull request required, do not forget to push your local master into the remote repository.

Tagging

Git supports lightweight and annotated tags. Lightweight tags can be used to create a reference to a specific commit. Both lightweight and annotated tags are in the form x.y.z and adhere to semantic versioning 2.0.0 specification.

```
$ git tag 0.0.1                              # lightweight tag, no -a, -s or -m
$ git tag -a 1.0.0 -m 'Release 1.0.0'        # annotated tag

$ git tag                                    # show existing tags
$ git show-ref -d --tags                     # show tag details
```

Releasing a project

When ready for release, with all relevant changes merged back into master, make an annotated tag on master branch and push it to the remote; **annotated tags enable deployment in production**. Please note that **not** every commit deserves an annotated tag. Moreover, not every tag has to be an annotated tag. Annotated tags mark a milestone in the project that deserves installation.

[3] It is mandatory to test your changes/additions before pushing to the master branch. See section "Testing" in "Software development procedures and installation policy for control systems".

```
$ git branch
* master
$ git tag -a 2.0.0 -m 'Release 2.0.0 - breaks everything'
$ git push origin 2.0.0
```

Whenever an annotated tag is made, the INAU service, if enabled for the specific repository, automatically clones the repository and builds the target. In case of successful build, the target is available for installation[4]. The developer is informed via email of the successful, or failed, build. In case of build failure, the developer has to fix the source and provide a new annotated tag. To avoid "thrashing" tags, it is recommended to check carefully that the repository builds correctly before creating an annotated tag. Tags cannot be deleted.

Working with Git Submodules

Submodules are used here in a very simple approach, just to track possible component dependencies. It is **forbidden** to make changes/develop in a submodule of a repository: please refer to the original repository and make any changes there.
To clone a repository that contains a submodule use:

```
$ git clone --recurse-submodules <repo>
```

The command works even for repositories that do not contain submodules, thus can be safely used any time. To add a submodule to a repository:

```
$ cd <repo>
$ git submodule add git@gitlab.elettra.eu:path/to/<repo>.git deps/<repo>
```

Please note that it is mandatory to use the folder **deps** to store any submodule in a repository. The Makefile also requires updating in order to deal with the submodule. An excerpt taken from the cs/ds/e651 repository is shown below; the e621 depends on the cs/ds/usb2 device and cs/cls/usb2client class:

```
$ cd ~/src/gitlab/cs/ds/e621
$ git submodule add git@gitlab.elettra.eu:cs/ds/usb2.git deps/usb2
$ git submodule add git@gitlab.elettra.eu:cs/cls/usb2client.git deps/usb2client
```

The related Makefile looks like:

```
NAME_SRV = e621-srv
SRC_FILES = $(wildcard deps/usb2*/src/USB2*.cpp)
CXXFLAGS = -Ideps/usb2client/src -Ideps/usb2/src
LDFLAGS = -lusb-1.0
```

[4] See Working with INAU, https://docs.google.com/document/d/1Dp3JD9_RGkqZ8PL53etHjxA2maJoIXo1T-R88Wphhng

```
include ../makefiles/Make-9.3.4.in
```

From CVS to Git

The following instructions presume **working on a Tango device**, which is hosted in the cs/ds group, but can be easily extended to the other repository types. For a Tango device, first of all you need to clone the “common” makefiles, which lives at the same level in the hierarchy as each repository:

```
$ cd ~/src/gitlab/cs/ds
$ git clone git@gitlab.elettra.eu:cs/ds/makefiles.git
```

Please note that the Make.in generic placeholder is no longer supported and the developer has to specify the exact Makefile to include, e.g. Make-9.3.4.in at this time of writing.

Then you can proceed with your new project. The easiest way:

- Login into gitlab.elettra.eu
- Using the web interface, create the new repository in the cs/ds group in gitlab. Make sure you comply with the naming convention. Initialise the repository with an empty README.md so it’s immediately clon-able. A short one-liner in the *Project description* box helps immediate “view” when using the web interface; a longer description goes into the README.md file. Also, set the appropriate language in the *Topics* box (e.g. C++, Python, Bash…).
- Clone the new repository. The remote is automatically set-up.
    - `$ cd ~/src/gitlab/cs/ds`
    - `$ git clone git@gitlab.elettra.eu:cs/ds/<repo>.git`

  Add LICENSE, .gitignore… and push to the master
- Check out the current CVS repository, **without the tag release**
    - `$ cd ~/another/path/here`
    - `$ cvs checkout <repo>`
- Copy the relevant files from CVS into the new Git repo; **do not make any change** to the **code** in files
    - `$ cd ~/src/gitlab/cs/ds/<repo>`
    - `$ cp ~/another/path/here/Makefile .`
    - `$ cp -r ~/another/path/here/src .`
    - `$ cp -r ~/another/path/here/doc .`
    - `$ cp ...`
- Remove the CVS history records from the source code files
- The Makefile may require fixing the path to include the reference Make-x.y.z.in, now provided by the repository “makefiles”, see after
- **Do not** copy, or remove them afterwards, hidden files and CVS folders
- Add the files to staging
    - `$ git add Makefile src/ doc/`
- Commit
    - `$ git commit -m ‘Initial import’`
- Push your local repository into the remote
    - `$ git push origin master`
- The first import into Git deserves the annotated tag 1.0.0

- `$ git tag -a 1.0.0 -m ‘Initial release’`
- `$ git push origin 1.0.0`

Also, remember to ask the sysadm to freeze the old CVS repository!

Resources

- S. Chacon, B. Straub - Pro Git - Apress Free download at https://git-scm.com/book/en/v2
- A. Bogani - IT Group technical meeting - https://drive.elettra.eu/lib/2898410a-d053-462a-b5dd-20b442bb1b70/file/Presentazioni/130412_Git.pdf
- Git - The simple guide - https://rogerdudler.github.io/git-guide
- ...