git(lab) : ABINIT specificities

Standard names (projects, ID, branches)

Let us suppose that you are one of the ABINIT developers ...

Every “physical” developer of ABINIT using gitlab has his/her specific user_id (e.g. gonze). Moreover, an additional “virtual” developer, called trunk, is also defined.

On the ABINIT gitlab server, an ABINIT project can be defined, specifically for you, with address git@gitlab.abinit.org:user_id/abinit.git. In order to do this, log on https://gitlab.abinit.org, go to “Explore projects”, find “Trunk / Abinit” (+perhaps select “All”), then click on the “Fork” button. You need to contact Jean-Michel Beuken to have access to the ABINIT gitlab.

In order to start to work locally (not on the ABINIT gitlab server, but on your own machine), you should setup the SSH environment for gitlab, as described in the last section of this document. In particular on some machines you need to have an ssh agent running with your rsa key already available, so that git finds it when it runs ssh.

You have by default a master branch and a develop branch in your repository, but can create or work on other branches, following the philosophy of git. In particular, according to gitflow, this master branch will be of little use for the “physical” developers (including you), as we will see later, while the develop and release-* branches will be quite important for you.

The visibility level of the ABINIT projects of the user_id developers and of the trunk developer is “internal”. The trunk/abinit.git (well, only the master branch) is however mirrored on github.

(Quick start) Cloning from gitlab

In order to clone from the gitlab repository to your gitlab repository on your local machine (either for the first time, or in order to restart from a fresh copy of the gitlab repository), the recommended command is :
git clone gitlab:user_id/abinit.git -b develop
where user_id is to be replaced by the adequate name.

[Additional explanation : the normal “clone” command from the gitlab repository to his/her gitlab repository on his/her local machine is :
git clone git@gitlab.abinit.org:user_id/abinit.git
However, with the proper SSH environment, the following “clone” command can be used :
git clone gitlab:user_id/abinit.git
This will clone the master, while the usual working branch is the develop branch, that can be obtained directly with the above-mentioned command.]

After some modifications, using the git commands (e.g. git add, git commit ...), you can push on the gitlab server thanks to :
git push
[ or
git push –tags ( double-dash before tags )
if tags have been introduced...

In order for the modifications to be merged in the trunk, a merge request has to be issued, as described later.

The same might be done for other branches, created from the local develop one, using the commands:

```
git branch another_branch
```
```
git checkout another_branch
```

For pushing, the first time, use:
```
git push -u origin another_branch.
```

You are able to create your own additional branches, either locally or on gitlab. The name is your own choice.

**Git branches**

In the ABINIT+git workflow:

- By default, all branches pushed on gitlab might be tested on-demand by the user.
- In order to be merged in the trunk, the branch has to be **on-track**, as explained later. You have to issue a “merge request” on gitlab (to the trunk virtual developer).
- Only branches that are **on-track** and that have succeeded (or passed) all the automatic tests are considered for merge (as was the case previously).

The develop branch in the ABINIT naming scheme corresponds to the development stage. The release-* branches in the ABINIT naming scheme corresponds to release candidates or hotfix branches. It is slightly simpler than the gitflow naming scheme.

In the git philosophy, branches are often created, then merged, then destroyed. The ABINIT developer is encouraged to take advantage of this flexibility. However, many developers work only on one main project, and will already be satisfied by simply using their develop branch.

**How to trigger the action of buildbot ? (On-demand)**

The developer has to use the **git on-demand form on the ABINIT buildbot portal** (https://bbportal.abinit.org). The authentication is made with login/password of gitlab.

One important feature has to be taken into account : with git, each commit is characterized by a SHA-1 code (40 hexadecimal characters). The four first characters actually define the commit with 1 combination out of 65536 4-character combinations, while the five first characters define the commit with 1 combination out of 1048576 5-character combinations.

Gitlab knows the repository of the different users, the existing branches, as well as the SHA-1 code of each commit. The three following fields must be defined by the developer for buildbot to know which is the revision of ABINIT to be run:

1. “User” : choose one of the existing User_ID from the menu. Default value : user_id.
2. “Branch” : choose one of the existing User_ID from the menu. Default value : develop.
What is an "on-track" branch ?

For a develop branch to be on-track, a specific commit must be contained in the past history of the branch. This specific commit will be tagged in the trunk/develop branch (actually, corresponding to the definition of the start of the version).

For a release-* branch to be on-track, not only a specific commit must be contained in the past history of the branch, but also the commit that starts the next development version cannot be present. Only one among the release-* branches is on-track at any time.

As an example, suppose we were on ABINIT v9.0.0, and want to start preparing a release 9.0.1 (for release) and a new v9.1.0 (for development) :

- a branch entitled release-9.0 will be forked from the develop branch ;
- after this branching, the first commit in the release-9.0 branch will be tagged start-9.0, while the first commit in the develop is the commit d51ce822
- for a develop branch to be considered on-track (and eligible for a merge request), the commit d51ce822 must be present ;
- for a release-9.0 branch to be considered on-track (and eligible for a merge request), the commit tagged start-9.0 must be present, while it will be forbidden to contain the commit d51ce822 .

In complement to the start of a X.Y.Z version being tagged as “start-X.Y.Z”, the commit that ends some X.Y.Z version of ABINIT will be tagged “X.Y.Z”.

What is shown in the Buildbot Status table ?

(See the Buildbot Status at https://bbportal.abinit.org/#/status)

The Buildbot Status table show a summary of the recent results obtained by the test farm for active branches. If you want to see all the results, select the “filters” : all + all , then click on the update button. The default selection of bots, that is the “night” set, should be adequate for all merge requests. The selection capabilities of the Buildbot Status table are rather extended, you have to play a bit wit them.

How and when will the merge in the master branch be done ?

In order for your development to be incorporated, there should be a gitlab “merge request” from your specific user_id to the trunk. For most branches of user_id, the merge request will be to the trunk/develop branch. (See the list of merge requests at https://bbportal.abinit.org/#/mr)
However, when a release-* branch is ready to be merged, the “merge request” should target the corresponding trunk/release-* branch.

The master branch is only used by the trunk. So, never issue a merge request to trunk/master.

You are supposed to merge, inside your branches, specific tagged branches from the trunk user, in order to avoid divergences. These specific tagged branches will be advertised. As mentioned earlier, the presence of such tagged commits allows one to identify whether the branch is on-track, before an automatic testing to be done.

**How to synchronize with the "trunk" virtual user?**

In order to keep your branches user_id/develop or user_id/release-* up to date with those of the trunk virtual user, you should first define git@gitlab.abinit.org:trunk/abinit.git as a remote:

```bash
git remote add trunk gitlab:trunk/abinit.git
```

Then, in order to synchronize, in a first step, issue:

```bash
git fetch trunk
```

then, if the develop branch is to be updated, supposing it is checked out, merge the trunk/develop in your develop:

```bash
git merge remotes/trunk/develop
```

You can combine the last two commands in one as:

```bash
git pull trunk develop
```

If, on the contrary, a new branch (e.g. a release branch, let's says 8.8 to fix the ideas) has to be created:

```bash

git branch release-8.8 start-8.8.1 (this creates the branch release-8.8 from the start-8.8.1 tag)  
git checkout release-8.8  
git merge remotes/trunk/release-8.8  
git push -u origin release-8.8
```

That's it! You can now make modifications in your release-8.8, then issue a merge request to the trunk/release-8.8.

**Additional info: how to not mess with your branches? ... show-branch ...**

To see which branches are present in your current copy of abinit type: git show-branch -a
The syntax is compact but a bit barbaric: the first section gives you the branches which are actually present, with a star for the one currently checked out. The other branches are offset by 1 space.

```
*   -    [develop] Merge branch 'develop' of gitlab:trunk/abinit into develop
  ! [master] Minor modif, to initialize v8.7.3 modified: KNOWN_PROBLEMS
  ! [origin/HEAD] Minor modif, to initialize v8.7.3 modified: KNOWN_PROBLEMS
  ! [origin/develop] Merge branch 'develop' of gitlab:trunk/abinit into develop
    ! [origin/master] Minor modif, to initialize v8.7.3 modified: KNOWN_PROBLEMS
  ! [trunk/develop] Import inside gitlab develop, the modifications from v8.6.2 to v8.6.3
    ! [trunk/master] Minor modif, to initialize v8.7.3 modified: KNOWN_PROBLEMS
```

The second section shows commits, and can tell you the relation between the branches. Commits are marked with a + in each column for each branch they are included in, and merges with a -. Note that the master branches are never used, the top commit is a merge of trunk into the current develop, and the second commit is contained in the currently active, the origin/develop (of which it is just a clone), and the trunk/develop branches.

### Additional info: Setup of the SSH environment

In order to avoid typing your password every time you issue a command that accesses gitlab, you have to introduce your public keys in your profile (https://gitlab.abinit.org/profile/keys).

On your local machine, generate a ssh key of RSA type only WITHOUT passphrase:

```
ssh-keygen -t rsa
```

and call it `id_rsa_gitlab`.

Then add a section in the `~/.ssh/config` file:

```
host gitlab
  Hostname gitlab.abinit.org
  User git
  KeepAlive yes
  IdentityFile ~/.ssh/id_rsa_gitlab
```

and then, copy the public key `id_rsa_gitlab.pub` on gitlab.
Now, you can use (on your local machine) the following syntax:

```bash
git clone gitlab:user_id/abinit.git
```

instead of the above-mentioned

```bash
git clone git@gitlab.abinit.org:user_id/abinit.git
```

To be sure the key is proposed each time git calls ssh, you can use ssh-agent:

```bash
ssh-agent # this starts the agent, and provides the process id
# execute the 3 lines of commands that ssh-agent proposes, e.g.
SSH_AUTH_SOCK=/tmp/ssh-ngsERHER3K1HS/agent.15589; export SSH_AUTH_SOCK;
SSH_AGENT_PID=15590; export SSH_AGENT_PID;
echo Agent pid 15590;
ssh add ~/.ssh/id_rsa_gitlab # add the corresponding ssh key for gitlab
```

For further details, please consult the official documentation

<https://gitlab.abinit.org/help/ssh/README.md>