10 steps to hike ABINIT

1. Installing and learning git

For an overview of git, see Wikipedia. Download git from the official git download page. Installing git is extremely easy on many platforms. To learn git, refer to the official git documentation, or the many tutorials available on the Web.

2. Getting the source tree

Follow the instructions described in the ABINIT Web pages describing the specificities of the use of git for ABINIT.

3. Running makemake

A certain number of files are generated by scripts. They cannot be managed by git, as they would cause a huge number of conflicts to occur at each merge operation. This means that they are missing just after you have the source code fetched, and the working tree is not yet functional. The build system provides a series of scripts which are in charge of producing a buildable source tree. They are located in the config/scripts directory, and their front-end is called makemake.

In order to do their job, these scripts will require that the following software be present on your machine:

- Perl ≥ 5.005 (5.8.8 recommended)
- Python ≥ 2.4 (2.5 in test)
- GNU M4 ≥ 1.4.4 (1.4.9 highly recommended)
- Autoconf ≥ 2.57 (2.61 highly recommended)
- automake ≥ 1.9 (1.10 highly recommended)
- Libtool ≥ 1.5.0 (1.5.22 highly recommended)

The recommended versions do not only fix bugs, they will produce better configure scripts. Please install either the minimal versions or the recommended ones. Do not mix them.

After checking that all these pieces of software are installed, please run the makemake script. Just type:

```
./config/scripts/makemake
or
./*/*/makemake
```

You will have to run it once each time you get a source tree, and each time you make a structural change to the sources.
4. Configuring ABINIT

There are two ways to configure ABINIT:

- giving command-line options to the configure script; a full list can be obtained by typing ".configure –help" from the top of the source tree;
- writing a config file.

Using a config file is a more convenient way to set-up options for ABINIT. And anyway, you can override it with command-line options at any time. A self-documented template is provided in doc/config/build-config.ac. Please follow the instructions contained therein.

This config file will be read by the configure script each time you want to build ABINIT, regardless of the version. This is good news for you, because it means that you can write it once and for all, adjusting it from time to time when your configuration evolves. The naming convention adopted makes it usable on platforms where a home directory is shared among several computers (e.g. through NFS).

Now we suggest you to create a build directory inside the ABINIT source tree. Let's call it tmp. Just type:

```bash
mkdir tmp && cd tmp
```

Then you can configure ABINIT by typing:

```bash
../configure
```

The configure script will display a lot of messages related to its attempts of auto-detecting your configuration. The command-line options you give it are hints for things it cannot guess by itself. At the end of the process, it will generate all the makefiles you need to build ABINIT.

5. Building ABINIT

To build ABINIT, just type “make”. This may take between 3 minutes and 1 hour, depending on your configuration and the level of optimisation you are requesting. If you want more information about what you can build, just type make help.

- a. Sequential binaries

If you want to (re)build a sequential binary, just type “make <program_name>”.

- b. Parallel binaries

**WARNING: this section is outdated.**

The MPI libraries available within most Linux distributions still come with Fortran 90 support disabled.
This means that you will have to build MPI yourself or have it built for you if you want to use the parallel binaries of ABINIT. For more information, you may have a look at the MPI options provided by configure.

6. Enabling external libraries

You may try to enable or disable optional features, either editing the config file (remember step 4), or providing command-line arguments to the configure script. Five libraries are enabled by default: NetCDF, ETSF I/O, LibXC, BigDFT, Wannier90. To disable them, use the appropriate “enable” flag in your config file, or use options like “--disable-etsf_io”.

XMLF90 library support is back from Spring 2007. The library should build well and be usable, but we have just made some tests on the PC platform. To enable it, just uncomment the “enable_xmlf90” flag in your config file or use the “--enable-xmlf90” option. Please note that this library is not well-maintained anymore, and that current development efforts are now found in the FoX library. Preliminary FoX support has been added for developers, but still require the current ABINIT bindings to be rewritten. Scalapack library might be enabled also, but is not yet considered in production on all machines.

7. Running tests

In order to perform tests, just go to the tests directory and type “./runtests.py”. You will get some help on how to perform them.

8. Sending patches

This step is only for developers using tarballs (not git)

When you have found and fixed a bug, the best is to send back your corrections upstream, so that they may be included in the upcoming releases. Here is what we suggest you to do:

1. Make a full backup of your changes (we'll say they are in NEWDIR).
2. Untar a pristine version of the source tree you have used for your fixes.
3. Create a patch with the command:

   <code>diff -urN OLDDIR NEWDIR > abinit-x.y.z-myfixes.patch</code>

   1. Compress the patch if it is too big to be sent directly by email.
   2. Attach your patch to a message explaining what you've done and send it to the project leader.

9. Commiting bug fixes

When you want to have bug fixes quickly included in the next release of ABINIT, remember to commit your changes in your git branch, then issue a merge request to the trunk.
10. Generating a source tarball

Being able to generate a source tarball is important when you want to build modified sources on a machine which does not provide the required build environment (i.e. the GNU Autotools).

To create a tarball, just type:

```
make dist
```

Generating the source tarball will take between 1 and 3 minutes on a modern PC. Once done, you'll find the file in the current directory.