Configuring the build of Abinit

Please read the Conventions and formatting page at least once before everything else.

Tutorials for beginners

- installing_abinit.pdf

The configure script

The configure script is a portable shell-script generated by Autoconf from a file called configure.ac or configure.in. It contains a sequence of commands depending on the software package it belongs to and the purpose of which is to detect the components needed to build this package.

The commonly-accepted default behavior is that the configure script should be able to provide working build parameters when run without option. However, to let users customize interact with it and customize its behavior, Autoconf provides a mechanism to transmit command-line options to configure. There are 2 kinds of options:

- switches, to turn simple features on and off, available through the enable_* shell variables and -enable-* / --disable-* on the command line;
- optional packages, to enable or disable support for external dependencies and complex features, available through the with_* shell variables and --with-* / --without-* on the command line.

In addition, the configure script comes with built-in standard options to customize the installation of the package and lets the developers mark some environment variables as important.

Whenever possible, we refer to the options of the configure script as shell variables, which is the way they appear in the config files of Abinit. Their names contain letters, digits, and underscores. On the command line, all underscores are replaced with dashes and option names are prefixed with 2 dashes. As an example, the option to specify the Fortran compiler vendor appears as with_fc_vendor in the config file, while it is typed --with-fc-vendor on the command line.

To obtain the full list of available options and their documentation, just type:

./configure --help

from the top source directory of any Autoconf-powered software package. With Abinit, you will get a long list of options and environment variables. Reading it at least once is a very good way to become aware of the possible optimizations offered at configure-time and get some familiarity with the underlying philosophy of the build system.
Environment variables

If you have run the `./configure --help` command, you will have already noticed that Abinit can be configured through various environment variables. A few others, not listed by `configure`, are also of great importance, and we will describe them in this section.

Using environment variables

Setting an environment variable is usually done using the variable name directly, e.g.:

```
LD_LIBRARY_PATH="/opt/my_package/lib"
```

while querying it is done by prefixing its name with a dollar sign, e.g.:

```
echo $LD_LIBRARY_PATH
```

Quite often, the situation happens to be a mix of both, e.g.:

```
export LD_LIBRARY_PATH="/opt/my_package/lib:$LD_LIBRARY_PATH"
```

Availability of environment variables

In order to be available beyond the current shell context, variables often have to be exported. This is done using the `export` instruction with Bourne-like shells, and `setenv` with C-like shells. If the `configure` script does not take an environment variable into account, it is likely because you did not export it.

PATH

The `PATH` environment variable tells the system where to look for relevant programs. It consists in a column-separated list of directories containing executable files. The following is a typical value that can be obtained from the command line:

```
$ echo $PATH
/home/abinit/bin:/usr/local/bin:/usr/bin:/bin
```

The `configure` script looks for various programs, in order to decide which features should be enabled and which ones not. In particular, if you use Abinit Fallbacks, you should make sure that your `PATH` contains the directory containing the `abinit-fallbacks-config` program before running `configure`, e.g.:

```
$ export PATH="/fallbacks/install/directory/**bin**:$PATH"
```
LD_LIBRARY_PATH

Equally essential, LD_LIBRARY_PATH is to libraries what PATH is to programs, and works exactly the same: it is a column-separated list of directories containing shared libraries. On Unix systems, these libraries have a “.so” extension. On MacOS, their extension is “.dylib”. On windows, it is “.dll”. A typical setting with a strict Bourne Shell is:

```
$ LD_LIBRARY_PATH="/path/to/abinit/dependency/**lib**:$LD_LIBRARY_PATH"
$ export LD_LIBRARY_PATH
```

Please note that LD_LIBRARY_PATH is called DYLD_LIBRARY_PATH on MacOS.

**IMPORTANT**

LD_LIBRARY_PATH must be set to the same value both before configuring the build of Abinit and before running the corresponding executables. These 2 actions can happen at very different times, hence the risk of forgetting to set the variable again. Misconfiguring LD_LIBRARY_PATH is actually a common mistake beginners make when learning Abinit and the source of many questions asked on the Abinit Forums.

LIBRARY_PATH

If you use GCC compilers, you may sometimes want to use the LIBRARY_PATH environment variable as well, in particular if the build fails while LD_LIBRARY_PATH has been correctly set. Contrary to the latter, LIBRARY_PATH is only used by the compiler at link-time and will be ignored when running an Abinit executable.

PYTHONPATH

Python is usually installed by system administrators in a standard way on most computers. However, if you use custom Python modules that you have installed yourself, please make sure that the PYTHONPATH variable is properly set before configuring Abinit.

Language-specific variables

From time to time, some compilers complain or do not display important messages because language support is improperly configured on your computer. Should this happen, we recommend you to type the following:

```
$ LANG=C; LC_ALL=C
$ export LANG LC_ALL
```

This will reset the language support to its most basic defaults and will make sure that you get all messages from the compilers – in English, of course.

Tips for ABINIT users and developers - https://wiki.abinit.org/
Using a build directory

We strongly recommend you to systematically use a build directory different from the source tree when compiling Abinit. Usually, this is done by creating a new subdirectory at the top of the Abinit source tree and is as simple as typing:

```bash
cd ~abinit && mkdir my_build_dir && cd my_build_dir && ../configure
```

Besides being a standard way to proceed with most Autotools-based software packages, this is much cleaner than building directly in the source tree, since it clearly separates the built files from the source files and keeps the latter untouched. If your build fails for any reason, you can always restart with the guarantee of a pristine source tree by simply removing the contents of the build directory or creating a new one.

If you wish, you can even build Abinit fully outside its source tree by creating a build directory anywhere and using the `--srcdir` option of `configure`.

**Naming convention for developers**

If you are a developer of Abinit, we recommend you to name your build directory 'tmp' or to use 'tmp-' as a prefix to its name, in order to have it automatically ignored by the Version Control System.

Using a config file

When using some options repeatedly, it quickly becomes tedious to type them again and again on the command line. On the other hand, if you only compile Abinit once in a while, you will likely forget the options that you used previously between 2 builds. This is why we provide an alternative way to provide options to the `configure` script, through the use of config files.

In order to accommodate as many situations as possible, we have designed a flexible mechanism to handle config files:

- first, the build system looks into the build directory for a config file, which lets you import parameters for a specific build;
- if not found, it looks into the source directory, which allows you to use build parameters related to a particular version of Abinit;
- if still not found, it looks into the ~/.abinit/build/ directory, which acts as a provider of default parameters corresponding to your particular preferences;
- finally, it looks into the /etc/abinit/build/ directory, to let system administrators provide system-wide parameters for a specific computer.

Of these 4 possibilities, the first file encountered is processed and the build system goes on without looking for more files. In other words, config files are not cumulative. For example, if a user has a ~/.abinit/build/my_host.ac, the system-wide defaults found in /etc/abinit/build/my_host.ac will always be ignored.
Since many developers of Abinit work on shared filesystems, the build system looks by default for a file called `hostname.ac`, where `hostname` represents the result of the following command, that you can type in a shell to know how to name your config files (do not type the dollar signs):

```
$ hostname
wiki
$
```

In the current example, you would name your config file `wiki.ac` if you want the build system to take it into account by default.

This operation has to be repeated once on every computer where you want to build Abinit using a config file. We chose this strategy to make sure that the build system will always take the correct config file even in the case several computers would share the same home directory.

You will find detailed instructions on how to set the various options of the configure script in the template file `~abinit/doc/build/config-template.ac`, including how to have your config file included automatically each time you build Abinit. There are only 2 undocumented options in this template:

- the `--with-config-file` option, which lets you use arbitrary names for your config files, as long as you specify it each time on the command line;
- the `--disable-config-file` option, to force Abinit not to use any config file.

For obvious reasons, they are the only options that cannot be stored in a config file.

**Tuning configure options**

*Important tip*

The options of `configure` are fully documented in the `~abinit/doc/build/config-template.ac` file. Even if the wiki is updated quickly to reflect the changes made to this file, there may sometimes be a slight delay before both documents are synchronized. If you find a discrepancy between the wiki and `config-template.ac`, please let us know as soon as possible so that we can fix it - or fix it yourself if you have write permission on this wiki.

The build system of Abinit is quite modular and is composed of logical blocks addressing each a specific configuration issue. The following documents will help you tune the build parameters depending on what you want to adjust.

*Configuring compilers* helps you select C, C++ and Fortran compilers, and find the best compromise between performance and reliability for your architecture. It includes a section on preprocessors and the corresponding platform-specific issues.

*Configuring multicore architectures* shows how to configure OpenMP-, MPI-, and GPU-enabled architectures, as well as how to force a build to remain serial.

*Configuring linear algebra* explains how to configure linear algebra support for your particular situation.
Configuring Fast Fourier Transforms guides you to select the best FFT algorithm for your particular situation.

Configuring feature triggers provides you with useful advice on how to enhance the capabilities of Abinit with external packages.

From: https://wiki.abinit.org/ - Tips for ABINIT users and developers


Last update: 2019/03/28 14:23