Fallbacks

Objectives

In case some dependencies are missing on your computers, Abinit provides fallback libraries that you can build and install from their sources before compiling Abinit itself. Although they do not offer the same level of reliability and performance as packages that a skilled administrator would build, test, and install, they will let you preview some advanced features and decide which ones are the most important to you and would be worth installing fully on your system.

Fallbacks are also useful for developers who want to test a new version of an external dependency before setting it as a default for future releases of Abinit and have it deployed on the Abinit Test Farm. Except when this new version fixes a bug affecting significantly the accuracy and/or stability of production calculations, such an operation is always performed on a development version of Abinit and tested by several developers before being made available to the users.

IMPORTANT NOTE
We do not perform any make check during the build, since the process has been automated and would stop at the first error. Unfortunately, the packages wrapped by the fallbacks system do not provide complete nor fully reliable test suites, and failures are often difficult to analyze. This is why the fallbacks should not be used for production calculations.

Project home

The Abinit Fallbacks project is stored on the main Abinit Forge and is owned by the Test Farm Team. Read-only access is provided by default.

Here are the most relevant links to get useful information about the project and its status:

- https://gitlab.abinit.org/buildbot/abinit-fallbacks
- https://gitlab.abinit.org/buildbot/abinit-fallbacks/issues
- https://gitlab.abinit.org/buildbot/abinit-fallbacks/milestones

For historical reasons, a secondary repository is still kept on Launchpad, where the contents of the home repository are periodically mirrored:

- https://launchpad.net/abinit-fallbacks

Minimum requirements

The configure script of the Abinit Fallbacks downloads the source packages it needs from internet. In
order to have this feature properly working, you will thus need:

- an internet access;
- the `Wget` and/or `cURL` utilities.

If your network configuration does not allow for the downloading of remote packages, you will have to place the source tarballs in the `~/.abinit/tarballs/` directory manually.

Since the recommended way of getting the Abinit Fallbacks is by cloning its repository from the Abinit Website, we highly recommend you to install the GNU Autotools on your system first:

- `Autoconf`;
- `Autoconf Archive`;
- `Automake`;
- `Libtool`.

To build the fallbacks, you will also need a working development environment, including in particular:

- a working C compiler;
- a working C preprocessor (usually provided along with your C compiler);
- a working Fortran 2003-compliant Fortran compiler.

Most Fortran compilers published after 2012 are Fortran 2003-compliant. Earlier versions may work but are known to produce bugs and issues difficult to diagnose.

### Building the fallbacks as a separate package (recommended)

#### Overview

Since the fallbacks evolve at a very different pace from Abinit and have very different objectives, they are available as a separate package. Typically, in one year, there will be 3 minor releases of Abinit, while 2 or 3 of the fallbacks at most will be upgraded asynchronously, mainly during active development phases. In parallel, the build system that wraps their installation will be refactored, upgraded, and tested.

The independent fallbacks can be downloaded either from the Abinit Website or Launchpad, at your preference. You can either download the source tarballs released with Abinit or clone one of the Git repositories directly.

#### Getting the source code from the Abinit Website

The official Abinit Fallbacks project page is there: [https://gitlab.abinit.org/buildbot/abinit-fallbacks](https://gitlab.abinit.org/buildbot/abinit-fallbacks)

If you are an active Abinit Developer, we highly recommend you to configure your access before...
anything else. We will suppose this is the case from now on.

To get the latest version, just type the following:

```
git clone abinit-forge:buildbot/abinit-fallbacks
```

### Getting the source code from Launchpad

The Launchpad mirror project page is there: [https://launchpad.net/abinit-fallbacks](https://launchpad.net/abinit-fallbacks)

It is provided for historical reasons and convenience, but all our activities related to the fallbacks take place on the Abinit Website.

The first time you download the fallbacks from Launchpad, we highly recommend you to set your environment as described in the Access configuration page. In the following, we will suppose you did so.

To get the latest version of the fallbacks, type:

```
git clone lp:abinit-fallbacks
```

### Downloading a source tarball

When a new major or minor version of Abinit is about to be released, we usually package a snapshot of the fallbacks and distribute it as a source tarball. This file is called `abinit-fallbacks-X.Y.Z.tar.gz`, where X and Y correspond to the highest X.Y Abinit version for which compatibility has been tested, and Z is a patch level starting from 0. Please note that Z is used exclusively by the fallbacks and has nothing to do with Abinit. For instance, the tarball `abinit-fallbacks-8.4.2.tar.gz` contains fallbacks compatible with all 8.4.* versions of Abinit.

Source tarballs generated so far can be downloaded from Launchpad ([https://launchpad.net/abinit-fallbacks/+download](https://launchpad.net/abinit-fallbacks/+download)), but this location might change in the future.

Since Abinit dependencies evolve slowly, a fallbacks bundle with version X.Y.Z is often compatible with the X.(Y-1).* and X.(Y+1).* versions of Abinit.

### Using a snapshot from the Abinit source tarball

For convenience, and though fallbacks are now fully decoupled from the Abinit build system, source tarballs of Abinit include a snapshot of the fallbacks source code in the corresponding `fallbacks/` subdirectory. You may use this version of the fallbacks if you wish, but please note that it is usually not up-to-date with respect to the official fallbacks repository.
Build steps

If you have downloaded the source with Git, you must have the `Autotools` installed on your computer and run `./autogen.sh` from the top source directory. If you have downloaded a source tarball, this step has already been performed for you, so that you do not have to install the Autotools.

We highly recommend you to create a build directory and go there before going further. Please note that if its name is “tmp” or starts with “tmp-”, it will be ignored by Git and removed when running the `wipeout.sh` script. Using a build directory will ensure that the source code remains completely untouched all along the build and is a very important aspect in producing high-quality software.

Before building the fallbacks, run `../configure --help` from the build directory and read carefully the parameter description provided by the script. Select the options which best fit your needs and run `configure` with them.

The build of the fallbacks follows the standard `configure / make / make install` procedure and will look like this:

```bash
mkdir tmp-mybuild
cd tmp-mybuild
../configure --prefix=/install/path ... options ...
make
make install
```

where you replace `/install/path` by the actual directory where you want to install the fallbacks. You may as well replace `tmp-mybuild` by whichever name suits you best.

You can try to use `make -j` to accelerate the build but may experience random build errors because some packages do not support parallel builds and may spuriously break the process. If you are annoyed by such errors, please report the issue to the developers of the faulty package directly, as we cannot do anything about it ourselves. Or better: fix it and send them a patch.

Building the fallbacks with Abinit

For convenience, a release of the fallbacks comes with the source tarball of Abinit. It is possible to build it along with Abinit, although much more efficient to build it and install it separately, since you do not have to rebuild the fallbacks each time you wish to make a new build of Abinit. Please note that the version of the fallbacks you may find within the Abinit source tree will always be outdated.

Building the fallbacks within Abinit is mainly of interest to developers who want to experiment with compile flags and explore issues related to compilers and portability. It can also help - to a reduced extent only - users who are stuck within very rigid configurations and restricted build environments. Here is the way to proceed:

```bash
tar xzvf abinit-x.y.z.tar.gz
```
cd abinit-x.y.z
mkdir tmp-build
cd tmp-build
../configure \
  --prefix="/abinit/install/path" ... options ... \
# Abinit drops a build script into the transient/ subdirectory and configure 
# fails because the fallbacks are missing and cannot be tested.
cd transient
# Edit the build-abinit-fallbacks.sh script and set its install prefix 
# as instructed therein
./build-abinit-fallbacks.sh
make
make install
cd ..
../configure --prefix="/abinit/install/path" \
  --with-fallbacks-prefix="/path/to/fallbacks/prefix" \
  ... options ... 
make -j8  # For example, if you have at least 8 cores available
make check
make install

What happens in this case is that we first configure Abinit, so that it generates all the necessary 
information to build the fallbacks for missing external dependencies. The configure script of Abinit 
always gives priority to properly-installed external dependencies and will only select the minimal 
necessary fallbacks, in order to save time, trigger less bugs, and aim at maximum reliability. At this 
stage, the configure script has to fail, because it cannot anticipate the successful build of the 
fallbacks.

Once the information is available to the configure script of the fallbacks, we can build them within the 
Abinit source tree and install them outside. Next, Abinit uses them during its own build and we can 
install it normally after checking that everything is fine. In further builds with the same compiler, the 
same fallbacks can be reused, i.e. no need to rebuild them.

One line must be changed in the transient/build-abinit-fallbacks.sh script for it to work. This script can 
indeed not guess where you want to install the fallbacks. This is why you have to replace the PREFIX 
keyword by the full installation path of your choice. Please note that this path must be absolute, i.e. 
start with a slash. The line to change is located at the top of the script:

#!/bin/sh

# Init
fallbacks_prefix="PREFIX"

# Check that the installation prefix has been set
if test "\${fallbacks_prefix}" = "PREFIX"; then
echo "Please edit \${0} and set"
echo "the 'fallbacks_prefix' variable at the top of the file"
echo "to the absolute path where you wish to install the fallbacks."
echo "Then run again this script."
exit 1
fi
Using installed fallbacks

The install procedure of the fallbacks is designed for the Abinit Test Farm and will remain so until the last fallback is abandoned. It might work properly in various other configurations, but adapting the path hierarchy to your own needs is entirely up to you. Feel free to use symbolic links if you need a different organization of the files.

The directory hierarchy is the following:

```
FALLBACKS_HOME
 |  
+- vendor1
 |  +- version1
 |   +- variant1
 |     +- package1
 |     |   +- bin
 |     |   +- include
 |     |   +- lib
 |     |   +- share
 |     |   +- package2
 |     |   +- ...
 |     +- variant2
 |     +- ...
 |   +- version2
 |     +- variant1
 |     +- ...
 |   +- version3
 |   +- ...
+- vendor2
 |  +- version1
 |  +- ...
```

where files are ordered by compiler vendor, then compiler version, and finally build variant. Each variant corresponds to a set of build parameters and is summarized by a keyword:

- `gpu` for a GPU-aware configuration;
- `mpi` for a MPI-aware configuration;
- `omp` for an OpenMP-aware configuration;
- `sdebug` for a serial configuration with enhanced debug flags;
- `serial` for a serial configuration with safe optimizations.

For example, a build of the fallbacks for an environment optimized for GCC 5.4.0 and MPI would be installed in `FALLBACKS_HOME/gnu/5.4/mpi`, and a debugging version for ifort 17.0 would go to
FALLBACKS_HOME/intel/17.0/sdebug.

Other variants with free-form keywords may appear depending on the needs of the Abinit Developers and Maintainers, in particular on the Abinit Test Farm.

The PREFIX install path which is specified when building the fallbacks usually corresponds to a FALLBACKS_HOME/vendor/version/variant hierarchy. It is however flexible enough to adapt to different situations.

Once the fallbacks are installed, the build system of Abinit can make use of them thanks to the PREFIX/bin/abinit-fallbacks-config script, which provides all kinds of information on how to access the installed packages. To get more information about what this script can do, just run it without arguments.

The support for fallbacks in the Abinit build system is enabled when using the –with-fallbacks=DIR option of the configure script of Abinit. It is possible either to specify a PREFIX-like directory or a FALLBACKS_HOME-like one. In the latter case, Abinit will define the vendor/version/variant part of the install path itself, depending on the current build configuration.

Troubleshooting

The configuration of a package fails

Errors may appear at configure time when you build the fallbacks. In order to help you diagnose the problems, most packages provide a config.log file. You will find it in ~fallbacks/sources/package-version/config.log, where you replace package by the name of the package and version by the corresponding version. Open this file and look for the last occurrence of the “failed” keyword, this is usually the place where you will get the most useful information. Looking around the corresponding line will give you useful hints about what made the configure script abort. With a little bit of practice, you will understand and fix most errors relatively quickly.

"make" fails

Most errors appear at compile time. When make fails, first look at the error message. Many times, you might have been running make -j on a package which does not support parallel builds. The first thing to do is to re-run make serially. If the error still appears and is about a missing file, look on internet for the package which provides this file and check whether it is correctly installed on your computer. If you fail in finding a solution, look on internet for the error message. If you still do not see how you could solve the issue, look at the bug tracker of the package. If you still find no hint of the problem, then contact the developers of the package through their preferred channel.

The compiler complains about invalid characters

When the compiler complains about invalid characters, in particular with Fortran, it is very likely a problem of preprocessing. Since the Fortran standard does not have any specification regarding preprocessing, most packages use the C preprocessor to prepare Fortran files before compilation. If
provided with incorrect options or wrongly set-up, the C preprocessor will leave C comments in the source files, that a Fortran compiler will be unable to process. This usually happens when the preprocessing is done separately from the compilation. A frequent solution is to play with the FPP and FPPFLAGS environment variables until the Fortran compiler is satisfied. For FPP, look at all executables named cpp on your system. For each of them, look at their documentation to find the options that will minimize the output and remove stale comments.

**The program complains about invalid instructions at run-time**

Among run-time errors, the complaint of invalid instructions by the system is quite typical on heterogeneous clusters. What usually happens is that the front-end node where the program has been compiled has a slightly different architecture than that of the nodes where the program runs. In this case, the solution is:

- either to ask the system administrator to provide you with the characteristics of the processors of all types of nodes and/or the optimization flags compatible with them all;
- or restrict the execution of the program to the nodes which are compatible with the optimization flags the program has been built with.

**Frequently Asked Questions**

**Do I need to build the fallbacks with the same build flags as Abinit?**

No, this is not mandatory, and not even recommended. All packages forming part of the fallbacks are **external to Abinit**. As such, each of them will work better with individually configured flags. These flags can be passed to the configure script of the fallbacks, e.g. `CFLAGS_LIBXC="-g -O3 -ffast-math"`. In any case, you should be very careful when passing aggressive optimization flags, as they may lead to numerically wrong results difficult to pinpoint.

**When do I need to build the fallbacks?**

If you want to use the fallbacks instead of fully external packages, you should always build and install them before configuring Abinit. Building them with Abinit is strongly discouraged and will only provide you with a partial build of the fallbacks.

More specifically, you will have to build and install a custom version of the fallbacks every time you change compiler vendors or compiler versions, since Fortran modules are incompatible even with a different version of the same compiler. But once they are installed, you can reuse the fallbacks as many times as you want without rebuilding them, provided you keep using the exact same configuration.

If the compiler vendor and version are strictly identical, you may manage to use a variant of the fallbacks with a different variant of Abinit, with a high risk of inaccurate or numerically wrong results, as well as random crashes at run-time.
Who is in charge of fixing the fallbacks?

If you encounter an issue which is obviously a defect related to how the packages are built and/or installed, it should be reported to us through the Gitlab Issue Tracker of Abinit.

If the problem is related to one or more packages, the only way to resolve it is that you collaborate with their respective upstream developers. Always remember that only complaining is counter-productive. Be kind and polite, provide as many details as possible in your description of the problem, and do your best to propose solutions. Even if they are sometimes overwhelmed, the developers of these packages will be grateful to be informed of possibly important issues related to their projects. Once the issues are solved, we will gladly upgrade the fallbacks to include the fixed versions of these packages.

Contributing to the fallbacks

Advanced developers may contribute to the Abinit Fallbacks. Please read our Hacking the Fallbacks page if you are interested.