



ABINIT School 2019  
January 21 – 25, 2019  
Bruyères-le-Château, France

---

## Installing ABINIT on a Laptop – Hands-on session

By following this step-by-step procedure, you should understand how to install ABINIT on a laptop using a standard Linux distribution. You will probably need an *Internet connection*.

First of all, it is mandatory to have a Fortran compiler, a MPI library and a linear algebra library installed. If not, you need to install these packages:

- On Ubuntu : `sudo apt-get install gfortran openmpi liblapack-dev`
- On Fedora/CentOS : `sudo yum install gcc-gfortran openmpi-devel lapack-devel`
- On macOS : `sudo port install gcc8 openmpi lapack`  
or `brew install gcc openmpi lapack`

In the directory dedicated to this tutorial, you should find the `abinit-x.y.z.tar.gz` file. If not, download this file from ABINIT website ([www.abinit.org](http://www.abinit.org)) and then extract it.

Enter in the `abinit-x.y.z` directory.

It is highly recommended to create a separate directory to compile the code and store the object files. Let's create the `build` directory by typing:

```
mkdir build
```

Enter in the `build` directory.

Let's first try to configure the build automatically.

Just type (in the `abinit-x.y.z/build` folder):

```
../configure
```

After a few seconds, you should obtain the **configuration report**.

What do we see in it (usual case)?

```
* OpenMP enabled : no  
* MPI      enabled : no  
* TRIO     flavor = none  
* DFT      flavor = none
```

The parallel build (MPI and *openMP*) is not activated; no plugin has been activated (no “Transferable I/O” (TRIO) plugin, no DFT plugin).

This is not optimal! Let's change this.

First of all, we will create a *configuration file*; this is more convenient than having a long command line to configure the code.

Type the command (in the terminal):

```
hostname
```

You should get the name of the computer.

Something like: `computername` or `computername.domain.domain`

Let's create a file named `computername.ac` in the `build` directory. Use your favorite editor for that.

---

Then, in this configuration file, add the following lines, and retry `./configure`:

```
enable_mpi="yes"  
enable_openmp="yes"
```

If the configuration script goes to the end, MPI has been detected.

If not, you should add a line locating the MPI installation.

A possibility is to locate the `mpif90` file (type: `which mpif90`). You should find it in a place like: `path_to_MPI/bin/mpif90`

Then add the following line in the configuration files and relaunch `./configure`:

```
with_mpi_prefix="path_to_MPI"
```

Note: another possibility is to use directly `mpif90` as Fortran compiler. For that, add the following lines in the configuration file:

```
FC="mpif90"  
CC="mpicc"  
CXX="mpicxx"
```

This time, everything should be OK; MPI has been detected.

---

Now, let's try to add the ABINIT plugins.

For this tutorial, let's first try to add *netCDF* ("Transferable I/O"=TRIO plugin). Add the following in the configuration file and re-run the configure script:

```
with_trio_flavor="netcdf"
```

If the configuration script goes to the end, you should see the following in the configuration report:

```
* TRIO flavor = netcdf
```

If not, and if you have an Internet connection, you should see:

```
* TRIO flavor = netcdf-fallback
```

This means that ABINIT will "fall back" on its own *netCDF* package, downloading it from the Internet.

If the configuration script stops with an error (no Internet connection), we have to manually provide the *netCDF* package.

In the directory dedicated to this tutorial, you should have a [ABINIT-PLUGINS](#) directory. Copy the *netCDF* file ([netcdf-4.1.1.tar.gz](#)) into the `$HOME/.abinit/tarballs` directory; or download it from [www.abinit.org/fallbacks](http://www.abinit.org/fallbacks)). Then use the *configure* script again.

At this stage, the detection of *netCDF* should be OK.

Let's now add one additional plugin: *libXC* (library of exchange-correlation functionals).

Copy the [libxc-x.y.z.tar.gz](#) file into `$HOME/.abinit/tarballs` (or download it from [www.abinit.org/fallbacks](http://www.abinit.org/fallbacks)).

Add the following in the configuration file:

```
with_dft_flavor="libxc"
```

In the final configuration report, you should see (ABINIT will “fall back” on its own *libXC* package):

```
* DFT flavor = libxc-fallback
```

---

Everything is almost ready for the compilation  
Just type:

```
make mj4
```

...and wait... wait..

Depending on your computer, the compilation can take 1 min. or 15 min.

At the end of the compilation process, you should get some messages explaining how to check the installation by running the automatic tests. Let's try if ABINIT can run; just type:

```
cd tests && ../../tests/runtests.py fast
```

And wait for the report.

Many other automatic tests are available. You can get the list with:

```
../../tests/runtests.py -show-info
```

If you want to install ABINIT in your PATH to have it accessible from everywhere, you can type, in the “build” directory:

```
make install
```

OK, this is done.

You have installed ABINIT, congratulation!

---

## To go further...

### Improving Linear Algebra routines.

Usually, the detected linear algebra is the *netlib* one (you can check that in the configuration report):

```
* LINALG flavor = netlib
```

The *netlib BLAS/Lapack* distribution is not really efficient. You could take advantage of a better implementation of the Linear Algebra routines.

For example, you could try to find the *Atlas* library.

If it is not present on your computer, install it:

- On Ubuntu : `sudo apt-get install libatlas-base-dev`
- On Fedora/CentOS : `sudo yum install atlas-devel`
- On macOS : `sudo port install atlas` or `brew install atlas`

Then add in the configuration file:

```
with_linalg_flavor="atlas"
```

If it is successfully detected, the final report should contain:

```
* LINALG flavor = atlas
```

You can also try to link manually the linear algebra library.

Example with *Atlas*:

```
with_linalg_libs="-llapack -lf77blas -lcblas -latlas"
```

### Improving FFT implementation

On the same model as Linear Algebra, you can take advantage of the FFTW library (if installed).

If it is not present on your computer, install it:

- On Ubuntu : `sudo apt-get install libfftw3-dev`
- On Fedora/CentOS : `sudo yum install fftw-devel`
- On macOS : `sudo port install fftw` or `brew install fftw`

The better way to try it is to add:

```
with_fft_flavor="fftw3"  
with_fft_libs="-lfftw3 -lfftw3f"
```

---

## A possible configuration file (laptop)

```
enable_mpi="yes".
with_mpi_prefix="/usr"
enable_openmp="yes"

with_trio_flavor="netcdf".
with_dft_flavor="libxc".

with_linalg_flavor="atlas"
with_linalg_libs="-llapack -lf77blas -lcblas -latlas"

with_fft_flavor="fftw3"
with_fft_libs="-lfftw3 -lfftw3f"
```