- Home
- People
- Publications
- Open positions
- Mustguseal platform
- Mustguseal
- Zebra2
- Zebra3D
  - Download
  - Installation
  - Algorithm
  - Input
  - Parameters
  - Statistical model
  - Results
  - Examples
  - Citing
- pocketZebra
- visualCMAT
- Yosshi
- parMATT
- mpiWrapper
- CASBench
- easyAmber
- Biomol2Clust
- vsFilt
- Teaching & tutorials
- Switch to https

Home » Zebra3D » Installation

Zebra3D Installation Instructions

Zebra3D is a command-line Python3-based program intended for desktop computers and notebooks running Linux-based OS or MS Windows. Installation of Zebra3D is straightforward and does not require significant investment of time from the user. We recommend to use 'virtual environment' in Python3 to satisfy the requirements. The software implements shared-memory parallel capabilities to scale efficiently on all cores/threads of a multi-core CPU. Zebra3D can also be executed on one node of a cluster/supercomputer and can automatically utilize all CPUs hosted within a shared address space (e.g. by dual-CPU motherboards). The PyMol molecular graphics engine should also be installed for full compatibility.

- General installation instructions
- Activating a 'virtual environment' in Python3
- Troubleshooting
- Support

General installation instructions

Basics. Zebra3D requires the latest Python3 and a Linux-based OS or MS Windows. The installation of Zebra3D dependencies (i.e., third-party libraries) can be performed automatically using the `pip3` package installer. The PyMol molecular graphics engine should also be installed for full compatibility.

The superuser/root privileges. The Zebra3D files can be installed into a local folder, but fulfilling the Python3-associated prerequisites may require superuser/root privileges to perform the commands listed below.

Installation Step 1. Extract the Zebra3D files to any location on your computer and enter the project folder:

```
tar xzf zebra3d.latest.tar.gz
cd Zebra3D_folder
```

Installation Step 2. You may need to upgrade the `pip` package installer to the most recent version by running the following command:

```
pip3 install --upgrade pip
```

Installation Step 3. Install Zebra3D dependencies automatically using the `requirements.txt` file included into the Zebra3D distribution:

```
pip3 install -r requirements.txt
```

Try running your Zebra3D:

```
python3 ./Zebra3D.py
```

Installation Step 4. Install the PyMol molecular graphics engine. The PyMol molecular graphics engine will be required to compile and operate the Zebra3D annotation files. PyMol can be installed from a Linux repository (e.g., in openSuSE you should try the "`zypper in pymol`" command), compiled from

sources (recommended for advanced users only) or purchased from the distributor **Schrödinger LLC** (external link).

The PyMol binary should be executable and the corresponding folder should be in the global PATH variable. E.g., in Linux OS the PyMol installation can be probed by the following command:

```
sda@sda:~> which pymol
/usr/bin/pymol
```

Try running the "`pymol -c`" command to probe the PyMol installation

Activating a 'virtual environment' in Python3

Different Python3 programs can depend on different versions of the same packages (i.e. which are stated in the `requirements.txt` file). By satisfying the requirements globally (when running the `pip3 install -r requirements.txt` command) you replace (i.e. overwrite) the existing versions of packages with the required ones, potentially breaking the dependencies of all previously installed software. To avoid this mess, you may wish to use virtual environments to install the required packages to designated local folders. Then, you don't have to worry about breaking the dependencies of other Python3 software.

Using virtual environments in Python is simple.

First, install the `virtualenv` package:

```
pip3 install virtualenv
```

Now, enter the project folder (e.g., the permanent folder with Zebra3D installation data) and create the virtual environment (this has to be done only once per project):

```
cd Zebra3D_folder
python3 -m virtualenv env
```

This will create the `env` subfolder to store the 'virtual environment', i.e. the settings and packages.

Activate the environment (this command has to be executed each time you want to use the virtual environment, e.g. to install packages or to use the software):

```
source env/bin/activate
```

You can now install the requirements as you would normally do (as was already explained above), using the command below. The difference is, this time these requirements will be installed into the local 'virtual environment' folder:

```
pip3 install -r requirements.txt
```

You can now run the Zebra3D software:

```
cd Example_03_HAR/
./linux_run_this_example.sh
```

To deactivate the virtual environment, simply run:

```
deactivate
```

Troubleshooting

Problems with compilation of Python3 packages using `pip3`. If you experience errors while building the requirements using `pip3` check the following:

- Make sure you are using Python3 and `pip3` (i.e. not Python2 and `pip2`) by checking version numbers of these executables:
  `> pip3 --version`
  `pip 20.3.3 from /webserver/Zebra3D-1.1/env/lib/python3.6/site-packages/pip` **`(python 3.6)`**
  `> python3 --version`
  `Python 3.6.12`

- Check that the Python3 development files were properly installed from the `python3-devel` package (the exact name can differ among Linux distributions, but should have the "-devel" postfix). In particular, if you see an error reporting that "Python.h" file is missing during compilation, it would mean that the `python3-devel` package was not properly installed. In openSuSE, run YaST with root privileges to install system packages;

- We noticed that some requirements failed to compile with an older version of the `gcc` compiler. We had no problems with **`gcc` version 7.5.0** which is the default in modern openSuSE Linux distributions. If the requirements fail to compile with no clear explanation, we recommend to check the version of your compiler (usually you should run the compiler with the '-v' flag, e.g. `gcc -v`) and consider an upgrade. In openSuSE, run YaST with root privileges to upgrade system packages. Upgrade of system compilers should be carried out with caution and curated by an advanced user or system administrator, as amateur actions may invalidate all previous installations.

`OSError: [Errno 24] Too many open files`. We experienced this error when running Zebra3D on a large number of CPU threads (i.e. 40 threads of a dual-CPU 20-core system). We found two easy workarounds to solve this issue, as explained below:

- **Workaround 1:** run using a smaller number of CPU threads, e.g. 20 instead of 40. By default, Zebra3D will utilize all available CPU threads. To limit the number of threads, use the `cpu_threads` command line flag, e.g. `cpu_threads=20`. This approach will not necessarily be slower, as in this example using 20 threads means that all physically available 20 CPU cores will by occupied;

- **Workaround 2:** upgrade the ulimit. Check the current ulimit and set it to a larger value, e.g.:

```
ulimit -n
1024
> ulimit -n 4096
> ulimit -n
4096
```

Problems with OPTICS. If you see an error like this when requesting OPTICS as the clustering method, it would mean that the OPTICS algorithm was not installed properly (probably, overwritten by running pip3 globally to satisfy requirements of other software, see **virtual environments** to learn how to avoid such problems):

```
AttributeError: module 'sklearn.cluster' has no attribute 'OPTICS'
```

Use the following command to update the scikit-learn package:

```
pip3 install -U scikit-learn
```

Support

If you are having problems with Zebra3D installation don't hesitate to request assistance:

Dmitry Suplatov d.a.suplatov@belozersky.msu.ru Ask for help