

## Exercise 1: Download and install the tutorial

Open a terminal. Then, enter the following commands.

```
1 $ cd ~
2 $ mkdir alpstat
3 $ cd alpstat
4 $ git clone https://github.com/ALPSCore/Tutorial2.git
5 $ cd Tutorial2
6 $ tutorial=$PWD
7 $ ls -l
```

You should see a list of files and no error messages.

## Exercise 2: Download/install prerequisites

### Ubuntu Linux

```
1 $ sudo apt-get install cmake
2 $ sudo apt-get install libhdf5-dev
3 $ sudo apt-get install libboost-all-dev
4 $ sudo apt-get install mpi-default-dev
```

### Mac OS X

```
1 $ port install alpscore
```

This will install the latest ALPSCore release (we don't need it!) and prerequisites.

### Check

```
1 $ cmake --version
2 $ g++ --version
3 $ h5cc --version
4 $ mpicxx --version
```

There should be no error messages.

## Exercise 3: Download and install ALPSCore.

```
1 $ git https://github.com/ALPSCore/ALPSCore
2 $ cd ALPSCore
3 $ mkdir build
4 $ cd build
5 $ export ALPSCore_DIR=$PWD/install
```

```
6 $ cmake -DCMAKE_INSTALL_PREFIX=$ALPSCore_DIR ..
7 $ make
8 $ make test
9 $ make install
```

See also the list of CMake variables on page 7.

## Exercise 4: Build and run a dummy program that uses ALPSCore and does nothing.

The code is at [\\$tutorial/step1\\_trivial](#).

CMake file online: [https://git.io/alpstut2\\_s1\\_cmake](https://git.io/alpstut2_s1_cmake)

Source file online: [https://git.io/alpstut2\\_s1\\_main](https://git.io/alpstut2_s1_main)

The directory online: [https://git.io/alpstut2\\_s1](https://git.io/alpstut2_s1)

```
1 $ cd $tutorial/step1_trivial
2 $ mkdir 000build
3 $ cd 000build
4 $ cmake ..
5 $ make
6 $ ./alpsdemo
```

## Exercise 5: Build and run a program that uses parameters.

The code is at [\\$tutorial/step2\\_params](#).

Online: [https://git.io/alpstut2\\_s2](https://git.io/alpstut2_s2)

1. Play with the different values of parameters.
2. Try to override them from the command line.
3. Change the program to make `--loud` parameter an integer, with value 0 meaning “be quiet”.

```
1 $ cd $tutorial/step2_params
2 $ mkdir 000build
3 $ cd 000build
4 $ cmake ..
5 $ make
6 $ ./alpsdemo
7 $ ./alpsdemo --help
8 $ ./alpsdemo ../params.ini
9 $ ./alpsdemo ../params.ini --count=3
10 ....
```

## Exercise 6: Build and run a trivial MC program

The code is at [\\$tutorial/step3\\_trivial\\_mc](#).

Online: [https://git.io/alpstut2\\_s3](https://git.io/alpstut2_s3)

Note: the simulation code is split into 2 files.

1. Build and run.

```
$ ./alpsdemo --help
```

2. Run with small counts:

```
1 $ ./alpsdemo --count=2
2 $ ./alpsdemo --count=2 --verbose
```

3. Run with large count and small timelimit; time the execution:

```
$ time -p your_command
```

```
1 $ time -p ./alpsdemo --count=10000000 --timelimit=1
```

4. Set large time limit and interrupt the program (via Ctrl-C).
5. Change `fraction_completed()` so that `--count=0` would mean “till timeout”.
6. Change the name of the `update()` method and see it does not compile any more.

## Exercise 7: Compute $\pi$ by Markov chain MC

The code is at [\\$tutorial/step4\\_pi](#).

Online: [https://git.io/alpstut2\\_s4](https://git.io/alpstut2_s4)

1. Build and run the program.

2. Run with various time limits:

```
1 $ ./alpsdemo --timelimit=5
2 $ ./alpsdemo --timelimit=10
```

3. Run with different step sizes, compare autocorrelation lengths.

```
1 $ ./alpsdemo --step=1
2 $ ./alpsdemo --step=10
3 $ ./alpsdemo --step=0.1
```

4. Replace `FullBinningAccumulator` to `NoBinningAccumulator`.
5. Run with low or high step length and see the underestimated error bars.

```
1 $ ./alpsdemo --step=1
2 $ ./alpsdemo --step=0.01
3 $ ./alpsdemo --step=10
```

## Exercise 8: Running and resuming

The code is in [\\$tutorial/step5\\_pi\\_checkpoint](#).

Online: [https://git.io/alpstut2\\_s5](https://git.io/alpstut2_s5)

1. Build the code.
2. Run the code. There is an error: find and fix it!
3. Build and run the corrected code (note more options available!).

```
1 $ ./alpsdemo --help
2 $ ./alpsdemo --step 1 --timelimit 5
```

4. Note new files appear:
  - “\*.out” file contains simulation results.
  - “\*.clone.h5” file contains checkpoint.
5. Restore the checkpoint and run for 10 more seconds:

```
1 $ ./alpsdemo alpsdemo.clone.h5 --timelimit 10
```

Note:

- compulsory `--step` is read from the checkpoint.
- parameters can be overridden (like `--timelimit`).

## Exercise 9: Parallel runs

The code is in [\\$tutorial/step6\\_pi\\_mpi](#).

Online: [https://git.io/alpstut2\\_s6](https://git.io/alpstut2_s6)

1. Build the MPI-parallelized program.
2. Do timed runs with different number of processes:

```
1 $ time -p mpiexec -n 1 ./alpsdemo --step=1 --timelimit=10
2 $ time -p mpiexec -n 2 ./alpsdemo --step=1 --timelimit=10
```

3. Observe checkpoint names.
4. Try to restore from checkpoints, see how statistics builds up.

```
1 $ mpiexec -n 2 ./alpsdemo alpsdemo.clone.h5
```

## Exercise 10: Parallelize the 2D Ising code

The code is in [\\$tutorial/step7\\_ising](#).

Online: [https://git.io/alpstut2\\_s7](https://git.io/alpstut2_s7)

Steps:

1. Initialize MPI environment.
2. Use `alps::mcmpiadapter` template.
3. Use the parallel parameter constructor.
4. Make sure each rank has its own checkpoint file.
5. Make sure only the master process outputs the results.

## Appendix: Git cheat-sheet for today

Clone (“download”) repository:

```
git clone repo local_directory
```

or

```
git clone repo
```

Example:

```
git clone https://github.com/ALPSCore/Tutorial2.git
```

See the current changes:

```
git diff
```

Revert changes (CAUTION — no way back!):

```
git reset --hard
```

## Appendix: Important links

ALPSCore site: <http://alpscore.org>

ALPSCore wiki: <https://github.com/ALPSCore/ALPSCore/wiki>

ALPSCore repo: <https://github.com/ALPSCore/ALPSCore>

Doxygen docs: [https://alpscore.ci.cloudbees.com  
/job/alpscore\\_fedora17\\_doc-only/ALPSCore\\_reference/](https://alpscore.ci.cloudbees.com/job/alpscore_fedora17_doc-only/ALPSCore_reference/)

Tutorial repo: <https://github.com/ALPSCore/Tutorial2>

Exercise solutions:

- Online: [https://git.io/alpstut2\\_solutions](https://git.io/alpstut2_solutions)
- Locally: [\\$tutorial/X.solutions](#)

# Appendix: ALPSCore CMake and environment variables

CMake variables: set as `cmake -Dvariable=value`.

| Variable              | Default value    | Comment   |
|-----------------------|------------------|---|
| CMAKE_CXX_COMPILER    | (system default) | Path to C++ compiler executable. Can be set only once.                                    |
| CMAKE_INSTALL_PREFIX  | /usr/local       | ALPSCore target install directory.  |
| CMAKE_BUILD_TYPE      |                  | Specifies build type; set to <b>Release</b> to maximize performance.                      |
| BOOST_ROOT            |                  | Boost install directory. Set if CMake fails to find Boost.                                |
| Boost_NO_SYSTEM_PATHS | false            | Disable search in default system directories. Set if the wrong version of Boost is found. |
| Boost_NO_BOOST_CMAKE  | false            | Disable search for Boost CMake file. Set if the wrong version of Boost is found.          |
| Documentation         | ON               | Build ALPSCore developer's documentation.   |
| ENABLE_MPI            | ON               | Enable MPI build.   |
| Testing               | ON               | Build unit tests (recommended).   |
| ALPS_BUILD_SHARED     | ON               | Build ALPSCore as shared libraries. Mutually exclusive with ALPS_BUILD_STATIC=ON.         |
| ALPS_BUILD_STATIC     | OFF              | Build ALPSCore as static libraries. Mutually exclusive with ALPS_BUILD_SHARED=ON.         |

The environment variables are set via:

```
$ export variable=value
```

before running CMake. The relevant variables are:

| Variable   | Comment  |
|------------|--|
| CXX        | Path to C++ compiler executable. Can be set only once.     |
| BOOST_ROOT | Boost install directory. Set if CMake fails to find Boost. |
| HDF5_ROOT  | HDF5 install directory. Set if CMake fails to find HDF5.   |