BOUT++ cheat sheet

Essential and useful commands, from getting and building BOUT++, to writing a physics model. More complete documentation at https://bout-dev.readthedocs.io/

git commands

```
# Initial checkout
git clone --recurse-submodules https://github.com/boutproject/BOUT-dev.git
# Checkout other branch
git checkout <branch or tag>
# Get branch updates
git pull
# Make sure submodules are up to date
git submodule update --init --recursive
```

Building BOUT++

Requirements

- C++11 compiler (gcc >= 4.9, Intel >= 14, Cray >= 8.4, Clang >= 3.3)
- MPI
- NetCDF >= 4.4.0

Optional dependencies:

- HDF5
- FFTW3
- OpenMP
- PETSc >= 3.4.0
- SLEPc >= 3.4.0
- SUNDIALS >= 2.7, or the separate components:
  - ARKODE
  - IDA
  - CVODE
- LAPACK

Most of the integrated tests required `PYTHONPATH` to be set to `tools/pylib`:

```
export PYTHONPATH=/path/to/BOUT++/tools/pylib:$PYTHONPATH
```

Configure

The basics:

```
# Detect compiler, libraries, etc.
./configure
# Build in parallel with <N> processors:
make -j<N>
```

```
# Run tests
make check
```

Production: full optimisation, OpenMP, PETSc, and SUNDIALS

```
./configure --with-optimize=3 \
    --enable-openmp \
    --with-petsc=${PETSC_DIR} \
    --with-sundials=${SUNDIALS_DIR}
```

Other useful commands:

```
# List available options
./configure --help
# Full debug mode
./configure --enable-checks=3 \
    --enable-track \
    --enable-debug
# Set MPI C++ compiler to (for example) Intel MPI
./configure MPICXX=mpiicpc
```

Developing BOUT++ library itself? If you touch a header file, need to rebuild the whole library:

```
make clean && make -j<N>
```

Need to change configuration options? Just rerun `configure`. Worst case:

```
make distclean && ./configure <options>
```

Need to modify `configure` somehow? Change `configure.ac` then run

```
autoreconf -fvi
```

CMake

The basics:

```
# Make out-of-source build directory
mkdir build && cd build
# Detect compiler, libraries, etc.
cmake ..
# Build in parallel with <N> processors:
make -j<N>
# Run tests
make check
```

Production: full optimisation, OpenMP, PETSc, and SUNDIALS

```
cmake .. -DCMAKE_BUILD_TYPE=Release \
    -DENABLE_OPENMP=ON \
    -DUSE_PETSC=ON -DPETSC_DIR=${PETSC_DIR} \
    -DUSE_SUNDIALS=ON -DSUNDIALS_ROOT=${SUNDIALS_DIR} \
    -DCMAKE_INSTALL_PREFIX=/BOUT++/install/path
```

Other useful CMake flags:

```
# Get latest version of CMake
pip3 install --user cmake
export PATH=~/.local/bin:$PATH
# Graphical interface
cmake-gui
# List all options and documentation
cmake .. -LH
# Export compilation commands for other tools
```

```
cmake .. -DCMAKE_EXPORT_COMPILE_COMMANDS=ON
# Set the MPI C++ compiler (for example) for Intel MPI:
cmake .. -DCMAKE_CXX_COMPILER=mpiicpc
```

Developing BOUT++ library itself? Just rerun `make`, CMake knows exactly what to rebuild, including if you touch headers

Need to change configuration options? Delete the build directory and make a fresh one:

```
rm -r build && mkdir build && cd build
cmake .. <options>
```

Can keep multiple build directories for different configurations

Building a physics model

Makefile

In `Makefile`:

```
BOUT_TOP = /path/to/bout++/
SOURCEC = my_model.cxx
include $(BOUT_TOP)/make.config
```

then run:

```
make
```

CMake

In `CMakeLists.txt`:

```
cmake_minimum_required(VERSION 3.13)
project(my_model LANGUAGES CXX)
find_package(bout++ REQUIRED)
add_executable(my_model my_model.cxx)
target_link_libraries(my_model PRIVATE bout++::bout++)
```

then run:

```
mkdir build && cd build
cmake .. -DCMAKE_PREFIX_PATH=/path/to/installed/bout++
```

Writing a physics model

1. Include the necessary headers

   ```
   // Required
   #include <bout/physicsmodel.hxx>
   // For Laplacian inversion
   #include <invert_laplace.hxx>
   ```

2. Write a `class` publicly inheriting from `PhysicsModel`:

   ```
   class MyModel : public PhysicsModel {
   };
   ```

3. Add members for the evolving fields, any runtime options, any objects that need to live between `rhs` calls:

   ```
   class MyModel : public PhysicsModel {
       Field3D temperature;
       BoutReal conductivity;
   };
   ```

4. Implement the two required virtual functions, `init` and `rhs`. `init` should contain a call to `SOLVE_FOR` for each evolving variable. Similarly, `rhs` should contain a call to the `ddt` for those variables.

```
class MyModel : public PhysicsModel {
    Field3D temperature;
    BoutReal conductivity;
public:
    int init(bool restarting) override {
        // Any other setup
        SOLVE_FOR(temperature);
    }
    int rhs(BoutReal time) override {
        ddt(temperature) = /* implementation */;
    }
};
```

5. Add `BOUTMAIN` to create default `main` function:

```
class MyModel : public PhysicsModel {
...
};

BOUTMAIN(MyModel)
```

6. Add a `Makefile` or `CMakeLists.txt` from above