# HEMCO

*Release 3.0.0-rc.0*

**GEOS-Chem Support Team**

**Jan 13, 2021**

# GETTING STARTED

**Important:** This is a prerelease of the HEMCO user guide. These pages are the most up-to-date and accurate instructions for HEMCO, but they are still a work in progress.

Contributions (e.g., suggestions, edits, revisions) would be greatly appreciated. See editing this guide and our contributing guidelines. If you find a something hard to understand—let us know!

CHAPTER

ONE

# QUICK START

This quickstart guide assumes your environment satisfies HEMCO's requirements. This means you should load a compute environment such that programs like **`cmake`** and **`mpirun`** are available, before continuing. You can find more detailed instructions in the user guide.

## 1.1 1. Clone HEMCO

Download the source code:

```
$ git clone https://github.com/geoschem/HEMCO.git ~/HEMCO
$ cd ~/HEMCO
```

Checkout the HEMCO version that you want to use:

```
$ git checkout 3.0.0-rc.0
```

## 1.2 2. Create a run directory

Navigate to the `run/` subdirectory. Create a run directory by running `./createRunDir.sh` and answering the prompts:

```
$ cd run/
$ ./createRunDir.sh
```

## 1.3 3. Configure your build

Create a build directory and **`cd`** into it. A good name for this directory is `build/`, and a good place for it is in the top-level of the source code:

```
$ mkdir ~/HEMCO/build
$ cd ~/HEMCO/build
```

Initialize your build directory by running **`cmake`** and passing it the path to your source code:

```
$ cmake ~/HEMCO
```

Now you can configure *build options*. These are persistent settings that are saved to your build directory. A common build option is `-DRUNDIR`. This option lets you specify one or more run directories that HEMCO is "installed" to when you do **`make install`**. Configure your build so it installs HEMCO to the run directory you created in Step 2:

```
$ cmake . -DRUNDIR="/path/to/rundir"
```

---

**Note:** The **`.`** in the **`cmake`** command above is important. It tells CMake that your current working directory (i.e., `.`) is your build directory.

---

## 1.4 4. Compile and install

Compile HEMCO:

```
$ make -j
```

Next, install the compiled executable to your run directory (or directories):

```
$ make install
```

This copies `build/bin/hemco_standalone` and supplemental files to your run directory.

---

**Note:** You can update build settings at any time:

1. Navigate to your build directory.
2. Update your build settings with **`cmake`**. See
3. Recompile with **`make -j`**. Note that the build system automatically figures out what (if any) files need to be recompiled.
4. Install the rebuilt executable with **`make install`**.

---

## 1.5 5. Configure your run directory

Now, navigate to your run directory:

```
$ cd path/to/rundir
```

Simulation settings are configured in the `.rc` files. The main configuration file is `HEMCO_sa_Config.rc`. The start end end time for your simulation can be modified in `HEMCO_sa_Time.rc`. The horizontal grid for your simulation can be modified in `HEMCO_sa_Grid.rc`. Emissions settings can be changed in the *HEMCO_Config.rc* file that has been copied from another model (e.g. GEOS-Chem).

## 1.6 6. Run HEMCO

HEMCO can be run interactively from within your run directory by typing:

```
$ ./hemco_standalone
```

You may also submit your HEMCO simulation as a batch job to a scheduler. A sample run script `runHEMCO.sh` is included in your run directory. To submit a HEMCO simulation using SLURM:

```
$ sbatch runHEMCO.sh
```

Those are the basics of using HEMCO! See the user guide, step-by-step guides, and reference pages for more detailed instructions.

CHAPTER

TWO

# REQUIREMENTS

## 2.1 Hardware requirements

### 2.1.1 Computer system requirements

Before you can run HEMCO, you will need to have one the following items.

| Item | Description |
|---|---|
| EITHER<br>A Unix-based computer system | You will need a Unix operating system environment in order to run HEMCO. Any flavor of Unix (e.g. CentOS, Ubuntu, Fedora, etc.) should work just fine. |
| OR<br>An account on the Amazon Web Services cloud | If your institution has computational resources (e.g. a shared computer cluster with many cores, sufficient disk storage and memory), then you can run GEOS-Chem there. Contact your IT staff for assistance.<br>If your institution lacks computational resources (or if you need additional computational resources beyond what is available), then you should consider signing up for access to the Amazon Web Services cloud. Using the cloud has the following advantages:<br>• You can run GEOS-Chem without having to invest in local hardware and maintenance personnel.<br>• You won't have to download any meteorological fields or emissions data. All of the necessary data input for GEOS-Chem will be available on the cloud.<br>• You can initialize your computational environment with all of the required software (e.g. compilers, libraries, utilities) that you need for GEOS-Chem.<br>• Your GEOS-Chem runs will be 100% reproducible, because you will initialize your computational environment the same way every time.<br>• You will avoid GEOS-Chem compilation errors due to library incompatibilities.<br>• You will be charged for the computational time that you use, and if you download data off the cloud.<br>GEOS-Chem 12.0.0 and later versions can be used on the Amazon Web Services cloud computing platform. You can learn more about how to use GEOS-Chem on the cloud by visiting this tutorial (cloud.geos-chem.org). |

### 2.1.2 Memory requirements

If you plan to run GEOS-Chem on a local computer system, please make sure that your system has sufficient memory and disk space:

## 2.2 Software requirements

### 2.2.1 Supported compilers for HEMCO

The table below lists the supported compilers for HEMCO.

HEMCO is written in the Fortran programming language. However, you will also need C and C++ compilers to install certain libraries (like netCDF) on your system.

| Item | Description | Versions | Best way to install |
|---|---|---|---|
| `Intel Compiler Suite (icc, icpc, ifort) | **The Intel Compiler Suite is our recommended proprietary compiler collection.** Intel compilers produce well-optimized code that runs extremely efficiency on machines with Intel CPUs. Many universities and institutions will have an Intel site license that allows you to use these compilers. | The GCST has tested with these versions (but others may work as well):<br>• 19.0.5.281<br>• 18.0.5<br>• 17.0.4<br>• 15.0.0<br>• 13.0.079<br>• 11.1.069 | Install from Intel (requires purchase of a site license or a student license) |
| GNU Compiler Collection (gcc, g++, gfortra n) | **The GNU Compiler Collection is our recommended open-source compiler collection.** Because the GNU Compiler Collection is free and open source, this is a good choice if your institution lacks an Intel site license, or if you are running GEOS-Chem on the Amazon EC2 cloud environment. | The GCST has tested with these versions (but others may work as well):<br>• 10.2.0<br>• 9.3.0<br>• 9.2.0<br>• 8.2.0<br>• 7.4.0<br>• 7.3.0<br>• 7.1.0<br>• 6.2.0 | Install via Spack |

### 2.2.2 Required software packages for HEMCO

| Item | Description | Best way to install |
|---|---|---|
| *Git (a source code management system)* | GEOS-Chem source code can be downloaded using the Git source code management system. GEOS-Chem software repositories are stored at the https://github.com/geoschem organization page. Please see our Guide to using Git with GEOS-Chem for more information about how to use Git with GEOS-Chem. | Direct install from Git-SCM.com |
| CMake | CMake is software that directs how the GEOS-Chem source code is compiled into an executable.<br>• CMake is optional for GEOS-Chem versions 12.6.0 through 12.9.3.<br>• CMake is **REQUIRED** for GEOS-Chem versions 13.0.0 and later. | Install with Spack |
| GNU Make | GNU Make is software that can build executables from source code.<br>• NOTE: While GNU Make is not required for GEOS-Chem 13.0.0 and later, some external libraries that you might need to build will require GNU Make. Therefore it is best to download GNU Make along with CMake. | Install with Spack |
| *netCDF and netCDF-Fortran*<br>• plus dependencies (e.g. HDF5, zlib, etc) | GEOS-Chem input and output data files use the netCDF file format. This is a self-describing file format that allows metadata (descriptive text) to be stored alongside data values. Please see our Guide to netCDF in GEO S-Chem for more information about the netCDF file format and software library. | Install with Spack |

### 2.2.3 Optional but recommended software packages

| Item | Description | Best way to install |
|---|---|---|
| GCPy | GCPy is our recommended python companion software to GEOS-Chem. While this is not a general-purpose plotting package, it does contain many useful functions for creating zonal mean and horizontal plots from GEOS-Chem output. It also contains scripts to generate plots and tables from GEOS-Chem benchmark simulations. | Install from conda-forge |
| gdb and cgdb | The GNU debugger (gdb) and its graphical interface (cgdb) are very useful tools for tracking down the source of GEOS-Chem errors, such as segmentation faults, out-of-bounds errors, etc. | Install with Spack |
| ncview | ncview is a netCDF file viewer. While it does not produce publication-quality output, ncview can let you easily examine the contents of a netCDF data file (such as those which are input and output by GEOS-Chem). Ncview is very useful for debugging and development. | Install with Spack |
| nco | NCO are the netCDF operators. These are very powerful command-line tools for editing and manipulating data in netCDF format. | Install with Spack |
| cdo | CDO are the Climate Data Operators. These are very powerful command-line tools for editing and manipulating data in netCDF format. | Install with Spack |
| The Kinetic PreProcessor (KPP) chemical solver | KPP translates a chemical mechanism specification from user-configurable input files to Fortran-90 source code. | Install with Git |
| flex | Flex is the Fast Lexical Analyzer. This is a required library for the KPP chemical solver. | Install with Spack |

### 2.2.4 Required source code and data for HEMCO

| Item | Description | Best way to install |
|---|---|---|
| A clone of the geoschem/HEMCO repository repository | The HEMCO (Harmonized Emissions Component codebase. | Git clone from geoschem/HEMCO |
| The GEOS-Chem shared data directories | This is the directory structure containing the meteorology and emissions data that GEOS-Chem reads as input. For more information, please see our Downloading GEOS-Chem data directories wiki page. | Perform a HEMCO dry run |

CHAPTER

THREE

# KEY REFERENCES

- GEOS-Chem was first described in Bey_et_al._2001.
- HEMCO is described in Keller_et_al._2014.

CHAPTER

FOUR

# DOWNLOADING HEMCO

When cloning HEMCO you will get the `main` branch by default.

```
$ git clone https://github.com/geoschem/hemco.git HEMCO
```

If you would like a different version of HEMCO you can checkout the branch or tag from the top-level directory. If you have any unsaved changes, make sure you commit those to a branch prior to updating versions.

```
$ cd HEMCO
$ git checkout tags/3.0.0
```

CHAPTER

FIVE

# COMPILING HEMCO

**Note:** This user guide assumes you have loaded a computing environment that satisfies HEMCO's software requirements.

**Note:** Another useful resource for HEMCO build instructions is our YouTube tutorial.

There are two steps to build HEMCO. The first step is configuring your build. To configure your build you use **`cmake`** to configure build settings. Build settings cover options like enabling or disabling components, specifying run directories to install HEMCO to, or whether HEMCO should be compiled in Debug mode.

The second step is compiling. To compile HEMCO you use **`make`**. This compiles HEMCO according to your build configuration.

## 5.1 Create your build directory

Create a build directory. This directory is going to be the working directory for your build. The configuration and compile steps generate a bunch of build files, and this directory is going to store those. You can think of a build directory as representing a HEMCO build. It stores configuration settings, information about your system, and intermediate files from the compiler.

A build directory is self contained, so you can delete it at any point to erase the build and its configuration. You can have as many build directories as you would like. Most users only need one build directory, since they only build HEMCO once; but, for example, if you were building HEMCO with Intel and GNU compilers to compare performance, you would have two build directories: one for the Intel build, and one for the GNU build. You can name your build directories whatever you want, but a good choice is `build/`. There is one rule for build directories: **a build directory should be a new directory**.

Create a build directory and initialize it. You initialize a build directory by running **`cmake`** with the path to the HEMCO source code. Here is an example of creating a build directory in the top-level of the HEMCO source code:

```
$ cd HEMCO
$ mkdir build
$ cd build
$ cmake ..
-- The Fortran compiler identification is Intel 18.0.5.20180823
-- Check for working Fortran compiler: /bin/intel64/ifort
-- Check for working Fortran compiler: /bin/intel64/ifort  -- works
...
-- Configuring done
```

(continues on next page)

(continued from previous page)

```
-- Generating done
-- Build files have been written to: HEMCO/build
```

## 5.2 Configuring your build

Build settings are controlled by **cmake** commands with the following form:

```
$ cmake . -D<NAME>="<VALUE>"
```

where <NAME> is the name of the setting, and <VALUE> is the value that you are assigning it. These settings are persistent and saved in your build directory. You can set multiple variables in a single command, and you can run **cmake** as many times as you need to configure your desired settings.

**Note:** The . argument is important. It is the path to your build directory which is . here.

HEMCO has no required build settings. You can find the complete list of *HEMCO's build settings here*. The most frequently used build setting is RUNDIR which lets you specify one or more run directories where CMake will install HEMCO. Here, "install" refers to copying the compiled executable, and some supplemental files with build settings, to your run directories.

**Note:** You can even update build settings after you compile HEMCO. Simply rerun **make** and (optionally) **make install**, and the build system will automatically figure out what needs to be recompiled.

Since there are no required build settings, for this tutorial we will stick with the default settings.

You should notice that when you run **cmake** it ends with:

```
...
-- Configuring done
-- Generating done
-- Build files have been written to: HEMCO/build
```

This tells you the configuration was successful, and that you are ready to compile.

## 5.3 Compile HEMCO

You compile HEMCO with:

```
$ make -j     # -j enables compiling in parallel
```

Optionally, you can use the VERBOSE=1 argument to see the compiler commands.

This step creates ./bin/hemco_standalone which is the compiled executable. You can copy this executable to your run directory manually, or you can do

```
$ make install
```

which copies ./bin/hemco_standalone (and some supplemental files) to the run directories specified in RUNDIR.

Now you have compiled HEMCO, and you are ready to move on to creating a run directory!

## 5.4 Recompiling

You need to recompile HEMCO if you update a build setting or make a modification to the source code. However, with CMake, you don't need to clean before recompiling. The build system automatically figure out which files need to be recompiled based on your modification. This is known as incremental compiling.

To recompile HEMCO, simply do

```
$ make -j   # -j enables compiling in parallel
```

and optionally, do **`make install`**.

**RUNDIR** Paths to run directories where **`make install`** installs HEMCO. Multiple run directories can be specified by a semicolon separated list. A warning is issues if one of these directories does not look like a run directory.

These paths can be relative paths or absolute paths. Relative paths are interpreted as relative to your build directory.

**CMAKE_BUILD_TYPE** The build type. Valid values are `Release` or `Debug`. Set this to `Debug` if you want to build in debug mode.

**HEMCO_Fortran_FLAGS_<COMPILER_ID>** Additional compiler options for HEMCO for build type `<BUILD_TYPE>`.

**HEMCO_Fortran_FLAGS_<BUILD_TYPE>_<COMPILER_ID>** Compiler options for HEMCO for all build types. Valid values for `<COMPILER_ID>` are `GNU` and `Intel`.

CHAPTER

SIX

# CREATING A RUN DIRECTORY

**Note:** Another useful resource for HEMCO run directory creation instructions is our YouTube tutorial.

HEMCO run directories are created from within the source code. A new run directory should be created for each different version of HEMCO you use. Git version information is logged to file `rundir.version` within the run directory upon creation.

To create a run directory, navigate to the `run/` subdirectory of the source code and execute shell script `createRunDir.sh`.

```
$ cd HEMCO/run
$ ./createRunDir.sh
```

During the course of script execution you will be asked a series of questions:

## 6.1 Enter ExtData path

The first time you create a HEMCO run directory on your system you will be prompted for a path to GEOS-Chem shared data directories, which are also used by HEMCO. The path should include the name of your `ExtData/` directory and should not contain symbolic links. The path you enter will be stored in file `~/.geoschem/config` in your home directory as environment variable `GC_DATA_ROOT`. If that file does not already exist it will be created for you. When creating additional run directories you will only be prompted again if the file is missing or if the path within it is not valid.

```
------------------------------------------------------------
Enter path for ExtData:
------------------------------------------------------------
```

## 6.2 Choose meteorology source

Enter the integer number that is next to the input meteorology source you would like to use.

```
------------------------------------------------------------
Choose meteorology source:
------------------------------------------------------------
  1. MERRA-2 (Recommended)
  2. GEOS-FP
```

## 6.3 Choose horizontal resolution

Enter the integer number that is next to the horizontal resolution you would like to use.

```
------------------------------------------------------------
Choose horizontal resolution:
------------------------------------------------------------
  1. 4.0 x 5.0
  2. 2.0 x 2.5
  3. 0.5 x 0.625
  4. 0.25 x 0.3125
  5. Custom
```

## 6.4 Enter HEMCO_Config.rc path

Provide the path to a HEMCO_Config.rc file with your emissions settings. This is typically obtained from another model (e.g. `~/GEOS-Chem/run/HEMCO_Config.rc.templates/HEMCO_Config.rc.fullchem`)

```
------------------------------------------------------------
Enter path to the HEMCO_Config.rc file with your emissions settings.

NOTE: This may be a HEMCO_Config.rc file from a GEOS-Chem run directory
or a HEMCO_Config.template file from the GEOS-Chem source code repository.
------------------------------------------------------------
```

## 6.5 Enter run directory path

Enter the target path where the run directory will be stored. You will be prompted to enter a new path if the one you enter does not exist.

```
------------------------------------------------------------
Enter path where the run directory will be created:
------------------------------------------------------------
```

## 6.6 Enter run directory name

Enter the run directory name, or accept the default. You will be prompted for a new name if a run directory of the same name already exists at the target path.

```
------------------------------------------------------------
Enter run directory name, or press return to use default:

NOTE: This will be a subfolder of the path you entered above.
------------------------------------------------------------
```

## 6.7 Enable version control (optional)

Enter whether you would like your run directory tracked with git version control. With version control you can keep track of exactly what you changed relative to the original settings. This is useful for trouble-shooting as well as tracking run directory feature changes you wish to migrate back to the standard model.

```
------------------------------------------------------------
Do you want to track run directory changes with git? (y/n)
------------------------------------------------------------
```

If a run directory has successfully been created, you should see something like:

```
Created /scratch/rundirs/hemco_4x5_merra2
```

CHAPTER

SEVEN

# CONFIGURING A RUN DIRECTORY

**Note:** Another useful resource for instructions on configuring HEMCO run directories is our YouTube tutorial.

Navigate to your new run directory, and examine the contents:

```
$ cd /scratch/rundirs/hemco_4x5_merra2
$ ls
build/            HEMCO_Diagn.rc        HEMCO_sa_Spec.rc  README
CodeDir@          HEMCO_sa_Config.rc    HEMCO_sa_Time.rc  rundir.version
HEMCO_Config.rc   HEMCO_sa_Grid.4x5.rc  OutputDir/        runHEMCO.sh*
```

The following files can be modified to set up your HEMCO standalone simulation.

**HEMCO_sa_Config.rc** Main configuration file for the HEMCO standalone simulation. This file points to the other configuration files used to set up your simulation (e.g. `HEMCO_sa_Grid.4x5`, `HEMCO_sa_Time.rc`). This file typically references a HEMCO_Config.rc file using `>>>include HEMCO_Config.rc` which contains the emissions settings. Settings in `HEMCO_sa_Config.rc` will always override any settings in the included `HEMCO_Config.rc`.

**HEMCO_Config.rc** Contains emissions settings. This file is typically obtained from another model (e.g. GEOS-Chem).

**HEMCO_Diagn.rc** Specifies which fields to save out to the HEMCO diagnostics file saved in `OutputDir` by default. The frequency to save out diagnostics is controlled by the `DiagnFreq` setting in `HEMCO_Config_sa.rc`

**HEMCO_sa_Grid.4x5.rc** Defines the grid specification. Sample files are provided for 4.0 x 5.0, 2.0 x 2.5, 0.5 x 0.625, and 0.25 x 0.3125 global grids in `HEMCO/run/` and are automatically copied to the run directory based on options chosen when running `createRunDir.sh`. If you choose to run with a custom grid or over a regional domain, you will need to modify this file manually.

**HEMCO_sa_Spec.rc** Defines the species to include in the HEMCO standalone simulation. By default, the species in a GEOS-Chem full-chemistry simulation are defined. To include other species, you can modify this file by providing the species name, molecular weight, and other properties.

**HEMCO_sa_Time.rc** Defines the start and end times of the HEMCO standalone simulation as well as the emissions timestep (s).

**runHEMCO.sh** Sample run script for submitting a HEMCO standalone simulation via SLURM.

CHAPTER

# EIGHT

# RUNNING HEMCO

**Note:** Another useful resource for instructions on running HEMCO is our YouTube tutorial.

## 8.1 Run interactively

HEMCO may be run interactively at the command line by typing the following within your run directory

```
$ ./hemco_standalone
```

You may also specify the path to the HEMCO standalone configuration file using:

```
$ ./hemco_stantalone -c HEMCO_sa_Config.rc
```

If not specified, `HEMCO_sa_Config.rc` will be used by default.

## 8.2 Run as batch job

Batch job run scripts will vary based on what job scheduler you have available. The example run script included in HEMCO run directories (`runHEMCO.sh`) is for use with SLURM. You may modify this file for your system and preferences as needed.

At the top of all batch job scripts are configurable run settings. Most critically are requested # cores, # nodes, time, and memory. Figuring out the optimal values for your run can take some trial and error.

To submit a batch job using SLURM:

```
$ sbatch runHEMCO.sh
```

Standard output will be sent to a log file `HEMCO_SA.log` once the job is started. Standard error will be sent to a file specific to your scheduler, e.g. `slurm-jobid.out` if using SLURM, unless you configure your run script to do otherwise.

If your computational cluster uses a different job scheduler, e.g. Grid Engine, LSF, or PBS, check with your IT staff or search the internet for how to configure and submit batch jobs. For each job scheduler, batch job configurable settings and acceptable formats are available on the internet and are often accessible from the command line. For example, type **`man sbatch`** to scroll through options for SLURM, including various ways of specifying number of cores, time and memory requested.

## 8.3 Verify a successful run

There are several ways to verify that your run was successful.

1. NetCDF files are present in the `OutputDir/` subdirectory
2. HEMCO log file `HEMCO.log` ends with `HEMCO X.Y.Z FINISHED.`
3. Standard output file `HEMCO_SA.log` ends with `HEMCO_STANDALONE FINISHED!`
4. The job scheduler log does not contain any error messages

If it looks like something went wrong, scan through the log files to determine where there may have been an error. Here are a few debugging tips:

- Review all of your configuration files to ensure you have proper setup
- Check to make sure you have downloaded all input files needed for your HEMCO standalone simulation

If you cannot figure out where the problem is please do not hesitate to create a GitHub issue.

CHAPTER

NINE

# SUPPORT GUIDELINES

HEMCO support is maintained by the GEOS-Chem Support Team (GCST). The GCST members are based at Harvard University and Washington University in St. Louis.

We track bugs, user questions, and feature requests through GitHub issues. Please help out as you can in response to issues and user questions.

## 9.1 How to report a bug

We use GitHub to track issues. To report a bug, open a new issue. Please include all the information that might be relevant, including instructions for reproducing the bug.

## 9.2 Where can I ask for help?

We use GitHub issues to support user questions. To ask a question, open a new issue and select the question template.

## 9.3 How to submit changes

Please see "Contributing Guidelines".

## 9.4 How to request an enhancement

Please see "Contributing Guidelines".

CHAPTER

TEN

# CONTRIBUTING GUIDELINES

Thank you for looking into contributing to HEMCO! HEMCO is a grass-roots model that relies on contributions from community members like you. Whether you're new to HEMCO or a longtime user, you're a valued member of the community, and we want you to feel empowered to contribute.

## 10.1 We use GitHub and ReadTheDocs

We use GitHub to host the HEMCO source code, to track issues, user questions, and feature requests, and to accept pull requests: https://github.com/geoschem/HEMCO. Please help out as you can in response to issues and user questions.

We use ReadTheDocs to host the HEMCO user documentation: https://hemco.readthedocs.io.

## 10.2 How to submit changes

We use GitHub Flow, so all changes happen through pull requests. This workflow is described here: GitHub Flow.

As the author you are responsible for:

- Testing your changes
- Updating the user documentation (if applicable)
- Supporting issues and questions related to your changes in the near-term

## 10.3 Coding conventions

The HEMCO codebase dates back several decades and includes contributions from many people and multiple organizations. Therefore, some inconsistent conventions are inevitable, but we ask that you do your best to be consistent with nearby code.

## 10.4 How to request an enhancement

We accept feature requests through issues on GitHub. To request a new feature, open a new issue and select the feature request template. Please include all the information that migth be relevant, including the motivation for the feature.

## 10.5 How to report a bug

Please see "Support Guidelines".

## 10.6 Where can I ask for help?

Please see "Support Guidelines".

CHAPTER

ELEVEN

# EDITING THIS USER GUIDE

This user guide is generated with Sphinx.

## 11.1 Quick start

To build this user guide on your local machine, you need to install Sphinx. Sphinx is a Python 3 package and it is available via **pip**. This user guide uses the Read The Docs theme, so you will also need to install `sphinx-rtd-theme`.

```
$ pip install sphinx sphinx-rtd-theme
```

To build this user guide locally, navigate to the `docs/` directory and make the `html` target.

```
gcuser:~$ cd gcpy/docs
gcuser:~/gcpy/docs$ make html
```

This will build the user guide in `docs/build/html`, and you can open `index.html` in your web-browser. The source files for the user guide are found in `docs/source`.

**Note:** You can clean the documentation with `make clean`.

## 11.2 Learning reST

Writing reST can be tricky at first. Whitespace matters, and some directives can be easily miswritten. Two important things you should know right away are:

- Indents are 3-spaces
- "Things" are separated by 1 blank line. For example, a list or code-block following a paragraph should be separated from the paragraph by 1 blank line.

You should keep these in mind when you're first getting started. Dedicating an hour to learning reST will save you time in the long-run. Below are some good resources for learning reST.

- reStructuredText primer: (single best resource; however, it's better read than skimmed)
- Official reStructuredText reference (there is *a lot* of information here)
- Presentation by Eric Holscher (co-founder of Read The Docs) at DjangoCon US 2015 (the entire presentation is good, but reST is described from 9:03 to 21:04)
- YouTube tutorial by Audrey Tavares's

A good starting point would be Eric Holscher's presentations followed by the reStructuredText primer.

## 11.3 Style guidelines

**Important:** This user guide is written in semantic markup. This is important so that the user guide remains maintainable. Before contributing to this documentation, please review our style guidelines (below). When editing the source, please refrain from using elements with the wrong semantic meaning for aesthetic reasons. Aesthetic issues can be addressed by changes to the theme.

For **titles and headers**:

- Section headers should be underlined by `#` characters
- Subsection headers should be underlined by `-` characters
- Subsubsection headers should be underlined by `^` characters
- Subsubsubsection headers should be avoided, but if necessary, they should be underlined by `"` characters

**File paths** (including directories) occuring in the text should use the `:file:` role.

**Program names** (e.g. **`cmake`**) occuring in the text should use the `:program:` role.

**OS-level commands** (e.g. **`rm`**) occuring in the text should use the `:command:` role.

**Environment variables** occuring in the text should use the `:envvar:` role.

**Inline code** or code variables occuring in the text should use the `:code:` role.

**Code snippets** should use `.. code-block:: <language>` directive like so

```
.. code-block:: python

   import gcpy
   print("hello world")
```

The language can be "none" to omit syntax highlighting.

For command line instructions, the "console" language should be used. The `$` should be used to denote the console's prompt. If the current working directory is relevant to the instructions, a prompt like `gcuser:~/path1/path2$` should be used.

**Inline literals** (e.g. the `$` above) should use the `:literal:` role.

# INDEX