ComCat Lab Welcome Guide

Welcome to the group! This guide introduces several of the key tools and services that are used in ComCat Lab and examples of their use.

You can view the complete guide in PDF form (`comcatlab-welcome-guide.pdf`), or, alternatively, you can view the guide as a static webpage.

The best way to learn is by doing! Start by picking a tutorial and trying to work through it. The tutorials are (relatively) well documented, though, they are not self-contained. While you're getting acquainted with the material and workflows, feel free to bounce back and forth between the tutorials and the software pages. For more in-depth treatment of the theory, you are encouraged to consult the "Fundamentals" collection in ComCat Lab's Zotero collection. There you'll find annotated original references, more extensive DFT guides, and references to relevant textbooks. Regarding software, the documentation pages are generally extensive. Of course, if all else fails, don't hesitate to reach out to a group member!

Highlights

This guide has several valuable resources to help you get up to speed:

- **Workflows**: an overview of frequent workflows used with links to relevant software pages
- **Software Pages**: introductions to the various tools used in ComCat Lab
- **Tutorials**: walkthroughs for several common workflows/softwares
- **Sample Scripts**: example Python and SLURM scripts
- **Tips**: common issues and their resolutions/work-arounds

Prerequisites

- **Set up your local machine**: this guide assumes that your local machine is setup up with the necessary software already installed (e.g., Hatch, Python, etc.); follow the steps outlined in local-setup to satisfy this prerequisite
- **Set up your cluster account**: some of the tutorials in the guide require that you connect to the remote clusters provided by the Alliance. For these tutorials, you will need a valid CCDB account and to set up your remote cluster environment

Quickstart

These instructions assume that you have already completed setup of your SFU workstation. If you don't have have access to a configured workstation, please

out to a group member to help you set that up, or, if you're feeling up for it, check out the local setup repository that will walk you through a comprehensive setup!

1. Create a virtual environment in the top-level directory of the Welcome Guide folder (in the same directory as this README).

   ```
   python3 -m venv .venv
   ```

2. Activate your environment.

   ```
   source .venv/bin/activate
   ```

3. Install the required Python dependencies.

   ```
   python3 -m pip install .
   ```

4. Pick a tutorial!

Computational Catalysis in a Nutshell

It takes several steps to go from a structure to a publishable result. This page contextualizes common steps and highlights the infrastructure required to execute those steps.

1. **Obtain a structure file.**

   Structure files can be:

   - generated within various Python packages (e.g., ASE, Pymatgen)
   - obtained from the supplementary information of publications
   - retrieved from databases (e.g., Materials Project)

   Common structure file formats:

   - `.traj`: ASE trajectory file
   - `.cif`: Crystallographic Information File
   - `.xyz`: XYZ File Format
   - software-specific formats (e.g., `CONTCAR`, `Gaussian.log`, etc.)

2. **Perform structure manipulations.**

   Depending on the project, this can any combination of the following:

   - modifying the structure (e.g., creating defects, substituting atoms, creating surfaces)
   - adding adsorbates
   - and more...

3. **Determine necessary calculations and parameters.**

This decision is informed by literature, experience, and the scientific questions that one wants to answer. The following papers provide a good starting point for making this decision.

- Best-Practice DFT Protocols for Basic Molecular Computational Chemistry
- Density functional theory methods applied to homogeneous and heterogeneous catalysis: a short review and a practical user guide

4. **Create input files.**

   The required input files for a calculation generally include:

   - a structure file
   - an input parameter file for the DFT code
   - a Python script for interfacing with DFT code or generating results
   - a submission script
   - additional files for the DFT code (e.g., pseudopotentials, wavefunction files, etc.)

   Python and submission script templates for several different job types are described here. The format for parameter files and additional files required by DFT codes are generally specified in the documentation for the corresponding computational code. However, for many DFT codes, there are utilities that can help with the the creation of input parameter files:

   - ASE: of provides Python interface to input file generation for a number of computational chemistry codes
   - MaterialsCloud: provides a graphical interface for generating QuantumEspresso input files
   - VSCode VASP INCAR Pilot Extension: VS Code extension providing help and support for writing VASP `INCAR`, `CONTCAR`, and `POSCAR` files

5. **Submit calculations.**

   Calculations are submitted to remote clusters managed by the Digital Research Alliance (formerly Compute Canada). This is typically done by logging into one of the clusters using `ssh` and executing something like

   ```
   sbatch run.sh
   ```

   from the command line.

6. **Retrieve Results.**

   Although the output files produced by the job contain the results of the calculation, these files are often quite large (~GB), so it is often useful to extract only the specific required outputs. These outputs may include:

   - the final positions of and forces on each atom in the final structure
   - the final energy for the system

- orbital occupancies
- trajectories for an optimization
- vibrational modes

While you can write a script to extract these data from the output files of your job, there is almost definitely a routine to extract this data in either ASE, Pymatgen, or cclib, so search the documentation of existing software for your use case.

7. **Analyze Results.**

   This almost inevitably involves importing data into Excel and performing further analyses there. For some data (e.g., density of states, valence charge density difference diagram, molecular orbital visualization), you may need another software. In such a case, check the tutorials for your use case.

Software Pages

Digital Research Alliance (formerly Compute Canada)

Documentation

This is the national organization that manages our main computational resources. Their documentation is comprehensive, including details about specific clusters, available software, and how to get started. To use these resources, you'll need to create a CCDB account.

Related:

- Creating a CCDB account
- Transferring data to remote clusters
- Setting up your cluster environment

Slurm Workload Manager

Documentation

Slurm is the job scheduler that runs on our remote clusters. You can find a useful cheatsheet here. `sbatch`, `squeue`, and `sacct` are commonly used for submitting and monitoring jobs.

bash

Documentation

bash is a shell program and shell-scripting language. Notably, it is the default on most compute clusters. Navigating and interacting with the file system on remote clusters is done using bash. For example, the command-line functions `ssh` and `scp` are used to connect to and transfer files between the Alliance clusters.

Git

Documentation

Git is a version control system (VCS) that allows you to keep track of multiple versions of files at once. With git it's easy to revert back to an old version of a file or merge different versions of files. This is very helpful when you're trying to determine what change resulted in a particular error. Chapters 1 to 3 provide a good enough background to get started. See Section 5.1 for a description of our git workflow, the "Integration-Manager Workflow". You can find a guide to git best practices here.

Python

Documentation

Python is a relevant programming language in which many useful utilities for our computational workflows are written. Python files can be identified by the `.py` extension.

Atomic Simulation Environment (ASE)

Documentation

The Atomic Simulation Environment (ASE) is a set of tools and Python modules for setting up, manipulating, running, visualizing and analyzing atomistic simulations.

Pymatgen

Documentation

Pymatgen (Python Materials Genomics) is an open-source Python library for materials analysis.

cclib

Documentation

cclib is an open source library, written in Python, for parsing and interpreting the results of computational chemistry packages.

Vienna Ab Initio Simulation Package (VASP)

Documentation

VASP is a proprietary, plane-wave-based code that we use for studying periodic systems.

Quantum Espresso

Documentation

From their webpage:

> Quantum Espresso is an integrated suite of Open-Source computer codes for electronic-structure calculations and materials modeling at the nanoscale. It is based on density-functional theory, plane waves, and pseudopotentials.

Gaussian

Documentation

Gaussian is a proprietary, gaussian-orbital based code that we mainly use for studying molecular systems. The Alliance also maintains a documentation page for running Gaussian on Alliance clusters.

ORCA

Documentation

ORCA is a free-for-academic gaussian-orbital based code that we mainly use for studying molecular systems.

Markdown

Documentation

Markdown is a markup language commonly used for writing documentation. A common extension for Markdown files is `.md`. The documentation for this Welcome Guide is written in Markdown. Markdown files can easily be converted into the pages of a static website using a tool like mkdocs.

Lmod

Documentation

Lmod is a software for managing software environments. The necessary commands and variables required for a software to be used are specified in 'module files' that are written in the Lua programming language. Modules are managed using commands like `module load`, `module purge`, and `module unload`.

Computational Chemistry Utilities

Documentation

CompChemUtils (Computational Chemistry Utilities; AKA `ccu`) is a Python package containing several useful Python classes and routines computational

chemistry such as adsorbate complex creation, defect creation, free energy diagram generation, and more.

Autojob

Documentation

`autojob` is a framework for executing high throughput calculations with the flexibility to granularly resubmit and modifying calculations between execution. Autojob also provides a codeless interface for generating input directories for calculations.

WIP

- Materials Cloud
- Materials Project
- Globus
- FileZilla
- Cyberduck
- bader
- Multiwfn
- Lobster
- chargemol
- Fireworks
- jobflow
- Custodian
- atomate2
- catmap
- matplotlib

Samples

Here, you can find sample files for:

- Python input files
- bash profile files
- Slurm input files
- Software-specific files (e.g., VASP, Quantum Espresso, etc.)

Samples

Here, you can find sample files for:

- Python input files
- bash profile files
- Slurm input files

- Software-specific files (e.g., VASP, Quantum Espresso, etc.)

`.bashrc`

A standard bash profile for use on remote clusters defining utility functions and environment variables for software.

```
py title="samples/bash/.bashrc" --8<-- "./samples/bash/.bashrc"
```

!!! important "Reminder" Replace `username` and `USERNAME` with your username.

Python Scripts

This page describes the collection of Python sample scripts contained in the Python sample scripts folder.

Run a VASP relaxation using the internal quasi-Newton optimization algorithm (RMM-DIIS) in VASP

```
py title="samples/python/run.py" --8<-- "./samples/python/run.py"
```

!!! important "Reminder" Replace `"in.traj"` with the name of your structure file.

Run a mixed-basis Gaussian relaxation using the internal Berny optimization algorithm in Gaussian

```
py title="samples/python/run_gaussian.py" --8<-- "./samples/python/run_gaussian.py"
```

!!! important "Reminder" Replace `"in.traj"` with the name of your structure file, and read this article about selecting which executable to pass to the `command` keyword argument to the `Gaussian` constructor. Note that the syntax is *slightly* different (`g16 < Gaussian.com vs. G16 Gaussian.com`).

Run a VASP relaxation using the BFGSLineSearch optimization routine in ASE

```
py title="samples/python/run_ase.py" --8<-- "./samples/python/run_ase.py"
```

!!! important "Reminder" Replace `"in.traj"` with the name of your structure file.

Run a VASP relaxation using `ccu`

```
py title="samples/python/run_ccu.py" --8<-- "./samples/python/run_ccu.py"
```

`ccu` is a set of tools for computational chemistry workflows. In particular, `run_relaxation` is a wrapper function around `ase.atoms.Atoms.get_potential_energy()` that handles the logging and archiving of a calculation's final results.

!!! important "Reminder" Replace `"in.traj"` with the name of your structure file.

Run an AIMD calculation with Quantum Espresso

```
py title="samples/python/run_espresso.py" --8<-- "./samples/python/run_espresso.py"
```

!!! important "Reminder" Replace `"in.traj"` with the name of your structure file, and don't forget to change the string passed to the `pseudo_dir` keyword argument of the `EspressoProfile` constructor to the absolute path to where your Quantum Espresso pseudopotentials are located.

Slurm Submission Files (WIP)

Submit a DFT calculation

=== "VASP"

```
``` py title="samples/slurm/vasp.sh"
--8<-- "./samples/slurm/vasp.sh"
```
```

=== "Espresso"

```
``` py title="samples/slurm/espresso.sh"
--8<-- "./samples/slurm/espresso.sh"
```
```

```
!!! important "Reminder"
    This script assumes that you are using a self-compiled version of
    Quantum Espresso and have created a corresponding module named
    `espresso`. See [this tutorial](../tutorials/espresso_compilation.md)
    for how to compile Quantum Espresso and create the necessary
    modulefile.
```

=== "Gaussian"

```
``` py title="samples/slurm/gaussian.sh"
--8<-- "./samples/slurm/gaussian.sh"
```
```

=== "ORCA"

```
``` py title="samples/slurm/orca.sh"
--8<-- "./samples/slurm/orca.sh"
```
```

!!! Reminder

```
Don't forget to replace `JOB_NAME`, `SFU_ID`, and `PYTHON_SCRIPT` with
appropriate values in addition to setting your desired SLURM parameters.
Also, if you don't define the path to a Python virtual environment in your
`.bashrc` file, then you should replace `$COMP_CHEM_ENV` with the path to
the `activate` script (usually, `path-to-environment/bin/activate`).
```

Tutorials (WIP)

This page highlights a number of helpful tutorials and resources for getting familiar with the tools we use and workflows that we employ.

Additional Tutorials

ASE Tutorials: how to do basic property calculations, adsorption studies, slab generation, and more...

matgenb: a collection of Jupyter notebooks for Pymatgen, Custodian, and Fireworks compiled by the creators of Materials Virtual Lab, Materials Project

ORCA Tutorials: walkthroughs of how to perform various calculations; part of the official ORCA documentation

ORCA Input Library: a collection of ORCA input files for various types of ORCA calculations previously maintained by Ragnar Bjornsson; a note from the maintainer "There are no immediate plans to update the ORCA Input Library for version 6 compatibility since the maintainer has no time."

Modeling materials using density functional theory: "The Kitchin book"; a textbook with examples of how to use ASE to perform various DFT calculations with VASP; also contains explanations of the underlying theory with references; a PDF version can be found here

Open Catalyst Intro Series: A YouTube playlist from the Open Catalyst Project motivation atomistic simulation

QE-2019: course materials from the 2019 Summer School on Advanced Materials and Molecular Modeling; contains background theory and examples with Quantum Espresso

Online Course from Ghent University: a free, online course in Computational Materials Physics offered by Ghent University; examples in Quantum Espresso

Hands-On Quantum Espresso: a collection of tutorials for installing Quantum Espresso and using the SCF (`PWscf`) and phonon (`PHonon`) modules in Quantum Espresso with theoretical background

Atomistic Computer Modeling Of Materials (SMA 5107): lecture notes, videos, and problem sets from the Ceder/Marzari graduate class taught at MIT

QE YouTube Tutorials: a YouTube playlist with beginner tutorials for Quantum Espresso

Calculation of Phonons with FHI-vibes: a comprehensive tutorial outlining how to calculate phonons and relevant theory

Tutorials (WIP)

This page highlights a number of helpful tutorials and resources for getting familiar with the tools we use and workflows that we employ.

Additional Tutorials

ASE Tutorials: how to do basic property calculations, adsorption studies, slab generation, and more...

matgenb: a collection of Jupyter notebooks for Pymatgen, Custodian, and Fireworks compiled by the creators of Materials Virtual Lab, Materials Project

ORCA Tutorials: walkthroughs of how to perform various calculations; part of the official ORCA documentation

ORCA Input Library: a collection of ORCA input files for various types of ORCA calculations previously maintained by Ragnar Bjornsson; a note from the maintainer "There are no immediate plans to update the ORCA Input Library for version 6 compatibility since the maintainer has no time."

Modeling materials using density functional theory: "The Kitchin book"; a textbook with examples of how to use ASE to perform various DFT calculations with VASP; also contains explanations of the underlying theory with references; a PDF version can be found here

Open Catalyst Intro Series: A YouTube playlist from the Open Catalyst Project motivation atomistic simulation

QE-2019: course materials from the 2019 Summer School on Advanced Materials and Molecular Modeling; contains background theory and examples with Quantum Espresso

Online Course from Ghent University: a free, online course in Computational Materials Physics offered by Ghent University; examples in Quantum Espresso

Hands-On Quantum Espresso: a collection of tutorials for installing Quantum Espresso and using the SCF (`PWscf`) and phonon (`PHonon`) modules in Quantum Espresso with theoretical background

Atomistic Computer Modeling Of Materials (SMA 5107): lecture notes, videos, and problem sets from the Ceder/Marzari graduate class taught at MIT

QE YouTube Tutorials: a YouTube playlist with beginner tutorials for Quantum Espresso

Calculation of Phonons with FHI-vibes: a comprehensive tutorial outlining how to calculate phonons and relevant theory

Creating a CCDB Account (WIP)

You should receive an email invitation.

Transferring Data to a Remote Cluster (WIP)

Using `scp`

Using `rsync`

Using Globus

Using another option

viz. FileZilla or CyberDuck.

:sparkles: Make your own Quantum ESPRESSO compilation :sparkles:

Last Updated: October 18, 2024

Overview

This guide will walk through compiling Quantum Espresso with the following options

- compiled with Intel OneAPI libraries
- signal trapping: This enables QE to recognize common signals sent by job schedulers (e.g., SIGTERM) and exit gracefully
- exit statuses
- libxc support: This expands the library of functionals that can be used
- Environ: This compiles Quantum Espresso with the Environ library, enabling the use of advanced solvent models (e.g., SCCS, SSCS)

Tips

You may need to replace `def-samiras` with a valid project directory. To see which project directories are available, run:

```
ls ~/projects
```

Note that as of 2024/10/20, the installation instructions for Environ on the documentation are out-of-date for Environ 3+. The GitHub repository contains sparse instructions for Environ 3+ here.

A good practice for managing the installed software is to store the raw source code in a separate folder from where it is installed. For example, one can keep the pre-compile source code in a `software_support` folder and install the software in a `software` folder. In this way, if one every has to compile the software, they can simply delete the associated subdirectory in `software` without having to worry about the source code. Additionally, for large source codebases, one can archive the directory in `software_support` to conserve the file quota. This can be done for the Intel OneAPI, QuantumEspresso, and libxc software in this tutorial.

Step-by-Step

Here's a (not really) quick step-by-step to compile Quantum Espresso (QE) 7.3.1

1. **Download the Intel OneAPI libraries** from here.
2. **Copy the folder/archive of Intel OneAPI files** to where you will install QE. This may be in your home directory (e.g., `/home/USER`) or a subdirectory of your project folder (e.g., `/home/USER/projects/def-samiras/USER/software_support`).

   ```
   scp -r oneapi_archive $USER@Cedar.computecanada.ca:/home/$USER/
   ```

   where `USER` is your Digital Research Alliance (DRA) username.

   !!! note

   ```
   These libraries are from the [OneAPI suite][oneapi-suite], however,
   only the current year's libraries are available for download for free. The
   2024 libraries were tested and they did not work for QE compilation on
   Cedar. The 2022 have been confirmed to worked. Both the 2022 and 2023
   libraries are included in the shared folder from step 1.
   ```

3. **Install the Base Toolkit.** (This could take up to 30 minutes.) Navigate to where you copied the folder on Cedar

   ```
   $SHELL l_BaseKit_p_2022.1.1.119.sh
   ```

   This will install the Base Toolkit under the directory `/home/USER/intel/oneapi` (where `USER` is your DRA username).

   Optionally, you can change the installation location by opting to customize the installation (select "Accept & customize installation" prior to starting the installation process).

   You may want to do this if you would like the libraries to be accessible for others, in which case, change the default installation directory to a

Base Toolkit Installer Homepage

Figure 1: Base Toolkit Installer Homepage

Base Toolkit Customization

Figure 2: Base Toolkit Customization

subdirectory of the project folder. For example,

!!! warning

`You may receive warnings about the operating system being "Unknown" or missing packages required for the Intel® VTune(TM) Profiler, but you can ignore these.`

4. **Install the HPC Toolkit.** (This shouldn't take more than a few minutes.)

   `$SHELL l_HPCKit_p_2022.1.1.97.sh`

   Again, you may want to specify a custom installation location. In that case, it is reasonable to create a `oneapi` directory as a subdirectory of the custom location specified for the Base Toolkit and use this as the installation location.

   At this point, the directory that was chosen as the installation location should be populated with the Intel libraries, and there should be a file called `setvars.sh`, which when sourced (e.g., `source setvars.sh`), should give you a list of modules loaded in the environment. Check this step prior to QE compilation.

5. **Download Quantum Espresso 7.3.1**. You can get QE here. Alternatively, you can just go to their homepage, register your email, and go to Downloads > to find the latest version.

6. **Copy the downloaded `QE.tar` file to Cedar and extract it**.

7. **Forcibly purge your loaded modules** and reload the Gentoo Linux module.

   ```
   module --force purge
   module load gentoo
   ```

   !!! warning

   `This is important: the main problem with Cedar seems to be some library/ies dependency, so make sure you run module --force purge before moving on.`

   !!! note

   `The Gentoo Linux module provides access to the` ` `git` ` `CLI utility which is needed to configure the Environ module.`

8. **Setup the Intel environment.**

   ```
   source /path/to/setvars.sh
   ```

9. **Locate the root directory for libxc.** On Cedar, this directory is found at `/cvmfs/soft.computecanada.ca/easybuild/software/2023/x86-64-v3/Compiler/gcc12/libxc/6.2`

   !!! note

   ```
   Note that the latest version of libxc (7.0.0) is not installed on
   Cedar. However, installing libxc is quite straightforward. Instructions can
   be found [here][libxc-installation].
   ```

10. Ensure that the local language is set to the standard, i.e. "C".

```shell
export LC_ALL=C
```

11. **Obtain a copy of Environ 3+** and copy it to the QE folder. You can either clone it from the git repository by running this command from inside of the QE folder.

    ```
    git clone https://github.com/environ-developers/Environ.git
    ```

    or you can download an archive, copy it to Cedar, and then extract it into the QE folder.

    !!! warning

    ```
    The `Environ` folder should be inside of the Quantum Espresso
    folder (e.g., `qe-X.Y.Z/Environ`).
    ```

12. **Configure the QE compilation.**

    For clarity, we define variables for the location of the Intel Base Toolkit and where you would like to install the QE binaries. If you didn't modify the default directory where the Intel libraries are installed, then you should define `intel_dir` as:

    ```
    intel_dir=/home/$USER/intel/oneapi
    ```

    If you used a custom directory, your variable definition may look like this:

    ```
    intel_dir=/home/$USER/projects/def-samiras/$USER/software/intel-2022.1.1
    ```

    Next, specify a path (outside of the current directory) where you would like to install the QE executables.

    ```
    espresso_dir=/home/$USER/projects/def-samiras/$USER/software/espresso-X.Y.Z
    ```

    !!! note

    ```
    Note that these directories must be **absolute** paths (i.e., starting
    with `/`).
    ```

Run the `configure` script from inside the `qe-X.Y.Z` folder (where `X.Y.Z` is the QE version number).

```
./configure LIBDIRS="$intel_dir/mkl $intel_dir/mpi $intel_dir/compiler" --enable-parall
```

!!! note

```
This will take a while, (between minutes and hour-ish), so be sure
to have some time at this step. Keep an eye on what will come up at the
screen, because this will inform whether or not QE found the libraries
you told it to, or if it skipped any of them. It will also let you know
if the parallel compilation was successfully identified (QE has different
compilations for serial and parallel execution). Make sure it found all
dependencies you need before moving on. If you are unable to see all the
output from the command in your terminal, it may be useful to redirect
the standard output and standard error from the `./configure` comand to a
file:

```shell
./configure LIBDIRS="$intel_dir/mkl $intel_dir/mpi $intel_dir/compiler" --enable-parall
```
```

This command will redirect all the configuration information to `espresso.log`.

13. **Modify the `make.inc` file to configure libxc.**

    This includes:

    - adding `-D__LIBXC` to `DFLAGS`
    - adding `-I/path/to/libxc/include` to `IFLAGS`
    - setting `LD_LIBS=-L/path/to/libxc/lib -lxcf03 -lxc`

    Note that `/path/to/libxc` should be the path determined in Step 9.

14. **Configure Environ.**

    Change the current directory to inside the `Environ` folder and run:

    ```
    ./configure LIBDIRS="$intel_dir/mkl $intel_dir/mpi $intel_dir/compiler" --enable-parall
    ```

    !!! note

    ```
    It is very important to use the same libraries and parallelization
    flags for Environ as for QE.
    ```

15. **Compile Environ.**

    ```
    make -jN compile
    ```

    where `N` is the number of processors to use for parallel compilation.

16. **Build the QE executables.**

    ```
    make all
    ```

:star2: The QE manual is really good and useful and easy to follow, in case you want to test different things on your own.

:red_circle: You can also compile their default version, which is mostly foolproof and easy, and test it. It usually finds its own way and works well. Beware that there is a problem with library dependencies in the cluster, so if you compile the default version, it will have the same libraries as the cluster version and likely have the same issues!

!!! note

```
**Tip**: To speed up the installation, you can spawn an interactive
job with multiple cores and run `make` in parallel. For example, if you
want to spawn a job with 16 cores, do the following:

```shell
salloc --mem=32GB --ntasks-per-node=16 --nodes=1 --account=def-samiras --time=00:10:00
make -j16 all
```
```

I've found that this scales linearly with 32 cores taking about 2 minutes to complete.

17. **Install the executables.**

    ```
    make install
    ```

    This step copies the built executables to the directory specified by the `--prefix` option (the `espresso_dir` variable defined in Step 11).

Modulefiles

Recall that the `setvars.sh` script **must** be sourced prior to executing QE in order to setup the Intel environment for Quantum Espresso. This has a few disadvantages, the most significant of which being that there is no straightforward way to undo the changes. Additionally, there is the issue of convenience and portability due to having to specify the *exact* location of the `setvars.sh` script in order to source it. This section covers how to create modulefiles for Quantum Espresso and the Intel OneAPI libraries.

Intel OneAPI

Modulefiles solve the above problems and offer some additional benefits. In short, modulefiles provide a convenient way to dynamically change the users' environment. This may involve modifying environment variables, defining shell functions, or loading other modulefiles. The Digital Research Alliance clusters use Lmod to manage modulefiles. Lmod provides the `sh_to_modulefile` utility. See this tip for how to convert `setvars.sh` into a modulefile.

Once you have created the modulefile, ensure that the file is in your `MODULEPATH` (this is controlled by the command `module use`). A reasonable strategy is to name the modulefile after the version of the libraries (e.g., 2022.1.1) and to place this file in a subdirectory of a path in `MODULEPATH`. The name of the subdirectory will be the name used to load the module (i.e., set up the environment), so a reasonable name would be something like `intel`. Additionally, it's useful to add the following metadata to the modulefile:

"'text title="intel/2022.1.1.lua" help([[Description =========== Intel C, C++ & Fortran compilers (classic and oneAPI)

More information

- Homepage: https://software.intel.com/content/www/us/en/develop/tools/oneapi/hpc-toolkit.html]]) whatis("Description: Intel C, C++ & Fortran compilers (classic and oneAPI)") whatis("Homepage: https://software.intel.com/content/www/us/en/develop/tools/o toolkit.html") whatis("URL: https://software.intel.com/content/www/us/en/develop/tools/oneapi/hpc-toolkit.html") whatis("Version: 2022.1.1") whatis("Keywords: HPC") conflict("intel")

```
This mostly just descriptive information about the module. The last line
ensures that you don't load conflicting modules and is especially helpful
since we've had troubles with the default DRA libraries. An error will
be thrown if you try to load this module when another version is already
loaded.

If you placed the created file in a directory called `intel` in your `MODULEPATH`
and called the file `2022.1.1` (after the version of the Intel OneAPI
libraries), then you perform the required set up with the command:

```shell
module load intel/2022.1.1
```

Quantum Espresso

You can also define a modulefile for your newly installed version of Quantum Espresso. Essentially, the only necessary change that must be made is to prepend the directory containing QE executables to the `PATH` environment variable. A sample modulefile is shown below:

"'text title="espresso/7.3.1.lua" help([[For detailed instructions, go to: https://www.quantum-espresso.org/documentation/package-specific-documentation/

]])

whatis("Version: 7.3.1") whatis("Keywords: QuantumEspresso")
```

conflict(“quantumespresso”) conflict(“espresso”)

depends_on(“intel/2022.1.1”) prepend_path(“PATH”, “espresso_dir/bin”)

```
Here, `espresso_dir` should be replaced with the path used to define
`espresso_dir` in Step 11. Also, note that the DRA module is named
`quantumespresso` whereas this module is named `espresso`. The `conflict`
commands reflect this fact. This is beneficial (but not necessary) since ASE's
Quantum Espresso calculator class is called `Espresso`, so both Python and
SLURM submission scripts can be templated with the same variable. Finally,
note that this modulefile "depends_on" the `intel/2022.1.1` modulefile. This
ensures that the `intel/2022.1.1` module is loaded prior to loading the
`espresso` module Optionally, one can explicitly load the intel module by
replacing that `depends_on` with `load`.

## Running a sample calculation

The following commands should be added to your SLURM submission script to run
calculations with QE.

```shell
module --force purge
source /home/USER/intel/oneapi/setvars.sh
export PATH=$PATH:/home/USER/qe-7.3.1/bin/
export PATH=$PATH:/home/USER/intel/oneapi/mpi/2021.5.0/bin
/home/YOUR$USERNAME/intel/oneapi/mpi/2021.5.0/bin/mpirun pw.x < espresso.in > espresso.out
```

where `USER` is your DRA username.

If you’ve defined modules as indicated above, add the following commands instead.

```
module --force purge
module load gentoo intel/2022.1.1 espresso/7.3.1
mpirun pw.x
```

If running Quantum Espresso with ASE in a script called `run.py`, add the following commands instead.

```
module --force purge
module load gentoo intel/2022.1.1 espresso/7.3.1

# Load your Python environment here, if necessary

python3 run.py
```

If you changed the `depends_on` command to `load`, then you don’t need to load the `intel` module.
```

Benchmarking

Finally, benchmark and test its performance using their provided examples or your job. What to look for:

- The performance should be equivalent to their examples for simple structures with similar parameters. For unit cells, it is useless to scale up too many cores, so test it with few at first (1 core vs 2 vs 4, for example). Check the last lines of the out file the total CPU time and resources. If the performance is worse with more cores, the compilation is not good.
- Check the performance of a decent sized job with your compilation against the one available in Cedar, the time should be around half but it will also depend on your own system (it was half for mine with a ~100 atoms supercell). If it is similar to Cedar, or worse, again, the compilation it not good.

How to Optimize Resource Requests (WIP)

This page outlines some considerations for balancing job performance and queue wait times.

It's a good idea to become familiar with how the clusters are partitioned. This can be useful in maximizing the number of nodes that can potentially pick up your job. More detailed information about each cluster can be found on their respective software page (e.g., Cedar).

For monitoring the live status of nodes on a cluster, there is the `partition-stats` command:

```
$ partition-stats
Node type |                    Max walltime
          |   3 hr   |  12 hr  |  24 hr  |  72 hr  |  168 hr |  672 hr |
----------|-------------------------------------------------------------
          Number of Queued Jobs by partition Type (by node:by core)
----------|-------------------------------------------------------------
Regular   |   12:170 |  69:7066|  70:7335| 386:961 |  59:509 |   5:165 |
Large Mem |    0:0   |   0:0   |   0:0   |   0:15  |   0:1   |   0:4   |
GPU       |    5:14  |   3:8   |  21:1   | 177:110 |   1:5   |   1:1   |
----------|-------------------------------------------------------------
Number of Running Jobs by partition Type (by node:by core)
----------|-------------------------------------------------------------
Regular   |    8:32  |  10:854 |  84:10  |  15:65  |   0:674 |   1:26  |
Large Mem |    0:0   |   0:0   |   0:0   |   0:1   |   0:0   |   0:0   |
GPU       |    5:0   |   2:13  |  47:20  |  19:18  |   0:3   |   0:0   |
----------|-------------------------------------------------------------
          Number of Idle nodes by partition Type (by node:by core)
----------|-------------------------------------------------------------
```

```
Regular   |   16:9   |  15:8   |  15:8   |   7:0   |   2:0   |   0:0   |
Large Mem |    3:1   |   3:1   |   0:0   |   0:0   |   0:0   |   0:0   |
GPU       |    0:0   |   0:0   |   0:0   |   0:0   |   0:0   |   0:0   |
----------|-----------------------------------------------------------
       Total Number of nodes by partition Type (by node:by core)
----------|-----------------------------------------------------------
Regular   |  871:431 | 851:411 | 821:391 | 636:276 | 281:164 |  90:50  |
Large Mem |   27:12  |  27:12  |  24:11  |  20:3   |   4:3   |   3:2   |
GPU       |  156:78  | 156:78  | 144:72  | 104:52  |  13:12  |  13:12  |
----------|-----------------------------------------------------------
```

Calculate IR Intensities with VASP (WIP)

This tutorial covers how to calculate IR spectra using VASP.

1. Relax the structure.
2. Conduct a phonon calculation.

   It is important to set the following `INCAR` tags:

   - `EDIFF`: should be no larger than 10-6
   - `IBRION`: set to 5 (do not use symmetry to reduce the number of directions) or 6 (use symmetry to reduce the number of directions)
   - `LEPSILON`: must be set to `.TRUE.` in order to calculate the Born effective field tensor
   - `NFREE`: set to either 2 or 4; determines how many displacements in each direction are used for the finite differences
   - `NSW`: must be set to exactly 1 in order for the finite differences step to be calculated
   - `POTIM`: should be set to a very small number; the default is 0.015 but a valid number can be obtained by progressively increasing from 0.015 and comparing with the results from a DFPT phonon calculation

   Additionally, the calculation must be a gamma-point only calculation since as of 4/17/2024, VASP DFPT only works for

   $$q = \Gamma$$

   . Also note that, depending on how you run VASP, a separate executable may be selected based on the k-points chosen for the calculation. DFPT calculations are not supported with the `vasp_gam` executable.

Resources

VASP Wiki for IBRION INCAR Tag Stack Exchange Question with Instructions

Useful Links

This page is a catch-all for any links on specific topics that you may find useful that don't easily fall into the other categories.

Selecting the Right Number of Cores for a VASP Calculation

Matter Modeling Stack Exchange

Gaussian Error Wiki

More Gaussian Error Help

Snippets

This page contains useful snippets for performing common tasks.

Modulefiles

Create a modulefile from a script

Lmod provides a handy utility `sh_to_modulefile` for creating modulefiles from scripts. `sh_to_modulefile` records the initial environment, runs the script, and compares the final environment to determine the changes. It then converts these changes to Lua commands and prints them to the terminal. Given a shell script, `my_script.sh`, one can run the following command to generate an Lmod-compatible modulefile written in Lua.

```
$LMOD_DIR/sh_to_modulefile my_script.sh > my_script.lua
```

The command `$LMOD_DIR/sh_to_modulefile` calls the utility by using the Lmod-defined environment variable `LMOD_DIR`, which points to the directory in which Lmod is installed. `my_script.sh` is the name of the script to be converted into a modulefile. The output of the command is redirected (`>`) to the file `my_script.lua`.

SLURM

Attaching to Existing Jobs

Say, you have a job running on Cedar (this could be an interactive job launched with `salloc` or a job submitted with `sbatch`) and you would like to run some commands on the compute node with the job's resources (cores, memory, GPU, etc.). If the job ID for the job is JOBID, you can "attach" to the job with the command:

```
srun --pty --overlap --jobid JOBID bash
```

This snippet calls the `srun` command of SLURM. The `--pty` option launches a task in "pseudo terminal mode", allowing for you to specify commands from the

command line as if you were running a terminal session on the compute node. The `--overlap` option allows your terminal session to share resources with the running job. The `--jobid` option specifies the SLURM job ID to attach to.

!!! warning

`Since the commands that you run in this terminal will share the resources of the running job, if you exceed the memory request for the job, you can cause your original job to fail due to an out-of-memory (OOM) error.`

Troubleshooting

This page contains potential solutions to whatever may currently be troubling you. Your mileage may vary.

VASP

VASP internal routines have requested a change of the k-point set

Context Running phonon calculations for nitrate reduction intermediates on M-BHT slabs.

Environment

- **Cluster**: Cedar
- **Date**: July 9, 2024
- **Software Versions**: Python 3.11.9, ASE 3.23.0, VASP 5.4.4

Relevant Files Input files for the calculation.

=== “INCAR”

```text
--8<-- "./docs/sources/resources/files/kpt_set_change_error/INCAR"
```

=== “run.py”

```py
--8<-- "./docs/sources/resources/files/kpt_set_change_error/run.py"
```

=== “vasp.sh”

```py
--8<-- "./docs/sources/resources/files/kpt_set_change_error/vasp.sh"
```

The Error

“text title=“vasp.out”

```
|
EEEEEEE RRRRRR RRRRRR OOOOOOO RRRRRR ### ### ### |
E R R R R O O R R ### ### ### |
E R R R R O O R R ### ### ### |
EEEEE RRRRRR RRRRRR O O RRRRRR # # # |
E R R R R O O R R |
E R R R R O O R R ### ### ### |
EEEEEEE R R R R OOOOOOO R R ### ### ### |
|
VASP internal routines have requested a change of the k-point set. |
Unfortunately this is only possible if NPAR=number of nodes. |
Please remove the tag NPAR from the INCAR file and restart the |
calculations. |
|
—-> I REFUSE TO CONTINUE WITH THIS SICK JOB ..., BYE!!! <—- |
|
```

“‘

This is an error due to symmetry breaking (see here).

The Solution Set `ISYM=0`. in your `INCAR` file.

ASE is not copying the `vdw-kernel.bindat` to the job directory (WIP)

Context Running any calculation with VDW corrections.

Environment

- **Cluster**: Cedar
- **Date**: July 9, 2024
- **Software Versions**: Python 3.11.9, ASE 3.23.0, VASP 5.4.4

The Error You set the `ivdw` keyword argument when configuring a VASP calculator in ASE, but you notice that the `vdw-kernel.bindat` file is not copied to the calculation directory.

The Solution As of ASE 3.23.0, the `vdw-kernel.bindat` is only copied if you also set the keyword argument `luse_vdw=True` for the VASP calculator.

Gaussian

Frequency Calculations

Watch out for internal rotations! If performing frequency calculations in Gaussian, you may get the following warning:

```
Warning -- explicit consideration of  23 degrees of freedom as
        vibrations may cause significant error
```

See here for a resolution.

General

Property Calculation

When calculating zero-point energies using computational codes, it is worth it to examine whether there is an internal routine. In the case of VASP, one can use the INCAR tag `IBRION=5` (or 6) to perform such a calculation using finite differences. The benefit in this case over using the ASE reliant method `ccu.thermo.vibration.run_vibration` is that all code-specific data is retained. For example, one can also go back and collect IR frequency data from the calculated dipoles of each image.

Additionally, one should also check whether IR frequency data is any more expensive than ZPE data. For example, in VASP, dipole moments are calculated at every optimization point during a phonon frequency calculation. This means that if the phonon calculation is executed in VASP, IR frequencies are obtained for free. This benefit highlights a major advantage to executing frequency calculations in VASP as opposed to with an external routine like ASE's `ase.vibration.vibration.Vibration`. ASE does not archive all calculated results for each image, but instead, only records calculated forces.

How to Contribute

This page highlights sections of the guide that need work and explains how to report issues/suggest new features.

Development Workflow

Assuming that you have set up your environment:

1. Create and checkout a branch for your local changes.

   ```
   git checkout -b name-of-feature-or-change
   ```

   !!! reminder It is really important that you do not push your changes directly to the `main` branch. In the case that someone makes a mistake with `git`, it is much easier to correct the issue on a single fork (these can be deleted and re-forked relatively painlessly) than for the whole project.

See Integration-Manager Workflow for a more detailed explanation of the workflow.)

2. Make your changes (e.g., add/change/remove files).
3. Commit your changes. Each commit should include a brief (50 characters or less) description of the changes. If you can't describe your changes in 50 characters or less, you should probably break the commit into smaller pieces! This is a great guide to good commits.

   ```
   git commit -S -m "Added VASP relaxation python script"
   ```

4. Push your changes to **your** remote.

   ```
   git push origin name-of-feature-or-change
   ```

5. Create a pull request with a summary of your changes using the pull request template. Please remember to complete the checklist!

Issues

Maybe you notice an error in the documentation, or maybe, a new version of a particular software causes one of the tutorials to no longer work. This is a perfect chance to raise an issue! Please use an issue template to describe the problem and try to include the following information, where relevant:

- **System**: operating system, software version (e.g., ASE 3.23.0 or VASP 5.4.4)
- **Intended behaviour**: what should be happening?
- **Problem**: what's actually happening?
- **Minimum Reproducible Rxample (MRE)**: once you have an example of the problem, try to remove as much as possible from your example while still retaining the error

Features

If you think that adding something would improve the guide, suggest it by submitting an issue. Please explain what the change would be and how it would benefit the guide. From there, you can create a pull request with the change!

Where to Contribute

- Writing GitHub templates for issues/MRs
- Configuring GH actions
- Writing software pages
- Creating tutorials for new calculations
- Uploading sample files
- Explaining solutions to technical issues
- Compiling useful links

- Any page/section marked with "WIP" ("work-in-progress")

Guidelines for Contributing

When adding new content to tutorials and sample files, please to try following the existing format. Some general tips:

- Try to limit markdown lines to 80 characters to promote readability. In some cases (e.g., URLs, code, etc.), this is not possible.
- Please try to use fenced code blocks to display any code that should be copied. This ensures that `mkdocs` formats the code correctly and adds copy buttons. This greatly simplifies the user experience.
- Specify a language for fenced code blocks. Often this will simply be `py`, `shell`, or `text`. The language is specified inline with the first three backticks used to indicate the start of the fenced code block.
- If referring to another document, please consider linking to it directly to make it easier for users to navigate.

The following sections have more specific guidelines for adding content to particular sections.

Samples

- Add new sample files to an appropriate subdirectory in the top-level `samples/` directory and add a corresponding explanation of the sample file in the `docs/sources/samples` directory
- Files can be "included" in markdown files using "Snippets Notation". The general syntax is `--8<-- ./path/to/file`. `./path/to/file` should be the path to the sample file.
- You can add a title to a fenced code block using "Code Block Title Headers". To specify the title, add `title=Name of the Title` inline with the code language. Check the Python sample files source code for examples.

Tutorials

- Specify a "Last Updated" date when writing tutorials. Software is always changing, so a tutorial written today may not work tomorrow. Indicating when the tutorial was last updated gives users an idea of its currency and us an idea of what may need updating.

Troubleshooting

- Describe the problem as completely as possible. This includes listing all relevant software and system information (and versions) and all necessary steps to reproduce the problem. If you consulted a resource to find the solution, it is also helpful to include links to these resources for reference.

Snippets

- Describe the use-case as clearly as possible
- Annotate your code so that someone who is not so familiar with the language will be able to follow along with what each part of the code is trying to achieve; if there are any tricky details, this is a good way to highlight them!

Building the Documentation

This page describes how the documentation pages of the Welcome Guide are built.

Build Tools

This guide uses `mkdocs` to convert the Markdown files to HTML files and build the webpage. All the configuration for using `mkdocs` is contained within the `mkdocs.yml` configuration file in the project root. For a detailed description, please see the mkdocs documentation. For theme options, please see the documentation for material for mkdocs.

Building the Webpage

To build the webpage,

1. Activate the development environment.

   If you are using Hatch:

   ```
   hatch shell
   ```

   If you created a virtual environment with `venv` (e.g., `python -m venv .venv`):

   ```
   source .venv/bin/activate
   ```

   Note that at the very least, you should have the `docs` extra installed (e.g., `pip install '.[docs]'`).

2. Build the HTML files.

   ```
   mkdocs build
   ```

   The output should be something like this:

   ```
   INFO    -  Cleaning site directory
   INFO    -  Building documentation to directory: /Users/ugo/Projects/nwt/welcome-guide/d
   INFO    -  The following pages exist in the docs directory, but are not included in the
                - resources/conferences.md
                - resources/links.md
                - resources/references.md
   ```

```
            - resources/social.md
            - software_pages/index.md
INFO    -  Documentation built in 0.19 seconds
```

3. Serve the webpage.

   ```
   mkdocs serve
   ```

   The output should be something like this:

   ```
   INFO    -  Building documentation...
   INFO    -  Cleaning site directory
   INFO    -  The following pages exist in the docs directory, but are not included in the
               - resources/conferences.md
               - resources/links.md
               - resources/references.md
               - resources/social.md
               - software_pages/index.md
   INFO    -  Documentation built in 0.16 seconds
   INFO    -  [16:49:00] Watching paths for changes: 'docs/sources', 'mkdocs.yml'
   INFO    -  [16:49:00] Serving on http://127.0.0.1:8000/
   ```

4. Enter the IP address into your browser (here, `127.0.0.1:8000/`)

Building the PDF Version of the Guide

To build the PDF version of the guide, you will need to install Pandoc and a LaTeX engine. You will then need to remove all emoji symbols from the markdown source documents. Finally, from the project root, run:

```
cd docs/sources || exit; pandoc --file-scope -s -o ../../comcat-lab-welcome-guide.pdf -f mar
```

If you have hatch configured, you can run:

```
hatch run docs:build-pdf
```

This command will collate the markdown source files into a single PDF **in the order that their filenames** are passed to the `pandoc` command. The PDF will be saved to the project root in a file called `comcat-lab-welcome-guide.pdf`.

!!! warning

```
As of 2024/10/23, the formatting for this is still problematic. Code
formatting is poor, included files (e.g., in `samples/`) are not rendered,
and symbols result in errors.
```

Setting Up for Local Development

This page only covers the basic setup to enable local development of the welcome guide. Building the documentation is covered elsewhere.

Installing Prerequisites

1. Install Python 3.10 or greater.
2. (Optional) Install Hatch. Handles environment management and provides some handy command-line functions to simplify documentation building.
3. (Optional) Install Pandoc. Only required to build the PDF version of the guide.

Getting the Source Files

1. Create your own fork of the Welcome Guide (look for the "Fork" button on GitHub).
2. Clone your fork.

   with `ssh` (requires that your `ssh` key is added to your GitHub account):

   ```
   git clone git@github.com:yourusername/welcome-guide.git
   ```

   with `https` (requires login to GitHub account):

   ```
   git clone https://github.com/yourusername/welcome-guide.git
   ```

3. Initialize the development virtual environment.

   with Hatch:

   ```
   hatch env create default
   ```

   with `venv` and pip:

   ```
   python3 -m venv .venv
   source .venv/bin/activate
   python3 -m pip install '.[test,dev,docs]'
   ```

4. Install the pre-commit hooks.

   ```
   pre-commit install-hooks
   ```