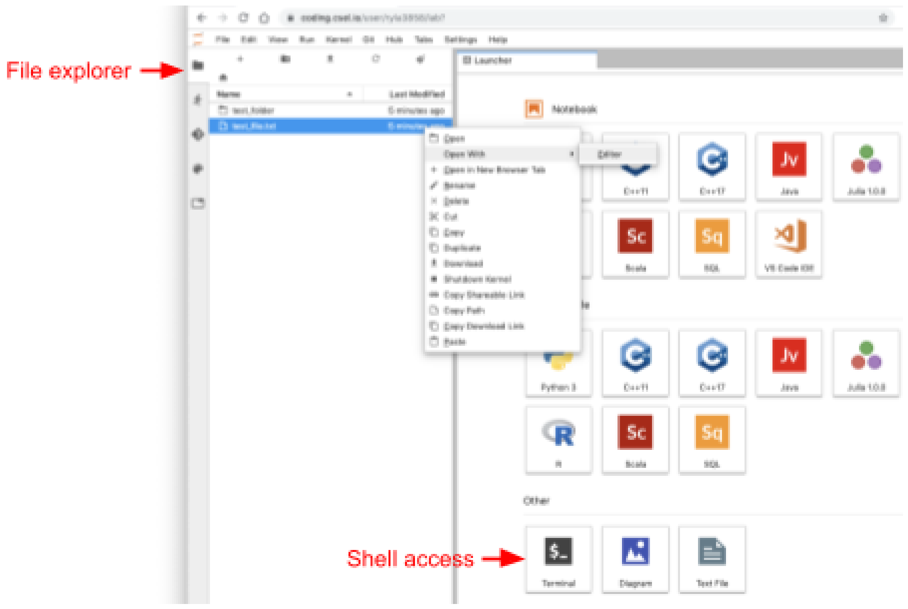
Development Environment
Software Engineering for Scientists

The reference environment is the CU CS JupyterHub at https://coding.csel.io/ Choose "Default Coding Environment". This interface includes a file explorer for creating and editing files and a command-line interface.

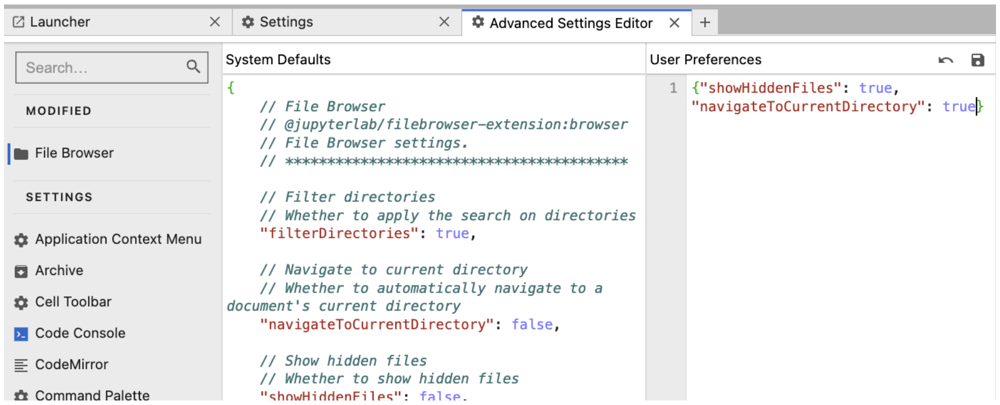


Initialize CU CS JupyterLab setup do this just once

Showing Hidden Files

In a Linux environment configurations are stored in *hidden* text files that start with a "." and are not listed by default. This prevents them from cluttering your environment and makes accidental changes less likely, but we will be working with hidden files and need to change the environment so we can see them in the file explorer. To do this:

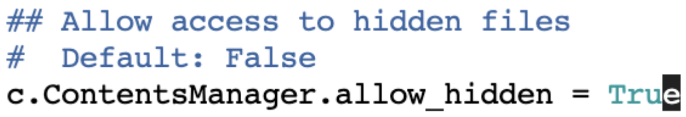
- Go to Settings->Advanced Settings Editor.
- Click "JSON Setting Editor" on top right.
- Click "File Browser" in the left pane. If you don't see it, you can get it by entering "File Browser" in the search bar.
- Add `"showHiddenFiles": true` between the braces (`{}`) in the User preferences file on the right.
- You can also add `"navigateToCurrentDirectory": false` to that list. While not required I find it helpful. This is a list, so don
- Save by clicking the floppy disk button.



```
{
    // File Browser
    // @jupyterlab/filebrowser-extension:browser
    // File Browser settings.
    // *****************************************

    // Filter directories
    // Whether to apply the search on directories
    "filterDirectories": true,

    // Navigate to current directory
    // Whether to automatically navigate to a
document's current directory
    "navigateToCurrentDirectory": false,

    // Show hidden files
    // Whether to show hidden files
    "showHiddenFiles": false,
```

```
{"showHiddenFiles": true,
"navigateToCurrentDirectory": true}
```

- Launch the terminal and run `$ jupyter server --generate-config`
- The config file will be saved into `~/.jupyter/jupyter_server_config.py` where we need to change "c.ContentsManager.allow_hidden" to True in the file.
    - Open the file with `$ nano .jupyter/jupyter_server_config.py`
    - Jump to line 1041 with Ctrl + _
    - Remove the `#` and change `False` to `True`



```
## Allow access to hidden files
#  Default: False
c.ContentsManager.allow_hidden = True
```

    - Save and exit with Ctrl X
- Log-out (File->Log Out) and log-in is required to make this change effective

Managing Dependencies

Ensuring that all software dependencies are met is a major obstacle to reproducible scientific software. There are many ways to help communicate these dependencies and simplify the installation process. In this class, we will use conda (https://conda.io/), which is an environment manager and a package manager. More specifically we will use a new, faster implementation of conda called mamba. Mamba is already installed in this environment, but we need to do a few setup steps. **NOTE: These steps are specific to this environment.** Keep reading for instructions for getting mamba installed in other environments (i.e., your laptop).

```
$ mamba init
...
Added mamba to /home/jovyan/.bashrc

==> For changes to take effect, close and re-open your current shell. <==
```

This command added everything needed to initialize mamba to the `.bashrc` file in your home directory. The issue is that in this environment the `.bashrc` is ignored, which is not normal. To fix this run

```
$ echo ". $HOME/.bashrc" >> .bash_profile
```

Log out, then log back in. If it worked, then you should see `(base)` in the command line prompt.

```
(base) $
```

To ensure that your environments are persistent, we need to change the directory they are stored in.

- Right click in the file explore and create a new file

- Name it `.condarc`
- Open it (now that you can see it)
- Add the lines:

```
envs_dirs:
- /home/jovyan/.conda/envs
```

Installing Conda in your environment if you don't have it (`$ conda` gives an error)

This is confusing. Anaconda is a python distribution that is meant to make data science easier. Anaconda provided **conda**. Conda became popular independent of anaconda, which led to **miniconda**, which cuts out all of the extra stuff in anaconda. Conda can be very slow, which led to the more efficient **mamba** (https://github.com/mamba-org/mamba). We will use mamba in this class.

Pick the version that best matches your environment here:

- https://github.com/conda-forge/miniforge#mambaforge-pypy3

For example in Linux you

```
$ cd $HOME
$ curl -L -O
https://github.com/conda-forge/miniforge/releases/latest/download/Mambaforge-pypy3-Linux-
x86_64.sh
$ bash Mambaforge-pypy3-Linux-x86_64.sh.sh
```

NOTE: The second command (`$ curl -L -O <url>`) is long and wraps across lines 2-4. Coping and pasting into your terminal may result in 2 commands, first `$ curl -L -O` then `$ https://....` To fix this copy and paste in two steps. First `curl -L -O` then the URL into the same line.

Once conda is installed, please update it and then install mamba

```
$ mamba update --yes mamba
```

Create an environment just once per environment

```
$ mamba create --yes -n swe4s
...
#
# To activate this environment, use
#
#     $ conda activate swe4s
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

Staring Conda do this every time you log in

```
$ mamba activate swe4s

(swe4s) $
```

Installing software/libraries

With Conda

```
(swe4s) $ mamba install -y pycodestyle
Looking for: ['pycodestyle']
...
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

With pip

```
(swe4s) $ pip install pycodestyle
Collecting pycodestyle
...
```

If pip is not installed:

```
$ sudo easy_install pip
$ sudo pip install --upgrade pip
```

Conda (Mamba) vs Pip

Pip is the basic Python Packaging Authority's recommended tool for installing packages from the Python Package Index, PyPI. You can use pip to install python packages on your computer, without being in a conda environment. Pip installs Python software packaged as wheels (pre-compiled) or source distributions (not pre-compiled). The latter may require that the system have compatible compilers, and possibly libraries, installed before invoking pip to succeed.

Conda is a cross-platform package *and* environment manager that installs and manages conda packages from the Anaconda repository as well as from the Anaconda Cloud. You can have different Conda environments for different projects, it is a nice way to keep all the dependencies your code needs together, not worry about them, and replicate the environment on another machine later.

Conda packages are binaries. There is never a need to have compilers available to install them. Additionally, conda packages are not limited to Python software. They may also contain C or C++ libraries, R packages, or any other software.

If you install packages with mamba, in your Conda environment, you do not need to ALSO pip install. However, if you change environments, you *will* need to install packages in the new environment as well. They do not transfer. We will be using one conda environment for this course so you do not need to worry about this.