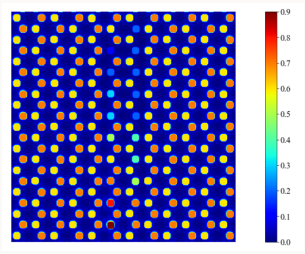
Setting environment efficiently on server

Xinzhe Dai dxz2019@mit.edu

STEM@UCAS

Version: 1.00

Update: August 16, 2022



0 Don't be afraid and let's start

Scientific computation is a huge project sometimes. For this reason, a good habit to manage your code is important. This user manual will guide you through out the basic environment setting, project management and advanced usage on server. Don't be afraid of Linux. You will gradually be familiar with it starting from this script, and find its efficiency later on.

1 Setting up computational environment

Once you log in the server for the first time with your account and password, you will arrive at the home directory.

1.1 Install Miniconda

Up to now, your home directory are nearly blank with only some basic files (probably you cannot see them because they are hidden files). We need firstly obtain a tool to help us install other useful packages, like abTEM for STEM simulation or atomap for analysis of STEM images. It is highly recommended to use conda as the tool. It is a commonly-used in scientific computation with plenty of python packages easy to be installed with.

Here we use miniconda - a minimal installer for conda - as example. Follow the installation instruction to acquire miniconda and install it. You could download it with command line as followed.

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-
   Linux-x86_64.sh
```

1.2 Setup environment

To activate it for the first time, try `source ~/.bashrc` and hopefully, you will find command line become as followed.

```
(bash) user@p3 [11:33:52]:~
$
```

By the way, you don't need to repeat this code next time you open the terminal, because it already been added to `.bashrc` file and will be automatically loaded.

Next, you'd better create an environment for further installing other packages. Basically, environment is a virtual space with an specified python interpreter. Each environment is isolated and cannot talk to each other. This is deliberately made to avert the possible incompatibility between different packages suitable for python's with distinct version. Now create a new environment called `py38env` with new python interpreter (version 3.8). You can show all environments with `conda env list`.

```
conda create -n py38env python==3.8 # To create a new env.
conda env list # To show all envs .
```

The parenthesis in the front of command line reveals that you still in the base environment. To change the environment to the new one you created, you could use `conda activate py38env`. The `(base)` changed to `(py38env)` correspondingly. Inside the new environment, the installed packages can be listed with `conda list`.

```
conda activate py38env # To activate an env .
conda list # To list all installed packages .
```

You probably already noticed that only a few packages are installed. To install other packages you will use, try conda install (packages to be installed).

Now let's install `numpy` and `matplotlib`, two of the most basic packages for matrix calculation and plotting figure.

```
conda install numpy # To install numpy .
conda install matplotlib # To install matplotlib.
```

Don't forget to update all the packages after you install them, because conda will not check the updating automatically for you like pip.

```
conda update --all # To update all packages installed.
```

Now we can try out the packages we installed via python.

```
import numpy as np # Import numpy.
import matplotlib.pyplot as plt # Import matplotlib.
x = np.arange(0,2*np.pi,0.01)
y = np.sin(x)
plt.plot(x,y) # Plot sinusoidal function.
plt.savefig('fig.png') # Save figure.
```

Now you could find the figure we plotted with ls to list all the files in current directory.

2 Project management

It is significant to manage the program clearly and efficiently for scientific computation.A good habit will save you much time for something more important.

Now create a folder for simulation with `mkdir` and get inside it with `cd`.

```
mkdir simulation # Create a folder called simulation.
cd simulation # Get inside the folder simulation.
```

You can create a new file and open it with `vim`. Vim actually is an Linux editor with its own usage. If you want to use other editor, gedit or vscode with ssh connection could be helpful.

```
vim inputs.py # Create and eedit file called inputs.py.
```

Inside the vim screen, hit `i` to activate the editing mode. Then you could edit the file like other common editor with your keyboard.

```
# Write python script here .
defocus = 40 # Define sefocus .
Cs = 1.2e5 # Define spherical aberration.
```

After editing, hit `Esc` to exit editing mode and press `:wq` to save the file and exit vim.

You could use `rm` to delete a file and `rm -r` to delete a folder.

```
rm inputs.py # Remove file.
cd .. # Return to the last directory.
rm -r simulation # Remove folder.
```

3 Running multi-batch missions

3.1 Setup

According to the instruction in Github, setup the preliminary works to run multi-batch mission.

After you setup the necessary codes, you could try to submit single job with `sbatch`or multi-jobs using `bash` file. Go to the directory where you put `runs.sh`, `multi_run.sh` and `run.sh`.

```
sbatch run.sh    # Single job submission.
bash runs.sh     # Multi-batch jobs submission.
```

You could track all running missions with `screen -ls`.

```
screen -ls  # Show all running missions.
```

3.2 Change parameters

Some parameters can be changed to modify path for output file, error file and job name, etc.

```
#!bin/bash
#SBATCH --output=run_out.log
#SBATCH --error=run_error.log
#SBATCH --job-name=run

python script.py
```

The `output` variable read the path for output file, while the `error` variable read the path for error file. You could name the submission by `job_name`.

For multi-batch submission, the usage is similar to single job submission. The only difference is that you need to write a bash script to submit all jobs with `sbatch` iteratively.