Python 及 Pymatgen 软件的安装

Pymatgen 的安装需要 Python2.7 以上版本，需要调用 Tck/Tk(要求版本 8.6 以上)，因此手动安装时配置如下:

Tcl/Tk 的安装配置

./configure --prefix=/share/home/jiangjun/Softs/tcl8.6.5 --enable-shared --enable-64bit --enable-symbols
make && make install
./configure --prefix=/share/home/jiangjun/Softs/tk8.6.5 --enable-shared --enable-64bit --enable-symbols
make && make install

lapack 和 atlas 的安装参见文档。建议**:** 用 **OpenBlas** 代替 **atlas** 更方便
安装 OpenBlas 非常简单: make CC=gcc FC=gfortran

请注意: 为了让 lapack 和 atlas 最后生成动态库函数，编译 lapack 库时，在 make.inc 中每个编译选项都加上-fPIC -m64
如
FORTRAN = gfortran
OPTS = -O3 -std=legacy -m64 -fno-second-underscore -fPIC -c
DRVOPTS = $(OPTS)
NOOPT = -O0 -frecursive -fPIC -m64
LOADER = gfortran
LOADOPTS = -fPIC -m64

CC = gcc
CFLAGS = -O3 -fPIC -m64

在非 root 权限下，atlas 推荐的编译选项:

../configure -b 64 -C ic gcc -C if gfortran -Fa alg -fPIC --prefix= 指定安装目录 --with-netlib-lapack= 静态库函数 liblapack.a 的绝对路径

手工将静态库编译为动态库
gfortran -fPIC -m64 -shared liblapack.a -o liblapack.so
gfortran -fPIC -m64 -shared libblas.a -o libblas.so

参见文档ATLAS_Numpy_Scipy_Theano 的环境搭建.pdf

python build finished successfully!
The necessary bits to build these optional modules were not found:
$_bz2_curses_curses_panel$
$_dbm_gdbm_hashlib$
$_lzma_sqlite3_ssl$
$_tkinterreadline$
$To find the necessary bits, look in setup.py in detect_modules() for the module's name.$

$The following modules found by detect_modules() in setup.py, have been$
$built by the Makefile instead, as configured by the Setup files :$
$_abcatexitpwd$
$time$

$Failed to build these modules :$
$_ctypes$

$Could not build the ssl module!$
$Python requires an OpenSSL 1.0.2 or 1.1 compatible libssl with X509\_VERIFY\_PARAM\_set1_host().$
$LibreSSL 2.6.4 and earlier do not provide the necessary APIs, https : //github.com/libressl - portable/portable/issues/381$

需要安装的库函数:

```
yum -y groupinstall "Development tools"
yum -y install zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel readline-devel tk-devel db4-devel libpcap-devel xz-devel
yum install -y libffi-devel zlib1g-dev
yum install zlib* -y
```

python 的源码包安装配置

./configure CC=gcc --prefix=/share/home/jiangjun/Softs/Python-2.7.11 --enable-universalsdk --with-universal-archs="64-bit" --with-cxx-main=g++ --with-tcltk-includes='-I/share/home/jiangjun/lib/tcltk/include' --with-tcltk-libs='/share/home/jiangjun/lib/tcltk/lib/libtcl8.6.so /share/home/jiangjun/lib/tcltk/lib/libtk8.6.so'
make && make install

类似地，**python3** 的源码包安装配置

./configure CC=gcc --prefix=/share/home/jiangjun/Softs/Python-3.9.5 --enable-universalsdk --with-universal-archs="64-bit" --with-cxx-main=g++ --with-tcltk-includes='-I/share/home/jiangjun/lib/tcltk/include' --with-tcltk-libs='/share/home/jiangjun/lib/tcltk/lib/libtcl8.6.so /share/home/jiangjun/lib/tcltk/lib/libtk8.6.so' --LDFLAGS='-L/usr/local/lib" CPPFLAGS="-I/usr/local/include' --with-openssl=/home/TEST/.local/openssl
make && make install

setuptools 安装 (参见文档 Python 的安装过程 (含 setuptools).pdf)

python setup.py build
python setup.py install

采用 setuptools，在可以实现在普通用户账号下安装模块 (ez_setup.py 是 python 官方给出的一个安装 setuptools 的工具)，如:

/share/software/python-2.7.10/bin/python ez_setup.py --user jiangjun

easy_install 会被安装在 ~/.local/bin/ 目录下)

在此基础上，通过选项 -d 可将各种模块安装到普通用户账号下的指定目录:
~/.local/bin/easy_install -d ~/.local/lib/python2.7/site-package 安装模块
在 ~/.bashrc 中加入下列变量 (即 PYTHONPATH 包含新增模块目录):
export PYTHONPATH=$PATHONPATH:~/.local/lib/python2.7/site-package

有此PYTHONPATH也可以用命令行，将有关模块直接安装到指定环境下
python setup.py install --prefix=~/.local
参见文档
Python 环境变量 PYTHONPATH 设置和 easy_install 简单使用.pdf
可以在配置文件 ~/.pydistutils.cfg 中指定下载镜像:
[easy_install]
index_url = http://pypi.douban.com/simple/
index_url = http://e.pypi.python.org/simple

通用 pip 安装模块命令:
pip install Module==version -i http://mirrors.aliyun.com/pypi/simple --trusted-host mirrors.aliyun.com --user

- 模块用 == 指定版本
- -i 和--trusted-host 指定 pip 下载库位置
- --user 指定在当前用户的~/.local/lib/python 版本下安装模块

如果采用 pip 安装模块，建议通过配置文件 ~/.pip/pip.conf 中指定有关参数:
[global]
timeout = 6000
index-url = http://pypi.douban.com/simple/
[install]
use-mirrors = true
mirrors = http://pypi.douban.com/simple/
trusted-host = pypi.douban.com
install-option=--prefix="~/.local " # 指定安装路径选项，而且引号不能少

在无法联网的服务器上用 pip 安装 Python 模块的一些处理:
下载安装模块: https://pypi.python.org/simple/模块.tar.gz (无需解压)
pip install 模块. 版本.tar.gz

Python 运行 pip 时,如果提示Can't connect to HTTPS URL because the SSL module is not available.-skipping,
就需要安装 openssl(1.1 或 1.02 版) 或 libressl 的 2.6.4 版本:

- wget http://www.openssl.org/source/openssl-xxx.tar.gz
- tar xvzf openssl-xxx.tar.gz
- ./config -prefix=/usr/local/openssl-xxx –openssldir=/usr/local/openssl-xxx/openssl no-zlib # 建议加上 no-zlib 否则会出现 undefined symbol: SSL_CTX_get0_param 错误

- make && make install
- echo “/usr/local/openssl-xxx/lib” » /etc/ld.so.conf
- ldconfig -v

重新编译和安装 Python，用选项--with-openssl=$openssl 的安装目录，在 Ubuntu 下记得修改 Module/Setup 文件:

- vim /root/Python-XXXX/Modules/Setup
- 修改结果如下:

```
# Socket module helper for socket(2)
_socket socketmodule.c timemodule.c
# Socket module helper for SSL support; you must comment out the other
# socket line above, and possibly edit the SSL variable:
SSL=/usr/local/openssl-xxx
_ssl _ssl.c
-DUSE_SSL  -I$(SSL)/include   -I$(SSL)/include/openssl
-L$(SSL)/lib -lssl -lcrypto
```

- 运行 python
- import ssl 测试正常即可

在此基础上，准备手动安装 numpy、scipy 等模块，建议指定环境变量 ATLAS(版本 3.8.4)、LAPACK(版本 3.6.0)、BLAS
注意: 环境变量要具体到绝对路径 (包括文件名)，如:
export ATLAS=/share/home/jiangjun/lib/atlas3.8.4/lib/libatlas.so
export LAPACK=/share/home/jiangjun/lib/atlas3.8.4/lib/liblapack.so
export BLAS=/share/home/jiangjun/lib/atlas3.8.4/lib/libblas.so
分别进入 numpy 和 scipy 目录
在文件 site.cfg 中指定有关库函数的参数 (注意: 写绝对路径)

```
[ALL]
library_dirs =/share/home/jiangjun/lib/atlas3.8.4/lib
include_dirs =/share/home/jiangjun/lib/atlas3.8.4/include
src_dir=/share/home/jiangjun/lib/atlas3.8
search_static_first=0
#
# Atlas
# -----
# Atlas is an open source optimized implementation of the BLAS and Lapack
# routines. Numpy will try to build against Atlas by default when available in
# the system library dirs. To build numpy against a custom installation of
# Atlas you can add an explicit section such as the following. Here we assume
# that Atlas was configured with “prefix=/opt/atlas“.
#
[atlas]
library_dirs = /share/home/jiangjun/lib/atlas3.8.4/lib
include_dirs = /share/home/jiangjun/lib/atlas3.8.4/include
atlas_libs=lapack, f77blas, cblas, atlas
```

[blas_opt]
library_dirs = /share/home/jiangjun/lib/atlas3.8.4/lib
include_dirs = /share/home/jiangjun/lib/atlas3.8.4/include
blas_libs=f77blas, cblas, atlas

[lapack_opt]
library_dirs = /share/home/jiangjun/lib/atlas3.8.4/lib
include_dirs = /share/home/jiangjun/lib/atlas3.8.4/include
lapack_libs=lapack, atlas

[amd]
amd_libs = amd
#
[umfpack]
umfpack_libs = umfpack

用 **Intel_mkl** 库支持 **numpy** 和 **scipy**
[mkl]
library_dirs = $intel 编译器安装目录/mkl/lib/intel64/
include_dirs = $intel 编译器安装目录/mkl/include
mkl_libs = mkl_rt mkl_blas95_lp64 mkl_core mkl_intel_lp64 mkl_intel_thread
lapack_libs =mkl_lapack95_lp64

修改 $numpy-目录/numpy/distutils/intelccompiler.py:
将self.cc_exe(class intel 或 class intelem 里的)为:
self.cc_exe = 'icc -O3 -g -fPIC -fp-model strict -fomit-frame-pointer -openmp -xhost'
numpy 编译:
python setup.py config --compiler=intel build_clib --compiler=intel build_ext --compiler=intel install
64 位将intel改为intelem
scipy 编译:
python setup.py config --compiler=intel --fcompiler=intel build_clib --compiler=intel --fcompiler=intel build_ext --compiler=intel --fcompiler=intel install
64 位同 numpy 修改命令行

python setup.py build
sudo python setup.py install

在 Ubuntu 系统中，如果有超级用户权限，则可以简单处理为:
sudo apt-get install libopenblas-dev liblapack-dev
export BLAS=/usr/lib/libblas.so
export LAPACK=/usr/lib/liblapack.so

在此基础上完成安装:
pip install numpy
pip install scipy

virtualenv: 用来建立一个虚拟的 python 环境，一个专属于项目的 python 环境。用 virtualenv 来保持一个干净的环境非常有用

1. 基本使用: 通过 pip 安装 virtualenv:
   pip install virtualenv
2. 测试安装:
   virtualenv --version
3. 为一个工程项目搭建一个虚拟环境:
   cd my_project
   virtualenv my_project$_{e}nv$python Python "python2.7" :
   virtualenv -p /usr/bin/python2.7 my_project_env
   my_project_env python Python pip Python ""Python
4. 要开始使用虚拟环境，其需要被激活:
   source my_project_env/bin/activate
5. 停用虚拟环境:
   deactivate
   停用后将回到系统默认的 Python 解释器

1 关于 MongoDB 的启动

MongoDB 可以通过控制文件 (如文件名为 mongodb.config) 来控制启动，mongodb.config 中指定各种参数:

```
fork = true
dbpath = /home/jun_jiang/WORKS/TEST/TEST_mongo/Data_Base
logpath = /home/jun_jiang/WORKS/TEST/TEST_mongo/mongo.log
logappend =true
journal = true
# repair = true
bind_ip = 127.0.0.1,192.168.113.42# 这里的 IP(192.168.113.42) 为数据库服务器的内部 IP(即 ifconfig 命令查询到的本机 IP)
port = 27017
```

如果需要在同一地点启动多个数据库，只需要修改 dbpath 即可

如 mongodb_atomate.config:

```
fork = true
dbpath = /home/jun_jiang/WORKS/TEST/TEST_mongo/Data_Base_Atomate
logpath = /home/jun_jiang/WORKS/TEST/TEST_mongo/mongo.log
logappend =true
journal = true
# repair = true
bind_ip = 127.0.0.1,192.168.113.42# 这里的 IP(192.168.113.42) 为数据库服务器的内部 IP(即 ifconfig 命令查询到的本机 IP)
port = 27017
```

如 mongodb_firework.config:

```
fork = true
dbpath = /home/jun_jiang/WORKS/TEST/TEST_mongo/Data_Base_Firework
logpath = /home/jun_jiang/WORKS/TEST/TEST_mongo/mongo.log
logappend =true
journal = true
# repair = true
bind_ip = 127.0.0.1,192.168.113.42# 这里的 IP(192.168.113.42) 为数据库服务器的内部 IP(即 ifconfig 命令查询到的本机 IP)
port = 27017
```

启动数据库的命令为: mongod -f mongodb_XXXX.config

进入数据库的命令为: mongo，然后用 help 检查 MongoDB 的各种命令

2 Pymatgen

采用 Pymatgen 生成 VASP 的 POTCAR 库函数时，用的命令是
pmg config -p Dir-source Dir-object
然后用
pmg config add PMG_VASP_PSP_DIR Dir-object
生效即可

pmg potcar 命令是用于根据 POTCAR 库产生对应元素的 POTCAR，在 ATOMATE 系统中， DIR-object 必须是

$$\text{POT\_GGA\_PAW\_PBE}, \text{POT\_LDA\_PAW}$$

3 ATOMATE 下的数据库配置

启动支持计算进程的 FireWorks 数据库:
mongod -f ./mongodb_fireworks.config
启动支持计算数据存储的 Atomate 数据库:
mongod -f ./mongodb_atomate.config.config

- ATOMATE 环境下，加载 FireWorks 的总体配置: /.bashrc 中写入环境变量:
  export FW_CONFIG_FILE=«INSTALL_DIR»/config/FW_config.yaml
  FW_config.yaml 的内容:

```
LAUNCHPAD_LOC: /home/TEST/WORKS/FireWorks_TEST/my_launchpad.yaml
FWORKER_LOC: /home/TEST/WORKS/FireWorks_TEST/my_fworker.yaml
QUEUEADAPTER_LOC: /home/TEST/WORKS/FireWorks_TEST/my_qadapter.yaml
CONFIG_FILE_DIR: /home/TEST/WORKS/FireWorks_TEST
```

- 文件 my_launchpad.yam 指定了 FireWorks 支持的计算流程存储的数据库的 (相关登录) 信息，(这个文件指定的参数一般不要改动)

```
authsource: fireworks # 指定支持计算流程的数据库为 FireWorks
host: localhost # 指定 IP 或域名
logdir: null
mongoclient_kwargs: {}
name: fireworks # 指定数据库登录账号为 FireWorks
password: null  # 指定数据库登录密码
port: 27017  # 指定数据库端口
ssl: false  # 默认为 false，以下全部空;true，以下填入合适的参数
ssl_ca_certs: null
ssl_certfile: null
ssl_keyfile: null
ssl_pem_passphrase: null
strm_lvl: INFO
uri_mode: false
user_indices: [ ]
username: null
wf_user_indices: [ ]
```

- 文件 my_fworker.yaml 指定了运行所需计算软件、计算数据存储数据位置信息

```
name: mgo-test # 指定计算对象 (任务) 的名称
category: ”
query: ’{}’
env:
    db_file: /home/TEST/WORKS/Mongo_TEST/Cal_Data_Base/db.json # 指定计算数据存储用数据库的位置，db.json 文件包含计算存储数据库的信息
    vasp_cmd: mpirun -np 4 /home/TEST/Softwares/vasp.5.4.4/bin/vasp_std # 指定核心计算所需的引擎
    scratch_dir: null
```

- 文件 my_qadapter.yaml # 这个文件指定了作业管理系统支持下提交计算作业的参数和形式， 这个文件是支持 qlaunch rapidfire

$$-m\ 3$$

命令的运行，类似于直接启动
rlaunch -l /home/TEST/WORKS/FireWorks_TEST/my_launchpad.yaml -w /home/TEST/WORKS/FireWorks_TEST/my_fworker.yaml rapidfire

```
_fw_name: CommonAdapter
# _fw_q_type: PBS # SLURM
_fw_q_type: LoadSharingFacility
# _fw_template_file: / share/ home/ czjiangjun/ WORKS/ TEST_ATOMATE/ LSF_template_custom.txt rocket_launch: rlaunch -w / home/ jun-jiang/ WORKS/ FireWorks/ my_fworker.yaml -l /home/jun-iang/ WORKS/ FireWorks/ my_launchpad.yaml rapidfire singleshot
# 本行指定的就是每个进程命令
# rocket_launch: rlaunch -c /share/home/czjiangjun/WORKS/TEST_FireWorks singleshot rapidfire
# rocket_launch: rlaunch -c /share/home/czjiangjun/WORKS/TEST_FireWorks singleshot
# rocket_launch: bsub ./stand.lsf nodes: 4
ntasks_per_node: 2
# walltime: 24:00:00
queue: para2
# account: czjiangjun
job_name: VASP_Cal
pre_rocket: null
post_rocket: null
# 以下是针对本单位 LSF 的改动脚本:
# NP_PER_NODE: 12 # (一个节点跑 n 个进程，不指定系统按照资源使用情况自主决定)
MPI_TYPE: openmpi # (选择 mpi 的类型)
MPI_HOME: /home/jun-jiang/Softswares/openmpi-1.8.4 # (mpi 的路径)
RUN: RAW
# OMP_NUM_THREADS=4 #(如果使用 OpenMP 或者是单节点启动多个进程，需要通过该参数修正资源的分配，该值表示一个进程中运行了 4 个线程)
_q_commands_override:
    submit_cmd: bsub
    # submit_cmd: ./
    # status_cmd: bstatus
```

计算结果数据库文件 db.json

```
{
    "host": "localhost", # 指定计算结果数据库的 IP 或域名
    "port": 27018, # 指定计算结果数据库的端口
    "database": "Cal_Data", # 指定计算结果数据的数据库名
    "collection": "tasks", #FireWorks 中的 FireTasks
    "admin_user": "", # 指定不同级别用户的账号和密码
    "admin_password": "",
    "readonly_user": "",
    "readonly_password": "",
    "aliases": {}
}
```

执行任务的步骤

- 启动计算进程管理数据库、启动计算数据存储数据库
- lpad reset # 清理以往进程 (如果是从头开始的进程)
- python3 mgo_bandstructure.py # 启动全部计算进程

- qlaunch rapidfire [-m 3] # 有任务管理系统支持作业提交 / rlaunch -l /home/TEST/WORKS/FireWorks_TEST/my_launchpad.yaml -w /home/TEST/WORKS/FireWorks_TEST/my_fworker.yaml rapidfire # 单机提交作业
  # 启动 FireWorks 支持的计算任务
- python3 mgo_analysis.py # 计算结果数据分析