Spack简明使用手册

> Spack 是由美国劳伦斯利弗莫尔国家实验室发起的一种针对于可续计算软件的包管理工具，旨在支持各种平台和环境上的多个版本和软件配置。它是为大型超级计算中心设计的，在这些中心，许多用户和应用程序团队使用没有标准 ABI 的库在具有异构的集群上共享软件的通用安装。Spack 是非破坏性的：安装新版本不会破坏现有安装，因此许多配置可以在同一系统上共存。
>
> 最重要的是，Spack 很简单。它提供了一个简单的规范语法，以便用户可以简洁地指定版本和配置选项。Spack 对于包作者来说也很简单：包文件是用纯 Python 编写的，规范允许包作者为同一包的许多不同构建维护一个文件。

1. 基础命令

常用命令如下：

命令	功能	例子
spack list	列出可用spack安装的全部软件	
spack info	查询可安装软件的基本信息	spack info vasp
spack find	查询已安装的软件	spack find gcc
spack load	加载已安装软件的环境变量	spack load cmake
spack unload	卸载加载的软件的环境变量	spack unload cmake
spack install	安装软件	spack install gromacs

2. 环境部署

在Spack的github的官方仓库下载其最新稳定版本：

https://github.com/spack/spack/releases

系统基础环境：

依赖的名字	支持版本	附加	依赖的原因
Python	2.7/3.5-3.9		Spack解释器
C/C++ Compilers			构建软件
make			构建软件
patch			构建软件
bash			编译器包装器
tar			提取/创建 存档
gzip			压缩/解压缩 档案
unzip			压缩/解压缩 档案
bzip			压缩/解压缩 档案
xz			压缩/解压缩 档案
zstd		可选	压缩/解压缩 档案
file			创建/使用构建 缓存
gnupg2			签名/验证 缓存
git			管理软件存储库
svn		可选	管理软件存储库
hg		可选	管理软件存储库
Python 头文件		可选	从源引导

2.1 加载spack包管理器环境变量

```
# 对于 bash/zsh/sh
source spack/share/spack/setup-env.sh
```

灵犀易算平台，默认地将spack稳定版本部署在 用户家目录的 `.spack` 文件夹下的 `spack` 文件夹中

> `$HOME/.spack` 文件夹还存放了spack的其他配置和软件的环境变量文件

在用户的bash配置文件 `~/.bashrc` 中，我们默认加上了 `source /es01/yeesuan/<user>/.spack/spack/share/spack/setup-env.sh` 使得用户打开终端时就加载好了spack包管理器的环境变量

2.2 将编译器加入spack编译器配置

执行`spack compiler find`可以将当前环境变量的编译器加载到spack编译器配置中,默认的配置文件在`$HOME/.spack/linux/compilers.yaml`

默认使用的系统是Centos7.6，所以默认的gcc、g++、gfortran版本是4.8.5

为了方便使用更多的软件，我们使用spack编译了gcc(包含g++、gfortran、ada、jit等)`6.5.0`、`8.5.0`、`10.2.0`,还安装了intel2019(兼容性最好)

2.3 纳入已安装的软件到spack

执行`spack exteranl find`可以将当前环境变量中包含bin目录的软件纳入spack管理，默认的配置文件在`$HOME/.spack/packages.yaml`

2.4 添加上游软件仓库

默认地软件仓库在全局共享文件系统下的`/es01/jnist/software/public`下，使用这个仓库安装好的软件，修改的上游软件仓库配置文件`$HOME/.spack/upstreams.yaml`：

```
upstreams:
  general:
    install_tree: /es01/jnist/software/public
```

我们默认地添加了这个软件仓库

2.5 添加软件源码镜像仓库

执行`spack mirror add <name> <path>`,例如添加软件源码镜像仓库`spack mirror add local /es01/jnist/mirror`,我们默认添加了这个镜像仓库

2.6 其他基础配置

通过`spack config get config > config.yaml`生成一份spack基础配置到`$HOME/.spack`下，这里可以修改最大的编译并行核数选项`build_jobs`，`install_tree`下的`root`条目为当前用户spack的软件安装目录

2.7 软件环境变量

由于spack集成了`Environment Modules`管理软件，你可以直接使用系统中`module avial`来查看可用软件的环境变量，默认的环境变量文件存放位置`$HOME/.spack/spack/share/spack/modules`

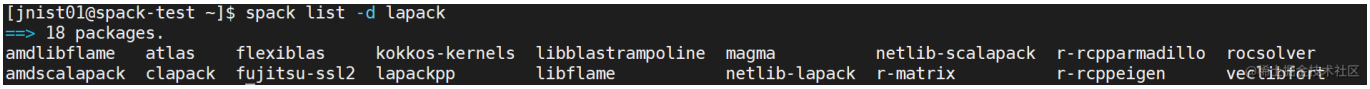
3.命令详解

3.1 查询可安装软件

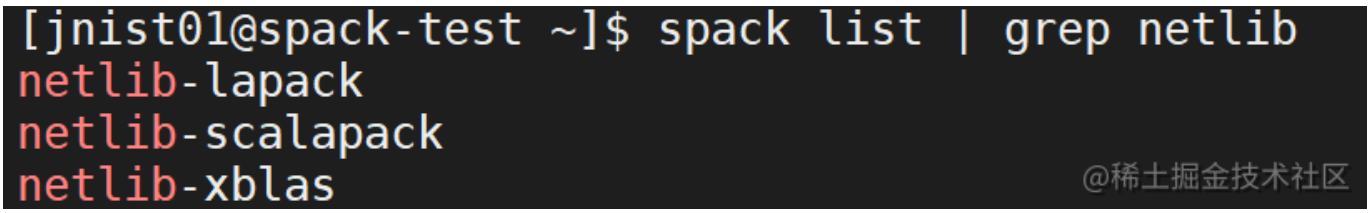
执行`spack list`可以查询可由spack安装的全部软件

有的时候，所需的软件的名字可能在spack软件列表不同，比如数学库blas、lapack、scalapack的实现会有好多种,下面我们展示下所有lapack的实现, 只需要在list后加`-d`参数,

```
spack list -d lapack
```



当然，我们也可以使用`grep`关键词过滤查找到所需要的软件的名字



3.2 查询可安装软件的基本信息

执行`spack info <package>`可以查询可安装软件的基本信息

例如，查询`vasp`的基本信息：

```
spack info vasp
```

这样会打印出：

```
MakefilePackage:   vasp

Description:
    The Vienna Ab initio Simulation Package (VASP) is a computer program for atomic
scale materials modelling, e.g. electronic structure calculations and quantum-
mechanical molecular dynamics, from first principles.

Homepage: https://vasp.at

Externally Detectable:
    False

Tags:
    None

Preferred version:
    6.1.1        file:///es01/home/jnist01/vasp.6.1.1.tgz

Safe versions:
    6.1.1        file:///es01/home/jnist01/vasp.6.1.1.tgz
    5.4.4.pl2    file:///es01/home/jnist01/vasp.5.4.4.pl2.tgz
    5.4.4        file:///es01/home/jnist01/vasp.5.4.4.tgz

Deprecated versions:
    None

Variants:
    Name [Default]     When    Allowed values    Description
    ===============    ====    ==============
================================================================================

    cuda [off]         --      on, off           Enables running on Nvidia GPUs
    scalapack [off]    --      on, off           Enables build with SCALAPACK
    vaspsol [off]      --      on, off           Enable VASPsol implicit solvation
model

https://github.com/henniggroup/VASPsol

Installation Phases:
    edit    build    install

Build Dependencies:
    blas  cuda  fftw  lapack  mpi  netlib-scalapack  qd  rsync

Link Dependencies:
    blas  cuda  fftw  lapack  mpi  netlib-scalapack  qd

Run Dependencies:
    mpi
```

```
  Virtual Packages:
      None
```

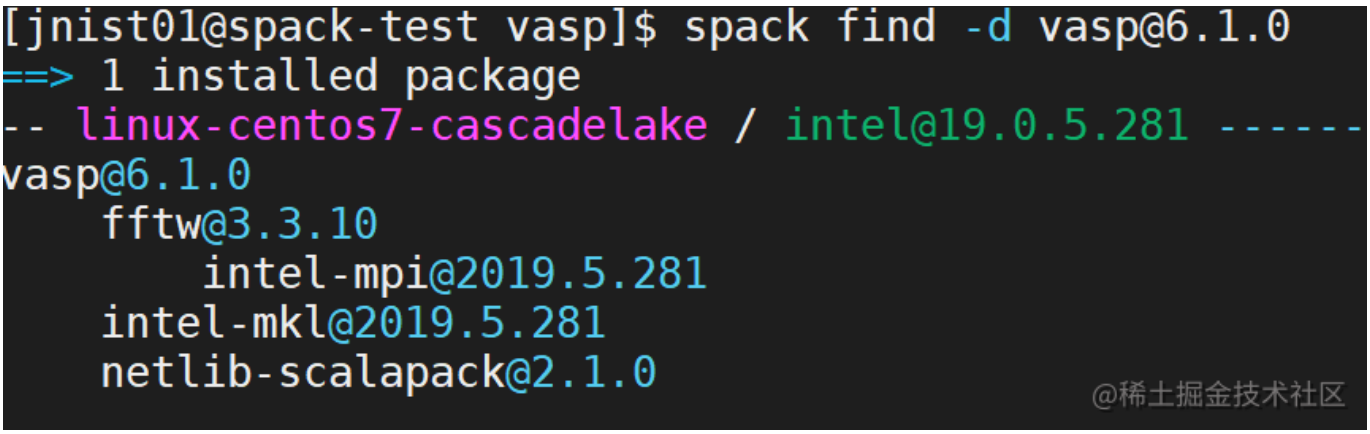
其中`Preferred version`项是安装此软件默认安装的版本号，`Build Dependencies`是所需的依赖,`Variants`项为该软件特性，安装软件的时候可以 用"+"或"-"选择开启或者关闭

3.2 查询已安装软件

执行`spack find`可以查询使用spack安装好的`全部`软件

如图所有软件根据编译目标架构和编译器分成了两部分

在`find`后加入`-d`（取依赖的英语单词dependencies的首字母）参数可以查看软件使用的依赖,例如查看vasp6.1.0使用的依赖




```
spack find -d vasp@6.1.0
```

3.2.1 更复杂的查询软件的方法

查找使用gcc10.2.0编译器编译的vasp:

```
spack find vasp %gcc@10.2.0
```

查找使用了intel-mkl数学库`依赖`的vasp:

```
spack find vasp ^intel-mkl
```

查找开启了vaspsol `特性`的vasp:

```
spack find vasp +vaspsol
```

查找满足以上三种条件的vasp:

```
spack find vasp %gcc@10.2.0 +vaspsol ^intel-mkl
```

3.3 软件环境变量

3.3.1 加载软件的环境变量

```
spack load <package>
```

有时候我们需要加载多个软件的环境变量，我们可以使用`spack load <package1> <package2> ...`

例如加载intel2019和gcc6.5.0

```
spack load intel-parallel-studio@cluster.2019.5 gcc@6.5.0
```

加载后`mpicc --version`的结果就不再是系统默认的4.8.5,而是6.5.0

多重条件加载指定软件：

加载使用了intel-mkl库的vasp5.4.4:

```
spack load vasp@5.4.4 ^intel5.4.4
```

加载使用了intel-mkl库的并由intel编译器编译的vasp5.4.4:

```
spack load vasp@5.4.4%intel ^intel5.4.4
```

3.3.2 使用hash值加载软件的环境变量

```
spack load /<hash>
```

有时候我们想加载某个软件，但它存在多个同名的软件时直接加载会报错

```
[jnist01@spack-test ~]$ spack load --sh vasp
==> Error: vasp matches multiple packages.
  Matching packages:
    itpysh5 vasp@5.4.4%intel@19.0.5.281 arch=linux-centos7-cascadelake
    7qgouzt vasp@6.1.0%intel@19.0.5.281 arch=linux-centos7-cascadelake
  Use a more specific spec.
```


例如加载vasp5.4.4:

```
spack load /itpysh5
```

我们可以加载其hash值, 上图灰色部分就是软件的md5的前7位

3.3.3 查看加载的环境变量

```
spack load --sh <package>
```

3.3.4 获取软件安装位置和可执行文件名字

```
spack location -i <package>
```

当然也可以使用hash值获取,

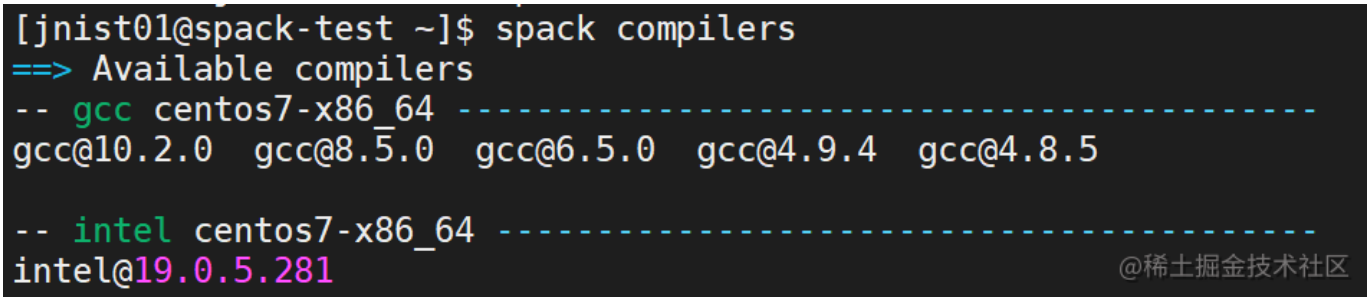
```
spack location -i /<hash>
```

获取之后，我们便可以`cd`到该目录，或直接`ls`该目录下的bin文件夹获取其可执行文件的名字

3.4 安装软件

```
spack install <package>
```

通常，我们需要指定编译软件所需要的编译器，使用`spack compilers`可以查看spack可以使用的编译器



使用`%`百分号作为编译器的界定符，它紧跟 软件或 软件和软件版本号 ， 之间可以有空格，也可以紧贴

3.4.1

例如我们需要使用gcc10.2.0编译`vasp`,则执行

```
spack install vasp@10.2.0
```

这将默认使用gcc10.2.0安装`spack info vasp`中`Preferred version`版本的vasp

3.4.2

我们需要vasp6.1.0版本，则执行

```
spack install vasp@6.1.0%gcc@10.2.0
```

3.4.2

为了在超算上达到更好的效率，我们更偏向于使用英特尔编译器和它的mpi、mkl数学库

```
spack install vasp@6.1.0%intel ^intel-mkl ^intel-mpi
```

使用`^`作为依赖的界定符，依赖有多个，可以指定多个依赖，否则使用默认的依赖，指定特殊的依赖需放在最后

3.4.3

vasp中有一个叫`vaspsol`的扩展包，如果想开启它

```
spack install vasp@6.1.0%intel +vaspsol ^intel-mkl ^intel-mpi
```

使用`+`开启它，特性位于编译后，空格隔开

有的时候，特性可能不是只有开启或关闭两种状态，而是提供多个可选项，例如下图这种

```
build_type [RelWithDebInfo]       [, ]    Debug, Release,          The build type to build
                                           RelWithDebInfo,
                                           MinSizeRel,
                                           Reference,
                                           RelWithAssert,
                                           Profile
```


构建类型`build_type`的值有多个，我们需要其中`Debug`、`Release`、`Profile`三个版本，可以这样描述

```
spack install <package>%gcc build_type=Debug,Release,Profile +opencl ^openmpi
```