

“探索 100”集群机 用户使用手册

清华信息科学与技术国家实验室(筹)

公共平台与技术部

2011-11-15

目录

| | |
|--|-----------|
| “探索 100” 集群机用户使用手册 | 1 |
| 1 硬件环境 | 4 |
| 1.1 登录节点 | 4 |
| 1.2 计算节点 | 4 |
| 1.3 存储节点 | 4 |
| 1.4 管理节点 | 4 |
| 1.5 网络互连 | 5 |
| 2 系统环境及磁盘共享 | 5 |
| 2.1 操作系统版本 | 5 |
| 2.2 磁盘共享 | 5 |
| 3 用户登录 | 5 |
| 3.1 远程访问软件 | 5 |
| 3.2 登录步骤 | 6 |
| 3.3 数据传输 | 6 |
| 3.4 使用集群 | 7 |
| 4 编译及测试环境 | 7 |
| 4.1 访问编译环境 | 7 |
| 4.2 软件资源: | 8 |
| 4.3 配置用户的环境变量: | 8 |
| 4.4 编译及测试 | 9 |
| 4.4.1 Intel 编译器编译串行程序及 Openmp 程序 | 9 |
| 4.4.2 Intel 编译器编译运行 mpi 并行程序 | 10 |
| 4.4.3 其他注意事项: | 13 |
| 5 作业提交 | 14 |
| 5.1 作业提交节点 | 14 |
| 5.2 用户目录说明 | 14 |
| 5.3 作业提交: | 14 |
| 6 Isf 使用说明 | 14 |
| 6.1 队列设定 | 15 |
| 6.2 提交作业(bsub) | 15 |
| 6.2.1 bsub 命令基本用法 | 15 |
| 6.2.2 OpenMP 并行作业提交 | 16 |
| 6.2.3 MPI 并行作业提交 | 16 |
| 6.2.4 大内存并行作业提交 | 16 |
| 6.2.5 使用脚本提交作业 | 17 |
| 6.3 状态查看 | 18 |
| 6.3.1 查看作业状态(bjobs) | 18 |
| 6.3.2 查看运行作业的标准(屏幕)输出(bpeek) | 19 |
| 6.3.3 查看作业历史运行情况(bhist) | 19 |
| 6.3.4 查看用户状态(busers) | 19 |
| 6.3.5 查看队列状态(bqueues) | 19 |
| 6.3.6 查询系统各主机状态(bhosts) | 19 |

| | | |
|--------|----------------------|----|
| 6.3.7 | 查询各主机系统状态 (lsload) | 20 |
| 6.4 | 控制作业执行 | 20 |
| 6.4.1 | 删除作业 (bkill) | 20 |
| 6.4.2 | 作业挂起 (bstop) | 20 |
| 6.4.3 | 作业恢复 (bresume) | 20 |
| 6.4.4 | 调整队列 (bwitch) | 20 |
| 6.4.5 | 改变作业排队次序 (btop/bbot) | 21 |
| 6.5 | 应用软件提交实例(待补充) | 21 |
| 6.5.1 | Matlab | 22 |
| 6.5.2 | Vasp | 22 |
| 6.5.3 | Siesta | 22 |
| 6.5.4 | Gaussian | 23 |
| 6.5.5 | Espresso | 23 |
| 6.5.6 | Nwchem | 23 |
| 6.5.7 | Abinit | 23 |
| 6.5.8 | Wien2K | 24 |
| 6.5.9 | NAMD | 24 |
| 6.5.10 | Gromacs | 24 |
| 6.5.11 | Fluent | 24 |
| 6.5.12 | Charmm | 24 |
| 6.5.13 | Rosseta | 24 |
| 6.5.14 | abaqus | 24 |
| 附录 1: | Linux 基本命令 | 24 |
| 附录 2: | Vi 使用 | 29 |

“探索 100” 集群机用户使用手册

1 硬件环境

“探索 100” 百万亿次集群系统由登录节点、740 个计算节点、24 个 I/O 存储节点及其他管理节点组成，节点间通过 InfiniBand 网络互连。集群系统理论峰值浮点计算性能达到 104TFlops，存储总容量 1000TB。

1.1 登录节点

“探索 100” 登录节点为 ln0。ln0 登录节点主要作用是实现用户登录、用户作业提交及集群系统作业的监控等。

1.2 计算节点

集群机计算节点共计 740 个：分 37 个刀片箱。编号形式为 c01bxx~c37bxx；每个刀片箱 20 个计算节点标号分别为 cxxb01~cxxb20。例如，第一个刀片箱第一个节点为 c01b01,第 37 个刀箱第 20 个节点为 c37b20。其中，c01b02~c01b03 为用户测试节点，用户可以直接登录进行程序开发和调试。其他节点需要通过 Isf 作业管理系统提交作业，加载程序。

单节点配置为：计算节点均采用两个 Intel Xeon X5670 六核处理器（2.93GHz，12MB Cache），160G SATA 硬盘。740 个节点中，360 个节点（c01bxx~c18bxx 及 c37b11~c37b19）配置 32GB 内存，370 个节点（c19bxx~c36bxx 及 c37b01~c37b10）配置 48GB 内存。

每个节点都是一个多核 SMP 服务器，计算节点用于运行串行和并行计算任务，支持 MPI、OpenMP 及 MPI/OpenMP 混行编程模式。“探索 100” 作业管理系统以 CPU 核作为并行作业的资源分配单位，实现并行作业的调度运行。“探索 100” 每个计算节点为 12 核的 SMP 服务器,可以最大支持 $740 * 12 = 8800$ 核并行作业的计算。

1.3 存储节点

IO 存储系统由 2 个管理节点 MDS0、MDS1 和 22 个 IO 存储节点 OSS1~OSS22 构成，提供集群系统的全局系统数据存储，可提供在线提供 160TB 存储容量。存储系统采用 LUSTRE 并行文件系统进行管理，实测写带宽 4GB/s。所有用户目录下/WORK 目录为全局共享，所有节点/WORK 目录都有读写权限。

1.4 管理节点

➤ 软件管理节点（appserver）：安装系统软件、编译器、并行库及各类应用软件。所有计算节点采用 NFS 方式共享软件管理节点上所有资源，共享目录为/apps。

➤ 用户目录管理节点 (homeserver): 管理用户自家目录下所有文件, 通过 NFS 方式共享 6T 存储, 提供稳定的存储访问。ln0、c01b02~c01b03 自家目录有读写权限, 其他计算节点自家目录用户有只读, 无写权限。

➤ 作业管理节点 (lsf0): 配置 lsf 作业管理系统, 根据实际资源情况及管理策略进行作业调度和分配。

1.5 网络互连

“探索 100”由 InfiniBand QDR 通信网络构成, 理论带宽 40Gb。所有节点间均可以通过 InfiniBand 网络实现高速通信。支持 MPI 并行任务间通信, 并实现全局文件系统的数据传输。

“探索 100”通过登录节点 ln0 接入校园网, 校内外用户通过以太网访问“探索 100”百万次集群系统。

2 系统环境及磁盘共享

2.1 操作系统版本

“探索 100”百万亿次集群系统所有节点均采用 RedHat Enterprise Linux 5.5 x86_64 版本, 遵循 POSIX, LSB 等标准, 提供了 64 位程序开发与运行环境。

2.2 磁盘共享

➤ 软件共享目录/apps: 通过 NFS 模式挂载软件管理节点 appserver 上所有软件资源。

➤ 用户目录~: 登录节点、所有计算节点通过 NFS 模式共享用户目录管理节点 6T 的存储空间。自家目录下所有文件在登录节点 ln0、计算节点 c01b02~c01b03 均有读写权限, 其他计算节点是只读权限。用户目录提供稳定的磁盘访问模式, 用户的软件、模型数据 (输入文件等) 建议存放在用户目录下。

➤ 工作目录./WORK: 每个用户自家目录下都有./WORK 目录。如用户 linjiao, 自家用户目录为/home/linjiao, /home/linjiao/WORK 即为用户 linjiao 的工作目录。登录节点、所有计算节点的工作目录通过 LUSTRE 并行文件系统共享 300T 存储空间, 用户计算主要在工作目录下进行。所有节点工作目录均具有读写权限。由于 LUSTRE 并行文件系统稳定性不强, 建议用户将重要计算结果随时备份到用户目录下。

3 用户登录

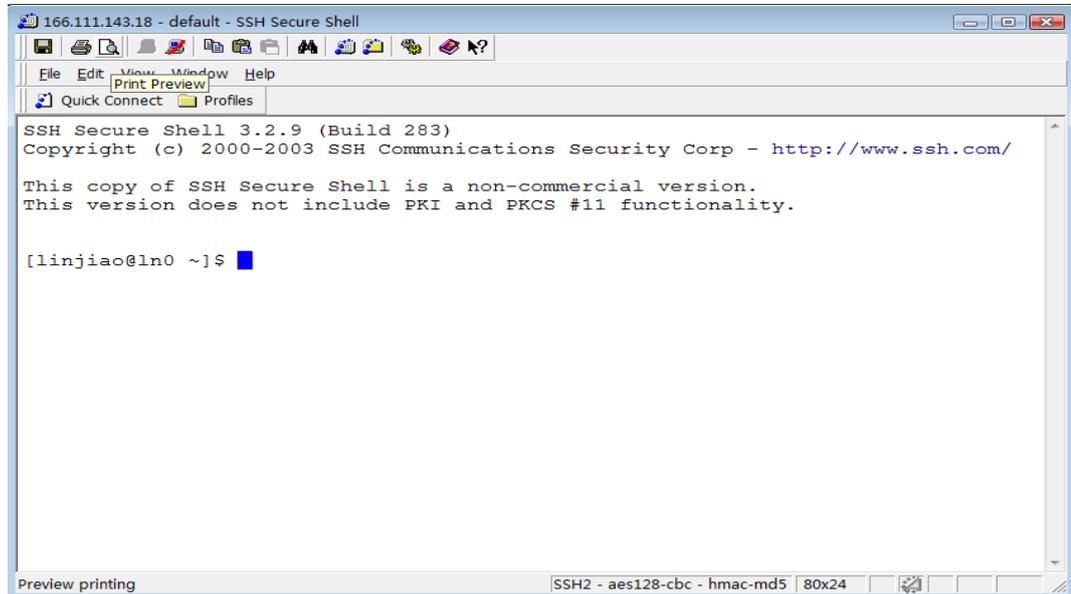
3.1 远程访问软件

用户需要使用支持 SSH 协议的相关软件访问系统, 我们推荐使用 SSH Secure Shell、

SecureCRT 等。

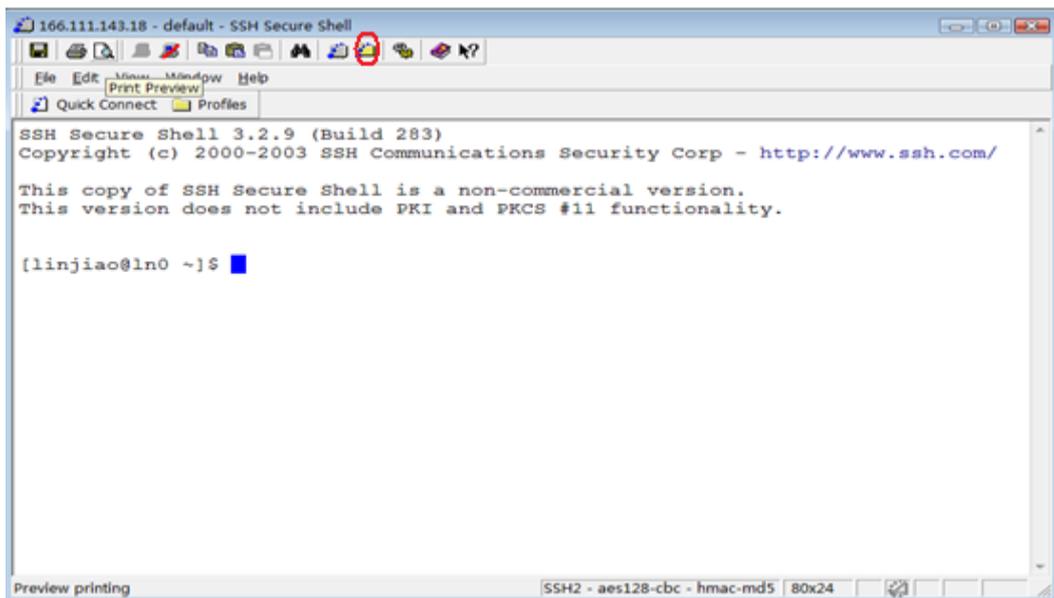
3.2 登录步骤

输入登录前端机 ln0，IP: 166.111.143.18，并键入申请的用户名密码。

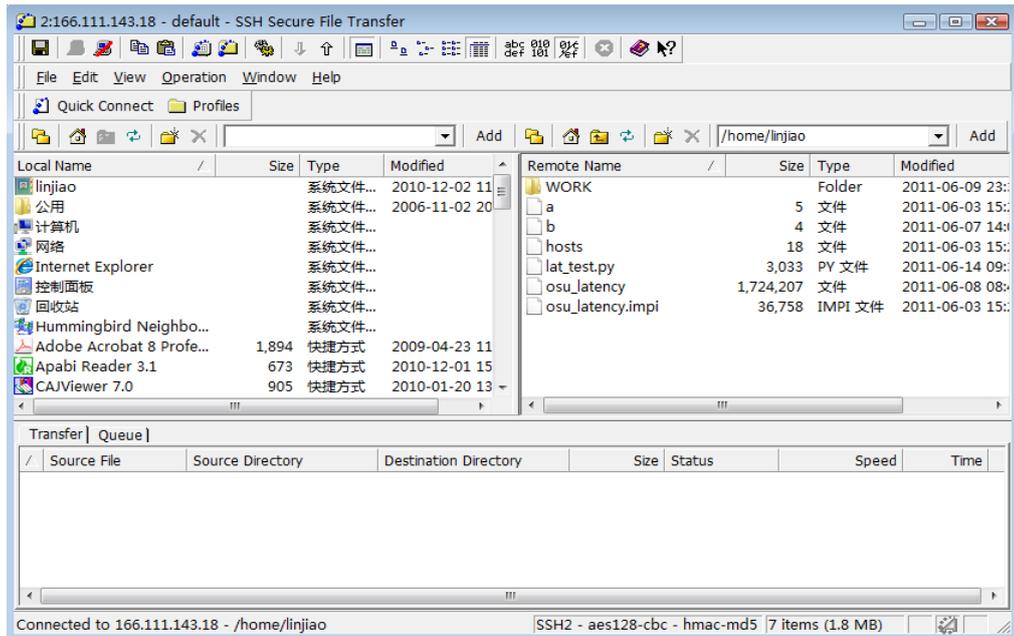


3.3 数据传输

单击 SSH Secure Shell 工具栏中的 File Transfer 键



得到如下窗口，将源程序及数据文件拷贝到登录节点上。



3.4 使用集群

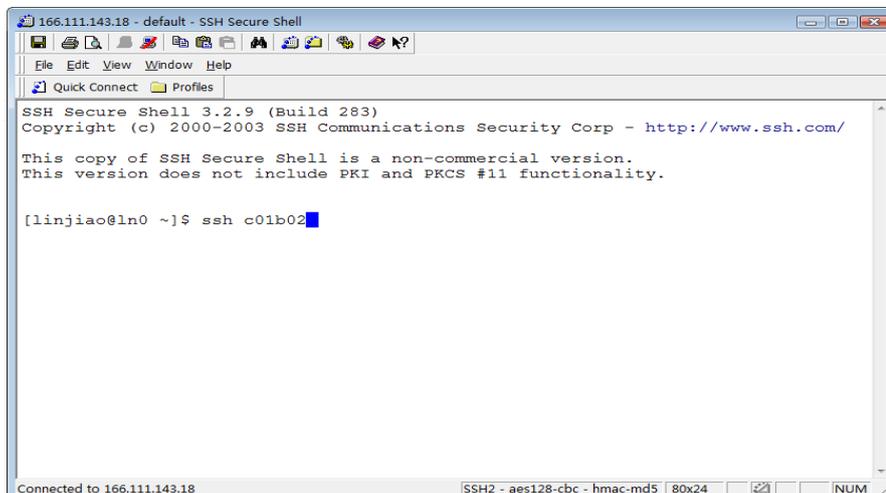
用户可以根据需求，进行程序调试（参见第 4 章编译及测试环境）或提交作业进行计算（参见第 5 章 Isf 使用方法）。

注意：初次使用集群系统的用户，必须在编译环境中调试软件。确保软件正常运行后，MPI 并行作业跨节点运行正常，再使用 Isf 提交作业，以免作业运行出错导致机器故障（死机或网络阻塞）！！

4 编译及测试环境

4.1 访问编译环境

集群计算节点 c01b02~c01b03 为用户测试节点。用户可以通过 ssh 命令，键入“ssh c01b02”，从 ln0 登录到测试节点编译、调试程序。



4.2 软件资源:

系统在软件服务器（appserver）预安装 intel 编译器，并基于 Intel 编译器安装各类 MPI 并行库、数学库及各类应用软件。所有节点共享软件服务器软件资源，共享目录为/apps。用户在 ln0、c01b02~c01b03 均可访问/apps 共享目录内容。表 1 为软件资源目录分布。系统为减少用户对软件服务器访问量，用户访问软件二级目录（/apps/intel）时才将目录挂载。即如果访问 intel 相关资源，运行命令（cd /apps/intel）;访问应用软件资源，运行命令（cd /apps/soft）;直接访问/apps（cd /apps），看不到软件资源。

集群系统采用 Intel X86_64 处理器，推荐用户优先使用 intel 编译器及 mkl 数学库进行软件安装优化，以提高程序执行效率。集群系统采用 infiniband 网络，系统安装了基于 infiniband 网络的 3 种 MPI 并行编程环境 impi-4.0.2、mvapich-2.1.7、openmpi-1.4.3，推荐用户使用以上 3 种 MPI 并行编程环境，获得高速网络通信。

| 目录 | 软件 |
|-------------|-----------------------------------|
| /apps/intel | Intel 软件（C/C++/fortran/mkl/mpi 等） |
| /apps/mpi | MPI 并行库 |
| /apps/lib | 数学库 |
| /apps/soft | 应用软件 |

表 1 软件资源目录分布

4.3 配置用户的环境变量:

用户在编译及运行程序之前，必须在用户自家的~/.bashrc 文件中配置环境变量指定使用的编译器、MPI 编译环境、数学库等相关路径，对 PATH 和 LD_LIBRARY_PATH 进行正确设置。手册提供了各类编译环境下的环境变量设置方法，请用户恰当选择编译环境，并将对应命令行添加在~/.bashrc 文件中，完成环境设置。

➤ Intel 编译器串行、OpenMP 并行程序环境设定:

```
source /apps/intel/Compiler/11.1/069/c/bin/iccvars.sh intel64
source /apps/intel/Compiler/11.1/069/f/bin/ifortvars.sh intel64
```

➤ Intel 编译器及 Intelmpi 环境设定:

```
source /apps/intel/Compiler/11.1/069/c/bin/iccvars.sh intel64
source /apps/intel/Compiler/11.1/069/f/bin/ifortvars.sh intel64
```

```
source /apps/intel/impi/4.0.2.003/bin64/mpivars.sh
```

➤ **Intel 编译器及 mvapich 环境设定:**

```
source /apps/intel/Compiler/11.1/069/c/bin/iccvars.sh intel64
```

```
source /apps/intel/Compiler/11.1/069/f/bin/ifortvars.sh intel64
```

```
export PATH=/apps/mpi/mvapich-2.1.7a-intel11.1/bin:$PATH
```

```
export LD_LIBRARY_PATH=/apps/mpi/mvapich-2.1.7a-intel11.1/lib:$LD_LIBRARY_PATH
```

➤ **Intel 编译器及 openmpi 环境设定:**

```
source /apps/intel/Compiler/11.1/069/c/bin/iccvars.sh intel64
```

```
source /apps/intel/Compiler/11.1/069/f/bin/ifortvars.sh intel64
```

```
export LD_LIBRARY_PATH=/apps/mpi/openmpi-1.4.3-intel11.1/lib:$LD_LIBRARY_PATH
```

```
export PATH=/apps/mpi/openmpi-1.4.3-intel11.1/bin:$PATH
```

```
export LD_LIBRARY_PATH= /apps/intel/Compiler/11.1/069/f/lib/intel64:$LD_LIBRARY_PATH
```

➤ **MKL 数学库环境设定:**

```
source /apps/intel/Compiler/11.1/069/c/mkl/tools/environment/mklvarsem64t.sh
```

➤ **其他环境设置模板:**

```
export LD_LIBRARY_PATH=库路径:$LD_LIBRARY_PATH
```

```
export PATH=可执行文件路径:$PATH
```

4.4 编译及测试

4.4.1 Intel 编译器编译串行程序及 Openmp 程序

➤ **icc:编译 C 程序:**

```
编译: icc -o prog prog.c
```

```
运行: ./prog
```

➤ **icpc:编译 C++程序:**

```
编译: icpc -o prog prog.cpp
```

```
运行: ./prog
```

➤ **ifort:编译 fortran 程序**

```
编译: ifort -o prog prog.f90
```

```
编译: ifort -o prog prog.for
```

```
运行: ./prog
```

➤ 编译 Openmp 程序

编译: `icc -o prog-omp -openmp prog-omp.c`

编译: `ifort -o prog-omp -openmp prog-omp.f90`

运行: `export OMP_NUM_THREADS=启动线程数` (启动线程数 ≤ 12)

`./ prog-omp`

常用编译选项:

(1) 优化选项 :

-O0: 禁止优化

-O1: 优化代码大小和代码局部性。

-O2 (缺省值): **优化代码速度 (推荐使用)**

-O3: -O2+激进的优化 (循环、存储访问转换、预取)。**需要注意的是, -O3 并不一**

定适合所有程序。

-fast: 打开-O3、-ipo、-static、-no-prec-div 和 -xP

-ipo: 过程间优化

(2) 输出和调试选项

-c: 只生成目标文件

-S: 只生成汇编文件

-g: 调试选项

-o <file>: 指定生成的输出文件名

(3) 浮点选项

-mp: 维持浮点精度 (禁止某些优化)

-mp1: 改善浮点精度。和-mp相比, -mp1 对性能影响较小

(4) 链接选项

-L<dir>: 指定链接时搜索的库路径

-l<string>: 链接特定库

-static: 静态链接

-shared: 生成共享库

4.4.2 Intel 编译器编译运行 mpi 并行程序

测试节点 c01b02~c01b03, 每个节点配置双 CPU6 核处理器 (即单节点 12 核), 最多可运行 24 核的例程。

系统基于 intel 编译器安装了多种 mpi, 安装目录在/apps/intel/impi 及/apps/mpi 下。impi openmpi mvapich 支持 infiniband 网络, 可获得较快的计算速度.

并行程序编译运行之前, 请参看 4.3 部分, 核对环境变量的设置, 确认无误后再进行 mpi 程序的编译。

➤ **intel mpi 的使用**

程序安装路径:

/apps/intel/impi/

impi 程序编译:

使用 mpiicc、mpiicpc、mpiifort 来编译 c、c++、fortran 程序, 底层调用的是 intel 编译器的 icc、icpc、ifort 进行编译。

编译方法如下:

```
mpiicc -o prog-mpi prog.c
```

```
mpiicpc -o prog-mpi prog.cpp
```

```
mpiifort -o prog-mpi prog.for
```

```
mpiifort -o prog-mpi prog.f90
```

impi 也提供 mpicc 和 mpif90 内部命令,其底层调用的 gcc 和 gfortran 编译程序。用户在编译及安装软件时请注意这一点。

impi 程序运行:

intel mpi 与其他 mpi 编程工具不同, 运行之前需要启动 MPD 守护进程, 再通过 -machinefile 文件指定进程分布。

1) 指定运行作业的节点。建立 hosts 文件, 内容为:

```
c01b02:12
```

```
c01b03:12
```

其中 c01b02 为运行节点, 12 为在 c01b02 运行的进程数。

2) 启动 impi 所需要的后台进程,

```
使用 mpdboot -n 2 -r ssh -f hosts
```

-n 2: 为启动两个节点

-r ssh:使用 ssh 协议

-f hosts:使用 hosts 文件作为进程启动说明文件。

3) 查看 mpd 进程是否启动

运行命令 `mpdtrace`，可以看到

`c01b02`

`c01b03`

说明启动成功

4) 运行 `mpi`

`mpiexec -machinefile hosts -n 24 ./prog-mpi`

`-n`:启动进程数

5) 关闭后台进程

程序结束后，运行 `mpdallexit`

➤ **mvapich 的使用**

程序安装路径:

`/apps/mpi/mvapich-2.1.7a-intel11.1`

mvapich 程序编译:

mvapich 使用 `mpicc`、`mpicxx`、`mpif77`、`mpif90` 来编译程序 `c`、`c++`、`f77`、`f90`，底层均调用的是 `intel` 编译器。

`mpicc -o prog-mpi prog.c`

`mpicxx -o prog-mpi prog.cpp`

`mpif77 -o prog-mpi prog.for`

`mpif90 -o prog-mpi prog.f90`

mvapich 的运行:

使用 `mpiexec` 或 `mpirun` 直接运行命令，无需启动 `mpd` 后台进程。

1) 指定运行作业的节点。建立 `hosts` 文件，内容为:

`c01b02:12`

`c01b03:12`

2) 运行 `mpi`

`mpiexec -machinefile hosts -n 24 ./prog-mpi`

`-n`:启动进程数

➤ **openmpi 的使用:**

程序安装路径:

`/apps/mpi/openmpi-1.4.3-intel11.1/`

openmpi 的编译:

openmpi 使用 mpicc、mpicxx、mpif77、mpif90 来编译程序 c、c++、f77、f90，底层均调用的是 intel 编译器。

```
mpicc -o prog-mpi prog.c
```

```
mpicxx -o prog-mpi prog.cpp
```

```
mpif77 -o prog-mpi prog.for
```

```
mpif90 -o prog-mpi prog.f90
```

openmpi 的运行:

使用 mpiexec 或 mpirun 直接运行命令，无需启动 mpd 后台进程。

- 1) 指定运行作业的节点。建立 hosts 文件，内容为:

```
c01b02:12
```

```
c01b03:12
```

- 2) 运行 mpi

```
mpiexec -machinefile hosts -n 24 ./prog-mpi
```

-n:启动进程数

常见问题:

- 1) warning: feupdateenv is not implemented and will always fail

解决: mpicc -o cpi -limf cpi.c

- 2) orted: error while loading shared libraries: libimf.so

解决: 各类库冲突,或者没有查找到。检查 intel 编译器及 openmpi 环境变量是否设置正确。在 LD_LIBRARY_PATH 中添加/apps/intel/Compiler/11.1/069/f/lib/intel64

4.4.3 其他注意事项:

- a) 测试节点 c01b02-c01b03 上用户目录被直接 mount 到系统存储中上,因此用户在目录下做任何文件操作,ln0、c01b02、c01b03 及其他计算节点都会有相应的改变。
- b) 程序运行以后想杀掉程序,直接按 ctrl+c,就可以杀掉一个 mpirun 启动的所有进程。
- c) 平台推荐用户使用 intel、及基于 Intel 编译器的 mpi 并行编程环境。用户如果需要其他环境配置,可直接和管理员联系,管理员将根据需求安装 gcc、pgi 编译器及基于相关编译器的 mpi 编程环境。
- d) c01b02-c01b03 为测试节点,仅供用户编译调试程序,为了所有用户使用方便,请大家不要长时间运行作业。管理员一旦发现,有权立即终止程序。

- e) 每个用户的自家目录都限制了磁盘限额，**请不要上传和计算无关文件，并及时做好数据备份和清理工作。**
- f) **系统/tmp 目录为内存虚拟目录，大小只为 100M。如果程序需要有临时文件写入，可将临时文件目录指定为：/scratch。**

5 作业提交

5.1 作业提交节点

ln0、c01b02-c01b03 均可提交作业。

5.2 用户目录说明

- 1) ln0、c01b02-c01b03 对自家目录~/有写权限，各计算节点均无写权限。用户只能通过 ln0、c01b02、c01b03 修改~/下的数据。
- 2) 每个用户自家目录下都有./WORK/目录，/WORK 目录挂载 lustre 并行文件系统。所有节点对./WORK/目录都有写权限。
- 3) 数据存放及使用：为保证计算模型数据安全，程序输入文件（计算模型）放在~/下；输出文件，放在 WORK/目录中；计算结束以后，及时将获得有用的数据结果保存到~/,防止并行文件系统不稳定，造成的数据丢失。

注意：如果用户程序必须把将输入输出文件放置在同级目录，则在 WORK/下对建立所有输入文件的链接或直接将输入文件拷贝到 WORK 目录下后，再开始运算。

如：input.dat 放在~/目录下

方法 1：:建立输入文件的连接

```
cd /WORK;ln -s ~/input.dat input.dat (单个文件建立连接)
```

```
cd /WORK;ln -s ~/input/* ./input (目录下所有文件建立连接)
```

方法 2：将输入文件拷贝到/WORK 下

```
cd /WORK;cp ~/input.dat input.dat
```

5.3 作业提交：

用户必须使用 Isf 作业管理软件提交作业，才能使用计算节点，Isf 使用方法见下第 6 部分。

6 Isf 使用说明

“探索 100”使用 Isf 作业管理系统进行作业的管理与分配。用户只需用 Isf 提交命令(bsub)

将作业提交到集群，系统就会按照管理员制定的作业分配策略自动进行调度，决定何时以及在哪些计算结点运行程序。作业管理系统不仅方便用户使用，更提高了整个系统使用效率。

6.1 队列设定

目前系统中建立了四个队列,可使用 `bqueues` 命令查看:

1. `normal` 队列: 可提交任意核数的作业。如果用户使用核数不是 12 倍数, `lsf` 自动将转移到 `normal` 队列中
2. `hpc_linux` 队列: 只允许提交以 12 的整数倍的作业。**作业核数为 12 倍数的作业, 请优先提交到该队列, 以保证独占计算节点, 提高计算效率。** 如果用户使用核数不是 12 倍数, 仍提交到该队列中, `lsf` 自动将其转移到 `normal` 队列中。
3. `priority` 队列: 优先队列。用户如有遇紧急需求, 可向管理员申请短时间使用 `priority` 队列提交作业, 以获得较高优先级, 抢先获得计算资源。
4. `short` 队列: 对于运行时间在 15 分钟以下作业, 用户可提交到 `short` 队列中, `short` 队列具有较高优先级。

6.2 提交作业(bsub)

6.2.1 bsub 命令基本用法

1. 提交作业:`bsub command`

```
$ bsub sleep 60
```

```
Job <3616> is submitted to default queue <normal>.
```

向 LSF 提交作业, 获得唯一 ID3616,作业提交成功。

2. 向某个队列提交作业: `bsub -q`。

```
$ bsub -q short sleep 60
```

```
Job <3628> is submitted to queue <short>.
```

3. 用 `-o`-`e` 制定标准输出和 `error` 文件位置

```
$ bsub -o output.%J -e errors.%J ls-l
```

```
Job <3640> is submitted to queue < normal >.
```

`%J` 代表作业 ID

注意: 用户的可执行程序必须写在 `-o -e` 选项后面

4. 用 `-i` 指定输入文件

有些可执行程序运行时采用 `<` 方式来输入可执行文件

如运行:mpirun -np 24 /apps/soft/siesta/siesta-3.0-rc1/Obj/siesta< SLG.fdf

lsf 可使用-i 指定输入文件, 命令如下:

```
bsub -a intelmpi -n 24 -i /home/zjy/WORK/SIESTA/bravais_graphene/SLG.fdf mpirun.lsf  
/apps/soft/siesta/siesta-3.0-rc1/Obj/siesta
```

5. 用-m 指定运行机器

```
$ bsub -m "hosta hostb" hostname
```

bsub 详细用法可以使用 man bsub, 参考说明

6.2.2 OpenMP 并行作业提交

使用 openmp 关键字

例 1: 提交作业 12 核 openmp, 并保证作业独占该计算节点。

```
bsub -a openmp -n 12 -R "span[hosts=1]" myOpenMPJob
```

例 2: 作业使用 32 核, 每个节点使用 4 核的 MPI 与 Openmp 混合程序

```
bsub -a openmp -n 32 -R "span[ptile=4]" myOpenMPJob
```

6.2.3 MPI 并行作业提交

用 mpirun.lsf 关键字提交作业, 并使用-a 选项指定所选用的 mpi。不同 mpi 要使用不同的关键字。

例 1: 提交 intelmpi 并行作业

```
bsub -a intelmpi -o output.%J -e error.%J -n 12 mpirun.lsf /examples/cpi
```

例 2: 提交 mvapich 并行作业

```
bsub -a mvapich -o output.%J -e error.%J -n 12 mpirun.lsf /examples/cpi
```

例 3: 提交 openmpi 并行作业

```
bsub -a openmpi -o output.%J -e error.%J -n 12 mpirun.lsf /examples/cpi
```

6.2.4 大内存并行作业提交

系统计算节点内存有两种配置, 48G 和 32G。因此, 当用户预计单计算节点内存使用量超过 26G (单进程内存占用量超过 2.2G), 用户必须使用大内存并行提交方式提交作业, 防治因内存不足造成计算缓慢或系统死机等问题。需要大内存的用户在提交作业时必须做好两件工作: 1) 使用-R 选项将作业提交到大内存节点(内存 48G)上。2) 使用-M 选项, 限制单进程内存使用量。否则, 作业内存使用量过大, 操作系统采用虚拟内存 swp 方式将大大影响计算速度, 并有可能造成计算节点死机。

1. 使用-R 选项大内存计算节点的选择

如：将作业提交到内存剩余总量超过 42G 的计算节点上

```
bsub -a intelmpi -R "select [mem>42000]" -n 12 mpirun.lsf /examples/large_mem
```

其中，单位为 MB

2. 使用-M 选项，限制单进程内存使用量

如：限制作业单进程内存使用量不要超过 3.5G

```
bsub -a intelmpi -M 3500000 -n 12 mpirun.lsf /examples/large_mem
```

其中,单位是 KB。作业单进程内存如果超过 3.5G,作业将被删除。用户需要通过增加计算节点的方式解决内存占用的问题。

大内存计算节点内存容量为 48G，除去操作系统及其他内存占用，**推荐用户单节点内存占用量不要超过 42G**。在各进程负载平衡的情况下，**推荐单进程内存使用量不要超过 42G/12=3.5G**。

因此，用户提交并行大内存作业的提交模板为：

```
bsub -a MPITYPE -R "select [mem>42000]" -M 3500000 -n Z mpirun.lsf ./large_mem
```

请大家务必做好限制，否则将很可能造成节点死机！！

6.2.5 使用脚本提交作业

为使用方便，用户可以自行撰写脚本提交作业，每次直接运行脚本即可。

撰写脚本有两种方式：

方法 1： 建立包含 bsub 的脚本

创建文件（如 job），在 job 中写入 bsub 提交命令，如：

```
bsub -a intelmpi -o output.%J -e error.%J -n 12 mpirun.lsf /examples/cpi
```

然后 chmod +x job

直接运行 ./job，就可以提交作业。

方法 2： 使用 bsub 脚本多次提交具有相同参数的作业，其格式如下：

```
#!/bin/sh
#BSUB -q QUEUENAME
#BSUB -a MPITYPE
#BSUB -n Z
#BSUB -o OUTPUTFILE
#BSUB -e ERRFILE
mpirun.lsf program
```

提交脚本,运行命令 bsub <脚本名，即可提交作业。

该脚本等同于命令：

```
bsub -q QUEUENAME -a MPITYPE -n Z -o OUTPUTFILE -e ERRFILE mpirun.lsf program
```

推荐用户使用方法 2 “**bsub 脚本模式**” 提交作业。

提交作业如果需要其他选项，如**-J**、**-R**、**-M**、**-W**、等请按照以上格式自己添加。

例如：提交 openmp 与 MPI 混合

1. 创建文件 job,内容如下:

```
#BSUB -q normal  
  
#BSUB -a intelmpi  
  
#BSUB -n 24  
  
#BSUB -R "span[ptile=12]"  
  
#BSUB -o output.%J  
  
#BSUB -e error.%J  
  
export OMP_NUM_THREADS=12  
  
mpirun.lsf ./mpi_openmp_hello
```

2. 用 bsub 提交作业:

```
bsub < job
```

6.3 状态查看

6.3.1 查看作业状态(bjobs)

作业提交后，用户使用 `bjob` 命令查看作业 ID 和状态

```
$ bjobs
```

```
JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME  
1266 user1 RUN normal hosta hostb sleep 60 Feb 5 17:39:58
```

一个作业提交后，将可能为以下几种状态之一：

| STAT | 状态 |
|-------|---------------------------------|
| PEND | 任务在队列中排队等待 |
| RUN | 任务正在执行 |
| PSUSP | 任务在队列中排队等待时被用户挂起 |
| SSUSP | 任务被系统挂起 |
| DONE | 作业正常结束， <code>exit</code> 代码为 0 |
| EXIT | 作业退出， <code>exit</code> 代码不为- |

常用选项：

-a: 除了可以查看已提交及尚未结束的作业，还可以看到刚结束不久的作业信息

-u: 查看系统其它用户作业情况，如：

查看 user1 的作业： `bjobs -u users1`

查看所有人的作业： `bjobs -u all`

-l: 查看某个作业详细信息

查看作业 JOBID 详细信息： `bjobs -l JOBID`

6.3.2 查看运行作业的标准（屏幕）输出（`bpeek`）

```
$ bpeek 1508
```

6.3.3 查看作业历史运行情况(`bhist`)

```
$ bhist -l 1508
```

6.3.4 查看用户状态（`busers`）

```
$ busers
```

| USER/GROUP | JL/P | MAX | NJOBS | PEND | RUN | SSUSP | USUSP | RSV |
|------------|------|-----|-------|------|-----|-------|-------|-----|
| linjiao | - | 96 | 0 | 0 | 0 | 0 | 0 | 0 |

Max: 用户可用核数上限

NJOBS: 已提交作业所需要全部核数

PEND: 在队列中等待执行的所有作业的核数

RUN: 正在运行作业的核数

SSUSP: 系统挂起用户作业核数

USUSP: 用户自行挂起作业的核数

RSV: 系统预约保留的核数

6.3.5 查看队列状态（`bqueues`）

```
$ bqueues
```

| QUEUE_NAME | PRIO | STATUS | MAX | JL/U | JL/P | JL/H | NJOBS | PEND | RUN | SUSP |
|------------|------|-------------|-----|------|------|------|-------|------|-----|------|
| priority | 45 | Open:Active | - | - | - | - | 0 | 0 | 0 | 0 |
| short | 40 | Open:Active | - | - | - | - | 0 | 0 | 0 | 0 |
| test | 35 | Open:Active | - | - | - | - | 0 | 0 | 0 | 0 |
| normal | 30 | Open:Active | - | - | - | - | 969 | 220 | 749 | 0 |
| hpc_linux | 30 | Open:Active | - | - | - | - | 0 | 0 | 0 | 0 |
| zjn | 30 | Open:Active | - | - | - | - | 0 | 0 | 0 | 0 |

`bqueues -l`: 查询某个队列的详细信息

```
$ bqueues -l normal
```

6.3.6 查询系统各主机状态（`bhosts`）

```
$ bhosts
```

| HOST_NAME | STATUS | JL/U | MAX | NJOBS | RUN | SSUSP | USUSP | RSV |
|-----------|--------|------|-----|-------|-----|-------|-------|-----|
|-----------|--------|------|-----|-------|-----|-------|-------|-----|

| | | | | | | | | |
|-------|--------|---|---|----|----|---|---|---|
| hosta | ok | - | - | 0 | 0 | 0 | 0 | 0 |
| hostb | ok | - | - | 0 | 0 | 0 | 0 | 0 |
| hostc | ok | - | - | 0 | 0 | 0 | 0 | 0 |
| hostd | closed | - | - | 12 | 12 | 0 | 0 | 0 |

OK: 该节点可以接收用户作业

Closed: 已经有作业运行或负载过高。

6.3.7 查询各主机系统状态 (lsload)

```
$ lsload
```

| HOST_NAME | status | r15s | r1m | r15m | ut | pg | ls | it | tmp | swp | mem |
|-----------|--------|------|-----|------|-----|-----|----|-----|-----|------|------|
| hostn | ok | 0.0 | 0.0 | 0.1 | 1% | 0.0 | 1 | 224 | 43M | 67M | 3M |
| hostk | -ok | 0.0 | 0.0 | 0.0 | 3% | 0.0 | 3 | 0 | 38M | 40M | 7M |
| hostg | busy | *6.2 | 6.9 | 9.5 | 85% | 1.1 | 30 | 0 | 5M | 400M | 385M |
| hostf | busy | 0.1 | 0.1 | 0.3 | 7% | *17 | 6 | 0 | 9M | 23M | 28M |

6.4 控制作业执行

6.4.1 删除作业 (bkill)

用 bkill 停止作业运行。

```
$ bkill 1266
```

```
Job <1266> is being terminated
```

使用 bkill 删除并行作业时，lsf 需要收集信息、发送信号等处理，用户执行 bkill 命令后，作业可能没有立即删除，使用 bjobs 命令还可以看到作业。**请用户耐心等待（大约 1 分钟），lsf 将完整作业删除工作。**

6.4.2 作业挂起 (bstop)

用 bstop 挂起正在运行的作业，需要指明作业 ID:

```
$ bstop 1266
```

```
Job <1266> is being stopped
```

```
$ bjobs
```

| JOBID | USER | STAT | QUEUE | FROM_HOST | EXEC_HOST | JOB_NAME | SUBMIT_TIME |
|-------|-------|-------|--------|-----------|-----------|----------|----------------|
| 1266 | user1 | USUSP | normal | hosta | hostb | sleep 60 | Feb 5 17:39:58 |

挂起之后，STAT 为 USUSP。

6.4.3 作业恢复 (bresume)

用 bresume 恢复作业运行

```
$ bresume 1266
```

```
Job <1266> is being resumed
```

```
$ bjobs
```

| JOBID | USER | STAT | QUEUE | FROM_HOST | EXEC_HOST | JOB_NAME | SUBMIT_TIME |
|-------|-------|------|--------|-----------|-----------|----------|----------------|
| 1266 | user1 | RUN | normal | hosta | hostb | sleep 60 | Feb 5 17:39:58 |

6.4.4 调整队列 (bwitch)

用 bswitch 将正在运行的作业调度到其他队列中

\$bswitch priority 5309

Job <5309> is switched to queue <priority>

6.4.5 改变作业排队次序 (btop/bbot)

用户可以使用 btop/bbot 改变本用户提交且处于“PEND”状态的作业调度次序。

btop: 指定队列中，所有同优先级作业最先获得调度。

bbot: 指定队列中，所有同优先级作业最后获得调度。

bjobs

| JOBID | USER | STAT | QUEUE | FROM_HOST | EXEC_HOST | JOB_NAME | SUBMIT_TIME |
|-------|-------|------|--------|-----------|-----------|----------|--------------|
| 5308 | user2 | RUN | normal | hostA | hostD | /s500 | Oct 23 10:16 |
| 5309 | user2 | PEND | night | hostA | | /s200 | Oct 23 11:04 |
| 5311 | user2 | PEND | night | hostA | | /s700 | Oct 23 18:17 |

btop 5311

Job <5311> has been moved to position 1 from top.

bjobs

| JOBID | USER | STAT | QUEUE | FROM_HOST | EXEC_HOST | JOB_NAME | SUBMIT_TIME |
|-------|-------|------|--------|-----------|-----------|----------|--------------|
| 5308 | user2 | RUN | normal | hostA | hostD | /s500 | Oct 23 10:16 |
| 5311 | user2 | PEND | night | hostA | | /s200 | Oct 23 18:17 |
| 5309 | user2 | PEND | night | hostA | | /s700 | Oct 23 11:04 |

6.5 应用软件提交实例(待补充)

| 软件名称 | 并行编译器 | 数学库 | 软件安装路径 |
|------------|----------|----------------|--|
| matlab | | | /apps/soft/MATLAB/R2011a/bin/matlab |
| vasp4.6 | intelmpi | MKL | /apps/soft/vasp/vasp.4.6/vasp.4.6/vasp |
| Vasp5.2 | intelmpi | MKL | /apps/soft/vasp/vasp.5.2/vasp.5.2/vasp |
| Siesta3.0 | intelmpi | MKL | /apps/soft/siesta/siesta-3.0-rc1/Obj/siesta |
| Gaussian09 | | | /apps/soft/gaussian/g09/g09 |
| espresso | intelmpi | MKL | /apps/soft/espresso/espresso-4.3/bin |
| nwchem | intelmpi | MKL | /apps/soft/nwchem/nwchem-6.0/bin/LINUX64/nwchem |
| abinit | mvapich | MKL | /apps/soft/abinit/abinit-6.8.2-mvapich/bin/abinit |
| NAMD | openmpi | fftw-2 .1.5 | /apps/soft/NAMD/NAMD_2.8_Source/Linux-x86_64-icc/namd2 |
| lammps | intelmpi | MKL | /apps/soft/lammps/lammps-23May11/src/lmp_mkl |

| | | | |
|-----------|----------|-----|--|
| autodock | gcc | | /apps/soft/Autodock/src/autodock/autodock4 |
| WIEN2K_11 | intelmpi | MKL | /apps/soft/WIEN2K/WIEN2K_11 |

6.5.1 Matlab

提交脚本:

```
#BSUB-q hpc_linux
#BSUB-n 12
#BSUB -a openmp
#BSUB-R "span[hosts=1]"
#BSUB -o output.%J
#BSUB -e error.%J
/apps/soft/MATLAB/R2011a/bin/matlab -r "eigenenergynyx(${LSB_JOBINDEX});quit"
```

6.5.2 Vasp

提交脚本:

```
#!/bin/sh
#BSUB -q hpc_linux
#BSUB -n 12
#BSUB -o output.%J
#BSUB -e error.%J
#BSUB -a intelmpi
mpirun.lsf -np 12 /apps/soft/vasp/vasp.4.6/vasp.4.6/vasp
```

6.5.3 Siesta

提交脚本:

```
#!/bin/sh
#BSUB -q hpc_linux
#BSUB -n 12
#BSUB -o output.%J
#BSUB -e error.%J
#BSUB -a intelmpi
```

```
mpirun.lsf -np 12 /apps/soft/siesta/siesta-3.0-rc1/Obj/siesta
```

6.5.4 Gaussian

6.5.5 Espresso

```
#!/bin/sh

#BSUB -q normal

#BSUB -n 4

#BSUB -o %J.output -e %J.err

#BSUB -W 360:00

#BSUB -a intelmpi

mpirun.lsf -np 4 /home/xuzp/bin/espresso/pw.x -npool 1 < relax.in >& relax_log
```

6.5.6 Nwchem

```
#BSUB -a intelmpi

#BSUB -R "span[ptile=12]"

#BSUB -n 12      # request number of processors.

#BSUB -o output.%J

#BSUB -e error.%J

#BSUB -q hpc_linux

export NWCHEM_TOP=/home/lijun/nwchem-6.0

export PATH=$NWCHEM_TOP/bin/LINUX64:$PATH

export PYTHONPATH=$PYTHONPATH:$NWCHEM_TOP/contrib/python/

mpirun.lsf      $NWCHEM_TOP/bin/LINUX64/nwchem      $LSB_JOBNAME.nw      >&

$LSB_JOBNAME.nwout
```

6.5.7 Abinit

提交脚本:

```
#!/bin/sh

#BSUB -q hpc_linux

#BSUB -n 12

#BSUB -o output.%J

#BSUB -e error.%J

#BSUB -a mvapich
```

```
mpirun.lsf /apps/soft/abinit/abinit-6.8.2-mvapich/bin/abinit< kpt.files
```

6.5.8 Wien2K

6.5.9 NAMD

提交脚本:

```
#!/bin/sh

#BSUB -q hpc_linux

#BSUB -n 12

#BSUB -o output.%J

#BSUB -e error.%J

#BSUB -a openmpi

mpirun.lsf /apps/soft/NAMD/NAMD_2.8_Source/Linux-x86_64-icc/namd2 mineq2.conf
```

6.5.10 Gromacs

6.5.11 Fluent

6.5.12 Charmm

6.5.13 Rosseta

6.5.14 abaqus

附录 1: Linux 基本命令

1. 目录操作

名称 : cd

语法 : cd [directory]

说明 : 把当前工作目录转到" directory"指定的目录。

实例 : 进入目录 /usr/bin/:

```
cd /usr/bin
```

名称 : ls

语法 : ls [options] [pathname-list]

说明：显示目录内的文件名和“pathname-list”中指定的文件名

实例：列出目前工作目录下所有名称是 s 开头的文件：

```
ls s*
```

名称：pwd

语法：pwd

说明：显示当前目录的绝对路径。

名称：mkdir

语法：mkdir [options] dirName

说明：创建名称为 dirName 的子目录。

实例：在工作目录下，建立一个名为 AA的子目录：

```
mkdir AA
```

名称：rmdir

语法：rmdir [-p] dirName

说明：删除空的目录。

实例：将工作目录下，名为 AA的子目录删除：

```
rmdir AA
```

2. 文件操作

名称：cp

语法：cp [options] file1 file2

说明：复制文件file1到file2。

常用选项：-r 整个目录复制

实例：将文件 aaa 复制(已存在)，并命名为 bbb：

```
cp aaa bbb
```

名称：mv

语法：mv [options] source... directory

说明：重新命名文件，或将数个文件移至另一目录。

范例：将文件 aaa 更名为 bbb：

```
mv aaa bbb
```

名称：rm

语法：rm [options] name...

说明：删除文件及目录。

常用选项：-f 强制删除文件

实例：删除除后缀名为.c的文件

```
rm *.c
```

名称：cat

语法：cat[options] [file-list]

说明：在标准输出上连接、显示文件列表file-list里的文件

实例1：显示file1和file2的内容

```
cat file1 file2
```

实例2：将file1和file2合并成file3

```
cat file1 file2 > file3
```

名称：more

语法：more[options] [file-list]

说明：在标准输出上连接、分页显示文件列表file-list里的文件

实例：分页显示文件AAA

```
more AAA
```

名称：head

语法：head[options] [file-list]

说明：显示文件列表file-list中的文件的起始部分，默认显示10行；

实例：显示文件AAA起始部分

```
head AAA
```

名称：tail

语法：tail[options] [file-list]

说明：显示文件列表file-list中的文件的尾部；默认显示10行；

实例：显示文件AAA尾部

```
tail AAA
```

名称：ln

语法：ln[options] existing-file new-file

```
ln[options] existing-file-list directory
```

说明：为“existing-file”创建链接，命名为new-file

在directory目录，为existing-file-list”中包含的每个文件创建同名链接

常用选项：-f 不管new-file是否存在，都创建链接

-s 创建软链接

实例1：建立软连接temp.soft,指向Chapter3

```
ln -s Chapter3 temp.soft
```

实例2：为examples目录下的所有文件和子目录建立软连接

```
ln -s ~/linuxbook/examples/* /home/faculty/linuxbook/examples
```

名称：chmod

语法：chmod [option] mode file-list

说明：改变或设置参数file-list中的读、写或执行权限

实例：添加文件job的可执行权限

```
chmod +x job
```

名称： tar

语法： chmod [option] [files]

说明： 备份文件。可用来建立备份文件，或还原备份文件。

实例1： 备份test目录下的文件，并命名为test.tar.gz，可执行命令：

```
tar -zcvf test.tar.gz test
```

实例2： 解压缩相关的test.tar.gz文件，可执行命令：

```
tar -zxvf test.tar.gz
```

3. 其他

名称： echo

语法： echo \$variable

说明： 显示变量 variable 的值。

实例 1： 显示当前用户路径 PATH 的值

```
echo $PATH
```

名称： ps

语法： \$ps [options]

说明： 用于查看当前系统中的活跃进程

实例 1： 显示当前所有进程

```
ps -aux
```

名称： kill

语法： \$kill [-signal] pid

说明： 终止指定进程

实例 1：终止 1511 号进程

```
kill 1511
```

附录 2：Vi 使用

1. 简要使用流程

- 1) 使用 “vi [选项] [文件 ..]” 命令打开要编辑的文件
- 2) 使用 “方向键” 浏览文件
- 3) 按下 “i” 进入编辑模式
- 4) 编辑
- 5) 按 “Esc” 键退出编辑模式
- 6) 输入 “:w” 回车保存，再输入 “:q” 回车退出。或者直接输入 “:wq” 回车，代表保存并退出

2. 两种操作模式

- 编辑模式：对文本进行编辑处理
- 命令模式：接收按键指令执行操作，如复制、粘贴、搜索、替换、保存、另存为等

3. 编辑模式

- i: 进入编辑模式
- a: 进入编辑模式，将光标向后移动一位
- o: 进入编辑模式，在光标处插入一个空行
- r: 按下 r 键，再按任意字符键，将光标所在处的字符替换成新输入的字符
- Esc: 退出编辑模式

4. 命令模式

➤ 移动光标

- ↑或 k: 把当前光标向上移动一行，保持光标的列位置。
- ↓或 j: 把当前光标向下移动一行，保持光标的列位置。
- 或 l: 把当前光标向右移动一个字符。
- ←或 h: 把当前光标向左移动一个字符。
- \$: 把当前光标移动到该行行末。

^: 把当前光标移动到该行行首。
w: 把当前光标移动到该行的下一个字的首字符上。
b: 把当前光标移动到该行的上一个字的首字符上。
e: 把当前光标移动到该行的该字的末尾字符上。
^F: 向前滚动一整屏正文。
^D: 向下滚动半个屏正文。
^B: 向后滚动一整屏正文。
^U: 向上滚动半个屏正文。

➤ 搜索与替换

/word: 从光标处开始, 向后搜索文本中出现 word 的字符串
?word: 从光标处开始, 向前搜索文本中出现 word 的字符串
:1,\$s/word1/word2/g: 在第 1 行与最后一行之间搜索 word1, 并将其替换为 word2
:n1,n2s/word1/word2/g: 在第 n1 行与第 n2 行之间搜索 word1, 并将其替换为 word2

➤ 删除 (剪切)、复制与粘贴

x,X: x 为向后删除一个字符, X 为向前删除一个字符
dd: 删除光标所在行
yy: 复制光标所在行的内容
nyy: 复制光标到第 1 行的所有内容
y1G: 复制光标到第 1 行的所有内容
yG: 复制光标到最后一行的所有内容
p,P: p 为将复制或剪切的内容粘贴在光标下一行, P 为粘贴在上一行
u: 撤消上一操作

➤ 管理命令

:w: 保存
:w!: 强制保存
:q: 退出 vi 编辑器
:q!: 强制退出
:w [文件名]: 另存为..
:r[文件名]: 读取另一个文档的内容, 内容追加到光标所在行之后
:set nu: 显示正文的行号。

:set nonu:取消行号。

:[命令]: 暂时离开 vi 编辑器, 并在 shell 中执行命令