浙医大 HPC 集群
用户手册

Version 1.0

by 郑光奇

Lenovo Present

目 录

一、 远程登录工具 XShell 使用方法 1
1.1、XShell 基本介绍主要内容 1
1.2、基本知识 1
二、 LSF 作业调度的使用方法 9
2.1、lsf 作业脚本的编写 9
2.2、提交作业 10
2.3、查看集群中 lsf 可调用的计算资源 11
2.4、查看集群中 lsf 队列信息 13
2.5、查看 lsf 节点的启动状态 13
2.6、查看 lsf 作业状态 15
2.7、查看 lsf 作业的详细信息 16
三、 Linux 常用命令说明 17
四、 Linux 平台软件安装方法 27
4.1、rpm 安装方法 27
4.2、yum 安装方法 27
4.3、源码编译安装方法 29
4.4、python 包安装方法 30
4.5、R 模块安装方法 31
4.6、perl 模块安装 31
五、 环境变量配置 33
六、Lenovo HPC 平台集群常见问题说明 34
6.1、 普通用户无法使用 SSH 登录集群 34
6.2、 普通用户 SSH 登录集群密码错误 34
6.3、 普通用户 SSH 登录集群后，用户家目录下无文件 34
6.4、 普通用户 SSH 登录集群后，登录计算节点和提交作业需要密码 34

一、 远程登录工具 XShell 使用方法

1.1、XShell 基本介绍主要内容

- 最简单的使用 -- 登录 SSH 主机
- XShell 常用配置的说明
    - 复制、粘贴
    - 保存会话
    - 注销
- XShell 的 X11 转发
- 如何用 XShell 建立 SSH 隧道
- 常见问题

除了上面的这些，还夹杂了一些 XShell 使用上的技巧、服务器配置的一些安全建议。这是一些有关 XShell 的使用教程，其实也就是 SSH 的参考教程，绝大多数的内容在其他系统或软件上也都是一样的。不同的是参数、配置、命令行之类的，只要会了 XShell，其他也就触类旁通了。

1.2、基本知识

用户需要使用 Baidu 了解 SSH、Telnet、Rlogin 基本概念，熟悉后可以进行下一步的阅读。

i. 简介

XShell 的官方网站：http://www.netsarang.com/products/xsh_overview.html，XShell 是一个跨平台的远程登录工具，为了发挥其强大的功能，还有如下程序，包括：

- XMing (用来转发 X11 图形界面)
- XFtp (文件传输客户端，类似于 FTP 的，只不过使用的是 SSH 的 22 端口，而非 FTP 的 21 端口)

如非特别说明，默认的登录的协议是 SSH。

ii. XShell 安装

XShell 是一个准绿色软件，说它绿色是因为直接就能使用，完全没有任何的安装程序。准绿色是指 XShell 的所有配置都保存到了注册表，如果不记得备份注册表中的相关内容，下次重装机器所有配置就没了，而且配置也不方便用闪存盘随身携带。

iii. 登录远程主机

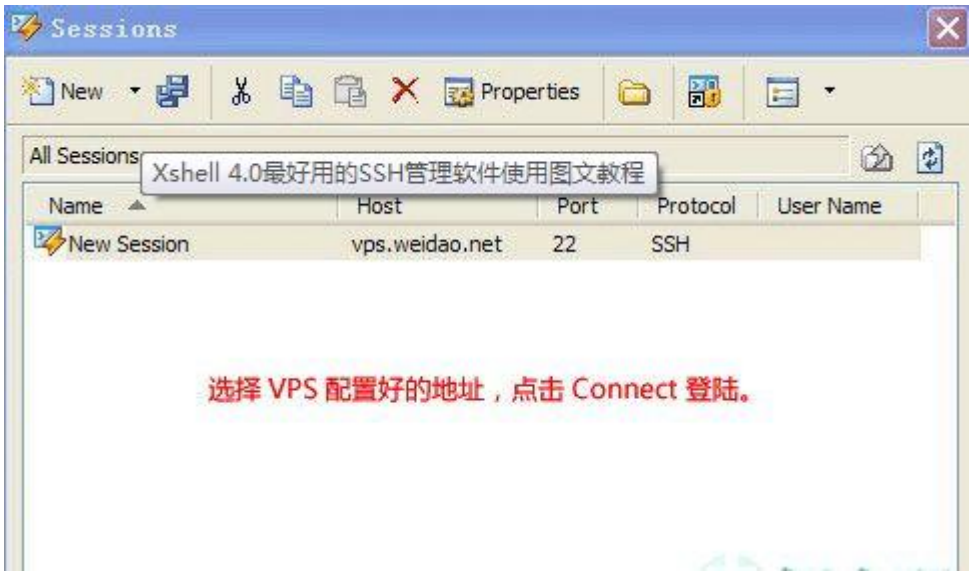
下载好 XShell 软件后，双机 XShell 图标即可运行 XShell 就可以看到下面这个界面：

安装完成之后，点击桌面 Xshell 4 打开，选择“新建”；

在这里输入服务器的 IP 或主机名，选择好登录协议，还有协议的端口，如果希望把这次的输入保存起来，以后就不需要再重新输入了，就在第 4 步输入好会话保存的名称，比如：

mail-server，或者干脆就是主机的地址，点击保存就可以了。

选择配置好的 VPS 地址，点 “Connect”登录；



最后点下面的 connect 按钮，输入正确的用户名和口令，就可以登录服务器了。

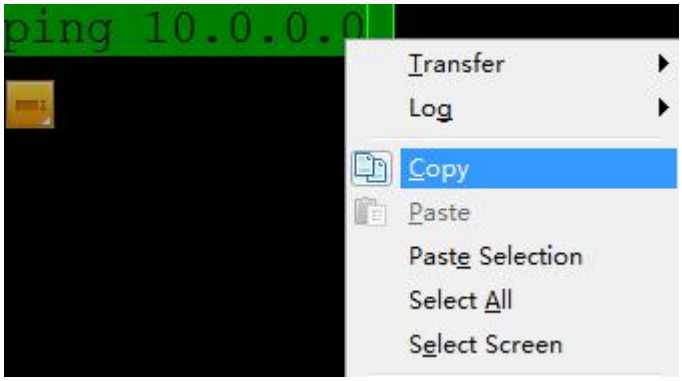
iv. 首次登录一台主机时

第一次登录时，会看到这个对话框：

点 Accept&Save 就保存起来，以后就不会再弹出这个窗口，然后就正常登录。点 Accept Once，下次还是要提示你，然后也可以正常登录。如果一台主机我们只是临时登录一下，当然就是点 Accept Once 了。Cancel 就是取消，也就是取消了这次登录。

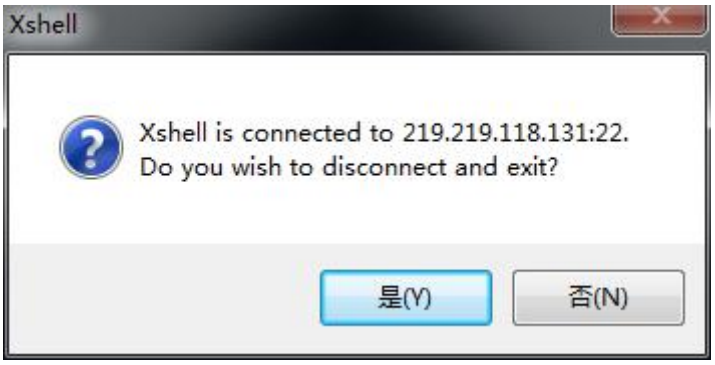
v. 在 XShell 里复制和粘贴

在 XShell 的窗口里面复制、粘贴可不能用 Windows 里的这些 Ctrl+C, Ctrl+V 这些快捷键，Ctrl+C 在控制台上可是终止当前的命令执行。XShell 的选择、复制、粘贴这些操作通过鼠标或者相应的快捷键来完成的。用鼠标操作的话：直接点击右键，即可选择相应的操作；快捷键操作：复制：Ctrl+Insert、粘贴：Shift+Insert



vi. 关于注销登录

成功登录主机后，也能正常看到中文了。这样，我们就可以完成大部分的工作。最后要关闭窗口了，该怎么办呢？我见过很多人，包括我们公司负责专职维护的同事，都是直接点击窗口上的关闭按钮，完全没有理会弹出警告窗口，直接点击了 Yes。



这样做是不对的，首先这不是正确的注销方式，应该输入命令 exit 来正常注销； 其次直接关闭窗口后，你的登录其实还在服务器上，如果一连多次的这样强制关闭窗口，用命令 who 查看时，可以看到很多的用户还在系统上登录，占用了系统的资源。最重要的是，你的这次登录可能只是为了启动什么应用服务器，直接关闭窗口后，可能会导致你的业务在随后的几分钟内也被终止，这应该不是你所希望看到的吧。

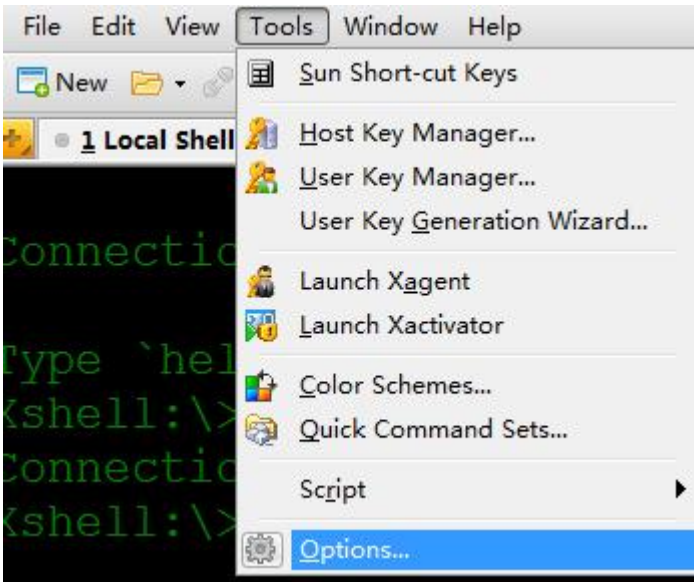
如果上述的理由是每次要输入 exit 然后回车，比较麻烦。你可以用快捷键 Ctrl+d 来注销登录，一般情况下，快捷键一按窗口都直接关闭了，还省了两次鼠标点击。

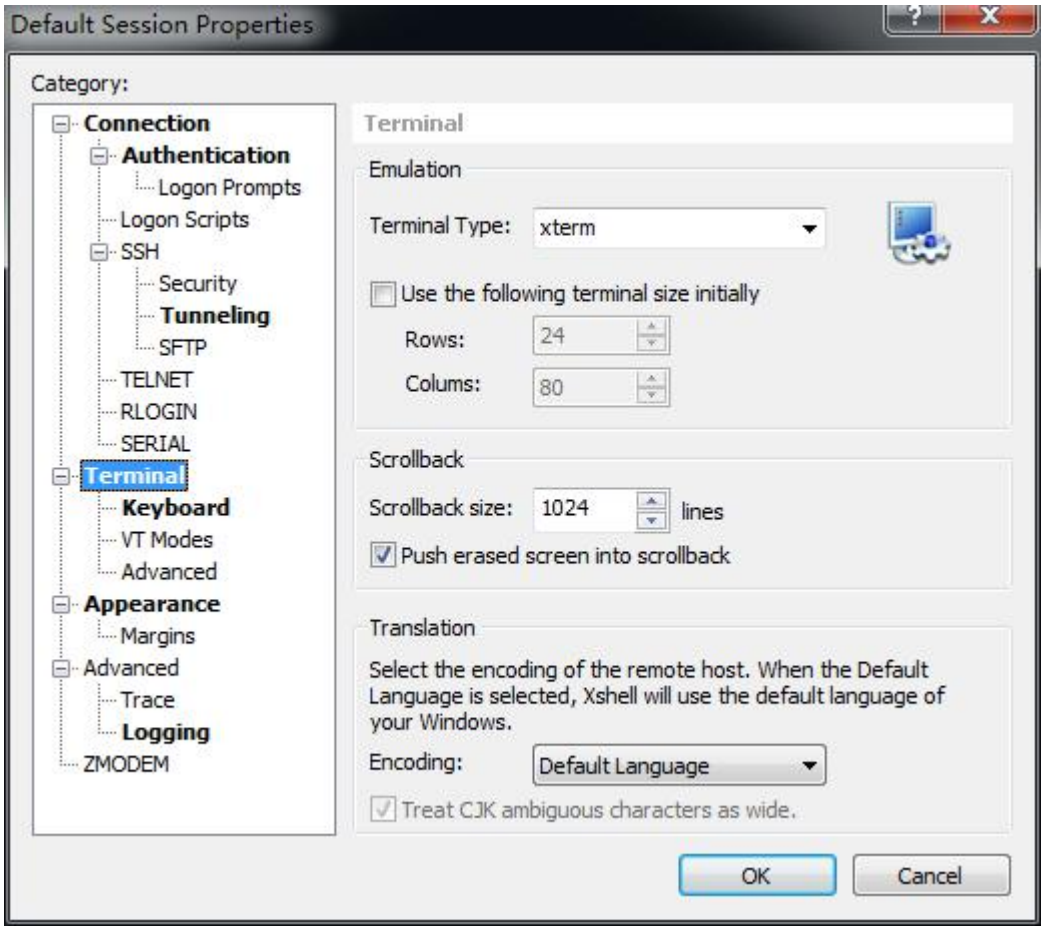
有的程序在执行时，虽然在命令最后面加上 “&”就能放到后台运行。但是正常注销登录后，窗口没有被自动关闭，还能看到程序的输出，这时强制关闭窗口还是可以的。为了避免这种情形，可以使用 nohup 命令。

用法为： nohup 命令 命令参数，这样就可以了。

vii. 窗口保存的输出

执行了一个命令，输出了好多东西，但是默认的配置下，XShell 只保存了最后 1024 行的内容，如果想适当调整的话，还是在标题栏上点选择 Tools——>options...，在配置窗口的左边选择 Terminal，修改右边的 Lines of scrollback



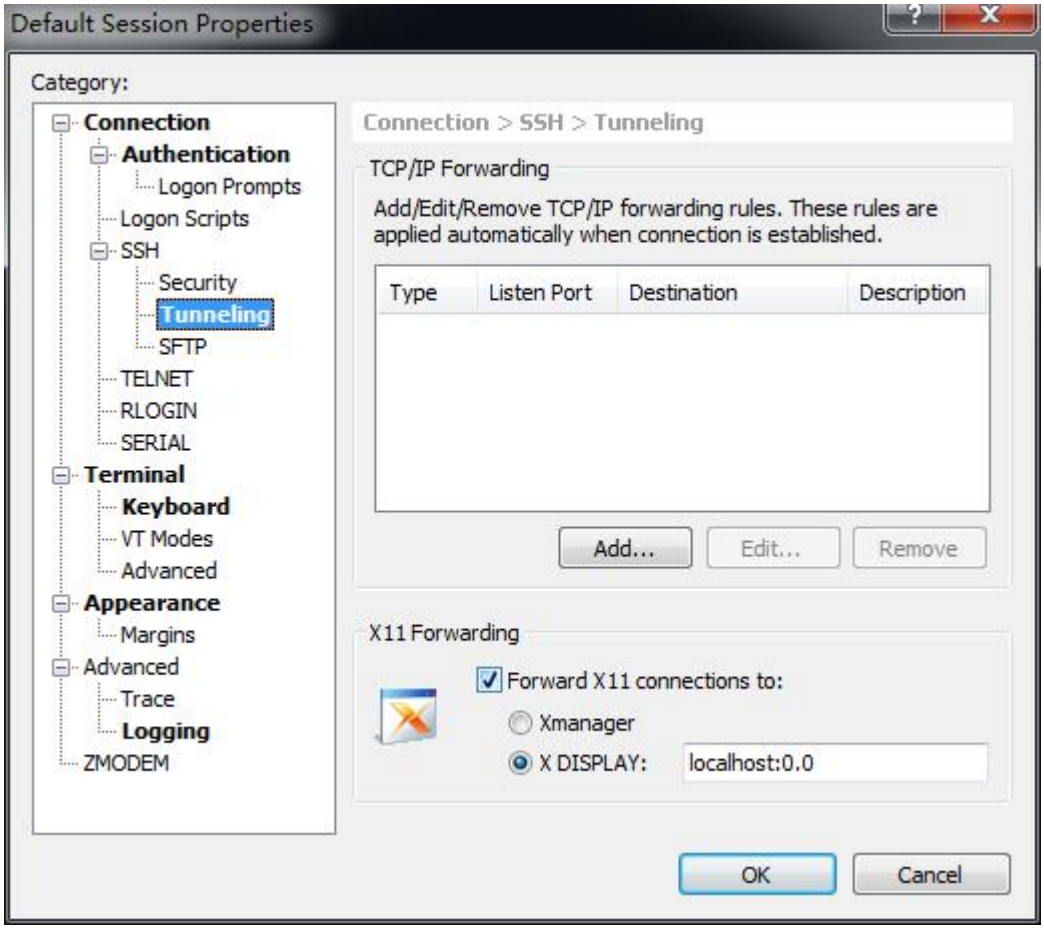


在上面的 Use the following terminal size initially 里设置的是窗口显示的行数和列数，默认是 24 行、80 列，根据自己的需要来修改吧。

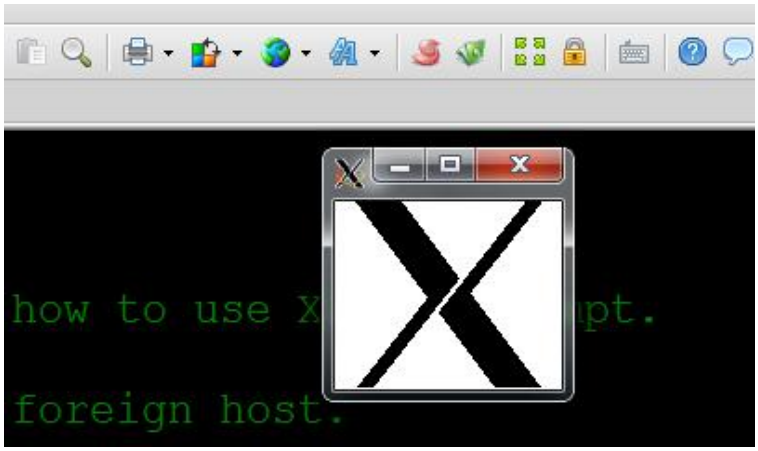
viii. 转发能够让你在 Windows 上使用 Linux 的程序

这里很简单，选中 Forward X11 connections to: X DISPLAY 后登录主机，记得在我

们本地运行 X 服务端程序（比如：免费好用的 Xming）。

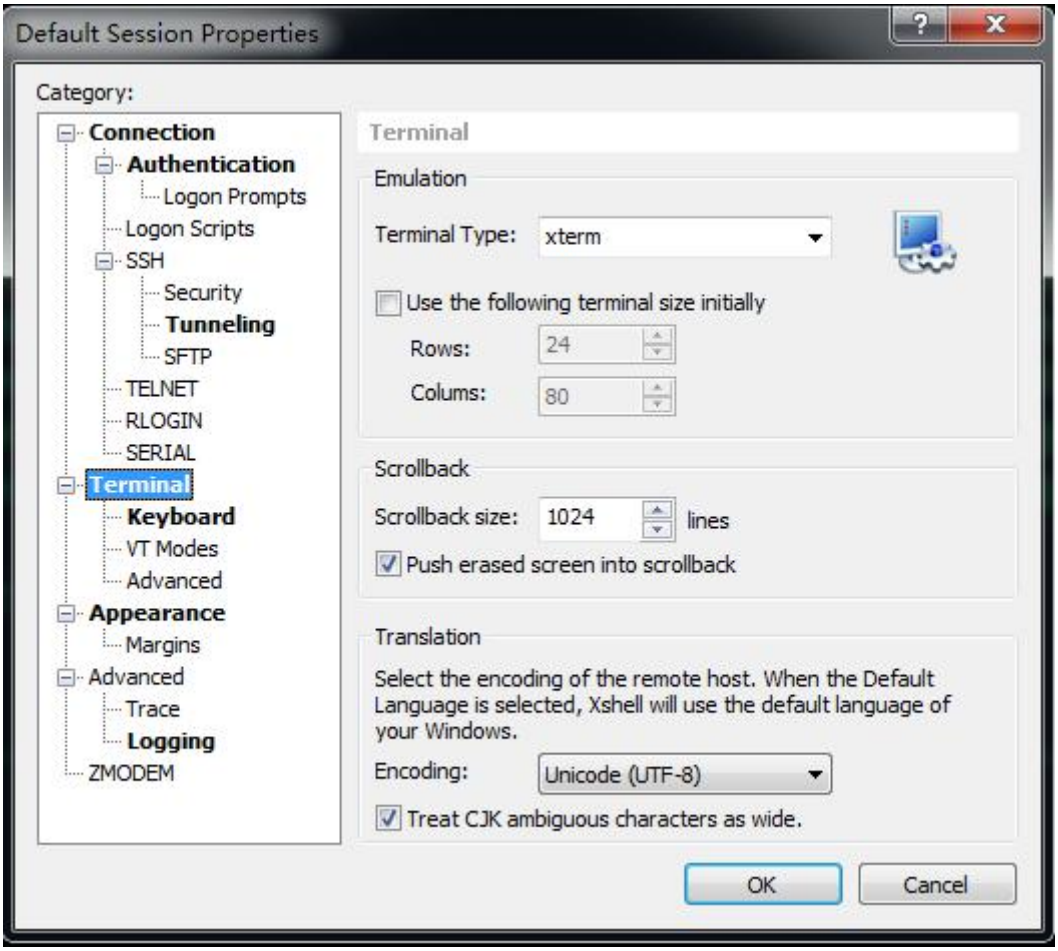


然后在控制台直接输入 X 环境下运行的程序，比如：xlogo，我们就可以看到 Linux 上的 GUI 界面的程序在 Windows 桌面上打开了。



ix. 显示中文乱码的解决方法

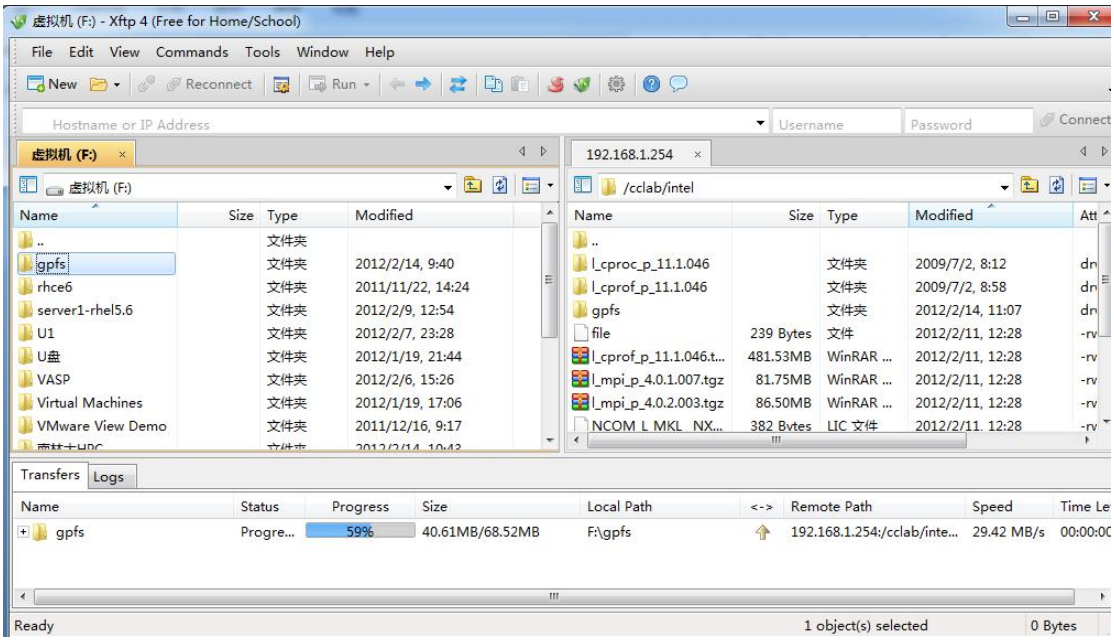
如果 XShell 中有中文显示为乱码的情况，点击 File——>Properties.....，然后在目录中选择 Terminal，在右边的选项 Translation 中，设置 Encoding 为 Unicode(UTF-8)，然后重新启动 XShell 即可解决中文显示为乱码的问题。



便捷的文件传输工具——XFTP

安装好 XShell 以及 XFTP 之后，打开 XShell 会在工具栏上看到 XFTP 已经嵌套其中

当你登录某一个服务器之后，进行文件传输时，直接点击 XFTP 的图标，然后弹出窗口，像在本地进行拷贝操作一样，支持复制、粘贴、删除等快捷键，同时也支持鼠标的拖拽动作，下面的进度条会显示当前操作的进度。



使用 Matlab 步骤：

1、程序设计阶段，在自己电脑或者登陆集群后输入 matlab，打开图形界面进行设计

2、将设计好的 matlab 程序通过 lsf 作业调度提交到集群上

注意：mgt 节点资源有限，任何人不得在 mgt 节点上直接运行程序，如果直接运行程序有可能导致 mgt 节点宕机，lsf 作业调度系统宕机，会影响所有的作业。

二、 LSF 作业调度的使用方法

2.1、lsf 作业脚本的编写

```
#BSUB -q normal                    指定队列
#BSUB -J  test                      作业名称
#BSUB -o  %J.out                   作业运行情况输出
#BSUB -e  %J.err                   错误日志输出
#BSUB  -n  16                      指定 CPU（core）个数
#BSUB  -R "span[ptile=8]"          指定分配的每节点 8 个 CPU
cd /gpfshpc/home/zql/test          进入工作目录
mpirun -np 16  ./tt                执行当前目录下的 tt 文件
```

Matlab 命令行作业：

```
#BSUB -q normal                    指定队列
#BSUB -J  matlab                   作业名称
#BSUB -o  %J.out                   作业运行情况输出
#BSUB -e  %J.err                   错误日志输出
#BSUB  -n  16                      指定 CPU（core）个数
#BSUB  -R "span[ptile=8]"          指定分配的每节点 8 个 CPU
cd /gpfshpc/home/zql/test          进入工作目录
mpirun matlab -nodisplay -r  example     #(example ----> example.m 文件)
或者
mpirun matlab -nodisplay -nodesktop -nosplash < potfit.m > potfit.out
```

2.2、提交作业

```
vim  job.sh
     添加“可执行命令”

bsub  <  job.sh
     提交作业
```

提交 gpu 作业

```
bsub  -q  phy_gpu  -n  28  -R  "span  [hosts=1]  select  [ngpus>0]  rusage  [ngpus_shared=2
8]"  sleep  10000
```

2.3、查看集群中 lsf 可调用的计算资源

```
[root@gpu15 ~]# lshosts
HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
login01      X86_64 Intel_EM  60.0    16  31.6G  62.4G    Yes (mg)
mgt          X86_64 Intel_EM  60.0    16  31.5G  31.2G    Yes (mg)
c01n01       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c01n02       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c01n03       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c01n04       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c01n05       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c01n06       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c01n07       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c01n08       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c01n09       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c01n10       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c01n11       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c01n12       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c02n01       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c02n02       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c02n03       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c02n04       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c02n05       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c02n06       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c02n07       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c02n08       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c02n09       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c02n10       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c02n11       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c02n12       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
```

```
c03n01       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c03n02       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c03n03       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c03n04       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c03n05       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c03n06       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c03n07       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c03n08       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c03n09       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
c03n10       X86_64 Intel_EM  60.0    36 255.5G  62.4G    Yes (mg)
fat01        X86_64 Intel_EM  60.0    12 767.5G  62.4G    Yes (mg)
fat02        X86_64 Intel_EM  60.0    36 511.4G  62.4G    Yes (mg)
fat03        X86_64 Intel_EM  60.0    36 511.5G  62.4G    Yes (mg)
fat04        X86_64 Intel_EM  60.0    36 511.5G  62.4G    Yes (mg)
gpu01        X86_64 Intel_EM  60.0     8 255.8G  62.4G    Yes (mg)
gpu02        X86_64 Intel_EM  60.0     8 255.8G  62.4G    Yes (mg)
gpu03        X86_64 Intel_EM  60.0     8 255.8G  62.4G    Yes (mg)
gpu04        X86_64 Intel_EM  60.0     8 255.8G  62.4G    Yes (mg)
gpu05        X86_64 Intel_EM  60.0     8 255.8G  62.4G    Yes (mg)
gpu06        X86_64 Intel_EM  60.0     8 255.8G  62.4G    Yes (mg)
gpu07        X86_64 Intel_EM  60.0     8 255.8G  62.4G    Yes (mg)
gpu08        X86_64 Intel_EM  60.0     8 255.8G  62.4G    Yes (mg)
gpu09        X86_64 Intel_EM  60.0     8 255.8G  62.4G    Yes (mg)
gpu10        X86_64 Intel_EM  60.0     8 255.8G  62.4G    Yes (mg)
gpu11        X86_64 Intel_EM  60.0     8 511.8G  62.4G    Yes (mg)
gpu12        X86_64 Intel_EM  60.0     8 511.8G  62.4G    Yes (mg)
gpu13        X86_64 Intel_EM  60.0     8 511.8G  62.4G    Yes (mg)
gpu14        X86_64 Intel_EM  60.0     8 511.8G  62.4G    Yes (mg)
gpu15       DEFAULT   POWER8 250.0    16   512G   412M    Yes (mg)
```

2.4、查看集群中 lsf 队列信息

```
[root@gpu15 ~]# bqueues
QUEUE_NAME      PRIO STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
priority         43  Open:Active       -    -    -    -     0     0     0     0
normal           30  Open:Active       -    -    -    -     0     0     0     0
fat-768g         30  Open:Active       -    -    -    -     0     0     0     0
fat-512g         30  Open:Active       -    -    -    -     0     0     0     0
power            30  Open:Active       -    -    -    -     0     0     0     0
tesla            30  Open:Active       -    -    -    -     0     0     0     0
geforce          30  Open:Active       -    -    -    -     0     0     0     0
quadro           30  Open:Active       -    -    -    -     0     0     0     0
interactive      30  Open:Active       -    -    -    -     0     0     0     0
idle             20  Open:Active       -    -    -    -     0     0     0     0
```

2.5、查看 lsf 节点的启动状态

```
[root@gpu01 ~]# bhosts
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
c01n01             ok              -     36      0      0      0      0      0
c01n02             ok              -     36      0      0      0      0      0
c01n03             ok              -     36      0      0      0      0      0
c01n04             ok              -     36      0      0      0      0      0
c01n05             ok              -     36      0      0      0      0      0
c01n06             ok              -     36      0      0      0      0      0
c01n07             ok              -     36      0      0      0      0      0
c01n08             ok              -     36      0      0      0      0      0
c01n09             ok              -     36      0      0      0      0      0
c01n10             ok              -     36      0      0      0      0      0
```

c01n11	ok	-	36	0	0	0	0	0
c01n12	ok	-	36	0	0	0	0	0
c02n01	ok	-	36	0	0	0	0	0
c02n02	ok	-	36	0	0	0	0	0
c02n03	ok	-	36	0	0	0	0	0
c02n04	ok	-	36	0	0	0	0	0
c02n05	ok	-	36	0	0	0	0	0
c02n06	ok	-	36	0	0	0	0	0
c02n07	ok	-	36	0	0	0	0	0
c02n08	ok	-	36	0	0	0	0	0
c02n09	ok	-	36	0	0	0	0	0
c02n10	ok	-	36	0	0	0	0	0
c02n11	ok	-	36	0	0	0	0	0
c02n12	ok	-	36	0	0	0	0	0
c03n01	ok	-	36	0	0	0	0	0
c03n02	ok	-	36	0	0	0	0	0
c03n03	ok	-	36	0	0	0	0	0
c03n04	ok	-	36	0	0	0	0	0
c03n05	ok	-	36	0	0	0	0	0
c03n06	ok	-	36	0	0	0	0	0
c03n07	ok	-	36	0	0	0	0	0
c03n08	ok	-	36	0	0	0	0	0
c03n09	ok	-	36	0	0	0	0	0
c03n10	ok	-	36	0	0	0	0	0
fat01	ok	-	12	0	0	0	0	0
fat02	ok	-	36	0	0	0	0	0
fat03	ok	-	36	0	0	0	0	0
fat04	ok	-	36	0	0	0	0	0
gpu01	ok	-	8	0	0	0	0	0

gpu02	ok	-	8	0	0	0	0	0
gpu03	ok	-	8	0	0	0	0	0
gpu04	ok	-	8	0	0	0	0	0
gpu05	ok	-	8	0	0	0	0	0
gpu06	ok	-	8	0	0	0	0	0
gpu07	ok	-	8	0	0	0	0	0
gpu08	ok	-	8	0	0	0	0	0
gpu09	ok	-	8	0	0	0	0	0
gpu10	ok	-	8	0	0	0	0	0
gpu11	ok	-	8	0	0	0	0	0
gpu12	ok	-	8	0	0	0	0	0
gpu13	ok	-	8	0	0	0	0	0
gpu14	ok	-	8	0	0	0	0	0
gpu15	unavail	-	128	0	0	0	0	0
login01	ok	-	-	0	0	0	0	0
mgt	ok	-	-	0	0	0	0	0

一般有几种状态：ok、close、unaliva、unreach

ok:该节点可以接受作业，可以被分配作业

close：该节点不可以接受作业

UNalive：该节点 lsf 相关服务不可用

unreach：该节点网络存在故障，不可到达

2.6、查看 lsf 作业状态

```
[test@hpc0 ~]$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
208     test    RUN   hpc        hpc0        hpc1        testjob    May  6 19:10
                                             hpc1
                                             hpc2
                                             hpc2
                                             hpc3
                                             hpc3
                                             hpc4
                                             hpc4
                                             hpc5
                                             hpc5
```

2.7、查看 lsf 作业的详细信息

```
[test@hpc0 ~]$ bjobs -lr 208

Job <208>, Job Name <testjob>, User <test>, Project <default>, Status <RUN>, Qu
                     eue <hpc>, Command <#!/bin/bash;#BSUB -q hpc;#BSUB -q hpc;
                     #BSUB -J testjob;#BSUB -e %J.err;#BSUB -o %J.out;#BSUB -n
                     10;#BSUB -R "span[ptile=2]"; data;sleep 100;data>
Fri May  6 19:10:38: Submitted from host <hpc0>, CWD <$HOME>, Output File <%J.o
                     ut>, Error File <%J.err>, 10 Processors Requested, Request
                     ed Resources <span[ptile=2]>;
Fri May  6 19:10:40: Started on 10 Hosts/Processors <hpc1> <hpc1> <hpc2> <hpc2>
                     <hpc3> <hpc3> <hpc4> <hpc4> <hpc5> <hpc5>, Execution Home
                     </data/home/test>, Execution CWD </data/home/test>;
Fri May  6 19:11:16: Resource usage collected.
                     MEM: 1 Mbytes;  SWAP: 326 Mbytes;  NTHREAD: 5
                     PGID: 13639;  PIDs: 13639 13640 13644 13646


 SCHEDULING PARAMETERS:
           r15s   r1m  r15m   ut      pg    io   ls    it    tmp    swp    mem
 loadSched   -     -     -     -       -     -    -     -     -      -      -
 loadStop    -     -     -     -       -     -    -     -     -      -      -
```

三、 Linux 常用命令说明

1. VI 命令(文本编辑器 VI)：

Vi 是 Unix 世界里极为普遍的全屏幕文本编辑器，VIM 是它的改进版本 Vi IMproved 的简称。几乎可以说任何一台 Unix 机器都会提供这套软体。Linux 当然也有，它的 vi 其实是 elvis（版权问题），不过它们都差不多。熟悉 DOS 下的文书处理后，也许会感到 vi 并不好用；Unix 上也已经发展出许多更新、更好用的文书编辑器，但是并不一定每一台 Unix 机器上都会安装这些额外的软体。所以，学习 vi 的基本操作还是有好处，让你在各个不同的机器上得心应手。

Unix 提供一系列的 ex 编辑器，包括 ex, edit 和 vi 。相对于全屏幕编辑器，现在可能很难想像如何使用 ex, edit 这种行列编辑器。 Vi 的原意是 "Visual"，它是一个立即反应的编辑程序，也就是说可以立刻看到操作结果。

也由于 vi 是全屏幕编辑器，所以它必须控制整个终端屏幕哪里该显示些什么。而终端机的种类有许多种，特性又不尽相同，所以 vi 有必要知道现在所使用的是哪一种终端机。这是由 TERM 这个环境变数来设定，设定环境变数方面请查看所使用 shell 的说明。

只要简单的在 Shell 下执行 vi 就可以进入 vi 的编辑环境。在实际操作之前先对它有个了解会比较好。Vi 有两种模式，输入模式以及指令模式。输入模式即是用来输入文字资料，而指令模式则是用来下达一些编排文件、存档、以及离开 vi 等等的操作指令。当执行 vi 后，会先进入指令模式，此时输入的任何字元都视为指令。对于 vi 的详细操作，可参考相关的 Unix 教程。

VI 命令可以说是 Unix/Linux 世界里最常用的编辑文件的命令了，但是因为它的命令集众多，很多人都不习惯使用它，其实您只需要掌握基本命令，然后 加以灵活运用，就会发现它的优势，并会逐渐喜欢使用这种方法。本文旨在介绍 VI 的一些最常用命令和高级应用技巧。

- 进入 vi 命令

>> vi filename :打开或新建文件，并将光标置于第一行首

>> vi +n filename ： 打开文件，并将光标置于第 n 行首

>> vi + filename ： 打开文件，并将光标置于最后一行首

>> vi +/pattern filename：打开文件，并将光标置于第一个与 pattern 匹配的串处

>> vi -r filename ： 在上次正用 vi 编辑时发生系统崩溃，恢复 filename

>> vi filename....filename ： 打开多个文件，依次进行编辑

- 基本命令介绍

1．光标命令

k、j、h、l——上、下、左、右光标移动命令。虽然您可以在 Linux 中使用键盘右边的 4 个 光标键，但是记住这 4 个命令还是非常有用的。这 4 个键正是右手在键盘上放置的基本位置。

nG——跳转命令。n 为行数，该命令立即使光标跳到指定行。

Ctrl+G——光标所在位置的行数和列数报告。

w、b——使光标向前或向后跳过一个单词。

2．编辑命令

i、a、r——在光标的前、后以及所在处插入字符命令(i=insert、a=append、r=replace)。

cw、dw——改变(置换)/删除光标所在处的单词的命令 (c=change、d=delete)。

x、d$、dd——删除一个字符、删除光标所在处到行尾的所有字符以及删除整行的命令。

3．查找命令

/string、?string——从光标所在处向后或向前查找相应的字符串的命令。

4．拷贝复制命令

yy、p——拷贝一行到剪贴板或取出剪贴板中内容的命令。

- 常见问题及应用技巧

1．在一个新文件中读/etc/passwd 中的内容，取出用户名部分。

>> vi file

>> :r /etc/passwd 在打开的文件 file 中光标所在处读入/etc/passwd

>> :%s/:.*//g 删除/etc/passwd 中用户名后面的从冒号开始直到行尾的所有部分。

您也可以在指定的行号后读入文件内容，例如使用命令“:3r /etc/passwd”从新文件的第 3 行开始读入 /etc/passwd 的所有内容。

我们还可以使用以下方法删掉文件中所有的空行及以#开始的注释行。

>> #cat squid.conf.default | grep -v ^$ | grep -v ^#

2．在打开一个文件编辑后才知道登录的用户对该文件没有写的权限，不能存盘，需要将所做修改存入临时文件。

>> vi file

>> :w /tmp/1 保存所做的所有修改，也可以将其中的某一部分修改保存到临时文件，例如仅仅把第 20～59 行之间的内容存盘成文件/tmp/1，我们可以键入如下命令。

>> vi file

>> :20,59w /tmp/1

3．用 VI 编辑一个文件，但需要删除大段的内容。

首先利用编辑命令“vi file”打开文件，然后将光标移到需要删除的行处按 Ctrl+G 显示行号，再到结尾处再按 Ctrl+G，显示文件结尾的行号。

>> :23,1045d 假定 2 次得到的行号为 23 和 1045，则把这期间的内容全删除，也可以在要删除的开始行和结束行中用 ma、mb 命令标记，然后利用“:a,bd”命令删 除。

4．在整个文件的各行或某几行的行首或行尾加一些字符串。

>> vi file

>> :3,$s/^/some string / 在文件的第一行至最后一行的行首插入“some string”。

>> :%s/$/some string/g 在整个文件每一行的行尾添加“some string”。

>> :%s/string1/string2/g 在整个文件中替换“string1”成“string2”。

>> :3,7s/string1/string2/ 仅替换文件中的第 3 行到第 7 行中的“string1”成“string2”。

注意: 其中 s 为 substitute，%表示所有行，g 表示 global。

5．同时编辑 2 个文件，拷贝一个文件中的文本并粘贴到另一个文件中。

>> vi file1 file2

yy 在文件 1 的光标处拷贝所在行

:n 切换到文件 2 (n=next)

p 在文件 2 的光标所在处粘贴所拷贝的行

:n 切换回文件 1

6. 替换文件中的路径。

使用命令“:%s#/usr/bin#/bin#g”可以把文件中所有路径/usr/bin 换成/bin。也可以使用命令“:%s//usr/bin //bin/g”实现，其中“”是转义字符，表明其后的“/”字符是具有实际意义的字符，不是分隔符。

- 更多高级命令和使用：

i. 光标移动：

h ： 光标左移一个字符

l ： 光标右移一个字符

space：光标右移一个字符

Backspace：光标左移一个字符

k 或 Ctrl+p：光标上移一行

j 或 Ctrl+n ： 光标下移一行

Enter ： 光标下移一行

w 或 W ： 光标右移一个字至字首

b 或 B ： 光标左移一个字至字首

e 或 E ： 光标右移一个字至字尾

) ： 光标移至句尾

(： 光标移至句首

}：光标移至段落开头

{：光标移至段落结尾

nG：光标移至第 n 行首

n+：光标下移 n 行

n-：光标上移 n 行

n$：光标移至第 n 行尾

H ： 光标移至屏幕顶行

M ： 光标移至屏幕中间行

L ：光标移至屏幕最后行

0：（注意是数字零）光标移至当前行首

$：光标移至当前行尾

ii. 屏幕翻滚类命令

Ctrl+u：向文件首翻半屏

Ctrl+d：向文件尾翻半屏

Ctrl+f：向文件尾翻一屏

Ctrl＋b；向文件首翻一屏

nz：将第 n 行滚至屏幕顶部，不指定 n 时将当前行滚至屏幕顶部。

iii. 插入文本类命令

i ：在光标前

I ：在当前行首

a：光标后

A：在当前行尾

o：在当前行之下新开一行

O：在当前行之上新开一行

r：替换当前字符

R：替换当前字符及其后的字符，直至按 ESC 键

s：从当前光标位置处开始，以输入的文本替代指定数目的字符

S：删除指定数目的行，并以所输入文本代替之

ncw 或 nCW：修改指定数目的字

nCC：修改指定数目的行

iv. 删除命令

ndw 或 ndW：删除光标处开始及其后的 n-1 个字

do：删至行首

d$：删至行尾

ndd：删除当前行及其后 n-1 行

x 或 X：删除一个字符，x 删除光标后的，而 X 删除光标前的

Ctrl+u：删除输入方式下所输入的文本

v. 搜索及替换命令

/pattern：从光标开始处向文件尾搜索 pattern

?pattern：从光标开始处向文件首搜索 pattern

n：在同一方向重复上一次搜索命令

N：在反方向上重复上一次搜索命令

：s/p1/p2/g：将当前行中所有 p1 均用 p2 替代

：n1,n2s/p1/p2/g：将第 n1 至 n2 行中所有 p1 均用 p2 替代

：g/p1/s//p2/g：将文件中所有 p1 均用 p2 替换

vi. 选项设置

all：列出所有选项设置情况

term：设置终端类型

ignorance：在搜索中忽略大小写

list：显示制表位(Ctrl+I)和行尾标志（$)

number：显示行号

report：显示由面向行的命令修改过的数目

terse：显示简短的警告信息

warn：在转到别的文件时若没保存当前文件则显示 NO write 信息

nomagic：允许在搜索模式中，使用前面不带“\”的特殊字符

nowrapscan：禁止 vi 在搜索到达文件两端时，又从另一端开始

mesg：允许 vi 显示其他用户用 write 写到自己终端上的信息

vii. 最后行方式命令

：n1,n2 co n3：将 n1 行到 n2 行之间的内容拷贝到第 n3 行下

：n1,n2 m n3：将 n1 行到 n2 行之间的内容移至到第 n3 行下

：n1,n2 d ：将 n1 行到 n2 行之间的内容删除

：w ：保存当前文件

：e filename：打开文件 filename 进行编辑

：x：保存当前文件并退出

：q：退出 vi

：q!：不保存文件并退出 vi

：!command：执行 shell 命令 command

：n1,n2 w!command：将文件中 n1 行至 n2 行的内容作为 command 的输入并执行之，若不指定 n1，n2，则表示将整个文件内容作为 command 的输 入

：r!command：将命令 command 的输出结果放到当前行

viii. 寄存器操作

"?nyy：将当前行及其下 n 行的内容保存到寄存器？中，其中?为一个字母，n 为一个数字

"?nyw：将当前行及其下 n 个字保存到寄存器？中，其中?为一个字母，n 为一个数字

"?nyl：将当前行及其下 n 个字符保存到寄存器？中，其中?为一个字母，n 为一个数字

"?p：取出寄存器？中的内容并将其放到光标位置处。这里？可以是一个字母，也可以是一个数字

ndd：将当前行及其下共 n 行文本删除，并将所删内容放到 1 号删除寄存器中。

tags 用法

在函数中移动光标

[{ 转到上一个位于第一列的"{"

}] 转到下一个位于第一列的"{"

{ 转到上一个空行

} 转到下一个空行 （[and] 也分别是两个指令）

ctrl+t 跳回

gd 转到当前光标所指的局部变量的定义

* 转到当前光标所指的单词下一次出现的地方

转到当前光标所指的单词上一次出现的地方

- **Mail 命令**

如果使用 test 用户帐号登录 Lenovo HPC 平台集群，在终端经常出现：

You have new mail in /var/spool/mail/test

这肯定是系统给 test 员发的一些通知邮件，我常用 mail 命令来查看。

- **Lenovo HPC 平台用户使用**

i. 进入系统查看邮件：

>>mail 进入 mail 系统

ii. 如果希望查看已经阅读过的 mail：

>> mail -f ~/mbox

- **Mail 使用说明：**

>> mail 进入 mail 系统

【注意】下面 mail 命令的 help 帮助是我个人随便翻译的，准确的请看英文说明

&help //如果不会使用或者忘记了什么命令，就输入 help 或者 ？来获取帮助

Mail Commands

t <message list> 打印出信息 【注意】多个信息用空格分开，如 t 1 7

n 打印出下一条信息

e <message list> 编辑信息

f <message list> 输出信息的头行

d <message list>　　删除信息

s <message list> file　　追加信息到文件 file

u <message list>　　不删除某信息

R <message list>　　回复发件人

r <message list>　　回复发件人和本信息所有的收件人

pre <message list>　　保留信息在 /usr/spool/mail　1*

m <user list>　　发邮件，多个收件人用空格分开【需要 sendmail 支持】

q　　quit, saving unresolved messages in mbox 2*

x　　quit, do not remove system mailbox

h　　print out active message headers

!　　让 shell 执行某命令，如 !ls 输出 ls 命令结果

cd [directory]　　改变目录，这里不用 !cd，但打印当前目录需要 !pwd

1* 如果阅读过某信息，执行 q 退出时候，mail 会把阅读过的信息“取回”放在 ~/mbox，执行 pre 后就不取回。如果退出时执行 x 命令则不会取回信息。

2* 这三句不难理解，但不好翻译（本人水平问题），也就保留原文

Example:

& t 7　　//阅读第 7 封信息，阅读时，按空格键就是翻页，按回车键就是下移一行

& d 10　　//删除第 10 封信息

& d 10-100　　//删除第 10-100 封信息

& top　　//显示当前指针所在的邮件的邮件头

& file　　// //显示系统邮件所在的文件，以及邮件总数等信息

& x　　//退出 mail 命令平台，并不保存之前的操作，比如删除邮件

& q　　//退出 mail 命令平台,保存之前的操作，比如删除

Linux 常用命令

- cd directory　　进入指定的目录
- cd ..　　进入上一级目录
- cd /directory　　进入目录
- cd　　进入用户自己的目录
- cp file_from file_to　　拷贝文件
- ln [-s] source linkname　　为一个文件建立连结
- ls [directory]　　查看指定目录下的文件
- ls -l [directory]　　查看指定目录下文件的详细
- ls -a [directory]　　查看指定目录下的所有文件
- mkdir new_directory　　建一个新目录
- more file　　查看一个文本文件的内容
- rm file　　删除一个文件
- rm -r directory　　删除一个目录
- rmdir directory　　删除一个目录
- find . -name "file"　　从当前目录开始查找指定的文件
- adduser　　创建新用户
- alias　　设置别名或替代名
- bg fg　　使挂起的进程继续运行
- ps ax　　查询当前进程
- mount　　连接文件系统
- more　　浏览文件内容
- less　　浏览文件内容
- chown chgrp　　改变文件的拥有者
- chmod　　改变文件属性
- halt　　关闭系统
- man　　显示手册页
- passwd　　改变用户口令
- grep　　查找字符串

- find 查找文件
- dd 复制磁盘或文件系统
- kill 杀掉一个进程
- killall 杀掉进程

四、Linux 平台软件安装方法

4.1、rpm 安装方法

rpm 执行安装包

二进制包（Binary）以及源代码包（Source）两种。二进制包可以直接安装在计算机中，而源代码包将会由 RPM 自动编译、安装。源代码包经常以 src.rpm 作为后缀名。

常用命令组合：

－ivh：安装显示安装进度--install--verbose--hash

－Uvh：升级软件包--Update；

－qpl：列出 RPM 软件包内的文件信息[Query Package list]；

－qpi：列出 RPM 软件包的描述信息[Query Package install package(s)]；

－qf：查找指定文件属于哪个 RPM 软件包[Query File]；

－－replacepkge ： 无论软件包是否已被安装，都强行安装软件包；

--nodeps ：忽略软件包的依赖关系强行安装；

--force ：忽略软件包及文件的冲突；

－e：删除包

rpm -q samba //查询程序是否安装

rpm -qa|grep samba //查询程序是否安装

rpm -ivh --relocate /=/opt/gaim gaim-1.3.0-1.fc4.i386.rpm //指定安装目录

rpm -Uvh --oldpackage gaim-1.3.0-1.fc4.i386.rpm //新版本降级为旧版本

4.2、yum 安装方法

yum 的理念是使用一个中心仓库(repository)管理一部分甚至一个 distribution 的

应用程序相互关系，根据计算出来的软件依赖关系进行相关的升级、安装、删除等等操作，减少了 Linux 用户一直头痛的 dependencies 的问题。这一点上，yum 和 apt 相同。apt 原为 debian 的 deb 类型软件管理所使用，但是现在也能用到 RedHat 门下的 rpm 了。

yum 主要功能是更方便的添加/删除/更新 RPM 包，自动解决包的倚赖性问题，便于管理大量系统的更新问题。

首先要配置好本地或者网络 yum 源

yum 仓库配置径：/etc/yum.repod/

```
#vim   /etc/yum.repod/server.repo       配置本地 yum 源
[rhel]
name=rhel
baseurl=file:///opt/iso
enabled=1
gpgcheck=0
```

Yum 常用指令

yum list　　#列出资源库中所有可以安装或更新的 rpm 包

yum list updates　　#列出资源库中所有可以更新的 rpm 包

yum list installed　　#列出已经安装的所有的 rpm 包

yum list extras　　#列出已经安装的但是不包含在官方资源库中的 rpm 包，例如安装了 epel 源的 rpm 包会列出来

yum grouplist　　#安装了的组和可以安装的组一览显示

yum search perl　　#搜索匹配特定字符的 rpm 包

yum provides libstdc++.so.6　　#反查包含特定文件名的 rpm 包，查询命令用 yum provides */ifconfig，也可用 yum whatprovides

yum install perl　　#安装 perl 包

yum install perl*　　#安装 perl 开头的包

yum remove perl*　　#会删除 perl* 所有包，以及相关依赖的包

yum groupinstall "Chinese Support"　　#安装指定的组

yum groupupdate "Chinese Support"　　#安装了的组成员软件包更新

yum groupremove "Chinese Support"　　#删除指定的组

yum groupinfo "Chinese Support" #指定组所包含的软件包显示

yum check-update #检查可更新的 rpm 包

yum update #更新所有的 rpm 包

yum update kernel kernel-source #更新指定的 rpm 包,如更新 kernel 和 kernel source

yum upgrade #大规模的版本升级,与 yum update 不同的是,连旧的淘汰的包也升级

yum clean packages #清除暂存中 rpm 包文件

yum clean headers #清除暂存中 rpm 头文件

yum clean oldheaders #清除暂存中旧的 rpm 头文件

yum clean all #清除暂存中旧的 rpm 头文件和包文件

4.3、源码编译安装方法

源码编译安装主要是用于安装一些开源的软件：

可以在软件开源社区下载到，如：https://github.com

源码的安装一般由 3 个步骤组成：

配置(configure)；

编译(make)；

安装(make install)；

rhel 源码编译安装：

默认编译器：

[root@mgt ~]# which gcc

/usr/bin/gcc

默认调用的文件位置：

/lib 和/usr/lib 这两个目录下的库文件

/usr/inlcude 这么目录下的头文件

默认安装位置：/usr/local/

4.4、python 包安装方法

4.4.1、离线源码包安装

下载 python 软件源码包

```
python  setup.py  install
```

4.4.2、在线安装

```
pip  install  软件-版本号
```

例如在线安装 Django-1.11.3：

```
pip install Django==1.11.3
```

4.4.3、修改 pip 的 Python 源站，提升下载安装的速度

修改方式：

1) 临时修改方法：

```
pip  install  -i  https://pypi.tuna.tsinghua.edu.cn/simple  gevent
```

2) 永久修改方法：

修改 ~/.pip/pip.conf (没有就创建一个)， 修改 index-url 至 tuna

```
[global]
index-url = https://pypi.tuna.tsinghua.edu.cn/simple
```

4.4.4、查看 Python 调用模块路径

```
#python
import os, sys
print >> sys.stderr,sys.path
```

注意点：集群上使用 Python 时，不要使用系统自带的 Python，因为系统默认的 Python 目录一般不作为集群的共享目录，所以如果用系统自带的会造成，在登录节点可用，但其他节点就不可用了

4.5、R 模块安装方法

4.5.1、离线安装

#R CMD INSTALL /.../mypackage.tar.gz

需要先把软件包下载好，此方法安装时会出现依赖关系的问题，类似于 rpm 安装的方式；

4.5.2、在线线安装

R

> install.packages('mypackage')

可以使用 install.packages()安装本地下载的包，尤其适用于在服务器上安装包，会自动解决依赖关系

4.5.3、 Bioconductor 的安装方法

> source("http://bioconductor.org/biocLite.R")

> biocLite("mypackage")

这类包的安装也是会自动解决依赖关系；

4.5.4、 卸载 package

remove.packages("mypackage")

4.5.5、 更新包

update.packages( ) 可以定期执行以下

4.6、perl 模块安装

4.6.1、安装 perl 离线安装包，先下载好软件包：

perl Makefile.PL

make

```
make test
make install
```

4.6.2、perl 模块在线安装命令如下：

```
#perl -MCPAN -e 'install perl模块'
```

例：perl -MCPAN -e 'install Archive::Extract'

修改 cpan 源：

```
vi /usr/share/perl5/CPAN/Config.pm
    'urllist' => [q[http://mirrors.aliyuncs.com/CPAN/]],
```

4.6.3、直接进入 cpanm 模式安装：

```
#cpanm  Archive::Zip
```

修改 cpanm 的源

```
#vim  .bashrc
alias cpanm='cpanm --sudo --mirror http://mirrors.163.com/cpan --mirror-only'
```

4.6.4、查看 perl 模块是否已经安装完成：

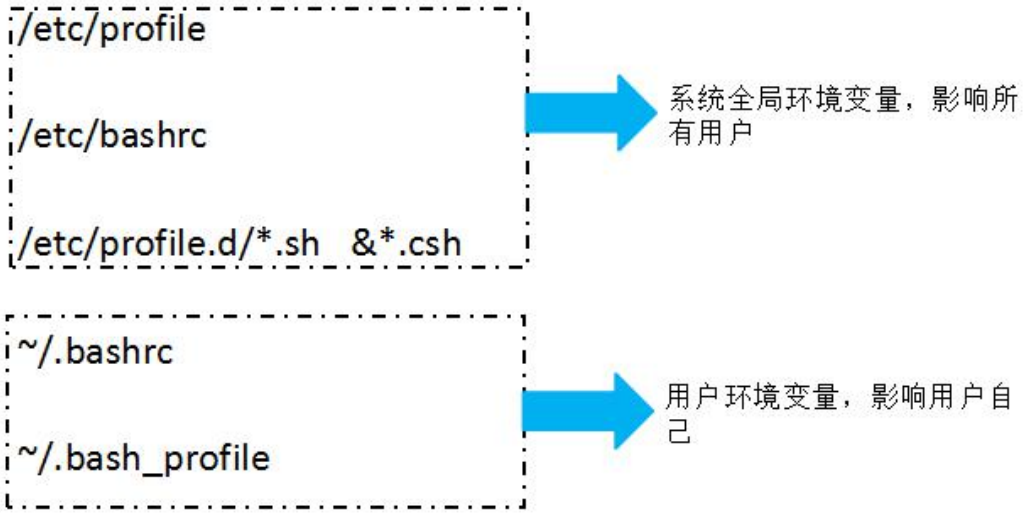
perldoc Archive::Zip 查看 perl 模块的使用方法（类似于 linux 的 man）

4.6.5、卸载方式：

```
# pm-uninstall MongoDB
```

五、环境变量配置

linux 中的环境变量配置文件：



如何查看当前用户环境变量？

执行命令：env

env 命令用于显示系统中已存在的环境变量，以及在定义的环境中执行指令

例：

```
[root@localhost ~]# env
hostname=LinServ-1
TERM=linux
SHELL=/bin/bash
HISTSIZE=1000
.........
```

如何编写环境变量？

格式如下：

export PATH=$PATH:/share/apps/openmpi/bin

或：export PATH=/share/apps/openmpi/bin:$PATH

常用变量名称:

PATH

LD_LIBRARY_PATH

C_CLUDE

.....

六、Lenovo HPC 平台集群常见问题说明

6.1、 普通用户无法使用 SSH 登录集群

a. 请查看远程主机 IP 是否输入正确

b. 请检查用户名和密码是否输入正确

c. 在远程主机 IP 和用户名、密码输入正确的情况下，如果仍然无法登陆远程主机，请使用 ping 命令，查看是否能 ping 通远程主机

d. 如果可以 ping 通远程主机，请联系管理员，请管理员检查文件系统是否挂载正确

e. 如果无法 ping 通远程主机，请联系管理员，请管理员检查集群是否开启，网络是否正常

6.2、 普通用户 SSH 登录集群密码错误

a. 请联系管理员，请管理员检查文件系统是否挂载正确

b. 如果文件系统挂载正确，请管理员重置用户密码

6.3、 普通用户 SSH 登录集群后，用户家目录下无文件

a. 请联系管理员，请管理员检查文件系统是否挂载正确

6.4、 普通用户 SSH 登录集群后，登录计算节点和提交作业需要密码

a. 请联系管理员，请管理员检查文件系统是否挂载正确