linux、vim 和 shell 的新手入门手册

李丛宇[1]　刘昊[1]　张景旭[1]　谢世卿[1]
柴新宇[1]　祁康辉[2]　王雄飞[1]

([1] 兰州大学物理科学与技术学院 甘肃 兰州 730107
[2] 中国科学院近代物理研究所 甘肃 兰州 730000)

前言

想要参与高能物理实验就必须会用 linux，而使用 linux 的过程中难免要用到 vi（vim）。而 shell 语言虽然不算是必须，但掌握一些知识，能写简单脚本，对于提高工作效率是很有帮助的。

笔者认为学习一种操作系统或者一种编程语言，最好是带有一定的目的性去学，为实现某种需求去学习某种命令，学完后立马实践看看是否能满足自己的需求，不能满足则要学习更高深的知识。我并不是反对通读教材，系统性的学习，但是在时间匮乏的情况下，“干中学”不失为一种好方法。

笔者至今也没有系统性的学习过有关知识，所以说下文我所写的内容只是我懂得的知识，各种命令一定没有写全，即使是对于我提到的命令，它可以带的各种参数我也没有全部掌握。如果以下内容仍然没有覆盖到读者的需求，请您上网搜索，多浏览几篇文章，应该就可以掌握。

本文将会不断更新，也欢迎各位读者批评指正。

目录

第一章　linux

linux 和 Windows、Mac OS 一样，都是电脑操作系统。因笔者未曾使用过苹果系统，下文涉及到系统操作的比较时，均以微软系统为例。

首先我们要通过软件来连接远端的服务器，这类软件有很多，笔者感觉比较独特的有 PuTTY 和 MobaXterm。

PuTTY 最大的优点是小巧，功能基本上也够用。缺点就是没有多窗口，这个大概可以通过 Windows10 的多桌面来弥补一下，但超过 3 个窗口恐怕就很难受了。另外 PuTTY 不能看图，这样 root 一半的功能就无法发挥了。

MobaXterm 弥补了 PuTTY 的两个缺点，而且还能记住密码，比较方便。另外 PuTTY 的左侧可以图形化显示 linux 下的文件，就类似于 Windows 文件资源管理器的左侧，而且可以随意把服务器上的东西下载（download）到本地。

软件下载下来直接使用不一定舒适，可以先调节一下字体颜色，字号大小。另外，大多数软件在长时间不操作后会与服务器断开，解决方法就是在设置 settings-SSH 中找到 SSH keepalive 并勾选（图 1.1），或者在 connection 的第一行中把 0 改为非 0 数（意思是每间隔多少秒向服务器发送一个空包以保持连接）（图 1.2）。

另外，至少以上两款软件是没有官方中文版的，其他汉化版不能保证其安全性，因此还是到官方下载为好。

linux 和 Windows 两种系统最直观的区别就是：Windows 有图形化界面，上手容易，给小孩子一个下午或许就能自己探索出如何使用，而 linux 一般是非图形化的，让小孩子自己探索恐怕就很难了，甚至可能还没探索出 ls 呢，先执行了 rm -r * 甚至 rm -rf /*。

非图形化界面的一切操作都是通过在屏幕上显示的哪一行里输入命令（command）来实现的，我们可以称之为命令行，鼠标基本上失去了作用，

图 1.1: MobaXterm

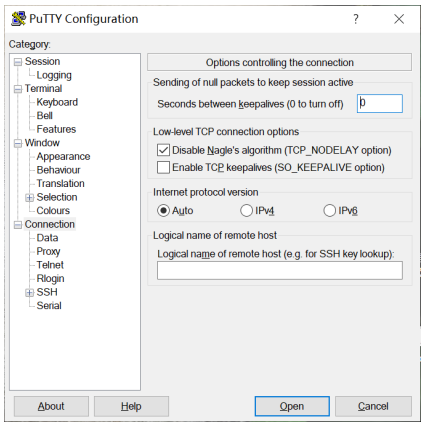

图 1.2: PuTTY

但用来复制粘贴还是很方便的。linux 中（包括 vim 里），选中文字即复制，粘贴只需要右键单击即可。

下面详细介绍一下常用命令。

1.1 cd

即 change directory。Windows 下可以通过点击返回来回到上一层目录，点击对应文件夹进入下一层目录，这在 linux 中都是通过 cd 命令来实现的，cd .. 返回上一层目录，cd xx 进入名为 xx 的文件夹。

另外在 linux 中，一个. 就是指当前所在目录，两个.. 指上一层目录。

cd 命令后面可以加绝对路径和相对路径。

绝对路径往往比较长，所以平时我们可能不太愿意使用它，不过也正是因为它长，所以定位准确。我们执行 pwd(print working directory) 命令，就能看到我们当前目录的绝对路径。

相对路径是以一个点. 和两个点.. 为基础的，而我们平时很少见到一个点，那是因为它被简化了。我们要进入当前目录下的名为 xx 的文件夹，按照相对路径写法，应该是 cd ./xx/，实际使用中，前面的./和最后的斜杠/都可以省略，于是就变成了我们喜闻乐见的 cd xx。

linux 下 tab 键可以自动补全，所以说在 Linux 下文件名可以命的长一点，反正打几个字母 tab 一下就能补全。如果你输入的文字对应到不止一个文件，Linux 会显示出来，如`vi ru` tab 会补全出`vi run`，上面会显

示run_conder.sh run.head run.foot（图 1.3），所以说有时用 tab 就可以实现下文的 ls 的功能。

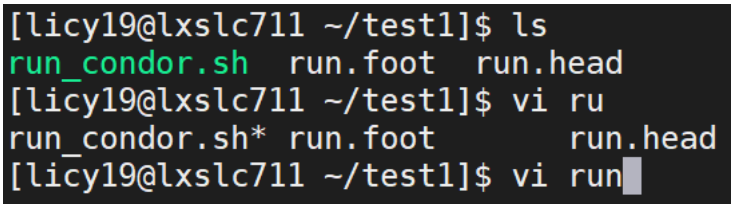

图 1.3: tab 自动补全

1.2 ls

即 list。你可能会问：Windows 每进一个目录都能看到下面的东西，通过右键 -属性还能看到关于文件的更多信息，linux 怎么什么都看不到？这是因为这也需要执行命令。

执行 ls，我们就能看到当前目录下的文件和文件夹（以蓝色显示）（不含隐藏文件），这个命令对于文件较少的目录还是能很快显示的，文件多了就会有明显的等待时间，而且这个命令也会对服务器造成冲击，而用/bin/ls 就只是显示文件名，执行速度就快多了。我们可以把这个命令设为快捷指令 l（后面会讲）。

ls -a 可以显示出以. 开头的隐藏文件，ls -l 或 ll 可以显示文件的详细信息。

linux 下有一种模糊功能，* 可以表示任何名称，所以我们执行 ls *.txt 就可以列出当前目录下所有以.txt 结尾的文件。

顺便说一下，linux 下的文件名后缀可能只是个后缀了，与文件属性没什么关系，但使用合理的后缀可以让我们的文件更加整齐。

cmd 命令行中，按上下键可以查看命令的历史记录，左右键自然就是左右移动光标了。

1.3 mkdir

即 make directory。linux 下创建文件夹用 mkdir 加名称，如果要创建多个文件夹，不是用 and & 连接，而是直接空格，其他命令对多个文件操

作也是如此。而创建多层目录需要加上参数 -p，如`mkdir -p test1/test2/test3/test4`。

1.4 rm(dir)

删除文件夹用 rmdir，不过这个命令只能删除空文件夹，不太常用，一般都用 rm -r 加文件夹名，rm 可以删除文件，-r 表示对文件夹进行操作（下面均以文件演示，文件夹只需要补加 -r），-i 开启互动模式，在删除前会询问使用者，避免误删。

1.5 cp

复制文件用`cp+`来源`+`去向，都可以使用绝对路径或者相对路径。复制到当前目录下则必须改名，如`cp test1 test2`，复制到其他地方则可改可不该，最后不写新的文件名就以原文件名复制过去。

1.6 mv

移动文件或者改名，和 cp 基本差不多，在当前目录下 mv 是改名，如`mv test1 test2` 将 test1 改名为 test2，`mv+`文件`+`文件夹是把文件移动到文件夹，也可以再加上/xx 顺便改名为 xx。

1.7 diff

比较两个文件，如有区别，会分别输出两个文件中的对应内容，方便比较（图 1.4）。

1.8 快捷键

设置指令的别名。在家目录 home 下一个神奇的隐藏文件.tcshrc 中输入 alias 别名“指令名称”，如`alias l "/bin/ls"`。

如果把 ls 设置为/bin/ls 的快捷指令，又想使用真正的 ls，则要加转义符\，即`\ls`，不过一般还是不要让快捷指令覆盖到真实指令为好。

```
[licy19@lxslc711 ~/test1]$ diff test1.txt test2.txt
1,5c1,5
< favourite
< colour
< flavour
< honour
< labour
---
> favorite
> color
> flavor
> honor
> labor
```

图 1.4: diff

1.9 文件内容查看

cat 正着看，tac 倒着看（由最后一行输出到第一行）；nl 带着行号看（可以替代 cat）；head 只看头几行，tail 只看尾几行，均默认显示 10 行，看所有以 log 结尾的文件的后 3 行即`tail -n3 *log`。

1.10 grep

grep ‘字符串’文件目录，引号可单可双，对于短字符串不加引号也行。不加任何参数则输出文件名和对应字符串。

`grep "xxx" * | wc -l` 用来查找含有 xxx 字符的文件数，而`wc -l` 本身也是用来统计行数的命令。例如

```
grep 'ApplicationMgr     INFO Application Manager Finalized successfully' *log | wc -l
```

的作用是输出 log 文件中含有这一行成功信息的文件数，如果输出结果和 log 文件数一致则说明所有作业都成功运行，建议将这条命令整合到快捷指令中。

1.11 sed

`sed -i 's/xxx/yyy/g'` 用于替换，其中参数 -i 表示直接修改原文件内容，不加则把修改后的内容显示在屏幕上 (输出到终端)，原文件内容不变，对我们常见的批量替换需求来说用处不大。引号可单可双，s、g、/可以简单的看作形式，而且这一形式和后面 vim 里的替换很像。

/是一种定界符，两个//之间的东西才是要查找和替换的内容，如果内容中也含有/（一般是程序语言的注释//或者路径），则可以用转义符 -反斜线\ 告诉计算机这是真正的斜线/而不是定界符/，也可以用井号 # 做定界符。有的时候内容中同时含有/和 #，那就必须使用转义符了，不过可以通过合理选择定界符来减少转义符的使用。例如`sed -i 's/1000/1/g' test*.txt` 就是将`test_00.txt`、`test_01.txt` 中的 1000 替换为 1.

Sed '1d'xx 即删除 xx 中的第一行，'2，5d' 表示删除第二到五行，特别的，$ 表示最后一行。

1.12 查找 + 替换

例如`sed -i "s/#path/path/g" `grep "#path" * -rl``，先查找含有 #path 的文件，得到文件名，再对这些文件进行替换操作（也就是去掉注释 #）。注意要用反引号 ‘将 grep 语句括起来。另外这种查找和替换是同一个字符串这种写法应该说仅仅是个人习惯，它和直接 grep 语句换成 * 没有本质区别。想要有区别的话，可以增长 grep 搜索的字符串，提高搜索精度。

另一种解决方案是分步进行，先搜索你想要替换的字符串，观察是否可以全部替换，如果可以，替换的操作对象可以打 *，否则就要明确对象，多个对象之间用空格隔开即可。

1.13 hadd

用来将多个 root 文件合成为一个，格式为`hadd output.root input.root`，例如`hadd data.root *.root`, 既然是把当前目录下所有 root 合并，那就可以直接写成 *.root 或者 *root。另外这条命令不同于大多数命令，是先写 output，再写 input 的。

1.14 root

root 博大精深，可以再写一篇文章，这里只提一下基本操作。在任何一个目录下输入 root（加不加具体.root 都行）回车，蹦出几行字，然后会出现新的命令行，再输入`TBrowser g`（参数可随意替换），就能打开一个新窗口（浏览器），就能看到 root，下面有树，再往下还有枝，退出就是输入`.q`(ctrl+z 也可以)

上面说的其实是 root+xx.root 的情况，其实 root 还可以加 c 或 cpp 程序，如果需要看图，就用`root -l x.c`，如果不需要看图，就用`root -l -q y.cxx`。

-l 在笔者看来最显著的作用就是提高运行速度，不过即使不加这个参数，也是可以正常运行的。-q 就是让程序运行完后自动退出，而不用人工输入.q 了。

1.15 上传和下载

常用 scp 指令。在你所用的软件中打开一个本地窗口，不做连接，执行 scp 远端地址本地（./），例如 scp licy19@lxslc7.ihep.ac.cn:/scratchfs/bes/ licy19/LamLambar/Data/Psipscan/ Data/*jpg /home/mobaxterm/Desktop。

1.16 输出重定向

`command > file` 将输出重定向到 file，» file 则将输出以追加的方式重定向到 file。

问题的关键在于追加，这里的区别在后面写 shell 脚本时会有更明显的体现，如果全程只向 file 中写入一次，那么 > 和 » 没有区别，如果目的是通过循环向 file 里写入，并且想保留每次写入的信息，那就必须使用 »，这样每次循环中写入都是以追加而的方式来进行的，之前的信息都不会丢失。而 > 这种方式就有一种覆盖的意味，即使循环写入了 9 次，保留的也只是最后一次写入的信息。

1.17 后台运行

command &：后台运行，这时理论上后台运行的任务对当前 cmd 窗口的操作不造成影响，但如果这个 command 是带有输出的，那么前台还是会有输出（这一点对于下面的 nohup 也是一样的）。& 对 ctrl+z（暂停）和 ctrl+c（终止）免疫，想要终止可以直接退出终端，command 就被停止，而如果用 exit 退出，该 command 不会被挂断，这一点能做到和 nohup 相同。

如果想要停止，应该要先输入 fg 将任务调到前台再暂停或停止。

注：ctrl+z 和 ctrl+c 当然是有区别的，不过在平时只用 ctrl+z 也没什么大问题，毕竟一般暂停之后也不会再让任务继续跑。

nohup command：表示不挂起（no hang up），也即，关闭终端或者退出某个账号，进程也继续保持运行状态，因为 nohup command 后该指令相当于被上传到远端的服务器，这时候无论你是 exit、退出终端、关机甚至突然断电、断网，该命令都是不会被挂断的，这对于那些执行时间较长的命令来说尤为重要，应所以说 nohup 的抗冲击性比 & 还要好，唯一不如 & 的地方就是当前 cmd 窗口不能再干其他事情，所以把 nohup sh 或者 nohup sh & 设为快捷指令也都是可以的。

另外，如果将`nohup shy yy.sh &`写入一个 xx.sh 脚本，在执行 xx.sh 后，yy.sh 的输出将被定向到 nohup.out 里，不会在屏幕上输出，屏幕上只会显示`nohup: ignoring input and appending output to 'nohup.out'`，加之 & 的后台运行功能，此时 cmd 窗口能正常使用，前台也不会不断输出提交作业的信息，上面那一句话出现的次数和 xx.sh 脚本中提交 yy.sh 的数量是一致的，出现频率也不会太高。

如果上文所说的脚本提交后发现有问题，该如何终止呢？答案是悲观的，可能根本停不下来。靠物理方法（指断网、退出终端等）不再适用，直接暂停或终止也不管用，fg、jobs 也调不出任务，这应该说是把 nohup command & 写在脚本里的坏处。连任务都找不到，更不用谈停止任务了。

总之，综合评判，至少在外部提交脚本时最好用 nohup sh，至于脚本内 nohup 和 & 用不用看个人习惯，如果都写上的话中途可能停止不了，写 & 可以干其他事情，但体验不好，因为屏幕上还在不断输出；写 nohup 实际上并没有什么用，只是能让屏幕输出变得简洁一些。

注：在高能所环境下，`hep_sub job.sh` 也可以将程序提交到后台，作用和 nohup 相似。

第二章　vim

Vim 和 vi 差不多，具体的区别对我们使用者来说也不重要，我们也可以将快捷指令 v 和 vi 都指向 vim。

vi 可以分为三种状态，分别是命令模式（command mode）、插入模式（Insert mode）和底行模式（last line mode），有时把底线模式也看作一种命令模式。插入模式自然不用说，就像 word 或者 notepad 一样输入文本即可。

在命令模式下按 a、i、o 都可以进入插入模式，a 光标后移一格，i 光标不动，o 光标下移一行（插入空白行）。按 esc 键退出插入模式。无论是命令模式还是插入模式，光标都可以通过上下左右键或 hjkl 键自由移动，通过 pg up、pg dn 或者 ctrl+b（backward）、ctrl+f（forward）可以实现向上向下翻一整页，而 ctrl+u/d 则是翻半页。

命令模式下，双击 g（gg）到达文本开头，双击 G（GG）到达文本结尾，输入行号 +gg（G）到达指定行号。

dd 删除光标所在行，delete 向后删除一个字符。

u 可以回到上一步操作，也就是撤销操作，类似于 Windows 下的 ctrl+z。

在命令模式下输入冒号进入底线模式，可以输入一些命令，回车执行，所以说底线模式看作命令模式是有一定道理的。

输入数字就可以到达对应行号，特别的，首行就是 1，末行则可以输入一个较大的数字，比如说你日常处理的文档不会超过 1000 行，那你可以输入 1111，这样就可以到达末行。

set nu 可标注行号，w 保存，q 退出，！是强制的意思（force），比较常用的是保存退出 wq，wq！是强制保存退出，q 退出只能在未作修改的条件下进行，否则必须用 wq 或者 q!，这是你应该思考一下到底需不需要保存，如果你是不小心做了更改，又不知道改了哪里，那就直接 q！强制退出，文

件不会变样。另外，长时间编辑文档，即使不用 wq 保存退出，也最好时不时用 w 保存一下，否则一旦断电断网，你所做的更改就消失殆尽了。

`:n1,n2s/word1/word2/g`，n1、n2 为行号，写成 1，$ 就是从第一行到最后一行，或者写成%，所以全局替换常用`%s/word1/word2/g`。

在命令模式下输入/或？可以用来搜索希望的关键字，一个不是可以按 n 向后（对应/）或向前（对应？）继续查找，另外 N 与 n 的作用相反，所以我们只用/也可以完成向前或向后查找。

在命令模式下按 ctrl+v 可进入块选择模式，可以对相邻多行进行增删工作。上下左右移动光标，被选中的部分会被高亮，此时 delete 就可以全部删除。如果想要批量插入一段话，那么就不能左右移动光标了，上下移动选中，输入大写的 I 进入插入模式，输入需要的文字，按 esc，过一会你所选中的行都会被做相同的改动。

第三章　shell

Shall 具体又分为 sh、bash 和其他很多种，至少在笔者写一些简单程序时，没发现 sh 和 bash 有什么区别，下面所讲的也就是思路方面的东西，毕竟变量和循环是各种语言几乎都有的东西。

3.1　shell 语言性质

shell 语言兼具批处理和脚本的性质。

批处理，顾名思义，成批的处理。shell 语言写出的程序很适合做重复性的工作。

脚本语言，相对于编译语言来说，它不需要经过编译过程，可以通过简单的命令被执行，它的执行过程也几乎是线性的，就是按照程序员所写的东西一行一行执行，即使碰到循环，也就是把那几行代码按顺序执行多次。shell 脚本里所写的东西拿出来放到 Linux 的命令行中都可以被执行，即使是循环，把它打开，变量添上，也可以执行。虽然如此，但每个命令单独执行都需要时间，所以为了省时间，我们把想让计算机做的事情整合到一个文本里，直接执行这个文本即可。

脚本语言比外部命令多的一些东西便是定义变量（数组）和执行循环。

3.2　变量

定义变量很自然，就是变量名 =”xx”，单引号和双引号有区别，这里不讨论。

引用变量也有多种形式，这里演示三种。定义 var=”shell”，则`$var=${var}=shell`, 这两种方式的结果相同，都返回变量值，推荐带有大括号的写法，避免歧义。而变量前再加井号 #，可输出字符串的长度，`${#var}=5`。

3.3 数组

对于数组，shell 只支持一维数组，对于多维数组就需要其他方法实现了，而且作为一种比较简单的脚本语言，也没必要让它用多维数组做事。定义数组也很简洁，就是`array=(x1 x2 ⋯ xn)`, 各数组元素之间用空格分隔。也可以指定数组元素赋值，如 array[0]=1,array[2]=3。和 c 语言一样，数组元素从 0 开始数，后面写循环也要注意从 0 开始，最后的上限用小于号，这样循环的次数可以对上。

引用数组元素，其实就是把数组的每个元素看成一个变量，用 ${} 引用，如`${array[2]}=3`。

3.4 判断语句

关于 shell 的 if 语句和 test 语句，仅举一例：

```
num1="avourite"
num2="favorite"
if test $num1 = $num2
then
    echo '两个字符串相等!'
else
    echo '两个字符串不相等!'
fi
```

字符串测试用 = 和！=，数值测试用 -eq（等于）和 -ne（不等于）以及其他的 -gt（大于）-lt（小于）-ge(大于等于）-le(小于等于)。

If 语句结尾用 fi，即将 if 反过来，后面的循环也是如此。

3.5 流程控制

循环以 for 举例：

```
for((i=1;i<=5;i++));
do
command1;
```

```
.....
Commandn;
done;
```

应该说除了 do 后面不可加分号，其他语句都是可加可不加的，如果已经比较熟悉 c 语言，那么每句都带上分号也无妨。

3.6 函数

格式如下：

```
[ function ] name [()]
{
    command;
    [return int;]
}
```

方括号里是可以省略不写的部分，但我们应该知道它们也都是有自己的含义的。function 是 shell 中的关键字，表示后面是一个函数（或者叫方法，不同语言对其称呼不同）；name 是函数名，后面的括号在其他语言中可能是需要填入参数的，但 shell 中不用，所以说也是可以省略的；函数中的 return 可以返回一个范围在 0 到 255 的整数。另外关键字 function 和参数括号必须要保留一个，不能全部省略。

函数体里的分号都是可加可不加的，除非你的函数只有一行，那才必须要加分号。

调用函数只需要键入函数名（不带括号）。如果需要传递参数，参数之间用空格隔开。如果想要调用的函数在另一个文件中，我们一样可以调用，只是要在函数名前面加一句 source function.sh。

3.7 开头

和 c 语言的 include ……类似，shell 脚本开头永远是 #!/bin/bash（sh），但是经过笔者亲测，不加一般也能正常执行。

3.8 执行与调试

执行 shell 脚本，最简单的方法就是（ba）sh test.sh，调试就是在中间加 -x，如 sh -x 1.sh。由于 shell 脚本本来就是顺序执行，调试的显示也十分清晰，以一个正号 + 表示一句命令，下面可能会展示命令的执行情况，如果这个 + 所执行的命令又会执行其他命令，则以更多的正号表示。

另外，也可以通过 chmod +x 命令是脚本具有可执行权限，再用./执行。

3.9 注释

Shell 中的注释符就是 #，该行在 # 后面的内容就被注释掉。

多行注释和 c 语言不同，不是基于 #，可以通过把不需要的部分构造成函数，不去调用，间接达到（多行）注释的效果。另外也可以通过以下命令来实现：

```
<<!
......
!
```

被注释掉的部分会变红。

第四章　shell 脚本举例

笔者学习 shell 就是为了写一些简单的脚本去完成复杂的事情。简单指的是语句易懂，虽然 linux 操作细节很多，但笔者也只会上文列举的那些基本操作，但仅凭不超过十种句型，我们就可以把原来机械化的操作都融合到脚本中，经过几次调试，最后就能达到一键完成工作的目的。

下面的脚本都是笔者自己写出，或者是将别人的操作步骤整合起来，都经过了调试和测试，基本不会出问题。笔者会把脚本所在路径写出，对于我们 BES 组内成员，可以直接去 copy。如若不能 copy，可以尝试复制代码；如果复制出来为乱码，则可以借助 OCR 工具将图片转为文字。当然如果您想自己敲出来，笔者觉得这也是极好的。

4.1　删除作业

/afs/ihep.ac.cn/users/l/licy19/others/rm.sh

```
#!/bin/bash
echo 这是一个按照区间删除作业的脚本
read -p "请输入需要删除作业的区间（整数）(num1 num2) > " num1 num2
echo "开头：$num1"
echo "末尾：$num2"
for((i=$num1;i<=$num2;i++));do
hep_rm ${i};
done;
```

该脚本用到了键盘读入的命令：read，其实也挺简单，就是读取参数，多个参数之间用空格隔开。-p 后面跟的是提示信息。

4.2 配置环境

/afs/ihep.ac.cn/users/l/licy19/Envir_config.sh（updateconfig.sh）

```
#!/bin/bash
#默认 705 环境，703 以及其他版本可移步其他脚本
#应用 bash config_envir.sh 执行，sh 执行会出现中断（原因未知）
read -p 'yourname(to replace $name)' name
sed -i 's/$name/$name/g' test.sh
cd /ihepbatch/bes/licy19
#/ihepbatch/bes/licy19/
##mkdir cmthome
##mkdir workarea
cd cmthome
#/ihepbatch/bes/licy19/cmthome/

cp /cvmfs/bes3.ihep.ac.cn/bes3sw/cmthome/cmthome-7.0.5Slc6Centos7Compat ./ -r
cd cmthome-7.0.5-Slc6Centos7Compat
#/ihepbatch/bes/licy19/cmthome/cmthome-7.0.5-Slc6Centos7Compat

sed -i "s/#macro WorkArea/macro WorkArea/g"
`grep "#macro WorkArea" * -rl`
sed -i "s/#path/path/g" `grep "#path" * -rl`
sed -i "s/maqm/licy19/g" `grep "maqm" * -rl`

source setupCVS.sh
source setupCMT.sh
cmt config
source setup.sh
echo $CMTPATH # 调试
cd
#/afs/ihep.ac.cn/users/l/licy19 （家目录）

# 获得 .tcshrc 文件
```

```
### cp /afs/ihep.ac.cn/users/l/licy19/.tcshrc ./

# 由于隐藏文件 .* 不可利用 sed 替换命令从外部修改，故先用 mv 移动
命令改名，完成 sed 后改回
mv .tcshrc tcshrc

#sed -i "s/cmthome-7.0.5-Slc6Centos7Compat/cmthome-7.0.3-Slc6
Centos7Compat/g" tcshrc
sed -i "s/cmthome-7.0.3-Slc6Centos7Compat/cmthome-7.0.5-Slc6
Centos7Compat/g" tcshrc

num1=86
num2=92
# 第一句将 num1 对应的 testrelease 注释掉
# 第二句将 num2 对应的 testrelease 的注释打开
sed -i "s#source /ihepbatch/bes/licy19/workarea/TestRelease/
TestRelease-00-00-$num1#\#source /ihepbatch/bes/licy19/workarea/
TestRelease/TestRelease-00-00-$num1#g" tcshrc
sed -i "s#\#source /ihepbatch/bes/licy19/workarea/TestRelease/
TestRelease-00-00-$num2#source /ihepbatch/bes/licy19/workarea/
TestRelease/TestRelease-00-00-$num2#g" tcshrc

mv tcshrc .tcshrc

cp /ihepbatch/bes/licy19/workarea/TestRelease/ /ihepbatch/bes/licy19/workarea -r
cd /ihepbatch/bes/licy19/workarea/TestRelease/TestRelease-00-00-$num2/cmt
#/ihepbatch/bes/licy19/workarea/TestRelease/TestRelease-00-00-92/cmt

sed -i "s#/ihepbatch/bes/licy19/workarea#/ihepbatch/bes/licy19/workarea#g"
 `grep /ihepbatch/bes/licy19/workarea *sh -rl`

cmt broadcast cmt config
cmt broadcast gmake
source setup.sh
```

```
echo $CMTPATH


PATH1="$CMTPATH"
PATH2="/ihepbatch/bes/licy19/workarea:/ihepbatch/bes/licy19/
workarea:/cvmfs/bes3.ihep.ac.cn/bes3sw/ExternalLib/SLC6/
ExternalLib/gaudi/GAUDI_v23r9:/cvmfs/bes3.ihep.ac.cn/bes3sw/
ExternalLib/SLC6/ExternalLib/LCGCMT/LCGCMT_65a"

if test $PATH1 = $PATH2
then
    echo 'done!'
else
    echo 'something wrong?'
fi
```

该脚本看起来复杂，实际上就是把配置环境的步骤整合起来，用户使用时只需要输入自己的名字（主用户名）即可实现一键配置自己的环境。

4.3 跑作业脚本集

高能物理实验需要进行大量计算，而提交这些计算任务的过程是枯燥而漫长的，笔者将这一过程写成了几个脚本，确实给我节省了很多时间。我们的梦想就是一键科研，把更多的时间节省出来做更有意义的工作。路径如下：

/afs/ihep.ac.cn/users/l/licy19/lam（xi）

两个过程有一些区别，xi 过程比 lam 过程要多交一些作业，相应的脚本也会复杂一些，下面以 lam 过程为例。

4.3.1 模拟:1sim.sh

```
#!/bin/bash
```

```
# 个能量点有关信息9
no=(
3581
3670
3680
3682
3684
3685
3686
3691
3709
)
ecms=(
3.58154
3.67016
3.68014
3.68275
3.68422
3.68526
3.68650
3.69136
3.70976
)
runno=(
-55375,0,-55461
-55462,0,-55541
-55542,0,-55635
-55636,0,-55662
-55663,0,-55690
-55691,0,-55716
-55717,0,-55737
-55738,0,-55795
-55796,0,-55859
```

```
)

#in the KKMC
cd /scratchfs/bes/licy19/LamLambar/KKMC # 绝对路径，下同

#下两句根据需要取舍，作者个人喜欢保留一个原始文件夹如2222
#cp -r 3* 2222
#rm -r 3*

for((i=0;i<9;i++));
do
cp -r 2222 ${no[i]};
cd ${no[i]};
sed -i "s/3.58154/${ecms[i]}/g" *;
sed -i "s/-59163,0,-59573/${runno[i]}/g" *;
nohup sh Mysimbar.tex;
nohup sh Mysim.tex;
cd ..;
done;

#再把 Data 跑上
#cd ../Data
cd /scratchfs/bes/licy19/LamLambar/Data/Psipscan/Data
nohup sh run_condor.sh

#还有 inc
#提前 copy 好run* 三件套（run09.sh 、run.foot 、run.head ）
#看看 run.head 里的分析包路径是否需要改。虽然share 里面只有一个文件，但不同分析包对应的名称还是不同的。
#cd ../Inc_MC
cd /scratchfs/bes/licy19/LamLambar/Inc_MC
nohup sh run09.sh
```

模拟一般不能和重建一起跑，但可以和数据（Data）和 Inc_MC 一起跑。

4.3.2 重建:2rec.sh

```
#!/bin/bash

# 9 个能量点编号
no=(
3581
3670
3680
3682
3684
3685
3686
3691
3709
)
runno=(
55375
55462
55542
55636
55663
55691
55717
55738
55796
)

#in the KKMC
cd /scratchfs/bes/licy19/LamLambar/KKMC

for((i=0;i<9;i++));
do
cd ${no[i]};
```

```
#复制出测试脚本，前台运行，输出重定向到 .log 中以计算，ISR 初态辐射因子（修正后比上修正前）
cp sim_00_00.txt test.txt;
sed -i 's/1000/1/g' test*.; txt
boss.exe test.txt > test.; log
sed -i 's/Correction = 1/Correction = 0/g' test*.txt;
boss.exe test.txt >> test.; log

#提交 rec 作业
nohup sh Myrecbar.tex;
cd ..;
done;
```

4.3.3 KKMC 到 Exc_MC:3kkmc.sh

```
#!/bin/bash

no=(
3581
3670
3680
3682
3684
3685
3686
3691
3709
)

#in the KKMC
cd /scratchfs/bes/licy19/LamLambar/KKMC
```

```
for((i=0;i<9;i++));
do
cd ${no[i]};
#ls -R *.dst > xx.txt;
ls LamLam_*.dst |
       while read file_name;
       do
              echo "/scratchfs/bes/licy19/LamLambar/KKMC/${no[i]}/
${file_name%.*}".dst"" >> ${no[i]}.txt
       done

sed -i '1d' *${no[i]}.txt # 删除合成文件的第一行，凑成100 个
cd ..;
done;

#from KKMC to MC
cd ../Exc_MC
#将合成 dst 的一个绝对路径/scratchfs/bes/licy19/LamLambr/KKMC/
3581/3581.txt 放到MC 中run_condor1.sh 中的recFile 里
for((i=0;i<9;i++));
do
cp -r run_condor.sh ${no[i]}.sh;
sed -i "s/3581/${no[i]}/g" ${no[i]}.sh;
nohup sh ${no[i]}.sh;
rm ${no[i]}.sh; # 清理一下没用的中间文件
done;
```

4.3.4 Data 中画图:4data.sh

```
no=(
3581
3670
3680
```

```
3682
3684
3685
3686
3691
3709
)
runno=(
55375
55462
55542
55636
55663
55691
55717
55738
55796
)

#in the Data
cd /scratchfs/bes/licy19/LamLambar/Data/Psipscan/Data

#合成一个data.文件root
hadd data.root *.root
cd ..
mkdir root
cd -
#scan_wangxf.不加，生成的cxxcutxxxx.用来画图root包括总图和各个能量点的图，共有个(10)
#scan_wangxf1.加cxxcut求出中心框粒子数，
#scan_wangxf2.加cxxcut求出边缘框粒子数，
cp data.root ../root

for((i=0;i<9;i++));
```

```
do
cp /afs/ihep.ac.cn/users/l/licy19/lam/code/scan_wangxf*.cxx ./
cp /afs/ihep.ac.cn/users/l/licy19/lam/code/box.c ./
sed -i "s/newdata/new${no[i]}/g" *.c*; 对#scan_wangxf.和cxxbox.进行操作c
sed -i "s#//if(!(run>${runno[i]}#if(!(run>${runno[i]}#g" scan_wangxf*.cxx;
root -l -q scan_wangxf1.cxx >> N_s.out; 共生成个#1N_s.root
root -l -q scan_wangxf2.cxx >> N_bkg.out; 共生成个#1N_bkg.root
root -l -q scan_wangxf.cxx; 生成#newxxxx.root画分图,
root -l box.c 命名为#xxxx
done;

#提取我们需要的N?, 放到=out.中txt
grep 'N=' N*.out >> out.txt

cp /afs/ihep.ac.cn/users/l/licy19/lam/code/Box.c ./
cp /afs/ihep.ac.cn/users/l/licy19/lam/code/scan_wangxf.cxx ./
root -l -q scan_wangxf.cxx
root -l -q Box.c 命名为#newdata
```

4.3.5 Exc_MC 中画图:4mc.sh

```
#!/bin/bash
no=(
3581
3670
3680
3682
3684
3685
3686
3691
3709
```

```
)

#in the Exc_MC
cd /scratchfs/bes/licy19/LamLambar/Exc_MC
#scan_wangxf.不加，生成的cxxcutxxxx.用来画图root且(中各能量点生成的文件分散，需要移到同一个文件夹中MCroot)
#scan_wangxf1.加cxxcut求出中心框粒子数，
#scan_wangxf2.加cxxcut求出边缘框粒子数，
mkdir root

for((i=0;i<9;i++));
do
cd ${no[i]};
hadd ${no[i]}.root *.root;
cp ${no[i]}.root ../root;
cd ..;
done;

cd root
hadd mc.root *.root

for((i=0;i<9;i++));
do
cp /afs/ihep.ac.cn/users/l/licy19/lam/code/scan_wangxf1.cxx ./
sed -i "s/3581/${no[i]}/g" scan_wangxf1.cxx; 对#scan_wangxf1.进行操作cxx
sed -i "s#//if(!(run>${runno[i]}#if(!(run>${runno[i]}#g" scan_wangxf1.cxx;
root -l -q scan_wangxf.cxx >> N.out; 共生成个#1N.root
done;

grep 'N=' N*.out > out.txt

cp /afs/ihep.ac.cn/users/l/licy19/lam/code/Box.c ./
sed -i "s/data/mc/g" Box.c
```

```
root -l -q Box.c 命名为#mc
```

最后要说一下，笔者为了在能力范围内尽可能提高脚本的自动化水平，对其中所用到的程序也做了一些改动，它们都放在/afs/ihep.ac.cn/users/l/licy19/lam/code 下。

参考文献

[1] 鸟哥: 鸟哥的 *Linux* 私房菜 [M]

[2] runoob.com:*Linux* 教程, https://www.runoob.com/linux/linux-tutorial.html[OL]

[3] 博客园: 单行多行注释，https://www.cnblogs.com/sssblog/p/10143138.html[OL]