Linux与生物信息数据处理
实验指导书

编制单位：生命健康信息学院

编制人：解增言

编制时间：2024年9月

课程说明

一、课程名称：Linux 与生物信息数据处理

二、总课时数： 理论 48 学时，实验 16 学时

三、先修课程：计算机基础，普通生物学

四、课程教材：

理论部分：解增言. Linux 与生物信息学数据处理. 自编讲义，2018

实验部分：解增言. Linux 与生物信息学数据处理实验指导书. 2022

五、上机实验要求：

本课程的上机实验要求：

（1）掌握 Linux 系统的基本操作和 Vim 编辑器的使用；

（2）了解 Linux 环境下 Python 语言的编写，C 语言的编写、编译及运行方法；

（3）掌握 Shell 编程的基本语法；

（4）掌握命令历史、环境变量、管道、重定向的概念及使用方法；

（5）能较熟练地运用 Linux 命令和 Shell 脚本程序处理生物数据。

六、考核方式：

平时成绩（考勤、平时表现等）：50%

实验报告：50%

目录

实验 1：Linux 常用命令（1）：基础命令

一、实验目的

1. 掌握 Linux 登录、退出方法；
2. 掌握常用 Linux 文件操作和目录操作命令；

二、实验环境

1. 操作系统：客户端 Windows，服务器端 Linux
2. 主要软件：PuTTY

三、实验原理

1. 文件内容查看

（1）cat

[功能]

显示文件内容

[命令格式]

cat [option] [file]

[常用选项]

[其它用法]

新建文件：$cat <<EOF >file

合并文件：$cat file1 file2 >file3

管道用法：$cat file |sort

（2）zcat

[功能]

显示压缩文件内容

[命令格式]

zcat [option] [file]

[常用选项]

（3）head

[功能]

显示文件头部内容

[命令格式]

head [option] [file]

[常用选项]

-n number 或-number：显示前 number 行

（4）tail

[功能]

显示文件尾部内容

[命令格式]

tail [option] [file]

[常用选项]

-n number 或-number：显示最后 number 行

（5）more

[功能]
分页显示文件内容
[命令格式]
more [option] [file]
[常用选项]
（6）less
[功能]
分页显示文件内容（功能比 more 强大）
[命令格式]
less [option] [file]
[使用技巧]
①搜索

当使用命令 less file-name 打开一个文件后，可以使用下面的方式在文件中搜索。搜索时整个文本中匹配的部分会被高亮显示。

向前搜索

/ - 使用一个模式进行搜索，并定位到下一个匹配的文本

n - 向前查找下一个匹配的文本

N - 向后查找前一个匹配的文本

向后搜索

? - 使用模式进行搜索，并定位到前一个匹配的文本

n - 向后查找下一个匹配的文本

N - 向前查找前一个匹配的文本

②全屏导航

ctrl + F - 向前移动一屏

ctrl + B - 向后移动一屏

ctrl + D - 向前移动半屏

ctrl + U - 向后移动半屏

③单行导航

j - 向前移动一行

k - 向后移动一行

④其它导航

G - 移动到最后一行

g - 移动到第一行

q / ZZ - 退出 less 命令

⑤其它有用的命令

v - 使用配置的编辑器编辑当前文件

h - 显示 less 的帮助文档

&pattern - 仅显示匹配模式的行，而不是整个文件

2. 文件操作

（1）cp

[功能]

复制文件或目录

[命令格式]

cp [option] source_file target_file

[常用选项]

-r：复制目录

-f：如果目标文件已存在，不提示直接覆盖

-i：覆盖之前提示

（2）mv

[功能]

移动或重命名文件或目录

[命令格式]

mv [option] source_file target_file

[常用选项]

-f：如果目标文件已存在，不提示直接覆盖

-i：覆盖之前提示

（3）rm

[功能]

删除文件或目录

[命令格式]

rm [option] file

rm –r directory

[常用选项]

-f：如果目标文件已存在，不提示直接覆盖

-i：覆盖之前提示

-r：删除目录及其中的内容

（4）ln

[功能]

建立连接

[命令格式]

ln [option] file link

[常用选项]

-s：建立软连接

（5）touch

[功能]

修改文件或目录的时间戳

[命令格式]

touch [option] file
[常用选项]
-t stamp：使用时间（格式[[CC]YY]MMDDhhmm[.ss]）代替当前时间戳
[其它用法]
生成新的空文件（touch 后面的文件不存在的话）

（6）chown

[功能]
修改文件或目录的属主
[命令格式]
chown [option] user file
[常用选项]
-R：修改目录及其中的所有文件和目录的属主

（7）chmod

[功能]
修改文件或目录的权限
[命令格式]
chmod mode file
[常用选项]
-R：修改目录及其中的所有文件和目录的权限
[示例]
chmod 755 at_cds.fa
chmod +x blast_parser.pl
chmod go-w index.php

（8）locate

[功能]
通过文件名查找文件
[命令格式]
locate [option] patern
[常用选项]

（9）find

[功能]
查找文件（功能比 locate 强大）
[命令格式]
find [option] expression
[常用选项]
-type FILETYPE：查找类型为 FILETYPE 的文件
-name FILENAME：查找文件名为 FILENAME 的文件

3. 文本处理

（1）grep

[功能]
显示匹配特定模式的行
[命令格式]
grep [option] pattern file
[常用选项]
-E：使用扩展的正则表达式匹配
-c：只显示匹配的行数
-i：匹配时忽略大小写

（2）sort

[功能]

排序文件内容

[命令格式]

sort [option] file

[常用选项]

-k：设定排序的字段

-n：按数字大小（而不是 ASCII 码顺序）排序

-r：反向排序

[示例]

sort –k2,2 pt.gff

sort –k2,2n –k3,3nr pt.gff

（3）cut

[功能]

从文件的每一行中取出特定的列（默认为制表符分隔）

[命令格式]

cut [option] file

[常用选项]

-f：（后跟数字 n）取出第 n 列

-d：（后跟字符 x）定义列的界定符

-b：取出特定字节

-c：取出特定字符

[示例]

cut –f2 pt.gff

cut –d' ' –f3 pt_modified.gff

cut –b2-10 pt.gff

cut –c11- pt.gff

（4）paste

[功能]

按列合并文件

[命令格式]

paste [option] file1 file2

[常用选项]

-d：定义合并时的分隔符（默认为制表符）

（5）sed

[功能]

过滤或转换文本的流编辑器

[命令格式]

sed [option] command file

[常用选项]

[示例]

sed 1,4d pt.gff

sed s/A/a/g at.gff

（6）tr

[功能]

转换或删除字符

[命令格式]

cat file |tr pattern1 [pattern2]

[常用选项]
-d：删除 pattern1

（7）awk

[功能]
awk 本身是一门脚本语言，有控制结构及变量定义。但常见的用法为重新排列一个文件的列。

[命令格式]
awk program-text file

[常用选项]
-F：定义输入文件的列分隔符
-f：执行脚本文件，而不是执行 program-text 脚本

[示例]
awk -F'\t' -v OFS='\t' '{print $2,$3,$1}' pt.gff

（8）comm

[功能]
对两个已经排好序的文件进行比较。其中 file1 和 file2 是已排序的文件。comm 读取这两个文件，然后生成三列输出：仅在 file1 中出现的行；仅在 file2 中出现的行；在两个文件中都存在的行。如果文件名用"- "，则表示从标准输入读取。

[命令格式]
comm [-123] file1 file2

[常用选项]
-1
-2
-3

选项 1 、2 或 3 抑制相应的列显示。例如
comm - 12 就只显示在两个文件中都存在的行；
comm - 23 只显示在第一个文件中出现而未在第二个文件中出现的行；
comm - 123 则什么也不显示。

（9）diff

[功能]
逐行比较两个文本文件，列出其不同之处。它比 comm 命令完成更复杂的检查。它对给出的文件进行系统的检查，并显示出两个文件中所有不同的行，不要求事先对文件进行排序。结果为将文件 1 改成文件 2 需要的步骤。

[命令格式]
diff [option] file1 file2

[常用选项]

4. 目录操作

（1）ls

[功能]
显示目录内容

[命令格式]
ls [option] [dirs]

[常用选项]
-l：显示详细信息
-a：显示所有文件（包括隐藏文件）

（2）cd

[功能]
　　改变当前目录
[命令格式]
　　cd [dir]
[常用选项]

（3）mkdir
[功能]
　　新建目录
[命令格式]
　　mkdir [option] directory
[常用选项]
　　-p：在目录中新建目录
（4）rmdir
[功能]
　　删除空目录。如果目录中有文件或目录，该命令无效，如果要删除非空目录及其内容，需使用 rm –r。
[命令格式]
　　rmdir [option] empty-directory
[常用选项]
　　-p：删除目录及其父目录

四、实验内容

1. Linux 服务器的远程登录

（1）在 Windows 下运行 SSH 客户端程序 PuTTY；
（2）主机一栏填 www.linuxstudio.cn，端口用默认的 22，字符编码设置选 UTF-8；
（3）点击 Open 按钮，输入用户名（每个人在该服务器上的帐号）和密码（注意：输入过程不显示*）。

2. 文件内容查看

（1）在个人主目录下新建目录 linux（如果已有就忽略该步骤），并在其中新建目录 exp，在 exp 目录中新建目录 exp_1；
（2）将/home/pub/seq/目录下的文件 at_LEC1_CDS.fa 和 pt_partial.gff.gz 复制到 exp_1 目录中；
（3）分别用 cat、zcat、head、tail、more、less 查看两个文件内容，比较各程序的异同。

3. 文件及目录操作

（1）在目录 exp_1 下新建目录 tmp；
（2）将文件 at_LEC1_CDS.fa 复制到目录 tmp 中，并命名为 at_LEC1_CDS_backup.fa；
（3）将目录 tmp 复制到 exp_1 下，并命名为 tmp1；
（4）将目录 tmp 复制到 exp_1 下，并命名为 tmp2；
（5）使用 rmdir 删除目录 tmp1 和 tmp2，看是否能成功；
（6）删除 tmp1 中的文件 at_LEC1_CDS_backup.fa；
（7）再次使用 rmdir 删除目录 tmp1，看是否能成功；；
（8）使用 rm -r 删除目录 tmp2；
（9）在目录 exp_1 中，为文件 at_LEC1_CDS.fa 建立软连接和硬链接，比较二者的异同。

4. 文件内容处理

　　将文件 pt_partial.gff.gz 中包含 CDS 的行取出，并且只保留序列名、起始位置和终止位置 3 列，再按序列名大小升序、起始位置降序排列，利用 awk 将起始位置和终止位置放到 1、

2 列，序列名放到第三列，最后将结果保存到 pt_result。

五、实验报告

1. 实验环境（包括操作系统和软件）实验步骤，结果文件记录；
2. 实验中遇到的问题,如何解决的。

实验 2：Linux 常用命令（2）：系统管理命令

一、实验目的

1. 掌握帮助、进程管理、压缩解压缩、网络连接与文件传输及其它命令；

二、实验环境

1. 操作系统：客户端 Windows，服务器端 Linux
2. 主要软件：PuTTY

三、实验原理

1.帮助命令

（1）man

[功能]

查看命令说明

[命令格式]

man command

[常用选项]

（2）info

[功能]

查看命令说明（比 man 详细）

[命令格式]

man command

[常用选项]

2. 进程管理命令

（1）top

[功能]

显示 Linux 任务

[命令格式]

top

[常用选项]

（2）ps

[功能]

显示进程信息

[命令格式]

ps [option]

[常用选项]

-e：显示所有进程

[示例]

ps –e

ps –aux

（3）kill

[功能]

终止进程

[命令格式]
kill [option] process-id
[常用选项]
（4）sleep
[功能]
系统在一段时间内什么都不做
[命令格式]
sleep number
[常用选项]
（5）bg
[功能]
将挂起的进程放到后台运行。ctrl-z 可以将正在运行的进程挂起，恢复挂起的进程时，有两种选择：用 fg 命令将挂起的作业放回到前台执行；用 bg 命令将挂起的作业放到后台执行。
[命令格式]
bg
[常用选项]
（6）fg
[功能]
将在后台运行的进程放到前台。
[命令格式]
fg [position-of-suspended-process]
[常用选项]

3. 压缩、解压缩命令
（1）zip/unzip
[功能]
压缩/解压缩 zip 格式文件
[命令格式]
zip file.zip file
unzip zip-file
[常用选项]
（2）gzip/gunzip
[功能]
压缩/解压缩 gzip 格式文件
[命令格式]
gzip file
gunzip gzip-file
[常用选项]
（3）bzip2/bunzip2
[功能]
压缩/解压缩 bzip 格式文件
[命令格式]
bzip2 file
bunzip2 bzip-file
[常用选项]
（4）tar
[功能]

目录打包（或调用压缩程序压缩）
[命令格式]
tar [cxvzjf] directory
[常用选项]
[示例]
tar xjf at.bz2
tar czf at.tar.gz at

4. 网络连接与文件传输命令

（1）ssh

[功能]
远程登录 Linux 主机
[命令格式]
ssh [option] host
[常用选项]
-p：设定登陆端口
-X：允许传送图形
[示例]
ssh –p 443 10.10.10.10

（2）scp

[功能]
在两个 Linux 服务器之间传送文件或目录
[命令格式]
scp [option] file host:path
[常用选项]
-r：传送目录
-P：设定端口

（3）wget

[功能]
下载网页或文件
[命令格式]
wget [option] url
[常用选项]
-i：从文件中读取 url
-c：续传

（4）lftp

[功能]
登陆 ftp 服务器
[命令格式]
lftp [option] ftp-host
[常用选项]

5. 其他命令

（1）who

[功能]
显示系统登录用户信息
[命令格式]
who
[常用选项]

（2）w

[功能]

显示系统登录用户详细信息

[命令格式]

w

[常用选项]

（3）date

[功能]

显示或设定系统时间

[命令格式]

date [option]

date [MMDDhhmm[[CC]YY][.ss]]

[常用选项]

（4）cal

[功能]

显示当月日历

[命令格式]

cal

[常用选项]

（5）clear

[功能]

清空屏幕

[命令格式]

clear

[常用选项]

（6）passwd

[功能]

修改用户密码

[命令格式]

passwd [option] [user]

[常用选项]

（7）time

[功能]

计算程序运行所需时间

[命令格式]

time command

[常用选项]

（8）echo

[功能]

显示一行文本或变量内容

[命令格式]

echo [string|variable]

[常用选项]

-n：不显示换行符

四、实验内容

1. 帮助命令

分别用 man 和 info 查看常用命令的帮助文档

2. 进程管理命令

用 sleep 命令测试前台后台相关命令及进程管理命令：Ctrl-z、fg、bg、&、jobs、kill、top、ps。

3. 压缩解压缩命令

（1）在个人主目录下的 linux/exp 目录中新建目录 exp_2；

（2）将/home/pub/seq/at_NFY_protein.fa 复制到 exp_2 目录中；

（3）分别将 at_NFY_protein.fa 压缩成 zip、gzip 和 bzip 格式，再解压缩。

4. 网络连接与文件传输命令

ssh、scp、lftp、wget 命令

5. 其他命令

who、w、date、cal、clear、passwd、time、echo 命令

五、实验报告

1. 实验环境（包括操作系统和软件）实验步骤，结果文件记录；
2. 实验中遇到的问题,如何解决的。

实验 3：Vim 编辑器的使用

一、实验目的

1. 了解 Vim 编辑器的两种操作模式：命令模式和插入模式；
2. 掌握 Vim 编辑器的操作方法。

二、实验环境

1. 操作系统：客户端 Windows，服务器端 Linux
2. 主要软件：PuTTY

三、实验原理

（一）vi 的基本概念

文本编辑器有很多，图形模式下有 gedit、kwrite 等编辑器，文本模式下的编辑器有 vi、vim（vi 的增强版本）和 nano。vi 和 vim 是 Linux 系统中最常用的编辑器。

vi 编辑器是所有 Linux 系统的标准编辑器，用于编辑任何 ASCII 文本，对于编辑源程序尤其有用。它功能非常强大，通过使用 vi 编辑器，可以对文本进行创建、查找、替换、删除、复制和粘贴等操作。

vi 编辑器有 3 种基本工作模式，分别是命令模式、插入模式和末行模式。在使用时，一般将末行模式也算入命令行模式。各模式的功能区分如下。

1．命令行模式

控制屏幕光标的移动，字符、字或行的删除，移动、复制某区域及进入插入模式，或者到末行模式。

2．插入模式

只有在插入模式下才可以做文本输入，按“ESC”键可回到命令行模式。

3．末行模式

将文件保存或退出 vi 编辑器，也可以设置编辑环境，如寻找字符串、列出行号等。

（二）vi 的基本操作

1．进入 vi 编辑器

在系统 shell 提示符下输入 vi 及文件名称后，就进入 vi 编辑画面。如果系统内还不存在该文件，就意味着要创建文件；如果系统内存在该文件，就意味着要编辑该文件。下面就是用 vi 编辑器创建文件的示例。

#vi filename

进入 vi 之后，系统处于命令行模式，要切换到插入模式才能够输入文字。

2．切换至插入模式编辑文件

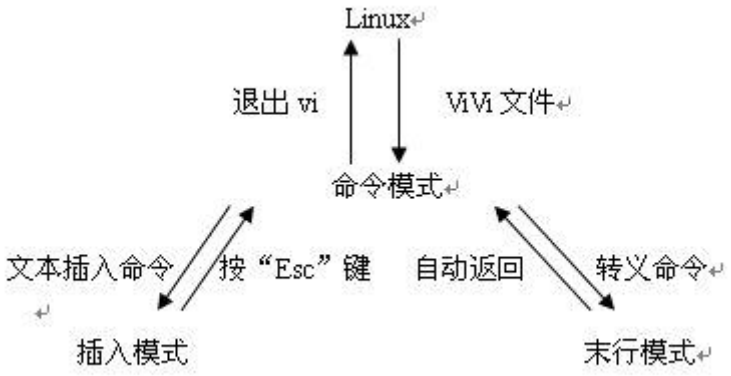
在命令行模式下按字母“i”就可以进入插入模式，这时候就可以开始输入文字了。

3．退出 vi 及保存文件

在命令行模式下，按冒号键“：”可以进入末行模式，例如：[:w filename]将文件内容以指定的文件名 filename 保存。

输入“wq”，存盘并退出 vi。输入“q!”，不存盘强制退出 vi。

下面表示 vi 编辑器的 3 种模式之间的关系。



（三）命令行模式操作

1．进入插入模式

按“i”：从光标当前位置开始输入文件。

按“a”：从目前光标所在位置的下一个位置开始输入文字。

按“o”：插入新的一行，从行首开始输入文字。

按“I”：在光标所在行的行首插入。

按“A”：在光标所在行的行末插入。

按“O”：在光标所在的行的下面插入一行。

按“s”：删除光标后的一个字符，然后进入插入模式。

按“S”：删除光标所在的行，然后进入插入模式。

2．从插入模式切换为命令行模式

按“ESC”键盘或 ctrl+c

3．移动光标

vi 可以直接用键盘上的光标来上下左右移动，但正规的 vi 是用小写英文字母“h”、“j”、“k”、“l”分别控制光标左、下、上、右移一格。

按“ctrl+b”：屏幕往后移动一页。

按“ctrl+f”：屏幕往前移动一页。

按“ctrl+u”：屏幕往后移动半页。

按“ctrl+d”：屏幕往前移动半页。

按数字“0”：移动到文本的开头。

按 “G”：移动到文件的最后。

按 “$”：移动到光标所在行的行尾。

按 “^”：移动到光标所在行的行首。

按 “w”：光标跳到下个字的开头。

按 “e”：光标跳到下个字的字尾。

按 “b”：光标回到上个字的开头。

按 “nl”：光标移动该行的第 n 个位置，例如：“5l”表示移动到该行的第 5 个字符。

4．删除文字

“x”：每按一次，删除光标所在位置的后面一个字符。

“nx”：例如：“6x”表示删除光标所在位置后面 6 个字符。

“X”：大写的 X，每按一次，删除光标所在位置的前面一个字符。

“xX”：例如：“20X”表示删除光标所在位置前面 20 个字符。

“dd”：删除光标所在行。

“ndd”：从光标所在行开始删除 n 行。例如：“4dd”表示删除从光标所在行开始的 4 行字符。

5．复制

“yw”：将光标所在之处到字尾的字符复制到缓冲区中。

“nyw”：复制 n 个字到缓冲区。

“yy”：复制光标所在行到缓冲区。

“nyy”：例如：“6yy”表示复制从光标所在行开始 6 行字符。

“p”：将缓冲区内的字符写到光标所在位置。

6．替换

“r”：替换光标所在处的字符。

“R”：替换光标所到处的字符，直到按下“ESC”键为止。

7．撤销上一次操作

“u”：如果误执行一个命令，可以马上按下“u”，回到上一个操作。按多次“u”可以执行多次撤销操作。

8．更改

“cw”：更改光标所在处的字到字尾处。

“cnw”：例如：“c3w”表示更改 3 个字。

9．跳至指定的行

“ctrl+g”：列出光标所在行的行号。

“nG”：例如：“15G”，表示移动光标到该文件的第 15 行行首。

10．存盘退出

“ZZ”：存盘退出

11．不存盘退出

“ZQ”：不存盘退出

（四）末行模式操作

在使用末行模式之前，请记住先按“ESC”键确定已经处于命令行模式后，再按冒号“：”即可进入末行模式。

1．列出行号

“set nu”：输入“set nu”后，会在文件中的每一行前面列出行号。

2．取消列出行号

“set nonu”：输入“set nonu”后，会取消在文件中的每一行前面列出行号。

3．搜索时忽略大小写

“set ic”：输入“set ic”后，会在搜索时忽略大小写。

4．取消搜索时忽略大小写

“set noic”：输入“set noic”后，会取消在搜索时忽略大小写。

5．跳到文件中的某一行

“n”：“n”表示一个数字，在冒号后输入一个数字，再按回车键就会跳到该行了，如输入数字 15,再回车就会跳到文本的第 15 行。

6．查找字符

“/关键字”：先按“/”，再输入想查找的字符，如果第一次查找的关键字不是想要的，可以一直按“n”，往后查找一个关键字。

“? 关键字”：先按“?”键，再输入想查找的字符，如果第一次查找的关键字不是想要的，可以一直按“？”，往后查找一个关键字。

7．运行 shell 命令

“!cmd”：运行 shell 命令 cmd。

8．替换字符

“s /SPARCH/REPLACE/g”：把当前光标所处的行中的 SEARCH 单词替换成

REPLACE，并把所有 SEARCH 高亮显示。

“%s /SPARCH/REPLACE”：把文档中所有 SEARCH 替换成 REPLACE。

“n1,n2 s /SPARCH/REPLACE/g”：n1、n2 表示数字，表示从 n1 行到 n2 行，把 SEARCH 替换成 REPLACE。

9．保存文件

“w”：在冒号输入字母“w”就可以将文件保存起来。

10．离开 vi

“q”：按“q”即退出 vi，如果无法离开 vi，可以在“q”后面一个“!”强制符离开 vi。

“qw”：一般建议离开时，搭配“w”一起使用，这样在退出的时候还可以保存文件。

（五）命令行内容说明

命令行模式：移动光标的方法

h 或向左方向键（←）：光标向左移动一个字符

j 或向下方向键（↓）：光标向下移动一个字符

k 或向上方向键（↑）：光标向上移动一个字符

l 或向右方向键（→）：光标向右移动一个字符

如果想要进行多次移动的话，例如；向下移动 30 行，可以使用“30j”或“30↓”的组合键，即加上想要进行的次数（数字）后，操作即可。

[Ctrl]+[f]：屏幕“向下”移动一页，相当于[Page Down]按键

[Ctrl]+[b]：屏幕“向上”移动一页，相当于[Page Up]按键

[Ctrl]+[d]：屏幕“向下”移动半页

[Ctrl]+[u]：屏幕“向上”移动半页

+：光标移动到非空格符的下一行

-：光标移动到非空格符的上一行

n<space>：n 表示“数字”，例如 20.按下数字后再按空格键，光标会向右移动这一行 n 个字符。例如 20<space>则光标会向后面移动 20 个字符距离

0：这是数字“0”：移动到这一行的最前面字符处（常用）

$：移动到这一行的最后面字符处（常用）

H：光标移动到屏幕的第一行

M：光标移动到屏幕的中间一行

L：光标移动到屏幕的最后一行

G：移动到文件的最后一行（常用）

nG：n 为数字。移动到这个文件的第 n 行。例如 20G 则会移动到这个文件的第 20 行（可配合：set nu）

gg：移动到这个文件的第一行，相当于 1G（常用）

n<Enter>：n 为数字。光标向下移动 n 行（常用）

命令行模式：搜索与替换

/word：从光标位置开始，向下寻找一个名为 word 的字符串。例如要在文件内搜索 vbird 这个字符串，就输入/vbird 即可（常用）

?word：从光标位置开始，向上寻找一个名为 word 的字符串

n：n 是英文按键。表示“重复前一个搜索的动作”。举例来说，如果刚刚执行/vbird 去向下搜索 vbird 字符串，则按下 n 后，会向下继续搜索下一个名称为 vbird 的字符串。如果是执行?vbird 的话，那么按下 n，则会向上继续搜索名称为 vbird 的字符串

N：这个 N 是英文按键。与 n 刚好相反，为“反向”进行前一个搜索操作。例如/vbird 后，按下 N 则表示“向上”搜索 vbird

:n1、n2s/word1/word2/g：n1 与 n2 为数字。在第 n1 与 n2 行之间寻找 word1 这个字符串，并将该字符串替换为 word2。举例来说，在 100 到 200 行之间搜索 vbird 并替换为 VBIRD 则：":100、200s/vbird/VBIRD/g"（常用）

:1、$s/word1/word2/g：从第一行到最后一行寻找 word1 字符串，并将该字符串替换为 word2（常用）

:1、$s/word1/word2/gc：从第一行到最后一行寻找 word1 字符串，并将该字符串替换为 word2。且在替换前显示提示符给用户确认（conform）是否需要替换（常用）

命令行模式：删除、复制与粘贴

p,P：p 为将已复制的数据粘贴到光标的下一行，P 则为贴在光标上一行。举例来说，当前光标在第 20 行，且已经复制了 10 行数据。则按下 p 后，那 10 行数据会粘在原来的 20 行之后，即由 21 行开始贴。但如果是按下 P，那么原来的第 20 行会被变成 30 行（常用）

J：将光标所在行与下一列的数据结合成同一行

c：重复删除多个数据，例如向下删除 10 行，[10cj]

u：复原前一个操作（常用）

[Ctrl]+r：重做上一个操作（常用）。U 与[Ctrl]+r 是很常用的命令。一个是复原，另一个则是重做一次。利用这两个功能按键，编辑起来就得心应手。

.：这就是不数点。意思是重复前一个动作。如果想重复删除、重复粘贴，按下小数点"."就可以（常用）

插入模式

i、I：插入：在当前光标所在处插入输入文字，已存在的文字会向后退；其中，i 为"从当前光标所在处插入"，I 为"在当前所在行的第一个非空格符处开始插入"（常用）

a、A：a 为"从当前光标所在的下一个字符处开始插入"，A 为"从光标所在行的最后一个字符处开始插入"（常用）

o、O：这是英文字母 o 的大小写。o 为"在当前光标所在的下一行处插入新的一行"，O 为"在当前光标所在处的上一行插入新的一行"（常用）

r、R：替换：r 会替换光标所在的那一个字符；R 会一直替换光标所在的文字，直到按下 Esc 键为止（常用）

使用上面这些按键时，在 vi 画面的左下角处会出现"—INSERT--"或"—REPLACE--"的字样。通过名称就知道是什么操作。特别注意，上面也提过了，想在文件中输入字符时，一定要在左下角处看到 INSERT/ REPLACE 才能输入。

Esc：退出插入模式，回到命令行模式中（常用）

末行命令模式

:w：将编辑的数据写入硬盘文件中（常用）

:w!：若文件属性为"只读"时，强制写入该文件。不过，到底能不能写入，与文件权限有关

:q：离开 vi（常用）

:q!：若曾修改过文件，又不想存储，使用！为强制离开不存储文件。注意一下，那个感叹号（!）在 vi 当中，常常具有"强制"的意思。

:wq：存储后离开，若为:wq!则为强制存储后离开（常用）

:e!：将文件还原到最原始的状态

ZZ：若文件没有更改，则不存储离开，若文件已经更改，则存储后离开

:w[filename]：将编辑的数据存储成另一个文件（类似另存新文件）

:r[filename]：在编辑的数据中，读入另一个文件的数据。即将"filename"这个文件内容加到光标所在行的后面

:n1、n2 w[filename]：将 n1 到 n2 的内容存储成 filename 文件

:!command：暂时离开 vi 到命令模式下执行 command 的显示结果。例如，“:! ls /home”，即可在 vi 中查看/home 中以 ls 输出的文件信息

:set nu：显示行号，设置之后，会在每一行的前缀显示该行的行号

:set nonu：与 set nu 相反，为取消行号

特别注意，在 vi 中，“数字”是很有意义的。数字通常表示重复做几次的意思。也有可能表示要去哪里的意思。举例来说，要删除 50 行，则是用“50dd”。数字加在动作之前。要向下移动 20 行，使用“20j”或“20↓”即可。

（http://hi.baidu.com/eao110/blog/item/0e2074f08fd3dfd77831aa0d.html）

四、实验内容

1. 在自己的主目录下的 linux/exp 目录中新建目录 exp_3，在其中完成下面的任务；
2. 用 Vim 编辑器写一个 C 语言的“Hello world!”程序 hello.c，并编译、运行。注意在 Vim 中不要使用方向键（箭头）移动光标；
3. 用 Vim 编辑器写一个 Python 语言的“Hello world!”程序 hello.py 并运行；
4. 比较脚本语言（Python）与编译语言（C）有哪些不同。

五、实验报告

1. 实验环境（包括操作系统和软件）实验步骤，结果文件记录；
2. 实验中遇到的问题,如何解决的。

实验 4：Shell 程序设计基础（1）：变量与特殊字符

一、实验目的

1. 掌握 Linux 环境变量的设置方法；
2. 了解 Shell 特殊字符的含义；
3. 掌握 Shell 变量的概念及其使用方法。

二、实验环境

1. 操作系统：客户端 Windows，服务器端 Linux
2. 主要软件：PuTTY

三、实验原理

1. 变量简介

Linux 是一个多用户的操作系统。每个用户登录系统后，都会有一个专用的运行环境。通常每个用户默认的环境都是相同的，这个默认环境实际上就是一组环境变量的定义。用户可以对自己的运行环境进行定制，其方法就是修改相应的系统环境变量。

2. 常见的环境变量

$PATH：决定了 shell 将到哪些目录中寻找命令或程序

$HOME：当前用户主目录

$MAIL：是指当前用户的邮件存放目录。

$SHELL：是指当前用户用的是哪种 Shell。

$HISTSIZE：是指保存历史命令记录的条数

$LOGNAME：是指当前用户的登录名。

$HOSTNAME：是指主机的名称，许多应用程序如果要用到主机名的话，通常是从这个环境变量中来取得的。

$LANG/LANGUGE：是和语言相关的环境变量，使用多种语言的用户可以修改此环境变量。

$PS1：是基本提示符，对于 root 用户是#，对于普通用户是$，也可以使用一些更复杂的值。

$PS2：是附属提示符，默认是“>”。可以通过修改此环境变量来修改当前的命令符，比如下列命令会将提示符修改成字符串“Hello,My NewPrompt :) ”。

```
# PS1=" Hello,My NewPrompt :) "
```

$IFS：输入域分隔符。当 shell 读取输入时，用来分隔单词的一组字符，它们通常是空格、制表符和换行符。

$0：shell 脚本的名字。

例如，在我的 Linux 系统中：

```
$ echo $0
/bin/bash
```

$#：传递给脚本的参数个数。

$$：shell 脚本的进程号，脚本程序通常会用它生成一个唯一的临时文件，如 /tmp/tmfile_$$

例如，在我的 Linux 系统中：

```
$ echo $$
31038                    #表示当前 shell 进程号为 31038
```

3. export 命令

export 命令将作为他参数的变量导入到子 shell 中，并使之在子 shell 中有效。export 命令把自己的参数创建为一个环境变量，而这个环境变量可以被其他脚本和当前程序调用的程序看见。如

```
export PATH=$PATH:/usr/local/bin
```

4. 定制环境变量

环境变量是和 Shell 紧密相关的，用户登录系统后就启动了一个 Shell 。对于 Linux 来说一般是 bash ，但也可以重新设定或切换到其它的 Shell 。根据发行版本的情况，bash 有两个基本的系统级配置文件： /etc/bashrc 和 /etc/profile 。这些配置文件包含两组不同的变量： shell 变量和环境变量。前者只是在特定的 shell 中固定（如 bash ），后者在不同 shell 中固定。很明显， shell 变量是局部的，而环境变量是全局的。环境变量是通过 Shell 命令来设置的，设置好的环境变量又可以被所有当前用户所运行的程序所使用。对于 bash 这个 Shell 程序来说，可以通过变量名来访问相应的环境变量，通过 export 来设置环境变量。下面通过几个实例来说明。

4.1 使用命令 echo 显示环境变量

```
#本例使用 echo 显示常见的变量 HOME
$ echo $HOME
/home/lqm
```

4.2 设置一个新的环境变量

```
$ export HELLO="Hello"
$ echo $HELLO
Hello!
```

4.3 使用 env 命令显示所有的环境变量

```
$ env
SSH_AGENT_PID=1875
HOSTNAME=lqm
SHELL=/bin/bash
TERM=xterm
HISTSIZE=1000
……
```

4.4 使用 set 命令显示所有本地定义的 Shell 变量

```
$ set
BASH=/bin/bash
……
```

4.5 使用 unset 命令来清除环境变量

```
$ export TEST="test"          # 增加一个环境变量 TEST
```

```
$ env | grep TEST                # 此命令有输出，证明环境变量 TEST 已经存在了
TEST=test
$ unset TEST                      # 删除环境变量 TEST
$ env | grep TEST                # 此命令无输出，证明环境变量 TEST 已经不存在了
```

4.6 使用 readonly 命令设置只读变量

如果使用了 readonly 命令的话，变量就不可以被修改或清除了。示例如下：

```
$ export TEST="Test"              # 增加一个环境变量 TEST
$ readonly TEST                   # 将环境变量 TEST 设为只读
$ unset TEST                      # 会发现此变量不能被删除
-bash: unset: TEST: cannot unset: readonly variable
$ TEST="New" #会发现此变量不能被修改
-bash: TEST: readonly variable
```

4.7 用 C 程序来访问和设置环境变量

对于 C 程序的用户来说，可以使用下列三个函数来设置或访问一个环境变量。

getenv() 访问一个环境变量。输入参数是需要访问的变量名字，返回值是一个字符串。如果所访问的环境变量不存在，则会返回 NULL 。

setenv() 在程序里面设置某个环境变量的函数。

unsetenv() 清除某个特定的环境变量的函数。

另外，还有一个指针变量 environ ，它指向的是包含所有的环境变量的一个列表。下面的程序可以打印出当前运行环境里面的所有环境变量：

```
#include <stdio.h>
extern char**environ;
int main ()
{
   char**var;
   for (var =environ;*var !=NULL;++var)
   printf ("%s \n ",*var);
   return 0;
}
```

4.8 通过修改环境变量定义文件来修改环境变量。

需要注意的是，一般情况下，这仅仅对于普通用户适用，避免修改根用户的环境定义文件，因为那样可能会造成潜在的危险。

```
$vi /etc/bashrc          #修改 shell 变量
$vi /etc/profile        #修改环境变量定义文件
```

然后编辑你的 PATH 声明，其格式为：

PATH=$PATH:<PATH 1>:<PATH 2>:<PATH 3>:------:<PATH N>

你可以自己加上指定的路径，中间用冒号隔开。环境变量更改后，在用户下次登陆时生效，如果想立刻生效，则可执行下面的语句：$source .bash_profile

需要注意的是，最好不要把当前路径 “./” 放到 PATH 里，这样可能会受到意想不到的攻击。完成后，可以通过 $echo $PATH 查看当前的搜索路径。这样定制后，就可以避免频

繁的启动位于 shell 搜索的路径之外的程序了。

5. Shell 特殊字符（具体内容参考课本）

5.1 通配符

（1）*（星号）

（2）？（问号）

（3）[字符组]

（4）！（感叹号）

5.2 引号

（1）双引号

（2）单引号

（3）反引号

5.3 输入输出重定向符号

（1）输入重定向：<

（2）输出重定向：>

（3）输出附加重定向：>>

（4）即时文件重定向：<<

（5）与文件描述符有关的重定向：n>

5.4 注释、管道和后台命令

（1）注释：#

（2）管道：|

（3）后台命令：&

5.5 命令执行操作符

（1）顺序执行：换行和分号（;）

（2）逻辑与：cmd1 && cmd2

（3）逻辑或：cmd1 || cmd2

5.6 成组命令

（1）{ cmd1; cmd2; ……; }

（2）(cmd1; cmd2; ……)

四、实验内容

1. 查看自己的所有环境变量；
2. 在自己的主目录下建目录 bin（已有的略过），将该目录添加到环境变量 PATH 中。写一个脚本，并添加执行权限，检查该脚本能否像其它系统命令一样使用；
3. 测试各种特殊字符的作用。

五、实验报告

1. 实验环境（包括操作系统和软件）实验步骤，结果文件记录；
2. 实验中遇到的问题,如何解决的。

实验 5：Shell 程序设计基础（2）：控制结构

一、实验目的

1. 了解程序设计控制结构的概念；
2. 掌握 Shell 编程常用的控制结构。

二、实验环境

1. 操作系统：客户端 Windows，服务器端 Linux
2. 主要软件：PuTTY

三、实验原理

控制结构分顺序结构、判断和循环三类。

（一）顺序结构

Shell 顺序结构包括换行和分号（;）两种，比较简单，不再详述。

（二）判断结构

Shell 判断结构分为 if 和 case 两种。

1. if 结构

```
if 条件表达式
then 命令表
[elif 表达式
then 命令表]
[else 命令表]
fi
```

可以把 if 和 then 放在同一行，用分号“;”分隔：

```
if 表达式; then
命令表
fi
```

条件表达式可以是[ ]、test、[[ ]]或命令，返回值为 0 的表达式为真，返回值为其他值的表达式为假。

2. case 结构

```
case 表达式 in
   模式 11 [ |模式 12 ]...) 命令表 1;;
   模式 21 [ |模式 22 ]...) 命令表 2;;
   ...
        *) 命令表 n;;
esac
```

每个分支以右括号“)”分隔模式与命令，管道符“|”分隔同一分支的各个模式，表示或，两个

分号“;;”表示分支结束。星号“*”表示除以上模式之外的情况。

（三）循环结构

Shell 循环结构包括 for、while、until 和 select 四种。

1. for 循环

```
for 变量 [in 列表]
do
   命令表
done
```

如果省略了列表，则隐含表示为“for 变量 in $@”，即位置参数列表。bash 里的 for 不能设定循环执行的次数。

2 while 结构

```
while 表达式
do
   命令表
done
```

3. until 循环

```
until
命令表 1
test 表达式
do
   命令表 2
done
```

命令表 1 在循环开始前执行，测试如果为假，则进入循环，执行命令表 2。之后再次执行命令表 1，测试…… 直到测试结果为真，终止循环。

4. select 结构

select 结构生成一个数字化的菜单，并提示用户进行选择，默认提示符为“#”。用户只需在提示符下输入对应菜单项的数字，即可完成选择。

```
select 变量 in 列表
do
   命令表
done
```

例：

```
select N in one two three
do
   case $N in
      one) echo I;;
```

```
        two) echo II;;
        three) echo III;;
        *) echo wrong
           break;
    esac
done
```

上面的代码会显示菜单：

```
1) one
2) two
3) three
#?
```

循环结构的循环体以 do 开始，以 done 结束。各循环结构均支持用 continue 和 break 跳出循环，其中 continue 跳过当前循环，继续下一次循环；break 跳出整个循环。

四、实验内容

1. 在自己的主目录下的 linux/exp 目录中新建目录 exp_5；
2. 在目录 exp_5 中编写 Shell 脚本 score.sh，实现下列功能：利用命令行参数提供给程序一个整数，程序判断：

    0-59 分，提示不及格；
    60-69 分，提示成绩为及格；
    70-79 分，提示成绩为中；
    80-89 分，提示成绩为良；
    90-100 分，提示成绩为优；
    其它，提示超出范围。
    要求程序运行时能判断是否提供了命令行参数，没有参数提示出错。

五、实验报告

1. 实验环境（包括操作系统和软件）实验步骤，结果文件记录；
2. 实验中遇到的问题,如何解决的。

实验 6：Shell 程序设计基础（3）：程序调试

一、实验目的

1. 掌握 Shell 程序的调试方法和技巧。

二、实验环境

1. 操作系统：客户端 Windows，服务器端 Linux
2. 主要软件：PuTTY

三、实验原理

（一） 在 shell 脚本中输出调试信息

通过在程序中加入调试语句把一些关键地方或出错的地方的相关信息显示出来是最常见的调试手段。Shell 程序员通常使用 echo(ksh 程序员常使用 print)语句输出信息，但仅仅依赖 echo 语句的输出跟踪信息很麻烦，调试阶段在脚本中加入的大量的 echo 语句在产品交付时还得再费力一一删除。针对这个问题，本节主要介绍一些如何方便有效的输出调试信息的方法。

1. 使用 trap 命令

trap 命令用于捕获指定的信号并执行预定义的命令。其基本的语法是:

```
trap 'command' signal
```

其中 signal 是要捕获的信号，command 是捕获到指定的信号之后，所要执行的命令。可以用 kill –l 命令看到系统中全部可用的信号名，捕获信号后所执行的命令可以是任何一条或多条合法的 shell 语句，也可以是一个函数名。

shell 脚本在执行时，会产生三个所谓的“伪信号”，(之所以称之为“伪信号”是因为这三个信号是由 shell 产生的，而其它的信号是由操作系统产生的)，通过使用 trap 命令捕获这三个“伪信号”并输出相关信息对调试非常有帮助。

表 1. shell 伪信号

信号名	何时产生
EXIT	从一个函数中退出或整个脚本执行完毕
ERR	当一条命令返回非零状态时(代表命令执行不成功)
DEBUG	脚本中每一条命令执行之前

通过捕获 EXIT 信号,我们可以在 shell 脚本中止执行或从函数中退出时，输出某些想要跟踪的变量的值，并由此来判断脚本的执行状态以及出错原因,其使用方法是：

```
trap 'command' EXIT    或    trap 'command' 0
```

通过捕获 ERR 信号,我们可以方便的追踪执行不成功的命令或函数，并输出相关的调试信息，以下是一个捕获 ERR 信号的示例程序，其中的$LINENO 是一个 shell 的内置变量，代表 shell 脚本的当前行号。

```
$ cat -n exp1.sh
     1  ERRTRAP()
     2  {
```

```
 3      echo "[LINE:$1] Error: Command or function exited with status $?"
 4  }
 5  foo()
 6  {
 7      return 1;
 8  }
 9  trap 'ERRTRAP $LINENO' ERR
10  abc
11  foo
```

其输出结果如下：

```
$ sh exp1.sh
exp1.sh: line 10: abc: command not found
[LINE:10] Error: Command or function exited with status 127
[LINE:11] Error: Command or function exited with status 1
```

在调试过程中，为了跟踪某些变量的值，我们常常需要在 shell 脚本的许多地方插入相同的 echo 语句来打印相关变量的值，这种做法显得烦琐而笨拙。而通过捕获 DEBUG 信号，我们只需要一条 trap 语句就可以完成对相关变量的全程跟踪。

以下是一个通过捕获 DEBUG 信号来跟踪变量的示例程序:

```
$ cat –n exp2.sh
 1  #!/bin/bash
 2  trap 'echo “before execute line:$LINENO, a=$a,b=$b,c=$c”' DEBUG
 3  a=1
 4  if [ "$a" -eq 1 ]
 5  then
 6      b=2
 7  else
 8      b=1
 9  fi
10  c=3
11  echo "end"
```

其输出结果如下：

```
$ sh exp2.sh
before execute line:3, a=,b=,c=
before execute line:4, a=1,b=,c=
before execute line:6, a=1,b=,c=
before execute line:10, a=1,b=2,c=
before execute line:11, a=1,b=2,c=3
end
```

从运行结果中可以清晰的看到每执行一条命令之后，相关变量的值的变化。同时，从运行结果中打印出来的行号来分析，可以看到整个脚本的执行轨迹，能够判断出哪些条件分支执行了，哪些条件分支没有执行。

2. 使用 tee 命令

在 shell 脚本中管道以及输入输出重定向使用得非常多，在管道的作用下，一些命令的执行结果直接成为了下一条命令的输入。如果我们发现由管道连接起来的一批命令的执行结果并非如预期的那样，就需要逐步检查各条命令的执行结果来判断问题出在哪儿，但因为使用了管道，这些中间结果并不会显示在屏幕上，给调试带来了困难，此时我们就可以借助于 tee 命令了。

tee 命令会从标准输入读取数据，将其内容输出到标准输出设备,同时又可将内容保存成文件。例如有如下的脚本片段，其作用是获取本机的 ip 地址：

```
ipaddr=`/sbin/ifconfig | grep 'inet addr:' | grep -v '127.0.0.1'
| cut -d : -f3 | awk '{print $1}'`
#注意=号后面的整句是用反引号(数字 1 键的左边那个键)括起来的。
echo $ipaddr
```

运行这个脚本，实际输出的却不是本机的 ip 地址，而是广播地址,这时我们可以借助 tee 命令，输出某些中间结果，将上述脚本片段修改为：

```
ipaddr=`/sbin/ifconfig | grep 'inet addr:' | grep -v '127.0.0.1'
| tee temp.txt | cut -d : -f3 | awk '{print $1}'`
echo $ipaddr
```

之后，将这段脚本再执行一遍，然后查看 temp.txt 文件的内容：

```
$ cat temp.txt
inet addr:192.168.0.1    Bcast:192.168.0.255    Mask:255.255.255.0
```

我们可以发现中间结果的第二列(列之间以:号分隔)才包含了 IP 地址，而在上面的脚本中使用 cut 命令截取了第三列，故我们只需将脚本中的 cut -d : -f3 改为 cut -d : -f2 即可得到正确的结果。

具体到上述的 script 例子，我们也许并不需要 tee 命令的帮助，比如我们可以分段执行由管道连接起来的各条命令并查看各命令的输出结果来诊断错误，但在一些复杂的 shell 脚本中，这些由管道连接起来的命令可能又依赖于脚本中定义的一些其它变量，这时我们想要在提示符下来分段运行各条命令就会非常麻烦了，简单地在管道之间插入一条 tee 命令来查看中间结果会更方便一些。

3. 使用"调试钩子"

在 C 语言程序中，我们经常使用 DEBUG 宏来控制是否要输出调试信息，在 shell 脚本中我们同样可以使用这样的机制，如下列代码所示：

```
if [ “$DEBUG” = “true” ]; then
echo “debugging”    #此处可以输出调试信息
fi
```

这样的代码块通常称之为“调试钩子”或“调试块”。在调试钩子内部可以输出任何您想输出的调试信息，使用调试钩子的好处是它是可以通过 DEBUG 变量来控制的，在脚本的开发调试阶段，可以先执行 export DEBUG=true 命令打开调试钩子，使其输出调试信息，而在把脚本

交付使用时，也无需再费事把脚本中的调试语句一一删除。

如果在每一处需要输出调试信息的地方均使用 if 语句来判断 DEBUG 变量的值，还是显得比较繁琐，通过定义一个 DEBUG 函数可以使植入调试钩子的过程更简洁方便，如下面代码所示:

```
$ cat –n exp3.sh
     1  DEBUG()
     2  {
     3  if [ "$DEBUG" = "true" ]; then
     4          $@
     5  fi
     6  }
     7  a=1
     8  DEBUG echo "a=$a"
     9  if [ "$a" -eq 1 ]
    10  then
    11          b=2
    12  else
    13          b=1
    14  fi
    15  DEBUG echo "b=$b"
    16  c=3
    17  DEBUG echo "c=$c"
```

在上面所示的 DEBUG 函数中，会执行任何传给它的命令，并且这个执行过程是可以通过 DEBUG 变量的值来控制的，我们可以把所有跟调试有关的命令都作为 DEBUG 函数的参数来调用，非常的方便。

（二） 使用 shell 的执行选项

上一节所述的调试手段是通过修改 shell 脚本的源代码，令其输出相关的调试信息来定位错误的，那有没有不修改源代码来调试 shell 脚本的方法呢？答案就是使用 shell 的执行选项，本节将介绍一些常用选项的用法：

-n 只读取 shell 脚本，但不实际执行
-x 进入跟踪方式，显示所执行的每一条命令
-c "string" 从 strings 中读取命令

“-n”可用于测试 shell 脚本是否存在语法错误，但不会实际执行命令。在 shell 脚本编写完成之后，实际执行之前，首先使用“-n”选项来测试脚本是否存在语法错误是一个很好的习惯。因为某些 shell 脚本在执行时会对系统环境产生影响，比如生成或移动文件等，如果在实际执行才发现语法错误，您不得不手工做一些系统环境的恢复工作才能继续测试这个脚本。

“-c”选项使 shell 解释器从一个字符串中而不是从一个文件中读取并执行 shell 命令。当需要临时测试一小段脚本的执行结果时，可以使用这个选项，如下所示：

```
sh -c 'a=1;b=2;let c=$a+$b;echo "c=$c"'
```

"-x"选项可用来跟踪脚本的执行，是调试 shell 脚本的强有力工具。“-x”选项使 shell 在执行脚

本的过程中把它实际执行的每一个命令行显示出来，并且在行首显示一个"+"号。 "+"号后面显示的是经过了变量替换之后的命令行的内容，有助于分析实际执行的是什么命令。 “-x”选项使用起来简单方便，可以轻松对付大多数的 shell 调试任务,应把其当作首选的调试手段。

如果把本文前面所述的 trap ‘command’ DEBUG 机制与“-x”选项结合起来，我们就可以既输出实际执行的每一条命令，又逐行跟踪相关变量的值，对调试相当有帮助。

仍以前面所述的 exp2.sh 为例，现在加上“-x”选项来执行它：

```
$ sh –x exp2.sh
+ trap 'echo "before execute line:$LINENO, a=$a,b=$b,c=$c"' DEBUG
++ echo 'before execute line:3, a=,b=,c='
before execute line:3, a=,b=,c=
+ a=1
++ echo 'before execute line:4, a=1,b=,c='
before execute line:4, a=1,b=,c=
+ '[' 1 -eq 1 ']'
++ echo 'before execute line:6, a=1,b=,c='
before execute line:6, a=1,b=,c=
+ b=2
++ echo 'before execute line:10, a=1,b=2,c='
before execute line:10, a=1,b=2,c=
+ c=3
++ echo 'before execute line:11, a=1,b=2,c=3'
before execute line:11, a=1,b=2,c=3
+ echo end
end
```

在上面的结果中，前面有“+”号的行是 shell 脚本实际执行的命令，前面有“++”号的行是执行 trap 机制中指定的命令，其它的行则是输出信息。

shell 的执行选项除了可以在启动 shell 时指定外，亦可在脚本中用 set 命令来指定。 "set -参数"表示启用某选项，"set +参数"表示关闭某选项。有时候我们并不需要在启动时用"-x"选项来跟踪所有的命令行，这时我们可以在脚本中使用 set 命令，如以下脚本片段所示：

```
set -x          #启动"-x"选项
要跟踪的程序段
set +x          #关闭"-x"选项
```

set 命令同样可以使用上一节中介绍的调试钩子—DEBUG 函数来调用，这样可以避免脚本交付使用时删除这些调试语句的麻烦，如以下脚本片段所示：

```
DEBUG set -x          #启动"-x"选项
要跟踪的程序段
DEBUG set +x          #关闭"-x"选项
```

（三） 对"-x"选项的增强

"-x"执行选项是目前最常用的跟踪和调试 shell 脚本的手段，但其输出的调试信息仅限于进行变量替换之后的每一条实际执行的命令以及行首的一个"+"号提示符，居然连行号这样的重要信息都没有，对于复杂的 shell 脚本的调试来说，还是非常的不方便。幸运的是，我们可以巧妙地利用 shell 内置的一些环境变量来增强"-x"选项的输出信息，下面先介绍几个 shell 内置的环境变量：

$LINENO
代表 shell 脚本的当前行号，类似于 C 语言中的内置宏__LINE__

$FUNCNAME
函数的名字，类似于 C 语言中的内置宏__func__,但宏__func__只能代表当前所在的函数名，而$FUNCNAME 的功能更强大，它是一个数组变量，其中包含了整个调用链上所有的函数的名字，故变量${FUNCNAME[0]}代表 shell 脚本当前正在执行的函数的名字，而变量${FUNCNAME[1]}则代表调用函数${FUNCNAME[0]}的函数的名字，余者可以依此类推。

$PS4
主提示符变量$PS1 和第二级提示符变量$PS2 比较常见，但很少有人注意到第四级提示符变量$PS4 的作用。我们知道使用“-x”执行选项将会显示 shell 脚本中每一条实际执行过的命令，而$PS4 的值将被显示在“-x”选项输出的每一条命令的前面。在 Bash Shell 中，缺省的$PS4 的值是"+"号。(现在知道为什么使用"-x"选项时，输出的命令前面有一个"+"号了吧？)。

利用$PS4 这一特性，通过使用一些内置变量来重定义$PS4 的值，我们就可以增强"-x"选项的输出信息。例如先执行 export PS4='+{$LINENO:${FUNCNAME[0]}} ', 然后再使用“-x”选项来执行脚本，就能在每一条实际执行的命令前面显示其行号以及所属的函数名。

以下是一个存在 bug 的 shell 脚本的示例，本文将用此脚本来示范如何用“-n”以及增强的“-x”执行选项来调试 shell 脚本。这个脚本中定义了一个函数 isRoot(),用于判断当前用户是不是 root 用户，如果不是，则中止脚本的执行

```
$ cat –n exp4.sh
     1  #!/bin/bash
     2  isRoot()
     3  {
     4          if [ "$UID" -ne 0 ]
     5                  return 1
     6          else
     7                  return 0
     8          fi
     9  }
    10  isRoot
    11  if ["$?" -ne 0 ]
    12  then
    13          echo "Must be root to run this script"
    14          exit 1
    15  else
    16          echo "welcome root user"
    17          #do something
    18  fi
```

首先执行 sh –n exp4.sh 来进行语法检查，输出如下：

```
$ sh –n exp4.sh
exp4.sh: line 6: syntax error near unexpected token `else'
exp4.sh: line 6: `            else'
```

发现了一个语法错误，通过仔细检查第 6 行前后的命令，我们发现是第 4 行的 if 语句缺少 then 关键字引起的(写惯了 C 程序的人很容易犯这个错误)。我们可以把第 4 行修改为 if ["$UID" -ne 0]; then 来修正这个错误。再次运行 sh –n exp4.sh 来进行语法检查，没有再报告错误。接下来就可以实际执行这个脚本了，执行结果如下：

```
$ sh exp4.sh
exp2.sh: line 11: [1: command not found
welcome root user
```

尽管脚本没有语法错误了，在执行时却又报告了错误。错误信息还非常奇怪“[1: command not found”。现在我们可以试试定制$PS4 的值，并使用“-x”选项来跟踪：

```
$ export PS4='+{$LINENO:${FUNCNAME[0]}} '
$ sh –x exp4.sh
+{10:} isRoot
+{4:isRoot} '[' 503 -ne 0 ']'
+{5:isRoot} return 1
+{11:} '[1' -ne 0 ']'
exp4.sh: line 11: [1: command not found
+{16:} echo 'welcome root user'
welcome root user
```

从输出结果中，我们可以看到脚本实际被执行的语句，该语句的行号以及所属的函数名也被打印出来，从中可以清楚的分析出脚本的执行轨迹以及所调用的函数的内部执行情况。由于执行时是第 11 行报错，这是一个 if 语句，我们对比分析一下同为 if 语句的第 4 行的跟踪结果：

```
+{4:isRoot} '[' 503 -ne 0 ']'
+{11:} '[1' -ne 0 ']'
```

可知由于第 11 行的[号后面缺少了一个空格，导致[号与紧挨它的变量$?的值 1 被 shell 解释器看作了一个整体，并试着把这个整体视为一个命令来执行，故有“[1: command not found”这样的错误提示。只需在[号后面插入一个空格就一切正常了。

shell 中还有其它一些对调试有帮助的内置变量，比如在 Bash Shell 中还有 BASH_SOURCE, BASH_SUBSHELL 等一批对调试有帮助的内置变量，您可以通过 man sh 或 man bash 来查看，然后根据您的调试目的,使用这些内置变量来定制$PS4，从而达到增强“-x”选项的输出信息的目的。

（四） 总结

调试 shell 脚本的过程：

首先使用“-n”选项检查语法错误，然后使用“-x”选项跟踪脚本的执行，使用“-x”选项之

前，别忘了先定制 PS4 变量的值来增强“-x”选项的输出信息，至少应该令其输出行号信息(先执行 export PS4='+[$LINENO]'，更一劳永逸的办法是将这条语句加到您用户主目录的.bash_profile 文件中去)，这将使你的调试之旅更轻松。也可以利用 trap,调试钩子等手段输出关键调试信息，快速缩小排查错误的范围，并在脚本中使用“set -x”及“set +x”对某些代码块进行重点跟踪。这样多种手段齐下，相信您已经可以比较轻松地抓出您的 shell 脚本中的臭虫了。如果您的脚本足够复杂，还需要更强的调试能力，可以使用 shell 调试器 bashdb，这是一个类似于 GDB 的调试工具，可以完成对 shell 脚本的断点设置，单步执行，变量观察等许多功能，使用 bashdb 对阅读和理解复杂的 shell 脚本也会大有裨益。关于 bashdb 的安装和使用，可参阅 http://bashdb.sourceforge.net/上的文档并下载试用。
（http://www.ibm.com/developerworks/cn/linux/l-cn-shell-debug/）

四、实验内容

1. 在自己的主目录下的 linux/exp 目录中新建目录 exp_6；
2. 在目录 exp_6 中编辑 shell 脚本程序 int_sum.sh，内容如下，并用上述方法调试。

```
$ cat int_sum.sh
#!/bin/sh
if [ $# -eq 0 ]
then
        echo "Usage: ./ex21.sh integers"
        exit 1
fi
sum=0
until [ $# -eq 0 ]
do
        ((sum=$sum+$1))
        shift
done
echo $sum
```

如利用 bash 的-x 选项调试：

```
$ bash -x ex21.sh 2 3 4
+ '[' 3 -eq 0 ']'
+ sum=0
+ '[' 3 -eq 0 ']'
+ (( sum=0+2 ))
+ shift
+ '[' 2 -eq 0 ']'
+ (( sum=2+3 ))
+ shift
+ '[' 1 -eq 0 ']'
+ (( sum=5+4 ))
+ shift
+ '[' 0 -eq 0 ']'
+ echo 9
9
```

五、实验报告

1. 实验环境（包括操作系统和软件）实验步骤，结果文件记录；
2. 实验中遇到的问题,如何解决的。

实验 7：Shell 程序开发（1）：GFF 文件处理

一、实验目的

1. 了解 GFF 文件的概念和特点；
2. 掌握 Shell 脚本处理 GFF 文件的方法。

二、实验环境

1. 操作系统：客户端 Windows，服务器端 Linux
2. 主要软件：PuTTY

三、实验原理

GFF 格式由 Sanger 研究所定义，是一种简单的、方便的对于 DNA、RNA 以及蛋白质序列的特征进行描述的一种数据格式，比如序列的哪里到哪里是基因。GFF 已经成为序列注释的通用格式，比如基因组的基因预测，许多软件都支持输入或者输出 GFF 格式。目前格式定义的最新版本是版本 3。原始定义见 http://www.sequenceontology.org/gff3.shtml。

GFF 是纯文本文件，由 tab 键隔开的 9 列组成，以下是各列的说明：

Column 1: “seqid”

序列的编号，编号的有效字符[a-zA-Z0-9.:^*$@!+_?-|]

Column 2: “source”

注释信息的来源，比如”Genescan”、”GenBank” 等，可以为空，为空用”.”点号代替

Column 3: “type”

注释信息的类型，比如 Gene、cDNA、mRNA 等，或者是 SO 对应的编号

Columns 4 & 5: “start” and “end”

开始与结束的位置，注意计数是从 1 开始的。结束位置不能大于序列的长度

Column 6: “score”

得分，数字，是注释信息可能性的说明，可以是序列相似性比对时的 E-values 值或者基因预测是的 P-values 值。“.”表示为空。

Column 7: “strand”

序列的方向， +表示正义链, -反义链 , ？ 表示未知.

Column 8: “phase”

仅对注释类型为 “CDS”有效，表示起始编码的位置，有效值为 0、1、2。

Column 9: “attributes”

以多个键值对组成的注释信息描述，键与值之间用“=”，不同的键值用“;”隔开，一个键可以有多个值，不同值用“,”分割。注意如果描述中包括 tab 键以及”,=;”，要用 URL 转义规则进行转义，如 tab 键用 %09 代替。键是区分大小写的，以大写字母开头的键是预先定义好的，在后面可能被其他注释信息所调用。

预先定义的键包括：

ID 注释信息的编号，在一个 GFF 文件中必须唯一；

Name 注释信息的名称，可以重复；

Alias 别名

Parent Indicates 该注释所属的注释，值为注释信息的编号，比如外显子所属的转录组编号，转录组所属的基因的编号。值可以为多个。

Target Indicates： the target of a nucleotide-to-nucleotide or protein-to-nucleotide alignment.

Gap：The alignment of the feature to the target if the two are not collinear (e.g. contain gaps).

Derives_from：Used to disambiguate the relationship between one feature and another when the relationship is a temporal one rather than a purely structural “part of” one. This is needed for

polycistronic genes.

Note 备注

Dbxref 数据库索引

Ontology_term： A cross reference to an ontology term.

GFF 文件示例：

```
##gff-version 3
##sequence-region ctg123 1 1497228
ctg123 . gene 1000 9000 . + . ID=gene00001;Name=EDEN
ctg123 . TF_binding_site 1000 1012 . + . Parent=gene00001
ctg123 . mRNA 1050 9000 . + . ID=mRNA00001;Parent=gene00001
ctg123 . mRNA 1050 9000 . + . ID=mRNA00002;Parent=gene00001
ctg123 . mRNA 1300 9000 . + . ID=mRNA00003;Parent=gene00001
ctg123 . exon 1300 1500 . + . Parent=mRNA00003
ctg123 . exon 1050 1500 . + . Parent=mRNA00001,mRNA00002
ctg123 . exon 3000 3902 . + . Parent=mRNA00001,mRNA00003
ctg123 . exon 5000 5500 . + . Parent=mRNA00001,mRNA00002,mRNA00003
ctg123 . exon 7000 9000 . + . Parent=mRNA00001,mRNA00002,mRNA00003
ctg123 . CDS 1201 1500 . + 0 ID=cds00001;Parent=mRNA00001
ctg123 . CDS 3000 3902 . + 0 ID=cds00001;Parent=mRNA00001
ctg123 . CDS 5000 5500 . + 0 ID=cds00001;Parent=mRNA00001
ctg123 . CDS 7000 7600 . + 0 ID=cds00001;Parent=mRNA00001
ctg123 . CDS 1201 1500 . + 0 ID=cds00002;Parent=mRNA00002
ctg123 . CDS 5000 5500 . + 0 ID=cds00002;Parent=mRNA00002
ctg123 . CDS 7000 7600 . + 0 ID=cds00002;Parent=mRNA00002
ctg123 . CDS 3301 3902 . + 0 ID=cds00003;Parent=mRNA00003
ctg123 . CDS 5000 5500 . + 2 ID=cds00003;Parent=mRNA00003
```

（http://boyun.sh.cn/bio/?p=1602）

四、实验内容

1. 在自己的主目录下的 linux/exp 目录中新建目录 exp_7；
2. 将/home/pub/gff/637000073.gff 复制到 exp_7 下，并在此目录中编辑 Shell 脚本 inter_gene_len.sh，实现计算所有在正义链上基因间的距离（方向为正的基因间的距离）。（提示：可以利用 Python 和 Shell 混合编程，运行时 Shell 脚本调用 Python 脚本处理一部分任务）

五、实验报告

1. 实验环境（包括操作系统和软件）实验步骤，结果文件记录；
2. 实验中遇到的问题,如何解决的。

实验 8：Shell 程序开发（2）：PubMed 文献下载

一、实验目的

1. 了解 NCBI 的 PubMed 文献资源库；
2. 掌握利用 Shell 脚本从 PubMed 批量下载文献信息的方法。

二、实验环境

1. 操作系统：客户端 Windows，服务器端 Linux
2. 主要软件：PuTTY

三、实验原理

PubMed 是由美国国家医学图书馆(NLM)的国家生物技术信息中心（NCBI）开发的基于 Web 的检索系统，通过 NCBI 平台提供基于 Web 的免费 MEDLINE 数据库检索服务，并提供部分免费的全文链接服务，此外还可以访问 NCBI 维护的完整的分子生物学数据库．1999 年 8 月 PubMed 加入 NCBI 开发的 Entrez 通用浏览器，更换了检索界面。

NCBI 提供批量下载工具 efetch（http://www.ncbi.nlm.nih.gov/entrez/query/static/efetchseq_help.html），可以批量下载基因序列、蛋白质序列、文献摘要等。如：wget "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=pubmed&id=7764678&retmode=text&rettype=medline"（注意，地址两侧要用引号，否则 wget 会认为是由"&"分割的多个地址），即可下载 PubMed ID 为 7764678 的文献的相关信息：

```
PMID- 7764678
OWN - NLM
STAT- MEDLINE
DA  - 19940606
DCOM- 19940606
LR  - 20081121
IS  - 8756-7938 (Print)
IS  - 1520-6033 (Linking)
Vim  - 10
IP  - 2
DP  - 1994 Mar-Apr
TI  - Intermolecular electrostatic interactions and their effect on flux and
protein
      deposition during protein filtration.
PG  - 207-13
AB  - Although membrane filtration is used extensively to process protein
solutions containing a variety of electrolytes, there is currently little
fundamental understanding of the effect of the solution environment (and in
particular, the solution pH) on the filtrate flux in these systems. We have
obtained data for the flux and sieving coefficients during the batch (stirred
cell) filtration of solutions of bovine serum albumin, immunoglobulins,
hemoglobin, ribonuclease A, and lysozyme through 0.16-micron microfiltration
```

membranes at different pH values. The flux declined significantly for all five proteins due to the formation of a protein deposit on the upper surface of the membrane. The quasi-steady ultrafiltrate fluxes at the individual protein isoelectric pH's were essentially identical, despite the large differences in molecular weight and physicochemical characteristics of these proteins. The flux increased at pH's away from the isoelectric point, with the data well-correlated with the protein surface charge density. These results were explained in terms of a simple physical model in which the protein deposit continues to grow, and thus the flux continues to decline, until the drag force on the proteins associated with the filtrate flow is no longer able to overcome the intermolecular repulsive interactions between the proteins in the bulk solution and those in the protein deposit on the surface of the membrane.
AD - Department of Chemical Engineering, University of Delaware, Newark 19716.
FAU - Palecek, S P
AU - Palecek SP
FAU - Zydney, A L
AU - Zydney AL
LA - eng
GR - RO1-HL-39455-02/HL/NHLBI NIH HHS/United States
PT - Journal Article
PT - Research Support, U.S. Gov't, P.H.S.
PL - UNITED STATES
TA - Biotechnol Prog
JT - Biotechnology progress
JID - 8506292
RN - 0 (Membrane Proteins)
RN - 0 (Proteins)
SB - B
MH - Chemistry, Physical
MH - Electrochemistry
MH - Hydrogen-Ion Concentration
MH - Isoelectric Focusing
MH - Membrane Proteins/chemistry
MH - Models, Chemical
MH - Molecular Weight
MH - Physicochemical Phenomena
MH - Protein Conformation
MH - Proteins/*chemistry
MH - Ultrafiltration
EDAT- 1994/03/01
MHDA- 1994/03/01 00:01
CRDT- 1994/03/01 00:00
AID - 10.1021/bp00026a010 [doi]
PST - ppublish
SO - Biotechnol Prog. 1994 Mar-Apr;10(2):207-13.

wget 功能强大，提供了很多参数，如-q（不输出提示）、-O（输出到文件，后跟文件名，如果跟 - 表示标准输出）、-i（从文件读取 url 地址）等。

四、实验内容

1. 在自己的主目录下的 linux/exp 目录中新建目录 exp_8；
2. 在目录 exp_8 中编辑 Shell 脚本 get_pubmed.sh，实现下载 PubMed ID 为 1768001 到 1768010 的文献信息，要求程序运行时提供两个命令行参数，第 1 个命令行参数是第一篇文献的 PubMed ID，第 2 个命令行参数是最后一篇文献的 PubMed ID。

五、实验报告

1. 实验环境（包括操作系统和软件）实验步骤，结果文件记录；
2. 实验中遇到的问题,如何解决的。