

第十一章、认识和学习 BASH

前面几章我们使用终端机下达指令的方式，就是透过 bash 的环境来处理的，这个章节几乎是所有指令列模式 (command line) 与主机维护和管理的重要基础 (这一章有 60 页 !!!)

1、回顾：管理计算机硬件的是操作系统的核心 (kernel)，而 Linux 使用者通过 Shell 与核心沟通。只要有操作系统，就离不开 Shell。

先了解一下计算机是怎么运作的，当计算机要播放音乐时：

(1) 硬件：声卡 (声卡可分为模数转换电路和数模转换电路两部分，模数转换电路负责将麦克风等声音输入设备采到的模拟声音信号转换为电脑能处理的数字信号；而数模转换电路负责将电脑使用的数字声音信号转换为喇叭等设备能使用的模拟信号。)

(2) 核心管理：操作系统核心支持此声卡芯片，且有对应的驱动程序

(3) 应用程序：使用者输入指令控制音乐播放

其中应用程序处于最外层，所以叫 Shell (壳)

Linux 是用 C 程序语言撰写的

检查一下 /etc/shells 这个档案，可见有以下几个可用的不同类型的 shells

```
[root@localhost vtest]# cat /etc/shells
/bin/sh
/bin/bash
/sbin/nologin
/usr/bin/sh
/usr/bin/bash
/usr/sbin/nologin
/bin/tcsh
/bin/csh
```

/bin/bash (就是 Linux 预设的 shell)

```
[root@localhost ~]# cat /etc/passwd | grep bash
root:x:0:0:root:/root:/bin/bash
Ray:x:1001:1001::/home/Ray:/bin/bash
```

bash 的优点：

1、能记忆用户输入过的指令 (按上下键就能找到前/后指令)

这些指令序列记录在家目录的 .bash_history 里，打开一看：

```
LANG=zh_CN.UTF-8
locale
locale -a | grep zh_CN
\locale -a
vim /etc/sysconfig/i18n
vim ~/.viminfo
vim Learning_Project.txt
vim /etc/vimrc
vim Learning_Project.txt
cd /tmp/vitest/
vim A.txt
unix2dos -k A.txt
yum install unix2dos
unix2dos -k A.txt
vim A.txt
dos2unix -k /root/Learning_Project.txt
vim /root/Learning_Project.txt
ls /etc/shells
cat /etc/shells
```

不过这些都是上一次登入所执行过的指令，而这一次登入执行过的指令都暂存在内存里，当注销系统时指令记忆才会被记录进 .bash_history 中

2、Tab 键 进行命令与文件名补全

- 3、命令别名设定 (alias)。可以设定 alias lm=' ls -al' ，然后就可以用 lm 替代 ls -al 这个指令使用
- 4、工作控制，前景背景控制，Ctrl+z 将工作放到后台执行
- 5、程序化脚本 (shell scripts) 可以将平时管理系统所需要常常下达的连续指令写成一个文档，运行该文档能进行日常工作
- 6、通配符。 * 等符号

如果指令太长，一行不够写，可以在第一行末尾加上反斜杠 \ ，然后 Enter 就可以在下一行继续写了，最后默认连接起来

当我们在终端机 tty 上登入时，Linux 就会根据/etc/passwd 文档的设定给我们分配一个 shell

变量

- 1、取用 PATH 之类的变量来显示：echo，启用变量时，变量前要加一个\$符号比如

echo \$PATH

```
[root@localhost ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
```

还有家目录

```
[root@localhost ~]# echo $HOME
/root
[root@localhost ~]# su Ray
[Ray@localhost root]$ echo $HOME
/home/Ray
```

- 2、修改变量

直接用『等号(=)』连接变量和它的内容

变量内容若有空格符可使用双引号『"』或单引号『'』将变量内容结合起来

```
[Ray@localhost root]$ var11='I am a stupid'
[Ray@localhost root]$
[Ray@localhost root]$ echo $var11
I am a stupid
```

若该变量为扩增变量内容时，则可用 "\$变量名称" 或 \${变量} 累加内容

```
[Ray@localhost root]$ var11="$var11"! I love inory'
[Ray@localhost root]$ echo $var11
I am a stupid! I love inory
```

取消变量：使用 unset

unset var11

```
[Ray@localhost root]$ name=Ray
[Ray@localhost root]$ bash
[Ray@localhost root]$ echo $name

[Ray@localhost root]$ exit
exit
[Ray@localhost root]$ export name
[Ray@localhost root]$ bash
[Ray@localhost root]$ echo $name
Ray
[Ray@localhost root]$ exit
exit
```

上面这个例子中，bash 表示进入子程序，exit 离开子程序。

第一次在子程序中 echo \$name 时输出为空，说明父程序的自定义变量无法在子程序中使用
但通过 export 将变量变成环境变量后，就能够在子程序里使用了

进入当前核心的驱动程序

```
cd /lib/modules/`uname -r`/kernel
```

以上指令中 uname -r 得到的是核心的版本号，注意这里用到的是反单引号而不是单引号

单引号与双引号的区别，鸟哥的例子：

```
[root@www ~]# name=VBird
[root@www ~]# echo $name
VBird
[root@www ~]# myname="$name its me"
[root@www ~]# echo $myname
VBird its me
[root@www ~]# myname='$name its me'
[root@www ~]# echo $myname
$name its me
```

即双引号内，\$name 会被转换成 name 这个变量的内容 VBird，而单引号内\$name 就是\$name

反单引号(`)这个符号代表乱意义：在一串指令中，在 ` 之内乱指令将会被先执行

使用变量的好处：

比如有一个常去的目录，名字很长/cluster/server/work/taiwan_2005/003/，那样每次想要 cd 进入该目录都要弄很久，但如果使用变量记录下来：

work= "/cluster/server/work/taiwan_2005/003/"，那每次进入目录只需要 cd \$work 即可

```
[root@localhost ~]# work="/home/Ray"
[root@localhost ~]# cd $work
[root@localhost Ray]#
```

如何查看当前 shell 环境中有多少默认的环境变量？

1、env 指令

```
[root@localhost Ray]# myname="zhongzhanhui"
[root@localhost Ray]# export myname
[root@localhost Ray]# env
LANG=zh-CN.UTF-8
SELINUX_LEVEL_REQUESTED=
myname=zhongzhanhui
HISTCONTROL=ignoredups
SHLVL=3
HOME=/root
```

HOME 代表家目录是/root，cd~能回到家目录就是取用的这个变量

切换 Ray 账号看到 HOME 是/home/Ray

```
HISTCONTROL=ignoredups
myname=zhongzhanhui
HOME=/home/Ray
SHLVL=4
```

2、set 指令。不仅列出环境变量，还能列出所有变量

变量 PS1：(提示字符的设定)

CentOS 预设的 PS1 内容：『\u@\h \W]\\$ 』

\u : 目前使用者的账号名称, 如『root』;
\h : 仅取主机名在第一个小数点之前名字, 如鸟哥主机则为『www』后面省略
\W : 利用 basename 函数取得工作目录名称, 所以仅会列出最后一个目录名。
\\$: 提示字符, 如果是 root 时, 提示字符为 # , 否则就是 \$
那你现在知道为何你的命令提示字符是: 『 [root@www ~]# 』了吧

假如我想要有类似底下的提示字符:

```
[root@www /home/dmtsai 16:50 #12]#
```

那个 # 代表第 12 次下达的指令。那么应该如何设定 PS1 呢?

\# : 下达的第几个指令

\w : 完整的工作目录名称, 由根目录写起的目录名称。但家目录会以 ~ 取代

\t : 显示时间, 为 24 小时格式的『HH:MM:SS』

```
[root@localhost /home/Ray 21:12:41 14]$PS1="[u@\h \w \t #\#]\$"  
[root@localhost /home/Ray 21:13:10 #15]$
```

export 把自定义变量变成环境变量

我们在原本的 bash 底下执行另一个 bash , 结果操作的环境接口会跑到第二个 bash 去(就是子程序), 那原本的 bash 就会在暂停的情况(睡着了, 就是 sleep)。整个指令运作的环境是实线的部分 若要回到原本的 bash 去, 就只有将第二个 bash 结束掉(下达 exit 或 logout)才行。

子程序仅会继承父程序的环境变量, 子程序不会继承父程序的自定义变量

read 之后不加任何参数, 直接加上变量名称, 那么底下就会主动出现一个空白行等待你的输入。

如果加上 -t 后面接秒数, 那么指定秒内没有任何动作时, 该指令就会自动略过了

如果是加上 -p , 在输入的光标前就会有比较多可以用的提示字符给我们参考!

```
[root@localhost /home/Ray 21:48:36 #17]$read readtry  
lalalalal  
[root@localhost /home/Ray 22:04:35 #18]$echo $readtry  
lalalalal  
[root@localhost Ray]$read -p "Input something baby:" -t 10 readtry1  
Input something baby:hahaha  
[root@localhost Ray]$echo $readtry1  
hahaha
```

变量宣告 declare

-a : 将后面名为 variable 的变量定义成为数组 (array) 类型

-i : 将后面名为 variable 的变量定义成为整数数字 (integer) 类型

-x : 用法不 export 一样, 就是将后面的 variable 发成环境变量;

-r : 将变量设定成为 readonly 类型, 该变量不可被更改内容, 也不能 unset

```
[root@localhost Ray]$sum=10+20+30
[root@localhost Ray]$echo $sum
10+20+30
[root@localhost Ray]$declare -i sum=10+20+30
[root@localhost Ray]$echo $sum
60
```

文件系统资源的限制：ulimit

为了防止过多的用户无节制使用系统资源，bash 可以限制用户的某些系统资源（包括可以开启的文档数量、可以使用的 CPU 时间、可使用的内存总量等）

参数很多，不记了，用到的时候再查询

1、列出当前账户所有限制资源的数值：

```
[root@localhost Ray]$ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority     (-e) 0
file size                (blocks, -f) unlimited
pending signals         (-i) 7207
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files              (-n) 1024
pipe size               (512 bytes, -p) 8
POSIX message queues    (bytes, -q) 819200
real-time priority      (-r) 0
stack size              (kbytes, -s) 8192
cpu time                (seconds, -t) unlimited
max user processes      (-u) 7207
virtual memory          (kbytes, -v) unlimited
file locks              (-x) unlimited
```

0 就是没有限制

file size 可建立的单一文档的大小

open files 可同时开启的文档数量

2、尝试限制用户只能新建 10MB 以下容量的文档

```
[root@localhost Ray]$ulimit -f 10240
[root@localhost Ray]$ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority     (-e) 0
file size                (blocks, -f) 10240
pending signals         (-i) 7207
```

尝试建立 20MB 的文件失败

```
[root@localhost Ray]$dd if=/dev/zero of=trytry bs=1M count=20
File size limit exceeded (core dumped)
```

注销账户后再登录，则 ulimit 恢复默认设定

变量内容的部分删除和替代：

部分删除：

1、建立 path 变量（copy PATH）

```
[root@localhost ~]# path=$PATH
[root@localhost ~]# echo $path
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
```

2、尝试删除前两个目录：

```
[root@localhost ~]# echo ${path#/*local/bin:}
/usr/sbin:/usr/bin:/root/bin
```


#表示由前面开始删除，可以透过通配符 * 删除从第 0 位 (/) 开始到后面所接字符串之间的无穷多个任意字符。

: 符合取代文字的『最短的』那一个；

: 符合取代文字的『最长的』那一个

还有其他的，有点麻烦复杂，直接贴个总结算了

| 变量设定方式 | 说明 |
|--------------------------------|---------------------------------|
| <code>\${变量#关键词}</code> | 若变量内容从头开始的数据符合『关键词』，则将符合的最短数据删除 |
| <code>\${变量##关键词}</code> | 若变量内容从头开始的数据符合『关键词』，则将符合的最长数据删除 |
| <code>\${变量%关键词}</code> | 若变量内容从尾向前的数据符合『关键词』，则将符合的最短数据删除 |
| <code>\${变量%%关键词}</code> | 若变量内容从尾向前的数据符合『关键词』，则将符合的最长数据删除 |
| <code>\${变量/旧字符串/新字符串}</code> | 若变量内容符合『旧字符串』则『第一个旧字符串会被新字符串取代』 |
| <code>\${变量//旧字符串/新字符串}</code> | 若变量内容符合『旧字符串』则『全部旧字符串会被新字符串取代』 |

变量的测试与内容替换：

```
username=${username-Ray}
```

如果 username 已经存在则不改变原内容，如果不存在则给予 Ray 这个内容

```
username=${username:-Ray}
```

多了一个冒号:，这样即使 username 存在，但如果是空字符串，则也会给予 Ray 这个内容。

但是即使如此，我还是更喜欢直接用 echo \$varname 这个方式看一下是不是有这个变量

命令别名：

直接用 vi 打开文档的话，就是黑白两色的

```
# This file is used by the man-db package to configure the man and cat paths.
# It is also used to provide a manpath for those without one by examining
# their PATH environment variable. For details see the manpath(5) man page.
#
# Lines beginning with '#' are comments and are ignored. Any combination of
# tabs or spaces may be used as 'whitespace' separators.
#
```

但如果使用命令别名，使 vi=vim 的话：

```
[root@localhost vittest]# alias vi='vim'
[root@localhost vittest]# vi man.config
```

```
CentOS
# This file is used by the man-db package to configure the man and cat paths.
# It is also used to provide a manpath for those without one by examining
# their PATH environment variable. For details see the manpath(5) man page.
#
# Lines beginning with '#' are comments and are ignored. Any combination of
```

每次用 vi 的时候就自动变成用 vim 打开了。

现在取消命令别名：

```
[root@localhost vittest]# unalias vi
[root@localhost vittest]# vi man.config
```

```
# This file is used by the man-db package to configure the man
# It is also used to provide a manpath for those without
# their PATH environment variable. For details see the ma
#
# Lines beginning with '#' are comments and are ignored.
```

恢复原状

历史命令 history：查询我们曾经下达过的指令

列出最近的 10 个历史指令：

```
[root@localhost vittest]# history 10
747  ls
748  vi man.config
749  alias vi='vim'
750  vi man.config
751  unalias vi='vim'
752  unalias vi
753  vi man.config
754  clear
755  history
756  history 10
```

在查询到历史指令的基础上，如果想再次执行其中一个指令，比如指令 750，则可以：

```
[root@localhost vittest]# !750
vi man.config
# This fs used by the man-db package to
# It is also used to provide a manpath
# their PATH environment variable. For
#
# Lines beginning with `#' are comments
# tabs or spaces may be used as `whitesp
#
```

既然一个指令有可能既是命令别名又是 PATH 搜寻等，那么按什么顺序来选择执行呢？

```
[root@localhost vittest]# type -a ls
ls 是 `ls --color=auto' 的别名
ls 是 /usr/bin/ls
```

可见是先别名、后按 PATH 搜寻执行文件/usr/bin/ls

bash 的进站与欢迎信息：/etc/issue /etc/motd

在中断机接口（tty1~tty6）登录时会有几行字符串构成进站画面：

比较神奇的是 ctrl+alt+F5 切换到 tty5 这个快捷键只能在 vmware 上用，xshell 不行。。。

```
CentOS Linux 7 (Core)
Kernel 3.10.0-693.el7.x86_64 on an x86_64
```

这个字符串记录在/etc/issue 内

```
[root@localhost vittest]# cat /etc/issue
\S
Kernel \r on an \m
[root@localhost vittest]#
```

| |
|--------------------------------------|
| \d 本地端时间的日期； |
| \l 显示第几个终端机接口； |
| \m 显示硬件的等级 (i386/i486/i586/i686...)； |
| \n 显示主机的网络名称； |
| \o 显示 domain name； |
| \r 操作系统的版本 (相当于 uname -r) |
| \t 显示本地端时间的时间； |
| \s 操作系统的名称； |
| \v 操作系统的版本。 |

例题：

如果你在 tty3 乱进站画面看到如下显示，该如何设定才能得到如下画面？

CentOS release 5.3 (Final) (terminal: tty3)

Date: 2009-02-05 17:29:19

Kernel 2.6.18-128.el5 on an i686

Welcome!

Answer：

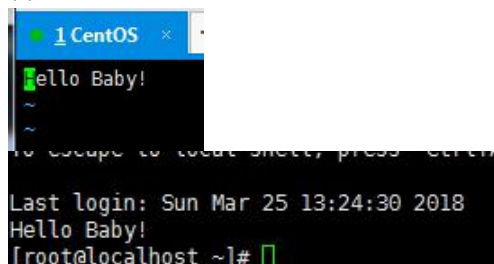
\s (terminal:\l)

Date: \d \t

Kernel \r on an \m

Welcome!

至于登录时显示的信息：在/etc/motd 里，本来是空的，我们可以加入一些信息给用户登录后看



```

1 CentOS x
Hello Baby!
~
~
To escape to local shell, press Ctrl-C.
Last login: Sun Mar 25 13:24:30 2018
Hello Baby!
[root@localhost ~]#

```

bash 的环境配置文件分为全体系统的配置文件和用户个人偏好配置文件

命令别名、自定义变量等，在注销 bash 后就会失效，所以如果想要保留设定，需要将其写入 bash 配置文件之中。

login shell：需要输入账号密码得到的 bash

non-login shell：不需要输入账号密码得到的 bash，如终端之间的切换、子程序 bash 等

以上两种情况读取的配置文件不同

login shell 会读取的配置文件：

1、/etc/profile：系统整体设定

2、~/.bash_profile, ~/.bash_login, ~/.profile：用户个人设定

1、bash 的 login shell 情况下所读取的整体环境配置文件其实只有 /etc/profile，但是 /etc/profile 还会呼叫出其他的配置文件
接着 bash 会读取使用者的个人配置文件，依次检测 ~/.bash_profile, ~/.bash_login, ~/.profile，一旦检测到有某一个文件，则只会读取那一个配置文件。

```
[root@localhost ~]# cat ~/.bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin
export PATH
```

首先读取 ~/.bashrc，然后再 PATH 变量后面加上家目录下的 bin 目录，然后 export 使之变成环境变量

/etc/profile 和 ~/.bash_profile 都是登录时才会读取的配置文件，如果修改了配置文件后，就得注销重登才生效，但用 source 指令能直接读入环境配置文件

source+配置文件名

当需要在主机上处理不同的工作，而它们又需要不同的环境配置时，可以各自编写一个配置文件，当切换工作时 source 这个配置文件即可简便的切换环境。

2、non-login shell 情况下 bash 仅会读取 ~/.bashrc 这个配置文件，它会呼叫/etc/bashrc 和 /etc/profile.d/*.sh

终端机的环境设定

在 tty1 ~ tty6 这六个文字接口的终端机 (terminal) 环境中登入

- 1、可以使用退格键(backspace) 来删除命令行上的字符，
- 2、也可以使用 [ctrl]+c 强制终止一个指令的运行
- 3、当输入错误时，就会有警告声

可以用 stty (setting tty) 来查阅：

stty -a 列出所有的按键及其作用，其中^表示 ctrl 按键

```
[root@localhost ~]# stty -a
speed 38400 baud; rows 19; columns 104; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>; swtch = <undef>;
start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd -cmspar cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff -iucLC -ixany -imaxbel
-iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
```

- eof : End of file 的意思，代表『结束输入』。
- erase : 向后删除字符，
- intr : 送出一个 interrupt (中断) 的讯号给目前正在 run 的程序；
- kill : 删除在目前指令列上的所有文字；
- quit : 送出一个 quit 的讯号给目前正在 run 的程序；
- start : 在某个程序停止后，重新启动他的 output
- stop : 停止目前屏幕的输出；
- susp : 送出一个 terminal stop 的讯号给正在 run 的程序。

修改设定： stty erase ^h 后，删除字符就要用 ctrl+h

此外还有 set 指令设定指令输出/输入环境

```
[root@www ~]# set [-uvCHmBx]
选项与参数：
-u : 预设不启用。若启用后，当使用未设定变量时，会显示错误讯息；
-v : 预设不启用。若启用后，在讯息被输出前，会先显示讯息的原始内容；
-x : 预设不启用。若启用后，在指令被执行前，会显示指令内容(前面有 ++ 符号)
-h : 预设启用。与历史命令有关；
-H : 预设启用。与历史命令有关；
-m : 预设启用。与工作管理有关；
-B : 预设启用。与刮号 [ ] 的作用有关；
-C : 预设不启用。若使用 > 等，则若档案存在时，该档案不会被覆盖。
```

比如启用 x : set -x，关闭 x : set +x

其实以上这些终端机环境本来默认设定就很好，没必要修改

一些 bash 的默认组合键

| 组合按键 | 执行结果 |
|----------|-----------------------|
| Ctrl + C | 终止目前的命令 |
| Ctrl + D | 输入结束 (EOF)，例如邮件结束的时候； |
| Ctrl + M | 就是 Enter 啦！ |
| Ctrl + S | 暂停屏幕的输出 |
| Ctrl + Q | 恢复屏幕的输出 |
| Ctrl + U | 在提示字符下，将整列命令删除 |
| Ctrl + Z | 『暂停』目前的命令 |

通配符

| 符号 | 意义 |
|-------|--|
| * | 代表『0 个到无穷多个』任意字符 |
| ? | 代表『一定有一个』任意字符 |
| [] | 同样代表『一定有一个在括号内』的字符(非任意字符)。例如 [abcd] 代表『一定有一个字符，可能是 a, b, c, d 这四个任何一个』 |
| [-] | 若有减号在中括号内时，代表『在编码顺序内的所有字符』。例如 [0-9] 代表 0 到 9 之间的所有数字，因为数字的语系编码是连续的！ |
| [^] | 若中括号内的第一个字符为指数符号 (^)，那表示『反向选择』，例如 [^abc] 代表一定有一个字符，只要是非 a, b, c 的其他字符就接受的意思。 |

数据流重导向：将某个指令执行完后本应出现在屏幕上的数据，传输到其他位置（比如文档或者打印机等装置）

standard output 标准输出：指令执行所回传的正确信息

standard error output 标准错误输出：指令执行失败后回传的错误信息

1. 标准输入 (stdin) : 代码为 0 , 使用 < 或 << ;
2. 标准输出 (stdout) : 代码为 1 , 使用 > 或 >> ;
3. 标准错误输出(stderr) : 代码为 2 , 使用 2> 或 2>> ;

比如平常的 ll

```
[root@localhost ~]# ll
总用量 146876
-rw-r--r--. 1 root root    1614 3月  7 20:20 anaconda-ks.cfg
-rw-r--r--. 1 root root 115998720 3月 23 13:35 boot.dump
-rw-r--r--. 1 root root 10577920 3月 23 13:57 boot.dump.1
-rw-r--r--. 2 root root    451 6月 10 2014 crontab
lrwxrwxrwx. 1 root root    12 3月 19 21:11 crontab2 -> /etc/crontab
-rw-r--r--. 1 root root    61 3月 23 19:53 cut.txt
drwxr-xr-x. 2 root root   4096 3月  8 19:57 Desktop
drwxr-xr-x. 2 root root   4096 3月  8 19:57 Documents
drwxr-xr-x. 2 root root   4096 3月  8 19:57 Downloads
drwxr-xr-x. 2 root root   4096 3月 23 00:25 etc
-rw-r--r--. 1 root root 14175785 3月 23 14:10 etc.dump.bz2
-rw-r--r--. 1 root root   10270 3月 23 00:49 etc.newerthanpasswd.tar.bz2
-rw-r--r--. 1 root root  9398803 3月 22 20:56 etc.tar.bz2
-rwxr--r--. 1 root root    1 3月 23 18:39 haha_copy.txt
```

将 ll 输出信息导入到文档 ll_res.txt 中

```
[root@localhost ~]# ll >> ll_res.txt
[root@localhost ~]# vi ll_res.txt
```

```
总用量 146876
-rw-r--r--. 1 root root    1614 3月  7 20:20 anaconda-ks.cfg
-rw-r--r--. 1 root root 115998720 3月 23 13:35 boot.dump
-rw-r--r--. 1 root root 10577920 3月 23 13:57 boot.dump.1
-rw-r--r--. 2 root root    451 6月 10 2014 crontab
lrwxrwxrwx. 1 root root    12 3月 19 21:11 crontab2 -> /etc/crontab
-rw-r--r--. 1 root root    61 3月 23 19:53 cut.txt
drwxr-xr-x. 2 root root   4096 3月  8 19:57 Desktop
drwxr-xr-x. 2 root root   4096 3月  8 19:57 Documents
drwxr-xr-x. 2 root root   4096 3月  8 19:57 Downloads
drwxr-xr-x. 2 root root   4096 3月 23 00:25 etc
-rw-r--r--. 1 root root 14175785 3月 23 14:10 etc.dump.bz2
-rw-r--r--. 1 root root   10270 3月 23 00:49 etc.newerthanpasswd.tar.bz2
-rw-r--r--. 1 root root  9398803 3月 22 20:56 etc.tar.bz2
-rwxr--r--. 1 root root    1 3月 23 18:39 haha_copy.txt
```

如果文档 ll_res.txt 不存在则新建

> : 如果文档已存在则覆盖 >> : 如果文档已存在则在文档最后累加写入

同时将正确数据和错误警告存入不同的文档 :

```
[root@localhost ~]# find /home -name.bashrc >list_right 2>list_error
[root@localhost ~]# cat list_right
[root@localhost ~]# cat list_error
find: 未知的断言“-name.bashrc”
[root@localhost ~]#
```

如果不想显示错误信息, 可以在指令后面接 2> /dev/null

导向/dev/null 这个装置的任何信息数据都直接被消除

如果接完> filename 后再接 2>&1, 则错误信息也写入正确信息的那个文档中

<

将原本需要由键盘输入的数据, 改由文档内容来取代

一次输入多重指令

cmd1;cmd2

分号前的指令执行完毕后接着执行后面的指令

cmd1&&cmd2

cmd1 执行正确则继续执行 cmd2；cmd1 执行错误则不执行 cmd2

cmd1||cmd2

cmd1 正确则不执行 cmd2；cmd1 错误则继续执行 cmd2

以 ls 测试 /tmp/vbirding 是否存在，若存在则显示 "exist"，若不存在，则显示 "not exist"！

```
[root@localhost ~]# ls /tmp/vbirding && echo "exist" || echo "not exist"
ls: 无法访问/tmp/vbirding: 没有那个文件或目录
not exist
```

一条指令如果执行成功，会回传 \$?=0，否则回传 \$?!=0

如果文件存在，则第一条 ls 指令成功执行（回传 \$?=0），则根据 && 符号，会继续执行第二条指令，输出 exist，即第二条指令也执行成功（回传 \$?=0），又根据 || 符号，第三条指令不继续执行；

如果文件不存在，则第一条指令执行失败（回传 \$?!=0），根据 &&，第二条指令不执行（即继续回传前面的 \$?!=0），根据 ||，第三条指令会执行，输出 not exist

管线命令：

管线命令能处理 standard output，无法处理 standrad error output

管线命令要能接收来自前一个指令的数据，将其作为 standrad input 继续处理才行

1、

```
[root@localhost ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
[root@localhost ~]# echo $PATH | cut -d ':' -f 3
/usr/sbin
```

这条 cut 指令，-d 后接的是分隔符(:)，-f 后接的是截取分割后的第几段，这里取的是第三段，即 /usr/sbin

2、

```
[root@localhost ~]# export
declare -x HISTCONTROL="ignoredups"
declare -x HISTSIZE="1000"
declare -x HOME="/root"
declare -x HOSTNAME="localhost.localdomain"
[root@localhost ~]# export | cut -c 10-20
x HISTCONTR
x HISTSIZE=
x HOME="/ro
x HOSTNAME=
x LANG="zh_
x LESSOPEN=
x LOGNAME="
```

这个有点鸡肋。-c 后面接字符区间，每一行都只截取那个区间显示出来

3、

```
[root@localhost ~]# last
root      pts/0      192.168.142.1    Mon Mar 26 21:33  still logged in
reboot    system boot  3.10.0-693.el7.x Mon Mar 26 21:31 - 22:38 (01:06)
root      pts/1      192.168.142.1    Sun Mar 25 15:24 - 19:52 (04:27)
```



```
[root@localhost ~]# last | cut -d ' ' -f 1
root
reboot
root
root
root
```

截取用户名

但是由于 root 与 pts/0 之间不止一个空格，因此取其他的有点麻烦

即 cut 在处理多空格相连的数据时，会比较困难

cut 是从每行信息中截取我们需要的那部分

grep 是找出包含我们想要的信息的那一行出来

grep [-acinv] [--color=auto] '搜寻字符串' filename

1、找出含有“root”的行

```
[root@localhost ~]# last | grep "root"
root pts/0 192.168.142.1 Mon Mar 26 21:33 still logged in
root pts/1 192.168.142.1 Sun Mar 25 15:24 - 19:52 (04:27)
root pts/0 192.168.142.1 Sun Mar 25 13:32 - 18:02 (04:29)
root tty4 Sun Mar 25 13:24 - 20:58 (07:33)
```

2、找出不含“root”的行

```
[root@localhost ~]# last | grep -v "root"
reboot system boot 3.10.0-693.el7.x Mon Mar 26 21:31 - 22:45 (01:13)
reboot system boot 3.10.0-693.el7.x Sun Mar 25 12:47 - 22:45 (1+09:57)
reboot system boot 3.10.0-693.el7.x Sat Mar 24 11:59 - 22:45 (2+10:45)
```

3、找出含有“root”的行，且只取第一栏（实话说这个指令没什么意义）

```
[root@localhost ~]# last | grep 'root' | cut -d ' ' -f 1
root
root
root
root
```

4、取出文件/etc/man.config 内含有 MANPATH 的那几行，且加上颜色

```
[root@localhost ~]# grep --color=auto 'MANPATH' /etc/man_db.conf
# MANDATORY MANPATH manpath_element
# MANPATH_MAP path_element manpath_element
# every automatically generated MANPATH includes these fields
#MANDATORY MANPATH /usr/src/pvm3/man
MANDATORY MANPATH /usr/man
MANDATORY MANPATH /usr/share/man
```

sort 指令

1、将账号按照字典顺序排序：

原本的顺序：

```
[root@localhost ~]# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
```

使用 sort（预设按字符字典顺序升序排序）

```
[root@localhost ~]# cat /etc/passwd | sort
abrt:x:173:173::/etc/abrt:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
alex:x:1002:1003::/home/alex:/bin/bash
arod:x:1003:1004::/home/arod:/bin/bash
```

2、按第三栏，即数字那一栏排序：

```
[root@localhost ~]# cat /etc/passwd | sort -t ':' -k 3
root:x:0:0:root:/root:/bin/bash
linuxidc:x:1000:1000::/home/linuxidc:/bin/bash
Ray:x:1001:1001::/home/Ray:/bin/bash
alex:x:1002:1003::/home/alex:/bin/bash
```

但它还是按字符类型来排序的，要按数字大小排序的话，最后还要再加上参数-n


```
[root@localhost ~]# cat /etc/passwd | sort -t ':' -k 3 -n
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
ln:x:4:7:ln:/var/spool/lnd:/sbin/nologin
```

3、只取 last 中的账号，且排序

```
[root@localhost ~]# last | cut -d ' ' -f 1 | sort
Ray
Ray
Ray
```

可见上面同一个账号有很多记录，如果每个账号只想它显示一次的话，就加上 uniq：

```
[root@localhost ~]# last | cut -d ' ' -f 1 | sort | uniq
Ray
reboot
root
wtmp
```

如果还想看到每个 uniq 的字符串的重复次数，可以加上参数 -c

```
wtmp
[root@localhost ~]# last | cut -d ' ' -f 1 | sort | uniq -c
  1
 22 Ray
 33 reboot
 58 root
  1 wtmp
[root@localhost ~]#
```

WC

```
[root@localhost ~]# cat /etc/man_db.conf | wc
 131   723  5171
[root@localhost ~]#
```

从左到右依次是行数、词数、字符数

tee

它的作用是既能在屏幕上输出我想要的东西，同时还能存一份数据到文档中

只需要在指令后面紧接着 | tee filename 即可

```
[root@localhost ~]# ls -l /home | tee ~/homefiletee | more
总用量 24948
drwx-----, 5 alex      alex      4096 3月  18 21:14 alex
drwx-----, 5 arod      arod      4096 3月  18 21:24 arod
drwx-----, 2 linuxidc linuxidc   4096 3月  7 20:30 linuxidc
-rw-r--r--, 1 root      root      536870912 3月 22 10:31 loopdev
drwx-----, 2 root      root      16384 3月  7 20:00 lost+found
-rw-r--r--, 1 root      root         0 3月 21 22:26 newFile3_21.txt
drwx-----, 14 Ray      Ray      4096 3月 24 22:54 Ray
[root@localhost ~]# cat homefiletee
总用量 24948
drwx-----, 5 alex      alex      4096 3月  18 21:14 alex
drwx-----, 5 arod      arod      4096 3月  18 21:24 arod
drwx-----, 2 linuxidc linuxidc   4096 3月  7 20:30 linuxidc
-rw-r--r--, 1 root      root      536870912 3月 22 10:31 loopdev
drwx-----, 2 root      root      16384 3月  7 20:00 lost+found
-rw-r--r--, 1 root      root         0 3月 21 22:26 newFile3_21.t
drwx-----, 14 Ray      Ray      4096 3月 24 22:54 Ray
```

字符转换命令

tr -d 删除字符

tr -s 替换字符

1、将小写转换成大写

```
[root@localhost ~]# cat passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
```

开始转换

```
[root@localhost ~]# cat passwd | tr -s '[a-z]' '[A-Z]'
ROT:X:0:0:ROT:/ROT:/BIN/BASH
BIN:X:1:1:BIN:/BIN:/SBIN/NOLOGIN
DAEMON:X:2:2:DAEMON:/SBIN:/SBIN/NOLOGIN
ADM:X:3:4:ADM:/VAR/ADM:/SBIN/NOLOGIN
LP:X:4:7:LP:/VAR/SPOL/LPD:/SBIN/NOLOGIN
SYNC:X:5:0:SYNC:/SBIN:/BIN/SYNC
```

当然这只是把显示内容转换了，文档内容并没有被改变

2、删除显示信息中的 ':'

```
[root@localhost ~]# cat passwd | tr -d ':'
rootx00root/root/bin/bash
binx11bin/bin/sbin/nologin
daemonx22daemon/sbin/sbin/nologin
admx34adm/var/adm/sbin/nologin
lp47lp/var/spool/lpd/sbin/nologin
```

3、tr 本来只能改显示信息不能改文档内容，但可以将显示信息存储进一个文件内

```
[root@localhost ~]# cat passwd | tr -d ':' > passwdtr
[root@localhost ~]# cat passwdtr
rootx00root/root/bin/bash
binx11bin/bin/sbin/nologin
daemonx22daemon/sbin/sbin/nologin
admx34adm/var/adm/sbin/nologin
```

tr 还有一个用法就是替代 dos2unix，只需要把文档中的 ^M 即 '\r' 删除掉即可

col :

1、col -x 将 tab 转换成对等的空格

```
[root@localhost ~]# vi coltabtry
[root@localhost ~]# cat -A coltabtry
qqqq^Ieeee$
```

^I 就是 tab

转换

```
[root@localhost ~]# cat coltabtry | col -x | cat -A
qqqq  eeee$
```

上面好像也没什么用。。。

2、将 man page 转存成纯文本文档，方便阅读

不管了，没什么意义

join 能将两个文档中，具有相同数据的一行整合起来

/etc/passwd

root:x:0:0:root:/root:/bin/bash

/etc/shadow

root:\$1\$/3AQpE5e\$y9A/D0bh6rElAs:14120:0:99999:7:::

整合：join -t ':' /etc/passwd /etc/shadow

结果：root:x:0:0:root:/root:/bin/bash:\$1\$/3AQpE5e\$y9A/D0bh6rElAs:14120:0:99999:7:::

可见，因为两行前面的 root 重复了，因此整合的时候第二段就去掉了 root 然后接上去

paste：相对比于 join，paste 更简单，不需要对比相关性，直接将两行数据整合起来，以 tab 键隔开

expand：也是把 tab 转换成空格，但可以自定义

expand [-t] file

-t：后面可以接数字。一般来说，一个 tab 按键可以用 8 个空格键替代。我们也可以自行定义一个 [tab] 按键代表多少个字符

split：分割文档 split [-bl] file PREFIX

-b：接分割后文档的大小，可加单位，例如 b, k, m 等；

-l：以行数来进行分割。

PREFIX：代表前导符的意思，可作为分割档案的前导文字。

下面这个文档有 5271KB 大小

```
[root@localhost ~]# ll /etc/man_db.conf
-rw-r--r--. 1 root root 5171 6月 10 2014 /etc/man_db.conf
[root@localhost ~]#
```

1、将它分成 1000（我忘了那是什么单位了）一个文档

```
[root@localhost tmp]# split -b 1000 /etc/man_db.conf man_db.split
[root@localhost tmp]# ll - man*
ls: 无法访问 -: 没有那个文件或目录
-rw-r--r--. 1 root root 1000 3月 27 12:20 man_db.splitaa
-rw-r--r--. 1 root root 1000 3月 27 12:20 man_db.splitab
-rw-r--r--. 1 root root 1000 3月 27 12:20 man_db.splitac
-rw-r--r--. 1 root root 1000 3月 27 12:20 man_db.splitad
-rw-r--r--. 1 root root 1000 3月 27 12:20 man_db.splitae
-rw-r--r--. 1 root root 1000 3月 27 12:20 man_db.splitaf
-rw-r--r--. 1 root root 171 3月 27 12:20 man_db.splitag
```

2、把它们再合成一个文档（用数据流重导向>>累加）：

```
[root@localhost tmp]# cat man_db.split* >> man_db.reunite
[root@localhost tmp]# ll man_db.reunite
-rw-r--r--. 1 root root 5171 3月 27 12:23 man_db.reunite
```

3、每 10 行记录成一个文件：

```
[root@localhost tmp]# ls -al | split -l 10 - lsroot
[root@localhost tmp]# wc -l lsroot*
 10 lsrootaa
 10 lsrootab
 10 lsrootac
 10 lsrootad
 10 lsrootae
 10 lsrootaf
  6 lsrootag
 66 总用量
```

参数替换 xargs，这个挺麻烦的不管了

减号-的作用：

比如 tar -cvf - /home | tar -xvf -

将/home 里的文档打包，但打包的数据不记录进文档（如果是记录进文档那/home 前应该是文件名而不是减号），而是传送到 stdout，经过管线（|）后传送给后面，后面的减号-取用前面的 stdout

```
[root@localhost tmp]# ls -al | split -l 10 - lsroot
[root@localhost tmp]# wc -l lsroot*
 10 lsrootaa
 10 lsrootab
 10 lsrootac
 10 lsrootad
 10 lsrootae
 10 lsrootaf
  6 lsrootag
 66 总用量
```

这个也是，lsroot 前的减号，就是取用前面的 stdout