第13章 学习 shell脚本之前的基础知识

学习Linux请加QQ群： 群1(163262181) 群2(148412746) 群3(246401509) 群4(173884211)

跟阿铭学Linux邀请函 (http://www.aminglinux.com)，**狼课已上线，请加微信aminglinux84索要配套视频教程。**

日常的linux系统管理工作中必不可少的就是shell脚本，如果不会写shell脚本，那么你就不算一个合格的管理员。目前很多单位在招聘linux系统管理员时，shell脚本的编写是必考的项目。有的单位甚至用shell脚本的编写能力来衡量这个linux系统管理员的经验是否丰富。阿铭讲这些的目的只有一个，那就是让你认真对待shell脚本，从一开始就要把基础知识掌握牢固，然后要不断的练习，只要你shell脚本写的好，相信你的linux求职路就会轻松的多。阿铭在这一章中并不会多么详细的介绍shell脚本，而只是带你进入shell脚本的世界，如果你很感兴趣那么请到网上下载相关的资料或者到书店购买相关书籍吧。

在学习shell 脚本之前，需要你了解很多关于shell的知识，这些知识是编写shell脚本的基础，所以希望你能够熟练的掌握。

什么是shell

简单点理解，就是系统跟计算机硬件交互时使用的中间介质，它只是系统的一个工具。实际上，在shell和计算机硬件之间还有一层东西那就是系统内核了。打个比方，如果把计算机硬件比作一个人的躯体，而系统内核则是人的大脑，至于shell，把它比作人的五官似乎更加贴切些。回到计算机上来，用户直接面对的不是计算机硬件而是shell，用户把指令告诉shell，然后shell再传输给系统内核，接着内核再去支配计算机硬件去执行各种操作。

阿铭接触的linux发布版本（Redhat/CentOS）系统默认安装的shell叫做bash，即Bourne Again Shell，它是sh（Bourne Shell）的增强版本。Bourn Shell 是最早行起来的一个shell，创始人叫Steven Bourne，为了纪念他所以叫做Bourn Shell，检称sh。那么这个bash有什么特点呢？

1. 记录命令历史

我们敲过的命令，linux是会有记录的，预设可以记录1000条历史命令。这些命令保存在用户的家目录中的.bash_history文件中。有一点需要你知道的是，只有当用户正常退出当前shell时，在当前shell中运行的命令才会保存至.bash_history文件中。

与命令历史有关的有一个有意思的字符那就是 '!' 了。常用的有这么几个应用：

1） !! 连续两个 '!', 表示执行上一条指令：

```
[root@localhost ~]# pwd
/root
[root@localhost ~]# !!
pwd
/root
```

2） !n 这里的n是数字，表示执行命令历史中第n条指令，例如 `!1002` 表示执行命令历史中第1002个命令：

```
[root@localhost ~]# history |grep 1002
 1002  pwd
 1015  history |grep 1002
[root@localhost ~]# !1002
pwd
/root
```

`history` 命令如果未改动过环境变量，默认可以把最近1000条命令历史打印出来。

3） !字符串 （字符串大于等于1），例如 `!pw` 表示执行命令历史中最近一次以 'pw' 为开头的指令。

```
[root@localhost ~]# !pw
pwd
/root
```

2. 指令和文件名补全

最开始阿铭就介绍过这个功能了，记得吗？它就是按tab键，它可以帮你补全一个指令，也可以帮你补全一个路径或者一个文件名。连续按两次tab键，系统则会把所有的指令或者文件名都列出来。

3. 别名

前面也出现过alias的介绍，这个就是bash所特有的功能之一了。我们可以通过alias把一个常用的并且很长的指令别名一个简洁易记的指令。如果不想用了，还可以用unalias解除别名功能。直接敲alias会看到目前系统预设的alias.

系统预设的alias指令也就这几个而已，你也可以自定义你想要的指令别名。alias语法很简单， `alias [命令别名]=['具体的命令']`

```
[root@localhost ~]# alias aming='pwd'
[root@localhost ~]# aming
/root
[root@localhost ~]# unalias aming
[root@localhost ~]# aming
bash: aming: command not found
```

使用 `unalias 命令别名` 就可以把设置的别名给解除了。

4. 通配符

在bash下，可以使用 `*` 来匹配零个或多个字符，而用 `?` 匹配一个字符。

```
[root@localhost ~]# ls -d test*
test1.txt  test2  test3  test.pl  test.txt
[root@localhost ~]# ls -d test?
test2  test3
```

5. 输入输出重定向

输入重定向用于改变命令的输入，输出重定向用于改变命令的输出。输出重定向更为常用，它经常用于将命令的结果输入到文件中，而不是屏幕上。输入重定向的命令是<，输出重定向的命令是>，另外还有错误重定向2>，以及追加重定向>>，稍后会详细介绍。

6. 管道符

前面已经提过过管道符“|”，就是把前面的命令运行的结果丢给后面的命令。

7. 作业控制

当运行一个进程时，你可以使它暂停（按Ctrl+z），然后使用fg命令恢复它，利用bg命令使他到后台运行，你也可以使它终止（按Ctrl+c）。

```
[root@localhost ~]# vi test1.txt
testtestssttstst
```

阿铭使用“vi”编辑test1.txt，随便输入一些内容，按“ESC”后， 使用“Ctrl + z”使任务暂停：

```
[root@localhost ~]# vi test1.txt

[1]+  Stopped                 vi test1.txt
```

可以看到提示“vi test1.txt”已经停止了，然后使用fg命令恢复它，此时又进入刚才的“vi”窗口了。再次使其暂停，然后输入 `jobs`，可以看到在被暂停或者在后台运行的任务：

```
[root@localhost ~]# jobs
[1]+  Stopped                 vi test1.txt
```

如果想把暂停的任务丢在后台跑起来，就使用bg命令：

```
[root@localhost ~]# bg
[1]+ vi test1.txt &

[1]+  Stopped                 vi test1.txt
```

但是 vi 似乎并不支持在后台运行，那阿铭换一个其他的命令：

```
[root@localhost ~]# vmstat 1 > /tmp/1.log
^Z
[2]+  Stopped                 vmstat 1 > /tmp/1.log
[root@localhost ~]# jobs
[1]-  Stopped                 vi test1.txt
[2]+  Stopped                 vmstat 1 > /tmp/1.log
[root@localhost ~]# bg 2
[2]+ vmstat 1 > /tmp/1.log &
```

在上面的例子中，又有一个新的知识点需要你知道，那就是多个被暂停的任务会有编号，使用 jobs 命令可以看到两个任务，那么使用bg或者fg的时候，就需要在后面加一个编号了，阿铭使用 `bg 2` 把第二个被暂停的任务丢到后台跑起来了，丢入后台需要使用在命令后边加一个 `&` 符号，中间有个空格。本例中的 `vmstat 1` 这个是用来观察系统状态的一个命令，后面章节阿铭再介绍。

丢到后台的任务如何关掉呢？如果你没有退出刚才的shell，那么先使用 `fg 编号` 把任务调到前台，然后使用“Ctrl + c”结束任务：

```
[root@localhost ~]# fg 2
vmstat 1 > /tmp/1.log
^C
```

另一种情况则是，关闭到当前的shell，再次打开另一个shell时，使用jobs命令并不会显示在后台运行或者被暂停的任务，要想停掉它的话，则需要先知道其pid，然后使用kill命令杀死那个进程。

```
[root@localhost ~]# vmstat 1 > /tmp/1.log &
[1] 9433
[root@localhost ~]# ps aux |grep vmstat
root      9433  0.0  0.0   6180   516 pts/2    S    09:57   0:00 vmstat 1
root      9435  0.0  0.0 103308   848 pts/2    S+   09:58   0:00 grep vmstat
```

使用 `&` 把任务丢入后台运行，它会显示pid信息，如果忘记这个pid，我们还可以使用 `ps aux` 命令找到那个进程，关于 ps 这个命令阿铭会在后面讲解的。想结束掉该进程，需要使用 kill 命令：

```
[root@localhost ~]# kill 9433
[1]+  已终止               vmstat 1 > /tmp/1.log
```

kill命令语法很简单，直接在后面加pid即可，如果遇到杀不死的进程时，可以在kill 后面加一个选项： `kill -9 [pid]`

变量

前面章节中阿铭曾经介绍过环境变量PATH，这个环境变量就是shell预设的一个变量，通常shell预设的变量都是大写的。变量，说简单点就是使用一个较简单的字符串来替代某些具有特殊意义的设定以及数据。就拿PATH来讲，这个PATH就代替了所有常用命令的绝对路径的设定。因为有了PATH这个变量，所以我们运行某个命令时不再去输入全局路径，直接敲命令名即可。你可以使用echo命令显示变量的值。

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

除了PATH, HOME, LOGNAME外，系统预设的环境变量还有哪些呢？

```
[root@localhost ~]# env
HOSTNAME=localhost.localdomain
```

```
TERM=xterm
SHELL=/bin/bash
HISTSIZE=1000
SSH_CLIENT=10.72.137.107 50947 22
SSH_TTY=/dev/pts/0
USER=root
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=01;05;37;41:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;
MAIL=/var/spool/mail/root
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
PWD=/root
LANG=zh_CN.UTF-8
HISTCONTROL=ignoredups
SHLVL=1
HOME=/root
LOGNAME=root
SSH_CONNECTION=10.72.137.107 50947 10.72.137.159 22
LESSOPEN=|/usr/bin/lesspipe.sh %s
G_BROKEN_FILENAMES=1
_=/bin/env
```

使 用 `env` 命令即可全部列出系统预设的全部系统变量了。不过登录的用户不一样这些环境变量的值也不一样。当前显示的就是root这个账户的环境变量了。下面阿铭简单介绍一下常见的环境变量：

P A T H 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　　HISTSIZE 历史记录数　　LOGNAME 当前用户的登录名 HOSTNAME 指主机的名称 SHELL 前用户Shell类型 LANG 语言相关的环境变量，多语言可以修改此环境变量 MAIL 当前用户的邮件存放目录 PWD 当前目录

env命令显示的变量只是环境变量，系统预设的变量其实还有很多，你可以使用set命令把系统预设的全部变量都显示出来。

```
[root@localhost ~]# set
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:expand_aliases:extquote:force_fignore:hostcomplete:interactive_comments:login_shell:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="4" [1]="1" [2]="2" [3]="1" [4]="release" [5]="i386-redhat-linux-gnu")
BASH_VERSION='4.1.2(1)-release'
COLORS=/etc/DIR_COLORS
COLUMNS=168
DIRSTACK=()
EUID=0
GROUPS=()
G_BROKEN_FILENAMES=1
HISTCONTROL=ignoredups
HISTFILE=/root/.bash_history
HISTFILESIZE=1000
HISTSIZE=1000
HOME=/root
HOSTNAME=localhost.localdomain
HOSTTYPE=i386
ID=0
IFS=$' \t\n'
LANG=zh_CN.UTF-8
LESSOPEN='|/usr/bin/lesspipe.sh %s'
LINES=44
LOGNAME=root
LS_COLORS='rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=01;05;37;41:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;
MACHTYPE=i386-redhat-linux-gnu
MAIL=/var/spool/mail/root
MAILCHECK=60
OLDPWD=/root
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
PIPESTATUS=([0]="0")
PPID=27377
PROMPT_COMMAND='printf "\033]0;%s@%s:%s\007" "${USER}" "${HOSTNAME%%.*}" "${PWD/#$HOME/~}"'
PS1='[\u@\h \W]\$ '
PS2='> '
PS4='+ '
PWD=/root
SHELL=/bin/bash
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor
SHLVL=1
SSH_CLIENT='10.72.137.107 50947 22'
SSH_CONNECTION='10.72.137.107 50947 10.72.137.159 22'
SSH_TTY=/dev/pts/0
TERM=xterm
UID=0
USER=root
_=-
colors=/etc/DIR_COLORS
```

set不仅可以显示系统预设的变量，也可以连同用户自定义的变量显示出来。

```
[root@localhost ~]# myname=Aming
[root@localhost ~]# echo $myname
Aming
[root@localhost ~]# set |grep myname
myname=Aming
```

虽然你可以自定义变量，但是该变量只能在当前shell中生效。

```
[root@localhost ~]# echo $myname
Aming
[root@localhost ~]# bash
[root@localhost ~]# echo $myname

[root@localhost ~]# exit
exit
[root@localhost ~]# echo $myname
Aming
```

使用 `bash` 命令即可再打开一个shell，此时先前设置的“myname”变量已经不存在了，退出当前shell回到原来的shell，“myname”变量还在。那要想设置的变量一直生效怎么办？有两种情况：

1） 要想系统内所有用户登录后都能使用该变量

需要在“/etc/profile”文件最末行加入 `export myname=Aming` 然后运行 `source /etc/profile` 就可以生效了。此时再运行bash命令或者直接 `su - test` 账户可以看到效果。

```
[root@localhost ~]# echo "export myname=Aming" >> /etc/profile
[root@localhost ~]# source !$
source /etc/profile
[root@localhost ~]# bash
[root@localhost ~]# echo $myname
Aming
[root@localhost ~]# exit
exit
[root@localhost ~]# su - test
[test@localhost ~]$ echo $myname
Aming
```

2）只想让当前用户使用该变量

需要在用户主目录下的 `.bashrc` 文件最后一行加入 `export myname=Aming` 然后运行 `source .bashrc` 就可以生效了。这时候再登录test账户，myname变量则不会生效了。上面用的source命令的作用是，将目前设定的配置刷新，即不用注销再登录也能生效。

阿铭在上例中使用 `myname=Aming` 来设置变量myname，那么在linux下设置自定义变量有哪些规则呢？

1. 设定变量的格式为“a=b”，其中a为变量名，b为变量的内容，等号两边不能有空格；
2. 变量名只能由英、数字以及下划线组成，而且不能以数字开头；
3. 当变量内容带有特殊字符（如空格）时，需要加上单引号；

```
[root@localhost ~]# myname='Aming Li'
[root@localhost ~]# echo $myname
Aming Li
```

有一种情况，需要你注意，就是变量内容中本身带有单引号，这就需要用到双引号了。

```
[root@localhost ~]# myname="Aming's"
[root@localhost ~]# echo $myname
Aming's
```

4. 如果变量内容中需要用到其他命令运行结果则可以使用反引号；

```
[root@localhost ~]# myname=`pwd`
[root@localhost ~]# echo $myname
/root
```

5. 变量内容可以累加其他变量的内容，需要加双引号；

```
[root@localhost ~]# myname="$LOGNAME"Aming
[root@localhost ~]# echo $myname
rootAming
```

在这里如果你不小心把双引号加错为单引号，将得不到你想要的结果

```
[root@localhost ~]# myname='$LOGNAME'Aming
[root@localhost ~]# echo $myname
$LOGNAMEAming
```

通过上面几个例子也许你能看得出，单引号和双引号的区别，用双引号时不会取消掉里面出现的特殊字符的本身作用（这里的$），而使用单引号则里面的特殊字符全部失去它本身的作用。

在前面的例子中阿铭多次使用了 `bash` 命令，如果在当前shell中运行bash指令后，则会进入一个新的shell，这个shell就是原来shell的子shell了，不妨你用pstree指令来查看一下。

```
[root@localhost ~]# pstree |grep bash
     |-login---bash
     |-sshd---sshd---bash-+-grep
[root@localhost ~]# bash
[root@localhost ~]# pstree |grep bash
     |-login---bash
     |-sshd---sshd---bash---bash-+-grep
```

`pstree` 这个指令会把linux系统中所有进程通过树形结构打印出来。限于篇幅阿铭没有全部列出，你可以直接输入pstree查看即可。在父shell中设定一个变量后，进入子shell后该变量是不会生效的，如果想让这个变量在子shell中生效则要用到export指令。

```
[root@localhost ~]# abc=123
[root@localhost ~]# echo $abc
123
[root@localhost ~]# bash
[root@localhost ~]# echo $abc

[root@localhost ~]# exit
exit
[root@localhost ~]# export abc
[root@localhost ~]# echo $abc
```

```
123
[root@localhost ~]# bash
[root@localhost ~]# echo $abc
123
```

export其实就是声明一下这个变量的意思，让该shell的子shell也知道变量abc的值是123.如果export后面不加任何变量名，则它会声明所有的变量。

```
[root@localhost ~]# export
declare -x G_BROKEN_FILENAMES="1"
declare -x HISTCONTROL="ignoredups"
declare -x HISTSIZE="1000"
declare -x HOME="/root"
declare -x HOSTNAME="localhost.localdomain"
declare -x LANG="zh_CN.UTF-8"
declare -x LESSOPEN="|/usr/bin/lesspipe.sh %s"
declare -x LOGNAME="root"
declare -x LS_COLORS="rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=01;05;37;41:su=37;41:sg=30;43:ca=30;41:tw=
declare -x MAIL="/var/spool/mail/root"
declare -x OLDPWD
declare -x PATH="/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin"
declare -x PWD="/root"
declare -x SHELL="/bin/bash"
declare -x SHLVL="3"
declare -x SSH_CLIENT="10.72.137.107 50947 22"
declare -x SSH_CONNECTION="10.72.137.107 50947 10.72.137.159 22"
declare -x SSH_TTY="/dev/pts/0"
declare -x TERM="xterm"
declare -x USER="root"
declare -x abc="123"
declare -x myname="\$LOGNAMEAming"
```

在最后面连同我们自定义的变量都被声明了。

前面光讲如何设置变量，如果想取消某个变量怎么办？只要输入 `unset 变量名` 即可。

```
[root@localhost ~]# echo $abc
123
[root@localhost ~]# unset abc
[root@localhost ~]# echo $abc
```

系统环境变量与个人环境变量的配置文件

上面讲了很多系统的变量，那么在linux系统中，这些变量被存到了哪里呢，为什么用户一登陆shell就自动有了这些变量呢？

/etc/profile ：这个文件预设了几个重要的变量，例如PATH, USER, LOGNAME, MAIL, INPUTRC, HOSTNAME, HISTSIZE, umask等等。

/etc/bashrc ：这个文件主要预设umask以及PS1。这个PS1就是我们在敲命令时，前面那串字符了，例如阿铭的linux系统PS1就是 [root@localhost ~]#, 我们不妨看一下PS1的值：

```
[root@localhost ~]# echo $PS1
[\u@\h \W]\$
```

`\u` 就是用户， `\h` 主机名， `\W` 则是当前目录，`\$` 就是那个‘#’了，如果是普通用户则显示为‘$’.

除了两个系统级别的配置文件外，每个用户的主目录下还有几个这样的隐藏文件：

.bash_profile ：定义了用户的个人化路径与环境变量的文件名称。每个用户都可使用该文件输入专用于自己使用的shell信息,当用户登录时,该文件仅仅执行一次。

.bashrc ：该文件包含专用于你的shell的bash信息,当登录时以及每次打开新的shell时,该该文件被读取。例如你可以将用户自定义的alias或者自定义变量写到这个文件中。

.bash_history ：记录命令历史用的。

.bash_logout ：当退出shell时，会执行该文件。可以把一些清理的工作放到这个文件中。

linux shell中的特殊符号

你在学习linux的过程中，也许你已经接触过某个特殊符号，例如”*”，它是一个通配符号，代表零个或多个字符或数字。下面阿铭就说一说常用到的特殊字符。

1. `*` 代表零个或多个任意字符。

```
[root@localhost ~]# ls -d test*
test  test1  test2  test3
```

2. `?` 只代表一个任意的字符

```
[root@localhost ~]# touch testa
[root@localhost ~]# touch testb.txt
[root@localhost ~]# ls -d test?
test1  test2  test3  testa
```

不管是数字还是字母，只要是一个都能匹配出来。

3. `#` 这个符号在linux中表示注释说明的意思，即 `#` 后面的内容linux忽略掉。

```
[root@localhost ~]# abc=123 #aaaaa
[root@localhost ~]# echo $abc
123
```

4. `\` 脱意字符，将后面的特殊符号（例如”*”）还原为普通字符。

```
[root@localhost ~]# ls -d test\*
```

```
ls: 无法访问test*: 没有那个文件或目录
```

5. `|` 管道符，前面多次出现过，它的作用在于将符号前面命令的结果丢给符号后面的命令。这里提到的后面的命令，并不是所有的命令都可以的，一般针对文档操作的命令比较常用，例如cat， less， head, tail, grep, cut, sort, wc, uniq, tee, tr, split, sed, awk等等，其中grep, sed, awk为正则表达式必须掌握的工具，在后续内容中详细介绍。

```
[root@localhost ~]# cat testb.txt |wc -l
0
```

`wc -l` 用来计算一个文档有多少行。在这里阿铭一下子列出来很多对你陌生的命令，其实这些命令在日常的处理文档工作中非常实用，所以阿铭需要先简单介绍一下它们，如果你记不住没有关系，以后用到的时候再过来查或者直接用man来查询帮助文档。

命令：cut

用来截取某一个字段

语法： `cut -d '分隔字符' [-cf] n` 这里的n是数字

-d ： 后面跟分隔字符，分隔字符要用单引号括起来

-c ： 后面接的是第几个字符

-f ： 后面接的是第几个区块

```
[root@localhost ~]# cat /etc/passwd |cut -d ':' -f 1 |head -n5
root
bin
daemon
adm
lp
```

-d 后面跟分隔字符，这里使用冒号作为分割字符，-f 1 就是截取第一段，-f和1之间的空格可有可无。

```
[root@localhost ~]# head -n2 /etc/passwd|cut -c2
o
i
[root@localhost ~]# head -n2 /etc/passwd|cut -c1
r
b
[root@localhost ~]# head -n2 /etc/passwd|cut -c1-10
root:x:0:0
bin:x:1:1:
[root@localhost ~]# head -n2 /etc/passwd|cut -c5-10
:x:0:0
x:1:1:
```

-c 后面可以是1个数字n，也可以是一个区间n1-n2，还可以是多个数字n1,n2,n3

```
[root@localhost ~]# head -n2 /etc/passwd|cut -c1,3,10
ro0
bn:
```

命令：sort

sort 用做排序

语法： `sort [-t 分隔符] [-kn1,n2] [-nru]` 这里的n1 < n2

-t 分隔符 ： 作用跟cut的-d一个意思

-n ： 使用纯数字排序

-r ： 反向排序

-u ： 去重复

-kn1,n2 ： 由n1区间排序到n2区间，可以只写-kn1，即对n1字段排序

```
[root@localhost ~]# head -n5 /etc/passwd |sort
adm:x:3:4:adm:/var/adm:/sbin/nologin
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
root:x:0:0:root:/root:/bin/bash
```

如果sort不加任何选项，则从首字符向后，依次按ASCII码值进行比较，最后将他们按升序输出。

```
[root@localhost ~]# head -n5 /etc/passwd |sort -t: -k3 -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
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

-t 后面跟分隔符，-k后面跟数字，表示对第几个区域的字符串排序，-n 则表示使用纯数字排序

```
[root@localhost ~]# head -n5 /etc/passwd |sort -t: -k3,5 -r
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
bin:x:1:1:bin:/bin:/sbin/nologin
root:x:0:0:root:/root:/bin/bash
```

-k3,5 表示从第3到第5区域间的字符串排序，-r表示反向排序

命令：wc

用于统计文档的行数、字符数、词数，常用的选项为：

-l ： 统计行数

-m ： 统计字符数

-w ： 统计词数

```
[root@localhost ~]# wc /etc/passwd
  27   37 1220 /etc/passwd
```

```
[root@localhost ~]# wc -l /etc/passwd
27 /etc/passwd
[root@localhost ~]# wc -m /etc/passwd
1220 /etc/passwd
[root@localhost ~]# wc -w /etc/passwd
37 /etc/passwd
```

wc 不跟任何选项，直接跟文档，则会把行数、词数、字符数依次输出。

命令：uniq

去重复的行，阿铭最常用的选项只有一个：

-c ： 统计重复的行数，并把行数写在前面

```
[root@localhost ~]# vim testb.txt
```

把下面的内容写入testb.txt, 保存。

```
111
222
111
333
```

使用uniq 的前提是需要先给文件排序，否则不管用。

```
[root@localhost ~]# uniq testb.txt
111
222
111
333
[root@localhost ~]# sort testb.txt |uniq
111
222
333
[root@localhost ~]# sort testb.txt |uniq -c
      2 111
      1 222
      1 333
```

命令：tee

后跟文件名，类似与重定向 “>”, 但是比重定向多了一个功能，在把文件写入后面所跟的文件中的同时，还显示在屏幕上。

```
[root@localhost ~]# echo "aaaaaaaaaaaaaaaaaaaaaaaaaaa" |tee testb.txt
aaaaaaaaaaaaaaaaaaaaaaaaaaa
[root@localhost ~]# cat testb.txt
aaaaaaaaaaaaaaaaaaaaaaaaaaa
```

tee 常用语管道符 “|” 后。

命令：tr

替换字符，常用来处理文档中出现的特殊符号，如DOS文档中出现的^M符号。常用的选项有两个：

-d ： 删除某个字符，-d 后面跟要删除的字符

-s ： 把重复的字符去掉

最常用的就是把小写变大写: tr ‘[a-z]’ ‘[A-Z]’

```
[root@localhost ~]# head -n2 /etc/passwd |tr '[a-z]' '[A-Z]'
ROOT:X:0:0:ROOT:/ROOT:/BIN/BASH
BIN:X:1:1:BIN:/BIN:/SBIN/NOLOGIN
```

当然替换一个字符也是可以的。

```
[root@localhost ~]# grep 'root' /etc/passwd |tr 'r' 'R'
Root:x:0:0:Root:/Root:/bin/bash
opeRatoR:x:11:0:opeRatoR:/Root:/sbin/nologin
```

不过替换、删除以及去重复都是针对一个字符来讲的，有一定局限性。如果是针对一个字符串就不再管用了，所以阿铭建议你只需简单了解这个tr即可，以后你还会学到更多可以实现针对字符串操作的工具。

命令：split

切割文档，常用选项：

-b ： 依据大小来分割文档，单位为byte

```
[root@localhost ~]# mkdir split_dir
[root@localhost ~]# cd !$
cd split_dir
[root@localhost split_dir]# cp /etc/passwd ./
[root@localhost split_dir]# split -b500 passwd
[root@localhost split_dir]# ls
passwd  xaa  xab  xac
```

如果split不指定目标文件名，则会以xaa xab... 这样的文件名来存取切割后的文件。当然我们也可以指定目标文件名：

```
[root@localhost split_dir]# split  -b500 passwd  123
[root@localhost split_dir]# ls
123aa  123ab  123ac  passwd
```

-l ： 依据行数来分割文档

```
[root@localhost split_dir]# rm -f 123a*
[root@localhost split_dir]# split -l10 passwd
[root@localhost split_dir]# wc -l *
  27 passwd
  10 xaa
  10 xab
  7 xac
  54 总用量
```

6. `$` 除了用于变量前面的标识符外，还有一个妙用，就是和 '!' 结合起来使用。

```
[root@localhost ~]# ls testb.txt
testb.txt
[root@localhost ~]# ls !$
ls testb.txt
testb.txt
```

'!$' 表示上条命中中最后一个变量（总之就是上条命令中最后出现的那个东西）例如上边命令最后是testb.txt那么在当前命令下输入!$则代表testb.txt.

7. `;` ：分号。平时我们都是在一行中敲一个命令，然后回车就运行了，那么想在一行中运行两个或两个以上的命令如何呢？则需要在命令之间加一个 ";" 了。

```
[root@localhost ~]# ls -d test*;  touch test111; ls -d test*
test  test1  test2  test3  testa  testb.txt
test  test1  test111  test2  test3  testa  testb.txt
```

8. `~` ：用户的家目录，如果是root则是 /root ， 普通用户则是 /home/username

```
[root@localhost ~]# cd ~
[root@localhost ~]# pwd
/root
[root@localhost ~]# su test
[test@localhost root]$ cd ~
[test@localhost ~]$ pwd
/home/test
```

9. `&` ： 如果想把一条命令放到后台执行的话，则需要加上这个符号。通常用于命令运行时间非常长的情况。

```
[root@localhost ~]# sleep 30 &
[1] 3260
[root@localhost ~]# jobs
[1]+  Running                 sleep 30 &
```

10. >, >>, 2>, 2>> 前面讲过重定向符号> 以及>> 分别表示取代和追加的意思，然后还有两个符号就是这里的2> 和 2>> 分别表示错误重定向和错误追加重定向，当我们运行一个命令报错时，报错信息会输出到当前的屏幕，如果想重定向到一个文本里，则要用2>或者2>>

```
[root@localhost ~]# ls aaaa
ls: 无法访问aaaa: 没有那个文件或目录
[1]+  Done                    sleep 30
[root@localhost ~]# ls aaaa
ls: 无法访问aaaa: 没有那个文件或目录
[root@localhost ~]# ls aaaa 2> /tmp/error
[root@localhost ~]# cat /tmp/error
ls: 无法访问aaaa: 没有那个文件或目录
[root@localhost ~]# ls aaaa 2>> /tmp/error
[root@localhost ~]# cat /tmp/error
ls: 无法访问aaaa: 没有那个文件或目录
ls: 无法访问aaaa: 没有那个文件或目录
```

11. [ ] 中括号，中间为字符组合，代表中间字符中的任意一个。

```
[root@localhost ~]# ls -d test*
test  test1  test111  test2  test3  testa  testb.txt
[root@localhost ~]# ls -d test[1-3]
test1  test2  test3
[root@localhost ~]# ls -d test[1a3]
test1  test3  testa
[root@localhost ~]# ls -d test[0-9]
test1  test2  test3
[root@localhost ~]# ls -d test[0-9a-z]
test1  test2  test3  testa
```

12. && 与 ||

在上面刚刚提到了分号，用于多条命令间的分隔符。另外还有两个可以用于多条命令中间的特殊符号，那就是"&&" 和 "||" 下面阿铭把这几种情况全列出：

1. command1 ; command2
2. command1 && command2
3. command1 || command2

使用 ";" 时，不管command1是否执行成功都会执行command2；

使用 "&&" 时，只有command1执行成功后，command2才会执行，否则command2不执行；

使用 "||" 时，command1执行成功后command2 不执行，否则去执行command2，总之command1和command2总有一条命令会执行。

在做实验前，阿铭想把所有的 test* 删除掉，可是删除的时候，却提示说权限不够，下面是阿铭排除问题的过程：

```
[root@localhost ~]# rm -rf test*
rm: 无法删除"test2/test1": 权限不够
rm: 无法删除"test2/test3": 权限不够
rm: 无法删除"test2/test4": 权限不够
[root@localhost ~]# ls test*
test1  test3  test4
[root@localhost ~]# lsattr test*
-----a-------e- test2/test1
```

```
----i--------e- test2/test3
-------------e- test2/test4
[root@localhost ~]# chattr -a test2/test1
[root@localhost ~]# chattr -i test2/test3
[root@localhost ~]# rm -rf test*
rm: 无法删除"test2/test1": 权限不够
rm: 无法删除"test2/test3": 权限不够
rm: 无法删除"test2/test4": 权限不够
[root@localhost ~]# ls test*
test1  test3  test4
[root@localhost ~]# ls -ld test*
drwxrwxr-x 2 root root 4096 5月  10 10:12 test2
[root@localhost ~]# ls -l test2/*
-rw-r--r-- 1 root root 6 5月  10 10:20 test2/test1
-rw-r--r-- 1 root root 0 5月  10 10:11 test2/test3
-rw-r--r-- 1 root root 0 5月  10 10:12 test2/test4
[root@localhost ~]# lsattr test2/*
-------------e- test2/test1
-------------e- test2/test3
-------------e- test2/test4
[root@localhost ~]# lsattr test2
-------------e- test2/test1
-------------e- test2/test3
-------------e- test2/test4
[root@localhost ~]# lsattr -d test2
----i--------e- test2
[root@localhost ~]# chattr -i test2/
[root@localhost ~]# rm -rf test2/
```

如果你之前跟着阿铭做过同样的实验，相信你也会出现同样的问题的。接下来阿铭要通过做实验来说明“&&”与“||”这两个特殊符号的作用：

```
[root@localhost ~]# touch test1 test3
[root@localhost ~]# ls test2 && touch test2
ls: 无法访问test2: 没有那个文件或目录
[root@localhost ~]# ls test2
ls: 无法访问test2: 没有那个文件或目录
[root@localhost ~]# ls test2 || touch test2
ls: 无法访问test2: 没有那个文件或目录
[root@localhost ~]# ls test*
test1  test2  test3
```

阿铭建议你最好再扩展学习一下： http://www.aminglinux.com/bbs/thread-5437-1-1.html

教程答疑：请移步这里.

欢迎你加入 阿铭学院 和阿铭一起学习Linux，让阿铭成为你Linux生涯中永远的朋友吧！