Shell 简介

不使用 shell 脚本的情况：

- 资源密集型的任务,尤其在需要考虑效率时(比如,排序,hash 等等)
- 需要处理大任务的数学操作,尤其是浮点运算,精确运算,或者复杂的算术运算
- (这种情况一般使用 C++或 FORTRAN 来处理)
- 有跨平台移植需求(一般使用 C 或 Java)
- 复杂的应用,在必须使用结构化编程的时候(需要变量的类型检查,函数原型,等等)
- 对于影响系统全局性的关键任务应用。
- 对于安全有很高要求的任务,比如你需要一个健壮的系统来防止入侵,破解,恶意破坏等等.
- 项目由连串的依赖的各个部分组成。
- 需要大规模的文件操作
- 需要多维数组的支持
- 需要数据结构的支持,比如链表或数等数据结构
- 需要产生或操作图形化界面 GUI
- 需要直接操作系统硬件
- 需要 I/O 或 socket 接口
- 需要使用库或者遗留下来的老代码的接口
- 私人的,闭源的应用(shell 脚本把代码就放在文本文件中,全世界都能看到)

在脚本的开头需要加上**#!**，意味着系统文件的执行需要一个解释器。解释脚本命令的程序有如下几个：

```
1 #!/bin/sh
2 #!/bin/bash
3 #!/usr/bin/perl
4 #!/usr/bin/tcl
5 #!/bin/sed -f
6 #!/usr/awk -f
```

上边每一个脚本头的行都指定了一个不同的命令解释器,如果是/bin/sh,那么就是默认shell (在Linux 系统中默认是Bash).[3]使用#!/bin/sh,在大多数商业发行的UNIX 上,默认是 Bourne shell,这将脚本可以正常的运行在非Linux 机器上,虽然这将会牺牲Bash 一些独特的特征.

注意: #! 后边给出的路径名必须是正确的,否则将会出现一个错误消息,通常是 "Command not found",这将是你运行这个脚本时所得到的唯一结果.

当然"#!"也可以被忽略,不过这样你的脚本文件就只能是一些命令的集合,不能够使用shell 内建
的指令了

调用脚本

方法：
sh scriptname
bash scriptname
如果脚本具有可执行权限则可以用下面方法进行调用：
./scriptname
注意事项:
[1] 那些具有UNIX 味道的脚本(基于4.2BSD)需要一个4 字节的魔法数字,在#!后边需要一个空格#! /bin/sh.
[2] 脚本中的#!行的最重要的任务就是命令解释器(sh 或者bash).因为这行是以#开始的,当命令解释器执行这个脚本的时候,会把它作为一个注释行.当然,在这之前,这行语句已经完成了它的任务,就是调用命令解释器.
如果在脚本的里边还有一个#!行,那么bash 将把它认为是一个一般的注释行.

```
#!/bin/bash

echo "Part 1 of script."
a=1
#!/bin/bash
# 这将不会开始一个新脚本.
echo "Part 2 of script."
echo $a # Value of $a stays at 1.
```

[3] 可移植的操作系统接口,标准化类UNIX 操作系统的一种尝试.POSIX 规范可以在http://www.opengroup.org/onlinepubs/007904975/toc.htm 中查阅.
[4] 小心:使用sh scriptname 来调用脚本的时候将会关闭一些Bash 特定的扩展,脚本可能因此而调用失败.
[5] 脚本需要读和执行权限,因为shell 需要读这个脚本.
[6] 为什么不直接使用scriptname 来调用脚本?如果你当前的目录下($PWD)正好有你想要执行的脚本,为什么它运行不了呢?失败的原因是,出于安全考虑,当前目录并没有被加在用户的$PATH 变量中.因此,在当前目录下调用脚本必须使用./scriptname 这种形式.

特殊字符

. 命令等价于source 命令(见Example 11-20).这是一个bash 的内建命令.
. 作为文件名的一部分.如果作为文件名的前缀的话,那么这个文件将成为隐藏文件.将不被 ls 命令列出.
. 命令如果作为目录名的一部分的话,那么.表达的是当前目录.".."表示上一级目录.
. 命令经常作为一个文件移动命令的目的地.
. 字符匹配,这是作为正则表达是的一部分,用来匹配任何的单个字符.

" 部分引用."STRING"阻止了一部分特殊字符,具体见第5 章.
' 全引用. 'STRING' 阻止了全部特殊字符,具体见第5 章.
, 逗号链接了一系列的算术操作,虽然里边所有的内容都被运行了,但只有最后一项被返回.
如:
1 let "t2 = ((a = 9, 15 / 3))" # Set "a = 9" and "t2 = 15 / 3"
\ 转义字符,如\X 等价于"X"或'X',具体见第5 章.
/ 文件名路径分隔符.或用来做除法操作.
` 后置引用,命令替换,具体见第14 章
: 空命令,等价于"NOP"(no op,一个什么也不干的命令).也可以被认为与shell 的内建命令(true)作用相同.":"命令是一个 bash 的内建命令,它的返回值为0,就是shell 返回的true.
! 取反操作符,将反转"退出状态"结果,(见Example 6-2).也会反转test 操作符的意义.比如修改=为!=.!操作是Bash 的一个关键字.
在一个不同的上下文中,!也会出现在"间接变量引用"见Example 9-22.
在另一种上下文中,!还能反转bash 的"history mechanism"(见附录J 历史命令)
需要注意的是,在一个脚本中,"history mechanism"是被禁用的.
* 万能匹配字符,用于文件名匹配(这个东西有个专有名词叫file globbing),或者是正则表达式中.注意:在正则表达式匹配中的作用和在文件名匹配中的作用是不同的.
bash$ echo *
abs-book.sgml add-drive.sh agram.sh alias.sh
* 数学乘法.
**是幂运算.
? 测试操作.在一个确定的表达式中,用?来测试结果.
(())结构可以用来做数学计算或者是写c 代码,那?就是c 语言的3 元操作符的一个.
在"参数替换"中,?测试一个变量是否被set 了.
? 在file globbing 中和在正则表达式中一样匹配任意的单个字符.
$ 变量替换
$ 在正则表达式中作为行结束符.
${} 参数替换,见9.3 节.
$*,$@ 位置参数
$? 退出状态变量.$?保存一个命令/一个函数或者脚本本身的退出状态.
$$ 进程ID 变量.这个$$变量保存运行脚本进程ID
() 命令组.如:
1 (a=hello;echo $a)
注意:在()中的命令列表,将作为一个子shell 来运行.
在()中的变量,由于是在子shell 中,所以对于脚本剩下的部分是不可用的.

{xxx,yyy,zzz...}
大括号扩展,
一个命令可能会对大括号中的以逗号分割的文件列表起作用[1]. file globbing 将对大括号中的文件名作扩展.
注意: 在大括号中,不允许有空白,除非这个空白是有意义的.
echo {file1,file2}\ :{\ A," B",' C'}

file1：A file1：B file1：C file2：A file2：B file2：C
{} 代码块.又被称为内部组.事实上,这个结构创建了一个匿名的函数.但是与函数不同的是,在其中声明的变量,对于脚本其他部分的代码来说还是可见的.如:
bash$
{
local a;
a= 123;
}
bash中的local 申请的变量只能够用在函数中.

```
a=123
{ a=321; }
echo "a = $a" # a = 321 (说明在代码块中对变量 a 所作的修改,影响了外边的变量 a)
```

注意: 与()中的命令不同的是,{}中的代码块将不能正常地开启一个新shell.[2]
{} \; 路径名.一般都在find 命令中使用.这不是一个shell 内建命令.
注意: ";"用来结束find 命令序列的-exec 选项.
[] test.
test的表达式将在[]中.
值得注意的是[是shell 内建test 命令的一部分,并不是/usr/bin/test 中的扩展命令的一个连接.
[[]] test.
test表达式放在[[]]中.(shell 关键字)
具体查看[[]]结构的讨论.
[] 数组元素
Array[1]=slot_1
echo ${Array[1]}
[] 字符范围
在正则表达式中使用,作为字符匹配的一个范围
(()) 数学计算的扩展
在(())结构中可以使用一些数字计算.
具体参阅((...))结构.
>&>>&>><
重定向.
scriptname >filename 重定向脚本的输出到文件中.覆盖文件原有内容.
command &>filename 重定向stdout 和stderr 到文件中
command >&2 重定向command 的stdout 到stderr
scriptname >>filename 重定向脚本的输出到文件中.添加到文件尾端,如果没有文件,则创建这个文件.
进程替换,具体见"进程替换部分",跟命令替换极其类似.
(command)>

<(command)
<和> 可用来做字符串比较
<和> 可用在数学计算比较
<< 重定向,用在"here document"
<<< 重定向,用在"here string"
<,> ASCII 比较
\<,\> 正则表达式中的单词边界.如:
bash$grep '\<the\>' textfile
| 管道.分析前边命令的输出,并将输出作为后边命令的输入。
管道是进程间通讯的一个典型办法,将一个进程的stdout 放到另一个进程的stdin 中.
标准的方法是将一个一般命令的输出,比如cat 或echo,传递到一个过滤命令中(在这个
过滤命令中将处理输入),得到结果,如:
cat $filename1 | $filename2 | grep $search_word
>| 强制重定向(即使设置了noclobber 选项--就是-C 选项).这将强制的覆盖一个现存文件.
|| 或-逻辑操作.
&& 与-逻辑操作.
& 后台运行命令.一个命令后边跟一个&,将表示在后台运行.
在一个脚本中,命令和循环都可能运行在后台.
- 之前工作的目录."cd -"将回到之前的工作目录,具体请参考"$OLDPWD"环境变量.
注意:一定要和之前讨论的重定向功能分开,但是只能依赖上下文区分.
- 算术减号.
= 算术等号,有时也用来比较字符串.
+ 算术加号,也用在正则表达式中.
+ 选项,对于特定的命令来说使用"+"来打开特定的选项,用"-"来关闭特定的选项.
% 算术取模运算.也用在正则表达式中.
~ home 目录.相当于$HOME 变量.~bozo 是bozo 的home 目录,并且ls ~bozo 将列出其
中的
~+ 当前工作目录,相当于$PWD 变量.
~- 之前的工作目录,相当于$OLDPWD 内部变量.
=~ 用于正则表达式,这个操作将在正则表达式匹配部分讲解,只有version3 才支持.
^ 行首,正则表达式中表示行首."^"定位到行首.

注意:命令是不能跟在同一行上注释的后边的,没有办法,在同一行上,注释的后边想
要再使用命令,只能另起一行.
当然,在echo 命令中被转义的#是不能作为注释的.
同样的,#也可以出现在特定的参数替换结构中或者是数字常量表达式中.

```
1 echo "The # here does not begin a comment."
2 echo 'The # here does not begin a comment.'
3 echo The \# here does not begin a comment.
4 echo The # 这里开始一个注释
5
6 echo ${PATH#*:} # 参数替换,不是一个注释
7 echo $(( 2#101011 )) # 数制转换,不是一个注释
8
9 # Thanks, S.C.
```

标准的引用和转义字符(""\)可以用来转义#
命令分隔符,可以用来在一行中来写多个命令.

```
if [ -x "$filename" ];   then
      echo "File $filename exists."
      cp $filename $filename.bak
else
      echo "File $filename not found."
      touch $filename
fi
echo "File test complete."
```

控制字符

修改终端或文本显示的行为.控制字符以CONTROL + key 组合.
控制字符在脚本中不能正常使用.
Ctl-B 光标后退,这应该依赖于bash 输入的风格,默认是emacs 风格的.
Ctl-C Break,终止前台工作.
Ctl-D 从当前shell 登出(和exit 很像)
"EOF"(文件结束符).这也能从stdin 中终止输入.
在 console 或者在xterm window 中输入的时候,Ctl-D 将删除光标下字符.
当没有字符时,Ctrl-D 将退出当前会话.在xterm window 也有关闭窗口
的效果.
Ctl-G beep.在一些老的终端,将响铃.
Ctl-H backspace,删除光标前边的字符
Ctl-I 就是tab 键.
Ctl-J 新行.
Ctl-K 垂直tab.(垂直tab?新颖,没听过)
作用就是删除光标到行尾的字符.
Ctl-L clear,清屏.
Ctl-M 回车
Ctl-Q 继续(等价于XON 字符),这个继续的标准输入在一个终端里
Ctl-S 挂起(等价于XOFF 字符),这个被挂起的stdin 在一个终端里,用Ctl-Q 恢复
Ctl-U 删除光标到行首的所有字符,在某些设置下,删除全行.
Ctl-V 当输入字符时,Ctl-V 允许插入控制字符.比如,下边2 个例子是等价的
echo -e '\x0a'
echo <Ctl-V><Ctl-J>
Ctl-V在文本编辑器中十分有用,在vim 中一样.
Ctl-W 删除当前光标到前边的最近一个空格之间的字符.
在某些设置下,删除到第一个非字母或数字的字符.
Ctl-Z 终止前台工作.

空白部分
分割命令或者是变量.包括空格,tab,空行,或任何它们的组合.
在一些特殊情况下,空白是不允许的,如变量赋值时,会引起语法错误.
空白行在脚本中没有效果.
"$IFS",对于某些命令输入的特殊变量分割域,默认使用的是空白.
如果想保留空白,使用引用.

变量替换及赋值

$ 变量替换操作符
只有在变量被声明,赋值,unset 或exported 或者是在变量代表一个signal 的时候,
变量才会是以本来的面目出现在脚本里.变量在被赋值的时候,可能需要使用"=",
read状态或者是在循环的头部.
在""中还是会发生变量替换,这被叫做部分引用,或叫弱引用.而在''中就不会发生变
量替换,这叫做全引用,也叫强引用.具体见第5 章的讨论.
注意:$var 与${var}的区别,不加{},在某些上下文将引起错误,为了安全,使用2.

= 赋值操作符(前后都不能有空白)
不要与-eq 混淆,那个是test,并不是赋值.
注意,=也可被用来做 test 操作,这依赖于上下文.

**不像其他程序语言一样,Bash 并不对变量区分"类型".本质上,Bash 变量都是字符串.
但是依赖于上下文,Bash 也允许比较操作和算术操作.决定这些的关键因素就是,变量中的
值是否只有数字.**

特殊的变量类型
---
local variables
这种变量只有在代码块或者是函数中才可见
environmental variables
这种变量将改变用户接口和 shell 的行为.
在一般的上下文中,每个进程都有自己的环境,就是一组保持进程可能引用的信息的
变量.这种情况下,shell 于一个一般进程是相同的.
每次当 shell 启动时,它都将创建自己的环境变量.更新或者添加新的环境变量,将导
致 shell 更新它的环境,同时也会影响所有继承自这个环境的所有子进程(由这个命令
导致的).
positional parameters
就是从命令行中传进来的参数,$0, $1, $2, $3...
$0就是脚本文件的名字,$1 是第一个参数,$2 为第2 个...,参见[1](有$0 的说明),$9
以后就需要打括号了,如${10},${11},${12}...
两个值得注意的变量$*和$@，表示所有的位置参数.

转义

转义是一种引用单个字符的方法.一个具有特殊含义的字符前边放上一个转义符(\)就告诉shell这个字符失去了特殊的含义.
对于特定的转义符的特殊的含义
在 echo 和sed 中所使用的
\n 意味着新的一行
\r 回车
\t tab 键
\v vertical tab(垂直tab),查前边的Ctl-K
\b backspace,查前边的Ctl-H
\a "alert"(如beep 或flash)
\0xx 转换成8 进制ASCII 解码,等价于oxx

退出以及其状态

exit 命令被用来结束脚本,就像C 语言一样.他也会返回一个值来传给父进程,父进程会判断是否可用.每个命令都会返回一个 exit 状态(有时候也叫return 状态).成功返回0,如果返回一个非0 值,通常情况下都会被认为是一个错误码.一个编写良好的UNIX 命令,程序,和工具都会返回一个0 作为退出码来表示成功,虽然偶尔也会有例外.
同样的,脚本中的函数和脚本本身都会返回退出状态.在脚本或者是脚本函数中执行的最后的命令会决定退出状态.在脚本中,exit nnn 命令将会把nnn 退出码传递给shell(nnn 必须是10 进制数0-255).

$?读取最后执行命令的退出码.一般情况下,0 为成功,非0 失败。

文件测试操作

-e 文件存在
-a 文件存在这个选项的效果与-e 相同.但是它已经被弃用了,并且不鼓励使用
-f file 是一个regular 文件(不是目录或者设备文件)
-s 文件长度不为0
-d 文件是个目录
-b 文件是个块设备(软盘,cdrom 等等)
-c 文件是个字符设备(键盘,modem,声卡等等)
-p 文件是个管道
-h 文件是个符号链接
-L 文件是个符号链接
-S 文件是个socket
-t 关联到一个终端设备的文件描述符
这个选项一般都用来检测是否在一个给定脚本中的 stdin[-t0]或[-t1]是一个终端

-r 文件具有读权限(对于用户运行这个test)
-w 文件具有写权限(对于用户运行这个test)
-x 文件具有执行权限(对于用户运行这个test)
-g set-group-id(sgid)标志到文件或目录上
如果一个目录具有 sgid 标志,那么一个被创建在这个目录里的文件,这个目录属于创建这个目录的用户组,并不一定与创建这个文件的用户的组相同.对于workgroup 的目录共享来说,这非常有用.见<<UNIX 环境高级编程中文版>>第58 页.
-u set-user-id(suid)标志到文件上
如果运行一个具有 root 权限的文件,那么运行进程将取得root 权限,即使你是一个普通用户.[1]这对于需要存取系统硬件的执行操作(比如pppd 和cdrecord)非常有用.如果没有 suid 标志的话,那么普通用户(没有root 权限)将无法运行这种程序.
-k 设置粘贴位,
对于"sticky bit",save-text-mode 标志是一个文件权限的特殊类型.如果设置了这个标志,那么这个文件将被保存在交换区,为了达到快速存取的目的.如果设置在目录中,它将限制写权限.对于设置了sticky bit 位的文件或目录,权限标志中有"t".
-O 你是文件的所有者.
-G 文件的group-id 和你的相同.
-N 从文件最后被阅读到现在,是否被修改.
f1 -nt f2 文件 f1 比f2 新
f1 -ot f2 f1比f2 老
f1 -ef f2 f1和f2 都硬连接到同一个文件.
! 非--反转上边测试的结果(如果条件缺席,将返回true)

其他比较操作

二元比较操作符,比较变量或者比较数字.注意数字与字符串的区别.
整数比较
-eq 等于,如:if ["$a" -eq "$b"]
-ne 不等于,如:if ["$a" -ne "$b"]
-gt 大于,如:if ["$a" -gt "$b"]
-ge 大于等于,如:if ["$a" -ge "$b"]
-lt 小于,如:if ["$a" -lt "$b"]
-le 小于等于,如:if ["$a" -le "$b"]
< 小于(需要双括号),如:(("$a" < "$b"))
<= 小于等于(需要双括号),如:(("$a" <= "$b"))
> 大于(需要双括号),如:(("$a" > "$b"))
>= 大于等于(需要双括号),如:(("$a" >= "$b"))
字符串比较
= 等于,如:if ["$a" = "$b"]
== 等于,如:if ["$a" == "$b"],与=等价
注意:==的功能在[[]]和[]中的行为是不同的,如下:
1 [[$a == z*]] # 如果$a 以"z"开头(模式匹配)那么将为true
2 [[$a == "z*"]] # 如果$a 等于z*(字符匹配),那么结果为true
3

4 [$a == z*] # File globbing 和word splitting 将会发生
5 ["$a" == "z*"] # 如果$a 等于z*(字符匹配),那么结果为true
一点解释,关于File globbing 是一种关于文件的速记法,比如"*.c"就是,再如~也是.
但是 file globbing 并不是严格的正则表达式,虽然绝大多数情况下结构比较像.
!= 不等于,如:if ["$a" != "$b"]
这个操作符将在[[]]结构中使用模式匹配.
< 小于,在ASCII 字母顺序下.如:
if [["$a" < "$b"]]
if ["$a" \< "$b"]注意:在[]结构中"<"需要被转义.
> 大于,在ASCII 字母顺序下.如:
if [["$a" > "$b"]]
if ["$a" \> "$b"]
注意:在[]结构中">"需要被转义.
-z 字符串为"null".就是长度为0.
-n 字符串不为"null"
注意：使用-n 在[]结构中测试必须要用""把变量引起来.使用一个未被""的字符串来使用! -z 或者就是未用""引用的字符串本身,放到[]结构中

混合比较
-a 逻辑与
exp1 -a exp2 如果exp1 和exp2 都为true 的话,这个表达式将返回true
-o 逻辑或
exp1 -o exp2 如果exp1 和exp2 中有一个为true 的话,那么这个表达式就返回true
这与 Bash 的比较操作符&&和||很相像.在[[]]中使用它.
[[condition1 && condition2]]
-o 和-a 一般都是和test 命令或者是[]一起工作.
if ["$exp1" -a "$exp2"]

操作符

等号操作符
变量赋值
初始化或者修改变量的值
= 无论在算术运算还是字符串运算中,都是赋值语句.

算术操作符
+ 加法
- 减法
* 乘法
/ 除法
** 幂运算

% 取模
+= 加等于(通过常量增加变量)
let "var += 5" #var 将在本身值的基础上增加5
-= 减等于
*= 乘等于
let "var *= 4"
/= 除等于
%= 取模赋值,算术操作经常使用expr 或者let 表达式.

位操作符.

<< 左移1 位(每次左移都将乘2)
<<= 左移几位,=号后边将给出左移几位
let "var <<= 2"就是左移2 位(就是乘4)
>> 右移1 位(每次右移都将除2)
>>= 右移几位
& 按位与
&= 按位与赋值
| 按位或
|= 按位或赋值
~ 按位非
! 按位否
^ 按位异或XOR
^= 异或赋值

逻辑操作:

&& 逻辑与
|| 逻辑或

变量

内部变量

Builtin variable

这些内建的变量,将影响bash 脚本的行为.

$BASH

这个变量将指向 Bash 的二进制执行文件的位置.

$BASH_ENV

这个环境变量将指向一个 Bash 启动文件,这个启动文件将在调用一个脚本时被读取.

$BASH_SUBSHELL

这个变量将提醒 subshell 的层次,这是一个在version3 才被添加到Bash 中的新特性.

$BASH_VERSINFO[n]

记录 Bash 安装信息的一个6 元素的数组.与下边的$BASH_VERSION 很像,但这个更加详细.

$BASH_VERSION
安装在系统上的 Bash 的版本号.
$DIRSTACK
在目录栈中最上边的值(将受到pushd 和popd 的影响).
这个内建的变量与 dirs 命令是保持一致的,但是dirs 命令将显示目录栈的整个内容.
$EDITOR
脚本调用的默认编辑器,一般是vi 或者是emacs.
$EUID
"effective"用户ID 号.
当前用户被假定的任何 id 号.可能在su 命令中使用.
注意:$EUID 并不一定与$UID 相同.
$FUNCNAME
当前函数的名字.
$GLOBIGNORE
一个文件名的模式匹配列表,如果在file globbing 中匹配到的文件包含这个列表中的某个文件,那么这个文件将被从匹配到的文件中去掉.
$GROUPS
当前用户属于的组.
这是一个当前用户的组 id 列表(数组),就像在/etc/passwd 中记录的一样.
$HOME
用户的 home 目录,一般都是/home/username(见Example 9-14)
$HOSTNAME
hostname 命令将在一个init 脚本中,在启动的时候分配一个系统名字.
gethostname()函数将用来设置这个$HOSTNAME 内部变量.(见Example 9-14)
$HOSTTYPE
主机类型
就像$MACHTYPE,识别系统的硬件.
$IFS
内部域分隔符.
这个变量用来决定 Bash 在解释字符串时如何识别域,或者单词边界.
$IFS默认为空白(空格,tab,和新行),但可以修改,比如在分析逗号分隔的数据文件时.
注意:$*使用$IFS 中的第一个字符,
$LC_CTYPE
这个内部变量用来控制 globbing 和模式匹配的字符串解释.
$LINENO
这个变量记录它所在的 shell 脚本中它所在行的行号.这个变量一般用于调试目的.
$MACHTYPE
系统类型
提示系统硬件
$OLDPWD
老的工作目录("OLD-print-working-directory",你所在的之前的目录)
$OSTYPE
操作系统类型.
$PATH

指向 Bash 外部命令所在的位置,一般为/usr/bin,/usr/X11R6/bin,/usr/local/bin 等.
$PIPESTATUS
数组变量将保存最后一个运行的前台管道的退出码.有趣的是,这个退出码和最后一个命令
$PPID
一个进程的$PPID 就是它的父进程的进程id(pid).[1]
使用 pidof 命令对比一下.
$PROMPT_COMMAND
这个变量保存一个在主提示符($PS1)显示之前需要执行的命令.
$PS1
主提示符,具体见命令行上的显示.
$PS2
第 2 提示符,当你需要额外的输入的时候将会显示,默认为">".
$PS3
第 3 提示符,在一个select 循环中显示(见Example 10-29).
$PS4
第 4 提示符,当使用-x 选项调用脚本时,这个提示符将出现在每行的输出前边.
默认为"+".
$PWD
工作目录(你当前所在的目录).
与 pwd 内建命令作用相同.
$SHELLOPTS
这个变量里保存 shell 允许的选项,这个变量是只读的.
$SHLVL
Shell层次,就是shell 层叠的层次,如果是命令行那$SHLVL 就是1,如果命令行执行的脚本中,$SHLVL 就是2,以此类推.
$TMOUT
如果$TMOUT 环境变量被设置为一个非零的时间值,那么在过了这个指定的时间之后,shell提示符将会超时,这会引起一个logout.

位置参数
$0, $1, $2,等等...
位置参数,从命令行传递给脚本,或者是传递给函数.或者赋职给一个变量.
$#
命令行或者是位置参数的个数.(见Example 33-2)
$*
所有的位置参数,被作为一个单词.
注意:"$*"必须被""引用.
$@
与$*同义,但是每个参数都是一个独立的""引用字串,这就意味着参数被完整地传递,并没有被解释和扩展.这也意味着,每个参数列表中的每个参数都被当成一个独立的

其他的特殊参数

$-
传递给脚本的 falg(使用set 命令).
$!
在后台运行的最后的工作的 PID(进程ID).
$?
命令,函数或者脚本本身的退出状态(见Example 23-7)
$$
脚本自身的进程 ID.

字符串操作

字符串长度
${#string}
expr length $string
expr "$string" : '.*'

提取子串
${string:position}
在 string 中从位置$position 开始提取子串.
如果$string 为"*"或"@",那么将提取从位置$position 开始的位置参数,[1]
${string:position:length}
在 string 中从位置$position 开始提取$length 长度的子串.

从字符串开始的位置匹配子串的长度
expr match "$string" '$substring'
$substring 是一个正则表达式
expr "$string" : '$substring'
$substring 是一个正则表达式

索引
expr index $string $substring
匹配到子串的第一个字符的位置.

提取子串
${string:position}
在 string 中从位置$position 开始提取子串.
如果$string 为"*"或"@",那么将提取从位置$position 开始的位置参数,[1]
${string:position:length}
在 string 中从位置$position 开始提取$length 长度的子串.

子串削除
${string#substring}
从$string 的左边截掉第一个匹配的$substring

${string##substring}
从$string 的左边截掉最后一个个匹配的$substring

子串替换
${string/substring/replacement}
使用$replacement 来替换第一个匹配的$substring.
${string//substring/replacement}
使用$replacement 来替换所有匹配的$substring.

参数替换

操作和扩展变量
${parameter}
与$parameter 相同,就是parameter 的值.在特定的上下文中,只有少部分会产生
${parameter}的混淆.可以组合起来一起赋指给字符串变量.

变量扩展/子串替换
这些结构都是从 ksh 中吸收来的.
${var:pos}
变量 var 从位置pos 开始扩展.
${var:pos:len}
从位置 pos 开始,并扩展len 长度个字符.见Example A-14(这个例子里有这种操作的一个
创造性用法)
${var/Pattern/Replacement}
使用 Replacement 来替换var 中的第一个Pattern 的匹配.
${var//Pattern/Replacement}
全局替换.在var 中所有的匹配,都会用Replacement 来替换.
向上边所说,如果 Replacement 被忽略的话,那么所有匹配到的 Pattern 都会被删除.

指定类型的变量:declare 或者typeset
---
declare 或者typeset 内建命令(这两个命令是完全一样的)允许指定变量的具体类型.在某些
特
定的语言中,这是一种指定类型的很弱的形式.declare 命令是在Bash 版本2 或之后的版本
才被加入的.typeset 命令也可以工作在ksh 脚本中.
declare/typeset 选项
-r 只读
(declare -r var1与readonly var1 是完全一样的)
这和 C 语言中的const 关键字一样,都是强制指定只读.如果你尝试修改一个只读变量
的值,那么你将得到一个错误消息.
-i 整形
-a 数组
declae -a indices
变量 indices 将被视为数组.

-f 函数
declare -f
如果使用 declare -f 而不带参数的话,将会列出这个脚本中之前定义的所有函数.
declare -f function_name
如果使用 declare -f function_name 这种形式的话,将只会列出这个函数的名字.
-x export
declare -x var3
这种使用方式,将会把 var3 export 出来.

$RANDOM: 产生随机整数

$RANDOM 是Bash 的内部函数(并不是常量),这个函数将返回一个范围在0 - 32767 之间的一个伪随机整数.它不应该被用来产生密匙.

双圆括号结构

((...))与let 命令很像,允许算术扩展和赋值.举个简单的例子a=$((5 + 3)),将把a 设为"5+3"或者8.

循环和分支

for loops
for arg in [list]
这是一个基本的循环结构.它与C 的相似结构有很大不同.
```
for arg in [list]; do
    command(s)...
done
```

while
这种结构在循环的开头判断条件是否满足,如果条件一直满足,那就一直循环下去(0 为退出码).与for 循环的区别是,这种结构适合用在循环次数未知的情况下.
```
while [condition]
  do
    command...
  done
```
和 for 循环一样,如果想把 do 和条件放到同一行上还是需要一个";".

until
这个结构在循环的顶部判断条件,并且如果条件一直为false 那就一直循环下去.(与while相反)
```
until [condition-is-true]
  do
```

```
    command...
  done
```

循环控制

影响循环行为的命令

break,continue

break 和continue 这两个循环控制命令[1]与其它语言的类似命令的行为是相同的.break 命令将会跳出循环,continue 命令将会跳过本次循环下边的语句,直接进入下次循环.

测试与分支(case 和select 结构)

---

case 和select 结构在技术上说不是循环,因为它们并不对可执行的代码块进行迭代.但是和循环

相似的是,它们也依靠在代码块的顶部或底部的条件判断来决定程序的分支.

在代码块中控制程序分支

case (in) / esac

在 shell 中的case 同C/C++中的switch 结构是相同的.它允许通过判断来选择代码块中多条路径中的一条.

```
case "$variable" in
  "$condition1")
    command...
    ;;
  "$condition1")
    command...
    ;;
esac
```

注意: 对变量使用""并不是强制的,因为不会发生单词分离.

每句测试行,都以右小括号)结尾.

每个条件块都以两个分号结尾;;.

case 块的结束以 esac(case 的反向拼写)结尾.

select

select 结构是建立菜单的另一种工具,这种结构是从ksh 中引入的.

```
select variable [in list]
  do
    command...
    break
  done
```

提示用户选择的内容比如放在变量列表中.注意:select 命令使用PS3 提示符[默认为(#?)]但是可以修改 PS3.

内部命令与内建

内建命令指的就是包含在Bash 工具集中的命令.这内建命令将比外部命令的执行得更快,外部命令通常需要fork 出一个单独的进程来执行.另外一部分原因是特定的内建命令需要直接存取 shell 内核部分.
当一个命令或者是 shell 本身需要初始化(或者创建)一个新的子进程来执行一个任务的时候,这种行为被称为 forking.这个新产生的进程被叫做子进程,并且这个进程是从父进程中分离出来的.当子进程执行它的任务时,同时父进程也在运行.
注意:当父进程取得子进程的进程ID 的时候,父进程可以传递给子进程参数,而反过来则不行.这将产生不可思议的并且很难追踪的问题.

一般的,脚本中的内建命令在执行时将不会fork 出一个子进程.但是脚本中的外部或过滤命令
通常会 fork 一个子进程.

I/O 类
echo
打印(到stdout)一个表达式或变量

printf
printf 命令,格式化输出,是echo 命令的增强.它是C 语言printf()库函数的一个有限的变形,并且在语法上有些不同.

read
从 stdin 中读取一个变量的值,也就是与键盘交互取得变量的值.使用-a 参数可以取得数组变量

文件系统类
cd
cd,修改目录命令,在脚本中用得最多的时候就是,命令需要在指定目录下运行时,需要用cd 修改当前工作目录.

pwd
打印当前的工作目录.这将给用户(或脚本)当前的工作目录(见Example 11-9).使用这个命令的结果和从内键变量$PWD 中读取的值是相同的.
pushd, popd, dirs
这几个命令可以使得工作目录书签化,就是可以按顺序向前或向后移动工作目录.
压栈的动作可以保存工作目录列表.选项可以允许对目录栈作不同的操作.
pushd dir-name 把路径dir-name 压入目录栈,同时修改当前目录到dir-name.
popd 将目录栈中最上边的目录弹出,同时修改当前目录到弹出来的那个目录.
dirs 列出所有目录栈的内容(与$DIRSTACK 便两相比较).一个成功的pushd 或者popd 将会自动的调用 dirs 命令.
对于那些并没有对当前工作目录做硬编码,并且需要对当前工作目录做灵活修改的脚本来说

,使用这些命令是再好不过的了.注意内建$DIRSTACK 数组变量,这个变量可以在脚本内存取,并且它们保存了目录栈的内容.

变量类

let

let 命令将执行变量的算术操作.在许多情况下,它被看作是复杂的expr 版本的一个简化版.

eval

eval arg1 [arg2] ... [argN]

将表达式中的参数,或者表达式列表,组合起来,并且评估它们.包含在表达式中的任何变量都将被扩展.结果将会被转化到命令中.这对于从命令行或者脚本中产生代码是很有用的.

set

set 命令用来修改内部脚本变量的值.一个作用就是触发选项标志位来帮助决定脚本的行为.另一个应用就是以一个命令的结果(set `command`)来重新设置脚本的位置参数.脚本将会从命令的输出中重新分析出位置参数.

unset

unset 命令用来删除一个shell 变量,效果就是把这个变量设为null.注意:这个命令对位置参数无效.

export

export 命令将会使得被export 的变量在运行的脚本(或shell)的所有的子进程中都可用.不幸的是,没有办法将变量export 到父进程(就是调用这个脚本或shell 的进程)中.

关于 export 命令的一个重要的使用就是用在启动文件中,启动文件是用来初始化并且设置环境变量,让用户进程可以存取环境变量.

declare, typeset

declare 和typeset 命令被用来指定或限制变量的属性.

readonly

与 declare -r 作用相同,设置变量的只读属性,也可以认为是设置常量.设置了这种属性之后如果你还要修改它,那么你将得到一个错误消息.这种情况与C 语言中的const 常量类型的情况是相同的.

getopts

可以说这是分析传递到脚本的命令行参数的最强力工具.这个命令与getopt 外部命令,和C语言中的库函数getopt 的作用是相同的.它允许传递和连接多个选项[2]到脚本中,并能分配多个参数到脚本中.

getopts 结构使用两个隐含变量.$OPTIND 是参数指针(选项索引),和$OPTARG(选项参数)(可选的)可以在选项后边附加一个参数.在声明标签中,选项名后边的冒号用来提示这个选项名已经分配了一个参数.

getopts 结构通常都组成一组放在一个while 循环中,循环过程中每次处理一个选项和参数,然后增加隐含变量$OPTIND 的值,再进行下一次的处理.

注意:

1.通过命令行传递到脚本中的参数前边必须加上一个减号(-).这是一个前缀,这样

getopts 命令将会认为这个参数是一个选项.事实上,getopts 不会处理不带"-"前缀的参数,如果第一个参数就没有"-",那么将结束选项的处理.
2.使用getopts 的while 循环模版还是与标准的while 循环模版有些不同.没有标准while循环中的[]判断条件.
3.getopts结构将会取代getopt 外部命令.

脚本行为

source, . (点命令)
这个命令在命令行上执行的时候,将会执行一个脚本.在一个文件内一个source file-name 将会加载 file-name 文件.source 一个文件(或点命令)将会在脚本中引入代码,并附加到脚本中(与C 语言中的#include 指令的效果相同).最终的结果就像是在使用"sourced"行上插入了相应文件的内容.这在多个脚本需要引用相同的数据,或函数库时非常有用.

exit
绝对的停止一个脚本的运行.exit 命令有可以随便找一个整数变量作为退出脚本返回shell 时的退出码.使用exit 0 对于退出一个简单脚本来说是种好习惯,表明成功运行.
注意: 如果不带参数的使用exit 来退出,那么退出码将是脚本中最后一个命令的退出码.等价于 exit $?.
exec
这个 shell 内建命令将使用一个特定的命令来取代当前进程.一般的当shell 遇到一个命令,它会 fork off 一个子进程来真正的运行命令.使用exec 内建命令,shell 就不会fork 了,并且命令的执行将会替换掉当前 shell.因此,当我们在脚本中使用它时,当命令实行完毕,它就会强制退出脚本.

shopt
这个命令允许 shell 在空闲时修改shell 选项(见Example 24-1 和Example 24-2).它经常出现在启动脚本中,但是在一般脚本中也可用.需要Bash 2.0 版本以上.

caller
将 caller 命令放到函数中,将会在stdout 上打印出函数调用者的信息

命令类

ture
一个返回成功(就是返回0)退出码的命令,但是除此之外什么事也不做.

flase
一个返回失败(非0)退出码的命令,但是除此之外什么事也不做.

type[cmd]
与 which 扩展命令很相像,type cmd 将给出"cmd"的完整路径.与which 命令不同的是,type 命令是 Bash 内建命令.一个很有用的选项是-a 选项,使用这个选项可以鉴别所识别的参数是关键字还是内建命令,也可以定位同名的系统命令.
hash[cmds]
在 shell 的hash 表中[4],记录指定命令的路径名,所以在shell 或脚本中在调用这个命令的

话,shell 或脚本将不需要再在$PATH 中重新搜索这个命令了.如果不带参数的调用hash 命令,它将列出所有已经被hash 的命令.-r 选项会重新设置hash 表.
bind
bind 内建命令用来显示或修改readline[5]的键绑定.
help
获得 shell 内建命令的一个小的使用总结.这与whatis 命令比较象,但是help 是内建命令.

作业控制命令

jobs
在后台列出所有正在运行的作业,给出作业号.

disown
从 shell 的当前作业表中,删除作业.
fg,bg
fg 命令可以把一个在后台运行的作业放到前台来运行.而bg 命令将会重新启动一个挂起的作业,并且在后台运行它.如果使用fg 或者bg 命令的时候没指定作业号,那么默认将对当前正在运行的作业做操作.
wait
停止脚本的运行,直到后台运行的所有作业都结束为止,或者直到指定作业号或进程号为选项的作业结束为止.
你可以使用 wait 命令来防止在后台作业没完成(这会产生一个孤儿进程)之前退出脚本.

suspend
这个命令的效果与 Control-Z 很相像,但是它挂起的是这个shell(这个shell 的父进程应该在合适的时候重新恢复它).
logout
退出一个登陆的 shell,也可以指定一个退出码.
times
给出执行命令所占的时间,使用如下形式输出:
0m0.020s 0m0.020s
这是一种很有限的能力,因为这不常出现于shell 脚本中.
kill
通过发送一个适当的结束信号,来强制结束一个进程

command
command 命令会禁用别名和函数的查找.它只查找内部命令以及搜索路径中找到的脚本或可执行程序
注意: 当象运行的命令或函数与内建命令同名时,由于内建命令比外部命令的优先级高,而函数比内建命令优先级高,所以bash 将总会执行优先级比较高的命令.这样你就没有选择的余地了.所以Bash 提供了3 个命令来让你有选择的机会.command 命令就是这3 个命

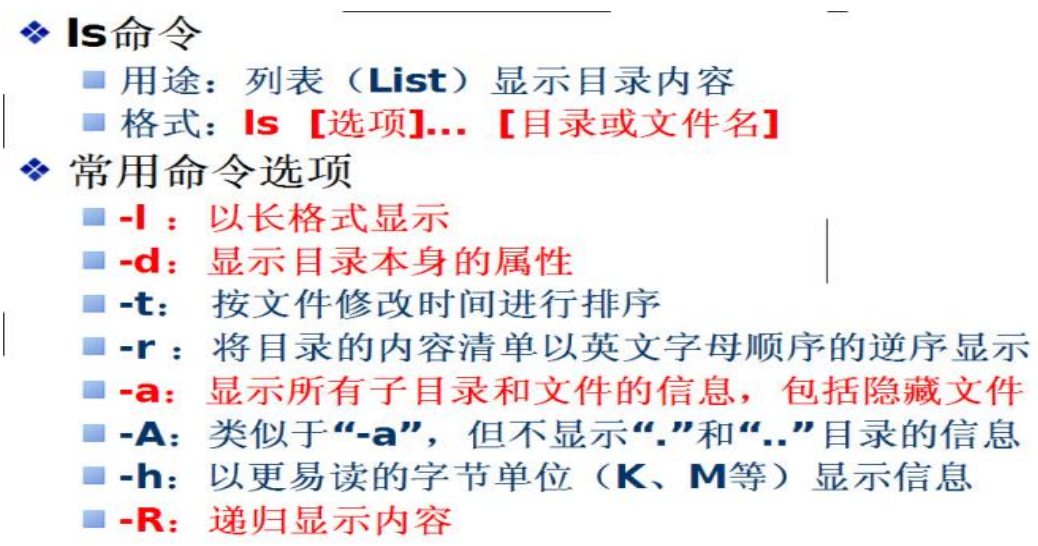
令之一.另外两个是 builtin 和enable.
builtin
在"builtin"后边的命令将只调用内建命令.暂时的禁用同名的函数或者是同名的扩展命令.
enable
这个命令或者禁用内建命令或者恢复内建命令.如: enable -n kill 将禁用kill 内建命令,所以当我们调用 kill 时,使用的将是/bin/kill 外部命令.

-a 选项将会恢复相应的内建命令,如果不带参数的话,将会恢复所有的内建命令.
选项-f filename 将会从适当的编译过的目标文件[6]中以共享库(DLL)的形式来加载一个内建命令.
autoload
这是从 ksh 的autoloader 命令移植过来的.一个带有"autoload"声明的函数,在它第一次被调用的时候才会被加载.[7] 这样做会节省系统资源.
注意: autoload 命令并不是Bash 安装时候的核心命令的一部分.这个命令需要使用命令enable -f(见上边 enable 命令)来加载.

作业标识符

记法	含义
%N	作业号[N]
%S	以字符串S 开头的被(命令行)调用的作业
%?S	包含字符串S 的被(命令行)调用的作业
%%	当前作业(前台最后结束的作业,或后台最后启动的作业)
%+	当前作业(前台最后结束的作业,或后台最后启动的作业)
%-	最后的作业
$!	最后的后台进程

基本命令



•ls命令后会产生多种颜色的对象

蓝色表示目录
红色表示压缩文件
绿色便是可执行文件
紫色便是图片文件
浅蓝色表示链接文件（类似于windows下的快捷方式）
黑色表示普通文件

- **ls命令**
  - 用途：列表（List）显示目录内容
  - 格式：ls [选项]... [目录或文件名]
- **常用命令选项**

文件类型	缩写	应用
常规文件	-	保存数据
目录	d	存放文件
符号链接	l	指向其它文件
字符设备节点	c	访问设备
块设备节点	b	访问设备

详细信息的第一个字母

r read
w write
x 执行

```
[root@teacher ~]# ls -dl /boot
dr-xr-xr-x. 6 root root 3072 Jul 19 11:32 /boot
结果解析：
dr-xr-xr-x. ：  共10个字符，最左侧表示对象类型，后面九个字符表示权限码
6：表示该目录下有几个子目录或者表示该文件有几个名字
第一个root:表示属主(主人)
第二个root:表示属组（组别）
3072:该对象的大小，单位字节bytes
Jul 19 11:32:创建时间
```

三位一看
所有者的权限
所有者所在组的权限
其他用户的权限
上图 6 对于文件而言是指有几个文件名，对于目录而言是指有几个子目录
• su - zhang
- 会进入到所切换用户的家目录下
不加 - 会在root下

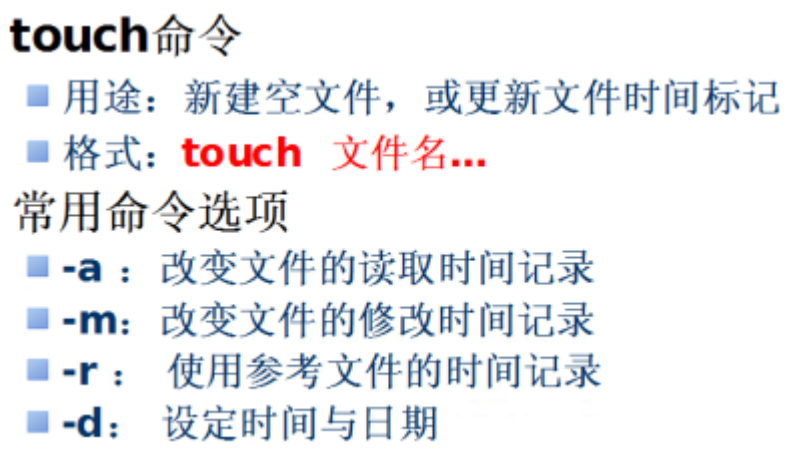
文件名前带. 是隐藏文件

递归文件
显示目录的同时显示子目录的内容
• mkdir

mkdir -p

du -sh 文件夹或文件 显示文件夹总和的大小
ls -ldh 显示文件夹的大小（不包含里面的文件
man中查找选项 /加关键字符
eg： /-s

•touch




```
[root@localhost ~]# touch file1.txt file2.doc
[root@localhost ~]# touch -ad 10:35 file1.txt
[root@localhost ~]# touch -md 11:25 file2.doc
[root@localhost ~]# touch -r file2.doc file1.txt
```

• cp

cp命令

- 用途：复制（Copy）文件或目录
- 格式：cp [选项]... 源文件或目录... 目标文件或目录
- -r：递归复制整个目录树
- -a：复制时保留链接、文件属性，并递归地复制目录

mv命令

- 用途：移动（Move）文件或目录
  - —— 若如果目标位置与源位置相同，则相当于改名
- 格式：mv [选项]... 源文件或目录... 目标文件或目录

自动继承目的地的主人权限

-ar

•file

•rmdir 删除空目录

rm 删除非空目录及空目录

rm -rf 【】

查看文件类型

stat命令：

```
[root@server1 ~]# stat install.log
  File: `install.log'
  Size: 37086           Blocks: 88         IO Block: 4096   regular file
)evice: 803h/2051d      Inode: 131075      Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2015-06-11 00:08:41.281997581 +0800
Modify: 2014-03-29 18:01:55.145999702 +0800
Change: 2014-03-29 18:02:03.993999698 +0800
```

stat命令很多结果与 ls -l 类似

```
[root@server1 ~]# #stat命令有atime,mtime,ctime分别表示最近的访问时间，最近修改内容的时间，最近修改属性的时间
```

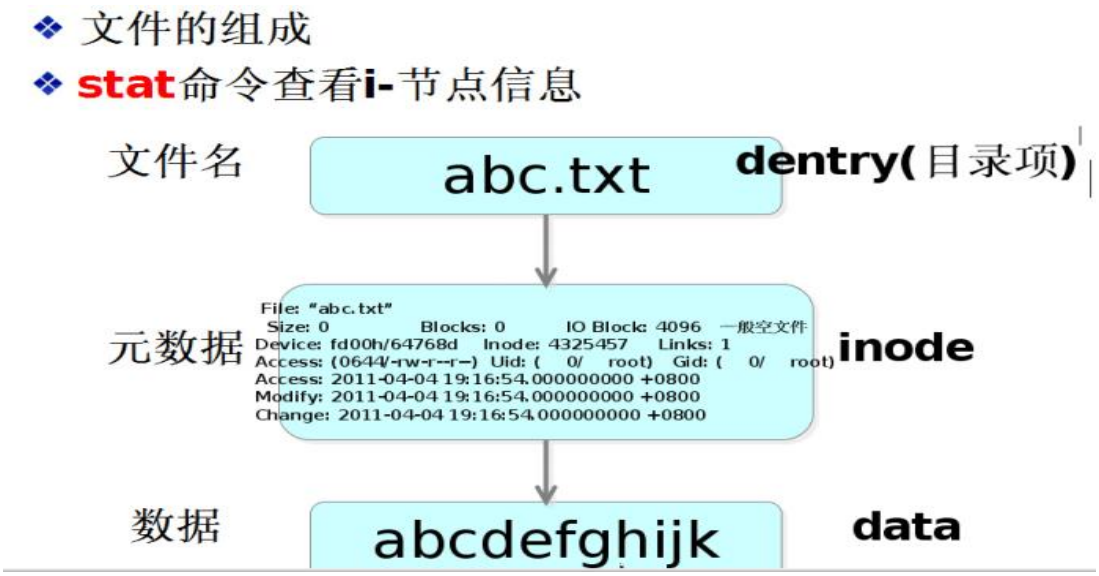
Inode 是指磁盘的编号，如果磁盘的编号用完，就算有空间也不能分配。

IO block 是指每个数据块的大小

ACCESS 是指最近的访问时间

Modify是指最近的修改文件内容的时间

Change是指最近的修改文件权限的时间



系统根据文件名找到Inode号

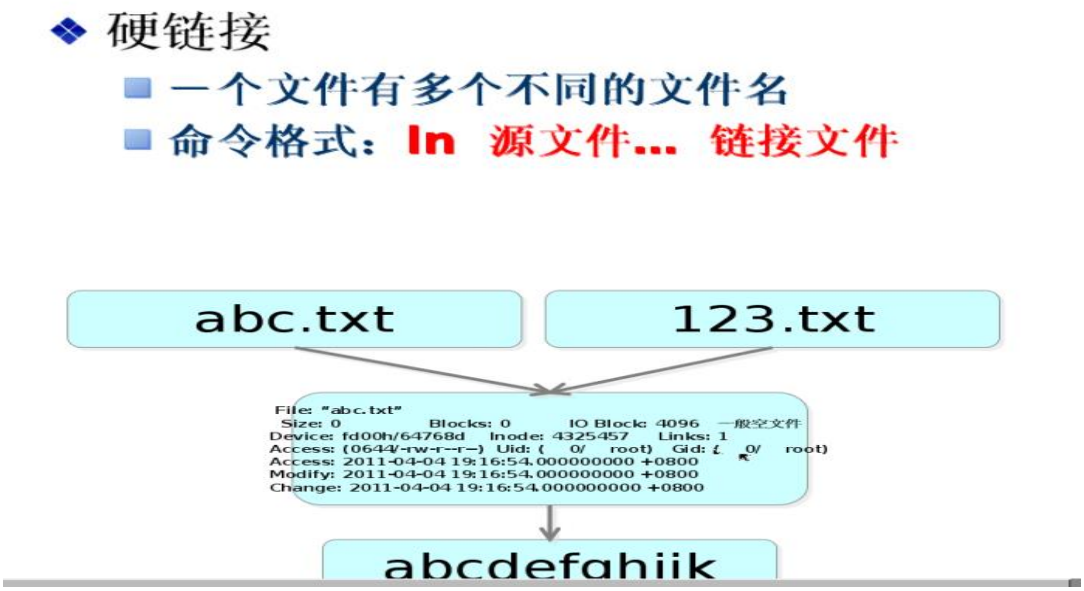
解释一个分区内怎么通过文件名读取具体数据：

在一个分区内有三张表，通过文件名在目录项表中找到对应的Inode号

根据Inode号在Inode table 找到对应的块号

通过块号在data表内找到存储的数据。

·硬链接，多个文件对同一个数据，同一个Inode号



执行ln install.log xj命令后：两个文件的对应的Inode号相同

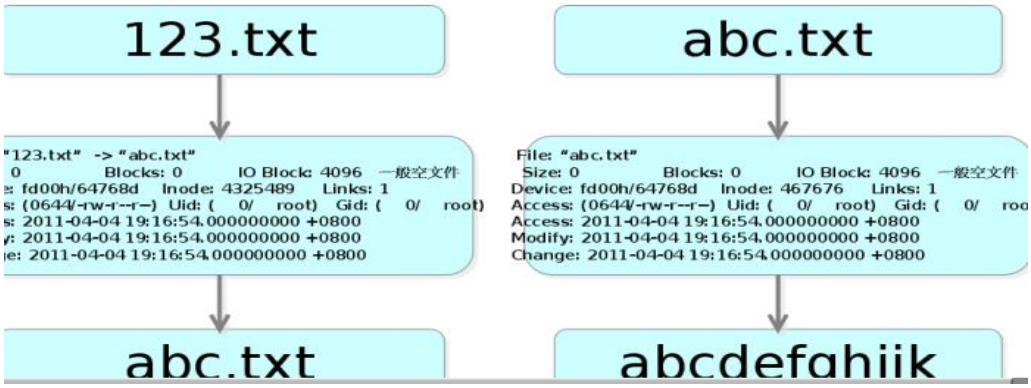
```
[root@localhost ~]# ls -i install.log
130563 install.log
[root@localhost ~]# ls -i xj
130563 xj
```

·软链接

- 软链接
  - 符号链接，表面上和硬连接相似
  - 文件类型和权限肯定是lrwxrwxrwx
  - 命令格式：ln -s 源文件... 链接文件

示例中123.txt和abc.txt位于同分区。软链接中的两个文件可以在不同分区

- 创建软链接
  - ln -s abc.txt 123.txt



- 硬链接和软链接比较
  - 软链接：指向原始文件所在的路径，又称为软链接
  - 硬链接：指向原始文件对应的数据存储位置
  - 不能为目录建立硬链接文件
  - 硬链接与原始文件必须位于同一分区（文件系统）中

- cat命令
  - 用途：显示出文件的全部内容
  - 格式：cat -n 文件名
- tac命令
  - 用途：从最后一行倒着显示出文件的全部内容

```
[root@localhost ~]# cat /etc/aaa
11111111111111111111
22222222222222222
[root@localhost ~]# tac /etc/aaa
22222222222222222
11111111111111111111
```

- **more命令**
  - 用途：全屏方式分页显示文件内容
  - 交互操作方法：
    - 按Enter键向下逐行滚动
    - 按空格键向下翻一屏、按b键向上翻一屏
    - 按q键退出
- **less命令**
  - 用途：与more命令相同，但扩展功能更多
  - 交互操作方法：
    - 与more命令基本类似，但个别操作会有些出入
    - 【page down】【page up】上翻下翻页

- **head命令**
  - 用途：查看文件开头的一部分内容（默认为10行）
  - 格式：head -n 文件名
- **tail命令**
  - 用途：查看文件结尾的少部分内容（默认为10行）
  - 格式：tail -n 文件名
    tail -f 文件名

```
[root@localhost ~]# tail -2 /var/log/messages
Sep  8 15:49:29 localhost scim-bridge: Cleanup, done.
    Exitting...
Sep  8 15:49:29 localhost Cleanup, done. Exitting...
```

tail -n +3 passwd
该命令的意思是从第三行开始显示
[root@localhost ~]# ps aux|tail -n +2|sort -k2 -nr
文件的第二行 开始显示，按第二列降序排序。|是指将左边的结果作为参数传递给右边

- **tail命令高级用法**
  - 格式：tail -n 数字 文件名
  - 数字：数字前有 +（加号），从文件开头指定的单元数开始输出；数字前有 -（减号），从文件末尾指定的单元 数开始输出；没有 +或 -，从文件末尾指定的单元 数开始输出。
  - 例如：
    - tail -n +3 /etc/passwd 从第三行开始显示
    - tail -n -3 /etc/passwd 显示最后三行
    - head -n -3 /etc/passwd 不显示最后三行
    - head -n +3 /etc/passwd 显示前三行

rev
把每一行中的内容反转，并且输出到 stdout 上. 这个命令与 tac 命令的效果是不同的，因为它并不反转行序，而是把每行的内容反转.

env 是查看当前环境变量的命令

```
[root@localhost ~]# env|grep PATH
PATH=/usr/lib64/qt-3.3/bin:/usr/local/sbin:/usr/sbin:/sbin:/usr/loc
in:/bin:/usr/local/mysql/bin:/root/bin
WINDOWPATH=1
[root@localhost ~]#
```

*过滤出带有 PATH 变量的行*也可以用 echo $PATH

```
[root@teacher ~]# //过滤出带有"PATH"字符串的行
bash: //过滤出带有PATH字符串的行: No such file or directory
[root@teacher ~]# #PATH环境变量是当前用户的命令搜索路径
```

所以过滤出 PATH 变量可以用 echo|grep PATH 以及 echo $PATH

```
[root@localhost ~]# a=4
[root@localhost ~]# echo $a
4
[root@localhost ~]# b=a
[root@localhost ~]# echo $b
a
[root@localhost ~]# echo ${!b}
4
```

- **which命令**
  - 用途：查找可执行文件并显示所在的位置
    —— 搜索范围由 **PATH** 环境变量指定
  - 格式：**which** 命令或程序名

```
[root@localhost ~]# which mkdir
/bin/mkdir
[root@localhost ~]# echo $PATH
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bi
n:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
[root@localhost ~]# which cd
/usr/bin/which: no cd in
(/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/loc
al/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin)
```

```
[root@localhost ~]# for ((i=1;i<=10;i++)); do  echo $i; done
1
2
3
4
5
6
7
8
9
10
[root@localhost ~]# seq 10
1
2
3
4
5
6
7
8
9
10
[root@localhost ~]#
```

•Seq 纵向排列 xargs 横向排列 tr " " + 是把字符串的空格变成加号

```
[root@localhost ~]# seq 100|xargs
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 3
1 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58
 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
[root@localhost ~]# seq 100|xargs|tr " " +
1+2+3+4+5+6+7+8+9+10+11+12+13+14+15+16+17+18+19+20+21+22+23+24+25+26+27+28+29+30+3
1+32+33+34+35+36+37+38+39+40+41+42+43+44+45+46+47+48+49+50+51+52+53+54+55+56+57+58
+59+60+61+62+63+64+65+66+67+68+69+70+71+72+73+74+75+76+77+78+79+80+81+82+83+84+85+
86+87+88+89+90+91+92+93+94+95+96+97+98+99+100
[root@localhost ~]# seq 100|xargs|tr " " +|bc
5050
[root@localhost ~]#
```

- **whereis命令**
  - 用途：查找文件的路径、该文件的帮助文件路径，原理和**which**类似
  - 格式：**whereis** 命令或程序名

```
[root@localhost ~]# whereis which
which: /usr/bin/which /usr/share/man/man1/which.1.gz
```

which 只搜素 PATH 环境变量下的可执行文件，并显示其绝对路径
whereis 也是搜索 PATH 环境变量下的路径，但不要求一定是执行文件，其他类型文件也可。当搜索到对象是一个命令时，还会显示该命令的帮助文档路径

- **locate命令**
  - 格式：**locate** 文件名
  - 根据每天更新的数据库**(/var/lib/mlocate)**查找，速度块

手动更新数据库命令：updatedb

```
[root@localhost ~]# touch wxj
[root@localhost ~]# locate wxj
[root@localhost ~]# updatedb
[root@localhost ~]# locate wxj
/root/wxj
```

刚开始没有更新，查找不到 wxj 文件的位置，更新数据库之后可以查找到位置

- **find命令**
  - 用途：用于查找文件或目录
  - 格式：find [查找范围] [查找条件][动作]
- 常用查找条件
  - **-name**：按文件名称查找
  - **-size**：按文件大小查找
  - **-user**：按文件属主查找
  - **-type**：按文件类型查找
  - **-perm**：按文件权限查找
  - **-mtime**：按文件更改时间查找
  - **-newer**：按比某个文件更新的查找

练习:查找/usr目录下设置了suid或者sgid权限的普通文件。
-perm +|- 权限数字描述
-perm +6000

-perm +|- 权限数字描述
-perm +6000

```
[root@bogon ~]# find /usr -type f -perm +6000
```

+表示后面有 1 的位有一个匹配就匹配
- 表示后面有 1 的位全部为 1 才匹配
没有+和- 表示和后面的权限码完全匹配
+6000
suid sgid sticky
1 1 0

·-maxdepth 查找最大深度，是否找所要找的对象的子目录

- 特殊查找条件
  - **-o**：逻辑或，只要所给的条件中有一个满足，寻找条件就算满足
  - **-not**:逻辑非，在命令中可用“!”表示。该运算符表示查找不满足所给条件的文件
  - **-a**：逻辑与，系统默认是与，可不加，表示只有当所给的条件都满足时，寻找条件才算满足。

find 命令 * 做通配符时表示任意多个任意字符
? 通配符代表一个字符

```
[root@localhost ~]# find /boot -name "vmlinuz*" -a -size +1M -exec cp  {} /root \;

[root@localhost ~]# ls
anaconda-ks.cfg     passwd                      公共的  图片  音乐
install.log         vmlinuz-2.6.32-358.el6.x86_64  模板    文档  桌面
install.log.syslog  wxj                         视频    下载
```

上图为找到/boot 目录下名字为 vmlinuz 开头的且文件大小大于 1M。并将文件复制到家目录下。

-exec 和 -ok 命令差不多 只不过 -ok 会让你进行确认。

```
[root@localhost ~]# find /home -nouser
/home/wbq
```

寻找无组对象

用户被删了，若其中的数据没有被删，则变成了无组对象。

- -name "文件名"
- -size 1M/+1M/-1M
- -user/-nouser

```
-type  类型
    f   (file)  普通文件
    d    (directory)  目录 (文件夹)
    c 字符设备 (character)
    b  块设备  (block)
    l  (link)  链接文件
    p  管道
```

-mtime +5 五天之前的文件

-mtime -5 五天之内的文件

- 删除/boot 下七天之前的文件 {}表示为查找到的文件

```
find /boot -mtime +7 -exec rm -rf {} \;
```

```
[root@localhost ~]# find /root -newer haha -type f
/root/xixi
```

寻找/boot 下比 haha 新的文件

- mount/umount 命令

```
[root@localhost ~]# #mount 挂载源 挂载点
[root@localhost ~]# mount /dev/sr0 /mnt
mount: block device /dev/sr0 is write-protected, mounting read-only
```

上述命令将光驱挂在在/mnt 上，因为/dev/sr0 不能直接访问

```
[root@server1 ~]# #umount /dev/sr0 或者 umount /mnt  卸载光驱
[root@teacher iso]# pwd
/var/ftp/iso
[root@teacher iso]# mount -o loop rhel6.4-x86_64.iso /mnt
```

★alias 别名功能

输入命令 vi .bashrc（记得有空格）按 i 进行输入， 输入完成后按【ESC】然后按【shift】和冒号 按 wq 保 存退出
完成之后

```
# .bashrc

# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
alias myvnc='vncviewer 10.0.2.253:6 -shared -viewonly &'
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
~
~
~
~
~
~
~
~
~
~
~
".bashrc" 12L, 232C
```

此时在 vi 编辑器内加入了
alias myvnc=’vncviewer 10.0.2.253:6 -shared -viewonly &’
把远程连接命令起别名 myvnc 下次再远程连接此电脑直接敲 myvnc 命令

```
alias myvnc='echo aixocm|vncviewer 10.0.2.253:6 -shared -viewonly -passwdinput'
# Source global definitions
```

- **zip命令**
  - 用途：制作压缩文件
  - 格式：**zip** 压缩文件名 文件1 文件2
- **unzip命令**
  - 用途：解压缩文件
  - 格式：**unzip** **[-l]** 压缩文件名

```
[root@localhost test]# touch aa bb cc
[root@localhost test]# ls
aa  bb  cc
[root@localhost test]# zip ff.zip ??
  adding: aa (stored 0%)
  adding: bb (stored 0%)
  adding: cc (stored 0%)
[root@localhost test]# ls
aa  bb  cc  ff.zip
```

两个？表示以两个字符命名的文件

```
[root@localhost test]# unzip ff.zip
Archive:  ff.zip
 extracting: aa
 extracting: bb
 extracting: cc
```

•dd 命令

```
[root@teacher lianxi]# #dd if=?    of=?   bs=?  count=?
[root@teacher lianxi]# dd if=/dev/zero of=saijie bs=1M count=100
100+0 records in
100+0 records out
104857600 bytes (105 MB) copied, 0.0765903 s, 1.4 GB/s
[root@teacher lianxi]# ls
auto.sys  saijie  vmlinuz-2.6.32-358.el6.x86_64
[@teacher lianxi]# du -sh saijie
    saijie
[@teacher lianxi]# #请用dd命令创建一个2048个字节的文件haha
[@teacher lianxi]# #dd if=/dev/zero of=saijie bs=1 count=2048
[@teacher lianxi]# #dd if=/dev/zero of=saijie bs=2048 count=1
[@teacher lianxi]# #dd if=/dev/zero of=saijie bs=1024 count=2
[@teacher lianxi]#
```

- **bzip2(gzip)命令**
  - 用途：制作压缩文件、解开压缩文件
  - 格式：**bzip2(gzip) [-9]** 文件名...
    bzip2(gzip) -d .bz2格式的压缩文件
- 常用命令选项
  - **-9**：表示高压缩比，取值**1-9**，默认为**6**
  - **-d**：用于解压缩文件，同**bunzip2(gunzip)**命令
  - **-c**：将输出重定向到标准输出
- **bzcat命令**
  - 用途：查看压缩文件内容
  - 格式：**bzcat** 压缩文件名

·运用 gzip 命令压缩的同时保留源文件命令：gzip< 文件名>文件名.gz

```
[root@bogon ~]# gzip<vmlinuz-2.6.32-358.el6.x86_64>vmlinuz-2.6.32-358.el6.x86_64
.gz
```

```
vmlinuz-2.6.32-358.el6.x86_64
vmlinuz-2.6.32-358.el6.x86_64.gz
```

- **tar命令**
  - 用途：制作归档文件、释放归档文件
  - 格式：**tar** [选项]... 归档文件名 源文件或目录
    tar [选项]... 归档文件名 [-C 目标目录]
- 常用命令选项
  - **-c**：创建 **.tar** 格式的包文件
  - **-x**：解开**.tar**格式的包文件
  - **-v**：输出详细信息
  - **-f**：表示使用归档文件
  - **-t**：列表查看包内的文件
  - **-p**：保持原文件的原来属性
  - **-P**：保持原文件的绝对路径

-r ：像原有的 tar 包里追加文件
对文件的备份文件归档 解包时对包内的备份文件解包。

- 常用命令选项
  - **-C**：建包或解包时进入指定的目标文件夹
  - **-z**：调用**gzip**程序进行压缩或解压
  - **-j**：调用**bzip2**程序进行压缩或解压

制作压缩包文件

```
[root@localhost ~]# tar jcf test.tar.bz2 /etc/httpd/
tar: 从成员名中删除开头的"/"
[root@localhost ~]# ls -lh test.tar.bz2
-rw-r--r-- 1 root root 21K 09-09 01:19 test.tar.bz2
[root@localhost ~]# tar xf test.tar.bz2 -C /tmp
[root@localhost ~]# ls -ld /tmp/etc/httpd/
drwxr-xr-x 4 root root 4096 09-08 16:37 /tmp/etc/httpd/
[root@localhost ~]# rm -rf /tmp/etc/
```

释放压缩包文件

解压缩：

```
[root@tom linux-4.8.12]# unxz patch-4.8.12.xz
```

在 tar 包里追加文件：

```
[root@bogon lianxi]# tar rf /tmp/sjjy.tar ~/.bashrc
```

释放到指定目录要先写-C 【指定目录】再写所要释放的文件

```
[root@bogon lianxi]# tar xf /tmp/sjjy.tar -C /home etc/group
```

```
[root@bogon lianxi]# tar cjf /tmp/home.tar.bz2 /home/*
tar: Removing leading `/' from member names
[root@bogon lianxi]# tar tvf /tmp/home.tar.bz2
drwxr-xr-x root/root         0 2016-10-15 21:04 home/etc/
-rw-r--r-- root/root       731 2016-09-25 13:59 home/etc/group
drwx------ tom/tom           0 2016-09-25 13:59 home/tom/
-rw-r--r-- tom/tom         176 2012-08-29 19:19 home/tom/.bash_profile
-rw-r--r-- tom/tom         124 2012-08-29 19:19 home/tom/.bashrc
drwxr-xr-x tom/tom           0 2016-09-25 13:47 home/tom/.mozilla/
drwxr-xr-x tom/tom           0 2009-12-03 10:21 home/tom/.mozilla/plugins/
drwxr-xr-x tom/tom           0 2009-12-03 10:21 home/tom/.mozilla/extensions/
-rw-r--r-- tom/tom          18 2012-08-29 19:19 home/tom/.bash_logout
drwxr-xr-x tom/tom           0 2010-07-14 23:55 home/tom/.gnome2/
```

```
 [root@bogon lianxi]# tar cvfz useradmin.tar.gz passwd shadow
 passwd
 shadow
 [root@bogon lianxi]# tar cvfj useradmin.tar.bz2 passwd shadow
 passwd
.shadow
```

```
[root@bogon lianxi]# tar xvf useradmin.tar.bz2 -C ./
passwd
shadow
[root@bogon lianxi]# tar xvf useradmin.tar.gz -C /tmp
passwd
shadow
```

以 ftp 的方式登入老师电脑的命令：lftp 10.0.2.253

然后下载 zabbix 软件

```
[root@bogon lianxi]# lftp 10.0.2.253
lftp 10.0.2.253:~> cd software
cd ok, cwd=/software
lftp 10.0.2.253:/software> get zabbix-2.0.6.tar.gz
```

mget 批量下载 get 是单个下载

```
[root@teacher lianxi]# *.rpm(二进制包)      *.tar.gz|*.tar.bz2(源码包)
```

date 命令

```
[root@bogon lianxi]# date
Sat Oct 15 21:09:46 CST 2016
[root@bogon lianxi]# date +%F
2016-10-15
[root@bogon lianxi]# date +%T
21:10:20
[root@bogon lianxi]# date "+%F %T"
2016-10-15 21:10:59
```

一对反引号的作用是将反引号里的值引出来

```
[root@teacher lianxi]# mkdir haha.`date +%F`
[root@teacher lianxi]# ls
auto.sys  etc  haha.2016-10-14  vmlinuz-2.6.32-358.el6.x86_64
```

•查看网卡 1 的 ip 地址

```
[root@bogon ~]# ifconfig eth0
```

• 杀死进程号为 5787 的进程

```
[root@bogon ~]# kill 5787
```

• 临时向 DHCP 服务器申请 IP

```
[root@bogon ~]# dhclient
```

• 为主机临时设置 ip

```
[root@bogon ~]# #ifconfig etho 10.0.2.181 netmask 255.255.0.0
```

•联通性测试用 ping 命令

- 访问权限
  - 可读(read)：允许查看文件内容、显示目录列表
  - 可写(write)：允许修改文件内容，允许在目录中新建、移动、删除文件或子目录
  - 可执行(execute)：允许运行程序、切换目录
- 归属（所有权）
  - 文件拥有者(owner)：拥有该文件或目录的用户帐号
  - 属组(group)：拥有该文件或目录的组帐号
  - 其它人(others)：除了属主和属组的其他人

```
[root@localhost ~]# ls -l install.log
-rw-r--r--  1  root  root  34298  04-02  00:23  install.log
```

文件类型　访问权限　所有者　属组

权限项	读	写	执行	读	写	执行	读	写	执行
字符表示	r	w	x	r	w	x	r	w	x
权限分配	文件所有者			文件所属组			其他用户		

	（r）读	（w）写	（x）可执行
文件	查看内容**cat**	修改内容**vi**	作为命令使用
文件夹	列出目录内容**ls**	添加、删除 **touch**、**rm**	进入文件夹或搜索 **cd**

- **chmod**命令
  - 格式**1**：**chmod** **[ugoa]** **[+-=]** **[rwx]** 文件或目录...

对应的权限字符

u、g、o、a 分别表示
属主、属组、其他用户、所有用户

+、-、= 分别表示
增加、去除、设置权限

-
  - **-R**：递归修改指定目录下所有文件、子目录的权限

•eg:

```
[root@bogon ~]# chmod g+w,o+w a
```

```
[root@bogon ~]# chmod o-wx haha
```

- **chmod**命令
  - 格式**2**：**chmod** **nnn** 文件或目录...

3位八进制数

权限项	读	写	执行	读	写	执行	读	写	执行
字符表示	r	w	x	r	w	x	r	w	x
数字表示	4	2	1	4	2	1	4	2	1
权限分配	文件所有者			文件所属组			其他用户		

- **chown**命令
  - 必须是**root**
  - 用户和组必须存在
  - 格式：**chown** 属主 文件
    chown :属组 文件
    chown 属主:属组 文件
- **chgrp**命令
  - 格式：**chgrp** 属组 文件
  - 必须是**root**或者是文件的所有者
  - 必须是新组的成员
- 常用命令选项
  - **-R**：递归修改指定目录下所有文件、子目录的归属

操作	可以执行的用户
chmod	root和文件所有者
chgrp	root和文件所有者（必须是组成员）
chown	只有root

- 在内核级别， 文件的初始权限666
- 在内核级别， 文件夹的初始权限777
- 用umask命令控制默认权限， 临时有效

```
[root@localhost ~]# umask
0022
[root@localhost ~]# umask -S
u=rwx,g=rx,o=rx
[root@localhost ~]# umask 077
[root@localhost ~]# umask
0077
```

- 不推荐修改系统默认umask

•umask 中的 022 是用来做减数的 是 rw- rw- rw-

减去 --- -w- -w-

得到文件的默认权限是 644(rw-r--r--)

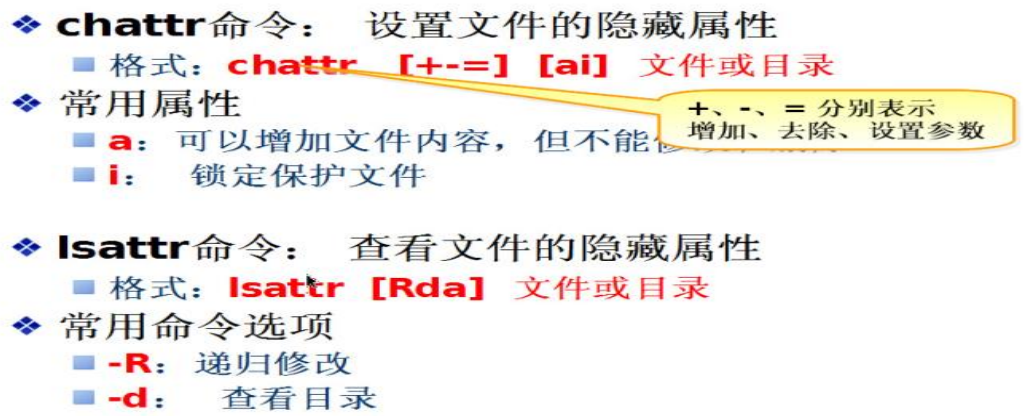
管理员 root 的 umask 为 0022

普通用户的 umask 为 0002

- 要求root在/tmp目录下创建/tmp/aa/bb这个目录，要求在这个bb目录下创建如下图所示的东东，要求（权限、属主属组、名称）完全一致。

```
drwxrwxr-x 4 haha root   4096 04-04 19:16 .
drwxr-xrwx 3 root root   4096 04-04 18:56 ..
drwx------ 2 root hello  4096 04-04 19:16 *_*
dr-xr-xr-x 2 xixi xixi   4096 04-04 19:14 <haha>
-r-x-wx--- 1 xixi haha      0 04-04 19:13 haha xixi
--w--w-r-x 1 root hello     0 04-04 19:07 .hello
```

- 使用vim文本编辑器手工创建一个用户sxjy(UID520),私有组是web(GID514)，密码是123，用户的主目录是/sxjy（注意不能用useradd,passwd,groupadd命令），最终要求可以使用sxjy用户成功登录后，在自己的主目录中新建的文件的默认权限是600，新建的文件夹的默认权限是700。



e 属性 支持属性扩展
文件由默认支持属性扩展的属性

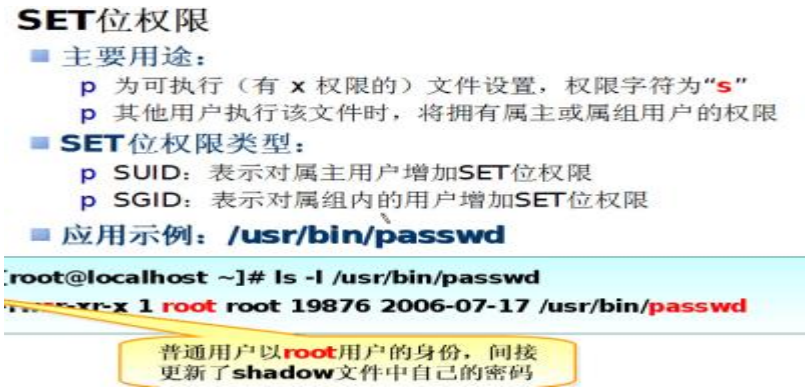
权限项	读	写	执行	读	写	执行	读	写	执行
字符表示	r	w	x	r	w	x	r	w	x
权限分配	文件所有者			文件所属组			其他用户		
特别权限	SUID			SGID			Sticky		
有x 特别权限	r	w	s	r	w	s	r	w	t
无x	r	w	S	r	w	S	r	w	T

set 位权限 （设置位权限）
SUID SGID 如果文件没有 x 权限 则不能设置为 s(t) 只能是 S(T)

Sticky 粘滞位权限

如果一个文件的 x 位为 s（含有 x 权限），S（不含有 x 权限） 则说明被设置了设置位权限

对可执行的二进制文件才可以加 s 权限



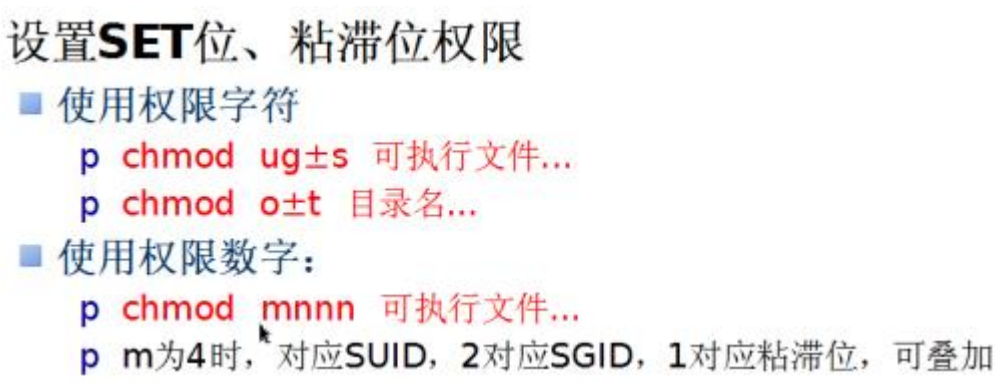
对文件设置了 set 位权限，则其他用户使用该文件时，可以有该用户属主或者属组的用户的身份去使用该文件

比如：

```
[root@server1 lianxi]# ll /usr/bin/passwd
-rwsr-xr-x. 1 root root 30768 Feb 17  2012 /usr/bin/passwd
[root@server1 lianxi]#
```

对公共目录设置了 t 确保用户只能操纵自己的资料

chmod o+(-)t 文件（夹）名



设置了 SGID 的文件夹中 创建的文件的组别是文件夹的组别

ACL(Access Control List)

- 一个文件/目录的访问控制列表，可以针对任意指定的用户/组使用权限字符分配**rwx**权限

设置ACL：setfacl指令

- 格式： **setfacl** 选项 规则 文件

常用选项

- **-m**：新增或修改**ACL**中的规则
- **-b**： 删除所有**ACL**规则
- **-x**： 删除指定的**ACL**规则

查看ACL：getfacl指令

- 格式： **getfacl** 文件

· -R 递归更改（目录下的子目录以及文件也应用此 ACL 规则）

设置ACL：setfacl指令

- 格式： **setfacl** 选项 规则 文件

常用规则

- 格式：类型:特定的用户或组:权限
- **user:(uid/name):(perms)** 指定某位使用者的权限
- **group:(gid/name):(perms)** 指定某一群组的权限
- **other::(perms)** 指定其它使用者的权限
- **mask::(perms)** 设定有效的最大权限

注意

- user、group、other、mask简写为：u，g，o，m

mask 用 m 简写

❖ 针对特殊用户

```
[root@sxkj ~]# ls -l test.txt
-rw-r--r-- 1 root root 0 Aug  4 09:10 test.txt
[root@sxkj ~]# setfacl -m u:hello:rw test.txt  =>对特定用户赋予权限
[root@sxkj ~]# getfacl test.txt                =>查看acl权限
# file: test.txt
# owner: root
# group: root
user::rw-
user:hello:rw-                                 =>对特定用户赋予权限
group::r--
mask::rw-
other::r--
```

命令的执行对象是文件或者文件夹

❖ 针对mask设置有效权限

```
[root@sxkj ~]# setfacl -m m::r test.txt          =>设置有效权限限
[root@sxkj ~]# getfacl test.txt                   =>查看acl权限
# file: test.txt
# owner: root
# group: root
user::rw-
user:hello:rw-              #effective:r--        =>有效的其实只有r
    权限
group::r--
group:sxkj:rw-              #effective:r--        =>有效的其实只有r
    权限
mask::r--
other::r--
```

mask 限制了文件或者文件夹的最大权限，即使单独给某个用户的权限，如果权限超过了 mask 的值，实际上也是只有 mask 所定的权限

ACL类型

- 存取型ACL(Access ACL)：文件或目录
- 预设型ACL(Default ACL)：只能对目录

预设型ACL(Default ACL)

- 格式：setfacl -m default:类型:特定的用户或组:权限

  setfacl -m d:类型:特定的用户或组:权限
- 设置了预设型ACL的目录，其下的所有文件或者子目录就都具有子主目录的ACL权限，并且子目录也同样有预设的ACl权限

❖ 设置预设ACL

```
[root@rhela ~]# setfacl -m d:u:hello:rw soft      =>设置默认权限
[root@rhela ~]# getfacl soft
...
user::rwx
group::r-x
other::r-x
default:user::rwx
default:user:hello:rw-
default:group::r-x
default:mask::rwx
default:other::r-x
```

复杂命令

xargs

这是给命令传递参数的一个过滤器，也是组合多个命令的一个工具.它把一个数据流分割为一些足够小的块，以方便过滤器和命令进行处理. 由此这个命令也是后置引用的一个强有

力的替换. 在一般使用过多参数的命令替换失败的时候,用xargs 来替换它一般都能成功.
[1] 通常情况下, xargs 从管道或者stdin 中读取数据, 但是它也能够从文件的输出中读取数据.
xargs 的默认命令是 echo. 这意味着通过管道传递给xargs 的输入将会包含换行和空白, 不过通过 xargs 的处理, 换行和空白将被空格取代.

expr
通用求值表达式: 通过给定的操作(参数必须以空格分开)连接参数,并对参数求值.可以使算术操作, 比较操作, 字符串操作或者是逻辑操作.

expr 3 + 5
返回 8
expr 5 % 3
返回 2
expr 1 / 0
返回错误消息, expr: division by zero
不允许非法的算术操作.
expr 5 * 3
返回 15
在算术表达式 expr 中使用乘法操作时, 乘法符号必须被转义.

时间/日期命令
时间/日期和计时
date
直接调用, date 就会打印日期和时间到 stdout 上.
at
at 命令是一个作业控制命令, 用来在指定时间执行给定的命令集合.它有点像 cron 命令, 然而, at 命令主要还是用来执行那种一次性执行的命令集合.

batch
batch 作业控制命令与 at 命令的行为很相像, 但 batch 命令被用来在系统平均载量降到 0.8 以下时执行一次性的任务. 与 at 命令相似的是, 它也可以使用 -f 选项来从文件中读取命令.
cal
从 stdout 中输出一个格式比较整齐的日历. 也可以指定年和月来显示那个月的日历.
sleep
这个命令与一个等待循环的效果一样. 你可以指定需要暂停的秒数, 这段时间将什么都不干.
usleep
Microsleep 睡眠微秒("u" 会被希腊人读成 "mu", 或者是 micro- 前缀). 与上边的 sleep 命令作用相同, 但这个命令是以百万分之一秒为单位的. 当需要精确计时, 或者需要非常频繁的监控一个正在运行的进程的时候, 这个命令非常有用.

hwclock, clock

hwclock 命令可以存取或调整硬件时钟. 这个命令的一些选项需要 root 权限. 在系统启动的时候, /etc/rc.d/rc.sysinit 这个启动文件,会使用 hwclock 来从硬件时钟中读取并设置系统时间.clock at bootup.
clock 命令与 hwclock 命令完全相同.

文本处理命令

sort
文件排序, 通常用在管道中当过滤器来使用. 这个命令可以依据指定的关键字或指定的字符位置, 对文件行进行排序. 使用 -m 选项, 它将会合并预排序的输入文件.

tsort
拓扑排序 ,读取以空格分隔的有序对, 并且依靠输入模式进行排序.

uniq
这个过滤器将会删除一个已排序文件中的重复行.这个命令经常出现在 sort 命令的管道后边.

expand, unexpand
expand 将会把每个tab 转化为一个空格.这个命令经常用在管道中.
unexpand 将会把每个空格转化为一个tab.效果与 expand 相反.

cut
一个从文件中提取特定域的工具. 这个命令与 awk 中使用的 print $N 命令很相似, 但是更受限. 在脚本中使用cut 命令会比使用 awk 命令来得容易一些. 最重要的选项就是 -d (字段定界符) 和 -f (域分隔符) 选项.

paste
将多个文件,以每个文件一列的形式合并到一个文件中, 合并后的文件没列就是原来的一个文件.对于创建系统log 文件来说, 使用 cut 命令与 paste 命令相结合是非常有用的.

join
这个命令与 paste 命令属于同类命令, 但是它能够完成某些特殊的目地. 这个强力工具能够以一种特殊的形式来合并 2 个文件, 这种特殊的形式本质上就是一个关联数据库的简单版本.
join 命令只能够操作2 个文件, 它可以将那些具有特定标记域(通常是一个数字标签)的行合并起来, 并且将结果输出到stdout. 被加入的文件应该事先根据标记域进行排序以便于能够正确的匹配.

head
将一个文件的头打印到 stdout 上 (默认为10 行, 可以自己修改)

tail
将一个文件的结尾输出到 stdout 中(默认为 10 行). 通常用来跟踪一个系统 logfile

的修改状况，使用 -f 选项的话, tail 命令将会继续显示添加到文件中的行.

grep

使用 正则表达式的一个多用途文本搜索工具. 这个命令本来是 ed 行编辑器中的一个命令/过滤器: g/re/p -- global - regular expression - print.

grep pattern [file...]

在文件中搜索所有 pattern 出现的位置, pattern 既可以是要搜索的字符串,也可以是一个正则表达式.

-i 选项在搜索时忽略大小写.

-w 选项用来匹配整词.

-l 选项仅列出符合匹配的文件，而不列出匹配行.

-r (递归) 选项不仅在当前工作目录下搜索匹配，而且搜索子目录.

-n 选项列出所有匹配行，并显示行号.

-v (或者--invert-match) 选项将会显示所有不匹配的行.

-c (--count) 选项将只会显示匹配到的行数的总数,而不会列出具体的匹配.

如果存在一个成功的匹配，那么 grep 命令将会返回 0 作为退出状态 ,这样就可以将 grep 命令的结果放在脚本的条件测试中来使用，尤其和 -q (禁止输出)选项组合时特别有用.

egrep - 扩展的 grep - 这个命令与 grep -E 等价. 这个命令用起来有些不同，由于正则表达式扩展，将会使得搜索更具灵活性.

fgrep - 快速的 grep - 这个命令与 grep -F 等价. 这是一种按照字符串字面意思进行的搜索(即不允许使用正则表达式)，这样有时候会使搜索变得容易一些.

look

命令 look 与命令 grep 很相似，但是这个命令只能做字典查询，也就是它所搜索的文件必须已经排过序的单词列表. 默认情况下，如果没有指定搜索那个文件，那就默认搜索 /usr/dict/words 文件(译者：感觉好像应该是/usr/share/dict/words)，当然也可以指定其他目录下的文件进行搜索.

sed, awk

这个两个命令都是独立的脚本语言，尤其适合分析文本文件和命令输出. 既可以单独使用，也可以结合管道和在 shell 脚本中使用.

sed

非交互式的 "流编辑器"，在批量模式下，允许使用许多 ex 命令.你会发现它在shell 脚本中非常有用.

awk

可编程的文件提取器和文件格式化工具，在结构化的文本文件中,处理或提取特定域(特定列)具有非常好的表现.它的语法与 C 语言很类似.

wc

wc 可以统计文件或 I/O 流中的单词数量.

wc -w 统计单词数量.

wc -l 统计行数量.

wc -c 统计字节数量.
wc -m 统计字符数量.
wc -L 给出文件中最长行的长度.

tr

字符转换过滤器.
注意: 必须使用引用或中括号, 这样做才是合理的. 引用可以阻止 shell 重新解释出现在 tr 命令序列中的特殊字符.中括号应该被引用起来防止被shell 扩展.
无论 tr "A-Z" "*" <filename 还是 tr A-Z * <filename 都可以将 filename 中的大写字符修改为星号(写到 stdout).但是在某些系统上可能就不能正常工作了, 而 tr A-Z '[**]' 在任何系统上都可以正常工作.
-d 选项删除指定范围的字符.
-c "complement" 选项将会反转 匹配的字符集. 通过这个选项, tr 将只会对那些不匹配的字符起作用.

fold

将输入按照指定宽度进行折行. 这里有一个非常有用的选项 -s ,这个选项可以使用空格进行断行.

fmt

一个简单的文件格式器, 通常用在管道中, 将一个比较长的文本行输出进行折行.

col

这个命令用来滤除标准输入的反向换行符号. 这个工具还可以将空白用等价的 tab 来替换. col 工具最主要的应用还是从特定的文本处理工具中过滤输出, 比如 groff 和 tbl.

column

列格式化工具. 这个过滤工具将会将列类型的文本转化为"易于打印"的表格式进行输出, 通过在合适的位置插入 tab.

colrm

列删除过滤器. 这个工具将会从文件中删除指定的列(列中的字符串)并且写到文件中, 如果指定的列不存在,那么就回到 stdout. colrm 2 4 <filename 将会在filename 文件中对每行删除第 2 到第4 列之间的所有字符.
注意: 如果这个文件包含tab 和不可打印字符, 那将会引起不可预期的行为. 在这种情况下, 应该通过管道的手段使用 expand 和 unexpand 命令来预处理 colrm.

nl

计算行号过滤器. nl filename 将会在 stdout 中列出文件的所有内容, 但是会在每个非空行的前面加上连续的行号. 如果没有 filename 参数, 那么就操作 stdin.
nl 命令的输出与 cat -n 非常相似, 然而, 默认情况下 nl 不会列出空行.

pr

格式化打印过滤器. 这个命令会将文件(或stdout)分页, 将它们分成合适的小块以便于硬拷贝打印或者在屏幕上浏览.使用这个命令的不同的参数可以完成好多任务, 比如对行和列的操作,加入行, 设置页边, 计算行号, 添加页眉, 合并文件等等. pr 命令集合了许多命令的功能, 比如 nl, paste, fold, column, 和 expand.

gettext

GNU gettext 包是专门用来将程序的输出翻译或者本地化为不同国家语言的工具集.在最开始的时候仅仅支持 C 语言, 现在已经支持了相当数量的其它程序语言和脚本语言.

msgfmt
一个产生2 进制消息目录的程序. 这个命令主要用来本地化.
iconv
一个可以将文件转化为不同编码格式(字符集)的工具. 这个命令主要用来本地化.

recode
可以认为这个命令时上边 iconv 命令的一个空想家版本. 这个非常灵活的并可以把整个文件都转换为不同编码格式的工具并不是 Linux 标准安装的一部分.

TeX, gs
TeX 和 Postscript 都是文本标记语言, 用来对打印和格式化的视频显示进行预拷贝.
TeX 是 Donald Knuth 精心制作的排版系统. 通常情况下, 通过编写脚本的手段来把所有的选项和参数封装起来一起传到标记语言中是一件很方便的事情.
Ghostscript (gs) 是一个遵循 GPL 的Postscript 解释器.

enscript
将纯文本文件转换为 PostScript 的工具
比如, enscript filename.txt -p filename.ps 产生一个 PostScript 输出文件 filename.ps.

groff, tbl, eqn
另一种文本标记和显示格式化语言是 groff. 这是一个对传统 UNIX roff/troff 显示和排版包的 GNU 增强版本.Man 页使用的就是 groff.
tbl 表处理工具可以认为是 groff 的一部分, 它的功能就是将表标记转化到 groff 命令中.
eqn 等式处理工具也是 groff 的一部分, 它的功能是将等式标记转化到 groff 命令中.

文件与归档命令

归档命令
tar
一些有用的 tar 命令选项:
1. -c 创建 (一个新的归档文件)
2. -x 解压文件 (从存在的归档文件中)
3. --delete 删除文件 (从存在的归档文件中)
注意: 这个选项不能用于磁带类型设备.
4. -r 将文件添加到现存的归档文件的尾部
5. -A 将 tar 文件添加到现存的归档文件的尾部
6. -t 列出现存的归档文件中包含的内容
7. -u 更新归档文件
8. -d 使用指定的文件系统比较归档文件
9. -z 用 gzip 压缩归档文件
(压缩还是解压, 依赖于是否组合了 -c 或 -x)选项

10. -j 用 bzip2 压缩归档文件
注意: 如果想从损坏的用 gzip 压缩过的 tar 文件中取得数据, 那将是很困难的. 所有当我们归档重要的文件的时候, 一定要保留多个备份.

shar

Shell 归档工具. 存在于 shell 归档文件中的所有文件都是未经压缩的, 并且本质上是一个 shell 脚本,以 #!/bin/sh 开头, 并且包含所有必要的解档命令. Shar 归档文件至今还在 Internet 新闻组中使用, 否则的话 shar 早就被 tar/gzip 所取代了. unshar 命令用来解档 shar 归档文件.

ar

创建和操作归档文件的工具, 主要在对2 进制目标文件打包成库时才会用到.

rpm

Red Hat 包管理器, 或者说 rpm 工具提供了一种对源文件或2 进制文件进行打包的方法. 除此之外, 它还包括安装命令, 并且还检查包的完整性.

cpio

这个特殊的归档拷贝命令(拷贝输入和输出)现在已经很少能见到了, 因为它已经被 tar/gzip 所替代了.现在这个命令只在一些比较特殊的地方还在使用,比如拷贝一个目录树.

rpm2cpio

这个命令可以从 rpm 归档文件中解出一个 cpio 归档文件.

压缩命令

gzip

标准的 GNU/UNIX 压缩工具, 取代了比较差的 compress 命令. 相应的解压命令是 gunzip,gzip -d 是等价的.

bzip2

用来压缩的一个可选的工具, 通常比 gzip 命令压缩率更高(所以更慢), 适用于比较大的文件. 相应的解压命令是 bunzip2.
注意: 新版本的 tar 命令已经直接支持 bzip2 了.

compress, uncompress

这是一个老的, 私有的压缩工具, 一般的商业 UNIX 发行版都会有这个工具. 更有效率的 gzip 工具早就把这个工具替换掉了. Linux 发行版一般也会包含一个兼容的 compress 命令, 虽然 gunzip 也可以加压用 compress 工具压缩的文件.
注意: znew 命令可以将 compress 压缩的文件转换为 gzip 压缩的文件.

sq

另一种压缩工具, 一个只能工作于排过序的 ASCII 单词列表的过滤器.这个命令使用过滤器标准的调用语法, sq < input-file > output-file. 速度很快, 但是效率远不及 gzip. 相应的解压命令为 unsq, 调用方法与 sq 相同.
注意: sq 的输出可以通过管道传递给 gzip 以便于进一步的压缩.

zip, unzip

跨平台的文件归档和压缩工具, 与 DOS 下的 pkzip.exe 兼容. zip 归档文件看起来在互联网上比 tar 包更流行.

unarc, unarj, unrar

这些 Linux 工具可以用来解档那些用 DOS 下的 arc.exe, arj.exe, 和 rar.exe 程序进

行归档的文件.文件信息

file

确定文件类型的工具. 命令 file file-name 将会用 ascii 文本或数据的形式返回 file-name 文件的详细描述. 这个命令会使用 /usr/share/magic, /etc/magic, 或 /usr/lib/magic 中定义的魔法数字 来标识包含某种魔法数字的文件, 上边所举出的这 3个文件需要依赖于具体的 Linux/UNIX 发行版.

-f 选项将会让 file 命令运行于批处理模式, 也就是说它会分析 -f 后边所指定的文件, 从中读取需要处理的文件列表, 然后依次执行 file 命令. -z 选项, 当对压缩过的目标文件使用时, 将会强制分析压缩的文件类型.

which

which command-xxx 将会给出 "command-xxx" 的完整路径. 当你想在系统中准确定位一个

特定的命令或工具的时候, 这个命令就非常有用了.

whereis

与上边的 which 很相似, whereis command-xxx 不只会给出 "command-xxx" 的完整路径,

而且还会给出这个命令的 man 页的完整路径.

whatis

whatis filexxx 将会在 whatis 数据库中查询 "filexxx". 当你想确认系统命令和重要的 配置文件的时候, 这个命令就非常重要了. 可以把这个命令认为是一个简单的 man 命令.

vdir

显示详细的目录列表. 与 ls -l 的效果类似.

locate, slocate

locate 命令将会在预先建立好的档案数据库中查询文件. slocate 命令是 locate 的安全版本

readlink

显示符号连接所指向的文件.

strings

使用 strings 命令在二进制或数据文件中找出可打印字符. 它将在目标文件中列出所有找到的可打印字符的序列.

比较命令

diff, patch

diff: 一个非常灵活的文件比较工具. 这个工具将会以一行接一行的形式来比较目标文件. 在某些应用中, 比如说比较单词词典, 在通过管道将结果传递给 diff 命令之前, 使用诸如 sort 和 uniq 命令来对文件进行过滤将是非常有用的.diff file-1 file-2 将会输出 2个文件不同的行,并会通过符号标识出每个不同行所属的文件.

diff 命令的 --side-by-side 选项将会把2 个比较中的文件全部输出, 按照左右分隔的形式, 并会把不同的行标记出来. -c 和 -u 选项也会使得 diff 命令的输出变得容易解释一些.
还有一些 diff 命令的变种, 比如 sdiff, wdiff, xdiff, 和 mgdiff.
注意: 如果比较的两个文件是完全一样的话, 那么 diff 命令会返回 0 作为退出码, 如果不同的话就返回 1 作为退出码. 这样 diff 命令就可以用在 shell 脚本的测试结构中了.
diff 命令的一个重要用法就是产生区别文件, 这个文件将用作 patch 命令的 -e 选项的参数, -e 选项接受 ed 或 ex 脚本.
patch: 灵活的版本工具.给出一个用 diff 命令产生的区别文件, patch 命令可以将一个老版本的包更新为一个新版本的包. 因为你发布一个小的区别文件远比重新发布一个大的软件包来的容易得多.对于频繁更新的 Linux 内核来说, 使用补丁包的形式来发布将是一种很好的方法.

```
1 patch -p1 <patch-file
2 # 在'patch-file'中取得所有的修改列表
3 # 然后把它们应用于其中索引到的文件上.
4 # 那么这个包就被更新为新版本了.
```

diff3
一个 diff 命令的扩展版本, 可以同时比较3 个文件. 如果成功执行那么这个命令就返回0, 但是不幸的是这个命令不给出比较结果的信息.

sdiff
比较 和/或编辑 2 个文件, 将它们合并到一个输出文件中. 因为这个命令的交互特性, 所以在脚本中很少使用这个命令.

cmp
cmp 命令是上边 diff 命令的一个简单版本. diff 命令会报告两个文件的不同之处, 而 cmp 命令仅仅指出那些位置有不同, 而不会显示不同的具体细节.
注意: 与 diff 一样,如果两个文件相同 cmp 返回0 作为退出码, 如果不同返回1. 这样就可以用在 shell 脚本的测试结构中了.

comm
多功能的文件比较工具. 使用这个命令之前必须先排序.

```
comm -options first-file second-file
```

comm file-1 file-2 将会输出3 列:
* 第 1 列 = 只在 file-1 中存在的行
* 第 2 列 = 只在 file-2 中存在的行
* 第 2 列 = 两边相同的行.

下列选项可以禁止1 列或多列的输出.
* -1 禁止显示第一栏 (在 File1 中的行)
* -2 禁止显示第二栏 (在 File2 中的行)
* -3 禁止显示第三栏 (File1 和 File2 公共的行)
* -12 禁止第一列和第二列, (就是说选项可以组合).

一般工具

basename

从文件名中去掉路径信息，只打印出文件名．结构 basename $0 可以让脚本知道它自己的名字，也就是，它被调用的名字．可以用来显示用法信息，比如如果你调用脚本的时候缺少参数，可以使用如下语句:

```
1 echo "Usage: `basename $0` arg1 arg2 ... argn"
```

dirname

从带路径的文件名中去掉文件名，只打印出路径信息．

注意: basename 和 dirname 可以操作任意字符串．参数可以不是一个真正存在的文件，甚至可以不是一个文件名

split, csplit

将一个文件分割为几个小段的工具．这些命令通常用来将大的文件分割，并备份到软盘上，或者是为了切成合适的尺寸用 email 上传．

csplit 根据上下文 来切割文件，切割的位置将会发生在模式匹配的地方．

sum, cksum, md5sum, sha1sum

这些都是用来产生 checksum 的工具. checksum 的目的是用来检验文件的完整性，是对文

件的内容进行数学计算而得到的．出于安全目的一个脚本可能会有一个 checksum 列表，这样可以确保关键系统文件的内容不会被修改或损坏．对于需要安全性的应用来说，应该使用 md5sum (message digest 5 checksum) 命令，或者更好的更新的 sha1sum (安全 Hash 算法).

shred

用随机字符填充文件，使得文件无法恢复，这样就可以保证文件安全的被删除．这个命令的效果与 Example 12-55 一样，但是使用这个命令是一种更优雅更彻底的方法．

这是一个 GNU fileutils.

注意: 即使使用了 shred 命令，高级的(forensic)辩论技术还是能够恢复文件的内容．

编码和解码

uuencode

这个工具用来把二进制文件编码成 ASCII 字符串,这个工具适用于编码e-mail 消息体,或者新闻组消息.

uudecode

这个工具用来把 uuencode 后的 ASCII 字符串恢复为二进制文件.

一些杂项工具

mktemp

使用一个"唯一"的文件名来创建一个临时文件 [4] ．如果不带参数的在命令行下调用这个命令时，将会在 /tmp 目录下产生一个零长度的文件.

make

build 和 compile 二进制包的工具．当源文件被增加或修改时就会触发一些操作，这个工具用来控制这些操作.

make 命令将会检查 Makefile, makefile 是文件的依赖和操作列表.

install

特殊目的的文件拷贝命令, 与 cp 命令相似, 但是具有设置拷贝文件的权限和属性的能力. 这个命令看起来是为了安装软件包所定制的, 而且就其本身而言, 这个命令经常出现在 Makefile 中(在 make install : 区中). 在安装脚本中也会看到这个命令的使用.

dos2unix

这个工具是由 Benjamin Lin 和其同事编写的, 目的是将 DOS 格式的文本文件 (以 CR-LF 为行结束符) 转换为 UNIX 格式 (以 LF 为行结束符), 反过来也一样.

ptx

ptx [targetfile] 命令将会输出目标文件的序列改变的索引(交叉引用列表). 如果必要的话, 这个命令可以在管道中进行更深层次的过滤和格式化.

more, less

分页显示文本文件或 stdout, 一次一屏.可以用来过滤 stdout 的输出 . . . 或一个脚本的输出.

more 命令的一个有趣的应用就是测试一个命令序列的执行, 来避免可能发生的糟糕的结果.

通讯命令

host

通过名字或 IP 地址来搜索一个互联网主机的信息, 使用 DNS.

ipcalc

显示一个主机 IP 信息. 使用 -h 选项, ipcalc 将会做一个 DNS 的反向查询, 通过 IP 地址找到主机(服务器)名.

nslookup

通过 IP 地址在一个主机上做一个互联网的 "名字服务查询".

dig

域信息查询. 与 nslookup 很相似, dig 在一个主机上做一个互联网的 "名字服务查询". 这个命令既可以交互运行也可以非交互运行, 换句话说, 就是在脚本中运行.

下边是一些 dig 命令有趣的选项,

+time=N 选项用来设置查询超时为 N 秒,

+nofail 选项用来持续查询服务器直到收到一个响应,

-x 选项会做反向地址查询.

traceroute

跟踪包发送到远端主机过程中的路由信息. 这个命令在 LAN, WAN, 或者在 Internet 上都可以正常工作. 远端主机可以通过 IP 地址来指定. 这个命令的输出也可以通过管道中的 grep 或 sed 命令来过滤.

ping

广播一个 "ICMP ECHO_REQUEST" 包到其他主机上, 既可以是本地网络也可以使远端网络.

这是一个测试网络连接的诊断工具，应该小心使用.

whois
执行 DNS (域名系统) 查询lookup. -h 选项允许指定需要查询的特定的 whois 服务器. 参见 Example 4-6 和 Example 12-36.

finger
取得网络上的用户信息. 另外这个命令可以显示一个用户的~/.plan, ~/.project, 和 ~/.forward 文件，如果存在的话.
处于安全上的考虑，许多网络都禁用了 finger 以及和它相关的幽灵进程.

chfn
修改 finger 命令所显示出来的用户信息.

vrfy
验证一个互联网的 e-mail 地址.

远端主机接入

sx, rx
sx 和 rx 命令使用 xmodem 协议，设置服务来向远端主机传输文件和接收文件. 这些都是通讯安装包的一般部分，比如 minicom.

sz, rz
sz 和 rz 命令使用 zmodem 协议，设置服务来向远端主机传输文件和接收文件. zmodem 协议在某些方面比 xmodem 强，比如使用更快的的传输波特率，并且可以对中断的文件进行续传.与 sx 一样 rx，这些都是通讯安装包的一般部分.

ftp
向远端服务器上传或下载的工具和协议. 一个ftp 会话可以写到脚本中自动运行.

uucp, uux, cu
uucp: UNIX 到 UNIX 拷贝. 这是一个通讯安装包，目的是为了在 UNIX 服务器之间传输文件. 使用 shell 脚本来处理 uucp 命令序列是一种有效的方法.
因为互联网和电子邮件的出现, uucp 现在看起来已经很落伍了，但是这个命令在互联网连接不可用或者不适合使用的地方，这个命令还是可以完美的运行. uucp 的优点就是它的容错性，即使有一个服务将拷贝操作中断了，那么当连接恢复的时候，这个命令还是可以在中断的地方续传.

telnet
连接远端主机的工具和协议.
注意:telnet 协议本身包含安全漏洞，因此我们应该适当的避免使用.

wget
wget 工具使用非交互的形式从 web 或 ftp 站点上取得或下载文件. 在脚本中使用正好.

lynx
lynx 是一个网页浏览器，也是一个文件浏览器. 它可以(通过使用 -dump 选项)在脚本中使用. 它的作用是可以从 Web 或 ftp 站点上非交互的获得文件.
使用 -traversal 选项, lynx 将从参数中指定的 HTTP URL 开始，遍历指定服务器上的所有链接. 如果与 -crawl 选项一起用的话，将会把每个输出的页面文本都放到一个 log 文件中.

rlogin
远端登陆，在远端的主机上开启一个会话．这个命令存在安全隐患，所以要使用 ssh 来代替．
rsh
远端 shell，在远端的主机上执行命令．这个命令存在安全隐患，所以要使用 ssh 来代替．
rcp
远端拷贝，在网络上的不同主机间拷贝文件．
rsync
远端同步，在网络上的不同主机间(同步)更新文件．
ssh
安全 shell，登陆远端主机并在其上运行命令．这个工具具有身份认证和加密的功能，可以安全的替换 telnet, rlogin, rcp, 和 rsh 等工具．
scp
安全拷贝，在功能上与 rcp 很相似，就是在2 个不同的网络主机之间拷贝文件，但是要通过鉴权的方式，并且使用与 ssh 类似的安全层．
Local Network
write
这是一个端到端通讯的工具．这个工具可以从你的终端上(console 或者 xterm)发送整行到另一个用户的终端上. mesg 命令当然也可以用来对于一个终端的写权限
因为 write 是需要交互的，所以这个命令通常不使用在脚本中．
netconfig
用来配置网络适配器(使用 DHCP)的命令行工具．这个命令对于红帽发行版来说是内置的．
Mail
mail
发送或读取 e-mail 消息．
如果把这个命令行的 mail 客户端当成一个脚本中的命令来使用的话，效果非常好．
mailto
与 mail 命令很相似, mailto 命令可以使用命令行或在脚本中发送 e-mail 消息．然而，mailto 命令也允许发送 MIME (多媒体) 消息．
vacation
这个工具可以自动回复 e-mail 给发送者，表示邮件的接受者正在度假暂时无法收到邮件．这个工具与 sendmail 一起运行于网络上，并且这个工具不支持拨号的 POPmail 帐号．

终端控制命令
影响控制台或终端的命令
tput
初始化终端或者从 terminfo data 中取得终端信息．不同的选项允许特定的终端操作．tput clear 与下边的 clear 等价. tput reset 与下边的 reset 等价. tput sgr0 也可以重置终端，但是并不清除屏幕．
infocmp
这个命令会打印出大量的当前终端的信息
reset
重置终端参数并且清除屏幕．与 clear 命令一样，光标和提示符将会重新出现在终端的

左上角.

clear

clear 命令只不过是简单的清除控制台或者 xterm 的屏幕. 光标和提示符将会重新出现在屏幕或者 xterm window 的左上角. 这个命令既可以用在命令行中也可以用在脚本中. 参见 Example 10-25.

script

这个工具将会记录(保存到一个文件中)所有的用户在控制台下的或在 xterm window 下的按键信息. 这其实就是创建了一个会话记录.

数学计算命令

"Doing the numbers"

factor

将一个正数分解为多个素数.

bash$ factor 27417

27417: 3 13 19 37

bc

Bash 不能处理浮点运算, 并且缺乏特定的一些操作,这些操作都是一些重要的计算功能. 幸运的是, bc 可以解决这个问题.

bc 不仅仅是个多功能灵活的精确的工具, 而且它还提供许多编程语言才具备的一些方便的功能.bc 比较类似于 C 语言的语法.

混杂命令

jot, seq

这些工具通过用户指定的范围和增量来产生一系列的整数.

每个产生出来的整数一般都占一行, 但是可以使用 -s 选项来改变这种设置.

getopt

getopt 命令将会分析以破折号开头的命令行选项. 这个外部命令与Bash 的内建命令 getopts 作用相同. 通过使用 -l 标志, getopt 可以处理长(多字符)选项, 并且也允许参数重置.

banner

将会把字符串用一个 ASCII 字符(默认是 '#')来画出来(就是将多个'#'拼出一副字符的图形).可以作为硬拷贝重定向到打印机上(译者注: 可以使用-w 选项设置宽度).

printenv

对于某个特定的用户, 显示出所有的环境变量.

lp

lp 和 lpr 命令将会把文件发送到打印队列中, 并且作为硬拷贝来打印. 这些命令会纪录它们名字的起始位置并传递到行打印机的另一个位置.

tee
这是一个重定向操作，但是有些不同．就像管道中的"三通"一样，这个命令可以将命令或者管道命令的输出抽出到一个文件中,而且并不影响结果．当你想将一个正在运行的进程的输出保存到文件中时，或者为了debug 而保存输出记录的时候，这个命令就非常有用了．

mkfifo
这个不大引人注意的命令可以创建一个命名管道，并产生一个临时的先进先出的buffer 用来在两个进程间传输数据. [3] 典型的使用是一个进程向FIFO 中写数据，另一个进程读出来.

pathchk
这个命令用来检查文件名的有效性．如果文件名超过了最大允许长度(255 个字符)，或者它所在的一个或多个路径搜索不到，那么就会产生一个错误结果.
不幸的是,并不能够返回一个可识别的错误码，因此它在脚本中几乎没有什么用．一般都使用文件测试操作.

dd
用途. dd 命令只不过是简单的拷贝一个文件 (或者 stdin/stdout)，但是它会做一些转换.

od
od(octal dump)过滤器，将会把输入(或文件)转换为8 进制或者其他进制.

hexdump
对二进制文件进行 16 进制, 8 进制, 10 进制，或者 ASCII 码的查阅动作．这个命令大体上与上边的 od 命令作用相同，但是远不及 od 命令有用.

objdump
显示编译后的 2 进制文件或2 进制可执行文件的信息，以16 进制的形式显示，或者显示反汇编列表(使用-d 选项).

mcookie
这个命令会产生一个"magic cookie"，这是一个128-bit (32-字符) 的伪随机16 进制数字,这个数字一般都用来作为 X server 的鉴权"签名".

units
这个工具用来在不同的计量单位之间互相转换.

dialog
dialog 工具集提供了一种从脚本中调用交互对话框的方法. dialog 的更好的变种版本是 -- gdialog, Xdialog，和 kdialog -- 事实上是调用的 X-Windows 的界面工具集.

sox
sox 命令, "sound exchange" (声音转换)命令，可以进行声音文件的转换.

系统与管理命令

在/etc/rc.d 目录中的启动和关机脚本中包含了好多有用的(和没用的)这些系统管理命令. 这些命令通常总是被 root 用户使用, 用与系统维护或者是紧急文件系统修复.一定要小心使用这些工具, 因为如果滥用的话, 它们会损坏你的系统.

Users 和 Groups 类命令

users

显示所有的登录的用户. 这个命令与 who -q 基本一致.

groups

列出当前用户和他所属于的组. 这相当于 $GROUPS 内部变量, 但是这个命令将会给出组名字, 而不是数字.

chown, chgrp

chown 命令将会修改一个或多个文件的所有权. 对于root 来说这是一种非常好的将文件的所有权从一个用户换到另一个用户的方法. 一个普通用户不能修改文件的所有权, 即使他是文件的宿主也不行.

```
root# chown bozo *.txt
```

chgrp 将会修改一个或个文件党组所有权. 你必须是这些文件的宿主, 并且是目的组的成员(或者root), 这样才能使用这个操作.

useradd, userdel

useradd 管理命令将会在系统上添加一个用户帐号, 并且如果指定的话, 还会为特定的用户创建 home 目录. 相应的userdel 命令将会从系统上删除一个用户帐号, 并且删除相应的文件.

注意: adduser 命令与useradd 是相同的, adduser 通常都是一个符号链接.

usermod

修改用户帐号. 可以修改密码, 组身份, 截止日期, 或者给定用户帐号的其他的属性. 使用这个命令, 用户的密码可能会被锁定, 因为密码会影响到帐号的有效性.

groupmod

修改指定组. 组名字或者ID 号都可以使用这个命令来修改.

id

id 将会列出当前进程的真实和有效用户ID, 还有用户的组ID. 这与Bash 的内部变量 $UID, $EUID, 和 $GROUPS 很相像.

who

显示系统上所有已经登录的用户.

w

显示所有的登录的用户和属于它们的进程. 这是一个who 的扩展版本. w 的输出可以通过管道传递到 grep 中, 这样就可以查找指定的用户或进程.

logname

显示当前用户的登录名(可以在/var/run/utmp 中找到). 这与上边的whoami 很相近.

su

使用一个代替的用户来运行一个程序或脚本. su rjones 将会以 rjones 来启动一个

shell. 一个不加参数的su 默认就是root.

sudo
以 root(或其他用户)的身份来运行一个命令. 这个命令可以运行在脚本中, 这样就允许以正规的用户身份来运行脚本.

passwd
设置, 修改, 或者管理用户的密码.

ac
显示用户登录的连接时间

last
用户最后登录的信息, 就像从/var/log/wtmp 中读出来一样

newgrp
不用登出就可以修改用户的组 ID. 并且允许存取新组的文件. 因为用户可能同时属于多个组, 这个命令很少被使用.

终端类命令
tty
显示当前用户终端的名字. 注意每一个单独的xterm 窗口都被算作一个不同的终端.

stty
显示并(或)修改终端设置. 这个复杂命令可以用在脚本中, 并可以用来控制终端的行为和其显示输出的方法.

setterm
设置特定的终端属性. 这个命令将向它的终端的stdout 写一个字符串, 这个字符串将修改终端的行为.

setserial
设置或者显示串口参数. 这个脚本只能被root 用户来运行, 并且通常都在系统安装脚本中使用.

getty, agetty
一个终端的初始化过程通常都是使用 getty 或agetty 来建立, 这样才能让用户登录. 这些命令并不用在用户的 shell 脚本中. 它们的行为与stty 很相似.
mesg
使能或禁用当前用户终端的存取权限. 禁用存取权限将会阻止网络上的另一用户向这个终端写消息.
注意: 当你正在编写文本文件的时候, 在文本中间突然来了一个莫名其妙的消息, 这对你来说是非常烦人的. 在多用户的网络环境下, 当你不想被打断的时候, 你可能因此希望禁用对你终端的写权限.

wall
这是一个缩写单词 "write all"，也就是，向登录到网络上的任何终端的所有用户都发送一个消息．最早这是一个管理员的工具，很有用，比如，当系统有问题的时候，管理可以警告系统上的所有人暂时离开

信息与统计类
uname
输出系统的说明(OS，内核版本，等等.)到stdout．使用 -a 选项，将会给出详细的信息使用-s 选项只会输出OS 类型.

arch
显示系统的硬件体系结构．等价于 uname –m

lastcomm
给出前一个命令的信息，存储在/var/account/pacct 文件中．命令名字与用户名字都可以使用选项来指定．这是GNU 的一个统计工具.
lastlog
列出系统上所有用户最后登录的时间．存在/var/log/lastlog 文件中.
lsof
列出打开的文件．这个命令将会把所有当前打开的文件列出一份详细的表格，包括文件的所有者信息，尺寸，与它们相关的信息等等．当然，lsof 也可以管道输出到 grep 和(或) awk 来分析它的结果.

strace
为了跟踪系统和信号的诊断和调试工具．调用它最简单的方法就是strace COMMAND.

nmap
网络端口扫描器．这个命令将会扫描一个服务器来定位打开的端口，并且定位这些端口相关的服务．这是一个防止网络被黑客入侵的一个重要的安全工具.

nc
nc(netcat)工具是一个完整的工具包，可以使用它来连接和监听TCP 和UDP 端口．它可以用来作为诊断和测试工具，也可以用来作为基于脚本的HTTP 客户端和服务器的组件.

free
使用表格形式来显示内存和缓存的使用情况.

procinfo
从/proc pseudo-filesystem 中提取和显示所有信息和统计资料．这个命令将给出更详细的信息.

lsdev
显示设备，也就是显示安装的硬件.

du
递归的显示(磁盘)文件的使用状况. 除非指定, 默认是当前工作目录.
df
使用列表的形式显示文件系统的使用状况.

dmesg
将所有的系统启动消息输出到 stdout 上. 方便出错,并且可以查出安装了哪些设备驱动和察看使用了哪些系统中断. dmesg 命令的输出当然也可以在脚本中使用 grep, sed, 或 awk 来进行分析.

stat
显示一个或多个给定文件(也可以是目录文件或设备文件)的详细的统计信息.

vmstat
显示虚拟内存的统计信息.

netstat
显示当前网络的统计和信息, 比如路由表和激活的连接. 这个工具存取/proc/net(第27 章)中的信息.

uptime
显示系统运行的时间, 还有其他一些统计信息.

hostname
显示系统的主机名字.

hostid
显示主机的 32 位的16 进制ID.

sar
sar (System Activity Reporter 系统活动报告) 命令将会给出系统统计的一个非常详细的概要.

readelf
显示指定的 elf 格式的2 进制文件的统计信息. 这个工具是binutils 工具包的一部分.

系统日志类
logger
附加一个用户产生的消息到系统日之中 (/var/log/messages). 不是root 用户也可以调用 logger.

logrotate
这个工具用来管理系统的 log 文件, 可以在合适的时候轮换, 压缩, 删除, 和(或)e-mail 它们. 这个工具将从老的log 文件中取得一些杂乱的记录保存在/var/log 中. 通常使用

cron 来每天运行logrotate.
在/etc/logrotate.conf 中添加合适的入口就可以管理自己的log 文件了，就像管理系统log 文件一样.
注意: Stefano Falsetto 创造了rottlog，他认为这是logrotate 的改进版本.

作业控制

ps

进程统计：通过进程所有者和PID(进程ID)来列出当前执行的进程．通常都是使用ax 选项来调用这个命令，并且结果可以通过管道传递到 grep 或 sed 中来搜索特定的进程

pgrep, pkill

ps 命令与grep 或kill 结合使用.

pstree

使用"树"形式列出当前执行的进程. -p 选项显示PID,和进程名字.

top

连续不断的显示 cpu 使用率最高的进程. -b 选项将会以文本方式显示，以便于可以在脚本中分析或存取.

nice

使用修改后的优先级来运行一个后台作业．优先级从19(最低)到-20(最高)．只有root 用户可以设置负的(比较高的)优先级．相关的命令是renice, snice, 和skill.

nohup

保持一个命令的运行，即使用户登出系统．这个命令做为前台进程来运行，除非前边加 &. 如果你在脚本中使用 nohup 命令，最好和wait 命令一起使用，这样可以避免创建一个孤儿进程或僵尸进程.

pidof

取得一个正在运行的作业的进程 ID(PID)．因为一些作业控制命令，比如kill 和renice 只能使用进程的PID(而不是它的名字)，所以有时候必须的取得PID. pidof 命令与$PPID 内部变量非常相似.

fuser

取得一个正在存取某个或某些文件(或目录)的进程ID．使用-k 选项将会杀掉这些进程．对于系统安全来说，尤其是在脚本中想阻止未被授权的用户存取系统服务的时候，这个命令就显得很有用了.

cron

管理程序调度器，执行一些日常任务，比如清除和删除系统log 文件，或者更新slocate 命令的数据库．这是at 命令的超级用户版本(虽然每个用户都可以有自己的crontab 文件，并且这个文件可以使用 crontab 命令来修改)．它以幽灵进程T 的身份来运行，并且从/ect/crontab 中获得执行的调度入口.

进程控制和启动类

init

init 命令是所有进程的父进程．在系统启动的最后一步调用, init 将会依据/etc/inittab 来决定系统的运行级别．只能使用root 身份来运行它的别名telinit.

telinit

init 命令的符号链接，这是一种修改系统运行级别的一个手段，通常在系统维护或者紧急的文件系统修复的时候才用．只能使用root 身份调用．调用这个命令是非常危险的 - 在你使用之前确定你已经很好地了解它．

runlevel

显示当前和最后的运行级别，也就是，确定你的系统是否终止(runlevel 为0)，还是运行在单用户模式(1)，多用户模式(2)，或者是运行在X Windows(5)，还是正在重启(6)．这个命令将会存取/var/run/utmp 文件．

halt, shutdown, reboot

设置系统关机的命令，通常比电源关机的优先级高．

service

开启或停止一个系统服务．启动脚本在/etc/init.d 中，并且/etc/rc.d 在系统启动的时候使用这个命令来启动服务．

网络类

ifconfig

网络的接口配置和调试工具．

iwconfig

这是为了配置无线网络的命令集合．可以说是上边的ifconfig 的无线版本．

route

显示内核路由表信息，或者查看内核路由表的修改．

chkconfig

检查网络配置．这个命令负责显示和管理在启动过程中所开启的网络服务

tcpdump

网络包的"嗅探器"．这是一个用来分析和调试网络上传输情况的工具，它所使用的手段是把匹配指定规则的包头都显示出来．

显示主机 bozoville 和主机caduceus 之间所有传输的ip 包．

bash$ tcpdump ip host bozoville and caduceus

当然,tcpdump 的输出可以被分析，可以用我们之前讨论的文本处理工具来分析结果．

文件系统类

mount

加载一个文件系统，通常都用来安装外部设备，比如软盘或CDROM．文件/etc/fstab 将会提供一个方便的列表，这个列表列出了所有可用的文件系统，分区和设备，另外还包括某些选项，比如是否可以自动或者手动的mount．文件/etc/mtab 显示了当前已经mount 的文件系统和分区(包括虚拟的，比如/proc)．

mount -a 将会mount 所有列在/ect/fstab 中的文件系统和分区，除了那些标记有非自动选项的．在启动的时候，在/etc/rc.d 中的一个启动脚本(rc.sysinit 或者一些相似的脚本)将会这么调用, mount 所有可用的文件系统和分区．

umount

卸除一个当前已经 mount 的文件系统．在正常删除之前已经mount 的软盘和CDROM 之前,这个设备必须被 unmount，否则文件系统将会损坏．

sync
强制写入所有需要更新的 buffer 上的数据到硬盘上(同步带有buffer 的驱动器). 如果不是严格必要的话,一个sync 就可以保证系统管理员或者用户刚刚修改的数据会安全的在突然的断点中幸存下来. 在比较早以前, 在系统重启前都是使用 sync; sync (两次, 这样保证绝对可靠), 这是一种很有用的小心的方法.
有时候, 比如当你想安全删除一个文件的时候(参见 Example 12-55), 或者当磁盘灯开始闪烁的时候, 你可能需要强制马上进行buffer 刷新.
losetup
建立和配置 loopback 设备.

mkswap
创建一个交换分区或文件. 交换区域随后必须马上使用swapon 来使能.
swapon, swapoff
使能/禁用交换分区或文件. 这两个命令通常在启动和关机的时候才有效.
mke2fs
创建 Linux ext2 文件系统. 这个命令必须以root 身份调用.

tune2fs
调整 ext2 文件系统. 可以用来修改文件系统参数, 比如mount 的最大数量. 必须以root 身份调用.
注意: 这是一个非常危险的命令. 如果坏了, 你需要自己负责, 因为它可能会破坏你的文件系统.
dumpe2fs
打印(输出到stdout 上)非常详细的文件系统信息. 必须以root 身份调用.

hdparm
列出或修改硬盘参数. 这个命令必须以root 身份调用, 如果滥用的话会有危险.
fdisk
在存储设备上(通常都是硬盘)创建和修改一个分区表. 必须以root 身份使用.
注意: 谨慎使用这个命令. 如果出错, 会破坏你现存的文件系统.
fsck, e2fsck, debugfs
文件系统的检查, 修复, 和除错命令集合.

fsck: 检查UNIX 文件系统的前端工具(也可以调用其它的工具). 文件系统的类型一般都是默认的 ext2.
e2fsck: ext2 文件系统检查器.
debugfs: ext2 文件系统除错器. 这个多功能但是危险的工具的用处之一就是(尝试)恢复删除的文件. 只有高级用户才能用.
上边的这几个命令都必须以 root 身份调用, 这些命令都很危险, 如果滥用的话会破坏文件系统.
badblocks
检查存储设备的坏块(物理损坏). 这个命令在格式化新安装的硬盘时或者测试备份的完整性的时候会被用到. [4] 举个例子, badblocks /dev/fd0 测试一个软盘.

badblocks 可能会引起比较糟糕的结果(覆盖所有数据), 在只读模式下就不会发生这种情况.如果root 用户拥有需要测试的设备(通常都是这种情况), 那么root 用户必须调用这个命令.

lsusb, usbmodules

lsusb 命令会列出所有USB(Universal Serial Bus 通用串行总线)总线和使用USB 的设备. usbmodules 命令会输出连接USB 设备的驱动模块的信息.

mkbootdisk

创建启动软盘, 启动盘可以唤醒系统, 比如当MBR(master boot record 主启动记录)坏掉的时候. mkbootdisk 命令其实是一个Bash 脚本, 由Erik Troan 所编写, 放在/sbin 目录中.

chroot

修改ROOT 目录. 一般的命令都是从$PATH 中获得的, 相对的默认的根目录是 /. 这个命令将会把根目录修改为另一个目录(并且也将把工作目录修改到那). 出于安全目的, 这个命令时非常有用的, 举个例子, 当系统管理员希望限制一些特定的用户, 比如telnet 上来的用户, 将他们限定到文件系统上一个安全的地方(这有时候被称为将一个guest 用户限制在"chroot 监牢"中). 注意, 在使用chroot 之后, 系统的二进制可执行文件的目录将不再可用了.

chroot /opt 将会使得原来的/usr/bin 目录变为/opt/usr/bin. 同样,

chroot /aaa/bbb /bin/ls 将会使得ls 命令以/aaa/bbb 作为根目录, 而不是以前的/.

如果使用 alias XX 'chroot /aaa/bbb ls', 并把这句放到用户的~/.bashrc 文件中的话, 这将可以有效地限制运行命令"XX"时, 命令"XX"可以使用文件系统的范围.

当从启动盘恢复的时候(chroot 到 /dev/fd0), 或者当系统从死机状态恢复过来并作为进入 lilo 的选择手段的时候, chroot 命令都是非常方便的. 其它的应用还包括从不同的文件系统进行安装(一个rpm 选项)或者从CDROM 上运行一个只读文件系统. 只能以root 身份调用,小心使用.

注意: 由于正常的$PATH 将不再被关联了, 所以可能需要将一些特定的系统文件拷贝到 chrooted 目录中.

lockfile

这个工具是 procmail 包的一部分(www.procmail.org). 它可以创建一个锁定文件, 锁定文件是一种用来控制存取文件, 设备或资源的标记文件. 锁定文件就像一个标记一样被使用, 如果特定的文件, 设备, 或资源正在被一个特定的进程所使用("busy"), 那么对于其它进程来说, 就只能受限进行存取(或者不能存取).

mknod

创建块或者字符设备文件(当在系统上安装新硬盘时可能是必要的). MAKEDEV 工具事实上具有 nknod 的全部功能, 而且更容易使用.

MAKEDEV

创建设备文件的工具. 必须在/dev 目录下, 并且以root 身份使用.

root# ./MAKEDEV

这是 mknod 的高级版本.

tmpwatch

自动删除在指定时间内未被存取过的文件. 通常都是被cron 调用, 用来删掉老的log 文件.

备份类

dump, restore

dump 命令是一个精巧的文件系统备份工具, 通常都用在比较大的安装和网络上. 它

读取原始的磁盘分区并且以二进制形式来写备份文件. 需要备份的文件可以保存到各种各样的存储设备上, 包括磁盘和磁带. restore 命令用来恢复dump 所产生的备份.

fdformat

对软盘进行低级格式化.

系统资源类

ulimit

设置使用系统资源的上限. 通常情况下都是使用-f 选项来调用, -f 用来设置文件尺寸的限制(ulimit -f 1000 就是将文件大小限制为1M). -c(译者注: 这里应该是作者笔误, 作者写的是-t)选项来限制coredump(译者注: 核心转储, 程序崩溃时的内存状态写入文件)尺寸(ulimit -c 0 就是不要coredumps). 一般情况下, ulimit 的值应该设置在/etc/profile 和(或)~/.bash_profile 中(参见 Appendix G).

注意: Judicious 使用ulimit 可以保护系统免受可怕的fork 炸弹的迫害.

quota

显示用户或组的磁盘配额.

setquota

从命令行中设置用户或组的磁盘配额.

umask

设定用户创建文件时权限的缺省 mask(掩码).

rdev

取得 root device, swap space, 或 video mode 的相关信息, 或者对它们进行修改. 通常说来 rdev 都是被lilo 所使用, 但是在建立一个ram disk 的时候, 这个命令也很有用. 小心使用, 这是一个危险的命令.

模块类

lsmod

列出所有安装的内核模块.