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

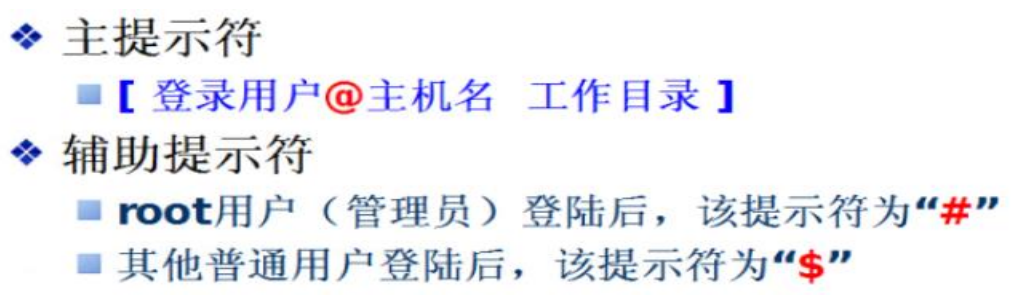
令之一.另外两个是 builtin 和enable.
builtin
在"builtin"后边的命令将只调用内建命令.暂时的禁用同名的函数或者是同名的扩展命令.
enable
这个命令或者禁用内建命令或者恢复内建命令.如: enable -n kill 将禁用kill 内建命令,所以当我们调用 kill 时,使用的将是/bin/kill 外部命令.

-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



- **Linux命令的通用命令格式**
  - 命令字 【选项】【参数】
- 选项及参数的含义
  - 选项：用于调节命令的具体功能
    - 以 “-”引导短格式选项（单个字符），例如“-l”
    - 以“--”引导长格式选项（多个字符），例如“--all”
    - 多个短格式选项可以写在一起，只用一个“-”引导，例如“-al”
  - 参数：命令操作的对象，如文件、目录名等

- 命令行编辑的几个辅助操作
  - **Tab**键： 自动补齐
  - 快捷键 **Ctrl+C**： 终止当前进程
  - 快捷键 **Ctrl+D**： 输入结束
  - 快捷键 **Ctrl+Z** ： 挂起程序
  - 快捷键 **Ctrl+L**：清屏，相当于**clear**命令
  - 快捷键 **Ctrl+K**：删除从光标到行末所有字符
  - 快捷键 **Ctrl+U**：删除从光标处到行首的字符
  - 快捷键 **Ctrl+A**：跳跃光标到行首的字符
  - 快捷键 **Ctrl+E**：跳跃光标到行尾的字符
  - 快捷键 **Ctrl+S**：锁屏
  - 快捷键 **Ctrl+Q**：解锁

- 绝对路径
  - 不考虑你当前的位置，从“/”到达目标文件需要经过的文件系统目录树的所有分支
  - **/home/hello/docs/share**
- 相对路径
  - 参照你当前的位置，到达目标文件需要经过的文件系统目录树的所有分支
  - 不以“/”开头

- 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
  - -l ：以长格式显示
  - -d：显示目录本身的属性
  - -t：按文件修改时间进行排序
  - -r ：将目录的内容清单以英文字母顺序的逆序显示
  - -a：显示所有子目录和文件的信息，包括隐藏文件
  - -A：类似于“-a”，但不显示“.”和“..”目录的信息
  - -h：以更易读的字节单位（K、M等）显示信息
  - -R：递归显示内容

•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命令

- 用途：统计目录及文件的空间占用情况（estimate file space usage）
- 格式：du [选项]... [目录或文件名]

常用命令选项

- -a：统计时包括所有的文件，而不仅仅只统计目录
- -h：以更易读的字节单位（K、M等）显示信息
- -s：只统计每个参数所占用空间总的大小

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

touch命令

- 用途：新建空文件，或更新文件时间标记
- 格式：touch 文件名...

常用命令选项

- -a ： 改变文件的读取时间记录
- -m：改变文件的修改时间记录
- -r ： 使用参考文件的时间记录
- -d： 设定时间与日期

```
[root@localhost ~]# touch file1.txt file2.doc
[root@localhost ~]# touch -ad 10:35 file1.txt
[root@localhost ~]# touch -m 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

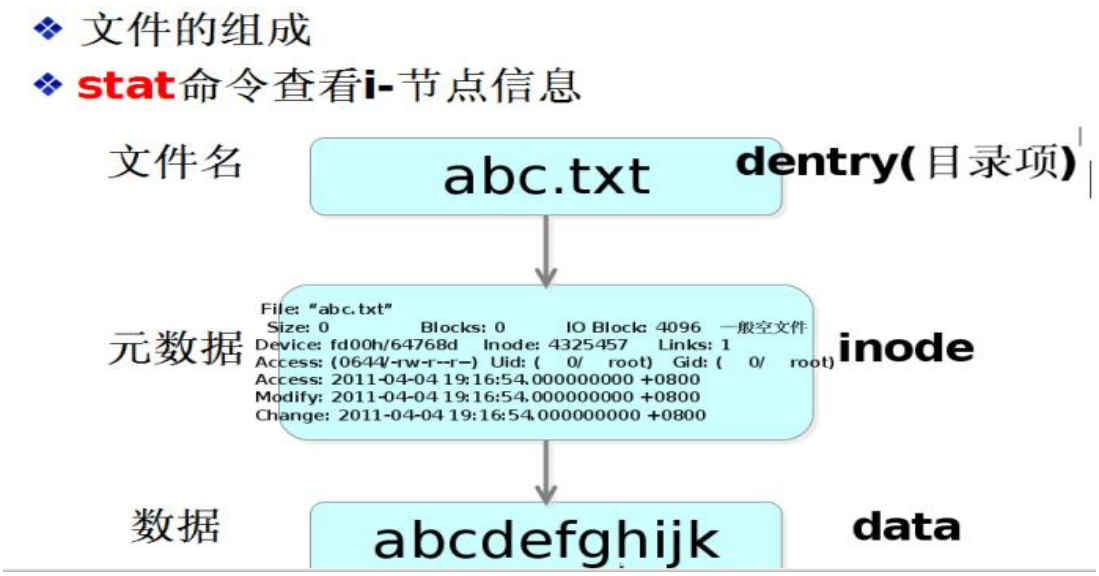
stat 命令：

```
[root@server1 ~]# stat install.log
  File: `install.log'
  Size: 37086           Blocks: 88         IO Block: 4096   regular file
Device: 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
```

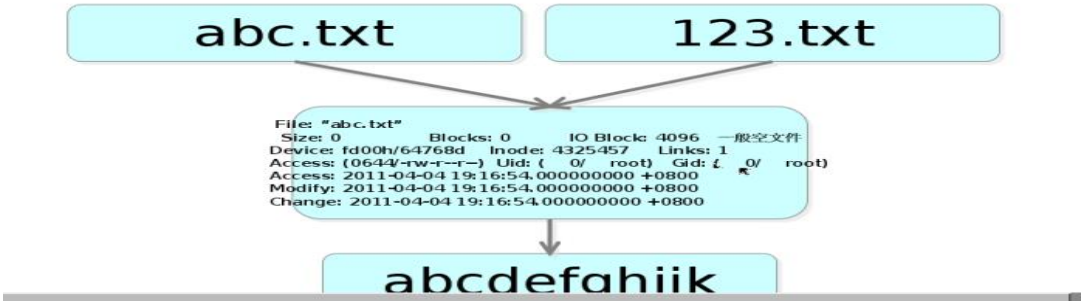
Inode 是指磁盘的编号，如果磁盘的编号用完，就算有空间也不能分配。
IO block 是指每个数据块的大小
ACCESS 是指最近的访问时间
Modify 是指最近的修改文件内容的时间
Change 是指最近的修改文件权限的时间



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

- 硬链接
  - 一个文件有多个不同的文件名
  - 命令格式：ln 源文件... 链接文件



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

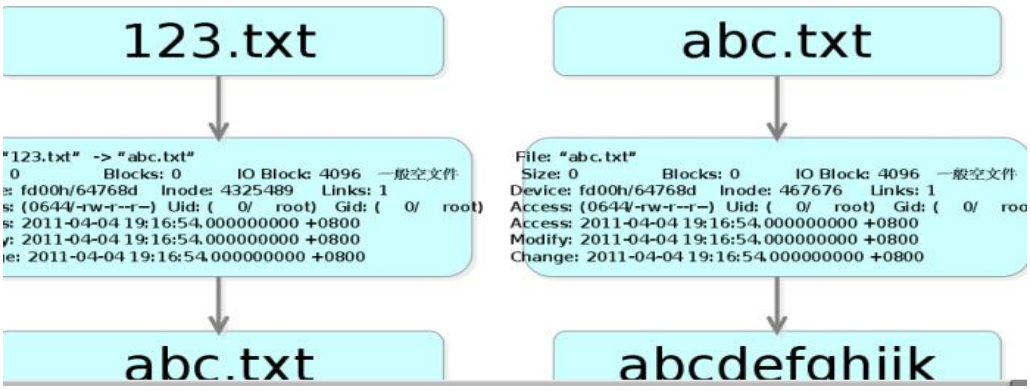
```
[root@localhost ~]# ls -i install.log
130563 install.log
[root@localhost ~]# ls -i xj
130563 xj
```

·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：
    - p **tail -n +3 /etc/passwd** 从第三行开始显示
    - p **tail -n -3 /etc/passwd** 显示最后三行
    - p **head -n -3 /etc/passwd** 不显示最后三行
    - p **head -n +3 /etc/passwd** 显示前三行

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'
```

- **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

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--)

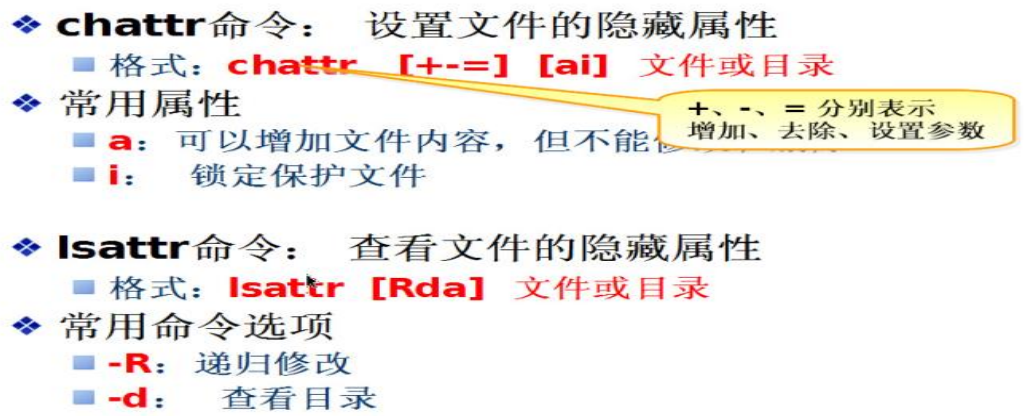
管理员 root 的 umask 为 0022

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

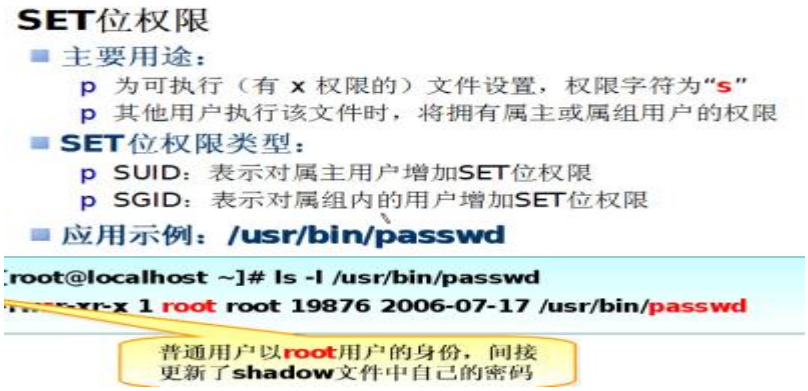
权限项	读	写	执行	读	写	执行	读	写	执行
字符表示	r	w	x	r	w	x	r	w	x
权限分配	文件所有者			文件所属组			其他用户		
特别权限	SUID			SGID			Sticky		
有x 特别权限	r	w	s	r	w	s	r	w	t
无x 特别权限	r	w	S	r	w	S	r	w	T

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



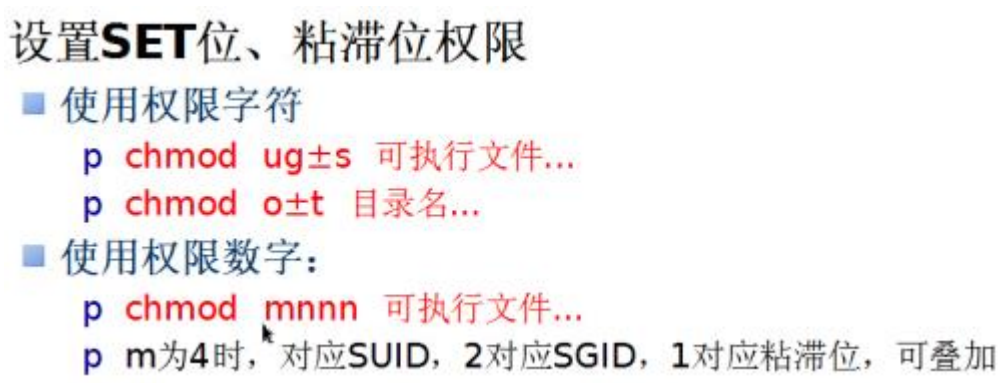
对文件设置了 set 位权限，则其他用户使用该文件时，可以有该用户属主或者属组的用户的身份去使用该文件

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
```