# [.exe的用法：理解bash解析的奥秘

if [.exe ! 8 -eq 9 ];then echo yes;fi

输出结果:yes

if [.exe ! 8 -eq 9 ;then echo yes;fi

[: missing “]” 也就是说，[是一个命令，但]不是一个命令，而[命令的最后一个参数必须是]。

$ [.exe 8 -eq 9 ] ] p p

结果[: missing "]"

$ ls /usr/bin |grep '\[' 找到[的路径，然后使用全路径名来引用

[.exe

**全路径方式**/usr/bin/[.exe 8 -eq 9 ]

总结：[不是bash的内建命令，只是一个具有返回值的普通函数

而[[则是一个内建的运算符

# 对于命令和关键字的区分

bash之中关键字不作为命令来分析，而直接截取关键字之后，结束符号之前的所有字符串作为命令来执行。

# 任何一门编程语言，掌握数组，循环，条件，输入和输出

数组 a=(I j k l m n)

循环 for I in ${a[@]};do ;done while do done

条件 if condi;then;else if;then;fi

case $i in

dd)

;;

bb)

;;

esac

# 高级部分字符串处理和一些常用函数

正则表达式:使用sed，awk

输入函数：read

# 文本处理，文件打开和关闭

输出到文本：IO重定向 > >> <<<EOF

输入cat

# 所以说这就是批处理的原则：数组

# 函数的定义

[function] f(){ statement}

当一个函数被调用时，脚本程序的位置参数（$\* $@ $#参数个数 $1 $2 $?上一条独立命令运行结果 ）等会被替换为函数的参数。

return 0//返回0表示一切正常

bash函数必有返回值，如果没有指定，就是最后执行的那条命令

# set

用set命令可以设置各种shell选项或者列出shell变量.单个选项设置常用的特性.在某些选项之后-o参数将特殊特性打开.在某些选项之后使用+o参数将关闭某些特性,不带任何参数的set命令将显示shell的全部变量.除非遇到非法的选项,否则set总是返回ture.

-a 自动向已经修改的变量或为导出后序命令的变量作出标志  
-b 不是在原提示符之前,而是立即引发终止后台任务的状态表表  
-e 如果命令带非零值返回,立即退出  
-f 禁止带扩展名的路径  
-h 定义函数时,定位和存储函数命令,当函数被执行时,通常查询函数命令  
-k 所有的关键词参数,而不只是那些命令名前的关键词参数,被放在环境命令中  
-m 监视器模式,启动任务控制.此选项默认支持系统shell交互.后台进程以单独的进程组运行,在每次完成任务时显示包含退出的状态行  
-n 读取命令但不执行命令.通常监查shell脚本的句法错误.交互shell被忽略  
-o option-name 选项名可以是下列之一:

set –o 查看所有的选项

-- Assign any remaining arguments to the positional parameters.

If there are no remaining arguments, the positional parameters

are unset.

--将会重置所有的位置参数，这在getopts之后很有用处，即参数规格化

# $的特殊参数

linux中shell变量$#,$@,$0,$1,$2的含义解释:   
变量说明:   
$$   
Shell本身的PID（ProcessID）   
$!   
Shell最后运行的后台Process的PID   
$?   
最后运行的命令的结束代码（返回值）   
$-   
使用Set命令设定的Flag一览   
$\*   
所有参数列表。如"$\*"用「"」括起来的情况、以"$1 $2 … $n"的形式输出所有参数。   
$@   
所有参数列表。如"$@"用「"」括起来的情况、以"$1" "$2" … "$n" 的形式输出所有参数。   
$#   
添加到Shell的参数个数   
$0   
Shell本身的文件名   
$1～$n   
添加到Shell的各参数值。$1是第1参数、$2是第2参数…。

# 编写脚本命令：getopts和参数规格化

无参脚本：这种脚本不处理输入，所以情况简单

## 规格化的参数

即形如command –o d –u x –long da – all

每一个命令都有非选项的参数，那个参数就代表默认要处理的数据，规格化的那类参数都在—之后。标准的规格化参数处理流程使用case

### getopt 注：getopts是内置命令，getopt是外置的

**处理命令的命令**，接受一个命令作为参数，然后接受-o选项的字符串参数，即所处理的命令能接受的参数，如下：

getopt -o nmlh – “$@”

$ getopt --help

用法：

getopt 选项字符串 参数

getopt [options] [--] 选项字符串 参数

getopt [options] -o|--options 选项字符串 [选项] [--] 参数

选项：

-a, --alternative 允许长选项以 - 开始

-h, --help 这个简短的用法指南

-l, --longoptions <长选项> 要识别的长选项

-n, --name <程序名> 将错误报告给的程序名

-o, --options <选项字符串> 要识别的短选项

-q, --quiet 禁止 getopt(3) 的错误报告

-Q, --quiet-output 无正常输出

-s, --shell <shell> 设置 shell 引用规则

-T, --test 测试 getopt(1) 版本

-u, --unquoted 不引用输出

-V, --version 输出版本信息

**在选项名中加入:表示需要一个参数**

**如** getopt -l hard: -l sj-- mke --hard yes 将被识别为两个选项

getopt -l hard -l sj-- mke --hard yes

# << here docment

cat <<EOF

afasdf

asdfa

<<EOF

# bash工具集合

cal –j 显示一年中的第几天

bc

date

cut

basename 显示基本名称

tar/gzip

seq 产生一系列的数

sort 排序

# bash其他命令集

login/logout 重新登录

stty –a 显示所有的终端设置

md5sum –c 校验md5的值

users 显示所有用户

hostid

whoami

whatis 即man –k

clear 清屏

cksum 检验crc

yes 一直输出y

talk 与另一个talk服务器的用户聊天，只要知道他的域名和用户名即可

talk [wangxz@tirc.cs.test.com](mailto:wangxz@tirc.cs.test.com)

who 显示登录的用户

wall wall this is a test line，向所有用户广播消息

hexdump/od 显示任何文件的16进制

head,tail

less,more 阅读命令

dd 文件复制，磁盘内容复制

touch 更改文件时间，如果没有指定文件名，则创建新文件

dirname 显示文件的目录

read 读取输入

let 和$(())同理

jobs,fg,bg,kill,wait 任务管理

type 查看命令是内部还是外部命令

enable [-n] linux shell命令执行时，shell总是先在自己的shell builtin中查找该命令，如果找到则执行该命令；如果找不到该命令，则会从环境变量$PATH指定的路径中依次去查找待执行的命令。因为了解了这一 点，所以看起来好像没有办法编写用户自己的命令来替代shell builtin命令。幸运的是，有了enable命令我们就能做到了。

env 显示环境变量

export 将一个变量添加或删除，从/到环境变量中

find diff whereis

strings 在一个文件中寻找可打印的字符串

uname –a 查看所有系统信息

# exec和网络编程

exec就是进程替换的意思，进程，必须理解。命令就是一个进程，一般的shell进程处于一个输入-执行-输入-执行的主循环之中，所以那种进程不会执行完就结束。

exec总是会开启一个新的进程来执行它要执行的命令，但是空间用的还是当前的空间。

**例外：**不过，要注意一个例外，当exec命令来对文件描述符操作的时候，就不会替换shell，而且操作完成后，还会继续执行接下来的命令。   
    exec 3<&0:这个命令就是将操作符3也指向标准输入。

**socket编程**

#exec 3<>/dev/tcp/www.google.com/80

#echo -e "GET / HTTP/1.1\r\nhost: http://www.google.com\r\nConnection: close\r\n\r\n" >&3

#cat <&3

也就是说，数字3在linux系统中代表了一个文件描述符，这个符号可以用于输入和输出

# 闭合符号的用法

() 开启新的进程

{…} 内部代码组，构成一个匿名函数，不会开启新的进程执行

{的前后必须没有多余的非空白字符，否则{不会被识别

[[ ]] 不同于[，这是一个关键字，里面不会发生扩展；这通常用于逻辑运算

(()) 用于执行c语言的算术，同expr。在（（））可以使用i++等语法

$(()) 表示执行其中的运算并显示结果

(())用于for循环

使用for循环读取数组：

adobe=('Flash' 'Flex' 'Photoshop' 'Dreamweaver' 'Premiere')

len=${#adobe[@]}

**for ((i=0;i<$len;i++));do**

**echo ${adobe[$i]}**

**done**

# 打印

# Flash

# Flex

# Photoshop

# Dreamweaver

# Premiere

用途说明

在shell中用于循环。类似于其他编程语言中的for，但又有些不同。for循环是Bash中最常用的语法结构。

常用格式

**格式一**

**for 变量**

**do**

**语句**

**done**

**格式二**

**for 变量 in 列表**

**do**

**语句**

**done**

**格式三**

**for ((变量=初始值; 条件判断; 变量变化))**

**do**

**语句**

**done**

## bash 的=~测试符号用于正则表达式

不能在正则表达式周围加上引号，这是不需要的。

if [[ "adfasdfadsfasd" =~ a\*.d ]];then echo yes; fi

能返回yes

if [[ "adfasdfadsfasd" =~ "a\*.d" ]];then echo yes; fi

不显示结果

**Bash字符串处理**

${} 系列函数的应用。当然，如果不喜欢，可以使用更具有代表性的python实现。