Linux 指令

# 一、文件管理

## **cat**

cat:由第一行开始显示文件内容

tac:从最后一行开始显示文件内容

cat (选项)(参数)

cat选项 -A 相当于-vET

-b 列出行号,只针对非空白行

-E 将结尾的断行字符$显示出来

-e 等价于"-vE"选项；

-n 打印行号,针对所有行

-s 当遇到有连续两行以上的空白行，就代换为一行的空白行；

-T 将[tab]键以^I显示出来

-t 等价于"-vT"选项；

-v 列出一些看不出来的特殊字符

## **chattr**

## **chgrp**

chgrp:改变文件的所属群组 root账号才可修改

chgrp -R 递归 例:chgrp users a.txt

## **[chmod](http://www.runoob.com/linux/linux-comm-chmod.html" \t "http://www.runoob.com/linux/_blank)**

chmod：改变文件权限 root账号才可修改

chmod -R 递归

xyz u/g/o的权限,r:4,w:2,x:1 例:chmod -R 770 folder1

u/g/o/a+-=rwx

r 对于文件:读取文件实际内容;对于目录:读取目录结构列表

w 对于文件:可修改文件内容;对于目录:可异动目录结构列表

x 对于文件:是否可被系统执行(与文件名无关);对于目录:可进入该目录成为工作目录

## **[chown](http://www.runoob.com/linux/linux-comm-chown.html" \t "http://www.runoob.com/linux/_blank)**

chown:改变文件拥有者 root账号才可修改

chown [-R] user[:group] file\_name/direct\_name -R 递归 例:chown root a.txt

[:group] 可直接修改群组 例:chown root:root a.txt

[.group] 可单纯修改所属群组 例:chown .root a.txt

## **cksum**

cksum:计算32位循环冗余校验(CRC)校验和每个文件的字节数. 例: cksum file.txt

## **cmp**

cmp [-clsv][-i <字符数目>][--help][第一个文件][第二个文件]

cmp 完全一样时,不显示任何信息;否则,会标示出第一个不同之处的字符和列数编号

-c或--print-chars 除了标明差异处的十进制字码之外，一并显示该字符所对应字符

-i, --ignore-initial=SKIP skip first SKIP bytes of both inputs, SKIP is a number.

-i, --ignore-initial=SKIP1:SKIP2 skip first SKIP1 bytes of FILE1 and first SKIP2 bytes of FILE2

-l或--verbose 标示出所有不一样的地方,预设只会标示出第一个

-s或--quiet或--silent 不显示错误信息

## **diff**

diff 原始比对文件 目的比对文件

diff 比较文件的差异

-B或--ignore-blank-lines 不检查空白行

-b或--ignore-space-change 不检查空格字符的不同，例: “about me” 视为与“about me”相同

-i或--ignore-case 不检查大小写的不同

## **diffstat**

## **file**

file:辨识文件类型

## **[find](http://www.runoob.com/linux/linux-comm-find.html" \t "http://www.runoob.com/linux/_blank)**

find [path] [option] [action]

find -mtime n 在n\*24-(n+1)\*24小时内被更动过内容的文件 find . -mtime 1 # find files modified between 24 and 48 hours ago

-mtime +n 在(n+1)\*24小时之前被更动过内容的文件 find . -mtime +1 # find files modified more than 48 hours ago

-mtime -n 在n\*24小时内被更动过的文件 find . -mtime -1 # find files modified less than 1 day ago # (i.e., within the past 24 hours, as before)

-atime/-ctime 同上

-amin/-cmin/-mmin 同上

-anewer/-cnewer/-mnewer/newer file newer预设与mnewer一致,列出比file还要新的文件名

-uid n

-gid n

-user name

-group name

-nouser

-nogroup

-name filename

-iname filename 忽略大小写

-size [+-]SIZE

-type TYPE 一般文件f,装置文件b/c,目录d,链接文件l,socket(s),FIFO(p)

-perm mode 文件权限刚好等于mode的文件

-perm -mode 文件权限囊括mode的权限, 如4755囊括0744, 例: -111

-perm +mode 文件权限有一个相同,就列出(已被/取代)

-perm /mode 文件权限有一个相同,就列出 例: /220; /005;/o=rx;

-exec command

-print

-empty 空的文件

-ipath p, -path p 路径名称符合 p 的文件, ipath 会忽略大小写, 路径应不以/结尾

-pid n process id 是 n 的文件

-mount, -xdev 只检查和指定目录在同一个文件系统下的文件，避免列出其它文件系统中的文件

-ok 和-exec相同，只不过在操作前要询用户

-regex<范本样式> 指定字符串作为寻找文件或目录的范本样式

-maxdepth<目录层级> 设置最大目录层级

-mindepth<目录层级> 设置最小目录层级

-prune 不寻找字符串作为寻找文件或目录的范本样式

-user<拥有者名称> 查找符和指定的拥有者名称的文件或目录

-group<群组名称> 查找符合指定之群组名称的文件或目录

-executable \! -readable 查找有执行权限但没有可读权限的文件

-link 查找硬连接连接数等于n的文件或目录, -link 1; -link +/-1(链接数大于/小于1)

-a; -o; -not 和, 或, 非

例 :find /etc/ -path '/etc/security' -a -prune -o -name "\*.conf" -print ---求-prune的值, 若目录为真, 则 -path '/etc/security' -a -prune为1, 则不执行后续内容, 当查找目录不为/etc/security时, 查找.conf后缀的文件

例 :find /etc ! -perm /222 -exec ls -ld {} \; ---查找/etc目录下所有用户都没有写权限的文件

例 :find /etc ! -perm -111 -exec ls -l {} \; ---查找/etc目录下至少有一类用户没有执行权限的文件

例 :find /etc -perm -113 -exec ls -ld {} \; ---查找/etc目录下，所有用户都有执行权限，且其它用户有写权限的文件

## **git**

## **gitview**

## **indent**

## **[cut](http://www.runoob.com/linux/linux-comm-cut.html" \t "http://www.runoob.com/linux/_blank)**

cut:把同一行里的数据进行分解,常用于管道符(|)后面.

cut -[n]b fields:以字节为单位进行分割, 例: 3, -3, 3-, 3-5

cut -c field:以字符为单位进行分割,例:cut -c 12-: 取得12字符以后的字符串(包括第12个字符).

cut -d[分隔符] -f[域] fields: cut -d : -f 2 :以冒号为分隔符, 选取第二个域里面的内容, 输出.

缺点:如果分隔符(如: 空格)为多个, 不易处理.

## **[ln](http://www.runoob.com/linux/linux-comm-ln.html" \t "http://www.runoob.com/linux/_blank)**

ln:建立一个文件/目录的hard link/symbolic link

ln [-sf] source\_file destination\_file

-s 不加此参数,代表建立硬链接,加上代表软链接,建立硬链接,不能跨Filesystem,不能link目录

-f 如果目标文件存在,就主动将目标文件移除再建立

## **[less](http://www.runoob.com/linux/linux-comm-less.html" \t "http://www.runoob.com/linux/_blank)**

less:与more类似,并可以往前翻页

less :查看文件内容, less的功能和more相似, 但是使用more无法向前翻页, 只能向后翻, 而less可以任意切换

空格 向下翻一页

pagedown 向下翻一页

pageup 向上翻一页

? 搜索字符串

n 重复前一个搜索

N 反方向重复前一个搜索

q 离开

## **locate**

## **lsattr**

## **mattrib**

## **mc**

## **mdel**

## **mdir**

## **mktemp**

## **[more](http://www.runoob.com/linux/linux-comm-more.html" \t "http://www.runoob.com/linux/_blank)**

more :查看文件内容(内容较多的文件，一页一页的翻动), 是将文件从第一行开始，根据输出窗口的大小，适当的输出文件内容。当一页无法全部输出时，可以用“回车键”向下翻行，用“空格键”向下翻页。退出查看页面，请按“q”键。另外，more还可以配合管道符“|”(pipe)使用，例如:ls -al | more

空格 向下翻一页

回车 向下翻一页

/字符串 查找字符串

:f 立即显示文件名和行数

b 翻到第一页

q 离开

## mmove

## mread

## mren

## mtools

## mtoolstest

## **[mv](http://www.runoob.com/linux/linux-comm-mv.html" \t "http://www.runoob.com/linux/_blank)**

mv:移动文件与目录,或更名

mv -f force,如果目标文件已经存在,不会询问而直接覆盖

-i 如果目标文件已经存在,会询问是否覆盖

-u 如果目标文件已经存在,且source比较新，才会覆盖

## **[od](http://www.runoob.com/linux/linux-comm-od.html" \t "http://www.runoob.com/linux/_blank)**

od:以二进制的方式读取文件内容

od -t a 利用默认的字符来输出

-t c 使用ASCII字符来输出

-t d [size] 利用十进制输出数据,每个整数占用size bytes

-t f [size] 利用浮点数输出数据,每个整数占用size bytes

-t o [size] 利用八进制输出数据,每个整数占用size bytes

-t x [size] 利用十六进制输出数据,每个整数占用size bytes

-t oCc 将文件内容以8进位列出存储值与ASCII的对照表

## paste

## patch

patch 修补文件

diff -Naur passwd.old passwd.new>passwd.patch :制作补丁文件

安装patch软件

patch -pN<patch\_file :更新

patch -R -pN<patch\_file :还原

-p 后面可以接’取消几层目录’的意思

-R 代表还原 ,将新的文件还原成旧的版本

## rcp

## **[rm](http://www.runoob.com/linux/linux-comm-rm.html" \t "http://www.runoob.com/linux/_blank)**

rm:移除文件或目录

rm -f force,忽略不存在的文件,不会出现警告信息

-i 互动模式,在删除前会询问

-r 递归删除,目录删除

## slocate

## split

## **[tee](http://www.runoob.com/linux/linux-comm-tee.html" \t "http://www.runoob.com/linux/_blank)**

tee

tee file:输出到标准输出的同时，保存到文件file中。如果文件不存在，则创建；如果已经存在，则覆盖之。

tee -a file:输出到标准输出的同时，追加到文件file中。如果文件不存在，则创建；如果已经存在，就在末尾追加内容，而不是覆盖。

tee -:输出到标准输出两次。

tee file1 file2 -:输出到标准输出两次，同时保存到file1和file2中。

## tmpwatch

## **[touch](http://www.runoob.com/linux/linux-comm-touch.html" \t "http://www.runoob.com/linux/_blank)**

touch:修改文件时间或建置新文档,不能修改ctime

touch -a 仅修改atime

-c 仅修改文件时间,若文件不存在则不建立

-d 欲修订的日期 例:touch -d "2 days ago" a.txt

-m 仅修改mtime

-t 欲修订的日期,YYYYMMDDhhmm

## **[umask](http://www.runoob.com/linux/linux-comm-umask.html" \t "http://www.runoob.com/linux/_blank)**

umask:目前用户在建立文件或目录时的权限默认值

umask 数字形态权限设定分数(如:0022,后三位与权限有关,表示该默认值需要减掉的分数) 若使用者建立为'文件',预设没有可执行权限

-S 字符形态权限

xyz 更新数字形态权限设定分数 例:umask 002

## **[which](http://www.runoob.com/linux/linux-comm-which.html" \t "http://www.runoob.com/linux/_blank)**

which:查找文件

例 :which bash --查看指令"bash"的绝对路径

## **[cp](http://www.runoob.com/linux/linux-comm-cp.html" \t "http://www.runoob.com/linux/_blank)**

cp:复制文件 复制后文件own/grp不变

cp -a 相当于-pdr

-i 若目标文件已经存在,在覆盖时会先询问动作的进行

-p 连同文件的属性(权限、用户、时间)一起复制过去，而非使用默认属性(备份常用)

-r 递归持续复制,目录的复制

-l 进行硬式连接文件的建立

-s 复制成符号链接文件.

-u destination比source旧才更新destination,或destination不存在的时候

-d 或源文件为链接文件,则复制链接文件属性而非文件本身

## **[whereis](http://www.runoob.com/linux/linux-comm-whereis.html" \t "http://www.runoob.com/linux/_blank)**

## mcopy

## mshowfat

## rhmask

## scp

## **[awk](http://www.runoob.com/linux/linux-comm-awk.html" \t "http://www.runoob.com/linux/_blank)**

(1)awk内置变量(预定义变量)

$0/$n :第n个字段的内容

FILENAME :当前输入文件的名.

FS :字段分隔符(默认是任何空格), 可以是正则表达式. 例 :awk 'BEGIN {FS="[:/]"}’ ---分隔符为: 或/

例s :awk ‘BEGIN{FS=”\n”;OFS=”\n”;RS=””;ORS=”\n\n”}’ ---换行符 awk -F[:/] '{print NF “:” $0}' list

NF :字段数 例 :print $NF, $(NF-1), NF

NR :记录数, 当前的行号.

CONVFMT :数字的输出格式(默认值是%.6g, 代表保留小数点后6位, POSIX之前, OFMT实现同样的功能).

当没有小数的时候, 会输出整个整数的内容.

例 :awk ‘BEGIN{a=10/3; CONVFMT=”%.2g”; printf a}’ ---3.3 ---为什么只能是printf?

例 :awk 'BEGIN{var1=1.1234567; OFMT="%.3f";print var1}' ---0.123 , 3代表小数点后还有3位

例 :awk ‘BEGIN{var=1.1234567;OFMT=”%.3g”;print var}’ ---1.12, 3代表一共有3个数字

例 :awk ‘BEGIN{var=12345;OFMT=”%.2g”;print var}’ --12345

OFS :输出字段分隔符(默认值是一个空格)

ORS :输出记录分隔符(默认值是一个换行符)

RS :记录分隔符(默认是一个换行符)

(2)awk匹配正则表达式

awk 可匹配拓展的正则表达式,

当想要匹配^和$, 不管它处于正则表达式的什么位置都要对其进行转义.

在[]中, \具有特殊意义, 且只适用于awk, 不适用于sed和grep.

在awk和egrep中, 使用{}不需要进行转义, 在grep 和sed中, 则需要进行转义 :\{\}

例 :awk 'BEGIN{FS=”:”}$1~/str1/{print}' filename ---不匹配~!

(3)awk进行变量类型转化

awk只有两种类型的变量 :数字和字符串

字符串转数字 :string + 0

数字转字符串 :number ""

(4)awk格式打印

printf没有指定自动换行功能, 需手动加上\n

printf语法 :printf(format\_expression, argument1, argument2..)

例 :printf(“%-10s\n”, string) ---左对齐 printf(“%\*.\*g\n”, 8, 3, string) ---宽度最小为8, 精度是3

(5)awk向脚本传递参数

awk ‘script’ var=value list, 这种变量在BEGIN的action不能获得

例 :awk -f script\_file high=100 low=60 list

awk -v var=value ‘script’ list

例 :awk -v high=100 -v low=$string ‘script’ list

(6)awk条件语句与循环

if条件语句 :if(expression1) {statement1;statement2} else if(expression2) statement3; else statement4

条件操作符?: expression?action1:action2

while循环 :while(condition){statement1;statement2}

do循环 :do{statement1;statement2} while(condition)

for循环 :for(set\_counter;test\_counter;increment\_counter){statement1;statement2}

continue :终止当前循环, 从循环顶部开始一个新的循环

break :退出循环

next :读入下一个输入行, 并返回到脚本的顶部. 这可以避免对当前输入行执行其他的操作过程.

例 :awk ‘FILENAME=”acronyms”{action;next}{print}’ list --如文件名为acronyms, 将不会执行print.

exit :使主输入循环退出并将控制转移到END, 如果END存在的话. 如果没有定义END规则, 或在END中应用exit语句, 则终止脚本的执行, 如果为exit语句设置一个初值, 然后END中再次调用没有参数的exit, 则使用第一个值.

(7)数组

awk数组都是关联数组, 下标可以是一个字符或一个数值, 下标都是字符串类型, awk数组是关联数组, 只要通过数组引用它的key，就会自动创建改序列。 例 :Array[1], Array["f"]

遍历 :for (i in arr){print i, arr[i]}

测试是否数组成员 :if(i in arr){statement1}else{statement2}

使用split创建数组 :z=split( string, array, separator ), string是即将被分解的字符串, array是分解之后得到的数组, 数组下标从1到n, 根据separator分隔符来分解, 若没有指定分隔符, 则使用FS, z是数组的长度.

删除数组元素 :delete array[item]

(8)多维数组

awk不支持多维数组, 但可模拟引用多维数组, 并不能创建多维数组, 下标的分隔符默认为”\034”(SUBSE P), 模拟多维数组的缺点是数组越大, 访问个别元素的速度就越慢.

测试是否数组成员 :if((i,j) in arr)

遍历数组 :for (item in arr)

访问单独的下标分量 :for (item in arr){split(item,arr2,”\034”);print arr2[1],arr2[2],arr[item]}

例 :awk 'BEGIN{for(i=1;i<=9;i++){for(j=1;j<=9;j++){arr[i,j]=i\*j; print i,"\*",j,"=",arr[i,j];}}}'

(9)awk系统变量数组

ARGV :命令行参数数组, 不包括脚本本身和任何调用awk指定的选项, 这个数组中的元素的个数可以从ARGC中获得, 数组元素下标以0开始(和awk中其他数组不同, 但和C一致), 最后一个下标是ARGC-1.

例 :awk ‘BEGIN{for(i=0;i<ARGC;i++){printf ARGV[i] “ ”}print ARGC}’ list ---awk list 2

ENVIRON :环境变量数组, 元素值是当前环境变量的值, 下标是环境变量的名字, 可以修改数组当中的任意元素, 但是并未改变用户的真实环境, 也未改变程序环境, 这些程序是awk使用getline()或system()调用的.

例 :awk ‘BEGIN{for(item in ENVIRON){print item “ : ”ENVIRON[item] }}’

(10)算数函数

atan2(y, x) :y/x的反正切 atan2(0,-1) ---结果为π的值

cos(x) :x的余弦(x是弧度)

sin(x) :x的正弦(x是弧度)

exp(x) :e的x幂 exp(1) ---返回的是e的值

log(x) :x的自然对数(以e为底)

sqrt(x) :x平方根

int(x) :x的整数部分的值, 不会遵循四舍五入的规则, 想要遵循四舍五入的规则可用 :printf(“%.0f\n”,a)

rand() :随机数n, 0<=n<1

srand(x) :建立rand()的新的种子数, 如果没有指定种子数, 就用当时的时间来生成一个种子数, 返回旧的种子值.

例 :awk 'BEGIN{srand();fr=int(100\*rand());print fr;}' ---保存为shell文件, 执行.

例 :awk -v number=$1-v top\_number=$2‘BEGIN{srand();if(number<=0){number=6}if (top\_number <=0){top\_number=30} for(i=1;i<=number;i++){do{select=int(rand()\*top\_number)+1} while(select in arr)arr[select]=select;print select}}’ | sort -n | awk ‘BEGIN{ORS=” ”}{print}’ ---pick number of top\_number

(11)字符串函数

gsub(re,replace,string ) :在字符串string中用字符串replace替换re的所有匹配, 并返回替换的个数, 如果没有给出string, 则默认为$0.(g :global)

例 :gsub(/UNIX/,”\\fB&\\fR”)

例 :awk 'BEGIN{info="this is a test2010test!";gsub(/[0-9]+/,"!",info);print info}' ---this is a test!test!

sub(re,replace,string ) :在字符串string中用字符串replace替换re的首次匹配, 成功返回1, 否则返回0,如果没有给出string, 则默认为$0.

index(string,sub\_string) :返回子串sub\_string在string中的位置, 由1开始编号, 如果sub\_string不在string中出现, 则返回0.

例 :awk ‘BEGIN{lower=”abcdefghijklmnopqrstuvwxyz”;upper=”ABCDEFGHIJKLMNOPQRSTUVWXYZ”} {index\_1=index(lower,substr($1,1,1));if(index\_1!=0){$1=substr(upper,index\_1,1)substr($1,2)}print $0}’

例 :position=index(“Mississippi”,”is”) ---返回值为2, 如果没有发现子串, 则返回0.

例 :awk 'BEGIN{info="this is a test2010test!";print index(info,"test")?"okay":"no found";}'

length(string) :返回string的长度(字符形式). 如果未给出string, 则返回$0的长度.

blength(string) :返回string的长度(以字节为单位), 如果未给出string, 则返回$0的长度.

substr(string,M,N) :返回string从M开始长度为N的子串, 如果未指定N， 返回M之后的字符串.

match(string,re) :如果re在string中出现, 返回起始位置(字符形式), 从1开始编号, 如果未出现, 返回0. match函数设置了两个系统变量RSTART和RLENGTH, RSTART为函数的返回值, 即匹配子串的开始位置, RLENGTH为匹配的字符串的长度. 当模式不匹配时, RSTART为0, RLENGTH为-1.

例 :awk ‘{if(match($0,re)){print substr($0,RSTART,RLENGTH)}}’ re=”$1” $2 ---提取匹配的子串

例 :if(match($1,/:[^:]\*:/)){before=substr($1,1,RSTART+RLENGTH-2);after=substr($1,RSTART+RLE

NGTH);$1=before";"after} ---用分号取代第二个冒号

split(string,arr,separator) :使用separator将string分解为arr的元素, 并返回元素个数, 未指定separator时, 使用FS.

tolower(string) :将string中所有的大写字符转变为小写字符, 并返回新字符, 大写和小写的映射由当前语言环境的LC\_CTYPE范畴定义.

toupper(string) :将string中所有的小写字符转变为大写字符, 并返回新字符.

例 :toupper($0)

spr intf(“format”,expression):对expression使用printf格式说明.

例 :awk 'BEGIN{for(i=97;i<=122;i++){arr[i]=sprintf("%c",i)}}'---产生a-z的ASCII字符, 赋值给arr.

(12)自定义函数

语法 :function function\_name(parameter\_list){statement1;statement2}

在函数中定义的变量为全局变量, 参数列表中的变量为局部变量.

例 :awk 'function sort(arr,NUM) { ---一个排序的函数, 亦可进行字符串排序.

for(j=NUM;j>=1;j--){

for(i=1;i<j;i++){

if(arr[i]>arr[i+1]){

c=arr[i+1];arr[i+1]=arr[i];arr[i]=c

}}} return}

BEGIN{a[1]=10;a[2]=8;a[3]=1;a[4]=66;a[5]=3;sort(a,5);for(i=1;i<=5;i++){print a[i]}}'

(13)高级输入输出

getline :从输入中读取另一行, 可以读取输入流和文件管道的输入, getline读取下一行但没有改变脚本的控制, 如果得到一条记录, 返回1, 如果到达文件末尾, 返回0, 如果出现错误, 返回-1.

getline从文件中读取 :getline <”data”

例 :awk ‘BEGIN{while((getline<”list”)>0){print}}’ --从文件中读取并打印

getline将输入赋值给一个变量 :getline input, 不会改变$0的值, 不会改变NF的值, 但会改变NR.

例 :awk ‘BEGIN{printf “Please enter your name:” ; getline<”-”;print “Hi ”$0}’ --用户的响应被赋给$0.

例 :awk ‘BEGIN{printf “Please enter your name:” ; getline name <”-” ; print “Hi ”name}’

getline从管道读取输入 :”who am i” | getline me, 执行一个命令并将输出结果用管道送到getline, 当一个命令的输出结果被管道输送给getline并包含多个行时, getline一次读取一行, 需创建循环以输出所有的行.

例 :awk ‘/^\.SH “?Name”?/{getline;print $1}’ list --打印.SH Name行的下一行的第一个字段, 并且新行赋值给$0, 打印的是下一行的$1.

例 :awk ‘BEGIN{“who am i” | getline ; name=$1;FS=”:”}$1~name{print $5}’ /etc/passwd

例 :awk ‘BEGIN{while(“ls -l” | getline){ls\_out[++i]=$0;print ls\_out[i]}}’

close() :关闭打开的文件和管道, 通常是当getline返回0或-1时, 关闭一个用过的管道, 例 :close(“who”).

例 :awk ‘BEGIN{FS=”:”}{print $7 | “sort > temp\_file”}END{close(“sort>temp\_file”);

while((getline<”temp\_file”)>0){print $0}}’ /etc/passwd

system() :执行一个以表达式给出的命令, 返回被执行的命令的退出状态, 退出状态为0是成功.

例 :awk ‘BEGIN{if (system(“mkdir dale”)!=0){print “Command failed”}}’ --当执行第二次时报错

例 :awk ‘BEGIN{FS=”:” ---基于命令的菜单生成器

if((getline<”test12”)>0) --- test12第一行是菜单标题, 后面的行包括:$1:描述, $2:命令

title=$0

while((getline<”test12”)>0){

++array\_size;menu[array\_size]=$1;command[array\_size]=$2

}

display\_menu()

}

function display\_menu(){

print title; for (i=1;i<=array\_size;i++){

printf(“\t%d. %s\n”,i,menu[i])

}

printf(\t%d. Exit\n,array\_size);printf “Choose one:”

}

{

if($1>0&&$1<=array\_size){

printf(“Executing... %s\n”,command[$1]);system(command[$1]);

printf “<Press ENTER to continue:>”;getline ---如果没有getline, 将会直接display\_menu()

}

else

exit

display\_menu()

}’ -

向文件及管道输出 :>, |.

例awk 'BEGIN{printf("hello world") >> "file.out"}'; awk 'BEGIN{printf("hello world") | wc -l}'

1. 数组相关函数

得到数组长度：

awk 'BEGIN{info="it is a test";lens=split(info,tA," ");print length(tA),lens;}' ---4 4

length返回字符串以及数组长度，split进行分割字符串为数组，也会返回分割得到数组长度。

awk 'BEGIN{info="it is a test";split(info,tA," ");print asort(tA);}' ---4

asort对数组进行排序，返回数组长度。

awk高级输入输出

1. 读取下一条记录

awk中next语句使用：在循环逐行匹配，如果遇到next，就会跳过当前行，直接忽略下面语句。而进行下一行匹配。next语句一般用于多行合并.

例：awk '/^web/{T=$0;next;}{print T":t"$0;}' test.txt 将包含有“web”行进行跳过，然后需要将内容与下面行合并为一行

1. 简单地读取一条记录
2. awk getline用法：输出重定向需用到getline函数。getline从标准输入、管道或者当前正在处理的文件之外的其他输入文件获得输入。它负责从输入获得下一行的内容，并给NF,NR和FNR等内建变量赋值。如果得到一条记录，getline函数返回1，如果到达文件的末尾就返回0，如果出现错误，例如打开文件失败，就返回-1。
3. getline语法：getline var，变量var包含了特定行的内容。

当其左右无重定向符|或<时：getline作用于当前文件，读入当前文件的第一行给其后跟的变量var或$0（无变量），应该注意到，由于awk在处理getline之前已经读入了一行，所以getline得到的返回结果是隔行的。

当其左右有重定向符|或<时：getline则作用于定向输入文件，由于该文件是刚打开，并没有被awk读入一行，只是getline读入，那么getline返回的是该文件的第一行，而不是隔行。

例：awk 'BEGIN{ "date" | getline out; print out }' test

awk 'BEGIN{ "date" | getline out; split(out,mon); print mon[2] }' test

awk 'BEGIN{ while( "ls" | getline) print }'

1. 关闭文件

awk中允许在程序中关闭一个输入或输出文件，方法是使用awk的close语句。

close("filename")

filename可以是getline打开的文件，也可以是stdin，包含文件名的变量或者getline使用的确切命令。或一个输出文件，可以是stdout，包含文件名的变量或使用管道的确切命令。

(4) 输出到一个文件

awk中允许用如下方式将结果输出到一个文件：

echo | awk '{printf("hello word!n") > "datafile"}'

或

echo | awk '{printf("hello word!n") >> "datafile"}'

awk内置函数，主要分以下3种类似：算数函数、字符串函数、其它一般函数、时间函数。

(1) 算术函数

(2) 字符串函数

(3) 一般函数

close( Expression ) 用同一个带字符串值的 Expression 参数来关闭由 print 或 printf 语句打开的或调用 getline 函数打开的文件或管道。如果文件或管道成功关闭，则返回 0；其它情况下返回非零值。如果打算写一个文件，并稍后在同一个程序中读取文件，则 close 语句是必需的。

system(command ) 执行 Command 参数指定的命令，并返回退出状态。等同于 system 子例程。

Expression | getline [ Variable ] 从来自 Expression 参数指定的命令的输出中通过管道传送的流中读取一个输入记录，并将该记录的值指定给 Variable 参数指定的变量。如果当前未打开将 Expression 参数的值作为其命令名称的流，则创建流。创建的流等同于调用 popen 子例程，此时 Command 参数取 Expression 参数的值且 Mode 参数设置为一个是 r 的值。只要流保留打开且 Expression 参数求得同一个字符串，则对 getline 函数的每次后续调用读取另一个记录。如果未指定 Variable 参数，则 $0 记录变量和 NF 特殊变量设置为从流读取的记录。

getline [ Variable ] < Expression 从 Expression 参数指定的文件读取输入的下一个记录，并将 Variable 参数指定的变量设置为该记录的值。只要流保留打开且 Expression 参数对同一个字符串求值，则对 getline 函数的每次后续调用读取另一个记录。如果未指定 Variable 参数，则 $0 记录变量和 NF 特殊变量设置为从流读取的记录。

getline [ Variable ] 将 Variable 参数指定的变量设置为从当前输入文件读取的下一个输入记录。如果未指定 Variable 参数，则 $0 记录变量设置为该记录的值，还将设置 NF、NR 和 FNR 特殊变量。

打开外部文件（close用法）

awk 'BEGIN{while("cat /etc/passwd"|getline){print $0;};close("/etc/passwd");}'

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

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

逐行读取外部文件(getline使用方法）

awk 'BEGIN{while(getline < "/etc/passwd"){print $0;};close("/etc/passwd");}'

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

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

awk 'BEGIN{print "Enter your name:";getline name;print name;}'

Enter your name:

chengmo

chengmo

调用外部应用程序(system使用方法）

awk 'BEGIN{b=system("ls -al");print b;}'

total 42092

drwxr-xr-x 14 chengmo chengmo 4096 09-30 17:47 .

drwxr-xr-x 95 root root 4096 10-08 14:01 ..

b返回值，是执行结果。

(4) 时间函数

mktime( YYYY MM dd HH MM ss[ DST]) 生成时间格式

strftime([format [, timestamp]]) 格式化时间输出，将时间戳转为时间字符串

具体格式，见下表.

systime() 得到时间戳,返回从1970年1月1日开始到当前时间(不计闰年)的整秒数

建指定时间(mktime使用）

awk 'BEGIN{tstamp=mktime("2001 01 01 12 12 12");print strftime("%c",tstamp);}'

2001年01月01日 星期一 12时12分12秒

awk 'BEGIN{tstamp1=mktime("2001 01 01 12 12 12");tstamp2=mktime("2001 02 01 0 0 0");print tstamp2-tstamp1;}'

2634468

求2个时间段中间时间差，介绍了strftime使用方法

awk 'BEGIN{tstamp1=mktime("2001 01 01 12 12 12");tstamp2=systime();print tstamp2-tstamp1;}'

308201392

strftime日期和时间格式说明符

格式 描述

%a 星期几的缩写(Sun)

%A 星期几的完整写法(Sunday)

%b 月名的缩写(Oct)

%B 月名的完整写法(October)

%c 本地日期和时间

%d 十进制日期

%D 日期 08/20/99

%e 日期，如果只有一位会补上一个空格

%H 用十进制表示24小时格式的小时

%I 用十进制表示12小时格式的小时

%j 从1月1日起一年中的第几天

%m 十进制表示的月份

%M 十进制表示的分钟

%p 12小时表示法(AM/PM)

%S 十进制表示的秒

%U 十进制表示的一年中的第几个星期(星期天作为一个星期的开始)

%w 十进制表示的星期几(星期天是0)

%W 十进制表示的一年中的第几个星期(星期一作为一个星期的开始)

%x 重新设置本地日期(08/20/99)

%X 重新设置本地时间(12：00：00)

%y 两位数字表示的年(99)

%Y 当前月份

%Z 时区(PDT)

%% 百分号(%)

## **[read](http://www.runoob.com/linux/linux-comm-read.html" \t "http://www.runoob.com/linux/_blank)**

read:读取来自键盘输入的变量

read [-pt] variable -p 提示字符

-t 等待秒数

## **head**

head :取出前面几行

head -n k k默认为10,k为负数时表示不打印最后n行, head -n k=head -n +k

-k 取出前面k行

例 :head -n 5 /etc/passwd ; ---打印前面5行

例 :head -5 /etc/passwd; ---打印前面5行

例 :head -n -5 /etc/passwd ---不打印最后5行

## **tail**

tail :取出文件最后几行

tail -n k k默认为10, tail -n k=tail -n -k

-n +k 第k行开始, 输出所有行

-k 输出最后k行

例 : tail -n 5 /etc/passwd ; ---打印最后5行

例 :tail -5 /etc/passwd; ---打印最后5行

例 :tail -n -5 /etc/passwd; ---打印最后5行

例 :tail -n +5 /etc/passwd; ---不打印前面5行

## + xargs

## +rename

rename :用字符串替换的方式批量改变文件名。

语法 :rename substring replace file; ---file支持通配符?和\*和[]

语法 :rename “s/substring/replace/” file ---支持正则表达式

例 :rename main1.c main.c main1.c ---将main1.c重命名为main.c

例 : rename foo foo0 foo? ---foo1到foo9的文件重命名为foo01到foo09, 因为foo?代表四个字符串的文件

例 :rename foo foo0 foo?? ---foo01到foo99的所有文件都被重命名为foo001到foo099

例 :rename foo foo0 foo\* ---所有以foo开头的文件都被重命名

例 :rename foo0 foo foo0[2]\* ---从foo0200到foo0299的所有文件都被重命名为foo200到foo299

例 :rename "s/AA/aa/" \* ---把文件名中的AA替换成aa

例 :rename "s//.html//.php/" \* ---把.html 后缀的改成 .php后缀

例 :rename "s/$//.txt/" \* ---把所有的文件名都以txt结尾

例 :rename "s//.txt//" \* ---把所有以.txt结尾的文件名的.txt删掉

## + test

test :检查某个条件是否成立, 它可以进行数值、字符和文件三个方面的测试, 与[ expression ] 等价,

[[ ]]为扩展的test命令, 可以使用通配符进行模式匹配，如&&，||，>, <等操作符可以应用于[[ ]]中

语法 :判断表达式if test ---表达式为真

if test ! ---表达式为假

test expression1 –a expression2 ---两个表达式都为真

test expression1 –o expression2 ---两个表达式有一个为真

test expression1 ! expression2 ---条件求反

语法 :判断字符串test –n string #字符串的长度非零

test –z string #字符串的长度是否为零

test string1＝string2 #字符串是否相等，若相等返回true

test string1!＝string2 #字符串是否不等，若不等返回false

语法 :判断整数 test int1 -eq int2 #整数相等

test int1 -ge int2 #整数1大于等于整数2

test int1 -gt int2 #整数1大于整数2

test int1 -le int 2 #整数1小于等于整数2

test int1 -lt int 2 #整数1小于整数2

test int1 -ne int2 #整数1不等于整数2

语法 :判断文件 test File1 –ef File2 两个文件是否为同一个文件，可用于硬连接。主要判断两个文件是否指向同一个inode。

test File1 –nt File2 判断文件1是否比文件2新

test File1 –ot File2 判断文件1比是否文件2旧

test –b file #文件是否块设备文件

test –c File #文件并且是字符设备文件

test –d File #文件并且是目录

test –e File #文件是否存在 （常用）

test –f File #文件是否为正规文件 （常用）

test –g File #文件是否是设置了组id

test –G File #文件属于的有效组ID

test –h File #文件是否是一个符号链接（同-L）

test –k File #文件是否设置了Sticky bit位

test –b File #文件存在并且是块设备文件

test –L File #文件是否是一个符号链接（同-h）

test –o File #文件的属于有效用户ID

-O file :如果文件存在并且归该用户所有，则为真；

test –p File #文件是一个命名管道

test –r File #文件是否可读

test –s File #文件是否是非空白文件

-S file :如果文件为一个套接字特殊文件，则为真；

test –t FD #文件描述符是在一个终端打开的

test –u File #文件存在并且设置了它的set-user-id位

test –w File #文件是否存在并可写

test –x File #文件是否存在并可执行

# 文档编辑

## col

## colrm

## comm

## csplit

## ed

## egrep

egrep :等同于grep -E,支持延伸正则表达式

在awk和egrep中, 使用{}不需要进行转义, 在grep 和sed中, 则需要进行转义 :\{\}

## ex

## fgrep

fgrep: 等同于 grep -F或--fixed-regexp 将范本样式视为固定字符串的列表, 其中正则表达式的.\*$等不能转义.

## fmt

## fold

## **[grep](http://www.runoob.com/linux/linux-comm-grep.html" \t "http://www.runoob.com/linux/_blank)**

grep:global search regular expression(RE) and print out the line, 全面搜索正则表达式并把行打印出来.

在awk和egrep中, 使用{}不需要进行转义, 在grep 和sed中, 则需要进行转义 :\{\}

grep [-acinvABC] ‘word’ filename/stdin ：分析一行讯息,若当中有需要的信息，就将该行拿出来，

针对的是行的筛选。

-a :不要忽略二进制数据。

-c ：计算找到搜寻字符串的行数 例:grep -c "text" file\_name

-E :将范本样式为延伸的普通表示法来使用，意味着使用能使用扩展正则表达式。

-i ：忽略大小写

-n ：在输出符合要求的行的同时连同行号一起输出 例:grep "text" -n file\_name

-v ：打印不符合要求的行

-A ：后跟一个数字（有无空格都可以），例如 –A2则表示打印符合要求的行以及下面两行

-B ：后跟一个数字，例如 –B2 则表示打印符合要求的行以及上面两行

-C ：后跟一个数字，例如 –C2 则表示打印符合要求的行以及上下各两行

-w或--word-regexp 只显示全字符合的列, 将w后的看作是一个单词来进行匹配,

-q 不显示任何信息。

-s 不显示不存在或无匹配文本的错误信息.?

-l --files-with-matches print only names of FILEs containing matches eg.grep -l ‘abc’ file1 file2

-r 在当前目录及其子目录下搜索

--include=FILE\_PATTERN search only files that match FILE\_PATTERN

例 :grep -w -E ‘books?’ list 等同于 grep -E ‘\<books?\>’ list :(ex和vi文件编辑器使用特殊的元字符\<和\>, 分别匹配单词开始处和结束处的字符串)

## ispell

## jed

## joe

## join

## look

## mtype

## pico

## rgrep

rgrep :递归查找文件里符合条件的字符串

## sed

sed 指令包括两个部分 :模式和过程. 模式是由/分割的正则表达式, 过程指定一个多多个将被执行的命令.

Tips: 在sed和grep中, 只有当^和$出现在正则表达式的开始或结尾时才是特殊的.

Tips: 在awk和egrep中, 使用{}不需要进行转义, 在grep 和sed中, 则需要进行转义 :\{\}

Tips: sed显示在屏幕上的结果是临时的, 输入文件中并没有发生改变, 只有将sed的输出重定向到另一个程序中, 才能够捕获文件中的输出. sed ‘s/a/b/g’ list >new\_list

Tips: sed可以使用大括号{}将一个一个地址嵌套在另一个地址中, 或者在相同的地址上应用多个命令.

sed -n 阻止自动输出每个输入行, 当指定该选项时, 每个要生成输出的指令都必须包含打印命令p, sed -n ‘s/a/b/gp’ list

sed -e 命令行有多个指令, sed -e ‘s/a/b/g’ -e ‘s/c/d/g ‘ list

sed -f 脚本文件 :sed -f scriptfile file

sed -r 表示支持延伸正则表达式

sed -i

sed ‘/line-address/a\

text’ 追加, 新的字串会在目前的下一行出现

sed ‘/line-address/i\

text’ 插入, 新的字串会在目前的上一行出现

sed ‘/address/c\

text’ 更改, 取代address之间的行, 以行为单位.

sed ‘/pattern/,/pattern2/d’ 删除

sed -n ‘n1,n2p’ 打印，亦即将某个选择的数据印出。通常 p 会与参数 sed -n 一起运行～

sed ‘[address]y/ab/he/’ 转换, 按位置将字符串ab中的每个字符, 都转换成字符串he中的等价字符.

sed ‘[address]s/pattern/replacement/flags’ 替换

replacement部分, 下列字符有特殊含义: & :用正则表达式所匹配的内容进行替换;

\n :匹配第n个字串, 在pattern中用\(\)指定;

\ :转义

flags 是 :g :对模式空间的所有出现的情况进行全局更改, 而没有g通常是第一次出现的情况被取代.

p :打印模式空间的内容

n :1到512之间的1个数字, 表示对文本模式中指定模式第n次出现的情况进行替换.

w file :将模式空间的内容写到file中

sed -n -e ‘l’ list :列表命令(l)用于显示模式空间的内容, 将非打印的字符显示为两个数字的ASCII代码.

sed ‘=’ list或sed -n ’=;p’ list :打印所有行行号, 会分两行显示, 第一行输出行号,第二行显示内容.

sed ‘[address] {=; p}’ list 或sed -n ‘[address]=; [address]p’ list :打印指定行行号及内容.

sed ‘[address]{n; [address2]d}’ list :下一步next, 在address下一行, 如果匹配address2, 则删除.

sed ‘[line-address]r file’ list :将文件file的内容附加到文件list指定的位置line-address.

sed ‘[address]w file’ list :将address 指定的内容写到file中, 若file不存在, 则新建, 若存在, 则覆写.

sed ‘[line-address]q’ list :退出命令q会使sed停止读取新的输入行, 并停止将它们发送到输出. 另一个用法是在文件中提取了想要的内容后退出脚本(因为sed 找到了需要寻找的东西之后会继续扫描庞大的文件).

sed ‘[address]{w file, [line-address]i\(换行)text(换行)d}’ list :将[address]里面的内容写到file里面, 并将address的内容更改为text, 最后删除.

sed ‘[line-address]{$!N(换行) s/ \*\n/ /(换行)s/string1/string2}’ list :

sed ‘/^ \*$/{N(换行)/^ \*\n$/D}’ list :删除多个空行为1行, 删除命令d清除的是整个模式空间, D代表删除模式空间的第一行.

sed -n ‘/(/,/)/p’ list :可以匹配跨行的括号并将相应的行打印出来

sed ‘s/@f1(\([^)]\*\))/---\1---/g;/@f1(/{N;s/@f1(\([^)]\*\n[\)]\*\))/---\1---/g;P;N}’ list :将@f1(text)更改为---text---.

P命令输出多行模式空间的第一部分, 直到第一个嵌入的换行符.

sed保持空间 :Hold/h/H :将模式空间的内容复制或追加到保持空间(在保持空间的内容之后放置一个换行符, 且后面跟随模式空间的内容)

Get/g/G :将保持空间的内容复制或追加到模式空间(同有换行符)

Exchange/x :交换模式空间和保持空间的内容.

sed ‘/^1/{h;d};/^2/G’ list

sed ‘${/^ \*$/!{H;s/.\*//}};/^ \*$/!{H;d};/^ \*$/{x;s/^\n/<p>/;s/$/<\\p>/;G}’ list

[address]b[label] :如果没有指令label, 控制就被转到脚本的结尾处, 如果有就执行标签(:label)后面的行.

[address]t[label] :如果没有指令label, 控制就被转到脚本的结尾处, 如果有就执行标签(:label)后面的行.

指定命令行上的多重指令的三种方式:

1. 用分号分隔 : sed ‘s/a/b/g; s/c/d/g’ list
2. 每个指令前放置-e : sed -e ‘s/a/b/g’ -e ‘s/c/d/g’ list
3. 使用bourne shell的分行指令功能, 在输入’后按enter, 就会出现多行输入的提示符(>) : (C shell不可用)

sed ‘s/a/b/g

>s/c/d/g’ list

例 :数据的搜寻并执行指令nl /etc/passwd | sed -n '/bash/{s/bash/blueshell/;p;q}'

例 :sed ‘s/a/b/’ 与sed ‘s/a/b/g’的区别 :前者只影响当前行上的第一次出现, 若要寻找同一行上的多次出现, 则需要加上g(global).

例 :sed -r -n ‘s/(\<books?\>)/\1ssss/gp’ list

例 :sed ‘/^\.TS/,/^\.TE/{

/^$/d

s/^\.ps10/.ps8/

}’ ---左大括号必须要在行末, 右大括号单独占一行, 且之后不能有空格.

例 :sed -n “/^\.de$1/,/^\.\.$/p” list ----如果使用单引号, shell不对$1进行解释.

例 :sed ‘/books/! s/book/hello/g’ list --!不能连续用, 连续否定 :sed -r ‘/book(s|worm)/! s/book/hello/g’ list

例 : ---会打印出第1,1+5,1+10...行

例 :sed -n ‘5,+5p’ list ---打印出5到5+5行

例 :sed -n ‘5,~3p’ list ---打印出5到下一个3的倍数(6)行.

例 :sed ‘/^12/s//ab/’ list ---会匹配以12开始的行, 并将12替换为ab.

## sort

sort :将文本文件内容加以排序

sort [-bcdfimMnr][-o<输出文件>][-t<分隔字符>][+<起始栏位>-<结束栏位>] [文件]

sort -b：忽略每行前面开始出的空格字符；

-c：检查文件是否已经按照顺序排序；

-d：排序时，处理英文字母、数字及空格字符外，忽略其他的字符；

-f：排序时，将小写字母视为大写字母；

-i：排序时，除了040至176之间的ASCII字符外，忽略其他的字符；

-m：将几个排序号的文件进行合并；

-M：将前面3个字母依照月份的缩写进行排序；

-n：依照数值的大小排序；

-o 输出文件 :将排序后的结果存入制定的文件；

-r：以相反的顺序来排序；

-t分隔字符 :指定排序时所用的栏位分隔字符; By default, fields are separated by the empty string between a non-blank character and a blank character.默认的分隔符为空白字符和非空白字符之间的空字符.

+<起始栏位>-<结束栏位>：以指定的栏位来排序，范围由起始栏位到结束栏位的前一栏位。

-u :忽略相同行, sort | uniq 与之作用相同, 但sort -n -u :只会检查排序字段数值部分的唯一性, sort -n | uniq :在sort对行中字段按数值排序后，uniq将检查整个行的唯一性。

-k :KEYDEF is F[.C][OPTS][,F[.C][OPTS]] for start and stop position, where F is a field number and C a character position in the field; both are origin 1, and the stop position defaults to the line's end. If neither -t nor -b is in effect, characters in a field are counted from the beginning of the preceding whitespace. OPTS is one or more single-letter ordering options [bdfgiMhnRrV],which override global ordering options for that key. If no key is given, use the entire line as the key. Use --debug to diagnose incorrect key usage.

例 :sort -u sort.txt

例 :sort -nk 2 -t: sort.txt

例 :sort -nrk 3 -t: sort.txt

例 :sort -t ' ' -k 1.2,1.2 -nrk 3,3 facebook.txt :对第一列的第二个字母排序 ,如果相同就按照第三列进行降序排序

例 :echo -e " 234 bar\n 123 car" | sort -t ' ' -b -k3 与echo -e " 234 bar\n 123 car" | sort -b -k2结果一致, 与 echo -e " 234 bar\n 123 car" | sort -b -k3结果不一致.

## spell

## tr

tr :对来自标准输入的字符进行替换、压缩和删除。它可以将一组字符变成另一组字符，经常用来编写优美的单行命令，作用很强大。

tr -c或--complement :反选设定字符. 也就是符合 SET1 的部份不做处理, 不符合的剩余部份才进行转换

-d或--delete :删除指令字符

-s或--squeeze-repeats :缩减连续重复的字符成指定的单个字符

-t或--truncate-set1 :削减 SET1 指定范围，使之与 SET2 设定长度相等

例 :cat text | tr '\t' ' '

例 :echo aa.,a 1 b#$bb 2 c\*/cc 3 ddd 4 | tr -d -c '0-9 \n' :补集中包含了数字0~9、空格和换行符\n，所以没有被删除，其他字符全部被删除了。

例 :echo "thissss is a text linnnnnnne." | tr -s ' sn' :将多个空格/s/n视为1个空格/s/n

## expr

expr :一个手工命令行计数器, 用于在UNIX/LINUX下求表达式变量的值, 一般用于整数值, 也可用于字符串.

expr length 例：expr length $str; expr "$str" : ".\*"

substr 例：expr substr "$str" 5 3 (第一位为1，最后一位为${#var}, 不可为空)

index 例：expr index "$str" "db"(会根据d或b最先出现的位置计算index)

整数运算 \\*,\>,\<,\|,\& 例:expr 5 + 3(注:用空格隔开每个项)

例 :expr $[2+3]; 等价于expr 2 + 3

例 :f=”text.pub”; expr “$f” : “.\*\.pub”---检查某个文件是否是某一类型或扩展名, 若是，则返回f的长度

例 :expr $i +0; echo $? ---判断一个变量是否是整数

例 :expr $a + 0 &>/dev/null; [$? -eq 0] && echo int || echo chars ---判断一个输入是否为整数的脚本

## uniq

uniq :检查及删除文本文件中重复出现的行列, stdin 必须是要已经排序好的.

uniq -c或--count 在每列旁边显示该行重复出现的次数。

-d或--repeated 仅显示重复出现的行列。

-f<栏位>或--skip-fields=<栏位> 忽略比较指定的栏位。

-s<字符位置>或--skip-chars=<字符位置> 忽略比较指定的字符。

-u或--unique 仅显示出一次的行列。

-w<字符位置>或--check-chars=<字符位置> 指定要比较的字符。

## wc

wc 统计文档的行数、字符数、词数

wc [-lmw] l 统计行数(line)

wc [-lmw] m 统计字符数

wc [-lmw] w 统计词数(word)

-c 统计字节数, 目测与-m无差异

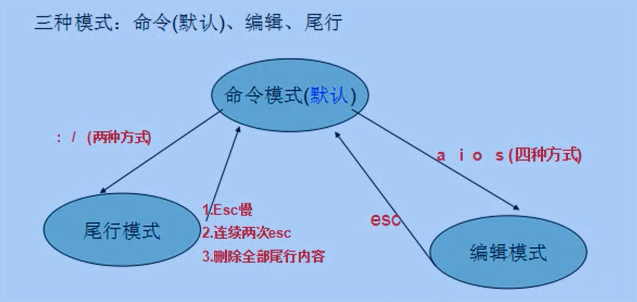
-L 打印最长行的长度

## let

let :BASH 中用于计算的工具，用于执行一个或多个表达式，变量计算中不需要加上 $ 来表示变量。

例 :let a=5+4(等同于a =$((5+4)), 但是没有(())计算效率高, 所以常用(()).);let no++

## + vi



命令模式

h 光标向左移动一个字符 例:5j

j 光标向下移动一个字符

k 光标向上移动一个字符

l 光标向右移动一个字符

Ctrl+f 屏幕向下移动一页

Ctrl+b 屏幕向上移动一页

Ctrl+d 屏幕向下移动半页

Ctrl+u 屏幕向上移动半页

0/[Home] 行首

$/[End] 行尾

G 文档最后一行

nG 文档第n行

gg 文档第一行

n+enter 向下移动n行

w 下个单词首字母

b 上个单词首字母

e 下个单词结尾

/word 向下寻找word字符串

:n1,n2s/word1/word2/g :1,$s/word1/word2/g

:1,$s/word1/word2/gc 交互模式

x,X x:向后删除一个字符,X:向前删除一个字符

dd ndd , 删除(剪切)

yy 复制当前行 nyy

p,P p:在下一行粘贴上,P:在上一行粘贴

u 撤销上一次动作

. 重复前一个动作

i,I(进入插入模式) i:在光标所在字符前开始插入,I:在光标所在行的行首开始插入 如果行首有空格则在空格之后插入

a,A(进入插入模式) a:在光标所在字符后开始插入,A:在光标所在你行的行尾开始插入

o,O(进入插入模式) o:在光标所在行的下面另起一新行插入,O:在光标所在行的上面另起一行开始插入

s,S(进入插入模式) s:删除光标所在的字符并开始插入,S:删除光标所在行并开始插入

编辑模式 [Esc] 退出编辑模式,进入指令模式

末行模式 :w :w!

:q :q!

:wq

set nu 显示行号

set nonu 取消显示行号

## + seq

seq :产生从某个数到另外一个数之间的所有整数。

语法 :seq [选项]... 尾数

语法 :seq [选项]... 首数 尾数

语法 :seq [选项]... 首数 增量 尾数

选项 -f, --format=格式 使用printf 样式的浮点格式

-s, --separator=字符串 使用指定字符串分隔数字（默认使用: \n）

-w, --equal-width 在列前添加0 使得宽度相同

例 :seq -f"%3g" 9 11 ---%后面指定数字的位数 默认是%g, %3g那么数字位数不足部分是空格。

例 :seq -f"%03g" 9 11 ---数字位数不足部分是0

例 :seq -f"str%03g" 9 11 ---%前面制定字符串

例 :seq -w 98 101 ---指定输出数字同宽, 不能和-f一起用，输出是同宽的。

例 :seq -s" " -f"str%03g" 9 11 ---str009 str010 str011

例 :seq -s"`echo -e "\t"`" 9 11 ---要指定/t做为分隔符号：

例 :seq -s"`echo -e "\n"`" 9 11 ---指定\n作为分隔符号：

# 文件传输

## lprm

## lpr

## lpq

## lpd

## bye

## ftp

## uuto

## uupick

## uucp

## uucico

## tftp

## ncftp

## ftpshut

## ftpwho

## ftpcount

# 四、磁盘管理

## **[cd](http://www.runoob.com/linux/linux-comm-cd.html" \t "http://www.runoob.com/linux/_blank)**

cd:变换目录(change directory)

cd . 此层目录

.. 上一层目录

- 前一个工作目录

~ 当前用户所在家目录

## **[df](http://www.runoob.com/linux/linux-comm-df.html" \t "http://www.runoob.com/linux/_blank)**

df :列出系统内的filesystem , 显示磁盘使用情况

df -a 列出所有的文件系统,包括系统特有的/proc 等文件系统

-k 以KBytes的容量显示各文件系统

-m 以MBytes的容量显示各文件系统

-h 以易阅读的GBytes,MBytes,KBytes等格式自行显示

-H 以M=1000K取代M=1024K的进位方式

-T 连同该partition的filesystem名称(例如xfs)也列出

-i 不用磁盘容量,而用inode的数量来显示

## **dirs**

## **[du](http://www.runoob.com/linux/linux-comm-du.html" \t "http://www.runoob.com/linux/_blank)**

du :列出目前目录下的所有文件总量

du [-ahskm] 文件/目录 -a 列出所有文件与目录总量,默认仅统计目录底下的文件量

-h 以易阅读的GBytes,MBytes,KBytes等格式自行显示

-s 列出总量而已，而不列出每个个别的目录占用总量

-S 显示个别目录大小时,并不含其子目录的大小

-k 以KBytes列出容量显示

-m 以MBytes列出容量显示

## edquota

## eject

## mcd

## mdeltree

## mdu

## **[mkdir](http://www.runoob.com/linux/linux-comm-mkdir.html" \t "http://www.runoob.com/linux/_blank)**

mkdir:建立一个新的目录

mkdir -m 设定目录权限,如果没有设定,系统会使用默认属性 mkdir -m 711 test2

mkdir -p 将所需要的目录递归建立起来 mkdir -p test1/test2

## **mlabel**

## **mmd**

## **mrd**

## **mzip**

## **quota**

## **[pwd](http://www.runoob.com/linux/linux-comm-pwd.html" \t "http://www.runoob.com/linux/_blank)**

pwd:显示当前工作目录(print working directory)

pwd -P 显示确实路径,而非使用链接路径

## **[mount](http://www.runoob.com/linux/linux-comm-mount.html" \t "http://www.runoob.com/linux/_blank)**

mount:文件系统挂载

mount /dev/hda1 /mnt

## **mmount**

## **rmdir**

## **rmt**

## **stat**

## **tree**

## **[umount](http://www.runoob.com/linux/linux-comm-umount.html" \t "http://www.runoob.com/linux/_blank)**

## **[ls](http://www.runoob.com/linux/linux-comm-ls.html" \t "http://www.runoob.com/linux/_blank)**

ls -l l表示链接文件(相当于快捷方式),b表示块设备文件(比如:硬盘),c表示字符设备文件,(例如:鼠标、键盘),p表示设备管道文件,s表示sock文件 查看文件及子目录的详细信息

ls -l |head -10 取10条记录,ls -l 查出来的时间是m-time

-a 列出全部文件,连同隐藏文件

-d 仅列出目录

## **quotacheck**

## **quotaoff**

## **lndir**

## **repquota**

## **quotaon**

# 五、磁盘维护

## badblocks

## cfdisk

## dd

## e2fsck

## ext2ed

## fsck

## fsck.minix

## fsconf

## fdformat

## hdparm

## mformat

## mkbootdisk

## mkdosfs

## mke2fs

## mkfs.ext2

## mkfs.msdos

## mkinitrd

## mkisofs

## mkswap

## mpartition

## swapon

## symlinks

## sync

## mbadblocks

## mkfs.minix

## fsck.ext2

## fdisklosetup

## mkfs

## sfdisk

## swapoff

# 六、网络通讯

## apachectl

## arpwatch

## dip

## getty

## mingetty

## uux

## telnet

## uulog

## uustat

## ppp-off

## netconfig

## nc

## httpd

## ifconfig / ip link

ifconfig 获取网络接口配置信息并对此进行修改, 不接任何参数时就会输出当前网络接口的情况.

ifconfig add<地址>：设置网络设备IPv6的ip地址；

del<地址>：删除网络设备IPv6的IP地址；

down：关闭指定的网络设备；

<hw<网络设备类型><硬件地址>：设置网络设备的类型与硬件地址；

io\_addr<I/O地址>：设置网络设备的I/O地址；

irq<IRQ地址>：设置网络设备的IRQ；

media<网络媒介类型>：设置网络设备的媒介类型；

mem\_start<内存地址>：设置网络设备在主内存所占用的起始地址；

metric<数目>：指定在计算数据包的转送次数时，所要加上的数目；

mtu<字节>：设置网络设备的MTU；

netmask<子网掩码>：设置网络设备的子网掩码；

tunnel<地址>：建立IPv4与IPv6之间的隧道通信地址；

up：启动指定的网络设备；

-broadcast<地址>：将要送往指定地址的数据包当成广播数据包来处理；

-pointopoint<地址>：与指定地址的网络设备建立直接连线，此模式具有保密功能；

-promisc：关闭或启动指定网络设备的promiscuous模式；

IP地址：指定网络设备的IP地址；

网络设备：指定网络设备的名称。

例 :ifconfig

eth0 Link encap:Ethernet HWaddr 00:16:3E:00:1E:51

---eth0表示第一块网卡, 连接类型 :Ethernet（以太网）, HWaddr（硬件mac地址）

inet addr:10.160.7.81 Bcast:10.160.15.255 Mask:255.255.240.0

---网卡的IP地址、子网、掩码。

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

---UP :网卡开启状态, RUNNING :网卡的网线被接上, MULTICAST :支持组播,

---MTU:1500(最大传输单元) :1500字节.

RX packets:61430830 errors:0 dropped:0 overruns:0 frame:0

---接收数据包情况统计.

TX packets:88534 errors:0 dropped:0 overruns:0 carrier:0

---发送数据包情况统计。

collisions:0 txqueuelen:1000

RX bytes:3607197869 (3.3 GiB) TX bytes:6115042 (5.8 MiB)

---接收、发送数据字节数统计信息

lo Link encap:Local Loopback

inet addr:127.0.0.1 Mask:255.0.0.0

UP LOOPBACK RUNNING MTU:16436 Metric:1

RX packets:56103 errors:0 dropped:0 overruns:0 frame:0

TX packets:56103 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:5079451 (4.8 MiB) TX bytes:5079451 (4.8 MiB)

---说明 :

lo是表示主机的回环地址，这个一般是用来测试一个网络程序，但又不想让局域网或外网的用户能够查看，只能在此台主机上运行和查看所用的网络接口。比如把 httpd服务器的指定到回环地址，在浏览器输入127.0.0.1就能看到你所架WEB网站了。但只是您能看得到，局域网的其它主机或用户无从知道。

例 :ifconfig eth0 up ---启动网卡eth0

例 :ifconfig eth0 down :关闭网卡eth0, ssh登陆linux服务器操作要小心, 关闭了就不能开启了, 除非有多网卡

例 :ifconfig eth0 add 33ffe:3240:800:1005::2/64 ---为网卡eth0配置IPv6地址

例 :ifconfig eth0 del 33ffe:3240:800:1005::2/64 ---为网卡eth0删除IPv6地址

例 :ifconfig eth0 hw ether 00:AA:BB:CC:dd:EE ---用ifconfig修改MAC地址

例 :ifconfig eth0 192.168.2.10 netmask 255.255.255.0 broadcast 192.168.2.255 ---配置IP地址

例 :ifconfig eth0 arp ---开启网卡eth0 的arp协议

例 :ifconfig eth0 -arp ---关闭网卡eth0 的arp协议

例 :ifconfig eth0 mtu 1500 ---设置能通过的最大数据包大小为 1500 bytes

**ip link**

ip link :用于网络接口的配置

ip link set：修改设备属性；

list|show：查看二层设备属性；

help：查看简要使用帮助；

①ip link|show：查看二层设备属性

例 :ip link ---查看主机上各网卡接口的二层属性, 此时list或show可省略

例 :ip link show ens33 ---查看ens33接口的二层属性, 查看指定接口属性时, show或list不能省略

②ip link set：修改设备属性

主要属性：

dev NAME (default)：指定要管理的设备；dev关键字可省略；

up | down：启用或禁用网络接口；

multicast { on | off }：启用或禁用组播功能；

name NEWNAME：重命名接口；

mtu MTU\_NUMBER：设置MTU的大小；默认为1500；

netns { PID | NAME }：将接口移动至指定的网络名称空间中; 要求Linux内核支持该功能(CentOS 7支持)

例 :ip link set ens38 down ---禁用ens38接口的使用

例 :ip link set ens38 up ---重新启用ens38接口

例 :ip link set ens38 name ens30 ---接口必须在DOWN状态下才可重命名

例 :ip link set ens30 mtu 1400 multicast off ---将接口ens30的MTU改为1400, 且关闭multicast功能

例 :---创建名称空间mynet，并将ens30接口移至名称空间mynet中

例 :ip netns add mynet ---创建名称空间mynet

例 :ip netns list ---查看已有的名称空间

例 :ip link set ens30 netns mynet ---将ens30接口移至名称空间mynet中；

例 :ip netns exec mynet ip link show ---在名称空间mynet之上运行命令"ip link show"，而"ip netns exec mynet"类似于进入虚拟的名称空间中，在此之上运行COMMAND.

例 : ip link show ---重新查看接口属性, 发现ens30已经消失了, 因为它已经跑到名称空间mynet里去了,基于netns这种机制可以搭建复杂的内部网络/虚拟网络。

## minicom

## mesg

## dnsconf

## wall

## netstat

## ping

ping :测试主机之间网络的连通性。执行ping指令会使用ICMP传输协议，发出要求回应的信息，若远端主机的网络功能没有问题，就会回应该信息，因而得知该主机运作正常。

ping -d：使用Socket的SO\_DEBUG功能；

-c<完成次数>：设置完成要求回应的次数；

-f：极限检测；

-i<间隔秒数>：指定收发信息的间隔时间；

-I<网络界面>：使用指定的网络界面送出数据包；

-l<前置载入>：设置在送出要求信息之前，先行发出的数据包；

-n：只输出数值；

-p<范本样式>：设置填满数据包的范本样式；

-q：不显示指令执行过程，开头和结尾的相关信息除外；

-r：忽略普通的Routing Table，直接将数据包送到远端主机上；

-R：记录路由过程；

-s<数据包大小>：设置数据包的大小；

-t<存活数值>：设置存活数值TTL的大小；

-v：详细显示指令的执行过程。

例 :ping [www.linuxde.net](http://www.linuxde.net) ----ping主机, 需要手动终止Ctrl+C

例 :ping -c 2 [www.w3cschool.cc](http://www.w3cschool.cc) ---指定接收包的次数, 收到两次包后, 自动退出

例 :ping -i 3 -s 1024 -t 255 g.cn ----i 3 发送周期为 3秒 -s 设置发送包的大小 -t 设置TTL值为 255

## pppstats

## samba

## setserial

## talk

## traceroute

## tty

## newaliases

## uuname

## netconf

## write

## statserial

## efax

## pppsetup

## tcpdump

## ytalk

## cu

## smbd

## testparm

## smbclient

## shapecfg

## + telinit

telinit :更改Linux系统的运行级别。运行级别可以是0～6之间的一个数字，其中0是关闭系统，1是进入单用户模式，2～5是多用户运行级别，6是重新启动系统。s或S表示单用户模式。

telinit -e<键=值>: 在运行级别事件中设置环境变量

-q: 减少输出，只有错误信息

-v: 增加输出，包括信息性消息

例 :telinit 3; runlevel ---更改Linux系统的运行级别为3。

## +sysctl

## +route

rout :来显示并设置Linux内核中的网络路由表，route命令设置的路由主要是静态路由。要实现两个不同的子网之间的通信，需要一台连接两个网络的路由器，或者同时位于两个网络的网关来实现。在Linux系统中设置路由通常是为了解决以下问题：该Linux系统在一个局域网中，局域网中有一个网关，能够让机器访问Internet，那么就需要将这台机器的ip地址设置为Linux机器的默认路由。要注意的是，直接在命令行下执行route命令来添加路由，不会永久保存，当网卡重启或者机器重启之后，该路由就失效了；可以在/etc/rc.local中添加route命令来保证该路由设置永久有效。

route -A：设置地址类型；

-C：打印将Linux核心的路由缓存；

-v：详细信息模式；

-n：不执行DNS反向查找，直接显示数字形式的IP地址；

-e：netstat格式显示路由表；

-net：到一个网络的路由表；

-host：到一个主机的路由表。

Add：增加指定的路由记录；

Del：删除指定的路由记录；

Target：目的网络或目的主机；

gw：设置默认网关；

mss：设置TCP的最大区块长度（MSS），单位MB；

window：指定通过路由表的TCP连接的TCP窗口大小；

dev：路由记录所表示的网络接口。

例 :route ---显示当前路由

Kernel IP routing table

Destination Gateway Genmask Flags Metric Ref Use Iface

112.124.12.0 \* 255.255.252.0 U 0 0 0 eth1

10.160.0.0 \* 255.255.240.0 U 0 0 0 eth0

192.168.0.0 10.160.15.247 255.255.0.0 UG 0 0 0 eth0

172.16.0.0 10.160.15.247 255.240.0.0 UG 0 0 0 eth0

10.0.0.0 10.160.15.247 255.0.0.0 UG 0 0 0 eth0

default 112.124.15.247 0.0.0.0 UG 0 0 0 eth1

例 :route -n

Kernel IP routing table

Destination Gateway Genmask Flags Metric Ref Use Iface

112.124.12.0 0.0.0.0 255.255.252.0 U 0 0 0 eth1

10.160.0.0 0.0.0.0 255.255.240.0 U 0 0 0 eth0

192.168.0.0 10.160.15.247 255.255.0.0 UG 0 0 0 eth0

172.16.0.0 10.160.15.247 255.240.0.0 UG 0 0 0 eth0

10.0.0.0 10.160.15.247 255.0.0.0 UG 0 0 0 eth0

0.0.0.0 112.124.15.247 0.0.0.0 UG 0 0 0 eth1

其中Flags为路由标志，标记当前网络节点的状态，Flags标志说明：

U Up表示此路由当前为启动状态。

H Host，表示此网关为一主机。

G Gateway，表示此网关为一路由器。

R Reinstate Route，使用动态路由重新初始化的路由。

D Dynamically,此路由是动态性地写入。

M Modified，此路由是由路由守护程序或导向器动态修改。

! 表示此路由当前为关闭状态。

例 :route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0 #增加一条到达244.0.0.0的路由。

例 :route add -net 224.0.0.0 netmask 240.0.0.0 reject #增加一条屏蔽的路由, 目的地址为224.x.x.x将被拒绝

例 :route del -net 224.0.0.0 netmask 240.0.0.0 ---删除路由记录

例 :route del -net 224.0.0.0 netmask 240.0.0.0 reject ---删除路由记录

例 :route del default gw 192.168.120.240 ---删除默认网关

例 :route add default gw 192.168.120.240 ---添加设置默认网关

# 七、系统管理

## **adduser**

## **chfn**

## **useradd**

useradd 建立用户帐号, 帐号建好之后, 再用passwd设定帐号的密码, 可用userdel删除帐号, 使用useradd指令所建立的帐号，实际上是保存在/etc/passwd文本文件中.(只有管理员能够操作)

useradd -c<备注>：加上备注文字。备注文字会保存在passwd的备注栏位中；

-d<登入目录>：指定用户登入时的启始目录；

-D：变更预设值；

-e<有效期限>：指定帐号的有效期限；

-f<缓冲天数>：指定在密码过期后多少天即关闭该帐号；

-g<群组>：指定用户所属的群组；

-G<群组>：指定用户所属的附加群组；

-m：自动建立用户的登入目录；

-M：不要自动建立用户的登入目录；

-n：取消建立以用户名称为名的群组；

-r：建立系统帐号；

-s<shell>：指定用户登入后所使用的shell；

-u<uid>：指定用户id

例 :useradd –g sales jack –G company,employees ---新建用户加入组 :-g :加入主要组、-G :加入次要组

例 :useradd caojh -u 544 ---建立一个新用户账户, 并设置ID :设定ID值时尽量要大于500, 以免冲突. 因为Linux安装后会建立一些特殊用户, 一般0到499之间的值留给bin、mail这样的系统账号.

例 :useradd tt ---添加一般用户

例 :useradd -r tt ---创建一个系统用户

例 :useradd -d /home/myd tt ---为新添加的用户指定home目录

## **[date](http://www.runoob.com/linux/linux-comm-date.html" \t "http://www.runoob.com/linux/_blank)**

date 显示或设置系统时间与日期

date -d “字符串” :显示字符串所指的日期与时间。字符串前后必须加上双引号；

-s “字符串” :设置系统时间, 设置完成后，写入CMOS :clock -w.

例 :date -s “2018 -01-01 00:00:00” ;clock -w

-u :显示GMT；

+”时间日期格式” :指定显示时使用的日期时间格式

小时 %H : 小时(00..23)

%I : 小时(01..12)

%k : 小时(0..23)

%l : 小时(1..12)

分钟 %M : 分钟(00..59)

AM/PM %p : 显示本地 AM 或 PM

秒 %S : 秒(00..61)

%s : 从 1970 年 1 月 1 日 00:00:00 UTC 到目前为止的秒数

时间 %T : 直接显示时间 (24 小时制) hh:mi:ss

%r : 直接显示时间 (12 小时制，格式为 hh:mm:ss [AP]M)

%X : 相当于 %H:%M:%S hh:mi:ss

年 %y : 年份的最后两位数字 (00.99)

%Y : 完整年份 (0000..9999)

月 %b : 月份 (Jan..Dec)

%B : 月份 (January..December)

%h : 同 %b

%m : 月份 (01..12)

日 %d : 日 (01..31)

礼拜几 %a : 星期几 (Sun..Sat)

%A : 星期几 (Sunday..Saturday)

日期时间%c : 直接显示日期与时间(aaa mmm dd hh:mi:ss yyyy)

日期 %D : 直接显示日期 (mm/dd/yy)

%x : 直接显示日期 (mm/dd/yy)

%Z : 显示时区

% :打印出 %

%n : 下一行

%t : 跳格

%j : 一年中的第几天 (001..366)

%U : 一年中的第几周 (00..53) (以 Sunday 为一周的第一天的情形)

%w : 一周中的第几天 (0..6)

%W : 一年中的第几周 (00..53) (以 Monday 为一周的第一天的情形)

例 :date -d "1 day ago" +"%Y-%m-%d" ---输出昨天日期

例 :date -d "2009-12-12" +"%Y/%m/%d %H:%M.%S" ---普通转格式

例 :date -d "Dec 5, 2009 12:00:37 AM" +"%Y-%m-%d %H:%M.%S" ---apache格式转换

## **[exit](http://www.runoob.com/linux/linux-comm-exit.html" \t "http://www.runoob.com/linux/_blank)**

exit 退出目前的shell, 执行exit可使shell以指定的状态值退出. 若不设置状态值参数, 则shell以预设值退出.状态值0代表执行成功, 其他值代表执行失败. exit也可用在script, 离开正在执行的script, 回到shell.

## finger

## fwhios

## **[sleep](http://www.runoob.com/linux/linux-comm-sleep.html" \t "http://www.runoob.com/linux/_blank)**

sleep :将目前动作延迟一段时间

sleep number : 时间长度，后面可接 s、m、h 或 d, 其中 s 为秒，m 为 分钟，h 为小时，d 为日数

## suspend

## groupdel

## groupmod

## halt

## **[kill](http://www.runoob.com/linux/linux-comm-kill.html" \t "http://www.runoob.com/linux/_blank)**

## **[last](http://www.runoob.com/linux/linux-comm-last.html" \t "http://www.runoob.com/linux/_blank)**

## lastb

## login

## logname

## logout

## **[ps](http://www.runoob.com/linux/linux-comm-ps.html" \t "http://www.runoob.com/linux/_blank)**

ps :显示当前进程 (process) 的状态

三种语法格式 :UNIX 风格, 选项可以组合在一起, 并且选项前必须有“-”连字符

BSD 风格，选项可以组合在一起, 但是选项前不能有“-”连字符

GNU 风格的长选项, 选项前有两个“-”连字符

Basic option:

a all with tty, except session leaders

-A, -e all processes

-a all with tty, except session leaders

-d all except session leaders.

--deselect, -N negate selection

g 显示现行终端机下的所有进程, 包括群组领导者的进程.

T all process on this terminal

r only running processes

x processes without control ttys

Select by list:

-C <command> command line

-G, --Group<GID> real group ID(RGID) or name.

-g , --group<group> session or effective group name

p, -p, --pid<PID> process ID

--ppid<PID> parent process ID

q, -q,--quick-pid<PID> process ID(quick mode)

-s, --sid<session> session ID

t, -t, --tty<tty> terminal

-u, U, --user<UID> effective user ID(EUID) or name

-U, --User<UID> real user ID or name

Output formats:

-c

--context display security context(for SELinux)

-f full-format, including command lines

-F extra full

f, --forest ascii art process tree

--format format 与’-o’和’o’相同

-H show process hierarchy

j BSD job control format

-j jobs format

l BSD long format

-l long format

-M, Z add security data(for SELinux)

O as -O, with BSD personality

-O<format> preloaded with default columns

-o, o, --format<format> user-defined format

s signal format, 采用程序信号的格式显示程序状况

u user-oriented format, 以用户为主的格式来显示程序状况

v virtual memory format

X register format

-y do not show flags, show rss vs. addr(used with -l)

--headers repeat header lines, one per page

--no-headers do not print header at all

--cols, --solumns,--width<num> set screen width

--rows, --lines<num> set screen height

show threads:

H as if there were processes

-L possibly with LWP and NLWP columns

-m, m after processes

-T possibly with SPID column

Miscellaneous options:

-c

c

e show the environment after command, 列出程序时, 显示每个程序所使用的环境变量

k, --sort

L

n

S, --cumulative include some dead child process data

-y

-w, w unlimited output width, 显示加宽可以显示较多的资讯

常用等

ps -ef 显示所有进程信息，连同命令行 ---与ps -e -f 一致

---输出格式 : UID PID PPID C STIME TTY TIME CMD

ps aux 列出目前所有的正在内存当中的程序, aux会截断command列

---USER, PID, %CPU, %MEM, VSZ, RSS, TTY, STAT, START, TIME, COMMAND

说明：

USER：该 process 属于那个使用者账号的

PID ：该 process 的号码

%CPU：该 process 使用掉的 CPU 资源百分比

%MEM：该 process 所占用的物理内存百分比

VSZ ：该 process 使用掉的虚拟内存量 (Kbytes)

RSS ：该 process 占用的固定的内存量 (Kbytes)

TTY ：该 process 是在那个终端机上面运作，若与终端机无关，则显示 ?，另外， tty1-tty6 是本机上面的登入者程序，若为 pts/0 等等的，则表示为由网络连接进主机的程序。

STAT：该程序目前的状态，主要的状态有

R ：该程序目前正在运作，或者是可被运作

S ：该程序目前正在睡眠当中 (可说是 idle 状态)，但可被某些讯号 (signal) 唤醒。

T ：该程序目前正在侦测或者是停止了

Z ：该程序应该已经终止，但是其父程序却无法正常的终止他，造成 zombie (疆尸) 程序的状态

START：该 process 被触发启动的时间

TIME ：该 process 实际使用 CPU 运作的时间

COMMAND：该程序的实际指令

ps -l --F ,S ,UID, PID, PPID, C, PRI, NI, ADDR, SZ, WCHAN, TTY, TIME, CMD

各相关信息的意义：

F 代表这个程序的旗标 (flag)， 4 代表使用者为 super user

S 代表这个程序的状态 (STAT)，关于各 STAT 的意义将在内文介绍

UID 程序被该 UID 所拥有

PID 就是这个程序的 ID ！

PPID 则是其上级父程序的ID

C CPU 使用的资源百分比

PRI 这个是 Priority (优先执行序) 的缩写，详细后面介绍

NI 这个是 Nice 值，在下一小节我们会持续介绍

ADDR 这个是 kernel function，指出该程序在内存的那个部分。如果是个 running的程序，一般就是 "-"

SZ 使用掉的内存大小

WCHAN 目前这个程序是否正在运作当中，若为 - 表示正在运作

TTY 登入者的终端机位置

TIME 使用掉的 CPU 时间。

CMD 所下达的指令为何

例 :ps -u root ---显示root正在运行的所有进程

例 :ps -e; ps -ef; ps -eF; ps -ely ---使用标准语法查看系统上的每个进程

例 :ps ax; ps axu ---使用BSD语法查看系统中的每个进程

例 :ps -ejH; ps axjf ---打印进程树

例 :ps -eLf; ps axms ---获取线程信息

例 :ps -eo euser, ruser, suser, fuser, f,comm, label; ps axZ; ps -eM ---获取安全信息

例 :ps -U root -u root u ---以用户格式查看每个进程作为根（实数和有效ID）运行

例 :ps -eo pid, tid, class, rtprio, ni, pri, psr, pcpu, stat, wchan:14, com

ps axo stat, euid, ruid, tty, tpgid, sess, pgrp, ppid, pid, pcpu,comm

ps -Ao pid tt,user, fname, tmout, f, wchan ---以用户定义的格式查看每个进程

例 :ps -C syslogd -o pid= ---只打印守护进程的PID

例 :ps -q 42 -o comm= ---只打印PID为42的命令

## nice

## procinfo

## **[top](http://www.runoob.com/linux/linux-comm-top.html" \t "http://www.runoob.com/linux/_blank)**

## pstree

## reboot

## rlogin

## rsh

## sliplogin

## screen

## shutdown

## rwho

## sudo

## gitps

## swatch

## tload

## logrotate

## uname

uname :可显示电脑以及操作系统的相关信息(内核版本号、硬件架构、主机名称和操作系统类型等)

uname -a, --all：显示全部的信息；

-m, --machine：显示电脑类型----x86\_64或i686

-n, -nodename：显示在网络上的主机名称----glot-runner/localhost

-r, --kernel-release：显示操作系统的发行编号----4.13.0-25-generic

-s, --kernel-name：显示操作系统名称----Linux

-v, --kernel-version：显示操作系统的版本----#1 SMP Tue May 21 15:34:22 EDT 2013

-p, --processor：输出处理器类型或"unknown"----x86\_64/i686

-i, --hardware-platform：输出硬件平台或"unknown"----x86\_64/i386

-o, --operating-system：输出操作系统名称----GNU/Linux

## chsh

## userconf

## userdel

userdel :删除给定的用户，以及与用户相关的文件。若不加选项，则仅删除用户帐号，而不删除相关文件。

userdel -f：强制删除用户，即使用户当前已登录；

-r：删除用户的同时，删除与用户相关的所有文件。

例 :userdel linuxde ---删除用户linuxde，但不删除其家目录及文件；

例 :userdel -r linuxde ---删除用户linuxde，其家目录及文件一并删除；

请不要轻易用-r选项；他会删除用户的同时删除用户所有的文件和目录，切记如果用户目录下有重要的文件，在删除前请备份。

其实也有最简单的办法，但这种办法有点不安全，也就是直接在/etc/passwd中删除您想要删除用户的记录；但最好不要这样做，/etc/passwd是极为重要的文件，可能您一不小心会操作失误。

## usermod

## vlock

## who

## whoami

whoami 命令用于显示自身用户名称, 相当于执行"id -un"指令

例 :whoami ---root

## whois

## newgrp

## renice

## **[su](http://www.runoob.com/linux/linux-comm-su.html" \t "http://www.runoob.com/linux/_blank)**

su 用于切换用户, 管理员切换到普通用户不需要密码，普通用户切换到管理员需要密码，普通用户之间切换同样需要密码。

su - 身份切换成root 使用exit回到当前身份

例 :su – linux00 ---切换到linux00用户

## skill

## w

## **[id](http://www.runoob.com/linux/linux-comm-id.html" \t "http://www.runoob.com/linux/_blank)**

id :显示真实有效的用户ID(UID)和组ID(GID)

id -g或--group :显示用户所属群组的ID。

-G或--groups :显示用户所属附加群组的ID。

-n或--name :显示用户，所属群组或附加群组的名称。

-r或--real :显示实际ID。

-u或--user :显示用户ID。

## free

## + hostname

hostname :显示和设置系统的主机名称。环境变量HOSTNAME也保存了当前的主机名。在使用hostname命令设置主机名后，系统并不会永久保存新的主机名，重新启动机器之后还是原来的主机名。如果需要永久修改主机名，需要同时修改/etc/hosts和/etc/sysconfig/network的相关内容。

hostname -v：详细信息模式；

-a：显示主机别名；

-d：显示DNS域名；

-f：显示FQDN名称；

-i：显示主机的ip地址；

-s：显示短主机名称，在第一个点处截断；

-y：显示NIS域名。

例 : hostname

## + groupadd

groupadd :创建一个新的工作组，新工作组的信息将被添加到系统文件中。

groupadd -g：指定新建工作组的id；

-r：创建系统工作组，系统工作组的组ID小于500；

-K：覆盖配置文件“/ect/login.defs”；

-o：允许添加组ID号不唯一的工作组。

例 :groupadd -g 344 linuxde ---建立一个新组，并设置组ID加入系统, 此时在/etc/passwd文件中产生一个组ID（GID）是344的项目。

## + service

service :是Redhat Linux兼容的发行版中用来控制系统服务的实用工具，它以启动、停止、重新启动和关闭系统服务，还可以显示所有系统服务的当前状态。

service --status-all :显示所服务的状态。

例 :service network status ----查看网络状态

例 :service network restart ---重启网络

例 :service mysqld status; service mysqld restart ---重启mysql

## + fg

fg :用于将后台作业（在后台运行的或者在后台挂起的作业）放到前台终端运行。与bg命令一样，若后台任务中只有一个，则使用该命令时，可以省略任务号。在Linux系统中执行某些操作时候，有时需要将当前任务暂停调至后台，或有时须将后台暂停的任务重启开启并调至前台，这一序列的操作将会使用到 jobs、bg、和 fg 三个命令以及两个快捷键来完成。

例 :fg 1 ---将任务号为1的任务从后台执行转换到前台执行

## + bg

bg :将作业放到后台运行，使前台可以执行其他任务。该命令的运行效果与在指令后面添加符号&的效果是相同的，都是将其放到系统后台执行。在Linux系统中执行某些操作时候，有时需要将当前任务暂停调至后台，或有时须将后台暂停的任务重启开启并调至前台，这一序列的操作将会使用到 jobs、bg、和 fg 三个命令以及两个快捷键来完成。

例 :bg 1 ---将任务号为1的任务放到后台继续执行

## +jobs

jobs :显示Linux中的任务列表及任务状态，包括后台运行的任务。该命令可以显示任务号及其对应的进程号。其中，任务号是以普通用户的角度进行的，而进程号则是从系统管理员的角度来看的。一个任务可以对应于一个或者多个进程号。

jobs -l :显示进程号

-p :仅任务对应的显示进程号

-n :显示任务状态的变化

-r :仅输出运行状态（running）的任务

-s :仅输出停止状态（stoped）的任务

例 :jobs -l ---使用jobs命令显示当前系统的任务列表.---[1] + 1903 运行中 find / -name password &

注意 :要得到以上输出信息, 必须在执行jobs命令之前执行命令find / -name password &. 否则, 执行jobs命令不会显示任何信息. 其中, 输出信息的第一列表示任务编号, 第二列表示任务所对应的进程号, 第三列表示任务的运行状态, 第四列表示启动任务的命令.

# 八、系统设置

## **reset**

## **[clear](http://www.runoob.com/linux/linux-comm-clear.html" \t "http://www.runoob.com/linux/_blank)**

## **[alias](http://www.runoob.com/linux/linux-comm-alias.html" \t "http://www.runoob.com/linux/_blank)**

alias:创建别名

alias 例:alias ll='ls -al'

1. **dircolors**
2. **aumix**
3. **bind**
4. **chroot**
5. **clock**
6. **crontab**
7. **declare**

declare -a 定义为数组型变量 例:declare -a cd='([0]="a" [1]="b" [2]="c")'

-i 定义为整型变量

-x 与export一样

-r 变量设定成read only

+ -变+为取消某个动作

1. **depmod**
2. **dmesg**

dmesg :用于显示开机信息, kernel会将开机信息存储在ring buffer中. 若是开机时来不及查看信息, 可利用dmesg来查看. 开机信息亦保存在/var/log目录中, 名称为dmesg的文件里.

dmesg -c 　显示信息后，清除ring buffer中的内容。

-s<缓冲区大小> 　预设置为8196，刚好等于ring buffer的大小。

-n 　设置记录信息的层级。

例 :dmesg |less ---显示开机信息

1. **enable**
2. **eval**

## **[export](http://www.runoob.com/linux/linux-comm-export.html" \t "http://www.runoob.com/linux/_blank)**

export:用于设置或显示环境变量,export可新增,修改或删除环境变量,供后续执行的程序使用.export的效力仅及于该次登陆操作.

export [-fnp][变量名称]=[变量设置值] 在shell脚本中可让子程序继承环境变量(仅限source和.)

## pwunconv

## grpconv

## rpm

## insmod

## kbdconfig

## lilo

## liloconfig

## lsmod

## minfo

## **[set](http://www.runoob.com/linux/linux-comm-set.html" \t "http://www.runoob.com/linux/_blank)**

set:显示所有变量(环境变量和自定义变量)

## modprobe

## ntsysv

## mouseconfig

## passwd

passwd :设置用户的认证信息，包括用户密码、密码过期时间等。只有管理者可以指定用户名称，一般用户只能变更自己的密码。

passwd -d, --delete 删除用户密码，仅能以root权限操作

-f, --force 强制操作；仅root权限才能操作

-i, --inactive=DAYS 在密码过期后多少天，用户被禁掉，仅能以root操作；

-k, --keep-tokens 保留即将过期的用户在期满后能仍能使用(keep non-expired authentication tokens)

-l, --lock 锁住用户无权更改其密码，仅能通过root权限操作

-n, --minimum=DAYS 两次密码修改的最小天数，后面接数字，仅能root权限操作；

-S, --status 查询用户的密码状态，仅能root用户操作；

-u, --unlock 解开已上锁的帐号

-w, --warning=DAYS 在距多少天提醒用户修改密码；仅能root权限操作；

-x, --maximum=DAYS 两次密码修正的最大天数，后面接数字；仅能root权限操作

例 :passwd -l ghf ---锁定用户ghf不能更改密码；

例 :passwd -d ghf ---清除beinan用户密码, 当我们清除一个用户的密码时, 登录时就无需密码

## pwconv

## rdate

## resize

## rmmod

## grpunconv

## modinfo

## time

time :统计给定命令所花费的总时间

time -o 或 --output=FILE：将执行时间写入到文件中。如果档案已经存在，系统将覆写其内容。

-a 或 --append：配合 -o 使用，会将结果写到档案的末端，而不会覆盖掉原来的内容。

-f FORMAT 或 --format=FORMAT：以 FORMAT 字串设定显示方式。当这个选项没有被设定的时候,会用系统预设的格式。不过你可以用环境变数 time 来设定这个格式，如此一来就不必每次登入系统都要设定一次。

## setup

## sndconfig

## setenv

## setconsole

## timeconfig

## ulimit

## **[unset](http://www.runoob.com/linux/linux-comm-unset.html" \t "http://www.runoob.com/linux/_blank)**

unset:取消变量

## chkconfig

## apmd

## hwclock

## mkkickstart

## fbset

## **[unalias](http://www.runoob.com/linux/linux-comm-unalias.html" \t "http://www.runoob.com/linux/_blank)**

unalias 例:unalias ll

## SVGATextMode

## **+ env**

env:显示系统当前环境变量

## **+ IFS**

shell会依据IFS（Internal Field Seperator）将command line所输入的文字给拆解为字段（word）。然后再针对特殊字符或者叫元字符（meta）先作处理，最后再重组整行command line,IFS 默认值是空格、制表符和换行符

IFS='\n' 普通字符反斜杠和n

IFS=$"\n" 换行符，不过只有在执行的时候才会进行转化

IFS=$'\n' 已经转换了的回车符NL，在屏幕上直接就回车

$var 会根据IFS的值来分割后合并输出 值用空格隔开 例:var=": a:b::c:" a b c

"$var" 用引号引起来，这个变量不用IFS替换 值用分隔符隔开 看到这个变量的"原始值" 例:var=": a:b::c:" : a:b::c:

$\* 会根据 IFS 的不同来组合值

$@ 会用" "来组合值

## + chage

chage :修改帐号和密码的有效期限

chage -m days： 密码可更改的最小天数。为零时代表任何时候都可以更改密码。

-M days： 指定口令有效的最多天数。当该选项指定的天数加上-d选项指定的天数小于当前的日期时，用户在使用该帐号前就必须改变口令。

-d days： 指定从1970年1月1日起，口令被改变的天数。

-I days： 指定口令过期后，帐号被锁前不活跃的天数。如果值为0，帐号在口令过期后就不会被锁。

-E date： 指定帐号被锁的日期。日期格式YYYY-MM-DD。若不用日期，也可以使用自1970年1月1日后经过的天数。

-W days： 指定口令过期前要警告用户的天数。

-l： 列出指定用户当前的口令时效信息，以确定帐号何时过期。

例 :chage -m 2 -M 30 -W 5 user1 ---设置用户user1两天内不能更改口令，并且口令最长的存活期为30天，并且口令过期前5天通知用户

## yum

yum :

yum check-update :列出所有可更新的软件清单命令

update :更新所有软件命令

search <keyword> :查找软件包

install <package\_name> :仅安装指定的软件命令

update <package\_name>: 仅更新指定的软件命令

list: 列出所有可安裝的软件清单命令

remove <package\_name>: 删除软件包命令

clean packages: 清除缓存目录下的软件包

clean headers: 清除缓存目录下的 headers

clean oldheaders: 清除缓存目录下旧的 headers

clean, yum clean all (= yum clean packages; yum clean oldheaders) :清除缓存目录下的软件包及旧的headers

# 备份压缩

## **gzip,zcat/zmore/zless/zgrep**

gzip:可解开由compress,zip,gzip等软件所压缩的程序

gzip [-cdtv#] file -c 将压缩的数据输出到屏幕上

-d 解压缩

-t 检验一个压缩文件的一致性(文件有无错误)

-v 可以显示出源文件/压缩文件的压缩比等比较信息

-# 压缩等级,预设是-6

## **bzip2,bzcat/bzmore/bzless/bzgrep**

bzip2 -c 将压缩的数据输出到屏幕上

-d 解压缩

-k 保留源文件,不删除原始文件

-z 压缩的参数(默认值,可以不加)

-v 可以显示出源文件/压缩文件的压缩比等比较信息

-# 压缩等级

## **xz ,xzcat/xzmore/xzless/xzgrep**

xz -c 将压缩的数据输出到屏幕上

-d 解压缩

-t 检验一个压缩文件的一致性(文件有无错误)

-l 列出压缩文件的相关信息

-k 保留源文件,不删除原始文件

-# 压缩等级

## **tar**

tar :打包

tar [-z|-j|-J] [cv] [-f filename] filename2 -c 建立打包文件 tar -cvf - /etc | tar -xvf -

tar [-z|-j|-J] [tv] [-f filenme] -t 查看打包文件的内容含有哪些文件名

tar [-z|-j|-J] [xv] [-f filenme] [-C dir] -x 解打包或解压缩的功能 tar -jxv -f /root/etc.tar.bz2 etc/shadow:解压一个文件

-z 通过gzip的支持进行压缩/解压缩, 命名 :\*.tar.gz

-j 通过bzip2的支持进行压缩/解压缩, 命名 :\*.tar.bz2

-J 通过xz的支持进行压缩/解压缩

-v 在压缩/解压缩过程中，将正在处理的文件名显示出来

-f filename filename为压缩文件

-C dir 指定压缩/解压缩的目录dir

-p 保留备份数据的原本权限与属性,常用于备份(-c)重要的配置文件

-P 保留绝对路径，亦即允许备份数据中含有跟目录存在之意

--exclude=file 在压缩过程中,不要将FILE打包

# 十、设备管理

# 十一、其他指令

## bc

bc 计算器 预设仅输出小数,csale=number,number就是小数点位数,输入quit离开软件

bash不支持浮点类型计算，浮点的运算需要使用bc/awk.

bc scale 设定小数精度；

ibase 表示输入的数的进制;

obase 表示输出的数的进制;

例:echo 0.5\*4-0.2|bc;

例:echo $(awk 'BEGIN{print 0.5\*4-0.2}');

例:echo 'scale=2; (2.777 - 1.4744)/1' | bc;

例:var1=$(echo "obase=10;ibase=2;$abc" | bc);

## cal

## printf

\a 警告声音

\b 退格键

\f 清除

\n 输出新的一行

\r enter

\t 水平Tab

\v 垂直Tab

\xNN NN为两位数字，可以转换数字为字符

printf %ns n个字符的字符串

printf %ni n个字符的整形数据

printf %N,nf 例:%8.2f代表的是整数部分5位，小数点1位，小数2位，8代表的是8个位数

## type

type:显示指定命令的类型，判断给出的指令是内部指令还是外部指令。

type [-tpa] name -t 输出'file'、'alias'或者'builtin',分别表示给定的指令为'外部指令'、'命令别名'或者'内部指令'

-p 如果给出的指令为外部指令,则显示其绝对路径

-a 在环境变量'PATH'指定的路径中,显示给定指令的信息,包括命令别名.

"命令类型:

alias" 别名

keyword 关键字,Shell保留字

function 函数,Shell函数

builtin 内建命令,Shell内建命令

file 文件,磁盘文件,外部命令

unfound 没有找到

## seq

$（seq 开始 步长 最后）,默认开始为1,步长为1,例:$(seq 9).

## shift

shift :用于对参数的移动(左移), 通常用于在不知道传入参数个数的情况下依次遍历每个参数然后进行相应处理, shift(shift 1) 命令每执行一次，变量的个数($#)减一（之前的$1变量被销毁,之后的$2就变成了$1）, 而变量值提前一位。

例 :shift 3

## trap

trap 运行格式

trap命令的参数分为两部分，前一部分是接收到指定信号时将要采取的行动，后一部分是要处理的信号名。

trap command signal

它有三种形式分别对应三种不同的信号回应方式。

第一种：

trap "commands" signal-list

当脚本收到signal-list清单内列出的信号时，trap命令执行双引号中的命令.

第二种：

trap signal-list

trap不指定任何命令，接受信号的默认操作，默认操作是结束进程的运行。

第三种：

trap " " signal-list

trap命令指定一个空命令串，允许忽视信号，我们用到的就是这一种。

※请记住，脚本程序通常是以从上到下的顺序解释执行的，所以必须在你想保护的那部分代码以前指定trap命令。

信号量详细列表可以trap -l即可显示

名称 默认动作 说明

SIGHUP 终止进程 终端线路挂断

SIGINT 终止进程 中断进程

SIGQUIT 建立CORE文件 终止进程，并且生成core文件

SIGILL 建立CORE文件 非法指令

SIGTRAP 建立CORE文件 跟踪自陷

SIGBUS 建立CORE文件 总线错误

SIGSEGV 建立CORE文件 段非法错误

SIGFPE 建立CORE文件 浮点异常

SIGIOT 建立CORE文件 执行I/O自陷

SIGKILL 终止进程 杀死进程

SIGPIPE 终止进程 向一个没有读进程的管道写数据

SIGALARM 终止进程 计时器到时

SIGTERM 终止进程 软件终止信号

SIGSTOP 停止进程 非终端来的停止信号

SIGTSTP 停止进程 终端来的停止信号

SIGCONT 忽略信号 继续执行一个停止的进程

SIGURG 忽略信号 I/O紧急信号

SIGIO 忽略信号 当子进程停止或退出时通知父进程

SIGTTOU 停止进程 后台进程写终端

SIGTTIN 停止进程 后台进程读终端

SIGXGPU 终止进程 CPU时限超时

SIGXFSZ 终止进程 文件长度过长

SIGWINCH 忽略信号 窗口大小发生变化

SIGPROF 终止进程 统计分布图用计时器到时

SIGUSR1 终止进程 用户定义信号1

SIGUSR2 终止进程 用户定义信号2

SIGVTALRM 终止进程 虚拟计时器到时

## + prev=(${curr[\*]})

prev=(${curr[\*]}) :初始化数组, 表示将curr[\*]的所有值分别传递给prev[\*], 值得注意的是, curr[\*]会根据\*值的大小进行排序,依次赋值给prev[\*], prev[\*] 中\*值由0开始.

注 :必须要加括号, 否则会将curr[\*]的所有值赋值给prev[0]

例 : a[1234]=1234; a[1]=1; a[0]=0; a[2]=2; b=(${a[\*]}); echo ${b[0]}, ${b[1]}, ${b[2]},${b[3]};echo ${b[\*]}

----返回结果----0, 1, 2,1234 \n 0 1 2 1234

## + pmrep

pmrep connect -r RS\_PMS\_SIT -d Domain\_PMSBISA1 -n Administrator -x Administrator

pmrep objectimport -i  wf\_dtmv\_achis\_upoup.xml -c control.XML -l rep.log(用于导入指定文件)

-i input\_xml\_file\_name :要导入的object(通常是xml格式)文件名称

-c control\_file\_name, 控制文件名

-l log\_file\_name, 日志文件名称

pmrep objectexport -n wf\_name -o workflow -f folder\_name -s -b -r -u exportfilename

-n object\_name，需要导出的workflow名称。

-o object\_type，导出的object类型，可以是”workflow”。

-f folder\_name ，需要导出的workflow的存放文件夹。

-s -b -r:这三个参数可以使导出的workflow中包含所有map的信息

-u xml\_output\_file\_name，要导出的workflow保存文件名。

-s -b -r:这三个参数可以使导出的workflow中包含所有map的信息

-o object\_type，导出的object类型，可以是”workflow”

## +db2

db2 list node directory :查看本地节点目录

db2 list db directory :查看数据库

db2 connect to DB\_alias\_name user username using password :连接到数据库

db2 -v “” :-v选项会回显当前执行的命令

db2 -x ”” :-x选项会省略列标题的打印

db2look -d db\_name -e -t table\_name -i user\_name -w password :查看表的DDL语句

db2 uncatalog node ND\_name :取消节点编目

db2 uncatalog db DB\_alias\_name :取消数据库的编目

db2 terminate :断开与数据库的连接, 相当于disconnect 后在quit

db2 catalog tcpip node ND\_name remote IP\_address sever Port\_name :编目一个TCP/IP节点

db2 catalog db DB\_name as DB\_alias\_name at node ND\_name :编目远端DB2数据库

db2 terminate

## +身份证校验位

1. 将前面的身份证号码17位数分别乘以不同的系数。从第一位到第十七位的系数分别为：7－9－10－5－8－4－2－1－6－3－7－9－10－5－8－4－2
2. 将这17位数字和系数相乘的结果相加
3. 用加出来和除以11，看余数是多少
4. 余数只可能有0－1－2－3－4－5－6－7－8－9－10这11个数字。其分别对应的最后一位身份证的号码为1－0－X－9－8－7－6－5－4－3－2

# Shell快捷键

[Ctrl]+a: 调到命令行的开头

[Ctrl]+c: 中断目前程序

[Ctrl]+d: EOF,EOI,exit

[Ctrl]+e: 调到命令行的末尾

[Ctrl]+k: 将光标处到命令行末尾的内容清除

[Ctrl]+l: 清屏

[Ctrl]+r: 在历史记录列表中搜索某一命令

[Ctrl]+u: 向前删除指令串

history 显示历史命令

# 变量内容删除、替代、替换等

Tip:加上冒号后,测试的变量未被设定或者空字符串时,都能够用后面的内容来替换与设定

注 :\*表示多个字符, ?表示一个字符

time : ${#string}< expr length “$string”<echo $string | wc -m

|  |  |  |  |
| --- | --- | --- | --- |
| 变量设定方式 | str没有设定 | str为空字符串 | str已设定为非空字符串 |
| var=${str-expr} | str=,var=expr | str=,var= | str不变,var=$str |
| var=${str:-expr} | str=,var=expr | var=expr | var=$str |
| var=${str+expr} | var= | var=expr | var=expr |
| var=${str:+expr} | var= | var= | var=expr |
| var=${str=expr} | var=expr,str=expr | str=,var= | str不变,var=$str |
| var=${str:=expr} | str=expr,var=expr | str=expr,var=expr | str不变,var=$str |
| var=${str?expr} | expr输出至stderr | var= | var=$str |
| var=${str:?expr} | expr输出至stderr | expr输出至stderr | var=$str |
| ${string#substring} | 删除最短匹配前缀 |  |  |
| ${string##substring} | 删除最长匹配前缀 |  |  |
| ${string%substring} | 删除最短匹配后缀 |  |  |
| ${string%%substring} | 删除最长匹配后缀 |  |  |
| ${string/substring/replace} | 替换第一次出现的匹配 |  |  |
| ${string//substring/replace} | 替换所有匹配 |  |  |
| ${str:n:m} | n,m为数字,提取第n个字节右边的m个字节 |  | ${file:0:5} |
| ${#string} | 变量值的长度 |  |  |
| ${A[@]} | 返回数组全部元素 | A=(a b c def) | a b c def |
| ${A[\*]} | 返回数组全部元素 | A=(a b c def) | a b c def |
| ${A[0]} | 返回数组第一个元素 | A=(a b c def) | a |
| ${#A[@]} | 返回数组元素总个数 | A=(a b c def) | 4 |
| ${#A[\*]} | 返回数组元素总个数 | A=(a b c def) | 4 |
| ${#A[3]} | 返回第四个元素的长度，即def的长度 | A=(a b c def) | 3 |
| A[3]=xyz | 则是将第四个组数重新定义为 xyz |  |  |
| $(( )) | 整数运算 | $(( (a+b)/c )) |  |
| $((n#m)) | n为2,8,16,将n进位的m数转为10进位 | $((16#2a)) | 42 |
| ${string:position} | 在$string中，从位置position之后开始提取子串,position是数字 |  |  |
| ${string:position:length} | 在$string中, 从位置position之后开始提取长度为length的子串 |  |  |
| ${string/#substring/replace} | 替换匹配前缀 |  |  |
| ${string/%substring/replace} | 替换匹配后缀 |  |  |

# 循环、函数等

## **if ... then ...**

if [expression1];then

statement1

elif((expression2));then

statement2

else

statement3

fi

例 :read -p "Please input your score: " score;

if [[ $score =~ [^0-9] ]] ; then echo "please input a positive integer" ; exit 10 --匹配正则

elif [ $score -gt 100 ];then echo "Your score is wrong" ; exit 20

elif [ $score -ge 85 ];then echo "Your score is very good" ;

elif [ $score -ge 60 ];then echo "Your score is soso"

else echo "You have to work hard"

fi

## **case ... in ...**

case $变量 in

"变量内容")

程序

;;

"变量内容2"）

程序2

;;

\*)

程序3

;;

esac

## **while ... do ... done**

while [condition]

do

程序段

done

## **until ... do ... done**

until [condition]

do

程序段

done

## **for ... do ... done**

for var in con1 con2...

do

程序段

done

for ((初始值;限制值;执行步阶))

do

程序段

done

for i in $(seq 9)与for((i=1;i<=9;i++)) 实现的结果一样

## **function**

function fname(){

程序段

}

# 正则表达式

|  |  |
| --- | --- |
| 正则表达式字符 | 意义/范例 |
| ^word | 行首 |
| Word$ | 行尾 |
| [ab] | 在方括号中, 标准的元字符会失去它们的含义,. |
| [^ab] | 方括号中具有特殊意义的字符有: \;-;^.(\只适用于awk中) |
| . | 任意一个字符 |
| \* | 重复前面的字符至少0次 |
| / | 跳脱字符 |
| \{m,n\} |  |
| [n1,n2] | 例:[0-9],[a-z] |
| POSIX字符类 :[:alnum:] | 英文大小写字符及数字，即0-9, a-z, A-Z |
| [:alpha:] | 英文大小写 |
| [:blank:] | 空格键，Tab键 |
| [:cntrl:] | 键盘上面的控制按键 |
| [:digit:] | 代表数字，0-9 |
| [:graph:] | 除了空格符和Tab按键以外的其他所有按键 |
| [:lower:] | 小写字符a-z |
| [:print:] | 任何可以被打印出来的字符 |
| [:punct:] | 标点符号，即：“ ‘ ？ ： |
| [:upper:] | 大写字符A-Z |
| [:spacde:] | 任何会产生空吧的字符，例：空格，”，’，: |
| [:xdigit:] | 16进位数字类型 |
|  |  |
| 扩展正则表达式egrep,grep -E | 意义/范例 |
| + | 至少匹配一次 |
| ? | 匹配0次或1次 |
| | | 或 |
| () | 常与|一起使用, 在egrep和awk中使用, 在sed -r与grep -E中也适用 |

sed, grep, egrep, awk 比较:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | sed | grep | egrep | awk |
| 正则表达式 | 支持 | 支持 | 支持 | 支持 |
| 延伸正则表达式 | sed -r | grep -E | 支持 | 支持 |
| \{m,n\} | \{m,n\},sed -r时用{} | \{m,n\} | {m,n} | {m,n} |
| (|) | sed -r | grep -E | 支持 | 支持 |
| 单词匹配\<\> | 支持 | 支持, grep -w | 支持 | 支持 |
| \(\),\1 | sed -r时用() |  |  |  |