第二十四章：shell脚本-正则、grep、sed、awk三剑客

**一、正则表达式及grep；**

**二、sed流编辑器；**

**三、awk：报告生成器；**

**四、awk的高级用法；**

**一、正则表达式及grep：**

**1.概述：**正则表达式--简写：regex、regexp或RE，就是为处理大量的字符串而定义的一台规则和方法。例：假设“@”代表oldboy，”!” 代表eitboy。则echo “@!” =”oldboy eitboy”。通过定义的这些特殊符号的辅助，系统管理员就可以快速过滤，替换或输出需要的字符串。

支持正则表达式的程序：

Linux正则表达式：grep、awk、sed

PHP、JAVA、Perl、Python（Perl兼容PCRE）

**2.学习目的：**在企业工作中，每天做的Linux运维工作中，时刻都会面对大量带有字符串的文本配置、程序、命令输出及日志文件等，而我们经常会有迫切的需要，从大量的字符串内容中查找符合工作需要的特定字符串。这就需要正则表达式，因此，可以说正则表达式就是为过滤这样的字符串而生的。

例：ifconfig的输出取IP，cat /var/log/messages输出取关键信息等。

**注意事项：**

1. 正则表达式应用非常广泛，存在于各种语言中，例：PHP、Python、JAVA、Perl等。但是，Linux 系统中的正则表达式，最常应用正则表达式的命令就是grep（egrep）、awk、sed，Linux三剑客想要更高效的工作，离不开正则表达式的配合。
2. 正则表达式与常用的通配符特殊字符是有本质区别的，一定要注意。

例：ls \*.log中的 \* 是通配符（表示所有），不是正则表达式。

**3.基础正则表达式：**

^word 搜索以word开头的。vi/vim中 ^ 一行的开头。

word$ 搜索以word结尾的。vi/vim中 $ 一行的结尾。

^$ 表示空行

. 代表且只能代表任意一个字符

\ 例：\. 只代表点本身，转义符号，让有特殊身份意义的字符，脱掉马甲，还原；

\n 换行符

\r 匹配回车

\w 匹配任意一个字符和数字

\* 重复0次或多次前面的一个字符。

.\* 匹配所有字符。例：^.\* 以任意多个字符开头，.\*$以任意多个字符结尾。

[abc] 匹配字符集内的任意一个字符。

[^abc] 匹配不包含 ^ 后的任意字符的内容。中括号里的 ^ 为取反。

[1-9] 表示匹配括号内的范围内的任意字符。

a\{n,m\} 重复n到m次前一个重复的字符。若用egrep、sed -r可以去掉斜线。

\{n,\} 重复至少n 次前一个重复的字符。若用egrep、sed -r可以去掉斜线。

\{n\} 重复n 次前一个重复的字符。若用egrep、sed -r可以去掉斜线。

\{,m}\ 重复少于m次。

**注：egrep，grep –E或sed –r 过滤一般特殊字符可以不转义。**

**4.扩展的正则表达式：**ERE（egrep或grep -E）

+ 重复一次或一次以上前面的一个字符。等同于\{n,} 磁盘

? 重复0次或一次前面的一个字符。等同于\{,m} \*

| 或者的意思，用或的方式查找多个符合的字符串。

() 找出括号内的字符串。

**5.理解正则：**（提示：若不在shell中运行，正则要生效必须借助支持正则的工具）；

^linux ##以linux开始

linux$ ##以linux结束

linuxfan. ##匹配linuxfans等

coo[kl] ##匹配cool或cook

9[^5689] ##匹配91,92等，但不匹配95,98等

[0-9] ##匹配任意一个所有的数字

[a-z]|[A-Z] ##匹配任意一个所有大小写字母,|属于扩展正则grep -E支持

colou?r ##匹配color或clolur，但是不能匹配colouur

rollno-9+ ##匹配rollno-9、rollno-99，rollno-999，但不匹配rollno-

co\*l ##匹配cl，col，cool，coool等

ma(tri)x ##匹配matrix

[0-9]{3} ##匹配任意一个三位数，等于[0-9][0-9][0-9]

[0-9]{2,} ##匹配任意一个两位数或更多位的数字

[0-9]{2,5} ##匹配从两位数到五位数之间的任意一个数字

Oct (1st|2nd) ##匹配Oct 1st或Oct 2nd

a\.b ##匹配a.b，但不能匹配ajb

[a-z0-9\_]+\@[a-z0-9\_]+\.[a-z]{2,4} ##匹配一个邮箱地址

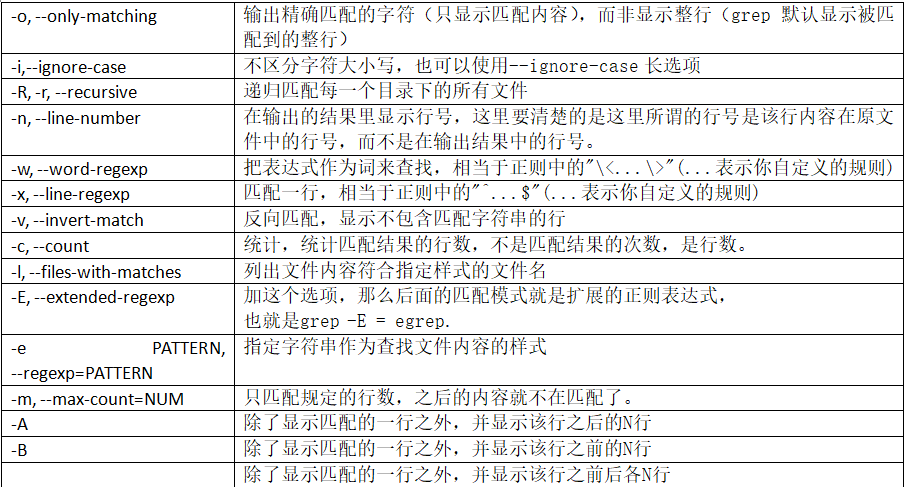
[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3} ##匹配IP地址

**6.grep用法：**

**作用：**grep命令是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来，过滤数据流或文件内容。

**语法格式：**grep [选项] [条件表达式] 目标文件

**常用选项：**



**4.案例：**

[root@ns ~]# cat /proc/meminfo |grep -e Mem -e Cache -e Swap ##查看系统内存、缓存、交换分区-e的作用是匹配多个表达式

[root@ns ~]# grep -R -o -n -E '[a-z0-9\_]+\@[a-z0-9\_]+\.[a-z]{2,4}' /etc/ ##查找/etc目录下的所有文件中的邮件地址；-R递归，-n表示匹配的行号，-o只输出匹配内容，-E支持扩展正则表达式，

[root@ns ~]# grep -R -c 'HOSTNAME' /etc/ |grep -v "0$" ##查找/etc/目录下文件中包含“HOSTNAME”的次数，-c统计匹配次数，-v取反

[root@ns ~]# grep -R -l 'HOSTNAME' /etc/ ##查找包含“HOSTNAME”的文件名，-l显示匹配的文件名，-L显示不匹配的文件名

[root@ns ~]# dmesg | grep -n --color=auto 'eth' ##查找内核日志中eth的行，显示颜色及行号

[root@ns ~]# dmesg | grep -n -A3 -B2 --color=auto 'eth' ##用 dmesg 列出核心信息，再以 grep 找出内含 eth 那行,在关键字所在行的前两行与后三行也一起找出出来显示

[root@ns ~]# cat /etc/passwd |grep -c bash$ ##统计系统中能登录的用户的个数

[root@ns tmp]# touch /tmp/{123,123123,456,1234567}.txt ##创建测试文件，以下三条命令是一样的效果，匹配文件名123，可以包含1个到多个

[root@ns tmp]# ls |grep -E '(123)+'

[root@ns tmp]# ls |grep '\(123\)\+'

[root@ns tmp]# ls |egrep '(123)+'

[root@ns ~]# ps -ef |grep -c httpd ##统计httpd进程数量

[root@ns ~]# grep -C 4 'games' --color /etc/passwd ##显示games匹配的“-C”前后4行

[root@ns ~]# grep ^adm /etc/group ##查看adm组的信息

[root@ns ~]# ip a |grep -E '^[0-9]' |awk -F : '{print $2}' ##获取网卡名称

[root@ns ~]# ip a |grep mtu|grep -o -E "[a-z]{3}[0-9]" ##获取网卡名称

[root@ns ~]# ifconfig eth0 |grep -E -o 'inet addr:[^ ]\*' |grep -o '[0-9.]\*' ##截取ip地址，[^ ]\*表示以非空字符作为结束符，[0-9.]\*表示数字和点的组合

[root@ns ~]# ip a |grep inet |grep eth0 |grep -o "inet[^/]\*" |grep -o "[0-9.]\*" ##截取ip地址

[root@ns ~]# ip a |grep -A 3 eth0 |grep global |grep -E -o "([0-9]{1,3}.){4}" |awk -F'/' '{print $1}' |head -1 ##截取ip地址

[root@ns ~]# ifconfig eth0 |grep -i hwaddr |awk '{print $5}' ##截取MAC地址

[root@ns ~]# ip a |grep -A 3 "eth0" |grep link/ether |grep -o "ether[^r]\*" |grep -o -E "[0-9a-f:]+"|grep -E "[0-9a-f:]{2}$" ##截取MAC地址

[root@oldboy ~]# alias grep='grep --color=auto'

[root@oldboy ~]# export LC\_ALL=C

[root@oldboy ~]# cat <<END >> oldboy.log

I am oldboy teacher!

I teach linux.

I like badminton ball,billiard ball and chinese chess!

my blog is http://oldboy.blog.51cto.com

our site is http://www.etiantian,org

my qq num is 49000448.

not 4900000448

my god,i am not oldbey,but OLDBOY!

END

[root@oldboy ~]# grep "^m" oldboy.log ##过滤输出以m开头的行

[root@oldboy ~]# grep "m$" oldboy.log

[root@oldboy ~]# grep -vn "^$" oldboy.log ##过滤空行

[root@oldboy ~]# grep -o "0\*" oldboy.log

000

00000

[root@oldboy ~]# grep -o "oldb.y" oldboy.log

oldboy

oldboy

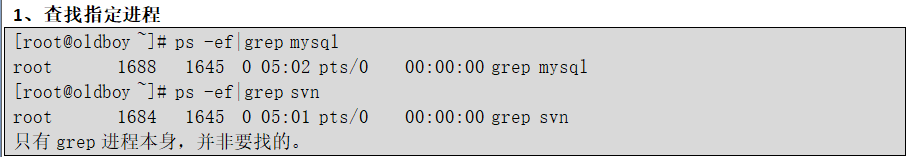
oldbey

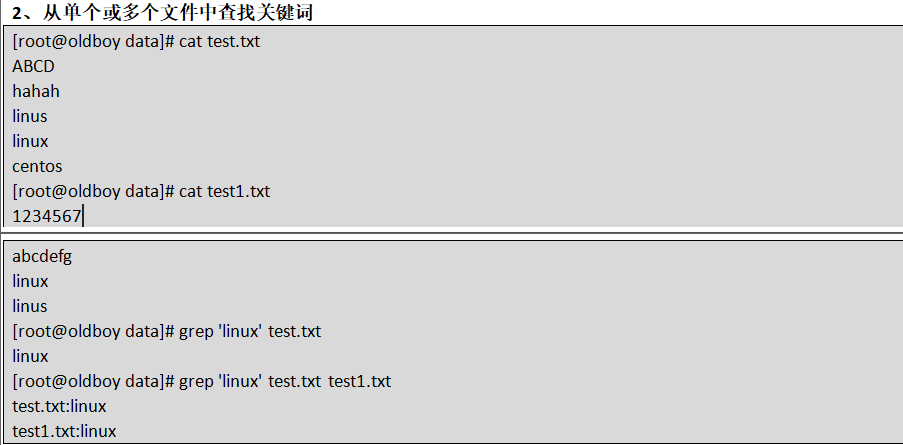
[root@oldboy ~]# grep "\.$" oldboy.log  ##以.结尾的行  
I teach linux.  
my qq num is 49000448.

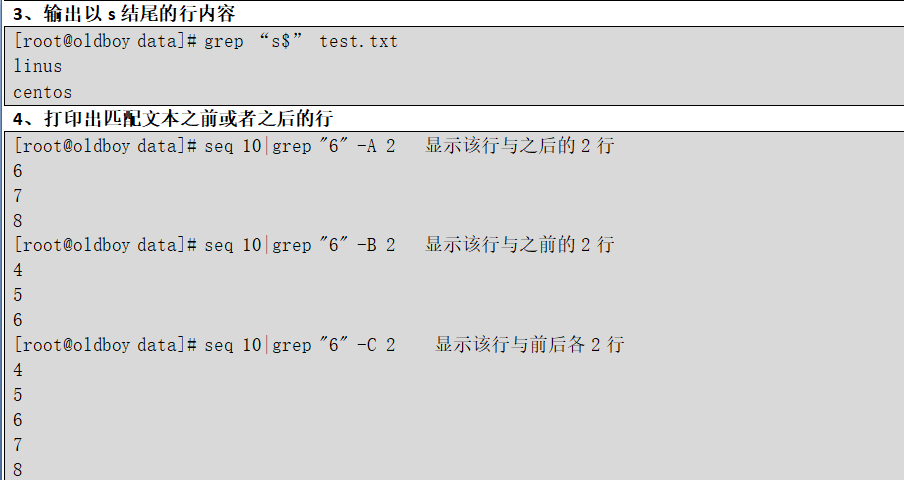
[root@oldboy ~]# grep "0\{3\}" oldboy.log ##重复三次

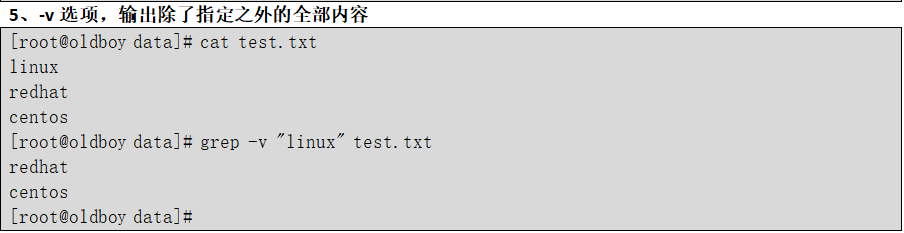
my qq num is 49000448.

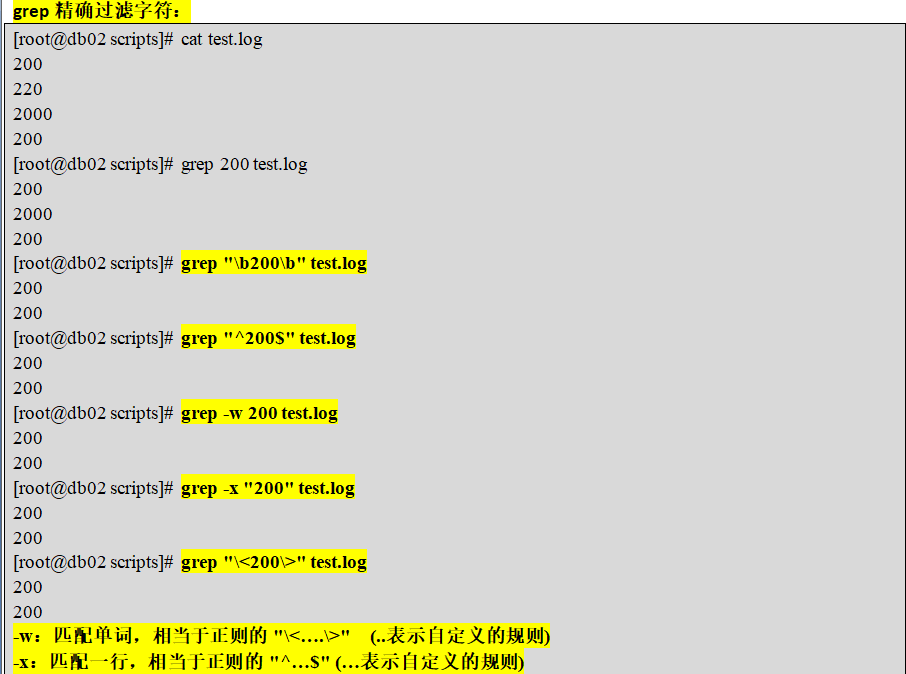
not 4900000448











**扩展正则表达式案例：**

[root@lwh tmp]# cat <<END >>2015.log

> good

> gd

> dod

> god

> goood

> END

[root@lwh tmp]# grep -E "go+d" 2015.log

good

god

goood

[root@lwh tmp]# grep -E "go?d" 2015.log

gd

god

[root@lwh tmp]# grep -E "dod|god" 2015.log

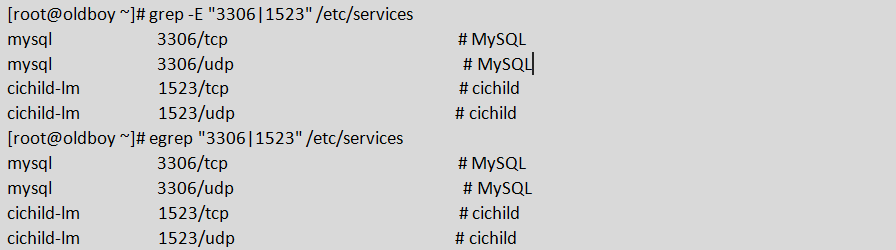
dod

god

[root@lwh tmp]# grep -E "(d|g)od" 2015.log

dod

god

****

**二、sed流编辑器：**

**概述：**一种流编辑器，它是文本处理中非常中的工具，能够完美的配合正则表达式使用。可删除(delete)、改变(change)、添加(append)、插入(insert)、合并、交换文件中的数据行，或读入其它文件的数据到文件中，也可替换(substuite)它们其中的字符串、或转换(tranfer)其中的字母等。

**语法：**sed [options] 'command' file(s)

**选项：**

-n 抑制自动打印pattern space，sed默认输出全部，-n用于取消默认输出；

-i 编辑文件；

-r 支持扩展正则表达式；

**1.改：**

**语法：**sed '/正则匹配条件/s/old/new/g' 文件

**常用语法：**sed -i 's/old/new/g' file ##所有内容进行替换

sed -i '2s/old/new/g' file ##基于行进行替换

sed -i '/reg/s/old/new/g' file ##基于正则匹配行进行替换

[root@ns ~]# sed 's/dhcp/static/g' /etc/sysconfig/network-scripts/ifcfg-eth1 ##只是显示，不修改

[root@ns ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth1

[root@ns ~]# sed -i 's/dhcp/static/g' /etc/sysconfig/network-scripts/ifcfg-eth1 ##只修改，不显示

[root@ns ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth1 ##验证

[root@ns ~]# vim ip ##编写测试文件

IP1=static

IP2=static

IP=static

:wq

[root@ns ~]# sed -i 's/dhcp/static/g' ip ##将所有的dhcp替换为static

[root@ns ~]# sed -i '/^IP1/s/static/dhcp/g' ip ##将IP1开头的行替换

[root@ns ~]# sed -i '2s/static/dhcp/g' ip ##指定特定行号2行替换

[root@ns ~]# cat -n /etc/selinux/config ##查看并显示行号

[root@ns ~]# sed -i '7s/disabled/enforcing/g' /etc/selinux/config ##开启selinux

[root@ns ~]# sed -i '7s/disabled/enforcing/g' /etc/selinux/config

**2.删：**

**语法：**sed '/表达式/d' 文件

**常用语法：**sed -i '2d' file ##基于行号进行删除

sed -i '/reg/d' file ##基于正则进行删除

[root@ns ~]# vim ip ##添加空行

[root@ns ~]# sed '/^$/d' ip ##删除空行并显示在屏幕上

[root@ns ~]# sed -i '/IP1/d' ip ##删除包含IP1的行

[root@ns ~]# sed -i '/^IP2/d' ip ##删除以IP2开头的行

[root@ns ~]# sed -i '2d' ip ##删除第二行

**3.增：**

**语法：**sed ' /表达式/a "需要添加的文字"' 文件

**常用语法：**sed -i '2a text' file ##在某行后添加内容

sed -i '2i text' file ##在某行前添加内容

sed -i '/reg/a text' file ##结合正则在匹配行后添加内容

[root@ns ~]# sed 'a IP3=static' ip ##每一行后都加上IP3=static

[root@ns ~]# sed '3a IP3=static' ip ##只在第3行后加上IP3=static，并显示不修改

[root@ns ~]# sed '3i IP3=static' ip ##只在第3行前加上IP3=static，显示不修改

[root@ns ~]# sed -i '3a IP3=static' ip ##修改，不显示

[root@ns ~]# sed -i '/^IP3/a "test add"' ip ##在以IP3开头的行后添加

**4.查：功能与grep相同**

**语法：**sed -n '/表达式/p' 文件

**常用语法：**sed -n '2p' file ##基于行号打印内容

sed -n '/reg/p' file ##基于正则匹配行打印内容

[root@ns ~]# sed -n '2p' /etc/hosts ##查看第二行

[root@ns ~]# sed -n '/www/p' /var/named/chroot/var/named/linuxfan.cn.zone ##查看包含www的解析记录

[root@ns ~]# sed -n '/.100$/p' /var/named/chroot/var/named/linuxfan.cn.zone ##查看以.100结尾的行

[root@ns ~]# sed -n '2~2p' ip ##从第二行，每隔两行显示

[root@oldboy ~]# ifconfig eth0|sed -n '2p'|sed 's#.\*dr:##g'|sed 's# Bc.\*##g' ##注：当sed命令处理的内容为多行内容，则以/作为表达式的分隔，若sed命令处理的内容为单行内容，作为截取的作用，以#号作为分隔符；

10.0.0.9

[root@oldboy ~]# ifconfig eth0|sed -n '2p'|sed -r 's#(.\*dr:)(.\*)(Bc.\*$)#\2#g' ##-r支持扩展正则，\2将2转义，打印出第二个范围(.\*)

10.0.0.9

[root@oldboy ~]# ifconfig eth0|sed -n '2p'|sed -r 's#.\*dr:(.\*) Bc.\*$#\1#g'

10.0.0.9

[root@oldboy ~]# ifconfig eth0|sed -nr '2s#^.\*dr:(.\*) Bc.\*$#\1#gp'

10.0.0.9

[root@oldboy ~]# ifconfig eth0|sed -nr '1s#^.\*dr (.\*)#\1#gp'

00:0C:29:33:C8:75

[root@oldboy ~]# ifconfig eth0|sed -n '1p'|sed -r 's#(^.\*dr )(.\*)#\2#g'

00:0C:29:33:C8:75

[root@oldboy ~]# ifconfig eth0|sed -n '1p'|sed 's#^.\*dr ##g'

00:0C:29:33:C8:75

[root@oldboy ~]# ifconfig eth0|sed -nr '1s#^.\*t (.\*) 00.\*$#\1#gp'

HWaddr

[root@oldboy ~]# stat /etc/hosts|sed -n '4p'

Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid: ( 0/ root)

[root@oldboy ~]# stat /etc/hosts|sed -n '4p'|sed 's#^.\*ss: (##g'|sed 's#/-.\*$##g'

0644

[root@oldboy ~]# stat /etc/hosts|sed -n '4p'|sed -r 's#^.\*s: \((.\*)/-.\*$#\1#g'

0644

[root@oldboy ~]# stat /etc/hosts|sed -nr '4s#^.\*s: \((.\*)/-.\*$#\1#gp'

0644

[root@oldboy ~]# stat /etc/hosts|sed -nr '4s#(^.\*s: \()(.\*)(/-.\*$)#\2#gp'

0644

**三、awk：报告生成器：**

**1.awk的作用：**一种编程语言，用于在linux/unix下对文本和数据进行处理。数据可以来自标准输入stdin、一个或多个文件，或其它命令的输出。简单来说awk就是把文件逐行的读入，以空格为默认分隔符将每行切片，切开的部分再进行各种分析处理。擅长取行、取列、过滤。

**2.语法：**

awk [选项] '模式{ 动作（action） }' 文件1 文件2 ...

选项： -F   指定输入分隔符，可以是字符串或正则表达式；

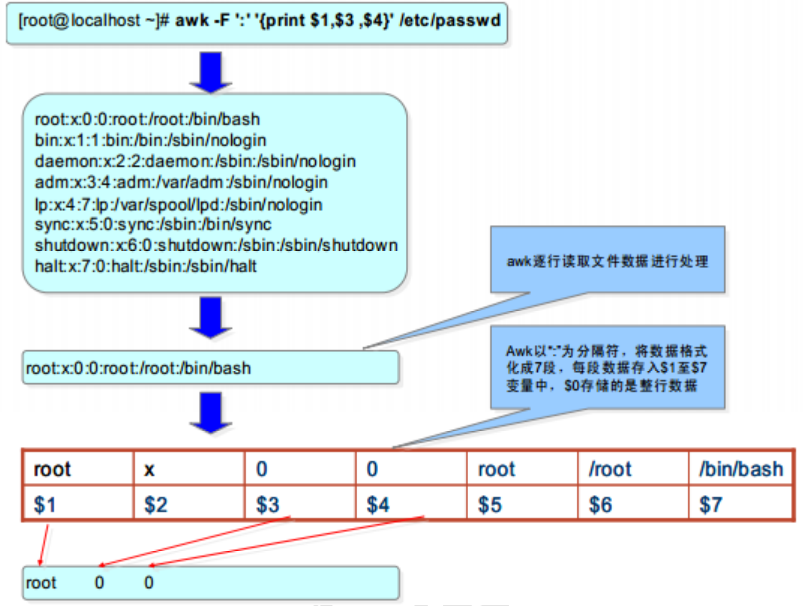
最常见的动作： print、printf

**3.awk的基本原理特征：**

1）每一次取一行

2）根据指定的分隔符（不指定是位空白字符）将该行切割位列，使用$0(整行),$1,$2,$3...（第一列,第二列,...）

3）可以指定行号，列号，切割符，操作后分隔符



**4.案例：**

chkconfig --list |grep 3:启用 |awk '{print $1}'

tail -1 /etc/passwd |awk -F ':' 'BEGIN{OFS="---"}{print $1,$6,$7}' ##OFS指定输出分隔符

ifconfig eth1 |awk -F '[ :]+' 'NR==2 {print $4}'

ifconfig eth1 |awk -F '[ :]+' 'NR==2 {print "eth1\_ip="$4}' ##可以加入显示内容

awk 'BEGIN {print "line one \nline two\nline three"}'

**5.匹配范围（ranges）：**指定的匹配范围，格式为part1,part2

**案例：**

awk -F : '$3==3,$3==10{print $1,$3,$7}' /etc/passwd

awk -F : '$1=="root",$1=="adm"{print $1,$3,$7}' /etc/passwd

awk -F : '/^r/,/^a/{print $1,$3,$7}' /etc/passwd

**6.awk区块原理：**

**区域构成：**

BEGIN { 动作 } ##开始处理第一行文本之前的操作

{ 动作 } ##针对每一行文本的处理操作

END { 动作 } ##处理完最后一行文本之后的操作

**执行流程：**

首先执行 BEGIN { } 区块中的初始化操作；

然后从指定的数据文件中循环读取一个数据行（自动更新 NF、 NR、 $0、 $1…… 等内建变量的值），并执行'模式或条件{ 动作 }'；

最后执行 END { } 区块中的后续处理操作。

**案例：**

awk -F : 'BEGIN{printf "%-10s%-10s%-20s\n","UserName","ID","Shell"}{printf "%-10s%-10s%-20s\n",$1,$3,$7}' /etc/passwd ##在awk处理之前打印头部BEGIN{}

ifconfig eth0 |awk -F':' 'NR==2{print $2,$4}'|awk 'BEGIN{OFS=" / "}{print "IP="$1,"MASK="$3}'

**7.awk的变量：**

**awk变量：**

FS：列分隔符，默认位空白

RS：行分隔符，默认位换行符

OFS：输出列分隔符

ORS ：输出行分隔符

**awk内置变量：**

NR：处理中行数

FNR：单个文件的行数

NF：列的个数

**案例：**

ifconfig eth1 |awk '{print NR}'

ifconfig eth1 |awk '{print NF}'

**自定义变量案例：**

awk 'BEGIN{test="www.linuxfan.cn";print test}'

awk -v test="linuxfan.cn" 'BEGIN{print test}'

**8.printf的使用：**

**格式：**printf "格式",列表1,列表2 ...

**特征：**

a.必须指定format(格式)，用于指定后面item（列表）的输出格式

b.printf语句不会自动打印换行符：\n

c.format格式以%加一个字符，如下：

%c:显示字符的ASCII码

%d，%i:十进制整数

%f:显示浮点数（小数）

%s:显示字符串

%u:无符号整数

%%:显示%

d.修饰符：N:显示宽度，-：左对齐，+：显示数值符号，如%-c(左对齐)

**案例：**

chkconfig --list |grep 3:启用 |awk '{printf "%-10s",$1}' ##在统一行显示

awk -F : '{printf "%-15s %-10d %-10s\n",$1,$3,$7}' /etc/passwd

**9.awk的操作符：**

**算数操作符：** x^y、x/y、x+y、x-y、x%y

**比较操作符：**>、<、>=、<=、==、!=

**逻辑操作符：**&&、||、！

**10.awk常见的模式类型**

**正则表达式（regexp）：**awk -F : '/^u/{print $1}' /etc/passwd

**表达式（expression）：**值位非0或位非空是满足条件，如$1 ~ /foo/或 $1 == "root"

**案例：**

awk -F : '$3>=500{print $1,$3,$7}' /etc/passwd ##打印普通用户

awk -F : '$3+1<=100&&$3+1>=10{print $1,$3,$7}' /etc/passwd ##UID在10-100之间的用户

awk -F : '$2=="!!"{print $1,$2}' /etc/shadow ##检查未初始化密码的用户

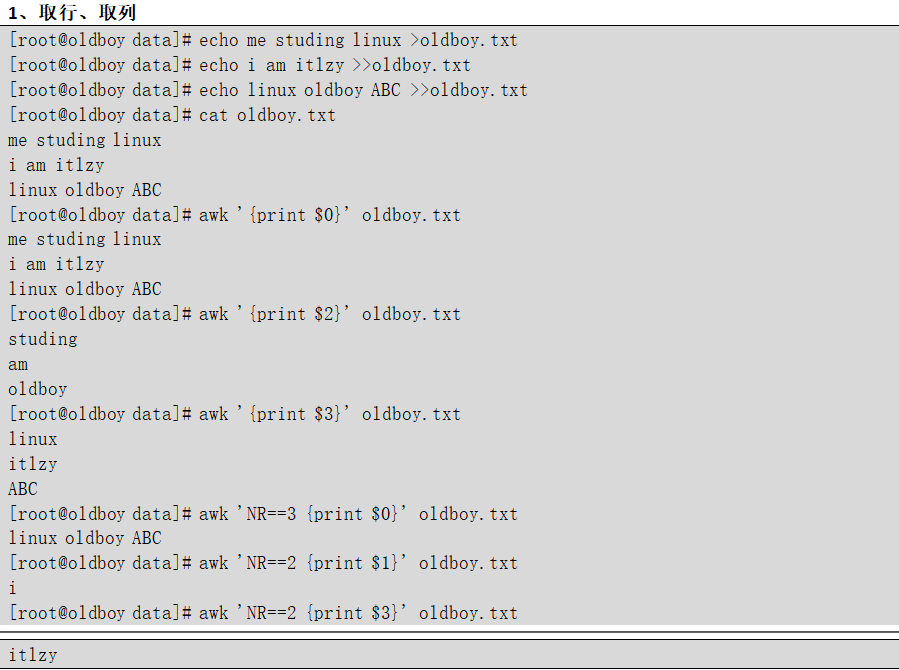
passwd -d u01

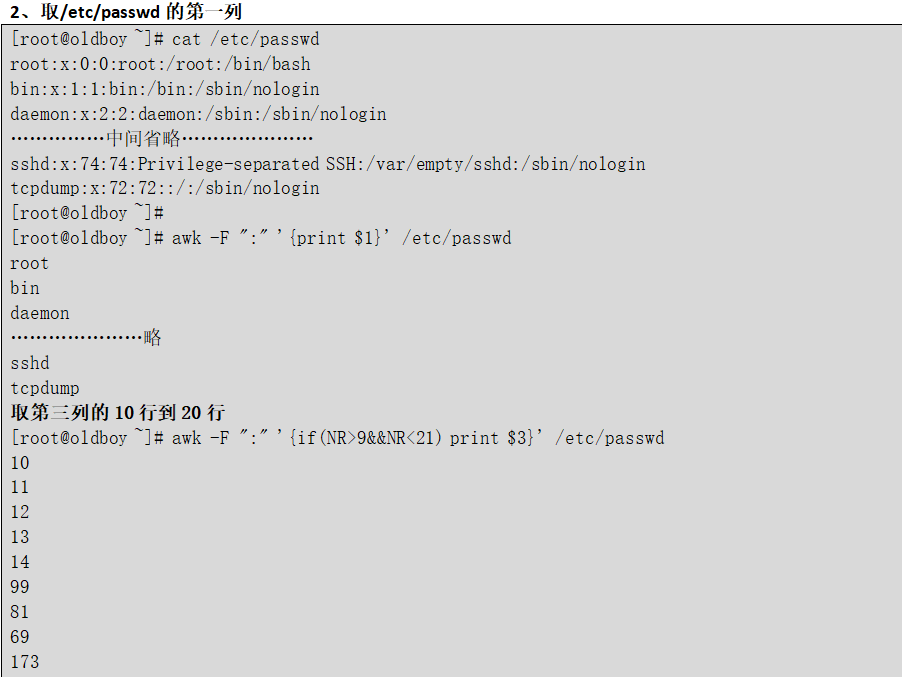
awk -F : '$2==""{print $1}' /etc/shadow ##打印密码为空的用户

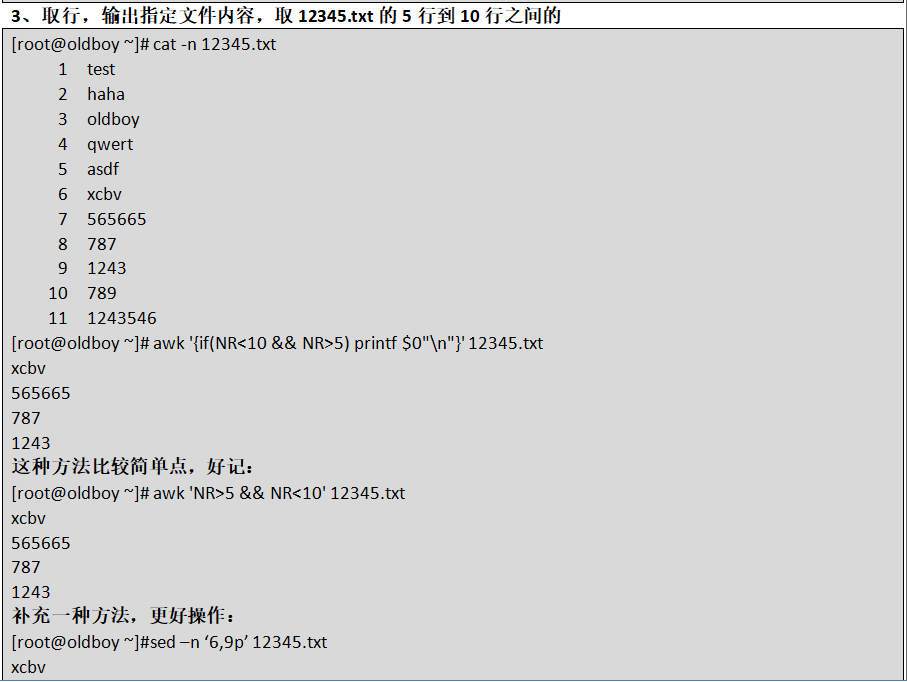
awk -F : '$7~"bash$"{print $1,$3,$7}' /etc/passwd ##匹配$7为bash结束行

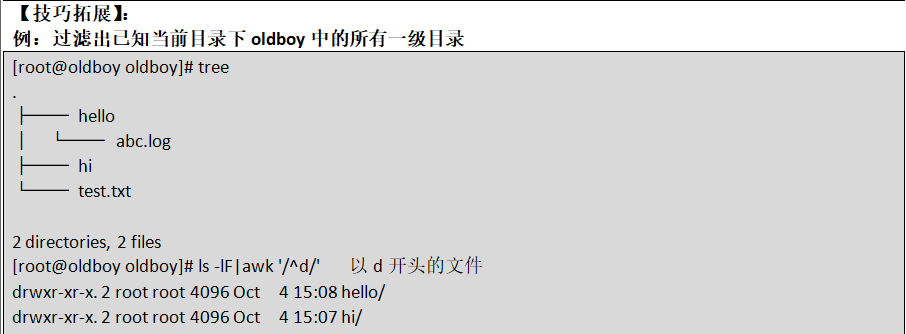
awk -F : '$7!~"bash$"{print $1,$3,$7}' /etc/passwd

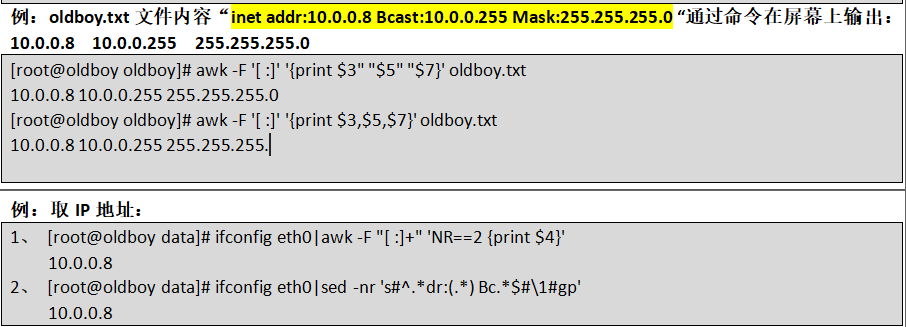
**案例：**











**四、awk的高级用法：**

**1.if-else：**判断

**语法：**if (条件表达式) 命令 操作1; else 命令 操作2

**案例：**

awk -F : '$7~"bash"{if ($1=="root") print $1, "admin";else print $1, "Common User"}' /etc/passwd ##第七列是bash，如果第一列是root打印root admin，否则打印出第一列 common user

awk -F : -v sum=0 '{if ($3>=500) sum++}END{print sum}' /etc/passwd ##输出普通用户的数量

awk -F : '$7~"bash$"{if ($1=="root") printf "%-15s: %s\n",$1,"admin";else printf "%-15s: %s\n",$1,"common user"}' /etc/passwd ##使用printf打印

**2.while循环：**循环字段（列），awk本身就是行的循环

**语法：**while (条件){语句1;语句2;...}

**案例：**

awk -F: '{i=1;while (i<=3) {print $i;i++}}' /etc/passwd ##循环打印前3列

awk -F : '{i=1;while (i<NF) {if (length($i)<=4) print $i;i++}}' /etc/passwd ##循环整行，并打印出长度小于4的字段列

**3.for：**循环

**语法：**for (初始变量;条件;变量自加){语句1, 语句2,...}

**案例：**

awk -F : '{for(i=1;i<=3;i++) print $i}' /etc/passwd

**4.数组**

**语法：**数组名[下标]

**解释：**awk中数组（array）的下标（index-expression）可以使用任意字符串；需要注意的是，如果某数据组元素事先不存在，那么在引用其时，awk会自动创建此元素并初始化为空串；因此，要判断某数据组中是否存在某元素，需要使用“下标 in 数组”的方式。要遍历数组中的每一个元素，需要使用如下的特殊结构：for (变量 in 数组) { 语句1, ... }

**案例：**

[root@localhost ~]# netstat -ant

Active Internet connections (servers and established)

Proto Recv-Q Send-Q Local Address Foreign Address State

tcp 0 0 0.0.0.0:22 0.0.0.0:\* LISTEN

tcp 0 0 127.0.0.1:25 0.0.0.0:\* LISTEN

tcp 0 0 192.168.1.106:22 192.168.1.101:36318 ESTABLISHED

以上命令显示的结果中，每一行最后一个字段为状态，若要统计每种状态的个数，只需要将state作为数组下标，由于awk本身就是一个行的循环，所以只需要给数字自加1（++）即可实现统计，具体操作如下：

[root@localhost ~]# netstat -ant |awk '/^tcp|^udp/{state[$6]++}END{for(i in state){print i,state[i]}}' ##state[$6]++表示定义数组state[]、$6是下标，++表示自加1

[root@localhost ~]# netstat -ant |awk '/^tcp/{S[$NF]++}END{for(a in S) print a, S[a]}'

[root@localhost ~]# awk '{counts[$1]++}END{for (url in counts) print url,"access times:",counts[url]}' /var/log/httpd/access\_log ##查看http的访问

[root@localhost ~]# awk '{AH[$1]++}END{for (i in AH) printf "%-20s:%s\n", i, AH[i]}' /var/log/httpd/access\_log