08.Shell正则应用

> 徐亮伟, 江湖人称标杆徐。多年互联网运维工作经验，曾负责过大规模集群架构自动化运维管理工作。擅长Web集群架构与自动化运维，曾负责国内某大型电商运维工作。
> 个人博客"徐亮伟架构师之路"累计受益数万人。
> 笔者Q:552408925、572891887
> 架构师群:471443208

正则表达式`regular expression, RE`是一种字符模式，用于在查找过程中匹配指定的字符。
在大多数程序里，正则表达式都被置于两个正斜杠之间;例如/l[oO]ve/就是由正斜杠界定的正则表达式，它将匹配被查找的行中任何位置出现的相同模式。在正则表达式中，元字符是最重要的概念。

正则表达式的作用

> 1.Linux正则表达式`grep,sed,awk`
> 2.大量的字符串文件需要进行配置，而且是非交互式的
> 3.过滤相关的字符串，匹配字符串，打印字符串

正则表达式注意事项

> 1.正则表达式应用非常广泛，存在于各种语言中，例如：php,python,java等。
> 2.正则表达式和通配符特殊字符是有本质区别的。

> 3.要想学好`grep、sed、awk`首先就要掌握正则表达式。

1.基础正则表达式

元字符意义BRE，正则表达式实际就是一些特殊字符，赋予了他特定的含义。

```
正则表达式    描述
\         转义符，将特殊字符进行转义，忽略其特殊意义
^         匹配行首，awk中，^则是匹配字符串的开始
$         匹配行尾，awk中，$则是匹配字符串的结尾
^$        表示空行
.         匹配除换行符\n之外的任意单个字符
[ ]       匹配包含在[字符]之中的任意一个字符
[^ ]      匹配[^字符]之外的任意一个字符
[ - ]     匹配[]中指定范围内的任意一个字符
?         匹配之前的项1次或者0次
+         匹配之前的项1次或者多次
*         匹配之前的项0次或者多次，.*
()        匹配表达式，创建一个用于匹配的子串
{ n }     匹配之前的项n次，n是可以为0的正整数
{n,}      之前的项至少需要匹配n次
{n,m}     指定之前的项至少匹配n次，最多匹配m次，n<=m
|         交替匹配|两边的任意一项ab(c|d)匹配abc或abd

特定字符:
>[[:space:]]     空格
[[:digit:]]      [0-9]
[[:lower:]]      [a-z]
[[:upper:]]      [A-Z]
[[:alpha:]]      [a-Z]
```

2.正则表达式实战

```
I am xuliangwei teacher!
I teach linux.
test

I like badminton ball ,billiard ball and chinese chess!
my blog is http://liangweilinux.blog.51cto.com
our site is http://www.xuliangwei.com
my qq num is 572891887.
not 572891888887.
```

```
//过滤以m开头的行
[root@Shell ~]# grep "^m" test.txt
my blog is http://liangweilinux.blog.51cto.com
my qq num is 572891887.
[root@Shell ~]# grep "m$" test.txt
my blog is http://liangweilinux.blog.51cto.com
our site is http://www.xuliangwei.com

//排除空行, 并打印行号
[root@student ~]# grep -vn "^$" xuliangwei.txt
//匹配任意一个字符,不包括空行
[root@student ~]# grep "." xuliangwei.txt
//.匹配所有
[root@student ~]# grep ".*" xuliangwei.txt
//匹配单个任意字符
[root@node1 ~]# grep "xuliangw.i" xuliangwei.txt
//以点结尾的
[root@student ~]# grep "\.$" xuliangwei.txt
//精确匹配到
[root@student ~]# grep -o "8*" xuliangwei.txt
//匹配有abc的行
[root@student ~]# grep "[abc]" xuliangwei.txt
//匹配数字所在的行"[^0-9]"
[root@student ~]# grep "[0-9]" xuliangwei.txt
//匹配所有小写字母[^a-z]
[root@student ~]# grep "[a-z]" xuliangwei.txt
//重复0三次
[root@student ~]# grep "8\{3\}" xuliangwei.txt
//重复3个000不用转义符
[root@student ~]# grep -E "8{3}" xuliangwei.txt
//重复数字8, 3-5次
[root@student ~]# grep -E "8{3,5}" test.txt
//至少1次或1次以上
[root@student ~]# grep -E "8{1,}" xuliangwei.txt
```

3.sed文本处理

sed是一个流编辑器, 非交互式的编辑器,它一次处理一行内容.
处理时，把当前处理的行存储在临时缓冲区中，称为“模式空间”（pattern space）
接着用 sed 命令处理缓冲区中的内容，处理完成后， 把缓冲区的内容送往屏幕。
接着处理下一行，这样不断重复，直到文件末尾。
文件内容并没有改变，除非你 使用重定向存储输出。
Sed 要用来自动编辑一个或多个文件；简化对文件的反复操作；编写转换程序等。

sed命令格式

`sed [options] 'command' file(s)`

sed正则使用

> 与 grep一样，sed 在文件中查找模式时也可以使用正则表达式(RE)和各种元字符。
> 正则表达式是括在斜杠间的模式，用于查找和替换，以下是sed支持的元字符。
> 使用基本元字符集 ' $, ., *, [], [], < >, (), {}
> 使用扩展元字符集 ?, +, { }, |, ( )
> 使用扩展元字符的方式 + sed -r

sed命令示例

`sed` 对指定行进行操作，包括打印、删除、修改、追加等。

sed选项参数

> -e 允许多项编辑
> -n 取消默认的输出
> -i 直接修改对应文件
> -r 支持扩展元字符

`sed` 命令参数

> a 在当前行后添加一行或多行
> c 在当前行进行替换修改
> d 在当前行进行删除操作
> i 在当前行之前插入文本
> p 打印匹配的行或指定行
> n 读入下一输入行，从下一条命令进行处理
> ! 对所选行以外的所有行应用命令
> h 把模式空间里的内容重定向到暂存缓冲区
> H 把模式空间里的内容追加到暂存缓冲区
> g 取出暂存缓冲区的内容，将其复制到模式空间，覆盖该处原有内容
> G 取出暂存缓冲区的内容，将其复制到模式空间，追加在原有内容后面

多重编辑选项 `e`

```
//先删除行,然后管道给后面的sed进行替换
[root@Shell ~]# sed '1,9d' passwd |sed 's#root#alex#g'
```

```
//使用-e进行多次编辑修改操作
[root@Shell ~]# sed -e '1,9d' -e 's#root#alex#g' passwd
```

打印命令 `p`

```
//打印匹配halt的行
[root@Shell ~]# sed -n  '/halt/p' passwd
halt:x:7:0:halt:/sbin:/sbin/halt
//打印第二行的内容
[root@Shell ~]# sed -n  '2p' passwd
bin:x:1:1:bin:/bin:/sbin/nologin
//打印最后一行
[root@Shell ~]# sed -n  '$p' passwd
```

追加命令 `a`

```
//给30行添加配置 \t tab键(需要转义) \n 换行符
[root@Shell ~]# sed -i  '30a listen 80;' passwd
```

修改命令 `c`

```
//指定某行进行内容替换
[root@Shell ~]# sed  -i '7c SELINUX=Disabled' /etc/selinux/config
//正则匹配对应内容，然后进行替换
sed -i '/^SELINUX=/c SELINUX=Disabled' /etc/selinux/config

//非交互式修改指定的配置文件
[root@Shell ~]# sed -ri '/UseDNS/cUseDNS no' /etc/ssh/sshd_config
[root@Shell ~]# sed -ri '/GSSAPIAuthentication/c#GSSAPIAuthentication no' /etc/ssh/
sshd_config
[root@Shell ~]# sed -ri '/^SELINUX=/cSELINUX=disabled' /etc/selinux/config
```

删除命令 `d`

```
//指定删除第三行，但不会改变文件内容
[root@Shell ~]# sed '3d' passwd
[root@Shell ~]# sed '3{d}' passwd
//从第三行删除到最后一行
[root@Shell ~]# sed '3,$d' passwd
//删除最后一行
[root@Shell ~]# sed '$d' passwd
//删除所有的行
[root@Shell ~]# sed '1,$d' passwd
```

```
//匹配正则进行该行删除
[root@Shell ~]# sed /mail/d passwd
```

插入命令 `i`

```
//在文件的某一行上面添加内容
[root@Shell ~]# sed -i  '30i listen 80;' passwd
```

写文件命令 `w`

```
//将匹配到的行写入到新文件中
[root@Shell ~]# sed -n '/root/w newfile' passwd

//将passwd文件的第二行写入到newfile中
[root@Shell ~]# sed -n '2w newfile' passwd
```

获取下一行命令 `n`

```
//匹配root的行，删除root行的下一列
[root@Shell ~]# sed '/root/{n;d}' passwd

//替换匹配root行的下一列
[root@Shell ~]# sed '/root/{n; s/bin/test/}' passwd
```

暂存和取用命令 `h H g G`

```
//将第一行的写入到暂存区，替换最后一行的内容
[root@Shell ~]# sed  '1h;$g' /etc/hosts

//将第一行的写入到暂存区，在最后一行调用暂存区的内容
[root@Shell ~]# sed  '1h;$G' /etc/hosts

//将第一行的内容删除但保留至暂存区，在最后一行调用暂存区内容追加至于尾部
[root@Shell ~]# sed -r '1{h;d};$G' /etc/hosts

//将第一行的内容写入至暂存区，从第二行开始进行重定向替换
[root@Shell ~]# sed -r '1h;2,$g' /etc/hosts

//将第一行重定向至暂存区，2-3行追加至暂存区，最后追加调用暂存区的内容
[root@Shell ~]# sed -r '1h; 2,3H; $G' /etc/hosts
```

反向选择命令 `!`

```
//除了第三行,其他全部删除
[root@Shell ~]# sed -r '3!d' /etc/hosts
```

sed匹配替换

> s 替换命令标志
> g 行内全局替换
> i 忽略替换大小写

替换命令 `s`

```
//替换每行出现的第一个root
[root@Shell ~]# sed 's/root/alice/' passwd
//替换以root开头的行
[root@Shell ~]# sed 's/^root/alice/' passwd
//查找匹配到的行, 在匹配的行后面添加内容
[root@Shell ~]# sed -r 's/[0-9][0-9]$/& .5/' passwd

//匹配包含有root的行进行替换
[root@Shell ~]# sed -r 's/root/alice/g' passwd
//匹配包含有root的行进行替换,忽略大小写
# sed -r 's/root/alice/gi' /etc/passwd

//后向引用
[root@Shell ~]# sed -r 's#(Roo)#\1-alice#g' passwd
[root@Shell ~]# ifconfig eth0|sed -n '2p'|sed -r 's#(^.*et) (.*) (net.*$)#\2#g'

//示例
[root@bgx ~]# vim a.txt
/etc/abc/456
etc
//删除文本中的内容,需加转义
[root@Shell ~]# sed -r '\/etc\/abc\/456/d' a.txt
//如果碰到/符号, 建议使用#符替换
[root@Shell ~]# sed -r 's#/etc/abc/456#/dev/null#g' a.txt
[root@Shell ~]# sed -r 's@/etc/abc/456@/dev/null@' a.txt
```

删除文件

```
//删除配置文件中#号开头的注释行, 如果碰到tab或空格是无法删除
[root@Shell ~]# sed '/^#/d' file
```

```
//删除配置文件中含有tab键的注释行
[root@Shell ~]# sed -r '/^[ \t]*#/d' file
//删除无内容空行
[root@Shell ~]# sed -r '/^[ \t]*$/d' file

//删除注释行及空行
[root@Shell ~]# sed -r '/^[ \t]*#/d; /^[ \t]*$/d' /etc/vsftpd/vsftpd.conf
[root@Shell ~]# sed -r '/^[ \t]*#|^[ \t]*$/d' /etc/vsftpd/vsftpd.conf
[root@Shell ~]# sed -r '/^[ \t]*($|#)/d' /etc/vsftpd/vsftpd.conf
```

给文件行添加注释

```
//将第二行到第六行加上注释信息
[root@Shell ~]# sed '2,6s/^/#/' passwd

//将第二行到第六行最前面添加#注释符
[root@Shell ~]# sed -r '2,6s/.*/#&/' passwd

//添加#注释符
[root@Shell ~]# sed -r '3,$ s/^#*/#/' passwd
# sed -r '30,50s/^[ \t]*#*/#/' /etc/nginx.conf
# sed -r '2,8s/^[ \t#]*/#/' /etc/nginx.conf
```

4.Awk文本处理

`awk` 是一种编程语言，用于在 `linux/unix` 下对文本和数据进行处理。
`awk` 数据可以来自标准输入、一个或多个文件，或其它命令的输出。
`awk` 通常是配合脚本进行使用, 是一个强大的文本处理工具。

awk 的处理文本和数据的方式如下:

> 1.进行逐行扫描文件, 从第一行到最后一行
> 2.寻找匹配的特定模式的行,在行上进行操作
> 3.如果没有指定处理动作,则把匹配的行显示到标准输出
> 4.如果没有指定模式，则所有被操作的行都被处理

awk 的两种形式语法格式

```
awk [options] 'commands' filenames
awk [options] -f awk-script-file filenames

options
```

```
-F 定义输入字段分隔符，默认的分隔符是空格或tab键

command
BEGIN{}        {}       END{}
行处理前        行处理     行处理后

[root@Shell ~]# awk 'BEGIN{print 1/2} {print "ok"} END {print "Game Over"}'  /etc/h
osts
0.5
ok
ok
ok
Game Over
```

awk命令格式

```
//awk 'pattern' filename 匹配文件
[root@Shell ~]# awk '/root/' /etc/passwd

//awk '{action}' filename 对文件行进行动作处理
[root@Shell ~]# awk -F: '{print $1}' /etc/passwd

//awk 'pattern {action}' filename 匹配+处理动作
[root@Shell ~]# awk -F ':' '/root/ {print $1,$3}' /etc/passwd
[root@Shell ~]# awk 'BEGIN{FS=":"} /root/{print $1,$3}' /etc/passwd

//command |awk 'pattern {action}' //判断大于多少则输出什么内容
[root@Shell ~]# df |awk '/\/$/ {if ($3>50000) print $4}'
```

Awk 工作原理

`# awk -F: '{print $1,$3}' /etc/passwd`

> 1.awk使用一行作为输入，并将这一行赋给内部变量`$0`每一行也可称为一个记录，以换行符结束
> 2.awk进行字段分解，每个字段存储在已编号的变量中，从`$1`开始
> 3.awk默认情况下的分隔符是空格，是由内部变量`FS`来确定字段分隔符。初始`FS`为空格
> 4.awk打印字段,将以设置的方法使用`print`函数打印
> 5.awk在打印的字段间加上空格，因为`$1,$3`之间有一个逗号。逗号它映射为另一个内部变量，称为输出字段分隔符`OFS`默认为空格
> 6.awk输出之后，将从文件中获取另一行，并将其存储在`$0`中，覆盖原来的内容，然后将新的字符串分隔成字段并进行处理。该过程将持续到所有行处理完毕

Awk内部变量

`$0` 保存当前记录的内容

```
[root@Shell ~]# awk '{print $0}' /etc/passwd
```

`NR` 控制输入的总行数

```
//打印整个文本的行号
[root@Shell ~]# awk '{print NR,$0}' /etc/passwd
//打印文件的前三行
[root@Shell ~]# awk 'NR<=3' /etc/passwd
```

`FNR` 记录输入文件的编号

```
[root@Shell ~]# awk '{print FNR,$0}' /etc/passwd /etc/hosts
```

`NF` 保存行的最后一列内容

```
[root@Shell ~]# awk -F ":" '{print $1,$NF}' /etc/passwd
```

`FS` 指定字段分割符,默认空格

```
//以冒号作为字段分隔符
[root@Shell ~]# awk -F: '/root/{print $1, $3}' /etc/passwd
[root@Shell ~]# awk 'BEGIN{FS=":"} {print $1,$3}' /etc/passwd
//以空格冒号tab作为字段分割
[root@Shell ~]# awk -F'[ :\t]' '{print $1,$2,$3}' /etc/passwd
```

`OFS` 输出字段分隔符

```
//,映射为OFS,初始情况下OFS变量是空格
[root@Shell ~]# awk -F: '/root/{print $1,$2,$3,$4}' /etc/passwd
[root@Shell ~]# awk 'BEGIN{FS=":"; OFS="+++"} /^root/{print $1,$2,}' /etc/passwd
```

`RS` 输入记录分隔符，默认为换行符

```
[root@Shell ~]# awk -F: 'BEGIN{RS=" "} {print $0}' /etc/hosts
```

`ORS` 将文件每一行合并为一行,以空格为分割

```
[root@Shell ~]# awk -F: 'BEGIN{ORS=" "} {print $0}' /etc/hosts

//通常情况下让输出分隔符为换行符，然后依次打印响应的字段
[root@Shell ~]# awk -F ":" 'BEGIN{OFS="\n"}{print $1,$2,$3,$4,$5,$6,$7}' passwd
//将RS记录值标记为冒号分割，打印后相当于将每行的内容进行切割
[root@Shell ~]# awk 'BEGIN{RS=":"}{print $0}' passwd
```

`print` 格式化输出函数

```
[root@Shell ~]# date|awk '{print $2,"5月份""\n",$NF,"今年"}'
[root@Shell ~]# awk -F: '{print "用户是:" $1 "\t 用户uid: " $3  "\t 用户gid:" $4}' /e
tc/passwd

printf 函数
[root@Shell ~]# awk -F: '{printf "%-15s %-10s %-15s\n", $1, $2, $3}' /etc/passwd
%s 字符类型
%d 数值类型
占 15 字符
- 表示左对齐，默认是右对齐
printf 默认不会在行尾自动换行，加\n
```

Awk模式动作

awk语句都由模式和动作组成。
模式部分决定动作语句何时触发及触发事件。
如果省略模式部分，动作将时刻保持执行状态。模式可以是条件语句或复合语句或正则表达式。

1.正则表达式

```
//匹配记录（整行）
[root@Shell ~]# awk '/^root/' /etc/passwd
[root@Shell ~]# awk '$0 ~ /^root/' /etc/passwd

//匹配字段：匹配操作符（~ !~）
[root@Shell ~]# awk '!/^root/' /etc/passwd
[root@Shell ~]# awk '$0 ~ !/^root/' /etc/passwd
```

2.比较表达式

比较表达式采用对文本进行比较，只有当条件为真，才执行指定的动作。
比较表达式使用关系运算符，用于比较数字与字符串。

关系运算符

```
运算符      含义       示例
<        小于       x<y
<=       小于或等于    x<=y
==       等于          x==y
!=       不等于      x!=y
>=       大于等于        x>=y
>        大于       x>y


//uid为0的列出来
[root@Shell ~]# awk -F ":" '$3==0' /etc/passwd
//uid小于10的全部列出来
[root@Shell ~]# awk -F: '$3 < 10' /etc/passwd
//用户登陆的shell等于/bin/bash
[root@Shell ~]# awk -F: '$7 == "/bin/bash" ' /etc/passwd
//第一列为alice的列出来
[root@Shell ~]# awk -F: '$1 == "alice" ' /etc/passwd
//为alice的用户列出来
[root@Shell ~]# awk -F: '$1 ~ /alice/ ' /etc/passwd
[root@Shell ~]# awk -F: '$1 !~ /alice/ ' /etc/passwd
//磁盘使用率大于多少则，则打印可用的值
[root@Shell ~]# df |awk '/\/$/'|awk '$3>1000000 {print $4}'
```

3.条件表达式

```
[root@Shell ~]# awk -F: '$3>300 {print $0}' /etc/passwd
[root@Shell ~]# awk -F: '{if($3>300) print $0}' /etc/passwd
[root@Shell ~]# awk -F: '{if($3>5555){print $3} else {print $1}}' /etc/passwd
```

4.运算表达式

```
[root@Shell ~]# awk -F: '$3 * 10 > 500000' /etc/passwd
[root@Shell ~]# awk -F: 'BEGIN{OFS="--"} { if($3*10>50000) {print $1,$3} } END {pri
nt "打印ok"}' /etc/passwd
[root@Shell ~]# awk '/southem/{print $5 + 10}' datafile
[root@Shell ~]# awk '/southem/{print $5 + 10.56}' datafile
[root@Shell ~]# awk '/southem/{print $8 - 10}' datafile
[root@Shell ~]# awk '/southem/{print $8 / 2 }' datafile
[root@Shell ~]# awk '/southem/{print $8 * 2 }' datafile
```

```
[root@Shell ~]# awk '/southem/{print $8 % 2 }' datafile
```

5.逻辑操作符和复合模式

```
&&逻辑与 || 逻辑或  !逻辑非

//匹配用户名为root并且打印uid小于15的行
[root@Shell ~]# awk -F: '$1~/root/ && $3<=15' /etc/passwd
//匹配用户名为root或uid大于5000
[root@Shell ~]# awk -F: '$1~/root/ || $3>=5000' /etc/passwd
```

awk示例1

```
# awk '/west/' datafile
# awk '/^north/' datafile
# awk '$3 ~ /^north/' datafile
# awk '/^(no|so)/' datafile
# awk '{print $3,$2}' datafile
# awk '{print $3 $2}' datafile
# awk '{print $0}' datafile
# awk '{print "Number of fields: "NF}' datafile
# awk '/northeast/{print $3,$2}' datafile
# awk '/^[ns]/{print $1}' datafile
# awk '$5 ~ /\. [7-9]+/' datafile
# awk '$2 !~ /E/{print $1,$2}' datafile
# awk '$3 ~ /^Joel/{print $3 "is a nice boy."}' datafile
# awk '$8 ~ /[0-9][0-9]$/{print $8}' datafile
# awk '$4 ~ /Chin$/{print "The price is $" $8 "."}' datafile
# awk '/Tj/{print $0}' datafile
# awk -F: '{print "Number of fields: "NF}' /etc/passwd
# awk -F"[ :]" '{print NF}' /etc/passwd
```

awk示例2

```
[root@Shell ~]# cat b.txt
bgx xuliangwei:is a:good boy!

[root@Shell ~]# awk '{print NF}' b.txt
4
[root@Shell ~]# awk -F ':' '{print NF}' b.txt
3
[root@Shell ~]# awk -F"[ :]" '{print NF}' b.txt
6
```

Awk条件判断

> if语句格式:{ if(表达式) {语句;语句;... } }

```
//打印当前管理员用户名称
[root@Shell ~]# awk -F: '{ if($3==0){print $1 "is adminisitrator"} }' /etc/passwd
//统计系统用户数量
[root@Shell ~]# awk -F: '{ if($3>0 && $3<1000){i++}} END {print i}' /etc/passwd
//统计普通用户数量
[root@Shell ~]# awk -F: '{ if($3>1000){i++}} END {print i}' /etc/passwd
```

> if...else 语句格式: {if(表达式) {语句;语句;... } else{语句;语句;...}}

```
# awk -F: '{if($3==0){print $1} else {print $7}}' /etc/passwd
# awk -F: '{if($3==0) {count++} else{i++} }' /etc/passwd
# awk -F: '{if($3==0){count++} else{i++}} END{print " 管理员个数: "count ; print "
系统用户数: "i}' /etc/passwd
```

if...else if...else 语句格式:

> {if(表达式 1) {语句;语句; ... } else if(表达式 2) {语句;语句; . .. } else {语句;语句; ... } }

```
[root@Shell ~]# awk -F: '{ if($3==0){i++} else if($3>0 && $3<1000){j++} else if($3>
1000) {k++}} END {print i;print j;print k}' /etc/passwd
[root@Shell ~]# awk -F: '{ if($3==0){i++} else if($3>0 && $3<1000){j++} else if($3>
1000) {k++}} END {print "管理员个数"i; print "系统用户个数" j; print "系统用户个 数" }
' /etc/passwd
管理员个数1
系统用户个数29
系统用户个数69
```

Awk循环语句

while循环

```
[root@Shell ~]# awk 'BEGIN{ i=1; while(i<=10){print i; i++} }'
[root@Shell ~]# awk -F: '{i=1; while(i<=NF){print $i; i++}}' /etc/passwd
[root@Shell ~]# awk -F: '{i=1; while(i<=10) {print $0; i++}}' /etc/passwd
[root@Shell ~]#cat b.txt
111 222
333 444 555
666 777 888 999
```

```
[root@Shell ~]# awk '{i=1; while(i<=NF){print $i; i++}}' b.txt
```

for循环

```
//C 风格 for
[root@Shell ~]# awk 'BEGIN{for(i=1;i<=5;i++){print i} }'
//将每行打印 10 次
[root@Shell ~]# awk -F: '{ for(i=1;i<=10;i++) {print $0} }' passwd
[root@Shell ~]# awk -F: '{ for(i=1;i<=10;i++) {print $0} }' passwd
[root@Shell ~]# awk -F: '{ for(i=1;i<=NF;i++) {print $i} }' passwd
```

awk数组实战

```
[root@Shell ~]# awk -F: '{username[++i]=$1} END{print username[1]}' /etc/passwd
[root@Shell ~]# awk -F: '{username[i++]=$1} END{print username[1]}' /etc/passwd
[root@Shell ~]# awk -F: '{username[i++]=$1} END{print username[0]}' /etc/passwd
```

> 注意:将需要统计的某个字段作为数组的索引，最后对索引进行遍历

1.按索引遍历

```
[root@Shell ~]# awk -F: '{username[x++]=$1} END{for(i in username) {print i,usernam
e[i]} }' /etc/passwd
[root@Shell ~]# awk -F: '{username[++x]=$1} END{for(i in username) {print i,usernam
e[i]} }' /etc/passwd
```

1.统计/etc/passwd 中各种类型 shell 的数量

```
# awk -F: '{shells[$NF]++} END{ for(i in shells){print i,shells[i]} }' /etc/passwd
```

2.网站访问状态统计<当前时实状态ss>

```
[root@Shell ~]# ss -an|awk '/:80/{tcp[$2]++} END {for(i in tcp){print i,tcp[i]}}'
```

3.统计当前访问的每个IP的数量<当前时实状态 netstat,ss>

```
[root@Shell ~]# ss -an|awk -F ':' '/:80/{ips[$(NF-1)]++} END {for(i in ips){print i
,ips[i]}}'
```

Awk数组案例

`Nginx`日志分析，日志格式如下：

```
log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                  '$status $body_bytes_sent "$http_referer" '
                  '"$http_user_agent" "$http_x_forwarded_for"';

52.55.21.59 - - [25/Jan/2018:14:55:36 +0800] "GET /feed/ HTTP/1.1" 404 162 "https:/
/www.google.com/" "Opera/9.80 (Macintosh; Intel Mac OS X 10.6.8; U; de) Presto/2.9.
168 Version/11.52" "-"
```

1.统计2018年01月25日,当天的PV量

```
[root@Shell ~]# grep "25/Jan/2018" log.bjstack.log |wc -l
[root@Shell ~]# awk "/25\/Jan\/2018/" log.bjstack.log |wc -l
[root@Shell ~]# awk '/25\/Jan\/2018/ {ips[$1]++} END {for(i in ips) {sum+=ips[i]} {
print sum}}' log.bjstack.log
//统计15-19点的pv量
[root@Shell ~]# awk '$4>="[25/Jan/2018:15:00:00" && $4<="[25/Jan/2018:19:00:00 {pri
nt $0}"' log.bjstack.log |wc -l
```

2.统计2018年01月25日,一天内访问最多的10个IP

```
[root@Shell ~]# awk '/25\/Jan\/2018/ {ips[$1]++} END {for(i in ips){ print ips[i],i
}}' log.bjstack.log |sort -rn|head
//统计15-19点访问次数最多的10个IP
[root@Shell ~]# awk '$4>="[25/Jan/2018:15:00:00" && $4<="[25/Jan/2018:19:00:00"' lo
g.bjstack.log |awk '{ips[$1]++} END {for(i in ips){print ips[i],i}}'|sort -rn|head
```

3.统计2018年01月25日,访问大于100次的IP

```
[root@Shell ~]# awk '/25\/Jan\/2018/ {ips[$1]++} END {for(i in ips){if(ips[i]>10){p
rint i,ips[i]}}}' log.bjstack.log
```

4.统计2018年01月25日,访问最多的10个页面($request top 10)

```
[root@Shell ~]# awk '/25\/Jan\/2018/ {request[$7]++} END {for(i in request){print r
equest[i],i}}' log.bjstack.log |sort -rn|head
```

5.统计2018年01月25日,每个URL访问内容总大小($body_bytes_sent)

```
[root@Shell ~]# awk '/25\/Jan\/2018/ {request[$7]++;size[$7]+=$10} END {for(i in re
quest){print request[i],i,size[i]}}' log.bjstack.log |sort -rn|head
```

6.统计2018年01月25日,每个IP访问状态码数量($status)

```
[root@Shell ~]# awk '{ip_code[$1 " " $9]++} END {for(i in ip_code){print ip_code[i]
,i}}' log.bjstack.log|sort -rn|head
```

7.统计2018年01月25日,访问状态码为404及出现的次数($status)

```
[root@Shell ~]# grep "404" log.bjstack.log |wc -l
[root@Shell ~]# awk '{if($9=="404") code[$9]++} END {for(i in code){print i,code[i]
}}' log.bjstack.log
```

8.统计2018年01月25日,8:30-9:00访问状态码是404

```
[root@Shell ~]# awk '$4>="[25/Jan/2018:15:00:00" && $4<="[25/Jan/2018:19:00:00" &&
$9=="404" {code[$9]++} END {for(i in code){print i,code[i]}}' log.bjstack.log
[root@Shell ~]# awk '$9=="404" {code[$9]++} END {for(i in code){print i,code[i]}}'
log.bjstack.log
```

9.统计2018年01月25日,各种状态码数量

```
[root@Shell ~]# awk '{code[$9]++} END {for(i in code){print i,code[i]}}' log.bjstac
k.log

[root@Shell ~]# awk '{if($9>=100 && $9<200) {i++}
else if ($9>=200 && $9<300) {j++}
else if ($9>=300 && $9<400) {k++}
else if ($9>=400 && $9<500) {n++}
else if($9>=500) {p++}}
END{print i,j,k,n,p,i+j+k+n+p}' log.bjstack.log

[root@Shell ~]# awk '{if($9>=100 && $9<200) {i++}
else if ($9>=200 && $9<300) {j++}
else if ($9>=300 && $9<400) {k++}
else if ($9>=400 && $9<500) {n++}
else if($9>=500) {p++}}
END{print i?i:0,j?j:0,k?k:0,n?n:0,p?p:0,i+j+k+n+p}' log.bjstack.log
```