三剑客之 shell 篇

快捷键：

Ctrl + 1　　标题 1

Ctrl + 2　　标题 2

Ctrl + 3　　标题 3

Ctrl + 4　　实例

Ctrl + 5　　程序代码

Ctrl + 6　　正文

格式说明：

蓝色字体：注释

黄色背景：重要

绿色背景：注意

老男孩linux运维实战培训

老男孩教育教学核心思想 6 重：重目标、重思路、重方法、重实践、重习惯、重总结

学无止境，老男孩教育成就你人生的起点！

目 录

第1章 shell 正则应用

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。

1.1 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
.*	匹配所有
[ ]	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]
```

1.2 grep 正则表达式实战

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


I like badminton ball ,billiard ball and chinese chess!
my blog is http: blog.51cto.com
our site is http:www.lizhenya.com
my qq num is 593528156
not 572891888887.
```

1.2.1 过滤以 m 开头的行

```
[root@Shell ~]# grep "^m" test.txt
my blog is http: blog.51cto.com
my qq num is 572891887.
```

1.2.2 过滤以 m 结尾的行

```
[root@Shell ~]# grep "m$" test.txt
my blog is http: blog.51cto.com
our site is http:www.lizhenya.com
```

1.2.3 排除空行，并打印行号

```
[root@student ~]# grep -vn "^$" lizhenya.txt
```

1.2.4 匹配任意一个字符,不包括空行

```
[root@student ~]# grep "." lizhenya.txt
```

1.2.5 匹配所有

```
[root@student ~]# grep ".*" lizhenya.txt
```

1.2.6 匹配单个任意字符

```
[root@node1 ~]# grep "lizhen.a" lizhenya.txt
```

1.2.7 以点结尾的

```
[root@student ~]# grep "\.$" lizhenya.txt
```

1.2.8 精确匹配到

```
[root@student ~]# grep -o "8*" lizhenya.txt
```

1.2.9 匹配有 abc 的行

```
[root@student ~]# grep "[abc]" lizhenya.txt
```

1.2.10 匹配数字所在的行"[0-9]"

```
[root@student ~]# grep "[0-9]" lizhenya.txt
```

1.2.11 匹配所有小写字母[a-z]

```
[root@student ~]# grep "[a-z]" lizhenya.txt
```

1.2.12 重复 8 三次

```
[root@student ~]# grep "8\{3\}" lizhenya.txt
```

1.2.13 重复数字 8, 3-5 次

```
[root@student ~]# grep -E "8{3,5}" test.txt
```

1.2.14 至少 1 次或 1 次以上

```
[root@student ~]# grep -E "8{1,}" lizhenya.txt
```

1.3 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。

1.3.1 sed 命令格式

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

1.3.2 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

1.3.3 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

1.3.4 sed`内置命令参数

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

1.3.5 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#oldboy#g'
```

1.3.6 使用-e 进行多次编辑修改操作

```
[root@Shell ~]# sed -e '1,9d' -e 's#root#oldboy#g' passwd
```

1.3.7 打印命令 p

1.3.7.1 打印匹配 halt 的行

```
[root@Shell ~]# sed -n  '/halt/p' passwd
```

1.3.7.2 打印第二行的内容

```
[root@Shell ~]# sed -n  '2p' passwd
bin:x:1:1:bin:/bin:/sbin/nologin
```

1.3.7.3 打印最后一行

```
[root@Shell ~]# sed -n  '$p' passwd
```

1.3.8 追加命令`a`

1.3.8.1 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
```

1.3.9 修改命令`c`

1.3.9.1 指定某行进行内容替换

```
[root@Shell ~]# sed  -i '7c SELINUX=Disabled' /etc/selinux/config
```

1.3.9.2 正则匹配对应内容, 然后进行替换

```
[root@Shell ~]# sed -i '/^SELINUX=/c SELINUX=Disabled' /etc/selinux/config
```

1.3.9.3 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
```

1.3.10 删除命令`d`

1.3.10.1 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
```

1.3.10.2 从第三行删除到最后一行

```
[root@Shell ~]# sed '3,$d' passwd
```

1.3.10.3 删除最后一行

```
[root@Shell ~]# sed '$d' passwd
```

1.3.10.4 删除所有的行

```
[root@Shell ~]# sed '1,$d' passwd
```

1.3.10.5 匹配正则进行该行删除

```
[root@Shell ~]# sed /mail/d passwd
```

1.3.11 **插入命令`i`**

1.3.11.1 在文件的某一行上面添加内容

```
[root@Shell ~]# sed -i  '30i listen 80;' passwd
```

1.3.12 **写文件命令`w`**

1.3.12.1 将匹配到的行写入到新文件中

```
[root@Shell ~]# sed -n '/root/w newfile' passwd
```

1.3.12.2 将 passwd 文件的第二行写入到 newfile 中

```
[root@Shell ~]# sed -n '2w newfile' passwd
```

1.3.13 **获取下一行命令`n`**

1.3.13.1 匹配 root 的行, 删除 root 行的下一列

```
[root@Shell ~]# sed '/root/{n;d}' passwd
```

1.3.13.2 替换匹配 root 行的下一列

```
[root@Shell ~]# sed '/root/{n; s/bin/test/}' passwd
```

1.3.14 **暂存和取用命令`h H g G`**

g: 将 hold space 中的内容拷贝到 pattern space 中，原来 pattern space 里的内容被覆盖

G：将 hold space 中的内容 append 到 pattern space\n 后

h: 将 pattern space 中的内容拷贝到 hold space 中，原来 hold space 里的内容被覆盖

H: 将 pattern space 中的内容 append 到 hold space\n 后

d: 删除 pattern 中的所有行，并读入下一新行到 pattern 中

D: 删除 multiline pattern 中的第一行，不读入下一行

1.3.14.1 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
```

1.3.14.2 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
```

1.3.14.3 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
```

1.3.14.4 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
```

1.3.14.5 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
```

4 图解 file

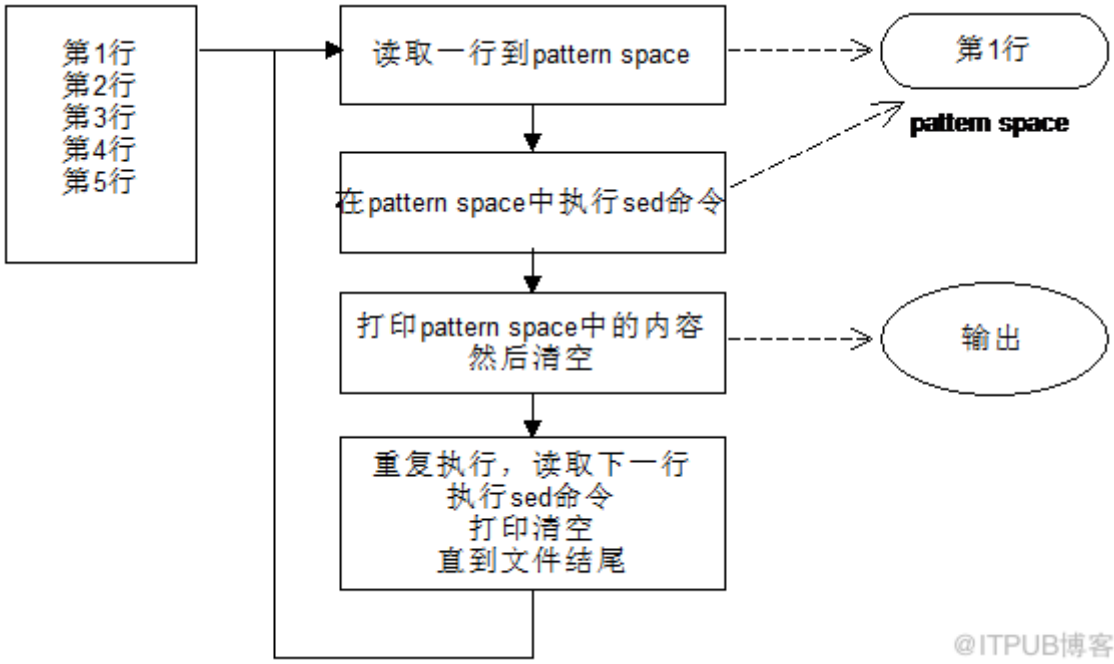
1 sed 简介

sed 是面向流的行编辑器。所谓面向流，是指接受标准输入的输入，输出内容到标准输出上。sed 编辑器逐行处理文件（或输入），并将结果发送到屏幕。

具体过程如下： sed 将处理的行读入到一个临时缓存区中（也称为模式空间 pattern space），sed 中的命令依次执行，直到所有命令执行完毕，完成后把该行发送到屏幕上，清理 pattern space 中的内容；接着重复刚才的动作，读入下一行，直到文件处理结束。

sed 每处理完一行就将其从 pattern space 中删除，然后将下一行读入，进行处理和显示。处理完输入 文件的最后一行后，sed 便结束运行。sed 把每一行都存在临时缓冲区中，对这个副本进行编辑，所以不会修改原文件。

2 sed 执行流程图



3 什么是 Pattern Space,Hold Space

Pattern Space 相当于车间，sed 把流内容在这里进行处理，Hold Space 相当于仓库，加工的半成品在这里进行临时存储。

由于各种原因，比如用户希望在某个条件下脚本中的某个命令被执行，或者希望模式空间得到保存以便下一次处理，都有可能使得 sed 在处理文件的时候不按照正常的流程来进行。这个时候，sed 设置了一些高级命令来满足用户的要求。

一些高级命令

g: 将 hold space 中的内容拷贝到 pattern space 中， 原来 pattern space 里的内容被覆盖

G：将 hold space 中的内容 append 到 pattern space\n 后

h: 将 pattern space 中的内容拷贝到 hold space 中，原来 hold space 里的内容被覆盖

H: 将 pattern space 中的内容 append 到 hold space\n 后

d: 删除 pattern 中的所有行，并读入下一新行到 pattern 中

D: 删除 multiline pattern 中的第一行，不读入下一行

4 图解 sed '1!G;h;$!d' file

1!G 第一行不执行 G 命令，从第二行开始执行

$!d 最后一行不删除

[root@localhost test]# cat file

1 1 1

2 2 2

3 3 3

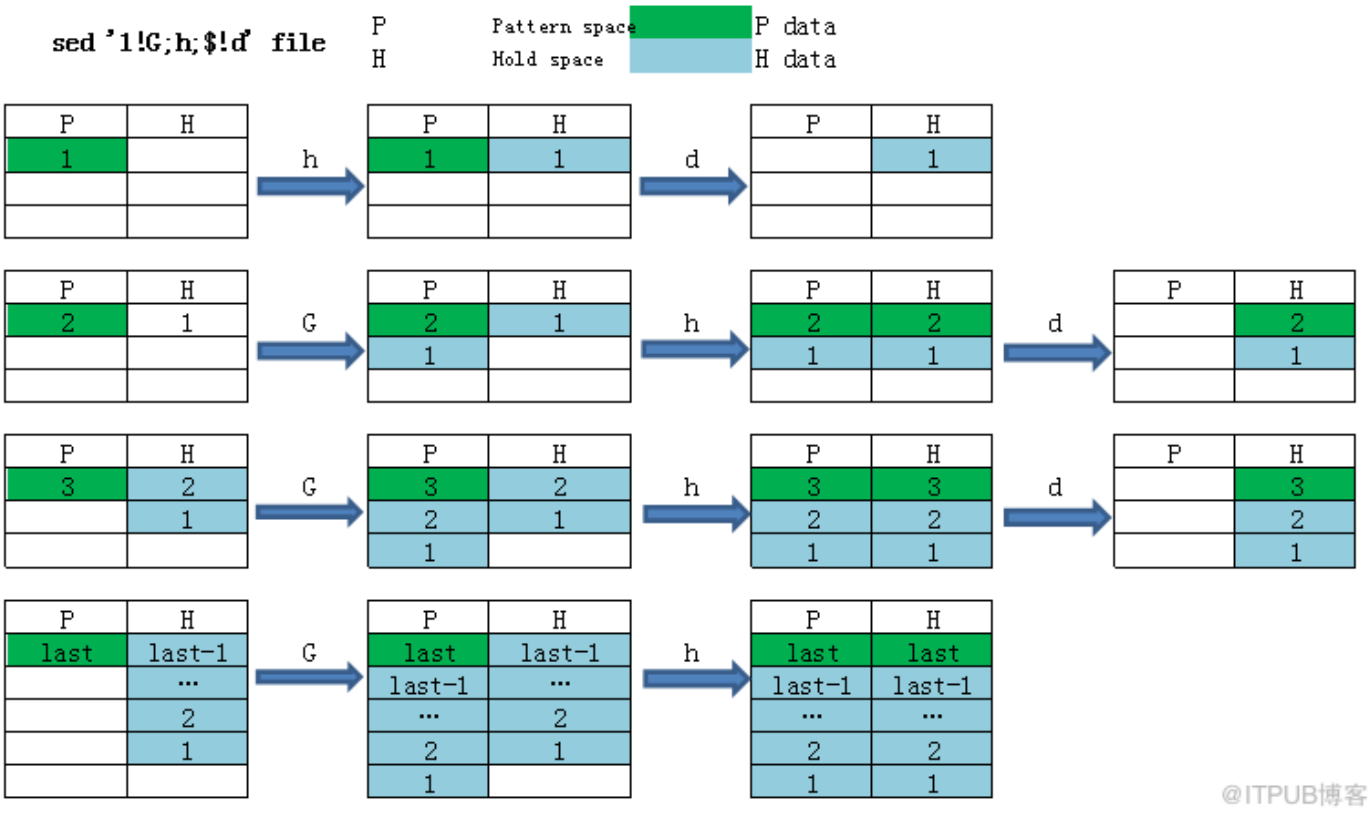
[root@localhost test]# sed '1!G;h;$!d' file

3 3 3

2 2 2

1 1 1

图中 P 代表 Pattern Space,H 代表 Hold Space。绿色代表 pattern space 中的数据，蓝色代表 hold space 中 的 数 据 。



1.3.15 反向选择命令`!`

1.3.15.1 除了第三行,其他全部删除

```
[root@Shell ~]# sed -r '3!d' /etc/hosts
```

1.3.16 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`
```

1.3.16.1 替换每行出现的第一个 root

```
[root@Shell ~]# sed 's/root/alice/' passwd
```

1.3.16.2 替换以 root 开头的行

```
[root@Shell ~]# sed 's/^root/alice/' passwd
```

1.3.16.3 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
```

1.3.16.4 匹配包含有 root 的行进行替换

```
[root@Shell ~]# sed -r 's/root/alice/g' passwd
```

1.3.16.5 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
```

1.3.16.6 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'


示例文件内容如下：
[root@lzy ~]# 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
```

1.3.17 删除文件

1.3.17.1 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
```

1.3.17.2 删除配置文件中含有 tab 键的注释行

```
[root@Shell ~]# sed -r '/^[ \t]*#/d' file
```

1.3.17.3 删除无内容空行

```
[root@Shell ~]# sed -r '/^[ \t]*$/d' file
```

1.3.17.4 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
```

1.3.18 给文件行添加注释

1.3.18.1 将第二行到第六行加上注释信息

```
[root@Shell ~]# sed '2,6s/^/#/' passwd
```

1.3.18.2 将第二行到第六行最前面添加#注释符

```
[root@Shell ~]# sed -r '2,6s/.*/#&/' passwd
```

1.3.19 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
```

1.4 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。
```

1.4.1 awk 的处理数据的方式

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

1.4.2 `awk`的语法格式

```
awk [options] 'commands' filenames

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

1.4.3 选项 options

```
-F 定义输入字段分隔符，默认的分隔符，空格或 tab 键


命令 command

行处理前       行处理   行处理后

BEGIN{}       {}        END{}


BEGIN 发生在读文件之前

[root@Shell ~]# awk 'BEGIN{print 1/2}'

0.5

BEGIN 在行处理前，修改字段分隔符

[root@Shell ~]# awk 'BEGIN{FS=":"} {print $1}' /etc/passwd

BEGIN 在行处理前，修改字段读入和输出分隔符

[root@Shell ~]# awk 'BEGIN{FS=":";OFS="---"} {print $1,$2}' /etc/passwd

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

0.5

ok

ok

ok
```

```
Game Over
```

1.4.4 `awk`命令格式

1.4.4.1 匹配 awk 'pattern' filename

```
[root@Shell ~]# awk '/root/' /etc/passwd
```

1.4.4.2 处理动作 awk '{action}' filename

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

1.4.4.3 匹配+处理动作 awk 'pattern {action}' filename

```
[root@Shell ~]# awk -F ':' '/root/ {print $1,$3}' /etc/passwd

[root@Shell ~]# awk 'BEGIN{FS=":"} /root/{print $1,$3}' /etc/passwd
```

1.4.4.4 判断大于多少则输出什么内容 command |awk 'pattern {action}'

```
[root@Shell ~]# df |awk '/\/$/ {if ($3>50000) print $4}'
```

1.4.5 Awk 工作原理

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

1.awk 将文件中的每一行作为输入，并将每一行赋给内部变量`$0`，以换行符结束

2.awk 开始进行字段分解，每个字段存储在已编号的变量中，从`$1`开始[默认空格分割]

3.awk 默认字段分隔符是由内部`FS`变量来确定，可以使用`-F`修订

4.awk 行处理时使用了`print`函数打印分割后的字段

5.awk 在打印后的字段加上空格，因为`$1,$3` 之间有一个逗号。逗号被映射至`OFS`内部变量中，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.
```

1.4.6 Awk 内部变量

```
*1.`$0`保存当前记录的内容*

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

2.`NR`记录输入总的编号(行号)

```
[root@Shell ~]# awk '{print NR,$0}' /etc/passwd
[root@Shell ~]# awk 'NR<=3' /etc/passwd
```

3.`FNR`当前输入文件的编号(行号)

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

4.`NF`保存行的最后一列

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

5.`FS`指定字段分割符，默认是空格

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
```

6.`OFS`指定输出字段分隔符

逗号映射为 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
```

7.`RS`输入记录分隔符，默认为换行符[了解]

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

8.`ORS`将文件以空格为分割每一行合并为一行[了解]

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

9.`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}' /etc/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

1.4.7 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 '$1~/^root/' /etc/passwd

[root@Shell ~]# awk '$NF !~ /bash$/' /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

[root@linuxnc ~]# awk -F: '{if($3==0){a++} else if($3>0&&$3<1000){b++} else{c++}}END{print "管理员数量: "a "\n 虚拟用户的数量: "b "\n 普通用户的数量是: "c}' passwd

管理员数量: 1

```
虚拟用户的数量: 24
普通用户的数量是: 1
```

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 {print "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
```

1.4.8 `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
```

1.4.9 awk 示例 2

```
[root@Shell ~]# cat b.txt
lzy lizhenya: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
```

1.4.10 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; ... } }

{if(表达式成立){执行的动作}else if(表达式成立{执行的动作} else{执行的动作})}

```
[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
```

1.4.11 Awk 循环语句

1.4.11.1 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
```

1.4.11.2 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<=NF;i++) {print $i} }' passwd
```

1.4.12 awk 数组概述

```
将需要统计的某个字段作为数组的索引，然后对索引进行遍历
```

1.4.12.1 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
```

1.4.12.2 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]}}'
```

1.4.12.3 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]}}'
```

1.4.13 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" "-"
```

1.4.13.1 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
```

1.4.13.2 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 {print $0}"'
log.bjstack.log |wc -l
```

1.4.13.3 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
```

1.4.13.4 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"' log.bjstack.log
|awk '{ips[$1]++} END {for(i in ips){print ips[i],i}}'|sort -rn|head
```

1.4.13.5 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){print
i,ips[i]}}}' log.bjstack.log
```

1.4.13.6 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 request[i],i}}'
log.bjstack.log |sort -rn|head
```

1.4.13.7 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 request){print
request[i],i,size[i]}}' log.bjstack.log |sort -rn|head
```

1.4.13.8 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
```

1.4.13.9 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
```

1.4.13.10 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
```

1.4.13.11 统计 2018 年 01 月 25 日,各种状态码数量，统计状态码出现的次数

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