awk 实战讲解

作者：张亚

归档：学习笔记

2021/04/13

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 awk 基础入门 1
1.1 awk 简介 1
1.2 学完 awk 你可以掌握 1
1.3 awk 环境简介 2
1.4 awk 的格式 2
1.5 模式动作 3
1.6 awk 执行过程 5
1.7 小结 awk 执行过程 6
1.8 记录和字段 6
1.8.1 记录(行) 7
1.8.2 awk 记录分隔符-RS 7
1.8.3 对$0 的认识 8
1.8.4 企业面试题:按单词出现频率降序排序(计算文件中每个单词的重复数量) 9
1.8.5 awk 记录知识小结 11
1.8.6 字段(列) 11
1.8.7 企业面试题:同时取出 oldboy 和 31333741 这两个内容(指定多分隔符) 13
1.8.8 ORS 与 OFS 简介 14
1.8.9 ORS 与 OFS 简介 15
1.9 awk 基础入门总结 16
第 2 章 awk 进阶 17
2.1 awk 模式与动作 17
2.2 正则表达式作为模式 18
2.3 awk 正则匹配操作符 19

2.4 awk 正则表达式匹配整行 19
2.5 awk 正则表达式匹配一行的某一列 19
2.6 某个区域中的开头和结尾 20
2.7 创建测试环境 20
2.8 测试文件说明 20
2.9 awk 正则表达式练习题 21
2.9.1 显示姓 Zhang 的人第二次捐款金额及她的名字 21
2.9.2 显示 Xiaoyu 的名字和 ID 号码 21
2.9.3 显示所有以 41 开头的 ID 号码的人的全名和 ID 号码 21
2.9.4 显示所有以一个 D 或 X 开头的人名全名 21
2.9.5 显示所有 ID 号码最后一位数字是 1 或 5 的人的全名 21
2.9.6 显示 Xiaoyu 的捐款,每个值是都已$开头的如:520$200$135 21
2.9.7 显示所有人的全名,以姓,名的格式显示如 Meng,Feixue 21
2.10 awk 正则表达式练习题-详解 21
2.10.1 显示姓 Zhang 的人的第二次捐款金额及她的名字 21
2.10.2 显示 Xiaoyu 的姓氏和 ID 号码 22
2.10.3 显示所有以 41 开头的 ID 号码的人的全名和 ID 号码 22
2.10.4 企业面试题: test.txt 内容如下,想输出不包含 oldboy 的行,仅仅是 oldboy,oldboylinux 正常输出: 22
2.10.5 找出所有人名的第一个字母是 D 或 X,显示她们的全名 23
2.10.6 显示所有 ID 号码最后一位数字是 1 或 5 的人的全名 23
2.10.7 显示 Xiaoyu 的捐款,每个值都有以$开头,如$520$200$135 24
2.10.8 显示所有人的全名,以名字,姓的格式显示如 Feixue,Meng 25
2.10.9 企业面试题:取出 eth0 的 ip 地址 26
2.10.10 企业案例 1:取出常用服务端口号 27
2.10.11 企业案例 2:取出常用服务端名称 28
2.10.12 正则表达式部分的疑问 28
2.11 比较表达式作为模式-需要一些例子 29
2.11.1 awk 变量导致的问题 29
2.11.2 企业面试题:取出/etc/services 的 23~30 行 30
2.11.3 如何判断某一列是否等于某个字符呢? 30

2.12 范围模式.....30
2.12.1 显示第二行到第五行的行号和整行的内容.....31
2.12.2 显示从以 bin 开头的行,到第五行中的行号和整行内容.....31
2.13 awk 特殊模式-BEGIN 模式与 END 模式.....32
2.13.1 BEGIN 模式.....32
2.13.2 第一个作用,内置变量的定义:去 eth0 的 ip 地址.....32
2.13.3 第二个作用,在读取文件之前,输出些提示性信息,表头.....33
2.13.4 第三种用法,使用 BEGIN 模块的特殊性质,进行一些测试.....33
2.13.5 第四种方法:配合 getline 读取文件,后面 awk 函数处讲解.....34
2.13.6 awk 中变量的概念简介.....34
2.13.7 END 模式.....34
2.13.8 企业案例 3:统计/etc/services 文件里面的空行数量.....35
2.13.9 通过/^$/{i=i+1;print i} 来查看 awk 执行过程.....36
2.13.10 企业面试题:文件 count.txt,文件内容是 1 到 100(由 seq 100 生成),请计算文件每行值加起来的结果(计算 1+.....+100).....37
2.13.11 awk 调试技巧:.....38
2.14 awk 中的动作.....38
2.14.1 awk 模式与动作小结:.....38
2.15 总结 awk 执行的过程.....39
2.16 awk 执行过程.....40
2.17 思想:.....40
2.18 [放到数组后面]awk 变量.....40
2.19 [放到数组后面]用户自定义变量.....41
2.19.1 举例:数组结构及创建.....41
2.19.2 举例:给数组赋值并打印.....42
2.19.3 也可以看下面的图片.....43
2.20 .企业面试题:统计域名访问次数.....44
2.21 总结:.....47

第 1 章 awk 基础入门

要弄懂 awk 程序,必须熟悉了解这个工具的规则,本书的目的是通过实际案例或面试题带大家熟练掌握 awk 在企业的用法名而不是 awk 程序的帮助手册

1.1 awk 简介

- 一种名字怪异的语言
- 模式扫描和处理==处理文本日志

awk 不仅仅是 linux 系统中的一个命令,而且是一种编程语言,可以用来处理数据和生成报告(excel),处理的数据可以是一个或多个文件,可以是来自标准输入,也可以通过关单获取标准输入,awk 可以在命令行上直接编辑命令进行操作,也可以编写成 awk 程序来进行更为复杂的运行,本章主要讲解 awk 命令行的运用.较为复杂的程序会包含在以后的书籍中.

1.2 学完 awk 你可以掌握

- 记录与字段
- 模式匹配:模式与动作
- 基本的 awk 执行过程
- awk 常用内置变量(预定义变量)
- awk 数组(工作常用)
- awk 语法:循环,条件
- awk 常用函数
- 向 awk 传递参数
- awk 引用 shell 变量
- awk 小程序及调试思路

1.3 awk 环境简介

本书涉及的 awk 为 gawk,即 GNU 版本的 awk.

```
[root@LSD ~]# cat /etc/redhat-release
CentOS release 6.8 (Final)
[root@LSD ~]# uname -r
2.6.32-642.13.1.el6.x86_64
[root@LSD ~]# ll $(which awk)
lrwxrwxrwx. 1 root root 4 Oct 11 23:42 /bin/awk -> gawk
[root@LSD ~]# awk --version
GNU Awk 3.1.7
```

1.4 awk 的格式

awk 指令是由模式,动作,或者模式和动作的组合组成.

模式即 pattern,可以类似理解成 sed 的模式匹配,可以由表达式组成,也可以使两个正斜杠之间的正则表达式.比如 NR==1,这就是模式,可以把他理解为一个条件.

动作即 action,是由在大括号里面的一条或多条语句组成,语句之间使用分号隔开,如下 awk 使用格式.

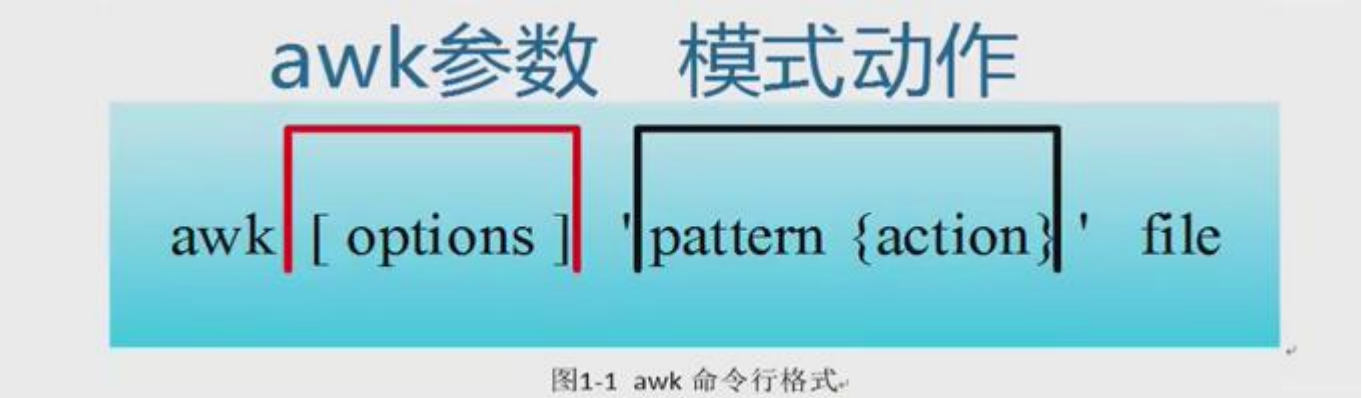


图1-1 awk 命令行格式

awk 处理的内容可以来自标准输入(<),一个或多个文本文件或管道.

pattern 即模式,可以理解为条件,也叫找谁,你找谁?高矮,胖瘦,男女?都是条件,即模式 action 即动作,可以

理解为干啥,找到人之后你要做什么.

模式和动作的详细介绍我们放在后面部分,现在大家先对 awk 结构有一个了解.我们在后面学习中慢慢渗透给大家更多内容.

1.5 模式动作

实例 1-1 基本的模式和动作

```
[root@LSD ~]# awk -F ":" 'NR>=2&&NR<=6 {print NR,$1}' /etc/passwd
2 bin
3 daemon
4 adm
5 lp
6 sync
```

说明:

-F 指定分隔符为冒号,相当于以":"为菜刀

'NR>=2&&NR<=6' 这部分表是模式,是一个条件,表示取第 2 行到第 6 行

{print NR,$1} 这部分表示动作,表示要输出 NR 行号和$1 第一列

这里只是给大家做模式动作简介,具体详细含义请继续往下看

实例 1-2 只有模式

```
[root@LSD ~]# awk -F ":" 'NR>=2&&NR<=6' /etc/passwd
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
```

说明:

-F 指定分隔为冒号

NR>=2&&NR<=6 这部分是条件,表示取第 2 行到第 6 行

但是这里没有动作,这里大家需要了解如果只有条件(模式)没有动作,awk 默认输出整行.

实例 1-3 只有动作

```
[root@LSD ~]# awk -F ":" '{print NR,$1}' /etc/passwd
1 root
2 bin
3 daemon
4 adm
....省略
```

说明:

-F 指定分隔符为冒号

这里没有条件,表示对每一行都处理.

{print NR,$1}表示动作,显示 NR 行号与$1 第一列.

这里要理解没有条件的时候,awk 会处理每一行.

实例 1-4　　多个模式和动作

```
[root@LSD ~]# awk -F ":" 'NR==1{print NR,$1}NR==2{print NR,$F}' /etc/passwd
1 root
2 bin:x:1:1:bin:/bin:/sbin/nologin
```

说明:

-F 指定分隔符为冒号

这里有多个条件与动作的组合

NR==1 表示条件,行号(NR)等于 1 的条件满足的时候,执行{print NR,$1}动作,输出行号与第一列

NR==1 表示条件,行号(NR)等于 2 的条件满足的时候,执行{print NR,$NF}动作,输出行号与最后一列($NF)

注意:

- Pattern 和{Action}需要用单引号引起来,防止 shell 作解释
- Pattern 是可选的,如果不指定,awk 将处理输入文件中的所有记录,.如果指定一个模式,awk 则只处理匹配指定的模式记录.
- {Action}为 awk 命令,可以是单个命令,也可以是多个命令,整个 Action(包括里面的所有命令)都必须放在{ 和 }之间
- Action 必须被{ } 包裹,没有被{ } 包裹的就是 Pattern
- file 要处理的目标文件

1.6 awk 执行过程

在深入了解 awk 前,我们需要知道 awk 如何处理文件的,

实例 1-5

```
[root@LSD ~]# mkdir awk
[root@LSD ~]# cd awk/
[root@LSD awk]# head /etc/passwd >awkfile.txt
[root@LSD awk]# cat awkfile.txt
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
```

这个文件仅包含十行文件,我们使用下面的命令.

实例 1-6 awk 执行过程演示

```
[root@LSD awk]# awk 'NR>=2{print $0}' awkfile.txt
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
```

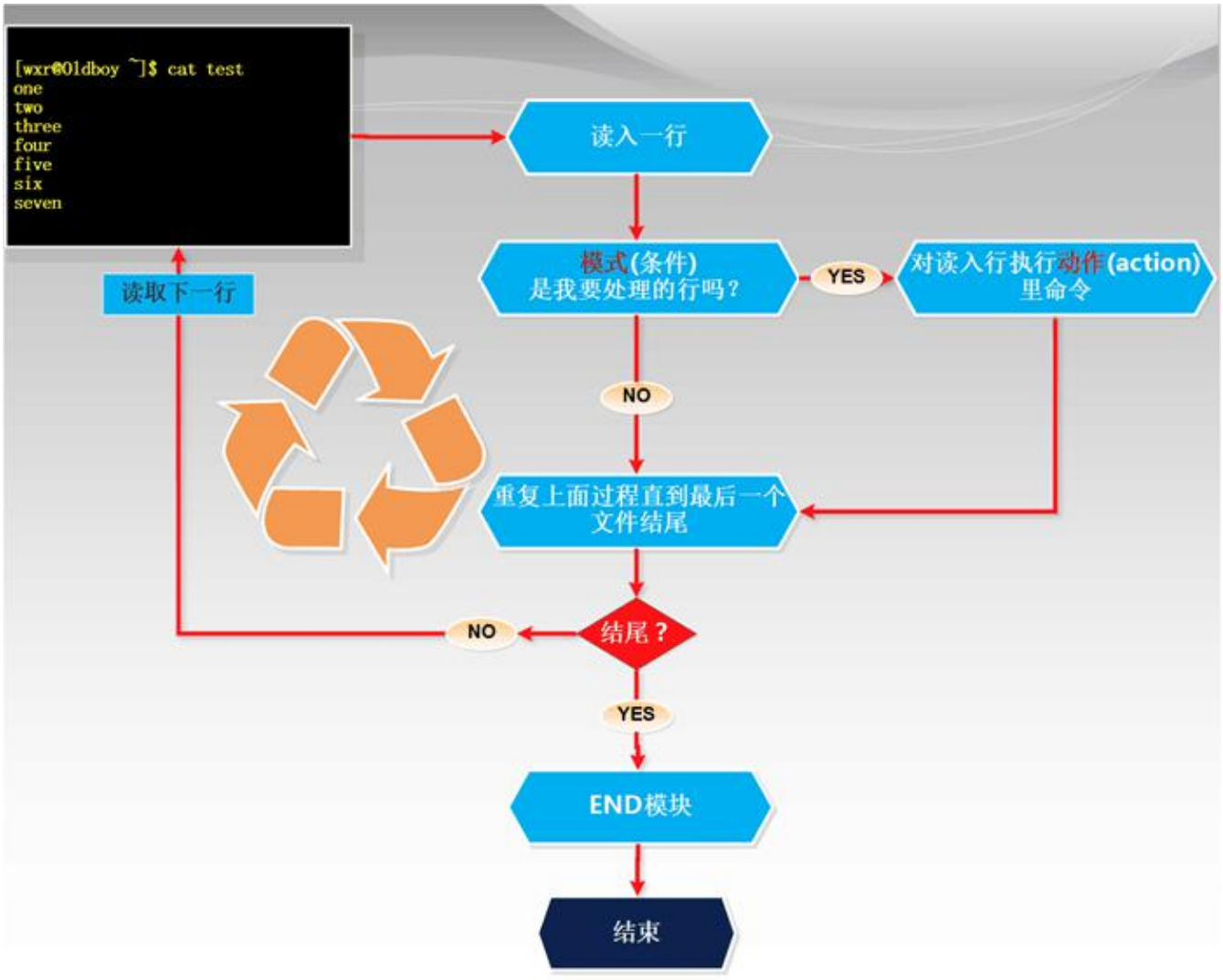
说明:

条件 NR>=2 表示行号大于等于 2 时候,执行{print $0}显示整行.

awk 是通过一行一行的处理文件,这条命令中包含模式部分(条件)和动作部分(动作),awk 将处理模式(条件)选定的行.

1.7 小结 awk 执行过程

1. awk 读入第一行内容
2. 判断是否符合模式中的条件:NR>=2
   如果匹配默认则执行对应的动作{print $0}
   如果不匹配条件,继续读取下一行
3. 继续读取下一行
4. 重复过程 1-3,直到读取到最后一行(EOF:end of file)



1.8 记录和字段

接下来我们给大家带来两个新概念记录和字段,这里为了方便大家理解可以把记录就当作行即记录行,字段相当于列.字段==列

名称	含义
record	记录,行
field	域,区域,字段,列

1.8.1 记录(行)

查看一下下面这段文字

```
[root@LSD awk]# cat /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
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

思考:

一共有多少行?如何知道的?通过什么标志?

awk 对每个要处理的输入数据认为都是具有格式和结构的,而不仅仅是一堆字符串,默认情况下,**每一行内容都是一条记录**,并以**换行符分隔(\n)结束**

1.8.2 awk 记录分隔符-RS

记录分隔符,每一个记录(行)是如何结束的.

图1-4 awk 记录分隔符

- awk 默认情况下每一行都是一个记录(record)
- RS 即 record separator 输入输入数据记录分隔符,每一行是怎么没的,表示每个记录输入的时候分隔符.即行与行之间如何分隔.
- NR 即 number of record 记录(行)号,表示当前正在处理的记录(行)的号码
- ORS 即 output record separator 输出记录分隔符

awk 使用内置变量 RS 来存放**输入记录分隔符**,RS 表示的是**输入**的**记录分隔符**,这个值可以通过 BEGIN 模块重新定义修改.

实例 1-7　使用"/"为默认记录分隔符

```
[root@LSD awk]# head -2 awkfile.txt
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
[root@LSD awk]# awk 'BEGIN{RS="/"}{print NR,$0}' awkfile.txt
1 root:x:0:0:root:
2 root:
3 bin
4 bash
bin:x:1:1:bin:
5 bin:
6 sbin
7 nologin
daemon:x:2:2:daemon:
8 sbin:
9 sbin
10 nologin
adm:x:3:4:adm:
```

第四行的换行符也算是一个符号,只要没遇到 /,就不算是新的一行.

在 awk 眼中,文件是从头到尾一段连续的字符串,恰巧中间有些\n(回车换行符),\n 也是字符.

\n 就是换行,所以后面的内容到了下一行了.

1.8.3 对$0 的认识

在 awk 中,$0 表示整行,其实 awk 使用$0 来表示整条记录.记录分隔符\n 保存在 RS 变量中,或者说每个记录以 RS 内置变量结束.

另外 awk 对每一行的记录号都有一个内置变量 NR 来保存,每处理一条记录 NR 的值就会自动+1.

下面通过示例来加强一下什么是记录,记录分隔符.

实例 1-8 NR 记录号

```
[root@LSD awk]# awk '{print NR,$0}' awkfile.txt
1 root:x:0:0:root:/root:/bin/bash
2 bin:x:1:1:bin:/bin:/sbin/nologin
3 daemon:x:2:2:daemon:/sbin:/sbin/nologin
4 adm:x:3:4:adm:/var/adm:/sbin/nologin
5 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

说明:

NR 即 number of record,当前记录的记录号,刚开始学也可以理解为行号

$0 表示整行或说整个记录

1.8.4 企业面试题:按单词出现频率降序排序(计算文件中每个单词的重复数量)

注(此处使用 sort 与 uniq 即可)

创建题目方法:

```
[root@LSD awk]# sed -r 's#[^a-zA-Z]+# #g' /etc/passwd >count.txt
```

```
[root@LSD awk]# cat count.txt
root x root root bin bash
bin x bin bin sbin nologin
daemon x daemon sbin sbin nologin
adm x adm var adm sbin nologin
lp x lp var spool lpd sbin nologin
sync x sync sbin bin sync
shutdown x shutdown sbin sbin shutdown
halt x halt sbin sbin halt
.....省略
```

思路:

1. 让所有的单词排成一列,这样每个单词都是单独的一行
2. 设置 RS 值为空格
3. 将文件里的所有空格替换成回车换行符"\n"
4. grep 所有连续的字母,egrep -o 参数让他们排成一列.
5. 整体思路:想办法让所有单词排成一列,站成一队,排序,合并重复的,显示重复数量.

答案:

```
[root@LSD awk]# sed 's# #\n#g' count.txt |sort -n|uniq -c|sort -nr
      45 x
      31 sbin
      25 nologin
      20 bin
      15 home
      14 bash
      12 var
       4 spool
[root@LSD awk]# tr " " "\n" < count.txt |sort -n|uniq -c|sort -nr
      45 x
      31 sbin
      25 nologin
      20 bin
      15 home
      14 bash
      12 var
       4 spool
[root@LSD awk]# egrep -o "[a-zA-Z]+" count.txt |sort|uniq -c|sort -nr
      45 x
      31 sbin
      25 nologin
      20 bin
      15 home
      14 bash
      12 var
[root@LSD awk]# awk 'BEGIN{RS="[:0-9/ ]+"}{print NR,$0}' count.txt|sort|uniq -c|sort -nr
[root@LSD awk]# awk 'BEGIN{RS="[^a-zA-Z]+"}{print NR,$0}' count.txt|sort|uniq -c|sort -nr
```

小结:

sort 默认是按照字母顺序排列

1.8.5 awk 记录知识小结

1. NR(number of recprd)存放着每个记录的号(行号)读取新行时候会自动+1
2. RS(record separator)是输入数据的记录的分隔符,简单理解就是可以指定每个记录的结尾标志
3. RS 作用就是表示一个纪录的结束
4. 当我们修改了 RS 的值,最好配合 NR(行)来查看变化,也就是修改了 RS 的值通过 NR 查看结果,调试 awk 程序
5. ORS 输出数据的纪录的分隔符

技巧:

一步一步来,先修改 RS,然后用 NR 调试,看看到底如何分隔的,

然后通过 sort,uniq 去重.

1.8.6 字段(列)

每条记录都是由多个区域(field)组成的,默认情况下区域之间的分隔符是由空格(即空格或制表符)来分隔,将分隔符记录在内置变量 FS 中.每行记录的区域数据保存在 awk 的内置变量 NF 中.

约定:

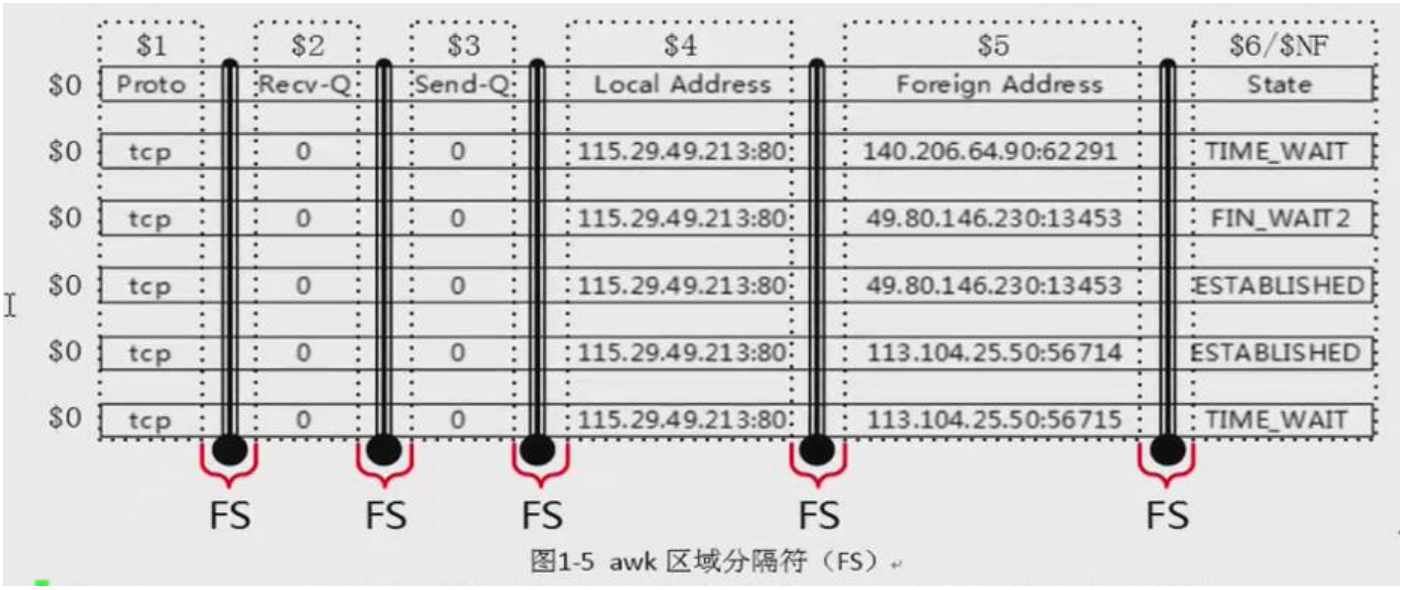


图1-5 awk 区域分隔符（FS）

FS 即 fileld separator,输入字段(列)分隔符,分隔符就是菜刀,把一行字符串切为很多区域

NF 即 number of fileds,表示一行中列(字段)的个数,可以理解为菜刀切过一行后,切成了多少分

实例 1-9　指定分隔符

```
[root@LSD awk]# awk -F ":" 'NR>=2&&NR<=5{print $1,$3}' awkfile.txt
bin 1
daemon 2
adm 3
lp 4
```

说明:

以(冒号)为分隔符,显示第 2 行到第 5 行之间的第一区域和第三区域

此处的 FS 只是一个字符,起始他可以指定多个,此时 FS 指定的值可以是一个正则表达式.

正常情况下,当你指定分隔符(非空格)的时候,例如指定多个区域分隔符,每个分隔符就是一把刀,把左右两边且为两个部分.

实例 1-10　查看并排版

```
[root@LSD awk]# awk 'NR==1,NR==10{print $NF,$0}' count.txt |column -t
bash      root      x  root      root  bin      bash
nologin   bin       x  bin       bin   sbin     nologin
nologin   daemon    x  daemon    sbin  sbin     nologin
nologin   adm       x  adm       var   adm      sbin     nologin
nologin   lp        x  lp        var   spool    lpd      sbin     nologin
sync      sync      x  sync      sbin  bin      sync
shutdown  shutdown  x  shutdown  sbin  sbin     shutdown
halt      halt      x  halt      sbin  sbin     halt
nologin   mail      x  mail      var   spool    mail     sbin     nologin
nologin   uucp      x  uucp      var   spool    uucp     sbin     nologin
```

实例 1-11　直接使用 count -t

```
[root@LSD awk]# column -t count.txt
root      x  root      root  bin      bash
bin       x  bin       bin   sbin     nologin
daemon    x  daemon    sbin  sbin     nologin
adm       x  adm       var   adm      sbin     nologin
lp        x  lp        var   spool    lpd      sbin     nologin
sync      x  sync      sbin  bin      sync
shutdown  x  shutdown  sbin  sbin     shutdown
halt      x  halt      sbin  sbin     halt
```

1.8.7 企业面试题:同时取出 oldboy 和 31333741 这两个内容(指定多分隔符)

```
[root@LSD awk]# echo "I am oldboy,my qq is 31333741" >>oldboy.txt
[root@LSD awk]# cat oldboy.txt
I am oldboy,my qq is 31333741
```

同时取出 oldboy 和 31333741 这两个内容

思路:

我们用默认的想法一次使用一把刀,需要配合管道的.如何同时使用两把刀呢?看下面的结果.

答案:

```
[root@LSD awk]# awk -F "[ ,]" '{print $3,$NF}' oldboy.txt
oldboy 31333741
```

实例 1-12 默认分隔符与指定分隔符会有些差异

```
[root@LSD awk]# ifconfig eth3|awk 'NR==2' >awkip.txt
[root@LSD awk]# cat awkip.txt
          inet addr:10.1.1.3  Bcast:10.1.1.255  Mask:255.255.255.0
[root@LSD awk]# awk -F"[ :]+" '{print $2}' awkip.txt
inet
[root@LSD awk]# awk -F"[ :]+" '{print $3}' awkip.txt
addr
[root@LSD awk]# awk -F"[ :]+" '{print $4}' awkip.txt
10.1.1.3
```

1.8.8 ORS 与 OFS 简介

现在说说 ORS 和 OFS 这两个内置变量的含义

RS 是输入记录分隔符,决定 awk 如何读取或分隔每行(记录)

ORS 表示输出记录分隔符(ooutput record separtor)决定 awk 如何输出一行(记录)的,默认是回车换行(\n)

FS 是输入区域分隔符,决定 awk 读入一行后如何在分为多个区域

OFS 表示输出区域分隔符,决定 awk 输出每个区域的时候使用什么分割他们

awk 无比强大,你可以通过 RS,FS 决定 awk 如何读取数据,你也可以通过修改 ORS,OFS 的值指定 awk 如何输出数据

注意:awk 默认的 FS 分隔符.空格序列.一个空格或多个空格 tab 都认为是一样的,一个整体,如下显示

```
[root@LSD awk]# cat nulltab.txt
                a               b
[root@LSD awk]# cat nulltab.txt |awk '{print $1}'
a
[root@LSD awk]# cat nulltab.txt |awk '{print $2}'
b
```

1.8.9 ORS 与 OFS 简介

现在说说 ORS 和 OFS 这两个内置变量的含义

RS 是输入记录分隔符,决定 awk 如何读取或分隔每行(记录)

ORS 表示输出记录分隔符(output record separator),决定 awk 如何输入一行(记录)的,默认是回车换行(\n)

FS 是输入区域分隔符,决定 awk 读入一行后如何再分为多个区域

OFS 表示输出区域分隔符,决定 awk 输出每个区域的时候使用什么分割他们

awk 无比强大,你可以通过 RS,FS 决定 awk 如何读取数据,你也可以通过修改 ORS,OFS 的值指定 awk 如何输出数据:

看下图:

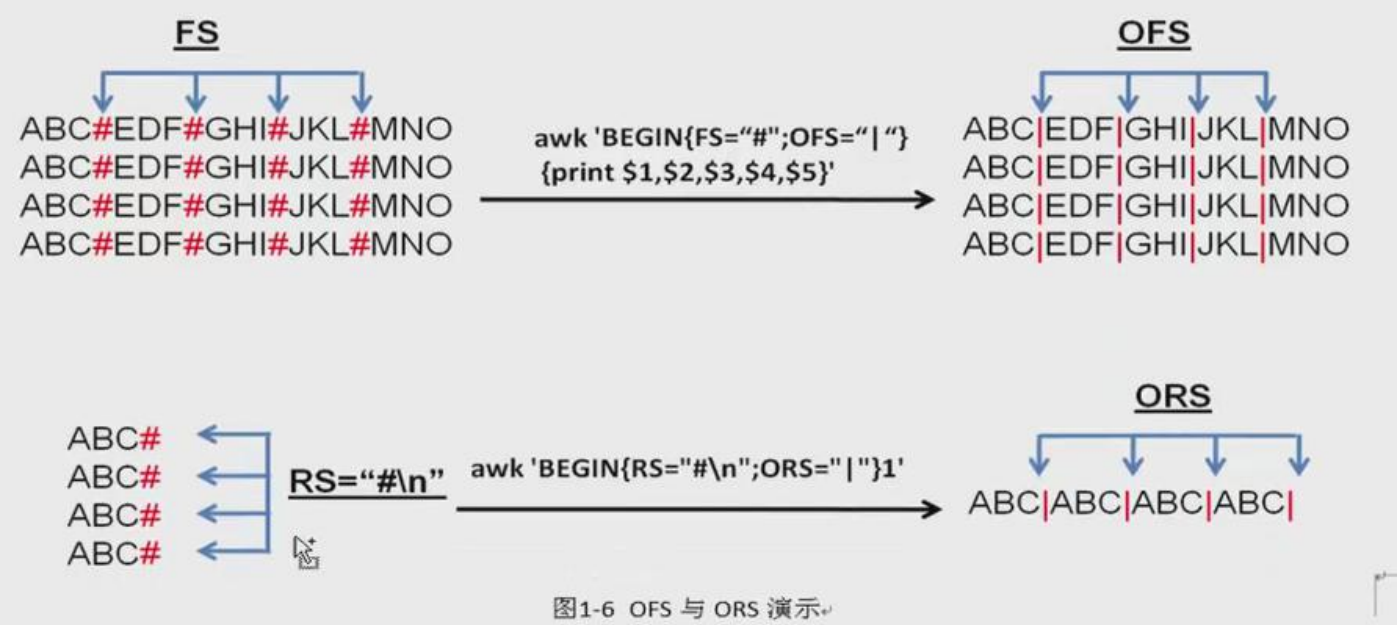


图1-6 OFS 与 ORS 演示

小结:

FS 文件的输入字段分隔符,文件的输入每一列的分隔符号

OFS awk 怎么输出每一列

```
[root@LSD awk]# cat OFS.txt
aaaa#bbb#ccc
aaaa#bbb#ccc
[root@LSD awk]# awk 'BEGIN{FS="#";OFS=":"}{print $1,$2,$3}' OFS.txt
aaaa:bbb:ccc
aaaa:bbb:ccc
```

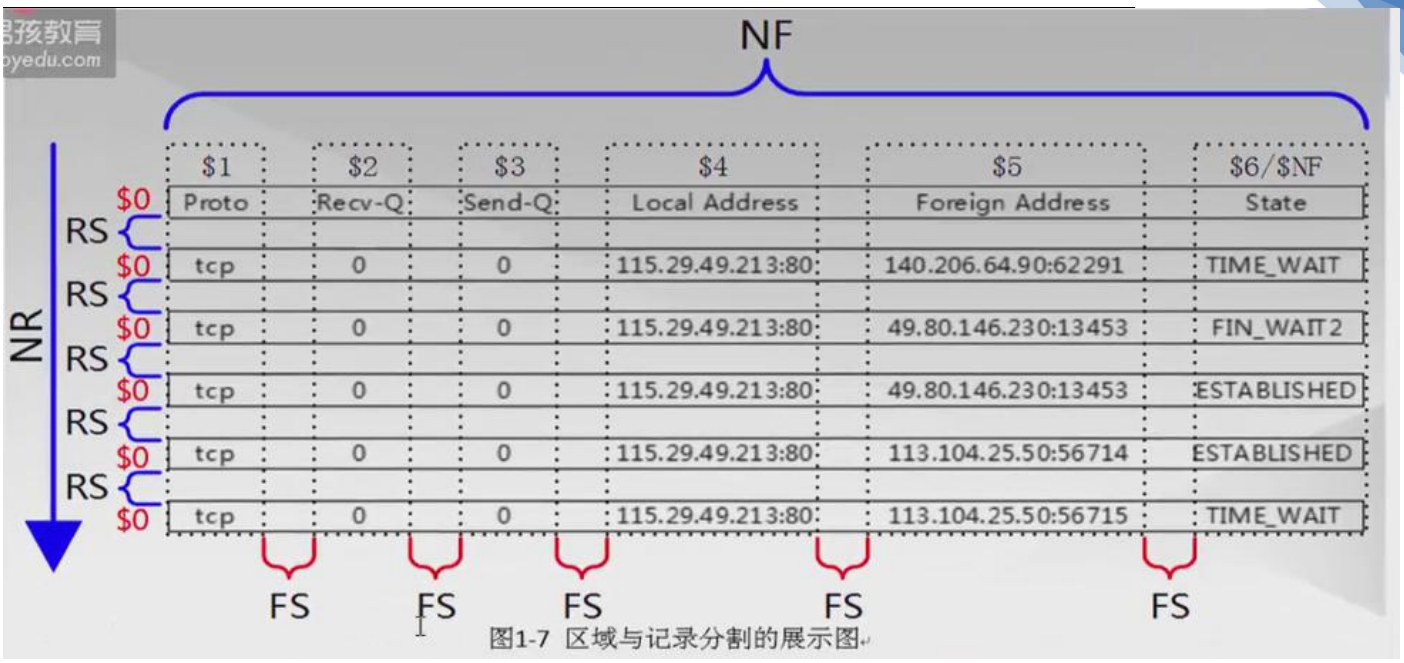


图1-7 区域与记录分割的展示图

- RS 记录分隔符,表示每行的结束标志
- NR 行号(记录号)
- FS 字段分隔符,每列的分割标志或结束标志
- NF 就是每行有多少列,每隔记录中字段的数量

- $符号表示取某个列(字段),$1,$2,$NF
- NF(number of filed)表示记录中的区域(列)数量,$NF 取最后一个列(区域)
- FS(-F) filed separator 字段(列)分隔符,-F(FS)":" <==> 'BEGIN(FS=":")'
- RS record separator 记录分隔符(行的结束标志)
- NR number of record 行号
- 选好合适的刀 FS(***),RS,OFS,ORS
- 分隔符===>结束标识
- 记录与区域,你就对我们所谓的行与列,有了新的认识(RS,FS)

1.9 awk 基础入门总结

到了这里我们回头看看,我们之前学习的内容
awk 的命令行结构
awk 的模式(条件 找谁)和动作(干什么)
awk 的记录和字段
比较核心常用的是字段

第 2 章 awk 进阶

2.1 awk 模式与动作

回顾下 awk 的格式,里面的模式动作,换句话说就是条件和做什么

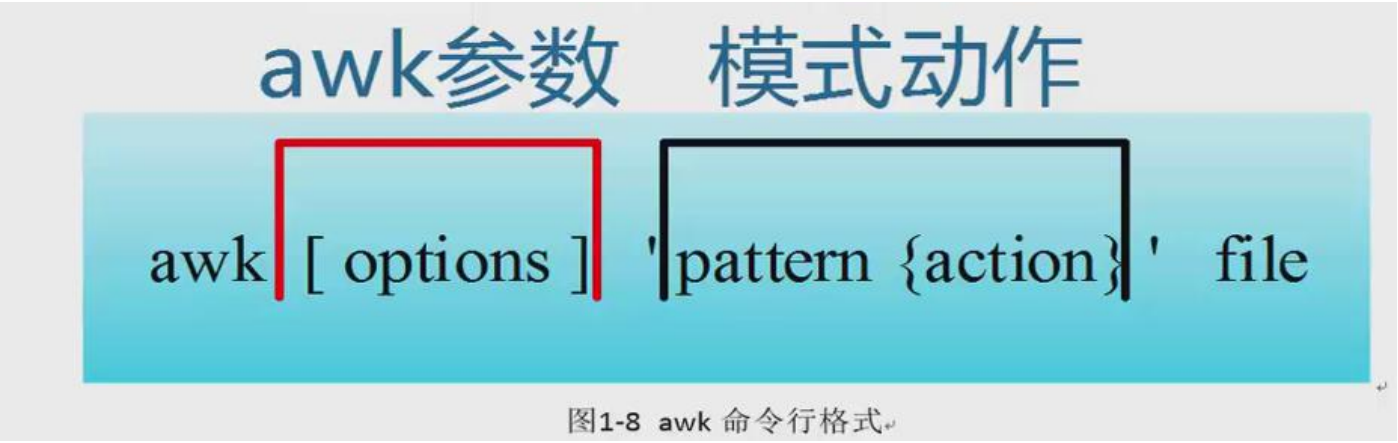


图1-8 awk 命令行格式

接下来就详细介绍下,awk 的模式都有几种

正则表达式作为模式

比较表达式作为模式

范围模式

特殊模式 BEGIN 和 END

awk 的模式是你玩好 awk 的必备也是最基础的内容,必须熟练掌握

2.2 正则表达式作为模式

awk 同 sed 一样也可以通过模式匹配来对输入的文本进行匹配处理.说道模式匹配,肯定少不了正则表达式 awk 也支持大量的正则表达式模式,大部分与 sed 支持的元字符类似,而且正则表达式是玩转三剑客的必备工具.

元字符	功能	示例	解释
^	字符串开头	/^oldboy/ $3~/^oldboy/	匹配所有以 oldboy 开头的字符串 匹配出所有第三列中以 oldboy 开头
$	字符串的结尾	/oldboy$/ $3~/oldboy$/	匹配所有以 oldboy 结尾的文本(字符串) 匹配第三列中以 oldboy 结尾的文本
.	匹配任意单个字符(包括回车符)	/c..l/	匹配字母 c 然后两个任意字符,再以 l 结尾的行,比如:ckkl,c@#l 等
*	重复 0 个或多个前一个字符	/a*cool/	匹配 0 个或多个 a 之后紧跟着 cool 的行,比如 cool aaacool
+	重复前一个字符一次或一次以上	/a+b/	匹配一个或多个 a 加 b 的行
?	匹配 0 个或一个前导字符	/a?b/	匹配 b 或 ab 的行
[]	匹配指定字符组内的任一个字符	/^[abc]/	匹配以字母 a 或 b 或 c 开头的行
[^]	匹配不在指定字符组内的任一字符	/^[^abc]/	匹配不以字母 a 或 b 或 c 开头的行
()	子表达式组合	/(oldboy)+/	表示一个或多个 oldboy 组合,当有一些字符需要组合时,使用括号括起来
|	或者的饿意思	/(oldboy)|B/	匹配 oldboy 或字母 B 的行

awkm 默认不支持的元字符,需要田间参数才能够支持的元字符			
x{m} x{m,} x{m,n}	x 重复 m 次 x 重复至少 m 次 x 重复至少 m 次,但不超过 n 次 需要指定参数: --posix 或--re-interval 没有该参数不能使用这种模式	/cool{5}/ /{cool}{2,}/ /{cool}{5,6}/	需要注意一点的是 cool 加括号或不加括号的区别,x 可以是字符串也可以只是一个字符,所以/cool\{5\}表示匹配 coo 再加上 5 个 l,及 coollllll (cool){2,}则表示匹配 coolcool coolcooolcool

正则表达式的运用,默认是在行内查找匹配的字符串,若有匹配则执行 action 操作,但是有时候仅需要固定的列来指定的正则表达式,比如:我想去/etc/passwd 文件中第五列($5)这一列查找匹配 mail 字符串的行,这样就需要用另外两个匹配操作符,并且 awk 里面只有这两个操作符来匹配正则表达式的.

2.3 awk 正则匹配操作符

~ 用于对记录或区域的表达式进行匹配

!~ 用于表达与~相反的意思

下面还是通过具体示例来看看,awk 如何通过正则表达式匹配字符串的

2.4 awk 正则表达式匹配整行

```
[root@LSD awk]# awk -F ":" '/^root/' awkfile.txt
root:x:0:0:root:/root:/bin/bash
```

和下面效果是一样的

```
[root@LSD awk]# awk -F ":" '$0~/^root/' awkfile.txt
root:x:0:0:root:/root:/bin/bash
```

说明:

awk 只用正则表达式的时候是默认匹配整行的即$0~/^root/和'/^root/' 是一样的

2.5 awk 正则表达式匹配一行的某一列

```
[root@LSD awk]# awk -F ":" '$5~/^shutdown/' awkfile.txt
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
```

2.6 某个区域中的开头和结尾

知道了如何使用正则表达式匹配操作符之后,我们来看看 awk 正则和 grep 和 sed 不同的地方.
awk 正则表达式

^ 匹配一个字符串的开头
$ 匹配一个字符串的结尾

在 sed 和 grep 这两个剑客中,我们都把他们当作行的开头和结尾,但是在 awk 中他表示的是字符串的开头和结尾.
接下来我们来通过练习题练习 awk 如何使用正则表达式

2.7 创建测试环境

```
[root@LSD awk]# cat reg.txt
Zhang Dandan     41117397      :250:100:175
Zhang Xiaoyu     390320151     :155:90:201
Meng   Feixue     80042789      :250:60:50
Wu     Waiwai    70271111      :250:80:75
Liu    Bingbing  41117483      :250:100:175
Wang   Xiaoai    3515064655    :50:95:135
Zi     Gege      1986787350    :250:168:200
Li     Youjiu    918391635     :175:75:300
Lao    Nanhai    918391635     :250:100:175
```

2.8 测试文件说明

第一列是姓氏
第二列是名字
第一第二列合起来就是姓名
第三列是对应的 ID 号码
最后三列是三次捐款数量

2.9 awk 正则表达式练习题

2.9.1 显示姓 Zhang 的人第二次捐款金额及她的名字

2.9.2 显示 Xiaoyu 的名字和 ID 号码

2.9.3 显示所有以 41 开头的 ID 号码的人的全名和 ID 号码

2.9.4 显示所有以一个 D 或 X 开头的人名全名

2.9.5 显示所有 ID 号码最后一位数字是 1 或 5 的人的全名

2.9.6 显示 Xiaoyu 的捐款,每个值是都已$开头的如:520$200$135

2.9.7 显示所有人的全名,以姓,名的格式显示如 Meng,Feixue

2.10 awk 正则表达式练习题-详解

2.10.1 显示姓 Zhang 的人的第二次捐款金额及她的名字

答案:

```
[root@LSD awk]# awk -F "[ :]+" '/Zhang/{print $1,$(NF-1)}' reg.txt
Zhang 100
Zhang 90
[root@LSD awk]# awk -F "[ :]+" '/Zhang/{print $1,$5}' reg.txt
Zhang 100
Zhang 90
```

说明:
-F 指定分隔符,现在大家应该知道了-F 即 FS 也是支持正则表达式的
[:]+ 表示连续的空格或冒号
-F "[:]+" 以连续的空格或冒号为分隔符
/Zhang/表示条件,整行中包含 Zhang 字符的这个条件
{print $1,$(NF-1)} 表示动作,满足条件后,执行显示第一列($1)和倒数第二列($(NF-1))当然$5 也可以
注意:
NF 是一行中有多少列,NF-1 整行就是倒数第二列
$(NF-1)就是取倒数第二列内容

2.10.2 显示 Xiaoyu 的姓氏和 ID 号码

答案:

```
[root@LSD awk]# awk -F "[ :]+" '/Xiaoyu/{print $1,$3}' reg.txt
Zhang 390320151
```

2.10.3 显示所有以 41 开头的 ID 号码的人的全名和 ID 号码

```
[root@LSD awk]# awk -F "[ :]+" '$3~/^41/' reg.txt
Zhang Dandan    41117397    :250:100:175
Liu    Bingbing 41117483    :250:100:175
```

说明:

-F"[:]+"以连续的空格或冒号为分隔符

$3~/^41/是一个条件,表示匹配第三列以 138 开头的行

满足条件后写任何动作,默认输出整行相当于 print$0

2.10.4 企业面试题: test.txt 内容如下,想输出不包含 oldboy 的行,仅仅是 oldboy,oldboylinux 正常输出:

```
egrep -vw "oldboy" test.txt
egrep -v "oldboy$" test.txt
egrep    "\boldboy\b" test.txt
sed '/oldboy$/d' test.txt
awk '!/oldboy$/' test.txt
awk '$0~/oldboy$/'
awk '/[^oldboy]/' test.txt
注意!这不是匹配 oldboy,而是字母,只要不在[o,l,d,b,o,y]这几个字符内,就匹配出来
```

2.10.5 找出所有人名的第一个字母是 D 或 X,显示她们的全名

答案:

```
[root@LSD awk]# awk -F "[ :]+" '$2~/^(D|X)/' reg.txt
Zhang Dandan     41117397        :250:100:175
Zhang Xiaoyu     390320151       :155:90:201
Wang   Xiaoai     3515064655      :50:95:135
[root@LSD awk]# awk -F "[ :]+" '$2~/^[DX]/' reg.txt
Zhang Dandan     41117397        :250:100:175
Zhang Xiaoyu     390320151       :155:90:201
Wang   Xiaoai     3515064655      :50:95:135
```

2.10.6 显示所有 ID 号码最后一位数字是 1 或 5 的人的全名

答案:

```
[root@LSD awk]# awk -F "[ :]+" '$3~/(1|5)$/{print $1}' reg.txt
Zhang
Wu
Wang
Li
Lao
[root@LSD awk]# awk -F "[ :]+" '$3~/[15]$/{print $1}' reg.txt
Zhang
Wu
Wang
Li
Lao
```

2.10.7 显示 Xiaoyu 的捐款,每个值都有以$开头,如$520$200$135

答案:

```
[root@LSD awk]# awk -F "[ :]" '/Xiaoyu/{print "$"$4"$"$5"$"$6}' reg.txt
$$390320151$
[root@LSD awk]# awk -F "[ :]" '$2~/Xiaoyu/{print "$"$4"$"$5"$"$6}' reg.txt
$$390320151$
```

gsub(/正则表达式/,"你要替换成啥",[目标($1 $2 $3)])

```
[root@LSD awk]# awk '$2~/^Xiaoyu$/{gsub(/:/,"$");print $NF}' reg.txt
$155$90$201
[root@LSD awk]# awk '$2~/^Xiaoyu$/{gsub(/:/,"$",$NF);print $NF}' reg.txt
$155$90$201
```

OFS

```
[root@LSD awk]# awk 'BEGIN{FS=":";OFS="$"}{print "",$2,$3,$4}' reg.txt
$250$100$175
$155$90$201
$250$60$50
$250$80$75
$250$100$175
$50$95$135
$250$168$200
$175$75$300
$250$100$175
```

2.10.8 显示所有人的全名,以名字,姓的格式显示如 Feixue,Meng

```
[root@LSD awk]# awk -F "[ :]+" '{print $2","$1}' reg.txt
Dandan,Zhang
Xiaoyu,Zhang
Feixue,Meng
Waiwai,Wu
Bingbing,Liu
Xiaoai,Wang
Gege,Zi
Youjiu,Li
Nanhai,Lao
[root@LSD awk]# awk -F "[ :]+" 'BEGIN{OFS=","}{print $2,$1}' reg.txt
Dandan,Zhang
Xiaoyu,Zhang
Feixue,Meng
Waiwai,Wu
Bingbing,Liu
Xiaoai,Wang
Gege,Zi
Youjiu,Li
Nanhai,Lao
[root@LSD awk]# awk -v OFS="," '{print $2,$1}' reg.txt
Dandan,Zhang
Xiaoyu,Zhang
Feixue,Meng
Waiwai,Wu
Bingbing,Liu
Xiaoai,Wang
Gege,Zi
Youjiu,Li
Nanhai,Lao
```

2.10.9 企业面试题:取出 eth0 的 ip 地址

答案:

```
[root@LSD awk]# ifconfig eth3|awk -F "(addr:)|( Bcast:)" 'NR==2{print $2}'
10.1.1.3
[root@LSD awk]# ifconfig eth3|awk -F "[ :]+" 'NR==2{print $4}'
10.1.1.3
[root@LSD awk]# ifconfig eth3|awk -F "[^0-9.]+" 'NR==2{print $2}'
10.1.1.3
```

方法一:

-F "(addr:)|(Bcast)" 这个还是比较容易理解的,看看我们的目标 10.1.1.3,左边是 Bcast:左一刀,右边一刀,中间的就是我们想要的.

但是还需要一个条件 ip 地址在第二行所以使用 NR==2 来表示

最后的形式就是 -F"(addr:)|(Bcast:)" 'NR==2{print $2}'了.

方法二:

我们再分析,10.1.1.3 两边还有什么可用的菜刀吗?

肯定有,左边的是:右边的空格

这里我们想指定空格或冒号(:)为菜刀,即-F"[:]"

但是你还能还会第二行开始每一行开头都有好多空格,连续的空格,怎么处理呢?

正则表达式加号(+),表示重复前面一个字符一次或一次以上.

结合一下就是

-F "[:]+" 以连续的空格或冒号 或者他们的组合为菜刀,最后就可以取到我们想要的 IP 地址.

即-F "[:]+" 'NR==2{print $4}'

+号的用法请参考前面 grep 正则表达式部分

方法三:

学会逆向思维,看看我们要的结果 10.1.1.50,IP 地址:数字和点(.),我是否可以指定分隔符,以数字和以外的字符为分隔符呢?

换句话说就是要排除数字和点(.)正则表达式与排除常用的就是[^0-9]即不匹配数字和点(.)

合起来就是 awk -F "[^0-9]" 'NR==2{print $2}'

2.10.10 企业案例 1:取出常用服务端口号

ftp,http,https,mysql,ssh 端口号

思路

linux 下面服务与端口信息的对应表格在/etc/service 里面,所以这道题我们要处理/etc/services 文件

我们简单分析下 services 文件

```
[root@LSD awk]# sed -n '23,30p' /etc/services
tcpmux          1/tcp                           # TCP port service multiplexer
tcpmux          1/udp                           # TCP port service multiplexer
rje             5/tcp                           # Remote Job Entry
rje             5/udp                           # Remote Job Entry
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
```

从 23 行开始基本上每一行第一列是服务名称,第二列的第一部分是端口号,第二列的第二部分是 tcp 或 udp 协议.

答案:

```
[root@LSD awk]# awk -F "[ /]+" '$1~/^(ftp|http|https|mysql|ssh)$/{print $1,$2}' /etc/services|uniq
ftp 21
ssh 22
http 80
https 443
mysql 3306
ftp 21
ssh 22
[root@LSD awk]# awk -F "[ /]+" '$1~/^(ftp|http|https|mysql|ssh)$/&&$3~/tcp/{print $1,$2}' /etc/services
ftp 21
ssh 22
http 80
https 443
mysql 3306
```

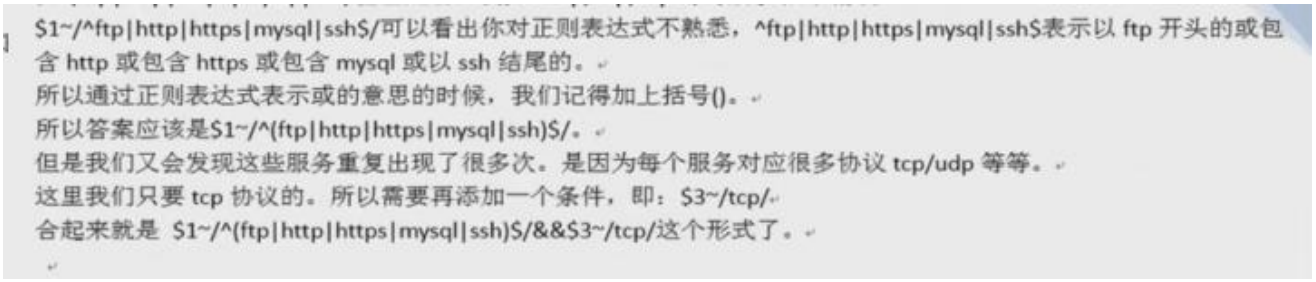
说明:

先看看几个错误的例子

```
awk -F "[ /]+" '$1~/ftp|http|https|mysql|ssh/{print $1,$2}' /etc/services
awk -F "[ /]+" '$1~/^ftp|http|https|mysql|ssh$/{print $1,$2}' /etc/services
```

上面两个结果就是错误的原因在于,会取出无关的项目

$1~/ftp|http|https|mysql|ssh/会匹配出,例如:ftp,sftp,yftp 等等很不精确



2.10.11 企业案例 2:取出常用服务端名称

80,443,3306,23,22,25,21

大家自己尝试下这个例子

```
Linux 源于生活,你的想法,把他变成字符或英文==>已经有了对应的功能
```

2.10.12 正则表达式部分的疑问

```
awk '/^Xiaoyu/' reg.txt
awk '$2~/^Xiaoyu/' reg.txt
```

/^Xiaoyu/意思以 Xiaoyu 为开头的字符串吗,为什么默认$0 匹配就是不行了?

/Xiaoyu/就可以了,这是为什么?

解答:

```
^     匹配字符串的开头的位置
$     匹配字符串结尾的位置
```

```
awk '/^Xiaoyu/' reg.txt
awk '/Xiaoyu/' reg.txt
```

```
grep "^Xiaoyu" reg.txt          ###awk 不会自动找到某一列,需要你指定
grep "Xiaoyu" reg.txt
```

2.11 比较表达式作为模式-需要一些例子

之前我们看了正则表达式在 awk 下的运用,下面在具体看看比较表达式如何在 awk 下工作.
某一区域作出相关的判断,比如打印成绩在 80 分以上的行,这样就必须对这一个区域做比较判断,表 1-4 列出了 awk 可以使用的相关的判断,可以用来比较数字或者字符串,还有正则表达式.是为真的时候,b 表达式结果为 1,否则为 0,只有表达式为真,awk 才执行相关的 action

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
>=	大于或等于	x>=y
>	大于	x<y
以上的运算符是针对数字,下面两个运算符之前已有示例,针对字符串		
~	与正则表达式	x~/y/
!~	与正则表达式不匹配	x!~y

2.11.1 awk 变量导致的问题

```
[root@LSD awk]# awk -F ":" '{print $5}' awkfile.txt
root
bin
....省略
[root@LSD awk]# awk -F ":" '$5==root' awkfile.txt
[root@LSD awk]#
```

在 awk 中,所有的字母,awk 会认为是变量.

```
[root@LSD awk]# awk -F ":" '$5=="root"' awkfile.txt
root:x:0:0:root:/root:/bin/bash
```

2.11.2 企业面试题:取出/etc/services 的 23~30 行

思路:

想表示一个范围,一个行的范围,就要用到 NR 这个内置变量了,同时也要用到比较表达式.

2.11.3 如何判断某一列是否等于某个字符呢?

在 awk 中,所有的字母,awk 会认为是变量.如果想找字符,需呀用双引号括起来

```
[root@LSD awk]# awk -F ":" '$5=="root"' awkfile.txt
root:x:0:0:root:/root:/bin/bash
[root@LSD awk]# awk -F ":" '$5~/^root$/' awkfile.txt
root:x:0:0:root:/root:/bin/bash
```

2.12 范围模式

pattern1	,	pattern2
从哪里来	到	哪里去
条件一	,	条件 2

范围模式简单理解就是从哪里来,到哪里去

匹配从条件 1 开始到条件 2 介绍的范围

范围模式处理的原则是:先匹配从第一个模式的首次出现到第二个模式的首次出现之间的内容,执行 action,然后匹配从第一个模式的下一次出现到第二个模式的下一次出现,直到文本结束,如果匹配到第一个模式而没有匹配到第二个模式,则 awk 处理从第一个模式开始直到文本结束全部的行.如果第一个模式不匹配,就算第一个模式匹配,awk 依旧不处理任何行.

```
awk '/start pos/,/end pos/{print $0}' passwd.oldboy
awk '/start pos/,/NR==XXX{print $0}' passwd.oldboy
```

范围模式的时候,范围条件的时候,表达式必须能匹配一行.

2.12.1 显示第二行到第五行的行号和整行的内容.

```
[root@LSD awk]# awk 'NR==2,NR==5{print NR,$0}' count.txt
2 bin x bin bin sbin nologin
3 daemon x daemon sbin sbin nologin
4 adm x adm var adm sbin nologin
5 lp x lp var spool lpd sbin nologin
```

说明:

条件是:从第二行,到第五行.

动作是:显示行号(NR)和整行($0)

合起来就是显示第二行到第五行的行号和整行的内容.

2.12.2 显示从以 bin 开头的行,到第五行中的行号和整行内容.

```
[root@LSD awk]# awk '/^bin/,NR==5{print NR,$0}' awkfile.txt
2 bin:x:1:1:bin:/bin:/sbin/nologin
3 daemon:x:2:2:daemon:/sbin:/sbin/nologin
4 adm:x:3:4:adm:/var/adm:/sbin/nologin
5 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

说明:

条件是:从以 bin 开头的行,到第五行

动作是:显示行号和整行内容

合起来就是显示从以 bin 开头的行,到第五行中的行号和整行内容.

2.13 awk 特殊模式-BEGIN 模式与 END 模式

2.13.1 BEGIN 模式

BEGIN 模块在 awk 读取文件之前就执行,一般用来定义我们的内置变量(预定义变量,eg:FS,RS),可以输出表头(类似 excel 表格名称)

BEGIN 模式之前我们有在示例中提到,自定义变量,给内容变量赋值等,都是用过.需要注意的是 BEGIN 模式后面要紧跟着一个 action 操作块,包含在大括号内.awk 必须在对输入文件进行任何处理前先执行 BEGIN 里的动作(action).我们可以不要任何输入文件,就可以对 BEGIN 模块进行测试,因为 awk 需要先执行完 BEGIN 模式,才对输入文件做处理.BEGIN 模式常常被用来修改内置变量 ORS,RS,FS,OFS 等的值.

2.13.2 第一个作用,内置变量的定义:去 eth0 的 ip 地址

```
[root@LSD awk]# ifconfig eth3|awk -F "(addr:)|( Bcast)" 'NR==2{print $2}'
10.1.1.3
[root@LSD awk]# ifconfig eth3|awk -F "[ :]+" 'NR==2{print $4}'
10.1.1.3
[root@LSD awk]# ifconfig eth3|awk -F "[^0-9.]+" 'NR==2{print $2}'
10.1.1.3
```

也可以写成:

```
[root@LSD awk]# ifconfig eth3|awk 'BEGIN{FS="(addr:)|( Bcast:)"}NR==2{print $2}'
10.1.1.3
[root@LSD awk]# ifconfig eth3|awk 'BEGIN{FS="[ :]+"}NR==2{print $4}'
10.1.1.3
[root@LSD awk]# ifconfig eth3|awk 'BEGIN{FS="[^0-9.]+"}NR==2{print $2}'
10.1.1.3
```

注:

命令行-F 本质就是修改的 FS 变量

2.13.3 第二个作用,在读取文件之前,输出些提示性信息,表头

实例 2-1 显示文件 awkfile.txt 的第一列和第三列(/etc/passwd 前 10 行)并在第一行输出 username 和 UID

答案:

```
[root@LSD awk]# awk -F ":" 'BEGIN{print "username","UID"}{print $1,$3}' awkfile.txt
username UID
root 0
bin 1
daemon 2
adm 3
lp 4
sync 5
shutdown 6
halt 7
```

2.13.4 第三种用法,使用 BEGIN 模块的特殊性质,进行一些测试

简单输出内容:

```
[root@LSD awk]# awk 'BEGIN{print "hellow world!"}'
hellow world!
```

进行计算:

```
[root@LSD awk]# awk 'BEGIN{print 10/3}'
3.33333
[root@LSD awk]# awk 'BEGIN{print 10/3+1}'
4.33333
[root@LSD awk]# awk 'BEGIN{print 10/3+1/4*9}'
5.58333
```

与变量有关的操作:

```
[root@LSD awk]# awk 'BEGIN{a=1;b=2;print a,b}'
1 2
[root@LSD awk]# awk 'BEGIN{a=1;b=2;print a,b,a+b}'
1 2 3
```

2.13.5 第四种方法:配合 getline 读取文件,后面 awk 函数处讲解

2.13.6 awk 中变量的概念简介

直接定义,直接使用即可

awk 中字符会被认为是变量,如果真的要给一个变量赋值(字符串),请使用双引号.

```
[root@LSD awk]# awk 'BEGIN{a=abcd;print a}'

[root@LSD awk]# awk 'BEGIN{abcd=123456;a=abcd;print a}'
123456
[root@LSD awk]# awk 'BEGIN{a="abcd";print a}'
abcd
```

说明:

没有文件 awk 依旧可以处理 BEGIN 模式下的动作(命令)

2.13.7 END 模式

END 在 awk 读取完所有的文件的时候,再执行 END 模块,一般用来输出一个结果(累加,数组结果),也可以是和 BEGIN 模块类似的结尾标识信息.

```
[root@LSD awk]# awk 'BEGIN{print "hello world!"}{print NR,$0}END{print "end of file"}' count.txt
hello world!
1 root x root root bin bash
.....省略
44 zhangyaya x home zhangyaya bin bash
end of file
[root@LSD awk]#
```

与 BEGIN 模式相对应的 END 模式,格式一样,但是 END 模式仅在 awk 处理完所有输入行后才进行处理

.

2.13.8 企业案例 3:统计/etc/services 文件里面的空行数量.

思路:

a) 空行通过正则表达式来实现:^$

b) 统计数量:

gergp -c

awk

答案:

```
[root@LSD awk]# egrep -c "^$" /etc/services
16
[root@LSD awk]# awk '/^$/{i++}END{print i}' /etc/services
16
```

说明:

方法一:

egrep 命令-c 表示 count 计数统计包含^$的行一共有多少

方法二:

使用了 awk 的基数功能,很常用

第一步:统计空行个数

/^$/表示条件,匹配出空行,然后执行{i++}

这里大家可能对 i++不是很理解

i++相当于 i=i+1

即:/^$/{i=i+1}

2.13.9 通过/^$/{i=i+1;print i} 来查看 awk 执行过程

```
[root@LSD awk]# awk '/^$/{i=i+1;print "the value of i is:"i}' /etc/services
the value of i is:1
the value of i is:2
the value of i is:3
the value of i is:4
the value of i is:5
the value of i is:6
the value of i is:7
the value of i is:8
the value of i is:9
the value of i is:10
the value of i is:11
the value of i is:12
the value of i is:13
the value of i is:14
the value of i is:15
the value of i is:16
```

2.13.10 企业面试题:文件 count.txt,文件内容是 1 到 100(由 seq 100 生成),请计算文件每行值加起来的结果(计算 1+.....+100)

思路:

文件每一行都有且只有一个数字,所以我们要让文件的每行的内容相加

回顾一下上一道题我们用的是 i++即 i=i+1

这里我们需要使用第二个常用的表达式

i=i+$0

对比一下,只是把上面的 1 换成了$0

```
[root@LSD awk]# seq 1 100|awk '{i=i+$0}END{print i}'
5050
```

说明:

awk 中的统计还是很简单的.

这里我们给大家分步执行,来解释本题

第一步:awk 处理每一行

{i=i+$0}这里只有动作(action),没有条件,所以会处理文件的每一行.

```
第 1 行: i=i+$0 <===> i=空+1 <==>i=1
第 2 行: i=i+$0 <===> i=1+2 <==>i=3
第 3 行: i=i+$0 <===> i=3+3 <==>i=6
第 4 行: i=i+$0 <===> i=6+4 <==>i=10
第 5 行: i=i+$0 <===> i=10+5 <==>i=15
第 6 行: i=i+$0 <===> i=15+6 <==>i=21
第 7 行: i=i+$0 <===> i=21+7 <==>i=28
第 8 行: i=i+$0 <===> i=28+8 <==>i=36
第 9 行: i=i+$0 <===> i=36+9 <==>i=45
.....
第 100 行: i=i+$0 <===> i= 4950+100 <==>i=5050
```

当然这里的$0 也可以换成$1,因为每一行只有一列

i+$0 的结果放在 i 里面即:i=i+$0

记住是一般情况先执行等号右边的,执行完成后在通过等号(=)把结果给等号左边的 i(小写字母)

还可以简写为 i+=$0 或 i+=$1 都可以

2.13.11 awk 调试技巧:

让 awk 显示出每一步的执行结果
一般通过 print 来配合完成
几种常用的运算表达式
i=i+1 ==>i++
i=i+2 ==>i+=2
i=i+$0==>i+=$0

2.14 awk 中的动作

在一个模式-动作语句中,模式决定动作什么时候执行,有时候动作会非常简单:一条单独的打印语句(print)或复制语句,在有些时候动作有可能是多条语句,语句之间用换行符或分号分开.
awk 的动作中如果有两个或两个以上的语句,需要用分号分隔
动作部分大家理解为花括号里面的内容即可,总体分为

- 表达式
- 流程控制语句
- 空语句
- 数组

我们会在后面 awk 高级知识里详细讲解

2.14.1 awk 模式与动作小结:

awk 命令核心由模式和动作组成,就是找谁(干啥)
模式就是条件,动作就是具体干什么
正则表达式:必须掌握正则,熟练
条件表达式:比大小,比较是否相等
范围表达式:从哪里来,到哪里去
注意 BEGIN 或 END 模块只能有一个 BEGIN{}BEGIN{}或 END{}END{}

2.15 总结 awk 执行的过程

回顾下 awk 的结构

awk -F 指定分隔符"BEGIN{}{}END{}文件,如下图

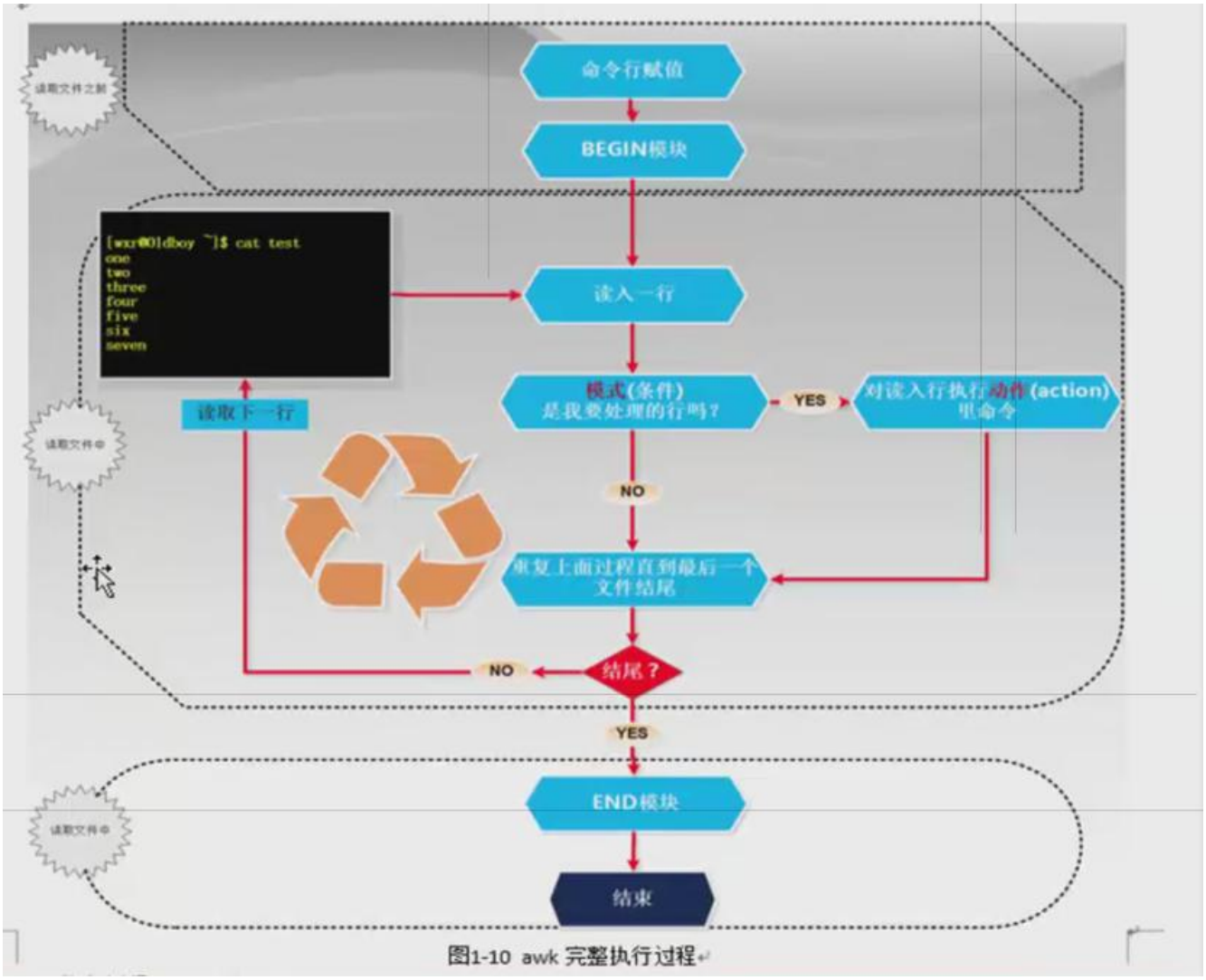


图1-10 awk 完整执行过程

2.16 awk 执行过程

A: 命令行的赋值(-F 或-v)
B: 执行 BEGIN 模式里面的内容
C: 开始读取文件
D: 判断条件(模式)是否成立
成立则执行对应动作里的内容
读取下一行
直到读取到最后一个文件的内容
E: 最后执行 END 模式里面的内容
F: 结束

2.17 思想:

1.awk 核心思想就是先处理,然后 END 模块输出.累加(a++;a+=$0,数组)
2.BEGIN 模块用于 awk 内置 FS,RS 的赋值,打印标题头的信息,要在 awk 执行前,先定义好
3.END 模块用来最后输出,统计信息,awk 数组信息(累加(a++;a+=$0,数组))
4,字段(列),记录(行)

2.18 [放到数组后面]awk 变量

awk 中的变量主要分为两种,一种是我们自己定义的变量,一般是字母,另一种是已经定义好的,叫预定义变量.

2.19 [放到数组后面]用户自定义变量

如图不难发现,awk 数组就和酒店一样,数组的名称就像是酒店名称,数组元素名称就是酒店房间号码,每个数组元素里面的内容就像是酒店里面的人.

2.19.1 举例:数组结构及创建

假设我们的酒店叫老男孩教育酒店

```
老男孩教育酒店<===>oldboyduhotel
```

酒店里面有几个房间,110,119,120,114 这几个房间

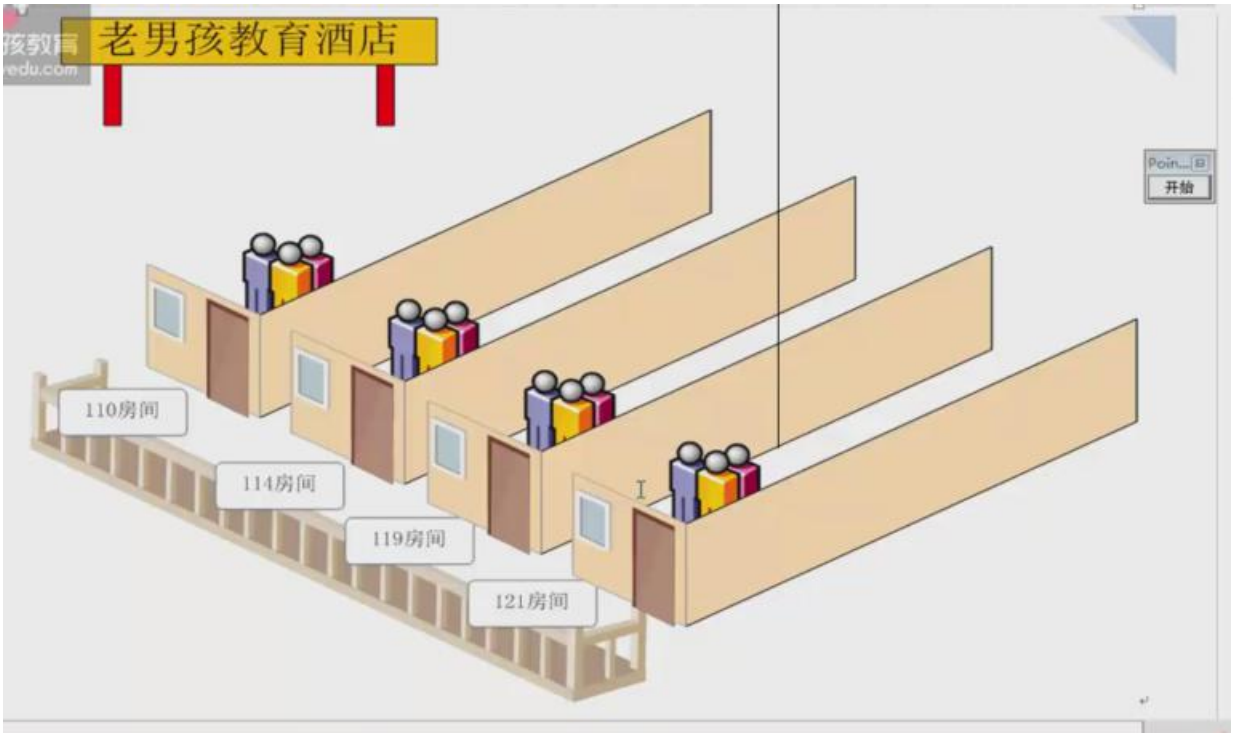
```
老男孩教育酒店<===>oldboyduhotel[101]
老男孩教育酒店<===>oldboyduhotel[119]
老男孩教育酒店<===>oldboyduhotel[120]
老男孩教育酒店<===>oldboyduhotel[114]
```

酒店房间里住的客人

```
老男孩教育酒店的 110 房间住着 xiaoyu<===>oldboyduhotel[110]="xiaoyu"
老男孩教育酒店的 119 房间住着 ruxue<===>oldboyduhotel[119]="xiadao"
老男孩教育酒店的 120 房间住着 xiaoyu<===>oldboyduhotel[120]="dandan"
老男孩教育酒店的 114 房间住着 xiaoyu<===>oldboyduhotel[114]="waiwai"
```

警察去酒店查房,自动循环的查

```
警察======房间号码
老男孩酒店[警察]=====房间里面的人
```



oldboyhotel====>数组的名字=======>酒店

oldboyhotel[110]=====>数组里面的元素===>房间

元素的内容/房间====>住的人

2.19.2 举例:给数组赋值并打印

```
[root@LSD awk]# awk 'BEGIN{hotel[110]="xiaoyu";hotel[114]="xiadao";hotel[121]="dandan";print
hotel[110],hotel[114],hotel[119],hotel[121]}'
xiaoyu xiadao  dandan
[root@LSD awk]# awk 'BEGIN{hotel[110]="xiaoyu";hotel[114]="xiadao";hotel[121]="dandan";print
hotel[110],hotel[114],hotel[119],hotel[121];for(pol in hotel)print pol,hotel[pol]}'
xiaoyu xiadao  dandan
110 xiaoyu
121 dandan
114 xiadao
119
执行过程分解:
awk 'BEGIN{hotel[110]="xiaoyu"
hotel[114]="xiadao"
hotel[121]="dandan"
print hotel[110],hotel[114],hotel[119],hotel[121]
for(pol in hotel)print pol,hotel[pol]}'
```

2.19.3 也可以看下面的图片

awk 数组小结

pol	in	hotel
变量	关键字	数组名
警察	找人	酒店名称
for 循环		
for(police in hotel)		

循环(警察 找人 酒店名称)
police ===>房间号码 ===>数组元素的名字 ===>数组的下标([])==>门把手
hotel[police]===>GPS 定位====>酒店名称[房间号码] hotel["b"] ===> print hotel["b"]

2.20 .企业面试题:统计域名访问次数

处理以下文件内容,将域名取出并根据域名进行计数排序处理

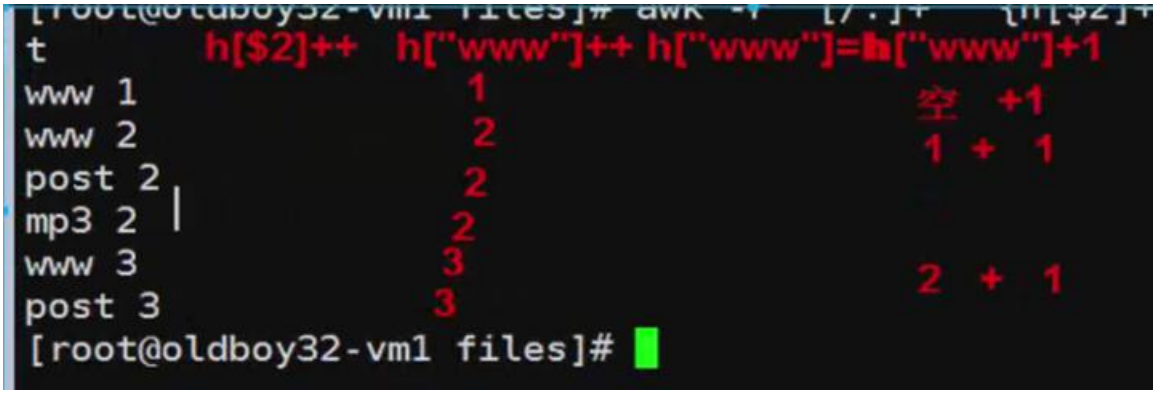
```
http://www.etiantian.org/indel.html
http://www.etiantian.org/1.html
http://post.etiantian.org/indel.html
http://mp3.etiantian.org/indel.html
http://www.etiantian.org/3.html
http://post.etiantian.org/2.html
```

思路:

1. 以斜线为菜刀取出第二列(域名)

```
[root@LSD awk]# awk -F "[/.]+" '{print $2}' url.txt
www
www
post
mp3
www
post
[root@LSD awk]#  awk -F "[/.]+" '{h[$2]++;print h[$2]}' url.txt
1
2
1
1
3
2
[root@LSD awk]#  awk -F "[/.]+" '{h[$2]++;print $2,h[$2]}' url.txt
www 1
www 2
post 1
mp3 1
www 3
post 2
```

过程分解:



最终结果:

```
[root@LSD awk]#  awk -F "[/.]+" '{h[$2]++}END{for(pol in h)print pol,h[pol]}' url.txt
www 3
mp3 1
post 2
[root@LSD awk]#  awk -F "[/]+" '{h[$2]++}END{for(pol in h)print pol,h[pol]}' url.txt
mp3.etiantian.org 1
post.etiantian.org 2
www.etiantian.org 3
```

指定分隔符	单引号	创建 awk 数组	取结果	单引号	文件
awk -F "[/.]+"	'	{hotel[$2]++}	END{for (pol in hotel) print pol,hotel[pol]}	'	url.txt
思路:先处理后输出(END 模块输出)					

```
[root@LSD awk]# awk -v RS="[^a-zA-Z]+" '{h[$0]++}END{for(pol in h)print pol,h[pol]}' awkfile.txt |sort -rnk2|column -t
sbin        12
x           10
nologin     6
bin         5
var         4
uucp        3
.....省略
```

```
[root@LSD awk]# awk -F "[/]+" '$2~/www/{c++;print c}' url.txt
1
2
3
[root@LSD awk]# awk -F "[/]+" '$2~/www/{c++;}$2~/post/{a++;print a}' url.txt
1
2
```

```
[root@LSD awk]# awk -F "[/]+" '$2~/www/{c++;print c}' url.txt
1
2
3
[root@LSD awk]# awk -F "[/]+" '$2~/www/{c++;}$2~/post/{a++;print a}' url.txt
1
2
[root@LSD awk]# #我想要一个方法 一个东西可以装下这三种情况并且互相不影响
[root@LSD awk]# #array[$2]++ >>> array["www"]++ array["post"]++ array["mp3"]++
```

```
[root@LSD awk]# awk -F "[/]+" '{a["www"]++;print $2,a["www"]}' url.txt
www.etiantian.org 1
www.etiantian.org 2
post.etiantian.org 3
mp3.etiantian.org 4
www.etiantian.org 5
post.etiantian.org 6
[root@LSD awk]# awk -F "[/]+" '{a[$2]++;print $2,a["www"]}' url.txt
www.etiantian.org
www.etiantian.org
post.etiantian.org
mp3.etiantian.org
www.etiantian.org
post.etiantian.org
```

```
[root@LSD awk]# awk -F "[/]+" '{a[$2]++;print $2,a["www.etiantian.org"]}' url.txt
```

```
www.etiantian.org 1
www.etiantian.org 2
post.etiantian.org 2
mp3.etiantian.org 2
www.etiantian.org 3
post.etiantian.org 3
```

2.21 总结:

赋值的两种方法:

BEGIN:

-F

-v RS=":"

END 模块输出

配合累加或者 awk 数组

正则