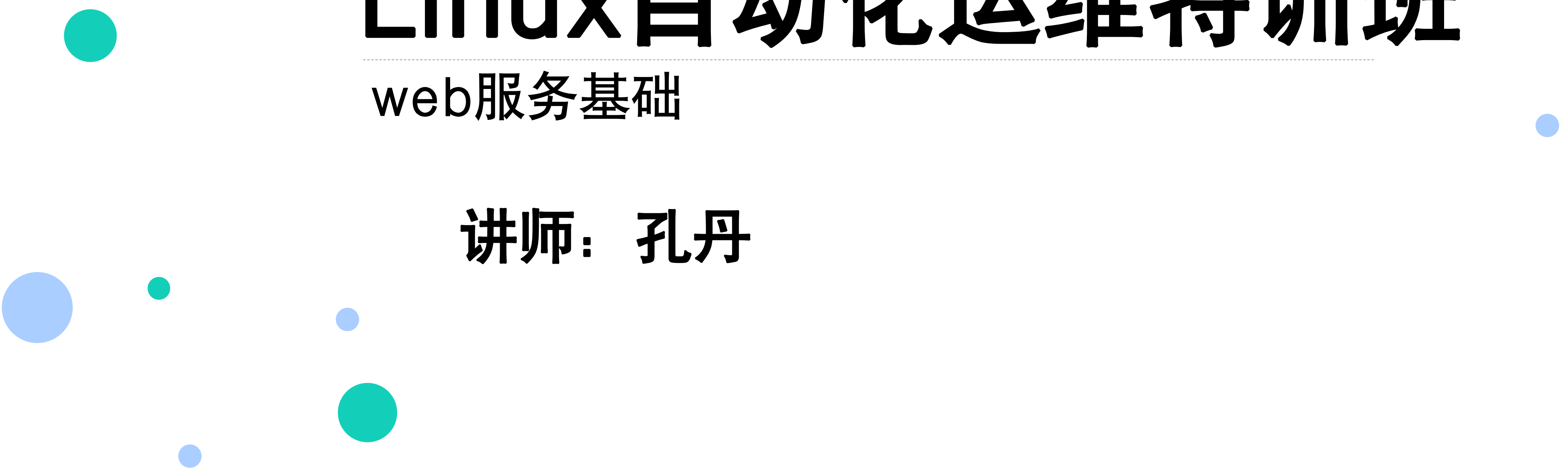




# Linux自动化运维特训班

awk实战

讲师：孔丹



# 大纲

- awk入门：主要介绍awk的功能、工作方式、基本语法以及工作流程等。
- awk的模式匹配：主要介绍awk命令中的匹配模式，包括BEGIN模式、END模式、关系表达式、正则表达式以及混合模式等。
- 变量：主要介绍awk程序中的变量的定义方法，常用的系统变量以及使用系统变量获取记录和字段的值。
- 运算符和表达式：主要介绍各种运算符，包括算术运算符、赋值运算符、条件运算符、逻辑运算符以及关系运算符等。
- 函数：主要介绍awk内置的常用字符串函数和算术函数。
- 数组：主要介绍数组的定义和引用方法以及数组的遍历等。
- 流程控制：主要介绍awk中的if、while、do...while、break、continue、next以及exit等流程控制语句的使用方法。
- awk程序的格式化输出：主要介绍awk程序中的输出语句，包括print以及printf等。
- awk的程序与Shell的交互：主要介绍通过管道和system函数实现awk程序与Shell的交互。

# 什么是awk

- Pattern scanning and text processing language
  - awk没有一个动听的名字，但它是一种很棒的语言。
  - awk适合于文本处理和报表生成。
  - awk是一种一旦学会了就会称为您战略编码库的主要部分的语言。
- 
- awk是一种非常强大的数据处理工具，其本身可以称为是一种程序设计语言，因而具有其他程序设计语言所共同拥有的一些特征，例如变量、函数以及表达式等。通过awk，用户可以编写一些非常实用的文本处理工具。本节将介绍awk的基础知识。

# awk的功能

- ❑ awk是Linux以及UNIX环境中现有的功能最强大的数据处理工具。简单地讲，awk是一种处理文本数据的编程语言。awk的设计使得它非常适合于处理由行和列组成的文本数据。而在Linux或者UNIX环境中，这种类型的数据是非常普遍的。
- ❑ 除此之外，awk还是一种编程语言环境，它提供了正则表达式的匹配，流程控制，运算符，表达式，变量以及函数等一系列的程序设计语言所具备的特性。它从C语言等中获取了一些优秀的思想。awk程序可以读取文本文件，对数据进行排序，对其中的数值执行计算已经生成报表等。

# awk的工作流程

- 对于初学者来说，搞清楚awk的工作流程非常重要。只有在掌握了awk的工作流程之后，才有可能用好awk来处理数据。在awk处理数据时，它会反复执行以下4个步骤：
  - (1) 自动从指定的数据文件中读取行文本。
  - (2) 自动更新awk的内置系统变量的值，例如列数变量NF、行数变量NR、行变量\$0以及各个列变量\$1、\$2等等。
  - (3) 依次执行程序中的所有匹配模式及其操作。
  - (4) 当执行完程序中所有的匹配模式及其操作之后，如果数据文件中仍然还有为读取的数据行，则返回到第（1）步，重复执行（1）~（4）的操作。

# awk程序执行方式

- 1. 通过命令行执行awk程序，语法如下：

`awk 'program-text' datafile`

- 2. 执行awk脚本

在awk程序语句比较多的情况下，用户可以将所有的语句写在一个脚本文件中，然后通过awk命令来解释并执行其中的语句。awk调用脚本的语法如下：

`awk -f program-file file ..`

在上面的语法中，-f选项表示从脚本文件中读取awk程序语句，program-file表示awk脚本文件名称，file表示要处理的数据文件。

- 3. 可执行脚本文件

在上面介绍的两种方式中，用户都需要输入awk命令才能执行程序。除此之外，用户还可以通过类似于Shell脚本的方式来执行awk程序。在这种方式中，需要在awk程序中指定命令解释器，并且赋予脚本文件的可执行权限。其中指定命令解释器的语法如下：

`#!/bin/awk -f`

以上语句必须位于脚本文件的第一行。然后用户就可以通过以下命令执行awk程序：`awk-script file`

其中，awk-script为awk脚本文件名称，file为要处理的文本数据文件。



# awk基本语法

- ❑ awk [options] 'script' file1 file2, ...
- ❑ awk [options] 'PATTERN { action }' file1 file2, ...

# awk基本语法

## □ awk输出

### 1) print的使用格式:

print item1, item2, ...

要点:

1. 各项目之间使用逗号隔开, 而输出时则以空白字符分隔;
2. 输出的item可以为字符串或数值、当前记录的字段(如\$1)、变量或awk的表达式; 数值会先转换为字符串, 而后再输出;
3. print命令后面的item可以省略, 此时其功能相当于print \$0, 因此, 如果想输出空白行, 则需要使用print "";

示例:

```
[root@localhost ~]# awk 'BEGIN { print "line one\nline two\nline three"}'
```

```
line one
```

```
line two
```

```
line three
```

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

```
root 0
```

```
bin 1
```

```
daemon 2
```



# awk基本语法

## □ awk输出

### 2) printf命令的使用格式:

- 1、其与print命令的最大不同是，printf需要指定format;
- 2、format用于指定后面的每个item的输出格式;
- 3、printf语句不会自动打印换行符; \n

format格式的指示符都以%开头，后跟一个字符；如下：

%c: 显示字符的ASCII码;

%d, %i: 十进制整数;

%e, %E: 科学计数法显示数值;

%f: 显示浮点数;

%g, %G: 以科学计数法的格式或浮点数的格式显示数值;

%s: 显示字符串;

%u: 无符号整数;

%%: 显示%自身;

修饰符:

N: 显示宽度;

-: 左对齐;

+: 显示数值符号;

示例:

```
[root@localhost ~]# awk -F: '{printf "%-15s %i\n",$1,$3}' /etc/passwd |head -n 3
```

# awk基本语法

## □ awk输出

### 3) 输出重定向

print items > output-file  
print items >> output-file  
print items | command

特殊文件描述符:

/dev/stdin: 标准输入

/dev/sdtout: 标准输出

/dev/stderr: 错误输出

/dev/fd/N: 某特定文件描述符, 如/dev/stdin就相当于

/dev/fd/0;

示例:

```
[root@localhost ~]# awk -F: '{printf "%-15s %i\n",$1,$3  
> "test1" }' /etc/passwd
```

# awk基本语法

## □ awk变量

### 1) awk内置变量之记录变量

FS: field separator, 读取文件本时, 所使用字段分隔符;

OFS: Output Filed Separator: 输出分割符

awk -F: F指定输入分割符

OFS=" #" 指定输出分割符

示例:

```
[root@localhost ~]# echo "this is" > test.txt
```

```
[root@localhost ~]# awk 'BEGIN {OFS="#"} {print $1,$2,"a","test"}'
```

test.txt

this#is#a#test

### 2) awk内置变量之数据变量

NR: The number of input records, awk命令所处理的记录数; 如果有多个文件, 这个数目会把处理的多个文件中行统一计数;

NF: Number of Field, 当前记录的field个数; 当前行的字段总数

FNR: 与NR不同的是, FNR用于记录正处理的行是当前这一文件中被总共处理的行数; awk可能处理多个文件, 各自文件计数

ENVIRON: 当前shell环境变量及其值的关联数组;

示例:

```
[root@localhost ~]# awk 'BEGIN{print ENVIRON["PATH"]}'
```

# awk基本语法

## □ awk变量

### 3) 用户自定义变量

gawk允许用户自定义自己的变量以便在程序代码中使用，变量名命名规则与大多数编程语言相同，只能使用字母、数字和下划线，且不能以数字开头。gawk变量名称区分字符大小写。

A、在gawk中给变量赋值使用赋值语句进行

示例：

```
[root@localhost ~]# awk 'BEGIN{test="hello";print test}'  
hello
```

B、在命令行中使用赋值变量

gawk命令也可以在“脚本”外为变量赋值，并在脚本中进行引用。例如，上述的例子还可以改写为：

```
[root@localhost ~]# awk -v test="hello" 'BEGIN {print  
test}'  
hello
```

# awk基本语法

## □ awk操作符

### 1) 算术操作符

-x: 负值

+x: 转换为数值

$x^y$ : 次方

$x**y$ : 次方

$x*y$ :

$x/y$ :

$x+y$ :

$x-y$ :

$x\%y$ :

示例:

```
[root@localhost ~]# awk 'BEGIN{x=2;y=3;print x**y,x^y,x*y,x/y,x+y,x-y,x%y}'
```

```
8 8 6 0.666667 5 -1 2
```

# awk基本语法

## □ awk操作符

### 2) 字符串操作符

只有一个，而且不用写出来，用于实现字符串拼接；

示例：

```
[root@localhost ~]# awk 'BEGIN{print "This","is","test"}'
```

```
This is test
```

### 3) 赋值操作符

=、+=、-=、\*=、/=、%=、^=、\*\*=、++、--

需要注意的是，如果某模式为=号，此时使用/=可能会产生语法错误，

应以/[=]/替代；

示例：

```
[root@localhost ~]# awk 'BEGIN{x=2;y=x;printf "%-5s %i\n%-5s
```

```
%i\n","++x=","++x,"--y=","--y}'
```

```
++x= 3
```

```
--y= 1
```



# awk基本语法

## □ awk操作符

### 4) 布尔值

awk中, 任何非0值或非空字符串都为真, 反之就为假;

### 5) 比较操作符

$x < y$  True if  $x$  is less than  $y$ .

$x \leq y$  True if  $x$  is less than or equal to  $y$ .

$x > y$  True if  $x$  is greater than  $y$ .

$x \geq y$  True if  $x$  is greater than or equal to  $y$ .

$x == y$  True if  $x$  is equal to  $y$ .

$x != y$  True if  $x$  is not equal to  $y$ .

$x \sim y$  True if the string  $x$  matches the regexp denoted by  $y$ .

$x !\sim y$  True if the string  $x$  does not match the regexp denoted by  $y$ .

$\text{subscript in array}$  True if the array  $\text{array}$  has an element with the subscript  $\text{subscript}$ .

示例:

### 6) 逻辑关系符

&&

||

# awk基本语法

## □ awk操作符

### 7) 条件表达式

selector?if-true-exp;if-false-exp

if selector; then

if-true-exp

else

if-false-exp

fi

a=3;b=4

a>b?a is max:b ia max

# cat num

5 8

7 2

1 9

6 4

7 2

使用条件测试表达式打印出每行的最大值:

# awk '{max=\$1>\$2?\$1:\$2;print NR,"max =",max}' num

1 max = 8

2 max = 7

3 max = 9

4 max = 6

5 max = 7

# awk基本语法

## □ awk模式

awk 'program' input-file1 input-file2 ...

其中的program为:

pattern { action }

pattern { action }

...

### 1) 常见的模式类型

1、Regexp: 正则表达式, 格式为/regular expression/

2、expression: 表达式, 其值非0或为非空字符时满足条件, 如: \$1 ~ /foo/ 或 \$1 == "uplook", 用运算符~(匹配)和!~(不匹配)。

3、Ranges: 指定的匹配范围, 格式为pat1,pat2

4、BEGIN/END: 特殊模式, 仅在awk命令执行前运行一次或结束前运行一次

5、Empty(空模式): 匹配任意输入行

### 2) 常见的action

1、Expressions:

2、Control statements

3、Compound statements

4、Input statements

5、Output statements

# awk基本语法

## □ awk模式

/正则表达式/：使用通配符的扩展集。

+：匹配其前的单个字符一次以上，是awk自有的元字符，不适用于grep或sed等

?：匹配其前的单个字符1次或0次，是awk自有的元字符，不适用于grep或sed等

关系表达式：可以用下面运算符表中的关系运算符进行操作，可以是字符串或数字的比较，如\$2>\$1选择第二个字段比第一个字段长的行。

模式匹配表达式：

模式，模式：指定一个行的范围。该语法不能包括BEGIN和END模式。

BEGIN：让用户指定在第一条输入记录被处理之前所发生的动作，通常可在这里设置全局变量。

END：让用户在最后一行输入记录被读取之后发生的动作。

# awk基本语法

## □ awk模式

#使用正则

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

```
root
```

```
rpc
```

```
rpcuser
```

#使用BEGIN

```
[root@localhost ~]# awk -F: 'BEGIN {printf "%-15s %-3s %-15s\n","user","uid","shell"} $3==0,$7~/nologin" {printf "%-15s %-3s %-15s\n",$1,$3,$7}' /etc/passwd
```

```
user          uid shell
```

```
root          0    /bin/bash
```

```
bin           1    /sbin/nologin
```

#使用END

```
[root@localhost ~]# awk -F: 'BEGIN {printf "%-15s %-3s %-15s\n","user","uid","shell"} $3==0,$7~/nologin" {printf "%-15s %-3s %-15s\n",$1,$3,$7} END {print "-----End file-----"}' /etc/passwd
```

```
user          uid shell
```

```
root          0    /bin/bash
```

```
bin           1    /sbin/nologin
```

```
-----End file-----
```

# awk控制语句

## □ awk控制语句

### 1) if-else

语法: if(表达式) {语句1} else if(表达式) {语句2} else {语句3}

示例:

```
[root@localhost ~]# awk -F: '{if ($1=="root") printf "%-10s %-15s\n", $1, "Admin"; else printf "%-10s %-15s\n",$1, "Common User"}' /etc/passwd | head -n 3
```

root	Admin
bin	Common User
daemon	Common User

将上面的两个数比较, 打印最大数的用if语句改写:

```
# awk '{if ($1>$2) print NR,"max =", $1;else print NR,"max =",$2}' num
```

1	max = 8
2	max = 7
3	max = 9
4	max = 6
5	max = 7



# awk控制语句

## □ awk控制语句

### 2) while

语法: while(表达式) {语句}

示例:

```
[root@localhost ~]# awk -F: '{i=1;while (i<=3) {print $i;i++}}' /etc/passwd
```

```
[root@localhost ~]# awk -F: '{i=1;while (i<=NF) { if (length($i)>=4) {print $i; i++ } }' /etc/passwd
```

计算1+2+3+...+100累加和

```
# awk 'BEGIN{while(i<=100){sum+=i;i++;}print "sum =",sum}'  
sum = 5050
```

### 3) do-while

语法: do{语句}while(条件)

示例:

```
[root@localhost ~]# awk -F: '{i=1;do {print $i;i++}while(i<=3)}' /etc/passwd | head -n 3
```

计算1+2+3+...+100累加和

```
# awk 'BEGIN{do{sum+=i;i++;}while(i<=100)print "sum =",sum}'  
sum = 5050
```

# awk控制语句

## □ awk控制语句

4) for

格式1:

语法: for(变量;条件;表达式){语句}

示例:

```
[root@localhost ~]# awk -F: '{for(i=1;i<=3;i++) print $i}'  
/etc/passwd
```

```
[root@localhost ~]# awk -F: '{for(i=1;i<=NF;i++) { if  
(length($i)>=4) {print $i}}}' /etc/passwd
```

计算1+2+3+...+100累加和

```
# awk 'BEGIN{for(i=1;i<=100;i++){sum+=i;}print "sum =",sum}'  
sum = 5050
```

for循环还可以用来遍历数组元素:

格式2:

语法: for(变量 in 数组){语句}

示例:

#查看用户的shell

```
[root@localhost ~]# awk -F: '$NF!~/^$/{BASH[$NF]++}END{for(A  
in BASH){printf "%15s:%i\n",A,BASH[A]}}' /etc/passwd
```

# awk控制语句

## □ awk控制语句

### 5) case

语法: switch (expression) { case VALUE or /REGEXP/:  
statement1, statement2,... default: statement1, ...}

### 6) break 和 continue

常用于循环或case语句中

### 7) next

提前结束对本行文本的处理, 并接着处理下一行; 例如, 下面的命令将显示其ID号为奇数的用户:

```
[root@localhost ~]# awk -F: '{if($3%2==0) next;print $1,$3}'  
/etc/passwd |head -n 3  
bin 1  
adm 3  
sync 5
```

# awk使用数组

## □ awk数组

### 1) 数组

`array[index-expression]`

`index-expression`可以使用任意字符串；需要注意的是，如果某数组元素事先不存在，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，要判断某数组中是否存在某元素，需要使用`index in array`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 (var in array) { statement1, ... }
```

其中，`var`用于引用数组下标，而不是元素值；

示例：

```
[root@localhost ~]# netstat -ant | awk '/^tcp/ {++S[$NF]}  
END {for(a in S) print a, S[a]}'  
ESTABLISHED 1  
LISTEN 10
```

每出现一被`/^tcp/`模式匹配到的行，数组`S[$NF]`就加1，`NF`为当前匹配到的行的最后一个字段，此处用其值做为数组`S`的元素索引；

### 2) 删除数组变量

从关系数组中删除数组索引需要使用`delete`命令。使用格式为：

```
delete array[index]
```

# awk内置函数

## □ awk内置函数

`split(string, array [, fieldsep [, seps ] ])`

功能：将string表示的字符串以fieldsep为分隔符进行分隔，并将分隔后的结果保存至array为名的数组中；数组下标为从0开始的序列；

```
# date +%T | awk '{split($0,a,":");print a[1],a[2],a[3]}'
```

`length([string])`

功能：返回string字符串中字符的个数；

```
# awk 'BEGIN{print length("wanmen")}'
```

`substr(string, start [, length])`

功能：取string字符串中的子串，从start开始，取length个；start从1开始计数；

```
# awk 'BEGIN{print substr("wanmen",w,3)}'
```

# awk内置函数

## □ awk内置函数

sysptime()

功能：取系统当前时间

```
# awk 'BEGIN{print sysptime()}'
```

tolower(s)

功能：将s中的所有字母转为小写

```
# awk 'BEGIN{print tolower("WWW.baidu.COM")}'
```

toupper(s)

功能：将s中的所有字母转为大写

```
# awk 'BEGIN{print toupper("WWW.baidu.COM")}'
```



# awk 案例

## □ 系统连接状态

### 1) 查看TCP连接状态

```
netstat -nat | awk '/^tcp/ {print $NF}' | sort | uniq -c | sort -rn  
netstat -nat | awk '/^tcp/ {++state[$NF]}; END {for(key in state) print  
state[key],key}' | sort -nr
```

### 2) 查找请求数20个IP (常用于查找攻击源)

```
netstat -nalp | awk '/^tcp/ {print $5}' | awk -F: '{print $1}' | sort | uniq -c  
| sort -nr | head -n 20  
netstat -nalp | awk '/^tcp/ {print $5}' | awk -F: '{++IP[$1]}; END {for (ip  
in IP) print IP[ip],ip}' | sort -nr | head -n 20
```

### 3) 用tcpdump嗅探80端口的访问看看谁最高

```
#tcpdump -i eth0 -tnn dst port 80 -c 1000 | awk -F"." '{print  
$1"."$2"."$3"."$4}' | sort | uniq -c | sort -nr | head -20
```

# awk 案例

## □ 网站日志分析

以apache为例

1) 获取访问前10位的ip地址

2) 统计404的连接

```
awk '($(NF-1) ~ /404/)' 2015_09_10_access_log | awk  
'{print $(NF-1),$7}' | sort | head
```

# awk 案例

## □ AWK应用实战20例

<1>输出当前系统所有用户的UID:

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

<2>输出当前系统所有用户的UID, 在首行加入UserUid:

```
#awk -F: 'BEGIN{print "UserUid"}{print $3}' /etc/passwd
```

<3>输出当前系统shell为/bin/bash的用户名, 在最后一行加入END

That is last line!!!

```
#awk -F: /bash$/{print $1}END{print "END That is last  
line!!!"}' /etc/passwd
```

<4>输出当前系统上GID为0的用户的用户名

```
# awk -F: '$4==0{print $1}' /etc/passwd
```

<5>输出当前系统上GID大于500的用户的用户名

```
# awk -F: '$4>500{print $1}' /etc/passwd
```

<6>输出当前系统上的所有用户名和UID, 以 " # #" 为分隔符

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

<7>输出/etc/passwd文件中以 ":" 为分隔符的最后一段。

```
# awk -F: '{print $NF}' /etc/passwd
```

<8>对/etc/passwd文件中输出的每一行计数

```
#awk '{print NR,$0}' /etc/passwd
```

<9>对/etc/passwd、/etc/fstab文件中输出的每一行分别计数。

```
# awk '{print FNR,$0}' /etc/passwd /etc/fstab
```

<10>自定义变量

```
# awk -v var="Linux.com.cn" BEGIN'{print var}'
```

# awk 案例

## □ AWK应用实战20例

<11>以printf格式输出用户名, UID、GID

```
# awk -F: '{printf "%-15s %d %8i\n",$1,$3,$4}' /etc/passwd
```

<12>检测当前系统上所有用户, 如果用户名为root输出: Admin

如果用户名不为root输出: Common User

```
# awk -F: '{if ($1=="root") printf "%-15s: %s\n", $1,"Admin";  
else printf "%-15s: %s\n", $1, "Common User"}' /etc/passwd
```

<13> 统计当前系统上UID大于500的用户的个数

```
# awk -F: -v sum=0 '{if ($3>=500) sum++}END{print sum}'  
/etc/passwd
```

<14>读取/etc/passwd文件中的每一行的每一个字段, 输出每个字段中字符个数大于等于四的字段。

```
#awk -F: '{i=1;while (i<=NF) { if (length($i)>=4) {print $i};  
i++ }}' /etc/passwd
```

<15>使用do-while语句输出/etc/passwd中每一行中的前三个字段

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

<16>使用for语句输出/etc/passwd中每一行中的前三个字段

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

# awk 案例

## □ AWK应用实战20例

<17>统计/etc/passwd文件中各种shell的个数

```
#awk -F: '$NF!~/^$/{BASHsum[$NF]++}END{for(A in BASHsum){printf "%-15s:%i\n",A,BASHsum[A]}}' /etc/passwd
```

注释:

\$NF!~/^\$/: 最后一个字段非空

BASHsum[\$NF]++: 最后一个字段相同的加一

<18> 显示当前系统上UID号为偶数的用户名和UID

```
# awk -F: '{if($3%2==1) next;{printf "%-15s%d\n",$1,$3}}' /etc/passwd
```

<19> 统计当前系统上以tcp协议工作的各端口的状态数

```
#netstat -ant | awk '/^tcp/ {++STATE[$NF]} END {for(a in STATE) print a, STATE[a]}'
```

<20>输出/etc/passwd中的每一行以||||隔开, 默认不换行

```
#awk -F: 'BEGIN{ORS="||||"}{print $0}' /etc/passwd
```

# 总结

- awk工作原理

- awk基本语法

- awk应用



# 作业

- 1、获取根分区剩余大小
- 2、获取当前机器ip地址
- 3、统计出apache的access.log中访问量最多的5个IP
- 4、打印/etc/passwd中UID大于500的用户名和uid
- 5、/etc/passwd 中匹配包含root或net或ucp的任意行
- 6、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(百度搜狐面试题)

test.txt

http://www.baidu.com/index.html

http://www.baidu.com/1.html

http://post.baidu.com/index.html

http://mp3.baidu.com/index.html

http://www.baidu.com/3.html

http://post.baidu.com/2.html

# 作业

- 7、请打印出/etc/passwd 第一个域，并且在第一个域所有的内容前面加上“用户帐号：”
- 8、请打印出/etc/passwd 第三个域和第四个域
- 9、请打印第一域，并且打印头部信息为：这个是系统用户，打印尾部信息为："=====
- 10、请打印出第一域匹配daemon的信息.
- 11、请将/etc/passwd 中的root替换成gongda，记住是临时替换输出屏幕看到效果即可.
- 12、请匹配passwd最后一段域bash结尾的信息，有多少条
- 13、请同时匹配passwd文件中，带mail或bash的关键字的信息



# 谢谢观看

更多好课，请关注[万门大学APP](#)

