**1、Linux cat 命令**

[[Linux 命令大全](https://www.runoob.com/linux/linux-command-manual.html)Linux 命令大全](https://www.runoob.com/linux/linux-command-manual.html)

cat（英文全拼：concatenate）命令用于连接文件并打印到标准输出设备上。

**1.1 使用权限**

所有使用者

**1.2 语法格式**

cat [-AbeEnstTuv] [--help] [--version] fileName

**1.3 参数说明：**

**-n 或 --number**：由 1 开始对所有输出的行数编号。

**-b 或 --number-nonblank**：和 -n 相似，只不过对于空白行不编号。

**-s 或 --squeeze-blank**：当遇到有连续两行以上的空白行，就代换为一行的空白行。

**-v 或 --show-nonprinting**：使用 ^ 和 M- 符号，除了 LFD 和 TAB 之外。

**-E 或 --show-ends** : 在每行结束处显示 $。

**-T 或 --show-tabs**: 将 TAB 字符显示为 ^I。

**-A, --show-all**：等价于 -vET。

**-e：**等价于"-vE"选项；

**-t：**等价于"-vT"选项；

**1.4 实例：**

把 textfile1 的文档内容加上行号后输入 textfile2 这个文档里：

cat -n textfile1 > textfile2

把 textfile1 和 textfile2 的文档内容加上行号（空白行不加）之后将内容附加到 textfile3 文档里：

cat -b textfile1 textfile2 >> textfile3

清空 /etc/test.txt 文档内容：

cat /dev/null > /etc/test.txt

cat 也可以用来制作镜像文件。例如要制作软盘的镜像文件，将软盘放好后输入：

cat /dev/fd0 > OUTFILE

相反的，如果想把 image file 写到软盘，输入：

cat IMG\_FILE > /dev/fd0

**注**：

* 1. OUTFILE 指输出的镜像文件名。
* 2. IMG\_FILE 指镜像文件。
* 3. 若从镜像文件写回 device 时，device 容量需与相当。
* 4. 通常用制作开机磁片。

# 2、Linux find 命令

[[Linux 命令大全](https://www.runoob.com/linux/linux-command-manual.html)Linux 命令大全](https://www.runoob.com/linux/linux-command-manual.html)

Linux find 命令用来在指定目录下查找文件。任何位于参数之前的字符串都将被视为欲查找的目录名。如果使用该命令时，不设置任何参数，则 find 命令将在当前目录下查找子目录与文件。并且将查找到的子目录和文件全部进行显示。

### 2.1 语法

find path -option [ -print ] [ -exec -ok command ] {} \;

**参数说明** :

find 根据下列规则判断 path 和 expression，在命令列上第一个 - ( ) , ! 之前的部份为 path，之后的是 expression。如果 path 是空字串则使用目前路径，如果 expression 是空字串则使用 -print 为预设 expression。

expression 中可使用的选项有二三十个之多，在此只介绍最常用的部份。

-mount, -xdev : 只检查和指定目录在同一个文件系统下的文件，避免列出其它文件系统中的文件

-amin n : 在过去 n 分钟内被读取过

-anewer file : 比文件 file 更晚被读取过的文件

-atime n : 在过去 n 天内被读取过的文件

-cmin n : 在过去 n 分钟内被修改过

-cnewer file :比文件 file 更新的文件

-ctime n : 在过去 n 天内创建的文件

-mtime n : 在过去 n 天内修改过的文件

-empty : 空的文件-gid n or -group name : gid 是 n 或是 group 名称是 name

-ipath p, -path p : 路径名称符合 p 的文件，ipath 会忽略大小写

-name name, -iname name : 文件名称符合 name 的文件。iname 会忽略大小写

-size n : 文件大小 是 n 单位，b 代表 512 位元组的区块，c 表示字元数，k 表示 kilo bytes，w 是二个位元组。

-type c : 文件类型是 c 的文件。

d: 目录

c: 字型装置文件

b: 区块装置文件

p: 具名贮列

f: 一般文件

l: 符号连结

s: socket

-pid n : process id 是 n 的文件

你可以使用 ( ) 将运算式分隔，并使用下列运算。

exp1 -and exp2

! expr

-not expr

exp1 -or exp2

exp1, exp2

### 2.2 实例

将当前目录及其子目录下所有文件后缀为 .c 的文件列出来:

# find . -name "\*.c"

将当前目录及其子目录中的所有文件列出：

# find . -type f

将当前目录及其子目录下所有最近 20 天内更新过的文件列出:

# find . -ctime 20

查找 /var/log 目录中更改时间在 7 日以前的普通文件，并在删除之前询问它们：

# find /var/log -type f -mtime +7 -ok rm {} \;

查找当前目录中文件属主具有读、写权限，并且文件所属组的用户和其他用户具有读权限的文件：

# find . -type f -perm 644 -exec ls -l {} \;

查找系统中所有文件长度为 0 的普通文件，并列出它们的完整路径：

# find / -type f -size 0 -exec ls -l {} \;

**3、Linux grep 命令**

[[Linux 命令大全](https://www.runoob.com/linux/linux-command-manual.html)Linux 命令大全](https://www.runoob.com/linux/linux-command-manual.html)

Linux grep 命令用于查找文件里符合条件的字符串。

grep 指令用于查找内容包含指定的范本样式的文件，如果发现某文件的内容符合所指定的范本样式，预设 grep 指令会把含有范本样式的那一列显示出来。若不指定任何文件名称，或是所给予的文件名为 -，则 grep 指令会从标准输入设备读取数据。

**3.1 语法**

grep [-abcEFGhHilLnqrsvVwxy][-A<显示行数>][-B<显示列数>][-C<显示列数>][-d<进行动作>][-e<范本样式>][-f<范本文件>][--help][范本样式][文件或目录...]

**参数**：

* **-a 或 --text** : 不要忽略二进制的数据。
* **-A<显示行数> 或 --after-context=<显示行数>** : 除了显示符合范本样式的那一列之外，并显示该行之后的内容。
* **-b 或 --byte-offset** : 在显示符合样式的那一行之前，标示出该行第一个字符的编号。
* **-B<显示行数> 或 --before-context=<显示行数>** : 除了显示符合样式的那一行之外，并显示该行之前的内容。
* **-c 或 --count** : 计算符合样式的列数。
* **-C<显示行数> 或 --context=<显示行数>或-<显示行数>** : 除了显示符合样式的那一行之外，并显示该行之前后的内容。
* **-d <动作> 或 --directories=<动作>** : 当指定要查找的是目录而非文件时，必须使用这项参数，否则grep指令将回报信息并停止动作。
* **-e<范本样式> 或 --regexp=<范本样式>** : 指定字符串做为查找文件内容的样式。
* **-E 或 --extended-regexp** : 将样式为延伸的正则表达式来使用。
* **-f<规则文件> 或 --file=<规则文件>** : 指定规则文件，其内容含有一个或多个规则样式，让grep查找符合规则条件的文件内容，格式为每行一个规则样式。
* **-F 或 --fixed-regexp** : 将样式视为固定字符串的列表。
* **-G 或 --basic-regexp** : 将样式视为普通的表示法来使用。
* **-h 或 --no-filename** : 在显示符合样式的那一行之前，不标示该行所属的文件名称。
* **-H 或 --with-filename** : 在显示符合样式的那一行之前，表示该行所属的文件名称。
* **-i 或 --ignore-case** : 忽略字符大小写的差别。
* **-l 或 --file-with-matches** : 列出文件内容符合指定的样式的文件名称。
* **-L 或 --files-without-match** : 列出文件内容不符合指定的样式的文件名称。
* **-n 或 --line-number** : 在显示符合样式的那一行之前，标示出该行的列数编号。
* **-o 或 --only-matching** : 只显示匹配PATTERN 部分。
* **-q 或 --quiet或--silent** : 不显示任何信息。
* **-r 或 --recursive** : 此参数的效果和指定"-d recurse"参数相同。
* **-s 或 --no-messages** : 不显示错误信息。
* **-v 或 --invert-match** : 显示不包含匹配文本的所有行。
* **-V 或 --version** : 显示版本信息。
* **-w 或 --word-regexp** : 只显示全字符合的列。
* **-x --line-regexp** : 只显示全列符合的列。
* **-y** : 此参数的效果和指定"-i"参数相同。

[[Linux 命令大全](https://www.runoob.com/linux/linux-command-manual.html)Linux 命令大全](https://www.runoob.com/linux/linux-command-manual.html)

**3.2 实例**

1、在当前目录中，查找后缀有 file 字样的文件中包含 test 字符串的文件，并打印出该字符串的行。此时，可以使用如下命令：

grep test \*file

结果如下所示：

$ grep test test\* #查找前缀有“test”的文件包含“test”字符串的文件

testfile1:This a Linux testfile! #列出testfile1 文件中包含test字符的行

testfile\_2:This is a linux testfile! #列出testfile\_2 文件中包含test字符的行

testfile\_2:Linux test #列出testfile\_2 文件中包含test字符的行

2、以递归的方式查找符合条件的文件。例如，查找指定目录/etc/acpi 及其子目录（如果存在子目录的话）下所有文件中包含字符串"update"的文件，并打印出该字符串所在行的内容，使用的命令为：

grep -r update /etc/acpi

输出结果如下：

$ grep -r update /etc/acpi #以递归的方式查找“etc/acpi”

#下包含“update”的文件

/etc/acpi/ac.d/85-anacron.sh:# (Things like the slocate updatedb cause a lot of IO.)

Rather than

/etc/acpi/resume.d/85-anacron.sh:# (Things like the slocate updatedb cause a lot of

IO.) Rather than

/etc/acpi/events/thinkpad-cmos:action=/usr/sbin/thinkpad-keys--update

3、反向查找。前面各个例子是查找并打印出符合条件的行，通过"-v"参数可以打印出不符合条件行的内容。

查找文件名中包含 test 的文件中不包含test 的行，此时，使用的命令为：

grep -v test \*test\*

结果如下所示：

$ grep-v test\* #查找文件名中包含test 的文件中不包含test 的行

testfile1:helLinux!

testfile1:Linis a free Unix-type operating system.

testfile1:Lin

testfile\_1:HELLO LINUX!

testfile\_1:LINUX IS A FREE UNIX-TYPE OPTERATING SYSTEM.

testfile\_1:THIS IS A LINUX TESTFILE!

testfile\_2:HELLO LINUX!

testfile\_2:Linux is a free unix-type opterating system.

**4、Linux egrep命令**

[[Linux 命令大全](https://www.runoob.com/linux/linux-command-manual.html)Linux 命令大全](https://www.runoob.com/linux/linux-command-manual.html)

Linux egrep命令用于在文件内查找指定的字符串。

egrep执行效果与"grep-E"相似，使用的语法及参数可参照grep指令，与grep的不同点在于解读字符串的方法。

egrep是用extended regular expression语法来解读的，而grep则用basic regular expression 语法解读，extended regular expression比basic regular expression的表达更规范。

**4.1 语法**

egrep [范本模式] [文件或目录]

**参数说明：**

* [范本模式] ：查找的字符串规则。
* [文件或目录] ：查找的目标文件或目录。

**4.2 实例**

显示文件中符合条件的字符。例如，查找当前目录下所有文件中包含字符串"Linux"的文件，可以使用如下命令：

egrep Linux \*

结果如下所示：

$ egrep Linux \* #查找当前目录下包含字符串“Linux”的文件

testfile:hello Linux! #以下五行为testfile 中包含Linux字符的行

testfile:Linux is a free Unix-type operating system.

testfile:This is a Linux testfile!

testfile:Linux

testfile:Linux

testfile1:helLinux! #以下两行为testfile1中含Linux字符的行

testfile1:This a Linux testfile!

#以下两行为testfile\_2 中包含Linux字符的行

testfile\_2:Linux is a free unix-type opterating system.

testfile\_2:Linux test

xx00:hello Linux! #xx00包含Linux字符的行

xx01:Linux is a free Unix-type operating system. #以下三行为xx01包含Linux字符的行

xx01:This is a Linux testfile!

xx01:Linux

# 5、Linux awk 命令

[[Linux 命令大全](https://www.runoob.com/linux/linux-command-manual.html)Linux 命令大全](https://www.runoob.com/linux/linux-command-manual.html)

AWK 是一种处理文本文件的语言，是一个强大的文本分析工具。

之所以叫 AWK 是因为其取了三位创始人 Alfred Aho，Peter Weinberger, 和 Brian Kernighan 的 Family Name 的首字符。

### 5.0 语法

awk [选项参数] 'script' var=value file(s)

或

awk [选项参数] -f scriptfile var=value file(s)

**选项参数说明：**

* -F fs or --field-separator fs  
  指定输入文件折分隔符，fs是一个字符串或者是一个正则表达式，如-F:。
* -v var=value or --asign var=value  
  赋值一个用户定义变量。
* -f scripfile or --file scriptfile  
  从脚本文件中读取awk命令。
* -mf nnn and -mr nnn  
  对nnn值设置内在限制，-mf选项限制分配给nnn的最大块数目；-mr选项限制记录的最大数目。这两个功能是Bell实验室版awk的扩展功能，在标准awk中不适用。
* -W compact or --compat, -W traditional or --traditional  
  在兼容模式下运行awk。所以gawk的行为和标准的awk完全一样，所有的awk扩展都被忽略。
* -W copyleft or --copyleft, -W copyright or --copyright  
  打印简短的版权信息。
* -W help or --help, -W usage or --usage  
  打印全部awk选项和每个选项的简短说明。
* -W lint or --lint  
  打印不能向传统unix平台移植的结构的警告。
* -W lint-old or --lint-old  
  打印关于不能向传统unix平台移植的结构的警告。
* -W posix  
  打开兼容模式。但有以下限制，不识别：/x、函数关键字、func、换码序列以及当fs是一个空格时，将新行作为一个域分隔符；操作符\*\*和\*\*=不能代替^和^=；fflush无效。
* -W re-interval or --re-inerval  
  允许间隔正则表达式的使用，参考(grep中的Posix字符类)，如括号表达式[[:alpha:]]。
* -W source program-text or --source program-text  
  使用program-text作为源代码，可与-f命令混用。
* -W version or --version  
  打印bug报告信息的版本。

## 5.1基本用法

log.txt文本内容如下：

2 this is a test

3 Do you like awk

This's a test

10 There are orange,apple,mongo

用法一：

awk '{[pattern] action}' {filenames} # 行匹配语句 awk '' 只能用单引号

实例：

# 每行按空格或TAB分割，输出文本中的1、4项

$ awk '{print $1,$4}' log.txt

---------------------------------------------

2 a

3 like

This's

10 orange,apple,mongo

# 格式化输出

$ awk '{printf "%-8s %-10s\n",$1,$4}' log.txt

---------------------------------------------

2 a

3 like

This's

10 orange,apple,mongo

用法二：

awk -F #-F相当于内置变量FS, 指定分割字符

实例：

# 使用","分割

$ awk -F, '{print $1,$2}' log.txt

---------------------------------------------

2 this is a test

3 Do you like awk

This's a test

10 There are orange apple

# 或者使用内建变量

$ awk 'BEGIN{FS=","} {print $1,$2}' log.txt

---------------------------------------------

2 this is a test

3 Do you like awk

This's a test

10 There are orange apple

# 使用多个分隔符.先使用空格分割，然后对分割结果再使用","分割

$ awk -F '[ ,]' '{print $1,$2,$5}' log.txt

---------------------------------------------

2 this test

3 Are awk

This's a

10 There apple

用法三：

awk -v # 设置变量

实例：

$ awk -va=1 '{print $1,$1+a}' log.txt

---------------------------------------------

2 3

3 4

This's 1

10 11

$ awk -va=1 -vb=s '{print $1,$1+a,$1b}' log.txt

---------------------------------------------

2 3 2s

3 4 3s

This's 1 This'ss

10 11 10s

用法四：

awk -f {awk脚本} {文件名}

实例：

$ awk -f cal.awk log.txt

## 5.2 运算符

| **运算符** | **描述** |
| --- | --- |
| = += -= \*= /= %= ^= \*\*= | 赋值 |
| ?: | C条件表达式 |
| || | 逻辑或 |
| && | 逻辑与 |
| ~ 和 !~ | 匹配正则表达式和不匹配正则表达式 |
| < <= > >= != == | 关系运算符 |
| 空格 | 连接 |
| + - | 加，减 |
| \* / % | 乘，除与求余 |
| + - ! | 一元加，减和逻辑非 |
| ^ \*\*\* | 求幂 |
| ++ -- | 增加或减少，作为前缀或后缀 |
| $ | 字段引用 |
| in | 数组成员 |

过滤第一列大于2的行

$ awk '$1>2' log.txt #命令

#输出

3 Do you like awk

This's a test

10 There are orange,apple,mongo

过滤第一列等于2的行

$ awk '$1==2 {print $1,$3}' log.txt #命令

#输出

2 is

过滤第一列大于2并且第二列等于'Are'的行

$ awk '$1>2 && $2=="Are" {print $1,$2,$3}' log.txt #命令

#输出

3 Are you

## 5.3 内建变量

| **变量** | **描述** |
| --- | --- |
| $n | 当前记录的第n个字段，字段间由FS分隔 |
| $0 | 完整的输入记录 |
| ARGC | 命令行参数的数目 |
| ARGIND | 命令行中当前文件的位置(从0开始算) |
| ARGV | 包含命令行参数的数组 |
| CONVFMT | 数字转换格式(默认值为%.6g)ENVIRON环境变量关联数组 |
| ERRNO | 最后一个系统错误的描述 |
| FIELDWIDTHS | 字段宽度列表(用空格键分隔) |
| FILENAME | 当前文件名 |
| FNR | 各文件分别计数的行号 |
| FS | 字段分隔符(默认是任何空格) |
| IGNORECASE | 如果为真，则进行忽略大小写的匹配 |
| NF | 一条记录的字段的数目 |
| NR | 已经读出的记录数，就是行号，从1开始 |
| OFMT | 数字的输出格式(默认值是%.6g) |
| OFS | 输出字段分隔符，默认值与输入字段分隔符一致。 |
| ORS | 输出记录分隔符(默认值是一个换行符) |
| RLENGTH | 由match函数所匹配的字符串的长度 |
| RS | 记录分隔符(默认是一个换行符) |
| RSTART | 由match函数所匹配的字符串的第一个位置 |
| SUBSEP | 数组下标分隔符(默认值是/034) |

$ awk 'BEGIN{printf "%4s %4s %4s %4s %4s %4s %4s %4s %4s\n","FILENAME","ARGC","FNR","FS","NF","NR","OFS","ORS","RS";printf "---------------------------------------------\n"} {printf "%4s %4s %4s %4s %4s %4s %4s %4s %4s\n",FILENAME,ARGC,FNR,FS,NF,NR,OFS,ORS,RS}' log.txt

FILENAME ARGC FNR FS NF NR OFS ORS RS

---------------------------------------------

log.txt 2 1 5 1

log.txt 2 2 5 2

log.txt 2 3 3 3

log.txt 2 4 4 4

$ awk -F\' 'BEGIN{printf "%4s %4s %4s %4s %4s %4s %4s %4s %4s\n","FILENAME","ARGC","FNR","FS","NF","NR","OFS","ORS","RS";printf "---------------------------------------------\n"} {printf "%4s %4s %4s %4s %4s %4s %4s %4s %4s\n",FILENAME,ARGC,FNR,FS,NF,NR,OFS,ORS,RS}' log.txt

FILENAME ARGC FNR FS NF NR OFS ORS RS

---------------------------------------------

log.txt 2 1 ' 1 1

log.txt 2 2 ' 1 2

log.txt 2 3 ' 2 3

log.txt 2 4 ' 1 4

# 输出顺序号 NR, 匹配文本行号

$ awk '{print NR,FNR,$1,$2,$3}' log.txt

---------------------------------------------

1 1 2 this is

2 2 3 Are you

3 3 This's a test

4 4 10 There are

# 指定输出分割符

$ awk '{print $1,$2,$5}' OFS=" $ " log.txt

---------------------------------------------

2 $ this $ test

3 $ Are $ awk

This's $ a $

10 $ There $

## 5.4 使用正则，字符串匹配

# 输出第二列包含 "th"，并打印第二列与第四列

$ awk '$2 ~ /th/ {print $2,$4}' log.txt

---------------------------------------------

this a

**~ 表示模式开始。// 中是模式。**

# 输出包含 "re" 的行

$ awk '/re/ ' log.txt

---------------------------------------------

3 Do you like awk

10 There are orange,apple,mongo

## 5.5 忽略大小写

$ awk 'BEGIN{IGNORECASE=1} /this/' log.txt

---------------------------------------------

2 this is a test

This's a test

## 5.6 模式取反

$ awk '$2 !~ /th/ {print $2,$4}' log.txt

---------------------------------------------

Are like

a

There orange,apple,mongo

$ awk '!/th/ {print $2,$4}' log.txt

---------------------------------------------

Are like

a

There orange,apple,mongo

## 5.7 awk脚本

关于 awk 脚本，我们需要注意两个关键词 BEGIN 和 END。

* BEGIN{ 这里面放的是执行前的语句 }
* END {这里面放的是处理完所有的行后要执行的语句 }
* {这里面放的是处理每一行时要执行的语句}

假设有这么一个文件（学生成绩表）：

$ cat score.txt

Marry 2143 78 84 77

Jack 2321 66 78 45

Tom 2122 48 77 71

Mike 2537 87 97 95

Bob 2415 40 57 62

我们的 awk 脚本如下：

$ cat cal.awk

#!/bin/awk -f

#运行前

BEGIN {

math = 0

english = 0

computer = 0

printf "NAME NO. MATH ENGLISH COMPUTER TOTAL\n"

printf "---------------------------------------------\n"

}

#运行中

{

math+=$3

english+=$4

computer+=$5

printf "%-6s %-6s %4d %8d %8d %8d\n", $1, $2, $3,$4,$5, $3+$4+$5

}

#运行后

END {

printf "---------------------------------------------\n"

printf " TOTAL:%10d %8d %8d \n", math, english, computer

printf "AVERAGE:%10.2f %8.2f %8.2f\n", math/NR, english/NR, computer/NR

}

我们来看一下执行结果：

$ awk -f cal.awk score.txt

NAME NO. MATH ENGLISH COMPUTER TOTAL

---------------------------------------------

Marry 2143 78 84 77 239

Jack 2321 66 78 45 189

Tom 2122 48 77 71 196

Mike 2537 87 97 95 279

Bob 2415 40 57 62 159

---------------------------------------------

TOTAL: 319 393 350

AVERAGE: 63.80 78.60 70.00

## 5.8 另外一些实例

AWK 的 hello world 程序为：

BEGIN { print "Hello, world!" }

计算文件大小

$ ls -l \*.txt | awk '{sum+=$5} END {print sum}'

--------------------------------------------------

666581

从文件中找出长度大于 80 的行：

awk 'length>80' log.txt

打印九九乘法表

seq 9 | sed 'H;g' | awk -v RS='' '{for(i=1;i<=NF;i++)printf("%dx%d=%d%s", i, NR, i\*NR, i==NR?"\n":"\t")}'

更多内容：

* [AWK 工作原理](https://www.runoob.com/w3cnote/awk-work-principle.html)
* [AWK 数组](https://www.runoob.com/w3cnote/awk-arrays.html)
* [AWK 条件语句与循环](https://www.runoob.com/w3cnote/awk-if-loop.html)
* [AWK 用户自定义函数](https://www.runoob.com/w3cnote/awk-user-defined-functions.html)
* [AWK 内置函数](https://www.runoob.com/w3cnote/awk-built-in-functions.html)
* [8 个有力的 Awk 内建变量](https://www.runoob.com/w3cnote/8-awesome-awk-built-in-variables.html)
* [AWK 官方手册](http://www.gnu.org/software/gawk/manual/gawk.html)

## 5.9 AWK 内置函数

### 5.9.1 分类 [编程技术](https://www.runoob.com/w3cnote_genre/code)

AWK 内置函数主要有以下几种：

* [算数函数](https://www.runoob.com/w3cnote/awk-built-in-functions.html#b1)
* [字符串函数](https://www.runoob.com/w3cnote/awk-built-in-functions.html#b2)
* [时间函数](https://www.runoob.com/w3cnote/awk-built-in-functions.html#b3)
* [位操作函数](https://www.runoob.com/w3cnote/awk-built-in-functions.html#b4)
* [其它函数](https://www.runoob.com/w3cnote/awk-built-in-functions.html#b5)

### 5.9.2 算数函数

|  |  |  |
| --- | --- | --- |
| **函数名** | **说明** | **实例** |
| atan2( y, x ) | 返回 y/x 的反正切。 | $ awk 'BEGIN {  PI = 3.14159265  x = -10  y = 10  result = atan2 (y,x) \* 180 / PI;  printf "The arc tangent for (x=%f, y=%f) is %f degrees\n", x, y, result  }'  输出结果为：  The arc tangent for (x=-10.000000, y=10.000000) is 135.000000 degrees |
| cos( x ) | 返回 x 的余弦；x 是弧度。 | $ awk 'BEGIN {  PI = 3.14159265  param = 60  result = cos(param \* PI / 180.0);  printf "The cosine of %f degrees is %f.\n", param, result  }'  输出结果为：  The cosine of 60.000000 degrees is 0.500000. |
| sin( x ) | 返回 x 的正弦；x 是弧度。 | $ awk 'BEGIN {  PI = 3.14159265  param = 30.0  result = sin(param \* PI /180)  printf "The sine of %f degrees is %f.\n", param, result  }'  输出结果为：  The sine of 30.000000 degrees is 0.500000. |
| exp( x ) | 返回 x 幂函数。 | $ awk 'BEGIN {  param = 5  result = exp(param);  printf "The exponential value of %f is %f.\n", param, result  }'  输出结果为：  The exponential value of 5.000000 is 148.413159. |
| log( x ) | 返回 x 的自然对数。 | $ awk 'BEGIN {  param = 5.5  result = log (param)  printf "log(%f) = %f\n", param, result  }'  输出结果为：  log(5.500000) = 1.704748 |
| sqrt( x ) | 返回 x 平方根。 | $ awk 'BEGIN {  param = 1024.0  result = sqrt(param)  printf "sqrt(%f) = %f\n", param, result  }'  输出结果为：  sqrt(1024.000000) = 32.000000 |
| int( x ) | 返回 x 的截断至整数的值。 | $ awk 'BEGIN {  param = 5.12345  result = int(param)  print "Truncated value =", result  }'  输出结果为：  Truncated value = 5 |
| rand( ) | 返回任意数字 n，其中 0 <= n < 1。 | $ awk 'BEGIN {  print "Random num1 =" , rand()  print "Random num2 =" , rand()  print "Random num3 =" , rand()  }'  输出结果为：  Random num1 = 0.237788  Random num2 = 0.291066  Random num3 = 0.845814 |
| srand( [Expr] ) | 将 rand 函数的种子值设置为 Expr 参数的值，或如果省略 Expr 参数则使用某天的时间。返回先前的种子值。 | $ awk 'BEGIN {  param = 10  printf "srand() = %d\n", srand()  printf "srand(%d) = %d\n", param, srand(param)  }'  输出结果为：  srand() = 1  srand(10) = 1417959587 |

### 5.9.3 字符串函数

|  |  |  |
| --- | --- | --- |
| **函数** | **说明** | **实例** |
| gsub( Ere, Repl, [ In ] ) | gsub 是全局替换( global substitution )的缩写。除了正则表达式所有具体值被替代这点，它和 sub 函数完全一样地执行。 | $ awk 'BEGIN {  str = "Hello, World"  print "String before replacement = " str  gsub("World", "Jerry", str)  print "String after replacement = " str  }'  输出结果为：  String before replacement = Hello, World  String after replacement = Hello, Jerry |
| sub(regex,sub,string) | sub 函数执行一次子串替换。它将第一次出现的子串用 regex 替换。第三个参数是可选的，默认为 $0。 | $ awk 'BEGIN {  str = "Hello, World"  print "String before replacement = " str  sub("World", "Jerry", str)  print "String after replacement = " str  }'  输出结果为：  String before replacement = Hello, World  String after replacement = Hello, Jerry |
| substr(str, start, l) | substr 函数返回 str 字符串中从第 start 个字符开始长度为 l 的子串。如果没有指定 l 的值，返回 str 从第 start 个字符开始的后缀子串。 | $ awk 'BEGIN {  str = "Hello, World !!!"  subs = substr(str, 1, 5)  print "Substring = " subs  }'  输出结果为：  Substring = Hello |
| index( String1, String2 ) | 在由 String1 参数指定的字符串（其中有出现 String2 指定的参数）中，返回位置，从 1 开始编号。如果 String2 参数不在 String1 参数中出现，则返回 0（零）。 | $ awk 'BEGIN {  str = "One Two Three"  subs = "Two"  ret = index(str, subs)  printf "Substring \"%s\" found at %d location.\n", subs, ret  }'  输出结果为：  Substring "Two" found at 5 location. |
| length [(String)] | 返回 String 参数指定的字符串的长度（字符形式）。如果未给出 String 参数，则返回整个记录的长度（$0 记录变量）。 | $ awk 'BEGIN {  str = "Hello, World !!!"  print "Length = ", length(str)  }'  输出结果为：  Substring "Two" found at 5 location. |
| blength [(String)] | 返回 String 参数指定的字符串的长度（以字节为单位）。如果未给出 String 参数，则返回整个记录的长度（$0 记录变量）。 |  |
| substr( String, M, [ N ] ) | 返回具有 N 参数指定的字符数量子串。子串从 String 参数指定的字符串取得，其字符以 M 参数指定的位置开始。M 参数指定为将 String 参数中的第一个字符作为编号 1。如果未指定 N 参数，则子串的长度将是 M 参数指定的位置到 String 参数的末尾 的长度。 | $ awk 'BEGIN {  str = "Hello, World !!!"  subs = substr(str, 1, 5)  print "Substring = " subs  }'  输出结果为：  Substring = Hello |
| match( String, Ere ) | 在 String 参数指定的字符串（Ere 参数指定的扩展正则表达式出现在其中）中返回位置（字符形式），从 1 开始编号，或如果 Ere 参数不出现，则返回 0（零）。RSTART 特殊变量设置为返回值。RLENGTH 特殊变量设置为匹配的字符串的长度，或如果未找到任何匹配，则设置为 -1（负一）。 | $ awk 'BEGIN {  str = "One Two Three"  subs = "Two"  ret = match(str, subs)  printf "Substring \"%s\" found at %d location.\n", subs, ret  }'  输出结果为：  Substring "Two" found at 5 location. |
| split( String, A, [Ere] ) | 将 String 参数指定的参数分割为数组元素 A[1], A[2], . . ., A[n]，并返回 n 变量的值。此分隔可以通过 Ere 参数指定的扩展正则表达式进行，或用当前字段分隔符（FS 特殊变量）来进行（如果没有给出 Ere 参数）。除非上下文指明特定的元素还应具有一个数字值，否则 A 数组中的元素用字符串值来创建。 | $ awk 'BEGIN {  str = "One,Two,Three,Four"  split(str, arr, ",")  print "Array contains following values"  for (i in arr) {  print arr[i]  }  }'  输出结果为：  Array contains following values  One  Two  Three  Four |
| tolower( String ) | 返回 String 参数指定的字符串，字符串中每个大写字符将更改为小写。大写和小写的映射由当前语言环境的 LC\_CTYPE 范畴定义。 | $ awk 'BEGIN {  str = "HELLO, WORLD !!!"  print "Lowercase string = " tolower(str)  }'  输出结果为：  Lowercase string = hello, world !!! |
| toupper( String ) | 返回 String 参数指定的字符串，字符串中每个小写字符将更改为大写。大写和小写的映射由当前语言环境的 LC\_CTYPE 范畴定义。 | $ awk 'BEGIN {  str = "hello, world !!!"  print "Uppercase string = " toupper(str)  }'  输出结果为：  Uppercase string = HELLO, WORLD !!! |
| sprintf(Format, Expr, Expr, . . . ) | 根据 Format 参数指定的 printf 子例程格式字符串来格式化 Expr 参数指定的表达式并返回最后生成的字符串。 | $ awk 'BEGIN {  str = sprintf("%s", "Hello, World !!!")  print str  }'  输出结果为：  Hello, World !!! |
| strtonum(str) | strtonum 将字符串 str 转换为数值。 如果字符串以 0 开始，则将其当作八进制数；如果字符串以 0x 或 0X 开始，则将其当作十六进制数；否则，将其当作浮点数。 | $ awk 'BEGIN {  print "十进制数 = " strtonum("123")  print "八进制数 = " strtonum("0123")  print "十六进制数 = " strtonum("0x123")  }'  输出结果为：  十进制数 = 123  八进制数 = 83  十六进制数 = 291 |

**注：**Ere 部分可以是正则表达式。

**1、gsub、sub 使用**

$ awk 'BEGIN{info="this is a test2012test!";gsub(/[0-9]+/,"||",info);print info}'

this is a test||test!

**2、查找字符串（index 使用）**

使用了三元运算符: 表达式 ? 动作1 : 动作2

$ awk 'BEGIN{info="this is a test2012test!";print index(info,"11111")?"ok":"no found";}'

no found

$ awk 'BEGIN{info="this is a test2012test!";print index(info,"is")?"ok":"no found";}'

ok

$ awk 'BEGIN{info="this is a test2012test!";print index(info,"test")?"ok":"no found";}'

ok

**3、正则表达式匹配查找(match 使用）**

$ awk 'BEGIN{info="this is a test2012test!";print match(info,/[0-9]+/)?"ok":"no found";}'

ok

**4、截取字符串(substr使用）**

从第 4 个 字符开始，截取 10 个长度字符串。

$ awk 'BEGIN{info="this is a test2012test!";print substr(info,4,10);}'

s is a tes

**5、字符串分割（split使用）**

$ awk 'BEGIN{info="this is a test";split(info,tA," ");print length(tA);for(k in tA){print k,tA[k];}}'

4

2 is

3 a

4 test

1 this

分割 info，将 info 字符串使用空格切分为动态数组 tA。注意 awk for …in 循环，是一个无序的循环。 并不是从数组下标 1…n ，因此使用时候需要特别注意。

**6、格式化字符串输出（sprintf使用）**

其中格式化字符串包括两部分内容: 一部分是正常字符，这些字符将按原样输出; 另一部分是格式化规定字符， 以 % 开始, 后跟一个或几个规定字符，用来确定输出内容格式。 需要特别注意的是使用 **printf** 时默认是不会换行的，而 **print** 函数默认会在每行后面加上 \n 换行符。

|  |  |
| --- | --- |
| **格式符** | **说明** |
| %d | 十进制有符号整数 |
| %u | 十进制无符号整数 |
| %f | 浮点数 |
| %s | 字符串 |
| %c | 单个字符 |
| %p | 指针的值 |
| %e | 指数形式的浮点数 |
| %x | %X 无符号以十六进制表示的整数 |
| %o | 无符号以八进制表示的整数 |
| %g | 自动选择合适的表示法 |

$ awk 'BEGIN{n1=124.113;n2=-1.224;n3=1.2345; printf("%.2f,%.2u,%.2g,%X,%o\n",n1,n2,n3,n1,n1);}'

124.11,4294967295,1.2,7C,174

$ awk 'BEGIN{n1=124.113;n2=-1.224;n3=1.2645; printf("%.2f,%.2u,%.2g,%X,%o\n",n1,n2,n3,n1,n1);}'

124.11,4294967295,1.3,7C,174

**注：**看上面的 n3 输出值会发现，在使用 printf 处理时一个比较智能的功能是可以进行四舍五入保留小数点位的。

### 5.9.4 时间函数

|  |  |  |
| --- | --- | --- |
| **函数名** | **说明** | **实例** |
| mktime( YYYY MM DD HH MM SS[ DST]) | 生成时间格式 | $ awk 'BEGIN {  print "Number of seconds since the Epoch = " mktime("2014 12 14 30 20 10")  }'  输出结果为：  Number of seconds since the Epoch = 1418604610 |
| strftime([format [, timestamp]]) | 格式化时间输出，将时间戳转为时间字符串  具体格式，见下表. | $ awk 'BEGIN {  print strftime("Time = %m/%d/%Y %H:%M:%S", systime())  }'  输出结果为：  Time = 12/14/2014 22:08:42 |
| systime() | 得到时间戳,返回从1970年1月1日开始到当前时间(不计闰年)的整秒数 | awk 'BEGIN{now=systime();print now}'  输出结果为：  1343210982 |

strftime 日期和时间格式说明符:

| **序号** | **描述** |
| --- | --- |
| %a | 星期缩写(Mon-Sun)。 |
| %A | 星期全称（Monday-Sunday）。 |
| %b | 月份缩写（Jan）。 |
| %B | 月份全称（January）。 |
| %c | 本地日期与时间。 |
| %C | 年份中的世纪部分，其值为年份整除100。 |
| %d | 十进制日期(01-31) |
| %D | 等价于 %m/%d/%y. |
| %e | 日期，如果只有一位数字则用空格补齐 |
| %F | 等价于 %Y-%m-%d，这也是 ISO 8601 标准日期格式。 |
| %g | ISO8610 标准周所在的年份模除 100（00-99)。比如，1993 年 1 月 1 日属于 1992 年的第 53 周。所以，虽然它是 1993 年第 1 天，但是其　ISO8601 标准周所在年份却是 1992。同样，尽管 1973 年 12 月 31 日属于 1973 年但是它却属于 1994 年的第一周。所以 1973 年 12 月 31 日的 ISO8610　标准周所在的年是 1974 而不是 1973。 |
| %G | ISO 标准周所在年份的全称。 |
| %h | 等价于 %b. |
| %H | 用十进制表示的 24 小时格式的小时(00-23) |
| %I | 用十进制表示的 12 小时格式的小时（00-12） |
| %j | 一年中的第几天（001-366） |
| %m | 月份（01-12） |
| %M | 分钟数（00-59) |
| %n | 换行符 (ASCII LF) |
| %p | 十二进制表示法（AM/PM） |
| %r | 十二进制表示法的时间（等价于 %I:%M:%S %p）。 |
| %R | 等价于 %H:%M。 |
| %S | 时间的秒数值（00-60） |
| %t | 制表符 (tab) |
| %T | 等价于 %H:%M:%S。 |
| %u | 以数字表示的星期(1-7),1 表示星期一。 |
| %U | 一年中的第几个星期（第一个星期天作为第一周的开始），00-53 |
| %V | 一年中的第几个星期（第一个星期一作为第一周的开始），01-53。 |
| %w | 以数字表示的星期（0-6），0表示星期日 。 |
| %W | 十进制表示的一年中的第几个星期（第一个星期一作为第一周的开始），00-53。 |
| %x | 本地日期表示 |
| %X | 本地时间表示 |
| %y | 年份模除 100。 |
| %Y | 十进制表示的完整年份。 |
| %z | 时区，表示格式为+HHMM（例如，格式要求生成的 RFC 822或者 RFC 1036 时间头） |
| %Z | 时区名称或缩写，如果时区待定则无输出。 |

### 5.9.5 位操作函数

|  |  |  |
| --- | --- | --- |
| **函数名** | **说明** | **实例** |
| and | 位与操作。 | $ awk 'BEGIN {  num1 = 10  num2 = 6  printf "(%d AND %d) = %d\n", num1, num2, and(num1, num2)  }'  输出结果为：  (10 AND 6) = 2 |
| compl | 按位求补。 | $ awk 'BEGIN {  num1 = 10  printf "compl(%d) = %d\n", num1, compl(num1)  }'  输出结果为：  compl(10) = 9007199254740981 |
| lshift | 左移位操作 | $ awk 'BEGIN {  num1 = 10  printf "lshift(%d) by 1 = %d\n", num1, lshift(num1, 1)  }'  输出结果为：  lshift(10) by 1 = 20 |
| rshift | 右移位操作 | $ awk 'BEGIN {  num1 = 10  printf "rshift(%d) by 1 = %d\n", num1, rshift(num1, 1)  }'  输出结果为：  rshift(10) by 1 = 5 |
| or | 按位或操作 | $ awk 'BEGIN {  num1 = 10  num2 = 6  printf "(%d OR %d) = %d\n", num1, num2, or(num1, num2)  }'  输出结果为：  (10 OR 6) = 14 |
| xor | 按位异或操作 | $ awk 'BEGIN {  num1 = 10  num2 = 6  printf "(%d XOR %d) = %d\n", num1, num2, xor(num1, num2)  }'  输出结果为：  (10 bitwise xor 6) = 12 |

### 5.9.6 其他函数

|  |  |  |
| --- | --- | --- |
| **函数名** | **说明** | **实例** |
| close(expr) | 关闭管道的文件 | $ awk 'BEGIN {  cmd = "tr [a-z] [A-Z]"  print "hello, world !!!" |& cmd  close(cmd, "to")  cmd |& getline out  print out;  close(cmd);  }'  输出结果为：  HELLO, WORLD !!!   * 第一条语句 cmd = "tr [a-z] [A-Z]" 在　AWK 中建立了一个双向的通信通道。 * 第二条语句 print 为 tr 命令提供输入。&| 表示双向通信。 * 第三条语句 close(cmd, "to") 完成执行后关闭 to 进程。 * 第四条语句 cmd |& getline out 使用 getline 函数将输出存储到 out 变量中。 * 接下来的输出语句打印输出的内容，最后 close 函数关闭 cmd。 |
| delete | 用于从数组中删除元素 | $ awk 'BEGIN {  arr[0] = "One"  arr[1] = "Two"  arr[2] = "Three"  arr[3] = "Four"  print "Array elements before delete operation:"  for (i in arr) {  print arr[i]  }  delete arr[0]  delete arr[1]  print "Array elements after delete operation:"  for (i in arr) {  print arr[i]  }  }'  输出结果为：  Array elements before delete operation:  One  Two  Three  Four  Array elements after delete operation:  Three  Four |
| exit | 终止脚本执行，它可以接受可选的参数 expr 传递 AWK 返回状态。 | $ awk 'BEGIN {  print "Hello, World !!!"  exit 10  print "AWK never executes this statement."  }'  输出结果为：  Hello, World !!! |
| flush | 刷新打开文件或管道的缓冲区 |  |
| getline | 读入下一行 | 使用 getline 从文件 marks.txt 中读入一行并输出：  $ awk '{getline; print $0}' marks.txt  ，AWK 从文件 marks.txt 中读入一行存储到变量 0 中。在下一条语句中，我们使用 getline 读入下一行。因此AWK读入第二行并存储到 0 中。最后，AWK 使用 print 输出第二行的内容。这个过程一直到文件结束。 |
| next | 停止处理当前记录，并且进入到下一条记录的处理过程。 | 当模式串匹配成功后程序并不执行任何操作：  $ awk '{if ($0 ~/Shyam/) next; print $0}' marks.txt |
| nextfile | 停止处理当前文件，从下一个文件第一个记录开始处理。 | 首先创建两个文件。 file1.txt 内容如下:  file1:str1  file1:str2  file1:str3  file1:str4  文件 file2.txt 内容如下：  file2:str1  file2:str2  file2:str3  file2:str4  现在我们来测试 nextfile 函数。  $ awk '{ if ($0 ~ /file1:str2/) nextfile; print $0 }' file1.txt file2.txt  输出结果为：  file1:str1  file2:str1  file2:str2  file2:str3  file2:str4 |
| return | 从用户自定义的函数中返回值。请注意，如果没有指定返回值，那么的返回值是未定义的。 | 创建文件 functions.awk，内容如下：  function addition(num1, num2)  {  result = num1 + num2  return result  }  BEGIN {  res = addition(10, 20)  print "10 + 20 = " res  }  执行该文件：  $ awk -f functions.awk  10 + 20 = 30 |
| system | 执行特定的命令然后返回其退出状态。返回值为 0 表示命令执行成功；非 0 表示命令执行失败。 | $ awk 'BEGIN { ret = system("date"); print "Return value = " ret }'  输出结果为：  Sun Dec 21 23:16:07 IST 2014  Return value = 0 |