



zkk

码龄7年

 暂无认证

10

原创

58万+

周排名

43万+

总排名

3527

访问



等级

95

积分

2

粉丝

0

获赞

0

评论

5

收藏



私信

关注

搜博文文章



- 热门文章
- 第六节、用户身份与权限

856
- 第十节、grep、find、sed和awk

521
- 第三节、Linux基础命令

471
- 第七节、文件系统与磁盘分区

421
- 第五节、Vim编辑器与Shell入门

292

分类专栏

 linux系统基础篇

10篇

 linux系统服务篇

您愿意向朋友推荐“博客详情页”吗？











强烈不推荐

不推荐

一般般

推荐

强烈推荐

最新文章

第九节、iptables与Firewalld	
第八节、RAID与LVM	
第七节、文件系统与磁盘分区	
2019年 6篇	2018年 4篇

目录

何谓正则表达式

支持 RE 的文本处理工具

正则表达式的组成

grep

grep 使用举例

find

find命令使用举例

sed

sed 使用举例

awk

awk 使用举例

第十节、grep、find、sed和awk

原创

zkk

于 2019-01-10 17:05:23 发布

524

收藏 2


版权

分类专栏：

linux系统基础篇

文章标签：

Linux运维基础

 linux系统基础篇 专栏收录该内容

0 订阅 10 篇文章

订阅专栏

何谓正则表达式

正则表达式是使用某种模式（pattern）去匹配（matching）一类字符串的一个公式。通常使用正则表达式进行查找、替换等操作。虽然复杂的正则表达式对于初学者来说晦涩难懂，但对于 **Linux** 使用者来说，学会正则表达式的使用是非常必要的。在适当的情况下使用正则表达式可以极大地提高工作效率。

有两种风格的正则表达式：

- POSIX 风格的正则表达式
- Perl 风格的正则表达式（Perl-compatible regular expression）

支持 RE 的文本处理工具

本节主要介绍 POSIX 风格的正则表达式及其工具的使用。下面列出 Linux 下常用的支持 POSIX 风格正则表达式的工具：基本的正则表达式 Basic regular expression (BRE)

- grep 按模式匹配文本
- ed 一个原始的行编辑器
- sed 一个流编辑器
- vim 一个屏幕编辑器
- emacs 一个屏幕编辑器

扩展的正则表达式 Extended regular expression (ERE)

- egrep 按模式匹配文本
- awk 进行简单的文本处理

正则表达式的组成

正则表达式由一些普通字符和一些元字符（metacharacters）组成。普通字符包括大小写的字母、数字（即所有非元字符），而元字符则具有特殊的含义。

下面列出 POSIX RE 的元字符及其含义：

特殊字符	含义	类型	举例	说明
^	匹配首字符	BRE	^x	以字符x开始的字符串
\$	匹配尾字符	BRE	x\$	以x字符结尾的字符串
.	匹配任意一个字符	BRE	l...e	love, life, live ...
?	匹配任意一个可选字符	ERE	xy?	x, xy
*	匹配零次或多次重复	BRE	xy*	x, xy, xyy, xyyy ...
+	匹配一次或多次重复	ERE	xy+	xy, xyy, xyyy ...
[...]	匹配任意一个字符	BRE	[xyz]	x, y, z
()	对正则表达式分组	ERE	(xy)+	xy, xyxy, xyxyxy, ...
{n}	匹配n次	BRE	go{2}gle	google
{n,}	匹配最少n次	BRE	go{2,}gle	google, gooogle, goooogle ...
{n,m}	匹配n到m次	BRE	go{2,4}gle	google, gooogle, goooogle
{n}	匹配n次	ERE	go{2}gle	google
{n,}	匹配最少n次	ERE	go{2,}gle	google, gooogle, goooogle ...
{n,m}	匹配n到m次	ERE	go{2,4}gle	google, gooogle, goooogle
	以或逻辑连接多个匹配	ERE	good	bon
\	转义字符	BRE	*	*

特殊字符	含义	类型	举例	说明
^	非（仅用于启始字符）	BRE	[^xyz]	匹配xyz之外的任意一个字符
-	用于指明字符范围（不能是首字符和尾字符）	BRE	[a-zA-Z]	匹配任意一个字母
\	转义字符	BRE	[.]	.

grep

grep是一个强大的文本搜索工具。grep 使用正则表达式搜索文本，并把匹配的行打印出来。UNIX 的 grep 家族包括 grep、egrep 和 fgrep：

- grep 使用 Basic regular expression (BRE) 书写匹配模式
- egrep 使用 Extended regular expression (ERE) 书写匹配模式，等效于 grep -E
- fgrep 不使用任何正则表达式书写匹配模式（以固定字符串对待），执行快速搜索，等效于 grep -F

grep 命令的格式如下：

- grep [options] PATTERN [FILE...]

其中：

- PATTERN 是用正则表达式书写的模式
- FILE 是要查找的文件，可以用空格间隔的多个文件，省略时表示在标准输入查找
- 常用的参数：
 - -c：只显示匹配行的次数

- -i : 搜索时不区分大小写
 - -n : 输出匹配行的行号
 - -v : 输出不匹配的行（反向选择）
 - -A NUM : 同时输出匹配行的后 NUM 行
 - -B NUM : 同时输出匹配行的前 NUM 行
 - -C NUM : 同时输出匹配行的前、后各 NUM 行
- FILE 可是使用 Shell 的通配符在多个文件中查找 PATTERN
- 通常必须用单引号将整个模式 PATTERN 括起来
- grep 命令不会对输入文件进行任何修改或影响，可以使用输出重定向将结果存为文件

grep 使用举例

在文件 myfile 中查找包含指定的字符串的行
如果使用 grep 命令查找指定的字符串（不使用正则表达式），PATTERN 可以不用单引号括起来

```
1 | [root@zkk ~]# grep osmond myfile
2 | [root@zkk ~]# fgrep osmond myfile
```

在文件 myfile1 myfile2 myfile3 中查找包含指定的字符串的行

```
1 | [root@zkk ~]# grep 'osmond' myfile1 myfile2 myfile3
2 | [root@zkk ~]# fgrep 'osmond' myfile1 myfile2 myfile3
```

在当前目录的所有文件中查找包含指定的字符串的行

```
1 | [root@zkk ~]# grep osmond *
2 | [root@zkk ~]# fgrep osmond *
```

在当前目录的所有文件中查找包含字符 * 的行

```
1 | [root@zkk ~]# grep '*' *
2 | [root@zkk ~]# fgrep '*' *
```

在文件 myfile 中查找包含字符 \$ 的行
为了强制 shell 将 \$（单反斜杠和美元符号）传递给 grep 命令，必须要使用 \（单反斜杠）。
\（单反斜杠）字符可通知 grep 命令将其后的字符（本例中为 \$）视作原义字符而不是元字符。
如果使用 fgrep 命令，则可以不必使用单反斜杠之类的转义字符。

```
1 | [root@zkk ~]# grep \$ myfile
2 | [root@zkk ~]# grep '\$' myfile
3 | [root@zkk ~]# fgrep '$' myfile
4 | [root@zkk ~]# fgrep $ myfile
```

匹配 myfile 中所有的空行

```
1 | [root@zkk ~]# grep '^$' myfile
```

显示 myfile 中第一个字符为字母的所有行

```
1 | [root@zkk ~]# grep '^[a-zA-Z]' myfile
```

在文件 myfile 中查找首字符不是 # 的行（即过滤掉注释行）

```
1 | [root@zkk ~]# grep -v '^#' myfile
```

显示所有包含每个字符串至少有5个连续小写字符的字符串的行

```
1 | [root@zkk ~]# grep '[a-z]\{5\}' myfile
2 | [root@zkk ~]# egrep '[a-z]{5}' myfile
```

在文件 myfile 中查找包含日期格式（形如：yyyy-mm-dd）的行

```
1 | [root@zkk ~]# grep '[12][0-9]\{3\}-[01][0-9]-[0-3][0-9]' myfile
2 | [root@zkk ~]# egrep '[12][0-9]{3}-[01][0-9]-[0-3][0-9]' myfile
```

在文件 myfile 中查找与abc 或 xyz 字符串匹配的所有行

```
1 | [root@zkk ~]# egrep 'abc|xyz' myfile
```

如果west被匹配，则es就被存储到内存中，并标记为1，然后搜索任意个字符（*），
这些字符后面紧跟着另外一个es（\1），找到就显示该行。

```
1 | [root@zkk ~]# grep 'w(es)t.*\1' myfile
2 | [root@zkk ~]# egrep 'w(es)t.*\1' myfile
```

通过管道过滤ls输出的内容，只显示以 l 开头的行

```
1 | [root@zkk ~]# ls -l | grep '^l'
```

通过管道过滤ls输出的内容，只显示以可写的文件或目录的行

```
1 | [root@zkk ~]# ls -l | grep '[-d].w..w..w.'
```

通过管道过滤ls输出的内容，只显示以 —— 或 - 或 .bak 结尾的行

```
1 | [root@zkk ~]# ls | egrep '(—|-\|.bak)$'
```

find

find命令用于在文件系统中查找满足条件的文件。find 命令功能强大，提供了相当多的查找条件。find 命令还可以对查找到的文件做操作，如执行 Shell 命令等。
find 命令的格式是：

- find [<起始目录> ...] [<选项表达式>] [<条件匹配表达式>] [<动作表达式>]

其中：

- <起始目录>：对每个指定的 <起始目录> 递归搜索目录树
 - 若在整个文件系统范围内查找，则起始目录是“/”

。若在当前目录下寻找，则起始目录是“.”，省略<起始目录>表示当前目录

- <选项表达式>：控制 find 命令的行为
- <条件匹配表达式>：根据匹配条件查找文件
- <动作表达式>：指定对查找结果的操作，默认为显示在标准输出（-print）

不带任何参数的 find 命令将在屏幕上递归显示当前目录下的文件列表。下面给出一些常用的表达式的解释。

选项表达式

表达式	说明
-follow	如果遇到符号链接文件，就跟踪链接所指的文件
-regextype TYPE	指定-regex和-iregex使用的正则表达式类型，默认为emacs，还可选择posix-awk, posix-basic, posix-egrep和posix-extended
-depth	查找进入子目录前先查找完当前目录的文件
-mount	查找文件时不跨越文件系统
-xdev	查找文件时不跨越文件系统
-maxdepth LEVELS	设置最大的查找深度
-help	显示 find 命令
-version	显示 find 的版本

条件匹配表达式

表达式	说明
-name PATTERN	匹配文件名
-iname PATTERN	匹配文件名（忽略大小写）
-lname PATTERN	匹配符号链接文件名
-ilname PATTERN	匹配符号链接文件名（忽略大小写）
-path PATTERN	匹配文件的完整路径（不把 '/' 和 '.' 作为特殊字符）PATTERN 使用 Shell 的匹配模式，可以使用 Shell 的通配符（*、? []），要用""或"括起来
-regex PATTERN	以正则表达式匹配文件名
-iregex PATTERN	以正则表达式匹配文件名（忽略大小写）
-amin N	查找 N 分钟以前被访问过的所有文件
-atime N	查找 N 天以前被访问过的所有文件
-cmin N	查找 N 分钟以前文件状态被修改过的所有文件（比如权限修改）
-ctime N	查找 N 天以前文件状态被修改过的所有文件（比如权限修改）
-mmin N	查找 N 分钟以前文件内容被修改过的所有文件
-mtime N	查找 N 天以前文件内容被修改过的所有文件
-uid N	查找属于 ID 号为 N 用户的所有文件
-gid N	查找属于 ID 号为 N 组的所有文件
-inum N	查找 i-node 是 N 的文件
-links N	查找硬链接数为 N 的文件
-size	N[bcwkMG] 查找大小为 N 的文件，b(块)默认单位; c(字节) ; w(双字节)N可以有三种输入方式，+N 或 -N 或 N。假设 N 为 20，则：(1) +20：表示20以上（21，22，23等）；(2) -20：表示20以内（19，18，17等）；(3) 20：表示正好是20。
-perm MODE	精确匹配权限模式为 MODE 的文件。
-perm -MODE	匹配权限模式至少为 MODE 的文件MODE : 与 chown 命令的书写方式一致，既可以使用字符模式也可以使用8进制模式
-anewer FILE	查找所有比 FILE 的访问时间新的文件
-cnewer FILE	查找所有比 FILE 的状态修改时间新的文件（比如权限修改）
-newer FILE	查找所有比 FILE 的内容修改时间新的文件
-samefile FILE	查找与 FILE 具有相同 i-node 的文件（硬链接）
-fstype TYPE	只查找指定类型的文件系统
-type [bcdpfls]	查找指定类型的文件 [块设备, 字符设备,目录, 管道, 普通文件, 符号链接, socket套接字]
-empty	内容为空的文件
-user NAME	查找用户名为 NAME 的所有文件
-group NAME	查找组名为 NAME 的所有文件
-nouser	文件属于不在 /etc/passwd 文件中的用户
-nogroup	文件属于不在 /etc/group 文件中的组

动作表达式

表达式	说明
-print	在标准输出上列出查找结果（每行一个文件）
-print0	在标准输出上列出查找结果（取消间隔符） 同样与
-fprint FILE	与 -print 一致，只是输出到文件 FILE

表达式	说明
-fprint0 FILE	与 print0 一致，只是输出到文件 FILE
-ls	使用 'ls -dils' 在标准输出上列出查找结果
-fls FILE	与 -ls 一致，只是输出到文件 FILE
-prune	忽略对某个目录的查找
-exec COMMAND {} ;	对符合查找条件的文件执行 Linux 命令
-ok COMMAND {} ;	对符合查找条件的文件执行 Linux 命令；与 -exec 不同的是，它会询问用户是否需要执行

组合条件表达式

在书写表达式时，可以使用逻辑运算符与、或、非组成的复合条件，并可以用()改变默认的操作符优先级。下面以优先级由高到低列出可用的逻辑操作符。若以空格作为各个表达式的间隔符，则各个表示式之间是与关系。

操作符	说明
(EXPR)	改变操作符优先次序，一些 UNIX 版的 find 命令要使用 (EXPR) 形式
! EXPR	表示对表达式取反
EXPR1 EXPR2	与逻辑，若 EXPR1 为假，将不再评估 EXPR2
EXPR1 -a EXPR2	与 EXPR1 EXPR2 功能一致
EXPR1 -o EXPR2	逻辑或，若 EXPR1 为真，将不再评估 EXPR2
EXPR1 , EXPR2	若 EXPR1 为假，继续评估 EXPR2

find命令使用举例

find 的版本和使用帮助信息

```
1 | [root@zkk ~]# find --help    # 显示 find 命令帮助信息
2 | [root@zkk ~]#   find --version # 显示 find 的版本
```

不指定匹配表达式，显示所有文件

递归显示当前目录的文件列表

```
1 | [root@zkk ~]#   find
```

递归显示 / 目录的文件列表

```
1 | [root@zkk ~]#   find /
```

递归显示 / 目录的文件列表（仅限于3层目录）

```
1 | [root@zkk ~]#   find / -maxdepth 3
```

递归显示 / 目录的文件列表（仅限于 / 文件系统）

```
1 | [root@zkk ~]#   find / -xdev
```

递归显示 /home、/www、/srv 目录的文件列表

```
1 | [root@zkk ~]#   find /home /www /srv
```

按文件名/路径名查找

查找特定的文件名

```
1 | [root@zkk ~]#   find -name myfile
2 | [root@zkk ~]#   find -maxdepth 2 -name symfony
```

使用通配符查找特定的文件名

```
1 | [root@zkk ~]# find -name 'd*'
2 | [root@zkk ~]# find -name '???'
3 | [root@zkk ~]# find -name '[afd]*'
4 | [root@zkk ~]# find -iname '[a-z]*'
5 | [root@zkk ~]# find -name 'ch[0-2][0-9].txt*'
```

匹配文件路径名

```
1 | [root@zkk ~]# find -path '*server'
2 | ./vbird/server
3 | ./server
4 | [root@zkk ~]# find -path '*server[12]'
```

```
5 | ./server1
6 | ./server2./server1/server2
7 | ./server2/server2
```

以正则表达式匹配文件路径名

```
1 | [root@zkk ~]# find -regex '.*'
2 | [root@zkk ~]# find -regex '.*ch0.*'
3 | ./ch01
4 | ./ch00
5 | ./vbird/server/1000results/ch09-01.jpg
6 | [root@zkk ~]# find -regex '.*ch[0-9]+'
7 | ./ch01
8 | ./ch21
9 | ./ch00
10 | ./ch333
11 | ./ch1
12 | ./ch41
13 | [root@zkk ~]# find -regex '.*ch[0-9]+\..txt'
```

```
14 | ./ch1.txt
15 | ./ch24.txt
```

按文件属性查找

只查找普通文件

```
1 | [root@zkk ~]# find . -type f
```

只查找符号链接文件

```
1 | [root@zkk ~]# find . -type l
```

查找硬连接数大于 1 的文件或目录

```
1 | [root@zkk ~]# find /home -links +1
```

查找 /tmp 目录下小于 10M 的文件

```
1 | [root@zkk ~]# find /tmp -size -10M
```

查找 /home 目录下大于 1G 的文件

```
1 | [root@zkk ~]# find /home -size +1G
```

查找系统中为空的文件或者目录

```
1 | [root@zkk ~]# find / -empty
```

查找在 /www 中最后10分钟访问过的文件

```
1 | [root@zkk ~]# find /www -amin -10
```

查找在 /www 中最后2天访问过的文件

```
1 | [root@zkk ~]# find /www -atime -2
```

查找在 /home 下最近2天内改动过的文件

```
1 | [root@zkk ~]# find /home -mtime -2
```

列出被改动过后 2 日内被存取过的文件或目录

```
1 | [root@zkk ~]# find /home -used -2
```

列出被改动过后 90 日前被存取过的文件或目录

```
1 | [root@zkk ~]# find /home -used +90
```

列出 /home 目录中属于用户 osmond 的文件或目录

```
1 | [root@zkk ~]# find /home -user osmond
```

列出 /home 目录中 UID 大于 501 的文件或目录

```
1 | [root@zkk ~]# find /home -uid +501
```

列出 /home 目录中组为 osmond 的文件或目录

```
1 | [root@zkk ~]# find /home -group osmond
```

列出 /home 目录中 GID 为 501 的文件或目录

```
1 | [root@zkk ~]# find /home -gid 501
```

列出 /home 目录中不属于本地用户的文件或目录

```
1 | [root@zkk ~]# find /home -nouser
```

列出 /home 目录中不属于本地组的文件或目录

```
1 | [root@zkk ~]# find /home -nogroup
```

精确查找权限为 664 的文件或目录

```
1 | [root@zkk ~]# find . -perm 664
```

查找权限至少为 664 的文件或目录

```
1 | [root@zkk ~]# find . -perm -664
```

使用逻辑运算构造复杂表达式

查找 /tmp 目录下21天之前访问过的大于 10G 的文件

```
1 | [root@zkk ~]# find /tmp -size +10M -a -atime +21
```

查找 / 目录下属主为 jjheng 或 osmond 的文件

```
1 | [root@zkk ~]# find / -user jjheng -o -user osmond
```

查找 /tmp 目录下的属主不是 osmond 的文件

```
1 | [root@zkk ~]# find /tmp ! -user osmond
```

在 /mnt 下查找 *.txt 且文件系统类型不为 vfat 的文件

```
1 | [root@zkk ~]# find /mnt -name '*.txt' ! -fstype vfat
```

在 /tmp 下查找名为 l 开头且类型为符号链接的文件

```
1 | [root@zkk ~]# find /tmp -name 'l*' -type l
```

查找以 server 开头的目录名

```
1 | [root@zkk ~]# find . -type d -name 'server*'
```

找出 /var/log 目录下所有的前5天修改过的.log 文件

```
1 | [root@zkk ~]# find /var/log -name '*.log' -mtime +5
```

按文件样本查找

查找所有比 FILE1 的访问时间新的文件

```
1 | [root@zkk ~]# find -anewer FILE1
```

查找所有比 FILE2 的访问时间旧的文件

```
1 | [root@zkk ~]# find ! -anewer FILE2
```

查找所有比 FILE1 的访问时间新的# 且比 FILE2 的访问时间旧的文件

```
1 | [root@zkk ~]# find -anewer FILE1 ! -anewer FILE2
```

查找所有比 FILE1 的内容修改时间新的文件

```
1 | [root@zkk ~]# find -newer FILE1
```

查找所有比 FILE2 的内容修改时间旧的文件

```
1 | [root@zkk ~]# find ! -newer FILE2
```

查找所有比 FILE1 的内容修改时间新的
且比 FILE2 的内容修改时间旧的文件

```
1 | [root@zkk ~]# find -newer FILE1 ! -newer FILE2
```

查找与 FILE 具有相同 i-node 的文件（硬链接）

```
1 | [root@zkk ~]# find -samefile FILE -ls
```

对查找到的文件实施命令操作

查找并列出当前目录下不安全的文件（世界可读写执行）

```
1 | [root@zkk ~]# find . -perm -007 -exec ls -l {} \;
```

查找 logs 目录下的所有的 .log 文件并查看它的详细信息

```
1 | [root@zkk ~]# find logs -name "*.log" -type f -exec ls -l {} \;
```

查找当天修改过的普通文件

```
1 | [root@zkk ~]# find . -type f -mtime -1 -exec ls -l {} \;
```

查找当前目录下的.php文件并用grep过滤出包含include的行

```
1 | [root@zkk ~]# find . -name "*.php" -exec grep "include" {} \; -print
```

查找并删除当前目录及其子目录下所有扩展名为 .tmp 的文件

```
1 | [root@zkk ~]# find . -name '*.tmp' -exec rm {} \;
```

在logs目录中查找7天之内未修改过的文件并在删除前询问

```
1 | [root@zkk ~]# find logs -type f -mtime +7 -exec -ok rm {} \;
```

查询并删除一周以来从未访问过的以 .o 结尾或名为 a.out
且不存在于 nfs 文件系统中的所有文件

```
1 | [root@zkk ~]# find / ( -name a.out -o -name '*.o' ) -atime +7 \  
2 | ! -fstype nfs -exec rm {} \;
```

查询并删除当前目录及其子目录下所有的空目录

```
1 | [root@zkk ~]# find . -depth -type d -empty -exec rmdir {} \;
```

将default目录下的文件由GBK编码转换为UTF-8编码

目录结构不变，转码后的文件保存在utf/default目录下

```
1 | From: http://www.xiaojb.com/archives/it/convert-gbk-utf-8.shtml  
2 | [root@zkk ~]# find default -type d -exec mkdir -p utf/{ } \;  
3 | [root@zkk ~]# find default -type f -exec iconv -f GBK -t UTF-8 {} -o utf/{ } \;
```

下面 find 命令的书写形式均等价

```
1 | [root@zkk ~]# find -name \*.sh -exec cp {} /tmp \;  
2 | [root@zkk ~]# find -name '*.sh' -exec cp {} /tmp ';' '  
3 | [root@zkk ~]# find -name "*.sh" -exec cp {} /tmp " ";" '  
4 | [root@zkk ~]# find -name \*.sh -exec cp {\ } /tmp \;  
5 | [root@zkk ~]# find -name '*.sh' -exec cp '{ }' /tmp ';' '  
6 | [root@zkk ~]# find -name "*.sh" -exec cp "{}" /tmp " ";" '
```

在查找中排除指定的目录

显示当前目录树

```
1 | [root@zkk ~]# tree -F -L 2  
2 | .  
3 | |-- bin/  
4 | |   |-- switch-lang.sh*  
5 | |   |-- sys2wiki.sh*  
6 | |-- book/  
7 | |   |-- basic/  
8 | |   |-- basic-utf8/  
9 | |   |-- basic.zip  
10 | |-- server/  
11 | |   |-- server-utf8/  
12 | |   |-- server.zip  
13 | |   |-- to-zh-CN-utf8.sh*  
14 | |-- bak.sh*
```

显示当前目录下除 book 目录的所有文件

```
1 | [root@zkk ~]# find . -name book -prune -o -print
```

查找当前目录下（除了 book 目录）的所有 .sh 文件


```
1 | [root@zkk ~]# find . -name book -prune -o -name '*.sh' -print
```

显示当前目录下除 book/server 目录的所有文件

```
1 | [root@zkk ~]# find . -path ./book/server -prune -o -print
```

使用绝对路径完成上述任务

```
1 | [root@zkk ~]# find /home/osmond -path /home/osmond/book/server -prune -o -print
```

查找当前目录下（除了 book/server 目录）的所有 .sh 文件

```
1 | [root@zkk ~]# find . -path ./book/server -prune -o -name '*.sh' -print
```

显示当前目录下除 book/server 和 book/server-utf8 目录的所有文件

```
1 | [root@zkk ~]# find . -path './book/server*' -prune -o -print
```

查找当前目录下（除了 book/server 和 book/server-utf8 目录）的所有 .sh 文件

```
1 | [root@zkk ~]# find . -path './book/server*' -prune -o -name '*.sh' -print
```

显示当前目录下除 book/server 和 book/basic 目录的所有文件

```
1 | [root@zkk ~]# find . \( -path ./book/server -o -path ./book/basic \) -prune -o -print
```

查找当前目录下（除了 book/server 和 book/basic 目录）的所有 .sh 文件

```
1 | [root@zkk ~]# find . \( -path ./book/server -o -path ./book/basic \) -prune -o -name '*.sh' -
```

sed

sed是一个流编辑器（stream editor）。sed 是一个非交互式的行编辑器，它在命令行中输入编辑命令、指定被处理的输入文件，然后在屏幕上查看输出。输入文件可以是指定的文件名，也可以来自一个管道的输出。sed 不改变输入文件的内容，且总是将处理结果输出到标准输出，可以使用输出重定向将 sed 的输出保存到文件中。

与 vi 不同的是 sed 能够过滤来自管道的输入。在 sed 编辑器运行的时候不必人工干涉，所以 sed 常常被称作批编辑器。此特性允许在脚本中使用编辑命令，极大的方便了重复性编辑任务。当对文件中大量的文本进行替换时， sed 将是一个有利的工具。

sed 以按顺序逐行的方式工作，过程为：

- 1.从输入读取一行数据存入临时缓冲区，此缓冲区称为模式空间（pattern space）
- 2.按指定的 sed 编辑命令处理缓冲区中的内容
- 3.把模式空间的内容送往屏幕并将这行内容从模式空间中删除
- 4.读取下面一行。重复上面的过程直到全部处理结束。

sed 命令的格式如下

- 格式1： sed [OPTION] [-e] command1 [[-e command2] ... [-e commandn]] [input-file]...
- 格式2： sed [OPTION] -f script-file [input-file]...

说明：

- 格式1：执行命令行上的sed编辑命令。可以指定多个编辑命令，每个编辑命令前都要使用 -e 参数，sed 将对这些编辑命令依次进行处理。若只有一个编辑命令时，-e 可以省略。
- 格式2：执行脚本文件中的sed编辑命令。当编辑命令很多时，可将所有的编辑命令存成sed脚本文件，然后在命令行上使用 -f 参数指定这个文件。
常用参数：
 - -n : sed 在将下一行读入pattern space之前，自动输出pattern space中的内容。此选项可以关闭自动输出，此时是否输出由编辑命令控制。
 - -r : 使用扩展正则表达式进行模式匹配。
- input-file: sed 编辑的文件列表，若省略，sed 将从标准输入中读取输入，也可以从输入重定向或管道获得输入sed的编辑命令包括地址和操作两部分。地址用于指定sed要操作的行；操作指定要进行的处理。
- 通常使用单引号将整个操作命令括起来
- 若操作命令中包含shell变量替换，应该使用双引号将整个操作命令括起来

地址的表示方法列表如下

分类	表示法	说明
0		省略地址部分，将对输入的每一行进行操作
1	n	表示第 n 行，特殊地：\$ 表示最后一行
1	f~s	表示从 f 开始的，步长为 s 的所有行
1	/regexp/	表示与正则表达式匹配的行
2	m,n	表示从第m行到第n行，特殊地： m,\$表示从m行到最后一行
2	m,+n	表示第 m 行以及其后的 n 行
2	/regexp1/,/regexp2/	表示从匹配 regexp1 的行开始到匹配 regexp2 的行
2	/regexp/,n	表示从匹配 regexp 的行开始到第 n 行
2	n,/regexp/	表示从第n行开始到匹配 regexp 的行

另外，在地址部分还可以使用 ! 表示反向选择，如 m,n! 表示除了m 到n之外的所有行。
sed 支持 25 个操作，下面列出常用的几个，更多的操作的使用方法请参考 sed 手册。

操作	说明
p	打印
l	显示所有字符，包括控制字符(非打印字符)
d	删除
=	显示匹配行的行号
s/regexp/replacement/	将指定行中第一个匹配 regexp 的内容替换为 replacement

操作	说明
s/regexp/replacement/g	将指定行中所有匹配regexp的内容替换为replacement（g表示全局）
s/regexp/replacement/p p	打印修改后的行
s/regexp/replacement/gp p	打印修改后的行（g表示全局）
s/regexp/replacement/w fname	将替换后的行内容写到指定的文件 fname 中
s/regexp/replacement/gw fname	将替换后的行内容写到指定的文件 fname 中（g表示全局）
r fname	将另外一个文件 fname 中的内容附加到指定行
w fname	将当前模式空间（pattern space）的内容写入指定的文件 fname
n	将指定行的下面一行读入模式空间（pattern space）
q	读取到指定行之后退出 sed
a\	在指定行后面追加文本（主要用于 sed 脚本）
i\	在指定行前面追加文本（主要用于 sed 脚本）
c\	用新文本替换指定的行（主要用于 sed 脚本）

sed 使用举例

以 p 操作说明地址的使用方法
显示 myfile 文件的全部内容

```
1 | [root@zkk ~]# sed -n p myfile
```

显示 myfile 文件中第 5 行的内容

```
1 | [root@zkk ~]# sed -n 5p myfile
```

显示 myfile 文件中最后一行的内容

```
1 | [root@zkk ~]# sed -n '$p' myfile
```

显示 myfile 文件从第 3 行开始步长为5的行的内容

```
1 | [root@zkk ~]# sed -n 3~5p myfile
2 | [root@zkk ~]# sed -n 3~5= myfile
```

显示 myfile 文件从第 3 行开始到第 10 行的内容

```
1 | [root@zkk ~]# sed -n 3,10p myfile
```

显示 myfile 文件第 10 行及其后的 10 行内容

```
1 | [root@zkk ~]# sed -n 3,+10p myfile
```

显示 myfile 文件从第 3 行开始到最后一行的内容

```
1 | [root@zkk ~]# sed -n '3,$p' myfile
```

显示 myfile 文件中所有包含 LANG 的行

```
1 | [root@zkk ~]#sed -n /LANG/p myfile
```

显示 myfile 文件中所有不包含 LANG 的行

```
1 | [root@zkk ~]# sed -n '/LANG/!p' myfile
```

显示 myfile 文件从第 3 行开始到其后第一次出现 LANG 的行

```
1 | [root@zkk ~]# sed -n 3,/LANG/p myfile
```

显示 myfile 文件从第一次出现 LANG 的行开始到最后一行的内容

```
1 | [root@zkk ~]# sed -n '/LANG/, $p' myfile
```

显示 myfile 文件从第一次出现以 case 开始的行到第一次出现以 esac 开始的行

```
1 | [root@zkk ~]# sed -n /^case/,/^esac/p myfile
```

以上 sed 命令中p 操作的地址使用也适用于其他操作。

替换命令使用举例
在每个输入行中, 将第一个出现的 Windows 替换为 Linux

```
1 | [root@zkk ~]# sed 's/Windows/Linux/' myfile
```

在每个输入行中, 将第一个出现的 Windows 替换为 Linux ，打印替换结果的行

```
1 | [root@zkk ~]# sed -n 's/Windows/Linux/p' myfile
```

在每个输入行中, 将出现的每个 Windows 替换为 Linux

```
1 | [root@zkk ~]# sed 's/Windows/Linux/g' myfile
```

在每个输入行中, 将出现的每个 Windows 替换为 Linux ，打印替换结果的行

```
1 | [root@zkk ~]# sed -n 's/Windows/Linux/g' myfile
```

在每个输入行中, 将出现的每个 Unix 替换为 Unix/Linux（&表示匹配到的字符串）

```
1 | [root@zkk ~]# sed -e 's/Unix/&\/Linux/g' myfile
```

将所有连续出现的c都压缩成单个的c

```
1 | [root@zkk ~]# sed 's/cc*/c/g' myfile
```

删除行首的一个空格


```
1 | [root@zkk ~]# sed 's/ //' myfile
```

删除每一行前导的连续“空白字符”（空格，制表符）

```
1 | [root@zkk ~]# sed 's/^[ \t]*//' myfile
```

删除以句点结尾的行中末尾的句点

```
1 | [root@zkk ~]# sed 's/\.$//g' myfile
```

删除每行的第一个字符

```
1 | [root@zkk ~]# sed 's/.//' myfile
```

删除每行结尾的所有空格

```
1 | [root@zkk ~]# sed 's/ *$//' myfile
```

在文件的每一行开始处插入两个空格

```
1 | [root@zkk ~]# sed 's/^/ /' myfile
```

在每一行开头加上一个尖括号和空格（引用信息）

```
1 | [root@zkk ~]# sed 's/^> /' myfile
```

将每一行开头处的尖括号和空格删除（解除引用）

```
1 | [root@zkk ~]# sed 's/^> //' myfile
```

删除路径前缀

```
1 | [root@zkk ~]# sed 's/.*/\/' myfile
2 | [root@zkk ~]# ls -d /usr/share/man/man1 |sed 's/.*/\/'
```

过滤掉所有标点符号（.、,、?、!）

```
1 | [root@zkk ~]# sed 's/\./g' -e 's/,/g' -e 's/\?/g' -e 's/!/g' myfile
```

对于 GNU sed 可以使用如下的等效形式

```
1 | [root@zkk ~]# sed 's/\./g ; s/,/g ; s/\?/g ; s/!/g' myfile
```

不论什么字符，紧跟着s命令的都被认为是分隔符，所以，“#”在这里是分隔符，代替了默认的“/”分隔符。尤其适用于替换文件路径

```
1 | [root@zkk ~]# sed 's#/some/path/old#/some/path/new#g' myfile
```

多个sed编辑命令是顺序执行的，例如下面的命令

```
1 | [root@zkk ~]# sed -e 's/Unix/UNIX/g' -e 's/UNIX System/UNIX Operating System/g' myfile
```

首先将 Unix 替换为 UNIX，然后将 UNIX System 替换为 UNIX Operating System下面的命令将不会得到预想的结果

```
1 | [root@zkk ~]# sed -e 's/Unix/UNIX/g' -e 's/Unix System/UNIX Operating System/g' myfile
```

因为Unix在缓冲区中已经被替换成了UNIX，所以再也找不到 Unix System 了。之所以没有使用下面的命令

```
1 | [root@zkk ~]# sed -e 's/Unix System/UNIX Operating System/g' myfile
```

而使用了两个替换命令，是为了将 UNIX System 也替换为 UNIX Operating System 。在支持扩展正则表达式的 sed 中也可以使用如下的命令

```
1 | [root@zkk ~]# sed -r 's/(Unix|UNIX) System/UNIX Operating System/g' myfile
```

替换的速度优化：可以考虑在替换命令（“s/.../...”）前面加上地址表达式来提高速度。举例来说：

```
1 | sed 's/foo/bar/g' filename # 标准替换命令
2 | sed '/foo/ s/foo/bar/g' filename # 速度更快
3 | sed '/foo/ s//bar/g' filename # 简写形式
```

若只替换第一次匹配 foo 的行，可以使用 q 短路后续行的执行。举例来说：

```
1 | sed '/foo/{s/foo/bar;q}' filename
```

其他命令使用举例
删除所有空白行

```
1 | [root@zkk ~]# sed '/^$/d' myfile
2 | [root@zkk ~]# sed '/./!d' myfile
```

删除文件顶部的所有空行

```
1 | [root@zkk ~]# sed '/./,$!d' myfile
```

从输入的开头一直删除到第1个空行(第一个空行也删除掉)

```
1 | [root@zkk ~]# sed '1,/^$/d' myfile
```

删除所有偶数行，与 sed -n ‘1~2p’ myfile 等效

```
1 | [root@zkk ~]# sed 'n;d' myfile
```

删除掉所有包含"GUI"的行

```
1 | [root@zkk ~]# sed '/GUI/d' myfile
```

将所有"GUI"都删除掉, 并保持剩余部分的完整性

```
1 | [root@zkk ~]# sed 's/GUI//g' myfile
```

在每一行后面增加一空行

```
1 | [root@zkk ~]# sed G myfile
```

在匹配“regex”的行之后插入一空行

```
1 | [root@zkk ~]# sed '/regex/G' myfile
```

将 myfile 中从case开始的行到esac结束的行写到文件 case-block

```
1 | [root@zkk ~]# sed '/^case/,/^esac/w case-block' myfile
```

在 myfile 末尾（\$）追加新行

反斜线 \ 是必需的，它表示将插入一个回车符。在任何要输入回车的地方您必须使用反斜线。

```
1 | [root@zkk ~]# sed '$a\ > newline1\ > newline2\
2 | > newline3' myfile
3 | 在匹配“regex”的行之后追加新行
4 | [root@zkk ~]# sed '/regex/a\ > newline1\
5 | > newline2\
6 | > newline3' myfile
7 | i\ 和 c\ 操作的格式与 上面的 a\ 操作的格式相同
```

awk

awk是一种用于处理文本的编程语言工具。它使用类似于C的语法，并在很多方面类似于 shell 编程语言。awk 名称是由它三个最初设计者的姓氏的第一个字母而命名的：Alfred Aho、Peter Weinberger 和 Brian Kernighan。gawk 是 GNU 版本 awk，gawk 最初在1986年完成，之后不断地被改进、更新。Linux 下的 awk 是 gawk 的符号链接。

与sed和grep很相似，awk 是一种模式扫描和处理语言。但其功能却大大强于sed和grep。awk尤其适合处理结构化的文本，如纯文本的表格等。awk提供了极其强大的功能：它几乎可以完成grep和sed所能完成的全部工作。同时，awk还支持流程控制、数学运算、进程控制语句甚至于内置的变量和函数。它具备了一个完整的语言所应具有的所有精美特性。

与 sed 一样，awk 不会修改输入文件的内容，可以使用输出重定向将 awk 的输出保存到文件中。

awk 命令的格式如下：

- 格式1： awk [OPTION] ‘program-statements’ [input-file]...
- 格式2： awk [OPTION] -f program-file [input-file]...

说明：

- 格式1：执行命令行上的awk程序语句。若在一行上书写多个awk程序语句时，各个语句使用分号 (;) 间隔。
 - 格式2：执行脚本文件中的awk程序语句。当awk程序语句很多时，可将所有的awk程序语句存成脚本文件，然后在命令行上使用 -f 参数指定这个文件。
- 常用参数：
- -F fs : 在awk中，缺省的字段分隔符一般是空格符或TAB。在- F后面跟着你想用的分隔符即可改变字符分隔符。
 - -v var=val : 对变量 var 赋初值为 val，变量既可以是 awk 的内置变量也可以是自定义变量。
 - input-file: awk 处理的文件列表，若省略，awk 将从标准输入中读取输入，也可以从输入重定向或管道获得输入。

awk 中每一个语句（statements）都由两部分组成：模式（pattern）和相应的动作（actions）。只要模式匹配，awk 就会执行相应的动作。动作部分由一个或多个命令、函数、表达式组成，之间由换行符或分号隔开，并位于大括号内。

- pattern 和 {actions} 可以省略，但不能同时省略；
- pattern 省略时表示对所有的输入行执行指定的 {actions}；
- {actions} 省略时表示打印匹配行，即 { print }。

模式（pattern）部分可以是：

- /regular expression/ : 使用扩展的正则表达式。
- relational expression : 使用关系表达式，可以使用与 C 语言类似的关系运算符。
- pattern1, pattern2 : 范围模式，匹配行的范围。表示从匹配pattern1的行到匹配pattern2的行。
- BEGIN : 指定在第一条输入记录被处理之前要执行的动作，通常可在此设置全局变量。
- END : 指定在最后一输入记录被读取之后要执行的动作，通常可在此输出统计数据。

动作（actions）部分可以是：

- 变量或数组赋值
- 输入/输出语句
- 内置函数和自定义函数
- 流程控制语句

awk 命令的一般形式为：

```
1 | awk 'BEGIN {actions}
2 | pattern1 {actions}
3 | .....
4 | patternN {actions}
5 | END {actions}' input-file
```

其中 BEGIN {actions} 和 END {actions} 是可选的。awk 的执行过程如下：

- 1.如果存在 BEGIN ，awk 首先执行它指定的 actions。
- 2.awk 从输入中读取一行，称为一条输入记录。
- 3.awk 将读入的记录分割成数个字段，并将第一个字段放入变量 \$1 中，第二个放入变量 \$2 中，以此类推；\$0 表示整条记录；字段分隔符可以通过选项 -F 指定，否则使用缺省的分隔符。
- 4.把当前输入记录依次与每一个语句中 pattern 比较：如果相匹配，就执行对应的 actions；如果不匹配，就跳过对应的 actions，直到完成所有的语句。
- 5.当一条输入记录处理完毕后，awk 读取输入的下一行，重复上面的处理过程，直到所有输入全部处理完毕。
- 6.如果输入是文件列表，awk 将按顺序处理列表中的每个文件。
- 7.awk 处理完所有的输入后，若存在 END，执行相应的 actions。

awk 常用的内置变量

变量	说明
NF	当前记录中的字段数。
NR	当前记录数。
FS	字段分隔符(默认是任何空格)。
RS	记录分隔符(默认是一个换行符)。
OFS	输出字段分隔符(默认值是一个空格)。
ORS	输出记录分隔符(默认值是一个换行符)。
IGNORECASE	如果为真，则进行忽略大小写的匹配。

awk 使用举例

下面给出一些使用 awk 的简单例子。

使用awk打印字符串

```
1 [root@zkk ~]# awk 'BEGIN { print "hello" }'
```

```
2 hello
```

使用awk进行浮点运算

```
1 [root@zkk ~]# awk 'BEGIN { print 1.05e+2/10.5+2.0**3-3.14 }'
```

```
2 14.86
```

显示要处理的输入文件

```
1 [root@zkk ~]# cat test.txt
```

```
2 F115!16201!1174113017250745 10.86.96.41 211.140.16.1 200703180718
```

```
3 F125!16202!1174113327151715 10.86.96.42 211.140.16.2 200703180728
```

```
4 F235!16203!1174113737250745 10.86.96.43 211.140.16.3 200703180738
```

```
5 F245!16204!1174113847250745 10.86.96.44 211.140.16.4 200703180748
```

```
6 F355!16205!1174115827252725 10.86.96.45 211.140.16.5 200703180758
```

显示输入文件的内容

```
1 [root@zkk ~]# awk '{print}' test.txt
```

```
2 F115!16201!1174113017250745 10.86.96.41 211.140.16.1 200703180718
```

```
3 F125!16202!1174113327151715 10.86.96.42 211.140.16.2 200703180728
```

```
4 F235!16203!1174113737250745 10.86.96.43 211.140.16.3 200703180738
```

```
5 F245!16204!1174113847250745 10.86.96.44 211.140.16.4 200703180748
```

```
6 F355!16205!1174115827252725 10.86.96.45 211.140.16.5 200703180758
```

使用正则表达式匹配行，{actions} 省略时表示 { print }

```
1 [root@zkk ~]# awk '/F[12].*/' test.txt
```

```
2 F115!16201!1174113017250745 10.86.96.41 211.140.16.1 200703180718
```

```
3 F125!16202!1174113327151715 10.86.96.42 211.140.16.2 200703180728
```

```
4 F235!16203!1174113737250745 10.86.96.43 211.140.16.3 200703180738
```

```
5 F245!16204!1174113847250745 10.86.96.44 211.140.16.4 200703180748
```

使用正则表达式匹配行，并打印匹配的第1和第3列（域、字段）

```
1 [root@zkk ~]# awk '/^F[12].*/ {print $1,$3}' test.txt
```

```
2 F115!16201!1174113017250745 211.140.16.1
```

```
3 F125!16202!1174113327151715 211.140.16.2
```

```
4 F235!16203!1174113737250745 211.140.16.3
```

```
5 F245!16204!1174113847250745 211.140.16.4
```

更改字段分隔符为!，执行上面的操作

```
1 [root@zkk ~]# awk -F\! '/^F[12].*/ {print $1,$3}' test.txt
```

```
2 F115 1174113017250745 10.86.96.41 211.140.16.1 200703180718
```

```
3 F125 1174113327151715 10.86.96.42 211.140.16.2 200703180728
```

```
4 F235 1174113737250745 10.86.96.43 211.140.16.3 200703180738
```

```
5 F245 1174113847250745 10.86.96.44 211.140.16.4 200703180748
```

使用空格或!做为字段分隔符（正则表达式 [!]）

```
1 [root@zkk ~]# awk -F '[ !]' '{print $1,$2,$3,$4,$5,$6}' test.txt
```

```
2 F115 16201 1174113017250745 10.86.96.41 211.140.16.1 200703180718
```

```
3 F125 16202 1174113327151715 10.86.96.42 211.140.16.2 200703180728
```

```
4 F235 16203 1174113737250745 10.86.96.43 211.140.16.3 200703180738
```

```
5 F245 16204 1174113847250745 10.86.96.44 211.140.16.4 200703180748
```

```
6 F355 16205 1174115827252725 10.86.96.45 211.140.16.5 200703180758
```

使用 awk 内置的取子串函数提取输入文件中的手机号

```
1 [root@zkk ~]# awk -F '[ !]' '{print substr($3,6)}' test.txt
```

```
2 13017250745
```

```
3 13327151715
```

```
4 13737250745
```

```
5 13847250745
```

```
6 15827252725
```

使用关系表达式书写模式，打印所有奇数行

```
1 [root@zkk ~]# awk 'NR % 2 == 1' test.txt
```

```
2 F115!16201!1174113017250745 10.86.96.41 211.140.16.1 200703180718
```

```
3 F235!16203!1174113737250745 10.86.96.43 211.140.16.3 200703180738
```

```
4 F355!16205!1174115827252725 10.86.96.45 211.140.16.5 200703180758
```

使用关系表达式书写模式，打印所有奇数行的第1和第3列（域、字段）

```
1 [root@zkk ~]# awk 'NR % 2 == 1 {print $1,$3}' test.txt
```

```
2 F115!16201!1174113017250745 211.140.16.1
```

```
3 F235!16203!1174113737250745 211.140.16.3
```

```
4 F355!16205!1174115827252725 211.140.16.5
```

打印输入文件的行数，类似于 wc -l test.txt

```
1 [root@zkk ~]# awk 'END { print NR }' test.txt5
```

为每一笔记录前添加行号，类似于 cat -n test.txt


```
1 [root@zkk ~]# awk '{print NR,$0}' test.txt
2 1 F115!16201!1174113017250745 10.86.96.41 211.140.16.1 200703180718
3 2 F125!16202!1174113327151715 10.86.96.42 211.140.16.2 200703180728
4 3 F235!16203!1174113737250745 10.86.96.43 211.140.16.3 200703180738
5 4 F245!16204!1174113847250745 10.86.96.44 211.140.16.4 200703180748
6 5 F355!16205!1174115827252725 10.86.96.45 211.140.16.5 200703180758
```

为每一笔记录前添加行号，使用制表符作为行号和记录的间隔符

```
1 [root@zkk ~]# awk '{print NR "\t" $0}' test.txt
2 1 F115!16201!1174113017250745 10.86.96.41 211.140.16.1 200703180718
3 2 F125!16202!1174113327151715 10.86.96.42 211.140.16.2 200703180728
4 3 F235!16203!1174113737250745 10.86.96.43 211.140.16.3 200703180738
5 4 F245!16204!1174113847250745 10.86.96.44 211.140.16.4 200703180748
6 5 F355!16205!1174115827252725 10.86.96.45 211.140.16.5 200703180758
```

下面再给出一些 awk 和其他命令结合使用的例子：

提取文件 test.txt 中的手机号

```
1 [root@zkk ~]# cat test.txt | awk -F\! '{print $3}' | awk '{print $1}'|cut -c6-16
2 13017250745
3 13327151715
4 13737250745
5 13847250745
6 15827252725
```

以文件修改顺序生成当前目录下带有时间的文件名

```
1 [root@zkk ~]# ls -alt * --time-style='+%F_%H:%M' | awk '{print $7"--"$6}'
2 file5--2007-12-27_12:00
3 file4--2007-12-26_12:00
4 file3--2007-12-25_12:00
5 file2--2007-12-24_12:00
6 file1--2007-12-23_12:00
```

计算当前目录中所有12月份创建的文件的字节数

```
1 [root@zkk ~]# ls -l | awk '$6 == "Dec" { sum += $5 } ; END { print sum }'
2 79878
3 [root@zkk ~]# who
4 root      tty1          2007-12-14   20:33
5 osmond    pts/0         2007-12-14   16:26 (192.168.0.77)
```

显示当前所有的登录用户和其使用的终端

```
1 [root@zkk ~]# who | awk '{print $1"\t"$2}'
2 root      tty1
3 osmond    pts/0
4 [root@zkk ~]# df -hPT -x tmpfs
5 Filesystem Type      Size    Used Avail Use% Mounted on
6 /dev/mapper/VolGroup00-LogVolRoot ext3    3.9G 1.1G 2.7G 28% /
7 /dev/mapper/VolGroup00-LogVolHome ext3    2.9G 106M 2.6G 4% /home
8 /dev/sda1  ext3     99M 12M 83M 13% /boot
```

使用 awk 筛选字段并格式化输出

```
1 [root@zkk ~]# df -hPT -x tmpfs| awk '{print "| " $1 " | " $2 " | " $3 " | " $7 " |"}'
2 | Filesystem | Type | Size | Mounted |
3 | /dev/mapper/VolGroup00-LogVolRoot | ext3 | 3.9G | / |
4 | /dev/mapper/VolGroup00-LogVolHome | ext3 | 2.9G | /home |
5 | /dev/sda1 | ext3 | 99M | /boot |
6 [root@zkk ~]# cat /proc/meminfo | grep MemTotal
7 MemTotal:    515476 kB
8 [root@zkk ~]# cat /proc/meminfo | grep MemTotal | awk -F\: '{print $2}'
9 515476 kB
10 [root@zkk ~]# cat /proc/meminfo | grep MemTotal | awk -F\: '{print $2}' | awk '{print $1 " "
11 515476 kB
12 [root@zkk ~]# cat /proc/cpuinfo | grep 'model name'
13 model name : Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz
14 model name : Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz
15 [root@zkk ~]# cat /proc/cpuinfo | grep 'model name' | awk -F\: '{print $2}'
16 Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz
17 Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz
18 [root@zkk ~]# cat /proc/cpuinfo | grep 'model name' | awk -F\: '{print $2}'|uniq|sed -e 's/ /
19 Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz
```

显示 ifconfig -a 的输出中以单词开头的行

```
1 [root@zkk ~]# ifconfig -a |grep '^w'
2 eth0      Link encap:Ethernet HWaddr 00:0C:29:B3:75:80
3 lo        Link encap:Local Loopback
4 sit0      Link encap:IPv6-in-IPv4
```

显示除了 lo 之外的所有网络接口

```
1 [root@zkk ~]# ifconfig -a |grep '^w'|awk '!/lo/{print $1}'
2 eth0
3 sit0
```

[root@zkk ~]# ifconfig eth0

```
1 eth0      Link encap:Ethernet HWaddr 00:0C:29:B3:75:80
2 inet addr:192.168.0.101 Bcast:192.168.0.255 Mask:255.255.255.0
3 inet6 addr: fe80::20c:29ff:feb3:7580/64 Scope:Link
4 UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
5 RX packets:44783 errors:0 dropped:0 overruns:0 frame:0
6 TX packets:53687 errors:0 dropped:0 overruns:0 carrier:0
7 collisions:0 txqueuelen:1000
8 RX bytes:3876638 (3.6 MiB) TX bytes:16922382 (16.1 MiB)
9 Interrupt:169 Base address:0x2000
```

匹配 inet 的行，以分号为字段间隔符打印第2个字段

```
1 [root@zkk ~]# ifconfig eth0 |awk -F\: '{print $2}'
2 192.168.0.101 Bcast
3 [root@zkk ~]# ifconfig eth0 |awk -F\: '{print $2}'|awk '{print $1}'
4 192.168.0.101
5 [root@zkk ~]# ifconfig eth0 | grep 'inet '
6 inet addr:192.168.0.101 Bcast:192.168.0.255 Mask:255.255.255.0
```

使用一个或多个空格 或 : 做为字段分隔符 （正则表达式 '+'|:|'）

```
1 | [root@zkk ~]# ifconfig eth0 | grep 'inet '| awk -F ' '+'|:' '{print $4}'
2 | 192.168.0.101
```

使用一个或多个空格 或 一个或多个: 做为字段分隔符 （正则表达式 '[':|+')

```
1 | [root@zkk ~]# ifconfig eth0 | grep 'inet '| awk -F '[':|+' '{print $4}'
2 | 192.168.0.101
```

删除所有名为 foo 的进程

```
1 | [root@zkk ~]# kill `ps ax|grep 'foo'|grep -v 'grep'|awk '{print $1}'`
```

查看Apache的并发请求数及其TCP连接状态

```
1 | FROM : http://blog.s135.com/read.php/269.htm
2 | [root@zkk ~]# netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a}]'
```

🔗 文章知识点与官方知识档案匹配，可进一步学习相关知识

Linux技能树 > Linux实用命令 > awk命令 8478 人正在系统学习中

文本 查找 和筛选工具(grep,find,sed)	12-21
文本 查找 和筛选工具(grep,find,sed)，介绍这三个命令的参数用法和大量实例	
中国知名网站(alexa top 500)	chensong7962的博文 1638
网站 类型 Baidu.com 搜索 QQ.COM 门户 Taobao.com 电子商务 sina.com....	
Linux 初级入门百篇- find 命令 _find [0-9]+_木泉说 Linux 的博客-CSDN博...	8-3
find [<起始 目录 > ...] [<选项表达式>] [<条件匹配表达式>] [<动作表达式>] 其中: n <起始 目录 >:对每个指定的 <起始 目录 > 递归搜索 目录 树 ...	
Linux 基础知识及相关命令_IT--Fly的博客	8-7
查找/tmp 目录 下 21 天之前访问过的大于 10G 的文件 \$ find / tmp -size +10M -a -atime + 21 查找 /home 目录 下属主为 jheng 或 osmond 的...	
redis-live 监控安装与测试 热门推荐	无人境域 25万+
1.下载安装依赖的软件：setuptools和pip 这需要使用两个脚本 文件 ez_setup.py和get-pip.py 这是从python官方提供的脚本 ez_setup.py #!...	
101 个 shell 脚本	weixin_34337381的博文 1万+
本文用于记录学习和日常中使用过的 shell 脚本 【脚本1】打印形状 打印等腰三角形、直角三角形、倒直角三角形、菱形 #!/bin/bash # 等...	
Linux 如何 查找 大 文件 内容_ linux 查看 10 多个 g 的 文件 _齐天-大圣的博客-C...	8-10
方法一:使用du命令 du -sh /* 先看看根 目录 下面 1.3G /usr 15G /var 然后再使用 du -sh /var/* 进行 查找 方法二:使用 find/-type f-size+ 10G ...	
Linux 查找 大 文件 的方法_ linux 查找 大 文件 _熊猫Jay的博客	8-5
比如,我要列出 /data/log/ 目录 中的 20 个最大 文件 ,可以: ls-lSh /data/log/ head-20 1 第二种:find find 本身就是 查找 命令,可以递归 查找 一个...	
Linux 高级命令 find,grep,sed,awk	weixin_44783506的博文 2208
Linux 高级命令[find,grep,sed,wak] 1. find find 命令用来在指定 目录 下 查找文件 语法： find path -option [-print] [-exec -ok command] {} ; ...	
vi_vim_awk_sed_grep_find 超级达人学习包	07-05
vi_vim_awk_sed_grep_find 超级达人学习包,包括一些vi的命令，图表等	
Linux--查找大文件 的几种方法_ linux 查找大文件 _Petter's Blog的博...	8-11
样例2: 查找 /etc 目录 下大于 200M 的 文件 find /etc -type f -size +200M xargs ls -lsh 样例3: 查找 /etc 目录 下 10 天前最大的 5 个 文件 find /etc -...	
win 10 计算机 查找大文件 ,教你如何在Win10系统中 查找大文件 ?_阿卞是宝...	8-10
如果选择Gigantic Filter(> 128 MB), 文件 资源管理器会自动搜索大于128MB的所有 文件 。 现在,如果要搜索大于500 MB的所有 文件 。 只需...	
shell 编程指南, shell 脚本,本书共分五部分全。 grep 家族, sed , AWK ,正则表达式,tr用法	10-08
第10章 sed 用法介绍 89 第11章 合并与分割 104 第12章 tr用法 119 第三部分 登录环境 第13章 登录环境 125 第14章 环境和shell变量 13...	
shell 编程和 unix 命令	01-15
001_文件安全与权限 002_使用find和xargs 003_后台执行命令nohup&at;&crontab; 004_文件名置换 008_ grep 家族 009_ AWK 介绍 01...	
Linux 常用命令_查看 tmp 目录 下的所有 文件 _西邮彭于晏的博客	7-27
(4) 拷贝 目录 : (5) 查看 目录 : 命令: find 目录 参数 文件 名称 find / tmp -name 'a'* 查找/tmp 目录 下所有以a开头的 文件 或 目录 . (6) 删除 目录 : (7)...	
linux 查看一个 g 以上的 文件夹 , Linux 系统中 查找大文件 的方法有哪些_weixi...	8-7
find 本身就是 查找 命令,可以递归 查找 一个 目录 的子 目录 ,所以用它是自然的。 比如, 查找 / 目录 下最大的一个 文件 : 1. sudo find / -type f -pr...	
linux unix 命令	12-21
这是我下载过的用于学习 linux /unix 最好的电子书。	
shell 知识-我认为很很不错的	08-31
内含每个知识点的详细描述，新手快速掌握，老手查询快速。	
... 查找文件 命令总结,这个很哇塞_查看 tmp 目录 下的指定 文件 命令_角落...	8-10
查找 /root/ 目录 下,以[1-3之间],结尾是.txt的 文件 [root@localhost ~]# ls 1.txt2.txt3.txt CatalogFile [root@localhost ~]# find/root/-name"[1-3]...	
Linux 运维常用命令 find 、 awk 、 sed 、 grep 、 vi 、 ps 、 ls of、 rpm	wohu1104的专栏 504
先用一个脚本，模拟创建 14 个测试 文件 ： #!/bin/bash for ((i=1;i<=10;i++)); do if [\$i -lt 3] then touch /home/mysql/test/test\$i.sh touch /...	
sed 替换_ find 结合 sed 查找 替换等命令总结	weixin_39953481的博文 817
致读者：点击上方“程序员爱好社区”→ 点击右上角“...”→ 点选“设为星标★” 加上星标，就不会找不到我啦！ sed 文本流编辑[root@admi...	
四剑客笔记1	weixin_44894262的博文 143
find 主要用于 查找文件 名，要以 :结束 固定格式 find .-name ".txt" -type d -mtime -1 xargs cp{ }/ tmp / ; 查找一天 以内以.txt结尾的 文件 并...	
AbstractQueuedSynchronizer浅析	吴大侠的博文 9万+
文章 目录 简介如何实现自定义同步器实现分析1. public final void acquire(int arg) 该方法以排他的方式获取锁，对中断不敏感，完成synch...	
find 、 sed 命令用法示例	东方隐 820
1、 查找 /etc 目录 下大于 1M 且类型为普通 文件 的所有 文件 。 # -type 过滤 文件 类型，f 表示普通 文件 # -size 过滤 文件 大小，+1M 表示大...	
linux 下查看 book 文件 夹, Linux 初级入门百篇- find 命令	weixin_39925959的博文 153
find内容提要1.熟悉find命令的格式2.掌握find命令中各种表达式的书写方法3.学会使用find命令 查找文件 find命令的格式find命令用于在文...	
四个强大的 linux 文本处理工具（ find 、 grep 、 sed 、 awk ）	青萍之末的博文 1929
一、find 因为 Linux 下面一切皆 文件 ，经常需要搜索某些 文件 来编写，所以对于 linux 来说find是一条很重要的命令。 linux 下面的find指...	
linux 查看指定 目录 下最大的 10 个 文件	奋斗、 9712
linux 查看指定 目录 下最大的 10 个 文件 ， linux 文件 按大小排序	
linux 中在当前 目录 下找出占用空间最大的前 10 大 文件	dexter159的博文 2万+
首先要了解三个常用命令： du : 计算出单个 文件 或者 文件 夹的磁盘空间占用.sort : 对 文件 行或者标准输出行记录排序后输出.head : 输出...	
find;grep;awk;sed 命令 最新发布	04-06
find、 grep 、 awk 、 sed 都是在 Linux 操作系统中常用的命令。 find用于在 文件 系统中搜索 文件 和 目录 。它可以按照 文件 名、属性等多种方...	

“相关推荐”对你有帮助么？

- 非常没帮助
- 没帮助
- 一般
- 有帮助
- 非常有帮助

关于我们 招贤纳士 商务合作 寻求报道 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00

公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕 1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心

