搞定Linux Shell文本处理工具，看完这篇集锦就够了

2017-09-26 大CC 马哥Linux运维

Linux Shell是一种基本功，由于怪异的语法加之较差的可读性，通常被Python等脚本代替。既然是基本功，那就需要掌握，毕竟学习Shell脚本的过程中，还是能了解到很多Linux系统的内容。

Linux脚本大师不是人人都可以达到的，但是用一些简单的Shell实现一些常见的基本功能还是很有必要的。

下面我介绍Linux下使用Shell处理文本时最常用的工具：
find、grep、xargs、sort、uniq、tr、cut、paste、wc、sed、awk；
提供的例子和参数都是最常用和最为实用的；
我对shell脚本使用的原则是命令单行书写，尽量不要超过2行；
如果有更为复杂的任务需求，还是考虑python吧；

1、find 文件查找

查找txt和pdf文件

```
find . \( -name "*.txt" -o -name "*.pdf" \) -print
```

正则方式查找.txt和pdf

```
find . -regex  ".*\(\.txt|\.pdf\)$"
```

-iregex： 忽略大小写的正则

否定参数

查找所有非txt文本

```
find . ! -name "*.txt" -print
```

指定搜索深度

打印出当前目录的文件（深度为1）

```
find . -maxdepth 1 -type f
```

定制搜索

按类型搜索：

```
find . -type d -print   //只列出所有目录
```

-type f 文件 / l 符号链接

按时间搜索：

-atime 访问时间 (单位是天，分钟单位则是-amin，以下类似）

-mtime 修改时间（内容被修改）

-ctime 变化时间（元数据或权限变化）

最近7天被访问过的所有文件：

```
find . -atime 7 -type f -print
```

按大小搜索：

w字 k M G

寻找大于2k的文件

```
find . -type f -size +2k
```

按权限查找：

```
find . -type f -perm 644 -print //找具有可执行权限的所有文件
```

按用户查找：

```
find . -type f -user weber -print// 找用户weber所拥有的文件
```

找到后的后续动作

删除：

删除当前目录下所有的swp文件：

```
find . -type f -name "*.swp" -delete
```

执行动作（强大的exec）

```
find . -type f -user root -exec chown weber {} \; //将当前目录下的所有权变更为weber
```

注：{}是一个特殊的字符串，对于每一个匹配的文件，{}会被替换成相应的文件名；

eg：将找到的文件全都copy到另一个目录：

```
find . -type f -mtime +10 -name "*.txt" -exec cp {} OLD \;
```

结合多个命令

tips: 如果需要后续执行多个命令，可以将多个命令写成一个脚本。然后 -exec 调用时执行脚本即可；

```
-exec ./commands.sh {} \;
```

-print的定界符

默认使用'\n'作为文件的定界符；

-print0 使用'\0'作为文件的定界符，这样就可以搜索包含空格的文件；

2、grep 文本搜索

grep match_patten file // 默认访问匹配行

常用参数

-o 只输出匹配的文本行 **VS** -v 只输出没有匹配的文本行

-c 统计文件中包含文本的次数

```
grep -c "text" filename
```

-n 打印匹配的行号

-i 搜索时忽略大小写

-l 只打印文件名

在多级目录中对文本递归搜索(程序员搜代码的最爱）：

```
grep "class" . -R -n
```

匹配多个模式

```
grep -e "class" -e "vitural" file
```

grep输出以\0作为结尾符的文件名：（ -z ）

```
grep "test" file* -lZ| xargs -0 rm
```

3、xargs 命令行参数转换

xargs 能够将输入数据转化为特定命令的命令行参数；这样，可以配合很多命令来组合使用。比如grep，比如find；

将多行输出转化为单行输出

cat file.txt| xargs

\n 是多行文本间的定界符

将单行转化为多行输出

cat single.txt | xargs -n 3

-n：指定每行显示的字段数

xargs参数说明

-d 定义定界符 （默认为空格 多行的定界符为 \n ）

-n 指定输出为多行

-I {} 指定替换字符串，这个字符串在xargs扩展时会被替换掉,用于待执行的命令需要多个参数时

eg：

```
cat file.txt | xargs -I {} ./command.sh -p {} -l
```

-0：指定\0为输入定界符

eg：统计程序行数

```
find source_dir/ -type f -name "*.cpp" -print0 |xargs -0 wc -l
```

4、sort 排序

字段说明：

-n 按数字进行排序 VS -d 按字典序进行排序

-r 逆序排序

-k N 指定按第N列排序

eg：

```
sort -nrk 1 data.txt
sort -bd data // 忽略像空格之类的前导空白字符
```

5、uniq 消除重复行

消除重复行

```
    sort unsort.txt | uniq
```

统计各行在文件中出现的次数

```
    sort unsort.txt | uniq -c
```

找出重复行

```
    sort unsort.txt | uniq -d
```

可指定每行中需要比较的重复内容：-s 开始位置 -w 比较字符数

6、用tr进行转换

通用用法

```
    echo 12345 | tr '0-9' '9876543210' //加解密转换，替换对应字符
    cat text| tr '\t' ' '   //制表符转空格
```

tr删除字符

```
    cat file | tr -d '0-9' // 删除所有数字
```

-c 求补集

```
    cat file | tr -c '0-9' //获取文件中所有数字
    cat file | tr -d -c '0-9 \n'   //删除非数字数据
```

tr压缩字符

tr -s 压缩文本中出现的重复字符；最常用于压缩多余的空格

```
cat file | tr -s ' '
```

字符类

tr中可用各种字符类：

alnum：字母和数字

alpha：字母

digit：数字

space：空白字符

lower：小写

upper：大写

cntrl：控制（非可打印）字符

print：可打印字符

使用方法：tr [:class:] [:class:]

```
eg: tr '[:lower:]' '[:upper:]'
```

7、cut 按列切分文本

截取文件的第2列和第4列：

```
cut -f2,4 filename
```

去文件除第3列的所有列：

```
cut -f3 --complement filename
```

-d 指定定界符：

```
cat -f2 -d";" filename
```

cut 取的范围

N- 第N个字段到结尾

-M 第1个字段为M

N-M N到M个字段

cut 取的单位

-b 以字节为单位

-c 以字符为单位

-f 以字段为单位（使用定界符）

eg:

```
cut -c1-5 file //打印第一到5个字符
cut -c-2 file  //打印前2个字符
```

8、paste 按列拼接文本

将两个文本按列拼接到一起;

```
cat file112cat file2
colin
book

paste file1 file21 colin2 book
```

默认的定界符是制表符，可以用-d指明定界符

paste file1 file2 -d ","

1,colin

2,book

9、wc 统计行和字符的工具

wc -l file // 统计行数

wc -w file // 统计单词数

wc -c file // 统计字符数

10、sed 文本替换利器

首处替换

```
seg 's/text/replace_text/' file    //替换每一行的第一处匹配的text
```

全局替换

```
 seg 's/text/replace_text/g' file
```

默认替换后，输出替换后的内容，如果需要直接替换原文件,使用-i：

```
seg -i 's/text/repalce_text/g' file
```

移除空白行：

```
sed '/^$/d' file
```

变量转换

已匹配的字符串通过标记&来引用.

```
echo this is en example | seg 's/\w+/[&]/g'$>[this]  [is] [en] [example]
```

子串匹配标记

第一个匹配的括号内容使用标记 \1 来引用

```
sed 's/hello\([0-9]\)/\1/'
```

双引号求值

sed通常用单引号来引用；也可使用双引号，使用双引号后，双引号会对表达式求值：

```
sed 's/$var/HLLOE/'
```

当使用双引号时，我们可以在sed样式和替换字符串中指定变量；

```
eg:p=patten
r=replaced
echo "line con a patten" | sed "s/$p/$r/g"$>line con a replaced
```

其它示例

字符串插入字符：将文本中每行内容（PEKSHA）转换为 PEK/SHA

```
sed 's/^.\{3\}/&\//g' file
```

11、awk 数据流处理工具

awk脚本结构

awk ' BEGIN{ statements } statements2 END{ statements } '

工作方式

1.执行begin中语句块；

2.从文件或stdin中读入一行，然后执行statements2，重复这个过程，直到文件全部被读取完毕；

3.执行end语句块；

print 打印当前行

使用不带参数的print时，会打印当前行;

```
echo -e "line1\nline2" | awk 'BEGIN{print "start"} {print } END{ print "End" }'
```

print 以逗号分割时，参数以空格定界;

```
echo | awk ' {var1 = "v1" ; var2 = "V2"; var3="v3"; \
print var1, var2 , var3; }'$>v1 V2 v3
```

使用-拼接符的方式（""作为拼接符）；

```
echo | awk ' {var1 = "v1" ; var2 = "V2"; var3="v3"; \
print var1"-"var2"-"var3; }'$>v1-V2-v3
```

特殊变量： NR NF $0 $1 $2

NR:表示记录数量，在执行过程中对应当前行号；

NF:表示字段数量，在执行过程总对应当前行的字段数；

$0:这个变量包含执行过程中当前行的文本内容；

$1:第一个字段的文本内容；

$2:第二个字段的文本内容；

```
echo -e "line1 f2 f3\n line2 \n line 3" | awk '{print NR":"$0"-"$1"-"$2}'
```

打印每一行的第二和第三个字段：

```
awk '{print $2, $3}' file
```

统计文件的行数：

```
awk ' END {print NR}' file
```

累加每一行的第一个字段：

```
echo -e "1\n 2\n 3\n 4\n" | awk 'BEGIN{num = 0 ;
print "begin";} {sum += $1;} END {print "=="; print sum }'
```

传递外部变量

```
var=1000echo | awk '{print vara}' vara=$var #  输入来自stdinawk '{print vara}' vara=$var file # 输入来自文件
```

用样式对awk处理的行进行过滤

awk 'NR < 5' #行号小于5

awk 'NR==1,NR==4 {print}' file #行号等于1和4的打印出来

awk '/linux/' #包含linux文本的行（可以用正则表达式来指定，超级强大）

awk '!/linux/' #不包含linux文本的行

设置定界符

使用-F来设置定界符（默认为空格）

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

读取命令输出

使用getline，将外部shell命令的输出读入到变量cmdout中；

```
echo | awk '{"grep root /etc/passwd" | getline cmdout; print cmdout }'
```

在awk中使用循环

for(i=0;i<10;i++){print $i;}

for(i in array){print array[i];}

eg:

以逆序的形式打印行：(tac命令的实现）

```
seq 9| \
awk '{lifo[NR] = $0; lno=NR} \
END{ for(;lno>-1;lno--){print lifo[lno];}
} '
```

awk实现head、tail命令

head:

```
awk 'NR<=10{print}' filename
```

tail:

```
awk '{buffer[NR%10] = $0;} END{for(i=0;i<11;i++){ \
print buffer[i %10]} } ' filename
```

打印指定列

awk方式实现：

```
ls -lrt | awk '{print $6}'
```

cut方式实现

```
ls -lrt | cut -f6
```

打印指定文本区域

确定行号

```
seq 100| awk 'NR==4,NR==6{print}'
```

确定文本

打印处于start_pattern 和end_pattern之间的文本；

```
awk '/start_pattern/, /end_pattern/' filename
```

eg:

```
seq 100 | awk '/13/,/15/'cat /etc/passwd| awk '/mai.*mail/,/news.*news/'
```

awk常用内建函数

index(string,search_string):返回search_string在string中出现的位置

sub(regex,replacement_str,string):将正则匹配到的第一处内容替换为replacement_str;

match(regex,string):检查正则表达式是否能够匹配字符串；

length(string)：返回字符串长度

```
echo | awk '{"grep root /etc/passwd" | getline cmdout; print length(cmdout) }'
```

printf 类似c语言中的printf，对输出进行格式化

eg：

```
seq 10 | awk '{printf "->%4s\n", $1}'
```

12、迭代文件中的行、单词和字符

1. 迭代文件中的每一行

while 循环法

```
while read line;doecho $line;done < file.txt改成子shell:cat file.txt | (while read line;do echo $line;done)
```

awk法：

cat file.txt| awk '{print}'

2.迭代一行中的每一个单词

```
for word in $line;do echo $word;done
```

3. 迭代每一个字符

${string:start_pos:num_of_chars}：从字符串中提取一个字符；(bash文本切片）

${#word}:返回变量word的长度

```
for((i=0;i<${#word};i++))doecho ${word:i:1};done
```

来自：大CC

链接：http://www.cnblogs.com/me115/p/3427319.html

看一看，这么多能力你还差什么？
