# 文本处理三剑侠

## sed

字符流编辑器(Stream Editor),可以很好地完成对多个文件的一系列编辑工作

使用sed的小结：

1. 在一个或多个文件上自动实现编辑操作
2. 简化对多个文件执行相同的编辑处理工作
3. 编写转换程序

## awk

非常适合结构化数据的处理和格式化报表的生成

awk能够完成的工作：

1. 将文本文件看做由记录和字段组成的文本数据库
2. 使用变量操作数据库
3. 使用算术和字符串操作符
4. 使用普通的程序设计结构，例如循环和条件
5. 生成格式化报告
6. 定义函数
7. 从脚本中执行UNIX命令
8. 处理UNUX命令结果
9. 更加巧妙的处理命令行的参数
10. 更容易地处理多个输入流

### 分隔符

-F --field-separator 指定文本的分隔符,默认以空格作为分隔符

示例：

显示用户及其默认的shell

]# cat /etc/passwd | awk -F ":" '{print $1"\t"$7}'

root /bin/bash

bin /sbin/nologin

daemon /sbin/nologin

adm /sbin/nologin

lp /sbin/nologin

sync /bin/sync

用冒号作为分隔符，\t 以tab隔开显示

### BEGIN,END

1.在最前面添加name shell，最后面添加 blue /bin/nosh

]# cat /etc/passwd | awk -F ":" 'BEGIN {print "name\tshell"} {print $1"\t"$7} END {print "blue\t/bin/nosh"}'

name shell

root /bin/bash

bin /sbin/nologin

daemon /sbin/nologin

adm /sbin/nologin

lp /sbin/nologin

blue /bin/nosh

2.用BEGIN指定分隔符

awk ‘BEGIN{OFS=”#”}{print $1,$2}’ filename.log

### print

print的使用格式：

print item1, item2, item3

### 内置变量

1.记录变量

FS:指定输入分隔符，字段分隔符

OFS:输出分隔符

RS:输入文本信息的换行符

ORS:输出行分隔符

2.数据变量

NR: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所有文件中行统一计数

NF：当前记录的field个数

$NF: 最后一个field

### 条件语句

#### if else

语法：if (condition) {then-body} else {[ else-body ]}

示例：查看nginx日志中，状态码是403的记录

awk '{ if ($9=="403") print $0}' sms\_access.log

### 循环

#### for

##### 查看tcp各状态信息

~]# netstat -n | awk '/^tcp/ {++y[$NF]} END {for(x in y) print x, y[x]}'

ESTABLISHED 3

定义数据

++y[$NF] -> y数组名

$NF 每行的最后一field

++ 递增

#### while

语法：while(condition){ statement1; statemtnt2;...}

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

#### do while

语法：do { statemtnt1; statement2;...}

#### awk练习：

##### 提取nginx access.log里的IP地址、日期

awk 'BEGIN{FS="-";print "IP", "Date", "Count"}END{print $1,substr($3,3,2), substr($3,6,3), substr($3, 10,4)}' test.log

1. awk传递变量到shell

detect\_dd()

{

dd\_file='/tmp/test.log'

eval $(awk 'BEGIN{FS="-"}END{print "ip=" $1, "tdate=" substr($3,3,2), "tmon=" substr($3,6,3), "tyear=" substr($3,10,4)}' $dd\_file)

echo $ip " " $tdate " " $tmon " " $tyear

}

#### 自定义变量

awk -v test=”hello,awk” ‘{print test}’

注意，打印变量不需要$符号

## grep

Global search REgular expression and Print out the line

作用：文本搜索工具，根据用户指定的“模式”对目标文本逐行匹配

模式：过滤条件

grep –E == egrep 支持扩展正则表达式

grep –F == fgrep 不支持正则表达式

options:

--color=auto 对匹配到的文本着色后高亮显示

-i: ignore case

-o:只显示匹配到的字符串本身

-v: --invert-match 不能被模式匹配到的行

-E：支持扩展正则

-q: --quiet --slient 静默模式

-A num : 匹配到的后num行也显示

-B num ：匹配到的前num行也显示

-C num : 前后各num行

基本正则与扩展正则

Basic vs Extended Regular Expressions

In basic regular expressions the meta-characters ?, +, {, |, (, and ) lose their special meaning; instead use the

backslashed versions \?, \+, \{, \|, \(, and \).

经个人实践，基本正则如果要使用元字符，必须要加反斜杠，如果不加，会被当作是普通字符串处理。

实践：

1. 查找nginx日志每行中月份为Mar或Feb 日期在11到16日的ip地址

日志格式

10.42.73.21 - - [16/Mar/2017:15:42:17 +0800] "POST /?name=queue\_xx&opt=put&ver=2 HTTP/1.1" 200 24 "-" "Python-urllib/3.7" "-"

1. ]# grep "1[1-6]\/\(Mar\|Feb\)/2017" access.log|awk '{print $1}'

[1-6] 匹配1~6号

\(Mar\|Feb\) 匹配Mar Feb

awk ‘{print $1}’ 获取第一列，awk –F 默认是以空格分割，所以不用写

使用示例：

1. 匹配数字

[root@lily tmp]# echo "aa11ee11bb3443" | grep -o "[[:digit:]]\{3\}" #定死匹配3位

344

[root@lily tmp]# echo "aa11ee11bb3443" | grep -o "[[:digit:]]\{3,\}" #可以匹配3位及以上

3443

1. 匹配字母
2. 匹配空白