Linux 给人的印象是黑乎乎的神秘窗口，文本操作和数据处理似乎没有 Windows 窗口界面直观方便。其实Linux 有自己的独特的法宝，称之为 **三剑客：grep，awk 和 sed** 。你可以用这三件法宝很方便的处理数据 ： **查找，分段，修改** ，而这三个功能对应着我们今天的主角：grep，awk，sed。
形象一点比喻，如果把数据比作人群，那么 **grep 就是照妖镜** ，用来找出妖精； **awk 就是尺子** ， 给人群分门别类；而 **sed 就是宝剑** ，用来除掉妖精。当你明白为什么要用三剑客时，就更容易拿这三把剑去斩妖除魔。

1. grep

> grep-global regular expression print - 全局正则表达式打印可用于数据查找定位

- 先列举出测试工作常用的grep命令和意义：

```
grep pattern file

```

```
grep -i pattern file 忽略大小写

```

```
grep -v pattern file  不显示匹配行

```

```
grep -o pattern file 只把每个匹配的内容独立的行显示

```

```
grep -E pattern file 使用拓展正则表达式

```

```
#注意：grep 'a[0-9]\{10\}' 等同于 grep -E 'a[0-9]{10}'

```

```
grep -A -B -C pattern file 打印命中数据的上下文

```

```
grep pattern -r dir/ 递归搜索

```

```
grep -m1 匹配匹配中的第一个

```

```
grep -n 顺便输出行号

```

- 下面以一个检查首页是否有死链的案例需求来展示 grep 的匹配用法

以目前国内最大的测试社区网站 testerhome 为例，访问 testerhome 主页，找出主页中包含的左右 url,分别进行访问，如果访问成功会返回状态码200，检查所有访问成功的url并打印出来，若没访问成功就打印ERR加上失败的url。

1. 先访问 Testerhome 社区主页，利用 grep href 过滤出所有包含 url 的内容。命令：

```
curl -s
https://testerhome.com
  | grep href

```

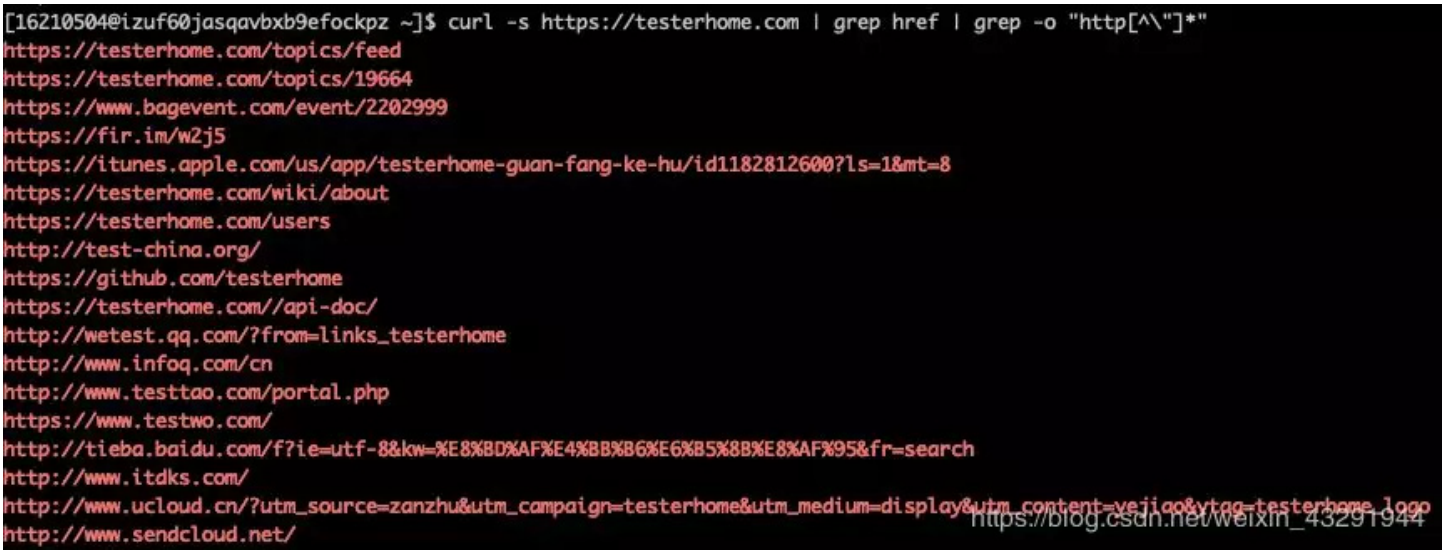
image968×461 202 KB

2.从返回的结果中取出 url,观察发现所有的 url 都被包在了双引号之中，那么在利用 grep -o 命令，加上正则表达式匹配，只打印从 http 开始到 url 结束双引号之前的内容。命令：

```
curl -s
https://testerhome.com
  | grep href | grep -o "http[^\"]*"

```

image861×326 90.5 KB



1. 从上一步中我们已经取出了完整的 url 了，现在我们需要对每个url进行访问取值判断

3.1. 先用curl -I 看看请求返回的头信息内容。命令：

```
curl -s -I
https://testerhome.com/topics/feed


```

``

3.2. 访问成功返回200，这时候我们一行一行去访问，再用grep命令匹配"200 OK"作为判断条件，筛选出成功的url并打印，然后将失败的 url 加上 ERR 标记也一起打印出来。命令

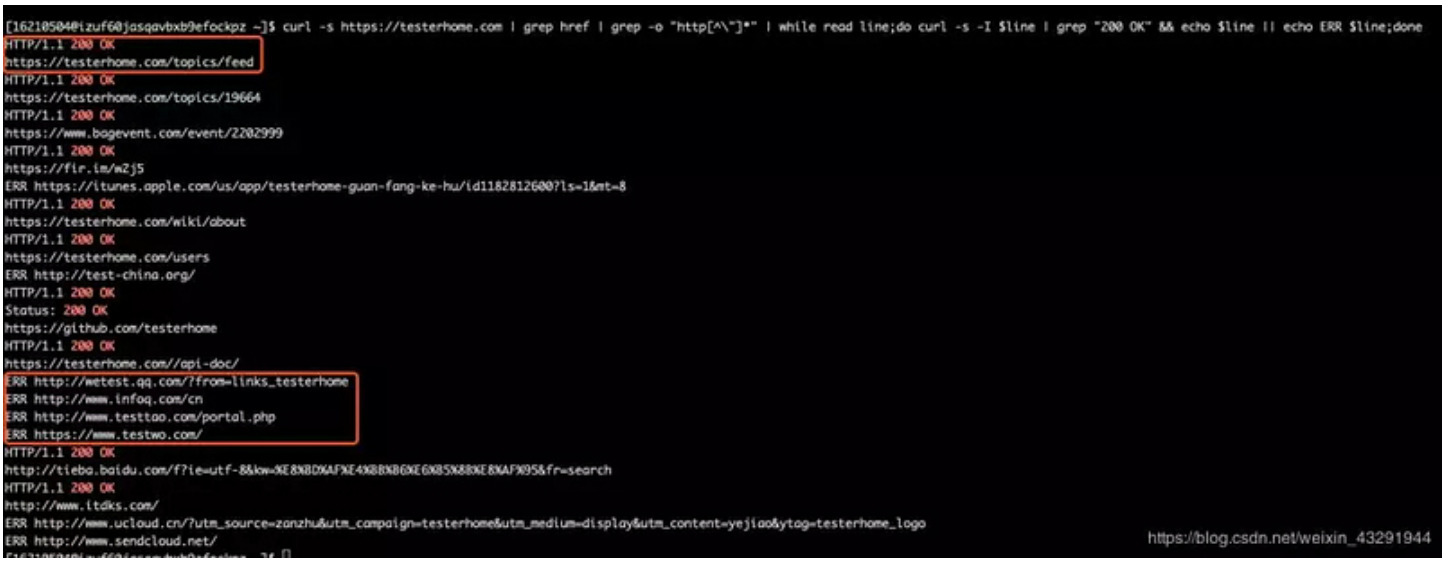
```

curl -s
```

```
https://testerhome.com
 | grep href | grep -o "http[^\"]*" | while read line;do curl -s -I
$line | grep 200 && echo $line || echo ERR $line;done
```

1. 最终结果展示

image1080×415 78.7 KB



2. awk

> awk = “Aho Weiberger and Kernighan” 三个作者的姓的第一个字母awk 是 Linux 下的一个命令，同时也是一种语言解析引擎awk 具备完整的编程特性。比如执行命令，网络请求等精通 awk,是一个 Linux 工作者的必备技能语法：awk ‘pattern{action}’

awk pattern语法

- awk 理论上可以代替 grep
- awk ‘pattern{action}’ ,默认以空格分隔

```
awk ‘BBEGIN{}END{}’ 开始和结束

```

```
awk ‘/Running/’ 正则匹配

```

```
awk ‘/aa/,/bb/’ 区间选择
```

```
2
```

```
1 awk ‘$2~/xxx/’ 字段匹配，这里指从第2个字段开始匹配包含xxx内容的行
2
```

```
1 awk ’NR==2’ 取第二行
2
```

```
1 awk ’NR>1’ 去掉第一行
2
```

awk的字段数据处理

- -F 参数指定字段分隔符
- BEGIN{FS=‘_’} 也可以表示分隔符

```
1 $0 代表原来的行
2
```

```
1 $1 代表第一个字段
2
```

```
1 $N 代表第N个字段
2
```

```
1 $NF 代表最后一个字段
2
```

下面以一个在nginx.log中查找返回状态码非200的请求响应数目的需求为例，演示awk的基础用法

有一份nginx.log文件，打开后内容格式如下：

```
1 220.181.108.111 - - [05/Dec/2018:00:11:42 +0000] "GET /topics/15225/show_wechat
  HTTP/1.1" 200 1684 "-" "Mozilla/5.0 (compatible; Baiduspider/2.0; +
  http://www.baidu.com/search/spider.html)
  " 0.029 0.029 .
2
```

```
1 216.244.66.241 - - [05/Dec/2018:00:11:42 +0000] "GET
  /topics/10052/replies/85845/reply_suggest HTTP/1.1" 301 5 "-" "Mozilla/5.0 (compatible;
  DotBot/1.1;
  http://www.opensiteexplorer.org/dotbot,
   help@moz.com)" 0.016 0.016 .
```

```
2
```

```
216.244.66.241 - - [05/Dec/2018:00:11:42 +0000] "GET /topics/10040?order_by=created_at
HTTP/1.1" 301 5 "-" "Mozilla/5.0 (compatible; DotBot/1.1;
http://www.opensiteexplorer.org/dotbot,
 help@moz.com)" 0.002 0.002 .
2
```

```
216.244.66.241 - - [05/Dec/2018:00:11:42 +0000] "GET
/topics/10043/replies/85544/reply_suggest HTTP/1.1" 301 5 "-" "Mozilla/5.0 (compatible;
DotBot/1.1;
http://www.opensiteexplorer.org/dotbot,
 help@moz.com)" 0.001 0.001 .
2
```

```
216.244.66.241 - - [05/Dec/2018:00:11:44 +0000] "GET /topics/10075/replies/89029/edit
HTTP/1.1" 301 5 "-" "Mozilla/5.0 (compatible; DotBot/1.1;
http://www.opensiteexplorer.org/dotbot,
 help@moz.com)" 0.001 0.001 .
2
```

```
216.244.66.241 - - [05/Dec/2018:00:11:44 +0000] "GET /topics/10075/replies/89631/edit
HTTP/1.1" 301 5 "-" "Mozilla/5.0 (compatible; DotBot/1.1;
http://www.opensiteexplorer.org/dotbot,
 help@moz.com)" 0.001 0.001 .
2
```

```
216.244.66.241 - - [05/Dec/2018:00:11:45 +0000] "GET /topics/10075?order_by=created_at
HTTP/1.1" 301 5 "-" "Mozilla/5.0 (compatible; DotBot/1.1;
http://www.opensiteexplorer.org/dotbot,
 help@moz.com)" 0.000 0.000 .
2
```

```
216.244.66.241 - - [05/Dec/2018:00:11:45 +0000] "GET /topics/10075?order_by=like
HTTP/1.1" 301 5 "-" "Mozilla/5.0 (compatible; DotBot/1.1;
http://www.opensiteexplorer.org/dotbot,
 help@moz.com)" 0.001 0.001 .
2
```

```
223.71.41.98 - - [05/Dec/2018:00:11:46 +0000] "GET /cable HTTP/1.1" 101 60749 "-"
"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:63.0) Gecko/20100101 Firefox/63.0" 2608.898
2608.898 .
2
```

```
113.87.161.17 - - [05/Dec/2018:00:11:39 +0000] "GET /cable HTTP/1.1" 101 3038 "-"
"Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/67.0.3396.62 Safari/537.36" 112.418 112.418 .

```

```
216.244.66.241 - - [05/Dec/2018:00:11:46 +0000] "GET /topics/10079/replies/119591/edit
HTTP/1.1" 301 5 "-" "Mozilla/5.0 (compatible; DotBot/1.1;
http://www.opensiteexplorer.org/dotbot,
 help@moz.com)" 0.001 0.001 .

```

```
216.244.66.241 - - [05/Dec/2018:00:11:46 +0000] "GET /topics/10089?locale=zh-TW
HTTP/1.1" 301 5 "-" "Mozilla/5.0 (compatible; DotBot/1.1;
http://www.opensiteexplorer.org/dotbot,
 help@moz.com)" 0.002 0.002 .

```

观察log内容，可以发现，以空格为分隔符，状态码在第九个字段位置；这里我们用awk命令从第九个字段位置开始匹配非200的状态码并打印出来。命令：

```
awk '$9!~/200/{print $9}' nginx.log

```

```
[avbxb9efockpz ~]$ awk '$9!~/200/{print $9}' nginx.log301

```

```
301

```

```
301

```

```
301

```

```
301

```

```
301

```

```
301
```

```
301
```

```
301
```

```
.....#剩余部分省略
```

再对取出的数据进行排序->去重->按数字的倒叙进行排列。命令：

```
awk '$9!~/200/{print $9}' nginx.log | sort | uniq -c | sort -nr
```

命令含义：

```
sort: 按从小到大进行排序
```

```
uniq -c :去重（相邻）
```

```
-nr: 按数字进行倒叙排序
```

```
-n:按数字进行排序
```

结果展示：

```

```

```
[sqavbxb9efockpz ~]$ awk '$9!~/200/{print $9}' nginx.log | sort | uniq -c | sort -nr
```

```
    433 101
```

```
```

```
    304 301
```

```
    266 404
```

```
    152 302
```

```
      7 401
```

```
      5 304
```

```
      2 499
```

```
      2 422
```

```
      1 500
```

再结合 awk ‘BBEGIN{}END{}’ 命令，以统计当前用户数目的例子来展示命令用法

使用 `cat /etc/passwd` 命令来查看本机用户，我们需要提取出用户名称并加上数字序号显示出来，达到这种效果：

```
1 nobody2 root
```

```
3 daemon
```

```
4 _uucp
```

```
5 _taskgated
```

```
6 _networkd
```

```
7 _installassistant
```

```
8 _lp
```

```
9 _postfix
```

```
......
```

用户信息：

```
localhost:~ qinzhen$ cat /etc/passwd
```

```
##
```

```
# User Database
```

```
#
```

```
# Note that this file is consulted directly only when the system is running
```

```
# in single-user mode.  At other times this information is provided by
```

```
# Open Directory.

```

```
#

```

```
# See the opendirectoryd(8) man page for additional information about

```

```
# Open Directory.

```

```
##

```

```
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false

```

```
root:*:0:0:System Administrator:/var/root:/bin/sh

```

```
daemon:*:1:1:System Services:/var/root:/usr/bin/false

```

```
_uucp:*:4:4:Unix to Unix Copy Protocol:/var/spool/uucp:/usr/sbin/uucico

```

```
_taskgated:*:13:13:Task Gate Daemon:/var/empty:/usr/bin/false

```

```
_networkd:*:24:24:Network Services:/var/networkd:/usr/bin/false

```

```
_installassistant:*:25:25:Install Assistant:/var/empty:/usr/bin/false

```

```
_lp:*:26:26:Printing Services:/var/spool/cups:/usr/bin/false
```

```

```

```
_postfix:*:27:27:Postfix Mail Server:/var/spool/postfix:/usr/bin/false

```

```
_scsd:*:31:31:Service Configuration Service:/var/empty:/usr/bin/false

```

```
_ces:*:32:32:Certificate Enrollment Service:/var/empty:/usr/bin/false

```

```
_appstore:*:33:33:Mac App Store Service:/var/empty:/usr/bin/false

```

```
_mcxalr:*:54:54:MCX AppLaunch:/var/empty:/usr/bin/false

```

```
......#后面的省略

```

```

```

思路：

```
* awk运行前先定义序号索引0，用来递增保存用户
* 利用awk将用户提取出来，按索引分别保存；
* 切片结束后再按行数进行循环，将数字序号与第一步保存的信息拼接打印
* 注意：

```

```
cat /etc/passwd

```

打印出的结果中，最上方的注释需要处理跳过

```
cat /etc/passwd | awk -F ':' 'BEGINE{userindex=0}
{user[userindex]=$1;userindex++}END{for(i=0;i<NR;i++)print i+1, user[i+10]}' |less

```

```
1 nobody
2 root
3 daemon
4 _uucp
5 _taskgated
6 _networkd
7 _installassistant
8 _lp
9 _postfix
10 _scsd
11 _ces
12 _appstore
13 _mcxalr
14 _appleevents
15 _geod
16 _devdocs
17 _sandbox
18 _mdnsresponder
19 _ard
20 _www
21 _eppc
22 _cvs
23 _svn
24 _mysql
25 _sshd
26 _qtss
27 _cyrus
28 _mailman
29 _appserver
```

3. sed

> sed:stream editor 根据定位到的数据行修改数据

```
sed [-nefri] [动作]

```

```
参数：

```

```
-n ：使用安静(slient)模式。只有经过sed特殊处理的那一行(或者操作)才会被列出来。一般与p配合使用

```

```
-e ：直接在命令行模式上进行sed的动作编辑
```

-f ：直接将sed动作写在一个文件内，-f filename则可以执行filename 内的sed动作。

-r ：sed的动作支持的是拓展正则表达式的语法(默认是基础正则表达式的语法)

-i ：直接修改读取的文件内容，而不是由屏幕输出

动作说明：[[n1][,n2]]function

n1,n2 ：不见得会存在，一般代表选择进行动作的行数，举例来说，如果我的动作是需要在10到20之间进行的，则“10,20[动作行为]”

function有下面这些参数：

a ：新增

d ：删除 （比较重要,测试工作中对数据处理时可快速去除无用信息，比如注释行，空白行等）

i ：插入

p ：打印 （一般与-n配合使用）

s ：替换（重中之重！！！，s参数可以说是日常测试工作中对数据用sed清理过滤时使用率最高的了）

```
2
```

sed 修改表达式： `sed 's/待修改/修改结果/'`

注意说明：

表达式单引号中的s表示修改，/ 符号表示分隔，实际上将/换成其他符号也可以，只要能起到分隔作用就OK

```
1 [16210504@izuf60jasqavbxb9efockpz ~]$ echo "aaa|bbb}|cccbbb" | sed 's/bbb/BBB/'
2
```

```
1 aaa|BBB}|cccbbb
2
```

```
1 [16210504@izuf60jasqavbxb9efockpz ~]$ echo "aaa|bbb}|cccbbb" | sed 's#bbb#BBB#'
2
```

```
1 aaa|BBB}|cccbbb
2
```

- 若想讲目标中所有的字段都替换，需要在命令最后加上g:

```
1 [16210504@izuf60jasqavbxb9efockpz ~]$ echo "aaa|bbb}|cccbbb" | sed 's/bbb/BBB/g'
2
```

```
1 aaa|BBB}|cccBBB
2
```

- sed还可以修改文件中的内容，现在有文件text.txt,内容如下：

```
1 [16210504@izuf60jasqavbxb9efockpz ~]$ cat text.txt
2
```

```
1 hello bash world
2
```

```
1 hi~ tester
2
```

```
1 go go go go!
2
```

用 `sed 's/hello/HELLO/' text.txt` 命令将文件中的 `hello` 替换成 `HELLO` ：

```
[16210504@izuf60jasqavbxb9efockpz ~]$ sed 's/hello/HELLO/' text.txt

```

```
HELLO bash world

```

```
hi~ tester

```

```
go go go go!

```

但是此时我们打开源text.txt文件发下源文件内容并未改变：

```
[16210504@izuf60jasqavbxb9efockpz ~]$ cat text.txt

```

```
hello bash world

```

```
hi~ tester

```

```
go go go go!

```

注意说明：

sed 在修改文件内容时，是另外开辟了一块模式空间，将修改后的内容放入并输出，源文件并未修改；这时如果想要修改源文件就需要借助 `-i` 命令，另外为了防止误操作修改文件，一般可以采取这种写法: `sed -i.bak 's/hello/HELLO/' text.txt` ，这种写法在修改源文件的同时还会生成一份以.bak结尾的备份文件，相较安全。

```
[16210504@izuf60jasqavbxb9efockpz ~]$ sed -i.bak 's/hello/HELLO/' text.txt

```

```
[16210504@izuf60jasqavbxb9efockpz ~]$ ls

```

```
1  1.sh  Allen_qin  nginx.log  test  text.txt  text.txt.bak  while_test
2
```

```
[16210504@izuf60jasqavbxb9efockpz ~]$ cat text.txt
```

```
HELLO bash world
```

```
hi~ tester
```

```
go go go go!
```

```
[16210504@izuf60jasqavbxb9efockpz ~]$ cat text.txt.bak
```

```
hello bash world
```

```
hi~ tester
```

```
go go go go!
```

`sed -e` 命令可以直接在命令行模式上进行sed的动作编辑，但看解释比较晦涩，来看一个实例：

需求： 现有一个1.txt的文本，内容如下：

```
a:
```

```
b:
```

```
c:
```

```
2
```

```
d:
```

```

```

要将其中每行末尾的 ： 都替换成 @ ，将 a 替换成 A ，并在文本末尾加上“ Sed Test ”

命令：

```
sed -i -e 's/:/@/g' \
```

```
-i -e 's/a/A/' \
```

```
-i -e '$a Sed Test' 1.txt
```

实例演示：

```
[16210504@izuf60jasqavbxb9efockpz ~]$ sed -i -e 's/:/@/g' -i -e 's/a/A/' -i -e '$a Sed Test' 1.txt
```

```
[16210504@izuf60jasqavbxb9efockpz ~]$ cat 1.txt
```

```
A@
```

```
b@
```

```
c@
```

```
d@
```

```
2
```

```
1 Sed Test
```