

# telnet命令

2022年2月18日 12:44

## 1. 命令格式:

telnet[参数][主机]

## 2. 命令功能:

执行telnet指令开启终端机阶段作业，并登入远端主机。

## 3. 命令参数:

- 8 允许使用8位字符资料，包括输入与输出。
- a 尝试自动登入远端系统。
- b<主机别名> 使用别名指定远端主机名称。
- c 不读取用户专属目录里的.telnetrc文件。
- d 启动排错模式。
- e<脱离字符> 设置脱离字符。
- E 滤除脱离字符。
- f 此参数的效果和指定"-F"参数相同。
- F 使用Kerberos V5认证时，加上此参数可把本地主机的认证数据上传到远端主机。
- k<域名> 使用Kerberos认证时，加上此参数让远端主机采用指定的领域名，而非该主机的域名。
- K 不自动登入远端主机。
- l<用户名称> 指定要登入远端主机的用户名称。
- L 允许输出8位字符资料。
- n<记录文件> 指定文件记录相关信息。
- r 使用类似rlogin指令的用户界面。
- S<服务类型> 设置telnet连线所需的IP TOS信息。
- x 假设主机有支持数据加密的功能，就使用它。

-X<认证形态> 关闭指定的认证形态。

#### 4. 使用实例:

##### 实例1: 远程服务器无法访问

命令:

```
telnet 192.168.120.206
```

输出:

```
[root@localhost ~]# telnet 192.168.120.209
Trying 192.168.120.209...
telnet: connect to address 192.168.120.209: No route to host
telnet: Unable to connect to remote host: No route to host
[root@localhost ~]#
```

说明:

处理这种情况方法:

- (1) 确认ip地址是否正确?
- (2) 确认ip地址对应的主机是否已经开机?
- (3) 如果主机已经启动, 确认路由设置是否正确? (使用route命令查看)
- (4) 如果主机已经启动, 确认主机上是否开启了telnet服务? (使用netstat命令查看, TCP的23端口是否有LISTEN状态的行)
- (5) 如果主机已经启动telnet服务, 确认防火墙是否放开了23端口的访问? (使用iptables-save查看)

##### 实例2: 域名无法解析

命令:

```
telnet www.baidu.com
```

输出:

```
[root@localhost ~]# telnet www.baidu.com
www.baidu.com/telnet: Temporary failure in name resolution
[root@localhost ~]#
```

说明:

处理这种情况方法:

- (1) 确认域名是否正确
- (2) 确认本机的域名解析有关的设置是否正确 (/etc/resolv.conf中nameserver的设置是否正确, 如果没有, 可以使用nameserver 8.8.8.8)
- (3) 确认防火墙是否放开了UDP53端口的访问 (DNS使用UDP协议, 端口53, 使用iptables-save查看)

##### 实例3:

命令:

```
telnet 192.168.120.206
```

输出:

```
[root@localhost ~]# telnet 192.168.120.206
Trying 192.168.120.206...
telnet: connect to address 192.168.120.206: Connection refused
telnet: Unable to connect to remote host: Connection refused
[root@localhost ~]#
```

说明:

处理这种情况:

- (1) 确认ip地址或者主机名是否正确?
- (2) 确认端口是否正确, 是否默认的23端口

#### 实例4: 启动telnet服务

命令:

```
service xinetd restart
```

输出:

```
[root@localhost ~]# cd /etc/xinetd.d/
```

```
[root@localhost xinetd.d]# ll
```

总计 124

```
-rw-r--r-- 1 root root 1157 2011-05-31 chargen-dgram
-rw-r--r-- 1 root root 1159 2011-05-31 chargen-stream
-rw-r--r-- 1 root root 523 2009-09-04 cvs
-rw-r--r-- 1 root root 1157 2011-05-31 daytime-dgram
-rw-r--r-- 1 root root 1159 2011-05-31 daytime-stream
-rw-r--r-- 1 root root 1157 2011-05-31 discard-dgram
-rw-r--r-- 1 root root 1159 2011-05-31 discard-stream
-rw-r--r-- 1 root root 1148 2011-05-31 echo-dgram
-rw-r--r-- 1 root root 1150 2011-05-31 echo-stream
-rw-r--r-- 1 root root 323 2004-09-09 eklogin
-rw-r--r-- 1 root root 347 2005-09-06 ekrb5-telnet
-rw-r--r-- 1 root root 326 2004-09-09 gssftp
-rw-r--r-- 1 root root 310 2004-09-09 klogin
-rw-r--r-- 1 root root 323 2004-09-09 krb5-telnet
-rw-r--r-- 1 root root 308 2004-09-09 kshell
-rw-r--r-- 1 root root 317 2004-09-09 rsync
-rw-r--r-- 1 root root 1212 2011-05-31 tcpmux-server
-rw-r--r-- 1 root root 1149 2011-05-31 time-dgram
-rw-r--r-- 1 root root 1150 2011-05-31 time-stream
```

```
[root@localhost xinetd.d]# cat krb5-telnet
```

```
# default: off
```

```
# description: The kerberized telnet server accepts normal telnet sessions, \
#               but can also use Kerberos 5 authentication.
```

```
service telnet
```

```
{
    flags          = REUSE
    socket_type    = stream
    wait           = no
    user           = root
    server         = /usr/kerberos/sbin/telnetd
    log_on_failure += USERID
    disable        = yes
}
```

```
[root@localhost xinetd.d]#
```

说明：

配置参数，通常的配置如下：

```
service telnet
{
  disable = no #启用
  flags = REUSE #socket可重用
  socket_type = stream #连接方式为TCP
  wait = no #为每个请求启动一个进程
  user = root #启动服务的用户为root
  server = /usr/sbin/in.telnetd #要激活的进程
  log_on_failure += USERID #登录失败时记录登录用户名
}
```

如果要配置允许登录的客户端列表，加入

```
only_from = 192.168.0.2 #只允许192.168.0.2登录
```

如果要配置禁止登录的客户端列表，加入

```
no_access = 192.168.0.{2,3,4} #禁止192.168.0.2、192.168.0.3、192.168.0.4登录
```

如果要设置开放时段，加入

```
access_times = 9:00-12:00 13:00-17:00 # 每天只有这两个时段开放服务（我们的上班时间）
```

如果你有两个IP地址，一个是私网的IP地址如192.168.0.2，一个是公网的IP地址如218.75.74.83，如果你希望用户只能从私网来登录telnet服务，那么加入

```
bind = 192.168.0.2
```

各配置项具体的含义和语法可参考xined配置文件属性说明（man xinetd.conf）

配置端口，修改services文件：

```
# vi /etc/services
```

找到以下两句

```
telnet 23/tcp
```

```
telnet 23/udp
```

如果前面有#字符，就去掉它。telnet的默认端口是23，这个端口也是黑客端口扫描的主要对象，因此最好将这个端口修改掉，修改的方法很简单，就是将23这个数字修改掉，改成大一点的数字，比如61123。注意，1024以下的端口号是internet保留的端口号，因此最好不要用，还应该注意不要与其它服务的端口冲突。

启动服务：

```
service xinetd restart
```

### 实例5：正常telnet

命令：

```
telnet 192.168.120.204
```

输出：

```
[root@andy ~]# telnet 192.168.120.204
```

```
Trying 192.168.120.204...
```

```
Connected to 192.168.120.204 (192.168.120.204).
```

```
Escape character is '^]'.
```

```
localhost (Linux release 2.6.18-274.18.1.el5 #1 SMP Thu Feb 9 12:45:44 EST  
2012) (1)
```

```
login: root
```

```
Password:
```

```
Login incorrect
```

说明：

一般情况下不允许root从远程登录，可以先用普通账号登录，然后再用su -切换到root用户。

# wget命令

2022年2月18日 14:38

wget是一个下载文件的工具，它用在命令行下。对于Linux用户是必不可少的工具，我们经常要下载一些软件或从远程服务器恢复备份到本地服务器。

wget支持HTTP，HTTPS和FTP协议，可以使用HTTP代理。所谓的自动下载是指，wget可以在用户退出系统的之后在后台执行。这意味这你可以登录系统，启动一个wget下载任务，然后退出系统，wget将在后台执行直到任务完成

wget 可以跟踪HTML页面上的链接依次下载来创建远程服务器的本地版本，完全重建原始站点的目录结构。这又常被称作”递归下载”。

wget 非常稳定，它在带宽很窄的情况下和不稳定网络中有很强的适应性. 如果是由于网络的原因下载失败，wget会不断的尝试，直到整个文件下载完毕。如果是服务器打断下载过程，它会再次联到服务器上从停止的地方继续下载。这对从那些限定了链接时间的服务器上下载大文件非常有用。

## 常用的命令展示

1、使用wget -O下载并以不同的文件名保存(-O: 下载文件到对应目录，并且修改文件名称)

wget -O wordpress.zip <http://www.minjieren.com/download.aspx?id=1080>

```
[root@localhost ~]# wget -O wordpress.zip http://www.minjieren.com/download.aspx?id=1080
--2018-07-04 07:21:21-- http://www.minjieren.com/download.aspx?id=1080
Resolving www.minjieren.com... 64.120.101.230
Connecting to www.minjieren.com[64.120.101.230]:80... connected.
HTTP request sent, awaiting response... 200 OK http://www.cnblogs.com/ftl1012/
Length: 7678 (7.5K) [text/html]
Saving to: 'wordpress.zip'
100%[=====>] 7,678
```

2、使用wget -b后台下载

wget -b <a href="http://www.minjieren.com/wordpress-3.1-zh\_CN.zip" rel="noopener">  
http://www.minjieren.com/wordpress-3.1-zh\_CN.zip</a>

备注： 你可以使用以下命令来察看下载进度：tail -f wget-log

3、利用-spider: 模拟下载，不会下载，只是会检查是否网站是否好着

wget --spider [www.baidu.com](http://www.baidu.com) #不下载任何文件

```
[root@localhost ~]# wget --spider www.baidu.com
Spider mode enabled. Check if remote file exists.
--2018-07-04 07:15:37-- http://www.baidu.com/
Resolving www.baidu.com... 220.181.112.244
Connecting to www.baidu.com[220.181.112.244]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 277 [text/html]
Remote file exists and could contain further links,
but recursion is disabled -- not retrieving.
```

4、模拟下载打印服务器响应

wget -S [www.baidu.com](http://www.baidu.com) # 打印服务器响应

```
[root@localhost ~]# wget -S www.baidu.com
Spider mode enabled. Check if remote file exists.--2018-07-04 07:16:14-- http://www.baidu.com/
Resolving www.baidu.com... 220.181.112.244
Connecting to www.baidu.com|220.181.112.244|:80... connected.
HTTP request sent, awaiting response...
HTTP/1.0 200 OK
Accept-Ranges: bytes
Cache-Control: private, no-cache, no-store, proxy-revalidate, no-transform
Content-Length: 2381
Content-Type: text/html http://www.cnblogs.com/ftl1012/
Date: Wed, 04 Jul 2018 14:16:14 GMT
Etag: "588604c8-94d"
Last-Modified: Mon, 23 Jan 2017 13:27:36 GMT
Pragma: no-cache
Server: bfe/1.0.8.18
Set-Cookie: BDORZ=27315; max-age=86400; domain=.baidu.com; path=/
Length: 2381 (2.3K) [text/html]
Saving to: "index.html"

100%[=====>] 2,381 --.-K/s
```

## 5、设定指定次数

wget -r --tries=2 [www.baidu.com](http://www.baidu.com) (指定尝试2次，2次后不再尝试)

wget -r --tries=2 -q [www.baidu.com](http://www.baidu.com) (指定尝试，且不打印中间结果)

```
[root@localhost ~]# wget -r --tries=2 www.baidu.com
--2018-07-04 07:18:19-- http://www.baidu.com/
Resolving www.baidu.com... 220.181.112.244, 220.181.111.188
Connecting to www.baidu.com|220.181.112.244|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2381 (2.3K) [text/html]
Saving to: "www.baidu.com/index.html"

100%[=====>] 2,381 --.-K/s in 0s

2018-07-04 07:18:19 (35.2 MB/s) - "www.baidu.com/index.html" saved [2381/2381]

Loading robots.txt; please ignore errors. http://www.cnblogs.com/ftl1012/
--2018-07-04 07:18:19-- http://www.baidu.com/robots.txt
Connecting to www.baidu.com|220.181.112.244|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2754 (2.7K) [text/plain]
Saving to: "www.baidu.com/robots.txt"

100%[=====>] 2,754 --.-K/s in 0s

2018-07-04 07:18:19 (7.33 MB/s) - "www.baidu.com/robots.txt" saved [2754/2754]

FINISHED --2018-07-04 07:18:19--
Downloaded: 2 files, 5.0K in 0s (11.6 MB/s)
```

## wget详细命令参数

GNU Wget 1.12，非交互式的网络文件下载工具。

用法： wget [选项]... [URL]...

长选项所必须的参数在使用短选项时也是必须的。

开始：

-V, --version	显示 Wget 的版本信息并退出。
-h, --help	打印此帮助。
-b, --background	启动后转入后台。
-e, --execute=COMMAND	运行一个 ‘.wgetrc’ 风格的命令。

登入并输入文件：

-o, --output-file=FILE	将信息写入 FILE。
-a, --append-output=FILE	将信息添加至 FILE。
-d, --debug	打印大量调试信息。
-q, --quiet	安静模式(无信息输出)。
-v, --verbose	详尽的输出(此为默认值)。
-nv, --no-verbose	关闭详尽输出，但不进入安静模式。
-i, --input-file=FILE	下载本地或外部 FILE 中的 URLs。
-F, --force-html	把输入文件当成 HTML 文件。
-B, --base=URL	解析与 URL 相关的 HTML 输入文件（由 -i -F 选项指定）。

#### 下载:

-t, --tries=NUMBER	设置重试次数为 NUMBER (0 代表无限制)。
--retry-connrefused	即使拒绝连接也是重试。
-O, --output-document=FILE	将文档写入 FILE。
-nc, --no-clobber	不要重复下载已存在的文件。
-c, --continue	继续下载部分下载的文件。
--progress=TYPE	选择进度条类型。
-N, --timestamping	只获取比本地文件新的文件。
-S, --server-response	打印服务器响应。
--spider	不下载任何文件。
-T, --timeout=SECONDS	将所有超时设为 SECONDS 秒。
--dns-timeout=SECS	设置 DNS 查寻超时为 SECS 秒。
--connect-timeout=SECS	设置连接超时为 SECS 秒。
--read-timeout=SECS	设置读取超时为 SECS 秒。
-w, --wait=SECONDS	等待间隔为 SECONDS 秒。
--waitretry=SECONDS	在取回文件的重试期间等待 1..SECONDS 秒。
--random-wait	取回时等待 0...2*WAIT 秒。
--no-proxy	关闭代理。
-Q, --quota=NUMBER	设置取回配额为 NUMBER 字节。
--bind-address=ADDRESS	绑定至本地主机上的 ADDRESS (主机名或是 IP)。
--limit-rate=RATE	限制下载速率为 RATE。
--no-dns-cache	关闭 DNS 查寻缓存。
--restrict-file-names=OS	限定文件名中的字符为 OS 允许的字符。
--ignore-case	匹配文件/目录时忽略大小写。
-4, --inet4-only	仅连接至 IPv4 地址。
-6, --inet6-only	仅连接至 IPv6 地址。
--prefer-family=FAMILY	首先连接至指定协议的地址 FAMILY 为 IPv6, IPv4 或是 none。
--user=USER	将 ftp 和 http 的用户名均设置为 USER。
--password=PASS	将 ftp 和 http 的密码均设置为 PASS。
--ask-password	提示输入密码。
--no-iri	关闭 IRI 支持。
--local-encoding=ENC	IRI 使用 ENC 作为本地编码。
--remote-encoding=ENC	使用 ENC 作为默认远程编码。

#### 目录:

-nd, --no-directories	不创建目录。
-x, --force-directories	强制创建目录。
-nH, --no-host-directories	不要创建主目录。
--protocol-directories	在目录中使用协议名称。
-P, --directory-prefix=PREFIX	以 PREFIX/... 保存文件
--cut-dirs=NUMBER	忽略 NUMBER 个远程目录路径。

#### HTTP 选项:

--http-user=USER	设置 http 用户名为 USER。
--http-password=PASS	设置 http 密码为 PASS。
--no-cache	不在服务器上缓存数据。
--default-page=NAME	改变默认页 (默认页通常是“index.html”)。
-E, --adjust-extension	以合适的扩展名保存 HTML/CSS 文档。
--ignore-length	忽略头部的‘Content-Length’区域。
--header=STRING	在头部插入 STRING。
--max-redirect	每页所允许的最大重定向。
--proxy-user=USER	使用 USER 作为代理用户名。
--proxy-password=PASS	使用 PASS 作为代理密码。
--referer=URL	在 HTTP 请求头包含‘Referer: URL’。
--save-headers	将 HTTP 头保存至文件。
-U, --user-agent=AGENT	标识为 AGENT 而不是 Wget/VERSION。
--no-http-keep-alive	禁用 HTTP keep-alive(永久连接)。
--no-cookies	不使用 cookies。
--load-cookies=FILE	会话开始前从 FILE 中载入 cookies。
--save-cookies=FILE	会话结束后保存 cookies 至 FILE。



<code>--keep-session-cookies</code>	载入并保存会话(非永久) cookies。
<code>--post-data=STRING</code>	使用 POST 方式; 把 STRING 作为数据发送。
<code>--post-file=FILE</code>	使用 POST 方式; 发送 FILE 内容。
<code>--content-disposition</code>	当选中本地文件名时 允许 Content-Disposition 头部(尚在实验)。
<code>--auth-no-challenge</code>	send Basic HTTP authentication information without first waiting for the server's challenge.

#### HTTPS (SSL/TLS) 选项:

<code>--secure-protocol=PR</code>	选择安全协议, 可以是 auto、SSLv2、 SSLv3 或是 TLSv1 中的一个。
<code>--no-check-certificate</code>	不要验证服务器的证书。
<code>--certificate=FILE</code>	客户端证书文件。
<code>--certificate-type=TYPE</code>	客户端证书类型, PEM 或 DER。
<code>--private-key=FILE</code>	私钥文件。
<code>--private-key-type=TYPE</code>	私钥文件类型, PEM 或 DER。
<code>--ca-certificate=FILE</code>	带有一组 CA 认证的文件。
<code>--ca-directory=DIR</code>	保存 CA 认证的哈希列表的目录。
<code>--random-file=FILE</code>	带有生成 SSL PRNG 的随机数据的文件。
<code>--egd-file=FILE</code>	用于命名带有随机数据的 EGD 套接字的文件。

#### FTP 选项:

<code>--ftp-user=USER</code>	设置 ftp 用户名为 USER。
<code>--ftp-password=PASS</code>	设置 ftp 密码为 PASS。
<code>--no-remove-listing</code>	不要删除 '.listing' 文件。
<code>--no-glob</code>	不在 FTP 文件名中使用通配符展开。
<code>--no-passive-ftp</code>	禁用 "passive" 传输模式。
<code>--retr-symlinks</code>	递归目录时, 获取链接的文件(而非目录)。

#### 递归下载:

<code>-r, --recursive</code>	指定递归下载。
<code>-l, --level=NUMBER</code>	最大递归深度( inf 或 0 代表无限制, 即全部下载)。
<code>--delete-after</code>	下载完成后删除本地文件。
<code>-k, --convert-links</code>	让下载得到的 HTML 或 CSS 中的链接指向本地文件。
<code>-K, --backup-converted</code>	在转换文件 X 前先将它备份为 X.orig。
<code>-m, --mirror</code>	-N -r -l inf --no-remove-listing 的缩写形式。
<code>-p, --page-requisites</code>	下载所有用于显示 HTML 页面的图片之类的元素。
<code>--strict-comments</code>	开启 HTML 注释的精确处理(SGML)。

#### 递归接受/拒绝:

<code>-A, --accept=LIST</code>	逗号分隔的可接受的扩展名列表。
<code>-R, --reject=LIST</code>	逗号分隔的要拒绝的扩展名列表。
<code>-D, --domains=LIST</code>	逗号分隔的可接受的域列表。
<code>--exclude-domains=LIST</code>	逗号分隔的要拒绝的域列表。
<code>--follow-ftp</code>	跟踪 HTML 文档中的 FTP 链接。
<code>--follow-tags=LIST</code>	逗号分隔的跟踪的 HTML 标识列表。
<code>--ignore-tags=LIST</code>	逗号分隔的忽略的 HTML 标识列表。
<code>-H, --span-hosts</code>	递归时转向外部主机。
<code>-L, --relative</code>	只跟踪有关系的链接。
<code>-I, --include-directories=LIST</code>	允许目录的列表。
<code>-X, --exclude-directories=LIST</code>	排除目录的列表。
<code>-np, --no-parent</code>	不追溯至父目录。

# yum命令

2022年2月18日 15:01

yum, 全称“Yellow dog Updater, Modified”, 是一个专门为了解决包的依赖关系而存在的软件包管理器。就好像 Windows 系统上可以通过 360 软件管家实现软件的一键安装、升级和卸载, Linux 系统也提供有这样的工具, 就是 yum。

可以这么说, yum 是改进型的 RPM 软件管理器, 它很好的解决了 RPM 所面临的软件包依赖问题。yum 在服务器端存所有的 RPM 包, 并将各个包之间的依赖关系记录在文件中, 当管理员使用 yum 安装 RPM 包时, yum 会先从服务器端下载包的依赖性文件, 通过分析此文件从服务器端一次性下载所有相关的 RPM 包并进行安装。

## yum查询命令

使用 yum 对软件包执行查询操作, 常用命令可分为以下几种:

- yum list: 查询所有已安装和可安装的软件包。例如

```
[root@localhost yum.repos.d]# yum list
#查询所有可用软件包列表
Installed Packages
#已经安装的软件包
ConsdeKit.i686 0.4.1-3.el6
@anaconda-CentOS-201207051201 J386/6.3
ConsdeKit-libs.i686 0.4.1-3.el6 @anaconda-CentOS-201207051201 J386/6.3
...省略部分输出...
Available Packages
#还可以安装的软件包
389-ds-base.i686 1.2.10.2-15.el6 c6-media
389-ds-base-devel.i686 1.2.10.2-15.el6 c6-media
#软件名 版本 所在位置 (光盘)
...省略部分输出...
```

- yum list 包名: 查询执行软件包的安装情况。例如:

```
[root@localhost yum.repos.d]# yum list samba
Available Packages samba.i686 3.5.10-125.el6 c6-media
#查询 samba 软件包的安装情况
```

- yum search 关键字: 从 yum 源服务器上查找与关键字相关的所有软件包。例如:

```
[root@localhost yum.repos.d]# yum search samba
#搜索服务器上所有和samba相关的软件包
```

```
=====N/S Matched:
samba =====
samba-client.i686: Samba client programs
samba-common.i686: Files used by both Samba servers and clients
samba-doc.i686: Documentation for the Samba suite
...省略部分输出...
Name and summary matches only, use "search all" for everything.
```

- `yum info` 包名: 查询执行软件包的详细信息。例如:

```
[root@localhost yum.repos.d]# yum info samba
#查询samba软件包的信息
Available Packages <-没有安装
Name : samba <-包名
Arch : i686 <-适合的硬件平台
Version : 3.5.10 <-版本
Release : 125.el6 <-发布版本
Size : 4.9M <-大小
Repo : c6-media <-在光盘上
...省略部分输出...
```

## yum安装命令

yum 安装软件包的命令基本格式为:

```
[root@localhost yum.repos.d]# yum -y install 包名
```

其中:

- `install`: 表示安装软件包。
- `-y`: 自动回答 `yes`。如果不加 `-y`, 那么每个安装的软件都需要手工回答 `yes`;

例如使用此 yum 命令安装 gcc:

```
[root@localhost yum.jepos.d]#yum -y install gcc
#使用yum自动安装gcc
```

gcc 是 C 语言的编译器, 鉴于该软件包涉及到的依赖包较多, 建议使用 yum 命令安装。

## yum 升级命令

使用 yum 升级软件包, 需确保 yum 源服务器中软件包的版本比本机安装的软件包版本高。

yum 升级软件包常用命令如下:

- `yum -y update`: 升级所有软件包。不过考虑到服务器强调稳定性, 因此该命令并不常用。
- `yum -y update 包名`: 升级特定的软件包。

## yum 卸载命令

使用 yum 卸载软件包时，会同时卸载所有与该包有依赖关系的其他软件包，即便有依赖包属于系统运行必备文件，也会被 yum 无情卸载，带来的直接后果就是使系统崩溃。

除非你能确定卸载此包以及它的所有依赖包不会对系统产生影响，否则不要使用 yum 卸载软件包。

yum 卸载命令的基本格式如下：

```
[root@localhost yum.repos.d]# yum remove 包名  
#卸载指定的软件包
```

例如，使用 yum 卸载 samba 软件包的命令如下：

```
[root@localhost yum.repos.d]# yum remove samba  
#卸载samba软件包
```

# netstat命令

2022年2月18日 15:39

## 一、介绍

Netstat是控制台命令,是一个监控TCP/IP网络的非常有用的工具,它可以显示路由表、实际的网络连接以及每一个网络接口设备的状态信息。Netstat用于显示与IP、TCP、UDP和ICMP协议相关的统计数据,一般用于检验本机各端口的网络连接情况。

## 二、输出信息描述

执行netstat后输出如下:

```
[root@sy-suz-srv51 ~]# netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 k8sdev.sui:sun-sr-https k8sdev.suiyi.com.:34880 SYN_RECV
tcp      0      0 k8sdev.suiyi.com.c:2379 10.1.62.21:47910      ESTABLISHED
tcp      0      0 k8sdev.suiyi.com.c:2379 k8sdev.suiyi.com.:37790 ESTABLISHED
tcp      0      0 sy-suz-srv:pcsync-https 10.1.62.162:49200     ESTABLISHED
tcp      0      0 k8sdev.suiyi.com.:52866 k8sdev.sui:sun-sr-https ESTABLISHED
tcp      0      0 k8sdev.suiyi.com.:37728 k8sdev.suiyi.com.c:2379 ESTABLISHED
tcp      0      0 k8sdev.sui:sun-sr-https k8sdev.suiyi.com.:52852 ESTABLISHED
tcp      0      0 k8sdev.sui:sun-sr-https 10.1.62.162:32841     ESTABLISHED
tcp      0      0 sy-suz-srv:pcsync-https sy-suz-srv51:60094    ESTABLISHED
tcp      0      0 localhost:webcache       localhost:40136       ESTABLISHED
tcp      0      0 k8sdev.suiyi.com.:35466 10.1.62.21:sun-sr-https ESTABLISHED
tcp      0      0 k8sdev.suiyi.com.:34358 10.1.62.21:sun-sr-https ESTABLISHED

Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags               Type               State              I-Node    Path
unix   3      [ ]                   DGRAM              18442             /run/systemd/notify
unix   2      [ ]                   DGRAM              18444             /run/systemd/cgroups-agent
unix   2      [ ]                   DGRAM              23822             /var/run/chrony/chronyd.sock
unix   8      [ ]                   DGRAM              18455             /run/systemd/journal/socket
unix  18      [ ]                   DGRAM              18457             /dev/log
unix   2      [ ]                   DGRAM              14151             /var/run/nscd/socket
unix   2      [ ]                   DGRAM              584               /run/systemd/shutdown
unix   3      [ ]                   STREAM             CONNECTED          124439388 /run/dbus/system_bus_socket
unix   3      [ ]                   STREAM             CONNECTED          42312          /run/systemd/journal/stdout
unix   3      [ ]                   STREAM             CONNECTED          39909
unix   3      [ ]                   STREAM             CONNECTED          21675
unix   3      [ ]                   STREAM             CONNECTED          47538
unix   3      [ ]                   STREAM             CONNECTED          124585242 /var/run/docker/containerd/docker-containerd.sock
unix   3      [ ]                   STREAM             CONNECTED          21658
unix   2      [ ]                   STREAM             CONNECTED          30160
unix   3      [ ]                   STREAM             CONNECTED          33750          /run/systemd/journal/stdout
unix   3      [ ]                   STREAM             CONNECTED          124614293 @/containerd-shim/moby/c44e49ee0f86d8a4109afb17670179
unix   3      [ ]                   STREAM             CONNECTED          124609611 @/containerd-shim/moby/a736ba153c07f0bbf099ae1a1060953
unix   3      [ ]                   STREAM             CONNECTED          124601653 @/containerd-shim/moby/20d3fd59d03455d45b1da2636fca25
```

netstat的输出结果可以分为两个部分

- 1、Active Internet connections 有源TCP连接, 其中"Recv-Q"和"Send-Q"指接收队列和发送队列。这些数字一般都应该都是0。如果不是则表示软件包正在队列中堆积。这种情况只能在非常少的情况见到。
- 2、Active UNIX domain sockets 有源Unix域套接口(和网络套接字一样, 但是只能用于本机通信, 性能可以提

高一倍)。

列名解释:

Proto: 显示连接使用的协议。

RefCnt: 表示连接到本套接口上的进程号。

Types: 显示套接口的类型。

State: 显示套接口当前的状态。

Path: 表示连接到套接口的其它进程使用的路径名。

### 三、netstat常见参数

- a (all) 显示所有选项, 默认不显示LISTEN相关。
- t (tcp) 仅显示tcp相关选项。
- u (udp) 仅显示udp相关选项。
- n 拒绝显示别名, 能显示数字的全部转化成数字。
- l 仅列出有在 Listen (监听) 的服务状态。
- p 显示建立相关链接的程序名
- r 显示路由信息, 路由表
- e 显示扩展信息, 例如uid等
- s 按各个协议进行统计
- c 每隔一个固定时间, 执行该netstat命令。

LISTEN和LISTENING的状态只有用-a或者-l才能看到。

### 四、常用netstat相关命令

- 1、列出所有端口 #netstat -a
- 2、列出所有 tcp 端口 #netstat -at
- 3、列出所有 udp 端口 #netstat -au
- 4、只显示监听端口 #netstat -l
- 5、只列出所有监听 tcp 端口 #netstat -lt
- 6、只列出所有监听 udp 端口 #netstat -lu
- 7、列出所有监听 UNIX 端口 #netstat -lx

8、显示所有端口的统计信息 #netstat -s

9、显示 TCP 或 UDP 端口的统计信息 #netstat -st 或 -su

10、输出中显示 PID 和进程名称 #netstat -p

11、netstat 输出中不显示主机，端口和用户名 (host, port or user)

当你不想让主机，端口和用户名显示，使用 netstat -n。将会使用数字代替那些名称。

同样可以加速输出，因为不用进行比对查询。

```
#netstat -an
```

如果只是不想让这三个名称中的一个被显示，使用以下命令

```
# netstat -a --numeric-ports
```

```
# netstat -a --numeric-hosts
```

```
# netstat -a --numeric-users
```

12、持续输出 netstat 信息 #netstat -c

13、找出程序运行的端口 #netstat -ap | grep ':80'

14、查看连接某服务端口最多的IP地址（前20个）

```
#netstat -nat | grep "10.1.62.23:443" | awk '{print $5}' | awk -F: '{print $1}' | sort | uniq -c | sort -nr | head -20
```

15、TCP各种状态列表

```
#netstat -nat | awk '{print $6}'
```

统计数量

```
#netstat -nat | awk '{print $6}' | sort | uniq -c
```

排序

```
#netstat -nat | awk '{print $6}' | sort | uniq -c | sort -rn
```

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

16、直接统计tcp数量监听的数量

```
#netstat -ant | wc -l
```

17、netstat -nlp | grep 80 //查看所有80端口使用情况

18、netstat -an | grep 3306 //查看所有3306端口使用情况

<[参数说明](https://www.runoob.com/linux/linux-comm-netstat.html)><https://www.runoob.com/linux/linux-comm-netstat.html>



# tcpdump命令

2022年2月21日 15:42

## 简介

用简单的话来定义tcpdump，就是：dump the traffic on a network，根据使用者的定义对网络上的数据包进行截获的包分析工具。 tcpdump可以将网络中传送的数据包的“头”完全截获下来提供分析。它支持针对网络层、协议、主机、网络或端口的过滤，并提供and、or、not等逻辑语句来帮助你去掉无用的信息。

## 实用命令实例

### 默认启动

```
Tcpdump
```

普通情况下，直接启动tcpdump将监视第一个网络接口上所有流过的数据包。

### 监视指定网络接口的数据包

```
tcpdump -i eth1
```

如果不指定网卡，默认tcpdump只会监视第一个网络接口，一般是eth0，下面的例子都没有指定网络接口。

### 监视指定主机的数据包

打印所有进入或离开sundown的数据包.

```
tcpdump host sundown
```

也可以指定ip, 例如截获所有210. 27. 48. 1 的主机收到的和发出的所有的数据包

```
tcpdump host 210.27.48.1
```

打印helios 与 hot 或者与 ace 之间通信的数据包

```
tcpdump host helios and \( hot or ace \)
```

截获主机210. 27. 48. 1 和主机210. 27. 48. 2 或210. 27. 48. 3的通信

```
tcpdump host 210.27.48.1 and \( 210.27.48.2 or 210.27.48.3 \)
```

打印ace与任何其他主机之间通信的IP 数据包，但不包括与helios之间的数据包.

```
tcpdump ip host ace and not helios
```

如果想要获取主机210. 27. 48. 1除了和主机210. 27. 48. 2之外所有主机通信的ip包，使用命令：

```
tcpdump ip host 210.27.48.1 and ! 210.27.48.2
```

截获主机hostname发送的所有数据

```
tcpdump -i eth0 src host hostname
```

监视所有送到主机hostname的数据包

```
tcpdump -i eth0 dst host hostname
```

### 监视指定主机和端口的数据包

如果想要获取主机210.27.48.1接收或发出的telnet包，使用如下命令  
`tcpdump tcp port 23 and host 210.27.48.1`

对本机的udp 123 端口进行监视 123 为ntp的服务端口  
`tcpdump udp port 123`

## 监视指定网络的数据包

打印本地主机与Berkeley网络上的主机之间的所有通信数据包(nt: ucb-ether, 此处可理解为'Berkeley网络'的网络地址, 此表达式最原始的含义可表达为: 打印网络地址为ucb-ether的所有数据包)  
`tcpdump net ucb-ether`

打印所有通过网关snup的ftp数据包(注意, 表达式被单引号括起来了, 这可以防止shell对其中的括号进行错误解析)  
`tcpdump 'gateway snup and (port ftp or ftp-data)'`

打印所有源地址或目标地址是本地主机的IP数据包  
(如果本地网络通过网关连到了另一网络, 则另一网络并不能算作本地网络. (nt: 此句翻译曲折, 需补充). localnet 实际使用时要真正替换成本地网络的名字)  
`tcpdump ip and not net localnet`

## 监视指定协议的数据包

打印TCP会话中的的开始和结束数据包, 并且数据包的源或目的不是本地网络上的主机. (nt: localnet, 实际使用时要真正替换成本地网络的名字)  
`tcpdump 'tcp[tcpflags] & (tcp-syn|tcp-fin) != 0 and not src and dst net localnet'`

打印所有源或目的端口是80, 网络层协议为IPv4, 并且含有数据, 而不是SYN, FIN以及ACK-only等不含数据的数据包. (ipv6的版本的表达式可做练习)  
`tcpdump 'tcp port 80 and (((ip[2:2] - ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0)'`

(nt: 可理解为, ip[2:2]表示整个ip数据包的长度, (ip[0]&0xf)<<2表示ip数据包包头的长度(ip[0]&0xf代表包中的IHL域, 而此域的单位为32bit, 要换算成字节数需要乘以4, 即左移2. (tcp[12]&0xf0)>>4表示tcp头的长度, 此域的单位也是32bit, 换算成比特数为((tcp[12]&0xf0)>>4)<<2, 即((tcp[12]&0xf0)>>2). ((ip[2:2] - ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0 表示: 整个ip数据包的长度减去ip头的长度, 再减去tcp头的长度不为0, 这就意味着, ip数据包中确实是有数据. 对于ipv6版本只需考虑ipv6头中的'Payload Length'与'tcp头的长度'的差值, 并且其中表达方式'ip[]'需换成'ip6[]'.)

打印长度超过576字节, 并且网关地址是snup的IP数据包  
`tcpdump 'gateway snup and ip[2:2] > 576'`

打印所有IP层广播或多播的数据包, 但不是物理以太网层的广播或多播数据报  
`tcpdump 'ether[0] & 1 = 0 and ip[16] >= 224'`

打印除'echo request'或者'echo reply'类型以外的ICMP数据包(比如, 需要打印所有非ping 程序产生的数据包时可用到此表达式.  
(nt: 'echo request'与'echo reply'这两种类型的ICMP数据包通常由ping程序产生))  
`tcpdump 'icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echo-reply'`

## tcpdump 与wireshark

Wireshark(以前是ethereal)是Windows下非常简单易用的抓包工具。但在Linux下很难找到一个好用的图形化抓包工具。

还好有Tcpdump。我们可以用Tcpdump + Wireshark 的完美组合实现：在 Linux 里抓包，然后在

Windows 里分析包。

tcpdump tcp -i eth1 -t -s 0 -c 100 and dst port ! 22 and src net 192.168.1.0/24 -w ./target.cap

(1)tcp: ip icmp arp rarp 和 tcp、udp、icmp这些选项等都要放到第一个参数的位置，用来过滤数据报的类型

(2)-i eth1 : 只抓经过接口eth1的包

(3)-t : 不显示时间戳

(4)-s 0 : 抓取数据包时默认抓取长度为68字节。加上-s 0 后可以抓到完整的数据包

(5)-c 100 : 只抓取100个数据包

(6)dst port ! 22 : 不抓取目标端口是22的数据包

(7)src net 192.168.1.0/24 : 数据包的源网络地址为192.168.1.0/24

(8)-w ./target.cap : 保存成cap文件，方便用ethereal(即wireshark)分析

## 使用tcpdump抓取HTTP包

tcpdump -XvvennSs 0 -i eth0 tcp[20:2]=0x4745 or tcp[20:2]=0x4854  
0x4745 为“GET”前两个字母“GE”，0x4854 为“HTTP”前两个字母“HT”。

tcpdump 对截获的数据并没有进行彻底解码，数据包内的大部分内容是使用十六进制的形式直接打印输出的。显然这不利于分析网络故障，通常的解决办法是先使用带-w参数的tcpdump 截获数据并保存到文件中，然后再使用其他程序(如Wireshark)进行解码分析。当然也应该定义过滤规则，以避免捕获的数据包填满整个硬盘。

<包解析><https://www.cnblogs.com/ggjucheng/archive/2012/01/14/2322659.html>

<详细使用><https://www.cnblogs.com/ct20150811/p/9431976.html>

## 5Wireshark分析tcpdump抓包结果

1、启动8080端口，tcpdump抓包命令如下：

```
tcpdump -i lo -s 0 -n -S host 10.37.63.3 and port 8080 -w ./Desktop/tcpdump_10.37.63.3_8080_20160525.cap
```

# 然后再执行curl

```
curl http://10.37.63.3:8080/atbg/doc
```

2、使用Wireshark打开tcpdump\_10.37.63.3\_8080\_20160525.cap文件

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.37.63.3	10.37.63.3	TCP	60	58305 → 8080 [SYN] Seq=3892987956 Win=65535 Len=0 MSS=16344 WS=32 TSval=921727188
2	0.000004	10.37.63.3	10.37.63.3	TCP	60	8080 → 58305 [SYN, ACK] Seq=3569295623 Ack=3892987956 Win=65535 Len=0 MSS=16344 WS=32
3	0.000075	10.37.63.3	10.37.63.3	TCP	56	58305 → 8080 [ACK] Seq=3892987956 Ack=3569295623 Win=48288 Len=0 TSval=921727188
4	0.000082	10.37.63.3	10.37.63.3	TCP	56	8080 → 58305 [ACK] Seq=3569295623 Ack=3892987956 Win=48288 Len=0 TSval=921727188
5	0.000110	10.37.63.3	10.37.63.3	HTTP	143	GET /atbg/doc HTTP/1.1
6	0.000135	10.37.63.3	10.37.63.3	TCP	56	8080 → 58305 [ACK] Seq=3569295623 Ack=389298838 Win=48288 Len=0 TSval=921727188
7	0.000175	10.37.63.3	10.37.63.3	HTTP	6031	HTTP/1.1 200 OK (text/html)
8	0.001395	10.37.63.3	10.37.63.3	TCP	56	58305 → 8080 [ACK] Seq=389298838 Ack=3569301618 Win=48288 Len=0 TSval=921727188
9	0.001414	10.37.63.3	10.37.63.3	TCP	56	8080 → 58305 [FIN, ACK] Seq=389298838 Ack=3569301618 Win=48288 Len=0 TSval=921727188
10	0.004715	10.37.63.3	10.37.63.3	TCP	56	8080 → 58305 [ACK] Seq=3569301618 Ack=389298838 Win=48288 Len=0 TSval=921727188
11	0.004721	10.37.63.3	10.37.63.3	TCP	56	58305 → 8080 [ACK] Seq=389298838 Ack=3569301618 Win=48288 Len=0 TSval=921727188
12	0.004788	10.37.63.3	10.37.63.3	TCP	56	8080 → 58305 [FIN, ACK] Seq=3569301618 Ack=389298838 Win=48288 Len=0 TSval=921727188
13	0.004803	10.37.63.3	10.37.63.3	TCP	56	58305 → 8080 [ACK] Seq=389298838 Ack=3569301618 Win=48288 Len=0 TSval=921727188

No. 1-4 行：TCP三次握手环节；

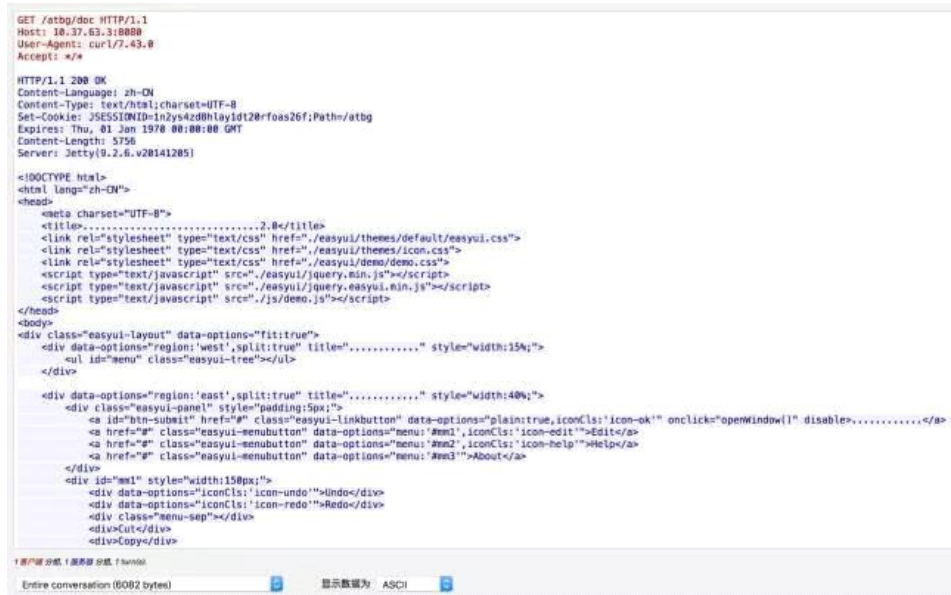
No. 5-8 行：TCP传输数据环节；

No. 9-13 行：TCP四次挥手环节；

3、顺便说一个查看 http 请求和响应的方法：



弹窗如下图所示，上面红色部分为请求信息，下面蓝色部分为响应信息：



1、抓本级服务器所有监听端口的包：

```
tcpdump -i any -X0 -vvv -s0 -w /root/`hostname`_anyport.cap
```

2、抓本机指定端口的包（例如抓6000监听端口的包）：

```
tcpdump -i any port 6000 -w /root/`hostname`_6000.cap
```



```
[root@localhost ~]# tcpdump -tttt
tcpdump: verbose output suppressed
listening on eth0, link-type EN10M
2014-08-08 06:34:32.594213 IP 192.
2014-08-08 06:34:32.594456 IP 192.
2014-08-08 06:34:32.659602 IP goog
2014-08-08 06:34:32.659686 IP 192.
```

3、抓本机多个指定端口的包（例如抓6000和6100监听端口的包）：

```
tcpdump -i any port 6000 or 6100 -w /root/`hostname`_6000_6100.cap
```

采用命令行方式对接口的数据包进行筛选抓取。

不带任何选项的tcpdump，默认抓取第一个网络接口。只有将tcpdump进程终止，抓包停止。

例子

(1) tcpdump -i eth0 host 192.168.1.1 【过滤主机】

注：抓取所有经过网卡1，目的ip为192.168.1.1

```
root@ubuntu:/home/peng# tcpdump -r tcpdump_test.log
reading from file tcpdump_test.log, link-type EN10MB (Ethernet)
00:52:46.461132 IP6 fe80::8abf:e4ff:fe5e:986a > fe80::20c:29ff:feb0:8013: ICMP6, neighbor
solicitation, who has fe80::20c:29ff:feb0:8013, length 32
00:52:46.461173 IP6 fe80::20c:29ff:feb0:8013 > fe80::8abf:e4ff:fe5e:986a: ICMP6, neighbor
advertisement, tgt is fe80::20c:29ff:feb0:8013, length 24
00:52:46.613994 IP6 fe80::20c:29ff:feb0:8013 > fe80::8abf:e4ff:fe5e:986a: ICMP6, neighbor
solicitation, who has fe80::8abf:e4ff:fe5e:986a, length 32
00:52:46.628554 IP6 fe80::8abf:e4ff:fe5e:986a > fe80::20c:29ff:feb0:8013: ICMP6, neighbor
advertisement, tgt is fe80::8abf:e4ff:fe5e:986a, length 24

00:52:47.180339 IP ubuntu.37884 > 192.168.43.48.domain: 857+ A? baidu.com. (27)
00:52:47.230918 IP 192.168.43.48.domain > ubuntu.37884: 857 2/0/0 A 39.156.69.79, A 220.1
1.38.148 (59)

00:52:47.231496 IP ubuntu > 39.156.69.79: ICMP echo request, id 4726, seq 1, length 64
00:52:47.292381 IP 39.156.69.79 > ubuntu: ICMP echo reply, id 4726, seq 1, length 64
00:52:47.292830 IP ubuntu.22247 > 192.168.43.48.domain: 36472+ PTR? 79.69.156.39.in-addr.
rpa. (43)
```

(2) tcpdump -i eth0 dst port 22 【过滤端口】

注：抓取所有经过网卡1，目的端口为22的网络数据

(3) tcpdump -i eth0 udp 【过滤协议】

注：抓取所有经过网卡1，协议为UDP的协议。

(4) tcpdump -i lo udp 【抓取本地环路数据包】

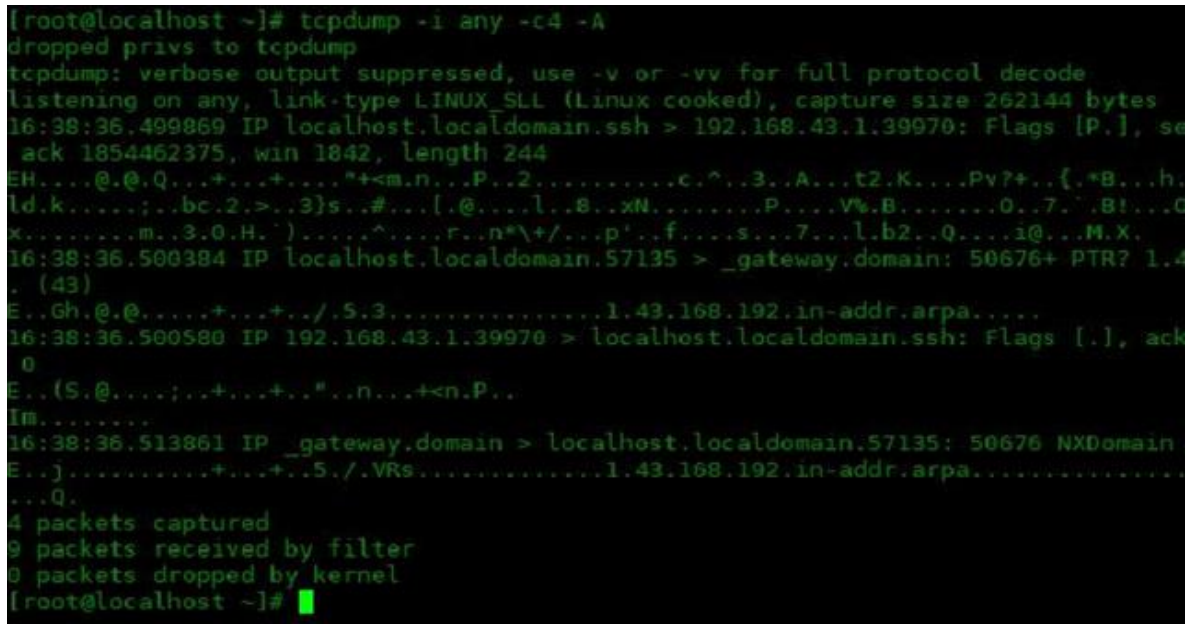
### (5) 特定协议特定端口

```
tcpdump udp port 22
```

### (6) 抓取特定类型的数据包

```
tcpdump -i eth0 'tcp[tcpflags] = tcp-syn' (抓取所有经过网卡1的SYN类型的数据包)
```

```
tcpdump -i eth0 udp dst port 53 (抓取经过网卡1的所有DNS数据包)
```



```
[root@localhost ~]# tcpdump -i any -c4 -A
dropped privs to tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
16:38:36.499869 IP localhost.localdomain.ssh > 192.168.43.1.39970: Flags [P.], seq
ack 1854462375, win 1842, length 244
EH....@.Q...+...+..."+<m.n...P..2.....c.^..3..A...t2.K....Pv7+...{.*B...h.
ld.k.....;..bc.2.>..3)s..#...[.@....l..8..xN.....P....v%.B.....0..7..!B!...0
x.....m..3.0.H..').....^....r..n*\+/...p'..f....s...7...l.b2..Q....i@...M.X.
16:38:36.500384 IP localhost.localdomain.57135 > _gateway.domain: 50676+ PTR? 1.4
. (43)
E..Gh.@.@.....+...+.../.5.3.....1.43.168.192.in-addr.arpa.....
16:38:36.500580 IP 192.168.43.1.39970 > localhost.localdomain.ssh: Flags [P.], ack
0
E..(S.@.....;...+...+...".n...+<n.P..
Im.....
16:38:36.513861 IP _gateway.domain > localhost.localdomain.57135: 50676 NXDomain
E..j.....+...+...5./..VRs.....1.43.168.192.in-addr.arpa.....
...Q.
4 packets captured
9 packets received by filter
0 packets dropped by kernel
[root@localhost ~]#
```

### (7) 逻辑语句过滤

```
tcpdump -i eth0 '((tcp) and ((dst net 192.168.0) and (not dst host 192.168.0.2)))'
```

注：抓取所有经过网卡1，目的网络是192.168.0，但目的主机不是192.168.0.2的TCP数据。

### (8) 抓包存取

```
tcpdump -i eth0 host 192.168.1.51 and port 22 -w /tmp/tcpdump.cap
```

注：抓取所有经过网卡1，目的主机为192.168.1.51的网络数据并存储。

# ps命令

2022年2月22日 11:12

## Linux ps 命令

Linux ps（英文全拼：process status）命令用于显示当前进程的状态，类似于 windows 的任务管理器。

### 语法

```
ps [options] [--help]
```

### 参数：

- ps 的参数非常多，在此仅列出几个常用的参数并大略介绍含义
- -A 列出所有的进程
- -w 显示加宽可以显示较多的资讯
- -au 显示较详细的资讯
- -aux 显示所有包含其他使用者的行程
- au(x) 输出格式：  
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
- USER: 行程拥有者
- PID: pid
- %CPU: 占用的 CPU 使用率
- %MEM: 占用的记忆体使用率
- VSZ: 占用的虚拟记忆体大小
- RSS: 占用的记忆体大小
- TTY: 终端的次要装置号码 (minor device number of tty)
- STAT: 该行程的状态:
- D: 无法中断的休眠状态 (通常 IO 的进程)
- R: 正在执行中
- S: 静止状态
- T: 暂停执行
- Z: 不存在但暂时无法消除
- W: 没有足够的记忆体分页可分配
- <: 高优先序的行程
- N: 低优先序的行程
- L: 有记忆体分页分配并锁在记忆体内 (实时系统或握A I/O)
- START: 行程开始时间
- TIME: 执行的时间
- COMMAND: 所执行的指令

### 实例

查找指定进程格式：

```
ps -ef | grep 进程关键字
```

例如显示 php 的进程：

```
# ps -ef | grep php
```

```
root      794      1  0  2020 ?        00:00:52 php-fpm: master process (/etc/php/7.3/fpm/php-fpm.conf)
www-data  951      794  0  2020 ?        00:24:15 php-fpm: pool www
www-data  953      794  0  2020 ?        00:24:14 php-fpm: pool www
www-data  954      794  0  2020 ?        00:24:29 php-fpm: pool www
...
```

显示进程信息：

```
# ps -A
```

PID	TTY	TIME	CMD
1	?	00:00:02	init
2	?	00:00:00	kthreadd
3	?	00:00:00	migration/0
4	?	00:00:00	ksoftirqd/0
5	?	00:00:00	watchdog/0
6	?	00:00:00	events/0
7	?	00:00:00	cpuset
8	?	00:00:00	khelper
9	?	00:00:00	netns
10	?	00:00:00	async/mgr
11	?	00:00:00	pm
12	?	00:00:00	sync_supers
13	?	00:00:00	bdi-default
14	?	00:00:00	kintegrityd/0
15	?	00:00:02	kblockd/0
16	?	00:00:00	kacpid
17	?	00:00:00	kacpi_notify
18	?	00:00:00	kacpi_hotplug
19	?	00:00:27	ata/0

.....省略部分结果

30749	pts/0	00:00:15	gedit
30886	?	00:01:10	qtcreator.bin
30894	?	00:00:00	qtcreator.bin
31160	?	00:00:00	dhclient
31211	?	00:00:00	aptd
31302	?	00:00:00	sshd
31374	pts/2	00:00:00	bash
31396	pts/2	00:00:00	ps

显示指定用户信息

```
# ps -u root //显示root进程用户信息
```

PID	TTY	TIME	CMD
1	?	00:00:02	init
2	?	00:00:00	kthreadd
3	?	00:00:00	migration/0
4	?	00:00:00	ksoftirqd/0
5	?	00:00:00	watchdog/0
6	?	00:00:00	events/0
7	?	00:00:00	cpuset
8	?	00:00:00	khelper
9	?	00:00:00	netns
10	?	00:00:00	async/mgr
11	?	00:00:00	pm
12	?	00:00:00	sync_supers
13	?	00:00:00	bdi-default
14	?	00:00:00	kintegrityd/0
15	?	00:00:02	kblockd/0
16	?	00:00:00	kacpid

.....省略部分结果

30487	?	00:00:06	gnome-terminal
30488	?	00:00:00	gnome-pty-helpe
30489	pts/0	00:00:00	bash
30670	?	00:00:00	debconf-communi
30749	pts/0	00:00:15	gedit
30886	?	00:01:10	qtcreator.bin
30894	?	00:00:00	qtcreator.bin
31160	?	00:00:00	dhclient
31211	?	00:00:00	aptd
31302	?	00:00:00	sshd
31374	pts/2	00:00:00	bash
31397	pts/2	00:00:00	ps

显示所有进程信息，连同命令行

```
# ps -ef //显示所有命令，连带命令行
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
-----	-----	------	---	-------	-----	------	-----



```
root    1    0 0 10:22 ?    00:00:02 /sbin/init
root    2    0 0 10:22 ?    00:00:00 [kthreadd]
root    3    2 0 10:22 ?    00:00:00 [migration/0]
root    4    2 0 10:22 ?    00:00:00 [ksoftirqd/0]
root    5    2 0 10:22 ?    00:00:00 [watchdog/0]
root    6    2 0 10:22 ?    /usr/lib/NetworkManager
.....省略部分结果
root    31302 2095 0 17:42 ?    00:00:00 sshd: root@pts/2
root    31374 31302 0 17:42 pts/2 00:00:00 -bash
root    31400 1 0 17:46 ?    00:00:00 /usr/bin/python /usr/sbin/aptd
root    31407 31374 0 17:48 pts/2 00:00:00 ps -ef
```

# tail命令

2022年2月22日 11:18

## Linux tail 命令

tail 命令可用于查看文件的内容，有一个常用的参数 **-f** 常用于查阅正在改变的日志文件。  
**tail -f filename** 会把 filename 文件里的最尾部的内容显示在屏幕上，并且不断刷新，只要 filename 更新就可以看到最新的文件内容。

### 命令格式:

```
tail [参数] [文件]
```

### 参数:

- -f 循环读取
- -q 不显示处理信息
- -v 显示详细的处理信息
- -c<数目> 显示的字节数
- -n<行数> 显示文件的尾部 n 行内容
- --pid=PID 与-f合用, 表示在进程ID, PID死掉之后结束
- -q, --quiet, --silent 从不输出给出文件名的首部
- -s, --sleep-interval=S 与-f合用, 表示在每次反复的间隔休眠S秒

### 实例

要显示 notes.log 文件的最后 10 行，请输入以下命令：

```
tail notes.log # 默认显示最后 10 行
```

要跟踪名为 notes.log 的文件的增长情况，请输入以下命令：

```
tail -f notes.log
```

此命令显示 notes.log 文件的最后 10 行。当将某些行添加至 notes.log 文件时，tail 命令会继续显示这些行。显示一直继续，直到您按下 (Ctrl-C) 组合键停止显示。

显示文件 notes.log 的内容，从第 20 行至文件末尾：

```
tail -n +20 notes.log
```

显示文件 notes.log 的最后 10 个字符：

```
tail -c 10 notes.log
```

# cat命令

2022年2月22日 11:24

## Linux cat 命令

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。

### 使用权限

所有使用者

### 语法格式

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

### 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，就替换为一行的空白行。
- 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；

### 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。

# grep命令

2022年2月22日 11:42

## Linux grep 命令

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行显示出来。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。

### 语法：

```
grep [-abcEFGhHilLnqrsVwxy] [-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 : 不要忽略二进制的文件。
- -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行之外，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。

### 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:hellLinux!
testfile1:Linis a free Unix-type operating system.
testfile1:Lin
testfile_1:HELLO LINUX!
testfile_1:LINUX IS A FREE UNIX-TYPE OPERATING SYSTEM.
testfile_1:THIS IS A LINUX TESTFILE!
testfile_2:HELLO LINUX!
testfile_2:Linux is a free unix-type operating system.
```

# sed命令

2022年2月22日 11:43

## Linux sed 命令

Linux sed 命令是利用脚本来处理文本文件。

sed 可依照脚本的指令来处理、编辑文本文件。

Sed 主要用来自动编辑一个或多个文件、简化对文件的反复操作、编写转换程序等。

### 语法：

```
sed [-hnV] [-e<script>] [-f<script文件>] [文本文件]
```

### 参数说明：

- -e<script>或--expression=<script> 以选项中指定的script来处理输入的文本文件。
- -f<script文件>或--file=<script文件> 以选项中指定的script文件来处理输入的文本文件。
- -h或--help 显示帮助。
- -n或--quiet或--silent 仅显示script处理后的结果。
- -V或--version 显示版本信息。

### 动作说明：

- a：新增，a 的后面可以接字符串，而这些字符串会在新的一行出现（目前的下一行）～
- c：取代，c 的后面可以接字符串，这些字符串可以取代 n1, n2 之间的行！
- d：删除，因为是删除啊，所以 d 后面通常不接任何东西；
- i：插入，i 的后面可以接字符串，而这些字符串会在新的一行出现（目前的上一行）；
- p：打印，亦即将某个选择的数据印出。通常 p 会与参数 sed -n 一起运行～
- s：取代，可以直接进行取代的工作哩！通常这个 s 的动作可以搭配正规表示法！例如 1,20s/old/new/g 就是啦！

### 实例

在testfile文件的第四行后添加一行，并将结果输出到标准输出，在命令行提示符下输入如下命令：

```
sed -e 4a\newLine testfile
```

首先查看testfile中的内容如下：

```
$ cat testfile #查看testfile 中的内容
```

```
HELLO LINUX!
```

```
Linux is a free unix-type operating system.
```

```
This is a linux testfile!
```

```
Linux test
```

使用sed命令后，输出结果如下：

```
$ sed -e 4a\newLine testfile #使用sed 在第四行后添加新字符串
```

```
HELLO LINUX! #testfile文件原有的内容
```

```
Linux is a free unix-type operating system.
```

```
This is a linux testfile!
```

```
Linux test
```

```
newline
```

### 以行为单位的新增/删除

将 /etc/passwd 的内容列出并且列印行号，同时，请将第 2~5 行删除！

```
[root@www ~]# nl /etc/passwd | sed '2,5d'
```

```
1 root:x:0:0:root:/root:/bin/bash
```

```
6 sync:x:5:0:sync:/sbin:/bin/sync
7 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
..... (后面省略).....
```

sed 的动作为 '2,5d'，那个 d 就是删除！因为 2-5 行给他删除了，所以显示的数据就没有 2-5 行罗  
 ~ 另外，注意一下，原本应该是要下达 sed -e 才对，没有 -e 也行啦！同时也要注意的，sed 后面接的动作，请务必以 ' ' 两个单引号括住喔！

只要删除第 2 行

```
nl /etc/passwd | sed '2d'
```

要删除第 3 到最后一行

```
nl /etc/passwd | sed '3,$d'
```

在第二行后 (亦即是加在第三行) 加上『drink tea?』字样！

```
[root@www ~]# nl /etc/passwd | sed '2a drink tea'
```

```
1 root:x:0:0:root:/root:/bin/bash
```

```
2 bin:x:1:1:bin:/bin:/sbin/nologin
```

```
drink tea
```

```
3 daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

```
..... (后面省略).....
```

那如果是要在第二行前

```
nl /etc/passwd | sed '2i drink tea'
```

如果是要增加两行以上，在第二行后面加入两行字，例如 Drink tea or ..... 与 drink beer?

```
[root@www ~]# nl /etc/passwd | sed '2a Drink tea or ..... \
```

```
> drink beer ?'
```

```
1 root:x:0:0:root:/root:/bin/bash
```

```
2 bin:x:1:1:bin:/bin:/sbin/nologin
```

```
Drink tea or .....
```

```
drink beer ?
```

```
3 daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

```
..... (后面省略).....
```

每一行之间都必须要以反斜杠『 \ 』来进行新行的添加喔！所以，上面的例子中，我们可以发现在第一行的最后面就有 \ 存在。

## 以行为单位的替换与显示

将第2-5行的内容取代成为『No 2-5 number』呢？

```
[root@www ~]# nl /etc/passwd | sed '2,5c No 2-5 number'
```

```
1 root:x:0:0:root:/root:/bin/bash
```

```
No 2-5 number
```

```
6 sync:x:5:0:sync:/sbin:/bin/sync
```

```
..... (后面省略).....
```

透过这个方法我们就能够将数据整行取代了！

仅列出 /etc/passwd 文件内的第 5-7 行

```
[root@www ~]# nl /etc/passwd | sed -n '5,7p'
```

```
5 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

```
6 sync:x:5:0:sync:/sbin:/bin/sync
```

```
7 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
```

可以透过这个 sed 的以行为单位的显示功能，就能够将某一个文件内的某些行号选择出来显示。

## 数据的搜寻并显示

搜索 /etc/passwd 有 root 关键字的行

```
nl /etc/passwd | sed '/root/p'
```

```
1 root:x:0:0:root:/root:/bin/bash
```

```
1 root:x:0:0:root:/root:/bin/bash
```

```
2 daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

```
3 bin:x:2:2:bin:/bin:/bin/sh
```

```
4 sys:x:3:3:sys:/dev:/bin/sh
```

```
5 sync:x:4:65534:sync:/bin:/bin/sync
```

```
.... 下面忽略
```

如果 root 找到，除了输出所有行，还会输出匹配行。

使用 -n 的时候将只打印包含模板的行。

```
nl /etc/passwd | sed -n '/root/p'
```

```
1 root:x:0:0:root:/root:/bin/bash
```

## 数据的搜寻并删除

删除/etc/passwd所有包含root的行，其他行输出

```
nl /etc/passwd | sed '/root/d'
2 daemon:x:1:1:daemon:/usr/sbin:/bin/sh
3 bin:x:2:2:bin:/bin:/bin/sh
.... 下面忽略
#第一行的匹配root已经删除了
```

## 数据的搜寻并执行命令

搜索/etc/passwd, 找到root对应的行，执行后面花括号中的一组命令，每个命令之间用分号分隔，这里把bash替换为blueshell，再输出这行：

```
nl /etc/passwd | sed -n '/root/{s/bash/blueshell/;p;q}'
1 root:x:0:0:root:/root:/bin/blueshell
最后的q是退出。
```

## 数据的搜寻并替换

除了整行的处理模式之外，sed 还可以用行为单位进行部分数据的搜寻并取代。基本上 sed 的搜寻与替代的与 vi 相当的类似！他有点像这样：

sed 's/要被取代的字串/新的字串/g'

先观察原始信息，利用 /sbin/ifconfig 查询 IP

```
[root@www ~]# /sbin/ifconfig eth0
eth0 Link encap:Ethernet HWaddr 00:90:CC:A6:34:84
inet addr:192.168.1.100 Bcast:192.168.1.255 Mask:255.255.255.0
inet6 addr: fe80::290:ccff:fea6:3484/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
..... (以下省略).....
```

本机的ip是192.168.1.100。

将 IP 前面的部分予以删除

```
[root@www ~]# /sbin/ifconfig eth0 | grep 'inet addr' | sed 's/^.*addr://g'
192.168.1.100 Bcast:192.168.1.255 Mask:255.255.255.0
```

接下来则是删除后续的部分，亦即： 192.168.1.100 Bcast:192.168.1.255 Mask:255.255.255.0

将 IP 后面的部分予以删除

```
[root@www ~]# /sbin/ifconfig eth0 | grep 'inet addr' | sed 's/^.*addr://g' | sed 's/Bcast.*$//g'
192.168.1.100
```

## 多点编辑

一条sed命令，删除/etc/passwd第三行到末尾的数据，并把bash替换为blueshell

```
nl /etc/passwd | sed -e '3,$d' -e 's/bash/blueshell/'
1 root:x:0:0:root:/root:/bin/blueshell
2 daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

-e表示多点编辑，第一个编辑命令删除/etc/passwd第三行到末尾的数据，第二条命令搜索bash替换为blueshell。

## 直接修改文件内容(危险动作)

sed 可以直接修改文件的内容，不必使用管道命令或数据流重导向！不过，由於这个动作会直接修改到原始的文件，所以请你千万不要随便拿系统配置来测试！我们还是使用文件 regular\_express.txt 文件来测试看看吧！

regular\_express.txt 文件内容如下：

```
[root@www ~]# cat regular_express.txt
runoob.
google.
taobao.
facebook.
zhihu-
weibo-
```



利用 sed 将 regular\_express.txt 内每一行结尾若为 . 则换成 !

```
[root@www ~]# sed -i 's/\.$/\!/g' regular_express.txt
[root@www ~]# cat regular_express.txt
runoob!
google!
taobao!
facebook!
zhihu-
weibo-
```

:q;q

利用 sed 直接在 regular\_express.txt 最后一行加入 # This is a test:

```
[root@www ~]# sed -i '$a # This is a test' regular_express.txt
[root@www ~]# cat regular_express.txt
runoob!
google!
taobao!
facebook!
zhihu-
weibo-
```

# This is a test

由於 \$ 代表的是最后一行，而 a 的动作是新增，因此该文件最后新增 # This is a test!

sed 的 -i 选项可以直接修改文件内容，这功能非常有帮助！举例来说，如果你有一个 100 万行的文件，你要在第 100 行加某些文字，此时使用 vim 可能会疯掉！因为文件太大了！那怎办？就利用 sed 啊！透过 sed 直接修改/取代的功能，你甚至不需要使用 vim 去修订！

# awk命令

2022年2月22日 11:44

## Linux awk 命令

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。

## 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 --assign 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-interval  
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。

## 基本用法

log.txt文本内容如下:

```
2 this is a test
3 Are 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 Are 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 Are 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
```

## 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 Are 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
```

# 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 $
```

## 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 Are you like awk
10 There are orange, apple, mongo
```

## 忽略大小写

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

```
-----
2 this is a test
This's a test
```

## 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
```

## 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
```

## 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")}'
```