

散尽浮华

重剑无锋，大巧不工！

博客园 首页 新随笔 联系 管理 订阅

随笔- 563 文章- 39 评论- 879

昵称: 散尽浮华  
园龄: 4年8个月  
粉丝: 2968  
关注: 23  
+加关注

Grep 高效用法实战总结 - 运维笔记

**grep** (global search regular expression(RE) and print out the line, 全面搜索正则表达式并把行打印出来) 是一种强大的文本搜索工具，能使用正则表达式搜索文本，并把匹配的行打印出来。**grep** 主要作用是过滤出指定的行，指定的行满足什么条件，满足的条件可配合正则表达式来表示，实现强大的文本处理。

**文本处理工具分类**  
常用的有：**grep**，**egrep**，**fgrep**。

**三者区别**  
**grep**：在没有参数的情况下，只输出符合RE（Regular Expression）字符。  
**egrep**：等同于**grep -E**，和**grep**最大的区别就是表现在转义符上，比如**grep** 做次数匹配时`\{n,m\}`，**egrep**则不需要直接`{n, m}`。**egrep**显得更方便简洁。  
**fgrep**：等同于**grep -f**，但是不能使用正则表达式。所有的字符匹配功能均已消失。

**grep格式**  
`grep [option] pattern filename`

grep的option选项说明

1 grep的option选项说明:

2 --color 显示颜色的参数，即搜索出来的关键字符带有颜色。使用"grep --color 关键字符" 或

3 -a 不要忽略二进制数据。使用"grep -a 关键字符"

4 -A 显示符合关键字符的行，以及其后面的n行。使用 "grep -An 关键字符" 或者 "grep -A n

5 -b 显示符合关键字符的行，以及其前后的各n行。使用"grep -bn 关键字符"，注意这个不能使用

6 -B 显示符合关键字符的行，以及其前面的n行。使用 "grep -An 关键字符" 或者 "grep -A n

7 -c 只输出匹配行的计数。即只显示出匹配关键字符的那行的行数，不显示内容！使用"grep -c 关键字符"

8 -C 显示符合关键字符的行，以及其前后的各n行。使用 "grep -Cn 关键字符" 或者 "grep -C n

9 -d 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命令将回报信息并停止动作。

10 -e 指定字符串作为查找文件内容的关键字符。使用"grep -e 关键字符"。grep -e "正则表达式"

11 -E 将关键字符为延伸的普通表示法来使用，意味着使用能使用扩展正则表达式！！通常用于满足多个条件。

12 -f 显示两个文件中相同的行。使用 "grep -f filename1 filename2 " 或者 "grep --file filename1 filename2"

13 -F 将关键字符视为固定字符串的列表。使用"grep -F 关键字符 filename1" 或者 "grep -F 关键字符 filename2"

14 -G 将关键字符视为普通的表示法来使用。

15 -h 对多文件搜索关键字符时不显示文件名，只显示关键字符。使用"grep -h 关键字符 filename1 filename2"

16 -H 对多文件搜索关键字符时显示文件名和关键字符，跟-h参数相反。使用"grep -H 关键字符 filename1 filename2"

17 -i 忽略关键字符的大小写。(跟-y参数相同)。使用"grep -i 关键字符 filename1 filename2"

18 -l 对多文件搜索关键字符时只显示文件名。使用"grep -l 关键字符 filename1 filename2 filename3"

19 -L 对多文件搜索关键字符时，只显示不匹配关键字符的文件名。使用"grep -L 关键字符 filename1 filename2 filename3"

20 -n 显示匹配关键字符的行号和行内容。使用"grep -n 关键字符 filename1 filename2 filename3"

21 -q 不显示任何信息。用于if逻辑判断,安静模式，不打印任何标准输出。如果有匹配的内容则立即返回0。

22 -R/-r 此参数的效果和指定"-d recurse"参数相同。

23 -s 当搜索关键字符，匹配的文件不存在时不显示错误信息。即不显示不存在或无匹配文本的错误信息。

24 -v 反转或过滤搜索，即过滤出来那些不匹配关键字符的行。使用"grep -v 关键字符 filename1 filename2 filename3"

25 -w 精准搜索关键字符，即只显示完全匹配关键字符的行，不显示那些包含关键字符的行。使用"grep -w 关键字符 filename1 filename2 filename3"

aws

12个月免费套餐

新用户注册立享  
11月限时专享 \$500 或 ¥2888 抵扣券

立即查看

< 2020年11月 >

日	一	二	三	四	五	六
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5
6	7	8	9	10	11	12

搜索

找找看

谷歌搜索

常用链接

我的随笔  
我的评论  
我的参与  
最新评论  
我的标签

随笔分类

Ansible(5)  
Apache(6)  
Ceph(4)  
ClusterShell(1)

- 26 -x 只显示整行都是关键字的行。使用 "grep -x 关键字"。
- 27 -y 忽略关键字的大小写。(跟-i参数相同)。
- 28 -o 只输出文件中匹配到的部分，不会打印多余的内容。
- 29 -P 使用perl的正则表达式语法，因为perl的正则更加多元化，能实现更加复杂的场景。典型用法

grep正则表达式元字符集（基本集）

- 1 ^  
2 匹配行的开始 如： '^grep' 匹配所有以grep开头的行。
- 3
- 4 \$  
5 匹配行的结束 如： 'grep\$' 匹配所有以grep结尾的行。
- 6
- 7 .  
8 匹配一个非换行符的字符 如： 'gr.p' 匹配gr后接一个任意字符，然后是p。
- 9
- 10 \*  
11 匹配零个或多个先前字符 如： '\*grep' 匹配所有有一个或多个空格后紧跟grep的行。 .\*一起用代表任
- 12
- 13 []  
14 匹配一个指定范围内的字符，如 '[Gg]rep' 匹配Grep和grep。即[mn]表示匹配m或者n关键字，相当
- 15
- 16 [^]  
17 匹配一个不在指定范围内的字符，如： '[^A-FH-Z]rep' 匹配不包含A-F和H-Z的一个字母，但是包含
- 18
- 19 \(. \. \)  
20 标记匹配字符，如 '\(love\)'，love被标记为1。
- 21
- 22 \  
23 匹配单词的开始，如： '\
- 24
- 25 \>  
26 匹配单词的结束，如 'str\>' 匹配包含以str结尾的单词的行。通常使用 "\<关键字\>" 作为精准匹
- 27
- 28 x\{m\}  
29 重复字符x，m次，如： '0\{5\}' 匹配包含5个o的行。
- 30
- 31 x\{m,\}  
32 重复字符x,至少m次，如： 'o\{5,\}' 匹配至少有5个o的行。
- 33
- 34 x\{m,n\}  
35 重复字符x，至少m次，不多于n次，如： 'o\{5,10\}' 匹配5--10个o的行。
- 36
- 37 \w  
38 匹配文字和数字字符，也就是[A-Za-z0-9]，如： 'Gw\*p' 匹配以G后跟零个或多个文字或数字字符，
- 39
- 40 \W  
41 \w的反置形式，匹配一个或多个非单词字符，如句号号等。
- 42
- 43 \b  
44 单词锁定符，如： \byang\b 表示只匹配yang。相当于grep -w "yang" 或者 grep "\<yang\>"
- 45
- 46 +  
47 匹配一个或多个先前的字符。如： '[a-z]+able'，匹配一个或多个小写字母后跟able的串，如love.
- 48
- 49 ?  
50 匹配零个或一个先前的字符。如： '(gr)?p' 匹配gr后跟一个或没有字符，然后是p的行。注意：先前
- 51
- 52 a|b|c

- DNS(5)
- Docker(36)
- DRBD(3)
- Elasticsearch(7)
- Expect(2)
- Fabric(1)
- FastDFS(1)
- FTP(4)
- GlusterFS (5)
- Haproxy(6)
- IP SAN(1)
- Iptables(6)
- Jenkins(9)
- Jira and Confluence(7)
- Jumpserver(4)
- Kafka(2)
- LB/HA高可用(23)
- LDAP(2)
- LVM(3)
- LVS(5)
- Maven/Nexus(1)
- Memcached(4)
- MongoDB(11)
- MooseFS(2)
- MQ消息队列(5)
- MySQL(65)
- NFS(2)
- Nginx(50)
- PHP(10)
- Puppet(3)
- Python(13)
- Redis(16)
- Rsync(8)
- Saltstack(4)
- Samba(3)
- Shell(36)
- Squid(4)
- SSH(7)
- Supervisor/Monit(3)
- Tomcat(11)
- Ubuntu(9)
- Varnish(1)
- VPN(7)
- Zookeeper(3)
- 安全性能(29)
- 版本控制(32)
- 常规运维(90)
- 监控系统(43)
- 日志分析(8)
- 虚拟化(30)
- 邮件服务(3)

随笔档案

- 2020年11月(6)
- 2020年4月(1)
- 2020年3月(1)
- 2019年11月(2)
- 2019年10月(3)
- 2019年9月(1)
- 2019年8月(4)
- 2019年7月(4)
- 2019年6月(1)
- 2019年4月(2)
- 2019年3月(6)
- 2019年2月(6)
- 2019年1月(6)
- 2018年12月(9)
- 2018年11月(7)

```
53 匹配a或b或c。如grep -E "a|b|c"，匹配a或b或c中的任意一个都可以。grep -v "a\b|c" 或
54
55 ()
56 分组符号，如：love(able|rs)ov+匹配loveable或lovers，匹配一个或多个ov。
57
58
59 =====
60 几个小示例：
61 [root@ss-server ~]# cat sys.list
62 AFG
63 AFS
64 BAU
65 CdC
66 000823
67 ERU
68 jNNNA
69 IFI
70 entegor
71 jKL
72 NNN
73 MYSQL
74 QQQ
75 UUU
76 CCS
77 ONLINE
78 MPB
79 IFF
80
81 如下，"A.."表示以A开头，后面跟两个单个字符。"."表示任意单个字符。
82 [root@ss-server ~]# cat sys.list |grep "A.."
83 AFG
84 AFS
85
86 重复N字母，至少重复了3次。
87 [root@ss-server ~]# cat sys.list|grep "N\{3\}"
88 jNNNA
89 NNN
90
91 如果匹配正则表达式，效果如下：
92 [root@ss-server ~]# cat sys.list|grep "^[A-Z]\{6\}" #至少连续出现6次大写的字母
93 ONLINE
94 [root@ss-server ~]# cat sys.list|grep "^[A-Z]\{6,6\}"
95 ONLINE
96 [root@ss-server ~]# cat sys.list|grep "^[A-Z]\{5\}"
97 MYSQL
98 ONLINE
99 [root@ss-server ~]# cat sys.list|grep "^[A-Z]\{5,6\}"
100 MYSQL
101 ONLINE
102 [root@ss-server ~]# cat sys.list|grep "^[A-Z]\{3\}"
103 AFG
104 AFS
105 BAU
106 ERU
107 IFI
108 NNN
109 MYSQL
110 QQQ
111 UUU
```

2018年10月(10)  
2018年9月(9)  
2018年8月(12)  
2018年7月(11)  
2018年6月(2)  
2018年5月(12)  
2018年4月(11)  
2018年3月(9)  
2018年2月(12)  
2018年1月(21)  
2017年12月(17)  
2017年11月(12)  
2017年10月(6)  
2017年9月(10)  
2017年8月(7)  
2017年7月(5)  
2017年6月(6)  
2017年5月(8)  
2017年4月(10)  
2017年3月(11)  
2017年2月(15)  
2017年1月(36)  
2016年12月(41)  
2016年11月(34)  
2016年10月(30)  
2016年9月(35)  
2016年8月(35)  
2016年7月(37)  
2016年6月(37)  
2016年3月(3)

Linux加油站

- Linux命令大全
- Git基础教程
- Prometheus中文手册
- Zabbix监控配置教程
- Centos的epel源下载
- Nginx官方文档
- Mysql源码下载
- Python自动化运维-案例源码
- 阿里开源镜像
- Docker镜像-hub.docker
- Python自动化运维之路
- Docker基础学习
- Redis命令参考
- Django基础教程
- RUNOOB.COM - 编程学习
- Elasticsearch 中文社区
- Tengine参考文档
- HTTP Status Codes
- Linux监控专注
- Puppet版本下载
- Open-falcon社区文档
- 每天一个Linux命令
- Python学习- 推荐网站
- 攻防安全指南
- Squid中文权威指南
- Ceph开源社区
- Linux运维日记本
- AWK使用手册
- Docker学习汇总
- Kubernetes 中文社区
- Technology blog1 of learning
- 精通Python自动化脚本
- Shell传递参数
- Rancher 部署文档
- Kubernetes 部署手册
- ElasticSearch性能监控指标

```
112 CCS
113 ONLINE
114 MPB
115 IFF
116 [root@ss-server ~]# cat sys.list|grep "^[A-Z]\{3,3\}"
117 AFG
118 AFS
119 BAU
120 ERU
121 IFI
122 NNN
123 MYSQL
124 QQQ
125 UUU
126 CCS
127 ONLINE
128 MPB
129 IFF
```

**POSIX字符类**（注意使用的时候，外面要套一层中括号才能生效）

为了在不同国家的字符编码中保持一致，POSIX(The Portable Operating System Interface)增加了特殊的字符类，如[:alnum:]是A-Za-z0-9的另一个写法。要把它们放到[]号内才能成为正则表达式，如[A-Za-z0-9]或[:alnum:]]。在linux下的grep除fgrep外，都支持POSIX的字符类。

```
1 [:alnum:]
2 文字数字字符。使用 grep [[:alnum:]] filename 表示打印filename文件中包括数字和字母（大
3
4 [:alpha:]
5 文字字符。使用 grep [[:alpha:]] filename 表示打印filename文件中包括字母（大小写字母）
6
7 [:digit:]
8 数字字符。使用 grep [[:digit:]] filename 表示打印filename文件中包括数字的行。
9
10 [:graph:]
11 非空字符（非空格、控制字符）
12
13 [:lower:]
14 小写字符。使用 grep [[:lower:]] filename 表示打印filename文件中包括小写字母的行。
15
16 [:cntrl:]
17 控制字符
18
19 [:print:]
20 非空字符（包括空格）
21
22 [:punct:]
23 标点符号
24
25 [:space:]
26 所有空白字符（新行，空格，制表符）。例如使用 sed -i 's/[[:space:]]//g' filename 表示
27
28 [:upper:]
29 大写字符。使用 grep [[:upper:]] filename 表示打印filename文件中包括大写字母的行。
30
31 [:xdigit:]
32 十六进制数字（0-9，a-f，A-F）
```

**grep引号使用问题**

```
1 单引号：
```

[Go语言学习速查手册](#)  
[Go语言中文网](#)  
[运维派](#)  
[优雅姿态监控Kubernetes](#)  
[K8S TLS bootstrapping](#)  
[Prometheus 操作指南](#)  
[Supervisor教程](#)  
[Prometheus 高可用](#)  
[Ansible Tower 实战](#)  
[Kubernetes 中文文档](#)  
[Kubernetes 学习指南](#)  
[Elasticsearch 权威指南](#)  
[Technology blog2 of learning](#)  
[Harbor 使用手册](#)  
[Kubernetes 使用手册](#)  
[Rancher 中文手册](#)  
[Rancher 中文文档](#)  
[kubernetes 最佳实践](#)  
[Istio 中文手册](#)

最新评论

1. [Re:Harbor镜像仓库（含clair镜像扫描） - 完整部署记录](#)  
您终于回来了，好想您  
--紫色飞猪
2. [Re:web cache server方案比较: varnish、squid、nginx](#)  
这么好的文章没人顶？  
--假的程序员
3. [Re:Elasticsearch 最佳运维实践 - 总结（一）](#)  
为啥第一篇没有评论呐。受益匪浅。谢了  
--Jackie、杰
4. [Re:无法直接在国内网络环境下从k8s.gcr.io下载镜像问题](#)  
kubeadm init --pod-network-cidr=xx.xx --image-repository registry.aliyuncs.com/google\_containers...  
--金枪语
5. [Re:Linux系统用户密码规则 - 运维总结](#)  
5次更改密码不能有重复，这个改了配置了，但是去测试，还是可以使用原来的密码？为啥呢？  
--海口-熟练工
6. [Re:挂载银行前置机Ukey到windows server2012虚拟机的操作记录](#)  
你好，请教您一下，一个kvm虚拟机可以挂载多个usb吗？  
我这边尝试的一直都报 no free usb prots  
--daquan1
7. [Re:日志切割方法小结 \[Logrotate、python、shell脚本实现\]](#)  
@豌豆花下猫 可以...
8. [Re:MySQL 半同步复制模式说明及配置示例 - 运维小结](#)  
要想不丢数据，首先要保证log buffer落盘成功，就得innodb\_flush\_log\_at\_trx\_commit=1代表完全同步log buffer->os buffer+fsync()落盘。...  
--只会一点java
9. [Re:CentOS6.9下升级默认的OpenSSH操作记录（升级到OpenSSH\\_7.6p1）](#)  
你好，大神。参照你的方法用systemctl restart sshd服务后一直卡着到最后报错。journalctl -xe报错信息中有“Registered Authentication Agent...”  
--顶梁柱Plus
10. [Re:MySQL 主从同步\(1\) - 概念和原理介绍 以及 主从/主主模式 部署记录](#)  
你好，测试了一下主从复制，发现一个问题，使用命令行插入数据从库不同步，但是使用可视化工具插入数据，从

```
2  即将单引号中内容原样输出，也就是单引号''是全引用。
3
4  双引号：
5  如果双引号的内容中有命令、变量等，会先把变量、命令解析成结果，再将结果输出。双引号""是部
6
7  单双引号：
8  常量用单引号''括起，而含有变量则用双引号""括起。单双可同时出现，单扩住双。
9
10 "" 号里面遇到$, \等特殊字符会进行相应的变量替换
11 '' 号里面的所有字符都保持原样
12
13 对于字符串，双引号和单引号两者相同，匹配模式也大致相同，但有一些区别非常容易混淆，例如：
14 grep "$a" file      #引用变量a,查找变量a的值。双引号识别变量。
15 grep 'a' file       #查找"a"字符串。单引号不识别变量。
16
17 grep "\" file      #grep: Trailing backslash (不知原因)
18 grep \' file      #查找\'字符
19
20 $  美元符
21 \  反斜杠
22 `  反引号
23 "  双引号
24 这四个字符在双引号中是具有特殊含义的，其他都没有，而单引号使所有字符都失去特殊含义！
25
26 如果用双引号，查找一个\，就应该用四个\\：
27 grep "\\\" file 这样就对了，这样等同于：
28 grep \' file
29
30 第一条命令shell把四个\,转义成2个\传递给grep，grep再把2个\转义成一个\查找；
31 第二条命令shell没转义，直接把2个\传递给grep，grep再把2个\转义成一个\查找；
32 其实grep执行的是相同的命令。
```

库就可以成功，不清楚这两者的区别，需要博主告知。  
--戴口罩的少侠

## 阅读排行榜

1. Git忽略提交规则 - .gitignore配置运维总结(461332)
2. ELK实时日志分析平台环境部署--完整记录(179948)
3. 完整部署CentOS7.2+OpenStack+kvm 云平台环境 (1) --基础环境搭建(138421)
4. MySQL 数据库误删除后的数据恢复操作说明(136203)
5. Linux终端复用神器-Tmux使用梳理(132888)
6. 执行git push出现"Everything up-to-date"(113817)
7. Mysql连接错误：Lost connection to Mysql server at 'waiting for initial communication packet'(83990)
8. MySQL 之binlog日志说明及利用binlog日志恢复数据操作记录(78811)
9. Gitlab利用Webhook实现Push代码后的jenkins自动构建(76108)
10. Linux下环境变量配置方法梳理 (.bash\_profile和.bashrc的区别) (75831)

## 评论排行榜

1. 完整部署CentOS7.2+OpenStack+kvm 云平台环境 (1) --基础环境搭建(126)
2. ELK实时日志分析平台环境部署--完整记录(24)
3. Gitlab利用Webhook实现Push代码后的jenkins自动构建(22)
4. [原创]CI持续集成系统环境--Gitlab+Gerrit+Jenkins完整对接(21)
5. kvm虚拟化管理平台WebVirtMgr部署-完整记录(1)(19)

## 推荐排行榜

1. Git忽略提交规则 - .gitignore配置运维总结(42)
2. ELK实时日志分析平台环境部署--完整记录(35)
3. 完整部署CentOS7.2+OpenStack+kvm 云平台环境 (1) --基础环境搭建(24)
4. Redis哨兵模式 (sentinel) 学习总结及部署记录 (主从复制、读写分离、主从切换) (22)
5. linux负载均衡总结性说明 (四层负载/七层负载) (18)

### grep -E 与 grep -P区别 [ 正则中的 ?= 、 ?<= 、 ?! 、 ?<! ]

```
1  grep -E 主要是用来支持扩展正则表达式，比如 | 符号，用于grep多条件查询，并非是使用标准i
2  grep -P 主要让grep使用perl的正则表达式语法，因为perl的正则更加多元化，能实现更加复杂的i
3
4  示例如下：
5  打印test.file文件中含有2018 或 2019 或 2020字符串的行
6  [root@localhost ~]# grep -E "2018|2019|2020" test.file
7
8  如下想在一句话"Hello, my name is kevin" 中匹配中间的一段字符串 "my name is"，可以这样
9  [root@localhost ~]# echo "Hello, my name is kevin"
10 Hello, my name is kevin
11 [root@localhost ~]# echo "Hello, my name is kevin"|grep -P '(?<=Hello)'
12 Hello, my name is kevin
13 [root@localhost ~]# echo "Hello, my name is kevin"|grep -P '(?<=Hello).*(?= kevin'
14 Hello, my name is kevin
15 [root@localhost ~]# echo "Hello, my name is kevin"|grep -Po '(?<=Hello).*(?= kevin'
16 , my name is
17 [root@localhost ~]# echo "Hello, my name is kevin"|grep -P '(?<=Hello, ).*(?= kevin'
18 Hello, my name is kevin
19 [root@localhost ~]# echo "Hello, my name is kevin"|grep -Po '(?<=Hello, ).*(?= kevin'
20 my name is
21
22 这里千万注意下：正则中的 ?= 、 ?<= 、 ?! 、 ?<! 四个表达的意思。结合grep使用的时候，一定要
23 ?= 表示询问后面跟着的东西是否等于这个。
24 ?<= 表示询问是否以这个东西开头。
25 ?! 表示询问后面跟着的东西是否不是这个。
```

```

26  ?<! 表示询问是否不是以这个东西开头 。
27
28  示例: grep -P后面跟上面四种正则时, 要在引号里使用小括号进行匹配。
29  [root@localhost ~]# cat test.txt
30  bac
31  ab
32  bb
33  bch
34  ban
35  [root@localhost ~]# grep -P 'b(?=a)' test.txt
36  bac
37  ban
38  [root@localhost ~]# grep -P '(?<=a)' test.txt
39  bac
40  ab
41  ban
42  [root@localhost ~]# grep -P '(?<=a)b' test.txt
43  ab
44  [root@localhost ~]# grep -P 'b(?!a)' test.txt
45  ab
46  bb
47  bch
48  [root@localhost ~]# grep -P '(?!a)b' test.txt
49  bac
50  bb
51  bch
52  ban
53
54  按照上面的思路, 如果想要取得本机ip地址, 可以如下操作:
55  [root@localhost ~]# ifconfig eth0
56  eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
57      inet 172.16.60.232  netmask 255.255.255.0  broadcast 172.16.60.255
58      inet6 fe80::e825:b3ff:fef6:1398  prefixlen 64  scopeid 0x20<link>
59      ether ea:25:b3:f6:13:98  txqueuelen 1000  (Ethernet)
60      RX packets 22911237  bytes 4001968461 (3.7 GiB)
61      RX errors 0  dropped 17058008  overruns 0  frame 0
62      TX packets 670762  bytes 98567533 (94.0 MiB)
63      TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
64
65  [root@localhost ~]# ifconfig eth0|grep -P "(?<=inet).*(?=netmask)"
66      inet 172.16.60.232  netmask 255.255.255.0  broadcast 172.16.60.255
67  [root@localhost ~]# ifconfig eth0|grep -Po "(?<=inet).*(?=netmask)"
68      172.16.60.232

```

### grep 常用操作技巧

1) 在文件中搜索一个单词, 命令会返回一个包含"match\_pattern"的文本行:

```
[root@test ~]# grep match_pattern file_name
```

2) 在多个文件中查找:

```
[root@test ~]# grep "match_pattern" file_1 file_2 file_3 ...
```

3) 输出除之外的所有行 -v 选项:

```
[root@test ~]# grep -v "match_pattern" file_name
```

4) 标记匹配颜色 --color=auto 选项:

```
[root@test ~]# grep "match_pattern" file_name --color=auto
```

5) 使用正则表达式 -E 选项:

```
[root@test ~]# grep -E "[1-9]+"
```

或

```
[root@test ~]# egrep "[1-9]+"
```

6) 只输出文件中匹配到的部分 -o 选项:

```
[root@test ~]# echo this is a test line. | grep -o -E "[a-z]+\."
```

```
line.
```



```
[root@test ~]# echo this is a test line. | egrep -o "[a-z]+\."
```

line.

7) 统计文件或者文本中包含匹配字符串的行数 -c 选项:

```
[root@test ~]# grep -c "text" file_name
```

8) 输出包含匹配字符串的行数 -n 选项:

```
[root@test ~]# grep "text" -n file_name
```

或

```
[root@test ~]# cat file_name | grep "text" -n
```

9) 多个文件

```
[root@test ~]# grep "text" -n file_1 file_2
```

10) 打印样式匹配所位于的字符或字节偏移:

```
[root@test ~]# echo gun is not unix | grep -b -o "not"
```

```
7:not
```

#一行中字符串的字符便宜是从该行的第一个字符开始计算, 起始值为0。选项 -b -o 一般总是配合使用。

11) 搜索多个文件并查找匹配文本在哪些文件中:

```
[root@test ~]# grep -l "text" file1 file2 file3...
```

### grep递归搜索文件

12) 在多级目录中对文本进行递归搜索:

```
[root@test ~]# grep "text" . -r -n
```

.表示当前目录。

13) 忽略匹配样式中的字符大小写:

```
[root@test ~]# echo "hello world" | grep -i "HELLO"
```

```
hello
```

14) 选项 -e 制动多个匹配样式:

```
[root@test ~]# echo this is a text line | grep -e "is" -e "line" -o
```

```
is
```

```
is
```

```
line
```

15) 也可以使用-f选项来匹配多个样式, 在样式文件中逐行写出需要匹配的字符。

```
[root@test ~]# cat patfile
```

```
aaa
```

```
bbb
```

```
[root@test ~]# echo aaa bbb ccc ddd eee | grep -f patfile -o
```

### 在grep搜索结果中包括或者排除指定文件

16) 只在目录中所有的.php和.html文件中递归搜索字符"main()"

```
[root@test ~]# grep "main()" . -r --include *.{php,html}
```

17) 在搜索结果中排除所有README文件

```
[root@test ~]# grep "main()" . -r --exclude "README"
```

18) 在搜索结果中排除filelist文件列表里的文件

```
[root@test ~]# grep "main()" . -r --exclude-from filelist
```

19) 使用0值字节后缀的grep与xargs:

#测试文件:

```
[root@test ~]# echo "aaa" > file1
```

```
[root@test ~]# echo "bbb" > file2
```

```
[root@test ~]# echo "aaa" > file3
```

```
[root@test ~]# grep "aaa" file* -lZ | xargs -0 rm
```

20) 执行后会删除file1和file3, grep输出用-Z选项来指定以0值字节作为终结符文件名 (\0), xargs

-0 读取输入并用0值字节终结符分隔文件名, 然后删除匹配文件, -Z通常和-l结合使用。**"grep -q"用于if逻辑判断, 特别好用! -q 参数意为安静模式, 不打印任何标准输出。如果有匹配的内容则立即返回状态值0。**

grep -q 静默输出:

```
[root@test ~]# grep -q "test" filename
```

不会输出任何信息, 如果命令运行成功返回0, 失败则返回非0值。一般用于if条件测试。

```
1  如下是某个脚本中使用的"grep -q"安静模式
2  #Deploy_Env参数有下划线则对Pkg_Env和Deploy_Env重新赋值
3  if `echo ${Deploy_Env}|grep -q  "_"`;then
4      Pkg_Env=`echo ${Deploy_Env} | awk -F'_' '{print $2}`
5      Deploy_Env=`echo ${Deploy_Env} | awk -F'_' '{print $1}`
6  fi
7
8  再如下一例
9  [root@test ~]# if $(echo "wang_shibo"|grep -q  "_");then kevin=`echo "wang_shibo"
```

**打印出匹配文本之前或者之后的行**

21) 显示匹配某个结果之后的3行, 使用 -A 选项:

```
[root@test ~]# seq 10 | grep "5" -A 3
```

```
5
6
7
8
```

22) 显示匹配某个结果之前的3行, 使用 -B 选项:

```
[root@test ~]# seq 10 | grep "5" -B 3
```

```
2
3
4
5
```

23) 显示匹配某个结果的前三行和后三行, 使用 -C 选项:

```
[root@test ~]# seq 10 | grep "5" -C 3
```

```
2
3
4
5
6
7
8
```

24) 如果匹配结果有多个, 即多重匹配的话, 中间会用"--"作为各匹配结果之间的分隔符:

```
[root@test ~]# echo -e "a\nb\nc\na\nb\nc" | grep a -A 1
```

```
a
b
--
a
b
```

```
[root@test ~]# grep "match_pattern" file_name
```

**grep取文件中每行的前两个字符/前2字节**

```
1  示例:
2  [root@localhost tmp]# cat test
3  00c3dd43f15eafab3b0db4bdaabb3f6d91c2f9a3b88e044ddf83393dd910eb9b
4  fb0772b142fe0d214c0ccfa9c7b8ed13277c35e9fce75ba1c151dd878dcda95c
5  d7c3167c320d7a820935f54cf4290890ea19567da496ecf774e8773b35d5f065
6  131f805ec7fd68d45a887e2ef82de61de0247b4eb934ab03b7c933650e854baa
7  322ed380e680a77f30528ba013e3a802a7b44948a0609c7d1d732dd46a9a310d
8  6ac240b130982ad1c3ba3188abbf18ba4e54bdd9e504ce2d5c2eff6d3e86b8dd
9  af40529340df51a38ca319c40e6b718f5952990b335aa703967520809061f677
10 a1bbb86f4147e007af7d3f7e987cf5ed3d33912bf430d8ab3528b1953225cca6
11 10ebb929a9def601aa26150b2dad6eae47ad07ac399e25ace3a5c1395fb2794
12
13 [root@localhost tmp]# cat /tmp/test|grep -o "^.\{0,2\}"
14 00
15 fb
16 d7
17 13
18 32
19 6a
20 af
21 a1
22 10
```

**grep同时筛选多个条件**

```
1  grep使用-E参数实现"满足多个条件中的任意一个", 如下:
2  1) 满足任意条件 (word1、word2和word3之一) 将匹配。
3  # grep -E "word1|word2|word3" file.txt
4
```



```

5  grep要想实现"同时满足多个条件", 需要执行多个grep过滤命令, 如下:
6  2) 必须同时满足三个条件 (word1、word2和word3) 才匹配。
7  # grep word1 file.txt | grep word2 |grep word3
8  # cat file.txt|grep word1 | grep word2 |grep word3
9
10 简单总结下grep常用过滤命令:
11 过滤的内容可以是一个词组等, 需要用引号包裹
12 1) 获取文件中的关键字key: # cat fileName | grep "key" #即获取fileName文件中的key
13 2) 获取文件中的某个关键字key1, key2, key3: # cat fileName | grep -E "key1|key2|key3"
14 3) 获取文件中的多个关键字, 同时满足: # cat fileName | grep key1 | grep key2| grep key3
15 4) 忽略文件中的某个关键字, 需要转义"|": # cat fileName | grep -v "key1\|key2\|key3"
16 4) 忽略文件中的多个关键字, : # cat fileName | grep -v "key1" | grep -v "key2" | grep -v "key3"
17
18 grep常用的还有:
19 grep -i: 表示不区分大小写
20 grep -w: 精准匹配 [或者使用单字边界"\<\>"或锁定单词"\b\b",即grep -w "str" 相当于 grep -w -e "str"]
21 grep -An: 获取匹配关键字所在行的后n行
22 grep -Bn: 获取匹配关键字所在行的前n行
23 grep -Cn: 获取匹配关键字所在行的前后各n行
24
25 另外注意:
26 grep -v "条件1\|条件2\|条件3" #需要加转义符
27 egrep -v "条件1|条件2|条件3" #不需要加转义符
28 =====
29 示例如下:
30 1) 排除test.txt文件中的haha、hehe字符
31 # cat test.txt |grep -v "haha\|hehe"
32
33 2) 删除/opt/data目录下创建时间是2017年 或 2018年的文件 ("ls -l"命令结果中的第8列是文件)
34 [root@localhost ~]# ll
35 total 15996
36 -rw-r--r-- 1 root root 781301 Nov 11 2017 a.sql
37 -rw-r--r-- 1 root root 460189 Nov 11 11:20 ncc_20180909.log
38 -rw-r--r-- 1 root root 112055 Nov 11 23:09 data.txt
39 -rw-r--r-- 1 root root 730029 Nov 11 2018 gate.tar.gz
40 .....
41
42 [root@localhost ~]# ls -l|grep -E -w "2017|2018"|xargs rm -rf

```

## grep精确匹配关键字

使用grep搜索某个关键字时, 默认搜索出来的是所有包含该关键字的行, 如下:

搜索/var/named/veredholdings.cn\_zone文件中172.16.50.24所在的行, 默认会把所有包含172.16.50.24所在的行打印出来。

```
[root@uatdns01 ~]# cat /var/named/veredholdings.cn_zone|grep 172.16.50.24
```

```

devzl-app01      IN      A      172.16.50.243
devzl-app02      IN      A      172.16.50.244
devzl-redis01    IN      A      172.16.50.245
devzl-redis02    IN      A      172.16.50.246
devzl-redis03    IN      A      172.16.50.247
devzl-oracle01   IN      A      172.16.50.242
wiki02           IN      A      172.16.50.24

```

```
[root@uatdns01 ~]# cat /var/named/veredholdings.cn_zone|grep 172.16.50.24 --color
```

```

devzl-app01      IN      A      172.16.50.243
devzl-app02      IN      A      172.16.50.244
devzl-redis01    IN      A      172.16.50.245
devzl-redis02    IN      A      172.16.50.246
devzl-redis03    IN      A      172.16.50.247
devzl-oracle01   IN      A      172.16.50.242
wiki02           IN      A      172.16.50.24

```

```
[root@uatdns01 ~]# cat /var/named/veredholdings.cn_zone|grep -o 172.16.50.24
172.16.50.24
172.16.50.24
172.16.50.24
172.16.50.24
172.16.50.24
172.16.50.24
172.16.50.24
```

要想精确地搜索出文件中某个单词所在的行，而不是打印所有包括该单词字样的行，可以使用grep -w 参数

**-w (--word-regexp) : 表示强制PATTERN仅完全匹配字词**

```
[root@uatdns01 ~]# cat /var/named/veredholdings.cn_zone|grep -w 172.16.50.24
wiki02 IN A 172.16.50.24
```

或者使用 \<\>单字边界也可以实现精确匹配（注意两边要加上双引号）

```
[root@uatdns01 named]# cat /var/named/veredholdings.cn_zone|grep "\<172.16.50.24\>"
wiki02 IN A 172.16.50.24
```

或者使用单词锁定符\b也可以实现精确匹配

```
[root@uatdns01 named]# cat /var/named/veredholdings.cn_zone|grep "\b172.16.50.24\b"
wiki02 IN A 172.16.50.24
```

### 两个小面试题

1) 精确地找出名为abc的进程名。

```
# ps -ef|grep -w "abc"
```

或者

```
# ps -ef|grep "\<abc\>"
```

或者

```
# ps -ef|grep "\babc\b"
```

2) 判断该进程的数量是否在3-5之间。

```
# ps -ef|grep -w abc|wc -l
```

或者

```
# ps -ef|grep "\<abc\>"|wc -l
```

[[ 需要注意 ]]: grep 使用 -w 或 \<\> 或 \b 进行精准匹配时，对于@, - 特殊字符是过滤不掉的，下划线\_ 字符可以过滤掉。

示例如下：

```

[root@localhost ~]# cat test.txt
finhub-crystalvar
finhub-crystal
finhub-mms-crt2000
finhub-crystal-2018
finhub-crystal_20180202
finhub-crystal@kevin
finhub-crystalcore-2018
finhub-cmp-201803
finhub-app-j2u89_20
[root@localhost ~]# cat test.txt|grep finhub-crystal
finhub-crystalvar
finhub-crystal
finhub-crystal-2018
finhub-crystal_20180202
finhub-crystal@kevin
finhub-crystalcore-2018
[root@localhost ~]# cat test.txt|grep -w finhub-crystal
finhub-crystal
finhub-crystal-2018
finhub-crystal@kevin
[root@localhost ~]# cat test.txt|grep '\<finhub-crystal\>'
finhub-crystal
finhub-crystal-2018
finhub-crystal@kevin
[root@localhost ~]# cat test.txt|grep '\bfinhub-crystal\b'
finhub-crystal
finhub-crystal-2018
finhub-crystal@kevin

```

### 小示例1

```

1  写一个shell脚本，检查服务器上的main进程在不在
2
3  [root@two002 tmp]# ps -ef|grep main
4  root      23448 23422  0 11:40 pts/0    00:00:00 grep --color=auto main
5  [root@two002 tmp]# ps -ef|grep main|grep -v grep|wc -l
6  0
7
8  [root@two002 tmp]# cat /tmp/main_check.sh
9  #!/bin/bash
10 NUM=$(ps -ef|grep main|grep -v grep|wc -l)
11 if [ $NUM -eq 0 ];then
12     echo "It's not good! main is stoped!"
13 else
14     echo "Don't worry! main is running!"
15 fi
16
17
18 执行检查脚本
19 [root@two002 tmp]# sh -x /tmp/main_check.sh
20 ++ grep main
21 ++ grep -v grep
22 ++ wc -l
23 ++ ps -ef
24 + NUM=2
25 + '[' 2 -eq 0 ']'
26 + echo 'Don't worry! main is running!'
27 Don't worry! main is running!
28

```

```

29 [root@two002 tmp]# sh /tmp/main_check.sh
30 Don't worry! main is running!
31
32
33 如上发现，执行脚本/tmp/main_check.sh的过程中，看到NUM参数值是2!
34 但是手动执行ps -ef|grep main|grep -v grep|wc -l的结果明明是0! !
35
36 这是由于grep匹配的问题，需要grep进行精准匹配，即"grep -w"
37 这就需要将main_check.sh脚本内容修改如下：
38 [root@two002 tmp]# cat /tmp/main_check.sh
39 #!/bin/bash
40 NUM=$(ps -ef|grep -w main|grep -v grep|wc -l)
41 if [ $NUM -eq 0 ];then
42     echo "Oh!My God! It's broken! main is stoped!"
43 else
44     echo "Don't worry! main is running!"
45 fi
46
47
48 再次执行检查脚本，就正确了
49 [root@two002 tmp]# sh -x /tmp/main_check.sh
50 ++ grep -w main
51 ++ grep -v grep
52 ++ wc -l
53 ++ ps -ef
54 + NUM=0
55 + '[' 0 -eq 0 ']'
56 + echo 'Oh!My God! It\'\'s broken! main is stoped!'
57 Oh!My God! It's broken! main is stoped!
58
59 [root@two002 tmp]# sh /tmp/main_check.sh
60 Oh!My God! It's broken! main is stoped!

```

## 小示例2

```

1 [root@localhost ABG]# ll /root/app/script/ansible/config/ABG/*.cfg|head -10
2 -rw-rw-rw- 1 root root 176 Aug 22 14:26 /root/app/script/ansible/config/ABG/absT:
3 -rw-rw-rw- 1 bxi bxi 435 Jun 11 2019 /root/app/script/ansible/config/ABG/accoi
4 -rw-rw-rw- 1 root root 234 Sep 4 16:11 /root/app/script/ansible/config/ABG/accoi
5 -rw-rw-rw- 1 bxi bxi 276 Aug 7 15:21 /root/app/script/ansible/config/ABG/adap:
6 -rw-rw-rw- 1 bxi bxi 359 Oct 29 14:47 /root/app/script/ansible/config/ABG/adap:
7 -rw-rw-rw- 1 root root 191 Dec 11 15:28 /root/app/script/ansible/config/ABG/aibai
8 -rw-rw-rw- 1 root root 218 Jun 11 2019 /root/app/script/ansible/config/ABG/aicai
9 -rw-rw-rw- 1 bxi bxi 261 Jun 11 2019 /root/app/script/ansible/config/ABG/aicai
10 -rw-rw-rw- 1 bxi bxi 288 Jun 11 2019 /root/app/script/ansible/config/ABG/aicai
11 -rw-rw-rw- 1 root root 177 Nov 14 09:59 /root/app/script/ansible/config/ABG/aica:
12
13 查看/root/app/script/ansible/config/ABG目录下的cfg结尾的配置文件中是否由带_A的配置，
14 [root@localhost ABG]# cat /root/app/script/ansible/config/ABG/mir-x-fund.cfg
15 [mir-x-fund_F]
16 172.16.60.20
17
18 [mir-x-fund_A]
19 172.16.60.22
20
21 [mir-x-fund:children]
22 mir-x-fund_F
23 mir-x-fund_A
24
25 [mir-x-fund:vars]

```

```

26 deploy_path=/opt/ABG/mir-x-fund/
27 start_time_out=90
28 stop_time_out=60
29 module=mir-x-fund
30
31 [root@localhost ABG]# cat /root/app/script/ansible/config/ABG/mir-x-fund.cfg|grep
32 [root@localhost ABG]# cat /root/app/script/ansible/config/ABG/mir-x-fund.cfg|grep
33 [mir-x-fund_A]
34 mir-x-fund_A
35
36 脚本如下（主要用到grep -w ".*_A"）：
37 [root@localhost ABG]# cat /root/ABG_A_file.sh
38 #!/bin/bash
39 for file in $(ls /root/app/script/ansible/config/ABG/*.cfg)
40 do
41     cat ${file}|grep -w ".*_A" >/dev/null 2>&1
42     if [ $? -ne 0 ];then
43         echo -e "ABG的$(echo ${file})|awk -F"/" '{print $NF}'|awk -F".cfg" '{print $
44     fi
45 done
46
47 执行脚本（配置文件是"应用模块.cfg"，ABG系统下的应用模块）：
48 [root@localhost ABG]# sh /root/ABG_A_file.sh
49 ABG的accounting_springBoot没有A环境的配置。
50 配置文件为/root/app/script/ansible/config/ABG/accounting_springBoot.cfg
51
52 ABG的aibank-service没有A环境的配置。
53 配置文件为/root/app/script/ansible/config/ABG/aibank-service.cfg
54
55 ABG的aireader-service没有A环境的配置。
56 配置文件为/root/app/script/ansible/config/ABG/aireader-service.cfg
57
58 ABG的backend-aiprogram-gateway没有A环境的配置。
59 配置文件为/root/app/script/ansible/config/ABG/backend-aiprogram-gateway.cfg
60
61 ABG的backend-transation没有A环境的配置。
62 配置文件为/root/app/script/ansible/config/ABG/backend-transation.cfg
63
64 ABG的backend-visitorguide没有A环境的配置。
65 配置文件为/root/app/script/ansible/config/ABG/backend-visitorguide.cfg
66
67 ABG的代码没有A环境的配置。
68 配置文件为/root/app/script/ansible/config/ABG/code.cfg

```

## grep 高效搜索用法大全

```

1 下面以/opt/aa.txt文件搜索为例
2 [root@ss-server ~]# cat /opt/aa.txt
3 beijing
4 beihai
5 this is test
6 you are good
7
8 通过管道过滤ls -l输出的内容，只显示以a开头的行。
9 [root@ss-server ~]# ls -l | grep '^a'
10
11 显示所有以d开头的文件中包含test的行。
12 [root@ss-server ~]# grep 'test' d*
13

```

显示在aa, bb, cc文件中匹配test的行。

```
[root@ss-server ~]# grep 'test' aa bb cc
```

显示aa文件中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。

```
[root@ss-server ~]# grep '[a-z]\{5\}' aa
```

能够使用-o仅仅打印匹配的字符

```
[root@ss-server ~]# echo this is line. |grep -o "[a-z]*.$"
```

```
line.
```

```
[root@ss-server ~]# echo this is line. |grep -o "[a-z]*\."
```

```
line.
```

```
[root@ss-server ~]# grep -o "bei" /opt/aa.txt
```

```
bei
```

```
bei
```

打印除匹配行之外的其它行，使用-v:

```
[root@ss-server ~]# echo -e "1\n2\n3\n4"|grep -v "1\|2\|3"
```

```
4
```

```
[root@ss-server ~]# grep -v "beihai\|this is test" /opt/aa.txt
```

```
beijing
```

```
you are good
```

统计匹配字符串的行数。使用-c

```
[root@ss-server ~]# echo -e "1\n2\n3\n4"|grep -v "1\|2" -c
```

```
2
```

统计字符串模式匹配的次数。能够结合-o。

```
[root@ss-server ~]# echo "beijing is good"|grep -o "bei"
```

```
bei
```

```
[root@ss-server ~]# echo "beijing is good"|grep -o "i"
```

```
i
```

```
i
```

```
i
```

假设须要显示行号，能够打开-n

```
[root@ss-server ~]# cat /opt/aa.txt |grep -o "bei" -n
```

```
1:bei
```

```
2:bei
```

-b选项能够打印出匹配的字符串相对于其所在的行起始位置的偏移量（从0开始）。通常配合-o使用:

```
[root@ss-server ~]# echo "012333456789" | grep -b -o 4
```

```
6:4
```

```
[root@ss-server ~]# echo "beijing ai ni"|grep -o "ai" -b
```

```
8:ai
```

```
[root@ss-server ~]# echo "beijing ai ni"|grep -o "jing" -b
```

```
3:jing
```

-P参数（声明grep后面要用的是perl的正则表达式）（\d+ 一个或多个数字）

```
[root@ss-server ~]# echo -e "\nline.123\nline."|grep -P "[a-z]*\.\d+"
```

```
line.123
```

```
[root@ss-server ~]# echo -e "\nline.123\nline."|grep -P "[a-z]*\."
```

```
line.123
```

```
line.
```

```
[root@ss-server ~]# echo -e "\nline.123\nline."|grep -P "[a-z]*\.$"
```

```
line.
```

```
[root@ss-server ~]# echo -e "\nline.123\nline."|grep -P "\.$"
```

```
line.
```

匹配多个字符串模式（grep -e 或者 grep -E）:



```
73 没有-o参数不会只打印匹配项
74 添加-o参数之后只打印匹配项
75 [root@ss-server ~]# cat /opt/aa.txt |grep -e "bei" -e "jing" -e "test"
76 beijing
77 beihai
78 this is test
79 [root@ss-server ~]# cat /opt/aa.txt |grep -e "bei" -e "jing" -e "test" -o
80 bei
81 jing
82 bei
83 test
84
85 [root@ss-server ~]# grep -E "bei|jing|test" /opt/aa.txt
86 beijing
87 beihai
88 this is test
89 [root@ss-server ~]# grep -E "bei|jing|test" -o /opt/aa.txt
90 bei
91 jing
92 bei
93 test
94
95 参数oP一起使用，会单独打印出要匹配的数字
96 [root@ss-server ~]# echo office365 | grep -oP "\d+"
97 365
98 [root@ss-server ~]# echo office365 | grep -oP "\d*"
99 365
100 [root@ss-server ~]# echo office365 | grep -oP "[0-9]*"
101 365
102 [root@ss-server ~]# echo 365beijing23 | grep -oP "\d+"
103 365
104 23
105 [root@ss-server ~]# echo 365beijing23 | grep -oP "\d*"
106 365
107 23
108 [root@ss-server ~]# echo 365beijing23 | grep -oP "[0-9]*"
109 365
110 23
111
112 只有参数-P，会完整显示匹配内容的一行，匹配内容高亮显示
113 [root@ss-server ~]# echo office365 | grep -P "\d+"
114 office365
115
116 只有参数-o，不会匹配任何内容，因为没有声明grep要使用正则表达式
117 如果单独使用参数-o，后面匹配的要是一个具体的字符串，不是正则
118 [root@ss-server ~]# echo office365 | grep -o "\d+"
119 [root@ss-server ~]#
120 [root@ss-server ~]# echo office365 | grep -o "365"
121 365
122
123 -Z选项在输出匹配文件名称时将以\0结尾配合xargs -0能够发挥非常多作用。
124 如下，将当前路径下包括wang字符串的文件全部删除
125 [root@ss-server ~]# echo "wang" >>11.txt
126 [root@ss-server ~]# echo "wangbo" >>12.txt
127 [root@ss-server ~]# echo "wanghu" >>13.txt
128 [root@ss-server ~]# echo "liru" >>14.txt
129 [root@ss-server ~]# grep -lZ "wang" *
130 11.txt 12.txt 13.txt
131 [root@ss-server ~]# grep -lZ "wang" *|xargs -0 rm
```

```

132 [root@ss-server ~]# ls
133 14.txt
134
135 限定全字匹配选项:-w, 即精准匹配, 三种方法: -w参数, "\<\>"单字边界, "\b\b"
136 [root@ss-server ~]# echo "linux" >> aa.list
137 [root@ss-server ~]# echo "li" >> bb.list
138 [root@ss-server ~]# grep -rn "li" *.list
139 aa.list:1:linux
140 bb.list:1:li
141 [root@ss-server ~]# grep -w "li" *.list
142 bb.list:li
143 [root@ss-server ~]# grep "\<li\>" *.list
144 bb.list:li
145 [root@ss-server ~]# grep "\bli\b" *.list
146 bb.list:li
147
148 "grep -E" 和 egrep 效果等同
149 [root@ss-server ~]# cat /etc/passwd|grep -E "linan|bobo"
150 linan:x:1000:1000::/home/linan:/bin/bash
151 bobo:x:1002:1002::/home/bobo:/bin/bash
152
153 [root@ss-server ~]# cat /etc/passwd|egrep "linan|bobo"
154 linan:x:1000:1000::/home/linan:/bin/bash
155 bobo:x:1002:1002::/home/bobo:/bin/bash
156
157 "grep -e"参数同样是匹配多个关键字
158 [root@ss-server ~]# cat /etc/passwd|grep -e "linan" -e "bobo"
159 linan:x:1000:1000::/home/linan:/bin/bash
160 bobo:x:1002:1002::/home/bobo:/bin/bash
161
162 #####
163 #####
164 grep -q : 安静模式, 不打印任何标准输出。用于if逻辑判断, 如果有匹配的内容则立即返回状态值
165 [root@localhost ~]# cat test.txt
166 17:41:54 up 67 days, 5 min, 2 users, load average: 0.00, 0.01, 0.05
167 [root@localhost ~]# grep -q "days" test.txt
168 [root@localhost ~]# echo $?
169 0
170 [root@localhost ~]# grep -q "days111" test.txt
171 [root@localhost ~]# echo $?
172 1
173
174 [root@localhost ~]# grep -q "days" test.txt && echo ok
175 ok
176 [root@localhost ~]# grep -q "days111" test.txt && echo ok
177 [root@localhost ~]# grep -q "days" test.txt || echo ok
178 [root@localhost ~]# grep -q "days111" test.txt || echo ok
179 ok
180
181 "grep -i" 或 "grep -y" 不区分大小写
182 [root@localhost ~]# echo -e "a\nA\nb\nc"|grep -i "a"
183 a
184 A
185 [root@localhost ~]# echo -e "a\nA\nb\nc"|grep -y "a"
186 a
187 A
188
189 grep使用正则关键匹配
190 [root@localhost ~]# cat test.txt

```

```

191 abc
192 adb123
193 abcde
194 abcdefg
195 [root@localhost ~]# cat test.txt|grep -e '[a-z]\{4\}'
196 abcde
197 abcdefg
198 [root@localhost ~]# cat test.txt|grep -e '[a-z]\{5\}'
199 abcde
200 abcdefg
201 [root@localhost ~]# cat test.txt|grep -e '[a-z]\{6\}'
202 abcdefg
203
204 grep查找以ab开头的行（使用^）
205 [root@localhost ~]# grep "^ab" test.txt
206
207 grep查找以de结尾的行（使用$）
208 [root@localhost ~]# grep "de$" test.txt
209
210 grep查找空行（使用^$）；查找空行及其行号
211 [root@localhost ~]# grep "^$" test.txt
212 [root@localhost ~]# grep -n "^$" test.txt
213
214 #####
215 #####
216 grep利用[]搜索集合字符
217 [] 表示其中的某一个字符，例如[ade] 表示a或d或e
218 [root@localhost ~]# cat test.txt
219 abc
220 adb8877
221 123456
222 [root@localhost ~]# cat test.txt|grep [8]
223 adb8877
224 [root@localhost ~]# cat test.txt|grep [a8]
225 abc
226 adb8877
227 [root@localhost ~]# cat test.txt|grep [85]
228 adb8877
229 123456
230
231 可以用^符号做[]内的前缀，表示除[]内的字符之外的字符。
232 比如搜索oo前没有g的字符串所在的行，使用 '[^g]oo' 作搜索字符串
233 [root@localhost ~]# cat haha.txt
234 abc
235 cbc
236 3bc
237 [root@localhost ~]# cat haha.txt|grep "[^a]bc"
238 cbc
239 3bc
240
241 [] 内可以用范围表示，比如[a-z] 表示小写字母，[0-9] 表示0~9的数字，[A-Z] 则是大写字母们。
242 [a-zA-Z0-9]表示所有数字与英文字符。当然也可以配合^来排除字符。
243 ^ 表示行的开头，$表示行的结尾（不是字符，是位置）那么'^$'就表示空行，因为只有行首和行尾
244 注意：^和$放在[]括号内或[]括号外都可以！！
245 [root@localhost ~]# grep [0-9] test.txt #搜索包含数字的行
246 [root@localhost ~]# grep [a-z] test.txt #搜索包含小写字母的行
247 [root@localhost ~]# grep [a-zA-Z0-9] test.txt #搜索包含数字和小写字母的行
248 [root@localhost ~]# grep [A-Z] test.txt #搜索包含大写字母的行
249 [root@localhost ~]# grep '^[a-z]' test.txt #搜索以小写字母开头的行。或者 grep

```

```

250 [root@localhost ~]# grep '^^[a-zA-Z]' test.txt #搜索开头不是英文字母的行
251 [root@localhost ~]# grep '\.$' test.txt #搜索末尾是.的行。由于.是正则表达式
252
253 =====
254 注意：在windows系统下生成的文本文件，换行会加上一个 ^M 字符。所以最后的字符会是隐藏的^M
255 可以用下面命令来删除^M符号。 ^M==\r
256 # cat dos_file | tr -d '\r' > unix_file
257
258 #####
259 #####
260 .
261 匹配一个非换行符的字符。即.符号匹配单个字符，能匹配空格
262 如：'gr.p'匹配gr后接一个任意字符，然后是p。例如'g??d' 可以用 'g..d' 表示。 good ,gxxd
263
264 *
265 匹配零个或多个先前字符
266 如：'*grep'匹配所有有一个或多个空格后紧跟grep的行。
267 如："aa*"表示搜索一个a以上的字符串！！ 其中第一个a一定存在，第二个a可以有一个或多个，
268
269 .* 一起用代表任意字符
270 .+ 字符必须出现 1 次
271 .? 字符出现 0 次或 1 次
272
273 [root@localhost ~]# grep 'g..d' test.txt #搜索
274 [root@localhost ~]# grep 'ooo*' test.txt #搜索两个o以上的字符串！！前两个o一定
275 [root@localhost ~]# grep 'goo*g' test.txt #搜索g开头和结尾，中间是至少一个o的字
276 [root@localhost ~]# grep 'g.*g' test.txt #搜索g开头和结尾的字符串在的行。 .*表
277
278 #####
279 #####
280 { } 表示限定连续重复字符的范围
281
282 .* 只能限制0个或多个， 如果要确切的限制字符重复数量，就用{范围}。范围是数字，用,隔开，
283
284 需要注意：由于{ } 在SHELL中有特殊意义，因此作为正则表达式用的时候要用\转义一下。
285
286 [root@localhost ~]# grep 'o\{2\}' test.txt #搜索包含两个o的字符串的行，2
287 [root@localhost ~]# grep 'go\{2,5\}g' test.txt #搜索g后面跟2~5个o,后面再跟一
288 [root@localhost ~]# grep 'go\{2,\}g' test.txt #搜索包含g后面跟2个以上o,后面
289
290 搜索test.txt文件中包含2次bo字符串的行（2个或2个以上的bo字符串都会被打印出来）
291 [root@localhost ~]# grep -n '\(bo\)\{2\}' test.txt
292
293 搜索test.txt文件中至少包含1次bo字符串的行
294 [root@localhost ~]# grep -n '\(bo\)\{1,\}' test.txt
295
296 搜索test.txt文件中出现1~3次包含bo字符串的行
297 [root@localhost ~]# grep -n '\(bo\)\{1,3\}' test.txt
298
299 显示test.txt 文件中至少有5个连续小写字母的字符串的行
300 [root@localhost ~]# grep -n '[a-z]\{5\}' test.txt
301
302 需要注意：
303 如果想让[]括号中的^ 不表现特殊意义，可以放在[]里面内容的后面。
304 例如：'[a-zA-z.!^ -]' 表示没有小写字母，没有. 没有!, 没有空格，没有- 的 串，注意[]里面
305
306 另外：shell 里面的反向选择为[!range]，正则里面是 [^range]。正则[^0-9]表示不以数字为开
307
308

```

```

309 ##### grep 的标签 #####
310 格式: grep '\(str\)\(\)\(\)[other]\1' filename
311
312 [root@localhost ~]# cat -n test1.txt
313     1 kevinboaakevin
314     2 kevinboeekevin
315     3 kevinbocccckevinaabb
316 [root@localhost ~]# grep -n "\(kevin\)\(bo\)\..1" test1.txt
317 1:kevinboaakevin
318 2:kevinboeekevin
319
320 [root@localhost ~]# grep 'w(es)t.*\1' test.txt
321 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
322 如果用egrep或grep -E, 就不用"\"号进行转义, 直接写成'w(es)t.*\1'就可以了。示例如下:
323 [root@localhost ~]# cat test.txt
324 westbwest
325 west123
326 westasdfwestsdf
327 [root@localhost ~]# grep 'w(es)t.*\1' test.txt
328 westbwest
329 westasdfwestsdf
330 [root@localhost ~]# grep -E 'w(es)t.*\1' test.txt
331 westbwest
332 westasdfwestsdf
333 [root@localhost ~]# egrep 'w(es)t.*\1' test.txt
334 westbwest
335 westasdfwestsdf
336
337 #####
338 #####
339 grep 扩展正则表达式是对基础正则表达式添加了几个特殊构成的。
340
341 例如: 去除空白行和行首为#的行
342 [root@localhost ~]# grep -v '^$' test.txt | grep -v '^#'
343 [root@localhost ~]# egrep -v '^$|^#' test.txt
344 [root@localhost ~]# grep -v '^$|^#' test.txt
345
346 这里列出几个扩展特殊符号:
347 + 匹配一个或多个先前的字符。如: '[a-z]+able', 匹配一个或多个小写字母后跟able的串, 如
348 ? 匹配零个或一个先前的字符。如: '(gr)?p' 匹配gr后跟一个或没有字符, 然后是p的行。注意:
349 | 表示或关系, 比如 'gd|good|dog' 表示有gd, good或dog的串
350 () 表示将部分内容合成一个单元组。 比如 要搜索 glad 或 good 可以这样 'g(la|oo)d'
351
352 ()的好处是可以对小组使用 + ? * 等。比如要搜索A开头和C结尾, 并且中间有至少一个(xyz)的字符
353
354 示例如下(注意要使用grep -E参数, 并且先前字符使用()或[]括起来):
355 [root@localhost ~]# grep -E '(gr)?p' test.txt      #搜索匹配gr后跟零个或一个字符,
356 [root@localhost ~]# grep -E '[gr]?p' test.txt      #搜索匹配gr后跟零个或一个字符,
357 [root@localhost ~]# grep -E "(2)?22" test.txt      #搜索匹配2后跟零个或一个字符,
358 [root@localhost ~]# grep -E "[2]?2" test.txt       #搜索匹配2后跟零个或一个字符,
359 [root@localhost ~]# grep -E '(2)+' test.txt        #搜索匹配2后跟一个或多个字符的
360 [root@localhost ~]# grep -E '(22)+' test.txt       #搜索匹配22后跟一个或多个字符的
361 [root@localhost ~]# grep -E '(asd)+' test.txt      #搜索匹配asd后跟一个或多个字符
362 [root@localhost ~]# grep -E 'ab|12|we' test.txt    #搜索匹配ab或12或we字符的行
363 [root@localhost ~]# grep -v 'ab|12|we' test.txt    #搜索不匹配ab或12或we字符的行
364 [root@localhost ~]# egrep -v 'ab|12|we' test.txt   #搜索不匹配ab或12或we字符的行
365 [root@localhost ~]# grep -E "glad|good" test.txt   #搜索 glad 或 good
366 [root@localhost ~]# grep -E "g(la|oo)d" test.txt   #搜索 glad 或 good
367 [root@localhost ~]# grep -E "g(lay)+h" test.txt    #搜索 g开头, h结尾, 并且中间有

```

```
368
369 再看下面一例
370 [root@localhost ~]# cat test1.txt
371 boruabcff
372 beibiaaacsf
373
374 nice to
375 meet you
376 123
377 123data
378 567
379 [root@localhost ~]# grep "1." test1.txt
380 123
381 123data
382 [root@localhost ~]# grep "aa*" test1.txt
383 boruabcff
384 beibiaaacsf
385 123data
386 [root@localhost ~]# grep "n.+" test1.txt
387 [root@localhost ~]# grep -E "n.+" test1.txt
388 nice to
389 [root@localhost ~]# echo "nb" >> test1.txt
390 [root@localhost ~]# grep -E "n." test1.txt
391 nice to
392 nb
393 [root@localhost ~]# grep -E "n.+" test1.txt
394 nice to
395 nb
396 [root@localhost ~]# grep -E "n.?" test1.txt
397 nice to
398 nb
399
400 通过上面，可以看到：
401 在 aa* 的时候出现了这么多，它的意思是匹配 a 字符后面的任意多个；
402 在直接 n.+ 的时候并没有出现 n 开头的字符，必须加上 -E 才能显示出；
403
404 那么 .+ 和 .? 的区别是什么呢？（可以从grep打印结果的标红字符串中看出区别）
405 .+ 是全部匹配出，而 .? 只是匹配出字符 n 后面紧跟的一个字符。
406
407 接着看下来看看 Perl 的正则表达式，grep 需要加-P参数：
408 .* 的贪婪匹配；
409 .*? 的惰性匹配。
410
411 上面二者之间的区别在于：贪婪匹配是全部匹配到整个字符串，而惰性匹配只是匹配到 tom 这个字
412
413 如下示例：通过打印结果中的标红字体就可以看出贪婪匹配和惰性匹配的区别
414 [root@localhost ~]# grep -P "kevin.*" test2.txt
415 kevin
416 kevinisgoodaiyadate
417 kevinbeijingshangkevin321
418 xiaoruiskevin
419 [root@localhost ~]# grep -P "kevin.*?" test2.txt
420 kevin
421 kevinisgoodaiyadate
422 kevinbeijingshangkevin321
423 xiaoruiskevin
424
425
426 ##### Re正则表达式的几个总结 #####
```



```

427 \    忽略正则表达式中特殊字符的原有含义
428 ^    匹配正则表达式的开始行
429 $    匹配正则表达式的结束行
430 \<    从匹配正则表达式的行开始
431 \>   到匹配正则表达式的行结束
432 [ ]  单个字符    如[A] 即A符合要求 , [bB]即b或B符合要求, [abc]即a或b或c符合要求
433 [ - ] 范围      如[A-Z] 即A, B, C一直到Z都符合要求
434 .    有的单个字符
435 *    所有字符, 长度可以为0
436
437 #####
438 #####
439 1) grep -f <范本文件> 或 grep --file=<范本文件>
440 表示指定范本文件, 其内容含有一个或多个范本样式, 让grep查找符合范本条件的文件内容, 格式为
441
442 简单示例:
443 下面命令就可以打印出文件test2.txt中与文件test1.txt中的相同行。
444 [root@localhost ~]# grep -f test1.txt test2.txt
445 [root@localhost ~]# grep --file test1.txt test2.txt
446 [root@localhost ~]# grep --file=test1.txt test2.txt
447 相当于
448 [root@localhost ~]# cat test1.txt test2.txt|sort|uniq -d
449
450 但是要注意一个细节:
451 grep -f 参数后面作为标准的第一个文件一定不能有空行才行!!! 否则打印结果就是最后一个文件
452
453 [root@localhost ~]# cat haha.txt
454
455 123
456 [root@localhost ~]# cat test.txt
457 123
458 beijing
459 [root@localhost ~]# grep -f test.txt haha.txt
460 123
461 [root@localhost ~]# grep -f haha.txt test.txt    #grep -f后的第一个文件一定不能有!
462 123
463 beijing
464
465 2) grep -F 关键字 filename1" 表示将关键字视为固定字符串的列表。
466 或者
467 "grep -F 关键字 filename1 filename2 filename" 表示会显示出来关键字所在的文件的号
468
469 [root@localhost ~]# cat test1.txt
470 beijing
471 anhui
472 shanghai
473 shenzhen
474 [root@localhost ~]# cat test2.txt
475 beijing
476 linux
477 kevin
478 [root@localhost ~]# grep -F beijing test1.txt
479 beijing
480 [root@localhost ~]# grep -F beijing test1.txt test2.txt
481 test1.txt:beijing
482 test2.txt:beijing
483
484 [root@localhost ~]# grep -F "anhui" test1.txt test2.txt
485 test1.txt:anhui

```

```
486
487 #####
488 #####
489 grep指定查找范围是目录时必用的参数:
490 grep -r 表示明确要求搜索子目录。等同于 grep -d recurse
491 grep -d skip 表示忽略子目录
492
493 小示例:
494 遍历当前目录及所有子目录查找匹配"kevin"的行,使用"grep -r"
495 [root@localhost ~]# grep -r "kevin" .
496 等同于
497 [root@localhost ~]# grep -d recurse "kevin" .
498
499 在当前目录及所有子目录下的txt结尾文件中查找"kevin"
500 需要注意: 命令中*.txt一定要加上引号,表示在当前及其子目录下;如果不加引号,则表示仅仅在当
501 [root@localhost ~]# grep -r kevin . --include "*.txt"
502 等同于
503 [root@localhost ~]# grep -d recurse kevin . --include "*.txt"
504
505 在当前目录下查找包含"kevin"字符的文件,忽略当前目录下的子目录里的文件。
506 下面命令只会对当前目录下的文件进行查找,当前目录下的子目录里的文件会忽略!
507 如果不加-d skip,直接使用grep "kevin" ./* 命令,则结果中会将当前目录的子目录打印出来,
508 [root@localhost ~]# grep -d skip "kevin" ./*
509 ./register.yml:- hosts: kevin
510 ./test1.txt:kevinboaakevin
511
512 [root@localhost ~]# grep "kevin" ./*
513 ./register.yml:- hosts: kevin
514 grep: ./test: Is a directory      #将当前目录下的子目录test作为错误提示信息打印出来
515 ./test1.txt:kevinboaakevin
516
517 利用 grep 和 find 命令查找文件内容
518 从根目录开始查找所有扩展名为.log的文本文件,并找出包含 "ERROR" 的行:
519 [root@localhost ~]# find / -type f -name "*.log" | xargs grep "ERROR"
520
521 从当前目录开始查找所有扩展名为.in的文本文件,并找出包含 "kevin_log" 的行:
522 [root@localhost ~]# find . -name "*.in" | xargs grep "kevin_log"
```

\*\*\*\*\* 当你发现自己的才华撑不起野心时, 就请安静下来学习吧! \*\*\*\*\*

分类: [常规运维](#), [Shell](#)

好文要顶

关注我

收藏该文



散尽浮华  
[关注 - 23](#)  
[粉丝 - 2968](#)

[+加关注](#)

30

« 上一篇: [Linux下针对服务器网卡流量和磁盘的监控脚本](#)  
» 下一篇: [confluence上传文件附件预览乱码问题 \(linux服务器安装字体操作\)](#)

posted @ 2018-07-12 14:18 散尽浮华 阅读(12453) 评论(1) 编辑 收藏

评论

#1楼 2020-09-04 12:01 | faithfu

很详细, 查漏补缺特别有帮助, 感谢