# linux 基础

运算器 + 控制器：CPU

存储器：RAM 易失性存储器 编址类型存储器

I/O设备 ： 硬盘，键盘，鼠标等

程序：指令和数据组成

控制器： 取数据 取指令

地址总线 ： 内存寻址

数据总线 ： 传输数据

控制总线 ： 控制指令

**总线是复用的**

寄存器： cpu暂存 存取速度最快

input设备：

output设备：

Linux系统设定

**标准输出： stdout 1**

**标准输入： stdin 0**

**标准错误输出：stderr 2**

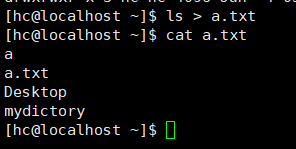
**默认的输出设备：**键盘

**默认的输入设备：**显示器

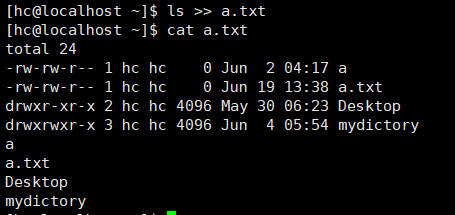
## 输出重定向

**改变默认的输出输入来源称I/O重定向**

### > 覆盖输出

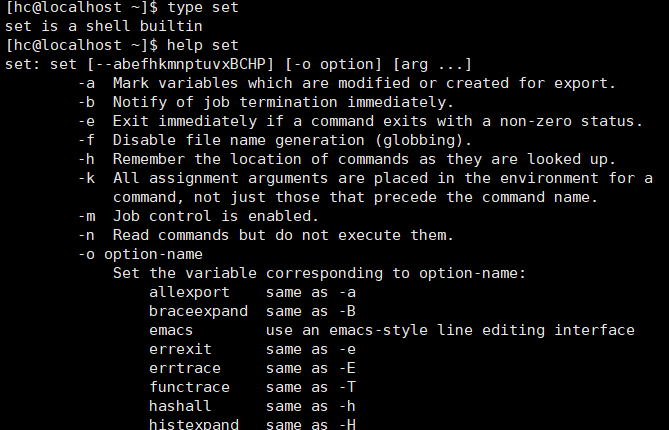


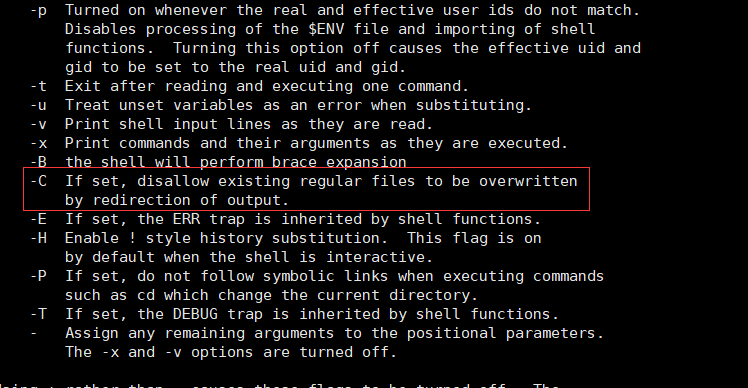
### >> 追加输出



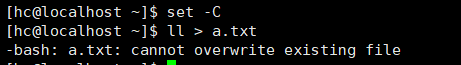
### set

开启关闭bash的一些功能

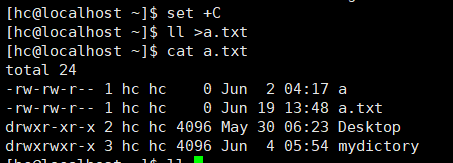




**set –C** 禁止对已经存在的文件使用覆盖重定向



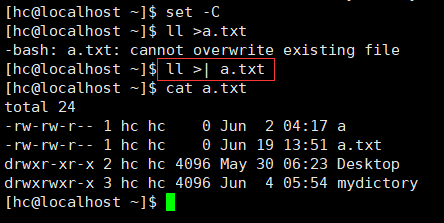
**set +C** 开启对文件重定向的覆盖功能



### >|

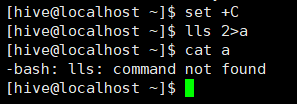
**强制覆盖输出**

ll >| a.txt 覆盖功能关闭，但是又想覆盖



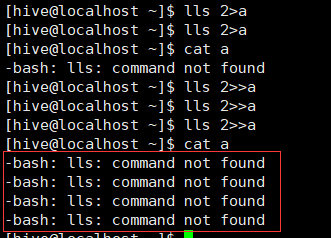
### 2>

重定向错误覆盖输出

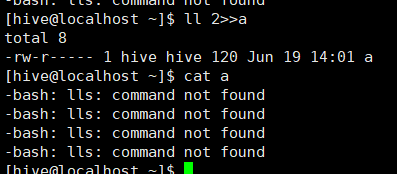


### 2>>

重定向错误追加输出

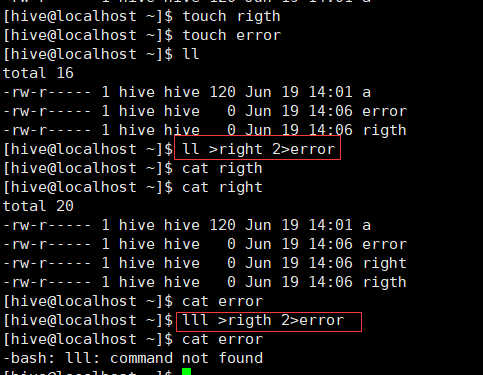


如果是正确的命令，那么又到显示器上来了，对于正确的又无能为力了



当然可以定义正确的到正确的文件中，错误的到错误的文件中去

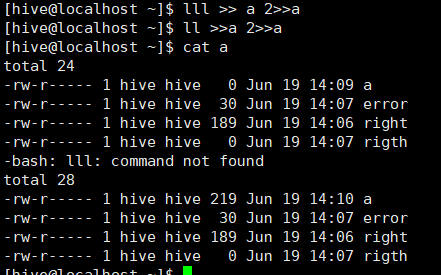
**ll >rigth 2<error**  **lll >rigth 2<error**



当然无论错误正确都可以保存到同一个文件中去

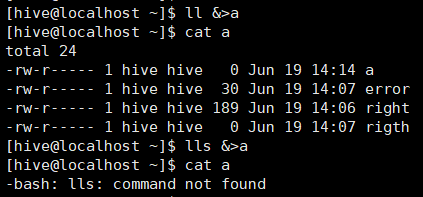
lll >>a 2>>a

ll >>a 2>>a



### &>

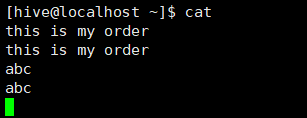
无论是正确还是错误都重定向到同一个文件中去



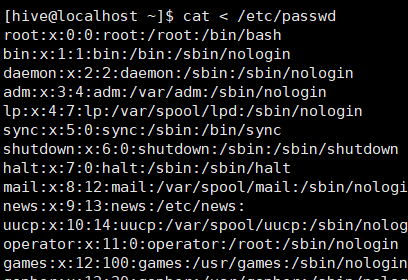
## 输入重定向

### <

有些输入命令带有默认输入重定向(默认键盘)，比如cat



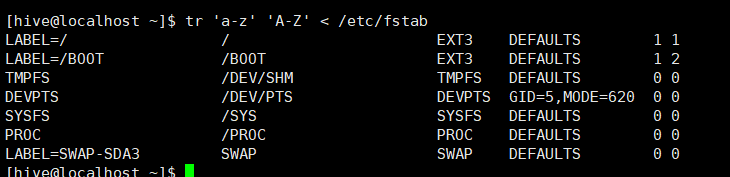
cat </etc/passwd



tr 字符处理也有默认的重定向（键盘）



tr 全部把输入的内容转换为大写，并把输入重定向到 /etc/fstab

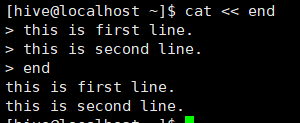


### <<

就不是追加的功能？而是**此处生成文档**。

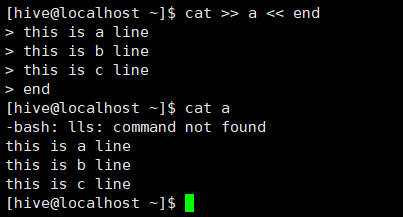
cat << END END 文档结束标记。

cat << EOF 文档结束标记



cat >> a << end 将输出重定向到a文件，并生成文档，使用结束符end

这样就可以使用脚本生成一个文件。



如果又想输出到文档中，有想输出到屏幕上咋办？

## 管道

### |

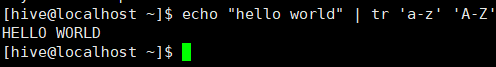
命令管道： 前一个命令的输出作为后一个命令的输入。

**命令1 | 命令2 | 命令3**

**命令1的输出作为命令2的输入，命令2 的输出作为命令3的输入**。

**echo “hello word” | tr ‘a-z’ ‘A-Z’**

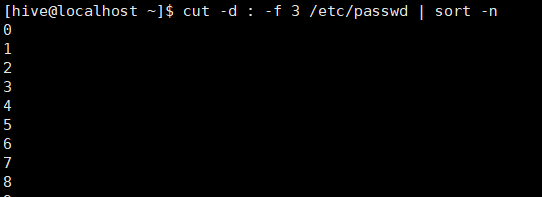
echo命令的输出当做tr命令的输入 全部将结果小写转换为大写。



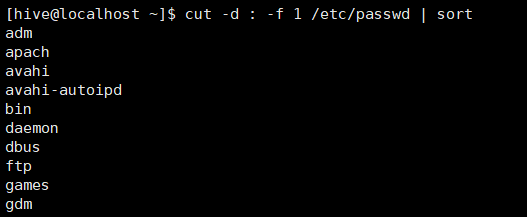
echo “redhat” | passwd -- stdin



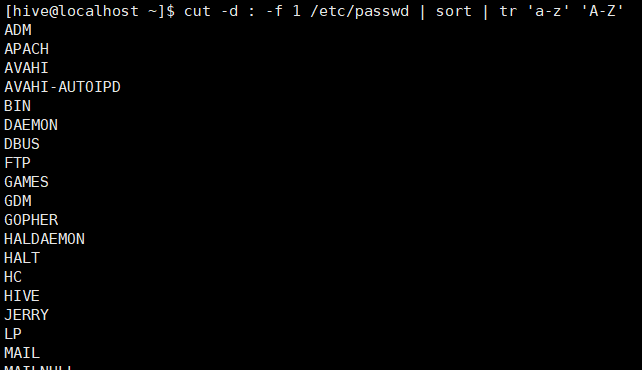
cut –d : -f 3 /etc/passwd | sort –n 将passwd文件中的第3个字段按数字大小排序



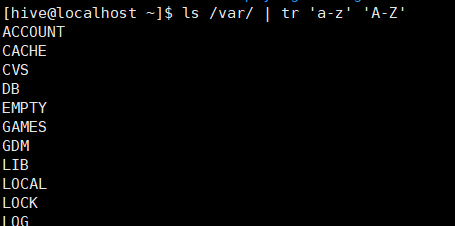
cut –d : -f 1 /etc/passwd | sort



cut –d : -f 1 /etc/passwd | sort | tr ‘a-z’ ‘A-Z’

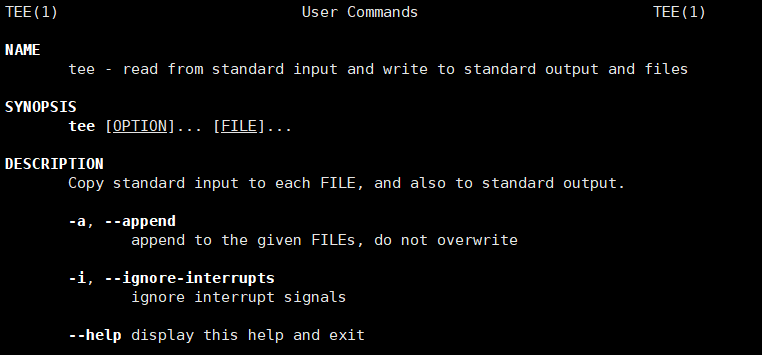


ls /var/ | tr ‘a-z’ ‘A-Z’



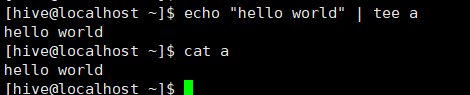
组合小命令完成复杂的任务，这是Linux重要的思想。

### tee



echo “hello world” | tee a

tee 可以将标准输入的内容保存到指定文件中，又可以将内容标准输出。

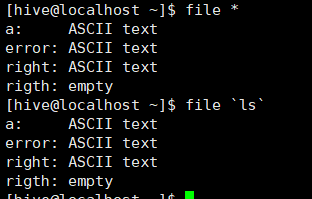


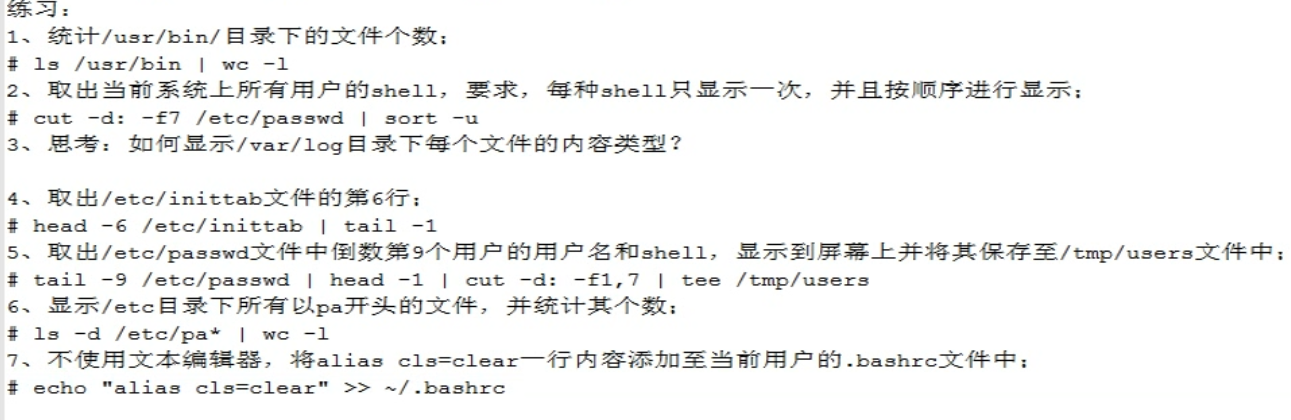
显示文件的行数。

wc –l /etc/passwd | cut –d ‘ ’ –f 1



查看目录下所有文件

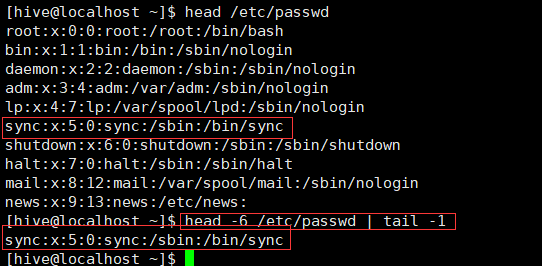




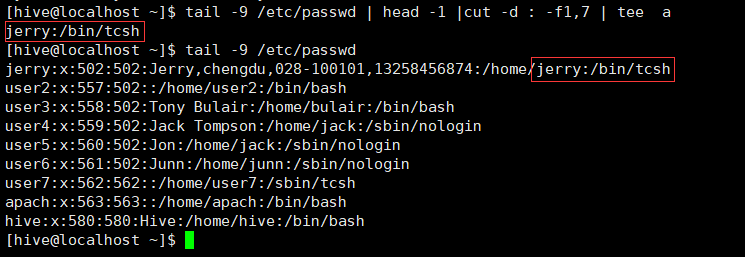
echo “alias cls = clear” >> ~/.bashrc

取出/etc/passwd的第6行

head -6 /etc/passwd | tail -1



tail -9 /etc/passwd | head -1 | cut –d : -f 1,7 |tee a





### grep

文本查找的需要

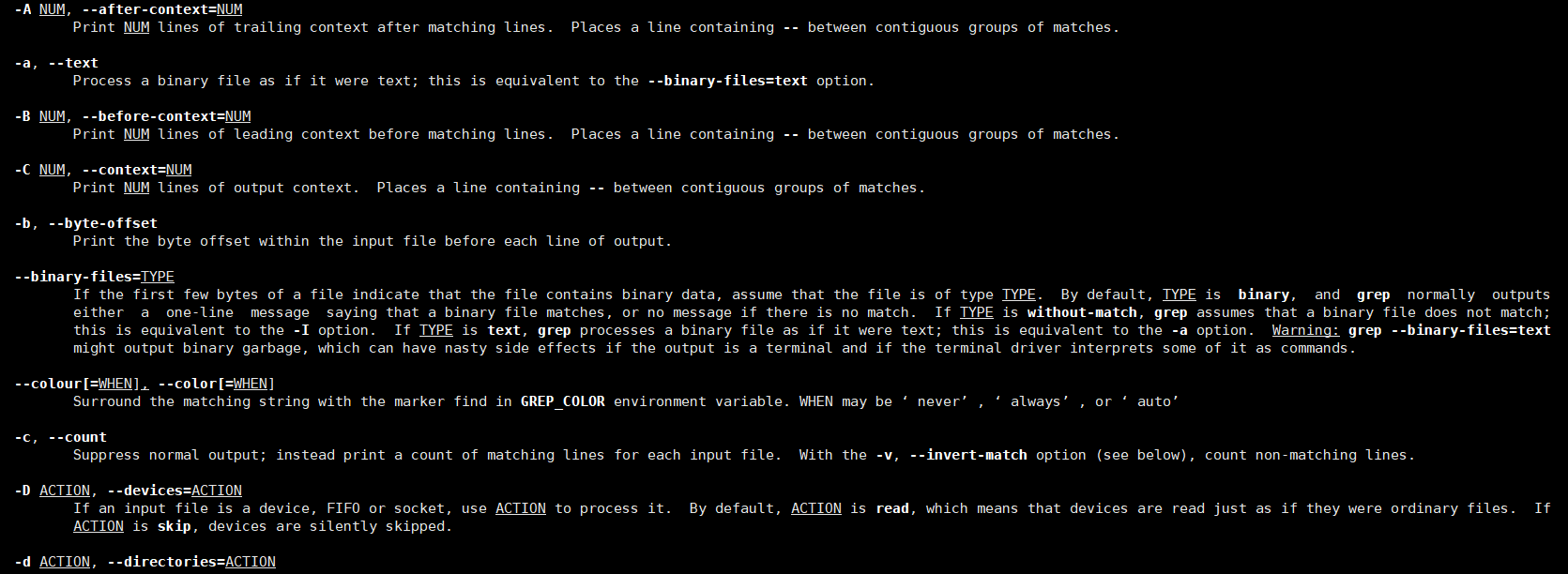
grep , egrep ,fgrep

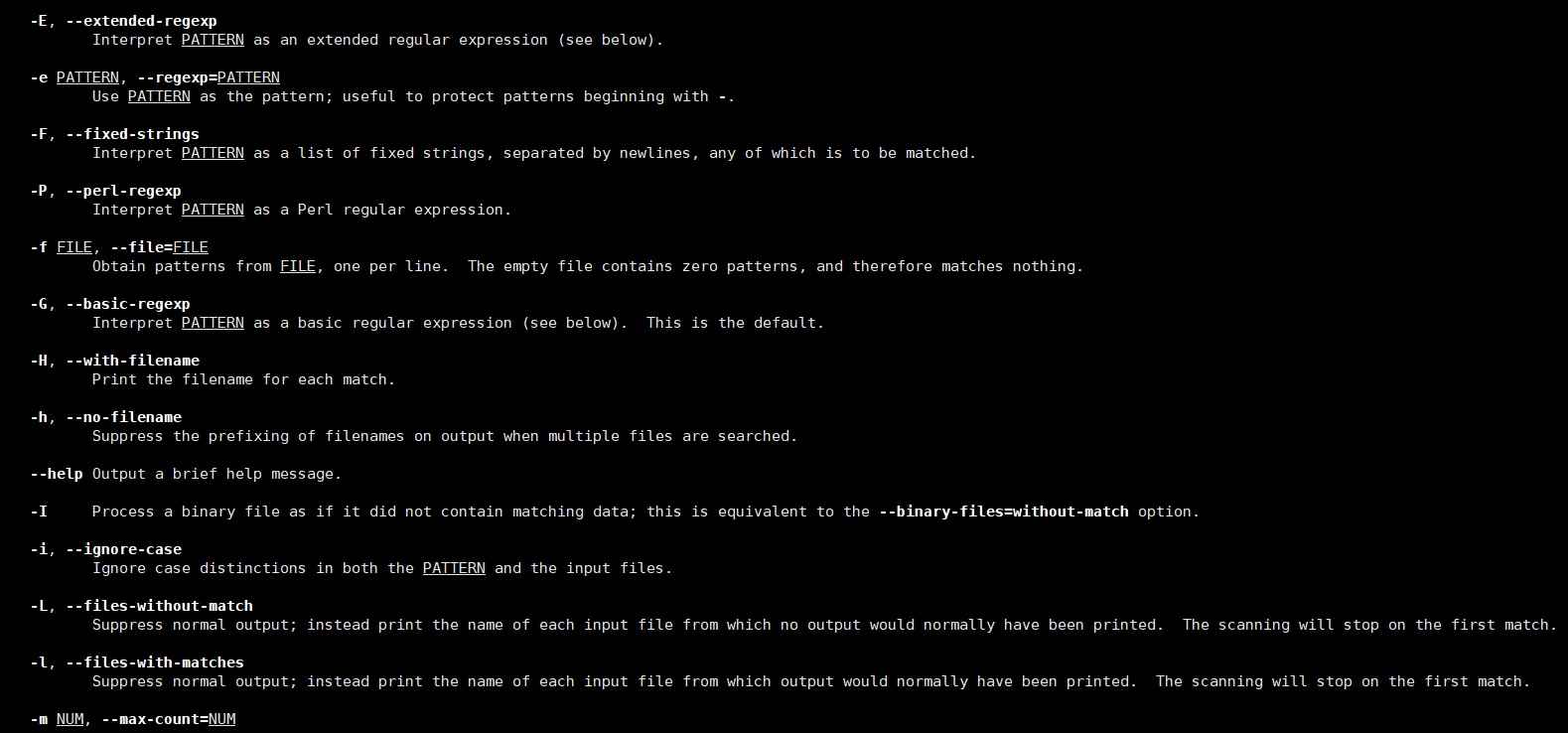
grep 根据模式去搜索文本，并将符合模式的文本显示出来

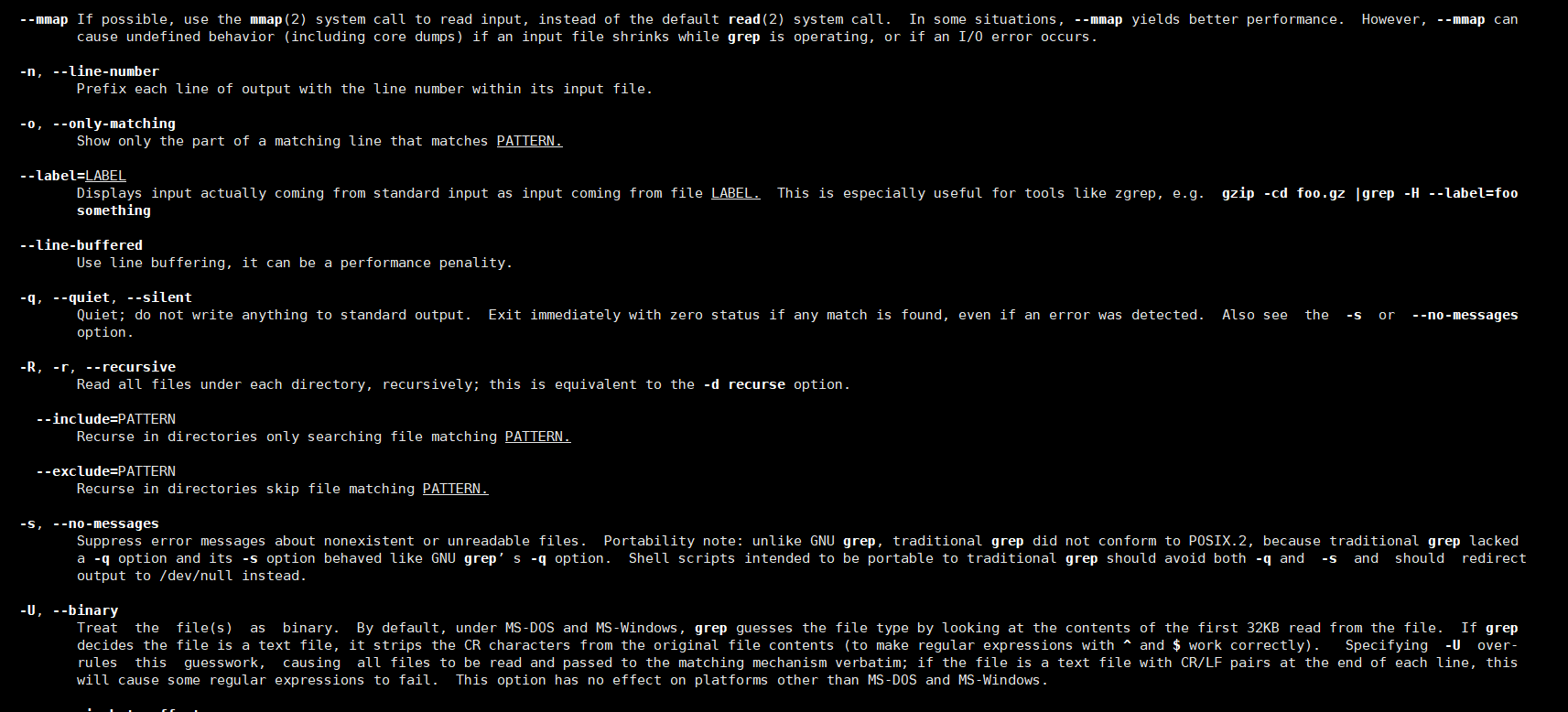
pattern:文本字符和正则表达式的元字符组合而成的匹配条件

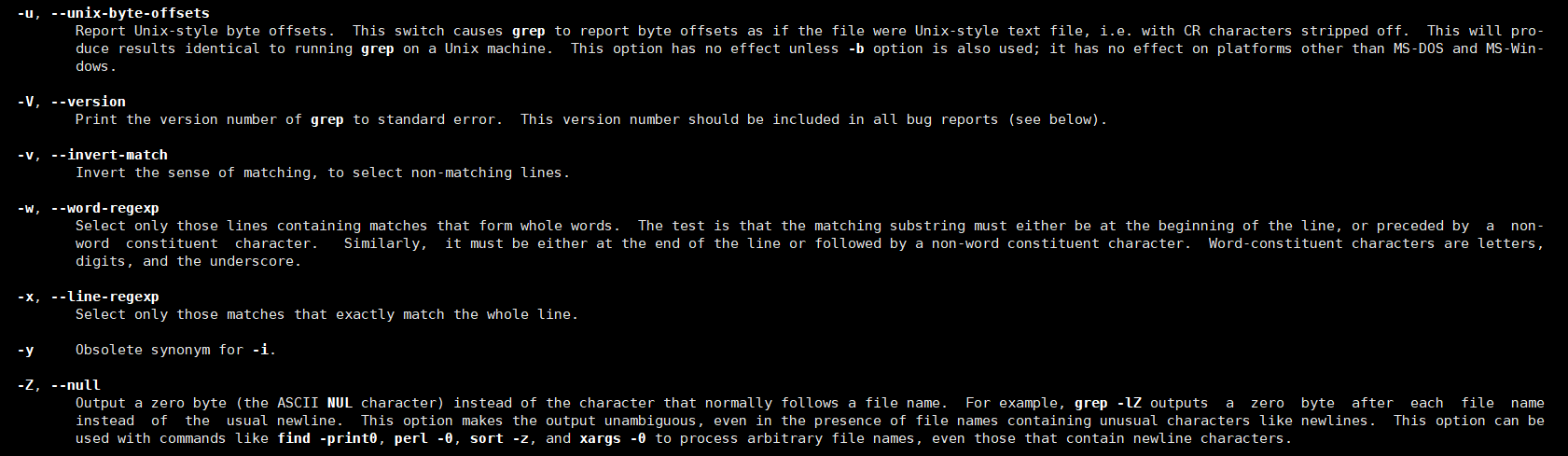
grep [options] PATTERN [FILE...]

grep [options] [-e PATTERN | -f FILE] [FILE...]



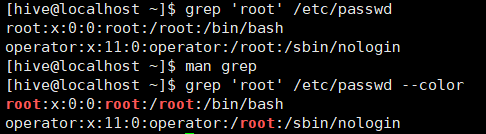






**-I ,** --ignore-case 忽略大小写

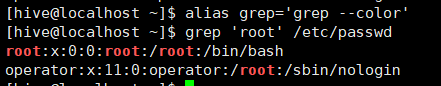
**--colour** [=WHEN], --color[=WHEN] 匹配到的用颜色显示



很方便，那就定义别名 alias grep = ‘grep --color’

echo “alias grep=’grep --color’” >> /home/hive/.bashrc

重新登录既可以使用



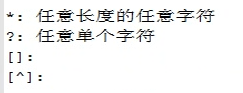
**-v**, --invert-match

显示没有被匹配到的行

**-o,** --only-matching

只显示被模式匹配到的字符串

文件名通配：



## 正则表达式

正则表达式：regular expression

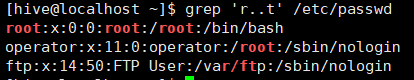
正则表达式分为两类：

**基本正则表达式** basic grep

**扩展正则表达式** extended grep

### 元字符

**.** ：表示匹配任意单个字符



### 匹配次数

**工作在贪婪模式下，尽可能长的匹配**

**\*** ：表示匹配前面的字符任意次

a, b, ab, aab, adb, amdb

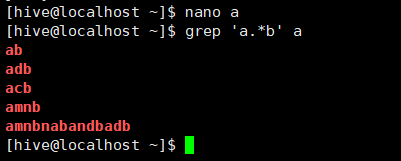
a\*b a出现任意次（0-n）以b结尾的 b , ab , aab

**.\***  : 表示任意长度的任意字符

a, b, ab, aab, adb, amdb

a.\*b 表示 **\*** 匹配的是 **.** 跟a没关系，意思就是a开头b结尾，中间是啥都行

ab, aab, adb, amdb



**\？** ： 匹配其前面的字符1次或者0次 就是可有可无

a, b, ab, aab, adb, amdb

a?b a可有可无，只有要b就可以 ，只要有部分匹配就可以

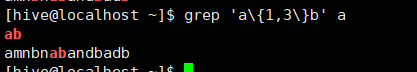


**\{m,n\}** : 匹配其前面字符最少m次，最多n次

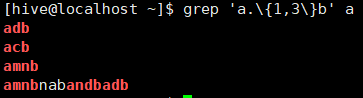
**\{1, \}** : 匹配其前面字符最少1次，最多无限次

**\{0, 3\}** : 匹配其前面字符可以没有，最多3次

grep 'a\{1,3\}b' a

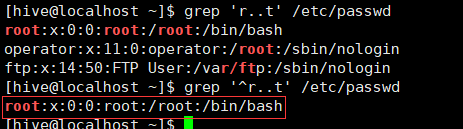


grep 'a.\{1,3\}b' a



### 位置锚定

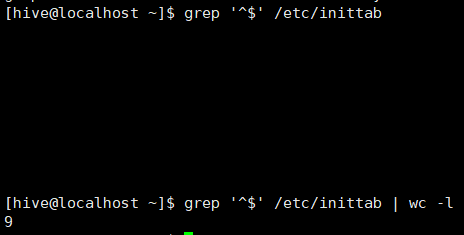
**^** : 锚定行首，此字符后面的内容必须出现在行首



**$**  : 锚定行尾，此字符后面的内容必须出现在行尾

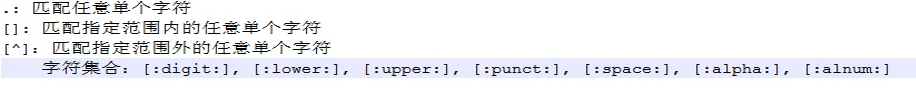


**^$** : 空白行



**[ ]** : 匹配指定范围内的任意单个字符

**[^ ]** : 匹配指定范围外的任意单个字符

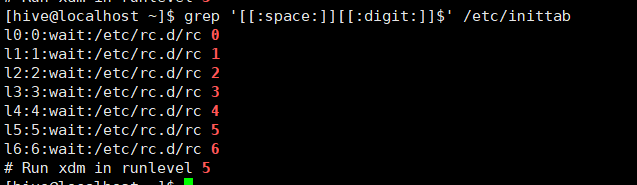


字符集合： [:digit:] , [:lower:] , [: upper:] , [:punct:] , [:sapce:] , [:alpha:] , [:alnum:]

grep '[[:digit:]]$' /etc/inittab 过滤以数字结尾的行



grep '[[:space:]][[:digit:]]$' /etc/inittab 匹配以空白开头，数字结尾的行



**\< 或者 \b**: 锚定词首，其后面的任意字符必须作为单词的首部出现

**\> 或者 \b**: 锚定词尾，其后面的任意字符必须作为单词的尾部出现

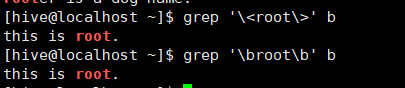
\<root\> : root这个单词必须精确匹配。



grep 'root\>' b 以root作为词尾的



grep '\<root' b 以root作为词首的



### 分组

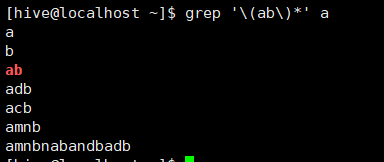
**\( \)** 后向引用

\1: 引用第一个左括号与之对应的右括号所包括的所有内容

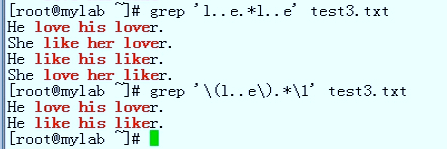
\2:

\n:

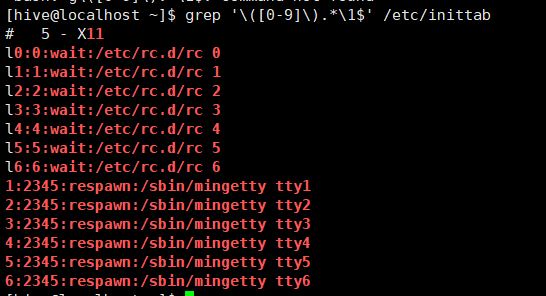
\(ab\)\* ab当作一个整体出现任意次，有或者没有



grep ‘\(l..e).\*\1’



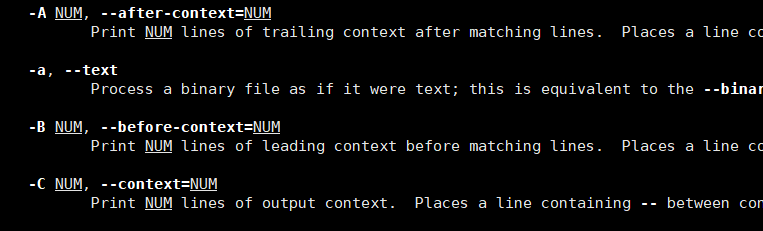
grep '\([0-9]\).\*\1$' /etc/inittab 行首数字与行尾的数字相同



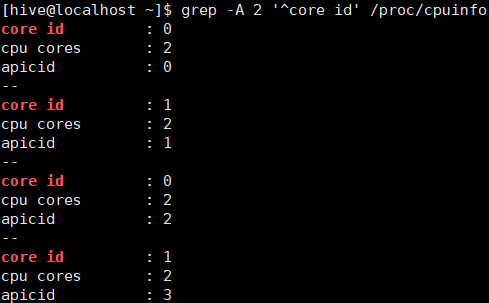
**-E,** --extended-regexp

Interpret PATTERN as an extended regular expression (see below).

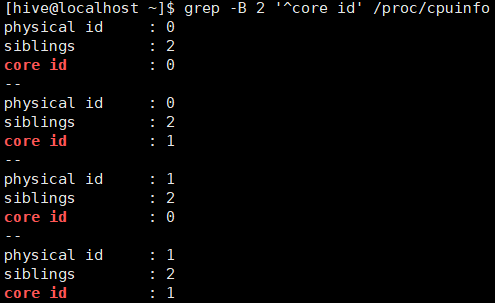
扩展的正则表达式



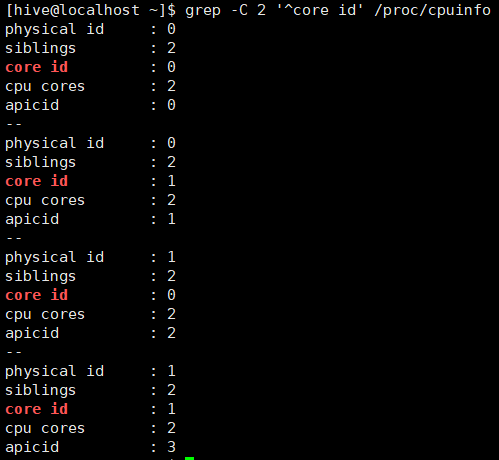
**-A** (after): -A 2 匹配到的内容还有其后面的两行显示出来



**-B** (before): -B 2 匹配到的内容还有其前面的两行显示出来



**-C** (context): -C 2 匹配到的内容还有其前后的两行显示出来



## 扩展正则表达式

工作在贪婪模式下

**grep -E == egrep**

### 字符匹配

**.** :任意字符

**[ ]** :字符范围内

**[^]** :字符范围外

和正则表达式是一样的

### 次数匹配

**\*** ：任意次数

**？** ：没有反斜线 匹配0次或者1次

**+** ： 匹配其前面字符至少1次

**{m,n}**: 次数匹配 ，就没有反斜线了

### 位置锚定

**^** : 锚定行首，此字符后面的内容必须出现在行首

**$**  : 锚定行尾，此字符后面的内容必须出现在行尾

**\<** : 锚定词首，其后面的任意字符必须作为单词的首部出现

**\>** : 锚定词尾，其后面的任意字符必须作为单词的尾部出现

用法都一样

### 分组

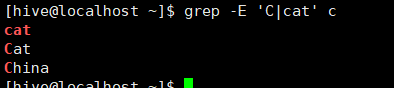
（）不需要加反斜线

\1 , \2 ……

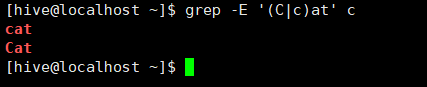
### 或者

| ： or

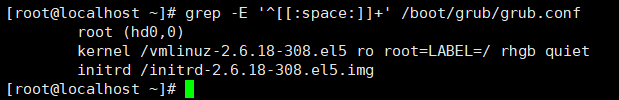
grep -E 'C|cat' c 匹配 | 的前面和后面



grep -E '(C|c)at' c 小写或者大写c ，at结尾的

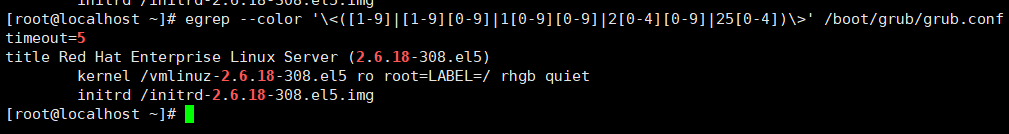


grep -E '^[[:space:]]+' /boot/grub/grub.conf 至少一个空白字符开头的行

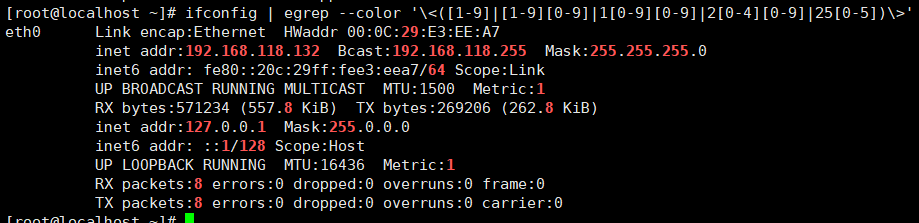


egrep --color '\<([1-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-4])\>' /boot/grub/grub.conf

1-254 个位 十位 百位 分别匹配



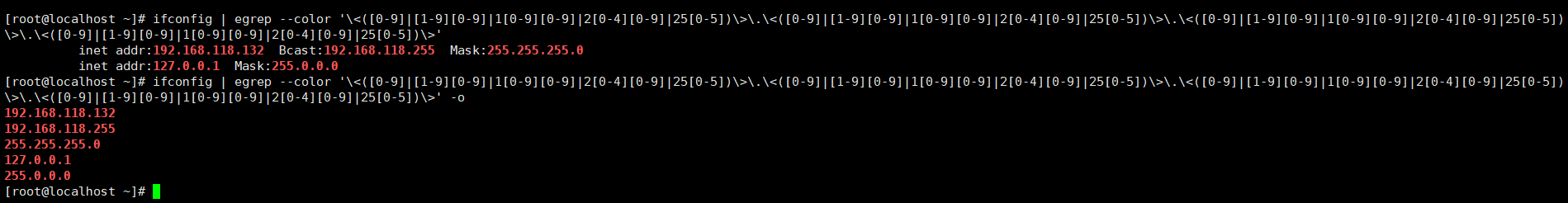
ifconfig | egrep --color '\<([1-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\>'



匹配文件中的IP地址

ifconfig | egrep --color '\<([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\>\.\<([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\>\.\<([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\>\.\<([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\>'

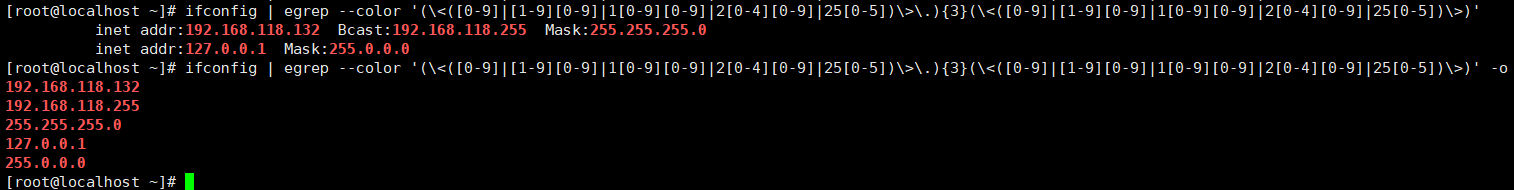
ifconfig | egrep --color '\<([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\>\.\<([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\>\.\<([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\>\.\<([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\>' -o

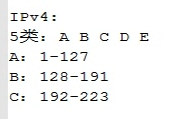


这样写太麻烦了，可以写短一点

ifconfig | egrep --color '(\<([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\>\.){3}(\<([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\>)' -o

为什么不是4次，那是最后一个没有.





精确找出IP地址，A B C 3类地址

比如 255.255.255.0 就是子网掩码，不能是IP D类和E类是组播和搞研究的不能用

ifconfig | egrep --color '\<[1-9]|[0-9]|1[0-9]{2}|2[0-1][0-9]|22[0-3]\>(\.\<([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-4])\>){2}\.\<([1-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-4])\>'

## 快速正则表达式

**fgrep** 不支持正则表达式，速度快