

# GNU/Linux 字符管理命令



# 字符管理命令 -grep

命令：grep

功能：通过正则表达式查找文件中的关键字

正则表达式：

又称正规表示法、常规表示法（英语：Regular Expression，在代码中常简写为 regex、regexp 或 RE）。正则表达式使用单个字符串来描述、匹配一系列符合某个句法规则的字符串。在很多文本编辑器里，正则表达式通常被用来检索、替换那些符合某个模式的文本。

# 字符管理命令 -grep

参数：

-i: 忽略大小写

-c: 打印匹配的行数

-C: 打印出匹配的上下文 ( 上 N 行 , 下 N 行 ) 的多少行

-l: 列出匹配的文件名

-L: 列出不匹配的文件名

-n : 打印包含匹配项的行和行标



# 字符管理命令 -grep

参数：

-w: 仅匹配指定的单词而非关键字

-e: 索引匹配字符串

-r: 递归查询

-v: 不输出匹配的行

-A < 行号 >: 显示所找的匹配字段，并显示下面指定的行数的信息

-B < 行号 >: 显示所找的匹配字段，并显示上面指定的行数的信息



## 字符管理命令 -grep

示例：

1. 递归且不区分大小写对 test 字段查找

```
#grep -ri "test" ./
```

2. 打印匹配" test" 关键字有多少行

```
#grep -c "test" grepcmd.txt
```

3. 打印匹配" test" 关键字的上下文各 1 行

```
#grep -C 1 "test" grepcmd.txt
```

4. 打印匹配 test 关键的行并显示行号

```
#grep -n "test" grepcmd.txt
```



# 字符管理命令 -grep

正则表达式表示方法：

\ 忽略正则表达式中特殊字符的原有含义

^ 匹配正则表达式的开始行

\$ 匹配正则表达式的结束行

\< 从匹配正则表达式的行开始

\> 到匹配正则表达式的行结束

[] 单个字符；如 [A] 即 A 符合要求

[n - m] 范围；如 [A-H] 即包含 A 至 H 都符合要求

. 所有的单个字符

\* 所有字符，长度可以为 0



## 字符管理命令 -grep

示例：

1. 查找开头为” #” 的行，并显示行号

```
#grep -n ^# grepcmd.txt
```

2. 查找在本地目录下 (含子目录) 结尾为” c” 的文件

```
#grep -r c$ ./
```

3. 查找以” man” 开头的单词

```
#grep '\<man'* grepcmd.txt
```

4. 查找” man” 仅匹配此三个字符

```
#grep '\<man\>' grepcmd.txt
```



# 字符管理命令 -grep

示例：

5. 查找含有以” D” 字符开头的行

```
#grep -n ^[D] grepcmd.txt
```

6. 查找含有以” A-F” 字符开头的行

```
#grep -r “[A-F]” ./
```

7. 查找含有以” A-F” 字符开头，第 2 个字符为 i 的行

```
#grep “[A-F]i” grepcmd.txt
```





# 字符管理命令 -grep

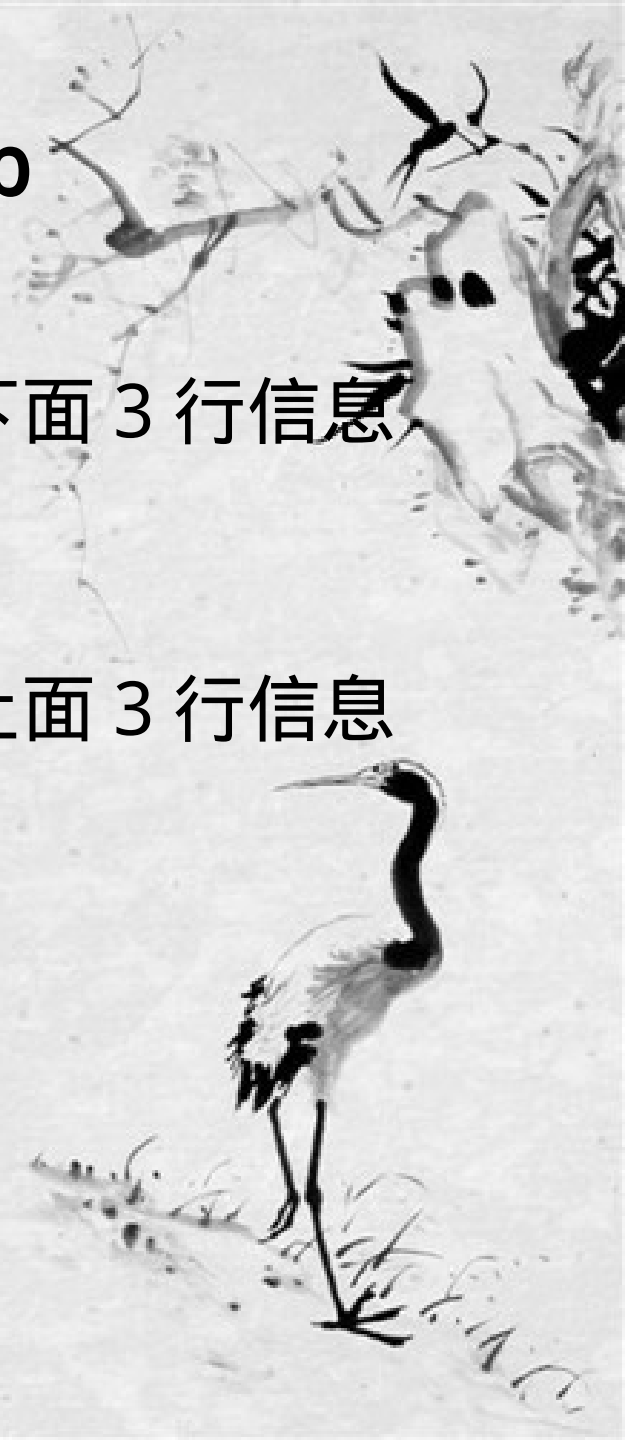
示例：

8. 查找含有以 'nobody' 字符的行及下面 3 行信息

```
#grep -A 3 'nobody' /etc/passwd
```

9. 查找含有以 'nobody' 字符的行及上面 3 行信息

```
#grep -B 3 'nobody' /etc/passwd
```



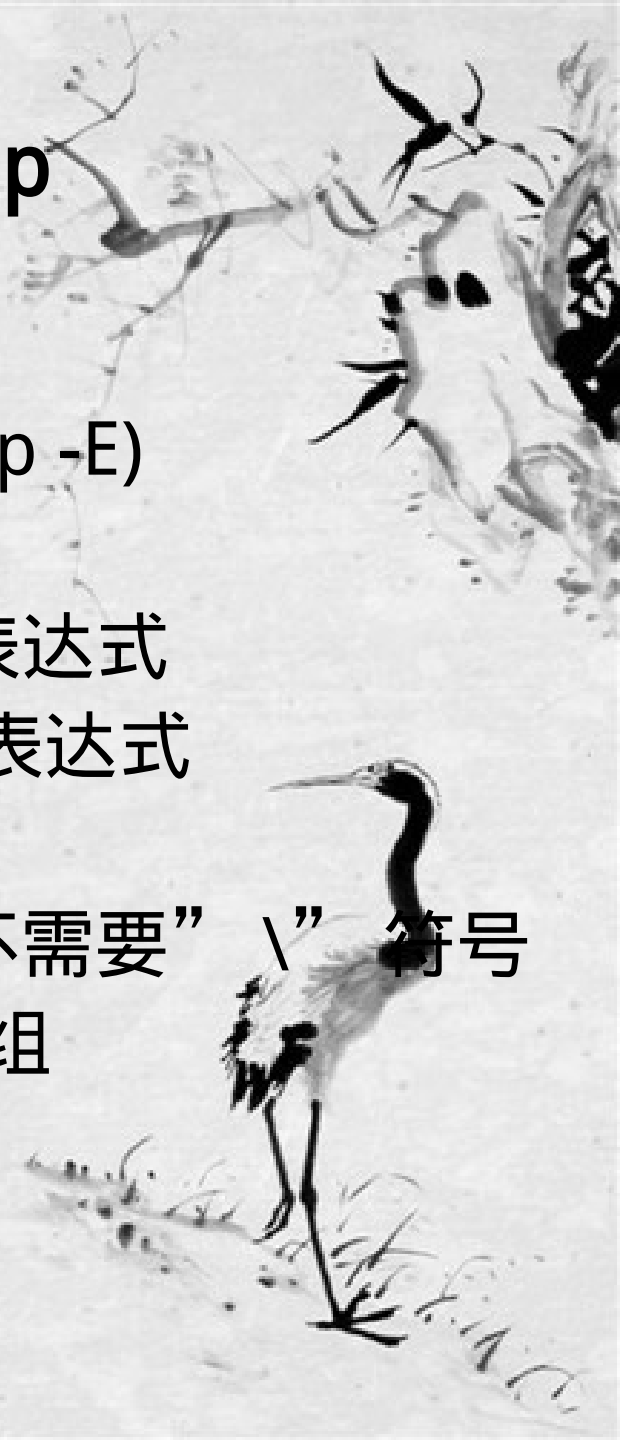
# 字符管理命令 -egrep

命令 : egrep

信息 : grep 的增强版 / 扩充版 ( 即 grep -E)

特点 :

1. 支持 ? 匹配 , 即匹配 0 到 1 个正则表达式
2. 支持 + 匹配 , 即匹配 1 到 N 个正则表达式
3. 支持 " 或关系 " 的匹配
4. 在查找范围时 , 可直接用 {a,z}, 而不需要 " \ " 符号
5. 可以被 () 来包含正则表达式进行分组
6. 参数与 grep 基本通用



# 字符管理命令 -egrep

示例：

1. 查找以 D 或 d 字符为开头的行

```
#egrep '^D|^d' egrepcmd.txt
```

2. 匹配以 D 开头的 0 个或 1 个字符

```
#egrep ^D? egrepcmd.txt
```

3. 查找不包含” chuai” 及” lisa” 的字段

```
#egrep -v '^(chuai|lisa)' egrepcmd.txt
```



# 字符管理命令 -cut

命令：cut

功能：对所需字符进行截取

-d “n”：定义分界符，即点位

-f n: 取第几位的字符



# 字符管理命令 -cut

示例：

1. 以空格符为分界符，进行第 2 位截取。

```
#cut -d " " -f 2 ./cutcmd.txt
```

2. 以空格符为分界符，进行第 1,3 位截取。

```
#cut -d " " -f 1,3 ./cutcmd.txt
```



# 字符管理命令 -sed

命令 :sed

功能：通过指定的正则表达式完成指定关键字的过滤、截取、修改等操作

语法格式：

sed 'command' filename(s)



# 字符管理命令 -sed

特点：

- 1.sed 属于一个流线式的非交互式的编辑器
2. sed 在输入命令和文件名后，将在屏幕上输出
3. 在不用重定向至文件之前，是不会改变文件现有内容。  
以避免修改文件时出现问题

# 字符管理命令 -sed

工作模式：

- 1.sed 一次处理一行文件的文本并将当前处理的行存储在临时缓冲区中（此又被称为模式空间“pattern space”）
2. Sed 一旦完成对模式空间中的行的处理，行将从空间中输出到屏幕。
3. sed 在继续读入下一行，在如此继续，直至全部执行完成



# 字符管理命令 -sed

工作模式：

- 1.sed 通过定址（指定行或者关键字等）来判断哪一行是用户所希望编辑的字段。
2. 定址可以通过数字、正则表达式或者二者相结合的方式来进行
3. 如果没有定址，则 sed 将处理输入文件的所有行

# 字符管理命令 -sed

## sed 之动作命令

命令	功能
a\	在当前行后面加入一行或者文本
b label	分支到脚本中带有标号的地方，如果标号不存在就分支到脚本的末尾
c\	用新文本改变或者替代本行的文本
d	从模式空间中制删除指定行
D	删除模式空间中第一行
i\	在当前行上面插入文本
h	拷贝模式空间到内存缓冲区
H	追加模式空间内容到内存缓冲区
g	获得内存缓冲区的内容，并替代当前模式空间中的文本
G	获得内存缓冲区的内容，并追加当前模式空间中的文本

# 字符管理命令 -sed

## Sed 之动作命令

命令	功能
l	列表不能打印所指定的字符清单
n	读取下一个输入行，用下一个命令处理新的行
N	追加下一个输入行到模式空间后面并在二者之间嵌入一个新的行，改变当前行的号码
p	打印模式空间的行
P	打印模式空间的第一行
q	退出 sed
r file	从 file 中读取行
t label	if 分支，从最后一行开始开始，一旦满足要求，将直接到带有标号的命令出，或者到脚本的末尾
T label	错误分支，从最后一行开始开始，一旦满足要求，将直接到带有标号的命令出，或者到脚本的末尾

# 字符管理命令 -sed

## Sed 之动作命令

命令	功能
w file	写并追加到模块空间 file 末尾
W file	写并追加到模块空间的第一行到 file 末尾
!	表示后面的命令对所有没有被选定的行发生作用
s/re/string/	用 string 替换正则表达式 re
=	打印当前行号码
#command	把注释扩展到下一个换行符以前
替换标记	
g	行内全面替换
p	打印行
w	把行写入一个文件

# 字符管理命令 -sed

## Sed 之动作命令

命令	功能
替换标记 (二)	
x	互换模块空间的文本和缓冲区的文本
y	把一个字符翻译为另外的字符 (此替换标记不可用正则表达式)
选项	
-e command	允许多点编辑
--expression=command	同上
-h,--help & --version	帮助 & 查看 sed 版本 (2 个选项)
-n,--quiet,--silent	取消默认输出
-f script-file	引导 sed 脚本文件名
file=script file	同上



# 字符管理命令 -sed

## Sed 匹配符号

元字符	功能	例子	匹配
^	指定行的开始	/^linux/	所有以 linux 开头的行
\$	指定行的末尾	/linux\$/	所有以 linux 结束的行
.	匹配一个非换行符的字符	/l...x/	匹配所有包含 l 后面 3 个字符任意，最后为 x 的行
*	匹配零或多个字符	/*linux/	匹配所有模板是一个或多个空格后紧跟 linux 的行
[]	匹配一个指定范围内的字符	[Ll]inux	匹配包含 Linux 或 linux 的行
[^]	匹配一个不再指定范围内的字符	/[^a-eg-z]tp/	匹配不再指定 a-e 及 g-z 区域内的开头并紧跟 tp 的字段

# 字符管理命令 -sed

## Sed 匹配符号

元字符	功能	例子	匹配
&	保存所搜字符用来替换其他字符	s/linux/**&*/	& 表示搜索字符串，因此 linux 将变为 **linux**
	指定单词的开始	/^<linux/	匹配包含 linux 开头的单词的行
	指定单词的结束	/linux\>/	匹配包含以 linux 结尾的单词的行
x\{m\}	重复字符 X,M 多少次	/o\{5\}/	匹配包含 5 个 o 的行
x\{m,\}	重复字符 X, 至少 M 次	/o\{5,\}\	匹配至少 5 个 o 的行

# 字符管理命令 -sed

示例：

1. 将 install.log 的第 1-3 行删除

```
$sed '1,3d' install.log
```

2. 对 install.log 查找以 A 或 a 开头且后面字符为 pache 的字串

```
$sed -n '/^[Aa]pache/p' install.log
```





## 字符管理命令 -sed

### 3. 打印匹配字段的行及所有行

```
$sed '/ftp/p' install.log
```

### 4. 打印匹配字段的行

```
$sed -n '/ftp/p' install.log
```

### 5. 默认删除第 3 行

```
$sed '3d' install.log
```



# 字符管理命令 -sed

6. 默认删除第 3 行至末尾行

```
$sed '3,$d' install.log
```

7. 删除含有指定字段的行

```
$sed '/data/d' install.log
```

8. 将 data 替换为 date

```
$sed -n 's/data/date/p' install.log
```



# 字符管理命令 -sed

9. 将行尾 noarch 为结尾后面增加 .chuai 字符

```
$sed 's/noarch$/&.chuai/' install.log
```

10. 将所有行尾 noarch 的单词替换为 chuai 单词

```
$sed -n 's/noarch$/chuai/gp' install.log
```

11. 显示包含 ra 及 data 字段的行

```
$sed -n '/ra/,/data/p' install.log
```

12. 从第 5 行开始显示直至包含有 data 的行

```
$sed -n '5,/data/p' install.log
```



# 字符管理命令 -sed

## 多点编辑使用方法

1. 删除第 1-3 行，将 data 字段改为 date 字段

```
sed -e '1,3d' -e 's/data/date/p'  
install.log
```

2. 显示指定 5,6 行并显示有 Disk 字段的行

```
$sed -ne '5,6p' -e '/Disk/p' anaconda-ks.cfg
```

# 字符管理命令 -sed

## 文件操作方法

1. 查找 tzdata 字段将 newtest 文件内容加入至此字段行下

```
$sed '/tzdata/r newtest' install.log
```

2. 将含有 tzdata 字段的行写入到 new2test 文件中

```
$sed -n '/tzdata/w new2test' install.log
```

3. 将含有 tzdata 字段的行下加入” ---Linux---” 字段

```
$sed '/tzdate/ a\---Linux---' install.log
```

# 字符管理命令 -sed

4. 将 noarch 字段 ( 小写 ) 改为 NOARCH 字段 ( 大写 )

```
$sed 'y/noarch/NOARCH/' install.log
```



# 字符管理命令 -awk

命令 :awk

功能：通过正则表达式，得到需要的行，列信息



# 字符管理命令 -awk

特点：

1. awk 是 Linux/UNIX 下用来操作数据和产生报告的程序语言
2. 数据可以来自标准输入、一个或多个文件或其他命令的输出
3. awk 逐行扫描文件，即从第一行至最后一行，寻找匹配特定模板的行，并执行“选择”动作





# 字符管理命令 -awk

Linux 下 gawk

是 Gnu 版本的 awk。

awk:

是 Alfred Aho, Peter Weinberger, Brian Kernighan 三位作者联合开发的。



# 字符管理命令 -awk

## awk 用法

\$awk 'pattern' filename

\$awk '{action}' filename

\$awk 'pattern' '{action}' filename



# 字符管理命令 -awk

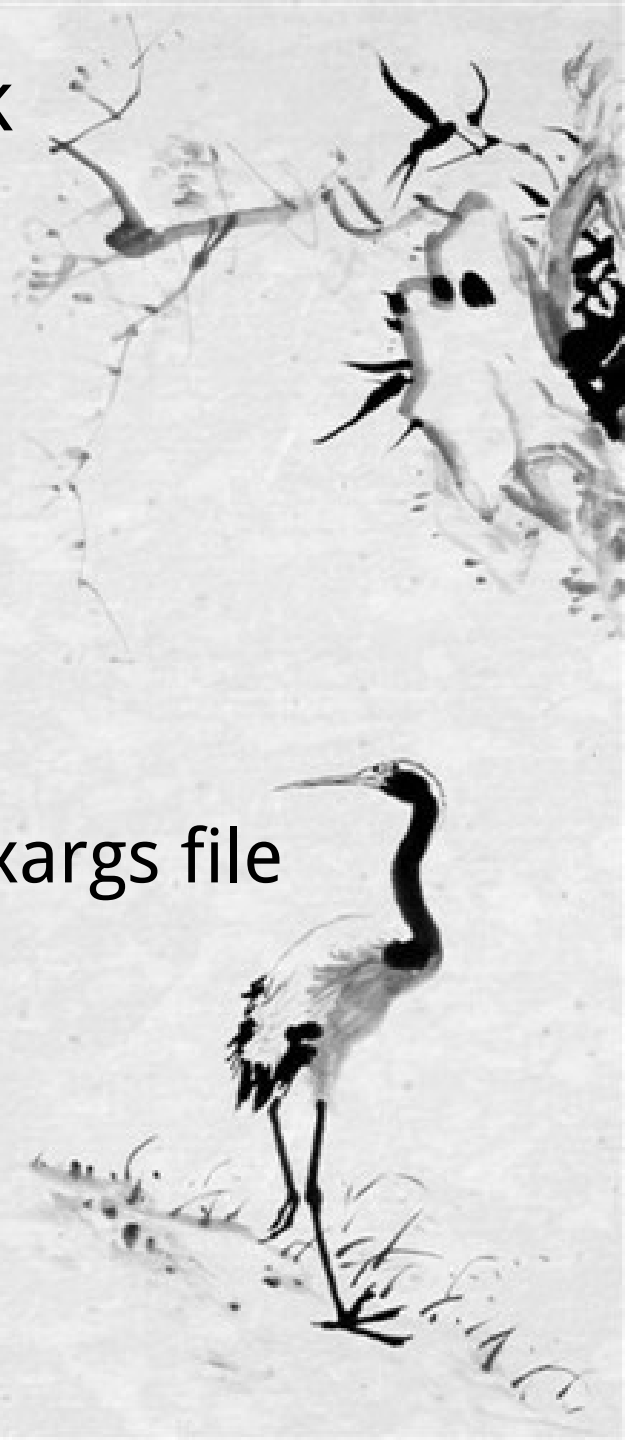
示例：

1. 查看 awk 版本

```
$awk --version
```

2. 确定 awk 命令为 gawk

```
$find /bin/ -name “awk” | xargs file
```



# 字符管理命令 -awk

示例：

1. 用 awk 打印所有包含有 data 字段的行

```
$awk '/data/' install.log
```

2. 查看 df -h 命令的第 2 列

```
$df -h | awk '{print $2}'
```

3. 查看 df -h 名 ing 的第 2,5 列

```
$df -h | awk '{print $2,$5}'
```



# 字符管理命令 -awk

示例：

4. 显示 install.log 的第四行

```
$awk 'NR==4' install.log
```

5. 打印 install.log 文件中包含 data 字段行的第二区域

```
$awk '/data/ {print $2}' install.log
```

6. 列示月份及年份 (\n 为换行符)

```
$date | awk '{print "Year:" $6  
"\nMonth:" $2}'
```



# 字符管理命令 -awk

示例：

7. 在有 /data 关键字的行的第 1 列后面增加 1 个 \t 制表符，并增加 RedHat, 第 2 列后面 ! 字符

```
$awk '/data/{print $1 "\tRedHat Linux" $2 "!" }' install.log
```

8. 在有 data 关键字的行第 1 列前面增加 1 个 \t 制表符，并增加 RedHat, 第 2 列后面 ! 字符)

```
$awk '/data/{print "\tRedHat Linux, " $1,$2 "!" }' install.log
```

# 字符管理命令 -awk

示例：

9. 在有 noarch 字段的行前增加记录号 (\$0 为行头前)

```
$awk '/noarch/{print NR, $0}' install.log
```

10. 在有 sda2 字段的行前增加行的序号及显示第 2 列内容

```
$df -h | awk '/sda2/{print NR,$2}'
```

# 字符管理命令 -awk

## 示例

11. 匹配 noarch 字段，如果有，则显示整行

```
$awk '$2 ~ /noarch/' install.log
```

12. 匹配不存在 noarch 字段的行，如果有  
则显示整行

```
$awk '$2 !~ /noarch/' install.log
```



# 字符管理命令 -sort

命令 :sort

功能：默认以排序 ASCII 方式进行排序 [a-z]

参数：

-u 去除重复的行

-r 降序排序 [z-a]

-n 数值排序，默认情况 10 比 2 小，主要因为  
sort 判断第一字符的值

-k 以文本的列进行判断

-t 设定分界符



# 字符管理命令 -sort

命令 :sort

功能：默认以排序 ASCII 方式进行排序 [a-z]

参数：

-u            去除重复的行

-r            降序排序 [z-a]

-n            数值排序，默认情况 10 比 2 小，主要因为  
sort 判断第一字符的值

-k            以文本的列进行判断

-t            设定分界符



# 字符管理命令 -sort

示例：

1. 对 /etc/passwd 文件进行升序排序

```
#sort /etc/passwd
```

2. 对 /etc/passwd 文件进行降序排序

```
#sort -r /etc/passwd
```

3. 对 /etc/passwd 第 3 列进行数值排序，分隔符为：

```
#sort -n -k 3 -t : /etc/passwd
```



# 字符管理命令 -sort

示例：

4. 对 test.txt 文件中重复的行删除并升序排序

```
#sort -u test.txt
```

5. 对 /etc/shadow 文件进行降序排序

```
#sort -r /etc/shadow
```

6. 对 /etc/passwd 第 3 列进行数值排序，分隔符为：

```
#sort -n -k 3 -t : /etc/passwd
```



# 字符管理命令 -sort

示例：

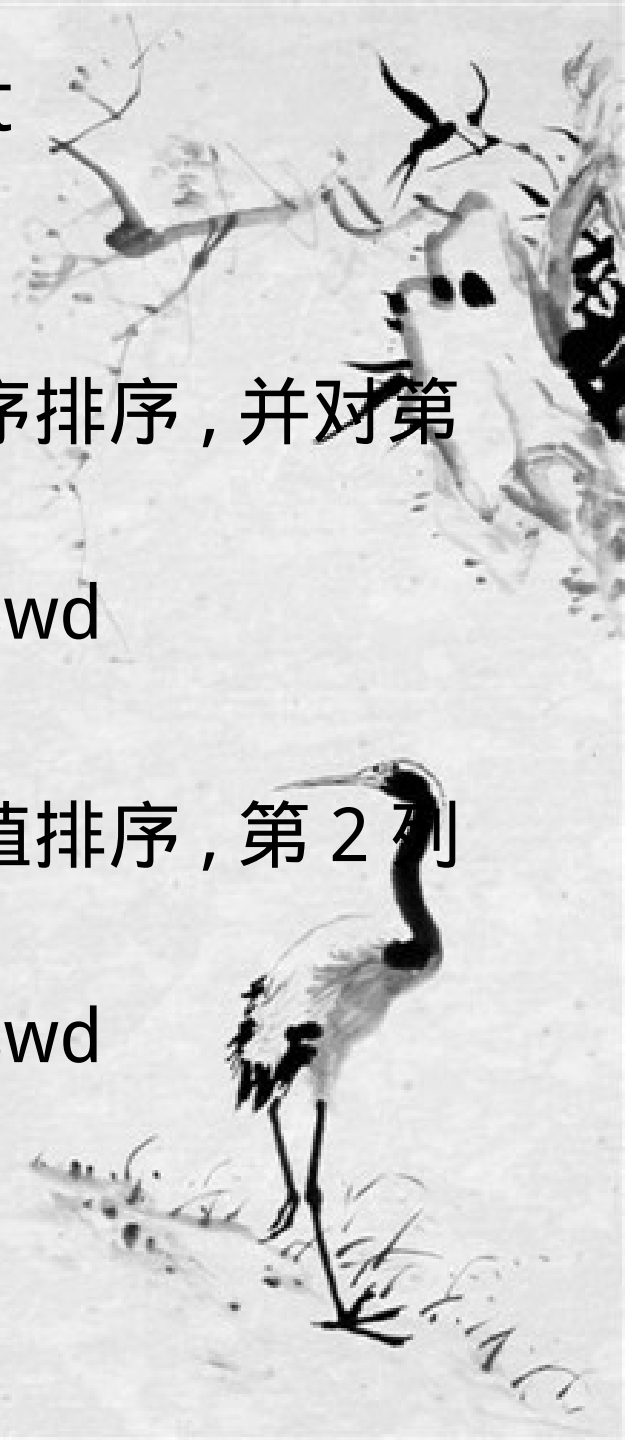
7. 对 /etc/passwd 的第 2 列进行逆序排序，并对第 3 列进行数值排序，优先级为第 2 列

```
#sort -k 2r -k 3n -t : /etc/passwd
```

8. 对 /etc/passwd 的第 3 列进行数值排序，第 2 列进行逆序排序，优先级为第 3 列

```
#sort -k 3n -k 2r -t : /etc/passwd
```

备注：哪列在前哪个为第 1 优先级



# 字符管理命令 -sort

示例：

9. 对第 1 列的第 2 个字符开始至本列最后 1 个字符排序

```
#sort -k 1.2 -t : /etc/passwd
```

10. 仅对第 1 列的第 2 个字符进行排序

```
#sort -k 1.2,1.2 -t : /etc/passwd
```

11. 对第 3 列的第 1-3 个字符进行数值排序

```
#sort -k 3,3n -t : /etc/passwd
```



# 字符管理命令 -wc

命令 :wc

功能：统计行数、字数、字符数、文件总统计数

参数：

-l 统计行数

-c 统计字节数

-w 统计字数（单词数）



# 字符管理命令 -wc

示例：

1. 统计 /etc/passwd 文件行数

```
#wc -l /etc/passwd
```

2. 统计 /etc/passwd 文件的字数

```
#wc -w /etc/passwd
```

3. 统计 /etc/passwd 文件的字节数

```
#wc -c /etc/passwd
```





# 字符管理命令 -wc

示例：

4. 统计 /etc/passwd 文件行数和字节数

```
#wc -lc /etc/passwd
```

5. 统计 /etc/passwd 与 /etc/fstab 各文件的行、字、字节数

```
#wc -lwc /etc/passwd /etc/fstab
```



# 字符管理命令 -uniq

命令 :uniq

功能：检查文本中重复出现的行

参数：

-c 显示输出，并在文本行前加出现的次数，但如果重复行不连续，则不认为是重复的行

-d 只显示重复的行

-u 只显示不重复的行



# 字符管理命令 -uniq

参数：

-f n 前 N 个字段和每个字段前的空白行一起被忽略，  
字段从 0 开始编号

-s n 前 N 个字符被忽略，字符从 0 开始编号

-w n 对 N 个字符以后的字符不在检查重复性



# 字符管理命令 -uniq

示例：

1. 对 test.txt 内容进行检查并显示次数

```
#uniq -c test.txt
```

2. 对 test.txt 的重复行不显示

```
#uniq -u test.txt
```

3. 忽略 test.txt 的第 1 列，对第 2 列进行检查

```
#uniq +1
```



# 字符管理命令 -uniq

示例：

4 对 test.txt 的显示重复行

```
#uniq -d test.txt
```

5. 忽略 test.txt 的第 1 个字符，从第 2 个字符开始进行检查

```
#uniq -s 1 test.txt
```

6. 对每行的第 2 个字符以后不在做检查

```
#uniq -w 2 -c test.txt
```



# 字符管理命令 -tee

命令 :tee

功能：读取标准输入的数据，并将其内容输出成文件

说明：指令会从标准输入设备读取数据，将其内容输出到标准输出设备，同时保存成文件



# 字符管理命令 -tee

参数：

-a: 附加到既有文件的后面，而非覆盖它。

-i: 忽略中断信号。

--help: 帮助。

--version: 显示版本信息。



# 字符管理命令 -tee

示例：

1. 查询当前账户并写入 who.txt 文件中

```
#who | tee who.txt
```

2. 将当前工作目录追加至 who.txt 文件中

```
#pwd | tee -a who.txt
```





# 字符管理命令 -tac

命令 :tac

功能：将行颠倒

说明：将最头行放置最底行，文本中所有的行均颠倒输出



# 字符管理命令 -getent

命令 :getent

功能：查询指定的数据库

语法格式 :getent [ 选项 ] [ 文件名 ] [ 关键字 ]



## 字符管理命令 -getent

1. 显示指定的数据库信息

```
#getent passwd
```

2. 显示指定的数据库信息中带有 snow 字段的条目

```
#getent passwd root
```

3. 显示 /etc/hosts 中带有 niliu 字段的条目

```
#getent hosts localhost
```



# 字符管理命令 -tr

命令 :tr

功能：大小写转换

语法格式 :tr 现有小写范围 转换大写范围  
tr 现有大写范围 转换小写范围

## 字符管理命令 -tr

示例：

1. 将全部小写转换为大写

```
#test=abcdefghijklmnopqrstuvwxyz  
#echo $test | tr a-z A-Z
```

2. 将指定小写范围转换为大写

```
#test=abcdefghijklmnopqrstuvwxyz  
#echo $test | tr a-c A-Z
```



## 字符管理命令 -tr

示例：

3. 将指定小写范围转换为大写 (A 和 B, 超过转换范围的则以 B 作为替换)

```
#test=abcdefghijklmnpqrstuvwxyz  
#echo $test | tr a-c A-B
```

4. 将指定大写范围替换为小写

```
#test=ABCDEFGHIJKLMNPOQRSTUVWXYZ  
#echo $test | tr A-Z a-z
```

# 从标准输入中建立与执行的命令

命令：xargs

功能：给命令传递参数的一个过滤器，也是组合多个命令的一个工具

特性：本身具备较强优势但与命令之间的配合非常完美强大，尤其是查找到目标后执行所希望的命令。



# 从标准输入中建立与执行的命令

示例：

1. 让不支持管道的命令实现管道支持

```
#find / -type f | xargs file
```

2. 将 .jpg 转换为 .png 格式

```
#ls *.jpg | xargs -l {} convert "{}" `echo {}  
| sed 's/jpg$/png'`
```

-l 为替换

{ } 为承接管道过滤的结果





# 从标准输入中建立与执行的命令

示例：

3. 将示例 2 的执行直接使用 CPU 中所有的 8 个核心

```
#ls *.jpg | xargs -l {} -P 8 convert "{}"  
`echo {} | sed 's/jpg$/png'`
```

-P 指定 CPU 核心数



# 从标准输入中建立与执行的命令

命令：parallel

功能：给命令传递参数的一个过滤器，也是组合多个命令的一个工具，可支持本地并发与远程并发计算

特性：本身具备较强优势但与命令之间的配合非常完美强大，尤其是查找到目标后执行所希望的命令。

注：centos7 及 EPEL 中没有此软件，需另行安装

# 从标准输入中建立与执行的命令

示例：

1. 批量修改普通文件权限

```
#find ./ -type f | parallel -m chmod 751
```

-m：启用化，如果不启用将逐条执行



# 从标准输入中建立与执行的命令

示例：

## 2. 并行批量下载

```
#cat downloads.txt
```

以下文件内容

```
#!/usr/bin/parallel
```

```
wget url1
```

```
wget url2
```

```
wget url3
```

以下为命令

```
#parallel -j+0 < downloads.txt
```



# 从标准输入中建立与执行的命令

示例：

参数说明

-j：指定至所有的 CPU 核心上执行所有任务，默认为每个 CPU 核心运行一个 jobs( 任务 )



# 从标准输入中建立与执行的命令

示例：

## 3. 对大文件的压缩

```
#cat bigfile.bin | parallel --pipe --recend "  
bzip2 -k --best > bigfile.bin.bz2
```

--pipe 从标准输入中读取一块数据，并分配给每个 jobs

--recend 记录数据结束位置

-k 按前后输入顺序进行显示



# 从标准输入中建立与执行的命令

示例：

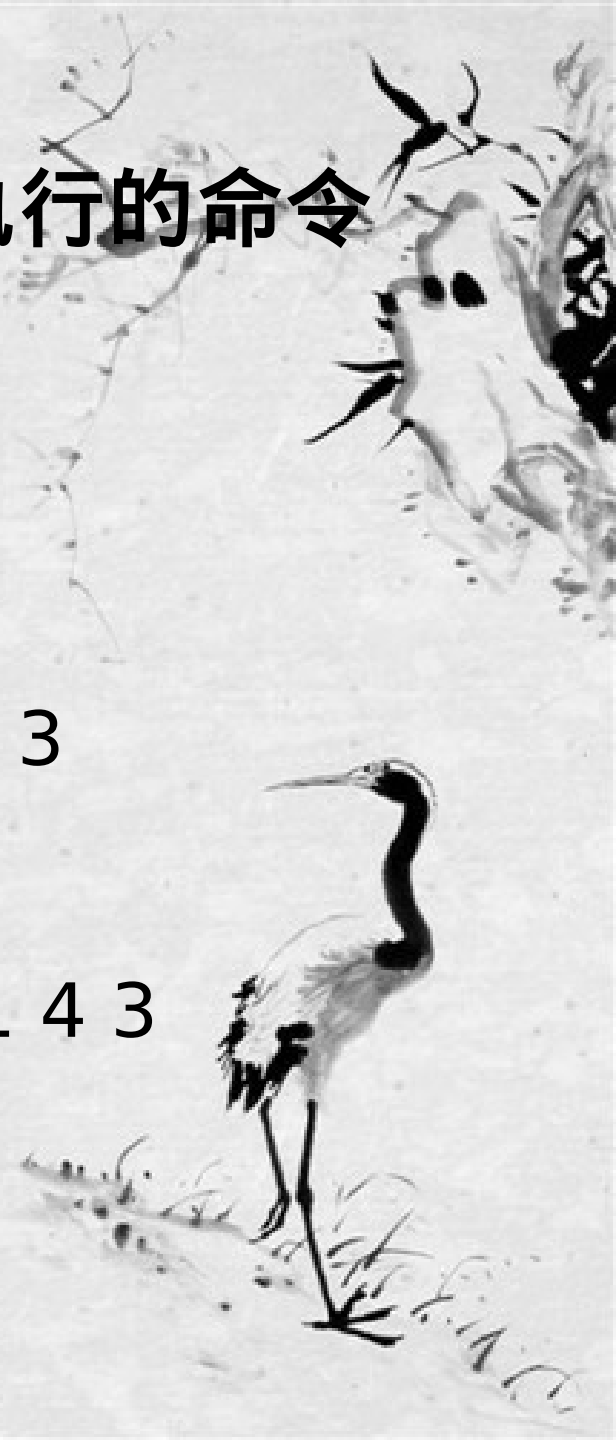
4. 对 -k 的作用

1) 无 -k

```
#parallel -j4 echo {} ::: 2 1 4 3
```

2) 有 -k

```
#parallel -j4 -k echo {} ::: 2 1 4 3
```



# 从标准输入中建立与执行的命令

示例：

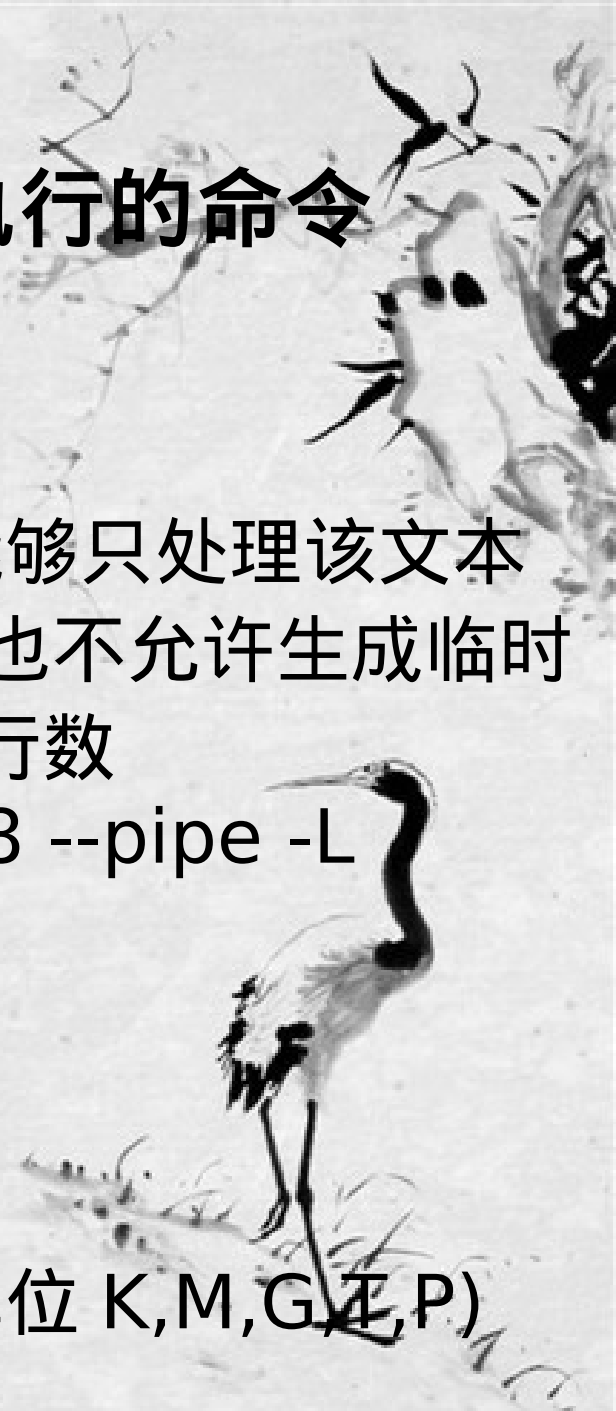
5. 如文本文件可能很大，希望每次能够只处理该文本文件的若干行，但不希望分割文件，也不允许生成临时文件。对大数据文件指定一次读取的行数

```
#cat giantfile.txt | parallel -j 8 --pipe -L  
10000 import_script
```

-j 表示并行任务 (jobs) 的数量

-L 一次读入 10000 行

--block 一次读取指定的字节 (单位 K,M,G,T,P)





# 从标准输入中建立与执行的命令

示例：

6. 将图片格式转换的任务分配到指定的计算机上，并指定每个设备的进程数

```
#ls *.jpg | parallel -l {} -S 32/niliu{1..4}  
convert "{}" `echo {} | sed 's/jpg$/png/'`
```

-S 参数：

将格式转换的任务分配到 niliu1,niliu2,niliu3,niliu4 的计算上，并且每台计算分配 32 个进程

注：前提 ssh 密钥需要配置完成