# 文本处理

曹东刚

caodg@sei.pku.edu.cn

Linux 程序设计环境

http://c.pku.edu.cn/

# 内容提要

# tr

*tr* 用于从标准输入中通过替换或删除操作进行字符转换

*tr* 命令格式

```
tr [OPTION] SET1 [SET2]
```

常用 OPTION

- -s：删除 SET1 中重复出现的字符序列
- -d：删除 SET1 中所有字符
- -c：求 SET1 的补集

# 字符范围

| | |
|---|---|
| [c1-c2] | ASCII 字符 c1 和 c2 之间的字符 |
| [c1c2] | ASCII 字符 c1 和 c2 |
| /oct | 三位 8 进制数, 对应 ASCII 字符 |
| [O*n] | 字符 O 重复出现 n 次 |

# 示例

- 去除重复出现的字符

```
tr -s "[a-z]"
```

- 删除空行

```
tr -s "\n"
```

- 大小写转换

```
tr "[a-z]" "[A-Z]"
```

# 示例 (cont)

- 将 DOS 格式 (回车\r、换行\n) 转换为 Unix 格式 (以\n 换行)

```
tr -d "\r"
```

- 找到文件中的所有单词, 每行一个输出

```
tr -cs "[a-zA-Z]" '\n'
```

# cut

*cut* 用来从标准输入或文本文件中抽取数据列或域

*cut* 命令格式

`cut [OPTION] [FILES]`

常用 OPTION

- -c LIST：指定剪切字符数. LIST 如下单表达式或由逗号分隔的表达式组合: N：第 N 个字符; N–M：第 N 到第 M 个字符; N–：从第 N 个字符到行尾
- -d delimiter-char：指定非 TAB 分隔字符
- -f LIST：指定剪切的域, LIST 形式同 -c

# 示例

- 从文件/etc/passwd 中剪切用户名及用户主目录

  passwd 文件格式:

  `ftp:x:103:65534::/home/ftp:/bin/false`

```
cut -d : -f1,6 /etc/passwd
```

- 显示当前谁在使用系统

```
who |  cut -c 1-8
```

# paste

*cut* 用来从文本文件或标准输入中抽取数据列或者域, *paste* 将按行将不同文件行信息放在一行

*paste* 命令格式

`paste [OPTION] [FILES]`

常用 OPTION

- -d delimiter-char：指定非 TAB 分隔字符
- -s：将每个文件粘贴为一行而不是逐行粘贴

# 示例

- 将文件 a.txt 和 b.txt 合并

```
paste a.txt b.txt
```

- 从管道输入，每行显示多列

  利用选项 “–”，意即对每一个 “–”，从标准输入中读一次数据。

```
ls | paste - - - -
```

# sort

- *sort* 命令将许多不同的域按不同的列顺序分类
- sort 假定工作文件已经被分过类
  - 文件是结构化的，符合一定的模式，如/etc/passwd 文件
  - 通常情况用 *awk* 将文件格式化输出后，再用 *sort* 进行排序
- 缺省情况下，*sort* 以一系列空格或 TAB 为分隔符

# 示例

- 基本排序，缺省按照第一域按字典进行分类

```
sort -t: /etc/passwd
```

- 逆向排序，对第三列按数字排序

```
sort -t: -r +2n /etc/passwd
sort -t: -r -k3n /etc/passwd
```

- 分类排序，先对 passwd 文件第三列排序，再对第七列排序

```
sort -t: -k3 -k7 /etc/passwd
```

- 将两个已分类文件合并

```
sort -m +0 patch.txt sorted.txt
```

# uniq

- *uniq* 用来从一个文本文件中去除或禁止重复行
- 一般 *uniq* 假定文件已分类，并且结果正确

*uniq* 命令格式

```
uniq [OPTION] [INPUT]
```

常用 OPTION

- -u：只显示不重复行
- -d：只显示有重复数据行，每种重复行只显示其中一行
- -c：打印每一重复行出现次数

# 统计文件中单词出现频率

- 找所有的单词

```
grep -o -P '\b[a-zA-Z]+\b'
```

- 统计

```
sort | uniq -c
```

# 统计文件中单词出现频率

- 找所有的单词

```
grep -o -P '\b[a-zA-Z]+\b'
```

- 统计

```
sort | uniq -c
```

# join

- *join* 用来将来自两个分类文本文件的行连在一起，类似 SQL 语言的 join 命令
- 文本文件中的域通常由空格或 tab 键分隔，也可以指定其他的域分隔符
- 一些系统要求使用 *join* 时文件域要少于 20；如果域大于 20，应使用 DBMS 系统
- 为有效使用 *join*，需分别将输入文件分类

# 示例: 数据文件

数据文件 name.dat

```
s00348228 1034 LiRuihang
s00348230 1036 MaJianzhu
s00348281 1035 RaoXiangrong
s00348282 1037 ChengZhiwen
```

数据文件 score.dat

```
s00348228 10   20
s00348229 8    18
s00348231 9    19
s00348282 7    20
```

# 示例

- 连接两个文件

  例: join name.dat score.dat 结果:

  s00348228 1034 LiRuihang 10 20

  s00348282 1037 ChengZhiwen 7 20

- 指定连接域

  例: join -1 1 -2 2 name.dat score.dat

- 指定输出域

  例: join -o1.1,2.2,2.3 结果:

  s00348228 10 20

  s00348282 7 20

# head 和 tail

head 和 tail 分别读取文件的前几行与后几行. 这在要读取的文件 (通常是 log) 很大 (如 200M), 要读取的内容恰好在文件的开始/结束位置时很好用.

```
tail -30 /var/log/syslog
```

# xargs 从标准输入重建并执行命令行

```
$cat t.txt
1 2 3
4 5 6
```

```
$cat t.txt | xargs echo
1 2 3 4 5 6
```

```
$cat t.txt | xargs -n 2
1 2
3 4
5 6
```

# xargs 示例

```
$echo "foo,bar,baz" | xargs -d, -L 1 echo
```

```
$ls | xargs -t -L 4 echo
```

```
$xargs -a foo -d, -L 1 echo
```

```
$find . -type f -name "*.c" -print0 | xargs -0 wc -l
```

避免 “Too Many Arguments” 问题

```
$find . -type d -name ".svn" -print | xargs rm -rf
```

# xargs 示例

```
$echo "foo,bar,baz" | xargs -d, -L 1 echo
```

```
$ls | xargs -t -L 4 echo
```

```
$xargs -a foo -d, -L 1 echo
```

```
$find . -type f -name "*.c" -print0 | xargs -0 wc -l
```

避免 “Too Many Arguments” 问题

```
$find . -type d -name ".svn" -print | xargs rm -rf
```

# xargs 示例

```
$echo "foo,bar,baz" | xargs -d, -L 1 echo
```

```
$ls | xargs -t -L 4 echo
```

```
$xargs -a foo -d, -L 1 echo
```

```
$find . -type f -name "*.c" -print0 | xargs -0 wc -l
```

避免 “Too Many Arguments” 问题

```
$find . -type d -name ".svn" -print | xargs rm -rf
```

# xargs 示例

```
$echo "foo,bar,baz" | xargs -d, -L 1 echo
```

```
$ls | xargs -t -L 4 echo
```

```
$xargs -a foo -d, -L 1 echo
```

```
$find . -type f -name "*.c" -print0 | xargs -0 wc -l
```

避免 “Too Many Arguments” 问题

```
$find . -type d -name ".svn" -print | xargs rm -rf
```

# xargs 示例

```
$echo "foo,bar,baz" | xargs -d, -L 1 echo
```

```
$ls | xargs -t -L 4 echo
```

```
$xargs -a foo -d, -L 1 echo
```

```
$find . -type f -name "*.c" -print0 | xargs -0 wc -l
```

避免 “Too Many Arguments” 问题

```
$find . -type d -name ".svn" -print | xargs rm -rf
```

# tee

tee 可以从标准输入读取数据，并数据输出到标准输出上，并输出到多个文件中.

```
ls -l | tee ls.txt lt.txt
```

# iconv 与 convmv

改变文本文件内容的编码.

```
iconv -f utf8 -t gbk
```

改变文本文件名的编码: convmv

可能需要首先探查文件编码: enca

# split

*split* 用来将大文件分割成小文件, 缺省以 1000 行为单位进行分割. 通过 -b 参数可以指定分割单位, 如 -b 50m 指定以 50M 为单位

- 例: 文件debian-installer.iso 为 110M, 现在将其分解为 60M 的小文件

  split -b 60m debian-installer.iso

  生成文件名为 xaa xab 的两个文件

- 例: 将文件 xaa xab 复原

  cat xaa xab > x.iso

# 综合示例: 统计出现频率最高的单词

```
#!/bin/sh
# 统计文本流中出现频率最高的前n个单词列表
# usage: wf [n]

tr -cs A-Za-z '\n'          |
 tr A-Z a-z                 |
   sort                     |
     uniq -c                |
       sort -k1nr -k2       |
        head -5
```

# 综合示例: 统计出现频率最高的单词

```
#!/bin/sh
# 统计文本流中出现频率最高的前n个单词列表
# usage: wf [n]

tr -cs A-Za-z '\n'      |
 tr A-Z a-z             |
   sort                 |
     uniq -c            |
       sort -k1nr -k2   |
        head -5
```

# 综合示例: 统计出现频率最高的单词

```
#!/bin/sh
# 统计文本流中出现频率最高的前n个单词列表
# usage: wf [n]

tr -cs A-Za-z '\n'       |
 tr A-Z a-z              |
   sort                  |
     uniq -c             |
       sort -k1nr -k2    |
        head -5
```

# 综合示例: 统计出现频率最高的单词

```
#!/bin/sh
# 统计文本流中出现频率最高的前n个单词列表
# usage: wf [n]

tr -cs A-Za-z '\n'       |
 tr A-Z a-z              |
   sort                  |
     uniq -c             |
       sort -k1nr -k2    |
        head -5
```

# 综合示例: 统计出现频率最高的单词

```
#!/bin/sh
# 统计文本流中出现频率最高的前n个单词列表
# usage: wf [n]

tr -cs A-Za-z '\n'       |
 tr A-Z a-z              |
   sort                  |
     uniq -c             |
       sort -k1nr -k2    |
        head -5
```

# 综合示例: 统计出现频率最高的单词

```
#!/bin/sh
# 统计文本流中出现频率最高的前n个单词列表
# usage: wf [n]

tr -cs A-Za-z '\n'       |
 tr A-Z a-z              |
   sort                  |
     uniq -c             |
       sort -k1nr -k2    |
        head -5
```

# 内容提要

# awk 和 sed 基础

*sed* 和 *awk* 是 Unix 环境中最强大的文本过滤工具, 他们有一些相似之处

- 激活语法相同:
  - `command 'script' filenames`
  - command 为 *sed* 或 *awk*, script 则是分别被 *sed* 或 *awk* 理解的命令清单
- 用户可为输入文件的每一行指定执行的指令
- 为匹配模式使用正则表达式

# 基本操作

当 *sed* 和 *awk* 命令运行时，执行如下操作

1. 从输入文件读入一行
2. 为该行做一个拷贝
3. 在该行上执行所给的脚本 script
4. 处理下一行，重复第1步

# 脚本结构及执行

*sed* 和 *awk* 指定的脚本 script 包含一行或多行记录, 格式为:

/pattern/action

脚本执行:

1. 顺序搜索每个 pattern, 直到发生一个匹配
2. 当发现匹配后, 为输入行执行相应的动作 action
3. 当动作执行完毕, 到达下一个模式, 并重复第1步
4. 当所有模式都试过后, 读取下一行

# sed

*sed* 中常用的动作

| 动作 | 描述 |
| --- | --- |
| p | 打印该行 |
| d | 删除该行 |
| s/pattern1/pattern2/ | 用 pattern2 替换 pattern1 |
| { COMMAND } | 一组命令 |
| : LABEL | 定义标号 |
| b LABEL | 跳到标号 |

# 示例: 数据文件

cat fruit.txt

| Fruit | Price/lbs |
|---|---|
| Banana | 0.89 |
| Paech | 0.79 |
| Kiwi | 1.50 |
| Pineapple | 1.29 |
| Apple | 0.99 |
| Mango | 2.20 |

# 示例

- 打印水果价格低于 1 美元的水果清单

```
sed -n '/0\.[0-9][0-9]$/p' fruit.txt
```

- 删除清单中关于芒果的信息

```
sed '/^[Mm]ango/d' fruit.txt
```

- 将清单中错误的 Paech 拼写纠正过来

```
sed 's/Paech/Peach/' fruit.txt
```

- 在清单中单价前面增加美元字符

```
sed 's/[0-9][0-9]*\.[0-9][0-9]$/\$&/' fruit.txt
```

# 示例

```
find  /home/tolstoy -type d -print |
 sed 's;/home/tolstoy/;/home/lt/;' |
  sed 's/^/mkdir /'                |
   sh -x
```

# awk

- *awk* 是一个完整的编程语言，支持按模式搜索文件并有条件的改变文件
- *awk* 可以自动将输入行分隔为域 (Field). 域是被一个或多个域分隔符分隔的字符串，缺省域分隔符为 TAB 和空格
- *awk* 有三个主要版本
  - 最初的 *awk*, 源自 1978 年的 Unix V7
  - 新版的 *nawk*, 1987 年发布为 SunOS 4.1 的一部分
  - POSIX/GNU 版的 *gawk*

# 示例: 数据文件

cat fruit.txt

| Fruit | Price/lbs | Quantity |
|---|---|---|
| Banana | $0.89 | 100 |
| Paech | $0.79 | 65 |
| Kiwi | $1.50 | 22 |
| Pineapple | $1.29 | 35 |
| Apple | $0.99 | 78 |
| Mango | $2.20 | 46 |

# 示例 -1

- 打印水果名称及数量

```
awk '{ print $1, $3 }' fruit.txt
```

格式化输出:

```
awk '{ printf "%-15s %s\n", $1, $3 }' fruit.txt
```

- 在价格高于 1 美元的水果行后加上一个"*"

```
1 #!/bin/sh
2  awk '
3     /\$[1-9][0-9]*\.[0-9][0-9]*/ { print $0, "*" }
4     /\$0\.[0-9][0-9]*/ { print }
5  ' fruit.txt
```

# 示例 –2

查找所有数量低于 70 的水果, 标记其为 “REORDER”

```
#!/bin/sh
 awk '
    $3 <= 70 { print $0, "REORDER" }
    $3 > 75 { print $0 }
 ' fruit.txt
```

# 示例 –3

查找单价高于 1 美元, 数量低于 70 的水果

```
#!/bin/sh
 awk '
    ($2 ~ /^\$[1-9][0-9]*\.[0-9][0-9]$/) && ($3 <= 70) {
        printf "%s\t%s\t%s\n", $0, "*", "REORDER"
    }
 ' fruit.txt
```

# 内容提要

# 利用多核/多机并行执行脚本

## GNU parallel

一个 Linux 下的脚本工具，用来并行执行本地/远程机器上的作业.

- 从标准输入读取, 每行输入启动一个命令或脚本执行.
  - 典型输入: 文件列表, URL 列表, 主机列表, 表格等
- 缺省每核启动一个进程
- 可以对大文件自动分块处理, 然后合并结果
- 完全可替换 `xargs`，`cat | bash`

# 替换 xargs

```
find . -type d -print | xargs echo Dir:
find . -type d -print | parallel -X echo Dir:
find . -type d -print | xargs -I {} echo Dir: {}
find . -type d -print | parallel echo Dir: {}
```

```
ls | xargs -t -L 4 echo
ls | parallel -t -L 4 echo
```

```
find . -type d -name ".svn" -print | xargs rm -rf
find . -type d -name ".svn" -print | parallel rm -rf
```

```
find . -type d -name ".svn" -print | parallel -m rm -rf
```

# 替换 xargs

```
find . -type d -print | xargs echo Dir:
find . -type d -print | parallel -X echo Dir:
find . -type d -print | xargs -I {} echo Dir: {}
find . -type d -print | parallel echo Dir: {}
```

```
ls | xargs -t -L 4 echo
ls | parallel -t -L 4 echo
```

```
find . -type d -name ".svn" -print | xargs rm -rf
find . -type d -name ".svn" -print | parallel rm -rf
```

```
find . -type d -name ".svn" -print | parallel -m rm -rf
```

# 替换 xargs

```
find . -type d -print | xargs echo Dir:
find . -type d -print | parallel -X echo Dir:
find . -type d -print | xargs -I {} echo Dir: {}
find . -type d -print | parallel echo Dir: {}
```

```
ls | xargs -t -L 4 echo
ls | parallel -t -L 4 echo
```

```
find . -type d -name ".svn" -print | xargs rm -rf
find . -type d -name ".svn" -print | parallel rm -rf
```

```
find . -type d -name ".svn" -print | parallel -m rm -rf
```

# 替换 xargs

```
find . -type d -print | xargs echo Dir:
find . -type d -print | parallel -X echo Dir:
find . -type d -print | xargs -I {} echo Dir: {}
find . -type d -print | parallel echo Dir: {}
```

```
ls | xargs -t -L 4 echo
ls | parallel -t -L 4 echo
```

```
find . -type d -name ".svn" -print | xargs rm -rf
find . -type d -name ".svn" -print | parallel rm -rf
```

```
find . -type d -name ".svn" -print | parallel -m rm -rf
```

# 从命令行读入

- 压缩所有的 html 文件

```
parallel gzip ::: *.html
```

- 将所有 wav 文件转化为 mp3 文件，每个核启动一个进程

```
parallel lame {} -o {.}.mp3 ::: *.wav
```

# 从命令行读入

- 压缩所有的 html 文件

```
parallel gzip ::: *.html
```

- 将所有 wav 文件转化为 mp3 文件，每个核启动一个进程

```
parallel lame {} -o {.}.mp3 ::: *.wav
```

# 替换

删除 pict0000.jpg 到 pict9999.jpg

逐个删除

```
seq -w 0 9999 | parallel rm pict{}.jpg
```

批量删除

```
seq -w 0 9999 | perl -pe 's/(.*)/pict$1.jpg/' |  \
    parallel -m rm
seq -w 0 9999 | parallel -X rm pict{}.jpg
```

# 替换

删除 pict0000.jpg 到 pict9999.jpg

逐个删除

```
seq -w 0 9999 | parallel rm pict{}.jpg
```

批量删除

```
seq -w 0 9999 | perl -pe 's/(.*)/pict$1.jpg/' |  \
     parallel -m rm
seq -w 0 9999 | parallel -X rm pict{}.jpg
```

# 加速

比较

```
seq -w 0 9999 | parallel touch pict{}.jpg
seq -w 0 9999 | parallel -X touch pict{}.jpg
```

如果脚本无法接受多个参数, 可以启动多个 parallel

```
seq -w 0 999999 | parallel -j10 --pipe \
    parallel -j0 touch pict{}.jpg
```

-j0 如果启动 512 个进程, 则总共启动最多 5120 个进程

# 结果收集

GNU parallel 缺省会将作业的输出收集后输出，因此只有作业结束后结果才会输出. 如果希望作业在运行时也会输出，可以使用-u 选项.

比较

```
parallel traceroute ::: sf.net debian.org
```

和

```
parallel -u traceroute ::: sf.net debian.org
```

# 使用远程主机

前提: 能够无密码 ssh 登陆远程主机, 可用 ssh-copy-id 设置

```
seq 10 | parallel --sshlogin server.example.com echo
seq 10 | parallel --sshlogin 8/server.example.com echo
seq 10 | parallel --sshlogin user1@server.example.com \
      --sshlogin user1@server2.example.net echo
```

# 传输文件

```
find logs/ -name '*.gz' | \
    parallel --sshlogin server1,server2,server3 \
        --transfer "zcat {} | bzip2 -9 >{.}.bz2"
```

```
find logs/ -name '*.gz' | \
    parallel --sshlogin server.example.com \
        --transfer --return {.}.bz2 "zcat {} | \
            bzip2 -9 >{.}.bz2"
```

```
find logs/ -name '*.gz' | \
    parallel -S server.example.com,: \
        --trc {.}.bz2 "zcat {} | bzip2 -9 >{.}.bz2"
```