

# 文本处理工具

讲师：王晓春

# 本章内容



马哥教育

IT 人的高薪职业学院

- ◆ 各种文本工具来查看、分析、统计文本
- ◆ `grep`
- ◆ `awk`
- ◆ `sed`
- ◆ 正则表达式
- ◆ 扩展正则表达式
- ◆ `vim`



# 抽取文本的工具

- ◆ 文件内容:less和 cat
- ◆ 文件截取：head和tail
- ◆ 按列抽取：cut
- ◆ 按关键字抽取：grep

## ◆ 文件查看命令：

cat , tac , rev

## ◆ cat [OPTION]... [FILE]...

-E: 显示行结束符\$

-n: 对显示出的每一行进行编号

-A : 显示所有控制符

-b : 非空行编号

-s : 压缩连续的空行成一行

## ◆ tac

## ◆ rev

# 分页查看文件内容

- ◆ more: 分页查看文件  
more [OPTIONS...] FILE...
    - d: 显示翻页及退出提示
  - ◆ less : 一页一页地查看文件或STDIN输出  
查看时有用的命令包括：
    - /文本 搜索 文本
    - n/N 跳到下一个 或 上一个匹配
- less 命令是man命令使用的分页器

# 显示文本前或后行内容



- ◆ head [OPTION]... [FILE]...
  - c #: 指定获取前#字节
  - n #: 指定获取前#行
  - # : 指定行数
- ◆ tail [OPTION]... [FILE]...
  - c #: 指定获取后#字节
  - n #: 指定获取后#行
  - # :
  - f: 跟踪显示文件fd新追加的内容,常用日志监控  
相当于 --follow=descriptor
  - F: 跟踪文件名, 相当于--follow=name --retry
- ◆ tailf 类似tail -f, 当文件不增长时并不访问文件

# 按列抽取文本cut和合并文件paste

◆ cut [OPTION]... [FILE]...

-d DELIMITER: 指明分隔符，默认tab

-f FILEDS:

#: 第#个字段

#,#[,#]: 离散的多个字段，例如1,3,6

#-#: 连续的多个字段, 例如1-6

混合使用：1-3,7

-c 按字符切割

--output-delimiter=STRING指定输出分隔符

# cut和paste



## ◆ 显示文件或STDIN数据的指定列

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

```
cat /etc/passwd | cut -d: -f7
```

```
cut -c2-5 /usr/share/dict/words
```

## ◆ paste 合并两个文件同行号的列到一行

```
paste [OPTION]... [FILE]...
```

-d 分隔符:指定分隔符，默认用TAB

-s : 所有行合成一行显示

```
paste f1 f2
```

```
paste -s f1 f2
```



# 分析文本的工具



马哥教育

IT 人的高薪职业学院

- ◆ 文本数据统计：wc
- ◆ 整理文本：sort
- ◆ 比较文件：diff和patch

# 收集文本统计数据wc

- ◆ 计数单词总数、行总数、字节总数和字符总数
- ◆ 可以对文件或STDIN中的数据运行

```
wc story.txt
```

```
39  237 1901 story.txt
```

行数 字数 字节数

## ◆ 常用选项

- -l 只计数行数
- -w 只计数单词总数
- -c 只计数字节总数
- -m 只计数字符总数
- -L 显示文件中最长行的长度



# 文本排序sort

- ◆ 把整理过的文本显示在STDOUT，不改变原始文件

sort [options] file(s)

- ◆ 常用选项

- -r 执行反方向（由上至下）整理
- -R 随机排序
- -n 执行按数字大小整理
- -f 选项忽略（fold）字符串中的字符大小写
- -u 选项（独特，unique）删除输出中的重复行
- -t c 选项使用c做为字段界定符
- -k X 选项按照使用c字符分隔的X列来整理能够使用多次

◆ uniq命令：从输入中删除前后相接的重复的行

◆ uniq [OPTION]... [FILE]...

-c: 显示每行重复出现的次数

-d: 仅显示重复过的行

-u: 仅显示不曾重复的行

注：连续且完全相同方为重复

◆ 常和sort 命令一起配合使用：

```
sort userlist.txt | uniq -c
```

## ◆ 比较两个文件之间的区别

```
diff foo.conf foo2.conf
```

```
5c5
```

```
< use_widgets = no
```

```
---
```

```
> use_widgets = yes
```

➤ 注明第5行有区别（改变）

# 复制对文件改变patch

- ◆ diff 命令的输出被保存在一种叫做“补丁”的文件中
    - 使用 -u 选项来输出“统一的 ( unified )” diff 格式文件，最适用于补丁文件
  - ◆ patch 复制在其它文件中进行的改变（要谨慎使用）
    - 适用 -b 选项来自动备份改变了的文件
- ```
$ diff -u foo.conf foo2.conf > foo.patch  
$ patch -b foo.conf foo.patch
```

- ◆ 1、找出ifconfig “网卡名” 命令结果中本机的IPv4地址
- ◆ 2、查出分区空间使用率的最大百分比值
- ◆ 3、查出用户UID最大值的用户名、UID及shell类型
- ◆ 4、查出/tmp的权限，以数字方式显示
- ◆ 5、统计当前连接本机的每个远程主机IP的连接数，并按从大到小排序

# Linux文本处理三剑客

- ◆ grep : 文本过滤(模式 : pattern)工具  
grep, egrep, fgrep ( 不支持正则表达式搜索 )
- ◆ sed : stream editor , 文本编辑工具
- ◆ awk : Linux上的实现gawk , 文本报告生成器



## ◆ grep: Global search REgular expression and Print out the line

作用：文本搜索工具，根据用户指定的“模式”对目标文本逐行进行匹配检查；打印匹配到的行

模式：由正则表达式字符及文本字符所编写的过滤条件

## ◆ grep [OPTIONS] PATTERN [FILE...]

```
grep root /etc/passwd
```

```
grep "$USER" /etc/passwd
```

```
grep '$USER' /etc/passwd
```

```
grep `whoami` /etc/passwd
```

# grep命令选项



- ◆ --color=auto: 对匹配到的文本着色显示
- ◆ -v: 显示不被pattern匹配到的行
- ◆ -i: 忽略字符大小写
- ◆ -n : 显示匹配的行号
- ◆ -c: 统计匹配的行数
- ◆ -o: 仅显示匹配到的字符串
- ◆ -q: 静默模式，不输出任何信息
- ◆ -A #: after, 后#行
- ◆ -B #: before, 前#行
- ◆ -C # : context, 前后各#行
- ◆ -e : 实现多个选项间的逻辑or关系  
        grep -e 'cat' -e 'dog' file
- ◆ -w : 匹配整个单词
- ◆ -E : 使用ERE
- ◆ -F : 相当于fgrep，不支持正则表达式

# awk



- ◆ awk介绍
- ◆ awk基本用法
- ◆ awk变量
- ◆ awk格式化
- ◆ awk操作符
- ◆ awk条件判断
- ◆ awk循环
- ◆ awk数组
- ◆ awk函数
- ◆ 调用系统命令

- ◆ awk : Aho, Weinberger, Kernighan , 报告生成器 , 格式化文本输出

- ◆ 有多种版本 : New awk ( nawk ) , GNU awk ( gawk )

- ◆ gawk : 模式扫描和处理语言

- ◆ 基本用法 :

awk [options] 'program' var=value file...

awk [options] -f programfile var=value file...

awk [options] 'BEGIN{action;... }pattern{action;... }END{action;... }' file ...

awk 程序可由 : BEGIN语句块、能够使用模式匹配的通用语句块、END语句块 , 共3部分组成

program 通常是被放在单引号中

- ◆ 选项 :

-F “分隔符”                      指明输入时用到的字段分隔符

-v var=value                      变量赋值

- ◆ 基本格式：awk [options] 'program' file...
- ◆ Program：pattern{action statements;..}
- ◆ pattern和action
  - pattern部分决定动作语句何时触发及触发事件  
BEGIN,END
  - action statements对数据进行处理，放在{}内指明  
print, printf
- ◆ 分割符、域和记录
  - awk执行时，由分隔符分隔的字段（域）标记\$1,\$2...\$n称为域标识。\$0为所有域，注意：此时和shell中变量\$符含义不同
  - 文件的每一行称为记录
  - 省略action，则默认执行 print \$0 的操作

- ◆ 第一步：执行BEGIN{action;... }语句块中的语句
- ◆ 第二步：从文件或标准输入(stdin)读取一行，然后执行pattern{ action;... }语句块，它逐行扫描文件，从第一行到最后一行重复这个过程，直到文件全部被读取完毕。
- ◆ 第三步：当读至输入流末尾时，执行END{action;...}语句块
- ◆ BEGIN语句块在awk开始从输入流中读取行之前被执行，这是一个可选的语句块，比如变量初始化、打印输出表格的表头等语句通常可以写在BEGIN语句块中
- ◆ END语句块在awk从输入流中读取完所有的行之后即被执行，比如打印所有行的分析结果这类信息汇总都是在END语句块中完成，它也是一个可选语句块
- ◆ pattern语句块中的通用命令是最重要的部分，也是可选的。如果没有提供pattern语句块，则默认执行{ print }，即打印每一个读取到的行，awk读取的每一行都会执行该语句块

◆ print格式：print item1, item2, ...

◆ 要点：

(1) 逗号分隔符

(2) 输出item可以字符串，也可是数值；当前记录的字段、变量或awk的表达式

(3) 如省略item，相当于print \$0

◆ 示例：

```
awk '{print "hello,awk"}'
```

```
awk -F: '{print}' /etc/passwd
```

```
awk -F: '{print "wang" }' /etc/passwd
```

```
awk -F: '{print $1}' /etc/passwd
```

```
awk -F: '{print $0}' /etc/passwd
```

```
awk -F: '{print $1" \t" $3}' /etc/passwd
```

```
grep "^UUID" /etc/fstab | awk '{print $2,$4}'
```

◆ 变量：内置和自定义变量

◆ FS：输入字段分隔符，默认为空白字符

```
awk -v FS=':' '{print $1,FS,$3}' /etc/passwd
```

```
awk -F: '{print $1,$3,$7}' /etc/passwd
```

◆ OFS：输出字段分隔符，默认为空白字符

```
awk -v FS=':' -v OFS=':' '{print $1,$3,$7}' /etc/passwd
```

◆ RS：输入记录分隔符，指定输入时的换行符

```
awk -v RS=' ' '{print }' /etc/passwd
```

◆ ORS：输出记录分隔符，输出时用指定符号代替换行符

```
awk -v RS=' ' -v ORS='###' '{print }' /etc/passwd
```

◆ NF：字段数量

```
awk -F: '{print NF}' /etc/fstab
```

 引用变量时，变量前不需加\$

```
awk -F: '{print $(NF-1)}' /etc/passwd
```

◆ NR：记录号

```
awk '{print NR}' /etc/fstab ; awk END '{print NR}' /etc/fstab
```



- ◆ FNR : 各文件分别计数,记录号

```
awk '{print FNR}' /etc/fstab /etc/inittab
```

- ◆ FILENAME : 当前文件名

```
awk '{print FILENAME}' /etc/fstab
```

- ◆ ARGV : 命令行参数的个数

```
awk '{print ARGV}' /etc/fstab /etc/inittab
```

```
awk 'BEGIN {print ARGV}' /etc/fstab /etc/inittab
```

- ◆ ARGV : 数组, 保存的是命令行所给定的各参数

```
awk 'BEGIN {print ARGV[0]}' /etc/fstab /etc/inittab
```

```
awk 'BEGIN {print ARGV[1]}' /etc/fstab /etc/inittab
```

## ◆ 自定义变量(区分字符大小写)

(1) -v var=value

(2) 在program中直接定义

## ◆ 示例：

```
awk -v test='hello gawk' '{print test}' /etc/fstab
```

```
awk -v test='hello gawk' 'BEGIN{print test}'
```

```
awk 'BEGIN{test="hello,gawk";print test}'
```

```
awk -F: '{sex= "male" ;print $1,sex,age;age=18}' /etc/passwd
```

```
cat awkscript
```

```
{print script,$1,$2}
```

```
awk -F: -f awkscript script= "awk" /etc/passwd
```

# printf命令



- ◆ 格式化输出：printf “FORMAT” , item1, item2, ...
  - (1) 必须指定FORMAT
  - (2) 不会自动换行，需要显式给出换行控制符，\n
  - (3) FORMAT中需要分别为后面每个item指定格式符
- ◆ 格式符：与item一一对应
  - %c：显示字符的ASCII码
  - %d, %i：显示十进制整数
  - %e, %E：显示科学计数法数值
  - %f：显示为浮点数
  - %g, %G：以科学计数法或浮点形式显示数值
  - %s：显示字符串
  - %u：无符号整数
  - %%：显示%自身
- ◆ 修饰符
  - #[.#] 第一个数字控制显示的宽度；第二个#表示小数点后精度，%3.1f
  - 左对齐（默认右对齐）%-15s
  - + 显示数值的正负符号 %+d

# printf示例



```
awk -F: '{printf "%s",$1}' /etc/passwd
```

```
awk -F: '{printf "%s\n",$1}' /etc/passwd
```

```
awk -F: '{printf "%-20s %10d\n",$1,$3}' /etc/passwd
```

```
awk -F: '{printf "Username: %s\n",$1}' /etc/passwd
```

```
awk -F: '{printf "Username: %s,UID:%d\n",$1,$3}' /etc/passwd
```

```
awk -F: '{printf "Username: %15s,UID:%d\n",$1,$3}' /etc/passwd
```

```
awk -F: '{printf "Username: %-15s,UID:%d\n",$1,$3}' /etc/passwd
```

## ◆ 算术操作符：

$x+y$ ,  $x-y$ ,  $x*y$ ,  $x/y$ ,  $x^y$ ,  $x\%y$

- x：转换为负数

+x：将字符串转换为数值

## ◆ 字符串操作符：没有符号的操作符，字符串连接

## ◆ 赋值操作符：

=, +=, -=, \*=, /=, %=, ^=, ++, --

### ➤ 下面两语句有何不同

- awk 'BEGIN{i=0;print ++i,i}'
- awk 'BEGIN{i=0;print i++,i}'

## ◆ 比较操作符：

`==, !=, >, >=, <, <=`

## ◆ 模式匹配符：

`~`：左边是否和右边匹配包含 `!~`：是否不匹配

### ➤ 示例：

```
awk -F: '$0 ~ /root/{print $1}' /etc/passwd
```

```
awk '$0~ "^root"' /etc/passwd
```

```
awk '$0 !~ /root/' /etc/passwd
```

```
awk -F: '$3==0' /etc/passwd
```

◆ 逻辑操作符：与&&，或||，非!

◆ 示例：

- `awk -F: '$3 >= 0 && $3 <= 1000 {print $1}' /etc/passwd`
- `awk -F: '$3 == 0 || $3 >= 1000 {print $1}' /etc/passwd`
- `awk -F: '!( $3 == 0 ) {print $1}' /etc/passwd`
- `awk -F: '!( $3 >= 500 ) {print $3}' /etc/passwd`

◆ 条件表达式（三目表达式）

`selector?if-true-expression:if-false-expression`

• 示例：

`awk -F: '{ $3 >= 1000 ? usertype = "Common User" : usertype = "SysUser"; printf "%15s: %-s\n", $1, usertype }' /etc/passwd`

- ◆ PATTERN:根据pattern条件，过滤匹配的行，再做处理

(1)如果未指定：空模式，匹配每一行

(2) /regular expression/：仅处理能够模式匹配到的行，需要用/ /括起来

```
awk '/^UUID/{print $1}' /etc/fstab
```

```
awk '!/^UUID/{print $1}' /etc/fstab
```

(3) relational expression: 关系表达式，结果为“真”才会被处理

真：结果为非0值，非空字符串

假：结果为空字符串或0值

- ◆ 示例：

```
awk -F: 'i=1;j=1{print i,j}' /etc/passwd
```

```
awk '!0' /etc/passwd; awk '!1' /etc/passwd
```

```
Awk -F: '$3>=1000{print $1,$3}' /etc/passwd
```

```
awk -F: '$3<1000{print $1,$3}' /etc/passwd
```

```
awk -F: '$NF==" /bin/bash"{print $1,$NF}' /etc/passwd
```

```
awk -F: '$NF ~ /bash$/{print $1,$NF}' /etc/passwd
```



## ◆ 4) line ranges : 行范围

startline,endline : /pat1/,/pat2/ 不支持直接给出数字格式

```
awk -F: '/^root\>/,/^nobody\>/{print $1}' /etc/passwd
```

```
awk -F: '(NR>=10&&NR<=20){print NR,$1}' /etc/passwd
```

## ◆ (5) BEGIN/END模式

BEGIN{}: 仅在开始处理文件中的文本之前执行一次

END{} : 仅在文本处理完成之后执行一次

# 示例

```
awk -F : 'BEGIN {print "USER USERID"} {print $1":"$3} END{print "end file"}'  
/etc/passwd
```

```
awk -F : '{print "USER USERID ";print $1":"$3} END{print "end file"}'  
/etc/passwd
```

```
awk -F: 'BEGIN{print " USER UID \n-----"}{print $1,$3}' /etc/passwd
```

```
awk -F: 'BEGIN{print " USER UID \n-----"}{print $1,$3}'END{print  
"====="} /etc/passwd
```

```
seq 10 |awk 'i=0'
```

```
seq 10 |awk 'i=1'
```

```
seq 10 | awk 'i=!i '
```

```
seq 10 | awk '{i=!i;print i} '
```

```
seq 10 | awk '!(i=!i)'
```

```
seq 10 |awk -v i=1 'i=!i'
```

## ◆ 常用的action分类

- (1) Expressions: 算术, 比较表达式等
- (2) Control statements : if, while等
- (3) Compound statements : 组合语句
- (4) input statements
- (5) output statements : print等

# awk控制语句



- ◆ { statements;... } 组合语句
- ◆ if(condition) {statements;...}
- ◆ if(condition) {statements;...} else {statements;...}
- ◆ while(conditon) {statments;...}
- ◆ do {statements;...} while(condition)
- ◆ for(expr1;expr2;expr3) {statements;...}
- ◆ break
- ◆ continue
- ◆ delete array[index]
- ◆ delete array
- ◆ exit

# awk控制语句if-else

- ◆ 语法：`if(condition){statement;...}[else statement]`  
`if(condition1){statement1}else if(condition2){statement2}else{statement3}`
- ◆ 使用场景：对awk取得的整行或某个字段做条件判断
- ◆ 示例：  
`awk -F: '{if($3>=1000)print $1,$3}' /etc/passwd`  
`awk -F: '{if($NF==" /bin/bash") print $1}' /etc/passwd`  
`awk '{if(NF>5) print $0}' /etc/fstab`  
`awk -F: '{if($3>=1000) {printf "Common user: %s\n",$1} else {printf "root or Sysuser: %s\n",$1}}' /etc/passwd`  
`awk -F: '{if($3>=1000) printf "Common user: %s\n",$1; else printf "root or Sysuser: %s\n",$1}' /etc/passwd`  
`df -h|awk -F% '/^ \dev/{print $1}|awk '$NF>=80{print $1,$5} '`  
`awk 'BEGIN{ test=100;if(test>90){print "very good "}  
else if(test>60){ print " good" }else{print "no pass" }}`

- ◆ while循环

- ◆ 语法：while(condition){statement;...}

- ◆ 条件“真”，进入循环；条件“假”，退出循环

- ◆ 使用场景：

  - 对一行内的多个字段逐一类似处理时使用

  - 对数组中的各元素逐一处理时使用

- ◆ 示例：

```
awk '/^[[:space:]]*linux16/{i=1;while(i<=NF)
    {print $i,length($i); i++}}' /etc/grub2.cfg
```

```
awk '/^[[:space:]]*linux16/{i=1;while(i<=NF) {if(length($i)>=10) {print
$i,length($i)); i++}}}' /etc/grub2.cfg
```

- ◆ do-while循环
- ◆ 语法：do {statement;...}while(condition)
- ◆ 意义：无论真假，至少执行一次循环体

◆ 示例：

```
awk 'BEGIN{ total=0;i=0;do{ total+=i;i++;}while(i<=100);print total}'
```

- ◆ for循环

- ◆ 语法：for(expr1;expr2;expr3) {statement;...}

- ◆ 常见用法：

- for(variable assignment;condition;iteration process)  
    {for-body}

- ◆ 特殊用法：能够遍历数组中的元素

- 语法：for(var in array) {for-body}

- ◆ 示例：

- awk '/^[[:space:]]\*linux16/{for(i=1;i<=NF;i++) {print \$i,length(\$i)}}'  
/etc/grub2.cfg



- ◆ `time (awk 'BEGIN{ total=0;for(i=0;i<=10000;i++){total+=i;};print total;}' )`
- ◆ `time(total=0;for i in {1..10000};do total=$((total+i));done;echo $total)`
- ◆ `time(for ((i=0;i<=10000;i++));do let total+=i;done;echo $total)`
- ◆ `time(seq -s " +" 10000|bc)`

## ◆ switch语句

◆ 语法：switch(expression) {case VALUE1 or /REGEXP/: statement1; case VALUE2 or /REGEXP2/: statement2; ...; default: statementn}

## ◆ break和continue

```
awk 'BEGIN{sum=0;for(i=1;i<=100;i++)  
    {if(i%2==0)continue;sum+=i}print sum}'
```

```
awk 'BEGIN{sum=0;for(i=1;i<=100;i++)  
    {if(i==66)break;sum+=i}print sum}'
```

- ◆ break [n]

- ◆ continue [n]

- ◆ next:

提前结束对本行处理而直接进入下一行处理（awk自身循环）

```
awk -F: '{if($3%2!=0) next; print $1,$3}' /etc/passwd
```

◆ 关联数组：array[index-expression]

◆ index-expression:

- (1) 可使用任意字符串；字符串要使用双引号括起来
- (2) 如果某数组元素事先不存在，在引用时，awk会自动创建此元素，并将其值初始化为“空串”
- (3) 若要判断数组中是否存在某元素，要使用“index in array”格式进行遍历

◆ 示例：

```
weekdays[ "mon" ]="Monday "
```

```
awk 'BEGIN{weekdays["mon"]="Monday";
```

```
    weekdays["tue"]="Tuesday";print weekdays["mon"]}'
```

```
awk '!arr[$0]++' dupfile
```

```
awk '{!arr[$0]++;print $0, arr[$0]}' dupfile
```

- ◆ 若要遍历数组中的每个元素，要使用for循环
- ◆ `for(var in array) {for-body}`
- ◆ 注意：var会遍历array的每个索引
- ◆ 示例：

```
awk 'BEGIN{weekdays["mon"]="Monday";weekdays["tue"]  
      ="Tuesday";for(i in weekdays) {print weekdays[i]}} '
```

```
netstat -tan | awk '/^tcp/{state[$NF]++}END  
      {for(i in state) { print i,state[i]}}'
```

```
awk '{ip[$1]++}END{for(i in ip) {print i,ip[i]}}' /var/log/httpd/access_log
```

## ◆ 数值处理：

rand()：返回0和1之间一个随机数

```
awk 'BEGIN{srand(); for (i=1;i<=10;i++)print int(rand()*100) }'
```

## ◆ 字符串处理：

- length([s])：返回指定字符串的长度

- sub(r,s,[t])：对t字符串搜索r表示模式匹配的内容，并将第一个匹配内容替换为s

```
echo "2008:08:08 08:08:08" | awk 'sub(/:/, "-", $1)'
```

- gsub(r,s,[t])：对t字符串进行搜索r表示的模式匹配的内容，并全部替换为s所表示的内容

```
echo "2008:08:08 08:08:08" | awk 'gsub(/:/, "-", $0)'
```

- split(s,array,[r])：以r为分隔符，切割字符串s，并将切割后的结果保存至array所表示的数组中，第一个索引值为1,第二个索引值为2,...

```
netstat -tn | awk '/^tcp\>/{split($5,ip,":");count[ip[1]]++}'
```

```
END{for (i in count) {print i,count[i]}}
```

## ◆ 自定义函数格式：

```
function name ( parameter, parameter, ... ) {  
    statements  
    return expression  
}
```

## ◆ 示例：

```
cat fun.awk  
function max(x,y) {  
    x>y?var=x:var=y  
    return var  
}  
BEGIN{a=3;b=2;print max(a,b)}  
awk -f fun.awk
```

- ◆ system命令
- ◆ 空格是awk中的字符串连接符，如果system中需要使用awk中的变量可以使用空格分隔，或者说除了awk的变量外其他一律用"引用起来

```
awk 'BEGIN'{system("hostname")}'
```

```
awk 'BEGIN{score=100; system("echo your score is " score)}'
```



◆ 将awk程序写成脚本，直接调用或执行

◆ 示例：

```
cat f1.awk
```

```
{if($3>=1000)print $1,$3}
```

```
awk -F: -f f1.awk /etc/passwd
```

```
cat f2.awk
```

```
#!/bin/awk -f
```

```
#this is a awk script
```

```
{if($3>=1000)print $1,$3}
```

```
chmod +x f2.awk
```

```
f2.awk -F: /etc/passwd
```

# 向awk脚本传递参数

## ◆ 格式：

awkfile var=value var2=value2... Inputfile

## ◆ 注意：在BEGIN过程中不可用。直到首行输入完成以后，变量才可用。可以通过-v 参数，让awk在执行BEGIN之前得到变量的值。命令行中每一个指定的变量都需要一个-v参数

## ◆ 示例：

```
cat test.awk
```

```
#!/bin/awk -f
```

```
{if($3 >=min && $3<=max)print $1,$3}
```

```
chmod +x test.awk
```

```
test.awk -F: min=100 max=200 /etc/passwd
```

- ◆ 1、文件ip\_list.txt如下格式，请提取“.magedu.com”前面的主机名部分并写入到回到该文件中
  - 1 blog.magedu.com
  - 2 www.magedu.com
  - ...
  - 999 study.magedu.com
- ◆ 2、统计/etc/fstab文件中每个文件系统类型出现的次数
- ◆ 3、统计/etc/fstab文件中每个单词出现的次数
- ◆ 4、提取出字符串Yd\$C@M05MB%9&Bdh7dq+YVixp3vpw中的所有数字
- ◆ 5、有一文件记录了1-100000之间随机的整数共5000个，存储的格式100,50,35,89...请取出其中最大和最小的整数
- ◆ 6、解决DOS攻击生产案例：根据web日志或者或者网络连接数，监控当某个IP并发连接数或者短时内PV达到100，即调用防火墙命令封掉对应的IP，监控频率每隔5分钟。防火墙命令为：iptables -A INPUT -s IP -j REJECT

- ◆ 7、将以下文件内容中FQDN取出并根据其进行计数从高到低排序

<http://mail.magedu.com/index.html>

<http://www.magedu.com/test.html>

<http://study.magedu.com/index.html>

<http://blog.magedu.com/index.html>

<http://www.magedu.com/images/logo.jpg>

<http://blog.magedu.com/20080102.html>

- ◆ 8、将以下文本以inode为标记，对inode相同的counts进行累加，并且统计出同一inode中，beginnumber的最小值和endnumber的最大值

inode|beginnumber|endnumber|counts|

106|3363120000|3363129999|10000|

106|3368560000|3368579999|20000|

310|3337000000|3337000100|101|

310|3342950000|3342959999|10000|

310|3362120960|3362120961|2|

311|3313460102|3313469999|9898|

311|3313470000|3313499999|30000|

311|3362120962|3362120963|2|

输出的结果格式为：

310|3337000000|3362120961|10103|

311|3313460102|3362120963|39900|

106|3363120000|3368579999|30000|

# Sed



马哥教育

IT 人的高薪职业学院

- ◆ Sed介绍
- ◆ Sed用法
- ◆ Sed高级用法

# 处理文本的工具sed

- ◆ Stream EEditor, 行编辑器
- ◆ sed是一种流编辑器，它一次处理一行内容。处理时，把当前处理的行存储在临时缓冲区中，称为“模式空间”（pattern space），接着用sed命令处理缓冲区中的内容，处理完成后，把缓冲区的内容送往屏幕。然后读入下行，执行下一个循环。如果没有使用诸如‘D’的特殊命令，那会在两个循环之间清空模式空间，但不会清空保留空间。这样不断重复，直到文件末尾。文件内容并没有改变，除非你使用重定向存储输出。
- ◆ 功能：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
- ◆ 参考：<http://www.gnu.org/software/sed/manual/sed.html>

## ◆ 用法:

`sed [option]... 'script' inputfile...`

## ◆ 常用选项:

-n: 不输出模式空间内容到屏幕, 即不自动打印

-e: 多点编辑

-f: /PATH/SCRIPT\_FILE: 从指定文件中读取编辑脚本

-r: 支持使用扩展正则表达式

-i.bak: 备份文件并原处编辑

## ◆ script:

'地址命令'



## ◆ 地址定界：

(1) 不给地址：对全文进行处理

(2) 单地址：

#: 指定的行, \$: 最后一行

/pattern/: 被此处模式所能够匹配到的每一行

(3) 地址范围：

#, #

#, + #

/pat1/, /pat2/

#, /pat1/

(4) ~: 步进

1~2 奇数行

2~2 偶数行

## ◆ 编辑命令：

|              |                         |
|--------------|-------------------------|
| d            | 删除模式空间匹配的行，并立即启用下一轮循环   |
| p            | 打印当前模式空间内容，追加到默认输出之后    |
| a [\]text    | 在指定行后面追加文本，支持使用\n实现多行追加 |
| i [\]text    | 在行前面插入文本                |
| c [\]text    | 替换行为单行或多行文本             |
| w /path/file | 保存模式匹配的行至指定文件           |
| r /path/file | 读取指定文件的文本至模式空间中匹配到的行后   |
| =            | 为模式空间中的行打印行号            |
| !            | 模式空间中匹配行取反处理            |

- ◆ `s///` 查找替换,支持使用其它分隔符, `s@@@`, `s###`
- ◆ 替换标记:
  - `g` 行内全局替换
  - `p` 显示替换成功的行
  - `w /PATH/FILE` 将替换成功的行保存至文件中

# sed示例



- ◆ `sed '2p' /etc/passwd`
- ◆ `sed -n '2p' /etc/passwd`
- ◆ `sed -n '1,4p' /etc/passwd`
- ◆ `sed -n '/root/p' /etc/passwd`
- ◆ `sed -n '2,/root/p' /etc/passwd` 从2行开始
- ◆ `sed -n '/^$/=' file` 显示空行行号
- ◆ `sed -n -e '/^$/p' -e '/^$/=' file`
- ◆ `sed '/root/a\superman' /etc/passwd` 行后
- ◆ `sed '/root/i\superman' /etc/passwd` 行前
- ◆ `sed '/root/c\superman' /etc/passwd` 代替行

# sed示例



- ◆ `sed '/^$/d' file`
- ◆ `sed '1,10d' file`
- ◆ `nl /etc/passwd | sed '2,5d'`
- ◆ `nl /etc/passwd | sed '2a tea'`
- ◆ `sed 's/test/mytest/g' example`
- ◆ `sed -n 's/root/&superman/p' /etc/passwd` 单词后
- ◆ `sed -n 's/root/superman&/p' /etc/passwd` 单词前
- ◆ `sed -e 's/dog/cat/' -e 's/hi/lo/' pets`
- ◆ `sed -i.bak 's/dog/cat/g' pets`

- ◆ P: 打印模式空间开端至\n内容，并追加到默认输出之前
- ◆ h: 把模式空间中的内容覆盖至保持空间中
- ◆ H: 把模式空间中的内容追加至保持空间中
- ◆ g: 从保持空间取出数据覆盖至模式空间
- ◆ G: 从保持空间取出内容追加至模式空间
- ◆ x: 把模式空间中的内容与保持空间中的内容进行互换
- ◆ n: 读取匹配到的行的下一行覆盖至模式空间
- ◆ N: 读取匹配到的行的下一行追加至模式空间
- ◆ d: 删除模式空间中的行
- ◆ D: 如果模式空间包含换行符，则删除直到第一个换行符的模式空间中的文本，并不会读取新的输入行，而使用合成的模式空间重新启动循环。如果模式空间不包含换行符，则会像发出d命令那样启动正常的新循环

# sed示例



- ◆ sed -n 'n;p' FILE
- ◆ sed '1!G;h;\$!d' FILE
- ◆ sed 'N;D' FILE
- ◆ sed '\$!N;\$!D' FILE
- ◆ sed '\$!d' FILE
- ◆ sed 'G' FILE
- ◆ sed 'g' FILE
- ◆ sed '/^\$/d;G' FILE
- ◆ sed 'n;d' FILE
- ◆ sed -n '1!G;h;\$p' FILE

- ◆ 1、删除centos7系统/etc/grub2.cfg文件中所有以空白开头的行行首的空白字符
- ◆ 2、删除/etc/fstab文件中所有以#开头，后面至少跟一个空白字符的行的行首的#和空白字符
- ◆ 3、在centos6系统/root/install.log每一行行首增加#号
- ◆ 4、在/etc/fstab文件中不以#开头的行的行首增加#号
- ◆ 5、处理/etc/fstab路径,使用sed命令取出其目录名和基名
- ◆ 6、利用sed 取出ifconfig命令中本机的IPv4地址
- ◆ 7、统计centos安装光盘中Package目录下的所有rpm文件的以.分隔倒数第二个字段的重复次数
- ◆ 8、统计/etc/init.d/functions文件中每个单词的出现次数，并排序（用grep和sed两种方法分别实现）
- ◆ 9、将文本文件的n和n+1行合并为一行，n为奇数行



- ◆ 博客: <http://mageedu.blog.51cto.com>
- ◆ 主页: <http://www.magedu.com>
- ◆ QQ: 1661815153, 113228115
- ◆ QQ群: 203585050, 279599283



# 祝大家学业有成

## 谢 谢

咨询热线 400-080-6560

- ◆ REGEXP：由一类特殊字符及文本字符所编写的模式，其中有些字符（元字符）不表示字符字面意义，而表示控制或通配的功能
- ◆ 程序支持：grep,sed,awk,vim, less,nginx,varnish等
- ◆ 分两类：
  - 基本正则表达式：BRE
  - 扩展正则表达式：ERE
  - grep -E, egrep
- ◆ 正则表达式引擎：
  - 采用不同算法，检查处理正则表达式的软件模块
  - PCRE ( Perl Compatible Regular Expressions )
- ◆ 元字符分类：字符匹配、匹配次数、位置锚定、分组
- ◆ man 7 regex

# 基本正则表达式元字符



## ◆ 字符匹配:

. 匹配任意单个字符

[] 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

[:alnum:] 字母和数字

[:alpha:] 代表任何英文大小写字符，亦即 A-Z, a-z

[:lower:] 小写字母    [:upper:] 大写字母

[:blank:] 空白字符（空格和制表符）

[:space:] 水平和垂直的空白字符（比[:blank:]包含的范围广）

[:cntrl:] 不可打印的控制字符（退格、删除、警铃...）

[:digit:] 十进制数字    [:xdigit:] 十六进制数字

[:graph:] 可打印的非空白字符

[:print:] 可打印字符

[:punct:] 标点符号

◆ 匹配次数：用在要指定次数的字符后面，用于指定前面的字符要出现的次数

\* 匹配前面的字符任意次，包括0次

贪婪模式：尽可能长的匹配

. \* 任意长度的任意字符

\? 匹配其前面的字符0或1次

\+ 匹配其前面的字符至少1次

\{n\} 匹配前面的字符n次

\{m,n\} 匹配前面的字符至少m次，至多n次

\{,n\} 匹配前面的字符至多n次

\{n,\} 匹配前面的字符至少n次

## ◆ 位置锚定：定位出现的位置

^ 行首锚定，用于模式的最左侧

\$ 行尾锚定，用于模式的最右侧

^PATTERN\$ 用于模式匹配整行

^\$ 空行

^[[:space:]]\*\$ 空白行

\< 或 \b 词首锚定，用于单词模式的左侧

\> 或 \b 词尾锚定；用于单词模式的右侧

\<PATTERN\> 匹配整个单词

- ◆ 分组：\(\) 将一个或多个字符捆绑在一起，当作一个整体进行处理，如：  
    \(\root\)\+
- ◆ 分组括号中的模式匹配到的内容会被正则表达式引擎记录于内部的变量中，这些变量的命名方式为：\1, \2, \3, ...
- ◆ \1 表示从左侧起第一个左括号以及与之匹配右括号之间的模式所匹配到的字符
- ◆ 示例： \(\string1\+\(\string2\)\*\)  
    \1 : string1\+\(\string2\)\*  
    \2 : string2
- ◆ 后向引用：引用前面的分组括号中的模式所匹配字符，而非模式本身
- ◆ 或者：\|  
    示例：a\|b: a或b    C\|cat: C或cat    \(\C\|c\)at: Cat或cat

# 正则表达式

| 元字符                  | 定义                              |
|----------------------|---------------------------------|
| <code>^</code>       | 行首                              |
| <code>\$</code>      | 行尾                              |
| <code>.</code>       | 任意单一字符                          |
| <code>[]</code>      | <code>[]</code> 内任意单一字符         |
| <code>[^]</code>     | 除 <code>[]</code> 内任意单一字符       |
| <code>*</code>       | <code>*</code> 前面字符重复不确定次数      |
| <code>\+</code>      | <code>\+</code> 前面字符重复一次以上不确定次数 |
| <code>\?</code>      | <code>\?</code> 前面字符重复0或1次      |
| <code>\</code>       | 转义符                             |
| <code>.*</code>      | 任意长度字符                          |
| <code>\{n\}</code>   | 前面字符重复n次                        |
| <code>\{n,\}</code>  | 前面字符重复n次以上                      |
| <code>\{m,n\}</code> | 前面字符重复m次和n次之间                   |



# 正则表达式

| 元字符       | 定义                            |
|-----------|-------------------------------|
| [alnum:]  | 字母和数字                         |
| [alpha:]  | 代表任何英文大小写字符，亦即 A-Z, a-z       |
| [lower:]  | 小写字母                          |
| [upper:]  | 大写字母                          |
| [blank:]  | 水平空白字符（空格和制表符）                |
| [space:]  | 所有水平和垂直的空白字符（比[blank:]包含的范围广） |
| [cntrl:]  | 不可打印的控制字符（退格、删除、警铃...）        |
| [digit:]  | 十进制数字                         |
| [graph:]  | 可打印的非空白字符                     |
| [print:]  | 可打印字符                         |
| [punct:]  | 标点符号                          |
| [xdigit:] | 十六进制数字                        |
|           |                               |

- ◆ 1、显示/proc/meminfo文件中以大小s开头的行(要求：使用两种方法)
- ◆ 2、显示/etc/passwd文件中不以/bin/bash结尾的行
- ◆ 3、显示用户rpc默认的shell程序
- ◆ 4、找出/etc/passwd中的两位或三位数
- ◆ 5、显示CentOS7的/etc/grub2.cfg文件中，至少以一个空白字符开头的且后面有非空白字符的行
- ◆ 6、找出“netstat -tan”命令结果中以LISTEN后跟任意多个空白字符结尾的行
- ◆ 7、显示CentOS7上所有系统用户的用户名和UID
- ◆ 8、添加用户bash、testbash、basher、sh、nologin(其shell为/sbin/nologin),找出/etc/passwd用户名和shell同名的行
- ◆ 9、利用df和grep，取出磁盘各分区利用率，并从大到小排序

# egrep及扩展的正则表达式



- ◆ `egrep = grep -E`
- ◆ `egrep [OPTIONS] PATTERN [FILE...]`
- ◆ 扩展正则表达式的元字符：
- ◆ 字符匹配：
  - . 任意单个字符
  - [] 指定范围的字符
  - [^] 不在指定范围的字符

# 扩展正则表达式



## ◆ 次数匹配：

\*：匹配前面字符任意次

?: 0或1次

+：1次或多次

{m}：匹配m次

{m,n}：至少m，至多n次

# 扩展正则表达式



## ◆ 位置锚定：

^ :行首

\$ :行尾

\<, \b :语首

\>, \b :语尾

## ◆ 分组：

()

后向引用：\1, \2, ...

## ◆ 或者：

a|b: a或b

C|cat: C或cat

(C|c)at: Cat或cat

- ◆ 1、显示三个用户root、mage、wang的UID和默认shell
- ◆ 2、找出/etc/rc.d/init.d/functions文件中行首为某单词(包括下划线)后面跟一个小括号的行
- ◆ 3、使用egrep取出/etc/rc.d/init.d/functions中其基名
- ◆ 4、使用egrep取出上面路径的目录名
- ◆ 5、统计last命令中以root登录的每个主机IP地址登录次数
- ◆ 6、利用扩展正则表达式分别表示0-9、10-99、100-199、200-249、250-255
- ◆ 7、显示ifconfig命令结果中所有IPv4地址
- ◆ 8、将此字符串：welcome to magedu linux 中的每个字符去重并排序，重复次数多的排到前面

# vim简介



- ◆ vi: Visual Interface , 文本编辑器
- ◆ 文本 : ASCII, Unicode
- ◆ 文本编辑种类 :
  - 行编辑器: sed
  - 全屏编辑器 : nano, vi
  - vim - Vi Improved
- ◆ 其他编辑器 :
  - gedit 一个简单的图形编辑器
  - gvim 一个Vim编辑器的图形版本

## ◆ # vim [OPTION]... FILE...

+ #: 打开文件后，让光标处于第#行的行首，+默认行尾

+ /PATTERN：打开文件后，直接让光标处于第一个被PATTERN匹配到的行的行首

-b file 二进制方式打开文件

-d file1 file2... 比较多个文件

-m file 只读打开文件

## ◆ ex file 或 vim -e 直接进入ex模式

## ◆ 如果该文件存在，文件被打开并显示内容

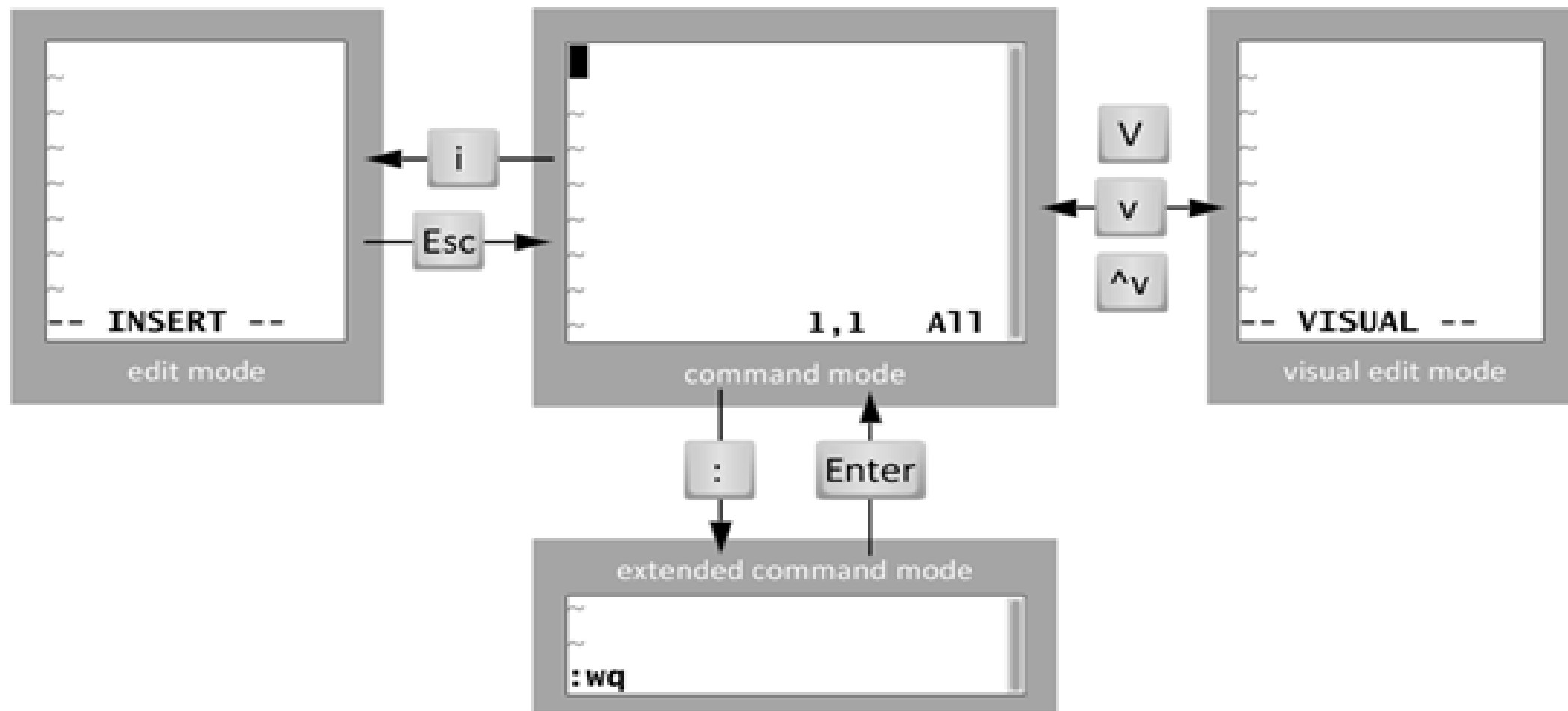
如果该文件不存在，当编辑后第一次存盘时创建它



# vim：一个模式编辑器

- ◆ 击键行为是依赖于 vim 的 “模式”
- ◆ 三种主要模式：
  - 命令(Normal)模式:默认模式，移动光标，剪切/粘贴文本
  - 插入(Insert)或编辑模式：修改文本
  - 扩展命令(extended command)模式：保存，退出等
- ◆ Esc键 退出当前模式
- ◆ Esc键 Esc键 总是返回到命令模式

# vim模式



## ◆ 命令模式 --> 插入模式

i: insert, 在光标所在处输入

I : 在当前光标所在行的行首输入

a: append, 在光标所在处后面输入

A : 在当前光标所在行的行尾输入

o: 在当前光标所在行的下方打开一个新行

O : 在当前光标所在行的上方打开一个新行

# 模式转换



◆ 插入模式 -----> 命令模式

ESC

◆ 命令模式 -----> 扩展命令模式

:

◆ 扩展命令模式 -----> 命令模式

ESC,enter

# 关闭文件



## ◆ 扩展模式：

:q 退出

:q! 强制退出，丢弃做出的修改

:wq 保存退出

:x 保存退出

## ◆ 命令模式

ZZ: 保存退出

ZQ：不保存退出

- ◆ 按 “:” 进入Ex模式
- ◆ 创建一个命令提示符： 处于底部的屏幕左侧
- ◆ 命令：
  - w      写（存）磁盘文件
  - wq    写入并退出
  - x      写入并退出
  - q      退出
  - q !    不存盘退出，即使更改都将丢失
  - r filename    读文件内容到当前文件中
  - w filename    将当前文件内容写入另一个文件
  - !command    执行命令
  - r!command    读入命令的输出

## ◆ 字符间跳转：

h: 左 l: 右 j: 下 k: 上

#COMMAND：跳转由#指定的个数的字符

## ◆ 单词间跳转：

w：下一个单词的词首

e：当前或下一单词的词尾

b：当前或前一个单词的词首

#COMMAND：由#指定一次跳转的单词数

## ◆ 当前页跳转：

H：页首 M：页中间行 L:页底

## ◆ 行首行尾跳转：

^: 跳转至行首的第一个非空白字符

0: 跳转至行首

\$: 跳转至行尾

## ◆ 行间移动：

#G、扩展模式：# ：跳转至由#指定行

G：最后一行

1G, gg: 第一行

## ◆ 句间移动：

)：下一句      (：上一句

## ◆ 段落间移动：

}：下一段      {：上一段



# 命令模式翻页操作



马哥教育

IT 人的高薪职业学院

- ◆ Ctrl+f: 向文件尾部翻一屏
- ◆ Ctrl+b: 向文件首部翻一屏
- ◆ Ctrl+d: 向文件尾部翻半屏
- ◆ Ctrl+u : 向文件首部翻半屏

## ◆ 字符编辑：

x: 删除光标处的字符

#x: 删除光标处起始的#个字符

xp: 交换光标所在处的字符及其后面字符的位置

~: 转换大小写

J: 删除当前行后的换行符

## ◆ 替换命令(r, replace)

r: 替换光标所在处的字符

R: 切换到REPLACE模式

## ◆ 删除命令：

d: 删除命令，可结合光标跳转字符，实现范围删除

d\$: 删除到行尾

d^: 删除到非空行首

d0: 删除到行首

dw:

de:

db:

#COMMAND

## ◆ dd: 删除光标所在的行

#dd: 多行删除

## ◆ D: 从当前光标位置一直删除到行尾，留空行，等同于d\$

# 命令模式操作



## ◆ 复制命令(y, yank) :

y: 复制，行为类似于d命令

y\$

y0

y^

ye

yw

yb

#COMMAND

## ◆ yy : 复制行

#yy: 复制多行

## ◆ Y: 复制整行

## ◆ 粘贴命令(p, paste) :

p : 缓冲区存的如果为整行 , 则粘贴当前光标所在行的下方 ; 否则 , 则粘贴至当前光标所在处的后面

P : 缓冲区存的如果为整行 , 则粘贴当前光标所在行的上方 ; 否则 , 则粘贴至当前光标所在处的前面

- ◆ 改变命令(c, change)

- c: 修改后切换成插入模式

- ◆ 命令模式 --> 插入模式

- c\$

- c^

- c0

- cb

- ce

- cw

- #COMMAND

- ◆ cc : 删除当前行并输入新内容，相当于S

- #cc:

- ◆ C : 删除当前光标到行尾，并切换成插入模式

# 命令模式：操作文本

|                 | Change (replace) | Delete (cut) | Yank (copy) |
|-----------------|------------------|--------------|-------------|
| Line            | cc               | dd           | yy          |
| Letter          | cl               | dl           | yl          |
| Word            | cw               | dw           | yw          |
| Sentence ahead  | c)               | d)           | y)          |
| Sentence behind | c(               | d(           | y(          |
| Paragraph above | c{               | d{           | y{          |
| Paragraph below | c}               | d}           | y}          |

- ◆ 100iwang [ESC] 粘贴 “wang” 100次
  - ◆ <start position> <command> <end position>
  - ◆ Command:
    - y 复制、d 删除、gU 变大写、gu 变小写
- 例如 0y\$ 命令意味着：
- 0 → 先到行头
  - y → 从这里开始拷贝
  - \$ → 拷贝到本行最后一个字符
- ye 从当前位置拷贝到本单词的最后一个字符



## ◆ 地址定界

:start\_pos,end\_pos

# 具体第#行，例如2表示第2行

#,# 从左侧#表示起始行，到右侧#表示结尾行

#,+# 从左侧#表示的起始行，加上右侧#表示的行数

      : 2,+3 表示2到5行

. 当前行

\$ 最后一行

.,\$-1 当前行到倒数第二行

% 全文, 相当于1,\$

## ◆ /pat1/,/pat2/

从第一次被pat1模式匹配到的行开始，一直到第一次被pat2匹配到的行结束

#,/pat/

/pat/, \$

## ◆ 使用方式：后跟一个编辑命令

d

y

w file: 将范围内的行另存至指定文件中

r file：在指定位置插入指定文件中的所有内容

## ◆ 查找

/PATTERN：从当前光标所在处向文件尾部查找

?PATTERN：从当前光标所在处向文件首部查找

n：与命令同方向

N：与命令反方向

# 扩展命令模式：查找并替换

## ◆ s: 在扩展模式下完成查找替换操作

格式：s/要查找的内容/替换为的内容/修饰符

要查找的内容：可使用模式

替换为的内容：不能使用模式，但可以使用\1, \2, ...等后向引用符号；还可以使用 "&" 引用前面查找时查找到的整个内容

修饰符：

i: 忽略大小写

g: 全局替换；默认情况下，每一行只替换第一次出现

gc: 全局替换，每次替换前询问

## ◆ 查找替换中的分隔符/可替换为其它字符，例如

s@/etc@/var@g

s#/boot#/#i

- ◆ 1、复制/etc/profile至/tmp/目录，用查找替换命令删除/tmp/profile文件中的行首的空白字符
- ◆ 2、复制/etc/rc.d/init.d/functions文件至/tmp目录，用查找替换命令为/tmp/functions的每行开头为空白字符的行的行首添加一个#号

# 命令模式：撤消更改



- ◆ u撤销最近的更改
- ◆ #u撤销之前多次更改
- ◆ U 撤消光标落在这行后所有此行的更改
- ◆ 按Ctrl - r重做最后的“撤消”更改
- ◆ . 重复前一个操作
- ◆ n.重复前一个操作n次

- ◆ 有26个命名寄存器和1个无命名寄存器，常存放不同的剪贴版内容，可以不同会话间共享
- ◆ 寄存器名称a, b,...,z,格式：“寄存器 放在数字和命令之间  
如：3"t yy 表示复制3行到t寄存器中  
"tp 表示将t寄存器内容粘贴
- ◆ 未指定，将使用无命名寄存器
- ◆ 有10个数字寄存器，用0, 1, ..., 9表示，0存放最近复制内容，1存放最近删除内容。当新的文本变更和删除时，1转存到2，2转存到3，以此类推。数字寄存器不能在不同会话间共享

- ◆ 以二进制方式打开文件

`vim -b binaryfile`

- ◆ 扩展命令模式下，利用xxd命令转换为可读的十六进制

`:%!xxd`

- ◆ 编辑二进制文件

- ◆ 扩展命令模式下，利用xxd命令转换回二进制

`:%!xxd -r`

- ◆ 保存退出



- ◆ 允许选择的文本块
  - ✓ 面向字符
  - ✓ 面向行
  - ctrl-v 面向块
- ◆ 可视化键可用于与移动键结合使用：
  - w ) } 箭头等
- ◆ 突出显示的文字可被删除，复制，变更，过滤，搜索，替换等

# 多文件模式



◆ vim FILE1 FILE2 FILE3 ...

:next 下一个

:prev 前一个

:first 第一个

:last 最后一个

:wall 保存所有

:qall 退出所有

:wqall

# 使用多个“窗口”



## ◆ 多文件分割

`vim -o|-O FILE1 FILE2 ...`

-o: 水平分割

-O: 垂直分割

在窗口间切换 : `Ctrl+w, Arrow`

## ◆ 单文件窗口分割 :

`Ctrl+w,s`: split, 水平分割

`Ctrl+w,v`: vertical, 垂直分割

`ctrl+w,q` : 取消相邻窗口

`ctrl+w,o`: 取消全部窗口

: `wqall` 退出

# 定制vim的工作特性



- ◆ 配置文件：永久有效

  - 全局：/etc/vimrc

  - 个人：~/.vimrc

- ◆ 扩展模式：当前vim进程有效

- ◆ (1) 行号

  - 显示：set number, 简写为set nu

  - 取消显示：set nonumber, 简写为set nonu

- ◆ (2) 忽略字符的大小写

  - 启用：set ic

  - 不忽略：set noic

- ◆ (3) 自动缩进

  - 启用：set ai

  - 禁用：set noai

# 定制vim的工作特性

## ◆ (4) 智能缩进

启用：smartindent 简写 set si

禁用：set nosi

## ◆ (5) 高亮搜索

启用：set hlsearch

禁用：set nohlsearch

## ◆ (6) 语法高亮

启用：syntax on

禁用：syntax off

## ◆ (7) 显示Tab和换行符 ^I 和\$显示

启用：set list

禁用：set nolist

# 定制vim的工作特性

## ◆ (8) 文件格式

启用windows格式 : set fileformat=dos

启用unix格式 : set fileformat=unix

简写 : set ff=dos|unix

## ◆ (9) 设置文本宽度

启用: set textwidth=65 (vim only)

禁用: set wrapmargin=15

## ◆ (10) 设置光标所在行的标识线

启用: set cursorline , 简写cul

禁用: set no cursorline

## ◆ (11) 复制保留格式

启用: set paste

禁用: set nopaste

# 了解更多



## ◆ Set 帮助

➤ :help option-list

➤ :set or :set all

## ◆ vi/vim内置帮助

:help

:help *topic*

Use :q to exit help

## ◆ vimtutor

# 练习

- ◆ 1、在vim中设置tab缩进为4个字符
- ◆ 2、复制/etc/rc.d/init.d/functions文件至/tmp目录，替换/tmp/functions文件中的/etc/sysconfig/init为/var/log
- ◆ 3、删除/tmp/functions文件中所有以#开头，且#后面至少有一个空白字符的行的行首的#号



# 关于马哥教育



马哥教育

IT 人的高薪职业学院

- ◆ 博客 : <http://mageedu.blog.51cto.com>
- ◆ 主页 : <http://www.magedu.com>
- ◆ QQ : 1661815153, 113228115
- ◆ QQ群 : 203585050, 279599283



# 祝大家学业有成

## 谢 谢

咨询热线 400-080-6560