Python开发之运维基础

讲师：王晓春

本章内容

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

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

- 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

- 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

- 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
```

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

- 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

正则表达式

- 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

正则表达式

- 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

正则表达式

- 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

正则表达式

- 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

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

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 —个简单的图形编辑器

  gvim —个Vim编辑器的图形版本

打开文件

- # 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

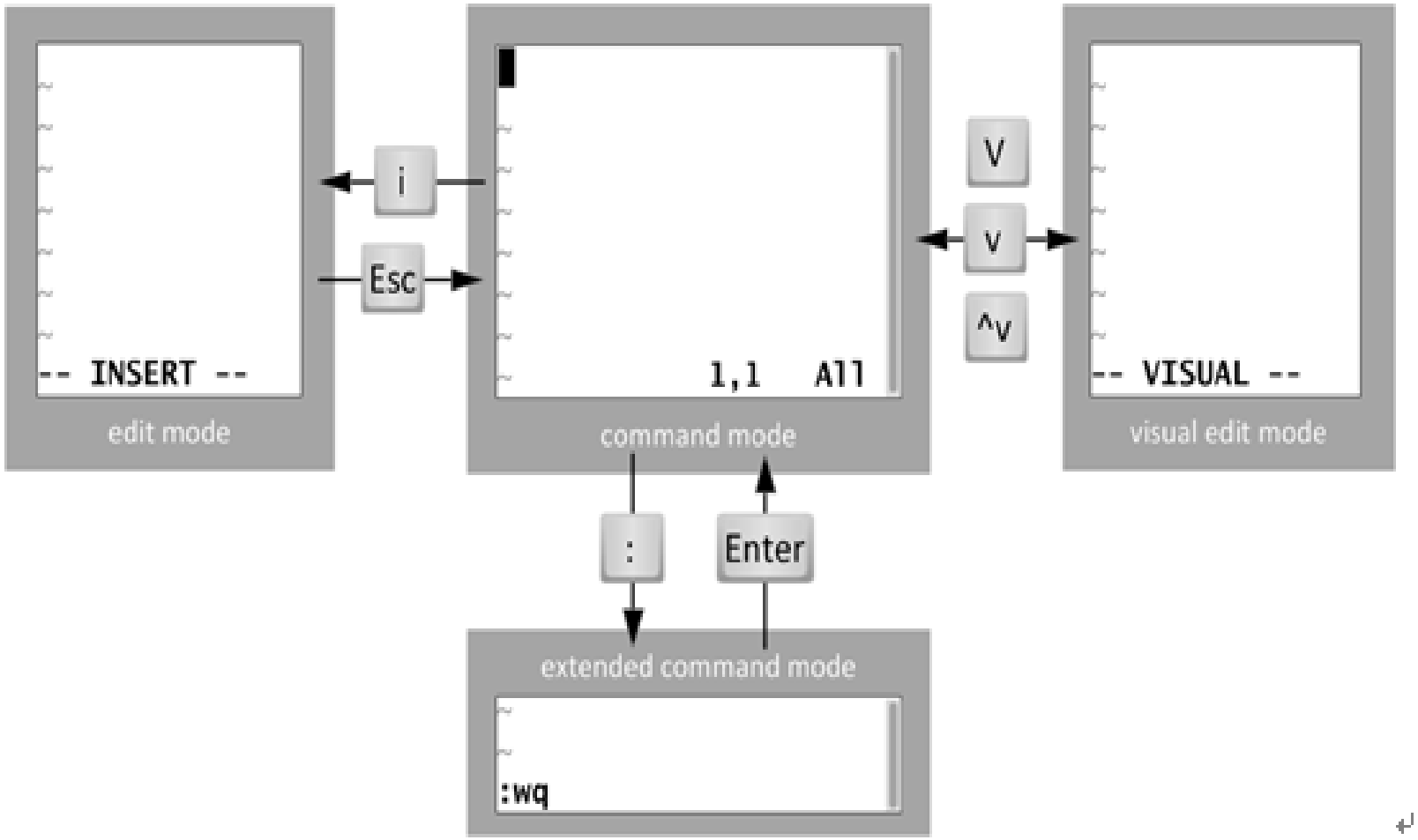
- ex file 或 vim –e 直接进入ex模式
- 如果该文件存在，文件被打开并显示内容

  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 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


-- INSERT --
edit mode
i
Esc
1,1 All
command mode
V
v
^v
-- VISUAL --
visual edit mode
:
Enter
extended command mode
:wq

模式转换

- 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

扩展模式

- 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

命令模式光标跳转

- 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

命令模式光标跳转

- 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

命令模式操作

- 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:切换成REPLACE模式

命令模式操作

- 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，行为相似于d命令
  - y$
  - y0
  - y^
  - ye
  - yw
  - yb
  - #COMMAND
- yy：复制行
  - #yy: 复制多行
- Y: 复制整行

命令模式操作

- 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

命令模式操作

- 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

扩展命令模式：地址定界

- 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,$

扩展命令模式：地址定界

- /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

扩展命令模式：查找

- 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

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

可视化模式

- 允许选择的文本块
  - v 面向字符
  - V 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

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

配置 vi and vim

- (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

处理文本的工具sed

- Stream EDitor, 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

sed工具

- 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'

sed工具

- 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

sed工具

- 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/somefile: 保存模式匹配的行至指定文件

  r /path/somefile：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

sed工具

- 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/TO/SOMEFILE：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

高级编辑命令

- 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

awk介绍

- 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 程序通常由：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通常是被单引号或双引号中
- 选项：

  -F 指明输入时用到的字段分隔符

  -v var=value: 自定义变量

awk语言

- 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。$0为所有域，注意：和shell中变量$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

awk工作原理

- 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

awk

- print格式：print item1, item2, ...
- 要点：
  - (1) 逗号分隔符
  - (2) 输出的各item可以字符串，也可以是数值；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
 tail –3    /etc/fstab |awk ‘{print $2,$4}’
```

awk变量

- 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,引用内置变量不用$
awk -F: '{print $(NF-1)}' /etc/passwd
```

- NR：记录号

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

awk

- 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

- ARGC：命令行参数的个数

  awk '{print ARGC}' /etc/fstab /etc/inittab

  awk ‘BEGIN {print ARGC}’ /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

awk变量

- 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

操作符

- 算术操作符：

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

  -x: 转换为负数

  +x: 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}’

操作符

- 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

操作符

- 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
- 函数调用： function_name(argu1, argu2, ...)
- 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="Sysadmin or SysUser";printf "%15s:%-s\n",$1,usertype}' /etc/passwd

awk PATTERN

◆ 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

awk PATTERN

- 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'

awk action

◆ 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"}}'
```

awk控制语句

- 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
```

awk控制语句

- 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}  '
```

awk控制语句

- 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
```

awk控制语句

- 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

awk数组

- 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

awk函数

- 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字符串进行搜索r表示的模式匹配的内容，并将第一个匹配的内容替换为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 -tan | awk '/^tcp\>/{split($5,ip,":");count[ip[1]]++}

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

祝大家学业有成

谢　谢