训练稿——Linux常用重点命令

目录

[简介 1](#_Toc322890023)

[命令格式&说明 2](#_Toc322890024)

[调用方法 3](#_Toc322890025)

[域切分&操作 3](#_Toc322890026)

[正则匹配 4](#_Toc322890027)

[内置变量&操作 5](#_Toc322890028)

[内置函数&操作 6](#_Toc322890029)

[格式化输出 7](#_Toc322890030)

[数组 8](#_Toc322890031)

[条件判断&流程控制 9](#_Toc322890032)

[与shell交互 10](#_Toc322890033)

[awk与变量 10](#_Toc322890034)

[错误排查 11](#_Toc322890035)

[其他 11](#_Toc322890036)

## 简介

|  |
| --- |
| AWK: 三位创造者Aho、Weinberger和Kernighan统称 |
| 官方定义：一个优秀的样式扫描与处理工具  定位：AWK 是一种用于处理文本的编程语言工具，主要用于格式化报文或从一个大的文本文件中抽取数据。提取原来文本中行的某些域进行计算组合显示，出结果  【扫描文件中的每一行，查找与命令行中所给定内容相匹配的模式。如果发现匹配内容，则进行下一个编程步骤。如果找不到匹配内容，则继续处理下一行, 直到文件末尾】  [Shell过滤工具中最难掌握的，语法复杂，错误提示不明确]  优点:  1.处理文件中的记录（与数据库相比）  2.简单，解释执行，不必编译（与C相比）  3.容易获得，unix自带（whereis awk） |
| 模式和动作  模式部分决定了动作语句何时触发及触发的事件【处理语句】  模式部分可省略，则动作对每一行执行；模式部分可以是任何条件语句、复合语句、正则表达式 |

## 命令格式&说明

|  |
| --- |
| 格式： |
| 语法组成：【查找 -🡪 执行处理】  awk ‘pattern{action}’ filename  【注意,pattern在非括号中，用于查询筛选匹配行，action在括号中，action 是在找到匹配内容时所执行的一系列命令，用于对筛选后的内容进行处理】  其中action内容可扩充，也可以有多个action。 执行顺序：awk一行行读入输入文件，顺序执行‘’内内容，按模式匹配来采取动作。 其他调用：awk可用内部变量和函数，条件与循环语句，也可执行数学运算和字符串操作。此外，可以使用BEGIN和END来执行处理前预操作和处理后后继操作。  格式：  awk [ -F re] [parameter...] ['prog'] [-f progfile] |

|  |  |
| --- | --- |
| 参数说明： | |
| -F re | 允许awk更改其字段分隔符，默认空格  A．-F参数后紧跟单个分隔符，则用双引号“”，例如 –F”+” B．-F参数后紧跟多个分隔符，则用单引号‘ ’并用[ ]，中间顺序无所谓，例如-F’[+$]’ |
| parameter | 该参数帮助为不同的变量赋值 |
| 'prog' | awk的程序语句段。这个语句段必须用单拓号：'和'括起，以防被shell解释 |
| -f progfile | 允许awk调用并执行progfile指定有程序文件  progfile是一个文本文件，他必须符合awk的语法 |
| in\_file | awk的输入文件，awk允许对多个输入文件进行处理。  值得注意的是awk不修改输入文件。  如果未指定输入文件，awk将接受标准输入，并将结果显示在标准输出上。  awk支持输入输出重定向。 |
| BEGIN{….}  {…..}  END{……} | 1 位置：’{}’  2 作用：BEGIN和END的作用是给程序赋予初始状态和在程序结束之后执行一些扫尾的工作  BEGIN{}：awk开始扫描输入之前执行 (显示变量和预置（初始化）变量 )  END{}：在扫描完全部的输入之后执行 (最终结果 )  {}：操作  BEGIN部分：设置计数和打印头  END部分：打印输出文本总数及结尾状态标识 |

## 调用方法

|  |  |
| --- | --- |
| 调用方式： | |
| 1 | 命令行; 普通UNIX命令 ,用于解决简单的问题 |
| 2 | 利用命令解释器调用awk程序;  写入脚本文件，并在首行加入#!/bin/awk –f ,执行之 |
| 3 | 使用-f选项调用awk程序  所有awk插入一个单独文件然后调用  awk –f awk-script-file input-files |

## 域切分&操作

|  |
| --- |
| 域的切分： |
| A#B#C#D  分隔符为#,则 $1:A $2:B ……注意，所有域为 $0  【print隶属于action，所以一定要放在{}中】  awk中，缺省的情况下总是将文本文件中的一行视为一个记录，而将一行中的某一部分作为记录中的一个字段。用$1,$2,$3...这样的方式来顺序地表示行（记录）中的不同字段，$0表示整个行（记录） |
| 示例: |
| 1. 打印所有行   awk ‘{print $0}’ data.txt   1. 打印所有第二个字段   awk ‘{print $2}’ data.txt   1. 打印第一列和最后一列，并且中间有分隔符   awk ‘{print $1,$NF}’ data.txt   1. 打印行号，第一列和最后一列，中间无分隔符   awk ‘{print NR $1 $NF}’ data.txt   1. 打印包含报头   awk ‘BEGIN{print “NAME AGE”}{print $1,$7}’ data.txt   1. 打印包含尾部   awk ‘BEGIN{print “NAME AGE”}{print $1,$7} END{print “END\_OF\_REPORT”}’ data.txt   1. 增加一个域【原来只有11个域】   awk ‘{$12=”AAA”;print $0}’ data  则显示结果中自动多了一个域   1. 修改文本域   awk '{if($1=="M.Tansley") $1="feidd";print $0}' awk.txt   1. 只显示修改的记录   awk ‘{if($1==“M.Tansley”) {$1=“feidd”;print $0}}’ awk.txt  （区分和修改文本域的区别）   1. awk ‘{(tot+=$6)};END {print tot}’ awk.txt（只打印tot值）   awk ‘(tot+=$6);END {print $0“\n”tot}’ awk.txt(打印所有域)   1. awk ‘{print “hello”,”there”,”Jim”}’   打印 hello there Jim |
| 设置域变量&域值比较 |
| 1. 设置输入域到域变量名   awk ‘{name=$1;belts=$4;if(belts ~/Yellow/) print name}’ data.txt  注：设置了 变量直接输出就是了，不需要使用$进行获取，否则拿不到东西，懂？   1. 域值比较   awk ‘{if($6<27) print $0}’ data.txt   1. 设置对所有记录有效的全局域变量【全局变量放在BEGIN中】   awk ‘BEGIN{LINE=”27”}{if($6<LINE) print $0}’ data.txt |
| 修改域值【对原文件无影响】 |
| 1. 修改域值   awk ‘{if($1==”MT”) $6=$6-1;print $1,$6,$7}’ data.txt  【if($1==”MT”) $6=$6-1;符合才执行，其他行都执行print $1,$6,$7】  awk '{if($1=="J.Lulu") $7=$7\*2;print $1,$7}' awk.txt   1. 赋新值   awk ‘{if($1==”MT”) $1=”AB”;print $1}’ data.txt   1. 打印每一列并进行了统计   awk ‘(tot+=$6);END{print tot}’ data.txt   1. 只进行统计   awk ‘{(tot+=$6)};END{print tot}’ data.txt  注：修改数值域取值（当在a w k中修改任何域时，重要的一点是要记住实际输入文件是不可修改的，修改的只是保存在缓存里的awk复本） |

## 正则匹配

|  |
| --- |
| 正则匹配 |
| 1. 模糊匹配   awk -F"^A" '{if($0 ~ /id=00001b/) print $0}' test.txt   1. 精确匹配   awk -F"^A" '{if($1 == "<doc>") print $0}' 1   1. 不匹配   awk -F"^A" '{if($0 !~ /id=000015b/) print $0}' test.txt   1. 特殊匹配   匹配大小写  awk '{if($4~/[Gg]reen/) print $0}' awk.txt  匹配任意字符（匹配第四个字符是a的）  awk '{if($1~/^...a/) print $0}' awk.txt  关系匹配（注意，使用竖线符时，语句必须用圆括号括起来）  awk '{if($4~/(Yellow|green)/) print $0}' awk.txt  5. 不指定域  a．awk ‘/[0-9]/’ data.txt 打印记录中任意列包含数字0－9的行 b．awk ‘/01/||/02/’ data.txt 打印包含01或者02的行 c．awk ‘/01/,/02/’ data.txt 打印既包含01又包含02的行；等同awk ‘/01/&&/02’ |

## 内置变量&操作

|  |  |
| --- | --- |
| awk内置变量 | |
| ARGC | 命令行参数个数 |
| ARGV | 命令行参数排列 |
| ENVIRON | 支持队列中系统环境变量的使用 |
| FILENAME | 实际操作的文件名 |
| FS | 设置输入域分隔符，等价于命令行-F选项 Field Separator  可在BEGIN中进行设置,然后执行的时候均以设置的符号为分隔符 |
| NF | 浏览记录 域的个数，在记录被读取时设置【number of fields】一共有多少个域 |
| NR | 当前记录数（为全部输入文件）  已读取记录数【number of rows】  1  2  3  4  …  全局行数 |
| FNR | 当前文件行数，<=NR |
| NR与FNR区别  awk ‘{print NR,$0}’ filea fileb  结果：行号是连续的，不区分两个文件  awk ‘{print FNR,$0}’ filea fileb  结果：行号区分两个文件 | |
| RS | 输入记录分隔符（默认为换行符）  缺省：新行\n  Row Separator记录分隔符  可以根据具体数据需求，设置读取一条记录的区间 |
| OFS | 输出域的分隔符，缺省空格  输出结果 print $1,$2默认加的是空格，可以在BEGIN中设置，改为其他分隔符 |
| ORS | 输出记录的分隔符，缺省：新行\n  整体记录的 |
| 注：可以在BEGIN中配置FS,OFS,RS,ORS已决定输入输出的分隔符【域和行的】 | |
| 示例： | |
| 1. 打印文件记录数   awk ‘END{print NR}’ data.txt   1. 打印每行记录数，已读记录数，记录内容   awk ‘{print NF,NR,$0} END{print FILENAME}’ data.txt   1. 打印当前所在文件夹   echo $PWD | awk –F / ‘{print $NF}’   1. 列转化成行   每四列转换成一行  awk '{if (NR%4==0){print $0} else {printf"%s ",$0}}' auction.xml   1. 行转换成列   awk 'BEGIN {RS="^A"} {print $0}' 1   1. 指定FS和OFS，多个分隔符 A．BEGIN中的FS无论是单个还是多个分隔，均用双引号“”，多个则需加[ ] 例如：nawk ‘BEGIN{FS=”[\t+$]”}{print $6,$8,$9}’data.txt B．BEGIN中若同时有FS和OFS，则建议中间用分号“；”来分隔开，否则提示出错 例如：nawk ‘BEGIN{FS=”[\t+$]”;OFS=”%”}{print $6,$7,$8,$9}’ data.txt C．半方括号[ ]也可以作为分隔符，顺序无所谓。例如：-F‘[[]]’ 或 –F‘[][]’ | |

## 内置函数&操作

|  |  |
| --- | --- |
| 内置字符串函数 | |
| gsub(r,s) | 在整个$0中用s代替r |
| gsub(r,s,t) | 在整个t中用s替代r |
| index(s,t) | 返回s中字符串t的第一位置 |
| length(s) | 返回s长度 |
| match(s,r) | 测试s是否包含匹配r的字符串 |
| split(s,a,fs) | 在fs上将s分成序列a.fs为分隔符 |
| sprint(fmt,exp) | 返回经fmt格式化后的exp |
| sub(r,s) | 用$0中最左边最长的子串代替s |
| substr(s,p) | 返回字符串s中从p开始的后缀部分 |
| substr(s,p,n) | 返回字符串s中从p开始长度为n的后缀部分 |
| 注意：字符串一定要用双引号括起来，否则当做变量解释，找不到即为空，变成了用空值作了原字符串 | |
| 示例： | |
| 1. 替换一个字符为另一个   awk ‘gsub(/4842/,4899){print $0}’ data.txt  替换字符串必须加双引号，否则当做变量处理，取到值为空，导致空值替换掉现有的东西  $awk –F : ‘gsub(/Mike/,”KEN”){print $0}’ data  只打印出存在替换操作的行  $awk –F : ‘{ gsub(/Mike/,”KEN”);print $0}’ data  打印出所有行，无论是否替换   1. 字符串中出现的第一位置   awk ‘{print index($0,”e”)}’ data.txt   1. 长度   awk ‘{print length($0)}’ data.txt   1. 匹配   awk ‘BEGIN{print match(“ANCD”,/D/)}’ 返回结果4，即开始下标  若不匹配，返回0   1. 切分   awk ‘BEGIN{split(“1#2#3”,myarray,”#”);print length(myarray)}’  $awk ‘BEGIN{STR=”1#2#3”;split(STR,myarray,”#”)}{for(x in myarray){print myarray[x]}}’   1. 发现并替换【注：sub函数的第三个参数不可直接用字符串，而必须用字符串变量】   awk ‘sub(/26/,”29”,$0)’ data.txt  将字符串变量指定字符中第一个符合含有o的用0替换调  awk ‘END{str=”How are you doing?”;sub(/o/,”0”,str);print str}’ data.txt   1. 截取字符串   awk ‘BEGIN{print substr(“abcdefg”,1,3)}’   1. 数学函数   awk ‘{print 2^5+sin(2.1)+int(0.9)}’ data.txt 在每一行都打印运算值（32.8632）   1. 大小写转换   awk ‘{print toupper($2), tolower($2)}’ data.txt 10. match【打印you在字符串中第一个匹配的位置以及长度（9，9，3）】  awk ‘END{print match(“How are you you?”,/you/),RSTART,RLENGTH}’ data.txt | |

## 格式化输出

【注意要自己换行，\n】

|  |  |
| --- | --- |
| awk使用printf进行格式化输出 | |
| %c | ASCII字符 |
| %d | 整数 |
| %e | 浮点数，可科学计数法 |
| %f | 浮点数，小数形式 |
| %g | 由awk决定使用哪种浮点数转换e或f |
| %o | 八进制 |
| %s | 字符串 |
| %x | 十六进制 |
| 示例 | |
| 1. 字符串转换   echo “65”| awk ‘{printf “%c\n”,$0 }’ 将输出A   1. 格式化输出   awk ‘{printf ‘%-15s %s \n’,$1,$3}’ data.txt 左对齐十五个字符等长  参数及脚本 | |

## 数组

|  |
| --- |
| 数组 |
| 1. awk中数组的使用   awk ‘BEGIN{split(“1#2#3”,myarray,”#”);for(I in myarray){print myarray[i]}}’   1. 数组   BEGIN{  myarray[0] = “jim”  myarray[2] = 456  }  for(x in myarray)  {  print myarray[x]  }  awk -v b=$line 'BEGIN {split(b,array,"^B")} END {for(i in array) {print array[i]}}'   1. Awk数组（打印数组）   #!/bin/sh  cat aa.txt | while read line  do  #传入参数line的值  echo "aaa" | awk -v b=$line 'BEGIN {split(b,array,"^B")} END {for(i in array) {print array[i]}}'  Done  awk 'BEGIN {RS="^B"} {print $0}' aa.txt |

## 条件判断&流程控制

|  |
| --- |
| 数组 |
| F. 流程控制：  1 if(){….}else{if()….}  2 while语句  3 do-while语句  4 for(;;){….}  5 exit; (不在END中，文件尾；END中，程序结束)  break(中断当前正在执行的循环并跳到循环外执行下一条语句)；  continue(从当前位置跳到循环开始处执行 ) |
| 条件测试 |
| < <= > >= == !=  ~匹配正则 !~不匹配正则 &&（与）、||（或）和括号（） |
| 1. 小于   awk '{if($6<$7) print $0}' awk.txt   1. 小于等于   awk '{if($6<=$7) print $0}' awk.txt   1. 大于   awk '{if($6>$7) print $0}' awk.txt   1. 打印最后一个域为44的   awk ‘$7==”44” {print $0}’ data.txt   1. 打印第四个字段为G开头的   awk ‘{if($4 ~/G\*/) print $0} data.txt   1. 打印第四字段非Brown的   awk ‘{if($4 !~/Brown/) print $0}’ data.txt   1. 比较最后两个字段符合的   awk ‘{if($6 < $7) print $0 “ABC”}’ data.txt   1. 多条件 && || !   awk ‘{if($1 == “A” && $2==”B”) print $0}’ data.txt |
| for循环 |
| for(x = 1 ; x<=4 ; x++)  {  Print “interation”,x  } |
| while循环 |
| x=1  while(x<NF){  print $x “\t”  x++  if(x == 2)  {  break  }  }  count=1 |
| do while |
| do{  print “”  }while(count != 1) |

## 与shell交互

|  |
| --- |
| 数组 |
| 交互 |
| 1. 将shell命令的执行结果送给awk处理 ：   s=`du -k ./diff" | awk '{ print $1 }'`   1. shell script程序读awk的执行结果 ：``   在awk中执行shell命令行 ：嵌入函数system()  END{print count;system("ls ./");} |
| 传参 |
| 向一行awk传递参数  awk ‘{if($5<AGE) print $0}’ AGE=10 data.txt  awk –v AGE=10 ‘{if($5<AGE) print $0}’ data.txt |

## awk与变量

|  |
| --- |
| 数组 |
| Awk 变量内置变量  1 直接引用，不加$($NF)  自定义变量  1 用户自定义(不要与内置变量冲突)  2 直接引用，不加$  3 不需要对变量进行初始化 (默认其为字符串类型 ) |
| 1. 自定义   awk '{line="44"} {if($7>=line) print $1 " is best"}' awk.txt  awk ‘{if($7>=“44”) print $1 “ is best”}’ awk.txt两者相同   1. 运用对比   A.  $awk –F : ‘BEGIN{NAME=”123131”}{print NAME}’ data  打印 123131  B.  $awk –F : ‘BEGIN{NAME=”123131”}{print $NAME}’ data  什么都不打印  C.  $awk –F : ‘BEGIN{NAME=”123131”}{print ‘$NAME’}’ data  打印出所有行，应该是$NAME 变成 $0了  D.  $awk –F : ‘BEGIN{NAME=”123131”}{print “$NAME”}’ data  打印出$NAME |

## 错误排查

|  |
| --- |
| 错误排查 |
| 在碰到awk错误时，可从下面几个方面查找：  确保整个awk命令用单引号括起来。  确保命令内所有引号成对出现。  确保用花括号括起动作语句，用圆括号括起条件语句。  切记有两条规则可以帮助您避免出现语法错误：  1. 确保命令位于括号中，而括号位于单引号中。没有使用这些字符之一必然导致程序无法运行。  2. 搜索命令需要位于斜线之间。要找出住在印第安那州的员工，您必须使用“/IN/”而不是“IN”。  其他注意：  A．从window拷贝语法句到Unix中，行结尾可能有不识别的字符会导致语法错误 B．写script文件时，BEGIN或END后必须紧跟{ C．所有action语句必须放在{ }中，否则提示语法错误。例如不可直接写print D．单独的赋值或pattern条件句没有{ }则默认print $0；例如{ }外面的i＝1 E．if条件块必须放在action中，也就是必须要有{ }来包含 F．条件的判断“相等”须用= =，而不是=（赋值）；例如，if($1= =100) G．输出多列须用“，”（逗号）隔开；for循环中间隔符为“；”而不是“，” |

## 其他

|  |
| --- |
| 其他 |
| 调用脚本 |
| stu.awk  !/bin/awk –f  BEGIN{  print”ABC”  print “DEF”  }  {total+=56}  END{  print “Total” total  print “AVG” total/WR  } |
| 自定义函数 |
| 定义方法如下：  function 函数名(参数表){  函数体  }  直接调用，但不执行参数有效性检查 (多余的参数会被awk所忽略，而不足的参数，awk将它们置为缺省值0或空字符串 )  返回值：return 返回值 |
| 重定向输出和特别函数getline实例 |
| a．awk ‘{print FILENAME,$0}’ data1.txt data2.txt >data\_all.txt 把第一个文件和第二个文件合并到data\_all.txt中,新文件第一列为原始文件名,后面列为原始文件内容。 b．awk ‘$1!=fd{close(fd);fd=$1} {print substr($0,index($0,“ ”)+1)>$1}’ data\_all.txt 把合并后的新文件data.txt重新拆开成原来的两个子文件，依照新文件的第一列来产生新文件名。生成一个完整子文件后，关闭之；再生成下一个子文件。 A．getline从整体上来说，应这么理解它的用法： 当其左右无重定向符 | 或 < 时，getline作用于当前文件，读入当前文件的第一行给其后跟的变量 var 或$0（无变量）；应该注意到，由于awk在处理getline之前已经读入了一行，所以getline得到的返回结果是隔行的。 当其左右有重定向符 | 或 < 时，getline则作用于定向输入文件，由于该文件是刚打开，并没有被awk读入一行，只是getline读入，那么getline返回的是该文件的第一行，而不是隔行。 B．getline用法大致可分为三大类（每大类又分两小类），即总共有6种用法。代码如下： nawk ‘BEGIN{“cat data.txt”|getline d; print d}’ data2.txt nawk ‘BEGIN{“cat data.txt”|getline; print $0}’ data2.txt nawk ‘BEGIN{getline d < “data.txt”; print d}’ data2.txt nawk ‘BEGIN{getline < “data.txt”; print $0}’ data2.txt 以上四行代码均实现“只打印data.txt文件的第一行”（若打印全部行，用循环） eg. nawk ‘BEGIN{FS=”:”;while(getline<”/etc/passwd”>0){print $1}}’ data.txt nawk ‘{getline d; print d”#”$3}’ data.txt awk首先读入第一行，接着处理getline函数，然后把下一行指定给变量d，再先打印d，由于d后面有换行符，所以后面紧跟的＃会覆盖d，后面的$3同样也会覆盖d。 nawk ‘{getline; print $0”#”$3}’ data.txt awk首先读入第一行接着处理getline函数，然后把下一行指定给$0，现在的$0已经是下一行内容，后面的#和$3（从$0中取）会覆盖$0的内容。 c．nawk ‘BEGIN{system(“echo \”input your name:\” ”);getline var; print “\nYour name is”,d,”\n”}’ 系统提示输入名字，然后将输入的名字打印出来。特别注意：该awk语句后没有输入文件名，而是利用键盘输入作为文件名 |
| 一个例子 |
| 从一个有6千多万行的文件中，每6千行提取一行； 解决：  第一种方法： lineNum=0; while read line do if [ $((lineNum++%6000)) -eq 0 ]; then echo $line fi done<$inputFile  第2种方法：  awk '(NR%6000==0){print $0}' $inputFile |