**前言**



**学习目标**

熟悉Linux系统常用命令与工具，掌握Shell脚本语言语法结构，能独立编写Shell脚本，完成自动化运维常规任务，提高工作效率，为以后学习其他语言打下坚实的基础。

**目标人群**

运维工程师、开发工程师、Linux系统爱好者或已经具备其他编程语言的人群。

**操作系统**

**本文档实验均采用CentOS7\_X64系统。需要注意的是，与CentOS6或者Ubuntu相比，个别命令使用方法会有点不同。**

**作者简介：**



**目录**

[第一章 Shell基础知识 7](#_Toc504048250)

[1.1 Shell简介 7](#_Toc504048251)

[1.2 Shell基本分两大类 8](#_Toc504048252)

[1.3 第一个Shell脚本 8](#_Toc504048253)

[1.4 Shell变量 9](#_Toc504048254)

[1.5 变量引用 11](#_Toc504048255)

[1.6 双引号和单引号 12](#_Toc504048256)

[1.7 注释 13](#_Toc504048257)

[1.8 小结 13](#_Toc504048258)

[第二章 Shell字符串处理之${} 13](#_Toc504048259)

[2.1 获取字符串长度 13](#_Toc504048260)

[2.2 字符串切片 13](#_Toc504048261)

[2.3 替换字符串 14](#_Toc504048262)

[2.4 字符串截取 14](#_Toc504048263)

[2.5 变量状态赋值 15](#_Toc504048264)

[2.6 字符串颜色 15](#_Toc504048265)

[第三章 Shell表达式与运算符 17](#_Toc504048266)

[3.1 条件表达式 17](#_Toc504048267)

[3.2 整数比较符 17](#_Toc504048268)

[3.3 字符串比较符 17](#_Toc504048269)

[3.4 文件测试 18](#_Toc504048270)

[3.5 布尔运算符 19](#_Toc504048271)

[3.6 逻辑判断符 19](#_Toc504048272)

[3.7 整数运算 19](#_Toc504048273)

[3.8 其他运算工具（let/expr/bc） 20](#_Toc504048274)

[3.9 Shell括号用途总结 21](#_Toc504048275)

[第四章 Shell流程控制 22](#_Toc504048276)

[4.1 if语句 22](#_Toc504048277)

[4.2 for语句 24](#_Toc504048278)

[4.3 while语句 26](#_Toc504048279)

[4.4 break和continue语句 27](#_Toc504048280)

[4.5 case语句 28](#_Toc504048281)

[4.6 select语句 29](#_Toc504048282)

[第五章 Shell函数与数组 31](#_Toc504048283)

[5.1 函数 31](#_Toc504048284)

[5.2 数组 33](#_Toc504048285)

[第六章 Shell正则表达式 34](#_Toc504048286)

[6.1正则表达式的概念 34](#_Toc504048287)

[6.2正则表达式的分类 35](#_Toc504048288)

[6.1 小结 38](#_Toc504048289)

[第七章 Shell文本处理三剑客 38](#_Toc504048290)

[7.1 grep 38](#_Toc504048291)

[7.2 sed 42](#_Toc504048292)

[7.2.1 匹配打印（p） 44](#_Toc504048293)

[7.2.2 匹配删除（d） 45](#_Toc504048294)

[7.2.3 替换（s///） 46](#_Toc504048295)

[7.2.4 多重编辑（-e） 49](#_Toc504048296)

[7.2.5 添加新内容（a、i和c） 50](#_Toc504048297)

[7.2.6 读取文件并追加到匹配行后（r） 51](#_Toc504048298)

[7.2.7 将匹配行写到文件（w） 52](#_Toc504048299)

[7.2.8 读取下一行（n和N） 52](#_Toc504048300)

[7.2.9 打印和删除模式空间第一行（P和D） 54](#_Toc504048301)

[7.2.10 保持空间操作（h与H、g与G和x） 55](#_Toc504048302)

[7.2.11 标签（:、b和t） 57](#_Toc504048303)

[7.2.12 忽略大小写匹配（I） 58](#_Toc504048304)

[7.2.13 获取总行数（#） 58](#_Toc504048305)

[8.3 awk 58](#_Toc504048306)

[8.3.1 选项 59](#_Toc504048307)

[8.3.2 模式 59](#_Toc504048308)

[8.3.3 内置变量 65](#_Toc504048309)

[8.3.4 操作符 69](#_Toc504048310)

[8.3.5 流程控制 73](#_Toc504048311)

[8.3.6 数组 76](#_Toc504048312)

[8.3.7 内置函数 79](#_Toc504048313)

[8.3.8 I/O语句 83](#_Toc504048314)

[8.3.9 printf语句 86](#_Toc504048315)

[8.3.10 自定义函数 88](#_Toc504048316)

[8.3.11 需求案例 88](#_Toc504048317)

[第八章 Shell标准输入、输出和错误 93](#_Toc504048318)

[8.1 标准输入、输出和错误 93](#_Toc504048319)

[8.2 重定向符号 94](#_Toc504048320)

[8.3 重定向输出 94](#_Toc504048321)

[8.4 重定向输入 94](#_Toc504048322)

[8.5 重定向标准输出和标准错误 95](#_Toc504048323)

[8.6 重定向到空设备 95](#_Toc504048324)

[8.7 read命令 96](#_Toc504048325)

[第九章 Shell信号发送与捕捉 97](#_Toc504048326)

[9.1 Linux信号类型 97](#_Toc504048327)

[9.2 kill命令 99](#_Toc504048328)

[9.3 trap命令 100](#_Toc504048329)

[第十章 Shell编程时常用的系统文件 101](#_Toc504048330)

[10.1 Linux系统目录结构 101](#_Toc504048331)

[10.2 环境变量文件 102](#_Toc504048332)

[10.3 系统配置文件 102](#_Toc504048333)

[10.4 /dev目录 103](#_Toc504048334)

[10.5 /proc目录 104](#_Toc504048335)

[10.5.1 /proc 104](#_Toc504048336)

[10.5.2 /proc/net 104](#_Toc504048337)

[10.5.3 /proc/sys 105](#_Toc504048338)

[第十一章 Shell常用命令与工具 107](#_Toc504048339)

[11.1 ls 107](#_Toc504048340)

[11.2 echo 108](#_Toc504048341)

[11.3 printf 108](#_Toc504048342)

[11.4 cat 109](#_Toc504048343)

[11.5 tac 110](#_Toc504048344)

[11.6 rev 110](#_Toc504048345)

[11.7 wc 110](#_Toc504048346)

[11.8 cp 111](#_Toc504048347)

[11.9 mkdir 111](#_Toc504048348)

[11.10 mv 111](#_Toc504048349)

[11.11 rename 112](#_Toc504048350)

[11.12 dirname 112](#_Toc504048351)

[11.13 basename 112](#_Toc504048352)

[11.14 du 113](#_Toc504048353)

[11.15 cut 113](#_Toc504048354)

[11.16 tr 113](#_Toc504048355)

[11.17 stat 114](#_Toc504048356)

[11.18 seq 114](#_Toc504048357)

[11.19 shuf 115](#_Toc504048358)

[11.20 sort 116](#_Toc504048359)

[11.21 uniq 117](#_Toc504048360)

[11.22 tee 118](#_Toc504048361)

[11.23 join 118](#_Toc504048362)

[11.24 paste 118](#_Toc504048363)

[11.25 head 119](#_Toc504048364)

[11.26 tail 119](#_Toc504048365)

[11.27 find 120](#_Toc504048366)

[11.28 xargs 121](#_Toc504048367)

[11.29 nl 121](#_Toc504048368)

[11.30 date 122](#_Toc504048369)

[11.31 wget 123](#_Toc504048370)

[11.32 curl 125](#_Toc504048371)

[11.33 scp 126](#_Toc504048372)

[11.34 rsync 127](#_Toc504048373)

[11.35 nohup 127](#_Toc504048374)

[11.36 iconv 128](#_Toc504048375)

[11.37 uname 128](#_Toc504048376)

[11.38 sshpass 129](#_Toc504048377)

[11.39 tar 129](#_Toc504048378)

[11.40 logger 130](#_Toc504048379)

[11.41 netstat 130](#_Toc504048380)

[11.42 ss 130](#_Toc504048381)

[11.43 lsof 131](#_Toc504048382)

[11.44 ps 132](#_Toc504048383)

[11.45 top 133](#_Toc504048384)

[11.46 free 134](#_Toc504048385)

[11.47 df 135](#_Toc504048386)

[11.48 vmstat 135](#_Toc504048387)

[11.49 iostat 136](#_Toc504048388)

[11.50 sar 136](#_Toc504048389)

[11.51 dstat 137](#_Toc504048390)

[11.52 ip 137](#_Toc504048391)

[11.53 nc 138](#_Toc504048392)

[11.54 time 139](#_Toc504048393)

[11.55 eval 140](#_Toc504048394)

[11.56 ssh 140](#_Toc504048395)

[11.57 iptables 141](#_Toc504048396)

[第十二章 Shell脚本编写实战 144](#_Toc504048397)

[注意事项 144](#_Toc504048398)

[12.1 获取随机字符串或数字 145](#_Toc504048399)

[12.2 定义一个颜色输出字符串函数 145](#_Toc504048400)

[12.3 批量创建用户 146](#_Toc504048401)

[12.4 检查软件包是否安装 146](#_Toc504048402)

[12.5 检查服务状态 147](#_Toc504048403)

[12.6 检查主机存活状态 147](#_Toc504048404)

[12.7 监控CPU、内存和硬盘利用率 148](#_Toc504048405)

[12.8 批量主机磁盘利用率监控 149](#_Toc504048406)

[12.9 检查网站可用性 150](#_Toc504048407)

[12.10 检查MySQL主从同步状态 151](#_Toc504048408)

[12.11 屏蔽网站访问频繁的IP 152](#_Toc504048409)

[12.12 判断输入是否为IP 153](#_Toc504048410)

[12.13 判断输入是否为数字 154](#_Toc504048411)

[12.14 找出包含关键字的文件 154](#_Toc504048412)

[12.15 监控目录，将新创建的文件名追加到日志中 155](#_Toc504048413)

[12.16 多个网卡选择 155](#_Toc504048414)

[12.17 查看网卡实时流量 156](#_Toc504048415)

[12.18 MySQL数据库备份 157](#_Toc504048416)

[12.19 Nginx启动脚本 157](#_Toc504048417)

[12.20 选择SSH连接主机 159](#_Toc504048418)

[12.21 FTP下载文件 159](#_Toc504048419)

[12.22 输入五个100数之内的字符，统计和、最小和最大 160](#_Toc504048420)

[12.23 将结果分别赋值给变量 160](#_Toc504048421)

[12.24 批量修改文件名 161](#_Toc504048422)

[12.25 统计当前目录中以.html结尾的文件总大小 161](#_Toc504048423)

[12.26 扫描主机端口状态 161](#_Toc504048424)

[12.27 Expect实现SSH免交互执行命令 162](#_Toc504048425)

[12.28 批量修改服务器用户密码 163](#_Toc504048426)

[12.29 打印乘法口诀 164](#_Toc504048427)

[12.30 getopts工具完善脚本命令行参数 164](#_Toc504048428)

# 第一章 Shell基础知识

## 1.1 Shell简介

Shell是一个C语言编写的脚本语言，它是用户与Linux的桥梁，用户输入命令交给Shell处理，Shell将相应的操作传递给内核（Kernel），内核把处理的结果输出给用户。

下面是流程示意图：

命令

用户

执行动作

解析命令

并传递给内核

Shell

Shell既然是工作在Linux内核之上，那我们也有必要了解下Linux相关知识。

Linux是一套免费试用和自由传播的类Unix操作系统，是一个基于POSIX和UNIX的多用户、多任务、支持多线程和多CPU的操作系统。

1983年9月27日，Richard Stallman（理查德-马修-斯托曼）发起GNU计划，它的目标是创建一套完全自由的操作系统。为保证GNU软件可以自由的使用、复制、修改和发布，所有的GNU软件都有一份在禁止其他人添加任何限制的情况下授权所有权利给任何人的协议条款，GNU通用公共许可证（GNU General Plubic License，GPL），说白了就是不能做商业用途。

GNU是"GNU is Not Unix"的递归缩写。UNIX是一种广泛使用的商业操作系统的名称。

1985年，Richard Stallman又创立了自由软件基金会（Free Software Foundation，FSF）来为GNU计划提供技术、法律以及财政支持。

1990年，GNU计划开发主要项目有Emacs（文本编辑器）、GCC（GNU Compiler Collection，GNU编译器集合）、Bash等，GCC是一套GNU开发的编程语言编译器。还有开发一些UNIX系统的程序库和工具。

1991年，Linuxs Torvalds（林纳斯- 托瓦兹）开发出了与UNIX兼容的Linux操作系统内核并在GPL条款下发布。

1992年，Linux与其他GUN软件结合，完全自由的GUN/Linux操作系统正式诞生，简称Linux。

1995年1月，Bob Young创办ACC公司，以GNU/Linux为核心，开发出了RedHat Linux商业版。

Linux基本思想有两点：第一，一切都是文件；第二，每个软件都有确定的用途。与Unix思想十分相近。

## 1.2 Shell基本分两大类

1.2.1 图形界面Shell（GUI Shell）

GUI为Unix或者类Unix操作系统构造一个功能完善、操作简单以及界面友好的桌面环境。主流桌面环境有KDE，Gnome等。

1.2.2 命令行界面Shell（CLI Shell）

CLI是在用户提示符下键入可执行指令的界面，用户通过键盘输入指令，完成一系列操作。

在Linux系统上主流的CLI实现是Bash，是许多Linux发行版默认的Shell。还有许多Unix上Shell，例如tcsh、csh、ash、bsh、ksh等。

## 1.3 第一个Shell脚本

本教程主要讲解在大多Linux发行版下默认Bash Shell。Linux系统是RedHat下的CentOS操作系统，完全免费。与其商业版RHEL（Red Hat Enterprise Linux）出自同样的源代码，不同的是CentOS并不包含封闭源代码软件和售后支持。

用vi打开test.sh，编写：

# vi test.sh

#!/bin/bash

echo "Hello world!"

第一行指定解释器，第二行打印Hello world！

写好后，开始执行，执行Shell脚本有三种方法：

方法1：直接用bash解释器执行

# bash test.sh

Hello world!

当前终端会新生成一个子bash去执行脚本。

方法2：添加可执行权限

# ll test.sh

-rw-r--r--. 1 root root 32 Aug 18 01:07 test.sh

# chmod +x test.sh

# ./test.sh

-bash: ./test.sh: Permission denied

# chmod +x test.sh

# ./test.sh # ./在当前目录

Hello world!

这种方式默认根据脚本第一行指定的解释器处理，如果没写以当前默认Shell解释器执行。

方法3：source命令执行，以当前默认Shell解释器执行

# source test.sh

Hello world!

## 1.4 Shell变量

1.4.1 系统变量

在命令行提示符直接执行env、set查看系统或环境变量。env显示用户环境变量，set显示Shell预先定义好的变量以及用户变量。可以通过export导出成用户变量。

一些写Shell脚本时常用的系统变量：

|  |  |
| --- | --- |
| $SHELL | 默认Shell |
| $HOME | 当前用户家目录 |
| $IFS | 内部字段分隔符 |
| $LANG | 默认语言 |
| $PATH | 默认可执行程序路径 |
| $PWD | 当前目录 |
| $UID | 当前用户ID |
| $USER | 当前用户 |
| $HISTSIZE | 历史命令大小，可通过HISTTIMEFORMAT变量设置命令执行时间 |
| $RANDOM | 随机生成一个0至32767的整数 |
| $HOSTNAME | 主机名 |

1.4.2 普通变量与临时环境变量

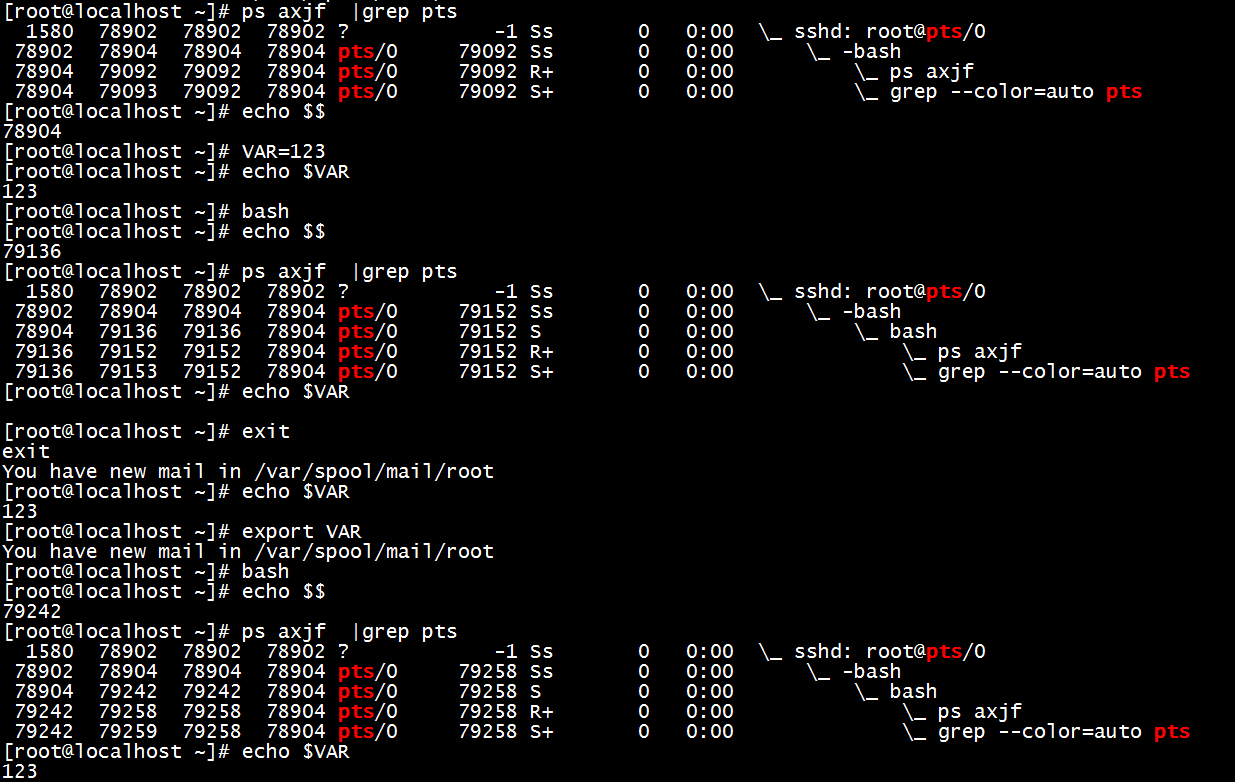
普通变量定义：VAR=value

临时环境变量定义：export VAR=value

变量引用：$VAR

下面看下他们之间区别：

Shell进程的环境变量作用域是Shell进程，当export导入到系统变量时，则作用域是Shell进程及其Shell子进程。



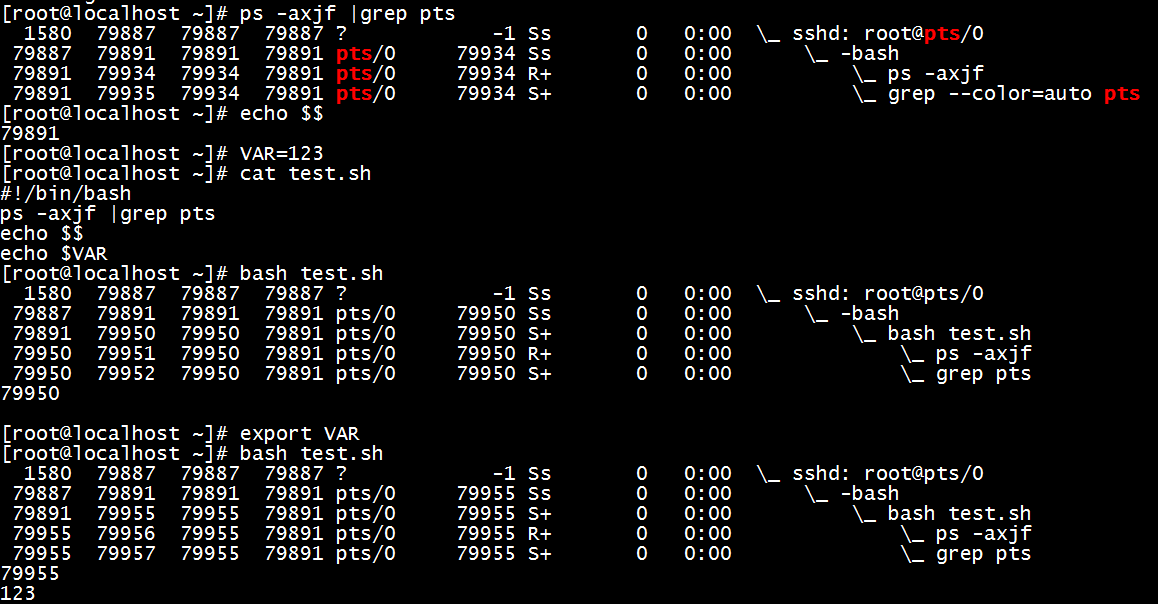


ps axjf输出的第一列是PPID（父进程ID），第二列是PID（子进程ID）

当SSH连接Shell时，当前终端PPID（-bash）是sshd守护程序的PID（root@pts/0），因此在当前终端下的所有进程的PPID都是-bash的PID，比如执行命令、运行脚本。

所以当在-bash下设置的变量，只在-bash进程下有效，而-bash下的子进程bash是无效的，当export后才有效。

进一步说明：再重新连接SSH，去除上面定义的变量测试下



所以在当前shell定义的变量一定要export，否则在写脚本时，会引用不到。

还需要注意的是退出终端后，所有用户定义的变量都会清除。

在/etc/profile下定义的变量就是这个原理，后面有章节会讲解Linux常用变量文件。

1.4.3 位置变量

位置变量指的是函数或脚本后跟的第n个参数。

$1-$n，需要注意的是从第10个开始要用花括号调用，例如${10}

shift可对位置变量控制，例如：

#!/bin/bash

echo "1: $1"

shift

echo "2: $2"

shift

echo "3: $3"

# bash test.sh a b c

1: a

2: c

3:

每执行一次shift命令，位置变量个数就会减一，而变量值则提前一位。shift n，可设置向前移动n位。

1.4.4 特殊变量

|  |  |
| --- | --- |
| $0 | 脚本自身名字 |
| $? | 返回上一条命令是否执行成功，0为执行成功，非0则为执行失败 |
| $# | 位置参数总数 |
| $\* | 所有的位置参数被看做一个字符串 |
| $@ | 每个位置参数被看做独立的字符串 |
| $$ | 当前进程PID |
| $! | 上一条运行后台进程的PID |

## 1.5 变量引用

|  |  |
| --- | --- |
| 赋值运算符 | 示例 |
| = | 变量赋值 |
| += | 两个变量相加 |

1.5.1 自定义变量与引用

# VAR=123

# echo $VAR

123

# VAR+=456

# echo $VAR

123456

Shell中所有变量引用使用$符，后跟变量名。

有时个别特殊字符会影响正常引用，那么需要使用${VAR}，例如：

# VAR=123

# echo $VAR

123

# echo $VAR\_   # Shell允许VAR\_为变量名，所以此引用认为这是一个有效的变量名，故此返回空

# echo ${VAR}

123

还有时候变量名与其他字符串紧碍着，也会误认为是整个变量：

# echo $VAR456

# echo ${VAR}456

123456

1.5.2 将命令结果作为变量值

# VAR=`echo 123`

# echo $VAR

123

# VAR=$(echo 123)

# echo $VAR

123

这里的反撇号等效于$()，都是用于执行Shell命令。

## 1.6 双引号和单引号

在变量赋值时，如果值有空格，Shell会把空格后面的字符串解释为命令：

# VAR=1 2 3

-bash: 2: command not found

# VAR="1 2 3"

# echo $VAR

1 2 3

# VAR='1 2 3'

# echo $VAR

1 2 3

看不出什么区别，再举个说明：

# N=3

# VAR="1 2 $N"

# echo $VAR

1 2 3

# VAR='1 2 $N'

# echo $VAR

1 2 $N

单引号是告诉Shell忽略特殊字符，而双引号则解释特殊符号原有的意义，比如$、！。

## 1.7 注释

Shell注释也很简单，只要在每行前面加个#号，即表示Shell忽略解释。

## 1.8 小结

本章我们了解了什么是shell，并编写了我们的第一个shell脚本！

接着我们了解到在shell中变量的定义和变量的调用，linux系统自带了一些特殊的变量我们可以直接调用，我们还可以自定义变量，但是要记住自定义变量的规范：shell定义的变量是由任何不超过20个的字母、数字或者下划线字符的文本字符串（变量只能以字母或下划线开头）命名的。并且变量是大小写敏感的，因此，变量Var1和变量var1是不同的变量。

脚本中的注释可以对一段代码进行说明，所以良好的注释习惯可以方便我们对脚本的阅读！

在shell脚本中，经常会涉及到字符串相关操作。下一章我们将会了解到在shell中对字符串的处理方式

# 第二章 Shell字符串处理之${}

上一章节讲解了为什么用${}引用变量，${}还有一个重要的功能，就是文本处理，单行文本基本上可以满足你所有需求。

## 2.1 获取字符串长度

# VAR='hello world!'

# echo $VAR

hello world!

# echo ${#VAR}

12

## 2.2 字符串切片

格式：

${parameter:offset}

${parameter:offset:length}

截取从offset个字符开始，向后length个字符。

截取hello字符串：

# VAR='hello world!'

# echo ${VAR:0:5}

hello

截取wo字符：

# echo ${VAR:6:2}

wo

截取world!字符串：

# echo ${VAR:5}

world!

截取最后一个字符：

# echo ${VAR:(-1)}

!

截取最后二个字符：

# echo ${VAR:(-2)}

d!

截取从倒数第3个字符后的2个字符：

# echo ${VAR:(-3):2}

ld

## 2.3 替换字符串

格式：${parameter/pattern/string}

# VAR='hello world world!'

将第一个world字符串替换为WORLD：

# echo ${VAR/world/WORLD}

hello WORLD world!

注意： 原来的变量var并没有改变

将全部world字符串替换为WORLD：

# echo ${VAR//world/WORLD}

hello WORLD WORLD!

替换正则匹配为空：

# VAR=123abc

# echo ${VAR//[^0-9]/}

123

# echo ${VAR//[0-9]/}

abc

patterm前面开头一个正斜杠为只匹配第一个字符串，两个正斜杠为匹配所有字符。

## 2.4 字符串截取

格式：

${parameter#word}   # 删除匹配前缀

${parameter##word}

${parameter%word}   # 删除匹配后缀

${parameter%%word}

# 去掉左边，最短匹配模式，##最长匹配模式。

% 去掉右边，最短匹配模式，%%最长匹配模式。

# URL="http://www.baidu.com/baike/user.html"

以//为分隔符截取右边字符串：

# echo ${URL#\*//}

www.baidu.com/baike/user.html

以/为分隔符截取右边字符串：

# echo ${URL##\*/}

user.html

以//为分隔符截取左边字符串：

# echo ${URL%%//\*}

http:

以/为分隔符截取左边字符串：

# echo ${URL%/\*}

http://www.baidu.com/baike

以.为分隔符截取左边：

# echo ${URL%.\*}

http://www.baidu.com/baike/user

以.为分隔符截取右边：

# echo ${URL##\*.}

html

## 2.5 变量状态赋值

${VAR:-string}  如果VAR变量为空则返回string

${VAR:+string}  如果VAR变量不为空则返回string

${VAR:=string}  如果VAR变量为空则重新赋值VAR变量值为string

${VAR:?string}  如果VAR变量为空则将string输出到stderr

如果变量为空就返回hello world!：

# VAR=

# echo ${VAR:-'hello world!'}

hello world!

如果变量不为空就返回hello world!：

# VAR="hello"

# echo ${VAR:+'hello world!'}

hello world!

如果变量为空就重新赋值：

# VAR=

# echo ${VAR:=hello}

hello

# echo $VAR

hello

如果变量为空就将信息输出stderr：

# VAR=

# echo ${VAR:?value is null}

-bash: VAR: value is null

${}主要用途大概就这么多了，另外还可以获取数组元素，在后面章节会讲到。

## 2.6 字符串颜色

再介绍下字符串输出颜色，有时候关键地方需要醒目，颜色是最好的方式：

|  |  |  |
| --- | --- | --- |
| **字体颜色** | **字体背景颜色** | **显示方式** |
| 30：黑  31：红  32：绿  33：黄  34：蓝色  35：紫色  36：深绿  37：白色 | 40：黑  41：深红  42：绿  43：黄色  44：蓝色  45：紫色  46：深绿  47：白色 | 0：终端默认设置  1：高亮显示  4：下划线  5：闪烁  7：反白显示  8：隐藏 |
| **格式：**  \033[1;31;40m  # 1是显示方式，可选。31是字体颜色。40m是字体背景颜色。  \033[0m     # 恢复终端默认颜色，即取消颜色设置。 | | |

示例：

#!/bin/bash

# 字体颜色

for i in {31..37}; do

    echo -e "\033[$i;40mHello world!\033[0m"

done

# 背景颜色

for i in {41..47}; do

    echo -e "\033[47;${i}mHello world!\033[0m"

done

# 显示方式

for i in {1..8}; do

    echo -e "\033[$i;31;40mHello world!\033[0m"

done



# 第三章 Shell表达式与运算符

## 3.1 条件表达式

|  |  |
| --- | --- |
| **表达式** | **示例** |
| [ expression ] | [ 1 -eq 1 ] |
| [[ expression ]] | [[ 1 -eq 1 ]] |
| test expression | test 1 -eq 1 ,等同于[] |

## 3.2 整数比较符

|  |  |  |
| --- | --- | --- |
| **比较符** | **描述** | **示例** |
| -eq，equal | 等于 | [ 1 -eq 1 ]为true |
| -ne，not equal | 不等于 | [ 1 -ne 1 ]为false |
| -gt，greater than | 大于 | [ 2 -gt 1 ]为true |
| -lt，lesser than | 小于 | [ 2 -lt 1 ]为false |
| -ge，greater or equal | 大于或等于 | [ 2 -ge 1 ]为true |
| -le，lesser or equal | 小于或等于 | [ 2 -le 1 ]为false |

## 3.3 字符串比较符

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **示例** |
| == | 等于 | [ "a" == "a"  ]为true |
| != | 不等于 | [ "a" != "a"  ]为false |
| > | 大于，判断字符串时根据ASCII码表顺序，不常用 | 在[]表达式中：[ 2 \> 1 ]为true  在[[]]表达式中：[[ 2 > 1 ]]为true  在(())表达式中：(( 3 > 2 ))为true |
| < | 小于，判断字符串时根据ASCII码表顺序，不常用 | 在[]表达式中：[ 2 \< 1 ]为false  在[[]]表达式中：[[ 2 < 1 ]]为false  在(())表达式中：(( 3 < 2 ))为false |
| >= | 大于等于 | 在(())表达式中：(( 3 >= 2 ))为true |
| <= | 小于等于 | 在(())表达式中：(( 3 <= 2 ))为false |
| -n | 字符串长度不等于0为真 | VAR1=1;VAR2=""  [ -n "$VAR1" ]为true  [ -n "$VAR2" ]为false |
| -z | 字符串长度等于0为真 | VAR1=1;VAR2=""  [ -z "$VAR1" ]为false  [ -z "$VAR2" ]为true |
| str | 字符串存在为真 | VAR1=1;VAR2=""  [ $VAR1 ]为true  [ $VAR2 ]为false |

需要注意的是，使用-z或-n判断字符串长度时，变量要加双引号。

举例说明：

# [ -z $a ] && echo yes || echo no

yes

# [ -n $a ] && echo yes || echo no

yes

# 加了双引号才能正常判断是否为空

# [ -z "$a" ] && echo yes || echo no

yes

# [ -n "$a" ] && echo yes || echo no

no

# 使用了双中括号就不用了双引号

# [[ -n $a ]] && echo yes || echo no

no

# [[ -z $a ]] && echo yes || echo no

yes

## 3.4 文件测试

|  |  |  |
| --- | --- | --- |
| **测试符** | **描述** | **示例** |
| -e | 文件或目录存在为真 | [ -e path ] path存在为true |
| -f | 文件存在为真 | [ -f file\_path ] 文件存在为true |
| -d | 目录存在为真 | [ -d dir\_path ] 目录存在为true |
| -r | 有读权限为真 | [ -r file\_path ] file\_path有读权限为true |
| -w | 有写权限为真 | [ -w file\_path ] file\_path有写权限为true |
| -x | 有执行权限为真 | [ -x file\_path ] file\_path有执行权限为true |
| -s | 文件存在并且大小大于0为真 | [ -s file\_path ] file\_path存在并且大小大于0为true |

## 3.5 布尔运算符

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **示例** |
| ! | 非关系，条件结果取反 | [ ! 1 -eq 2 ]为true |
| -a | 和关系，在[]表达式中使用 | [ 1 -eq 1 -a 2 -eq 2 ]为true |
| -o | 或关系，在[]表达式中使用 | [ 1 -eq 1 -o 2 -eq 1 ]为true |

## 3.6 逻辑判断符

|  |  |  |
| --- | --- | --- |
| **判断符** | **描述** | **示例** |
| && | 逻辑和，在[[]]和(())表达式中或判断表达式是否为真时使用 | [[ 1 -eq 1 && 2 -eq 2 ]]为true  (( 1 == 1 && 2 == 2 ))为true  [ 1 -eq 1 ] && echo yes 如果&&前面表达式为true则执行后面的 |
| || | 逻辑或，在[[]]和(())表达式中或判断表达式是否为真时使用 | [[ 1 -eq 1 || 2 -eq 1 ]]为true  (( 1 == 1 || 2 == 2 ))为true  [ 1 -eq 2 ] || echo yes 如果||前面表达式为false则执行后面的 |

## 3.7 整数运算

|  |  |
| --- | --- |
| **运算符** | **描述** |
| + | 加法 |
| - | 减法 |
| \* | 乘法 |
| / | 除法 |
| % | 取余 |

|  |  |
| --- | --- |
| **运算表达式** | **示例** |
| $(()) | $((1+1)) |
| $[] | $[1+1] |

上面两个都不支持浮点运算。

$(())表达式还有一个用途，三目运算：

# 如果条件为真返回1，否则返回0

# echo $((1<0))

0

# echo $((1>0))

1

指定输出数字：

# echo $((1>0?1:2))

1

# echo $((1<0?1:2))

2

注意：返回值不支持字符串

## 3.8 其他运算工具（let/expr/bc）

除了Shell本身的算数运算表达式，还有几个命令支持复杂的算数运算：

|  |  |  |
| --- | --- | --- |
| **命令** | **描述** | **示例** |
| let | 赋值并运算，支持++、-- | let VAR=(1+2)\*3 ; echo $VAR  x=10 ; y=5  let x++;echo $x 每执行一次x加1  let y--;echo $y 每执行一次y减1  let x+=2 每执行一次x加2  let x-=2 每执行一次x减2 |
| expr | 乘法\*需要加反斜杠转义\\* | expr 1 \\* 2  运算符两边必须有空格  expr \( 1 + 2 \) \\* 2  使用双括号时要转义 |
| bc | 计算器，支持浮点运算、平方等 | bc本身就是一个计算器，可直接输入命令，进入解释器。  echo 1 + 2 |bc 将管道符前面标准输出作为bc的标准输入  echo "1.2+2" |bc  echo "10^10" |bc  echo 'scale=2;10/3' |bc  用scale保留两位小数点 |

由于Shell不支持浮点数比较，可以借助bc来完成需求：

# echo "1.2 < 2" |bc

1

# echo "1.2 > 2" |bc

0

# echo "1.2 == 2.2" |bc

0

# echo "1.2 != 2.2" |bc

1

看出规律了嘛？运算如果为真返回1，否则返回0，写一个例子：

# [ $(echo "2.2 > 2" |bc) -eq 1 ] && echo yes || echo no

yes

# [ $(echo "2.2 < 2" |bc) -eq 1 ] && echo yes || echo no

no

expr还可以对字符串操作：

获取字符串长度：

# expr length "string"

6

截取字符串：

# expr substr "string" 4 6

ing

获取字符在字符串中出现的位置：

# expr index "string" str

1

# expr index "string" i

4

获取字符串开始字符出现的长度：

# expr match "string" s.\*

6

# expr match "string" str

3

## 3.9 Shell括号用途总结

看到这里，想一想里面所讲的小括号、中括号的用途，是不是有点懵逼了。那我们总结一下！

|  |  |
| --- | --- |
| ( ) | 用途1：在运算中，先计算小括号里面的内容  用途2：数组  用途3：匹配分组 |
| (( )) | 用途1：表达式，不支持-eq这类的运算符。不支持-a和-o，支持<=、>=、<、>这类比较符和&&、||  用途2：C语言风格的for(())表达式 |
| $( ) | 执行Shell命令，与反撇号等效 |
| $(( )) | 用途1：简单算数运算  用途2：支持三目运算符 $(( 表达式?数字:数字 )) |
| [ ] | 条件表达式，里面不支持逻辑判断符 |
| [[ ]] | 条件表达式，里面不支持-a和-o，不支持<=和>=比较符，支持-eq、<、>这类比较符。支持=~模式匹配，也可以不用双引号也不会影响原意，比[]更加通用 |
| $[ ] | 简单算数运算 |
| { } | 对逗号（,）和点点（...）起作用，比如touch {1,2}创建1和2文件，touch {1..3}创建1、2和3文件 |
| ${ } | 用途1：引用变量  用途2：字符串处理 |

# 第四章 Shell流程控制

流程控制是改变程序运行顺序的指令。

## 4.1 if语句

格式：if list; then list; [ elif list; then list; ] ... [ else list; ] fi

4.1.1 单分支

if 条件表达式; then

    命令

fi

示例：

#!/bin/bash

N=10

if [ $N -gt 5 ]; then

    echo yes

fi

# bash test.sh

yes

4.1.2 双分支

if 条件表达式; then

    命令

else

    命令

fi

示例1：

#!/bin/bash

N=10

if [ $N -lt 5 ]; then

    echo yes

else

    echo no

fi

# bash test.sh

no

示例2：判断crond进程是否运行

#!/bin/bash

NAME=crond

NUM=$(ps -ef |grep $NAME |grep -vc grep)

if [ $NUM -eq 1 ]; then

echo "$NAME running."

else

echo "$NAME is not running!"

fi

示例3：检查主机是否存活

#!/bin/bash

if ping -c 1 192.168.1.1 >/dev/null; then

echo "OK."

else

echo "NO!"

fi

if语句可以直接对命令状态进行判断，就省去了获取$?这一步！

4.1.3 多分支

if 条件表达式; then

  命令

elif 条件表达式; then

命令

else

命令

fi

当不确定条件符合哪一个时，就可以把已知条件判断写出来，做相应的处理。

示例1：

#!/bin/bash

N=$1

if [ $N -eq 3 ]; then

echo "eq 3"

elif [ $N -eq 5 ]; then

echo "eq 5"

elif [ $N -eq 8 ]; then

echo "eq 8"

else

echo "no"

fi

如果第一个条件符合就不再向下匹配。

示例2：根据Linux不同发行版使用不同的命令安装软件

#!/bin/bash

if [ -e /etc/redhat-release ]; then

yum install wget -y

elif [ $(cat /etc/issue |cut -d' ' -f1) == "Ubuntu" ]; then

apt-get install wget -y

else

Operating system does not support.

exit

fi

## 4.2 for语句

格式：for name [ [ in [ word ... ] ] ; ] do list ; done

for 变量名 in 取值列表; do

    命令

done

示例：

#!/bin/bash

for i in {1..3}; do

    echo $i

done

# bash test.sh

1

2

3

for的语法也可以这么写：

#!/bin/bash

for i in "$@"; {    # $@是将位置参数作为单个来处理

    echo $i

}

# bash test.sh 1 2 3

1

2

3

默认for循环的取值列表是以空白符分隔，也就是第一章讲系统变量里的$IFS:

#!/bin/bash

for i in 12 34; do

    echo $i

done

# bash test.sh

12

34

如果想指定分隔符，可以重新赋值$IFS变量：

#!/bin/bash

OLD\_IFS=$IFS

IFS=":"

for i in $(head -1 /etc/passwd); do

echo $i

done

IFS=$OLD\_IFS # 恢复默认值

# bash test.sh

root

x

0

0

root

/root

/bin/bash

for循环还有一种C语言风格的语法，常用于计数、打印数字序列：

for (( expr1 ; expr2 ; expr3 )) ; do list ; done

#!/bin/bash

for ((i=1;i<=5;i++)); do  # 也可以i--

    echo $i

done

示例1：检查多个主机是否存活

#!/bin/bash

for ip in 192.168.1.{1..254}; do

if ping -c 1 $ip >/dev/null; then

echo "$ip OK."

else

echo "$ip NO!"

fi

done

示例2：检查多个域名是否可以访问

#!/bin/bash

URL="www.baidu.com www.sina.com www.jd.com"

for url in $URL; do

HTTP\_CODE=$(curl -o /dev/null -s -w %{http\_code} http://$url)

if [ $HTTP\_CODE -eq 200 -o $HTTP\_CODE -eq 301 ]; then

echo "$url OK."

else

echo "$url NO!"

fi

done

## 4.3 while语句

格式：while list; do list; done

while 条件表达式; do

    命令

done

示例1：

#!/bin/bash

N=0

while [ $N -lt 5 ]; do

    let N++

    echo $N

done

# bash test.sh

1

2

3

4

5

当条件表达式为false时，终止循环。

示例2：条件表达式为true，将会产生死循环

#!/bin/bash

while [ 1 -eq 1 ]; do

    echo "yes"

done

也可以条件表达式直接用true：

#!/bin/bash

while true; do

    echo "yes"

done

还可以条件表达式用冒号，冒号在Shell中的意思是不做任何操作。但状态是0，因此为true：

#!/bin/bash

while :; do

    echo "yes"

done

示例3：逐行处理文本

文本内容：

# cat a.txt

a b c

1 2 3

x y z

要想使用while循环逐行读取a.txt文件，有三种方式：

方式1：

#!/bin/bash

cat ./a.txt | while read LINE; do

    echo $LINE

done

方式2：

#!/bin/bash

while read LINE; do

    echo $LINE

done < ./a.txt

方式3：

#!/bin/bash

exec < ./a.txt  # 读取文件作为标准输出

while read LINE; do

    echo $LINE

done

与while关联的还有一个until语句，它与while不同之处在于，是当条件表达式为false时才循环，实际使用中比较少，这里不再讲解。

## 4.4 break和continue语句

break是终止循环。

continue是跳出当前循环。

示例1：在死循环中，满足条件终止循环

#!/bin/bash

N=0

while true; do

    let N++

    if [ $N -eq 5 ]; then

        break

    fi

    echo $N

done

# bash test.sh

1

2

3

4

里面用了if判断，并用了break语句，它是跳出循环。与其关联的还有一个continue语句，它是跳出本次循环。

示例2：举例子说明continue用法

#!/bin/bash

N=0

while [ $N -lt 5 ]; do

    let N++

    if [ $N -eq 3 ]; then

        continue

    fi

    echo $N

done

# bash test.sh

1

2

4

5

当变量N等于3时，continue跳过了当前循环，没有执行下面的echo。

**注意：continue与break语句只能循环语句中使用。**

## 4.5 case语句

case语句一般用于选择性来执行对应部分块命令。

格式：case word in [ [(] pattern [ | pattern ] ... ) list ;; ] ... esac

case 模式名 in

    模式1)

        命令

        ;;

    模式2)

        命令

        ;;

    \*)

        不符合以上模式执行的命令

esac

每个模式必须以右括号结束，命令结尾以双分号结束。

示例：根据位置参数匹配不同的模式

#!/bin/bash

case $1 in

    start)

        echo "start."

        ;;

    stop)

        echo "stop."

        ;;

    restart)

        echo "restart."

        ;;

    \*)

        echo "Usage: $0 {start|stop|restart}"

esac

# bash test.sh

Usage: test.sh {start|stop|restart}

# bash test.sh start

start.

# bash test.sh stop

stop.

# bash test.sh restart

restart.

上面例子是不是有点眼熟，在Linux下有一部分服务启动脚本都是这么写的。

模式也支持正则，匹配哪个模式就执行那个：

#!/bin/bash

case $1 in

[0-9])

echo "match number."

;;

[a-z])

echo "match letter."

;;

'-h'|'--help')

echo "help"

;;

\*)

echo "Input error!"

exit

esac

# bash test.sh 1

match number.

# bash test.sh a

match letter.

# bash test.sh -h

help

# bash test.sh --help

help

模式支持的正则有：\*、?、[ ]、[.-.]、|。后面有章节单独讲解Shell正则表达式。

## 4.6 select语句

select是一个类似于for循环的语句。

格式：select name [ in word ] ; do list ; done

select 变量 in 选项1 选项2; do

    break

done

示例：

#!/bin/bash

select mysql\_version in 5.1 5.6; do

    echo $mysql\_version

done

# bash test.sh

1) 5.1

2) 5.6

#? 1

5.1

#? 2

5.6

用户输入编号会直接赋值给变量mysql\_version。作为菜单用的话，循环第二次后就不再显示菜单了，并不能满足需求。

在外面加个死循环，每次执行一次select就break一次，这样就能每次显示菜单了：

#!/bin/bash

while true; do

    select mysql\_version in 5.1 5.6; do

        echo $mysql\_version

        break

    done

done

# bash test.sh

1) 5.1

2) 5.6

#? 1

5.1

1) 5.1

2) 5.6

#? 2

5.6

1) 5.1

2) 5.6

如果再判断对用户输入的编号执行相应的命令，如果用if语句多分支的话要复杂许多，用case语句就简单多了。

#!/bin/bash

PS3="Select a number: "

while true; do

    select mysql\_version in 5.1 5.6 quit; do

        case $mysql\_version in

            5.1)

                echo "mysql 5.1"

                break

                ;;

            5.6)

                echo "mysql 5.6"

                break

                ;;

            quit)

                exit

                ;;

            \*)

                echo "Input error, Please enter again!"

                break

        esac

    done

done

# bash test.sh

1) 5.1

2) 5.6

3) quit

Select a number: 1

mysql 5.1

1) 5.1

2) 5.6

3) quit

Select a number: 2

mysql 5.6

1) 5.1

2) 5.6

3) quit

Select a number: 3

如果不想用默认的提示符，可以通过重新赋值变量PS3来自定义。这下就比较完美了！

# 第五章 Shell函数与数组

## 5.1 函数

格式：

func() {

    command

}

function关键字可写，也可不写。

示例1：

#!/bin/bash

func() {

    echo "This is a function."

}

func

# bash test.sh

This is a function.

Shell函数很简单，函数名后跟双括号，再跟双大括号。通过函数名直接调用，不加小括号。

示例2：函数返回值

#!/bin/bash

func() {

    VAR=$((1+1))

    return $VAR

    echo "This is a function."

}

func

echo $?

# bash test.sh

2

return在函数中定义状态返回值，返回并终止函数，但返回的只能是0-255的数字，类似于exit。

示例3：函数传参

#!/bin/bash

func() {

    echo "Hello $1"

}

func world

# bash test.sh

Hello world

通过Shell位置参数给函数传参。

函数也支持递归调用，也就是自己调用自己。

例如：

#!/bin/bash

test() {

    echo $1

    sleep 1

    test hello

}

test

执行会一直在调用本身打印hello，这就形成了闭环。

像经典的fork炸弹就是函数递归调用：

:(){ :|:& };: 或 .(){.|.&};.

这样看起来不好理解，我们更改下格式：

:() {

:|:&

};

:

再易读一点：

bomb() {

bomb|bomb&

};

bomb

分析下：

:(){ } 定义一个函数，函数名是冒号。

: 调用自身函数

| 管道符

: 再一次递归调用自身函数

:|: 表示每次调用函数":"的时候就会生成两份拷贝。

& 放到后台

; 分号是继续执行下一个命令，可以理解为换行。

: 最后一个冒号是调用函数。

因此不断生成新进程，直到系统资源崩溃。

一般递归函数用的也少，了解下即可！

## 5.2 数组

数组是相同类型的元素按一定顺序排列的集合。

格式：

array=(元素1 元素2 元素3 ...)

用小括号初始化数组，元素之间用空格分隔。

定义方法1：初始化数组

array=(a b c)

定义方法2：新建数组并添加元素

array[下标]=元素

定义方法3：将命令输出作为数组元素

array=($(command))

数组操作：

获取所有元素：

# echo ${array[\*]}  # \*和@ 都是代表所有元素

a b c

获取元素下标：

# echo ${!a[@]}

0 1 2

获取数组长度：

# echo ${#array[\*]}

3

获取第一个元素：

# echo ${array[0]}

a

获取第二个元素：

# echo ${array[1]}

b

获取第三个元素：

# echo ${array[2]}

c

添加元素：

# array[3]=d

# echo ${array[\*]}

a b c d

添加多个元素：

# array+=(e f g)

# echo ${array[\*]}

a b c d e f g

删除第一个元素：

# unset array[0]   # 删除会保留元素下标

# echo ${array[\*]}

b c d e f g

删除数组：

# unset array

数组下标从0开始。

示例1：讲seq生成的数字序列循环放到数组里面

#!/bin/bash

for i in $(seq 1 10); do

    array[a]=$i

    let a++

done

echo ${array[\*]}

# bash test.sh

1 2 3 4 5 6 7 8 9 10

示例2：遍历数组元素

方法1：

#!/bin/bash

IP=(192.168.1.1 192.168.1.2 192.168.1.3)

for ((i=0;i<${#IP[\*]};i++)); do

    echo ${IP[$i]}

done

# bash test.sh

192.168.1.1

192.168.1.2

192.168.1.3

方法2：

#!/bin/bash

IP=(192.168.1.1 192.168.1.2 192.168.1.3)

for  IP  in ${IP[\*]}; do

     echo $IP

done

# 第六章 Shell正则表达式

## 6.1正则表达式的概念

正则表达式是由普通字符和元字符组成的表达式, 通过此模式，用来匹配特定的内容。

普通字符包括大小写的字母和数字。

元字符，是一些特殊符号，这些字符并不表示字面上的含义, 而会被解释为一些特定的含义。

正则表达式严格区分大小写。

## 6.2正则表达式的分类

Shell正则表达式分为两种：

基础正则表达式：BRE（basic regular express）

扩展正则表达式：ERE（extend regular express），扩展的表达式有+、?、|和()

下面是一些常用的正则表达式符号，我们先拿grep工具举例说明。

|  |  |  |
| --- | --- | --- |
| **符号** | **描述** | **示例** |
| . | 匹配除换行符(\n)之外的任意单个字符 | 匹配123：  echo -e "123\n456" |grep '1.3' |
| ^ | 匹配前面字符串开头 | 匹配以abc开头的行：  echo -e "abc\nxyz" |grep ^abc |
| $ | 匹配前面字符串结尾 | 匹配以xyz结尾的行：  echo -e "abc\nxyz" |grep xyz$ |
| \* | 匹配前一个字符零个或多个 | 匹配x、xo和xoo：  echo -e "x\nxo\nxoo\no\noo" |grep "xo\*"  x是必须的，批量了0零个或多个 |
| + | 匹配前面字符1个或多个 | 匹配abc和abcc：  echo -e "abc\nabcc\nadd" |grep -E 'ab+'  匹配单个数字：echo "113" |grep -o '[0-9]'  连续匹配多个数字：echo "113" |grep -E -o '[0-9]+' |
| ？ | 匹配前面字符0个或1个 | 匹配ac或abc：  echo -e "ac\nabc\nadd" |grep -E 'a?c' |
| [ ] | 匹配中括号之中的任意一个字符 | 匹配a或c：  echo -e "a\nb\nc" |grep '[ac]' |
| [ .-.] | 匹配中括号中范围内的任意一个字符 | 匹配所有字母：  echo -e "a\nb\nc" |grep '[a-z]' |
| [^] | 匹配[^字符]之外的任意一个字符 | 匹配a或b：  echo -e "a\nb\nc" |grep '[^c-z]'  匹配末尾数字：echo "abc:cde;123" |grep -E '[^;]+$' |
| ^[^] | 匹配不是中括号内任意一个字符开头的行 | 匹配不是#开头的行：  grep '^[^#]' /etc/httpd/conf/httpd.conf |
| {n}或{n,} | 匹配花括号前面字符至少n个字符 | 匹配abc字符串（至少三个字符以上字符串）：  echo -e "a\nabc\nc" |grep -E '[a-z]{3}' |
| {n,m} | 匹配花括号前面字符至少n个字符，最多m个字符 | 匹配12和123（不加边界符会匹配单个字符）：  echo -e "1\n12\n123\n1234" |grep -E -w -o '[0-9]{2,3}' |
| \< | 边界符，匹配字符串开始 | 匹配开始是123和1234：  echo -e "1\n12\n123\n1234" |grep '\<123' |
| \> | 边界符，匹配字符串结束 | 匹配结束是1234：  echo -e "1\n12\n123\n1234" |grep '4\>' |
| ( ) | 单元或组合：将小括号里面作为一个组合  分组：匹配小括号中正则表达式或字符。\n反向引用，n是数字，从1开始编号，表示引用第n个分组匹配的内容 | 单元：匹配123a字符串  echo "123abc" |grep -E -o '([0-9a-z]){4}'  分组：匹配11  echo "113abc" |grep -E -o '(1)\1'  匹配出现xo出现零次或多次：  echo -e "x\nxo\nxoo\no\noo" |egrep "(xo)\*" |
| | | 匹配竖杠两边的任意一个 | 匹配12和123：  echo -e "1\n12\n123\n1234" |grep -E '12\>|123\>' |
| \ | 转义符，将特殊符号转成原有意义 | 1.2，匹配1.2：1\.2，否则112也会匹配到 |

|  |  |
| --- | --- |
| **Posix字符** | **描述** |
| [:alnum:] | 等效[a-zA-Z0-9] |
| [:alpha:] | 等效[a-zA-Z] |
| [:lower:] | 等效[a-z] |
| [:upper:] | 等效[A-Z] |
| [:digit:] | 等效[0-9] |
| [:space:] | 匹配任意空白字符，等效[\t\n\r\f\v] |
| [:graph:] | 非空白字符 |
| [:blank:] | 空格与定位字符 |
| [:cntrl:] | 控制字符 |
| [:print:] | 可显示的字符 |
| [:punct:] | 标点符号字符 |
| [:xdigit:] | 十六进制 |

示例：

echo -e "1\n12\n123\n1234a" |grep '[[:digit:]]'

在Shell下使用这些正则表达式处理文本最多的命令有下面几个工具：

|  |  |
| --- | --- |
| **命令** | **描述** |
| grep | 默认不支持扩展表达式，加-E选项开启ERE。如果不加-E使用花括号要加转义符\{\} |
| egrep | 支持基础和扩展表达式 |
| awk | 支持egrep所有的正则表达式 |
| sed | 默认不支持扩展表达式，加-r选项开启ERE。如果不加-r使用花括号要加转义符\{\} |

|  |  |
| --- | --- |
| **支持的特殊字符** | **描述** |
| \w | 匹配任意数字和字母，等效[a-zA-Z0-9\_] |
| \W | 与\w相反，等效[^a-zA-Z0-9\_] |
| \b | 匹配字符串开始或结束，等效\<和\> |
| \s | 匹配任意的空白字符 |
| \S | 匹配非空白字符 |

|  |  |
| --- | --- |
| **空白符** | **描述** |
| \n | 换行符 |
| \r | 回车符 |
| \t | 水平制表符 |
| \v | 垂直制表符 |
| \0 | 空值符 |
| \b | 退后一格 |

## 6.1 小结

如果你在shell脚本中处理数据文件，比如分析日志，寻找同种类型的内容或者文件等，你都必须要熟悉正则表达式的。通过结合字符和特殊字符，你可以匹配出大多数数据类型的模式，然后在使用grep、awk、sed对数据流进行过滤操作。

下一章我们将会结合grep、awk和sed这三个工具对文本数据进行处理。

# 第七章 Shell文本处理三剑客

## 7.1 grep

过滤来自一个文件或标准输入匹配模式内容。

除了grep外，还有egrep、fgrep。egrep是grep的扩展，相当于grep -E。fgrep相当于grep -f，用的少。

Usage: grep [OPTION]... PATTERN [FILE]...

|  |  |
| --- | --- |
| **支持的正则** | **描述** |
| -E，--extended-regexp | 模式是扩展正则表达式（ERE） |
| -P，--perl-regexp | 模式是Perl正则表达式 |
| -e，--regexp=PATTERN | 使用模式匹配，可指定多个模式匹配 |
| -f，--file=FILE | 从文件每一行获取匹配模式 |
| -i，--ignore-case | 忽略大小写 |
| -w，--word-regexp | 模式匹配整个单词 |
| -x，--line-regexp | 模式匹配整行 |
| -v，--invert-match | 打印不匹配的行 |

|  |  |
| --- | --- |
| **输出控制** | **描述** |
| -m，--max-count=NUM | 输出匹配的结果num数 |
| -n，--line-number | 打印行号 |
| -H，--with-filename | 打印每个匹配的文件名 |
| -h，--no-filename | 不输出文件名 |
| -o，--only-matching | 只打印匹配的内容 |
| -q，--quiet | 不输出正常信息 |
| -s, --no-messages | 不输出错误信息 |
| -r，--recursive | 递归目录 |
| -c，--count | 只打印每个文件匹配的行数 |
| --include=FILE\_PATTERN  --exclude=FILE\_PATTERN  --exclude-from=FILE  --exclude-dir=PATTERN | 只检索匹配的文件  跳过匹配的文件  跳过匹配的文件，来自文件模式  跳过匹配的目录 |

|  |  |
| --- | --- |
| **内容行控制** | **描述** |
| -B，--before-context=NUM | 打印匹配的前几行 |
| -A，--after-context=NUM | 打印匹配的后几行 |
| -C，--context=NUM | 打印匹配的前后几行 |
| --color[=WHEN], | 匹配的字体颜色 |

示例：

1）输出b文件中在a文件相同的行

# grep -f a b

2）输出b文件中在a文件不同的行

# grep -v -f a b

3） 匹配多个模式

# echo "a bc de" |xargs -n1 |grep -e 'a' -e 'bc'

a

bc

4）去除空格http.conf文件空行或开头#号的行

# grep -E -v "^$|^#" /etc/httpd/conf/httpd.conf

5） 匹配开头不分大小写的单词

# echo "A a b c" |xargs -n1 |grep -i a

或

# echo "A a b c" |xargs -n1 |grep '[Aa]'

A

a

6）只显示匹配的字符串

# echo "this is a test" |grep -o 'is'

is

is

7）输出匹配的前五个结果

# seq 1 20 |grep -m 5 -E '[0-9]{2}'

10

11

12

13

14

8）统计匹配多少行

# seq 1 20 |grep -c -E '[0-9]{2}'

11

9） 匹配b字符开头的行

# echo "a bc de" |xargs -n1 |grep '^b'

bc

10） 匹配de字符结尾的行并输出匹配的行

# echo "a ab abc abcd abcde" |xargs -n1 |grep -n 'de$'

5:abcde

11） 递归搜索/etc目录下包含ip的conf后缀文件

# grep -r '192.167.1.1' /etc --include \*.conf

12） 排除搜索bak后缀的文件

# grep -r '192.167.1.1' /opt --exclude \*.bak

13） 排除来自file中的文件

# grep -r '192.167.1.1' /opt --exclude-from file

14） 匹配41或42的数字

# seq 41 45 |grep -E '4[12]'

41

42

15） 匹配至少2个字符

# seq 13 |grep -E '[0-9]{2}'

10

11

12

13

16） 匹配至少2个字符的单词，最多3个字符的单词

# echo "a ab abc abcd abcde" |xargs -n1 |grep -E -w -o '[a-z]{2,3}'

ab

abc

17） 匹配所有IP

# ifconfig |grep -E -o "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}"

18） 打印匹配结果及后3行

# seq 1 10 |grep 5 -A 3

5

6

7

8

19） 打印匹配结果及前3行

# seq 1 10 |grep 5 -B 3

2

3

4

5

20） 打印匹配结果及前后3行

# seq 1 10 |grep 5 -C 3

2

3

4

5

6

7

8

21） 不显示输出

不显示错误输出：

# grep 'a' abc

grep: abc: No such file or directory

# grep -s 'a' abc

# echo $?

2

不显示正常输出：

# grep -q 'a' a.txt

grep支持上一章的基础和扩展正则表达式字符。

## 7.2 sed

流编辑器，过滤和替换文本。

工作原理：sed命令将当前处理的行读入模式空间进行处理，处理完把结果输出，并清空模式空间。然后再将下一行读入模式空间进行处理输出，以此类推，直到最后一行。还有一个空间叫保持空间，又称暂存空间，可以暂时存放一些处理的数据，但不能直接输出，只能放到模式空间输出。

这两个空间其实就是在内存中初始化的一个内存区域，存放正在处理的数据和临时存放的数据。

Usage:

sed [OPTION]... {script-only-if-no-other-script} [input-file]...

sed [选项] '地址 命令' file

|  |  |
| --- | --- |
| **选项** | **描述** |
| -n | 不打印模式空间 |
| -e | 执行脚本、表达式来处理 |
| -f | 执行动作从文件读取执行 |
| -i | 修改原文件 |
| -r | 使用扩展正则表达式 |

|  |  |
| --- | --- |
| **命令** | **描述** |
| s/regexp/replacement/ | 替换字符串 |
| p | 打印当前模式空间 |
| P | 打印模式空间的第一行 |
| d | 删除模式空间，开始下一个循环 |
| D | 删除模式空间的第一行，开始下一个循环 |
| = | 打印当前行号 |
| a \text | 当前行追加文本 |
| i \text | 当前行上面插入文本 |
| c \text | 所选行替换新文本 |
| q | 立即退出sed脚本 |
| r | 追加文本来自文件 |
| : label | label为b和t命令 |
| b label | 分支到脚本中带有标签的位置，如果分支不存在则分支到脚本的末尾 |
| t label | 如果s///是一个成功的替换，才跳转到标签 |
| h H | 复制/追加模式空间到保持空间 |
| g G | 复制/追加保持空间到模式空间 |
| x | 交换模式空间和保持空间内容 |
| l | 打印模式空间的行，并显示控制字符$ |
| n N | 读取/追加下一行输入到模式空间 |
| w filename | 写入当前模式空间到文件 |
| ! | 取反、否定 |
| & | 引用已匹配字符串 |

|  |  |
| --- | --- |
| **地址** | **描述** |
| first~step | 步长，每step行，从第first开始 |
| $ | 匹配最后一行 |
| /regexp/ | 正则表达式匹配行 |
| number | 只匹配指定行 |
| addr1,addr2 | 开始匹配addr1行开始，直接addr2行结束 |
| addr1,+N | 从addr1行开始，向后的N行 |
| addr1,~N | 从addr1行开始，到N行结束 |

借助以下文本内容作为示例讲解：

# tail /etc/services

nimgtw        48003/udp            # Nimbus Gateway

3gpp-cbsp       48049/tcp            # 3GPP Cell Broadcast Service Protocol

isnetserv       48128/tcp            # Image Systems Network Services

isnetserv       48128/udp            # Image Systems Network Services

blp5 48129/tcp            # Bloomberg locator

blp5 48129/udp            # Bloomberg locator

com-bardac-dw    48556/tcp            # com-bardac-dw

com-bardac-dw    48556/udp            # com-bardac-dw

iqobject        48619/tcp            # iqobject

iqobject        48619/udp            # iqobject

### 7.2.1 匹配打印（p）

1）打印匹配blp5开头的行

# tail /etc/services |sed -n '/^blp5/p'

blp5            48129/tcp               # Bloomberg locator

blp5            48129/udp               # Bloomberg locator

2）打印第一行

# tail /etc/services |sed -n '1p'

nimgtw          48003/udp               # Nimbus Gateway

3）打印第一行至第三行

# tail /etc/services |sed -n '1,3p'

nimgtw          48003/udp            # Nimbus Gateway

3gpp-cbsp        48049/tcp            # 3GPP Cell Broadcast Service Protocol

isnetserv        48128/tcp            # Image Systems Network Services

4）打印奇数行

# seq 10 |sed -n '1~2p'

1

3

5

7

9

5）打印匹配行及后一行

# tail /etc/services |sed -n '/blp5/,+1p'

blp5            48129/tcp               # Bloomberg locator

blp5            48129/udp               # Bloomberg locator

6）打印最后一行

# tail /etc/services |sed -n '$p'

iqobject        48619/udp               # iqobject

7）不打印最后一行

# tail /etc/services |sed -n '$!p'

3gpp-cbsp        48049/tcp             # 3GPP Cell Broadcast Service Protocol

isnetserv        48128/tcp             # Image Systems Network Services

isnetserv        48128/udp             # Image Systems Network Services

blp5          48129/tcp            # Bloomberg locator

blp5          48129/udp             # Bloomberg locator

com-bardac-dw    48556/tcp             # com-bardac-dw

com-bardac-dw    48556/udp             # com-bardac-dw

iqobject        48619/tcp             # iqobject

iqobject        48619/udp             # iqobject

感叹号也就是对后面的命令取反。

8）匹配范围

# tail /etc/services  |sed -n '/^blp5/,/^com/p'

blp5            48129/tcp               # Bloomberg locator

blp5            48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

匹配开头行到最后一行：

# tail /etc/services |sed -n '/blp5/,$p'

blp5            48129/tcp               # Bloomberg locator

blp5            48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

以逗号分开两个样式选择某个范围。

9）引用系统变量，用引号

# a=1

# tail /etc/services |sed -n ''$a',3p'

或

# tail /etc/services |sed -n "$a,3p"

sed命令用单引号时，里面变量用单引号引起来，或者sed命令用双引号，因为双引号解释特殊符号原有意义。

### 7.2.2 匹配删除（d）

删除与打印使用方法类似，简单举几个例子。

# tail /etc/services |sed '/blp5/d'

nimgtw          48003/udp               # Nimbus Gateway

3gpp-cbsp        48049/tcp               # 3GPP Cell Broadcast Service

isnetserv        48128/tcp               # Image Systems Network Services

isnetserv        48128/udp               # Image Systems Network Services

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

# tail /etc/services |sed '1d'

3gpp-cbsp        48049/tcp               # 3GPP Cell Broadcast Service Protocol

isnetserv        48128/tcp               # Image Systems Network Services

isnetserv        48128/udp               # Image Systems Network Services

blp5          48129/tcp               # Bloomberg locator

blp5          48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

# tail /etc/services |sed '1~2d'

3gpp-cbsp        48049/tcp               # 3GPP Cell Broadcast Service

isnetserv        48128/udp               # Image Systems Network Services

blp5          48129/udp               # Bloomberg locator

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/udp               # iqobject

# tail /etc/services |sed '1,3d'

isnetserv      48128/udp               # Image Systems Network Services

blp5          48129/tcp               # Bloomberg locator

blp5          48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

去除空格http.conf文件空行或开头#号的行：

# sed '/^#/d;/^$/d' /etc/httpd/conf/httpd.conf

打印是把匹配的打印出来，删除是把匹配的删除，删除只是不用-n选项。

### 7.2.3 替换（s///）

1）替换blp5字符串为test

# tail /etc/services |sed 's/blp5/test/'

3gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

isnetserv       48128/udp               # Image Systems Network Services

test          48129/tcp               # Bloomberg locator

test          48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

全局替换加g：

# tail /etc/services |sed 's/blp5/test/g'

2）替换开头是blp5的字符串并打印

# tail /etc/services |sed -n 's/^blp5/test/p'

test          48129/tcp               # Bloomberg locator

test          48129/udp               # Bloomberg locator

3）使用&命令引用匹配内容并替换

# tail /etc/services |sed 's/48049/&.0/'

3gpp-cbsp       48049.0/tcp              # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

isnetserv       48128/udp               # Image Systems Network Services

blp5          48129/tcp               # Bloomberg locator

blp5          48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

IP加单引号：

# echo '10.10.10.1 10.10.10.2 10.10.10.3' |sed -r 's/[^ ]+/"&"/g'

"10.10.10.1" "10.10.10.2" "10.10.10.3"

4）对1-4行的blp5进行替换

# tail /etc/services | sed '1,4s/blp5/test/'

3gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

isnetserv       48128/udp               # Image Systems Network Services

test          48129/tcp               # Bloomberg locator

blp5          48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

5）对匹配行进行替换

# tail /etc/services | sed '/48129\/tcp/s/blp5/test/'

3gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

isnetserv       48128/udp               # Image Systems Network Services

test          48129/tcp               # Bloomberg locator

blp5          48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

6）二次匹配替换

# tail /etc/services  |sed 's/blp5/test/;s/3g/4g/'

4gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

isnetserv       48128/udp               # Image Systems Network Services

test          48129/tcp               # Bloomberg locator

test          48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

7）分组使用，在每个字符串后面添加123

# tail /etc/services |sed -r 's/(.\*) (.\*)(#.\*)/\1\2test \3/'

3gpp-cbsp       48049/tcp              test # 3GPP Cell Broadcast Service

isnetserv       48128/tcp              test # Image Systems Network Services

isnetserv       48128/udp              test # Image Systems Network Services

blp5          48129/tcp              test # Bloomberg locator

blp5          48129/udp              test # Bloomberg locator

com-bardac-dw    48556/tcp              test # com-bardac-dw

com-bardac-dw    48556/udp              test # com-bardac-dw

iqobject        48619/tcp              test # iqobject

iqobject        48619/udp              test # iqobject

matahari        49000/tcp              test # Matahari Broker

第一列是第一个小括号匹配，第二列第二个小括号匹配，第三列一样。将不变的字符串匹配分组，再通过\数字按分组顺序反向引用。

8）将协议与端口号位置调换

# tail /etc/services |sed -r 's/(.\*)(\<[0-9]+\>)\/(tcp|udp)(.\*)/\1\3\/\2\4/'

3gpp-cbsp       tcp/48049               # 3GPP Cell Broadcast Service

isnetserv       tcp/48128               # Image Systems Network Services

isnetserv       udp/48128               # Image Systems Network Services

blp5          tcp/48129               # Bloomberg locator

blp5          udp/48129               # Bloomberg locator

com-bardac-dw    tcp/48556               # com-bardac-dw

com-bardac-dw    udp/48556               # com-bardac-dw

iqobject        tcp/48619               # iqobject

iqobject        udp/48619               # iqobject

matahari        tcp/49000               # Matahari Broker

9）位置调换

替换x字符为大写：

# echo "abc cde xyz" |sed -r 's/(.\*)x/\1X/'

abc cde Xyz

456与cde调换：

# echo "abc:cde;123:456" |sed -r 's/([^:]+)(;.\*:)([^:]+$)/\3\2\1/'

abc:456;123:cde

10）注释匹配行后的多少行

# seq 10 |sed '/5/,+3s/^/#/'

1

2

3

4

#5

#6

#7

#8

9

10

11）注释指定多行

# seq 5 |sed -r 's/^3|^4/&#/'

1

2

3#

4#

5

# seq 5 |sed -r '/^3|^4/s/^/#/'

1

2

#3

#4

5

# seq 5 |sed -r 's/^3|^4/#\0/'

1

2

#3

#4

5

12）去除开头和结尾空格或制表符

# echo "  1 2 3  " |sed 's/^[ \t]\*//;s/[ \t]\*$//'

1 2 3

### 7.2.4 多重编辑（-e）

# tail /etc/services |sed -e '1,2d' -e 's/blp5/test/'

isnetserv       48128/udp               # Image Systems Network Services

test          48129/tcp               # Bloomberg locator

test          48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

也可以使用分号分隔：

# tail /etc/services |sed '1,2d;s/blp5/test/'

### 7.2.5 添加新内容（a、i和c）

1）在blp5上一行添加test

# tail /etc/services |sed '/blp5/i \test'

3gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

isnetserv       48128/udp               # Image Systems Network Services

test

blp5          48129/tcp               # Bloomberg locator

test

blp5          48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

2）在blp5下一行添加test

# tail /etc/services |sed '/blp5/a \test'

3gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

isnetserv       48128/udp               # Image Systems Network Services

blp5          48129/tcp               # Bloomberg locator

test

blp5          48129/udp               # Bloomberg locator

test

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

3）将blp5替换新行

# tail /etc/services |sed '/blp5/c \test'

3gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

isnetserv       48128/udp               # Image Systems Network Services

test

test

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

4）在指定行下一行添加一行

# tail /etc/services |sed '2a \test'

3gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

test

isnetserv       48128/udp               # Image Systems Network Services

blp5          48129/tcp               # Bloomberg locator

blp5          48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

5）在指定行前面和后面添加一行

# seq 5 |sed '3s/.\*/txt\n&/'

1

2

txt

3

4

5

# seq 5 |sed '3s/.\*/&\ntxt/'

1

2

3

txt

4

5

### 7.2.6 读取文件并追加到匹配行后（r）

# cat a.txt

123

456

# tail /etc/services |sed '/blp5/r a.txt'

3gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

isnetserv       48128/udp               # Image Systems Network Services

blp5          48129/tcp               # Bloomberg locator

123

456

blp5          48129/udp               # Bloomberg locator

123

456

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

### 7.2.7 将匹配行写到文件（w）

# tail /etc/services |sed '/blp5/w b.txt'

3gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

isnetserv       48128/udp               # Image Systems Network Services

blp5          48129/tcp               # Bloomberg locator

blp5          48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

# cat b.txt

blp5          48129/tcp               # Bloomberg locator

blp5          48129/udp               # Bloomberg locator

### 7.2.8 读取下一行（n和N）

n 读取下一行到模式空间。

N 追加下一行内容到模式空间，并以换行符\n分隔。

1）打印匹配的下一行

# seq 5 |sed -n '/3/{n;p}'

4

2）打印偶数

# seq 6 |sed -n 'n;p'

2

4

6

sed先读取第一行1，执行n命令，获取下一行2，此时模式空间是2，执行p命令，打印模式空间。 现在模式空间是2，sed再读取3，执行n命令，获取下一行4，此时模式空间为4，执行p命令，以此类推。

3）打印奇数

# seq 6 |sed 'n;d'

1

3

5

sed先读取第一行1，此时模式空间是1，并打印模式空间1，执行n命令，获取下一行2，执行d命令，删除模式空间的2，sed再读取3，此时模式空间是3，并打印模式空间，再执行n命令，获取下一行4，执行d命令，删除模式空间的3，以此类推。

# seq 6 |sed -n 'p;n'

1

3

5

4）每三行执行一次p命令

# seq 6 |sed 'n;n;p'

1

2

3

3

4

5

6

6

sed先读取第一行1，并打印模式空间1，执行n命令，获取下一行2，并打印模式空间2，再执行n命令，获取下一行3，执行p命令，打印模式空间3。sed读取下一行3，并打印模式空间3,以此类推。

5）每三行替换一次

方法1：

# seq 6 |sed 'n;n;s/^/=/;s/$/=/'

1

2

=3=

4

5

=6=

我们只是把p命令改成了替换命令。

方法2：

这次用到了地址匹配，来实现上面的效果：

# seq 6 |sed '3~3{s/^/=/;s/$/=/}'

1

2

=3=

4

5

=6=

当执行多个sed命令时，有时相互会产生影响，我们可以用大括号{}把他们括起来。

6）再看下N命令的功能

# seq 6 |sed 'N;q'

1

2

将两行合并一行：

# seq 6 |sed 'N;s/\n//'

12

34

56

第一个命令：sed读取第一行1，N命令读取下一行2，并以\n2追加，此时模式空间是1\n2，再执行q退出。

为了进一步说明N的功能，看第二个命令：执行N命令后，此时模式空间是1\n2，再执行把\n替换为空，此时模式空间是12，并打印。

# seq 5 |sed -n 'N;p'

1

2

3

4

# seq 6 |sed -n 'N;p'

1

2

3

4

5

6

为什么第一个不打印5呢？

因为N命令是读取下一行追加到sed读取的当前行，当N读取下一行没有内容时，则退出，也不会执行p命令打印当前行。

当行数为偶数时，N始终就能读到下一行，所以也会执行p命令。

7）打印奇数行数时的最后一行

# seq 5 |sed -n '$!N;p'

1

2

3

4

5

加一个满足条件，当sed执行到最后一行时，用感叹号不去执行N命令，随后执行p命令。

### 7.2.9 打印和删除模式空间第一行（P和D）

P 打印模式空间的第一行。

D 删除模式空间的第一行。

1）打印奇数

# seq 6 |sed -n 'N;P'

1

3

5

2）保留最后一行

# seq 6 |sed 'N;D'

6

读取第一行1，执行N命令读取下一行并追加到模式空间，此时模式空间是1\n2，执行D命令删除模式空间第一行1，剩余2。

读取第二行，执行N命令，此时模式空间是3\n4，执行D命令删除模式空间第一行3，剩余4。

以此类推，读取最后一行打印时，而N获取不到下一行则退出，不再执行D，因此模式空间只剩余6就打印。

### 7.2.10 保持空间操作（h与H、g与G和x）

h 复制模式空间内容到保持空间（覆盖）。

H 复制模式空间内容追加到保持空间。

g 复制保持空间内容到模式空间（覆盖）。

G 复制保持空间内容追加到模式空间。

x 模式空间与保持空间内容互换

1）将匹配的内容覆盖到另一个匹配

# seq 6 |sed -e '/3/{h;d}' -e '/5/g'

1

2

4

3

6

h命令把匹配的3复制到保持空间，d命令删除模式空间的3。后面命令再对模式空间匹配5，并用g命令把保持空间3覆盖模式空间5。

2）将匹配的内容放到最后

# seq 6 |sed -e '/3/{h;d}' -e '$G'

1

2

4

5

6

3

3）交换模式空间和保持空间

# seq 6 |sed -e '/3/{h;d}' -e '/5/x' -e '$G'

1

2

4

3

6

5

看后面命令，在模式空间匹配5并将保持空间的3与5交换，5就变成了3,。最后把保持空间的5追加到模式空间的。

4）倒叙输出

# seq 5 |sed '1!G;h;$!d'

5

4

3

2

1

分析下：

1!G 第一行不执行把保持空间内容追加到模式空间，因为现在保持空间还没有数据。

h 将模式空间放到保持空间暂存。

$!d 最后一行不执行删除模式空间的内容。

读取第一行1时，跳过G命令，执行h命令将模式空间1复制到保持空间，执行d命令删除模式空间的1。

读取第二行2时，模式空间是2，执行G命令，将保持空间1追加到模式空间，此时模式空间是2\n1，执行h命令将2\n1覆盖到保持空间，d删除模式空间。

读取第三行3时，模式空间是3，执行G命令，将保持空间2\n1追加到模式空间，此时模式空间是3\n2\n1，执行h命令将模式空间内容复制到保持空间，d删除模式空间。

以此类推，读到第5行时，模式空间是5，执行G命令，将保持空间的4\n3\n2\n1追加模式空间，然后复制到模式空间，5\n4\n3\n2\n1，不执行d，模式空间保留，输出。

由此可见，每次读取的行先放到模式空间，再复制到保持空间，d命令删除模式空间内容，防止输出，再追加到模式空间，因为追加到模式空间，会追加到新读取的一行的后面，循环这样操作， 就把所有行一行行追加到新读取行的后面，就形成了倒叙。

5）每行后面添加新空行

# seq 10 |sed G

1

2

3

4

5

6）打印匹配行的上一行内容

# seq 5 |sed -n '/3/{x;p};h'

2

读取第一行1，没有匹配到3，不执行{x;p}，执行h命令将模式空间内容1覆盖到保持空间。

读取第二行2，没有匹配到3，不执行{x;p}，执行h命令将模式空间内容2覆盖到保持空间。

读取第三行3，匹配到3，执行x命令把模式空间3与保持空间2交换，再执行p打印模式空间2.

以此类推。

7）打印匹配行到最后一行或下一行到最后一行

# seq 5 |sed -n '/3/,$p'

3

4

5

# seq 5 |sed -n '/3/,${h;x;p}'

3

4

5

# seq 5 |sed -n '/3/{:a;N;$!ba;p}'

3

4

5

# seq 5 |sed -n '/3/{n;:a;N;$!ba;p}'

4

5

匹配到3时，n读取下一行4，此时模式空间是4，执行N命令读取下一行并追加到模式空间，此时模式空间是4\n5，标签循环完成后打印模式空间4\n5。

### 7.2.11 标签（:、b和t）

标签可以控制流，实现分支判断。

: lable name  定义标签

b lable  跳转到指定标签，如果没有标签则到脚本末尾

t lable  跳转到指定标签，前提是s///命令执行成功

1）将换行符替换成逗号

方法1：

# seq 6 |sed 'N;s/\n/,/'

1,2

3,4

5,6

这种方式并不能满足我们的需求，每次sed读取到模式空间再打印是新行，替换\n也只能对N命令追加后的1\n2这样替换。

这时就可以用到标签了：

# seq 6 |sed ':a;N;s/\n/,/;b a'

1,2,3,4,5,6

看看这里的标签使用，:a 是定义的标签名，b a是跳转到a位置。

sed读取第一行1，N命令读取下一行2，此时模式空间是1\n2$，执行替换，此时模式空间是1,2$，执行b命令再跳转到标签a位置继续执行N命令，读取下一行3追加到模式空间，此时模式空间是1,2\n3$，再替换，以此类推，不断追加替换，直到最后一行N读不到下一行内容退出。

方法2：

# seq 6 |sed ':a;N;$!b a;s/\n/,/g'

1,2,3,4,5,6

先将每行读入到模式空间，最后再执行全局替换。$!是如果是最后一行，则不执行b a跳转，最后执行全局替换。

# seq 6 |sed ':a;N;b a;s/\n/,/g'

1

2

3

4

5

6

可以看到，不加$!是没有替换，因为循环到N命令没有读到行就退出了，后面的替换也就没执行。

2）每三个数字加个一个逗号

# echo "123456789" |sed -r 's/([0-9]+)([0-9]+{3})/\1,\2/'

123456,789

# echo "123456789" |sed -r ':a;s/([0-9]+)([0-9]+{3})/\1,\2/;t a'

123,456,789

# echo "123456789" |sed -r ':a;s/([0-9]+)([0-9]+{2})/\1,\2/;t a'

1,23,45,67,89

执行第一次时，替换最后一个，跳转后，再对123456匹配替换，直到匹配替换不成功，不执行t命令。

### 7.2.12 忽略大小写匹配（I）

# echo -e "a\nA\nb\nc" |sed 's/a/1/Ig'

1

1

b

c

### 7.2.13 获取总行数（#）

# seq 10 |sed -n '$='

## 8.3 awk

awk是一个处理文本的编程语言工具，能用简短的程序处理标准输入或文件、数据排序、计算以及生成报表等等。

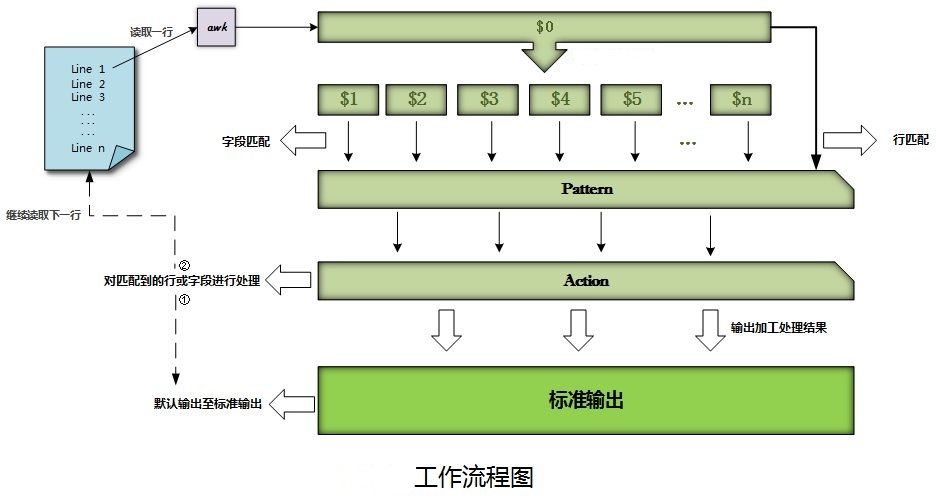
在Linux系统下默认awk是gawk，它是awk的GNU版本。可以通过命令查看应用的版本：ls -l /bin/awk

基本的命令语法：awk option 'pattern {action}' file

其中pattern表示AWK在数据中查找的内容，而action是在找到匹配内容时所执行的一系列命令。花括号用于根据特定的模式对一系列指令进行分组。

awk处理的工作方式与数据库类似，支持对记录和字段处理，这也是grep和sed不能实现的。

在awk中，缺省的情况下将文本文件中的一行视为一个记录，逐行放到内存中处理，而将一行中的某一部分作为记录中的一个字段。用1,2,3...数字的方式顺序的表示行（记录）中的不同字段。用$后跟数字，引用对应的字段，以逗号分隔，0表示整个行。



### 8.3.1 选项

|  |  |
| --- | --- |
| **选项** | **描述** |
| -f program-file | 从文件中读取awk程序源文件 |
| -F fs | 指定fs为输入字段分隔符 |
| -v var=value | 变量赋值 |
| --posix | 兼容POSIX正则表达式 |
| --dump-variables=[file] | 把awk命令时的全局变量写入文件，  默认文件是awkvars.out |
| --profile=[file] | 格式化awk语句到文件，默认是awkprof.out |

### 8.3.2 模式

常用模式有：

|  |  |
| --- | --- |
| **Pattern** | **Description** |
| BEGIN{ } | 给程序赋予初始状态，先执行的工作 |
| END{ } | 程序结束之后执行的一些扫尾工作 |
| /regular expression/ | 为每个输入记录匹配正则表达式 |
| pattern && pattern | 逻辑and，满足两个模式 |
| pattern || pattern | 逻辑or，满足其中一个模式 |
| ! pattern | 逻辑not，不满足模式 |
| pattern1, pattern2 | 范围模式，匹配所有模式1的记录，直到匹配到模式2 |

而动作呢，就是下面所讲的print、流程控制、I/O语句等。

**示例：**

1）从文件读取awk程序处理文件

# vi test.awk

{print $2}

# tail -n3 /etc/services |awk -f test.awk

48049/tcp

48128/tcp

49000/tcp

2）指定分隔符，打印指定字段

打印第二字段，默认以空格分隔：

# tail -n3 /etc/services |awk '{print $2}'

48049/tcp

48128/tcp

48128/udp

指定冒号为分隔符打印第一字段：

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

root

bin

daemon

adm

lp

sync

......

还可以指定多个分隔符，作为同一个分隔符处理：

# tail -n3 /etc/services |awk -F'[/#]' '{print $3}'

 iqobject

 iqobject

 Matahari Broker

# tail -n3 /etc/services |awk -F'[/#]' '{print $1}'

iqobject        48619

iqobject        48619

matahari        49000

# tail -n3 /etc/services |awk -F'[/#]' '{print $2}'

tcp

udp

tcp

# tail -n3 /etc/services |awk -F'[/#]' '{print $3}'

 iqobject

 iqobject

 Matahari Broker

# tail -n3 /etc/services |awk -F'[ /]+' '{print $2}'

48619

48619

49000

[]元字符的意思是符号其中任意一个字符，也就是说每遇到一个/或#时就分隔一个字段，当用多个分隔符时，就能更方面处理字段了。

3）变量赋值

# awk -v a=123 'BEGIN{print a}'

123

系统变量作为awk变量的值：

# a=123

# awk -v a=$a 'BEGIN{print a}'

123

或使用单引号

# awk 'BEGIN{print '$a'}'

123

4）输出awk全局变量到文件

# seq 5 |awk --dump-variables '{print $0}'

1

2

3

4

5

# cat awkvars.out

ARGC: number (1)

ARGIND: number (0)

ARGV: array, 1 elements

BINMODE: number (0)

CONVFMT: string ("%.6g")

ERRNO: number (0)

FIELDWIDTHS: string ("")

FILENAME: string ("-")

FNR: number (5)

FS: string (" ")

IGNORECASE: number (0)

LINT: number (0)

NF: number (1)

NR: number (5)

OFMT: string ("%.6g")

OFS: string (" ")

ORS: string ("\n")

RLENGTH: number (0)

RS: string ("\n")

RSTART: number (0)

RT: string ("\n")

SUBSEP: string ("\034")

TEXTDOMAIN: string ("messages")

5）BEGIN和END

BEGIN模式是在处理文件之前执行该操作，常用于修改内置变量、变量赋值和打印输出的页眉或标题。

例如：打印页眉

# tail /etc/services |awk 'BEGIN{print "Service\t\tPort\t\t\tDescription\n==="}{print $0}'

Service         Port                    Description

===

3gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

isnetserv       48128/udp               # Image Systems Network Services

blp5          48129/tcp               # Bloomberg locator

blp5          48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

END模式是在程序处理完才会执行。

例如：打印页尾

# tail /etc/services |awk '{print $0}END{print "===\nEND......"}'

3gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

isnetserv       48128/udp               # Image Systems Network Services

blp5          48129/tcp               # Bloomberg locator

blp5          48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

===

END......

6）格式化输出awk命令到文件

# tail /etc/services |awk --profile 'BEGIN{print "Service\t\tPort\t\t\tDescription\n==="}{print $0}END{print "===\nEND......"}'

Service Port Description

===

nimgtw 48003/udp # Nimbus Gateway

3gpp-cbsp 48049/tcp # 3GPP Cell Broadcast Service Protocol

isnetserv 48128/tcp # Image Systems Network Services

isnetserv 48128/udp # Image Systems Network Services

blp5 48129/tcp # Bloomberg locator

blp5 48129/udp # Bloomberg locator

com-bardac-dw 48556/tcp # com-bardac-dw

com-bardac-dw 48556/udp # com-bardac-dw

iqobject 48619/tcp # iqobject

iqobject 48619/udp # iqobject

===

END......

# cat awkprof.out

# gawk profile, created Sat Jan 7 19:45:22 2017

# BEGIN block(s)

BEGIN {

print "Service\t\tPort\t\t\tDescription\n==="

}

# Rule(s)

{

print $0

}

# END block(s)

END {

print "===\nEND......"

}

7）/re/正则匹配

匹配包含tcp的行：

# tail /etc/services |awk '/tcp/{print $0}'

3gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

blp5          48129/tcp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

matahari        49000/tcp               # Matahari Broker

匹配开头是blp5的行：

# tail /etc/services |awk '/^blp5/{print $0}'

blp5          48129/tcp               # Bloomberg locator

blp5          48129/udp               # Bloomberg locator

匹配第一个字段是8个字符的行：

# tail /etc/services |awk '/^[a-z0-9]{8} /{print $0}'

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

如果没有匹配到，请查看你的awk版本（awk --version）是不是3，因为4才支持{}

8）逻辑and、or和not

匹配记录中包含blp5和tcp的行：

# tail /etc/services |awk '/blp5/ && /tcp/{print $0}'

blp5          48129/tcp               # Bloomberg locator

匹配记录中包含blp5或tcp的行：

# tail /etc/services |awk '/blp5/ || /tcp/{print $0}'

3gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

blp5          48129/tcp               # Bloomberg locator

blp5          48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

matahari        49000/tcp               # Matahari Broker

不匹配开头是#和空行：

# awk '! /^#/ && ! /^$/{print $0}' /etc/httpd/conf/httpd.conf

或

# awk '! /^#|^$/' /etc/httpd/conf/httpd.conf

或

# awk '/^[^#]|"^$"/' /etc/httpd/conf/httpd.conf

9）匹配范围

# tail /etc/services |awk '/^blp5/,/^com/'

blp5            48129/tcp               # Bloomberg locator

blp5            48129/udp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

对匹配范围后记录再次处理，例如匹配关键字下一行到最后一行：

# seq 5 |awk '/3/,/^$/{printf /3/?"":$0"\n"}'

4

5

另一种判断真假的方式实现：

# seq 5 |awk '/3/{t=1;next}t'

4

5

1和2都不匹配3，不执行后面{}，执行t，t变量还没赋值，为空，空在awk中就为假，就不打印当前行。匹配到3，执行t=1，next跳出，不执行t。4也不匹配3，执行t，t的值上次赋值的1，为真，打印当前行，以此类推。（非0的数字都为真，所以t可以写任意非0数字）

如果想打印匹配行都最后一行，就可以这样了：

# seq 5 |awk '/3/{t=1}t'

3

4

5

### 8.3.3 内置变量

|  |  |
| --- | --- |
| **变量名** | **描述** |
| FS | 输入字段分隔符，默认是空格或制表符 |
| OFS | 输出字段分隔符，默认是空格 |
| RS | 输入记录分隔符，默认是换行符\n |
| ORS | 输出记录分隔符，默认是换行符\n |
| NF | 统计当前记录中字段个数 |
| NR | 统计记录编号，每处理一行记录，编号就会+1 |
| FNR | 统计记录编号，每处理一行记录，编号也会+1，与NR不同的是，处理第二个文件时，编号会重新计数。 |
| ARGC | 命令行参数数量 |
| ARGV | 命令行参数数组序列数组，下标从0开始，ARGV[0]是awk |
| ARGIND | 当前正在处理的文件索引值。第一个文件是1，第二个文件是2，以此类推 |
| ENVIRON | 当前系统的环境变量 |
| FILENAME | 输出当前处理的文件名 |
| IGNORECASE | 忽略大小写 |
| SUBSEP | 数组中下标的分隔符，默认为"\034" |

示例：

1）FS和OFS

在程序开始前重新赋值FS变量，改变默认分隔符为冒号，与-F一样。

# awk 'BEGIN{FS=":"}{print $1,$2}' /etc/passwd |head -n5

root x

bin x

daemon x

adm x

lp x

也可以使用-v来重新赋值这个变量：

# awk -vFS=':' '{print $1,$2}' /etc/passwd |head -n5      # 中间逗号被换成了OFS的默认值

root x

bin x

daemon x

adm x

lp x

由于OFS默认以空格分隔，反向引用多个字段分隔的也是空格，如果想指定输出分隔符这样：

# awk 'BEGIN{FS=":";OFS=":"}{print $1,$2}' /etc/passwd |head -n5

root:x

bin:x

daemon:x

adm:x

lp:x

也可以通过字符串拼接实现分隔：

# awk 'BEGIN{FS=":"}{print $1"#"$2}' /etc/passwd |head -n5

root#x

bin#x

daemon#x

adm#x

lp#x

2）RS和ORS

RS默认是\n分隔每行，如果想指定以某个字符作为分隔符来处理记录：

# echo "www.baidu.com/user/test.html" |awk 'BEGIN{RS="/"}{print $0}'

www.baidu.com

user

test.html

RS也支持正则，简单演示下：

# seq -f "str%02g" 10 |sed 'n;n;a\-----' |awk 'BEGIN{RS="-+"}{print $1}'

str01

str04

str07

str10

将输出的换行符替换为+号：

# seq 10 |awk 'BEGIN{ORS="+"}{print $0}'

1+2+3+4+5+6+7+8+9+10+

替换某个字符：

# tail -n2 /etc/services |awk 'BEGIN{RS="/";ORS="#"}{print $0}'

iqobject        48619#udp               # iqobject

matahari        49000#tcp               # Matahari Broker

3）NF

NF是字段个数。

# echo "a b c d e f" |awk '{print NF}'

6

打印最后一个字段：

# echo "a b c d e f" |awk '{print $NF}'

f

打印倒数第二个字段：

# echo "a b c d e f" |awk '{print $(NF-1)}'

e

排除最后两个字段：

# echo "a b c d e f" |awk '{$NF="";$(NF-1)="";print $0}'

a b c d

排除第一个字段：

# echo "a b c d e f" |awk '{$1="";print $0}'

 b c d e f

4）NR和FNR

NR统计记录编号，每处理一行记录，编号就会+1，FNR不同的是在统计第二个文件时会重新计数。

打印行数：

# tail -n5 /etc/services |awk '{print NR,$0}'

1 com-bardac-dw    48556/tcp               # com-bardac-dw

2 com-bardac-dw    48556/udp               # com-bardac-dw

3 iqobject        48619/tcp               # iqobject

4 iqobject        48619/udp               # iqobject

5 matahari        49000/tcp               # Matahari Broker

打印总行数：

# tail -n5 /etc/services |awk 'END{print NR}'

5

打印第三行：

# tail -n5 /etc/services |awk 'NR==3'

iqobject        48619/tcp               # iqobject

打印第三行第二个字段：

# tail -n5 /etc/services |awk 'NR==3{print $2}'

48619/tcp

打印前三行：

# tail -n5 /etc/services |awk 'NR<=3{print NR,$0}'

1 com-bardac-dw    48556/tcp               # com-bardac-dw

2 com-bardac-dw    48556/udp               # com-bardac-dw

3 iqobject      48619/tcp               # iqobject

看下NR和FNR的区别：

# cat a

a

b

c

# cat b

c

d

e

# awk '{print NR,FNR,$0}' a b

1 1 a

2 2 b

3 3 c

4 1 c

5 2 d

6 3 e

可以看出NR每处理一行就会+1，而FNR在处理第二个文件时，编号重新计数。同时也知道awk处理两个文件时，是合并到一起处理。

# awk 'FNR==NR{print $0"1"}FNR!=NR{print $0"2"}' a b

a1

b1

c1

c2

d2

e2

当FNR==NR时，说明在处理第一个文件内容，不等于时说明在处理第二个文件内容。

一般FNR在处理多个文件时会用到，下面会讲解。

5）ARGC和ARGV

ARGC是命令行参数数量

ARGV是将命令行参数存到数组，元素由ARGC指定，数组下标从0开始

# awk 'BEGIN{print ARGC}' 1 2 3

4

# awk 'BEGIN{print ARGV[0]}'

awk

# awk 'BEGIN{print ARGV[1]}' 1 2

1

# awk 'BEGIN{print ARGV[2]}' 1 2

2

6）ARGIND

ARGIND是当前正在处理的文件索引值，第一个文件是1，第二个文件是2，以此类推，从而可以通过这种方式判断正在处理哪个文件。

# awk '{print ARGIND,$0}' a b

1 a

1 b

1 c

2 c

2 d

2 e

# awk 'ARGIND==1{print "a->"$0}ARGIND==2{print "b->"$0}' a  b

a->a

a->b

a->c

b->c

b->d

b->e

7）ENVIRON

ENVIRON调用系统变量。

# awk 'BEGIN{print ENVIRON["HOME"]}'

/root

如果是设置的环境变量，还需要用export导入到系统变量才可以调用：

# awk 'BEGIN{print ENVIRON["a"]}'

# export a

# awk 'BEGIN{print ENVIRON["a"]}'

123

8）FILENAME

FILENAME是当前处理文件的文件名。

# awk 'FNR==NR{print FILENAME"->"$0}FNR!=NR{print FILENAME"->"$0}' a b

a->a

a->b

a->c

b->c

b->d

b->e

9）忽略大小写

# echo "A a b c" |xargs -n1 |awk 'BEGIN{IGNORECASE=1}/a/'

A

a

等于1代表忽略大小写。

### 8.3.4 操作符

|  |  |
| --- | --- |
| **运算符** | **描述** |
| （....） | 分组 |
| $ | 字段引用 |
| ++ -- | 递增和递减 |
| + - ! | 加号，减号，和逻辑否定 |
| \* / % | 乘，除和取余 |
| + - | 加法，减法 |
| | |& | 管道，用于getline，print和printf |
| < > <= >= != == | 关系运算符 |
| ~ !~ | 正则表达式匹配，否定正则表达式匹配 |
| in | 数组成员 |
| && || | 逻辑and，逻辑or |
| ?: | 简写条件表达式：  expr1 ? expr2 : expr3  第一个表达式为真，执行expr2，否则执行expr3 |
| = += -= \*= /= %= ^= | 变量赋值运算符 |

**须知：**

**在awk中，有3种情况表达式为假：数字是0，空字符串和未定义的值。**

**数值运算，未定义变量初始值为0。字符运算，未定义变量初始值为空。**

举例测试：

# awk 'BEGIN{n=0;if(n)print "true";else print "false"}'

false

# awk 'BEGIN{s="";if(s)print "true";else print "false"}'

false

# awk 'BEGIN{if(s)print "true";else print "false"}'

false

示例：

1）截取整数

# echo "123abc abc123 123abc123" |xargs -n1 | awk '{print +$0}'

123

0

123

# echo "123abc abc123 123abc123" |xargs -n1 | awk '{print -$0}'

-123

0

-123

2）感叹号

打印奇数行：

# seq 6 |awk 'i=!i'

1

3

5

打印偶数行：

# seq 6 |awk '!(i=!i)'

2

4

6

读取第一行：i是未定义变量，也就是i=!0，!取反意思。感叹号右边是个布尔值，0或空字符串为假，非0或非空字符串为真，!0就是真，因此i=1，条件为真打印当前记录。

没有print为什么会打印呢？因为模式后面没有动作，默认会打印整条记录。

读取第二行：因为上次i的值由0变成了1，此时就是i=!1，条件为假不打印。

读取第三行：上次条件又为假，i恢复初始值0，取反，继续打印。以此类推...

可以看出，运算时并没有判断行内容，而是利用布尔值真假判断输出当前行。

2）不匹配某行

# tail /etc/services |awk '!/blp5/{print $0}'

3gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service isnetserv       48128/tcp               # Image Systems Network Services

isnetserv       48128/udp               # Image Systems Network Services

com-bardac-dw    48556/tcp               # com-bardac-dw

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

iqobject        48619/udp               # iqobject

matahari        49000/tcp               # Matahari Broker

3）乘法和除法

# seq 5 |awk '{print $0\*2}'

2

4

6

8

10

# seq 5 |awk '{print $0%2}'

1

0

1

0

1

打印偶数行：

# seq 5 |awk '$0%2==0{print $0}'

2

4

打印奇数行：

# seq 5 |awk '$0%2!=0{print $0}'

1

3

5

4）管道符使用

# seq 5 |shuf |awk '{print $0|"sort"}'

1

2

3

4

5

5）正则表达式匹配

# seq 5 |awk '$0~3{print $0}'

3

# seq 5 |awk '$0!~3{print $0}'

1

2

4

5

# seq 5 |awk '$0~/[34]/{print $0}'

3

4

# seq 5 |awk '$0!~/[34]/{print $0}'

1

2

5

# seq 5 |awk '$0~/[^34]/{print $0}'

1

2

5

6）判断数组成员

# awk 'BEGIN{a["a"]=123}END{if("a" in a)print "yes"}' </dev/null

yes

7）三目运算符

# awk 'BEGIN{print 1==1?"yes":"no"}'  # 三目运算作为一个表达式，里面不允许写print

yes

# seq 3 |awk '{print $0==2?"yes":"no"}'

no

yes

no

替换换行符为逗号：

# seq 5 |awk '{print n=(n?n","$0:$0)}'

1

1,2

1,2,3

1,2,3,4

1,2,3,4,5

# seq 5 |awk '{n=(n?n","$0:$0)}END{print n}'

1,2,3,4,5

说明：读取第一行时，n没有变量，为假输出$0也就是1，并赋值变量n，读取第二行时，n是1为真，输出1,2 以此类推，后面会一直为真。

每三行后面添加新一行：

# seq 10 |awk '{print NR%3?$0:$0 "\ntxt"}'

1

2

3

txt

4

5

6

txt

7

8

9

txt

10

在

两行合并一行：

# seq 6 |awk '{printf NR%2!=0?$0" ":$0" \n"}'

1 2

3 4

5 6

# seq 6 |awk 'ORS=NR%2?" ":"\n"'

1 2

3 4

5 6

# seq 6 |awk '{if(NR%2)ORS=" ";else ORS="\n";print}'

8）变量赋值

字段求和：

# seq 5 |awk '{sum+=1}END{print sum}'

5

# seq 5 |awk '{sum+=$0}END{print sum}'

15

### 8.3.5 流程控制

**1）if语句**

格式：if (condition) statement [ else statement ]

单分支：

# seq 5 |awk '{if($0==3)print $0}'

3

也支持正则匹配判断，一般在写复杂语句时使用：

# echo "123abc#456cde 789aaa#aaabbb " |xargs -n1 |awk -F# '{if($2~/[0-9]/)print $2}'

456cde

# echo "123abc#456cde 789aaa#aaabbb " |xargs -n1 |awk -F# '{if($2!~/[0-9]/)print $2}'

aaabbb

或

# echo "123abc#456cde 789aaa#aaabbb" |xargs -n1 |awk -F# '$2!~/[0-9]/{print $2}'

aaabbb

双分支：

# seq 5 |awk '{if($0==3)print $0;else print "no"}'

no

no

3

no

no

多分支：

# cat file

1 2 3

4 5 6

7 8 9

# awk '{if($1==4){print "1"} else if($2==5){print "2"} else if($3==6){print "3"} else {print "no"}}' file

no

1

no

**2）while语句**

格式：while (condition) statement

遍历打印所有字段：

# awk '{i=1;while(i<=NF){print $i;i++}}' file

1

2

3

4

5

6

7

8

9

awk是按行处理的，每次读取一行，并遍历打印每个字段。

**3）for语句C语言风格**

格式：for (expr1; expr2; expr3) statement

遍历打印所有字段：

# cat file

1 2 3

4 5 6

7 8 9

# awk '{for(i=1;i<=NF;i++)print $i}' file

1

2

3

4

5

6

7

8

9

倒叙打印文本：

# awk '{for(i=NF;i>=1;i--)print $i}' file

3

2

1

6

5

4

9

8

7

都换行了，这并不是我们要的结果。怎么改进呢？

# awk '{for(i=NF;i>=1;i--){printf $i" "};print ""}' file  # print本身就会新打印一行

3 2 1

6 5 4

9 8 7

或

# awk '{for(i=NF;i>=1;i--)if(i==1)printf $i"\n";else printf $i" "}' file

3 2 1

6 5 4

9 8 7

在这种情况下，是不是就排除第一行和倒数第一行呢？我们正序打印看下

排除第一行：

# awk '{for(i=2;i<=NF;i++){printf $i" "};print ""}' file

2 3

5 6

8 9

排除第二行：

# awk '{for(i=1;i<=NF-1;i++){printf $i" "};print ""}' file

1 2

4 5

7 8

IP加单引号：

# echo '10.10.10.1 10.10.10.2 10.10.10.3' |awk '{for(i=1;i<=NF;i++)printf "\047"$i"\047"}

'10.10.10.1'  '10.10.10.2'  '10.10.10.3'

\047是ASCII码，可以通过showkey -a命令查看。

**4）for语句遍历数组**

格式：for (var in array) statement

# seq -f "str%.g" 5 |awk '{a[NR]=$0}END{for(v in a)print v,a[v]}'

4 str4

5 str5

1 str1

2 str2

3 str3

**5）break和continue语句**

break跳过所有循环，continue跳过当前循环。

# awk 'BEGIN{for(i=1;i<=5;i++){if(i==3){break};print i}}'

1

2

# awk 'BEGIN{for(i=1;i<=5;i++){if(i==3){continue};print i}}'

1

2

4

5

**6）删除数组和元素**

格式：

delete array[index]  删除数组元素

delete array  删除数组

# seq -f "str%.g" 5 |awk '{a[NR]=$0}END{delete a;for(v in a)print v,a[v]}'

空的…

# seq -f "str%.g" 5 |awk '{a[NR]=$0}END{delete a[3];for(v in a)print v,a[v]}'

4 str4

5 str5

1 str1

2 str2

**7）exit语句**

格式：exit [ expression ]

exit退出程序，与shell的exit一样。[ expr ]是0-255之间的数字。

# seq 5 |awk '{if($0~/3/)exit (123)}'

# echo $?

123

### 8.3.6 数组

数组：存储一系列相同类型的元素，键/值方式存储，通过下标（键）来访问值。

awk中数组称为关联数组，不仅可以使用数字作为下标，还可以使用字符串作为下标。

数组元素的键和值存储在awk程序内部的一个表中，该表采用散列算法，因此数组元素是随机排序。

数组格式：array[index]=value

1）自定义数组

# awk 'BEGIN{a[0]="test";print a[0]}'

test

2）通过NR设置记录下标，下标从1开始

# tail -n3 /etc/passwd |awk -F: '{a[NR]=$1}END{print a[1]}'

systemd-network

# tail -n3 /etc/passwd |awk -F: '{a[NR]=$1}END{print a[2]}'

zabbix

# tail -n3 /etc/passwd |awk -F: '{a[NR]=$1}END{print a[3]}'

user

3）通过for循环遍历数组

# tail -n5 /etc/passwd |awk -F: '{a[NR]=$1}END{for(v in a)print a[v],v}'

zabbix 4

user 5

admin 1

systemd-bus-proxy 2

systemd-network 3

# tail -n5 /etc/passwd |awk -F: '{a[NR]=$1}END{for(i=1;i<=NR;i++)print a[i],i}'

admin 1

systemd-bus-proxy 2

systemd-network 3

zabbix 4

user 5

上面打印的i是数组的下标。

第一种for循环的结果是乱序的，刚说过，数组是无序存储。

第二种for循环通过下标获取的情况是排序正常。

所以当下标是数字序列时，还是用for(expr1;expr2;expr3)循环表达式比较好，保持顺序不变。

4）通过++方式作为下标

# tail -n5 /etc/passwd |awk -F: '{a[x++]=$1}END{for(i=0;i<=x-1;i++)print a[i],i}'

admin 0

systemd-bus-proxy 1

systemd-network 2

zabbix 3

user 4

x被awk初始化值是0，没循环一次+1

5）使用字段作为下标

# tail -n5 /etc/passwd |awk -F: '{a[$1]=$7}END{for(v in a)print a[v],v}'

/sbin/nologin admin

/bin/bash user

/sbin/nologin systemd-network

/sbin/nologin systemd-bus-proxy

/sbin/nologin zabbix

6）统计相同字段出现次数

# tail /etc/services |awk '{a[$1]++}END{for(v in a)print a[v],v}'

2 com-bardac-dw

1 3gpp-cbsp

2 iqobject

1 matahari

2 isnetserv

2 blp5

# tail /etc/services |awk '{a[$1]+=1}END{for(v in a)print a[v],v}'

2 com-bardac-dw

1 3gpp-cbsp

2 iqobject

1 matahari

2 isnetserv

2 blp5

# tail /etc/services |awk '/blp5/{a[$1]++}END{for(v in a)print a[v],v}'

2 blp5

第一个字段作为下标，值被++初始化是0，每次遇到下标（第一个字段）一样时，对应的值就会被+1，因此实现了统计出现次数。

想要实现去重的的话就简单了，只要打印下标即可。

7）统计TCP连接状态

# netstat -antp |awk '/^tcp/{a[$6]++}END{for(v in a)print a[v],v}'

9 LISTEN

6 ESTABLISHED

6 TIME\_WAIT

8）只打印出现次数大于等于2的

# tail /etc/services |awk '{a[$1]++}END{for(v in a) if(a[v]>=2){print a[v],v}}'

2 com-bardac-dw

2 iqobject

2 isnetserv

2 blp5

9）去重

只打印重复的行：

# tail /etc/services |awk 'a[$1]++'

isnetserv       48128/udp               # Image Systems Network Services

blp5          48129/udp               # Bloomberg locator

com-bardac-dw    48556/udp               # com-bardac-dw

iqobject        48619/udp               # iqobject

不打印重复的行：

# tail /etc/services |awk '!a[$1]++'

3gpp-cbsp       48049/tcp               # 3GPP Cell Broadcast Service

isnetserv       48128/tcp               # Image Systems Network Services

blp5          48129/tcp               # Bloomberg locator

com-bardac-dw    48556/tcp               # com-bardac-dw

iqobject        48619/tcp               # iqobject

matahari        49000/tcp               # Matahari Broker

先明白一个情况，当值是0是为假，非0整数为真，知道这点就不难理解了。

只打印重复的行说明：当处理第一条记录时，执行了++，初始值是0为假，就不打印，如果再遇到相同的记录，值就会+1，不为0，则打印。

不打印重复的行说明：当处理第一条记录时，执行了++，初始值是0为假，感叹号取反为真，打印，如果再遇到相同的记录，值就会+1，不为0为真，取反为假就不打印。

# tail /etc/services |awk '{if(a[$1]++)print $1}'

isnetserv

blp5

com-bardac-dw

iqobject

使用三目运算：

# tail /etc/services |awk '{print a[$1]++?$1:"no"}'

no

no

isnetserv

no

blp5

no

com-bardac-dw

no

iqobject

no

# tail /etc/services |awk '{if(!a[$1]++)print $1}'

3gpp-cbsp

isnetserv

blp5

com-bardac-dw

iqobject

matahari

10）统计每个相同字段的某字段总数：

# tail /etc/services |awk -F'[ /]+' '{a[$1]+=$2}END{for(v in a)print v, a[v]}'

com-bardac-dw 97112

3gpp-cbsp 48049

iqobject 97238

matahari 49000

isnetserv 96256

blp5 96258

11）多维数组

awk的多维数组，实际上awk并不支持多维数组，而是逻辑上模拟二维数组的访问方式，比如a[a,b]=1，使用SUBSEP（默认\034）作为分隔下标字段，存储后是这样a\034b。

示例：

# awk 'BEGIN{a["x","y"]=123;for(v in a) print v,a[v]}'

xy 123

我们可以重新复制SUBSEP变量，改变下标默认分隔符：

# awk 'BEGIN{SUBSEP=":";a["x","y"]=123;for(v in a) print v,a[v]}'

x:y 123

根据指定的字段统计出现次数：

# cat file

A 192.168.1.1 HTTP

B 192.168.1.2 HTTP

B 192.168.1.2 MYSQL

C 192.168.1.1 MYSQL

C 192.168.1.1 MQ

D 192.168.1.4 NGINX

# awk 'BEGIN{SUBSEP="-"}{a[$1,$2]++}END{for(v in a)print a[v],v}' file

1 D-192.168.1.4

1 A-192.168.1.1

2 C-192.168.1.1

2 B-192.168.1.2

### 8.3.7 内置函数

|  |  |
| --- | --- |
| **函数** | **描述** |
| int(expr) | 截断为整数 |
| sqrt(expr) | 平方根 |
| rand() | 返回一个随机数N，0和1范围，0 < N < 1 |
| srand([expr]) | 使用expr生成随机数，如果不指定，默认使用当前时间为种子，如果前面有种子则使用生成随机数 |
| asort(a, b) | 对数组a的值进行排序，把排序后的值存到新的数组b中，新排序的数组下标从1开始 |
| asorti(a,b) | 对数组a的下标进行排序，同上 |
| sub(r, s [, t]) | 对输入的记录用s替换r正则匹配，t可选针对某字段替换，但只替换第一个字符串 |
| gsub(r, s [, t]) | 对输入的记录用s替换r正则匹配，t可选针对某字段替换，否则替换所有字符串 |
| gensub(r, s, h [, t]) | 对输入的记录用s替换r正则匹配，h替换指定索引位置 |
| index(s, t) | 返回s中字符串t的索引位置，0为不存在 |
| length([s]) | 返回s的长度 |
| match(s, r [, a]) | 测试字符串s是否包含匹配r的字符串，如果不包含返回0 |
| split(s, a [, r [, seps] ]) | 根据分隔符seps将s分成数组a |
| substr(s, i [, n]) | 截取字符串s从i开始到长度n，如果n没指定则是剩余部分 |
| tolower(str) | str中的所有大写转换成小写 |
| toupper(str) | str中的所有小写转换成大写 |
| systime() | 当前时间戳 |
| strftime([format [, timestamp[, utc-flag]]]) | 格式化输出时间，将时间戳转为字符串 |

**示例：**

1）int()

截断为整数：

# echo -e "123abc\nabc123\n123abc123" | awk '{print int($0)}'

123

0

123

# awk 'BEGIN{print int(10/3)}'

3

2）sqrt()

获取9的平方根：

# awk 'BEGIN{print sqrt(9)}'

3

3）rand()和srand()

rand()并不是每次运行就是一个随机数，会一直保持一个不变：

# awk 'BEGIN{print rand()}'

0.237788

当执行srand()函数后，rand()才会发生变化，所以一般在awk着两个函数结合生成随机数，但是也有很大几率生成一样：

# awk 'BEGIN{srand();print rand()}'

0.31687

如果想生成1-10的随机数可以这样：

# awk 'BEGIN{srand();print int(rand()\*10)}'

4

如果想更完美生成随机数，还得做相应的处理！

4）asort()和asorti()

排序数组：

# seq -f "str%.g" 5 |awk '{a[x++]=$0}END{s=asort(a,b);for(i=1;i<=s;i++)print b[i],i}'

str1 1

str2 2

str3 3

str4 4

str5 5

# seq -f "str%.g" 5 |awk '{a[x++]=$0}END{s=asorti(a,b);for(i=1;i<=s;i++)print b[i],i}'

0 1

1 2

2 3

3 4

4 5

asort将a数组的值放到数组b，a下标丢弃，并将数组b的总行号赋值给s，新数组b下标从1开始，然后遍历。

5）sub()和gsub()

替换正则匹配的字符串：

# tail /etc/services |awk '/blp5/{sub(/tcp/,"icmp");print $0}'

blp5            48129/icmp              # Bloomberg locator

blp5            48129/udp               # Bloomberg locator

# tail /etc/services |awk '/blp5/{gsub(/c/,"9");print $0}'

blp5            48129/t9p               # Bloomberg lo9ator

blp5            48129/udp               # Bloomberg lo9ator

# echo "1 2 2 3 4 5" |awk 'gsub(2,7,$2){print $0}'

1 7 2 3 4 5

# echo "1 2 3 a b c" |awk 'gsub(/[0-9]/, '0'){print $0}'

0 0 0 a b c

在指定行前后加一行：

# seq 5 | awk 'NR==2{sub('/.\*/',"txt\n&")}{print}'

1

txt

2

3

4

5

# seq 5 | awk 'NR==2{sub('/.\*/',"&\ntxt")}{print}'

1

2

txt

3

4

5

6）index()

获取字段索引起始位置：

# tail -n 5 /etc/services |awk '{print index($2,"tcp")}'

7

0

7

0

7

7）length()

统计字段长度：

# tail -n 5 /etc/services |awk '{print length($2)}'

9

9

9

9

9

统计数组的长度：

# tail -n 5 /etc/services |awk '{a[$1]=$2}END{print length(a)}'

3

8）match

# echo "123abc#456cde 789aaa#234bbb 999aaa#aaabbb" |xargs -n1 |awk '{print match($0,234)}'

0

8

0

如果记录匹配字符串234，则返回索引位置，否则返回0。

那么，我们只想打印包含这个字符串的记录就可以这样：

# echo "123abc#456cde 789aaa#234bbb 999aaa#aaabbb" |xargs -n1 |awk '{if(match($0,234)!=0)print $0}'

789aaa#234bbb

9）split()

切分记录为数组a：

# echo -e "123#456#789\nabc#cde#fgh" |awk '{split($0,a);for(v in a)print a[v],v}'

123#456#789 1

abc#cde#fgh 1

以#号切分记录为数据a：

# echo -e "123#456#789\nabc#cde#fgh" |awk '{split($0,a,"#");for(v in a)print a[v],v}'

123 1

456 2

789 3

abc 1

cde 2

fgh 3

10）substr()

截取字符串索引4到最后：

# echo -e "123#456#789\nabc#cde#fgh" |awk '{print substr($0,4)}'

#456#789

#cde#fgh

截取字符串索引4到长度5：

# echo -e "123#456#789\nabc#cde#fgh" |awk '{print substr($0,4,5)}'

#456#

#cde#

11）tolower()和toupper()

转换小写：

# echo -e "123#456#789\nABC#cde#fgh" |awk '{print tolower($0)}'

123#456#789

abc#cde#fgh

转换大写：

# echo -e "123#456#789\nabc#cde#fgh" |awk '{print toupper($0)}'

123#456#789

ABC#CDE#FGH

12)时间处理

返回当前时间戳：

# awk 'BEGIN{print systime()}'

1483297766

将时间戳转为日期和时间

# echo "1483297766" |awk '{print strftime("%Y-%m-%d %H:%M:%S",$0)}'

2017-01-01 14:09:26

### 8.3.8 I/O语句

|  |  |
| --- | --- |
| **语句** | **描述** |
| getline | 读取下一个输入记录设置给$0 |
| getline var | 读取下一个输入记录并赋值给变量var |
| command | getline [var] | 运行Shell命令管道输出到$0或var |
| next | 停止当前处理的输入记录后面动作 |
| print | 打印当前记录 |
| printf fmt, expr-list | 格式化输出 |
| printf fmt, expr-list >file | 格式输出和写到文件 |
| system(cmd-line) | 执行命令和返回状态 |
| print ... >> file | 追加输出到文件 |
| print ... | command | 打印输出作为命令输入 |

**示例：**

1）getline

获取匹配的下一行：

# seq 5 |awk '/3/{getline;print}'

4

# seq 5 |awk '/3/{print;getline;print}'

3

4

在匹配的下一行加个星号：

# seq 5 |awk '/3/{getline;sub(".\*","&\*");print}'

4\*

# seq 5 |awk '/3/{print;getline;sub(".\*","&\*")}{print}'

1

2

3

4\*

5

2）getline var

把a文件的行追加到b文件的行尾：

# cat a

a

b

c

# cat b

1 one

2 two

3 three

# awk '{getline line<"a";print $0,line}' b

1 one a

2 two b

3 three c

把a文件的行替换b文件的指定字段：

# awk '{getline line<"a";gsub($2,line,$2);print}' b

1 a

2 b

3 c

把a文件的行替换b文件的对应字段：

# awk '{getline line<"a";gsub("two",line,$2);print}' b

1 one

2 b

3 three

3）command | getline [var]

获取执行shell命令后结果的第一行：

# awk 'BEGIN{"seq 5"|getline var;print var}'

1

循环输出执行shell命令后的结果：

# awk 'BEGIN{while("seq 5"|getline)print}'

1

2

3

4

5

4）next

不打印匹配行：

# seq 5 |awk '{if($0==3){next}else{print}}'

1

2

4

5

删除指定行：

# seq 5 |awk 'NR==1{next}{print $0}'

2

3

4

5

如果前面动作成功，就遇到next，后面的动作不再执行，跳过。

或者：

# seq 5 |awk 'NR!=1{print}'

2

3

4

5

把第一行内容放到每行的前面：

# cat a

hello

1 a

2 b

3 c

# awk 'NR==1{s=$0;next}{print s,$0}' a

hello 1 a

hello 2 b

hello 3 c

# awk 'NR==1{s=$0}NF!=1{print s,$0}' a

hello 1 a

hello 2 b

hello 3 c

5）system()

执行shell命令判断返回值：

# awk 'BEGIN{if(system("grep root /etc/passwd &>/dev/null")==0)print "yes";else print "no"}'

yes

6）打印结果写到文件

# tail -n5 /etc/services |awk '{print $2 > "a.txt"}'

# cat a.txt

48049/tcp

48128/tcp

48128/udp

48129/tcp

48129/udp

7）管道连接shell命令

将结果通过grep命令过滤：

# tail -n5 /etc/services |awk '{print $2|"grep tcp"}'

48556/tcp

48619/tcp

49000/tcp

### 8.3.9 printf语句

格式化输出，默认打印字符串不换行。

格式：printf [format] arguments

|  |  |
| --- | --- |
| **Format** | **描述** |
| %s | 一个字符串 |
| %d，%i | 一个小数 |
| %f | 一个浮点数 |
| %.ns | 输出字符串，n是输出几个字符 |
| %m.nf | 输出浮点数，m是输出整数位数，n是输出的小数位数 |
| %x | 不带正负号的十六进制，使用a至f表示10到15 |
| %X | 不带正负号的十六进制，使用A至F表示10至15 |
| %% | 输出单个% |
| %-5s | 左对齐，对参数每个字段左对齐,宽度为5 |
| %-4.2f | 左对齐，宽度为4，保留两位小数 |
| %5s | 右对齐，不加横线表示右对齐 |

示例：

将换行符换成逗号：

# seq 5 |awk '{if($0!=5)printf "%s,",$0;else print $0}'

1,2,3,4,5

小括号中的5是最后一个数字。

输出一个字符：

# awk 'BEGIN{printf "%.1s\n","abc"}'

a

保留一个小数点：

# awk 'BEGIN{printf "%.2f\n",10/3}'

3.33

格式化输出：

# awk 'BEGIN{printf "user:%s\tpass:%d\n","abc",123}'

user:abc        pass:123

左对齐宽度10：

# awk 'BEGIN{printf "%-10s %-10s %-10s\n","ID","Name","Passwd"}'

ID         Name       Passwd

右对齐宽度10：

# awk 'BEGIN{printf "%10s %10s %10s\n","ID","Name","Passwd"}'

        ID       Name     Passwd

打印表格：

# vi test.awk

BEGIN{

print "+--------------------+--------------------+";

printf "|%-20s|%-20s|\n","Name","Number";

print "+--------------------+--------------------+";

}

# awk -f test.awk

+--------------------+--------------------+

|Name |Number |

+--------------------+--------------------+

格式化输出：

# awk -F: 'BEGIN{printf "UserName\t\tShell\n-----------------------------\n"}{printf "%-20s %-20s\n",$1,$7}END{print "END...\n"}' /etc/passwd

打印十六进制：

# awk 'BEGIN{printf "%x %X",123,123}'

7b 7B

### 8.3.10 自定义函数

格式：function name(parameter list) { statements }

示例：

# awk 'function myfunc(a,b){return a+b}BEGIN{print myfunc(1,2)}'

3

### 8.3.11 需求案例

1）分析Nginx日志

日志格式：'$remote\_addr - $remote\_user [$time\_local] "$request" $status $body\_bytes\_sent "$http\_referer" "$http\_user\_agent" "$http\_x\_forwarded\_for"'

统计访问IP次数：

# awk '{a[$1]++}END{for(v in a)print v,a[v]}' access.log

统计访问访问大于100次的IP：

# awk '{a[$1]++}END{for(v in a){if(a[v]>100)print v,a[v]}}' access.log

统计访问IP次数并排序取前10：

# awk '{a[$1]++}END{for(v in a)print v,a[v] |"sort -k2 -nr |head -10"}' access.log

统计时间段访问最多的IP：

# awk '$4>="[02/Jan/2017:00:02:00" && $4<="[02/Jan/2017:00:03:00"{a[$1]++}END{for(v in a)print v,a[v]}' access.log

统计上一分钟访问量：

# date=$(date -d '-1 minute' +%d/%d/%Y:%H:%M)

# awk -vdate=$date '$4~date{c++}END{print c}' access.log

统计访问最多的10个页面：

# awk '{a[$7]++}END{for(v in a)print v,a[v] |"sort -k1 -nr|head -n10"}' access.log

统计每个URL数量和返回内容总大小：

# awk '{a[$7]++;size[$7]+=$10}END{for(v in a)print a[v],v,size[v]}' access.log

统计每个IP访问状态码数量：

# awk '{a[$1" "$9]++}END{for(v in a)print v,a[v]}' access.log

统计访问IP是404状态次数：

# awk '{if($9~/404/)a[$1" "$9]++}END{for(i in a)print v,a[v]}' access.log

2）两个文件对比

找出b文件在a文件相同记录：

# seq 1 5 > a

# seq 3 7 > b

方法1：

# awk 'FNR==NR{a[$0];next}{if($0 in a)print $0}' a b

3

4

5

# awk 'FNR==NR{a[$0];next}{if($0 in a)print FILENAME,$0}' a b

b 3

b 4

b 5

# awk 'FNR==NR{a[$0]}NR>FNR{if($0 in a)print $0}' a b

3

4

5

# awk 'FNR==NR{a[$0]=1;next}(a[$0]==1)' a b   # a[$0]是通过b文件每行获取值，如果是1说明有

# awk 'FNR==NR{a[$0]=1;next}{if(a[$0]==1)print}' a b

3

4

5

方法2：

# awk 'FILENAME=="a"{a[$0]}FILENAME=="b"{if($0 in a)print $0}' a b

3

4

5

方法3：

# awk 'ARGIND==1{a[$0]=1}ARGIND==2 && a[$0]==1' a b

3

4

5

找出b文件在a文件不同记录：

方法1：

# awk 'FNR==NR{a[$0];next}!($0 in a)' a b

6

7

# awk 'FNR==NR{a[$0]=1;next}(a[$0]!=1)' a b

# awk 'FNR==NR{a[$0]=1;next}{if(a[$0]!=1)print}' a b

6

7

方法2：

# awk 'FILENAME=="a"{a[$0]=1}FILENAME=="b" && a[$0]!=1' a b

方法3：

# awk 'ARGIND==1{a[$0]=1}ARGIND==2 && a[$0]!=1' a b

3）合并两个文件

将a文件合并到b文件：

# cat a

zhangsan 20

lisi 23

wangwu 29

# cat b

zhangsan man

lisi woman

wangwu man

# awk 'FNR==NR{a[$1]=$0;next}{print a[$1],$2}' a b

zhangsan 20 man

lisi 23 woman

wangwu 29 man

# awk 'FNR==NR{a[$1]=$0}NR>FNR{print a[$1],$2}' a b

zhangsan 20 man

lisi 23 woman

wangwu 29 man

将a文件相同IP的服务名合并：

# cat a

192.168.1.1: httpd

192.168.1.1: tomcat

192.168.1.2: httpd

192.168.1.2: postfix

192.168.1.3: mysqld

192.168.1.4: httpd

# awk 'BEGIN{FS=":";OFS=":"}{a[$1]=a[$1] $2}END{for(v in a)print v,a[v]}' a

192.168.1.4: httpd

192.168.1.1: httpd tomcat

192.168.1.2: httpd postfix

192.168.1.3: mysqld

说明：数组a存储是$1=a[$1] $2，第一个a[$1]是以第一个字段为下标，值是a[$1] $2，也就是$1=a[$1] $2，值的a[$1]是用第一个字段为下标获取对应的值，但第一次数组a还没有元素，那么a[$1]是空值，此时数组存储是192.168.1.1=httpd，再遇到192.168.1.1时，a[$1]通过第一字段下标获得上次数组的httpd，把当前处理的行第二个字段放到上一次同下标的值后面，作为下标192.168.1.1的新值。此时数组存储是192.168.1.1=httpd tomcat。每次遇到相同的下标（第一个字段）就会获取上次这个下标对应的值与当前字段并作为此下标的新值。

4）将第一列合并到一行

# cat file

1 2 3

4 5 6

7 8 9

# awk '{for(i=1;i<=NF;i++)a[i]=a[i]$i" "}END{for(v in a)print a[v]}' file

1 4 7

2 5 8

3 6 9

说明：

for循环是遍历每行的字段，NF等于3，循环3次。

读取第一行时：

第一个字段：a[1]=a[1]1" " 值a[1]还未定义数组，下标也获取不到对应的值，所以为空，因此a[1]=1 。

第二个字段：a[2]=a[2]2" " 值a[2]数组a已经定义，但没有2这个下标，也获取不到对应的值，为空，因此a[2]=2 。

第三个字段：a[3]=a[3]3" " 值a[2]与上面一样，为空,a[3]=3 。

读取第二行时：

第一个字段：a[1]=a[1]4" " 值a[2]获取数组a的2为下标对应的值，上面已经有这个下标了，对应的值是1，因此a[1]=1 4

第二个字段：a[2]=a[2]5" " 同上，a[2]=2 5

第三个字段：a[3]=a[3]6" " 同上，a[2]=3 6

读取第三行时处理方式同上，数组最后还是三个下标，分别是1=1 4 7，2=2 5 8，3=3 6 9。最后for循环输出所有下标值。

5）字符串拆分，统计出现的次数

字符串拆分：

方法1：

# echo "hello world" |awk -F '' '{print $1}'

h

# echo "hello" |awk -F '' '{for(i=1;i<=NF;i++)print $i}'

h

e

l

l

o

方法2：

# echo "hello" |awk '{split($0,a,"''");for(v in a)print a[v]}'

l

o

h

e

l

统计字符串中每个字母出现的次数：

# echo "a.b.c,c.d.e" |awk -F '[.,]' '{for(i=1;i<=NF;i++)a[$i]++}END{for(v in a)print v,a[v]}'

a 1

b 1

c 2

d 1

e 1

6）统计平均成绩

# cat file

job 80

dave 84

tom 75

dave 73

job 72

tom 83

dave 88

# awk '{a[$1]+=$2;b[$1]++}END{for(i in a)print i,a[i]/b[i]}' file

job 76

dave 81.6667

tom 79

7）费用统计

# cat file

zhangsan 8000 1

zhangsan 5000 1

lisi 1000 1

lisi 2000 1

wangwu 1500 1

zhaoliu 6000 1

zhaoliu 2000 1

zhaoliu 3000 1

# awk '{name[$1]++;cost[$1]+=$2;number[$1]+=$3}END{for(v in name)print v,cost[v],number[v]}' file

zhangsan 5000 1

lisi 3000 2

wangwu 1500 1

zhaoliu 11000 3

8）获取数字字段最大值

# cat file

a b 1

c d 2

e f 3

g h 3

i j 2

获取第三字段最大值：

# awk 'BEGIN{max=0}{if($3>max)max=$3}END{print max}' file

3

打印第三字段最大行：

# awk 'BEGIN{max=0}{a[$0]=$3;if($3>max)max=$3}END{for(v in a)print v,a[v],max}' a

g h 3 3 3

e f 3 3 3

c d 2 2 3

a b 1 1 3

i j 2 2 3

# awk 'BEGIN{max=0}{a[$0]=$3;if($3>max)max=$3}END{for(v in a)if(a[v]==max)print v}' a

g h 3

e f 3

9）去除第一行和最后一行

# seq 5 |awk 'NR>2{print s}{s=$0}'

2

3

4

读取第一行，NR=1，不执行print s，s=1

读取第二行，NR=2，不执行print s，s=2 （大于为真）

读取第三行，NR=3，执行print s，此时s是上一次p赋值内容2，s=3

最后一行，执行print s，打印倒数第二行，s=最后一行

获取Nginx负载均衡配置端IP和端口：

# cat nginx.conf

upstream example-servers1 {

server 127.0.0.1:80 weight=1 max\_fails=2 fail\_timeout=30s;

}

upstream example-servers2 {

server 127.0.0.1:80 weight=1 max\_fails=2 fail\_timeout=30s;

server 127.0.0.1:82 backup;

}

# awk '/example-servers1/,/}/{if(NR>2){print s}{s=$2}}' nginx.conf

127.0.0.1:80

# awk '/example-servers1/,/}/{if(i>1)print s;s=$2;i++}' nginx.conf

# awk '/example-servers1/,/}/{if(i>1){print s}{s=$2;i++}}' nginx.conf

127.0.0.1:80

读取第一行，i初始值为0，0>1为假，不执行print s，x=example-servers1，i=1

读取第二行，i=1，1>1为假，不执行print s，s=127.0.0.1:80,i=2

读取第三行，i=2，2>1为真，执行print s，此时s是上一次s赋值内容127.0.0.1:80，i=3

最后一行，执行print s，打印倒数第二行，s=最后一行。

这种方式与上面一样，只是用i++作为计数器。

10）知道上述方式，就可以实现这种需求了，打印匹配行的上一行

# seq 5 |awk '/3/{print s}{s=$0}'

2

其他参考资料：<http://www.gnu.org/software/gawk/manual/gawk.html>

# 第八章 Shell标准输入、输出和错误

文件描述符（fd）：文件描述符是一个非负整数，在打开现存文件或新建文件时，内核会返回一个文件描述符，读写文件也需要使用文件描述符来访问文件。

内核为每个进程维护该进程打开的文件记录表。文件描述符只适于Unix、Linux操作系统。

## 8.1 标准输入、输出和错误

|  |  |  |
| --- | --- | --- |
| **文件描述符** | **描述** | **映射关系** |
| 0 | 标准输入，键盘 | /dev/stdin -> /proc/self/fd/0 |
| 1 | 标准输出，屏幕 | /dev/stdout -> /proc/self/fd/1 |
| 2 | 标准错误，屏幕 | /dev/stderr -> /proc/self/fd/2 |

## 8.2 重定向符号

|  |  |
| --- | --- |
| **符号** | **描述** |
| > | 符号左边输出作为右边输入（标准输出） |
| >> | 符号左边输出追加右边输入 |
| < | 符号右边输出作为左边输入（标准输入） |
| << | 符号右边输出追加左边输入 |
| & | 重定向绑定符号 |

输入和输出可以被重定向符号解释到shell。

shell命令是从左到右依次执行命令。

下面n字母是文件描述符。

## 8.3 重定向输出

1）覆盖输出

一般格式：[n]>word

如果n没有指定，默认是1

示例：

打印结果写到文件：

# echo "test" > a.txt

当没有安装bc计算器时，错误输出结果写到文件：

# echo "1 + 1" |bc 2 > error.log

2）追加重定向输出

一般格式：[n]>>word

如果n没有指定，默认是1

示例：

打印结果追加到文件：

# echo "test" >> a.txt

当没有安装bc计算器时，错误输出结果追加文件：

# echo "1 + 1" |bc 2 > error.log

## 8.4 重定向输入

一般格式：[n]<word

如果n没有指定，默认是0

示例：

a.txt内容作为grep输入：

# grep "test" --color < a.txt

## 8.5 重定向标准输出和标准错误

1）覆盖重定向标准输出和标准错误

&>word和>&word 等价于 >word 2>&1

&将标准输出和标准输入绑定到一起，重定向word文件。

示例：

当不确定执行对错时都覆盖到文件：

# echo "1 + 1" |bc &> error.log

当不确定执行对错时都覆盖到文件：

# echo "1 + 1" |bc > error.log 2>&1

2）追加重定向标准输出和标准错误

&>>word 等价于>>word 2>&1

示例：

当不确定执行对错时都追加文件：

# echo "1 + 1" |bc &>> error.log

将标准输出和标准输入追加重定向到delimiter：

<< delimiter

   here-document

delimiter

从当前shell读取输入源，直到遇到一行只包含delimiter终止，内容作为标准输入。

将eof标准输入作为cat标准输出再写到a.txt：

# cat << eof

123

abc

eof

123

abc

# cat > a.txt << eof

> 123

> abc

> eof

## 8.6 重定向到空设备

/dev/null是一个空设备，向它写入的数组都会丢弃，但返回状态是成功的。与其对应的还有一个/dev/zero设备，提供无限的0数据流。

在写Shell脚本时我们经常会用到/dev/null设备，将stdout、stderr输出给它，也就是我们不想要这些输出的数据。

通过重定向到/dev/null忽略输出，比如我们没有安装bc计算器，正常会抛出没有发现命令：

# echo "1 + 1" |bc >/dev/null 2>&1

这就让标准和错误输出到了空设备。

忽略标准输出：

# echo "test" >/dev/null

忽略错误输出：

# echo "1 + 1" |bc 2>/dev/null

## 8.7 read命令

read命令从标准输入读取，并把输入的内容复制给变量。

命令格式： read [-ers] [-a array] [-d delim] [-i text] [-n nchars] [-N nchars] [-p prompt] [-t timeout] [-u fd] [name ...]

|  |  |
| --- | --- |
| -e | 在一个交互shell中使用readline获取行 |
| -r | 不允许反斜杠转义任何字符 |
| -s | 隐藏输入 |
| -a array | 保存为数组，元素以空格分隔 |
| -d delimiter | 持续读取直到遇到delimiter第一个字符退出 |
| -n nchars | 读取nchars个字符返回，而不是等到换行符 |
| -p prompt | 提示信息 |
| -t timeout | 等待超时时间，秒 |
| -u fd | 指定文件描述符号码作为输入，默认是0 |
| name | 变量名 |

示例：

获取用户输入保存到变量：

# read -p "Please input your name: " VAR

Please input your name: lizhenliang

# echo $VAR

lizhenliang

用户输入保存为数组：

# read -p "Please input your name: " -a ARRAY

Please input your name: a b c

# echo ${ARRAY[\*]}

a b c

遇到e字符返回：

# read -d e VAR

123

456

e

# echo $VAR

123 456

从文件作为read标准输入：

# cat a.txt

adfasfd

# read VAR < a.txt

# echo $VAR

adfasfd

while循环读取每一行作为read的标准输入：

# cat a.txt |while read LINE; do echo $LINE; done

123

abc

分别变量赋值：

# read a b c

1 2 3

# echo $a

1

# echo $b

2

# echo $c

3

# echo 1 2 3 | while read a b c;do echo "$a $b $c"; done

1 2 3

# 第九章 Shell信号发送与捕捉

## 9.1 Linux信号类型

信号（Signal）：信号是在软件层次上对中断机制的一种模拟，通过给一个进程发送信号，执行相应的处理函数。

进程可以通过三种方式来响应一个信号：

1）忽略信号，即对信号不做任何处理，其中有两个信号不能忽略：SIGKILL及SIGSTOP。

2）捕捉信号。

3）执行缺省操作，Linux对每种信号都规定了默认操作。

Linux究竟采用上述三种方式的哪一个来响应信号呢？取决于传递给响应的API函数。

Linux支持的信号有：

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **信号名称** | **缺省动作** | **描述** |
| 1 | SIGHUP | 终止 | 终止进程，挂起 |
| 2 | SIGINT | 终止 | 键盘输入中断命令，一般是CTRL+C |
| 3 | SIGQUIT | CoreDump | 键盘输入退出命令，一般是CTRL+\ |
| 4 | SIGILL | CoreDump | 非法指令 |
| 5 | SIGTRAP | CoreDump | trap指令发出，一般调试用 |
| 6 | SIGABRT | CoreDump | abort(3)发出的终止信号 |
| 7 | SIGBUS | CoreDump | 非法地址 |
| 8 | SIGFPE | CoreDump | 浮点数异常 |
| 9 | SIGKILL | 终止 | 立即停止进程，不能捕获，不能忽略 |
| 10 | SIGUSR1 | 终止 | 用户自定义信号1，像Nginx就支持USR1信号，用于重载配置，重新打开日志 |
| 11 | SIGSEGV | CoreDump | 无效内存引用 |
| 12 | SIGUSR2 | 终止 | 用户自定义信号2 |
| 13 | SIGPIPE | 终止 | 管道不能访问 |
| 14 | SIGALRM | 终止 | 时钟信号，alrm(2)发出的终止信号 |
| 15 | SIGTERM | 终止 | 终止信号，进程会先关闭正在运行的任务或打开的文件再终止，有时候有的进程在运行任务会忽略此信号。不能捕捉 |
| 16 | SIGSTKFLT | 终止 | 处理器栈错误 |
| 17 | SIGCHLD | 可忽略 | 子进程结束时，父进程收到的信号 |
| 18 | SIGCONT | 可忽略 | 让终止的进程继续执行 |
| 19 | SIGSTOP | 停止 | 停止进程，不能忽略，不能捕获 |
| 20 | SIGSTP | 停止 | 停止进程，一般是CTRL+Z |
| 21 | SIGTTIN | 停止 | 后台进程从终端读数据 |
| 22 | SIGTTOU | 停止 | 后台进程从终端写数据 |
| 23 | SIGURG | 可忽略 | 紧急数组是否到达socket |
| 24 | SIGXCPU | CoreDump | 超出CPU占用资源限制 |
| 25 | SIGXFSZ | CoreDump | 超出文件大小资源限制 |
| 26 | SIGVTALRM | 终止 | 虚拟时钟信号，类似于SIGALRM，但计算的是进程占用的时间 |
| 27 | SIGPROF | 终止 | 类似与SIGALRM，但计算的是进程占用CPU的时间 |
| 28 | SIGWINCH | 可忽略 | 窗口大小改变发出的信号 |
| 29 | SIGIO | 终止 | 文件描述符准备就绪，可以输入/输出操作了 |
| 30 | SIGPWR | 终止 | 电源失败 |
| 31 | SIGSYS | CoreDump | 非法系统调用 |

CoreDump（核心转储）：当程序运行过程中异常退出时，内核把当前程序在内存状况存储在一个core文件中，以便调试。执行命令ulimit -c如果是0则没有开启，也不会生成core dump文件，可通过ulimit -c unlimited命令临时开启core dump功能，只对当前终端环境有效，如果想永久生效，可修改/etc/security/limites.conf文件，添加一行 "\* soft core unlimited"

默认生成的core文件保存在可执行文件所在的目录下，文件名为core。如果想修改core文件保存路径，可通过修改内核参数：echo "/tmp/corefile-%e-%p-%t" > /proc/sys/kernel/core\_pattern 则文件名格式为core-命名名-pid-时间戳

Linux支持两种信号：

一种是标准信号，编号1-31，称为非可靠信号（非实时），不支持队列，信号可能会丢失，比如发送多次相同的信号，进程只能收到一次，如果第一个信号没有处理完，第二个信号将会丢弃。

另一种是扩展信号，编号32-64，称为可靠信号（实时），支持队列，发多少次进程就可以收到多少次。

信号类型比较多，我们只要了解下，记住几个常用信号就行了，红色标记的我觉得需要记下。

发送信号一般有两种情况：

一种是内核检测到系统事件，比如键盘输入CTRL+C会发送SIGINT信号。

另一种是通过系统调用kill命令来向一个进程发送信号。

## 9.2 kill命令

kill命令发送信号给进程。

命令格式：kill [-s sigspec | -n signum | -sigspec] pid | jobspec ...

kill -l [sigspec]

-s # 信号名称

-n # 信号编号

-l # 打印编号1-31信号名称

示例：

给一个进程发送终止信号：

kill -s SIGTERM pid

或

kill -n 15 pid

或

kill -15 pid

或

kill -TREM pid

## 9.3 trap命令

trap命令定义shell脚本在运行时根据接收的信号做相应的处理。

命令格式：trap [-lp] [[arg] signal\_spec ...]

-l # 打印编号1-64编号信号名称

arg # 捕获信号后执行的命令或者函数

signal\_spec # 信号名或编号

一般捕捉信号后，做以下几个动作：

1）清除临时文件

2）忽略该信号

3）询问用户是否终止脚本执行

示例1：按CTRL+C不退出循环

#!/bin/bash

trap "" 2    # 不指定arg就不做任何操作，后面也可以写多个信号，以空格分隔

for i in {1..10}; do

   echo $i

   sleep 1

done

# bash a.sh

1

2

3

^C4

5

6

^C7

8

9

10

示例2：循环打印数字，按CTRL+C退出，并打印退出提示

#!/bin/bash

trap "echo 'exit...';exit" 2

for i in {1..10}; do

   echo $i

   sleep 1

done

# bash test.sh

1

2

3

^Cexit...

示例3：让用户选择是否终止循环

#!/bin/bash

trap "func" 2

func() {

   read -p "Terminate the process? (Y/N): " input

   if [ $input == "Y" ]; then

        exit

   fi

}

for i in {1..10}; do

   echo $i

   sleep 1

done

# bash a.sh

1

2

3

^CTerminate the process? (Y/N): Y

# bash a.sh

1

2

3

^CTerminate the process? (Y/N): N

4

5

6

...

# 第十章 Shell编程时常用的系统文件

## 10.1 Linux系统目录结构

|  |  |
| --- | --- |
| / | 根目录，所有文件的第一级目录 |
| /home | 普通用户家目录 |
| /root | 超级用户家目录 |
| /usr | 用户命令、应用程序等目录 |
| /var | 应用数据、日志等目录 |
| /lib | 库文件和内核模块目录 |
| /etc | 系统和软件配置文件 |
| /bin | 可执行程序目录 |
| /boot | 内核加载所需的文件，grub引导 |
| /dev | 设备文件目录，比如磁盘驱动 |
| /tmp | 临时文件目录 |
| /opt | 第三方软件安装目录 |

## 10.2 环境变量文件

**系统级**

系统级变量文件对所有用户生效。

/etc/profile   # 系统范围内的环境变量和启动文件。不建议把要做的事情写在这里面，最好创建一个自定义的，放在/etc/profile.d下

/etc/bashrc   # 系统范围内的函数和别名

**用户级**

用户级变量文件对自己生效，都在自己家目录下。

~/.bashrc      # 用户指定别名和函数

~/.bash\_logout   # 用户退出执行

~/.bash\_profile  # 用户指定变量和启动程序

~/.bash\_history  # 用户执行命令历史文件

开启启动脚本顺序：/etc/profile -> /etc/profile.d/\*.sh -> ~/.bash\_profile -> ~/.bashrc -> /etc/bashrc

因此，我们可以把写的脚本放到以上文件里执行。

## 10.3 系统配置文件

|  |  |
| --- | --- |
| /etc/issue | 系统版本 |
| /etc/hosts | 主机名与IP对应关系 |
| /etc/resolv.conf | DNS服务器地址 |
| /etc/hostname | 主机名 |
| /etc/sysctl.conf | 系统参数配置文件 |
| /etc/sudoers | sudo权限配置 |
| /etc/init.d | 服务启动脚本 |
| /etc/sysconfig/network-scripts | 网卡信息配置目录 |
| /etc/rc.d/rc.local | 系统init初始化完后执行，不建议将启动服务写在这里面，应创建自己的systemd或udev |
| /etc/fstab | 硬盘自动挂载配置 |
| /etc/inittab | 系统启动运行级别 |
| /etc/crontab | 系统级任务计划 |
| /var/spool/cron | 用户级任务计划，此目录下以用户名命名对应每个用户的任务计划 |
| /etc/cron.d | 描述计算机任务计划 |
| /etc/hosts.allow | TCP包访问列表 |
| /etc/hosts.deny | TCP包拒绝列表 |
| /usr/share/doc | 各软件的文档 |
| /etc/sshd\_config | SSH服务配置文件 |
| /var/log | 系统和应用程序日志目录 |
| /var/spool/mail | 邮件目录 |

crontab任务计划说明：

# Example of job definition:

# .---------------- minute (0 - 59)

# |  .------------- hour (0 - 23)

# |  |  .---------- day of month (1 - 31)

# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...

# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat

# |  |  |  |  |

# \*  \*  \*  \*  \* user-name command to be executed

## 10.4 /dev目录

/dev目录下存放的是一些设备文件。

|  |  |
| --- | --- |
| /dev/hd[a-t] | IDE设备 |
| /dev/sd[a-z] | SCSI设备 |
| /dev/dm-[-9] | LVM逻辑磁盘 |
| /dev/null | 黑洞 |
| /dev/zero | 无限0数据流 |

## 10.5 /proc目录

/proc是一个虚拟目录，在Linux系统启动后生成的，数据存储在内存中，存放内核运行时的参数、网络信息、进程状态等等。

### 10.5.1 /proc

|  |  |
| --- | --- |
| /proc/[0-9]+ | 此目录下数字命名的目录是运行进程信息，目录名为PID |
| /proc/meminfo | 物理内存、交换空间等信息，free |
| /proc/loadavg | 系统负载 |
| /proc/uptime | 系统运行时间  计算系统启动时间：  date -d "$(awk -F. '{print $1}' /proc/uptime) second ago" +"%Y-%m-%d %H:%M:%S"  或 who -b |
| /proc/cpuinfo | CPU信息 |
| /proc/modules | 系统已加载的模块或驱动，lsmod |
| /proc/mounts | 文件系统挂载信息，mount |
| /proc/swaps | swap分区信息 |
| /proc/partitions | 系统分区信息 |
| /proc/version | 内核版本 |
| /proc/stat | CPU利用率，磁盘，内存页 |
| /proc/devices | 可用的设备列表 |

### 10.5.2 /proc/net

/proc/net 目录存放的是一些网络协议信息。

|  |  |
| --- | --- |
| /proc/net/tcp | TCP状态连接信息，netstat |
| /proc/net/udp | UDP状态连接信息 |
| /proc/net/arp | arp信息表 |
| /proc/net/dev | 网卡流量 |
| /proc/net/snmp | 网络传输协议的收发包信息 |
| /proc/net/sockstat | socket使用情况，比如已使用，正在使用 |
| /proc/net/netstat | 网络统计数据，netstat -s |
| /proc/net/route | 路由表 |

### 10.5.3 /proc/sys

这个目录下的文件可被读写，存了大多数内核参数，可以修改改变内核行为。所以修改这些文件要特别小心，修改错误可能导致内核不稳定。

有四个主要的目录：

fs      # 文件系统各方面信息，包括配额、文件句柄、inode和目录项。

kernel   # 内核行为的信息

net     # 网络配置信息，包括以太网、ipx、ipv4和ipv6。

vm      # Linux内核的虚拟内存子系统，通常称为交换空间。

|  |  |
| --- | --- |
| /proc/sys/fs/file-max | 内核分配所有进程最大打开文件句柄数量，可适当增加此值 |
| /proc/sys/fs/file-nr | 只读，第一个值已分配的文件句柄数量，第二个值分配没有使用文件句柄数量，第三个值文件句柄最大数量。lsof |
| /proc/sys/kernel/ctrl-alt-del | 组合键重启计算机，只为0同步缓冲区到磁盘，1为不同步 |
| /proc/sys/kernel/domainname | 配置系统域名 |
| /proc/sys/kernel/exec-shield | 配置内核执行保护功能，防止某类型缓冲区溢出攻击。0为禁用，1开启 |
| /proc/sys/kernel/hostname | 配置系统主机名 |
| /proc/sys/kernel/osrelease | 内核版本号 |
| /proc/sys/kernel/ostype | 操作系统类型 |
| /proc/sys/kernel/shmall | 设置共享内存的总量，以字节为单位 |
| /proc/sys/kernel/shmmax | 设置最大共享内存段 |
| /proc/sys/kernel/shmmni | 设置共享内存段最大数量 |
| /proc/sys/kernel/threads-max | 设置最大允许线程数量 |
| /proc/sys/kernel/pid\_max | 设置最大允许创建的pid数量 |
| /proc/sys/kernel/version | 显示最后一次编译内核时间 |
| /proc/sys/kernel/random/uuid | 生成uuid |
| /proc/sys/kernel/core\_pattern | 控制生成core dump文件位置和保存格式 |
| /proc/sys/net/core/netdev\_max\_backlog | 设置数据包队列允许最大数量 |
| /proc/sys/net/core/optmem\_max | 设置socket允许最大缓冲区大小 |
| /proc/sys/net/core/somaxconn | 每个端口最大监听队列长度 |
| /proc/sys/net/core/rmem\_default | 设置socket接收默认缓冲区大小，单位字节 |
| /proc/sys/net/core/rmem\_max | 设置socket接收最大缓冲区大小 |
| /proc/sys/net/core/wmem\_default | 设置socket发送默认缓冲区大小 |
| /proc/sys/net/core/wmem\_max | 设置socket发送最大缓冲区大小 |
| /proc/sys/net/ipv4/icmp\_echo\_ignore\_all和icmp\_echo\_ignore\_broadcasts | 设置是否忽略icmp响应包和广播包，0为不忽略，1为忽略 |
| /proc/sys/net/ipv4/ip\_default\_ttl | 设置默认生存时间 |
| /proc/sys/net/ipv4/ip\_forward | 允许系统接口转发数据包，默认0为关闭，1为开启 |
| /proc/sys/net/ipv4/ip\_local\_port\_range | 指定使用本地TCP或UDP端口范围，第一个值最低，第二个值最高 |
| /proc/sys/net/ipv4/tcp\_syn\_retries | 限制重新发送syn尝试建立连接次数 |
| /proc/sys/net/ipv4/tcp\_synack\_retries | syn ack确认包尝试次数 |
| /proc/sys/net/ipv4/tcp\_syncookies | 是否启用syn cookie，0为关闭，默认1为开启 |
| /proc/sys/net/ipv4/tcp\_max\_tw\_buckets | 系统保持TIME\_WAIT最大数量 |
| /proc/sys/net/ipv4/tcp\_tw\_recycle | 是否启用TIME\_WAIT快速收回，默认0为关闭，1为开启 |
| /proc/sys/net/ipv4/tcp\_tw\_reuse | 是否启用TIME\_WAIT复用，默认0为关闭，1为开启 |
| /proc/sys/net/ipv4/tcp\_keepalive\_time | TCP连接保持时间（默认2小时），当连接活动，定时器会重新复位。 |
| /proc/sys/vm/swappiness | 内核按此值百分比来使用swap，值越小越不考虑使用物理内存，0为尽可能不使用swap |
| /proc/sys/vm/overcommit\_memory | 控制内存分配，默认0为内核先评估可用内存，如果足够允许申请，否则拒绝，1为允许分配所有物理内存，2为允许分配超过物理内存和交换空间总和的内存 |
| /proc/sys/vm/overcommit\_ratio | 指定物理内存比率，当overcommit\_memory=2时，用户空间进程可使用的内存不超过物理内存\*overcommit\_ratio+swap |

参考资料：

<https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/s2-proc-dir-sys.html>

# 第十一章 Shell常用命令与工具

本章节学习一些在编写Shell时的常用命令或工具及使用技巧。有人说Shell脚本是命令堆积的一个文件，按顺序去执行。还有人说想学好Shell脚本，要把Linux上各种常见的命令或工具掌握了，这些说法都没错。由于Shell语言本身在语法结构上比较简单，是面向过程编程，想实现复杂的功能有点强人所难！而且Shell本身又工作在Linux内核之上，在用户态调用Linux命令会很方面，所以大多数情况下我们都是依靠这些命令来完成脚本中的某些功能，比如文本处理、获取系统状态等等，然后通过Shell语法结构组织代码逻辑。不管是学Linux系统好还是写Shell脚本也好，有些命令都是必须要会的，以下是根据个人经验总结的一些常用的命令。

怎么更好的学习命令呢？

当然查看官方帮助文档了，可以通过man cmd、cmd --help、help cmd、info cmd等方式查看命令的使用。

## 11.1 ls

功能：列出目录内容

常用选项：

-a  显示所有文件，包括隐藏的

-l  长格式列出信息

-i  显示文件inode号

-t  按修改时间排序

-r  按修改时间倒序排序

-h 打印易读大小单位

示例：

按修改时间排序：

# ls -t

按修改时间倒序排序：

# ls -rt

长格式列出：

# ls -lh

查看文件inode：

# ls -i file

## 11.2 echo

功能：打印一行

常用选项：

-n  不加换行符

-e  解释转义符

示例：

解释换行符：

# echo -e "1\n2\n3"

1

2

3

## 11.3 printf

功能：格式化打印数据。默认打印字符串不换行。

格式：printf format [arguments]

|  |  |
| --- | --- |
| **Format** | **描述** |
| %s | 一个字符串 |
| %d，%i | 一个小数 |
| %f | 一个浮点数 |
| %.ns | 输出字符串，n是输出几个字符 |
| %m.nf | 输出浮点数，m是输出整数位数，n是输出的小数位数 |
| %x | 不带正负号的十六进制，使用a至f表示10到15 |
| %X | 不带正负号的十六进制，使用A至F表示10至15 |
| %% | 输出单个% |
| %-5s | 左对齐，对参数每个字段左对齐,宽度为5 |
| %-4.2f | 左对齐，宽度为4，保留两位小数 |
| %5s | 右对齐，不加横线表示右对齐 |

一些常用的空白符：

\n  换行

\r  回车

\t  水平制表符

示例：

输出一个字符：

# printf "%.1s" abc

a

保留一个小数点：

# printf "%.1f" 1.333

1.3

输出换行：

# printf "%.1f\n" 1.333

1.3

格式化输出：

# printf "user: %s\tpass: %d\n" abc 123

user: abc pass: 1

左对齐宽度10：

# printf "%-10s %-10s %-10s\n" ID Name Number

ID         Name       Number

右对齐宽度10：

# printf "%10s %10s %10s\n" ID Name Number

        ID       Name     Number

每段对齐：

# printf "%10s\n" ID Name Number

        ID

      Name

    Number

# printf "%-10s\n" ID Name Number

ID

Name

Number

## 11.4 cat

功能：连接文件和标准输出打印

常用选项：

-b  显示非空行行号

-n  显示所有行行号

-T  显示tab，用^I表示

-E  显示以$结尾

示例：

连接两个文件：

# cat a b

# cat << EOF

> 123

> abc

> EOF

123

abc

将eof标准输入作为cat标准输出再写到a.txt：

# cat > a.txt << eof

> 123

> abc

> eof

## 11.5 tac

功能：连接文件和倒序打印文件

常用选项：

示例：

倒序打印每一行：

# tac a.txt

## 11.6 rev

功能：反向打印每一行

常用选项：

示例：

# echo "123" |rev

321

## 11.7 wc

功能：统计文件行数、字节、字符数

常用选项：

-c  打印文件字节数，一个英文字母1字节，一个汉字占2-4字节（根据编码）

-m  打印文件字符数，一个汉字占2个字符

-l  打印多少行

-L 打印最长行的长度，也可以统计字符串长度

示例：

统计文件多少行：

# wc -l file

统计字符串长度：

# echo "hello" |wc -L

5

## 11.8 cp

功能：复制文件或目录

常用选项：

-a  归档

-b  目标文件存在创建备份，备份文件是文件名跟~

-f  强制复制文件或目录

-r  递归复制目录

-p  保留原有文件或目录属性

-i  覆盖文件之前先询问用户

-u  当源文件比目的文件修改时间新时才复制

-v  显示复制信息

示例：

复制目录：

# cp -rf test /opt

## 11.9 mkdir

功能：创建目录

常用选项：

-p 递归创建目录

-v  显示创建过程

示例：

创建多级目录：

# mkdir /opt/test/abc

创建多个目录：

# mkdir {install,tmp}

创建连续目录：

# mkdir {a..c}

## 11.10 mv

功能：移动文件或重命名

常用选项：

-b  目标文件存在创建备份，备份文件是"文件名后跟~"

-u  当源文件比目的文件修改时间新时才移动

-v  显示移动信息

示例：

移动文件：

# mv a.txt /opt

重命名文件：

# mv a.txt b.txt

## 11.11 rename

功能：重命名文件，支持通配符

常用选项：

示例：批量命名文件

将foo1-foo9替换为foo01-foo09：

# rename foo foo0 foo?

将以.htm后缀的文件替换为.html：

# rename .htm .html \*.htm

## 11.12 dirname

功能：去除路径的最后一个名字

常用选项：

示例：

# dirname /usr/bin/

/usr

# dirname dir1/str dir2/str

dir1

dir2

# dirname stdio.h

.

## 11.13 basename

功能：打印路径的最后一个名字

常用选项：

-a  支持多个参数

-s  删除后面的后缀

示例：

# basename /usr/bin/sort

sort

# basename include/stdio.h .h

stdio

# basename -s .h include/stdio.h

stdio

# basename -a any/str1 any/str2

str1

str2

## 11.14 du

功能：估算文件磁盘空间使用

常用选项：

-h  易读格式显示（K，M，G）

-b  单位bytes显示

-k  单位KB显示

-m  单位MB显示

-s  只显示总大小

--max-depth=<目录层数>，超过层数的目录忽略

--exclude=file 排除文件或目录

--time  显示大小和创建时间

示例：

查看目录大小：

# du -sh /opt

排除目录某个文件：

# du -sh --exclude=test /opt

## 11.15 cut

功能：选取文件的每一行数据

常用选项：

-b  选中第几个字符

-c  选中多少个字符

-d  指定分隔符分字段，默认是空格

-f  显示选中字段

示例：

打印b字符：

# echo "abc" |cut -b "2"

b

截取abc字符：

# echo "abcdef" |cut -c 1-3

abc

以冒号分隔，显示第二个字段：

# echo "a:b:c" |cut -d: -f2

b

## 11.16 tr

功能：替换或删除字符

格式：Usage: tr [OPTION]... SET1 [SET2]

常用选项：

-c  替换SET1没有SET2的字符

-d  删除SET1中字符

-s  压缩SET1中重复的字符

-t  将SET1用SET2转换，默认

示例：

替换SET1没有SET2的字符：

# echo "aaabbbccc" | tr -c c 1

111111ccc

去重字符：

# echo "aaacccddd" | tr -s '[a-z]'

acd

删除字符：

# echo "aaabbbccc" | tr -d bbb

aaaccc

删除换行符：

# echo -e "a\nb\nc" | tr -d '\n'

abc

替换字符：

# echo "aaabbbccc" | tr '[a-z]' '[A-Z]'

AAABBBCCC

## 11.17 stat

功能：显示文件或文件的系统状态

常用选项：

-Z 显示selinux安全上下文

-f  显示文件系统状态

-c  指定格式输出内容

-t  以简洁的形式打印

示例：

显示文件信息：

# stat file

只显示文件修改时间：

# stat -c %y file

## 11.18 seq

功能：打印序列化数字

常用选项：

-f  使用printf样式格式

-s  指定换行符，默认是\n

-w  等宽，用0填充

示例：

数字序列：

# seq 3

1

2

3

带0的数字序列：

# seq -w 03

01

02

03

范围数字序列：

# seq 2 5

2

3

4

5

步长序列：

# seq 1 2 5   # 2是步长

1

3

5

以冒号分隔序列：

# seq -s "+" 5

1+2+3+4+5

等宽并在数字前面加字符串：

# seq -f "str%02g" 3  # %g是默认数字位数，02是数字不足2位时用0填充。

str01

str02

str03

## 11.19 shuf

功能：生成随机序列

常用选项：

-i  输出数字范围

-o  结果写入文件

示例：

输出范围随机数：

# seq 5 |shuf

2

1

5

4

3

# shuf -i 5-10

8

10

7

9

6

5

## 11.20 sort

功能：排序文本，默认对整列有效

常用选项：

-f  忽略字母大小写

-M  根据月份比较，比如JAN、DEC

-h  根据易读的单位大小比较，比如2K、1G

-g 按照常规数值排序

-n  根据字符串数值比较

-r  倒序排序

-k  位置1,位置2 根据关键字排序，在从第位置1开始，位置2结束

-t  指定分隔符

-u  去重重复行

-o  将结果写入文件

示例：

随机数字排序：

# seq 5 |shuf |sort

随机字母排序：

# printf "%c\n" {a..f} |shuf |sort

倒序排序：

# seq 5 |shuf |sort -r

分隔后的字段排序：

# cat /etc/passwd |sort -t : -k 3 -n

去重重复行：

# echo -e "1\n1\n2\n3\n3" |sort -u

大小单位排序：

# du -h |sort -k 1 -h -r

分隔后第一个字段的第二个字符排序：

# echo -e "fa:1\neb:2\ncc:3" |sort -t : -k 1.2

tab作为分隔符：

# sort -t $"\t"

file文件内容：

zhangsan 6 100

lisi 8 80

wangwu 7 90

zhaoliu 9 70

对file文件的第二列正序排序，再次基础再对第三列倒序排序（多列排序）：

~~# sort -k 2,2 -n -k 3,3 -nr file~~

# sort -k 2 -n -k 3 -nr file

zhaoliu 9 70

lisi 8 80

wangwu 7 90

zhangsan 6 100

对两个文件同时排序：

# sort file1 file2

## 11.21 uniq

功能：去除重复行,只会统计相邻的

常用选项：

-c  打印出现的次数

-d  只打印重复行

-u 只打印不重复行

-D  只打印重复行，并且把所有重复行打印出来

-f N 比较时跳过前N列

-i  忽略大小写

-s N 比较时跳过前N个字符

-w N 对每行第N个字符以后内容不做比较

示例：

测试文本如下：

# cat file

abc

cde

xyz

cde

xyz

abd

去重复行：

# sort file |uniq

abc

abd

cde

xyz

打印每行重复次数：

# sort file |uniq -c

1 abc

1 abd

2 cde

2 xyz

打印不重复行：

# sort file |uniq -u

abc

abd

打印重复行：

# sort file |uniq -d

cde

xyz

打印重复行并统计出现次数：

# sort file |uniq -d -c

2 cde

2 xyz

根据前几个字符去重：

# sort file |uniq -w 2

abc

cde

xyz

## 11.22 tee

功能：从标准输入读取写到标准输出和文件

常用选项：

-a  追加到文件

示例：

打印并追加到文件：

# echo 123 |tee -a a.log

## 11.23 join

功能：连接两个文件

常用选项：

-i  忽略大小写

-o  按照指定文件栏位显示

-t  使用字符作为输入和输出字段分隔符

示例：

# cat file1

1 a

2 b

3 c

# cat file2

1 x

2 y

3 z

将两个文件相同字段合并一列：

# join file1 file2

1 a x

2 b y

3 c z

打印file1第二列和file2第二列：

# join -o 1.2 2.2 file1 file2

a x

b y

c z

# join -t ':' -o 1.1 2.1 /etc/passwd /etc/shadow

user1:user1

……

## 11.24 paste

功能：合并文件

常用选项：

-d  指定分隔符，默认是tab键分隔

-s  将文件内容平行合并，默认tab键分隔

示例：

# seq 1 3 > file1

# seq 4 6 > file2

两个文件合并：

# paste file1 file2

1 4

2 5

3 6

两个文件合并，+号分隔：

# paste -d "+" file1 file2

1+4

2+5

3+6

文件内容平行显示：

# paste -s file1 file2

1 2 3

4 5 6

## 11.25 head

功能：输出文件的前几行

常用选项：

-c  打印前多少K，M

-n  打印前多少行

示例：

打印文件前50行：

# head -n 50 file

## 11.26 tail

功能：输出文件的后几行

常用选项：

-c  打印后多少K，M

-f  实时读文件，随着文件输出附加输出

-n  输出最后几行

--pid  与-f一起使用，表示pid死掉后结束

-s  与-f一起使用，表示休眠多少秒输出

示例：

打印文件后50行：

# tail -n 50 file

实时输出新增行：

# tail -f file

## 11.27 find

功能：目录层次结构中搜索文件

格式：find path -option actions

常用选项：

-name  文件名，支持(‘\*’, ‘?’)

-type  文件类型，d目录，f常规文件等

-perm  符合权限的文件，比如755

-atime -/+n  在n天以内/过去n天被访问过

-ctime -/+n  在n天以内/过去n天被修改过

-amin -/+n   在n天以内/过去n分钟被访问过

-cmin -/+n   在n天以内/过去n分钟被修改过

-size -/+n   文件大小小于/大于，b、k、M、G

-maxdepth levels  目录层次显示的最大深度

-regex pattern   文件名匹配正则表达式模式

-inum 通过inode编号查找文件

动作：

-detele  删除文件

-exec command {} \;  执行命令，花括号代表当前文件

-ls   列出当前文件，ls -dils格式

-print 完整的文件名并添加一个回车换行符

-print0  打印完整的文件名并不添加一个回车换行符

-printf format  打印格式

其他字符：

！ 取反

-or/-o 逻辑或

-and 逻辑和

示例：

查找文件名：

# find / -name "\*http\*"

查找文件名并且文件类型：

# find /tmp -name core -type f -print

查找文件名并且文件类型删除：

# find /tmp -name core -type f -delete

查找当前目录常规文件并查看文件类型：

# find . -type f -exec file '{}' \;

查找文件权限是664：

# find . -perm 664

查找大于1024k的文件：

# find . -size -1024k

查找3天内修改的文件：

# find /bin -ctime -3

查找3分钟前修改的文件：

# find /bin -cmin +3

排除多个类型的文件：

# find . ! -name "\*.sql" ! -name "\*.txt"

或条件查找多个类型的文件：

# find . -name '\*.sh' -o -name '\*.bak'

# find . -regex ".\*\.sh\|.\*\.bak"

# find . -regex ".\*\.\(sh\|bak\)"

并且条件查找文件：

# find . -name "\*.sql" -a -size +1024k

只显示第一级目录：

# find /etc -type d -maxdepth 1

通过inode编号删除文件：

# rm `find . -inum 671915`

# find . -inum 8651577 -exec rm -i {} \;

## 11.28 xargs

功能：从标准输入执行命令

常用选项：

-a file  从指定文件读取数据作为标准输入

-0  处理包含空格的文件名,print0

-d  delimiter  分隔符，默认是空格分隔显示

-i  标准输入的结果以{}代替

-I  标准输入的结果以指定的名字代替

-t  显示执行命令

-p  交互式提示是否执行命令

-n  最大命令行参数

--show-limits  查看系统命令行长度限制

示例：

删除/tmp下名字是core的文件：

# find /tmp -name core -type f -print | xargs /bin/rm -f

# find /tmp -name core -type f -print0 | xargs -0 /bin/rm -f

列转行（去除换行符 ）：

# cut -d: -f1 < /etc/passwd | sort | xargs echo

行转列：

# echo "1 2 3 4 5" |xargs -n1

最长两列显示：

# echo "1 2 3 4 5" |xargs -n2

创建未来十天时间：

# seq 1 10 |xargs -i date -d "{} days " +%Y-%m-%d

复制多个目录：

# echo dir1 dir2 |xargs -n1 cp a.txt

清空所有日志：

# find ./ -name "\*.log" |xargs -i tee {} # echo ""> {} 这样不行，>把命令中断了

rm在删除大量的文件时，会提示参数过长，那么可以使用xargs删除：

# ls |xargs rm –rf

或分配删除 rm [a-n]\* -rf   # getconf ARG\_MAX 获取系统最大参数限制

## 11.29 nl

功能：打印文件行号

常用选项：

-b <a|t>   指定行号显示方式，a表示所有行都打印行号，b表示空行不显示行号，默认是a

-n <ln|rn|rz>  行号显示方法，ln左对齐，rn右对齐，rz右边显示，左边空白用0填充。

-w   行号栏位在左边占用的宽度

示例：

打印行号，空行不显示：

# nl a.txt

左对齐打印行号：

# nl -n ln a.txt

行号右移动五个空格：

# nl -w 5 a.txt

## 11.30 date

功能：打印或设置系统日期和时间

常用选项：

-d string  显示指定字符串所描述的时间，而非当前时间

-f datefile 从日期文件中按行读入时间描述

-I  输出ISO 8601格式的日期和时间

-r  显示文件的最后修改时间

-R  输出RFC 2822格式的日期和时间

-s string  设置时间所描述的字符串

-u  打印或设置UTC时间

控制输出格式：

%% 一个文字的 %

%a 当前locale 的星期名缩写(例如： 日，代表星期日)

%A 当前locale 的星期名全称 (如：星期日)

%b 当前locale 的月名缩写 (如：一，代表一月)

%B 当前locale 的月名全称 (如：一月)

%c 当前locale 的日期和时间 (如：2005年3月3日 星期四 23:05:25)

%C 世纪；比如 %Y，通常为省略当前年份的后两位数字(例如：20)

%d 按月计的日期(例如：01)

%D 按月计的日期；等于%m/%d/%y

%e 按月计的日期，添加空格，等于%\_d

%F 完整日期格式，等价于 %Y-%m-%d

%g ISO-8601 格式年份的最后两位 (参见%G)

%G ISO-8601 格式年份 (参见%V)，一般只和 %V 结合使用

%h 等于%b

%H 小时(00-23)

%I 小时(00-12)

%j 按年计的日期(001-366)

%k 时(0-23)

%l 时(1-12)

%m 月份(01-12)

%M 分(00-59)

%n 换行

%N 纳秒(000000000-999999999)

%p 当前locale 下的"上午"或者"下午"，未知时输出为空

%P 与%p 类似，但是输出小写字母

%r 当前locale 下的 12 小时时钟时间 (如：11:11:04 下午)

%R 24 小时时间的时和分，等价于 %H:%M

%s 自UTC 时间 1970-01-01 00:00:00 以来所经过的秒数

%S 秒(00-60)

%t 输出制表符 Tab

%T 时间，等于%H:%M:%S

%u 星期，1 代表星期一

%U 一年中的第几周，以周日为每星期第一天(00-53)

%V ISO-8601 格式规范下的一年中第几周，以周一为每星期第一天(01-53)

%w 一星期中的第几日(0-6)，0 代表周一

%W 一年中的第几周，以周一为每星期第一天(00-53)

%x 当前locale 下的日期描述 (如：12/31/99)

%X 当前locale 下的时间描述 (如：23:13:48)

%y 年份最后两位数位 (00-99)

%Y 年份

示例：

设置系统日期和时间：

# date -s "2016-12-15 00:00:00"

查看当前系统时间戳：

# date +%s

查看当前系统时间：

# date +'%F %T'

把日期和时间转换成时间戳：

# date -d "2016-12-15 18:00:00" +%s

把时间戳转成时间：

# date -d '@1481842800' '+%F %T'

时间加减：

显示前30秒：date -d '-30 second' +'%F %T'

显示前一分钟：date -d '-1 minute' +'%F %T'

显示前一个时间：date -d '-1 hour' +'%F %T'

显示前一个天：date -d '-1 day' +'%F %T'

显示上一周：date -d '-1 week' +'%F %T'

显示上一个月日期：date -d '-1 month' +%F

显示上一年日期：date -d '-1 year' +%F

或

显示前一天日期：date -d yesterday +%F

显示后一天日期：date -d tomorrow +%F

时间比较：

NOW\_DATE=$(date +%s)

AGO\_DATE=$(date -d "2016-12-15 18:00:00" +%s)

[ $NOW\_DATE -gt $AGO\_DATE ] && echo yes || echo no

## 11.31 wget

功能：非交互式网络下载，类似于HTTP客户端

常用选项：

-b,  --background   后台运行

日志记录和输入文件：

-o,  --output-file=FILE   日志写到文件

-a,  --append-output=FILE 日志追加到文件

-d,  --debug   打印debug信息，会包含头信息

-q,  --quiet   退出，不输出

-i,  --input-file=FILE   从文件中读取URL下载

下载选项：

-t,  --tries=NUMBER   设置链接重试次数

-O,  --output-document=FILE  写入内容到文件

-nc, --no-clobber   跳过下载现有的文件

-c,  --continue 断点续传

--progress=TYPE   设置进度条（dot和bar）

-S,  --server-response   打印服务器响应头信息

--spider   不下载任何内容

-T,  --timeout=SECONDS 设置相应超时时间（还有--dns-timeout、--connect-timeout和--read-timeout）

-w,  --wait=SECONDS   两次重试间隔等待时间

--bind-address=ADDRESS   设置绑定地址

--limit-rate=RATE   限制下载速度

--user=USER   设置ftp和http用户名

--password=PASS   设置ftp和http密码

目录：

-P,  --directory-prefix=PREFIX  保存文件目录

HTTP选项：

--http-user=USER   设置http用户名

--http-password=PASS   设置http密码

--proxy-user=USER 设置代理用户名

--proxy-password=PASS  设置代理密码

--referer=URL   设置Referer

--save-headers  保存头到文件

--default-page=NAME   改变默认页面名字，默认index.html

-U,--user-agent=AGENT  设置客户端信息

--no-http-keep-alive   禁用HTTP keep-alive（长连接）

--load-cookies=FILE   从文件加载cookies

--save-cookies=FILE   保存cookies到文件

--post-data=STRING   使用POST方法，发送数据

FTP选项：

--ftp-user=USER   设置ftp用户名

--ftp-password=PASS   设置ftp密码

--no-passive-ftp   禁用被动传输模式

递归下载：

-r, --recursive   指定递归下载

-l, --level=NUMBER   最大递归深度

-A, --accept=LIST   逗号分隔下载的扩展列表

-R, --reject=LIST 逗号分隔不被下载的扩展列表

-D, --domains=LIST   逗号分隔被下载域的列表

--exclude-domains=LIST 排除不被下载域的列表

示例：

下载单个文件到当前目录：

# wget http://nginx.org/download/nginx-1.11.7.tar.gz

放到后台下载：

# wget -b http://nginx.org/download/nginx-1.11.7.tar.gz

对于网络不稳定的用户使用-c和--tries参数，保证下载完成，并下载到指定目录：

# wget -t 3 -c http://nginx.org/download/nginx-1.11.7.tar.gz -P down

不下载任何内容，判断URL是否可以访问：

# wget --spider http://nginx.org/download/nginx-1.11.7.tar.gz

下载内容写到文件：

# wget http://www.baidu.com/index.html -O index.html

从文件中读取URL下载：

# wget -i url.list

下载ftp文件：

# wget --ftp-user=admin --ftp-password=admin ftp://192.168.1.10/ISO/CentOS-6.5-i386-minimal.iso

伪装客户端，指定user-agent和referer下载：

# wget -U "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.157 Safari/537.36" --referer "http://nginx.org/en/download.html" http://nginx.org/download/nginx-1.11.7.tar.gz

查看HTTP头信息：

# wget -S http://nginx.org/download/nginx-1.11.7.tar.gz

# wget --debug http://nginx.org/download/nginx-1.11.7.tar.gz

## 11.32 curl

功能：发送数据到URL，类似于HTTP客户端

常用选项：

-k, --insecure 允许HTTPS连接网站

-C, --continue-at  断点续传

-b, --cookie STRING/FILE  从文件中读取cookie

-c, --cookie-jar   把cookie保存到文件

-d, --data   使用POST方式发送数据

--data-urlencode   POST的数据URL编码

-F, --form   指定POST数据的表单

-D, --dump-header  保存头信息到文件

--ftp-pasv     指定FTP连接模式PASV/EPSV

-P, --ftp-port  指定FTP端口

-L, --location  遵循URL重定向，默认不处理

-l, --list-only 指列出FTP目录名

-H, --header   自定义头信息发送给服务器

-I, --head   查看HTTP头信息

-o, --output FILE   输出到文件

-#, --progress-bar  显示bar进度条

-x, --proxy [PROTOCOL://]HOST[:PORT]  使用代理

-U, --proxy-user USER[:PASSWORD] 代理用户名和密码

-e, --referer   指定引用地址referer

-O, --remote-name  使用远程服务器上名字写到本地

--connect-timeout  连接超时时间，单位秒

--retry NUM   连接重试次数

--retry-delay   两次重试间隔等待时间

-s, --silent   静默模式，不输出任何内容

-Y, --speed-limit  限制下载速率

-u, --user USER[:PASSWORD] 指定http和ftp用户名和密码

-T, --upload-file 上传文件

-A, --user-agent   指定客户端信息

示例：

下载页面：

# curl -o badu.html http://www.baidu.com

不输出下载信息：

# curl -s -o baidu.html http://www.baidu.com

伪装客户端，指定user-agent和referer下载：

# curl -A "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.157 Safari/537.36" -e "baike.baidu.com" http://127.0.0.1

模拟用户登录，并保存cookies到文件：

# curl -c ./cookies.txt -F NAME=user -F PWD=123 http://www.example.com/login.html

使用cookie访问：

# curl -b cookies.txt http://www.baidu.com

访问HTTP认证页面：

# curl -u user:pass http://www.example.com

FTP上传文件：

# curl -T filename ftp://user:pass@ip/a.txt

# curl ftp://ip -u user:pass-T filename

FTP下载文件：

# curl -O ftp://user:pass@ip/a.txt

# curl ftp://ip/filename -u user:pass -o filename

FTP下载多个文件：

# curl ftp://ip/img/[1,3,5].jpg

查看HTTP头信息：

# curl -I http://www.baidu.com

## 11.33 scp

功能：基于SSH的安全远程服务器文件拷贝

常用选项：

-i  指定私钥文件

-l  限制速率，单位Kb/s，1024Kb=1Mb

-P  指定远程主机SSH端口

-p  保存修改时间、访问时间和权限

-r  递归拷贝目录

-o  SSH选项，有以下几个比较常用的：

ConnectionAttempts=NUM   连接失败后重试次数

ConnectTimeout=SEC   连接超时时间

StrictHostKeyChecking=no 自动拉去主机key文件

PasswordAuthentication=no 禁止密码认证

示例：

本地目录推送到远程主机：

# scp -P 22 -r src\_dir root@192.168.1.10:/dst\_dir

远程主机目录拉取到本地：

# scp -P 22 -r root@192.168.1.10:dst\_dir src\_dir

同步文件方式一样，不用加-r参数

## 11.34 rsync

功能：远程或本地文件同步工具

常用选项：

-v  显示复制信息

-q  不输出错误信息

-c  跳过基础效验，不判断修改时间和大小

-a  归档模式，等效-rlptgoD，保留权限、属组等

-r  递归目录

-l  拷贝软连接

-z  压缩传输数据

-e  指定远程shell，比如ssh、rsh

--progress   进度条，等同-P

--bwlimit=KB/s   限制速率，0为没有限制

--delete   删除那些DST中SRC没有的文件

--exclude=PATTERN   排除匹配的文件或目录

--exclude-from=FILE 从文件中读取要排除的文件或目录

--password-file=FILE 从文件读取远程主机密码

--port=PORT 监听端口

示例：

本地复制目录：

# rsync -avz abc /opt

本地目录推送到远程主机：

# rsync -avz SRC root@192.168.1.120:DST

远程主机目录拉取到本地：

# rsync -avz root@192.168.1.10:SRC DST

保持远程主机目录与本地一样：

# rsync -avz --delete SRC root@192.168.1.120:DST

排除某个目录：

# rsync -avz --exclude=no\_dir SRC root@192.168.1.120:DST

指定SSH端口：

# rsync -avz /etc/hosts -e "ssh -p22" root@192.168.1.120:/opt

## 11.35 nohup

功能：运行程序，忽略挂起信号

示例：

后台运行程序，终端关闭不影响：

# nohup bash test.sh &>test.log &

## 11.36 iconv

功能：将文件内容字符集转成其他字符集

常用选项：

-l  列出所有已知的字符集

-f  原始文本编码

-t  输出编码

-o  输出到文件

-s  关闭警告

示例：

将文件内容转换UTF8：

# iconv -f gbk -t utf8 old.txt -o new.txt

将csv文件转换GBK：

# iconv -f utf8 -t gbk old.txt -o new.txt

解决邮件乱码：

# echo $(echo "content" | iconv -f utf8 -t gbk) | mail -s "$(echo "title" | iconv -f utf8 -t gbk)" example@mail.com

## 11.37 uname

功能：输出系统信息

常用选项：

-a  输出以下所有信息

-s  输出内核名称

-n  输出主机名

-r  输出内核发行版

-v  输出内核版本

-m  输出主机的硬件架构名称

-p  输出处理器类型或"unknown"

-i  输出硬件平台或"unknown

-o  输出操作系统名称

示例：

输出所有系统信息：

# uname -a

输出主机名：

# uname -a

输出内核版本：

# uname -r

输出操作系统：

# uname -o

## 11.38 sshpass

功能：非交互SSH登录（需要安装）

常用选项：

-f  从文件中获取密码

-d  用文件描述符数字获取密码

-p  指定SSH密码

-e  密码作为环境变量传递，变量名是SSHPASS

示例：

免交互SSH登录：

# sshpass -p 123456 ssh root@192.168.1.10

免交互传输文件：

# sshpass -p 123456 scp a.txt 192.168.1.10:/root

密码传入系统变量：

# SSHPASS=123456 rsync -avz /etc/hosts -e "sshpass -e ssh" root@192.168.1.221:/opt

## 11.39 tar

功能：归档目录或文件

常用选项：

-c 创建新归档

-d  比较归档和文件系统的差异

-r  追加文件到归档

-t  存档的内容列表

-x 提取归档所有文件

-C  改变解压目录

-f  使用归档文件或设备归档

-j  bzip2压缩

-z gzip压缩

-v  输出处理过程

示例：

创建归档文件来自foo和bar：

# tar -cf archive.tar foo bar

提取归档的所有文件：

# tar -xf archive.tar

列出所有归档文件内容：

# tar -tvf archive.tar

创建归档并gzip压缩：

# tar -zcvf archive.tar.gz log

提取归档文件并gzip解压：

# tar -zxvf log.tar.gz

创建归档并bzip2压缩：

# tar -jcvf log.tar.bz log

提取归档并解压到指定目录：

# tar -zxvf log.tar.gz -C /opt

## 11.40 logger

功能：系统日志的shell命令行接口

常用选项：

-i 每行记录进程ID

-f 指定输出日志到文件

-p 设置记录的优先级

-t 添加标签

示例：

# logger -i -t "my\_test" -p local3.notice "test\_info"

## 11.41 netstat

功能：打印网络连接、路由表、接口统计信息、伪装连接和多播成员

常用选项：

-r  显示路由表

-i  显示接口表

-n  不解析名字

-p  显示程序名 PID/Program

-l  显示监听的socket

-a  显示所有socket

-o  显示计时器

-Z  显示上下文

-t  只显示tcp连接

-u  只显示udp连接

-s  显示每个协议统计信息

示例：

显示所有监听：

# netstat -anltu

显示所有TCP连接：

# netstat -antp

显示所有UDP连接：

# netstat -anup

显示路由表：

# netstat -r

## 11.42 ss

功能：比netstat更强大的socket查看工具

格式：ss [options] [ FILTER ]

常用选项：

-n  不解析名字

-a  显示所有socket

-l  显示所有监听的socket

-o  显示计时器

-e  显示socket详细信息

-m  显示socket内存使用

-p  显示进程使用的socket

-i  显示内部TCP信息

-s  显示socket使用汇总

-4  只显示IPV4的socket

-0  显示包socket

-t  只显示TCP socket

-u  只显示UDP socket

-d  只显示DCCP socket

-w  只显示RAW socket

-x  只显示Unix域socket

-f  FAMILY  只显示socket族类型（ unix, inet, inet6, link, netlink）

-A  查询socket {all|inet|tcp|udp|raw|unix|packet|netlink}[,QUERY]

-D 将原始的TCP socket转储到文件

-F 从文件中读取过滤信息

过滤：

-o  state  显示TCP连接状态信息

示例：

显示所有TCP连接：

# ss -t -a

显示所有UDP连接：

# ss -u –a

显示socket使用汇总：

# ss -s

显示所有建立的连接：

# ss -o state established

显示所有的TIME-WAIT状态：

# ss -o state TIME-WAIT

搜索所有本地进程连接到X Server：

# ss -x src /tmp/.X11-unix/\*

## 11.43 lsof

功能：列出打开的文件

常用选项：

-i [i]  监听的网络地址，如果没有指定，默认列出所有。

[i]来自[46][protocol][@hostname|hostaddr][:service|port]

-U  列出Unix域socket文件

-p  指定PID

-u  指定用户名或UID所有打开的文件

+D 递归搜索

示例：

列出所有打开的文件：

# lsof

查看哪个进程占用文件：

# lsof /etc/passwd

列出所有打开的监听地址和unix域socket文件：

# lsof -i -U

列出80端口监听的进程：

# lsof -i:80

列出端口1-1024之间的所有进程：

# lsof -i:1-1024

列出所有TCP网络连接：

# lsof -i tcp

列出所有UDP网络连接：

# lsof -i udp

根据文件描述符列出打开的文件：

# lsof -d 1

列出某个目录被打开的文件：

# lsof +D /var/log

列出进程ID打开的文件：

# lsof -p 5373

打开所有登录用户名abc或user id 1234，或PID 123或PID 456：

# lsof -p 123,456 -u 123,abc

列出COMMAND列中包含字符串sshd：

# lsof -c sshd

## 11.44 ps

功能：报告当前进程的快照

常用选项：

-a  显示所有进程

-u  选择有效的用户ID或名称

-x  显示无控制终端的进程

-e  显示所有进程

-f  全格式

-r  只显示运行的进程

-T  这个终端的所有进程

-p  指定进程ID

--sort  对某列排序

-m  线程

-L  格式化代码列表

-o  用户自定义格式

CODE   NORMAL   HEADER

%C     pcpu     %CPU

%G     group    GROUP

%P     ppid     PPID

%U     user     USER

%a     args     COMMAND

%c     comm     COMMAND

%g     rgroup   RGROUP

%n     nice     NI

%p     pid      PID

%r     pgid     PGID

%t     etime    ELAPSED

%u     ruser    RUSER

%x     time     TIME

%y     tty      TTY

%z     vsz      VSZ

示例：

打印系统上所有进程标准语法：

# ps -ef

打印系统上所有进程BSD语法：

# ps aux

打印进程树：

# ps axjf 或 ps -ejH

查看进程启动的线程：

# ps -Lfp PID

查看当前用户的进程数：

# ps uxm 或 ps -U root -u root u

自定义格式显示并对CPU排序：

# ps -eo user,pid,pcpu,pmem,nice,lstart,time,args --sort=-pcpu

或 ps -eo "%U %p %C %n %x %a"

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

USER 进程所有者

PID 进程ID

%CPU 占用CPU时间

%MEM 物理内存

VSZ 虚拟内存大小（kb）

RSS 驻留集内存页数量（kb）

TTY 终端

STAT 进程状态；R运行，S休眠，D不可中断，T停止，Z僵尸，N表示普通优先级更低的优先级

START 进程启动时间

TIME使用CPU总时间

COMMAND 命令名称和参数

## 11.45 top

功能：动态显示活动的进程和系统资源利用率

常用选项：

-d  信息刷新时间间隔

-p  只监控指定的进程PID

-i  只显示正在使用CPU的进程

-H  显示线程

-u  只查看指定用户名的进程

-b  将输出编排成易处理格式，适合输出到文件处理

-n  指定最大循环刷新数

交互命令：

f  添加或删除显示的指标

c  显示完整命令

P  按CPU使用百分比排序

M  按驻留内存大小排序

T  按进程使用CPU时间排序

1  显示每个CPU核心使用率

k  终止一个进程

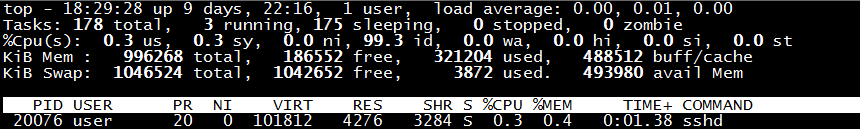
示例：

刷新一次并输出到文件：

# top -b -n 1 > top.log

只显示指定进程的线程：

# top -Hp 123



第一行：当前系统时间，系统运行了多长时间（9天22小时16分钟），CPU负载：1分钟、5分钟、15分钟

第二行：系统总共178个进程，3个CPU正在处理，175在休眠等待处理，0个停止，0个僵尸进程

第三行：us用户空间使用CPU时间0.3%，sy内核空间使用CPU时间0.3%，ni系统调整进程优先级使用CPU时间0.0%，id空闲CPU时间99.3%，wa等待I/O响应CPU时间0.0%，hi硬中断，si软中断。

第四行和第五行：物理内存与交换分区使用率

第六行：

PID 进程ID

USER 进程所有者

PR 进程优先级

NI 负值表示高优先级，正值表示低优先级

VIRT 进程启动后使用虚拟内存总量（KB），VIRT=SWAP+RES

RES 实际物理内存使用大小（KB），RES=CODE+DATA

SHR 共享内存大小（KB）,可能与其他进程共享的内存；计算进程使用物理内存大小：RES-SHR

S 进程状态；R运行，S休眠，D不可中断，T停止，Z僵尸

%CPU 上次更新到现在的CPU时间占用百分比

%MEM 使用物理内存百分比

TIME+ 使用CPU总时间

COMMAND命令名称和参数

## 11.46 free

功能：查看内存使用率

常用选项：

-b  bytes显示

-k  KB显示

-m  M显示

-g  G显示

-h  易读单位显示

-s  每几秒重复打印

-c 重复打印几次退出

示例：

查看物理内存：

# free -m

易读单位显示：

# free -h

## 11.47 df

功能：查看文件系统的磁盘空间使用情况

常用选项：

-a  包含虚拟文件系统

-h  可易读单位显示

-i  显示inode 信息而非块使用量

-k  1K块的数量

-t  只显示指定文件系统为指定类型的信息

-T  显示文件系统类型

示例：

查看所有文件系统：

# df -ah

输出指定文件系统：

# df -t xfs

## 11.48 vmstat

功能：报告虚拟内存、swap、io、上下文和CPU统计信息。

分析了这些文件：

/proc/meminfo

/proc/stat

/proc/\*/stat

常用选项：

-a  打印活跃和不活跃的内存页

-d  打印硬盘统计信息

-D  打印硬盘表

-p  打印硬盘分区统计信息

-s  打印虚拟内存表

-m  打印内存分配（slab）信息

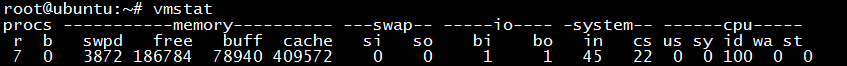
-t  添加时间戳到输出

-S  显示单位，默认k、KB、m、M，大写是\*1024

示例：

每秒刷新一次，统计五次：

# vmstat -t 1 5

r：CPU正在运行的进程数

b：在等待I/O的进程数

swpd：已经使用的交换内存（kb）

free：空闲的物理内存（kb）

buff：已经使用的缓冲区内存（kb）；一般对设备数据缓存，写入到磁盘的数据。

cache：已经使用的缓冲区内存（kb）；一般对文件数据缓存，从磁盘读取的数据。

si：从磁盘交换到内存的交换页数量（kb/s）

so：从内存交换到磁盘的交换页数据（kb/s）

bi：块设备接收的块数量（kb/s）

bo：块设备发送的块数量（kb/s）

in：每秒CPU中断次数

cs：每秒CPU上下文切换次数

us：用户进程使用CPU时间（%）

sy：系统进程使用CPU时间（%）

id：CPU空闲时间（%）

wa：等待I/O响应所消耗的CPU时间（%）

st：从虚拟设备中获得的时间（%）

## 11.49 iostat

功能：报告CPU利用率和磁盘I/O

用法: iostat [ 选项 ] [ <时间间隔> [ <次数> ] ]

常用选项：

-c  显示CPU使用率

-d  只显示磁盘使用率

-k  单位KB/s代替Block/s

-m  单位MB/s代替Block/s

-N 显示所有映射设备名字

-t  打印报告时间

-x  显示扩展统计信息

示例：

显示CPU使用率：

# iostat -c 1 3

显示I/O磁盘统计信息：

# iostat -d -x -k 1 3 # 间隔1秒，输出3次

## 11.50 sar

功能：查看系统资源综合方面利用率

常用选项：

-u, CPU

-r, memory

-b, disk

-n DEV, NIC traffic

-q, systemload

-b, TPS（Transaction Per Second，每秒事务处理量）

-o, output to file

示例：

# sar -u 2 3 #每两秒执行一次，采集三次

# sar -u 2 3 -o cpu.out

# sar -f cpu.out #读取文件

## 11.51 dstat

功能：查看系统资源综合方面利用率（需要安装）

常用选项：

-c cpu统计

-d 磁盘统计

-m 内存统计

-n 网络统计

-s swap统计

-l 负载统计

--tcp tcp状态统计

--udp udp状态统计

--socket socket数量统计

-t 输出时间

--output 写入csv文件

插件：

--list 支持的插件

--top-bio-adv 详细显示I/O进程写入block量，包括pid、r、w和cpu

--top-io-adv 进程写入磁盘总量

--top-cpu 占用CPU进程

--top-cpu-adv 查看最高CPU进程

--top-mem 内存进程

示例：

查看CPU利用率：

# dstat -c

查看TCP连接状态：

# dstat --tcp

## 11.52 ip

功能：查看/操作路由表、设备、路由策略和隧道

格式：ip [ OPTIONS ] OBJECT { COMMAND | help }

常用选项：

-b, -batch <FILENAME>  从文件或标准输入读取命令并调用他们，第一次失败将终止

-force 批量模式有错误不终止，如果有错误则状态返回非0

-s, -statistics 输出更多的统计信息

-l, -loops <COUNT>  指定最大的循环数

操作对象（OBEJECT）：

address  网络设备地址

12tp   以太网IP隧道

link   配置网络设备

maddress 多播地址

monitor 动态监控网络连接

mroute   多播路由缓存条目

mrule   角色在多播路由策略数据库

neighbour 管理ARP或NDISC缓存条目

netns   管理网络命名空间

ntable   管理neighbour缓存操作

route   路由表

rule   角色在路由策略数据库

tpc\_metrics/tcpmetrics 管理TCP指标

tunnel   IP隧道

tuntap   管理TUN/TAP设备

xfrm   管理IPSec策略

可通过ip OBEJECT help再查看对象的操作方法。

示例：

查看网络设备地址：

# ip addr

查看网卡统计信息：

# ip -s link

查看单个网卡统计信息：

# ip -s link ls eth0

查看ARP缓存表：

# ip neighbour

查看路由表：

# ip route

查看路由策略：

# ip rule

网卡设置/删除IP：

# ip addr add/del 192.168.1.201/24 dev eth0

添加/删除默认路由：

# ip route add default via 192.168.1.1 dev eth0

# ip route del 192.168.1.0/24 via 192.168.1.1

添加静态路由：

# ip route add 172.17.2.0/24 via 192.168.2.1 dev eth0

开启/关闭网卡：

# ip link set dev eth0 up/down

设置最大传输单元：

# ip link set dev eth0 mtu 1500

设置MAC地址：

# ip link set dev eth0 address 00:0c:29:52:73:8e

## 11.53 nc

功能：TCP和UDP连接和监听

常用选项：

-i interval 指定间隔时间发送和接受行文本

-l  监听模式，管理传入的连接

-n 不解析域名

-p  指定本地源端口

-s  指定本地源IP地址

-u  使用udp协议，默认是tcp

-v  执行过程输出

-w  timeout 连接超时时间

-x  proxy\_address[:port]  请求连接主机使用代理地址和端口

-z  指定扫描监听端口，不发送任何数据

示例：

端口扫描：

# nc -z 192.168.1.10 1-65535

TCP协议连接到目标端口：

# nc -p 31337 -w 5 192.168.1.10 22

UDP协议连接到目的端口：

# nc -u 192.168.1.10 53

指定本地IP连接：

# nc -s 192.168.1.9 192.168.1.10 22

探测端口是否开启：

# nc -z -w 2 192.168.1.10 22

创建监听Unix域Socket：

# nc -lU /var/tmp/ncsocket

通过HTTP代理连接主机：

# nc -x10.2.3.4:8080 -Xconnect 10.0.0.10 22

监听端口捕获输出到文件：

# nc -l 1234 > filename.out

从文件读入到指定端口：

# nc host.example.com 1234 < filename.in

收发信息：

# nc -l 1234

# nc 127.0.0.1 1234

执行memcahced命令：printf "stats\n" |nc 127.0.0.1 11211

发送邮件：

# nc [-C] localhost 25 << EOF

           HELO host.example.com

           MAIL FROM: <user@host.example.com>

           RCPT TO: <user2@host.example.com>

           DATA

           Body of email.

           .

           QUIT

           EOF

# echo -n "GET / HTTP/1.0\r\n\r\n" | nc host.example.com 80

## 11.54 time

功能：执行脚本时间

常用选项：

示例：

查看执行ls所需的时间：

# time ls

## 11.55 eval

功能：执行参数作为shell命令

示例：

for i in $@; do

eval $i

done

echo ---

echo $a

echo $b

# bash test.sh a=1 b=2

---

1

2

## 11.56 ssh

功能：SSH客户端

常用选项：

-p 指定远程主机端口

-i 指定认证文件

-L [bind\_address:]port:host:hostport

-R [bind\_address:]port:host:hostport]

-D [bind\_address:]port

-o SSH选项，有以下几个比较常用的：

ConnectionAttempts=NUM   连接失败后重试次数

ConnectTimeout=SEC   连接超时时间

StrictHostKeyChecking=no 自动拉去主机key文件

PasswordAuthentication=no 禁止密码认证

示例：

登录到远程主机：

# ssh user@192.168.1.120

远程主机执行命令：

# ssh user@192.168.1.120 'ifconfig'

本地文件内容写到远程主机文件：

# ssh user@192.168.1.120 'cat >> file' < /etc/passwd

SSH还提供了一个非常有用的功能，就是端口转发，能帮你解决一些无法建立的连接。

1. 本地端口转发

应用场景1：A不能访问C，B能访问A和C，实现通过B能让A访问C

在主机A执行：

# ssh -L 2222:主机C:22 主机B # ssh -L [绑定地址:]本地端口:主机C:C端口 主机B

将SSH绑定本地端口2222，本地2222端口数据转发主机B，主机B的所有数据转发到主机C的22端口；这样一来，只要在主机A ssh -p 2222 localhost，就等于连上了主机C的22端口。

应用场景2：一台Squid代理服务器，限制了本机可以清理缓存，但是我想从远程服务器清理

在远程服务器执行：

# ssh -L 31280:localhost:3128 SquidHost

在远程服务器上执行清理命令到本机31280端口，31280收到的数据加密转发到SquidHost的SSH Server上，SSH Client解密收到的数据并转发到监听的3128端口上，最后将Squid返回的数据原路返回。

1. 远程端口转发

应用场景1：A不能访问C，B能访问A和C，但A不能访问B，比如A在外网，B在内网

在主机B执行：

# ssh -R 2222:主机C:22 主机A

将SSH绑定本机2222端口，与主机A建立SSH通道，当主机A访问本地2222端口，就等于访问主机B的2222端口，主机B的2222端口把数据转发到主机C的22端口。

应用常见2：公司有一台内网服务器，还有一台云主机不能SSH直接连接这台公司内网服务器，但内网服务器可以SSH连接云主机

在公司内网服务器执行：

# ssh -R 2222:localhost:22 云主机

将云主机上的2222端口数据转发到内网服务器SSH Client上，SSH Client解密收到的数据并转发到监听的22端口上，最后再将返回的数据原路返回。

3）动态端口转发（不限定端口，全权代理）

应用场景：翻墙访问国外网站

如果是MAC系统直接在终端执行：ssh -D 2222 国外云主机

如果是Windows系统可借助putty工具实现，在putty里面端口转发->本地端口转发属性里面添加一个本地端口，并勾选SOCKS4/5动态转发，连接即可。

将SSH绑定本机8080端口，SSH就会创建一个SOCKS代理服务，直接在浏览器上设置代理本机127.0.0.1的8080端口即可，当浏览器访问国外网站时，本地代理把请求转发到国外云主机的SSH Server，SSH解密并转发给指定的网站。

注意：再Linux终端执行ssh绑定命令后，默认会进入一个新的shell，只要这个shell不退出，此端口转发就一直有效。如果要想放到后台执行就加-Nf两个选项，-N是不执行命令，-f后台执行，这样就转入后台运行，就可以在本地shell执行操作了，如果想关闭后台就kill这个进程。

参考资料：https://www.ibm.com/developerworks/cn/linux/l-cn-sshforward/

## 11.57 iptables

常见几种类型防火墙？

包过滤防火墙：包过滤是IP层实现，包过滤根据数据包的源IP、目的IP、协议类型（TCP/UDP/ICMP）、源端口、目的端口等包头信息及数据包传输方向灯信息来判断是否允许数据包通过。

应用层防火墙：也称为应用层代理防火墙，基于应用层协议的信息流检测，可以拦截某应用程序的所有封包，提取包内容进行分析。有效防止SQL注入或者XSS（跨站脚本攻击）之类的恶意代码。

状态检测防火墙：结合包过滤和应用层防火墙优点，基于连接状态检测机制，将属于同一连接的所有包作为一个整体的数据流看待，构成连接状态表（通信信息，应用程序信息等），通过规则表与状态表共同配合，对表中的各个连接状态判断。

iptables是Linux下的配置防火墙的工具，用于配置Linux内核集成的IP信息包过滤系统，使增删改查信息包过滤表中的规则更加简单。

iptables分为四表五链，表是链的容器，链是规则的容器，规则指定动作。

四表：

|  |  |
| --- | --- |
| filter | 用于包过滤 |
| nat | 网络地址转发 |
| mangle | 对特定数据包修改 |
| raw | 不做数据包链接跟踪 |

五链：

|  |  |
| --- | --- |
| INPUT | 本机数据包入口 |
| OUTPUT | 本机数据包出口 |
| FORWARD | 经过本机转发的数据包 |
| PREROUTING | 防火墙之前，修改目的地址（DNAT） |
| POSTROUTING | 防火墙之后，修改源地址（SNAT） |

表中的链：

|  |  |
| --- | --- |
| **表** | **链** |
| filter | INPUT、OUTPUT和FORWARD |
| nat | PREROUTING、POSTROUTING和OUTPUT |
| mangle | PREROUTING、POSTROUTING、INPUT、OUTPUT和FORWARD |
| raw | PREROUTING和OUTPUT |

命令格式：iptables [-t table] 命令 [chain] 匹配条件 动作

|  |  |
| --- | --- |
| **命令** | **描述** |
| -A，append | 追加一条规则 |
| -I，insert | 插入一条规则，默认链头，后跟编号，指定第几条 |
| -D，delete | 删除一条规则 |
| -F，flush | 清空规则 |
| -L，list | 列出规则 |
| -P，policy | 设置链缺省规则 |
| -m，module | 模块，比如state、multiport |

|  |  |
| --- | --- |
| **匹配条件** | **描述** |
| -i | 入口网卡 |
| -o | 出口网卡 |
| -s | 源地址 |
| -d | 目的地址 |
| -p | 协议类型 |
| --sport | 源端口 |
| --dport | 目的端口 |

|  |  |
| --- | --- |
| **动作** | **描述** |
| ACCEPT | 允许数据包通过 |
| DROP | 丢弃数据包不做处理 |
| REJECT | 拒绝数据包，并返回报错信息 |
| SNAT | 一般用于nat表的POSTROUTING链，进行源地址转换 |
| DNAT | 一般用于nat表的PREROUTING链，进行目的地址转换 |
| MASQUERADE | 动态源地址转换，动态IP时使用 |

|  |  |
| --- | --- |
| **模块** | **描述** |
| state | 包状态，有四个：NEW、RELATED、ESTABLISHED和INVALID |
| mac | 源MAC地址 |
| limit | 包速率限制 |
| multiport | 多端口，以逗号分隔 |
| iprange | 端口范围，以逗号分隔 |

示例：常用的规则配置方法

iptables -F         # 清空表规则，默认filter表

iptables -t nat -F   # 清空nat表

iptables -A INPUT -p tcp --dport 22 -j ACCEPT   # 允许TCP的22端口访问

iptables -I INPUT -p udp --dport 53 -j ACCEPT   # 允许UDP的53端口访问，插入在第一条

iptables -A INPUT -p tcp --dport 22:25 -j ACCEPT # 允许端口范围访问

iptables -D INPUT -p tcp --dport 22:25 -j ACCEPT # 删除这条规则

# 允许多个TCP端口访问

iptables -A INPUT -p tcp -m multiport --dports 22,80,8080 -j ACCEPT

iptables -A INPUT -s 192.168.1.0/24 -j ACCEPT # 允许192.168.1.0段IP访问

iptables -A INPUT -s 192.168.1.10 -j DROP    # 对1.10数据包丢弃

iptables -A INPUT -i eth0 -p icmp -j DROP    # eth0网卡ICMP数据包丢弃，也就是禁ping

# 允许来自lo接口，如果没有这条规则，将不能通过127.0.0.1访问本地服务

iptables -A INPUT -i lo -j ACCEPT

# 限制并发连接数，超过30个拒绝

iptables -I INPUT -p tcp --syn --dport 80 -m connlimit --connlimit-above 30 -j REJECT

# 限制每个IP每秒并发连接数最大3个

iptables -I INPUT -p tcp --syn -m limit --limit 1/s --limit-burst 3 -j ACCEPT

iptables -A FORWARD -p tcp --syn -m limit --limit 1/s -j ACCEPT

# iptables服务器作为网关时，内网访问公网

iptables –t nat -A POSTROUTING -s [内网IP或网段] -j SNAT --to [公网IP]

# 访问iptables公网IP端口，转发到内网服务器端口

iptables –t nat -A PREROUTING -d [对外IP] -p tcp --dport [对外端口] -j DNAT --to [内网IP:内网端口]

# 本地80端口转发到本地8080端口

iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 8080

# 允许已建立及该链接相关联的数据包通过

iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# ASDL拨号上网

iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o pppo -j MASQUERADE

# 设置INPUT链缺省操作丢弃所有数据包，只要不符合规则的数据包都丢弃。注意要在最后设置，以免把自己关在外面！

iptables -P INPUT DROP

# 第十二章 Shell脚本编写实战

## 注意事项

1. 开头加解释器：#!/bin/bash
2. 语法缩进，使用四个空格；多加注释说明。

3）命名建议规则：变量名大写、局部变量小写，函数名小写，名字体现出实际作用。

4）默认变量是全局的，在函数中变量local指定为局部变量，避免污染其他作用域。

5）有两个命令能帮助我调试脚本：set -e 遇到执行非0时退出脚本，set -x 打印执行过程。

6）写脚本一定先测试再到生产上。

## 12.1 获取随机字符串或数字

获取随机8位字符串：

方法1：

# echo $RANDOM |md5sum |cut -c 1-8

471b94f2

方法2：

# openssl rand -base64 4

vg3BEg==

方法3：

# cat /proc/sys/kernel/random/uuid |cut -c 1-8

ed9e032c

获取随机8位数字：

方法1：

# echo $RANDOM |cksum |cut -c 1-8

23648321

方法2：

# openssl rand -base64 4 |cksum |cut -c 1-8

38571131

方法3：

# date +%N |cut -c 1-8

69024815

cksum：打印CRC效验和统计字节

## 12.2 定义一个颜色输出字符串函数

方法1：

function echo\_color() {

if [ $1 == "green" ]; then

echo -e "\033[32;40m$2\033[0m"

elif [ $1 == "red" ]; then

echo -e "\033[31;40m$2\033[0m"

fi

}

方法2：

function echo\_color() {

case $1 in

green)

echo -e "\033[32;40m$2\033[0m"

;;

red)

echo -e "\033[31;40m$2\033[0m"

;;

\*)

echo "Example: echo\_color red string"

esac

}

使用方法：echo\_color green "test"

function关键字定义一个函数，可加或不加。

## 12.3 批量创建用户

#!/bin/bash

DATE=$(date +%F\_%T)

USER\_FILE=user.txt

echo\_color(){

    if [ $1 == "green" ]; then

        echo -e "\033[32;40m$2\033[0m"

    elif [ $1 == "red" ]; then

        echo -e "\033[31;40m$2\033[0m"

    fi

}

# 如果用户文件存在并且大小大于0就备份

if [ -s $USER\_FILE ]; then

    mv $USER\_FILE ${USER\_FILE}-${DATE}.bak

    echo\_color green "$USER\_FILE exist, rename ${USER\_FILE}-${DATE}.bak"

fi

echo -e "User\tPassword" >> $USER\_FILE

echo "----------------" >> $USER\_FILE

for USER in user{1..10}; do

    if ! id $USER &>/dev/null; then

        PASS=$(echo $RANDOM |md5sum |cut -c 1-8)

        useradd $USER

        echo $PASS |passwd --stdin $USER &>/dev/null

        echo -e "$USER\t$PASS" >> $USER\_FILE

        echo "$USER User create successful."

    else

        echo\_color red "$USER User already exists!"

    fi

done

## 12.4 检查软件包是否安装

#!/bin/bash

if rpm -q sysstat &>/dev/null; then

    echo "sysstat is already installed."

else

    echo "sysstat is not installed!"

fi

## 12.5 检查服务状态

#!/bin/bash

PORT\_C=$(ss -anu |grep -c 123)

PS\_C=$(ps -ef |grep ntpd |grep -vc grep)

if [ $PORT\_C -eq 0 -o $PS\_C -eq 0 ]; then

    echo "内容" | mail -s "主题" dst@example.com

fi

## 12.6 检查主机存活状态

方法1：将错误IP放到数组里面判断是否ping失败三次

#!/bin/bash

IP\_LIST="192.168.18.1 192.168.1.1 192.168.18.2"

for IP in $IP\_LIST; do

NUM=1

while [ $NUM -le 3 ]; do

if ping -c 1 $IP > /dev/null; then

echo "$IP Ping is successful."

break

else

# echo "$IP Ping is failure $NUM"

FAIL\_COUNT[$NUM]=$IP

let NUM++

fi

done

if [ ${#FAIL\_COUNT[\*]} -eq 3 ];then

echo "${FAIL\_COUNT[1]} Ping is failure!"

unset FAIL\_COUNT[\*]

fi

done

方法2：将错误次数放到FAIL\_COUNT变量里面判断是否ping失败三次

#!/bin/bash

IP\_LIST="192.168.18.1 192.168.1.1 192.168.18.2"

for IP in $IP\_LIST; do

    FAIL\_COUNT=0

    for ((i=1;i<=3;i++)); do

        if ping -c 1 $IP >/dev/null; then

            echo "$IP Ping is successful."

            break

        else

            # echo "$IP Ping is failure $i"

            let FAIL\_COUNT++

        fi

    done

    if [ $FAIL\_COUNT -eq 3 ]; then

        echo "$IP Ping is failure!"

    fi

done

方法3：利用for循环将ping通就跳出循环继续，如果不跳出就会走到打印ping失败

#!/bin/bash

ping\_success\_status() {

    if ping -c 1 $IP >/dev/null; then

        echo "$IP Ping is successful."

        continue

    fi

}

IP\_LIST="192.168.18.1 192.168.1.1 192.168.18.2"

for IP in $IP\_LIST; do

    ping\_success\_status

    ping\_success\_status

    ping\_success\_status

    echo "$IP Ping is failure!"

done

## 12.7 监控CPU、内存和硬盘利用率

1）CPU

借助vmstat工具来分析CPU统计信息。

#!/bin/bash

DATE=$(date +%F" "%H:%M)

IP=$(ifconfig eth0 |awk -F '[ :]+' '/inet addr/{print $4}') # 只支持CentOS6

MAIL="example@mail.com"

if ! which vmstat &>/dev/null; then

echo "vmstat command no found, Please install procps package."

exit 1

fi

US=$(vmstat |awk 'NR==3{print $13}')

SY=$(vmstat |awk 'NR==3{print $14}')

IDLE=$(vmstat |awk 'NR==3{print $15}')

WAIT=$(vmstat |awk 'NR==3{print $16}')

USE=$(($US+$SY))

if [ $USE -ge 50 ]; then

echo "

Date: $DATE

Host: $IP

Problem: CPU utilization $USE

" | mail -s "CPU Monitor" $MAIL

fi

2）内存

#!/bin/bash

DATE=$(date +%F" "%H:%M)

IP=$(ifconfig eth0 |awk -F '[ :]+' '/inet addr/{print $4}')

MAIL="example@mail.com"

TOTAL=$(free -m |awk '/Mem/{print $2}')

USE=$(free -m |awk '/Mem/{print $3-$6-$7}')

FREE=$(($TOTAL-$USE))

# 内存小于1G发送报警邮件

if [ $FREE -lt 1024 ]; then

echo "

Date: $DATE

Host: $IP

Problem: Total=$TOTAL,Use=$USE,Free=$FREE

" | mail -s "Memory Monitor" $MAIL

fi

1. 硬盘

#!/bin/bash

DATE=$(date +%F" "%H:%M)

IP=$(ifconfig eth0 |awk -F '[ :]+' '/inet addr/{print $4}')

MAIL="example@mail.com"

TOTAL=$(fdisk -l |awk -F'[: ]+' 'BEGIN{OFS="="}/^Disk \/dev/{printf "%s=%sG,",$2,$3}')

PART\_USE=$(df -h |awk 'BEGIN{OFS="="}/^\/dev/{print $1,int($5),$6}')

for i in $PART\_USE; do

PART=$(echo $i |cut -d"=" -f1)

USE=$(echo $i |cut -d"=" -f2)

MOUNT=$(echo $i |cut -d"=" -f3)

if [ $USE -gt 80 ]; then

echo "

Date: $DATE

Host: $IP

Total: $TOTAL

Problem: $PART=$USE($MOUNT)

" | mail -s "Disk Monitor" $MAIL

fi

done

## 12.8 批量主机磁盘利用率监控

前提监控端和被监控端SSH免交互登录或者密钥登录。

写一个配置文件保存被监控主机SSH连接信息，文件内容格式：IP User Port

#!/bin/bash

HOST\_INFO=host.info

for IP in $(awk '/^[^#]/{print $1}' $HOST\_INFO); do

USER=$(awk -v ip=$IP 'ip==$1{print $2}' $HOST\_INFO)

PORT=$(awk -v ip=$IP 'ip==$1{print $3}' $HOST\_INFO)

TMP\_FILE=/tmp/disk.tmp

ssh -p $PORT $USER@$IP 'df -h' > $TMP\_FILE

USE\_RATE\_LIST=$( awk 'BEGIN{OFS="="}/^\/dev/{print $1,int($5)}' $TMP\_FILE)

for USE\_RATE in $USE\_RATE\_LIST; do

PART\_NAME=${USE\_RATE%=\*}

USE\_RATE=${USE\_RATE#\*=}

if [ $USE\_RATE -ge 80 ]; then

echo "Warning: $PART\_NAME Partition usage $USE\_RATE%!"

fi

done

done

## 12.9 检查网站可用性

1）检查URL可用性

方法1：

check\_url() {

HTTP\_CODE=$(curl -o /dev/null --connect-timeout 3 -s -w "%{http\_code}" $1)

if [ $HTTP\_CODE -ne 200 ]; then

echo "Warning: $1 Access failure!"

fi

}

方法2：

check\_url() {

if ! wget -T 10 --tries=1 --spider $1 >/dev/null 2>&1; then

#-T超时时间，--tries尝试1次，--spider爬虫模式

echo "Warning: $1 Access failure!"

fi

}

使用方法：check\_url www.baidu.com

2）判断三次URL可用性

思路与上面检查主机存活状态一样。

方法1：利用循环技巧，如果成功就跳出当前循环，否则执行到最后一行

#!/bin/bash

check\_url() {

HTTP\_CODE=$(curl -o /dev/null --connect-timeout 3 -s -w "%{http\_code}" $1)

if [ $HTTP\_CODE -eq 200 ]; then

continue

fi

}

URL\_LIST="www.baidu.com www.agasgf.com"

for URL in $URL\_LIST; do

check\_url $URL

check\_url $URL

check\_url $URL

echo "Warning: $URL Access failure!"

done

方法2：错误次数保存到变量

#!/bin/bash

URL\_LIST="www.baidu.com www.agasgf.com"

for URL in $URL\_LIST; do

FAIL\_COUNT=0

for ((i=1;i<=3;i++)); do

HTTP\_CODE=$(curl -o /dev/null --connect-timeout 3 -s -w "%{http\_code}" $URL)

if [ $HTTP\_CODE -ne 200 ]; then

let FAIL\_COUNT++

else

break

fi

done

if [ $FAIL\_COUNT -eq 3 ]; then

echo "Warning: $URL Access failure!"

fi

done

方法3：错误次数保存到数组

#!/bin/bash

URL\_LIST="www.baidu.com www.agasgf.com"

for URL in $URL\_LIST; do

NUM=1

while [ $NUM -le 3 ]; do

HTTP\_CODE=$(curl -o /dev/null --connect-timeout 3 -s -w "%{http\_code}" $URL)

if [ $HTTP\_CODE -ne 200 ]; then

FAIL\_COUNT[$NUM]=$IP #创建数组，以$NUM下标，$IP元素

let NUM++

else

break

fi

done

if [ ${#FAIL\_COUNT[\*]} -eq 3 ]; then

echo "Warning: $URL Access failure!"

unset FAIL\_COUNT[\*] #清空数组

fi

done

## 12.10 检查MySQL主从同步状态

#!/bin/bash

USER=bak

PASSWD=123456

IO\_SQL\_STATUS=$(mysql -u$USER -p$PASSWD -e 'show slave status\G' |awk -F: '/Slave\_.\*\_Running/{gsub(": ",":");print $0}')  #gsub去除冒号后面的空格

for i in $IO\_SQL\_STATUS; do

    THREAD\_STATUS\_NAME=${i%:\*}

    THREAD\_STATUS=${i#\*:}

    if [ "$THREAD\_STATUS" != "Yes" ]; then

        echo "Error: MySQL Master-Slave $THREAD\_STATUS\_NAME status is $THREAD\_STATUS!"

    fi

done

## 12.11 屏蔽网站访问频繁的IP

1）屏蔽每分钟访问超过200的IP

方法1：以Nginx日志作为测试

DATE=$(date +%d/%b/%Y:%H:%M)

ABNORMAL\_IP=$(tail -n5000 access.log |grep $DATE |awk '{a[$1]++}END{for(i in a)if(a[i]>100)print i}')

#先tail防止文件过大，读取慢，数字可调整每分钟最大的访问量。awk不能直接过滤日志，因为包含特殊字符。

for IP in $ABNORMAL\_IP; do

if [ $(iptables -vnL |grep -c "$IP") -eq 0 ]; then

iptables -I INPUT -s $IP -j DROP

fi

done

方法2：通过建立连接数

ABNORMAL\_IP=$(netstat -an |awk '$4~/:80$/ && $6~/ESTABLISHED/{gsub(/:[0-9]+/,"",$5);{a[$5]++}}END{for(i in a)if(a[i]>100)print i}')

#gsub是将第五列（客户端IP）的冒号和端口去掉

for IP in $ABNORMAL\_IP; do

if [ $(iptables -vnL |grep -c "$IP") -eq 0 ]; then

iptables -I INPUT -s $IP -j DROP

fi

done

2）屏蔽每分钟SSH暴力破解超过10次的IP

方法1：通过lastb获取登录状态:

DATE=$(date +"%a %b %e %H:%M") #星期月天时分 %e单数字时显示7，而%d显示07

ABNORMAL\_IP=$(lastb |grep "$DATE" |awk '{a[$3]++}END{for(i in a)if(a[i]>10)print i}')

for IP in $ABNORMAL\_IP; do

if [ $(iptables -vnL |grep -c "$IP") -eq 0 ]; then

iptables -I INPUT -s $IP -j DROP

fi

done

方法2：通过日志获取登录状态

DATE=$(date +"%b %d %H")

ABNORMAL\_IP="$(tail -n10000 /var/log/auth.log |grep "$DATE" |awk '/Failed/{a[$(NF-3)]++}END{for(i in a)if(a[i]>5)print i}')"

for IP in $ABNORMAL\_IP; do

if [ $(iptables -vnL |grep -c "$IP") -eq 0 ]; then

iptables -A INPUT -s $IP -j DROP

echo "$(date +"%F %T") - iptables -A INPUT -s $IP -j DROP" >>~/ssh-login-limit.log

fi

done

## 12.12 判断输入是否为IP

方法1:

function check\_ip(){

IP=$1

VALID\_CHECK=$(echo $IP|awk -F. '$1<=255&&$2<=255&&$3<=255&&$4<=255{print "yes"}')

if echo $IP|grep -E "^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$">/dev/null; then

if [ $VALID\_CHECK == "yes" ]; then

echo "$IP available."

else

echo "$IP not available!"

fi

else

echo "Format error!"

fi

}

check\_ip 192.168.1.1

check\_ip 256.1.1.1

方法2：

function check\_ip(){

IP=$1

if [[ $IP =~ ^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$ ]]; then

FIELD1=$(echo $IP|cut -d. -f1)

FIELD2=$(echo $IP|cut -d. -f2)

FIELD3=$(echo $IP|cut -d. -f3)

FIELD4=$(echo $IP|cut -d. -f4)

if [ $FIELD1 -le 255 -a $FIELD2 -le 255 -a $FIELD3 -le 255 -a $FIELD4 -le 255 ]; then

echo "$IP available."

else

echo "$IP not available!"

fi

else

echo "Format error!"

fi

}

check\_ip 192.168.1.1

check\_ip 256.1.1.1

增加版：加个死循环，如果IP可用就退出，不可用提示继续输入，并使用awk判断

function check\_ip(){

local IP=$1

VALID\_CHECK=$(echo $IP|awk -F. '$1<=255&&$2<=255&&$3<=255&&$4<=255{print "yes"}')

if echo $IP|grep -E "^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$" >/dev/null; then

if [ $VALID\_CHECK == "yes" ]; then

return 0

else

echo "$IP not available!"

return 1

fi

else

echo "Format error! Please input again."

return 1

fi

}

while true; do

read -p "Please enter IP: " IP

check\_ip $IP

[ $? -eq 0 ] && break || continue

done

## 12.13 判断输入是否为数字

方法1：

if [[ $1 =~ ^[0-9]+$ ]]; then

echo "Is Number."

else

echo "No Number."

fi

方法2：

if [ $1 -gt 0 ] 2>/dev/null; then

echo "Is Number."

else

echo "No Number."

fi

方法3：

echo $1 |awk '{print $0~/^[0-9]+$/?"Is Number.":"No Number."}' #三目运算符

## 12.14 找出包含关键字的文件

DIR=$1

KEY=$2

for FILE in $(find $DIR -type f); do

if grep $KEY $FILE &>/dev/null; then

echo "--> $FILE"

fi

done

## 12.15 监控目录，将新创建的文件名追加到日志中

需安装inotify-tools软件包。

#!/bin/bash

MON\_DIR=/opt

inotifywait -mq --format %f -e create $MON\_DIR |\

while read files; do

echo $files >> test.log

done

## 12.16 多个网卡选择

function local\_nic() {

    local NUM ARRAY\_LENGTH

    NUM=0

    for NIC\_NAME in $(ls /sys/class/net|grep -vE "lo|docker0"); do

        NIC\_IP=$(ifconfig $NIC\_NAME |awk -F'[: ]+' '/inet addr/{print $4}')

        if [ -n "$NIC\_IP" ]; then

            NIC\_IP\_ARRAY[$NUM]="$NIC\_NAME:$NIC\_IP"    #将网卡名和对应IP放到数组

            let NUM++

        fi

    done

    ARRAY\_LENGTH=${#NIC\_IP\_ARRAY[\*]}

    if [ $ARRAY\_LENGTH -eq 1 ]; then     #如果数组里面只有一条记录说明就一个网卡

        NIC=${NIC\_IP\_ARRAY[0]%:\*}

        return 0

    elif [ $ARRAY\_LENGTH -eq 0 ]; then   #如果没有记录说明没有网卡

        echo "No available network card!"

        exit 1

    else

        #如果有多条记录则提醒输入选择

        for NIC in ${NIC\_IP\_ARRAY[\*]}; do

            echo $NIC

        done

        while true; do

            read -p "Please enter local use to network card name: " INPUT\_NIC\_NAME

            COUNT=0

            for NIC in ${NIC\_IP\_ARRAY[\*]}; do

                NIC\_NAME=${NIC%:\*}

                if [ $NIC\_NAME == "$INPUT\_NIC\_NAME" ]; then

                    NIC=${NIC\_IP\_ARRAY[$COUNT]%:\*}

                    return 0

                else

                   COUNT+=1

                fi

            done

            echo "Not match! Please input again."

        done

    fi

}

local\_nic

如果有只有一个网卡就不选择。

## 12.17 查看网卡实时流量

#!/bin/bash

# Description: Only CentOS6

traffic\_unit\_conv() {

local traffic=$1

if [ $traffic -gt 1024000 ]; then

printf "%.1f%s" "$(($traffic/1024/1024))" "MB/s"

elif [ $traffic -lt 1024000 ]; then

printf "%.1f%s" "$(($traffic/1024))" "KB/s"

fi

}

NIC=$1

echo -e " In ------ Out"

while true; do

OLD\_IN=$(awk -F'[: ]+' '$0~"'$NIC'"{print $3}' /proc/net/dev)

OLD\_OUT=$(awk -F'[: ]+' '$0~"'$NIC'"{print $11}' /proc/net/dev)

sleep 1

NEW\_IN=$(awk -F'[: ]+' '$0~"'$NIC'"{print $3}' /proc/net/dev)

NEW\_OUT=$(awk -F'[: ]+' '$0~"'$NIC'"{print $11}' /proc/net/dev)

IN=$(($NEW\_IN-$OLD\_IN))

OUT=$(($NEW\_OUT-$OLD\_OUT))

echo "$(traffic\_unit\_conv $IN) $(traffic\_unit\_conv $OUT)"

sleep 1

done

# 也可以通过ifconfig命令获取收发流量

while true; do

OLD\_IN=$(ifconfig $NIC |awk -F'[: ]+' '/bytes/{print $4}')

OLD\_OUT=$(ifconfig $NIC |awk -F'[: ]+' '/bytes/{print $9}')

sleep 1

NEW\_IN=$(ifconfig $NIC |awk -F'[: ]+' '/bytes/{print $4}')

NEW\_OUT=$(ifconfig $NIC |awk -F'[: ]+' '/bytes/{print $9}')

IN=$(($NEW\_IN-$OLD\_IN))

OUT=$(($NEW\_OUT-$OLD\_OUT))

echo "$(traffic\_unit\_conv $IN) $(traffic\_unit\_conv $OUT)"

sleep 1

done

## 12.18 MySQL数据库备份

#!/bin/bash

DATE=$(date +%F\_%H-%M-%S)

HOST=192.168.1.120

DB=test

USER=bak

PASS=123456

MAIL="zhangsan@example.com lisi@example.com"

BACKUP\_DIR=/data/db\_backup

SQL\_FILE=${DB}\_full\_$DATE.sql

BAK\_FILE=${DB}\_full\_$DATE.zip

cd $BACKUP\_DIR

if mysqldump -h$HOST -u$USER -p$PASS --single-transaction --routines --triggers -B $DB > $SQL\_FILE; then

zip $BAK\_FILE $SQL\_FILE && rm -f $SQL\_FILE

if [ ! -s $BAK\_FILE ]; then

echo "$DATE 内容" | mail -s "主题" $MAIL

fi

else

echo "$DATE 内容" | mail -s "主题" $MAIL

fi

find $BACKUP\_DIR -name '\*.zip' -ctime +14 -exec rm {} \;

## 12.19 Nginx启动脚本

#!/bin/bash

# Description: Only support RedHat system

. /etc/init.d/functions

WORD\_DIR=/data/project/nginx1.10

DAEMON=$WORD\_DIR/sbin/nginx

CONF=$WORD\_DIR/conf/nginx.conf

NAME=nginx

PID=$(awk -F'[; ]+' '/^[^#]/{if($0~/pid;/)print $2}' $CONF)

if [ -z "$PID" ]; then

    PID=$WORD\_DIR/logs/nginx.pid

else

    PID=$WORD\_DIR/$PID

fi

stop() {

    $DAEMON -s stop

    sleep 1

    [ ! -f $PID ] && action "\* Stopping $NAME"  /bin/true || action "\* Stopping $NAME" /bin/false

}

start() {

    $DAEMON

    sleep 1

    [ -f $PID ] && action "\* Starting $NAME"  /bin/true || action "\* Starting $NAME" /bin/false

}

reload() {

    $DAEMON -s reload

}

test\_config() {

    $DAEMON -t

}

case "$1" in

    start)

        if [ ! -f $PID ]; then

            start

        else

            echo "$NAME is running..."

            exit 0

        fi

        ;;

    stop)

        if [ -f $PID ]; then

            stop

        else

            echo "$NAME not running!"

            exit 0

        fi

        ;;

    restart)

        if [ ! -f $PID ]; then

            echo "$NAME not running!"

            start

        else

            stop

            start

        fi

        ;;

    reload)

        reload

        ;;

    testconfig)

        test\_config

        ;;

    status)

        [ -f $PID ] && echo "$NAME is running..." || echo "$NAME not running!"

        ;;

    \*)

        echo "Usage: $0 {start|stop|restart|reload|testconfig|status}"

        exit 3

        ;;

esac

## 12.20 选择SSH连接主机

写一个配置文件保存被监控主机SSH连接信息，文件内容格式：主机名 IP User Port

#!/bin/bash

PS3="Please input number: "

HOST\_FILE=host

while true; do

select NAME in $(awk '{print $1}' $HOST\_FILE) quit; do

[ ${NAME:=empty} == "quit" ] && exit 0

IP=$(awk -v NAME=${NAME} '$1==NAME{print $2}' $HOST\_FILE)

USER=$(awk -v NAME=${NAME} '$1==NAME{print $3}' $HOST\_FILE)

PORT=$(awk -v NAME=${NAME} '$1==NAME{print $4}' $HOST\_FILE)

if [ $IP ]; then

echo "Name: $NAME, IP: $IP"

ssh -o StrictHostKeyChecking=no -p $PORT -i id\_rsa $USER@$IP # 密钥登录

break

else

echo "Input error, Please enter again!"

break

fi

done

done

## 12.21 FTP下载文件

#!/bin/bash

if [ $# -ne 1 ]; then

    echo "Usage: $0 filename"

fi

dir=$(dirname $1)

file=$(basename $1)

ftp -n -v << EOF   # -n 自动登录

open 192.168.1.10

user admin adminpass

binary   # 设置ftp传输模式为二进制，避免MD5值不同或.tar.gz压缩包格式错误

cd $dir

get "$file"

EOF

## 12.22 输入五个100数之内的字符，统计和、最小和最大

COUNT=1

SUM=0

MIN=0

MAX=100

while [ $COUNT -le 5 ]; do

    read -p "请输入1-10个整数：" INT

    if [[ ! $INT =~ ^[0-9]+$ ]]; then

        echo "输入必须是整数！"

        exit 1

    elif [[ $INT -gt 100 ]]; then

        echo "输入必须是100以内！"

        exit 1

    fi

    SUM=$(($SUM+$INT))

    [ $MIN -lt $INT ] && MIN=$INT

    [ $MAX -gt $INT ] && MAX=$INT

    let COUNT++

done

echo "SUM: $SUM"

echo "MIN: $MIN"

echo "MAX: $MAX"

## 12.23 将结果分别赋值给变量

方法1：

for i in $(echo "4 5 6"); do

eval a$i=$i

done

echo $a4 $a5 $a6

方法2：将位置参数192.168.18.1{1,2}拆分为到每个变量

num=0

for i in $(eval echo $\*);do #eval将{1,2}分解为1 2

let num+=1

eval node${num}="$i"

done

echo $node1 $node2 $node3

# bash a.sh 192.168.18.1{1,2}

192.168.18.11 192.168.18.12

方法3：

arr=(4 5 6)

INDEX1=$(echo ${arr[0]})

INDEX2=$(echo ${arr[1]})

INDEX3=$(echo ${arr[2]})

## 12.24 批量修改文件名

# touch article\_{1..3}.html

# ls

article\_1.html  article\_2.html  article\_3.html

现在想把article改为bbs：

方法1：

for file in $(ls \*html); do

    mv $file bbs\_${file#\*\_}

    # mv $file $(echo $file |sed -r 's/.\*(\_.\*)/bbs\1/')

    # mv $file $(echo $file |echo bbs\_$(cut -d\_ -f2)

done

方法2：

for file in $(find . -maxdepth 1 -name "\*html"); do

     mv $file bbs\_${file#\*\_}

done

方法3：

# rename article bbs \*.html

## 12.25 统计当前目录中以.html结尾的文件总大小

方法1：

# find . -name "\*.html" -maxdepth 1 -exec du -b {} \; |awk '{sum+=$1}END{print sum}'

方法2：

for size in $(ls -l \*.html |awk '{print $5}'); do

    sum=$(($sum+$size))

done

echo $sum

递归统计：

# find . -name "\*.html" -exec du -k {} \; |awk '{sum+=$1}END{print sum}'

## 12.26 扫描主机端口状态

#!/bin/bash

HOST=$1

PORT="22 25 80 8080"

for PORT in $PORT; do

if echo &>/dev/null > /dev/tcp/$HOST/$PORT; then

echo "$PORT open"

else

echo "$PORT close"

fi

done

## 12.27 Expect实现SSH免交互执行命令

需要先安装expect工具。

expect涉及用法说明：

|  |  |
| --- | --- |
| 命令 | 描述 |
| set | 可以设置超时，也可以设置变量 |
| timeout | 超时等待时间，默认10s |
| spawn | 执行一个命令 |
| expect "" | 匹配输出的内容 |
| exp\_continue | 继续执行下面匹配 |
| \r | 回车 |
| $argc | 统计位置参数数量 |
| [lindex $argv 0] | 位置参数 |
| puts | 打印字符串，类似于echo |
| expect{...} | 输入多行记录 |

方法1：EOF标准输出作为expect标准输入

#!/bin/bash

USER=root

PASS=123.com

IP=192.168.1.120

expect << EOF

set timeout 30

spawn ssh $USER@$IP

expect {

"(yes/no)" {send "yes\r"; exp\_continue}

"password:" {send "$PASS\r"}

}

expect "$USER@\*" {send "$1\r"}

expect "$USER@\*" {send "exit\r"}

expect eof

EOF

方法2：

#!/bin/bash

USER=root

PASS=123.com

IP=192.168.1.120

expect -c "

spawn ssh $USER@$IP

expect {

\"(yes/no)\" {send \"yes\r\"; exp\_continue}

\"password:\" {send \"$PASS\r\"; exp\_continue}

\"$USER@\*\" {send \"df -h\r exit\r\"; exp\_continue}

}"

方法3：将expect脚本独立出来

login.exp登录文件：

#!/usr/bin/expect

set ip [lindex $argv 0]

set user [lindex $argv 1]

set passwd [lindex $argv 2]

set cmd [lindex $argv 3]

if { $argc != 4 } {

puts "Usage: expect login.exp ip user passwd"

exit 1

}

set timeout 30

spawn ssh $user@$ip

expect {

"(yes/no)" {send "yes\r"; exp\_continue}

"password:" {send "$passwd\r"}

}

expect "$user@\*" {send "$cmd\r"}

expect "$user@\*" {send "exit\r"}

expect eof

执行命令脚本：

#!/bin/bash

HOST\_INFO=user\_info

for ip in $(awk '{print $1}' $HOST\_INFO)

do

user=$(awk -v I="$ip" 'I==$1{print $2}' $HOST\_INFO)

pass=$(awk -v I="$ip" 'I==$1{print $3}' $HOST\_INFO)

expect login.exp $ip $user $pass $1

done

SSH连接信息文件：

# cat user\_info

192.168.1.120 root 123456

## 12.28 批量修改服务器用户密码

旧密码SSH主机信息old\_info文件：

# ip user passwd port

#--------------------------------------

192.168.18.217 root 123456 22

192.168.18.218 root 123456 22

修改密码脚本：

#!/bin/bash

OLD\_INFO=old\_info

NEW\_INFO=new\_info

for IP in $(awk '/^[^#]/{print $1}' $OLD\_INFO); do

USER=$(awk -v I=$IP 'I==$1{print $2}' $OLD\_INFO)

PASS=$(awk -v I=$IP 'I==$1{print $3}' $OLD\_INFO)

PORT=$(awk -v I=$IP 'I==$1{print $4}' $OLD\_INFO)

NEW\_PASS=$(mkpasswd -l 8)

echo "$IP $USER $NEW\_PASS $PORT" >> $NEW\_INFO

expect -c "

spawn ssh -p$PORT $USER@$IP

set timeout 2

expect {

\"(yes/no)\" {send \"yes\r\";exp\_continue}

\"password:\" {send \"$PASS\r\";exp\_continue}

\"$USER@\*\" {send \"echo \'$NEW\_PASS\' |passwd --stdin $USER\r exit\r\";exp\_continue}

}"

done

生成新密码new\_info文件：

192.168.18.217 root n8wX3mU% 22

192.168.18.218 root c87;ZnnL 22

## 12.29 打印乘法口诀

方法1：

# awk 'BEGIN{for(n=0;n++<9;){for(i=0;i++<n;)printf i"x"n"="i\*n" ";print ""}}'

方法2：

for ((i=1;i<=9;i++)); do

for ((j=1;j<=i;j++)); do

result=$(($i\*$j))

echo -n "$j\*$i=$result "

done

echo

done

## 12.30 getopts工具完善脚本命令行参数

getopts是一个解析脚本选项参数的工具。

命令格式：getopts optstring name [arg]

初次使用你要注意这几点：

1. 脚本位置参数会与optstring中的单个字母逐个匹配，如果匹配到就赋值给name，否则赋值name为问号；
2. optstring中单个字母是一个选项，如果字母后面加冒号，表示该选项后面带参数，参数值并会赋值给OPTARG变量；
3. optstring中第一个是冒号，表示屏蔽系统错误（test.sh: illegal option -- h）；
4. 允许把选项放一起，例如-ab

下面写一个打印文件指定行的简单例子，用于引导你思路，扩展你的脚本选项功能：

#!/bin/bash

while getopts :f:n: option; do

case $option in

f)

FILE=$OPTARG

[ ! -f $FILE ] && echo "$FILE File not exist!" && exit

;;

n)

sed -n "${OPTARG}p" $FILE

;;

?)

echo "Usage: $0 -f <file\_path> -n <line\_number>"

echo "-f, --file specified file"

echo "-n, --line-number print specified line"

exit 1

;;

esac

done

# bash test.sh -h

Usage: test.sh -f <file\_path> -n <line\_number>

-f, --file specified file

-n, --line-number print specified line

# bash test.sh -f /etc/passwd -n 1

root:x:0:0:root:/root:/bin/bash

思路扩展：限定脚本参数，将参数保存变量，下面调用变量继续操作。