Contents

[Install Linux 3](#_Toc455947268)

[基本概念 3](#_Toc455947269)

[Linux内核主要功能： 3](#_Toc455947270)

[系统内存管理 3](#_Toc455947271)

[软件程序管理 3](#_Toc455947272)

[硬件设备管理 4](#_Toc455947273)

[文件系统管理 4](#_Toc455947274)

[GNU工具链 4](#_Toc455947275)

[Linux桌面环境 4](#_Toc455947276)

[Linux发行版 4](#_Toc455947277)

[编辑器 5](#_Toc455947278)

[VI编辑器使用 5](#_Toc455947279)

[GNOME编辑器：gedit 6](#_Toc455947280)

[linux shell脚本 6](#_Toc455947281)

[脚本开发流程 6](#_Toc455947282)

[变量 6](#_Toc455947283)

[环境变量 6](#_Toc455947284)

[用户变量 6](#_Toc455947285)

[反引号 7](#_Toc455947286)

[重定向 7](#_Toc455947287)

[管道 7](#_Toc455947288)

[数字运算 7](#_Toc455947289)

[退出脚本 8](#_Toc455947290)

[结构化命令 8](#_Toc455947291)

[处理用户输入 11](#_Toc455947292)

[输入参数 11](#_Toc455947293)

[解析命令行参数 12](#_Toc455947294)

[通用的Linux命令选项 13](#_Toc455947295)

[获取用户输入 14](#_Toc455947296)

[从文件中读取 14](#_Toc455947297)

[数据输出 14](#_Toc455947298)

[文件描述符 14](#_Toc455947299)

[在脚本中重定向输出 15](#_Toc455947300)

[列出打开的文件描述符 15](#_Toc455947301)

[阻止命令输出 16](#_Toc455947302)

[创建临时文件 16](#_Toc455947303)

[记录消息 16](#_Toc455947304)

[控制脚本 16](#_Toc455947305)

[处理信号 16](#_Toc455947306)

[以后台模式运行脚本 17](#_Toc455947307)

[在非控制台下运行脚本 17](#_Toc455947308)

[作业控制 18](#_Toc455947309)

[函数 20](#_Toc455947310)

[函数定义 20](#_Toc455947311)

[函数调用 20](#_Toc455947312)

[函数退出状态码 20](#_Toc455947313)

[函数输出值 21](#_Toc455947314)

[函数输入参数 21](#_Toc455947315)

[可变数组 21](#_Toc455947316)

[函数递归 22](#_Toc455947317)

[怎么调用函数库？ 22](#_Toc455947318)

[软件程序管理 23](#_Toc455947319)

[dpkg 24](#_Toc455947320)

[apt-get 24](#_Toc455947321)

[进程管理 25](#_Toc455947322)

[监测进程 $ps 25](#_Toc455947323)

[实时监测进程 $top 25](#_Toc455947324)

[结束进程 $kill 26](#_Toc455947325)

[磁盘管理 26](#_Toc455947326)

[文件系统 27](#_Toc455947327)

[常见Linux目录名称 27](#_Toc455947328)

[文件系统操作 27](#_Toc455947329)

[查看文件内容 28](#_Toc455947330)

[排序文件内容 28](#_Toc455947331)

[搜索文件内容 28](#_Toc455947332)

[统计文件内容 29](#_Toc455947333)

[文件权限 29](#_Toc455947334)

[用户信息 29](#_Toc455947335)

[用户组 30](#_Toc455947336)

[共享文件做法 30](#_Toc455947337)

[探索Linux文件系统 31](#_Toc455947338)

[压缩及解压工具 32](#_Toc455947339)

[bzip2工具 32](#_Toc455947340)

[gzip工具 32](#_Toc455947341)

[zip工具 32](#_Toc455947342)

[tar命令 32](#_Toc455947343)

[Linux环境变量 32](#_Toc455947344)

[全局环境变量 32](#_Toc455947345)

[局部环境变量 33](#_Toc455947346)

[系统环境变量查找流程： 33](#_Toc455947347)

[数据库 33](#_Toc455947348)

[install MySQL 33](#_Toc455947349)

[install PostgreSQL 34](#_Toc455947350)

[python使用 35](#_Toc455947351)

# Install Linux

usb+\*.iso 安装ubuntu, … 见ubuntu网站

// 解决ubuntu卡，原因是virtualBox资源分配

关闭ubuntu

控制->设置->系统->内存大小，处理器数量，硬件加速，

控制->设置->显示->显卡->3D,2D加速

打开终端 Ctrl+Alt+T

Ubuntu桌面入门指南

https://help.ubuntu.com/

<http://wiki.ubuntu.org.cn/Ubuntu%E6%A1%8C%E9%9D%A2%E5%85%A5%E9%97%A8%E6%8C%87%E5%8D%97>

桌面 -> 控制台 Ctrl + Alt + F1 (F2, …F6)

控制台 –> 桌面 Ctrl + Alt + F7

Switch between window Alt + Tab

# 基本概念

Linux = Linux内核 + GNU工具组件 + 图形化桌面环境 + 应用软件

## Linux内核主要功能：

系统内存管理

内核通过硬盘上的存储空间来实现虚拟内存，这块区域称为交换空间(swap space)。

内存存储单元 = 页面+页面+…

内存会记录哪些内存页面正在使用，并自动把一段时间未访问的内存页面复制到交换空间（称为换出，swapping out），即使还有可用内存。当程序要访问一个已被换出的内存页面，内核必须从物理内存换出另外一个内存页面来给它让出空间，然后从交换空间换入(swapping in)请求的内存页面。只要Linux系统在运行，为运行中的程序换出内存页面的过程就不会停歇。

默认情况下，运行在linux系统上的每个进程都有各自的内存页面。进程不能访问其他进程正在使用的内存页面。

$cat /proc/meminfo //查看系统虚拟内存

软件程序管理

内核创建第一个进程(init进程)来启动系统上所有其他进程。一些Linux发行版使用一个表来管理在系统开机时要自动启动的进程，Ubuntu则采用/etc/init.d目录，将开机时启动或停止某个应用的脚本放在这个目录下。

Linux操作系统有7个启动运行级：

运行级为1时，只有基本的系统进程会启动，同时会启动唯一一个控制台终端进程。（单用户模式）

标准的启动运行级为3,大多数应用软件，比如网络支持程序，都会启动

运行级为5，系统会启动图形化的X Window系统，同时允许用户通过图形化桌面窗口登录系统。

$ps ax //查看运行在linux系统上的进程

硬件设备管理

Linux系统将硬件设备当成特殊的文件，称为设备文件，有三种：

字符型设备文件：指处理数据时每次只能处理一个字符的设备，比如终端

块设备文件：指处理数据时每次能处理大块数据的设备，比如硬盘

网络设备文件：指采用数据包发送和接收数据的设备，包括网卡和回环设备。

Linux为系统上的每个设备都创建一种特殊的文件，称为节点，与设备的的所有通信都是通过设备节点来完成。

文件系统管理

内核必须在编译时就加入对所有可能用到的文件系统的支持。

Linux内核采用虚拟文件系统（Virtual File Sytem, VFS）作为和每个文件系统交互的接口。

## GNU工具链

处理文件

操作文本

管理进程

Shell：提供启动程序、管理文件和进程

## Linux桌面环境

X Window系统

KDE桌面：类windows桌面

GNOME桌面：Red Hat默认桌面

图形化桌面环境会占用很多的系统资源。而Linux的卖点不是可以运行在功能较弱的早期PC上

## Linux发行版

完整的Linux系统包。大多发行版是为某个特定用户群定制的。

Red Hat: 用于Internet服务器的商业发行版

Fedora: 从Red Hat分享出的家用发行版

Debian: 在Linux专家和商用Linux产品中流行的发行版

Ubuntu: 免费的学校和家族Linux发行版

# 编辑器

## VI编辑器使用

Vi编辑器使用控制台图形模式来模拟文本编辑窗口，允许查看文件中的行，在文件中移动，以及插入、编辑和替换文本

Vim编辑器在内存缓冲区中处理数据。

$vi myprog.c

若启动vi时未指定文件名，或者这个文件不存在，vi会新开一段缓冲区来编辑。

若命令行来已有文件，vi会将文件的整个内容都读到一块缓冲区来准备编辑

刚打开文件时，vi编辑会进入普通模式。在普通模式中，vi编辑器会将按键解释成命令

按下i键，进入插入模式。在插入模式中，vi会将你在当前光标位置输入的每个键都插入到缓冲区。

按退出键ESC，则又进入普通模式

在普通模式下，你可以用方向键，page down/up在文本区域移动光标

在普通模式下，编辑缓冲区中的数据

x删除字符 （vi编辑器会将删除键识别成x命令，但不识别退格键）

dw删除单词

dd删除行

d$删除光标至行尾

a光标后追加数据

A光标行尾追加数据

复制：

移动光标至起始位置，接下v键，移动光标选择要复制的内容，按y键，激活copy

移动光标至目标位置，按p，激活粘贴

查找：

按斜线/, 输入查找文件，按回车键。光标会跳至匹配的文本，按n键，跳至下一个

替换：

:s/old/new old -> new

在普通模式下，按下冒号键:,进入命令行模式。在命令行模式下，将缓冲区的数据保存到文件中并退出vi

:q 未修改缓冲区数据，退出

:q! 取消对缓冲区数据的修改，退出

:w filename save as功能

:wq save功能

## GNOME编辑器：gedit

$gedit fact.sh myprog.c

Edit –> Preferences -> Editor -> Tab Width: 4

->Font&Colors 调用背景和字体

# linux shell脚本

## 脚本开发流程

1. 构建脚本文件

$vi test.sh

#!/bin/bash

脚本命令

1. 赋予执行权限

$chmod u+x test.sh

1. 执行脚本

$./test.sh 在提示符用绝对或相对路径引用shell脚本文件

echo “installing …” 显示消息

## 变量

### 环境变量

echo “HOME: $HOME” 访问环境变量HOME

shell维护着一组环境变量，用来记录特定的系统信息。比如系统的名称，登录到系统上的用户名称，用户的系统ID，用户的默认主目录，shell查找程序的搜索路径等等，可以用set命令显示全部环境变量列表

$set 但也可以在环境变量前加$来访问

### 用户变量

在脚本的整个生命周期里，shell脚本中定义的变量会一直保持它们的值，但在shell脚本完成时删除掉

guest=”qzlin” 变量、等号和值之间不能出现空格

var1=10 shell脚本会自动决定变量值的数据类型

var2=$guest 获取变量值需要使用$

## 反引号

output=`date` 反引号允许将shell命令的输出赋给变量

today=`date +%y%m%d`

ls /usr/bin -al > log.$today

## 重定向

command > output file 输出重定向

command < input file 输入重定向

date > log 如果存在输出文件，则新文件数据覆盖

date >> log 追加数据到输出文件

wc < file 输入文件file流向命令wc,进行统计

command << EOF 内联输入重定向，允许在命令行指定数据

数据

EOF

wc << EOF

>test string 1

>test string 2

>EOF

## 管道

command1 | command2

Linux系统会同时运行这两个命令，在系统内部将它们连接起来，在第一个命令产生输出的同时，输出会被立即送给第二个命令。

dpkg -l | sort > log.list

## 数字运算

使用bash计算器bc + 内联输入重定向

variable=`bc << EOF

options

statements

expressions

EOF

`

var1=10.46

var2=43.33

var3=33.2

var4=71

var5=`bc << EOF

scale=4

a1=($var1\*$var2)

b1=($var3\*$var4)

a1+b1

EOF

`

echo “The final answer for this mess is $var5”

## 退出脚本

shell中运行的每个命令都使用退出状态码来告诉shell它完成了处理。退出状态码是0~255之间的整数值，在命令结束运行时由命令传给shell.

exit 0

## 结构化命令

Bash shell的if语句会运行if行定义的命令。如果该命令的退出状态码是0(该命令成功运行)，位于then部分的命令就会被执行。如果该命令的退出状态码是其他什么值，那then部分的命令就不会被执行，bash shell会继续执行命令脚本中的下一个命令

if command; then

commands

fi

if command; then

commands

else

commands

fi

if command; then

commands

elif command; then

commands

elifc command; then

commands

fi

记住，bash shell会依次执行if语句，只有第一个返回退出状态码0的语句中的then部分被执行

if [ condition ]; then

commands

fi

test命令[], condition可以是数值比较，文件串比较，文件比较。

test命令提供了在if-then语句中测试不同条件的途径。如果test命令中列出的条件成立，test命令就会退出并返回退出状态码0；如果条件不成立，test命令就会退出并返回退出状态码1。

if [ $val1 \> $val2 ] 比较操作

if [ -n “$val1” ] 检查一个变量是否含有数据, 相当于!isEmpty(val1)

if [ -z “$val1” ] 相当于isEmpty(val1)

if [ -e $HOME ] 检查文件或目录对象是否存在

if [ -d $HOME ] 检查指定的文件名是否是目录

if [ -f $HOME ] 检查指定的文件名是否是文件

if [ -r $file ] 检查文件能否可读

if [ -w $file ] 检查文件是否有可写权限

if [ -x $file ] 检查文件是否可执行

if [ -s $file ] 检查文件是否为空

if [ -o /etc/passwd ] 检查你是否是文件的属主

if [ ./test19 –nt ./test18 ] 检查某文件是否比另一文件更新

if [ ./test19 –ot ./test18 ] 检查某文件是否比另一文件更老

if [ condition1 ] && [ condition2 ] || [ condition3 ]

if (( $val \*\* 2 > 90 )); then

双圆括号命令允许将高级数学表达式放入比较中

If [[ $USER == r\* ]]

双方括号命令针对字符串，表达式可以使用模式匹配

case variable in

pattern1 | pattern2)

commands1;;

pattern3)

commands2;;

\*)

default commands;;

esac

处理某个目录下的所有文件、系统上的所有用户或是某个文本文件中的所有行

for var in list

do

name=`basename $var` basename命令返回程序名而不包括路径

commands

done

for test in Nevada “New Ham” “new Mexico” “new York”

do

echo $test

done

从命令输出取值

for state in `cat /tmp/file` 以文件行读取

do

echo $state

done

环境变量IFS（称为内部字段分隔符internal field separator）. IFS环境变量定义了bash shell用作字段分隔符的一系列字符。默认情况下，bash shell会将下列字符当作字段分隔符:空格，制表符，换行符。你也可以在shell脚本中临时更改IFS环境变量的值来限制一下被bash shell当作字段分隔符的字符

IFS.OLD=#IFS

IFS=$’\n’ or IFS=:

for state in `cat /tmp/file`

do

echo $state 以字段分隔符读取

done

用通配符读取目录

for file in /home/\*

do

if [ -d “$file” ]; then

echo “$file is a directory”

elif [ -f “$file”]; then

echo “$file is a file”

fi

done

for (( variable assignment; condition; iteration process ))

for (( a=1, b=10; a <= 10; a++, b-- ))

while命令某种意义上是if-then语句和for循环的混杂体

while [ $var –gt 0 ]

do

commands

done

until命令和while命令工作的方式完全相反

until [ condition ]

do

commands

done

for (( i=1; i<=3; i++))

do

for (( j=1; j<=3; j++))

do

echo “$i, $j”

done

done

vi

break命令 and continue命令同c语言

for var in 1 2 3 4

do

if [ $var –eq 3 ]

then

break or continue

fi

done

循环输出可以直接重定向到文件中,或管接到下一命令中

for file in /home/\*

do

echo “$file”

done > output.txt

## 处理用户输入

### 输入参数

位置参数：分配给命令行输入的所有参数。

位置参数变量是标准的数字：$0是程序名，$1是第一个参数，…

每个参数要么空格分割，要么是引号分割

$# 命令行参数个数

$\* 命令行所有参数， 按字符串保存

$@ 参数数组

for param in “$@”

do

echo “$param”

done

shift命令：移位参数

-a 选项options

-- 双破折线表明选项结束，双破折线之后的命令行参数当做参数来处理，而不是选项

### 解析命令行参数

getopt optstring options parameters

$getout ab:cd -a -b test1 -cd test2 test3

处理结果：-a -b test1 -c -d test2 test3

在脚本中使用getopt

set -- `getopt -q ab:c “$@”`

while [ -n “$1” ]; do

case “$1” in

-a) …

-b) param=”$2”

shift ;;

-c) …

--) shift

break;;

\*) echo “$1” is not an option;;

esac

shift

done

set命令的选项之一是双破折线，它会将命令行参数替换为set命令的命令行的值。

推荐使用getopts

getopts optstring variable

while getopts :ab:c opt

do

cat “$opt” in

1. echo “Found the –a option” ;;
2. echo “Found the –b option, with value $OPTARG”
3. echo “Found the –c option” ;;

\*) echo “Unknown option: $opt” ;;

esac

done

shift $[ $OPTIND – 1 ]

count=1

for param in “$@”

do

echo “Parameter $count: $param”

count=$[ $count + 1]

done

有效的选项字母都会列在optstring中，如果选项字母要求有个参数值，就加一个冒号。要去掉错误消息的话，可以在optstring之前加一个冒号

getopts命令会用到两个环境变量，如果选项需要跟一个参数值，

OPTARG环境变量就会保存这个值。

OPTIND环境变量保存了参数列表中getopts正在处理的参数位置

$./test19 -ab test1 -c test2 test3 test4

Found the -a option

Found the -b option, with value test1

Found the -c option

Parameter 1: test2

Parameter 2: test3

Parameter 3: test4

### 通用的Linux命令选项

-a 显示所有对象

-c 生成一个计数

-d 指定一个目录

-e 扩展一个对象

-f 指定读入数据的文件

-h 显示命令的帮助信息

-I 忽略文本大小写

-n 使用非交互模式（批量）

-o 指定将所有输出重定向到输出文件

-q 以安静模式运行

-r 递归地处理目录和文件

-s 以安静模式运行

-v 生成详细输出

-x 排队某个对象

-y 对所有问题回答yes

### 获取用户输入

echo -n “Enter your name:”

read -t 5 name

echo “Hello $name”

read命令接受从标准输入（键盘）或另一个文件描述符的输入

-t 5选项来指定一个计时器，指定等待的秒数，当计时器过期后，read命令会返回珍上非0退出状态码

read -n1 -p “Do you want to continue [Y/N]? “ answer

case $answer in

Y | y) echo “find, continue on…”;;

N | n) echo OK, goodbye exit;

esac

-n选项和值1一起使用，告诉read命令在接受单个字符后退出。只要你按下单个字符回答后，read命令就会接受输入并将它传给变量，而不必按回车键

-p选项允许你直接在read命令行指定提示符

read -s -p “Enter your password: “ pass

echo “Is your password really $pass?”

-s选项会阻止将传给read命令的数据显示在显示器上（实际上，数据会被显示，只是read命令会将文本颜色设成跟背景一样）

### 从文件中读取

使用read命令来读取Linux系统上文件里保存的数据。每次调用read命令会从文件中读取一行文本。当文件中再没有内容时，read命令会退出并返回非零退出状态码

最常见的方法是：将文件运行cat命令后的输出通过管道直接传给含有read命令的while命令

cat test | while read line

do

echo “$line”

done

## 数据输出

### 文件描述符

Linux系统将每个对象当做文件来处理。这包括输入和输出的过程。Linux用文件描述符来标识每个文件对象。文件描述符是一个非负整数，可以唯一地标识会话中打开的文件。每个过程一次最多可以有9个文件描述符

STDIN (0: 文件描述符) 标准输入

STDIN文件描述符代表shell的标准输入，对终端界面来说，标准输入是键盘

在使用输入重定向符号<时，Linux会用重定向指定的文件来替换输入文件描述符。

许多bash命令能接受STDIN的输入，尤其是没有在命令行上指定文件的话。如

$cat

$cat < testfile

STDOUT （1：文件描述符）标准输出

文件描述符代表标准的shell输出，在终端界面上，标准输出就是终端显示器

默认情况下，很多bash命令会定向输出到STDOUT文件描述符

通过输出重定向符号，通常会显示出显示器的所有输出会被shell重定向到指定的重定向文件。你也可以将数据追加到某个文件。可以用>>符号来完成

$ls -l > testfile

$who >> testfile

STDERR (2: 文件描述符) 标准错误

shell通过特殊的STDERR文件描述符来处理错误消息

默认情况下，STDERR文件描述符会和STDOUT文件描述符指向同样的地方，也就是说默认情况下，错误消息也会输出到显示器输出中

重定向STDOUT并不会自动重定向STDERR。

$ls -al badfile 2> errFile

只重定向错误消息，将文件描述符2放在重定向符号前

$ls -al test test1 test2 2> errFile 1> outFile

Shell利用1>符号将ls命令的本该输出STDOUT的正常输出重定向到了errFile

2>符号将ls命令的本该输出STDERR被重定向outFile

### 在脚本中重定向输出

echo “this is normal output” 标准输出

echo “This is an error message” >&2

在脚本中生成错误消息，并输出到STDERR文件描述符

### 列出打开的文件描述符

$lsof -a -p $$ -d 0,1,2

lsof显示当前Linux系统上打开的每个文件的有关信息

-p $$选项指示进程的PID

-d 0,1,2指定显示的文件描述符

-a选项用来对其他两个选项的结果执行布尔AND运算

### 阻止命令输出

不想显示脚本的输出，将脚本作为后台进程执行时很常见。要解决这个问题，可以将STDERR重定向到一个称作null文件的特殊文件。null文件跟它的名称很像，文件里什么都没有。Shell输出到null文件的任何数据都不会保存，这样它们就都被丢掉了

在Linux系统上null文件的标准位置是/dev/null.你重定向到该位置的任何数据都会被丢掉

$ls -al > /dev/null

$cat /dev/null 显示为空

你也可以在输入重定向将/dev/null作为输入文件。由于/dev/null文件不含有任何内容，程序员通常用它来快速移除现有文件中的数据而不用先删除文件再创建

$cat /dev/null > log

### 创建临时文件

Linux发行版配置了系统在启动时自动删除/tmp目录的所有文件

系统上的任何用户帐户都有权限在/tmp目录中读和写。这个特性为你提供了简单地创建临时文件的途径，而不用管理清理工作。

mktemp命令可以轻松地在/tmp目录中创建一个唯一的临时文件.一旦创建了文件，你就在脚本中有了完整的读写权限，但其他人没法访问它（除了root）

$tempfile=`mktemp -t testing.XXXXXX`

mktemp命令会用6个字符码替换这6个x,从而保证文件名在目录中是唯一的。

-t选项指示mktemp命令会返回临时文件的全路径

### 记录消息

$date | tee logfile

Tee命令相当于管道的一个T型接头。它将从STDIN过来的数据同时发给两个目的地。一个目的地是STDOUT, 另一个目的是tee命令行所指定的文件名

$date | tee -a logfile

-a表示数据追加到文件中

## 控制脚本

### 处理信号

Linux通过信号来在运行在系统上的进程之间通信，用这些信号来控制shell脚本的运行。通过对脚本进行编程使其在收到Linux系统的特定信号时执行特定的命令

如果bash shell收到信号，shell会将这些信号传给shell脚本程序来处理。而shell脚本的默认行为是忽略这些信号。你可以脚本中加入识别信号和处理信号结果的命令。

终止进程

Ctrl+C生成SIGINT信号(2),并将其发送给shell中当前运行的所有进程

暂停进程

Ctrl+Z生成SIGTSTP信号（18），停止shell中运行的任何进程。停止一个进程跟终止进程不同，停止进程会让程序继续保留在内存中，并能从上次停止的位置继续运行。

Shell将shell中运行的每个进程称为作业，并为每个作业分配一个唯一的作业号。它会给第一个作业分配作业号1,第二个作业号2,…

$exit 退出shell

$ps au 查看已停止的作业

$kill -9 停止作业的PID 发送SIGKILL信号来终止进程

捕捉信号

Trap命令允许你来指定shell脚本要观察哪些Linux信号并从shell中拦截。如果脚本收到了trap命令中列出的信号，它会阻止它被shell处理，而在本地处理它。

trap commands signals

trap “echo ‘ Sorry! I have trapped Ctrl-C’” SIGINT SIGTERM

### 以后台模式运行脚本

在后台background运行进程，进程运行时不会和终端会话上的STDIN，STDOUT 和STDERR关联

$ ./test1 &

要在命令行界面以后台模式运行shell脚本，只要在命令后加个&符就行了。

当&符放到命令后时，它会将命令和bash shell分享开来，并将它作为系统上的独立后台进程运行。

每当启动新作业时，Linux系统都会为其分配一个新的作业号和PID.通过ps命令，你可以看到所有脚本都在运行中

$ps au

以上每个后台进程都绑定到了该终端会话的终端上了(pts/0)。如果进程会话退出了，后台进程也会退出

### 在非控制台下运行脚本

$nohup ./test1 &

若想退出控制台后，脚本继续以后台形式运行

nohup命令运行了另外一个命令来阻止所有发送给该进程的SIGHUP信号。这会在退出终端会话时阻止进程退出

nohup命令会从终端解除进程的关联，进程会丢掉到STDOUT and STDERR的链接。为了保存该命令产生的输出，nohub命令会自动将STDOUT and STDERR的消息重定向一个名为nohup.out的文件中

$cat nohup.out

### 作业控制

启动、停止、无条件终止以及恢复作业的这些功能统称为作业控制，通过作业控制，你就能完全控制shell中正在运行的所有作业

$jobs

jobs命令允许查看shell当前正在处理的作业

jobs命令输出中的加号和减号。带加号的作业会被当做默认的作业。带减号的作业则会在当前默认作业完成处理时成为下一个默认作业。

在bash作业控制中，你可以将已停止的作业作为后台进程或前台进程重启。

$bg 2 以后台模式重启一个作业，可用bg命令加上作业号

$fg 1 以前台模式重启作业，用fg命令加作业号

#### 调整进程优先级

在多任务操作系统中，内核负责将CPU时间分配给系统上运行的每个进程。实际上每次只有一个进程可以运行在CPU上，所以内核将CPU时间轮流分配给每个进程

默认情况下，从shell启动的所有进程在Linux系统上都有同样的调试优先级。调度优先级是内核分配给该进程的相对于分配给其他进程的CPU时间总量

调度优先级是个整数值，从-20（最高优先级）到+20（最低优先级）。默认情况下bash shell以优先级0来启动所有进程

$nice -n 10 ./test4 > test4out & 指定新的优先级

$renice 10 -p PID号 重新指定进程的优先级

#### 定时运行作业

at命令允许指定Linux系统何时运行脚本。at命令会将作业提交到队列中指定shell何时运行该作业。at的守护进程atd会以后台模式运行，并检查作业队列来运行作业。大多数Linux发行版会在启动时运行时守护进程

atd守护进程会检查系统上的一个特殊目录（通常位于/var/spool/at）来获取用at命令提交的作业。默认情况下，atd守护进程会每60s检查一下这个目录。有作业时，atd守护进程会检查作业设置运行的时间。

如果指定了一个已经过去的时间，at命令会在第二天该时间运行该作业

at -f filename 10:15~PM

当作业在Linux系统上运行时，没有屏幕会关联到该作业。取而代之的是，Linux系统会将提交该作业的用户的email地址作为STDOUT and STDERR。任何发到STDOUT or STDERR的输出都会通过邮件系统发送给该用户

$atq 查看系统中有哪些作业在等待

$atrm 作业号 删除等待中的作业

#### 定期执行脚本

cron程序来计划要定期执行的作业。cron程序会在后台运行并检查特殊的称做cron时间表，来获得计划执行的作业。

cron时间表

min hour dayofmonth month dayofweek command

15 10 \* \* \* /home/rich/test4 > test4out

每天的10:15运行命令

cron目录

当你创建的脚本不要求有精确的执行时间时，用预配置的cron脚本目录会更方便。

$ls /etc/cron.\*ly

/etc/cron.daily

/etc/cron.hourly

/etc/cron.monthly

/etc/cron.weekly

因此，如果你有脚本需要每天运行一次，只要将脚本复制到daily目录，cron就会每天执行它

#### 开机时自动运行脚本

开机过程：

在你开始运行Linux系统时，Linux内核会加载到内存中并运行。这个过程叫System V init过程。System V init过程会读取/etc/inittab文件，inittab文件会列出系统的运行级run level,不同的Linux运行级会启动不同的程序和脚本。Ubuntu会以运行级2运行。

每个运行级将会定义System V init过程启动或停止哪些脚本。这些开机脚本都是shell脚本，通过提供必要的环境变量来启动应用程序

Ubuntu开机脚本/etc/init.d/rc

最好不要和Linux发行版上已有的独立开机脚本文件混起来。通常发行版会提供一些工具来在你添加服务器程序时自动构建这些脚本，而手动修改这些脚本可能会造成各种问题。大多数Linux发行版提供了一个本地开机文件专门让系统管理员添加开机时运行的脚本

在本地开机文件中，你可以指定特定命令或语句，或者输入任何你想在开机时启动的脚本。记住，如果你要用脚本，你必须指定该脚本的全路径名，这样系统才能在开机时找到

Ubuntu本地开机文件： /etc/rc.local

#### 新shell中自动运行脚本

每个用户的主目录下都有两个文件，bash shell会用它们来自动启动脚本并设置环境变量：

.bash\_profile

.bashrc

当新shell是新的登录生成的话，bash shell会运行.bash\_profile文件。可以把任何登录时运行的脚本放到该文件中。

如果你想为系统中的所有用户运行一个脚本，大多数Linux发行版提供了/etc/bashrc文件.每次有用户在系统上启动一个新的bash shell时，bash shell就会执行该文本中的语句

## 函数

### 函数定义

function name {

commands

}

或者

name() {

commands

}

### 函数调用

name

重定义了函数，新定义会覆盖原函数的，不会产生任何错误消息

### 函数退出状态码

bash shell会把函数当做小型脚本，运行结束时会返回一个退出状态码。

默认情况下，函数的退出状态码是函数中最后一条命令返回的退出状态码。可以使用return命令来退出函数并返回特定的退出状态码。return命令允许指定一个整数值来定义函数的退出状态码。

function dbl {

read –p “Enter a value: “ value

return $[ $value \* 2 ]

}

### 函数输出值

function dbl {

read –p “Enter a value: “ value

echo $[ $value \* 2 ]

}

result=`dbl` 将函数dbl的输出存入变量result

### 函数输入参数

function addem {

if [ $# -eq 0 ] || [$# -gt 2 ]

then

echo -1

elif [ $# -eq 1 ]

then

echo $[ $1 + $1 ]

else

echo $[ $1 + $2 ]

fi

}

value=`addem 10` 结果是20

value=`addem 10 15` 结果是25

特殊变量$#是函数的参数数目

函数名$0, 函数命令行的任何参数会是$1, $2, …

默认情况下，在脚本中定义的任何变量都是全局变量，在函数外定义的变量可以在函数内正常访问。

函数内部定义的local变量是局部变量

local temp

## 可变数组

$mytest=(one two three four five) 初始化数组

$mytest[0] = seven 修改数组元素

$ unset mytest[0] 删除数组元素

$echo ${mytest[0]} 显示数组元素

$echo ${mytest[\*]} 显示整个数组

函数传数组作为参数

function addarray {

local sum=0

local newarray

newarray=(`echo “$@”`) 函数内接受数组

for value in ${newarray[\*]} 遍历数组

do

sum=$[ $sum + $value ]

done

echo $sum 函数返回值

}

myarray=(1 2 3 4 5)

addarray ${ myarray [\*]} 函数调用时传入数组参数

echo ${myarray[\*]} 函数返回数组

## 函数递归

局部函数变量提供的一个特性是自成体系(self-containment)。自成体系的函数不需要使用任何外部资源，除了脚本在命令行上传给它的变量

function fac {

if [ $1 -eq 1 ]

then

echo 1

else

local temp=$[ $1 – 1 ]

local result=`fac $temp`

echo $[ $result \* $1 ]

fi

}

$chmod u+x fac.sh

## 怎么调用函数库？

可否直接$./fac.sh，然后$fac 3? 不可以

问题出在shell函数的作用域上。和环境变量一样，shell函数只对定义它的shell会话有效。如果你在shell命令行界面的提示符下运行./fac.sh脚本，shell会创建一个新的shell并在新shell中运行这个脚本。它会为那个新shell定义函数，但当你运行另外一个用到这些函数的脚本时，它们却不可用。

解决方法：创建库

bash shell允许创建函数库文件，然后在需要时在多个脚本中引用该文件

Source命令会在当前的shell上下文中执行命令，而不是创建一个新的shell来执行命令。可以用source命令在shell脚本中运行库文件脚本。Source命令有个快捷别名，称为点操作符

$. ./fac.sh

$fac 3

新shell启动自动运行函数库文件

bash shell会在每次启动时在主目录查找.bashrc文件。只要在文件末尾加上自定义函数或者. /home/rich/lib/fac.sh

命令别名

允许为通用命令（和它们的参数一起）创建一个别名，这样就能通过最少的键入调用想要的命令

$ lias -p 查看已有的别名列表

$ alias li=’ls -il’ 创建命令别名

命令别名的行为和局部环境变量差不多，只在定义它们的shell进程中有效

$man man //man命令，访问存储在linux系统上的手册页面

$man bash

…

# 软件程序管理

包管理系统Package Management System, PMS

每个主要的Linux发行版都利用包管理系统的某些形式来控制安装软件应用和库。PMS利用一个数据库来记录：

Linux系统上已安装了什么软件包

每个包安装了什么文件

每个已安装软件包的版本

软件包存储在服务器上，并通过运行在本地Linux系统上的PMS工具通过互联网访问。这些服务器称为库repository。你可以用PMS工具来搜索新的软件包。或者是系统上已安装软件包的更新。

Linux中广泛使用的两个主要PMS基础工具：dpkg and rpm

基于Debian的发行版，比如Ubuntu and Linux Mint,在它们PMS工具的底层用的是dpkg命令

基于Red Hat的发行版，比如Fedora, openSUSE and Mandriva,在它们PMS的底层用的是rpm命令。

dpkg命令是基于Debian系统PMS工具的核心。包含在这个PMS中的其他工具有：apt-get, apt-cache, aptitude.

## dpkg

$man dpkg

$dpkg -? 查询dpkg的使用命令

$dpkg -l 显示所有安装包

($dpkg -l | wc) 显示行数，字数，字节数

$dpkg -L python 列出安装包的所有文件清单

$dpkg -s python 显示包详细信息

## apt-get

默认的软件库位置：/etc/apt/sources.list

$sudo vi /etc/apt/sources.list

格式如下：

deb http://site.example.com/debian distribution component1 component2 component3

deb-src http://site.example.com/debian distribution component1 component2 component3

deb代表编译好的包（des-src代表源代码）

http是软件库的web地址

distribution代表发行版本，比如 12.07 是 precise

component1, …表明库里面有什么类型的包

更改了/etc/apt/sources.list，要用用 apt-get update 更新 index 文件, 更新的 index 文件在 /var/lib/apt/lists/

PPA：Personal Package Archives，也就是个人软件包集。

有很多软件因为种种原因，不能进入官方的 Ubuntu 软件仓库。为了方便 Ubuntu 用户使用，launchpad.net 提供了ppa，允许用户建立自己的软件仓库，自由的上传软件。PPA也被用来对一些打算进入 Ubuntu 官方仓库的软件，或者某些软件的新版本进行测试。

sudo add-apt-repository ppa:user/ppa-name

apt-get update //更新软件列表

apt-get -y install packagename //安装一个新软件包

-y跳过系统提示，直接安装

apt-get remove --purge packagename //卸载一个已安装的软件包

--purge删除配置文档, 若没有，则保留配置文档

apt-get autoremove --purge packagname //删除包及其依赖的软件包

--purge删除彻底，包括配置文件

apt-get upgrade //更新软件

apt-get dist-upgrade //更新系统版本

命令帮助

apt-get --help

源码安装(.tar, .tar.gz, .tar.bz2, .tar.Z)

1. 解压

解xx.tar.gz，xx.tar.Z，xx.tgz，xx.tar：tar -zxf xx.tar.gz

（或tar -zxf xx.tar.Z 或tar -zxf xx.tgz 或tar -xf xx.tar）

解xx.bz2：bunzip2 xx.bz2

1. 进入到解压出的目录中

建议先读一下README之类的说明文件，因为此时不同源代码包或者预编译包可能存在差异，然后建议使用ls -F --color或者ls -F命令（实际上我的只需要 l 命令即可）查看一下可执行文件，可执行文件会以\*号的尾部标志。

1. 编译，链接，创建可执行文件

$ ./configure

$ make

1. 安装可执行文件

$ sudo make install

即可完成安装。

# 进程管理

## 监测进程 $ps

$ps

默认情况下,ps命令只会显示运行在当前控制台下的属于当前用户的进程。

$ps -efH

-e显示所有进程

-f显示完整格式的输出

-H层级格式显示进程

## 实时监测进程 $top

$top

ps命令只能显示某个特定时间点的信息；top命令能够实时显示进程信息，具体见<linux命令行与shell脚本编程大全>p82

## 结束进程 $kill

Linux没用了Unix进行进程间通信的方法。在Linux上，进程之间通过信号来通信。进程的信号就是预定义好的一个消息，进程能识别它并决定忽略还是作出反应。大多数写得好的程序都能接收和处理标准Unix进程信号

$kill 3940

默认情况下,kill命令会向命令行中列出的全部PID发送一个TERM信号，注意发送进程信号的必须是进程的属主或root

TERM信号告诉进程可能的话停止运行。

推荐杀死进程流程：先试试TERM信号。如果进程忽略它，可用INT或HUP信号。程序收到这些信号，会在关掉进程前有序地停止它正在做的事。KILL信号的强制性最强。当进程接收到这个信号时，它会立即停止运行。这可能会导致文件损坏。

$kill -s HUP 3940

$killall http\*

killall命令支持通过进程名而不是进程号来结束进程。Killall命令也支持通配符，这在系统因负载过大而变得很慢时很有用

# 磁盘管理

Linux系统将所有磁盘都挂载到一个虚拟目录下。在使用新的存储媒体之前，你需要把它放到虚拟目录下。这项工作称为挂载mounting

在今天的图形化桌面环境里，大多数Linux发行版都能自动挂载指定的可移动存储媒体。

$mount //输出当前系统上挂载的设备列表

$mount -t ntfs /dev/sdb1 /media/disk

媒体设备挂载到了虚拟目录后，root用户就有了对该设备的所有访问权限，而其他用户的访问则会被限制

$cd /media/disk //进入磁盘操作

从Linux系统上移除一个可移动设备时，不能直接从系统上移除，而应该先卸载它。

$umount [directory | device]

umount命令支持通过设备文件或者是挂载点来指定要卸载的设备

$lsof [directory | device]

如果在卸载设备时，系统提示设备繁忙，无法卸载设备，通常是有进程还在访问该设备或使用该设备上的文件。这时可以lsof命令获得使用它的进程信息，然后在应用中停止使用该设备或停止该进程。

$df -h 查看所有已挂载磁盘的使用情况

$du -h 显示某个特定目录的磁盘使用情况，默认是当前目录,一般用来判断某个目录下是不是有超大文件

# 文件系统

Linux将文件存储在单个目录结构中，这个目录我们称之为虚拟目录。

## 常见Linux目录名称

/bin 存放GNU用户级的工具

/dev 设备目录Linux在这里创建设备节点

/etc 系统配置文件目录

/mnt 挂载目录，存放设备挂载点

/opt 可选目录，存放可选的软件包

/usr 用户安装软件的目录

/var 可变目录，比如日志文件

$cd //切换目录，若没有目标路径，则切换至用户主目录

## 文件系统操作

$ls -Fal

-F区分文件和目录

-a显示隐藏文件，普通文件，普通目录

-l详细信息

$ls -l mypro\* //过滤输出列表，ls命令配合模式匹配

$ mkdir -p src/main/scala // create directory

$ mv src dst // rename directory

$ rm -rf directoryName // delete directory

$ touch build.gradle // create file

$mv test1 test2 //重命令文件名或目录名

$ rm filename // delete files

$stat test1 //查看文件状态信息

$file test1 //查看文件类型（文本，可执行，数据文件）

$cp src dst

若src and dst都是文件名，则cp命令将源文件复制至一个新文件具以dst命名。如果目标文件已存在，会提示是否覆盖，按y

若src是文件，dst是目录，则将文件复制到目录中

若src and dst都是目录，配合-R参数递归复制整个目录$cp -R srcDir dstDir

链接：虚拟的副本

软链接：符号链接

硬链接

默认情况下ln命令会创建硬链接

硬链接文件采用和源文件相同的索引节点号。硬链接会一直维持这个索引节点号来保留数据，直到你删除了最后一个硬链接它的文件。

软链接文件都知道它所指向的文件不在了，所以指向的也就是一个无效文件

$ unlink /usr/local/bin/sts // delete link

## 查看文件内容

$cat test1 显示整个文件内容

$more test1 按页显示内容

$less test1 前后翻动和搜索

$tail -n 20 log 默认显示文件末尾10行，

-f允许在其他进程使用该文件时查看文件的内容。Tail命令会保持活动状态并不断地显示添加到文件中的内容。这是实时监测系统日志的绝妙方式。

$head log 显示文件开头

## 排序文件内容

$sort test1 默认情况下，sort命令按字符排序

$sort -n test1 按值排序

$sort -M log 按月排序，Linux日志文件经常会在每行的起始位置有一个时间戳

$sort -t’: ‘ -k 3 -n /etc/passwd 按字段分隔的数据进行排序

-t指定字符分配符

-k指定排序的字段

$du -sh \* | sort -nr 查看目录下的哪些文件占用空间最多

## 搜索文件内容

$grep [options] pattern [file]

grep命令会到输入中或你指定的文件中查找包含匹配指定模式的字符的行，输出是包含了匹配模式的行

$grep -n three file1 在文件file1中搜索能匹配模式three的文本

$grep -v three file1 输出不匹配该模式的行

$grep [tf] file1 正则表达式来匹配模式，grep搜索包含t或者f字符的匹配

## 统计文件内容

$wc file1 行数，字数，字节数

## 文件权限

### 用户信息

Linux系统使用/etc/passwd文件保存与用户有关的信息

/etc/passwd文件：包含了所有系统用户帐户列表以及每个用户的基本配置信息

用户名:用户密码:用户ID:组ID:用户全名:用户默认主目录:用户默认shell程序

Linux系统会为各种各样的功能创建不同的用户帐户，而这些帐户并不真的用户。这些帐户称作系统帐户，是系统上运行的各种服务进程访问资源用的特殊帐户。所有运行在后台的服务都需要一个系统用户帐户登录到linux系统上。

在安全成为一个大问题之前，这些服务经常会用根帐户登录。但是，如果有非授权的用户攻入了这些服务中的一个，他就能作为root用户进入整个系统了。为了防止这种情况发生，现在几乎每个Linux服务器上后台运行的服务都有自己的用户帐户。这样，即使有人攻入了某个服务，他也无法访整个系统。

Linux系统将用户密码保存在/etc/shadow文件中，只有特定的程序才能访问这个文件，比如登录程序。

#### 创建用户

$ useradd -D 显示创建新用户的默认值

创建新用户时，管理员会创建一份默认的HOME目录配置，（作为创建新用户HOME目录的模板），自动在每个新用户的HOME目录里放置默认的系统文件

$ useradd -m test 创建新用户test

$ls -al /home/test

#### 删除用户

$userdel -r test

默认情况下，userdel命令只删除/etc/passwd文件中的用户信息，而不会删除系统中属于该账户的任何文件,所以需要加-r

#### 修改用户

usermod命令用来修改/etc/passwd文件中的大部分字段，只需要用与想修改的字段对应的命令行参数就可以

$usermod -i qizhonglin test 修改用户名test为qizhonglin

$usermod -L qizhonglin 锁定帐户

$usermod -U qizhonglin 解锁帐户

$passwd qizhonglin 改帐户密码

### 用户组

组权限允许多个用户共享一组共用的权限来访问系统上的对象，比如文件、目录或设备之类的

有些Linux发行版会创建一个组，把所有用户都当做这个组的成员。有些发行版会为每个用户创建一个单独的组。如Ubuntu会为每个用户创建一个单独的与用户帐户同名的组

$groupadd shared 创建组shared

$usermod -G shared qizhonglin 将用户qizhonglin添加到组shared

$groupmod -n sharing shared 修改组名sharing <- shared

文件权限符：

详细见《linux命令行与shell脚本编程大全》p133

Ubuntu系统为例

$umask 显示或设置用户创建文件和目录的默认权限

0002 只是掩码，屏蔽不想授予该安全级别的权限

文件全权限值是666, 通过掩码，得创建新文件的默认权限是664即rw-rw-r--

目录全权限值是777,通过掩码，得创建新目录的默认权限是775即rwxrwxr-x

$chmod options mode file 改变文件或目录的安全性设置

$chmod [ugoa…][+-=][rwxXstugo…] file

$chmod u+x file1 给文件file1添加执行权限

### 共享文件做法

$mkdir shared-dir

$chgrp shared shared-dir 将目录shared-dir的默认属组改为共享组

$chmod g+s shared-dir

$umask 002 所有组成员都需把他们的umask值设置成文件对属组成员可写。

$cd shared-dir 进入共享文件夹工作

$touch shared-file

## 探索Linux文件系统

安装时大多数Linux发行版会为系统提供一个默认的文件系统

每种文件系统都在存储设备上实现了虚拟目录结构

使用虚拟目录来操作硬件设备，在物理设备上按定长的块来存储数据

Ext文件系统

采用索引节点的系统来存放虚拟目录中所存储文件的信息。索引节点系统在每个物理设备中创建一个单独的表（称为索引节点表）来存储这些文件的信息。文件系统通过索引节点号而不是文件全名及路径来标识文件

Ext2文件系统

改进：

扩展了索引节点表的格式来保存系统上每个文件的更多信息

通过按组分配磁盘块来减轻碎片化

缺点：文件系统每次存储或更新文件，它都要用新信息来更新索引节点表，如果在存储文件和更新索引节点表的过程中，由于系统崩溃或断电，会造成不同步

日志文件系统

改进：

先将文件的更改写入到临时文件（称作日志，journal）中，然后在数据成功写到存储设备和索引节点表之后，再删除对应的日志条目。

如果系统在数据被写入到存储设备之前崩溃了或断电了，日志文件系统下次会读取日志文件并处理上次留下的未写入的数据。

有三种方法：（起不同等级的保护）

数据模式

索引节点和文件都会写入日志，性能差，因为写到存储设备上的数据必须写两次，第一次写到日志，第二次写到真正的存储设备

排序模式

回写模式

Ext3文件系统

用排序模式的日志功能————只将索引节点信息写入日志文件，直到数据块都被成功写入存储设备才删除

**Ext4文件系统**

在2008年时被Linux内核官方支持，现在已是大多数流行的Linux发行版采用的默认文件系统，如Fedora and Ubuntu

改进：支持数据压缩和加密

1. 分区$fdisk /dev/sdc 假定分到的第一个区sdc1
2. 格式化（创建文件系统）$mkfs.ext4 /dev/sdc1

挂载到虚拟目录下的某个挂载点

1. $mkdir /mnt/testing
2. $mount –t ext4 /dev/sdb1 /mnt/testing

这种挂载文件系统的方法只会临时挂载该文件系统。当重启Linux系统时，文件系统不会自动挂载。要强制Linux在启动时自动挂载这个新文件系统，可以将文件系统添加到/etc/fstab文件中。

逻辑卷管理器 <- 卷组 <- 物理卷1，物理卷2，… <- 分区1, … <- 硬盘1…

见《Linux命令行与shell脚本编程大全》p151

# 压缩及解压工具

bzip2工具

bzip2: 压缩文件

bzcat: 查看压缩文件内容

bunzip2: 解压文件

gzip工具

gzip: 压缩文件

gzcat: 查看压缩文件内容

gunzip: 解压文件

zip工具

zip: 压缩文件

zipcloak: 加密压缩文件

zipnote: 从zip文件提取批注

zipsplit: 将大zip文件分割成多个固定小文件

unzip: 解压文件

zip工具的强大之处：能够将整个目录下的文件都压缩进单个文件。这让它成为归档整个目录结构的理想工具

## tar命令

$tar -cvf test.tar test/ test2/ 将test and test2目录内容归档到test.tar文件中

$tar -xvf test.tar 在当前目录解压test.tar内容

下载了开源软件之后，经常看到文件名以.tgz结尾。这些是gzip压缩过的tar文件，可以用命令tar -zxvf filename.tgz来解压

# Linux环境变量

## 全局环境变量

系统环境变量一律用大写字母，以区别普通用户的环境变量

$printenv 查看所有

$echo $HOME 查看单个

$test=testing && export test 设置全局环境变量=局部+export

$unset test 删除变量

常用默认的shell环境变量：

$ echo $PATH 冒号分隔的shell查找命令的目录列表

PATH定义了命令行输入命令的搜索路径

$ PATH=$PATH:/home/user/test

$ PATH=$PATH:. 将当前目录加入PATH环境变量里

## 局部环境变量

局部环境变量只能在定义它们的进程中可见

$set 显示某个特定进程设置的所有环境变量。包括全局环境变量

$test=testing 设置局部环境变量

注意等号两边没有空格，若放了空格，如$test = testing,那么bash shell就会把值当成一个单独的命令

$echo $test

## 系统环境变量查找流程：

若登录时当做默认登录shell

启动文件次序：

/etc/profile bash shell的主启动文件

/etc/profile.d/\*.sh 用户登录时应用专属的启动文件

$HOME/.profile

$HOME/.bashrc

作为非登录shell的交互式shell

$HOME/.bashrc

# 数据库

## install MySQL

//安装过程会要求输入root用户的密码，所以有user: root, password: q

$sudo apt-get install mysql-server

//客户端程序访问MySQL服务器

$mysql -u root -p //记得输入root密码

mysql命令

mysql>help 显示帮助信息

mysql>status 从MySQL服务器提取状态信息

mysql>exit 退出

//操作数据库

mysql> show databases ;

mysql> use infomation\_schema;

mysql> show tables ;

//root授权给用户test\_user,密码为test\_password使用数据库test\_db，有所有权限

mysql> GRANT ALL on test\_db.\* to test\_user IDENTIFIED by 'test\_password';

mysql> exit;

mysql> CREATE DATABASE test;

mysql> USE test;

mysql> CREATE TABLE employees (

empid int not null,

lastname varchar(30),

firstname varchar(30),

salary float,

primary key (empid) );

mysql> INSERT INTO employees VALUES (1, ‘ Blum’, ‘Rich’, 25000.00);

mysql> DELETE FROM employees WHERE empid = 2;

mysql> SELECT \* FROM employees WHERE salary>40000;

在脚本中使用数据库

MYSQL=` which mysql` 查找mysql客户端程序

$MYSQL test -u test -ptest <<EOF

SHOW TABLES;

SELECT \* FROM employees WHERE salary > 4000;

EOF

## install PostgreSQL

$sudo apt-get install postgresql

//启动postgresql服务器

$sudo /etc/init.d/postgresql start

$sudo /etc/init.d/postgresql stop

//访问服务器

$sudo -u postgres psql

postgres=# \l 显示所有数据库q

postgres=# CREATE DATABASE test; 创建数据库

postgres=# \c test 使用(连接)数据库

创建数据表employees

test=# CREATE ROLE rich login; 创建登录角色

test=# GRANT SELECT, INSERT, DELETE, UPDATE ON public.employees TO rich;

使用web

cURL

允许使用特定URL来自动从命令行传送文件。现在它支持FTP, FTPS, HTTP, HTTPS, SCP, SFTP, TFTP, telnet, DICT, LDAP, LDAPS, FILE协议作为URL中指定的协议

//install cURL

$ sudo apt-get install curl

$curl http://www.baidu.com

默认情况下，cURL会将完整的Web页面HTML代码返回到STDOUT上；

curl -s -o /home/qzlin/curl-7.18.0.tar.gz http://curl.haxx.se/download/curl-7.18.0.tar.gz

-s选项让cURL工作在安静模式下，不会发送任何数据到STDOUT

-o选项会将输出重定向到一个文件名

# python使用

## 包管理器pip

pip类似RedHat的yum, Ubuntu的apt-get, Ruby的gem

PyPI is a repository for open-source third-party Python packages. It's similar to RubyGems in the Ruby world, PHP's Packagist, CPAN for Perl, and NPM for Node.js.

$ sudo easy\_install pip

Python has primitive package manager called easy\_install, which is installed automatically when you install Python itself

$pip --help

$pip list 列出已安装包

$pip install 安装包

$pip install --upgrade 升级包

$pip uninstall 卸载包

$pip show 显示包详细信息

$pip search 搜索包，类似yum里的search

## virtual environment

virtualenv is a tool for creating isolated Python environments containing their own copy of python, pip, and their own place to keep libraries installed from PyPI.

simply a directory that contains a complete copy of everything needed to run a Python program, including a copy of the python binary itself, a copy of the entire Python standard library, a copy of the pip installer, and (crucially) a copy of the site-packages directory mentioned above.

$sudo pip install virtualenv

若有代理：

$export https\_proxy=http://web-proxy.mydomain.com

$sudo -E pip install virtualenv

Usually pip and virtualenv are the only two packages you ever need to install globally.In fact, virtualenv comes with a copy of pip which gets copied into every new environment you create, so virtualenv is really all you need.

$ cd ~/code/myproject/

$ virtualenv env

Start by changing directory into the root of your project directory, and then use the virtualenv command-line tool to create a new environment

It's a common convention to call this directory env, and to put it inside your project directory (so, say you keep your code at ~/code/projectname/, the environment will be at ~/code/projectname/env/ - each project gets its own env).

Note: if you're using a version control system like git, you shouldn't commit the env directory. Add it to your .gitignore file (or similar).

projectname

env

bin

include

lib

helloworld.py

helloworld.py

import requestsprint requests.get('http://www.baidu.com')

$env/bin/pip install requests

$env/bin/python helloworld.py

## Python脚本编程和运行方式

### Interactive Mode Programming

$ python

>>> print "Hello, Python!"

>>> exit()

### Script Mode Programming

test.py

print "Hello, Python"

$ python test.py

### like shell

#!/usr/bin/python

print "Hello, Python!"

$ chmod +x test.py # This is to make file executable

$./test.py

## 语法

### Standard Data Types

Numbers

String

List

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

Tuple

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

The main differences between lists and tuples are: Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated

Dictionary

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

### Types of Operator

Arithmetic Operators： +， -， \*， 、， %， \*\*， //

Comparison (Relational) Operators： ==, !=, <>, >, <, >=, <=

Assignment Operators： =, +=, -=, \*=, /=, %=, \*\*=, //=

Logical Operators (Bitwise Operators)： &, |, ^, ~, <<, >>

Membership Operators： in, not in

Identity Operators： is, is not

### 函数

#### 函数定义

def functionname( parameters ):

"function\_docstring"

function\_suite

return [expression]

sum = lambda arg1, arg2: arg1 + arg2;

#### 函数传参

Pass by reference vs value

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example −

def changeme( mylist ):

mylist.append([1,2,3,4]);

print "Values inside the function: ", mylist

return

mylist = [10,20,30];

changeme( mylist );

print "Values outside the function: ", mylist

Values inside the function: [10, 20, 30, [1, 2, 3, 4]]

Values outside the function: [10, 20, 30, [1, 2, 3, 4]]

argument is being passed by reference and the reference is being overwritten inside the called function.

def changeme( mylist ):

mylist = [1,2,3,4]; # This would assig new reference in mylist

print "Values inside the function: ", mylist

return

mylist = [10,20,30];

changeme( mylist );

print "Values outside the function: ", mylist

Values inside the function: [1, 2, 3, 4]

Values outside the function: [10, 20, 30]

#### 函数参数

Required arguments

Keyword arguments：

def printinfo( name, age ):

...

printinfo( age=50, name="miki" )

Default arguments

def printinfo( name, age = 35 ):

...

printinfo( name="miki" )

Variable-length arguments

def printinfo( arg1, \*vartuple ):

...

printinfo( 70, 60, 50 )

结果：arg1=70, vartuple=(60, 50)

An asterisk (\*) is placed before the variable name that holds the values of all nonkeyword variable arguments.

### 模块

#### 模块定义

A module can define functions, classes and variables.

support.py

def print\_func( par ):

print "Hello : ", par

return

#### 模块调用

# Import module support

import support

# Now you can call defined function that module as follows

support.print\_func("Zara")

The from...import Statement

The from...import \* Statement:

#### 模块搜索

1. The current directory.
2. If the module isn't found, Python then searches each directory in the shell variable PYTHONPATH.
3. If all else fails, Python checks the default path. On UNIX, this default path is normally /usr/local/lib/python/.

### Packages in Python

Phone/Isdn.py file having function Isdn()

Phone/G3.py file having function G3()

Phone/\_\_init\_\_.py

from Pots import Pots

from Isdn import Isdn

from G3 import G3

To make all of your functions available when you've imported Phone, you need to put explicit import statements in \_\_init\_\_.py as above

# Now import your Phone Package.

import Phone

Phone.Pots()

Phone.Isdn()

Phone.G3()

输入输出

STDOUT

#!/usr/bin/python

print """Hello, Python!

I am qzlin,

please welcome me"""

$./helloworld.py | grep py

STDIN

#!/usr/bin/python

import sys

for line in sys.stdin:

print line

$./helloworld.py | ./stream-read.py

文件迭代器

当文件对象用于for loop时，Python会自动调用readline方法

#文件内容如果可以很小，一次性读取所有内容:

f = open(filename)

for line in f.readlines():

process(line)

f.close()

#文件内容很大，流式处理,一行一行读取

import fileinput

for line in fileinput.input(filename):

process(line)

json输出 | python -m json.tool

open/close port

ufw: uncomplicated firewall

Nmap: Network Mapper(网络映射器)

<https://nmap.org/man/zh/>

设计目标是快速地扫描大型网络，使用原始IP报文来发现网络上有哪些主机，那些主机提供什么服务(应用程序名和版本)，那些服务运行在什么操作系统(包括版本信息)，它们使用什么类型的报文过滤器/防火墙,以及一堆其它功能。

通常用于安全审核,也可做一些日常的工作，比如查看整个网络的信息，管理服务升级计划，以及监视主机和服务的运行。

“所感兴趣的端口表格”是其中的关键。那张表列出端口号，协议，服务名称和状态。状态可能是 open(开放的)，filtered(被过滤的)， closed(关闭的)，或者unfiltered(未被过滤的)。

Open(开放的)意味着目标机器上的应用程序正在该端口监听连接/报文。

filtered(被过滤的) 意味着防火墙，过滤器或者其它网络障碍阻止了该端口被访问，Nmap无法得知 它是 open(开放的) 还是 closed(关闭的)。

$sudo apt-get install –y nmap

SSH: secure shell 安全外壳协议

传统的网络服务程序，如：ftp、pop和telnet在本质上都是不安全的，因为它们在网络上用明文传送口令和数据，很容易受到“中间人”（man-in-the-middle）这种方式的攻击。所谓“中间人”的攻击方式， 就是“中间人”冒充真正的服务器接收你传给服务器的数据，然后再冒充你把数据传给真正的服务器.

通过使用SSH，你可以把所有传输的数据进行加密.

结构：SSH是由客户端和服务端的软件组成的,服务端是一个守护进程(daemon)，他在后台运行并响应来自客户端的连接请求。服务端一般是sshd进程，提供了对远程连接的处理，一般包括公共密钥认证、密钥交换、对称密钥加密和非安全连接。客户端包含ssh程序以及像scp（远程拷贝）、slogin（远程登陆）、sftp（安全文件传输）等其他的应用程序。

工作机制：本地的客户端发送一个连接请求到远程的服务端，服务端检查申请的包和IP地址再发送密钥给SSH的客户端，本地再将密钥发回给服务端，自此连接建立。

Linux一般会打开ssh网络服务(端口默认是22)，window通过Putty.exe访问

MD5

MD5即Message-Digest Algorithm 5（信息-摘要算法5）

将数据（如汉字）运算为另一固定长度值, 用于确保信息传输完整一致, 除了MD5以外，其中比较有名的还有sha-1、RIPEMD以及Haval等。

特点

1、压缩性：任意长度的数据，算出的MD5值长度都是固定的。

2、容易计算：从原数据计算出MD5值很容易。

3、抗修改性：对原数据进行任何改动，哪怕只修改1个字节，所得到的MD5值都有很大区别。

4、强抗碰撞：已知原数据和其MD5值，想找到一个具有相同MD5值的数据（即伪造数据）是非常困难的。

MD5应用

一致性验证:

在Unix下有很多软件在下载的时候都有一个文件名相同，文件扩展名为.md5的文件，在这个文件中通常只有一行文本，大致结构如：MD5 (tanajiya.tar.gz) = 38b8c2c1093dd0fec383a9d9ac940515

这就是tanajiya.tar.gz文件的数字签名。MD5将整个文件当作一个大文本信息，通过其不可逆的字符串变换算法，产生了这个唯一的MD5信息摘要。

利用MD5算法来进行文件校验的方案被大量应用到软件下载站、论坛数据库、系统文件安全等方面。

数字签名

MD5的典型应用是对一段Message(字节串)产生fingerprint(指纹），以防止被“篡改”

安全访问认证

MD5还广泛用于操作系统的登陆认证上，如Unix、各类BSD系统登录密码、数字签名等诸多方面。如在Unix系统中用户的密码是以MD5（或其它类似的算法）经Hash运算后存储在文件系统中。当用户登录的时候，系统把用户输入的密码进行MD5 Hash运算，然后再去和保存在文件系统中的MD5值进行比较，进而确定输入的密码是否正确。通过这样的步骤，系统在并不知道用户密码的明码的情况下就可以确定用户登录系统的合法性。这可以避免用户的密码被具有系统管理员权限的用户知道。MD5将任意长度的“字节串”映射为一个128bit的大整数，并且是通过该128bit反推原始字符串是困难的

这种加密技术被广泛的应用于Unix系统中，这也是为什么Unix系统比一般操作系统更为坚固一个重要原因。

Ubuntu自带md5

$md5sum -help

$md5sum \*.txt

常用开发软件安装

// config proxy

$ gedit ~/.bashrc

在打开的文件里添加

export http\_proxy=http://165.225.96.34:10015/

export https\_proxy=https://165.225.96.34:10015/

$ sudo gedit /etc/apt/apt.conf

Acquire::http::proxy "http://165.225.96.34:10015/";

Acquire::https::proxy "https://165.225.96.34:10015/";

$ sudo gedit /etc/environment

http\_proxy=http://165.225.96.34:10015/

https\_proxy=https://165.225.96.34:10015/

//有时候无法apt-getq

$sudo gedit /etc/resolvconf/resolv.conf.d/base

nameserver 8.8.8.8

nameserver 8.8.4.4

// install chrome

$ wget -q -O - https://dl-ssl.google.com/linux/linux\_signing\_key.pub | sudo apt-key add -

$ sudo sh -c 'echo "deb http://dl.google.com/linux/chrome/deb/ stable main" >> /etc/apt/sources.list.d/google.list'

$ sudo apt-get update

$ sudo apt-get install google-chrome-stable

// install notepad++

$ sudo -E add-apt-repository ppa:notepadqq-team/notepadqq

$ sudo apt-get update

$ sudo apt-get install notepadqq

//install jdk8 on ubuntu

open terminal via Ctrl+Alt+T (or search terminal)

$ sudo add-apt-repository ppa:webupd8team/java

(如果无法连接，用sudo -E add-apt-repository ppa:webupd8team/java)

$ sudo apt-get update

$ sudo apt-get install oracle-java8-installer

// verify installed java version

$ java -version

// configure java environment

$ sudo apt-get install oracle-java8-set-default

//install Intellij

Download ideaIU-2016.1.3.tar.gz

$cd /opt

$sudo tar –xvf ideaIU-2016.1.3.tar.gz

$sudo /opt/idea-IU-145.1617.8/bin/idea.sh //启动

// install spring tool suite

$ cd /opt

$ wget http://....tar.gz

$ sudo tar -xvf spring-tool-suite...tar.gz

$ ln -s /opt/sts-bundle/sts-...RELEASE/STS /usr/local/bin/sts

$ sts

if could not connect to network, please only set http and https, not set socks

// install gradle

download gradle

$sudo unzip <path to gradle-<>-all.zip>/gradle-<>-all.zip -f /opt/gradle

$ gedit ~/.bashrc

GRADLE\_HOME=<path to gradle bin file> (e.g. /opt/gradle/gradle-1.5/bin )

export GRADLE\_HOME

PATH=$PATH:$GRADLE\_HOME

export PATH

$source ~/.bashrc

$gradle -version

$gradle build –x test //exclude any task

$gradle bootRun

// install docker

$ wget -qO- https://get.docker.com/ | sh

$ sudo reboot

$ sudo gedit /etc/default/docker

export HTTP\_PROXY="http://161.92.51.225:8080/"

$ sudo service docker restart

// check port used and kill

$sudo netstat - tupln | grep 27017

$sudo kill -9 914 (914 is the pid to use port 27017)

// install git

$sudo apt-get install git-core

$git --version

// install hadoop

添加JAVA\_HOME (查看java安装路径$ update-alternatives --config java)

安装ssh

$ sudo apt-get install ssh

$ sudo apt-get install rsync

下载并解压hadoop-x.y.z.tar.gz,然后放入到opt目录中

$sudo mv Documents/hadoop/tools/hadoop-2.7.1 /opt/

设置环境变量

$gedit ~/.bashrc

HADOOP\_HOME=/opt/hadoop-2.7.1

PATH=$PATH:$HADOOP\_HOME/bin

export HADOOP\_HOME

export PATH

$source ~/.bashrc

检查hadoop version

$ hadoop version

java project

add \*.jar from $HADOOP\_HOME/bin/mr-jobhistory

mapper.java

reducer.java

JobTask.java

build

project$ export HADOOP\_CLASSPATH=bin

project$ hadoop project input/ncdc/sample.txt output