Contents

[Contents 1](#_Toc168413358)

[Linux 常用快捷键 2](#_Toc168413359)

[Linux = 内核 + GNU工具 + 图形化桌面环境 + 应用软件 4](#_Toc168413360)

[文件系统 6](#_Toc168413361)

[Linux文件系统和常见Linux目录 6](#_Toc168413362)

[文件和内容操作: ls, mkdir, rm, mv, cp, ln, find, cat, diff, split, grep, tar 7](#_Toc168413363)

[帐号和文件权限: 用户管理，文件权限，共享文件 10](#_Toc168413364)

[磁盘管理: mount, /etc/fstab, df, du 15](#_Toc168413365)

[进程 16](#_Toc168413366)

[ps, top, kill and renice 16](#_Toc168413367)

[进程通信，后台模式，定时定期自动执行脚本 17](#_Toc168413368)

[网络 20](#_Toc168413369)

[ping, nslookup, whois, arp/rarp, hostname, traceroute, netstat, nmap, tcmdump, wget/curl 20](#_Toc168413370)

[远程登录Linux系统 24](#_Toc168413371)

[linux and win 互传文件，文件共享 26](#_Toc168413372)

[虚拟机与主机通信 28](#_Toc168413373)

[GPU(显卡) 29](#_Toc168413374)

[X11 : X window system的第11个版本，Linux的工具包 30](#_Toc168413375)

[软件 34](#_Toc168413376)

[软件程序管理: dpkg (apt-get), rpm (yum) 34](#_Toc168413377)

[常用软件：md5, 39](#_Toc168413378)

[开发软件: git, intellij, jdk, spring, gradle, docker, .NET 44](#_Toc168413379)

[linux shell脚本 47](#_Toc168413380)

[开发流程，编辑器vim和运行 47](#_Toc168413381)

[语法：变量，条件判断，循环，可变数组，函数，函数库 50](#_Toc168413382)

[常见命令: su/sudo, >, >>, <, >/dev/null, tee, grep, sed and awk 57](#_Toc168413383)

[数据输入(命令行,用户输入,文件读取) 和数据输出 (文件,临时文件,日志文件) 61](#_Toc168413384)

[系统编程 68](#_Toc168413385)

[进程间通信 68](#_Toc168413386)

[Appendix： 70](#_Toc168413387)

[Development Environment for Deep Learning 70](#_Toc168413388)

[安装ubuntu 71](#_Toc168413389)

参考书籍：

《Linux宝典》

《Linux命令行大全》

《Linux命令行与shell脚本编程大全》

# Linux 常用快捷键

Ctrl + Alt + L 锁屏 // 相当于 Win + L

Super + Tab 菜单栏应用程序切换

Alt + Tab 已打开的窗口切换

[http://wiki.ubuntu.org.cn/Ubuntu%E6%A1%8C%E9%9D%A2%E5%85%A5%E9%97%A8%E6%8C%87%E5%8D%97](http://wiki.ubuntu.org.cn/Ubuntu桌面入门指南)

$ poweroff #关机

$ reboot #重启系统

$ shutdown -r +10 #10分钟后重启系统， -r表示重启系统

$ shutdown -h now #马上关闭系统，-h表示关闭系统

桌面 -> 控制台 Ctrl + Alt + F1 (F2, …F6)

控制台 –> 桌面 Ctrl + Alt + F7

Ctrl+Alt+T #打开终端

$ echo $OS<Tab> #Tab补齐命令

$ ls Do<Tab><Tab> #双Tab对文本扩展的可能方式进行检查

$ exit #退出shell

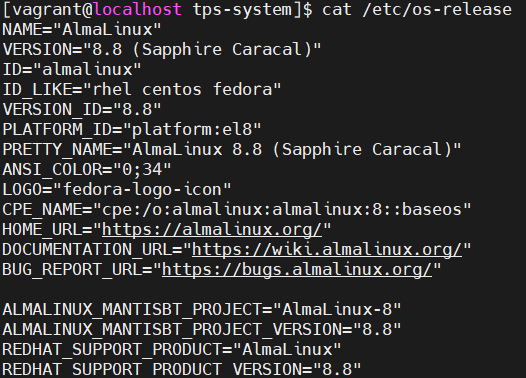
// 查看操作系统信息

GUI -> Settings -> About

A screenshot of a computer

Description automatically generated

$ cat /etc/os-release



// list PCI devices

Most of the graphics cards, in general, are plugged into PCI slots of the motherboard.

$ lspci -v

A screenshot of a computer program

Description automatically generated

$ uname -a # 查系统

$ cat /proc/cpuinfo # 查看cpu信息

$ cat /proc/cpuinfo | grep 'processor' | wc -l 逻辑CPU个数

$ cat /proc/cpuinfo | grep 'physical id' | sort | uniq | wc -l 物理CPU个数

$ cat /proc/cpuinfo | grep 'cpu cores' | wc -l 每个物理CPU中Core的个数

# 查看内存

$ cat /proc/meminfo

$ free -h # 对/proc/meminfo收集到的信息的概述

$ top # 根据内存的使用和大小排序

$ df -h # 查看磁盘

$ du -sh \* # 查看文件夹大小

$ nvidia-smi watch 查看CUDA version, driver version …

$ watch –n 10 nvidia-smi 每10秒显示GPU的情况

$ gnome-system-monitor 类似windows任务管理器，ubuntu系统监视器

$ alias -p 查看已有的别名列表

#查看软件安装位置

$ sudo find / -name libopenblas-dev

$ dpkg -L libopenblas-dev //若是用apt-get install安装，可以用dpkg查找

$ whereis libopenblas-dev

$ locate libopenblas-dev

$ which libopenblas-dev

$ type libopenblas-dev

How to adjust menu bar ?

$ gsettings set com.canonical.Unity.Launcher launcher-position Bottom

# Linux = 内核 + GNU工具 + 图形化桌面环境 + 应用软件

* Linux内核主要功能：

1. 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)**。

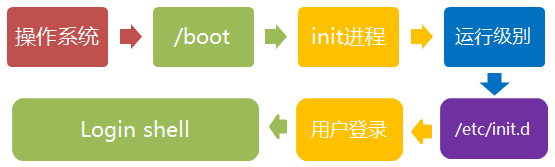
内存存储单元 = 页面+页面+…

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。

1. 软件程序管理

Linux系统启动过程：



* 内核引导：

计算机打开电源后，BIOS开机自检，读入/boot目录下的内核文件

* 运行init进程：

**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。**

* 运行级别：

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。

* 系统初始化/etc/init.d

配置基本系统环境，启动各种守护进程

* 建立终端

打开6个终端(tty1 – 6)，以便用户登录系统。

1. 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**。

1. 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　（简称X)

几乎每一个提供了桌面界面的主要Linux发行版都是基于X Windows系统。X Windows系统提供了一个框架，根据该框架，可以构建不同类型的桌面环境

客户端/服务器模型工作方式

X服务器运行在大型计算机上，含应用程序, 磁盘存储和处理能力

X客户端可能运行在大型计算机上（即本地系统上），更一般是通过网络在瘦客户端显示和管理

**Linux系统可能只提供一个纯文本的命令行界面。可以在以后添加一个桌面或多个桌面环境**，在安装完桌面之后，当计算机启动时可以选择是否启用该桌面

KDE桌面：类windows桌面

GNOME桌面：类mac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

**CentOS**: 基于Red Hat，社区赞助的版本

Fedora: 基于Red Hat，关注最新技术，较少关注稳定性的的版本

RPM包管理

Red Hat创建了RPM封装格式，通过使用该格式，一个软件包不仅可以包含共享的文件，还可以包含包版本的相关信息（比如谁是创建者，哪些文件是文档或是配置文件以及何时被创建）。通过安装以RPM格式封装的软件，可以在本地的RPM数据库中存储每个软件包相关的上述信息，从而便于查找安装了什么软件以及更新或者删除软件

使用GNOME界面

Debian: 在Linux专家和商用Linux产品中流行的发行版

**Ubuntu**: 免费的学校和家用Linux发行版

使用deb包装技术和工具来管理系统中的所有软件包

使用Unity桌面

# 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。

## Linux文件系统和常见Linux目录

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。缺点：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。

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

$fdisk /dev/sdc # 分区，假定分到的第一个区sdc1

$mkfs.ext4 /dev/sdc1 # 格式化（创建文件系统）

挂载到虚拟目录下的某个挂载点

$mkdir /mnt/testing

$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。**

* 常见Linux目录名称

/etc 系统配置文件目录

/home 用户主目录，每个用户自己的目录，如/home/qzlin

/bin binary，存放GNU用户级的工具如ls

/lib library，动态连接共享库 (类似于 Windows DLL)

/opt optional，存放可选的软件包如postgre数据库, 默认是空的

/usr unix shared resources(共享资源)，用户安装软件的目录，

类似于 windows 下的 program files 目录

/usr/src 内核源代码默认的放置目录。

/usr/bin 系统用户使用的应用程序

/usr/sbin 超级用户使用的比较高级的管理程序和系统守护程序。

/var variable，存放不断扩充的东西，比如日志文件

/tmp temporary，存放临时文件

/run 临时文件系统，存储系统启动以来的信息。

当系统重启时，这个目录下的文件应该被删掉或清除

/proc process，伪文件系统（也即虚拟文件系统），记录当前内核运行状态

/proc是虚拟目录，仅仅是系统内存的映射，访问/proc获取系统信息

子目录以系统中的PID命名，子目录中的文件代表的是进程的各种状态，特别/proc/cpuinfo文件包含大量的内核和硬件系统信息

/media linux 自动识别设备(如U盘光驱)，然后挂载到这个目录下

/mnt 挂载目录，存放设备挂载点，让用户临时挂载别的文件系统

如：将光驱挂载在 /mnt/ 上，然后进入该目录就可以查看光驱里的内容了

/dev 设备目录存放Linux的外部设备

在这里创建设备节点，表示I/O接口，

**在 Linux 中访问设备的方式和访问文件的方式是相同的**

/root 系统管理员(超级权限者)的用户主目录

/sbin Superuser Binaries，存放的是系统管理员使用的系统管理程序。

/src 存放一些服务启动之后需要提取的数据

/sys sysfs 文件系统，内核设备树的一个直观反映

/boot linux内核文件

## 文件和内容操作: ls, mkdir, rm, mv, cp, ln, find, cat, diff, split, grep, tar

$ ls -Fal // list files

-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

$ cd <directory> // change directory

$ pwd // print work directory

**$ mkdir -p** src/main/scala // make directory

**$ rm -rf** directoryName // remove directory or file

$ **mv** test1 test2 // move files or remove file/directory name

$ **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 //copy file to file

$ cp file1 file2 dir //copy file to dir

**$ cp -R** srcDir dstDir //copy dir to dir recursively

**软链接：符号链接，**类似windows中的快捷方式

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

**ln -s** 源文件 软链接 //创建软链接

//创建软链接，将可执行文件放入命令行搜索路径中

ln -s /home/q/Downloads/node-v8.1.0-linux-x64/bin/node /usr/local/bin/node

ln -s /home/q/Downloads/node-v8.1.0-linux-x64/bin/npm /usr/local/bin/npm

$node -v && npm -v

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

**$ touch** build.gradle // create file

查找文件

$ find dir file //在目录下查找文件

$ sudo find / -name replace\_fess\_certificate.sh

$ locate //在系统创建的文件索引中查找文件

文件内容操作: 查看，排序，搜索，统计，大文件split and cat

// 查看文件内容

$ **cat** test1 显示整个文件内容

$ nl test1 显示整个文件内容，且显示行号

**$ more** test1 按页显示内容

$ **less** test1 按页显示内容内容，可以前后翻动和搜索

$ **head** log 显示文件开头

**$ 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。

$ **diff** file1 file2 查看两个文件的不同

// 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

// 文件切割和组装

split [选项] [要切割的文件] [输出文件名前缀］

cat file1 file2 ... > allfile

举例：

将ResNet-101-model.caffemodel 分解成50M大小的文件集

$ **split** -b 50m ResNet-101-model.caffemodel ResNet-101-model-

$ cat ResNet-101-model-\* > ResNet-101-model.caffemodel

**// 压缩及解压工具: tar, bzip2,gzip,zip**

// tar命令 (推荐）

$**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

-c表示创建

-f表示文件

-v表示显示详细信息

-x表示解压

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

// 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

## 帐号和文件权限: 用户管理，文件权限，共享文件

// 用户管理

$ sudo passwd [root] #root用户设置密码

注意：root没有默认密码，需要手动设定。以安装ubuntu时输入的用户名登陆，该用户在admin组中，有权限给root设定密码

$ su 切换至root #用户帐号切换至root

$ su – qzlin #切换至普通帐户

$ sudo passwd #修改密码

$ 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 [-d] [-g] 用户名**

-d 目录 指定用户主目录，如果此目录不存在，则同时使用-m选项，可以创建主目录。

-g 用户组 指定用户所属的用户组。

-G 用户组，用户组 指定用户所属的附加组。

-s Shell文件 指定用户的登录Shell。

刚添加的账号是被锁定的，无法使用。

$ useradd -s /bin/sh -g group –G adm,root test

新建用户test，该用户登录Shell是/bin/sh，

属于group用户组，同时又属于adm和root用户组

$ useradd -m test #创建新用户test, 从而有用户文件夹/home/test

在/etc/passwd增加新用户记录，同时更新其他系统文件如/etc/shadow, /etc/group等

$ **userdel** -r test #删除用户

默认情况下，userdel命令只删除/etc/passwd文件中的用户信息

-r 删除系统中属于该账户的任何文件如用户主目录：/home/test

**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

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

$ passwd qizhonglin #改帐户密码

用户账号刚创建时没有口令，但是被系统锁定，无法使用，必须为其指定口令后才可以使用。

chage修改密码到期时间，更改密码时间等

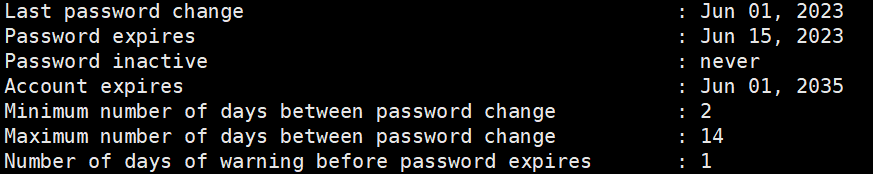
$ chage -m 2 qzlin # 要求用户qzlin密码更改间隔最小天数为2天

$ chage -M 14 qzlin # 要求用户qzlin密码更改间隔最大天数为2天

$ chage -W 7 qzlin # 密码到期前一天提示用户qzlin密码要过期

$ chage -E 2036-6-1 qzlin # 设置密码到期时间

$ chage -l qlin # 显示密码相关信息



// 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

新组的组标识号是在当前已有的最大组标识号的基础上加1。

# groupadd -g 1100 pinnacle

-g GID 指定新用户组的组标识号（GID）。

# **groupdel** pinnacle # 创建用户组pinnacle

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

// 与用户帐号相关的系统文件：/etc/passwd, /etc/shadow, /etc/group

**# cat /etc/passwd** // 用户帐户列表以及每个用户的基本配置信息

root:x:0:0:Superuser:/:

daemon:x:1:1:System daemons:/etc:

bin:x:2:2:Owner of system commands:/bin:

sys:x:3:3:Owner of system files:/usr/sys:

/etc/passwd中一行记录对应着一个用户，每行记录又被冒号(:)分隔为7个字段

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

用户密码：现在Linux 系统使用shadow技术，把用户口令加密后存放到/etc/shadow，而在/etc/passwd口令字段只存放特殊字符x

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。即**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**(也称为系统帐户，是系统上运行的各种服务进程访问资源用的特殊帐户**) 。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。**

系统用户 ：

bin 拥有可执行的用户命令文件

sys 拥有系统文件

adm 拥有帐户文件

uucp UUCP使用

lp lp或lpd子系统使用

nobody NFS使用

# cat /etc/shadow # 由pwconv命令根据/etc/passwd中的数据自动产生

登录名:加密口令:最后一次修改时间:最小时间间隔:最大时间间隔:警告时间:不活动时间:失效时间:标志

# cat /etc/group # 存放用户组的所有信息

组名:口令:组标识号:组内用户列表

//添加批量用户

user.txt // 按照/etc/passwd密码文件的格式书写, 密码栏为空

user001::600:100:user:/home/user001:/bin/bash

user002::601:100:user:/home/user002:/bin/bash

# /usr/sbin/newusers < user.txt // 创建用户

$ cat /etc/passwd //检查是否出现user001等用户，以及/home/user001是否创建

# /usr/sbin/pwunconv // 取消shadow password功能

passwd.txt

user001:123456

user002:123456

# /usr/sbin/chpasswd < passwd.txt // 创建用户密码

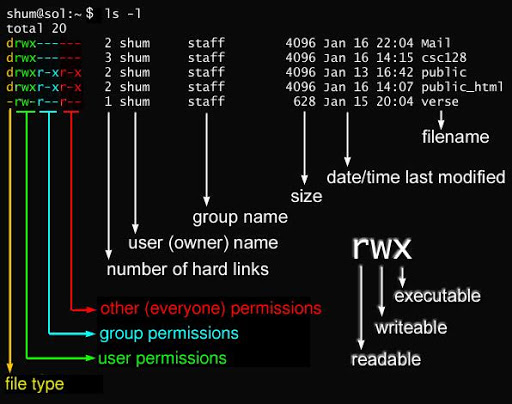
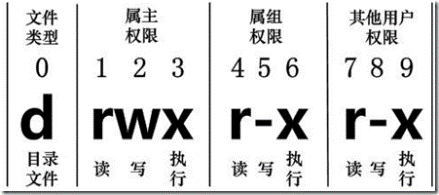
chpasswd 会将经过 /usr/bin/passwd 命令编码过的密码写入 /etc/passwd 的密码栏。

# /usr/sbin/pwconv // 密码编码为 shadow password，并写入 /etc/shadow

文件权限：

不同的用户处于不同的地位，拥有不同的权限。

Linux 系统对不同的用户访问同一文件（包括目录文件）的权限做了不同的规定。

第一个字符d/-/l代表file type文件类型是目录/文件/链接文件等

接下来字符，以三个rwx为一组。代表read/write/execute，若没有权限就会出现减号-。

每个文件的属性由左边第一部分的 10 个字符来确定, 第 0 位确定文件类型，第 1-3 位确定属主（**文件所有者**）拥有该文件的权限。第4-6位确定属组（所有者的**同组用户**）拥有该文件的权限，第7-9位确定**其他用户**拥有该文件的权限。

**chgrp** [-R] 属组名 文件名 // 更改文件属组

-R 该目录下的所有文件的属组都会更改

**// chown (change owner) 修改文件或目录所属用户与组。**

chown [-R] 所有者 文件名 // 更改文件所有者(owner)

chown [-R] 所有者: 属组名 文件名 // 更改文件所有者和文件所属组。

// chmod (change mode) 修改文件或目录所属用户的权限。

Linux 文件的基本权限就有九个，分别是 owner/group/others(拥有者/组/其他) 三种身份各有自己的 read/write/execute 权限。

**// 通过文件权限数字改变权限**

**文件的权限字符为： -rwxrwxrwx** ， 这九个权限是三个三个一组的！其中，我们可以使用数字来代表各个权限，各权限的分数对照表如下：r:4， w:2， x:1

每种身份(owner/group/others)各自的三个权限(r/w/x)分数是需要累加的，例如当权限为： -rwxrwx--- 分数则是：

owner = rwx = 4+2+1 = 7

group = rwx = 4+2+1 = 7

others= --- = 0+0+0 = 0

则该**文件的权限数字就是 770**

**chmod [-R] xyz 文件或目录**

xyz 文件的权限数字

$ chmod 777 .bashrc // 所有用户都有读写执行权限

$ chmod 754 .bashrc // 修改权限为rwxr-xr--

**// 通过符号类型改变文件权限**

**$ chmod [ugoa…][+-=][rwx] file or 目录**

u,g,o 表示user用户,group组, others其他

+-= 表示加入，除去，设定

rwx 表示读，写，执行

$ chmod u+x file1 #给文件file1添加用户执行权限

$ chmod u=rwx,g=rx,o=r test1 # 修改 test1 权限

# chmod a-x test1 # 去除全部人的可执行权限

// root创建或解压的文件夹及文件，普通用户无法查看

$sudo chmod -R 777 example/ // 修改文件夹权限给普通用户

$ 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

// 共享文件做法

$ mkdir shared\_dir

$ chgrp shared shared\_dir #将共享目录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

创建和管理加密文件夹：encfs

<https://linux.cn/article-2234-1.html>

$ sudo apt-get install encfs

$ encfs ~/.tuxtweaks ~/tuxtweaks //用encfs创建加密文件夹

~/.tuxtweaks包含被加密的信息

~/tuxtweaks解密的信息

$ echo "..." >~/tuxtweaks/test.txt // 自动更新加密版本~/.tuxtweaks

$ fusermount -u ~/tuxtweaks //卸载解密版本

//访问加密数据

$ encfs ~/.tuxtweaks ~/tuxtweaks

# 磁盘管理: mount, /etc/fstab, df, du

**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

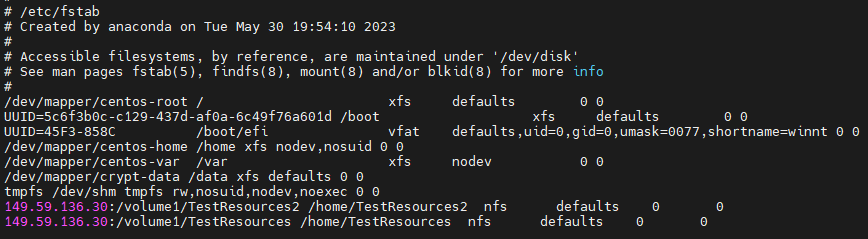
//自动挂载文件/etc/fstab

磁盘被手动挂载之后都必须把挂载信息写入/etc/fstab这个文件中，否则下次开机启动时仍然需要重新挂载。

**系统开机时会主动读取/etc/fstab这个文件中的内容，根据文件中的配置挂载磁盘。**

挂载约束：先挂载根目录，挂载点必须是已经存在的目录

[root@philips ~] # cat /etc/fstab



第一列/dev/mapper/centos-root表示要安装的块特殊设备或远程文件系统

第二列/ 表示存储挂载点

第三列xfs 表示挂载的文件系统类型

第四列defaults 表示文件系统挂载参数

第五列0 表示是否转储

第六列0 表示是否要fsck程序在重新启动时执行文件系统检查的顺序

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

$ df -hT #disk free查看所有已挂载磁盘的使用情况

-h ：以人们较易阅读的 GBytes, MBytes, KBytes 等格式自行显示；

-T ：显示文件系统类型, 连同该 partition 的 filesystem 名称 (例如 ext3) 也列出；

$ du -h #disk used显示某个特定目录的磁盘使用情况，默认是当前目录

-a ：列出所有的文件与目录容量，因为默认仅统计目录底下的文件量而已。

-h ：以人们较易读的容量格式 (G/M) 显示；

-s ：仅显示指定目录或文件的总大小，而不显示其子目录的大小。

$ du -h --max-depth=1 # 显示子目录的磁盘使用情况

$ sudo fdisk -l # 查一下磁盘或移动硬盘，假定要挂载/dev/sda

$ sudo mount /dev/sda /disk4T

$ 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] #通过设备文件或者是挂载点来指定要卸载的设备

搜索栏 -> disks #磁盘分区，格式化等

并不是所有磁盘分区都包含文件系统，系统可以通过使用磁盘空间来扩展内存容量。如果出现内存空间不足的情况，**linux虚拟内存系统会自动将内存中的进程移出至磁盘以及从磁盘移入内存。我们称其为交换swap, 因为空闲的进程被移出到磁盘，同时被激活的进程从磁盘移入到内存。用来保存内存页面的磁盘空间称为交换空间**

# 进程

## ps, top, kill and renice

$ ps --help all # 显示ps帮助信息

$ ps -auxwf #查看运行在linux系统上的所有进程

-a # all with tty, except session leaders

-x # processes without controlling ttys

-u #显示更详细的细息

-w #显示命令的全名

-f #显示完整格式的输出

**$ ps -auxwf | grep p3rtp** #查看p3rtp帐户正在运行的进程

$ tail -f /proc/<pid>/fd/1 #查看进程输出，1表示stdout

**$ top** #ps命令显示特定时间点信息，top命令实时显示进程信息

**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

推荐杀死进程流程：

1. 先试试TERM信号
2. 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。
3. 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 pid #等效于kill -TERM 告诉进程可能的话停止运行

$ kill -STOP pid #暂停进程

$ kill -CONT pid #恢复进程

$ kill -INT pid #等效于ctrl+c终止当前进程

$ kill -KILL pid #等效于kill -9

$ 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

$ netstat -tulpn | grep 5000 #查看哪个进程（服务）占用端口5000

$ sudo kill 19067 #杀死该进程，从而释放端口，假定是5000对应的进程id

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

## 进程通信，后台模式，定时定期自动执行脚本

* **进程通信(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,…

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.** 以后台模式运行shell脚本，脚本命令会和shell分开，作为系统上的独立后台进程运行。

$ ./test1 & #以后台模式运行shell脚本，终端关了，脚本运行也终止

以上方式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

$ **nohup** ./test1 **&** #即使退出终端，脚本仍以后台形式运行

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

退出终端，虽然以后台模式运行，但看不到stdout(虽然可以写到文件中，但想实时看呢？),解决方法：使用 screen 管理你的远程会话

https://www.ibm.com/developerworks/cn/linux/l-cn-screen/

$ screen #打开一个screen会话和screen窗口

$ python \*.py #在该screen shell运行脚本,实时输出stdout内容

ctrl+a, d #暂时断开该screen会话，或者终端被断电了

$ screen -ls #在shell中列出screen窗口

比如结果如下：

There is a screen on:

16582.pts-1.tivf06 (Detached)

1 Socket in /tmp/screens/S-root.

$ screen -r 16582 #重新进入之前detached 的screen窗口

* **定时/定期/自动执行脚本**

// 定时执行脚本

**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。

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

$ at -f filename 10:15~PM #若指定时间已经过去，第二天该时间运行该作业

$ 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 服务（守护进程）**在系统后台运行，并且会持续地检查**cron时间表, /etc/crontab 文件, /etc/cron.\*/目录和 /var/spool/cron/ 目录**,来获得计划执行的作业

crontab 是用来安装、卸载或者列出定时任务列表的命令。

cron 配置文件则用于驱动cron守护进程

每个用户都可以拥有自己的 crontab 文件

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

**$ crontab -e** #编辑定时任务

m h dom mon dow /path/to/command arg1 arg2

m字段：分钟 (0-59)

h字段：小时 (0-23)

dom字段：日期 (1-31)

mon字段：月份 (1-12 [12 代表 December])

dow字段：一周当中的某天 (0-7 [7 或 0 代表星期天])

/path/to/command - 计划执行的脚本或命令的名称

e.g.,

\*/5 \* \* \* \* /root/backupscript.sh #每隔 5 分钟运行一次 backupscript 脚本

0 1 \* \* \* /root/backupscript.sh #每天的凌晨 1 点运行 backupscript 脚本

15 3 1 \* \* /root/backupscript.sh #每月第一个凌晨 3:15 运行 backupscript 脚本

#每隔1分钟运行一次thread-ex.py,并且将output输出

**\*/1 \* \* \* \*** …/threading-ex.py >> …/out-threading-ex.txt

// 自动运行脚本

开机过程：

1. BIOS或者启动固件加载并运行**引导装载程序**
2. 引导装载程序在磁盘上找到**内核映像**，将其载入内存并启动
3. 内核初始化**设备及其驱动程序**
4. 内核挂载root**文件系统**
5. 内核使用PID 1来运行一个叫init的程序，用户空间在此时开始启动
6. Init启动其他的系统进程

基础的底层服务如udevd and syslogd

网络配置

中高层服务如cron and 打印服务等

登录提示符，GUI及其他应用程序

Init有三种实现版本System V init -> Upstart -> systemd (目前最新的版本)

6.1 systemd首先激活runlevel<N>.target，其中N是运行级别

6.2 systemd为/etc/rc<N>.d中的每一个符号链接在/etc/init.d中标识出对应脚本

6.3 systemd将脚本名和服务单元关联起来（如/etc/init.d/foo对应foo.service)

6.4 systemd根据rc<N>.d中的名称，激活服务单元，使用参数start or stop运行脚本

6.5 systemd尝试关联脚本进程和服务单元

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。

/etc/init.d/rc #ubuntu开机脚本

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

/etc/rc.local#Ubuntu本地开机文件,在此加入脚本全路径名

/etc/.bashrc #为所有用户自动运行脚本

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

~/.bash\_profile #为当前用户自动运行脚本

~/.bashrc #为当前用户自动运行脚本

# 网络

ping, nslookup, whois, arp/rarp, hostname, traceroute, netstat, nmap, tcmdump, wget/curl

<http://ping.eu/><https://www.wormly.com/tools>

$ **ping** google.com

ping: unknown host google.com

but $ping 8.8.8.8 works

method: DNS servers

/etc/network/interfaces

dns-nameservers 8.8.8.8

dns-nameservers 8.8.4.4

$ sudo /etc/init.d/networking restart

ping doesn't work under HTTP Proxy

ping needs a direct network connection on the IP level to do its work. A proxy works on a higher layer of the TCP/IP network model, where there is no direct access to the IP protocol.

web-based ping services (<https://www.wormly.com/tools>) will work.

$ **nslookup** google.com #主机名（域名） -> IP地址

$ **whois** google.com

whois就是一个用来查询域名是否已经被注册，以及注册域名的详细信息的数据库（如域名所有人、域名注册商）, WHOIS就是链接到域名数据库的搜索引擎，一般来说是属于网络信息中心（NIC）所提供和维护的名字服务之一。

$ hostname #Linux(or Window)系统查主机名,假定结果是QQ000CNG131W0P4

$ nslookup QQ000CNG131W0P4 #主机名 -> IP

$ ping QQ000CNG131W0P4

$ traceroute QQ000CNG131W0P4

<https://www.whois365.com/tw/domain/google.com>

$ arp -a #获取ARP缓存表（记录IP地址和MAC地址的对应关系)

$ rarp

**netstat：**显示路由表、网络连接以及每个网络接口设备的状态信息，一般用于检验本机各端口的网络连接情况

$ netstat -h

-a: (all) 显示所有活动中的网络连接和机器正在监听的 TCP/UDP 端口

-r: (route): 显示路由表

-t: (tcp) 显示tcp连接

-u: (udp) 显示udp连接

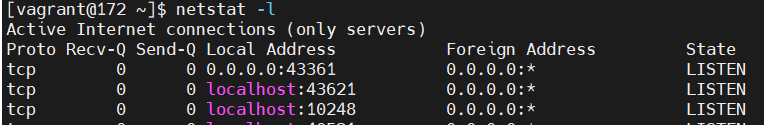
-l: (Listen) 只显示监听状态的套接字

-p: 输出中显示 PID 和进程名称

-s: 显示所有端口的统计信息

-i: 查看网卡接口信息

$ netstat -l



Proto：显示连接使用的协议。

Recv-Q：发送队列 Send-Q：接收队列

Local Address：本地地址：端口。 Foreign Address：外部地址：端口。

state：显示套接口当前的状态。

$ netstat -i #查看网卡接口信息

$ netstat -s #显示所有端口的统计信息

$ netstat -at #查看当前已经建立的tcp连接

$ netstat -lt #只列出所有监听中的 tcp类型的 端口

$ netstat -au #列出所有 udp 端口

$ netstat -lu #只列出所有监听中的 udp类型的端口

$ netstat -st #显示 TCP 端口的统计信息

$ netstat -su #显示 UDP 端口的统计信息

$ netstat -rn #获取路由控制表

**nmap:** 网络探测工具和安全/端口扫描器

<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。

端口表格：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)。

nmap还能提供关于目标机的进一步信息，包括反向域名，操作系统猜测，设备类型和MAC地址

$ sudo apt-get install –y nmap

$ nmap -A -T4 localhost #-A: 操作系统及版本的探测; -T4：可以加快执行速度

# 主机发现 （ping扫描）

在许多网络上，在给定的时间，往往只有小部分的IP地址是活动的。主机发现能够找到零星分布于IP地址海洋上的那些机器。

$ nmap -sP 192.168.10.0/24 #网络上有多少机器正在运行

# 端口扫描

nmap <target>这个简单的命令扫描主机<target>上的超过 1660个TCP端口。

$ nmap localhost #获取本机开放的端口和服务

$ nmap 192.168.10.0/24 #扫描192.168.10.0~255主机

**tcpdump: 抓取流动在网卡上的数据包**

<https://www.tcpdump.org/>

抓包原理：通过注册一种虚拟的底层网络协议来完成对网络报文的处理。

当网卡接收到一个网络报文之后，它会遍历系统中所有已经注册的网络协议，例如以太网协议处理模块来尝试进行报文的解析处理，这一点和文件系统的挂载相似，就是让系统中所有已注册的文件系统来进行尝试挂载，如果哪一个认为自己可以处理，那么就完成挂载。

当抓包模块把自己伪装成一个网格协议的时候，系统在收到报文的时候就会给这个伪协议一次机会，让它来对网卡收到的报文进行一次处理，此时该模块就会趁机对报文进行窥探。

// 截获主机[www.baidu.com](http://www.baidu.com)发送与接收所有的数据包

$ tcpdump -i eth0 host [www.baidu.com](http://www.baidu.com)

A close-up of a computer code

Description automatically generated

$ wget [www.baidu.com](http://www.baidu.com)

A screenshot of a computer code

Description automatically generated

测试130.147.128.227是否给149.59.136.46发送API请求？

$ tcpdump -i eth0 host 130.147.128.227 # 在46号机抓包来自227号机

在130.147.128.227号机，CT console -> launch mmsim

查看46号机stdout，是否有接收到信号

$ **wget** <http://www.website-name.com/file> #下载一个单独的文件

$ wget -r http://www.website-name.com #递归下载整个网站

$ wget -r -A png,pdf http://www.website-name.com #下载指定类型

$ wget -r -np -nd <ftp://ftp> #用户名:ftp密码@ftp地址/packages/

-r表示递归子目录

-np 的作用是不遍历父目录（–no-parent）

-nd 表示不在本机重新创建目录结构（–no-directories）

$ wget -r -np -nd -accept=iso http://example.com/fedora13/i386/

–accept=iso表示 wget 仅下载 i386 目录中所有扩展名为 iso 的文件。你也可以指定多个扩展名，只需用逗号分隔即可。

–reject=iso表示不下载扩展名为iso的文件，其他文件全部下载

$ wget -i address.txt #实现批量下载，所有下载地址存放到 address.txt

$ wget -c -t 100 http://example.com/fedora13.iso

-c选项实现断点续传

-t表示重新连接的次数,100则表示重新连接100次，如果设置成-t 0则表示无限次重新连接，直到下载完成。

**cURL**是支持FTP, HTTP, FTPS, TFTP, TELNET, IMAP, POP3等协议的客户端应用

$ sudo apt-get install curl

$ curl Is google.com # 测试能否访问google.com

$ curl <http://www.centos.org> #下载单个文件

$ curl -s -o /home/curl.tar.gz http://download/curl.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：将文件保存为命令行中指定的文件名的文件中

$ curl -u username:password -C -O URL1 -O URL2 #同时获取多个文件

-u username:password cURL授权

-O：使用URL中默认的文件名保存文件到本地

-C选项断点续传

$ curl -u ftpuser:ftppass -O [ftp://.../public/](ftp://ftp_server/public_html/) #从FTP服务器下载文件

$ curl -x proxysever.test.com:3128 <http://google.co.in> #为cURL设置代理

$ curl -u user --data "p1=value1&p2=value" <https://api.com> # POST

$ curl -u ftpuser:ftppass -T "{file1,file2}" <ftp://ftp.com> #上传文件到FTP服务器

$ curl --form "fileupload=@filename.txt" #上传文件

$ curl -I -X DELETE [https://api.com](https://api.github.cim) -X # 选项指定其它协议

## 远程登录Linux系统

* **无界面访问 （通过xshell 如putty, ssh-> ubuntu SSH）**

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开启sshd服务(端口默认是22)

$ sudo apt-get install openssh-server -y

$ sudo ps –e | grep ssh #查看ssh服务是否启动

$ sudo service ssh start #开启sshd服务

// linux通过ssh访问linux

$ ssh -X [user@ip](mailto:user@ip) 然后输入密码

// Win10 通过自带OpenSSH访问linux

打开Powershell

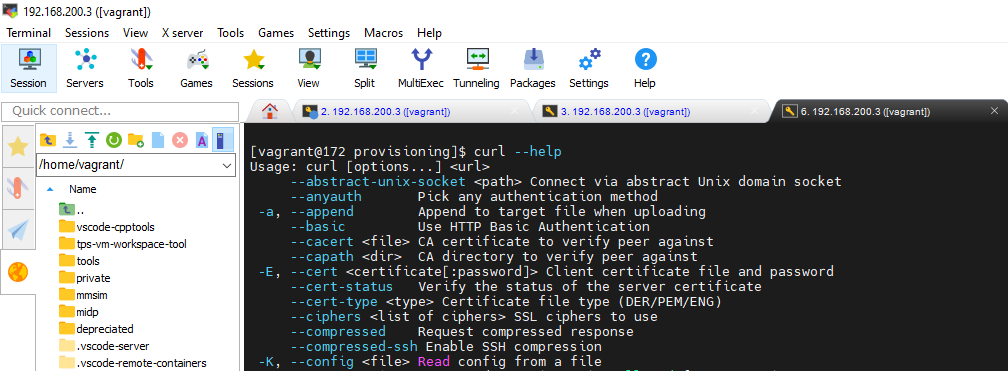
> ssh [qzlin@192.168.1.7](mailto:qzlin@192.168.1.7)

// window通过putty访问linux

192.168.1.13:22 SSH 假定开启sshd服务的服务器地址是192.168.1.13

输入用户名和密码即可登录

// Win10通过MobaXterm软件，ssh访问linux



* **有界面访问（通过window remote control->Ubuntu vnc4server）**

<https://www.htcp.net/880.html>

**Ubuntu安装vnc4server**

// 安装软件

apt update # 更新源及系统

apt install ubuntu-mate-core ubuntu-mate-desktop #安装 ubuntu-mate-desktop

apt install vnc4server -y # 安装 VNC server

// 配置VNC server

vncserver :1

passward: xxxxxx

vncserver -kill :1

gedit ~/.vnc/xstartup

#!/bin/sh

# Uncomment the following two lines for normal desktop:

# unset SESSION\_MANAGER

# exec /etc/X11/xinit/xinitrc

[ -x /etc/vnc/xstartup ] && exec /etc/vnc/xstartup

[ -r $HOME/.Xresources ] && xrdb $HOME/.Xresources

xsetroot -solid grey

vncconfig -iconic &

# x-terminal-emulator -geometry 80x24+10+10 -ls -title "$VNCDESKTOP Desktop" &

mate-session &

vncserver :1

开机自启动

$ sudo crontab –e

@reboot /usr/bin/vncserver :1

**Win10**

install TigerVNC #安装 VNC 客户端

启动TigerVNC -> 161.92.142.107:1

Passward: 160810

若要改变远程屏幕大小，在/usr/bin/vncserver里设置

$ sudo vncserver –kill :1

$ sudo gedit /usr/bin/vncserver

Search: geometry=“1920x1080“ 根据客户端电脑屏幕分辨率设置大小

$ sudo vncserver :1

If wrong type or access mode of /home/gdp/.vnc

$ vncserver :1

## linux and win 互传文件，文件共享

**ftp服务器**

// Win10架ftp服务器：

<http://jingyan.baidu.com/article/6079ad0e67acf828ff86db3f.html> <https://blog.csdn.net/a254342594/article/details/124740312>

已测试，假定IP:192.168.1.4, user=admin, password=qzlin

Win10客户端 filezilla (需要信息：ip:port, user/password)

// Ubuntu:16.04 架ftp服务器

<https://www.ostechnix.com/install-vsftpd-server-ubuntu-16-04-lts/>$ sudo apt-get install vsftpd

$ sudo nano /etc/vsftpd.conf #Configure VSFTPD

anonymous\_enable=NO #禁止匿名访问

local\_enable=YES #允许本地访问:ftp localhost

write\_enable=YES #可以上传

$ sudo service vsftpd restart

or $sudo /etc/init.d/vsftpd restart

or $sudo systemctl restart vsftpd

$ sudo systemctl status vsftpd #check service status

$ sudo adduser uftp #Create FTP users

terminal will ask you set password

knowledge$ sudo ln –s . /home/uftp/ 将共享文件软连接至uftp

knowledge$ sudo chmod 777 –R knowledge/ 设置共享文件内容权限

**客户端**

$ sudo apt-get install filezilla // 安装filezilla

$ sudo apt-get install filezilla-locales // 装个语言包

$ filezilla (需要信息：ip:port, user/password)

Window10安装filezilla 客户端

Sftp://ip user password port:22

**网络传输文件**

拷贝本地目录至服务器

$ rsync -va /home/qzlin/Pictures linqizhong@10.10.52.51:dl

注意：假定本地Pictures/... -> /home/linqizhong/dl/Pictures...

若指定文件夹Pictures/，那么只拷贝Pictures/文件夹下内容

Copy from remote to local

$ rsync -va [linqizhong@10.10.52.51:dl/RefineDet/models](mailto:linqizhong@10.10.52.51:dl/RefineDet/models) .

$ ftp ip (需要信息：user/password） // ubuntu访问通过ftp客户端

系统用户登录ftp便进入他们的home目录

ftp> help

ftp> ls

ftp> get

$ wget -rc <ftp://qzlin:q@192.168.1.14/Documents/script/>

注意：ftp服务端根目录是/home/qzlin/ 如果你是用qzlin登录

-r 下载目录及内容

-c 断点续传

Xmanager (window界面软件连接unix环境）

Xshell: 用于MS Windows平台的强大的SSH，TELNET，和RLOGIN终端仿真软件。它使得用户能轻松和安全地从Windows PC上访问Unix/Linux主机。

Xftp： 用于MS Windows平台的强大的FTP和SFTP文件传输程序。Xftp能安全地在Unix/Linux和Windows PC之间传输文件。

若是virtualbox,主机和虚拟机可以共享文件夹<http://blog.csdn.net/longerzone/article/details/32119457>

virtualbox -> 共享文件夹 ->

共享文件夹路径(E:/knowledge) + 共享文件夹名称(knowledge)

在虚拟机里：$sudo /sbin/mount.vboxsf knowledge /home/uftp

**文件共享**

Linux访问Win10 网盘

方法1：在win10安装FileZilla, open FileZilla 连接Linux

Protocol: SFTP – SSH File Transfer Protocol

ip: 161.92.168.32

user: qzlin

password: 1

本地站点：[\\dephbgpflhxx015.pfh.research.philips.com\export\_TraumaCTC](file:///\\\\dephbgpflhxx015.pfh.research.philips.com\\export_TraumaCTC)

方法二：将网盘直接挂载到Linux系统上

1. 安装cifs

$ sudo apt install cifs-utils

1. 挂载网络驱动器

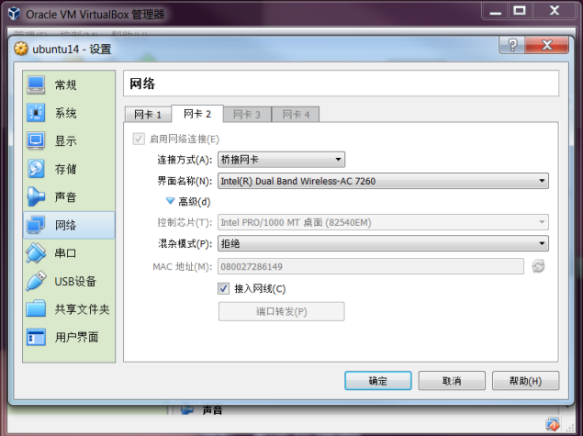
$ sudo [mount.cifs](https://wx.qq.com/cgi-bin/mmwebwx-bin/webwxcheckurl?requrl=http%3A%2F%2Fmount.cifs&skey=%40crypt_40deaefc_742038f9dadae4c5c76eb1837bb7b653&deviceid=e423832393435143&pass_ticket=HmxvEEcqPzqKsTd2mzitoq%252BLfC2o2nllR9%252Bx0x5mUmGagpXxjH2BWm0YPQJNYC4H&opcode=2&scene=1&username=@355abf08b97b706a03cfe2b75f6a09dfaee1595aad7cc5a014c06a15b9c5bad7" \t "_blank) //dephbgpflhxx015.pfh.research.philips.com/export\_TraumaCTC export\_TraumaCTC -o user=320029857,pass=Yuping@803,domain=CODE1

其中user中填你的code1账号，pass填你的code1密码，然后在你挂载的路径下就能看到汉堡的数据。后面就可以从/你要挂载的路径/下复制了

## 虚拟机与主机通信

* <http://www.2cto.com/os/201205/133370.html>使虚拟机能够上网 --NAT(网络地址转换模式)

**使用NAT模式，就是让虚拟系统借助NAT(网络地址转换)功能，通过宿主机器所在的网络来访问公网。**NAT模式下的虚拟系统的TCP/IP配置信息是由VMnet8(NAT)虚拟网络的DHCP服务器提供的，无法进行手工修改，因此虚拟系统也就无法和本局域网中的其他真实主机进行通讯。所以主机不能ping到虚拟机

* 使主机能ping到虚拟机 --bridged(桥接模式)

　　在桥接模式下，你需要手工为虚拟系统配置IP地址、子网掩码，而且还要和宿主机器处于同一网段，这样虚拟系统才能和宿主机器进行通信。由于这个虚拟系统是局域网中的一个独立的主机系统，那么就可以手工配置它的TCP/IP配置信息，以实现通过局域网的网关或路由器访问互联网。 **使用桥接模式的虚拟系统和宿主机器的关系，就像连接在同一个Hub上的两台电脑。想让它们相互通讯，你就需要为虚拟系统配置IP地址和子网掩码**，否则就无法通信。

启动虚机，发现多了一个未配置的网卡eht1

为该网卡添加一个网络配置，手动输入IP地址、子网掩码和网关（与宿主机要在同一网段），参考宿主机$ipconfig,假定主机

IP: 192.168.1.11

Subnet Mask: 255.255.255.0

Default Gateway: 192.168.1.1

$ping 192.168.1.11

# GPU(显卡)

* **GPU Driver**

<https://ubuntu.com/server/docs/nvidia-drivers-installation>

# 查看 GPU driver version

$ apt list –installed nvidia-driver\*server



$ cat /proc/driver/nvidia/version

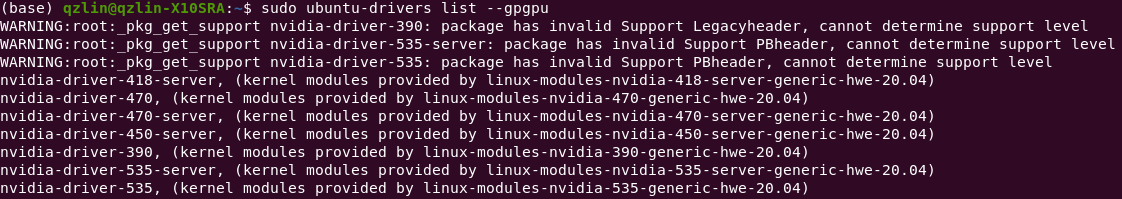


# Check the available drivers for your hardware

$ apt list nvidia-driver\*server

$ sudo ubuntu-drivers list # For desktop

$ sudo ubuntu-drivers list --gpgpu # for servers



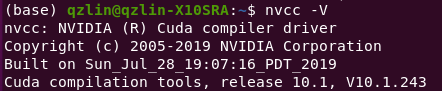
# upgrade GPU driver

$ sudo apt-get purge "\*nvidia\*470\*" # remove old driver

$ sudo apt install -y linux-generic # Install the latest kernel.

$ sudo ubuntu-drivers install nvidia:535 # install new driver

$ nvcc -V



$ cat /usr/local/cuda/include/cudnn.h 查看cuDNN version

$ dpkg –l | grep cudnn 通过.deb安装的，没有头文件

* **Monitor GPU resource**

$ watch -n 1 nvidia-smi # gpu driver, cuda version,

cat /usr/local/cuda/version.txt # cuda 版本

cat /usr/local/cuda/include/cudnn.h | grep CUDNN\_MAJOR -A 2 # cudnn 版本

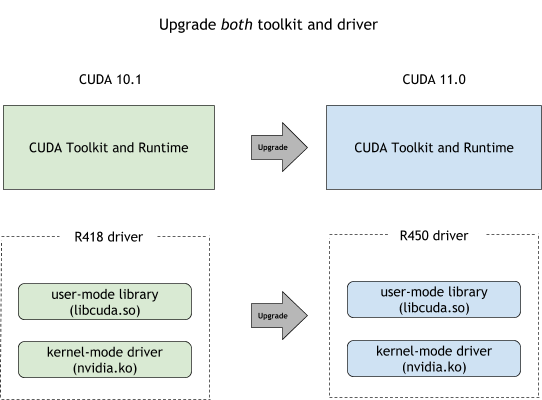
or sudo apt search cudnn if you install via dpkg \*.deb

#清空GPU资源

$ fuser -v /dev/nvidia\* 查看占用显卡进程号

$sudo kill -9 pid # pid 可以从nvidia-smi显示的process看出来

* **upgrading a CUDA Toolkit** involves upgrading both the toolkit and the driver

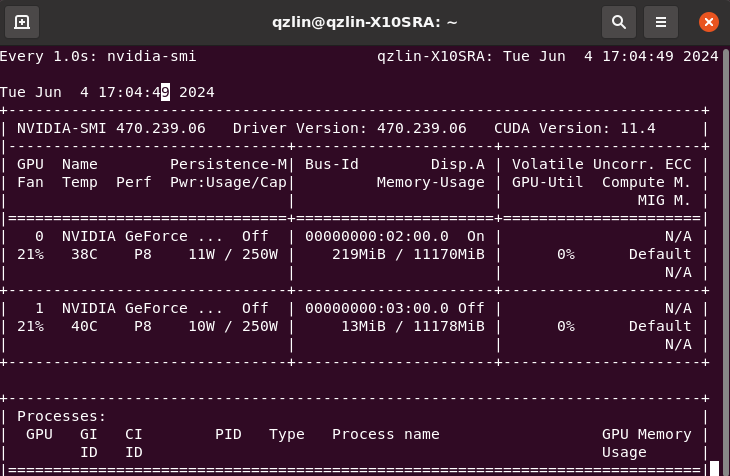


A black and white box with text

Description automatically generated

Check workstation with GPU

$ watch -n 1 -d nvidia-smi



Driver Version 470.xxx, 查表CUDA Toolkit and Driver compatibility, 可以支持CUDA Toolkit 11.xxx

从而可以安装Pytorch, 自带的cuda runtime 11.8

A screenshot of a computer

Description automatically generated

# X11 : X window system的第11个版本，Linux的工具包

WINDOWS 95及以后的版本，图形界面是操作系统的一部分，在系统内核中实现图形界面。

linux只是一个基于命令行的操作系统。

The X Window System 显示图形用户界面的标准工具包和网络协议。X 仅为 GUI 环境构建提供基本框架，在屏幕上绘图和移动窗口，以及与鼠标和键盘交互。

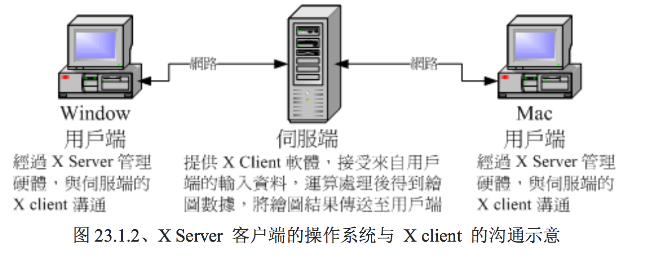
软件架构：

X基于Client-Server模型。X Server管理硬件，X Client是应用程序，X Client应用程序将要呈现的界面告知X Server，X Server将结果通过硬件绘制出来。(X takes the perspective of the application, rather than that of the end-user: X provides display and I/O services to applications, so it is a server; applications use these services, thus they are clients.)

每部客户端主机需要安装**X Server** : 管理客户端硬件(如接收键盘鼠标等输入信息告知X Client)，并将图形绘制到屏幕上。

服务器端提供**X Client**软件 : 根据X Server发送的事件（输入设备行为）进行处理，从而产生绘图的数据。

X Client不需要知道X Server的硬件配备与操作系统。因为X Client单纯只处理绘图数据，本身不绘图。



**X Window Manager** 负责全部X Client的管理，提供许多的控制元素如任务栏和背景桌面，管理虚拟桌面和提供窗口控制参数如窗口大小移动重叠最小化等。常见的X Window Manager有KDE, GNOME等

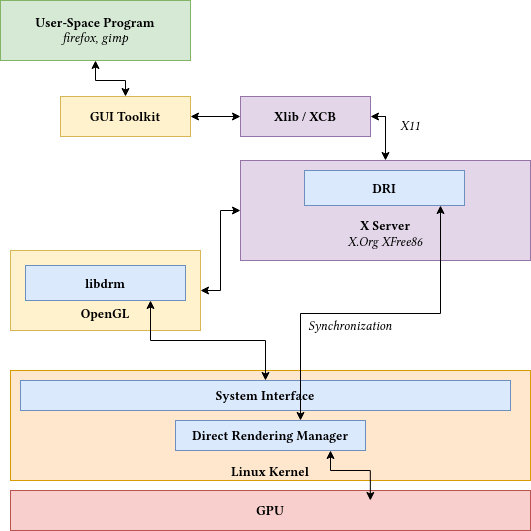
Remote desktop (远程桌面)

To run an X client application on a remote machine, the user may do the following:

* on the local machine, open a terminal window
* use ssh -X command to connect to the remote machine
* request a local display/input service (e.g., export DISPLAY=[user's machine]:0 if not using SSH with X forwarding enabled)

The remote X client application will then make a connection to the user's local X server, providing display and input to the user.

Alternatively, the local machine may run a small program that connects to the remote machine and starts the client application.



**GUI toolkit:** The X Client can draw primitive shapes like rectangles and text to the screen. However, it doesn’t have the ability to draw complex widgets like buttons and drop-downs, which we can achieve using a GUI toolkit.

**Xlib (X Library)** **is the client-side implementation**, which is used by GUI toolkits like GTK to present an enhanced front-end for the graphical applications.

**The X Server (X.Org or X)** is the most popular **server-side implementation** of the X Window System. X is responsible for access to the graphics card, display screen, and I/O. Events generated by the X clients are handled, and the result is communicated back to its respective clients by X. The X Server is usually started by a display manager such as GDM.

**Xlib** : C Language X Interface to the X Window System protocol

<https://www.x.org/releases/X11R7.5/doc/libX11/libX11.html>

The X Window System supports one or more screens containing overlapping **windows** or subwindows.

A single X server can provide display services for any number of screens.

A set of screens for a single user with one keyboard and one pointer (usually a mouse) is called a **display**. Each **display** can have multiple screens. A **screen** is a physical monitor.

All the windows in an X server are arranged in strict hierarchies. A **window** has a border zero or more pixels in width, which can be any pattern (pixmap) or solid color. Each window and pixmap has its own coordinate system. **The coordinate system** has the X axis horizontal and the Y axis vertical with the origin [0, 0] at the upper-left corner.

X also provides off-screen storage of graphics objects, called **pixmaps**. Windows and pixmaps together are referred to as drawables.

Many Xlib functions will return an **integer resource ID**, which allows you to refer to objects stored on the X server. These can be of type **Window, Font, Pixmap, Colormap, Cursor, and GContext**, as defined in the file <X11/X.h>

**Input events** (for example, a key pressed or the pointer moved) arrive asynchronously from the server and are queued until they are requested by an explicit call (for example, XNextEvent or XWindowEvent).

<X11/Xlib.h> // main header file for Xlib. The majority of all Xlib symbols are declared

<X11/Xcms.h> // contains symbols for color management facilities.

<X11/Xutil.h> // functions, types, and symbols used for inter-client communication

<X11/Xresource.h> // functions, types, and symbols for the resource manager facilities

<X11/Xatom.h> // predefined atoms, with the prefix ‘‘XA\_’’

<X11/cursorfont.h> // cursor symbols for the standard cursor font, with the prefix ‘‘XC\_’.

<X11/keysymdef.h> // standard KeySym values, with the prefix ‘‘XK\_’’.

<X11/keysym.h> //preprocessor symbols XK\_MISCELLANY, XK\_XKB\_KEYS, …

**test.cpp**

#include <X11/Xlib.h>

#include <unistd.h>

main() {

Display \*d = XOpenDisplay(0); // Open a display.

If (!d) return 0;

// Create the window

Window w = XCreateWindow(d, DefaultRootWindow(d), 0, 0, 200,

100, 0, CopyFromParent, CopyFromParent, CopyFromParent, 0, 0);

// Show the window

XMapWindow(d, w);

XFlush(d);

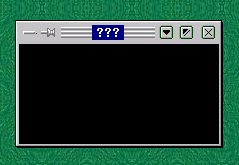
// Sleep long enough to see the window.

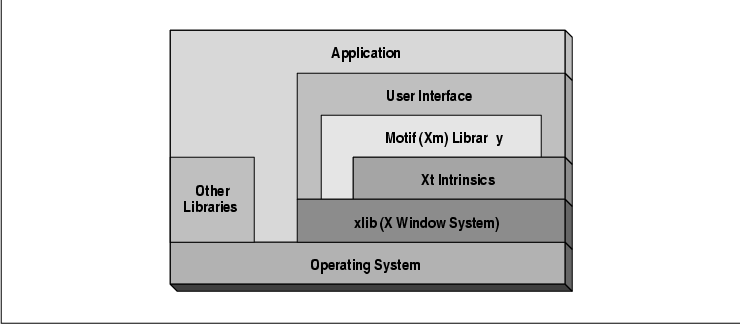
sleep(10);

}

$ g++ test.cpp -L/usr/X11R6/lib -lX11

$ ./a.out





**Motif-style graphical user interface (GUI)**: is created using both the Motif **Xm** library and the Intrinsics' **Xt** library

**X Toolkit Intrinsics (Xt):** Xt provides an object-oriented framework for creating reusable, configurable user-interface components called widgets.

**Motif (Xm) Library:**  provides widgets for such common user-interface elements as labels, buttons, menus, dialog boxes, scrollbars, and text-entry or display areas.

**hello.c**

#include <Xm/PushB.h>

main(int argc, char \*argv[]) {

XtSetLanguageProc (NULL, NULL, NULL);

XtAppContext app;

Widget toplevel = XtVaAppInitialize (&app, "Hello", NULL, 0, &argc, argv, NULL, NULL);

XmString label = XmStringCreateLocalized ("Push here to say hello");

Widget button = XtVaCreateManagedWidget ("pushme",

xmPushButtonWidgetClass, toplevel, XmNlabelString, label, NULL);

XmStringFree (label);

XtAddCallback (button, XmNactivateCallback, button\_pushed, NULL);

XtRealizeWidget (toplevel);

XtAppMainLoop (app);

}

void button\_pushed(Widget widget, XtPointer client\_data, XtPointer call\_data) {

printf ("Hello Yourself!0);

}

// library order is important. Xm relies on Xt, and both Xm and Xt rely on Xlib

$ gcc -O -o hello hello.c -lXm -lXt -lX11

A grey rectangular sign with black text

Description automatically generated

# 软件

## 软件程序管理: dpkg (apt-get), rpm (yum)

* 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CentOS**, 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.

**//基于Debian的发行版(e.g. Ubuntu): dpkg, apt-get, apt**

$ man dpkg

$ dpkg -? 查询dpkg的使用命令

$ dpkg -i \*.deb 安装包

$ 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

若用dpkg -i 安装某个包，出现依赖包没有安装，可以用$ apt-get -f install自动安装依赖包

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$ 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 --help

$ 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

apt(Advanced Packaging Tool)

# apt [options] [command] [package ...]

options：选项-h(帮助), -y(当安装过程提示选择全部为"yes”), -q(不显示安装的过程)等。

command：要进行的操作。

package：安装的包名。

# apt list --installed // 列出所有已安装的包

# apt list --all-versions // 列出所有已安装的包的版本信息

# sudo apt search <keyword> // 查找软件包

# sudo apt install <package\_name> // 安装指定的软件

# sudo apt install <package\_name>=<version\_number>

# sudo apt install <package\_1> <package\_2> <package\_3> // 安装多个软件包

# sudo apt update // 列出所有可更新的软件清单

# sudo apt update <package\_name> // 更新指定的软件

# apt list --upgradeable // 列出可更新的软件包及版本信息

# sudo apt upgrade // 升级软件包

# sudo apt full-upgrade // 升级软件包，升级前先删除需要更新软件包

# sudo apt show <package\_name> // 软件包信息,如版本号，安装大小，依赖关系等等

# sudo apt remove <package\_name> // 删除软件包

# sudo apt autoremove // 清理不再使用的依赖和库文件

# sudo apt purge <package\_name> // 移除软件包及配置文件

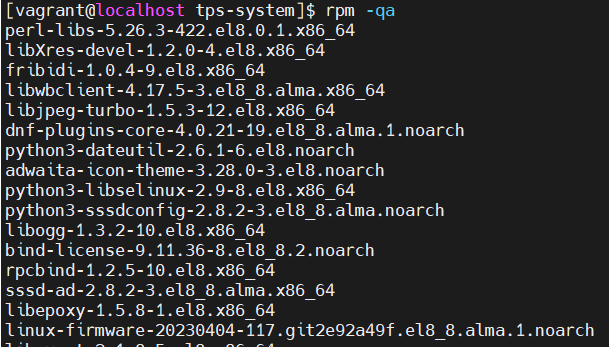
**// 基于Red Hat的发行版(e.g. CentOS, AlmaLinux): rpm, yum**

CentOS源于RHEL（Red Hat Enterprise Linux）依照开放源代码规定释出的源码所编译而成的Linux操作系统, Centos8的维护已到期，而Centos7也将于2024年6月结束维护。AlmaLinux和Rocky Linux是为了替代Centos而上线的Linux项目

rpm: repository package manager, which is used to build, install, query, verify, update and erase software packages on RPM based distro.

$ rpm --help

$ rpm -qa // list all installed packages



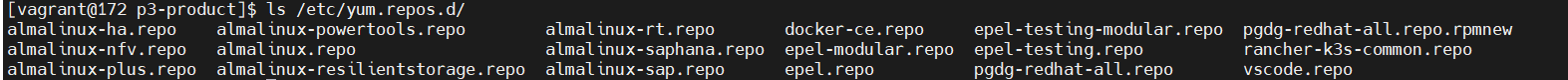
**yum (Yellow dog Updater, Modified),** 基于 RPM 包管理，能够从指定的服务器自动下载 RPM 包并且安装，可以自动处理依赖性关系，并且一次安装所有依赖的软件包。

原理：yum在服务器端存有所有的RPM包，并将各个包之间的依赖关系记录在文件中，当管理员使用yum安装RPM包时，yum会先从服务器端下载包的依赖性文件，通过分析此文件从服务器端一次性下载所有相关的RPM包并进行安装。

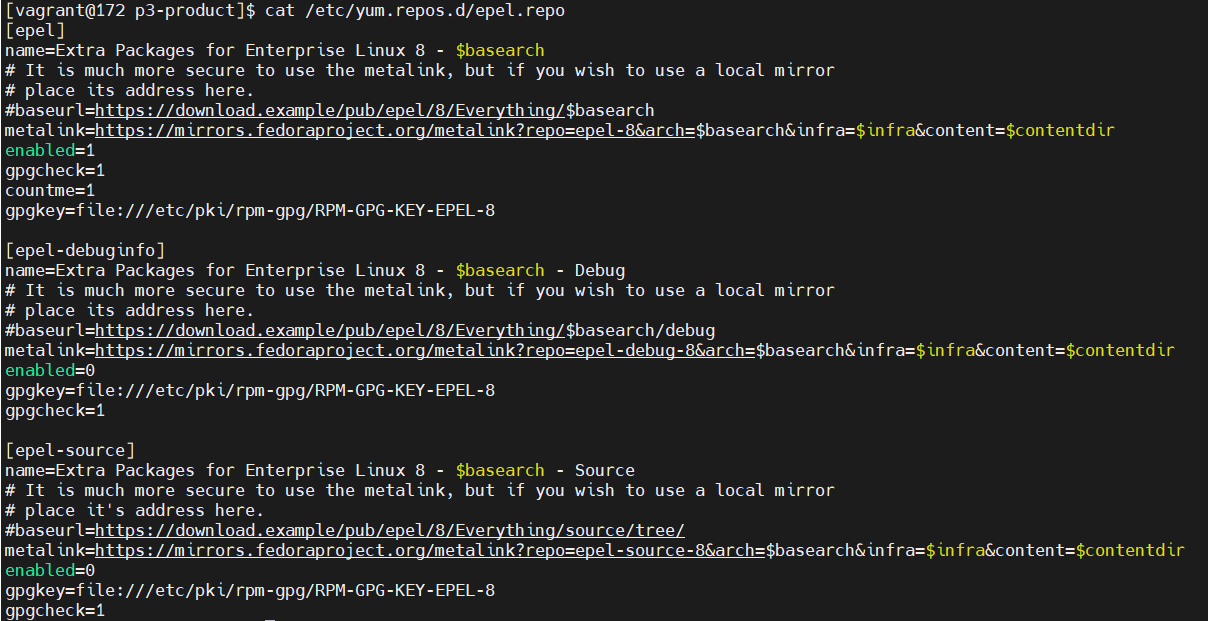
yum源配置: 使用yum安装软件包之前，需指定yum下载RPM包的位置，此位置称为yum源。Yum源即可以使用网络yum源，也可以将本地光盘作为yum源。

网络yum源配置文件位于/etc/yum.repos.d/目录下，文件扩展名为.repo

$ ls /etc/yum.repos.d/



$ cat /etc/yum.repos.d/epel.repo



[epel] : 容器名称

name : 容器说明

mirrorlist :镜像站点

baseurl : yml 源服务器的地址

enabled : 0表示此容器不生效

gpgcheck : 0表示RPM的数字证书不生效

gpgkey : 数字证书的公钥文件保存位置。不用修改

yum源更换:

$ cd /etc/yum.repos.d

$ mv CentOS-Base.repo CentOS-Base.repo.backup #备份源

$ wget -P /etc/yum.repos.d <https://mirrors.xxx/CentOS7-Base.repo> #下载repo文件

$ yum clean all # 更新缓存

$ yum makecache

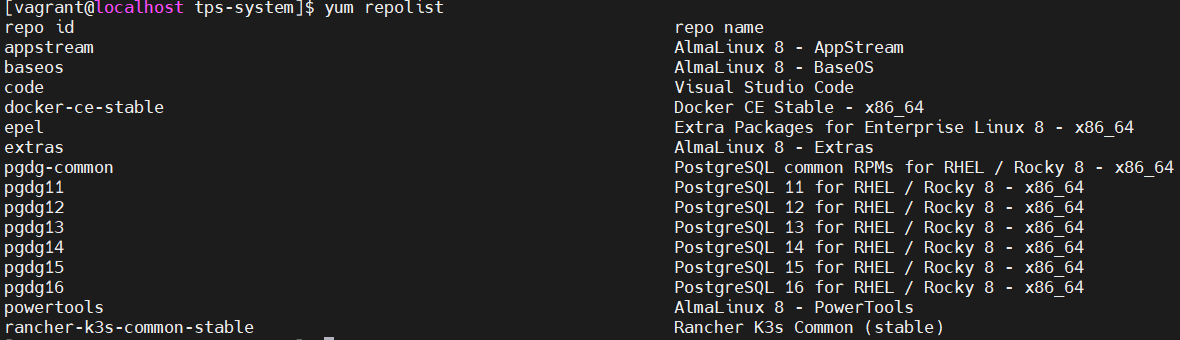
yum [options] [command] [package ...]

options：如-h帮助，-y当安装过程提示选择全部为 "yes"，-q不显示安装的过程等

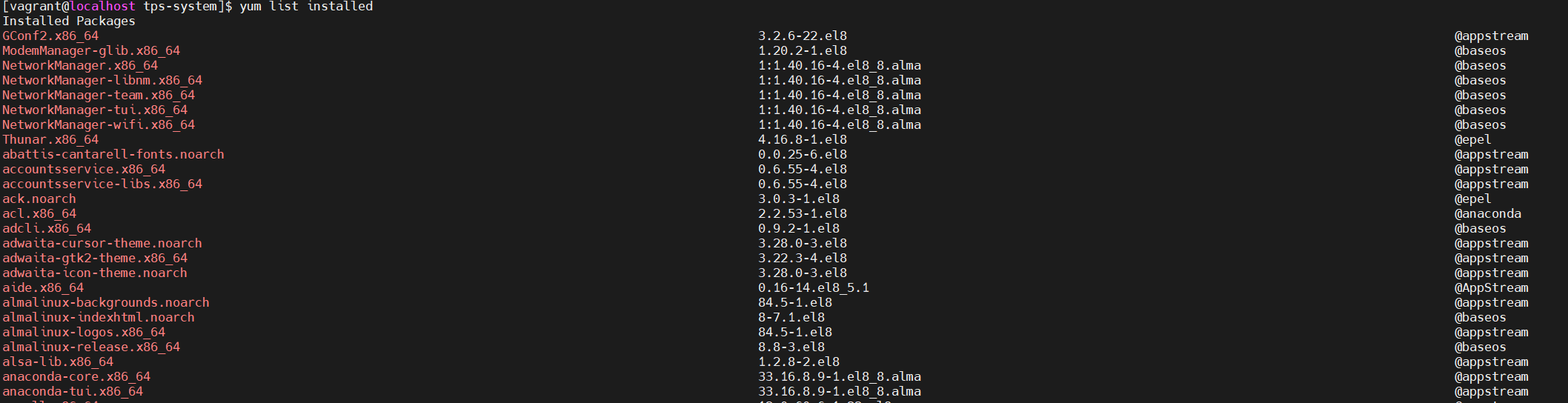
command：要进行的操作。

package：安装的包名。

$ yum repolist // list repos



**$ yum list installed // list all installed packages**



$ yum list installed | wc -l // 统计安装包数目

$ yum list // 列出所有可安裝的软件清单

$ yum search <keyword> // 查找软件包

$ yum install <package\_name>

$ yum -y install <package\_name>

-y表示自动回答结果为yes，如果不加-y,那么每个安装软件都需要手动回答yes

$ yum check-update // 列出所有可更新的软件清单

$ yum update <package\_name>

$ yum update // 更新所有软件

$ yum remove <package\_name> // 删除软件包

$ yum clean packages // 清除缓存目录下的软件包

$ yum clean headers // 清除缓存目录下的 headers

$ yum clean oldheaders // 清除缓存目录下旧的 headers

yum clean, yum clean all (= yum clean packages; yum clean oldheaders) :清除缓存目录下的软件包及旧的 headers

* 源码安装

1. 解压xx.tar.gz, xx.tar.Z, xx.tgz, xx.tar, xx.bz2

$ tar -zxf xx.tar.gz

$ tar -zxf xx.tar.Z

$ tar -zxf xx.tgz

$ tar -xf xx.tar

$ 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

## 常用软件：md5,

* **MD5(Message-Digest Algo 5): 转换数据为固定长度值, 用于确保信息传输完整一致**

$ md5sum \*.jpg

$ md5sum \*.txt

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。

* 通过MD5查找并删除重复文件

$ sudo apt install fdupes

$ fdupes -rdN ./

r: 表示递归查找

d: 表示删除重复图片并保留一张

N: 表示不确认直接删除

删除视觉上一样或非常相似的图片

基于perceptual hash算法http://www.phash.org/

$sudo apt install findimagedupes

* [**https://www.linuxbabe.com/desktop-linux/uget-aria2-download-manger-linux**](https://www.linuxbabe.com/desktop-linux/uget-aria2-download-manger-linux)**视频下载/播放/剪辑**

Transmission BitTorrent Client

youtube-dl #tools for YouTube.com, 土豆

$sudo pip install --upgrade youtube\_dl

$youtube-dl –help

$youtube-dl --extractor-descriptions #查询支援的網站

$ youtube-dl --list-formats #查询网站支持的视频格式

$youtube-dl "影片的網址"

$ youtube-dl --batch-file YouTube\_URLs.txt #下载多部影片

$youtube-dl #下载土豆视频

$sudo apt-get install ktorrent

$sudo apt-get install amult

$ktorrent #GUI方式下载视频(磁力，torrent等)

<http://www.tudou.com/programs/view/FcmdkdPx6aA/>

$ffmpeg #合并视频文件

第一步：获取视频所在目录下的全部flv格式文件，并将它们输出成列表打印出来；

# CD切换到视频所在目录下

#现将文件名小改一下，不然顺序不对

mv 1.flv 01.flv

mv 2.flv 02.flv

# bash获取当前目录下的所有flv文件

for f in \*.flv; do echo "file '$f'" >> mylist.txt; done

第二步：开始批量合并flv视频

ffmpeg -f concat -i mylist.txt -c copy output.flv

#字幕文件转换

ffmpeg -i subtitle.srt subtitle.ass .srt > .ass

ffmpeg -i subtitle.ass subtitle.srt .ass -> .srt

#字幕文件嵌入视频文件

ffmpeg -i video.avi -vf subtitles=subtitle.srt out.avi #字幕文件为subtitle.srt

#video.mkv字幕嵌入到out.avi文件

ffmpeg -i video.mkv -vf subtitles=video.mkv out.avi

ffmpeg -i video.avi -vf "ass=subtitle.ass" out.avi #嵌入ASS字幕到视频文件

$sudo apt-get install vlc browser-plugin-vlc

$vlc #视频文件播放器，gui操作

* **图像处理imagemagick**

$**eog** \*.jpg

$display \*.jpg 显示图像

$sudo apt-get install **imagemagick** --fix-missing

$ convert image.jpg image.png 转换格式

$ convert \*.jpg images.gif

$ convert ‘images.gif[0]’ image.png 取某图像

$ convert ‘images.gif[3,2,4]’ images.mng multi-frame file

$ convert image.png -resize 135x135 output.png 缩放

$convert @myimages.txt mymovie.gif

假定myimages.txt

Frame001.jpg

Frame002.jpg

Frame003.jpg

$ convert foo.png -crop 640x480+50+100 out.png

crop rectangle to start at top corner X: 50 Y: 100 and the crop rectangle to be of size W: 640 H:480

* **截图软件**

sudo apt-get install shutter

$shutter

* **音乐软件**

<http://music.163.com/#/download>

网易云音乐

$dpkg -i netease-cloud-music\*

$netease-cloud-music

* **Office**

$sudo apt install libreoffice-gnome #Libreoffice (ubuntu self-contains)

Install WPS

https://www.cnblogs.com/gisalameda/p/6839482.html

Install wubi

$sudo apt install fcitx fcitx-table-wbpy

System setting -> language support -> Fcitx (Keyboard input method system)

Restart computer

Language icon -> configure fcitx -> click “+” -> select wubi

解决WPS不能中文输入问题？

Refer to: http://blog.csdn.net/duxu24/article/details/52711693

WPS文字

$ sudo vim /usr/bin/wps

export XMODIFIERS="@im=fcitx"

export QT\_IM\_MODULE="fcitx"

WPS表格

$ sudo vim /usr/bin/et

export XMODIFIERS="@im=fcitx"

export QT\_IM\_MODULE="fcitx"

WPS演示

$ sudo vim /usr/bin/wpp

export XMODIFIERS="@im=fcitx"

export QT\_IM\_MODULE="fcitx"

PDF Viewer

$evince \*.pdf 系统自带

* **邮件客户端**

<https://www.csdn.net/article/2014-04-22/2819419>

Thunderbird

安装 -> 搜索栏: thunderbird -> 运行

配置username and password

* **Web Broswer**

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

$google-chrome \*.html \*.pdf

Firefox已自带

* **文本编辑器**

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

$ sudo apt-get update

$ sudo apt-get install notepadqq #notepad++

* **文件格式互换**

html, Markdown, Latex, MediaWiki, docx, pdf...格式转换

$docker run -ti --rm -v $PWD:/pdf bwits/pdf2htmlex pdf2htmlEX --zoom 1.3 test.pdf #Pdf2htmlEX for pdf -> html

运行完后，会在$PWD目录下生成test.html文件

http://pandoc.org/demos.html

<http://pandoc.org/try/> #online try:

$wget pandoc-1.17.2-1-amd64.deb #pandoc

$dpkg -i pandoc…

$pandoc --help

$pandoc -f mediawiki -t html5 -s haskell.wiki -o example32.html //MediaWiki -> html5

$pandoc MANUAL.txt -o example1.html //markdown -> html

$pandoc -s MANUAL.txt -o example4.tex //markdown -> LateX

$pandoc -s -r html http://www.gnu.org/software/make/ -o example12.text //html -> markdown

$pandoc MANUAL.txt --latex-engine=xelatex -o example13.pdf //markdown -> PDF

## 开发软件: git, intellij, jdk, spring, gradle, docker, .NET

Unix的哲学是一组UNIX操作系统的程序员建立的软件设计原则。它认为：编写小的、目的有限的、能互操作的程序，胜过大的、功能丰富的应用程序

较小的程序更容易理解，通过能够互操作，它们可以是强大的应用程序的构建块。

* **Proxy配置**

$ gedit ~/.bashrc // config proxy if needed

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

* **开发工具(git, Intellij)**

// install git

$ sudo apt-get install git-core

$ git –version

// 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

* **开发工具包(jdk, spring, gradle, docker, .NET)**

// installjdk8 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

$ java -version

$ sudo apt-get install oracle-java8-set-default // configure java environment

* 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

export GRADLE\_HOME=/opt/gradle/gradle-1.5/bin

export PATH=$PATH:$GRADLE\_HOME

$ 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 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

export HADOOP\_HOME=/opt/hadoop-2.7.1

export PATH=$PATH:$HADOOP\_HOME/bin

$ source ~/.bashrc

$ hadoop version

开发具体内容见hadoop.docx

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

* **install angular-cli for linux**

download node-v8.1.0-linux-x64 & untar

测试可执行文件可以直接使用

$ cd node-v-8.1.0-linux-x64/bin

$ ./node -v

软链接，从而放入命令行搜索路径中

ln -s /home/q/Downloads/node-v8.1.0-linux-x64/bin/node /usr/local/bin/node

ln -s /home/q/Downloads/node-v8.1.0-linux-x64/bin/npm /usr/local/bin/npm

$ node -v && npm -v

$ sudo npm install -g angular-cli

若出现无法安装，改为执行

$ sudo npm install angular-cli

$ ln -s /home/qzlin/Documents/web-front/tools/node\_modules/@angular/cli/bin/ng

/usr/local/bin/ng

$ng -v #测试是否安装成功

* Install .NET on Linux

<https://docs.microsoft.com/en-us/dotnet/core/install/linux-ubuntu>

Install the SDK (which includes the runtime) if you want to develop .NET apps

If you only need to run apps, install the Runtime (ASP.NET Core Runtime includes .NET and ASP.NET)

$dotnet --list-sdks

$donet --list-runtimes

wget https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb

sudo dpkg -i packages-microsoft-prod.deb

rm packages-microsoft-prod.deb

sudo apt-get update; \

sudo apt-get install -y apt-transport-https && \

sudo apt-get update && \

sudo apt-get install -y dotnet-sdk-6.0

# linux shell脚本

## 开发流程，编辑器vim和运行

* **脚本开发流程**

$ vi test.sh #构建脚本文件

#!/bin/bash 定义shebang,告诉系统用哪个可执行文件来解释命令

脚本命令

建议使用#!/usr/bin/env bash, #!/usr/bin/env python因为如果bash或python可执行文件的安装目录并非/usr/bin,那么这个脚本就再也无法执行。而env命令行工具总能获知bash and python的安装目录，使用env会使脚本可移植性更好

$ chmod u+x test.sh #赋予执行权限

$ ./test.sh #执行脚本，用绝对或相对路径引用shell脚本文件

为什么test.sh不行，要./test.sh?

linux 系统会去 PATH 里寻找有没有叫 test.sh 的，而只有 /bin, /sbin, /usr/bin，/usr/sbin 等在 PATH 里，你的当前目录通常不在 PATH 里，所以写成 test.sh 是会找不到命令的，要用 ./test.sh 告诉系统说，就在当前目录找。

Shell如何执行一条命令？

1. 宏替换：执行之前，shell会查找其中的变量，通配符以及其他代词，如果有的话，就将它们进行替换
2. 执行替换后的命令

* **脚本编辑vim**

所有的 Unix Like 系统都会内建 vi /vim文本编辑器。

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。

三种模式：

命令模式(Command Mode)/普通模式: 执行命令

输入模式/插入模式(Insert Mode): 内容输入到缓冲区

命令行模式(Command-Line Mode)：缓冲区输出到文件

//启动vi/vim, **默认进入命令模式(按键解释成命令)**

$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

i // 按下i键，进入Insert**输入模式**，在光标当前位置开始输入文本。

a // 光标前进一格，进入输入模式

A // 光标前进行尾，进入输入模式

o/O // 在当前行下方(上方)插入新行，并进入输入模式。

r/R // 进入Replace mode, 取代光标所有字符一次/一直取代直到ESC

方向键 // 在文本中移动光标

n方向键 //如输入10然后按上方向键，即向上移动光标10行

HOME/END // 移动光标到行首/行尾

Page Up/Page Down //上/下翻页

gg // 移动光标到文档第一行

x // 删除当前光标所在处的字符。

dd // 删除当前行。

d$ // 删除光标至行尾

dG // 删除光标至文档末尾

dw // 删除单词

u // 撤销上一次操作。

Ctrl + r // 重做上一次操作。

p/P // 粘贴剪贴板内容到光标下方/上方。

Copy & Paste：

1. 移动光标(方向键)至起始位置，**按下v键(进入visual mode)，移动光标选择要复制的内容，按y键(激活copy)**

或移动光标至当前行，按yy 复制当前行。

1. 移动光标至目标位置，**按p(激活粘贴)**

Cutting & Paste:

1. 移动光标至起始位置，**接下v键(进入visual mode)，移动光标选择要剪切的内容，按d键(激活copy)**
2. 移动光标至目标位置，**按p(激活粘贴)**

查找：

/word // 从光标下查找word

按enter后会跳至第一个word，按n/N键跳至下一个/上一个

?word // 从光标之前查找word

替换：

:%s/old/new/g // 全文查找替换old -> new

:100,200s/old/new/g //在100到200行之间查找替换old -> new

:%s/old/new/gc // 替换前，需要用户确认

**// 在命令模式下，按下: 进入命令行模式**(即在最底一行输入命令)

:w // write,即保存文件，将缓冲区的数据保存到文件中

:q // quit,即退出 Vim 编辑器。

:q! // !是指强制，强制退出Vim 编辑器，不保存修改。

:w filename // save as功能

:wq // write and quit

ESC退出键 // 切换到命令模式

**// 在命令模式下，输入i, a/A, o/O等进入输入模式**

ENTER回车键 // 换行

BACK SPACE退格键 // 删除光标前一个字符

DEL删除键 // 删除光标后一个字符

方向键 // 在文本中移动光标

HOME/END // 移动光标到行首/行尾

Page Up/Page Down //上/下翻页

Insert // 切换光标为输入/替换模式，光标将变成竖线/下划线

ESC退出键 // 切换到命令模式

// 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

* **Shell是一种命令语言解析器**

记住：shell脚本主要用来操控简单的文件和命令，若脚本写得很复杂，特别涉及复杂的字符串或数学处理时，应该用python脚本语言

[[user@hostname](mailto:user@hostname) [user@hostname](mailto:user@hostname) 当前目录名] $

$表示可以被任何用户运行的命令

#表示应该以root用户运行的命令

Shell查找命令路径：$PATH

/home/qzlin/bin: 当前用户

/usr/local/bin: 所有普通用户

/usr/bin:

/bin:

/usr/local/sbin: 管理员

/usr/sbin:

/sbin:

路径目录顺序是很重要的，目录从左到右被检查

如何想要使命令对所有用户可用，则需要将它们添加到/usr/local/bin中

$ history 8 #查看历史命令列表

$ history | grep front-end-server

$ !n #运行命令编号，n表示命令行编号，由history获得

在关闭Shell之后，历史命令列表将被保存到~/.bash\_history文件中，最多保存1000条历史命令

$ cat /etc/passwd | sort | less #进程通过管道通信

$ date; …; date #连接命令

$ ./pycharm.sh & #后台命令

扩展命令

$vi `find /home | grep xyzzy` #编辑含xyzzy的文件

echo “There are $(ls | wc -w) files in this directory.”

## 语法：变量，条件判断，循环，可变数组，函数，函数库

* **变量**

1. 环境变量 (全局变量，所有终端窗口都可以访问)

保存操作系统的信息、状态和配置等，比如系统名称，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，

$ env #查看环境变量

$ set // 显示全部环境变量列表

$ $PATH //显示指定的环境变量

设置PATH环境变量来定义操作系统寻找可运行程序的路径

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

$ setenv PATH /usr/bin:/usr/local/bin:/sbin:/usr/sbin:$PATH //设置全局环境变量

使用setenv命令设置的环境变量的作用范围仅限于当前的终端窗口。如果需要设置全局的环境变量，可以将环境变量添加到/etc/rc.local或者/etc/profile等全局的配置文件中。

如果修改了环境变量后需要让其生效，可以在当前终端窗口中执行source命令，例如：

$ source /etc/profile

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

**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

$ unset test 删除变量

常用默认的shell环境变量：

$ echo $PATH #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

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**

1. 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=$guest **获取变量值需要使用$**

$ grep ‘r.\*t’ /etc/passwd #单引号表示字面量，不会被宏替换

$ echo “no \* in my path:$PATH” #shell会对双引号中的所有变量进行扩展

$ declare # 显示已定义的变量

* **条件判断**

**if** command**;** **then**

commands

**elif** command; then

commands

elif command; then

commands

else

commands

**fi**

**如果if关键字之后命令的退出状态码是0(表示该命令成功运行)，位于then部分的命令就会被执行**。

如果退出状态码非0，执行else or elif语句

if之后的内容必须是命令，所以需要分号然后then, 若没有分号，则then会被当成参数

若if后面是条件判断的表达式，需要用条件测试命令[] or test包装

condition可以是数值比较，文件串比较，文件比较，

**若condition成立，则条件测试命令返回退出码0 (表示条件判断成立)**

if [ condition]; then

commands

fi

逻辑结构&& and ||

if c1 **&&** c2; then #仅当命令c1和c2都执行成功

..

fi

if c1 **||** c2; then #命令c1或c2执行成功

..

fi

文件测试

if [ -e $file ] 检查文件或目录对象是否存在

if [ -d $file ] 检查指定的文件名是否是目录

if [ -f $file ] 检查指定的文件名是否是文件

if [ -h $file ] 检查指定的文件名是否是符号链接

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 $file ] 检查你是否是文件的属主

if [ $file1 –nt $file2 ] 检查某文件是否比另一文件更新

if [ $file1 –ot $file2 ] 检查某文件是否比另一文件更老

字符串测试 （字符串表示变量内容）

if [“$1” = hi] 若字符串相同，返回true

if [“$1” != hi] 若字符串不同，返回true

if [ -z “$val1” ] 若字符串为空，返回true, 相当于isEmpty(val1)

if [ -n “$val1” ] 若字符串非空, 返回true, 相当于!isEmpty(val1)

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

算术测试 双圆括号命令允许将高级数学表达式放入比较中

val=10

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

echo “greater”

fi

If [ $val -gt 0 ] 数值比较-eq(=), -ne(!=), -lt(<), -le(<=), -ge(>=)

**case** $var in

pattern1 | pattern2)

commands1

**;;**

pattern3)

commands2

;;

\*)

default commands

;;

**esac**

运行过程：

脚本将$var与每个)字符前的案例进行对比

如果匹配，就会执行该案例后的命令，直至遇到;;时，跳到esac关键字

整个条件判断结果以esac关键字结束

* **循环**

for var in list #注意list是以空格区分的列表

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” #以空格区分的字符串列表

do

echo $test

done

for state in `cat /tmp/file` #从命令输出取值， 以文件行读取

do

echo $state

done

for file in /home/\* #用通配符读取目录

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-- ))

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

do

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

do

echo “$i, $j”

done

done

for var in 1 2 3 4

do

if [ $var –eq 3 ]

then

break or continue #break命令 and continue命令同c语言

fi

done

for file in /home/\*

do

echo “$file”

done > output.txt #循环输出可以直接重定向到文件中,或管接到下一命令中

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

* **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

// 声明数组变量

$ declare -a IMAGES\_ALL=(\

"dicom-service" \

"mm-sim" \

"patient-list-service" \

"patient-list-ui-lib" \

"session-management-service" \

"tps-service"

)

$ IMAGES\_ALL[0]=hello 修改数组元素

$ echo ${IMAGES\_ALL[0]} 显示数组元素

$ echo ${IMAGES\_ALL[\*]} 显示整个数组

$ echo "${IMAGES\_ALL[@]}" 显示整个数组

function update\_all {

for IMAGE in "${IMAGES\_ALL[@]}"

do

update "$IMAGE"

done

}

* **函数**

// 函数定义, 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

function name {

commands

}

或者

name() {

commands

}

// 函数调用

name

// 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**

// 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 ./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source /home/rich/lib/fac.sh

## 常见命令: su/sudo, >, >>, <, >/dev/null, tee, grep, sed and awk

su [OPTION] [USER] // su指Switch User，切换当前用户的身份到其他用户

su - vagrant //切换到目标用户的登录shell，并获取其环境变量。

su //如果不指定目标用户，`su`命令默认会切换到超级用户（root）

`su`命令的功能有限，通常用于在非图形界面的终端中切换用户身份。对于图形界面的用户切换，建议使用其他工具，如`sudo`

sudo //指super user do, 允许普通用户执行部分root命令，如halt, reboot, su等

在Linux系统操作时，通常不会直接采用root用户。但当某些命令需要root权限执行时，往往会通过sudo命令提升当前用户的执行权限来完成。如果普通用户并未进行配置相应的配置，则在使用sudo命令时会提示“xxx 不在 sudoers 文件中。此事将被报告。“

用户是否拥有sudo命令的执行权限，要看是否在于 /etc/sudoers 文件进行了设置。

如果想要控制某个用户（或某个组用户）只能执行root权限中的一部分命令，或者允许某些用户使用sudo时不需要输入密码

格式：

授权用户/组 主机 =(切换到哪些用户或组) 是否需要输入密码验证 命令1, 命令2

字段1 字段2=(字段3) 字段4 字段5

字段1: 不以%开头的表示“将要授权的用户” root

以%开头的表示“将要授权的组”

字段2: 表示允许登录的主机，ALL表示所有

字段3: 如果省略相当于(root:root), 表示可以通过sudo提权到root

如果(ALL) or (ALL:ALL), 表示能够提权到（任何用户：任意用户组）

字段4: NOPASSWD: 表示执行sudo时不需要输入密码，只影响其后的第一条命令

字段5: 授权给用户的操作

e.g.,

jack mycomputer=/usr/sbin/reboot, /usr/sbin/shutdown

表示普通用户jack在主机（或主机组）mycomputer, 可以通过sudo执行reboot and shutdown

Lucy ALL=(ALL) NOPASSWD: /bin/useradd

表示普通用户luch可以在任何主机上，通过sudo执行/bin/useradd命令，并且不需要密码

**/etc/sudoers**

…

#includedir /etc/sudoers.d

任何在/etc/sudoers.d/目录下，添加/etc/sudoers的内容

**/etc/sudoers.d/designer**

%pinnacle ALL=(vagrant) NOPASSWD: /usr/local/pinnacle/bin/.designer-elev.sh

Defaults!/usr/local/pinnacle/bin/.designer-elev.sh env\_reset

Defaults!/usr/local/pinnacle/bin/.designer-elev.sh env\_keep+="DISPLAY"

Defaults!/usr/local/pinnacle/bin/.designer-elev.sh setenv

表示用户组pinnacle可以在任何主机(ALL)上，通过sudo提权到用户vagrant，执行命令

/usr/local/pinnacle/bin/.designer-elev.sh，且不需要输入密码(NOPASSWD)

Linux为了保证安全，sudo执行时，会创建一个小的执行环境，在这个执行环境中，只保留了一些必要的环境变量。导致经常出现的问题是sudo获取自定义环境变量失败。如何做？

Defaults! Command env\_reset 生成sudo执行环境时，将用户当前环境变量保留，不重置

Defaults! Command env\_keep+="DISPLAY" 添加环境变量DISPLAY

Defaults! Command setenv

output=`date` #命令的输出赋给变量。传统方法：反引号

today=$(date +%y%m%d) #命令的输出赋给变量。新方法：$()

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

// shell 输入输出重定向

command > output\_file #输出重定向

command >> output\_file #输出以追加方式重定向到output\_file

command < input\_file #输入重定向

$ command > /dev/null #不希望在屏幕上显示输出结果

# 同时替换输入和输出，执行command，从文件infile读取内容，输出到outfile

$ command < infile > outfile

$ tee file1 file2 # 标准输入 -> 标准输出和指定文件

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 << tag #将EOF之前的内容作为输入发送给command命令

数据

Tag

// 将一段文本写入到脚本文件

tee ping.txt > /dev/null <<ENDOFTEXT

ping 192.168.200.3

ENDOFTEXT

执行命令产生结果ping.txt, 文本内容是ping 192.168.200.3

// wrapper script

PINNACLE\_BIN\_DIR="/usr/local/pinnacle/bin"

DESIGNER\_LAUNCH\_SCRIPT="$PINNACLE\_BIN\_DIR/designer.sh"

DESIGNER\_LAUNCH\_ELEV\_SCRIPT="$PINNACLE\_BIN\_DIR/.designer-elev.sh"

sudo **tee** "$DESIGNER\_LAUNCH\_SCRIPT" > /dev/null <<ENDOFTEXT

su - vagrant $DESIGNER\_LAUNCH\_ELEV\_SCRIPT

ENDOFTEXT

执行命令产生结果

/usr/local/pinnacle/bin/designer.sh

… //原有内容

su - vagrant /usr/local/pinnacle/bin/.designer-elev.sh

command1 | command2 #管道

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。

exit 0 #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. 0表示成功

$ alias -p #查看已有的别名列表

$ alias li=’ls -il’ #创建命令别名

$man man #man命令，访问存储在linux系统上的手册页面

$man bash

…

# 目录切换命令

$ cd <work-dir>

//用于频繁切换两个目录

~$ pushd work-dir // 将目录work-dir压入栈中，然后切换到work-dir

~/work-dir ~

~/work-dir$ pushd // 退回到原来目录

~ ~/work-dir

~$ pushd // 切换到work-dir

~/work-dir ~

~/work-dir$ pop // 删除栈顶目录，退回到原来目录

~

~$

$ set -e //若后续bash命令返回code非0，脚本退出，后续脚本不执行

**grep, sed and awk**

awk -F fs '对每行执行命令'

逐行扫描文件(for i=1:N), 若行匹配成功，则在该行执行操作

awk 在读取一行文本时，会用**字段分隔符划分**每个数据字段，然后自动给一行中的每个数据元素分配一个变量。$0 代表整个文本行，$1 代表文本行中的第 1 个数据字段，$2 代表文本行中的第 2 个数据字段。

-F fs 批以fs作为输入行的分隔符，**默认分隔符为空格或制表符**

$ echo "My name is Rich" | awk '{$4="Christine"; print $0}'

将第4个字段Rich改为Christine, 然后打印整行

My name is Christine

$ cat /etc/passwd

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

adm:x:3:4:adm:/var/adm:/sbin/nologin

$ cat /etc/passwd | awk -F: '{print $1}' //逐行处理，按 :分隔，打印第1个字段

root

bin

daemon

adm

$ kubectl get pods -A

NAMESPACE NAME READY STATUS RESTARTS AGE

kube-logging es-cluster-0 1/1 Running 0 21m

kube-logging fluentd-mcz4r 1/1 Running 0 21m

kube-system coredns-5cfbb9f57c-5cgxb 1/1 Terminating 1 57m

kube-system coredns-5cfbb9f57c-lbwrc 1/1 Running 3 18m

testing mockoon-8659c45b55-mkv6j 1/1 Running 0 21m

# delete pods with Running status

$ kubectl get pod --all-namespaces | awk '{

if ($4 != "Running")

system ("kubectl -n " $1 " delete pods " $2 " --grace-period=0 " " --force ")

}'

# delete pods with Terminating status

$ kubectl get pv -A | awk '{

if ($5 == "Terminating")

system ("kubectl delete pv " $1 " --grace-period=0 " " --force ")

}'

sed [-nefri] [n1, n2] function

-n 仅处理的那几行显示到终端上

-i 直接修改读取的文件内容，而不是输出到终端

n1, n2 表示指定行范围

function: a 新增, c取代, d删除,i插入,p打印,s直接取代

$ sed -n ‘3p’ file # 取出文件的第3行

$ sed -n ‘2,5p’ file # 取出文件的第2-5行

$ sed -i ‘s/原字符串/新字符串/g’ file

-i 表示对文本文件直接进行操作

g表示全局替换

e.g.,

$ cat readme.txt

d

dd

$ sed -i ‘s/d/hello/’ readme.txt

hello

hellod

$ sed -i ‘s/d/hello/g’ readme.txt

hello

hellohello

$ sed -i ‘s/^@//’ file 去掉行首带@的首字母@

$ sed -i ‘/字符串/i新行字符串’ file 字符串前插入新行

$ sed -i ‘/字符串/a新行字符串’ file 字符串后插入新行

$ sed -i ‘/字符串/d file 字符串的删除

$ sed 3,6d file #3,6表示行的范围，d表示运算符delete

function update\_mmsim {

sudo sed -i "/^export PINN\_IMAGE\_TAG=/c\export PINN\_IMAGE\_TAG=$image"

/usr/local/pinnacle/bin/.env

}

在文件/usr/local/pinnacle/bin/.env查找export PINN\_IMAGE\_TAG=并替换成

export PINN\_IMAGE\_TAG=$image

## 数据输入(命令行,用户输入,文件读取) 和数据输出 (文件,临时文件,日志文件)

* **输入参数，解析命令行参数，获取用户输入，从文件中读入**

// 命令行输入参数

位置参数：分配给命令行输入的所有参数，每个参数要么空格分割，要么是引号分割

-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

$0是程序名，$1是第一个参数，… #位置参数变量

$# 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

// 解析命令行参数

在脚本中使用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

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

cat test | while **read** line #从文件test读取行，赋给变量line

do

echo “$line”

done

比如好的写法

**while read line**

do

echo "Sending invitation to **${line}**."

done **< data/emails.txt**

* **数据输出 (文件,临时文件,日志文件)**

文件描述符，阻止文件输出，临时文件，日志文件

// 文件描述符

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

对终端界面来说，标准输入是键盘

默认情况下，很多bash命令会定向输入到STDIN文件描述符

$ cat **<** testfile #**通过输入重定向符<，重定向指定的文件来替换输入文件描述符**

STDOUT (1：文件描述符) 标准输出

对终端界面来说，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

$ ls -l **>** testfile **#通过输出重定向符>，重定向到指定的重定向文件**

$ who **>>** testfile #将数据追加到某个文件

STDERR (2: 文件描述符) 标准错误

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

$ ls -al badfile **2>** errFile #重定向错误消息

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

echo “this is normal output” 标准输出

echo “This is an error message” >&2 #错误消息输出到STDERR文件描述符

$ lsof -a -p $$ -d 0,1,2 #列出打开的文件描述符

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，等效于and

// 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

$cat /dev/null > log #删除文件内容，但不删除文件

注：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

// 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

// 日志文件

$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

常见命令：yq, jq

Yq is yaml processor, jq is json processor

Example:

**update\_kube.sh**

#!/bin/bash

set -e

export DOCKER\_ART\_REPO="radonc-testing-docker-local.artifactory-ehv.ta.philips.com"

// 定义函数kc, 封装kubectl命令，"$@"为函数参数数组

function kc {

sudo NO\_PROXY="\*" kubectl --kubeconfig /etc/kubernetes/admin.conf "$@"

}

function update\_deployment {

docker login $DOCKER\_ART\_REPO

local IMAGE=$1

local DEPLOYMENT=$2

image="$DOCKER\_ART\_REPO/radonc/components/$IMAGE:edge"

// update docker image

docker rmi image|| true

docker pull image

# Retag and push from local registry to Nexus registry

// 获取docker inspect image的Id字段

hash=$(docker inspect --format="{{.Id}}" image | awk '{print substr($1, 8)}')

hash\_short=${hash: 0: 12}

docker tag "$hash" "repo-0:8072/radonc/components/$IMAGE:$hash\_short"

docker login repo-0:8072

docker push "repo-0:8072/radonc/components/$IMAGE:$hash\_short" || true

if [ -n "$DEPLOYMENT" ]; then

// update deployment

kc set image -n p3 deployment $DEPLOYMENT

"$DEPLOYMENT"="repo-0:8072/radonc/components/$IMAGE:$hash\_short"

kc patch deployment -n p3 $DEPLOYMENT

-p { "spec": {

"template": {

"spec": {

"containers": [{

"name": "$DEPLOYMENT",

"imagePullPolicy": "IfNotPresent"

}]

}}}}

kc rollout restart -n "$P3\_NAMESPACE" "deployment/$DEPLOYMENT-deployment"

fi

}

function update\_mmsim {

image=" $DOCKER\_ART\_REPO\/radonc\/components\/tps-system:$VERSION\_TAG"

sudo sed -i "/^export PINN\_IMAGE\_TAG=/c\export PINN\_IMAGE\_TAG=$image"

/usr/local/pinnacle/bin/.env

}

// map table

function update {

case $1 in

"dicom-service")

update\_deployment "$1" adac-dicom-service

;;

"mm-sim")

update\_deployment tps-system ""

update\_mmsim

;;

"patient-list-service")

update\_deployment "$1" "$1"

;;

\*)

echo "The service or deployment was not recognized"

;;

esac

}

function update\_all {

for IMAGE in "${IMAGES\_ALL[@]}"

do

update "$IMAGE"

done

}

function get\_login\_details {

if [ -z "$RT\_ARTIFACTORY\_API\_KEY\_USR" ];

then

echo "Code1 user:"

read -rs RT\_ARTIFACTORY\_API\_KEY\_USR

fi

if [ -z "$RT\_ARTIFACTORY\_API\_KEY\_PSW" ];

then

echo "Code1 password to connect to artifactory with user $RT\_ARTIFACTORY\_API\_KEY\_USR:"

read -rs RT\_ARTIFACTORY\_API\_KEY\_PSW

fi

}

// 脚本输入参数-s mm-sim -v edge -u 320029857 -p xxxxx

while getopts s:v:u:p option

do

case "${option}"

in

s)

IMAGE\_REQUEST=${OPTARG}

;;

v)

VERSION\_TAG=${OPTARG}

;;

u)

RT\_ARTIFACTORY\_API\_KEY\_USR=${OPTARG}

;;

p)

RT\_ARTIFACTORY\_API\_KEY\_PSW=${OPTARG}

;;

\*)

echo "Did not recognize the requested option."

echo "Available are:"

echo "-s item to update"

echo "-v tag to update to"

echo "-u Code1 username for artifactory"

echo "-p Code1 user password for artifactory"

echo "Update can be:"

for IMAGE in "${IMAGES\_ALL[@]}"

do

echo " $IMAGE"

done

exit 1

;;

esac

done

get\_login\_details // 要求输入Code1 and Password

if [ -z "$IMAGE\_REQUEST" ];

then

echo "No deployment selected, will update all."

**update\_all**

else

**update "$IMAGE\_REQUEST"**

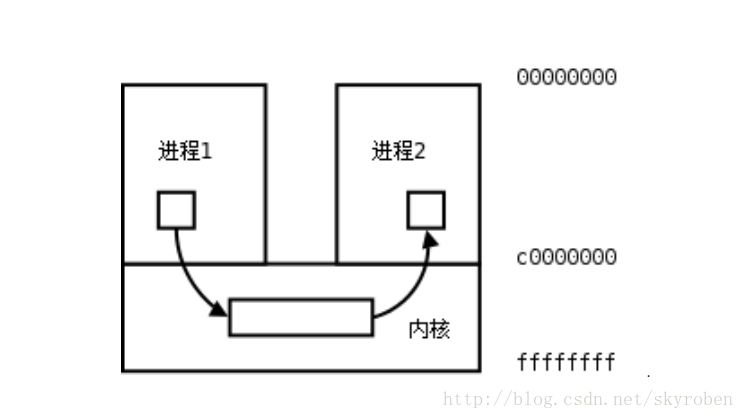
fi

# 系统编程

## 进程间通信

<https://www.cnblogs.com/imreW/p/17238050.html>

每个进程各自有不同的用户地址空间,任何一个进程的全局变量在另一个进程中都看不到，所以进程之间要交换数据必须通过内核,在内核中开辟一块缓冲区,进程A把数据从用户空间拷到内核缓冲区,进程B再从内核缓冲区把数据读走,内核提供的这种机制称为进程间通信。



不同进程间的通信本质：进程之间可以看到一份公共资源；而提供这份资源的形式或者提供者不同，造成了通信方式不同，而 pipe就是提供这份公共资源的形式的一种。

1. 管道的创建： 管道是由调用pipe函数来创建

#include <unistd.h>

int pipe (int fd[2]);

//返回:成功返回0，出错返回-1

// fd参数返回两个文件描述符,fd[0]指向管道的读端,fd[1]指向管道的写端。fd[1]的输出是fd[0]的输入。

1. 管道如何实现进程间的通信
2. 父进程创建管道，得到两个⽂件描述符指向管道的两端。
3. 父进程fork出子进程，⼦进程也有两个⽂件描述符指向同⼀管道。
4. 父进程关闭fd[0],子进程关闭fd[1]，即⽗进程关闭管道读端,⼦进程关闭管道写端（因为管道只支持单向通信）。⽗进程可以往管道⾥写,⼦进程可以从管道⾥读,管道是⽤环形队列实现的,数据从写端流⼊从读端流出,这样就实现了进程间通信。

#include <unistd.h>

int main(void)

{

int fd[2];

if (**pipe(fd)** == -1) sys\_err("pipe");

**pid\_t pid = fork();**

char \*p = "test for pipe\n";

if (pid < 0) { sys\_err("fork err");

} else if (pid == 0) {

printf("i am child begin\n");

**close(fd[1]);**

char buf[15];

**int len = read(fd[0], buf, sizeof(buf));** //子进程从管道里读数据

write(STDOUT\_FILENO, buf, len);

close(fd[0]);

printf("i am child end\n");

} else {

printf("i am parent begin\n");

**close(fd[0]);**

for (int i = 0; i < 10000; ++i) {

**write(fd[1], p, strlen(p));** //父进程往管道里写数据

}

wait(NULL);

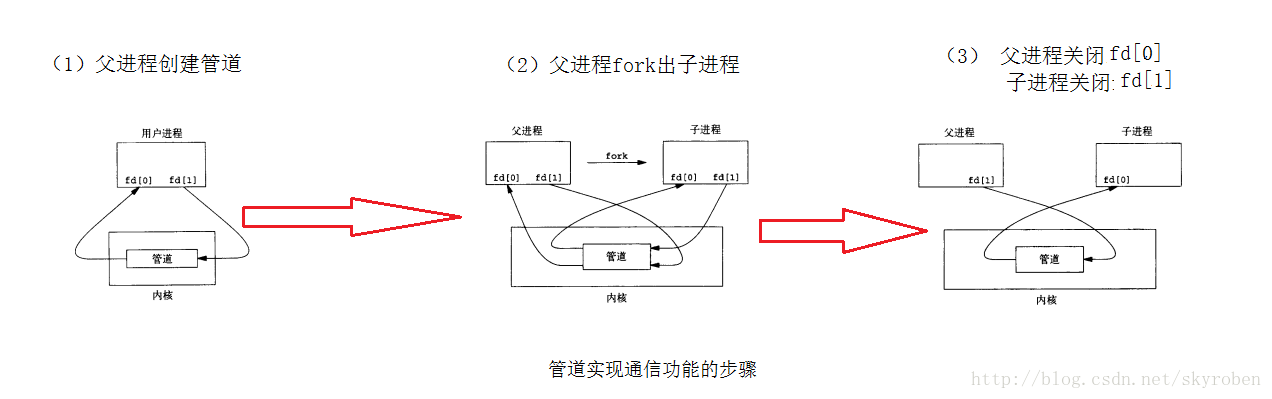
close(fd[1]);

printf("i am parent end\n");

}

return 0;

}



Appendix：

## Development Environment for Deep Learning

1. Reinstall Ubuntu20.04.

Settings -> Network Proxy

1. Install required tools

Firefox: ubuntu drivers

$sudo apt-get update

**$ubuntu-drivers devices**

$sudo ubuntu-drivers autoinstall

$sudo reboot

若无法联网，

$ sudo gedit /etc/apt/apt.conf

Acquire ::http ::Proxy "http://185.46.212.34:10015"

Acquire ::http ::Proxy ‘https:// 185.46.212.34:10015’

$bash Anaconda3… # Download and install anaconda

$./pycharm.py # download and install pycharm IDE

$**conda create -n your\_env\_name** **python** #其它包可以进虚拟环境再装

$ **conda activate your\_env\_name**

**(your\_env\_name)$ conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch**

若出现错误，一般是python版本与pytorch不兼容，找不到，降低python版本python=3.8

1. Window 访问ubuntu

方法a : 利用ubuntu20.04自带远程访问

<https://blog.csdn.net/xingyu97/article/details/111091528>

服务端：设置允许远程访问， 见链接网页

客户端window 安装vnc viewer, 输入ip

方法b :

服务端安装xrdp

客户端window远程桌面连接

以上远程访问速度比较慢，如果想实时操作，操作如下

<https://linuxize.com/post/how-to-install-and-configure-vnc-on-ubuntu-20-04/>

服务端安装，见链接网页

服务端启动：vncserver :2 -localhost no

客户端window安装tigervnc, 输入161.92.168.32:2

若需要服务员重启，直接$reboot, 然后过几分钟，用putty连接， 161.92.168.32

输入用户名和密码，然后在terminal 输入vncserver :2 -localhost no

window安装filezilla

(确保ubuntu 打开ssh-server)

$ sudo apt-get install openssh-server 安装ssh-server服务

1. Linux 分配虚拟内存(swap空间)

<https://blog.csdn.net/u014663203/article/details/84661807>

安装ubuntu

见Ubuntu桌面入门指南 <https://help.ubuntu.com/>

* usb+\*.iso 安装ubuntu, … 见ubuntu网站
* virtualBox + Ubuntu

// 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

FAQ:

* 无法卸载USB？

$ 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。