Linux 学习笔记-基于 Deepin 环境

目录

1 前言.....5
2 deepin 系统启动流程.....5
2.1 加载 BIOS.....5
2.2 读取主引导记录.....5
2.3 启动加载器.....5
2.4 加载内核.....6
2.5 初始化环境.....6
2.6 配置系统环境.....6
2.7 启动内核模块.....6
2.8 进入登录状态.....7
3 linux 文本编辑器 VIM 的使用.....7
4 Deepin 文件系统.....7
5 U 盘和硬盘挂载.....8
5.1 默认情况下的挂载.....8
5.2 如何永久性挂载某个驱动器到其他目录？.....8
6 软件源.....9
6.1 软件源配置文件.....10
7 软件包管理.....10
7.1 什么是包-package？.....10
7.2 软件包类型.....10
7.3 不确定软件包类型时，怎么?.....10
7.4 软件包的命名.....10
7.5 软件包的优先级.....10
7.6 查看软件包的状态.....11
7.6.1 查看当期系统所有已安装软件包.....11
7.6.2 查看某个软件的详细信息.....11
7.6.3 列出属于指定软件包的所有文件.....11
7.7 软件包依赖.....11
7.8 软件包安装方法.....12
7.8.1 本地安装.....12
7.8.2 网络安装.....12

7.9 常见问题.....12
8 增加字体种类.....13
9 如何加快系统启动速度.....14
9.1 分析那些服务导致启动速度慢.....14
10 desktop entry 文件.....15
11 Linux 命令.....16
11.1 常用基本命令.....17
11.1.1 查看文件信息命令.....17
11.1.2 清屏.....17
11.1.3 切换工作目录命令.....17
11.1.4 显示当前路径.....17
11.1.5 创建目录.....18
11.1.6 删除文件命令.....18
11.2 平时探索记录下的命令.....18
11.2.1 如何修改系统软件包下载源？.....18
11.2.2 查看系统启动时间.....18
11.2.2.1 查看系统启动时间.....18
11.2.2.2 分析系统各个服务组件启动耗时.....18
11.2.3 系统服务管理指令.....19
11.2.4 环境变量相关.....19
11.2.4.1 显示系统已经存在的环境变量.....19
11.2.4.2 管理环境变量.....19
11.2.4.2.1 用法 1.....19
11.2.4.2.2 用法 2.....20
11.2.5 输出一组系统信息.....21
12 Deepin 服务器的远程管理.....21
12.1 服务器端配置.....21
12.1.1 配置文件.....21
12.1.2 服务器的启动.....22
12.1.3 客户端远程登录访问.....22
13 Deepin 远程协助.....22
13.1 使用商店安装.....22
13.2 使用终端安装.....22
14 一些重要的环境变量的定义位置.....23
14.1 用户级变量.....23
14.2 对所有用户有效的变量(系统级）.....23
15 linux shell 脚本基础.....23
15.1 基础概念.....23
15.1.1 什么是 linux 脚本？.....23
15.1.2 linux 脚本有什么用？.....23

15.1.3 脚本中的指令执行流程.....23
15.1.4 脚本解释器是什么？.....24
15.2 基本语句.....24
15.2.1 变量定义语句.....24
15.2.1.1 变量的种类.....24
15.2.1.2 语法格式.....25
15.2.1.3 表达式中常用算术运算符.....27
15.2.1.4 变量定义语句用法示例.....27
15.2.1.5 临时环境变量定义.....28
15.2.1.5.1 示例.....28
15.2.1.6 表达式中的内置变量.....28
15.2.2 数组定义语句.....28
15.2.2.1 语法.....28
15.2.2.2 示例.....29
15.2.3 关联数组定义语句.....29
15.2.3.1 语法.....30
15.2.3.2 示例.....30
15.2.4 赋值语句.....31
15.2.4.1 给用户变量赋值.....31
15.2.4.2 给环境变量赋值.....31
15.2.4.3 将命令结果赋给变量.....31
15.2.5 管理变量语句.....32
15.2.5.1 删除变量语句.....32
15.2.5.2 设变量只读.....32
15.2.6 linux 命令语句.....32
15.2.6.1 它是什么.....32
15.2.6.2 常用命令语句.....32
15.2.6.2.1 输入命令语句.....33
15.2.6.2.2 输出语句.....33
15.2.6.2.3 test 命令语句.....33
15.3 流程控制.....33
15.3.1 关系运算符.....34
15.3.2 逻辑运算符.....34
15.3.3 逻辑表达式.....35
15.3.3.1 中括号表示法.....35
15.3.3.2 test 函数表示法.....35
15.3.4 分支流程.....35
15.3.4.1 if 分支.....35
15.3.4.2 case ... esac 多分支.....36
15.3.4.3 示例.....36

15.3.5 循环流程.....38
15.3.5.1 for 循环.....38
15.3.5.2 while 循环.....38
15.3.5.3 util 循环.....38
15.3.5.4 无限循环.....38
15.3.6 示例.....39
15.4 函数定义与调用.....40

1 前言

本文是给从 windows 转向 linux 的学习者的材料，是本人学习时的经验总结（许多部分来自于互联网，版权归原作者所有），绝大部分的用法都经过了测试，测试系统为 deepin 操作系统，适用于其他 linux 发行版（如 UOS 等）。

在 1515 的逻辑结构和组织方式与这方面传统的逻辑结构有所不同，即：基本语句➡流程控制➡模块（这里是函数），这是所有编程语言的通用知识结构笔者认为，按照这种逻辑结构学习，学习者的某种编程语言经验，可以更快更顺利地迁移到新的编程语言。

因水平和时间都有限，难免存在疏漏（如错别字），请多多包涵。

2 deepin 系统启动流程

指的是从计算机加电到显示登录界面的过程。

2.1 加载 BIOS

bios 里存储了系统的硬件信息，知道从哪里开始（如硬盘、CPU 等等），最后 BIOS 将系统的控制权交给硬盘的第 1 个扇区后，linux 就来控制系统了。

2.2 读取主引导记录

磁盘上第 0 磁道第 1 个扇区，就是 MBR（Master Boot Record）它的大小是 512B，里面存储了预启动、分区表信息。分为两个部分：

（1）Pre-BOOT 分区，446 个字节

（2）分区表，64B，记录硬盘的分区信息

预引导分区的作用是找到活动（Active）分区的引导区，读入内存。

BIOS 帮助找到 MBR 后，就会将其复制到 0x7c00 地址所在的物流内存，被复制到的物理内存中的内容就是 Boot loader，具体到电脑，就是 lilo 或 grub 了。

2.3 启动加载器

Boot Loader 是操作系统内核运行之前的一道小程序（初始化硬件设备，建

立内存空间映射图、将系统硬件设备带到一个合适的状态，为最终调用操作系统内核做好准备）。

linux 的引导扇区的内容使用汇编写的程序。源代码在 arch/i386/boot 中（不同体系的 CPU 有其各自的 boot 目录）

加载器有好几种类型，Grub、Lilo、spfdisk。系统读取内存中的 grub 信息，一般为 menu.lst 或 grub.lst,用于配置启动不同的操作系统。

2.4 加载内核

根据 grub 所设定的内核映像所在路径，解压缩并放入内存。然后调用 start_kernel()函数（在 init/main.c 中定义），启动初始化工作。

2.5 初始化环境

内核加载后，运行程序/sbin/init,该程序将读取/etc/inittab,依据次文件初始化。

inittab 作用：设定登记。

5 个等级

0：关机

1：单用户模式

2：无网络支持的多用户模式

3：有网络支持的多用户模式

4：保留，未使用

5：有网络支持有 X-Window 支持的多用户模式

6：重新引导系统，即重启

2.6 配置系统环境

执行，/etc/rc.d/rc.sysinit 脚本程序

2.7 启动内核模块

具体是依据 /etc/modules.conf 文件或 /etc/modules.d 目录下的文件来装载内

核模块。

2.8 进入登录状态

执行 /bin/login 程序，等待用户输入账号。

3 linux 文本编辑器 VIM 的使用

作为 linux 开发者，如果不知道文本编辑器 vim 的使用，就如同外国人来到中国，吃中国菜，却不会使用筷子一样。

在许多需要编辑文本文件（特别是配置文件）的场景中，需要使用 vim 进行文本文件的创建、编辑与保存。

入门 vim，不需要专门的书籍，只需要打开 deepin 终端，键入如下命令，即可进入“在做中学的环境”，花个 30 分钟的时间，就能基本搞定 vim。

$ vimtutor

我认为，还没有哪一个资料比得上这个官方教程的有效性了。

4 Deepin 文件系统

Deepin 默认文件系统为 Ext4（第 4 代可扩展文件系统）.

查看当前系统分区命令：

df -Th

```
imagan@imagan-PC:~$ df -Th
文件系统        类型      容量  已用  可用 已用% 挂载点
udev            devtmpfs  3.8G     0  3.8G    0% /dev
tmpfs           tmpfs     786M  1.7M  784M    1% /run
/dev/sda5       ext4      239G   23G  205G   10% /
tmpfs           tmpfs     3.9G   76M  3.8G    2% /dev/shm
tmpfs           tmpfs     5.0M  4.0K  5.0M    1% /run/lock
tmpfs           tmpfs     4.0M     0  4.0M    0% /sys/fs/cgroup
tmpfs           tmpfs     786M  108K  786M    1% /run/user/1000
/dev/sda3       fuseblk   157G   53G  105G   34% /media/imagan/1098145410981454
/dev/sda4       fuseblk   370G   85G  285G   23% /media/imagan/109A1018109A1018
/dev/sda2       fuseblk   162G   34G  129G   21% /media/imagan/Data
/dev/nvme0n1p4  fuseblk   995M  538M  458M   55% /media/imagan/WinRE_DRV
/dev/nvme0n1p3  fuseblk   118G   84G   35G   71% /media/imagan/Windows
```

左边的文件系统这一列，类似于 windows 的分区名称，类型即为该分区的文件系统类型，/为根分区，所有其他分区都是以树形目录的形式挂载到根分区下的。

udev 是设备节点。

tmpfs,临时内存文件系统

5 U 盘和硬盘挂载

5.1 默认情况下的挂载

默认情况下，都是以目录的形式挂载到“/media/(用户名)/外部磁盘设备的名称或 uuid”目录。

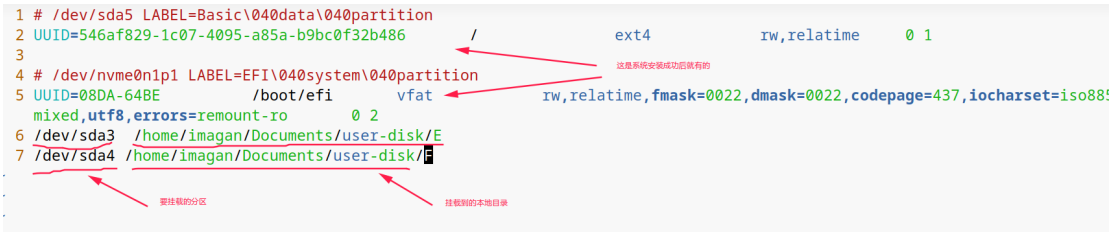
5.2 如何永久性挂载某个驱动器到其他目录？

使用 mount 和 unmount 能实现启动后的硬盘驱动器的挂载和卸载，但重启系统后，挂载失效。如何实现永久性挂载：

其实只需要修改文件“/etc/fstab”，即在该文件末尾添加若干行如下格式的挂载信息即可：

要挂载的分区　挂载的本地目录

如下就是示例：



这些要挂载的分区路径如何查看？使用如下命令：

df -h

将显示类似如下结果：

```
imagan@imagan-PC:~$ df -h
文件系统        容量  已用  可用 已用% 挂载点
udev            3.8G     0  3.8G    0% /dev
tmpfs           786M   89M  697M   12% /run
/dev/sda5       239G   25G  203G   11% /
tmpfs           3.9G   12M  3.9G    1% /dev/shm
tmpfs           5.0M  4.0K  5.0M    1% /run/lock
tmpfs           4.0M     0  4.0M    0% /sys/fs/cgroup
/dev/sda3       157G   53G  105G   34% /home/imagan/Documents/user-disk/E
/dev/sda4       370G   85G  285G   23% /home/imagan/Documents/user-disk/F
tmpfs           786M   96K  786M    1% /run/user/1000
/dev/sda2       162G   34G  129G   21% /media/imagan/Data
/dev/nvme0n1p3  118G   84G   35G   71% /media/imagan/Windows
imagan@imagan-PC:~$
```

插图 1 查看 UUID 命令

左边的文件系统就是可以挂载的分区名称。

要使用 UUID 挂载各个磁盘分区，需要查询 UUID，使用如下命令即可查询

$blkid

```
imagan@imagan-PC:~$ df -h
文件系统        容量  已用  可用 已用% 挂载点
udev            3.8G     0  3.8G    0% /dev
tmpfs           785M   26M  760M    4% /run
/dev/sda5       239G   35G  193G   16% /
tmpfs           3.9G   77M  3.8G    2% /dev/shm
tmpfs           5.0M  4.0K  5.0M    1% /run/lock
tmpfs           4.0M     0  4.0M    0% /sys/fs/cgroup
/dev/sda4       370G   89G  281G   24% /home/imagan/Documents/user-disk/F
/dev/sda3       157G   53G  105G   34% /home/imagan/Documents/user-disk/E
tmpfs           785M   88K  785M    1% /run/user/1000
/dev/sda2       162G   34G  129G   21% /media/imagan/Data
/dev/nvme0n1p3  118G   86G   33G   73% /media/imagan/Windows
imagan@imagan-PC:~$ blkid -L
blkid: 选项需要一个参数 -- L
Try 'blkid --help' for more information.
imagan@imagan-PC:~$ blkid
/dev/nvme0n1p1: LABEL="SYSTEM" UUID="08DA-64BE" TYPE="vfat" PARTLABEL="EFI system partition" PARTUUID="b0a34b52-9086-45fb-9c3c-5138ccbcb745"
/dev/nvme0n1p3: LABEL="Windows" UUID="7A8EDD068EDCBBB7" TYPE="ntfs" PARTLABEL="Basic data partition" PARTUUID="13f33812-57a0-41a6-bda8-97b651c2380b"
/dev/nvme0n1p4: LABEL="WinRE_DRV" UUID="E69CDDC99CDD9501" TYPE="ntfs" PARTLABEL="Basic data partition" PARTUUID="2b61c5e1-ef58-4e5a-8e68-964f759407f6"
/dev/sda2: LABEL="Data" UUID="B2E0E0B0E0E07C4B" TYPE="ntfs" PARTLABEL="Basic data partition" PARTUUID="7e273d01-4dcf-11eb-97f5-e96593f20baa"
/dev/sda3: UUID="1098145410981454" TYPE="ntfs" PARTLABEL="Basic data partition" PARTUUID="7e273d2b-4dcf-11eb-97f5-e96593f20baa"
/dev/sda4: UUID="109A1018109A1018" TYPE="ntfs" PARTLABEL="Basic data partition" PARTUUID="962ddb62-1f68-11ed-9882-a4ae127e821c"
/dev/sda5: UUID="546af829-1c07-4095-a85a-b9bc0f32b486" TYPE="ext4" PARTLABEL="Basic data partition" PARTUUID="962ddb63-1f68-11ed-9882-a4ae127e821c"
imagan@imagan-PC:~$
```

6 软件源

linux 提供的免费应用程序安装仓库。按类型分为：

软件仓库：包含软件的二进制包和源码

ISO 仓库：发行版的 ISO 文件。

软件源包括：网络服务器、光盘、本地目录。

6.1 软件源配置文件

软件源配置文件为：/etc/apt/sources.list,里面写了软件源地址。

可以对其进行编辑：

sudo edit /etc/apt/sources.list

7 软件包管理

7.1 什么是包-package？

linux术语，类似于windows的程序、库或工具，包是linux上所有可由用户启动或其他程序调用的所有程序及其文档的总称。

7.2 软件包类型

deepin下保护两种类型的软件包：

（1）二进制软件包：包含可执行文件，库文件，配置文件、man/info页面、版权声明、其他文档。

（2）源码包：包含软件源代码、版本修改说明、构建指令以及编译工具等。先由tar工具归档为.tar.gz文件，然后再打包成.dsc文件。

7.3 不确定软件包类型时，怎么？

用户不确定一个软件包具体类型时，可以使用file命令查看文件类型。

7.4 软件包的命名

约定：

```
Filename_Version-Reversion_Architecture.deb
```

Filename：软件名

Version:版本号

Reversion:修订版本号

Architecture:适用的计算机架构

以g++_4.1.2-9ubuntu2_i386.deb软件包为例，g++是软件包名，4.1.2是软件版本号，9ubuntu2是修订版本号，i386是适用的计算机架构。

7.5 软件包的优先级

Deepin为每个软件包指定了优先级，高优先级的软件包不能依赖于低优先

级的软件包。可以实现按照优先级由高到底层层冻结系统，实现系统的维护。

优先级：Required 级>Important 级>Optional>Extra

7.6 查看软件包的状态

描述的是用户安装、卸载软件包的行为结果，linux 系统对于软件包的状态，由两种类型的描述：

期望状态：用户期望某个软件包的状态；

当前状态：标记用户操作该软件包后的最终状态。

7.6.1　查看当期系统所有已安装软件包

简明扼要地列出本系统已安装的所有软件包

dpkg -l

7.6.2　查看某个软件的详细信息

aptitude show package_name

例如：

aptitude show gcc

7.6.3　列出属于指定软件包的所有文件

我们希望知道安装了某个软件后，它到底包含哪些文件，这个在 windows 系统中比较好办，但在 linux 系统，需要使用如下命令：

dpkg -L package_name

7.7 软件包依赖

依赖性是描述当前软件与系统中已经存在的软件之间联系的紧密程度。软

件包管理器根据依赖关系完成组件的安装核卸载。

例：包含GNU C编译器的软件包依赖于包含链接器和汇编器的软件包binutils，若在没有安装号bintils软件包的情况下，安装GNU C编译器，就会发生中断，并提示用户先安装好binutils。

7.8 软件包安装方法

分为本地安装和网络安装。

7.8.1　本地安装

本地安装指的是用户先下载程序的二进制包（包括压缩文档或deb文件等），然后使用本地安装程序进行安装。

deepin提供了本地安装命令：

sudo dpkg -i deb包

若要同时安装一个目录下的所有deb包，如何安装？

sudo dpkg -i *.deb

7.8.2　网络安装

sudo apt-get install 软件包名

网络安装，需要软件包源的确实有那个软件包，其设置方法，请参考“6 软件源”。

7.9 常见问题

在深度操作系统源上，提供了几乎所有的deb格式的软件包，供用户下载和使用。

不推荐使用Rpm转换成的deb包进行安装，因为存在依赖问题。

一定要转换，则先安装“alien”：

sudo apt-get install alien

转换命令：

alien XXX.rpm

该命令将在 XXX.rpm 目录下产生 XXX.deb 软件包。

8 增加字体种类

deepin 系统自带字体太少，例如无宋体、黑体等等。如何把 windows 系统中的字体引入 deepin 系统，是一个值得探讨的问题。

以下假设你用的是双系统，你进入 linux 系统后，能看到 windows 系统的文件夹。

操作步骤：

（1）启动字体管理器，如下图所示。



（2）单击界面上的“+”按钮，浏览到“原 windows 系统盘/windows/fonts”目录，选择所有的文件，即可。

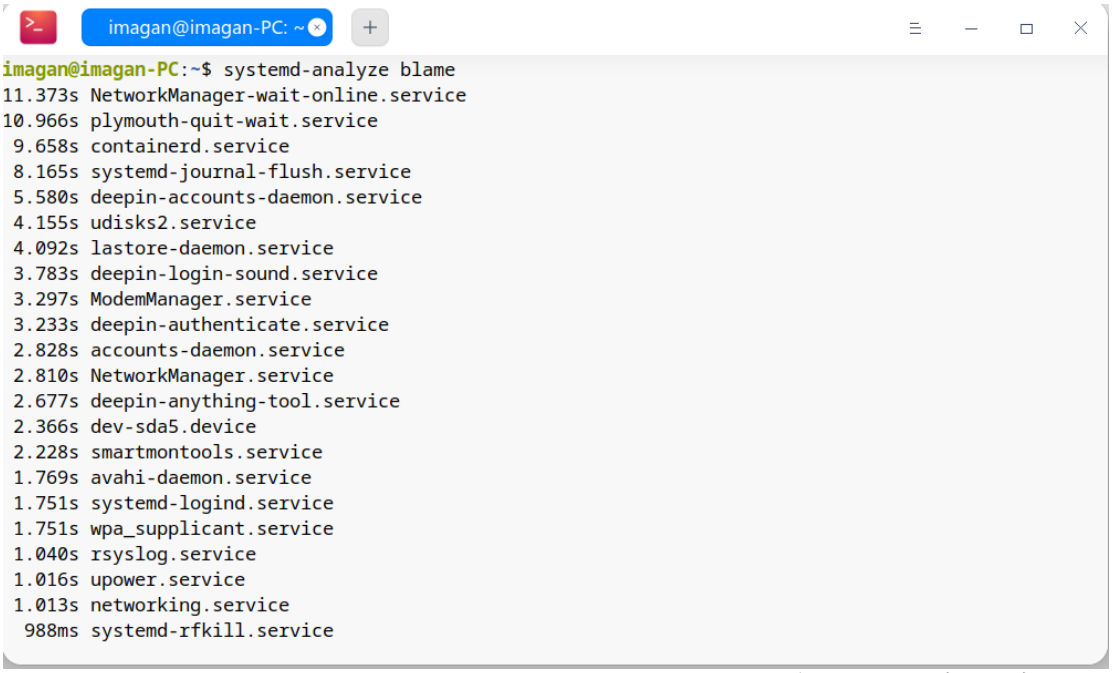
9 如何加快系统启动速度

9.1 分析那些服务导致启动速度慢

使用：systemd-analyze 进行分析。

命令：systemd-analyze blame

该命令可以显示启动时各个服务所耗费的时间。



从中可见，Network...这一项耗时最大，是否可以禁止，要查一查，是否为系统启动关键服务，若不是，则改为手动启动。

要禁用一个服务，可以使用：

sudo systemctl disable 服务名称

这里的服务名称就是上面看到的第 2 列的字符串。

若要启动，则 sudo systemctl enable 服务名称

若要完全禁用某个服务，则可以：

sudo systemctl mask 服务名称

也可以通过 deepin 自带的系统监视器对某个服务进行禁用或改为手动。

10 desktop entry 文件

用于配置程序启动方式的文件，它相当于 windows 中的快捷方式。

典型的 desktop entry 文件的配置如下：

[Desktop Entry] #每个 desktop 文件都以这个标签开始，说明这是一个 Desktop Entry 文件
Version = 1.0 #标明 Desktop Entry 的版本（可选）
Name = Firefox #程序名称（必须），这里以创建一个 Firefox 的快捷方式为例
GenericName = Web Browser #程序通用名称（可选）
Comment = A Web Browser #程序描述（可选）
Exec = firefox %u #程序的启动命令（必选），可以带参数运行,当下面的 Type 为 Application，此项有效
Icon = firefox #设置快捷方式的图标（可选）
Terminal = false #是否在终端中运行（可选），当 Type 为 Application，此项有效
Type = Application #desktop 的类型（必选），常见值有“Application”和“Link”
Categories = GNOME;Application;Network; #注明在菜单栏中显示的类别（可选）

注意：desktop 文件必须具有可执行权限才行。

一个完整示例：

[Desktop Entry]

Version=1.0

Name=Firefox Web Browser

Name[zh_CN]=Firefox 浏览器

Comment=Browse the World Wide Web

Comment[zh_CN]=浏览互联网

GenericName=Web Browser

GenericName[zh_CN]=网络浏览器

Exec=/home/abi/下载/firefox/firefox

Terminal=false

X-MultipleArgs=false

Type=Application

Icon=/home/abi/下载/firefox/icons/mozicon128.png

Categories=GNOME;GTK;Network;WebBrowser;

MimeType=text/html;text/xml;application/xhtml+xml;application/xml;application/vnd.mozilla.xul+xml;application/rss+xml;application/rdf+xml;image/gif;image/jpeg;image/png;x-scheme-handler/http;x-scheme-handler/https;x-scheme-handler/ftp;x-scheme-handler/chrome;video/webm;

StartupWMClass=Firefox

StartupNotify=true

X-Ayatana-Desktop-Shortcuts=NewWindow;

[NewWindow Shortcut Group]

Name=Open a New Window

Name[zh_CN]=新建窗口

Exec=firefox -new-window about:blank

TargetEnvironment=Unity

11 Linux 命令

指的是在终端中可以执行的命令。

11.1 如何获取命令的用法

```
$ command_name  --help
```

注意，上面的 help 前面两个短横线。

例如，要查看“apt-get”用法，可以使用如下命令：

apt-get --help

个别命令提供了更友好的帮助信息，可以使用如下命令查看帮助：

$ man command_name

11.2 常用基本命令

11.2.1 查看文件信息命令

ls [参数] [目标目录或文件]

[]表示可选。

可选参数有：

（1）-a :显示所有子目录和文件（包括隐藏文件（以.开头的文件））

（2）-l:以列表方式显示详细信息

（3）-h -l:以人性化方式显示目标文件或目录的信息（包括文件大小）

注意：目录或文件名可以包含通配符（*？等）

11.2.2 清屏

clear

11.2.3 切换工作目录命令

cd

cd ~ :切换到当前用户主目录

cd . :切换到当前目录

cd .. :切换到上一级目录

cd _ :切换到上一次目录

11.2.4　显示当前路径

pwd

11.2.5　创建目录

mkdir

11.2.6　删除文件命令

rm [参数] [待删除的文件或目录]

参数：

（1）-i，交互式方式执行

（2）-f,强制删除

（3）-r 递归第删除某个目录下的一切，删除文件夹时需要带此次参数

11.3 平时探索记录下的命令

11.3.1　如何修改系统软件包下载源？

sudo vim /etc/apt/sources.list

这个 sources.list 文件存放了软件包下载源。

11.3.2　查看系统启动时间

11.3.2.1 查看系统启动时间

systemd-analyze

11.3.2.2 分析系统各个服务组件启动耗时

systemd-analyze blame

11.3.3　系统服务管理指令

systemctl

11.3.4　环境变量相关

11.3.4.1 显示系统已经存在的环境变量

env [选项] [参数]

选项：

值	说明
-i	开始一个新的空的环境
-u<变量名>	从当前环境中删除指定的变量

参数：

变量定义参数：变量名=值

指定：指定要执行的指令和参数

11.3.4.2 管理环境变量

11.3.4.2.1　　用法 1

export [-fn] [name[=word]]...

-f 指向函数

-n 删除变量的导出属性，所有导出属性指的是存在该变量，且它的值可读。

name：表示变量名

word：表示值

示例：

export -fn fun1 #导出脚本中定义的函数fun1

export m=10 #修改变量m的值为10，若m不存在，则创建一个新的变量，并赋给初始值10

首先删除要演示的函数名

unset func_1 func_2

创建函数

function func_1(){ echo '123'; }

function func_2(){ echo '890'; }

为已定义函数增加导出属性

export -f func_1 func_2

删除函数的导出属性

export -fn a b

添加环境变量（JAVA）到`~/.bashrc`

PATH=/usr/local/jdk1.7.0/bin:$PATH

添加当前位置到动态库环境变量

export LD_LIBRARY_PATH=$(pwd):${LD_LIBRARY_PATH}

11.3.4.2.2　用法2

export -p

显示全部拥有导出属性的变量。

11.3.5　输出一组系统信息

使用命令：

uname -a

该命令可输出操作系统/硬件架构等信息。

12 Deepin服务器的远程管理

假设我想使用我的电脑远程管理公司的一台linux服务器，怎么实现？

上述任务需要两方面的配合：

（1）服务器端（要管理的服务器）：安装了服务器程序，且程序在运行状态；

（2）我的电脑端（客户端）：知道服务器程序运行的端口、服务器的ip地址、服务器的账户信息。

上面所说的服务器程序，有很多种，当前，最简单的一种是ssh-server。Deepin 20.X版自带了这个软件，且在其终端程序中集成了远程管理的功能，因此它默认具备远程ssh登录等功能，用户仅仅需要对服务器端进行配置，并使用客户端深度终端工具，即可完成远程管理。

12.1 服务器端配置

12.1.1　配置文件

SSH服务器程序的名称叫做：OPEN_BSD，其运行时的参数通过配置文件

“/etc/ssh/sshd_config”进行指定，具体意义，请参考该文件。

若要修改，建议备份该文件进行修改。

12.1.2　服务器的启动

默认情况下，服务器的ssh-server是没有启动的，因此远程无法登录它进行管理。

设置服务器自启动（开机启动）：

```
systemctl enable ssh
```

注：上述操作也可以通过Deepin自带的“系统监视器”的GUI界面进行操作，其中ssh服务就是对应的程序

启动服务器程序：

```
systemctl restart sshd
```

12.1.3　客户端远程登录访问

客户端的深度终端集成了远程管理功能，具体请参考用户手册。

13 Deepin远程协助

远程协助，指的是网络中的两台电脑中的一台主动发起请求（提供协助方），请求对另一台电脑（获得协助方）进行控制，协助另一台电脑操作者接近他在使用过程中遇到的问题。

远程协助中，获得协助方和提供协助方都要安装远程协助软件。

13.1 使用商店安装

略

13.2 使用终端安装

sudo apt-get install uos-remote-assistance

具体用法，请参考相关帮助。

14 一些重要的环境变量的定义位置

14.1 用户级变量

~/.bashrc

14.2 对所有用户有效的变量(系统级）

/etc/profile

15 linux shell脚本基础

15.1 基础概念

15.1.1 什么是linux脚本？

用来指示操作系统进行一序列语句定义的操作的文本文件。linux自带用C语言编写的Shell（外壳程序）可以读入linux脚本文件并依次执行里面的操作。

linux脚本必须根据linux shell语法规则编写，才能被shell解析器解释执行。

15.1.2 linux脚本有什么用？

（1）脚本是shell能识别并自动执行的一组指令，因此，可以实现linux主机管理的自动化。

（2）linux shell脚本在linux内核、用户、用户程序之间可以建立桥梁，让用户或用户程序更好地使用linux系统。

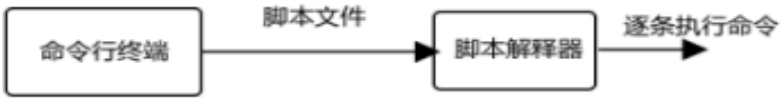
（3）...

15.1.3 脚本中的指令执行流程

脚本解释器读取脚本→按基本语句或语句块（循环语句块、分支语句块）出现的先后执行，其中语句块内部执行顺序与语句块对应的流程控制规范有关。

15.1.4　脚本解释器是什么？

用来读取和运行脚本命令的程序。



与编译器类似，脚本解释器也有不同的开发者开发的产品序列。

目前，linux 下最流行的还是：Bourne Again Shell（/bin/bash）（大多数 Linux 系统默认的 Shell）

15.2 基本语句

脚本由基本语句、语句块（流程控制块、函数定义块）构成，语句块也是由基本语句构成，因此，基本语句是构成脚本的最小功能单位。基本语句具体形式千变万化，但万变不离其宗，基本语句的类型只有有限的若干种。因此，学好基本语句是基础。每一条语句占据一行。

15.2.1　变量定义语句

15.2.1.1 变量的种类

环境变量：为系统内核、命令或应用程序运行提供的预定义的变量，不用重新定义。

内部变量：bash 中内置变量，是固定不变的。

用户变量：用户在脚本中定义的变量，又可分为局部和全局变量。

环境变量、内部变量都是预定义的变量，不需要用户重新定义，只需要使用。因此这里重点介绍用户变量的定义。

15.2.1.2 语法格式

变量名=表达式

变量名只能包含字母、数字、下划线，不能以数字开头。

表达式是能够最终评估为某个字符串的式子。

“=”左右两边不能有空格。

表达式可以包含“常量”、“已定义变量”、“运算符”、“返回值为字符串的函数”等要素的式子。

若表达式只包含常量，则常量可以加双引号，也可以不加双引号,总是对应字符串。例如，以下三个变量初始值相等。

a=1

b=”1”

c=’1’

单引号常量中所有子字，都是按原样编码；

若表达式包含已经定义的变量（假设该变量名为“A”），意味着要把那个已定义的变量的值作为表达式的一部分，则“A”应该写作：

$A 或 ${A}

这里的A可以是预定义的环境变量、shell中的内置变量或前面已定义的用户变量。

单引号常量中所有字符，都是按原样编码；而双引号常量中的${变量名}，将以该变量的值取代。

若表达式中存在运算符，且希望运算符及其对应的操作数构成的整体（称为数学表达式）具有运算意义，则需要使用特殊符号把数学表达式包括起来，

否则运算符会被当做普通的字符。常见的几种用于数学表达式计算的特殊符号为：

形式 1：$((数学表达式))

形式 2：$[数学表达式]

形式 3：`expr 数学表达式`

形式 4：let 数学表达式

注意，上面的操作只能针对整数进行运算。

若要针对小数，需要用到 bc 或 awk 计算器程序，具体用法要参考帮助。这里以 bc 为例，其用法示例如下：

```
1 #!/bin/bash
2 a=23.5
3 echo $a
4 b=13.7
5 echo $b
6 c=`echo "scale=3;$a+$b" | bc`
7 echo $c
8 d="$a+$b" | bc
9 echo $d
10
```

注意，第 9 行的代码没有实际作用，仅仅在 bc 的进程中执行了 23.5+13.7 操作，可是没有把这个操作重定向到当前终端输出。因此，d 并没有获得结果。要实现这一点，需要使用 echo,把结果输出到终端，若希望把 echo 输出到终端的结果存入一个变量，则需要使用``把它引用起来，如第 6 行代码所示。这里的 scale=3 用来设定小数点后最多保留多少位有效数字。

要访问环境变量的值，可以使用：$环境变量名。

环境变量查看命令：

env

15.2.1.3 表达式中常用算术运算符

算术运算符：+（加）、-（减）、*（乘）、/（除）、**（乘幂）、%（求余数）

15.2.1.4 变量定义语句用法示例

```
#!/bin/bash
a=10 #定义一个变量a
b="19" #定义一个变量b
echo $a #输出a的值
echo $b #输出b的值
c=a+b #定义一个变量c，其初始值为字符串"a+b"
d=$((a+b)) #定义一个变量d，其初始值为数学表达式a+b的结果
echo $c
echo $d
echo $((a**2-b**2))

```

上面的脚本文件为 test4.sh,在终端中运行它的指令为：

bash ./test4.sh

这里，.表示当前目录.执行结果如下图所示。

```
imagan@imagan-PC: ~/Documents/programming/..
imagan@imagan-PC:~/Documents/programming/shell$ bash ./test4.sh
10
19
a+b
29
-261
imagan@imagan-PC:~/Documents/programming/shell$
```

15.2.1.5 临时环境变量定义

export 临时环境变量名=表达式

15.2.1.5.1　　示例

a=10

b=”hello”

export C=23

15.2.1.6 表达式中的内置变量

shell 保留了一些特殊的变量，在 shell 运行脚本时，这些变量的值就是确定的，可以用于变量定义或赋值语句中的表达式中。

变量	含义
$0	当前脚本的文件名
$n	传递给脚本或函数的参数。n 是一个数字，表示第几个参数。例如，第一个参数是$1，第二个参数是$2。
$#	传递给脚本或函数的参数个数。
$*	传递给脚本或函数的所有参数。
$@	传递给脚本或函数的所有参数。被双引号(" ")包含时，与 $* 稍有不同，下面将会讲到。
$?	上个命令的退出状态，或函数的返回值。
$$	当前Shell进程ID。对于 Shell 脚本，就是这些脚本所

15.2.2　数组定义语句

15.2.2.1 语法

数组名=(val1 val2 val3 ...)

数组的元素 valk 可以为表达式。

要访问数组元素，则：

${数组名[下标]} #下标从 0 开始

要访问数组所有元素，则：

${数组名[*]} # * 或 @都表示所有的意思。

若要创建一个空数组，则：

数组名=()

可以给空数组赋值，则数组长度自动增长。

例如：

myArray=()

myArray[10]=10 #数组长度增至 11

15.2.2.2 示例

```
1 #!/bin/bash
2 my_array=(1 3 4 "7")
3 echo ${my_array[0]} #输出第1个元素
4 echo ${my_array[*]} #输出所有元素的值
5 echo ${my_array[@]} #与上一行语句等价
```

15.2.3　关联数组定义语句

关联数组，相当于字典，里面有多个元素，每个元素是一个键值对。

15.2.3.1 语法

declare -A 数组名

这条语句声明了一个关联数组。只有给他赋值，才有意义，因此，接下来可以使用下面的语句赋值：

数组名[键名 1]=值 1

数组名[键名 2]=值 2

...

关联数组元素的访问，对于单个元素，使用键名访问。

15.2.3.2 示例

```
#!/bin/bash
declare -A student
student["name"]=张三
student["age"]=18
student["gender"]=男
echo ${student["name"]},${student["age"]} #访问单个元素
echo ${student[*]} #访问所有元素
echo ${student[@]} #同上
```

若要获取数组元素的长度，可以使用#符号。

${#数组名[*]) 或 ${#数组名[@]}

示例：

```
#!/bin/bash
declare -A student
student["name"]=张三
student["age"]=18
student["gender"]=男
echo ${student["name"]},${student["age"]} #访问单个元素
echo ${student[*]} #访问所有元素
echo ${student[@]} #同上
echo 数组元素个数为： ${#student[*]}
```

15.2.4　赋值语句

赋值语句的作用是修改已定义的变量或数组元素的值。

15.2.4.1 给用户变量赋值

格式：变量名=新的值

注意：=两边都不能有空格

例如：

```
a=2
a=$a+3   # a=”2+3”
```

15.2.4.2 给环境变量赋值

export 环境变量名=新值

15.2.4.3 将命令结果赋给变量

格式 1：变量名=`命令`

格式 2：变量名=$(命令)

15.2.5　管理变量语句

对已有的变量进行删除、属性设置等操作。

15.2.5.1 删除变量语句

unset 变量名

15.2.5.2 设变量只读

readonly 已经定义的变量名

设置后，该变量不可再赋值了。

15.2.6　linux命令语句

15.2.6.1 它是什么

linux操作系统对外提供的一序列命令，本质上是一些命令行程序。用户可以通过如下格式调用：

命令名称 参数1的值 参数2的值...

linux命令语句是脚本功能的体现。

命令语句可以通过终端逐条执行。

15.2.6.2 常用命令语句

linux有很多命令语句，用户可通过linux shell手册查询。这里只给出shell编程中常用的命令语句。

15.2.6.2.1　输入命令语句

read -p “提示文本” 变量名

作用：创建一个变量，通过终端提示用户输入，用户输入后，把值赋给这个变量。

例如：read -p “请输入您的姓名:” your_name

该命令运行后，将会提示用户从终端输入一个字符串，用户输入并回车后，将会把输入的字符串赋值给 your_name 变量。

15.2.6.2.2　输出语句

echo 表达式

例如：echo $a

15.2.6.2.3　test 命令语句

作用：检查某个逻辑表达式是否为真。

格式:

test 逻辑表达式

注意:

（1）这里的逻辑表达式不需要加[]，例如： if test $a -lt 5

（2）test 的强大功能还体现在可以测试文件,具体请参考帮助。

15.3 流程控制

流程控制是通过一定的条件控制是否执行某若干语句块的机制。

流程控制需要给定条件表达式，条件表达式是由关系运算符和逻辑运算符

连接起来的式子。因此，要熟悉流程控制，必须学会使用关系运算符和逻辑运算符，并进一步学习如何由这两种运算符构成逻辑表达式。

15.3.1　关系运算符

用于比较两个表达式的关系（相等、大于、小于、大于或等于、小于或等于等）。

写法：表达式1 关系运算符 表达式2

数值关系运算符有：

运算符	意义	举例
-eq	是否相等	[$a -eq 4]
-ne	不等于	[$a -ne 5]
-gt	大于	[$a -gt 4]
-lt	小于	[$a -lt 4]
-ge	大于或等于	[$a -ge 4]
-le	小于或等于	[$a -le 4]

注意：(1)由于是表达式，因此需要用[]把式子括起来。下面的逻辑运算符也是类似。(2)与中括号相邻的表达式与中括号之间必须留有空格。（3）上面的运算符只针对整数。

字符串关系运算符有：=(是否相同)、!=(是否不同）、-z（字符串长度是否为0）、-n(字符串长度是否不为0）

15.3.2　逻辑运算符

```
与：-a
或：-o
非：！
```

举例：[$a -gt 5 -a $a -lt 10] #逻辑运算符优先级高于关系运算符.

与运算符还可以用&&表示，而或运算符可以用||表示，但此时必须要再加上一层括号。

[[$a -gt 5 -a $a -lt 10]]

15.3.3　逻辑表达式

由关系运算符、逻辑运算符等要素连接起来的、最终计算得到结果为真或假的式子，称为逻辑表达式。

逻辑表达式有两种描述方法。

15.3.3.1 中括号表示法

就是把表达式用一对[], 中括号和里面的内容要有空格作为间隔。具体如 15.3.1、15.3.2 所述。

15.3.3.2 test 函数表示法

详见“test 命令语句”

15.3.4　分支流程

15.3.4.1 if 分支

形式 1:

```
if contidition
then
…
fi
```

形式 2:

```
if condition
then
   ...
else
   ...
```

```
fi
```

形式 3：

```
if condition
then
    ...
elif ...
then ...
    ...
else
   ...
fi
```

15.3.4.2 case ... esac 多分支

```
case 值 in
模式 1)
    command1
    command2
    ...
    commandN
    ;;
模式 2)
    command1
    command2
    ...
    commandN
    ;;
esac
```

15.3.4.3 示例

以下是一个脚本示例，充分地展示了分支流程的用法。

```
#!/bin/bash
read -p "请输入账号(gcj):" acount
read -p "请输入密码(1234):" password
if [[ $acount = "gcj" &&  $password = "1234" ]]
then
        echo '欢迎访问本系统'
else
        echo '你输入的用户名或密码错误'
fi
read -p "请输入整数:" n
if test $n -lt 0
then
        echo "n<0"
elif test $n -eq 0
then
        echo "n=0"
else
        echo "n>0"
fi
if test -e "/etc/profile"
then
        echo "存在profile文件"
else
        echo "不存在profile文件"
fi
read -p "请选择1-3之间的整数:" n
case $n in
1) echo "你输入了1"
;;
2) echo "你输入了2"
;;
3) echo "你输入了3"
;;
*) echo "你输入的不是1-3"
esac
```

15.3.5　循环流程

15.3.5.1 for 循环

多行格式：

```
for var in item1 item2 ... itemN
do
    command1
    command2
    ...
    commandN
done
```

单行格式：

```
for var in item1 item2 ... itemN; do command1; command2… done;
```

15.3.5.2 while 循环

```
while condition
do
    command
done
```

15.3.5.3 util 循环

```
until condition
do
    command
done
```

15.3.5.4 无限循环

```
while :
do
    echo -n "输入 1 到 5 之间的数字: "
    read aNum
```

```
    case $aNum in
        1|2|3|4|5) echo "你输入的数字为 $aNum!"
        ;;
        *) echo "你输入的数字不是 1 到 5 之间的!"
            continue
            echo "游戏结束"
        ;;
    esac
done
```

15.3.6　示例

```
#!/bin/bash
#for循环示例
myArray=()#创建一个空数组
k=0
while [ $k -lt 10 ] #分支、循环结构中，条件表达式周围的括号与它的边界存在空格
do
        m=$[$k+1]
        myArray[$k]=$m
        k=$m
done
echo ${myArray[*]}
for ele in ${myArray[*]}
do
        if [ $[$ele%2] -eq 0 ] #注意数学表达式的中括号格式$[],里面的表达式与中括号没有空格
        then
                echo $ele
        fi
done
echo "myArray的元素个数：${#myArray[*]}" #元素个数
```

15.4 函数定义与调用

```
#!/bin/bash
#定义一个名为myfun的函数
#该函数在屏幕上输出一串字符
myfun1()
{
        echo "this is my first fun"
}
myfun1  #调用函数
#定义一个add函数，实现两个数相加
#调用格式为：add 3 5
# 始终注意，return x中的x是一个表达式，其最终的估值只能是0-255，否则就会出错
add()
{
        if [ ${#} -ne 2 ]  #若函数调用时传入的参数个数不为2，则输出消息
        then
                echo "函数的参数个数必须为2"
                return 0
        fi
        echo "您传入的参数为：$* "
        return `expr $1 + $2` #是一个数学表达式,$k表示函数调用时传入的第k个参数值
}
read -p "a=" a
read -p "b=" b
add $a $b
echo a+b=$?
add $a
echo a+b=$?
```

注意：在函数调用后，$?表示函数的返回值。

从函数的定义可见，shell 脚本中的函数与真正编程语言（如 C、C++）相比，是有局限性的。