目录

第二部分 嵌入式Linux系统移植篇

附录

嵌入式Linux入门与实践 – Embedded Linux Learning and Practice

迟思堂工作室 李迟

如果出现此行文字，则说明本书还没有完稿，诸多地方还在编写、修改、审校中。敬请注意。

目标读者

本书是一本入门读物，针对嵌入式Linux初学者或想提高的人员，如果你已经沉浸嵌入式Linux多年以上，那么，这本书可能不合适你。本书涉及到Linux系统的介绍以及嵌入式开发两方面，有部分知识点属于工具性的，笔者坚持的理念是工具性只需要懂如何应用即可，无须深入研究，比如git版本管理工具，自动化编译，等等。但涉及开发原理，运行机制等则建议深入，如系统调用原理，内核驱动模型，等等。

写书目的

笔者从事嵌入式Linux开发多年，以此书作为笔者职业生涯的一个阶段总结。因此，本书很多知识点和思想都是笔者个人积累所得，具有少许实践性和较强的个人色彩。愿这本书给大家在嵌入式Linux道路的探索带来一点帮助。

本书使用的环境、代码

本书宿主机为ubuntu 16.04 64bit系统，u-boot、内核和根文件系统等嵌入式实验环境则使用qemu模拟器。
本书排版约定如下：

- 命令行使用“$”表示，而一些以root权限执行的，统一加上sudo。使用“#”作为命令的注释。因为root权限的shell提示符也是“#”，为了避免混乱，书中做此约定。但是在实际操作时，大家可以使用sudo -s切换为root用户。行文所称的**root**权限，包括使用root用户登陆，以及普通用户用sudo -s切换到root用户。
- 书中代码均有标行号，以方便行文表述。但是代码行数少的除外。
- ~~书中代码标题与github源码仓库保持一致，方便大家直接对照。~~

主要开源项目及版本如下：

- u-boot版本：2018.01
- coreboot版本： 4.7
- Linux内核版本：4.15.9
- busybox版本：1.28.1
- qemu版本：4.0.0
- gcc版本：宿主机为5.4.0，交叉编译器为6.3.0

在线资源

本书勘误地址为：https://github.com/cststudio/embedded-linux-learning-and-practice-ebook。
本书源码地址为：https://github.com/cststudio/embedded-linux-learning-and-practice-code。

捐赠

如果觉得本书有用，可扫码赞助作者：

联系作者

邮箱：li@latelee.org(备用：latelee@163.com)。

FAQ

Q：本书会持续更新吗？
A：会，不影响工作生活情况下会更新。

Q：本书有些内容还是过于简单，可否深入？
A：限于个人精力，无法深入研究。想提高，建议自行研究。

Q：本书会出版吗？
A：目前定位为开源电子书，但不保证以后因为经济原因新加内容再收费或出版。

Q：本书可多人编写吗？
A：为确保文笔一致性，目前只限笔者，各位可提pr。笔者酌情合并。

迟思堂工作室 李迟
2017年年底构思，2018年4月开始编写。
2018年下半年~2019年上半年，因故搁置，2019年下半年重新拾起

第1章 嵌入式Linux基础概念

嵌入式系统应用十分广泛，在消费类电子产品、安防、互联网、工业控制、医疗领域等都可见其身影。
嵌入式系统的定义为：以应用为中心，以计算机技术为基础，软硬件可裁剪，适用于应用系统，对功能、体积、成本、功有耗严格要求的专用计算机系统。首先它是一个计算机系统（因此涉及软硬件），其次它是专用的（因此需要裁剪）。说通俗直白一些，嵌入式系统就是专门针对某种应用的计算机系统。
从定义上看，最简单的单机系统，如51单片机、AVR单片机，在某种程度上也可以称之为“嵌入式系统”，但因为功能较简单，且处理性能不高，因此一般称之为单片机系统。通常意义上的嵌入式系统，其硬件平台有x86、ARM、MIPS和PowerPC，等等，而软件操作系统则有Linux、WinCE、VxWorks，等等。
本书专注ARM+Linux，两者的资源和应用都非常多，是入门嵌入式的一个比较好的选择。

如果出现此行文字，则说明本书本章节还没有完稿，诸多地方还在编写、修改、审校中。敬请注意。

第1章 嵌入式Linux基础概念

嵌入式系统应用十分广泛，在消费类电子产品、安防、互联网、工业控制、医疗领域等都可见其身影。
嵌入式系统的定义为：以应用为中心，以计算机技术为基础，软硬件可裁剪，适用于应用系统，对功能、体积、成本、功有耗严格要求的专用计算机系统。首先它是一个计算机系统（因此涉及软硬件），其次它是专用的（因此需要裁剪）。说通俗直白一些，嵌入式系统就是专门针对某种应用的计算机系统。
从定义上看，最简单的单机系统，如51单片机、AVR单片机，在某种程度上也可以称之为“嵌入式系统”，但因为功能较简单，且处理性能不高，因此一般称之为单片机系统。通常意义上的嵌入式系统，其硬件平台有X86、ARM、MIPS和PowerPC，等等，而软件操作系统则有Linux、WinCE、VxWorks，等等。
本书专注ARM+Linux，两者的资源和应用都非常多，是入门嵌入式的一个比较好的选择。

1.1 嵌入式Linux是什么

在真正开始介绍嵌入式Linux之前，我们先来了解一下Linux有关的历史。

GNU/Linux
GNU是“GNU's Not Unix!”的递归缩写，意指其设计目标是类Unix操作系统，但又不包含Unix的代码，且是自由软件。其发音与“革努”相近。
GNU，是由Richard Stallman在1983年公开发起的。它的目标是创建一套完全自由的操作系统。Richard Stallman最早是在net.unix-wizards新闻组上公布该消息，并附带一份《GNU宣言》等解释为何发起该计划的文章，其中一个理由就是要“重现当年软件界合作互助的团结精神”。
Stallman宣布GNU应当发音为Guh-NOO，与canoe发音相同，以避免与gnu（非洲牛羚，发音与new相同）这个单词混淆。UNIX是一种广泛使用的商业操作系统的名称。由于GNU将要实现UNIX系统的接口标准，因此GNU计划可以分别开发不同的操作系统部件。GNU计划采用了部分当时已经可自由使用的软件，例如TeX排版系统和X Window视窗系统等。不过GNU计划也开发了大批其他的自由软件。
为保证GNU软件可以自由地“使用、复制、修改和发布”，所有GNU软件都在一份在禁止其他人添加任何限制的情况下授权所有权利给任何人的协议条款，GNU通用公共许可证（GNU General Public License，GPL）。这个就是被称为“反版权”（或称Copyleft）的概念。 1985年Richard Stallman又创立了自由软件基金会（Free Software Foundation）来为GNU计划提供技术、法律以及财政支持。尽管GNU计划大部分时候是由个人自愿无偿贡献，但FSF有时还是会聘请程序员帮助编写。当GNU计划开始逐渐获得成功时，一些商业公司开始介入开发和技术支持。当中最著名的就是之后被Red Hat兼并的Cygnus Solutions。
到了1990年，GNU计划已经开发出的软件包括了一个功能强大的文字编辑器Emacs，C语言编译器GCC，以及大部分UNIX系统的程序库和工具。唯一依然没有完成的重要组件就是操作系统的内核。

再来看看Linux发展历史。
Linux操作系统（Linux），是一种计算机操作系统。Linux操作系统的内核的名字也是“Linux”。Linux操作系统也是自由软件和开放源代码发展中最著名的例子。
Linux内核最初是为英特尔386微处理器设计的。现在Linux内核支持从个人电脑到大型主机甚至包括嵌入式系统在内的各种硬件设备。
在开始的时候，Linux只是个人狂热爱好的一种产物。但是现在，Linux已经成为了一种受到广泛关注和支持的一种操作系统。包括IBM和惠普在内的一些计算机业巨头也开始支持Linux。很多人认为，和其他的商用Unix系统以及微软Windows相比，作为自由软件的Linux具有低成本，安全性高，更加可信赖的优势。
Linux内核最初只是由芬兰人林纳斯•托瓦兹(Linus Torvalds)在赫尔辛基大学上学时出于个人爱好而编写的。最初的设想中，Linux是一种类似Minix这样的一种操作系统。Linux的第一个版本在1991年9月被发布在Internet上，随后在10月份第二个版本就发布了。
Linux的历史是和GNU紧密联系在一起的。从1983年开始的GNU计划致力于开发一个自由并且完整的类Unix操作系统，包括软件开发工具和各种应用程序。到1991年Linux内核发布的时候，GNU已经几乎完成了除了系统内核之外的各种必备软件的开发。在Linus Torvalds和其他开发人员的努力下，GNU组件可以运行于Linux内核之上。
——GNU系统需要内核，而Linux内核需要开发工具和应用程序，两者就如此奇妙地组合起来，一个完全自由的操作系统正式诞生。这个系统应该叫做“GNU/Linux操作系统”，但人们经常称之为“Linux操作系统”。
现在，许多GNU工具还被广泛地移植到Windows和Mac OS上。
GNU的logo如图1.1所示。Linux标志为企鹅(名为Tux)，如图1.2所示。

图1.1 GNU logo——角马

图1.2 linux logo——Tux

ARM处理器
ARM(Advanced RISC Machine)，即可以认为是一个公司的名字，也可以认为是对一类微处理器的能称，还可以认为是一种技术的名称。ARM公司是嵌入式RISC微处理器技术的领导者，创办于1990。ARM公司并不生产芯片，而是出售芯片技术授权。
ARM处理器架构发展的天梯图如图1.3所示。

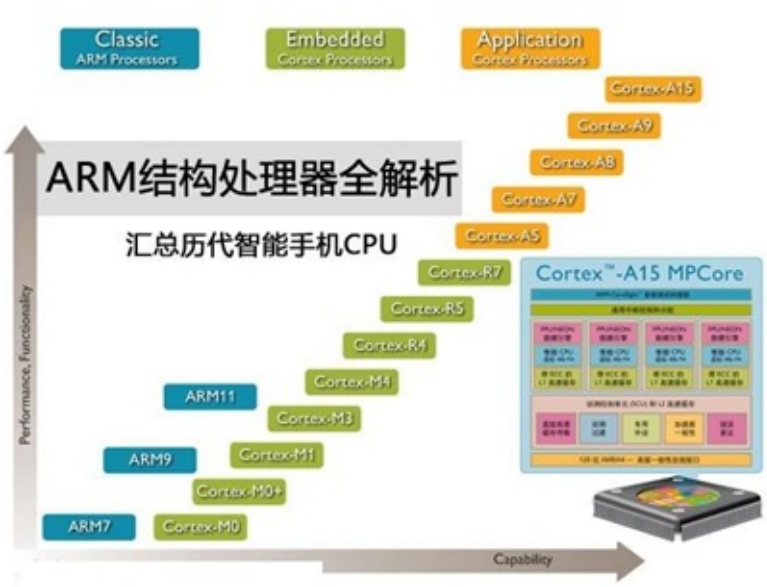

图1.3 ARM处理器架构发展图

不同人对嵌入式Linux的定义不同，涉及到的范围亦有不同。本书使用狭义的定义：将Linux源码移植到特定芯片（包括但不限于X86、ARM、MIPS芯片），从而实现某一种或多种应用目的。本文仅以ARM芯片为硬件平台。

1.2 嵌入式Linux由哪些部分组成

嵌入式Linux系统结构分为三部分，应用软件层、操作系统层、硬件设备层，它们是上下级关系。如图1.4所示。

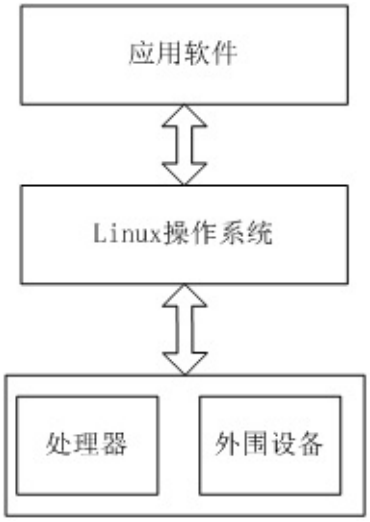

图1.4 嵌入式Linux系统结构图

如果以功能划分，嵌入式Linux有输入、处理、输出三大层面。其结构如图1.5所示。

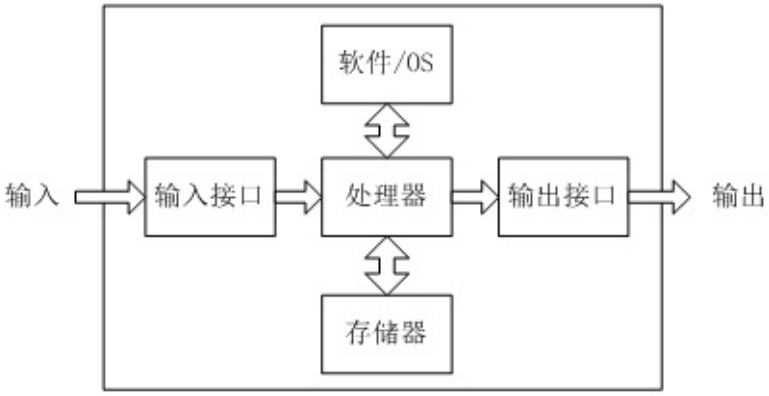

图1.5 嵌入式Linux功能结构图

很多功能都可以用图1.5的结构图来分析，如视频监控系统，从输入到输出流程描述如下：

1、从摄像头获取数据，比如USB摄像头，通过USB协议传输数据到Linux系统，此即为输入层。

2、CPU进行处理，其中包含了底层驱动的处理，比如在Linux内核进行视频的编码。此为处理层。

3、应用层通过V4L2接口获取视频数据，使用使用QT界面显示，或者通过网络使用RTSP协议传输到PC上显示。这是输出层。

其它的应用也是类似的。

图1.5下方有一个框图为存储器，存储器存储着系统固件、系统程序等等。嵌入式Linux系统典型存储结构如图1.6所示。存储器可以是flash，也可以是flash+sd卡混合形式。

图1.6 嵌入式Linux系统存储结构

图1.6中，最左侧为bootloader，即启动装载程序，接着是参数区，用于传递配置参数，接着是内核映射，亦即kernel，最后是文件系统。实际应用中，还可能有多个分区，以适用更复杂的场合，比如有应用层程序分区、升级预留分区，等等。关于存储方面的内容，这里仅仅提到一些概念，后文将会有专门章节介绍。

1.3 如何学习嵌入式Linux

在给出学习路线之前，先看看嵌入式Linux有哪些基础的或重要的知识。

1.3.1 掌握Linux系统

Linux系统的使用必须熟练掌握。使用越熟练，越能提高对Linux系统的认识，并能提高开发效率。所以必须要安装一个Linux发行版本系统，并动手实践。使用Linux是一个循序渐进的过程，只能靠自己日积月累达成，没有捷径，如果有捷径，那就是不停敲命令，不停做笔记。

除此以外，一些学习工作中用到的工具，也需要掌握，如思维导图、云笔记、版本控制等等。

1.3.2 上层应用开发

C语言

C语言是嵌入式Linux的精髓，C语言的掌握程度，决定了嵌入式Linux路子走多远。从底层的bootloader到kernel到BusyBox，包括应用层程序，大部分代码都有C语言的身影，而且这些代码中往往包括着一般教科书学不到的技巧。

当然，嵌入式Linux底层依然还是汇编的天下，因为汇编语言更接近机器语言。如果真到修改汇编代码的时候，相信读者对嵌入式Linux已经掌握得很深的程度了。

自动化编译
在Linux系统（包括类Unix、MacOS）中，很多开放源码的软件，都使用make进行自动化编译，因为要学习Makefile文件的使用，了解开源代码编译的“三步曲”。熟悉其编译过程，才能更好地定制我们所需要的库，甚至修改源代码文件。

界面开发
嵌入式Linux系统的界面，很多都用QT实现。QT是跨平台的界面库，类似于微软出品的MFC。如果能掌握界面的开发，为嵌入式产品添色不少。

1.3.3 系统驱动开发

bootloader（启动装载程序）
嵌入式Linux常见的bootloader是u-boot，而X86领域中一般称为BIOS。u-boot的学习没有捷径，只能通过接触代码的方法学习。最好是在qemu模拟环境或购买ARM开发板情况下进行研究，通过打印信息的方法跟踪其流程，而不是仅仅阅读其代码。但是，看懂u-boot代码还需要电路基本知识、芯片手册知识等等。——这些知识，同样适用于内核驱动的开发。
首先要建立的是整体概念和认识。建议先把厂商提供的u-boot源码编译通过，并下载到开发板上看到正常结果后，再用串口打印信息搜索代码，以了解u-boot的代码流程。然后再慢慢研究。如果bootloader不是学习重点，在有一定概念前提下就可以跳过到内核驱动层了，不过就笔者经验来看，bootloader和内核关系十分密切且部分代码是相通的。
一般初始化的代码是汇编代码，对于入门者而言，初学阶段不用追究，等有一定基础后再回头研究也不晚。

kernel(内核**)**
内核涉及内容十分多。作为入门者，与u-boot类似，首先要建立的整体概念。先把厂商提供的内核源码编译通过，并下载到开发板上，串口会打印很多启动信息，这些启动信息能帮助我们学习内核，自己也可以在内核中打印语句，以了解其流程。
每一个平台芯片不同，外设不同，内核均不同，需要进行移植。所谓的“移植”，就是找到合适的驱动，修改适应到该平台的过程。比如，某平台使用2个LCD屏，一个是3.5寸的，另一个是4.3寸的，这需要对内核进行修改。再比如，这个平台使用nand flash是1GB的，另一个平台使用的是512MB的，也需要修改内核。其它如EEPROM、电源芯片、网卡，等等，均如此。
内核知识点分两部分，一是kernel本身的知识点，如内存管理机制(MMU)、时间管理、同步机制，等等。二是外设驱动，如USB、Flash、SPI、LED、GPIO，等等。
初学者建议学习：
1、了解内核编译的过程：配置内核、编译uImage。
2、了解platform驱动模型(笔者文章有现成的模板，已经应用于很多个平台上)。
3、了解一般外设驱动模型。建议从简单的LED、GPIO入门。

高阶知识点：
1、学习各种子系统，如MTD、USB、IIC、SPI、RTC、WDT。
2、学习内核知识，如延时队列、时间管理，同步机制，等等。

rootfs(根文件系统**)** 一般情况下，开发板厂商会提供根文件系统，如果没有，则可以自己去官网下载并编译制作。一般嵌入式Linux使用busybox制作文件系统必要的程序、库、配置文件。因为busybox编译出来的内容体积小，节省空间，所以很多ARM开发板上都是用busybox的。另外还涉及到文件系统格式，像Yaffs2、ramfs、ext4、UBI，等等。所有这些知识点，可以逐步学习研究。

1.3.4 硬件应用

电路原理图和**datasheet**
作为底层开发人员来说，能看懂硬件原理图和datasheet是必要的一项技能。
看懂硬件原理图，就可以知道这个系统上有什么器件，哪些器件有什么功能，如何连接(使用什么协议)，提供什么接口。有了这些认知后，才会对系统有一个全局整体的认识掌握。对于开发人员来说绝对是有利的。

看懂datasheet，才能知道如何访问器件，如何操作器件。
另外，对于嵌入式经常接触到的如I2C协议，SPI协议等等的协议也要掌握。
掌握程度：对于不是偏向计算机硬件专业的同学而言，不需要像他们那样学习数电、模电课程。但起码要知道、掌握上升沿、下降沿的概念，以及高电平、低电平概念，懂得看I2C协议的时序图。懂得如何找到datasheet中关键信息(寄存器说明、时序图)。

1.3.5 学习路线

下面以偏重嵌入式ARM+Linux开发为例，列出入门的路线。这个路线仅是笔者的个人经历，读者一定要结合自己实际情况进行选择。
1、搭建qemu模拟开发环境，或者买一款ARM开发板。无论是哪种方式，最好选用使用广泛、资料多的芯片平台，因为使用的人多，遇到的问题很可能别人早就解决了，这样的话，能省很多时间，并且提高自信心。
2、使用vmware安装一个ubuntu系统。在vmware软件中设置物理桥接方式上网。在ubuntu设置好samba服务、nfs服务、tftp服务。
3、自己动手编译u-boot、kernel，将得到的镜像文件使用qemu启动，或者将其烧写到板子上（烧写方法因不同开发板而不同）。
4、自己修改kernel，添加一个简单的字符驱动。
5、制作busybox根文件系统，使用qemu启动，或者烧写到板子上，再挂载NFS，然后在虚拟机交叉编译一个Helloworld程序，并在板子上运行。
6、根据个人兴趣，开始研究不同方向，比如u-boot、kernel、应用层开发、界面QT开发。如果时间允许，每一方面都建议接触。
7、选择自己重点关注方面，继续深入研究。

这里要说明一下，对于初学嵌入式Linux的人，笔者并不推荐开始就学汇编、ARM体系结构，因为这类内容对初学者太过高深了。

1.4 其它知识点补充

前面提到的都属于“内功心法”，嵌入式Linux应用在各种领域中，我们在实际工作中，还要接触与特定专业领域相关的知识。下面简单介绍一下。

音视频
音视频(包括图像显示)应用比较广泛，在安防监控、智能交通、无人机、图像识别等领域都有应用。涉及技术有：音频编码解码（mp3）、视频编码解码（H.264、H.265）、颜色空间（YUV、RGB）。

通信协议
通信协议涉及非常广泛，如常见的TCP/IP协议，还有IEEE 802.11(无线协议)、安防行业会用到 onvif协议和gb28181协议。

事实上，任何一个领域都会涉及多种技术，因此需要围绕领域知识点扩散学习掌握。

以上提到的嵌入式Linux相关知识点，不同的人基础不同，目标也不同。如计算机硬件专业的同学，数电、模电和电路图已经掌握，就要加强C和Linux系统的学习。反之，计算机软件专业的同学，就要去学习硬件知识，等等。

对于书籍，有的可能一下子无法理解，那是因为功力不到。有的可能觉得没有用处，那是因为还没有涉及此方面。比如，非计算专业的人看《编译原理》、《操作系统原理》、《计算机体系结构》等书籍，肯定是看不懂的，而且初学者也不必去看。又比如，嵌入式有的领域使用到H264编码、MPEG编码，802.11、CDMA，如果不是进入有关行业的，也不必学习。另外网上很多人写的嵌入式入门文章提到0.11内核版本、2.4内核版本的书，还有离散数字、算法导论的书，并不是都适用所有人。相信对于初学嵌入式Linux的人来说，先去啃ARM体系结构和Linux内核剖析，肯定会看得云里雾里。 所以大家一定要量体裁衣，有多大胃口吃多少饭，有多大头戴多大帽，根据个人已有的知识和所处的阶段进行选择。
另外要说明的是，很多知识点是密不可分的，且界线是很模糊的。比如应用层和底层。因为有时出现问题，并不知道具体哪里的问题，这就需要站在比较高的层面(系统视野)看问题，才会快速定位并解决问题。建议以某一部分为核心点，

另外的部分则了解掌握。如果有能力，最好都学。理论上知识点学得越多，越有利于看问题深度、广度的提高。
但是，熟练使用Linux使用、了解Linux系统，能提高我们工作效率。而C/C++语言、内核知识、硬件基本知识、编码规范，英文阅读能力，等，在嵌入式Linux领域都会应用到，都能应用到。

1.5 本书代码示例说明

本书中的命令、shell脚本以及代码示例，都经过实际测试。在分析Linux内核驱动代码时，截取的代码都带上行号，以方便读者跟踪。
读者在阅读本书时，可以先行下载本书配套代码示例。但是建议读者能亲自动手敲代码，并自己做实验，将结果与书中例子进行对照学习。
如果没有特别说明，书中所用例子均与具体芯片平台无关。理论在Linux系统中都可以编译、运行。
Linux命令的执行分别root权限和非root权限，非root权限提示符默认为“$”，root权限提示符默认为“#”，而“#”恰好又是Linux系统各种脚本文件的注释，为了避免混乱，书中的命令统一使用“$”提示符，在需要root权限情况下，则在命令前面添加sudo。在实际操作中，读者可以用sudo –s切换到root权限进行操作。

1.6 本书实验平台说明

本书使用的实验平台为qemu，qemu为一款优秀的模拟器，可以模拟出很多种平台，如X86、ARM、MIPS，等。对于经济还没条件购买开发板的人来说，qemu是一个低成本的学习平台。

X86平台
对于X86平台，使用的是机器类型为440FX[注1]，在qemu中称之为主板（Mainboard）。440FX是英特尔的一个芯片组（chipset），支持奔腾Pro和奔腾II。我们可以将44FX理解为一款X86的板子。在第10章的coreboot中，将会使用这个“板子”。

ARM平台
对于ARM平台，选择的板子为vexpress-a9[注2]。Vexpress系列为ARM公司推出的开发板，面向SoC设计厂商。其设计结构是母板+子板组合。我们并不十分关心其内部硬件逻辑实现，因为我们使用该平台只是为了在qemu进行嵌入式Linux的移植。而实际工作中，我们会遇到各式型号的芯片，每种芯片的硬件都不相同，都需要去了解。

1.7 本章小结

本章根据作者的实践经验，对嵌入式Linux的组成和学习路线进行了总结，并对本书实验环境进行说明。这里要再次强调的是，每个人的目标不同，研究领域也不同，因而学习的重点也会不同。读者朋友们一定要根据自己的实际情况做出选择。本书围绕着嵌入式Linux这一中心点展开，在有限的物理条件下，使用qemu模板器，抛却芯片硬件细节，专注于软件方面的实践。

> 注1. 参见Wikipedia：https://en.wikipedia.org/wiki/Intel_440FX。 ↩

> 注2. 实物开发板参考：https://www.arm.com/files/pdf/Datasheet_CoreTile_Express_A9x4.pdf 。 ↩

第2章 Ubuntu系统安装

Linux系统发行版本十分多，red hat系（包括red hat、centos、fedora）和debian系(包括debian、ubuntu)是使用比较广泛的发行版本。每个发行版本都有自己的特点、社区支持，软件安装方式。但其建成是类似的：使用Linux内核，再加上内核之上的应用软件和图形界面。如此多的发行版本，如何选择合适自己的呢？这取决于发行版支持力度和支持时间，界面友好性，等等。但我们要记住，不同发行版本，其本质上都是Linux系统。

本书使用Ubuntu系统（本章以Ubuntu指代Linux）。因为无论是软件包安装或支持，都比较好，而且Ubuntu更新比较快，社区也比较活跃。

Ubuntu系统镜像分为服务版本(server)和桌面版本(desktop)，每种版本都有32位和64位系统。Ubuntu每年都发行2个版本，分别是当年的4月份和10月份。每隔2年的4月份发行长期支持版本(LTS: Long Term Support)，LTS版本维护时间是5年。版本号由年份+月份组成。比如，截至写书时，Ubuntu最新版本为17.10，即表示该版本是2017年10月份发行的。最新的LTS版本为16.04。而最近将发行的18.04（2018年4月份）还没有正式发布。本书使用Ubuntu 16.06 64位系统。
Ubuntu发行版本ISO镜像下载地址为：
http://old-releases.ubuntu.com/releases/。

Ubuntu16.04版本ISO镜像下载地址为：
http://old-releases.ubuntu.com/releases/xenial/。

本书使用的镜像名称为 `ubuntu-16.04.2-desktop-amd64.iso` 。
可能有的读者朋友们已经了解过Ubuntu的安装过程，甚至还捣腾过几种发行版。那么，这章可以直接跳过。因为本书的实验环境是由安装系统一步一步进行的，因此，为保持完整性，会给出Ubuntu的系统的过程及配置。
本章在物理机 windows 10系统中，使用虚拟机软件VMware Workstation 12 pro安装Ubuntu，在安装后，进行基本配置。使用虚拟机 安装Ubuntu，容易快速上手，且节省了购买真实机器的资金，使用起来与在真实机器安装的Ubuntu并无差别，另外，还可以充分利用Windows系统的优势。在本书的大部分实验操作，代码编写、查阅，都是在Windows上进行的。

如果出现此行文字，则说明本书本章节还没有完稿，诸多地方还在编写、修改、审校中。敬请注意。

第2章 Ubuntu系统安装

Linux系统发行版本十分多，red hat系（包括red hat、centos、fedora）和debian系(包括debian、ubuntu)是使用比较广泛的发行版本。每个发行版本都有自己的特点、社区支持，软件安装方式。但其建成是类似的：使用Linux内核，再加上内核之上的应用软件和图形界面。如此多的发行版本，如何选择合适自己的呢？这取决于发行版支持力度和支持时间，界面友好性，等等。但我们要记住，不同发行版本，其本质上都是Linux系统。

本书使用Ubuntu系统（本章以Ubuntu指代Linux）。因为无论是软件包安装或支持，都比较好，而且Ubuntu更新比较快，社区也比较活跃。

Ubuntu系统镜像分为服务版本(server)和桌面版本(desktop)，每种版本都有32位和64位系统。Ubuntu每年都发行2个版本，分别是当年的4月份和10月份。每隔2年的4月份发行长期支持版本(LTS: Long Term Support)，LTS版本维护时间是5年。版本号由年份+月份组成。比如，截至写书时，Ubuntu最新版本为17.10，即表示该版本是2017年10月份发行的。最新的LTS版本为16.04。而最近将发行的18.04（2018年4月份）还没有正式发布。本书使用Ubuntu 16.06 64位系统。
Ubuntu发行版本ISO镜像下载地址为：
http://old-releases.ubuntu.com/releases/。

Ubuntu16.04版本ISO镜像下载地址为：
http://old-releases.ubuntu.com/releases/xenial/。

本书使用的镜像名称为 `ubuntu-16.04.2-desktop-amd64.iso` 。
可能有的读者朋友们已经了解过Ubuntu的安装过程，甚至还捣腾过几种发行版。那么，这章可以直接跳过。因为本书的实验环境是由安装系统一步一步进行的，因此，为保持完整性，会给出Ubuntu的系统的过程及配置。
本章在物理机 windows 10系统中，使用虚拟机软件VMware Workstation 12 pro安装Ubuntu，在安装后，进行基本配置。使用虚拟机 安装Ubuntu，容易快速上手，且节省了购买真实机器的资金，使用起来与在真实机器安装的Ubuntu并无差别，另外，还可以充分利用Windows系统的优势。在本书的大部分实验操作，代码编写、查阅，都是在Windows上进行的。

2.1 使用虚拟机VMware Workstation安装Ubuntu

本节介绍VMware Workstation安装Ubuntu系统，关于VMware Workstation的下载安装，此处从略。下面从创建虚拟机开始，一步一步安装Ubuntu系统。

双击打开VMware Workstation软件，在主页面中点击“创建新的虚拟机”图标，进入向导选择。默认选择“自定义(高级)”。如图2.1所示。

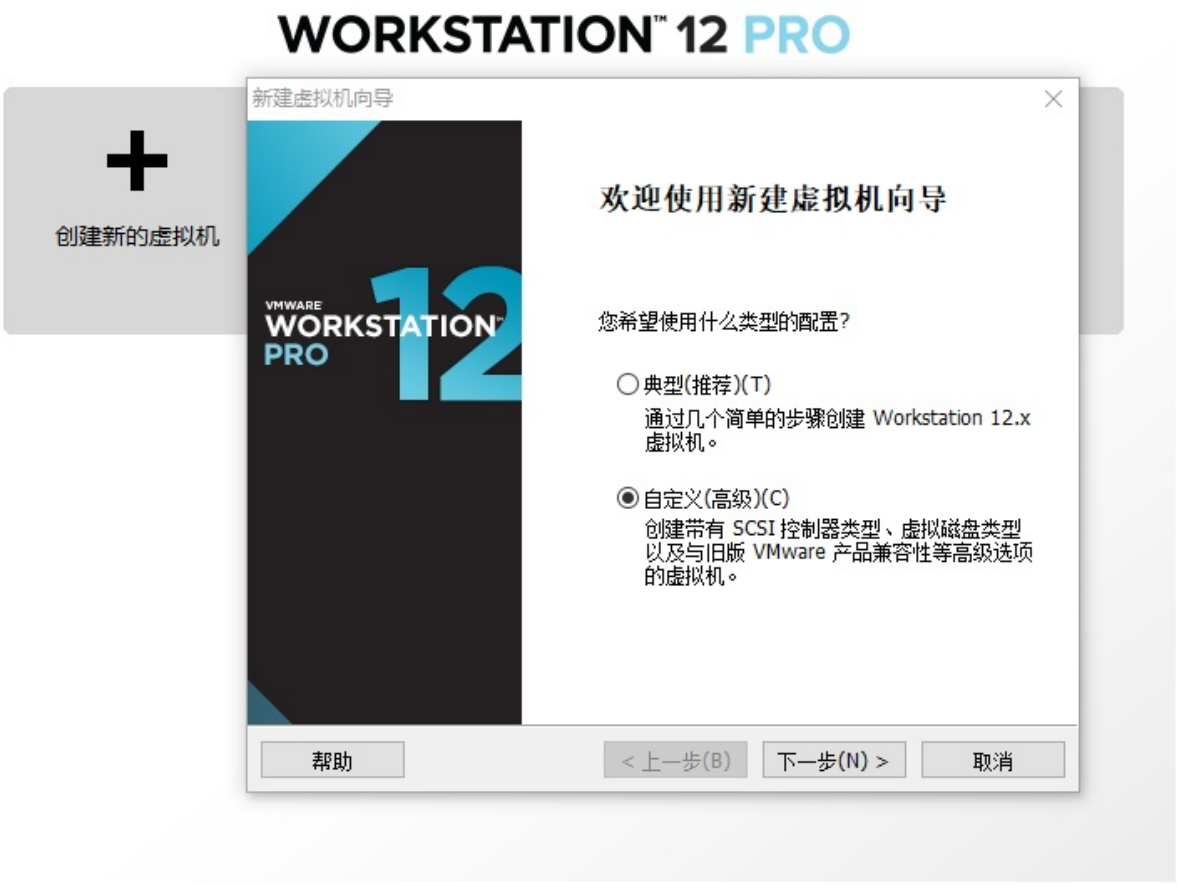

图2.1 虚拟机向导

点击“下一步”，选择虚拟机硬件兼容性，此处使用默认性。如图2.2所示。

图2.2 选择虚拟机硬件兼容性

点击“下一步”，选择操作系统镜像位置。在“安装程序光盘映像文件（iso）”处选择下载好的Ubuntu镜像ISO文件。这时，VMware会自动识别出操作系统类型以及系统的位数。如图2.3所示。

图2.3 安装客户机操作系统

点击“下一步”，输入用户信息，这些就是Ubuntu登陆所需的用户名和密码。如图2.4所示。这里需要说明的是，由于是在个人电脑使用Ubuntu，因此密码可以设置相对简单一些，以减少输入密码的工作量。图2.3密码为6位数字，相信聪明的读者能猜测密码是123456 ^_^。

图2.4 填写用户信息

点击“下一步”继续，设置虚拟机命名（使用默认值）以及虚拟机文件安装目录（如目录不存在，则会自动创建）。如图2.5所示。

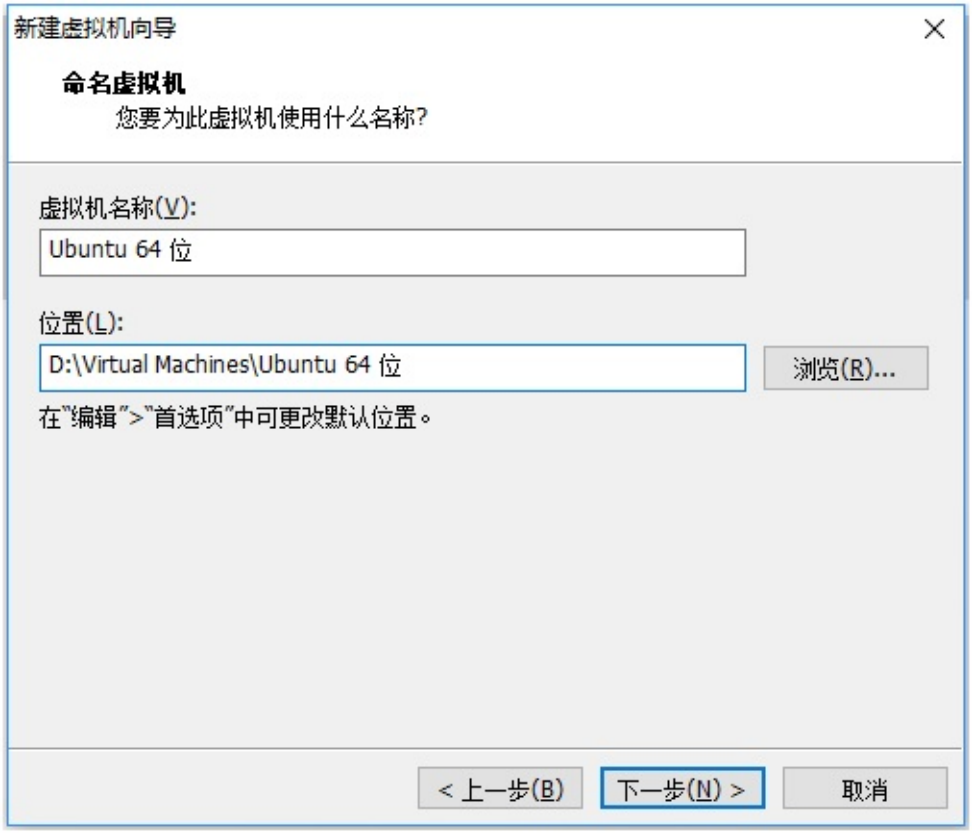

图2.5 设置虚拟机名称和安装目录

点击“下一步”继续，设置处理器个数。此处一共有4个处理器核心。如图2.6所示。

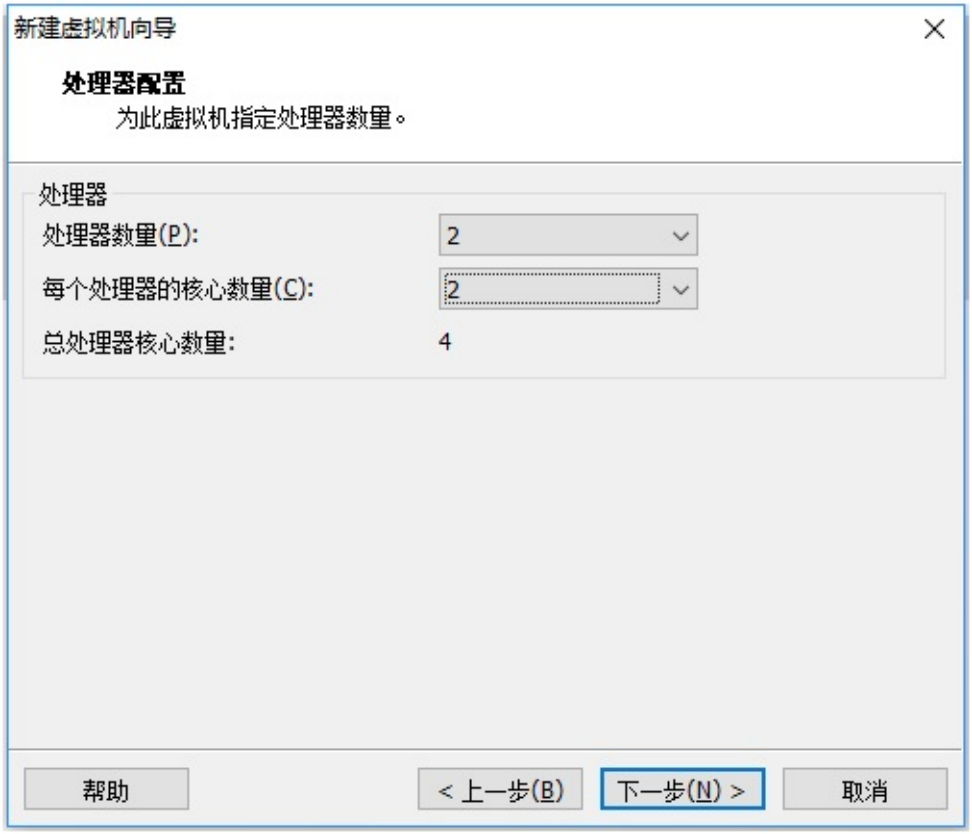

图2.6 设置处理器数目

点击“下一步”继续，设置虚拟机占用内存，这里设置的容量为2048MB，即2GB。如图2.7所示。需要说明，总处理器核心数量和内存容量不应过多，否则会影响物理机的性能。

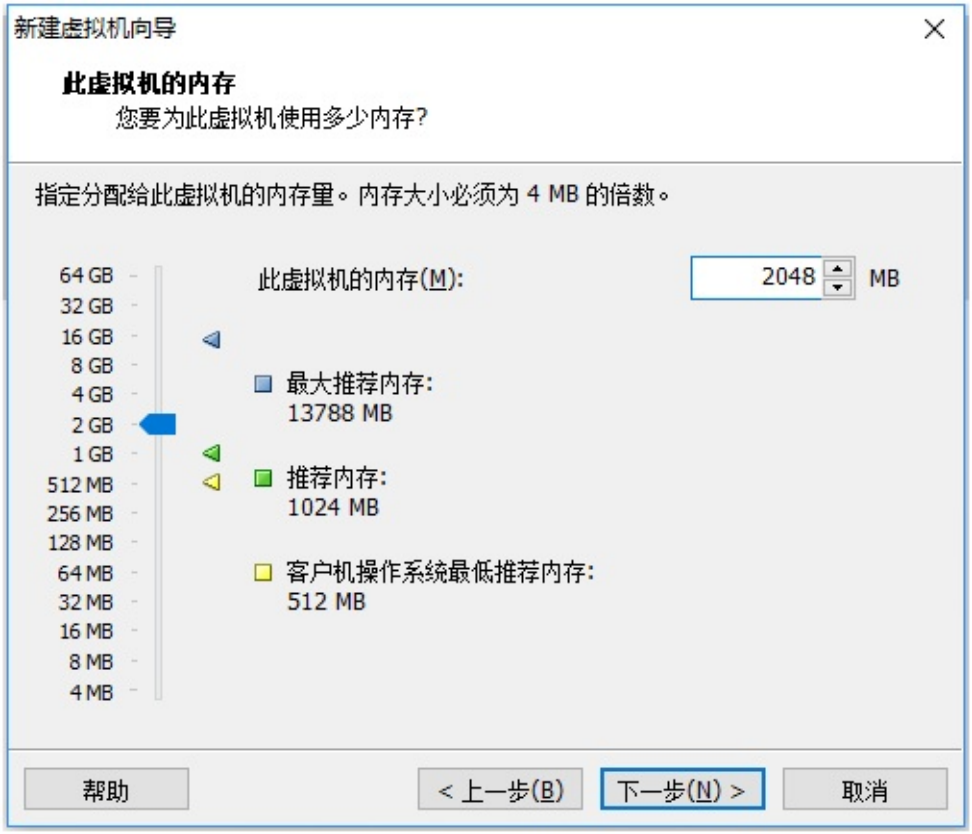

图2.7 设置内存容量

点击“下一步”继续，选择网络连接类型，这里选择的是桥接网络，这种方式使得虚拟机内的Ubuntu直接从路由器自动获取IP地址，相对来说比较方便。如图2.8所示。

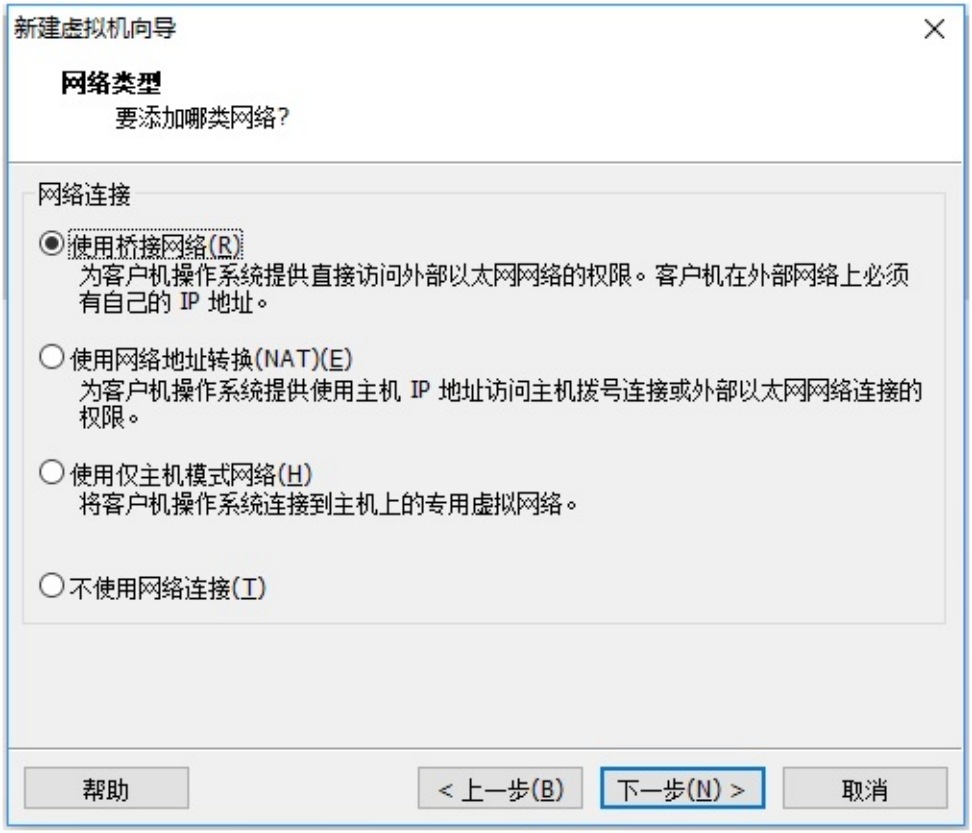

图2.8 选择网络连接类型

点击“下一步”继续，选择I/O控制器类型，此处保持默认值。如图2.9所示。

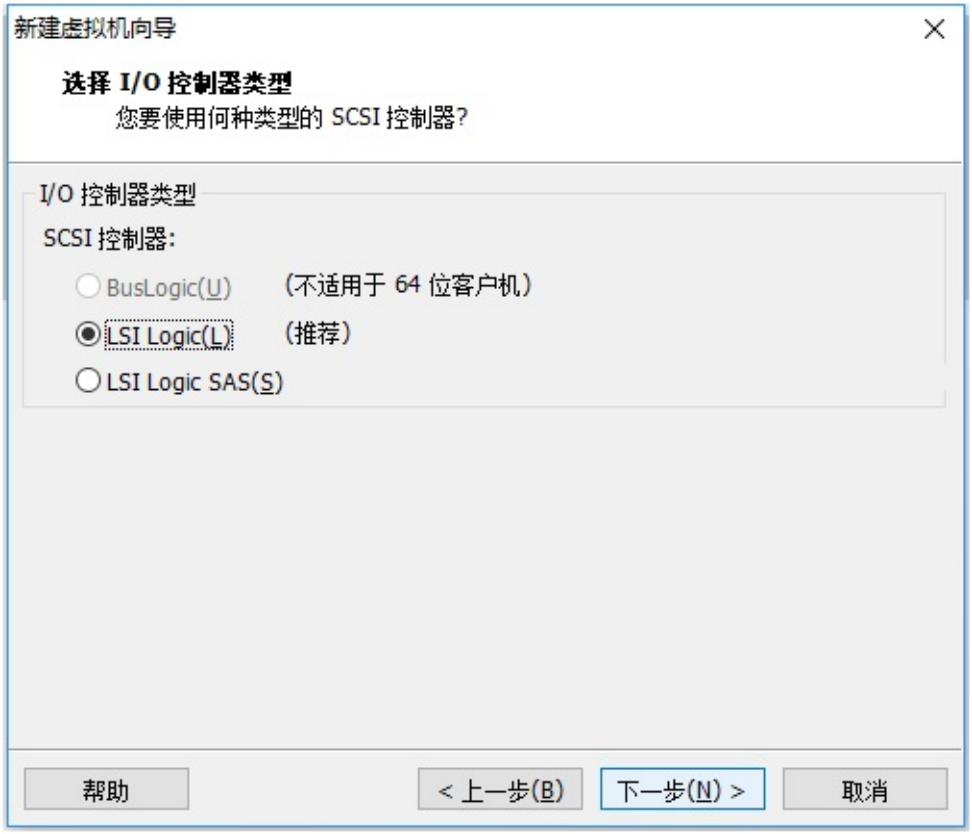

图2.9 选择I/O控制器类型

点击“下一步”继续，选择磁盘类型，同样保持默认值。如图2.10所示。

图2.10 选择磁盘类型

点击“下一步”继续，选择磁盘，因为是从零创建虚拟机的，所以选择“创建新虚拟磁盘”。如图2.11所示。

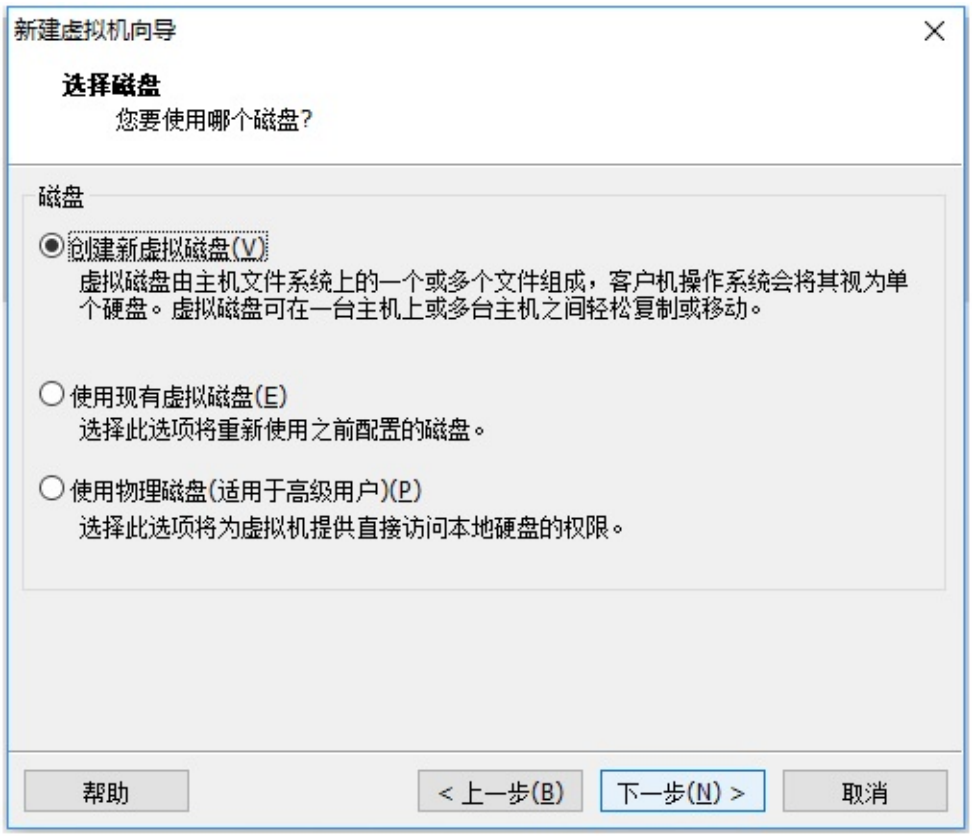

图2.11选择磁盘

点击“下一步”继续，设置磁盘容量。要说明的是，这里的容量并不是立刻分配的空间（注意，容量下方的“立刻分配所有磁盘空间”并未打勾）。这里设置200GB，应该可以应对开发所用。万一哪天虚拟机的磁盘空间不够使用，还可以向虚拟机再添加一块硬盘。如图2.12所示。

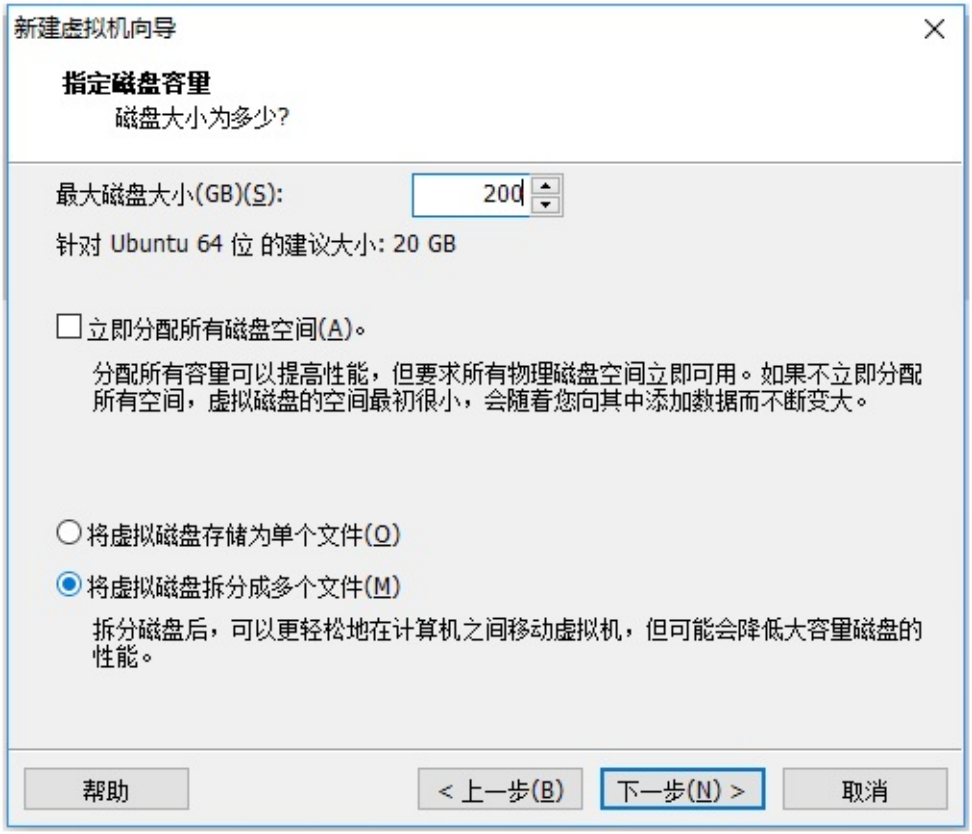

图2.12 指定磁盘容量

点击“下一步”继续，指定磁盘文件，使用默认值即可。如图2.13所示。

图2.13 指定磁盘文件

点击“下一步”，完成虚拟机的创建，此时会列出刚刚创建的虚拟机的信息。如图2.14所示。

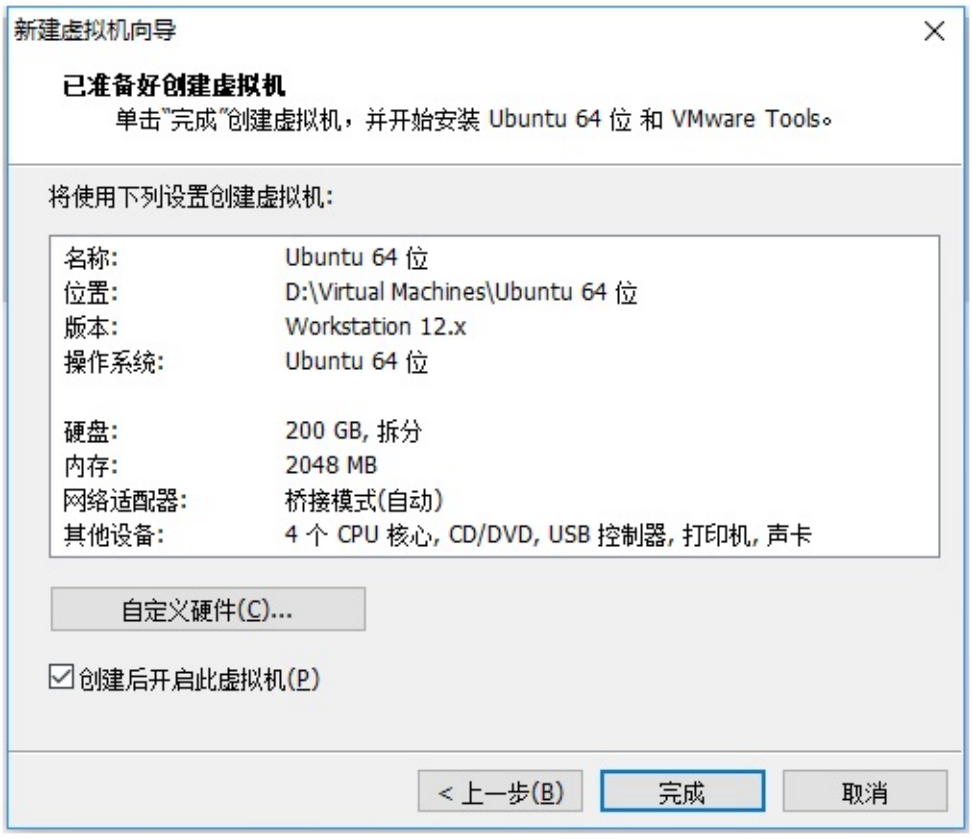

图2.14 完成虚拟机的创建

点击“完成”按钮后，将启动虚拟机。然后正式进入Ubuntu安装之旅。
稍等片刻后，出现Ubuntu图标，接着自动进入安装过程，如图2.15、图2.16所示。

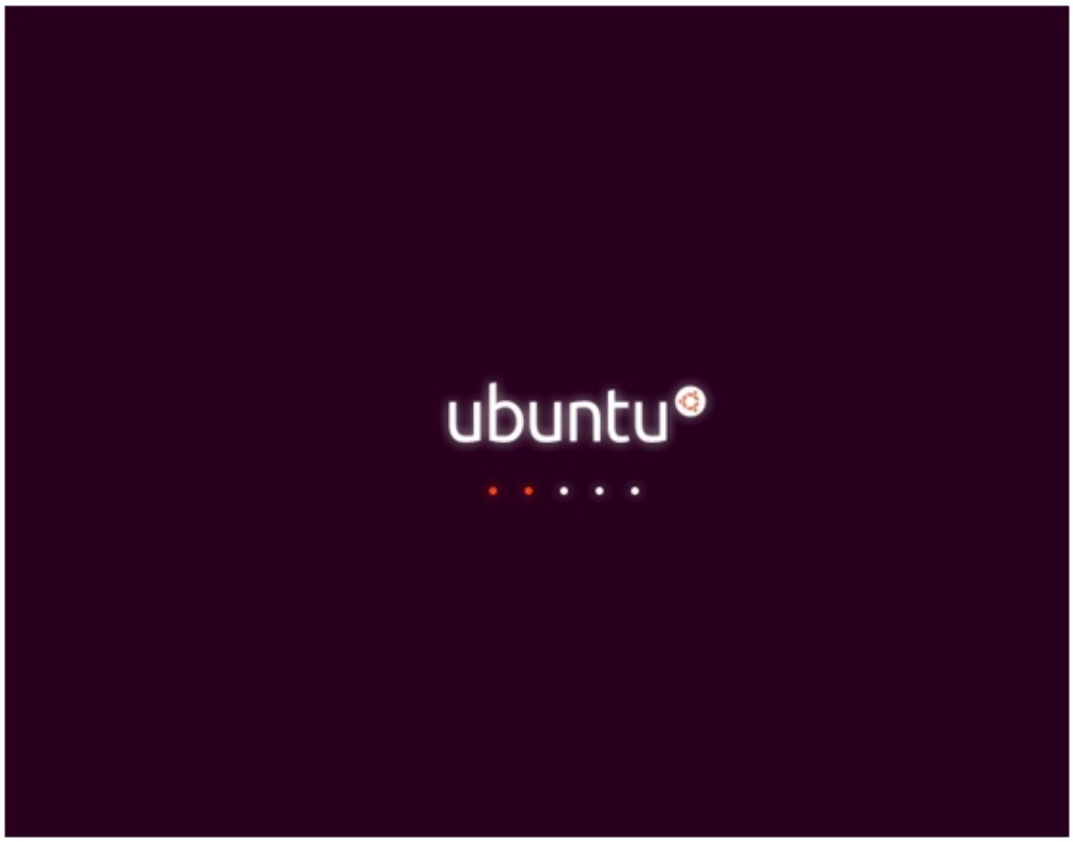

图2.15 启动出现Ubuntu图标

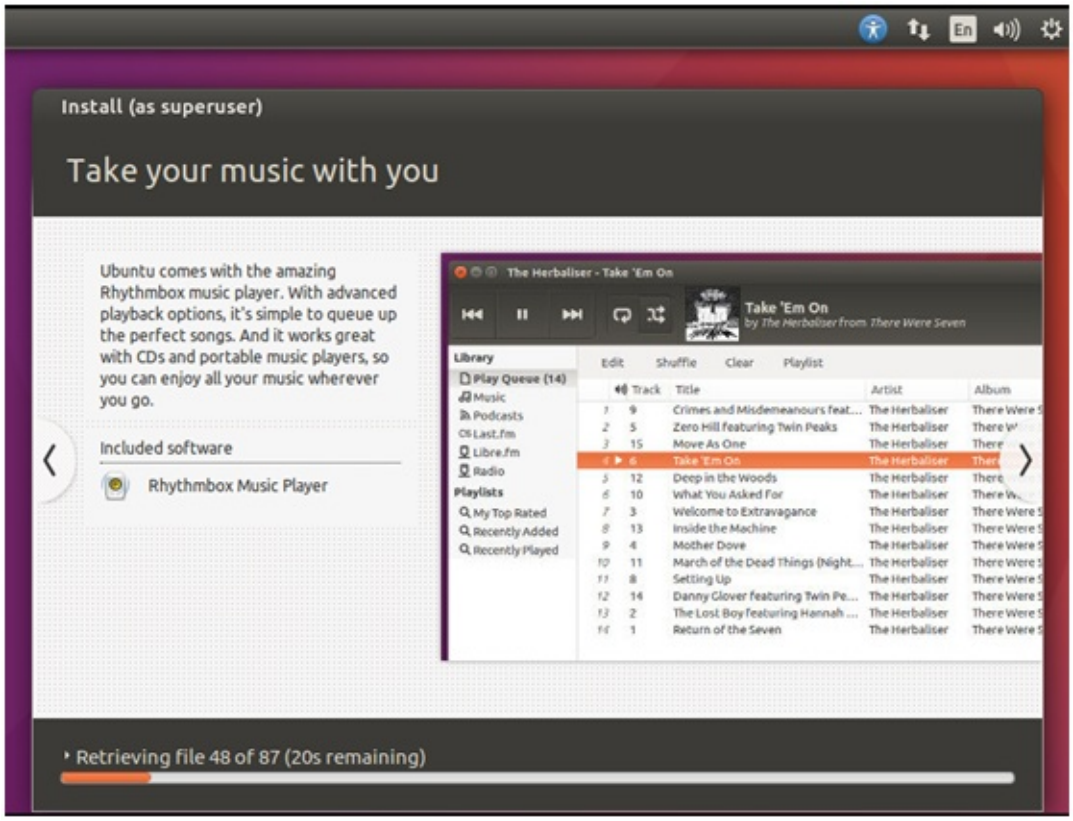

图2.16 Ubuntu安装过程

在作者所用的电脑中安装，大约20分钟左右，完成安装。如图2.17所示。

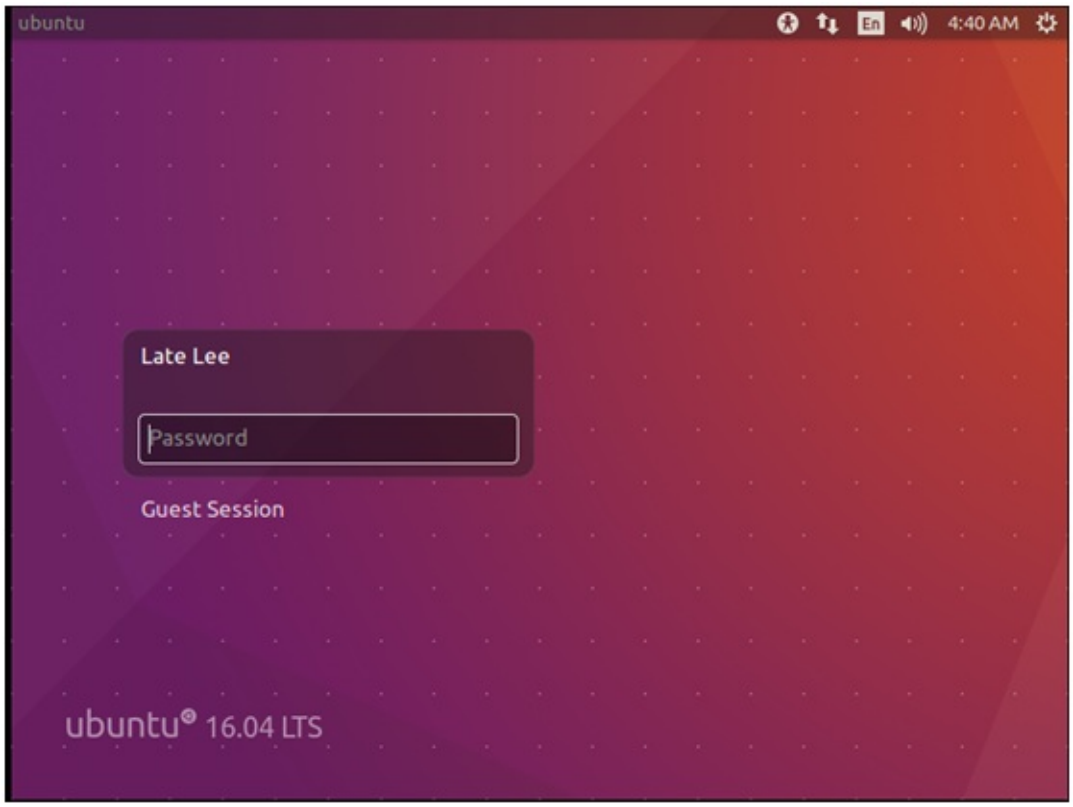

图2.17 完成安装

此时，输入刚才设置的密码，即可登陆到Ubuntu系统中。图2.18是系统内核版本和发行版本信息。

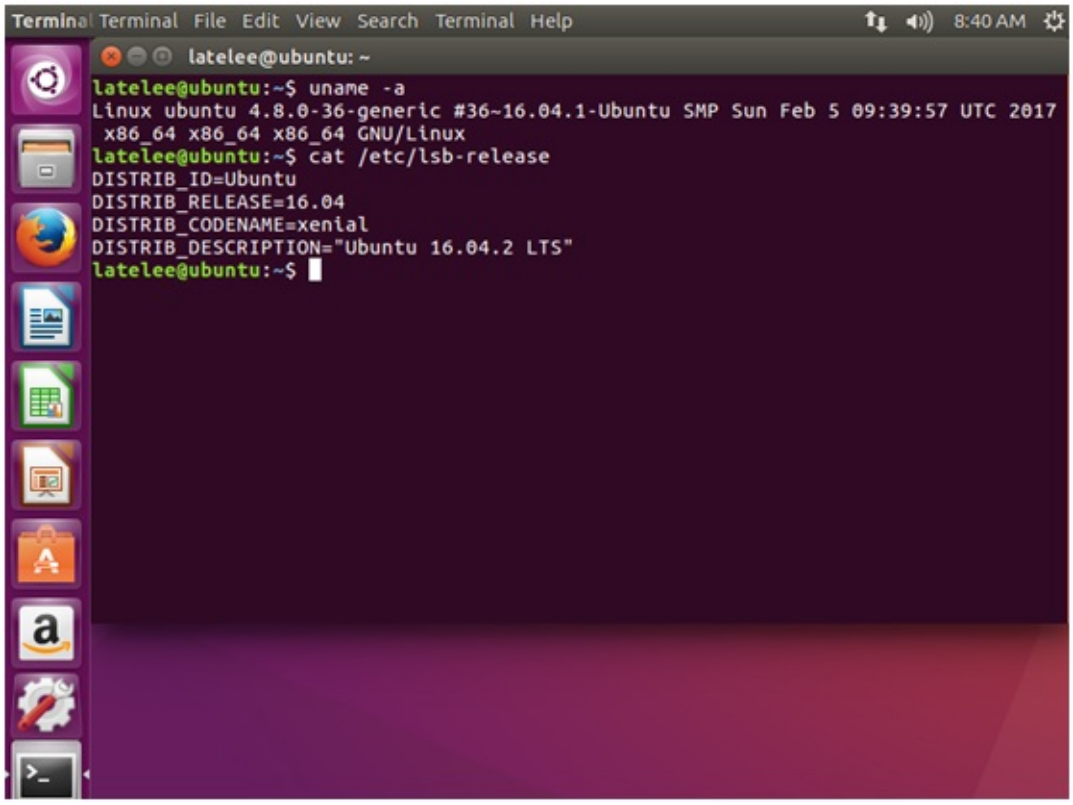

图2.18 系统内核版本及发行版本

使用桥接网络方式，无需额外设置网络参数，即可上网，如图2.19所示。

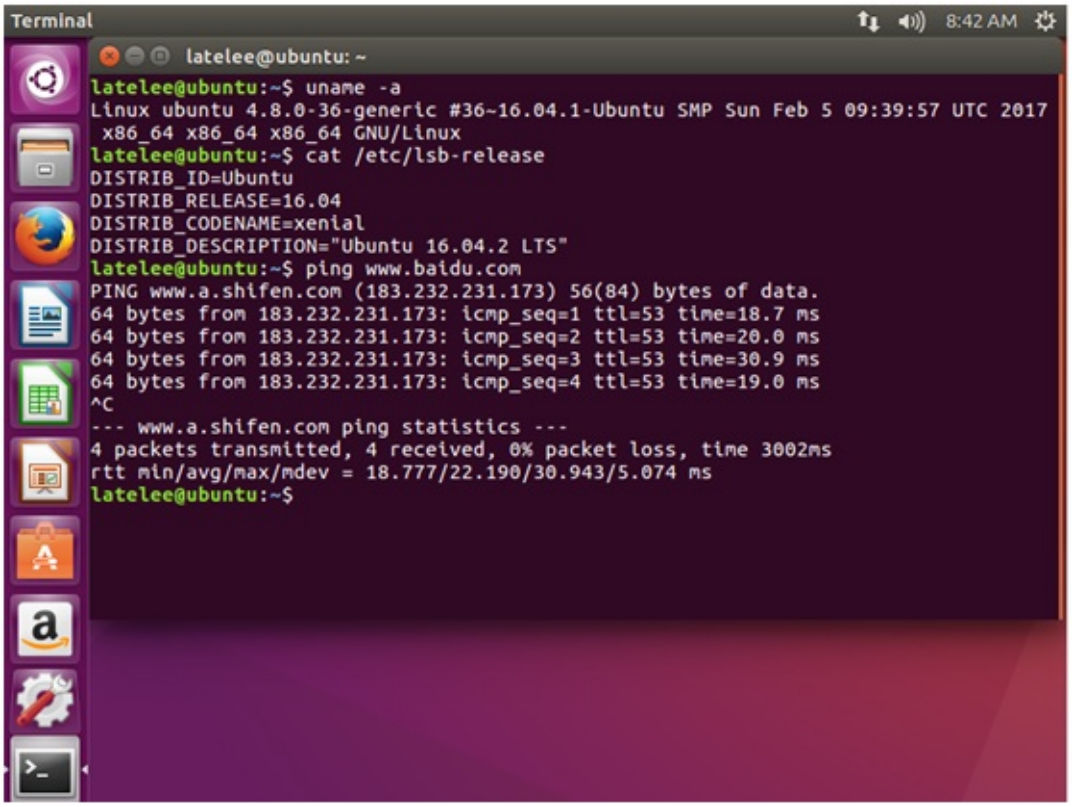

图2.19 与外网连通性测试

最后要说明的是，在VMware Workstation软件中，可以对虚拟机系统的硬件进行重新配置，在VMware 菜单“虚拟机”->“设置”界面中进行设置，如图2.20所示。

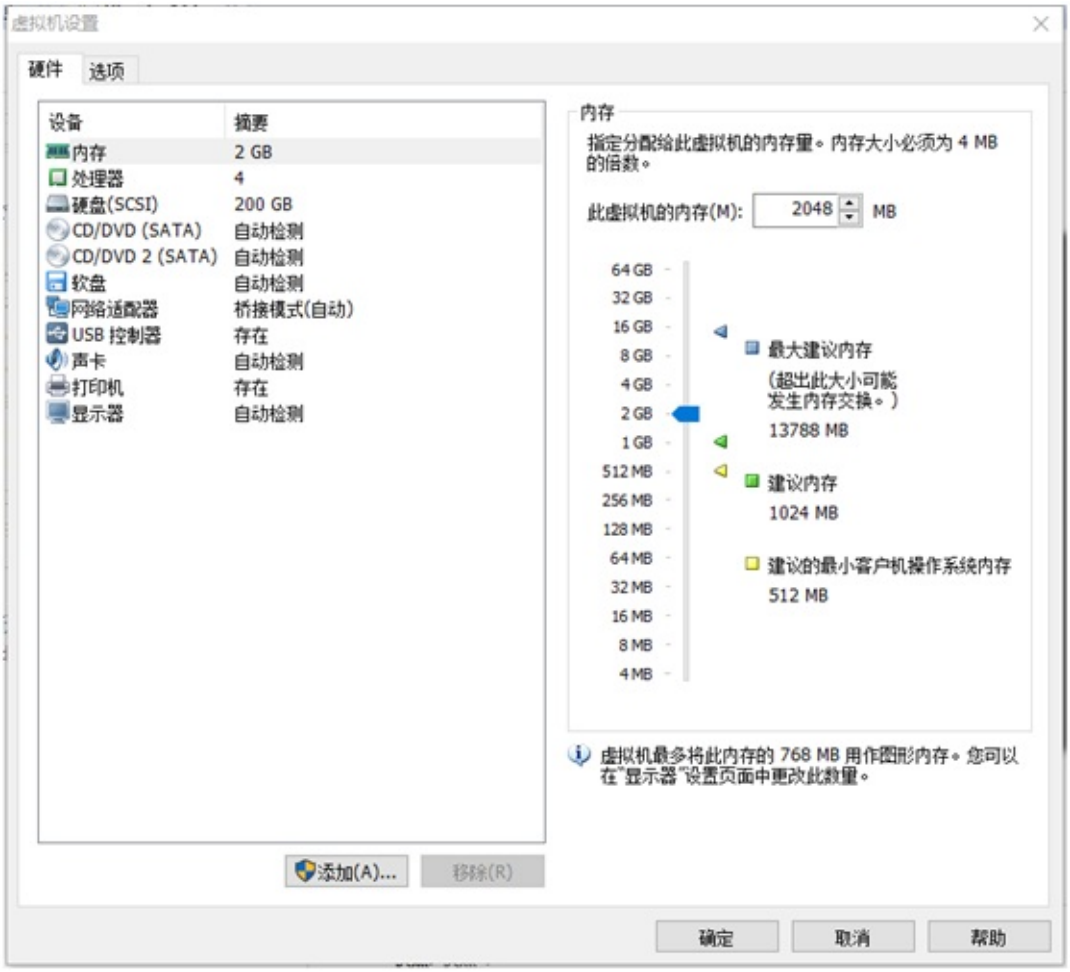

图2.20 虚拟机设置

2.2 安装Ubuntu后要做的配置

刚刚安装完成的Ubuntu系统，离开发使用，还有一段距离。本节就进行一些安装后的配置。

2.2.1 网络设置

由2.1小节知道，虚拟机Ubuntu使用的是桥接方式，是从路由器自动获取地址（DHCP），这种方式存在一个问题，那就是每次关机重启后，IP地址可能会发生变化，这会带来一定麻烦，为此，需要将系统IP设置为静态IP方式。在图2.21选择要配置的网络，点击“Edit”进入编辑界面。

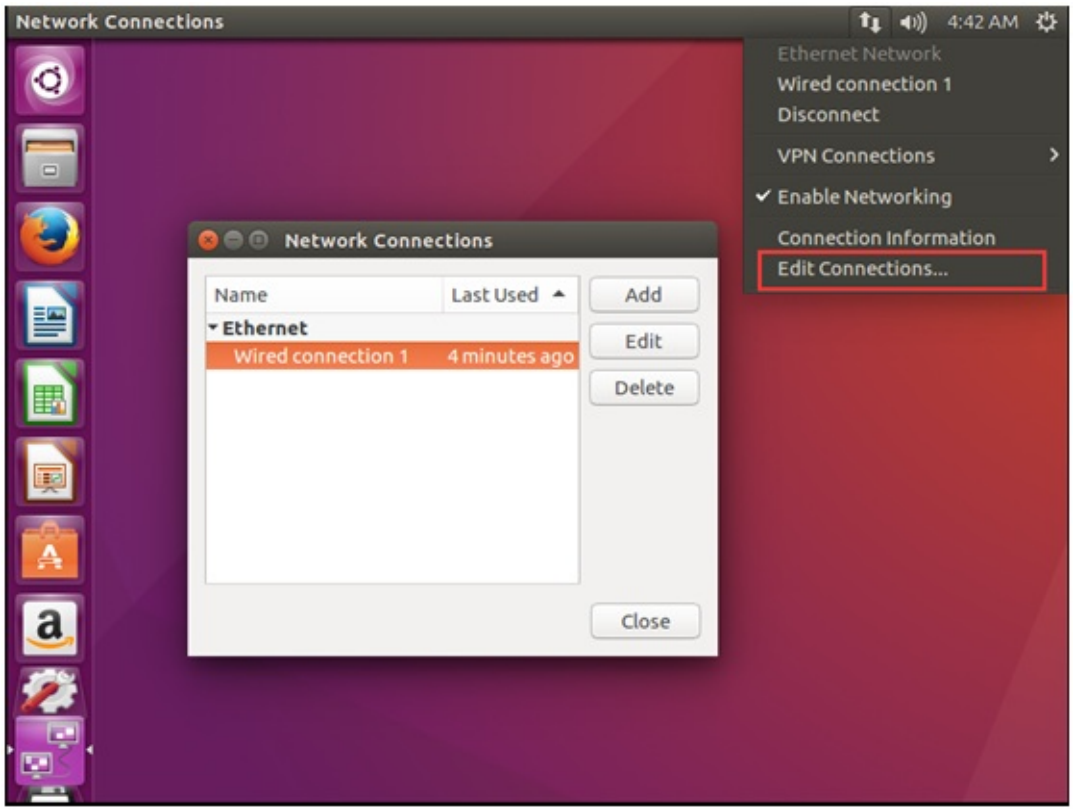

图2.21 编辑网络信息

默认情况下是DHCP，如图2.22所示。

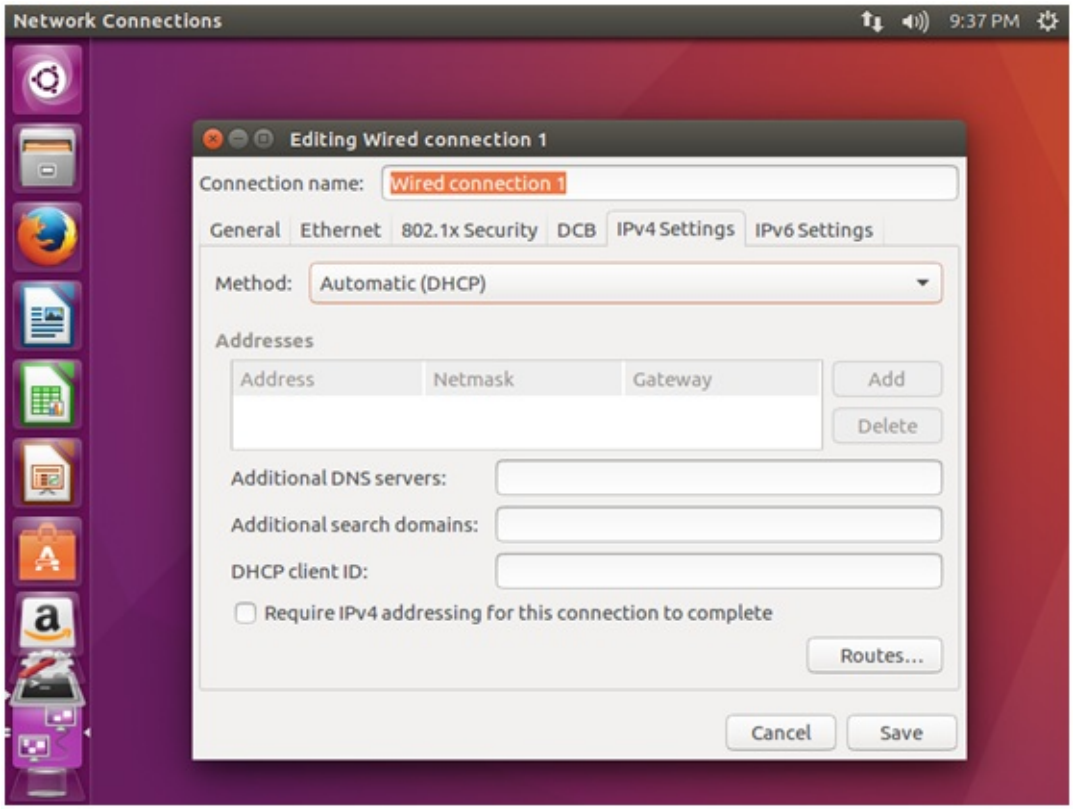

图2.22 IP配置为DHCP方式

由于DHCP获取的IP可能会发生变化，为了使用固定的IP，本书将IP设置为静态IP，如图2.23所示。

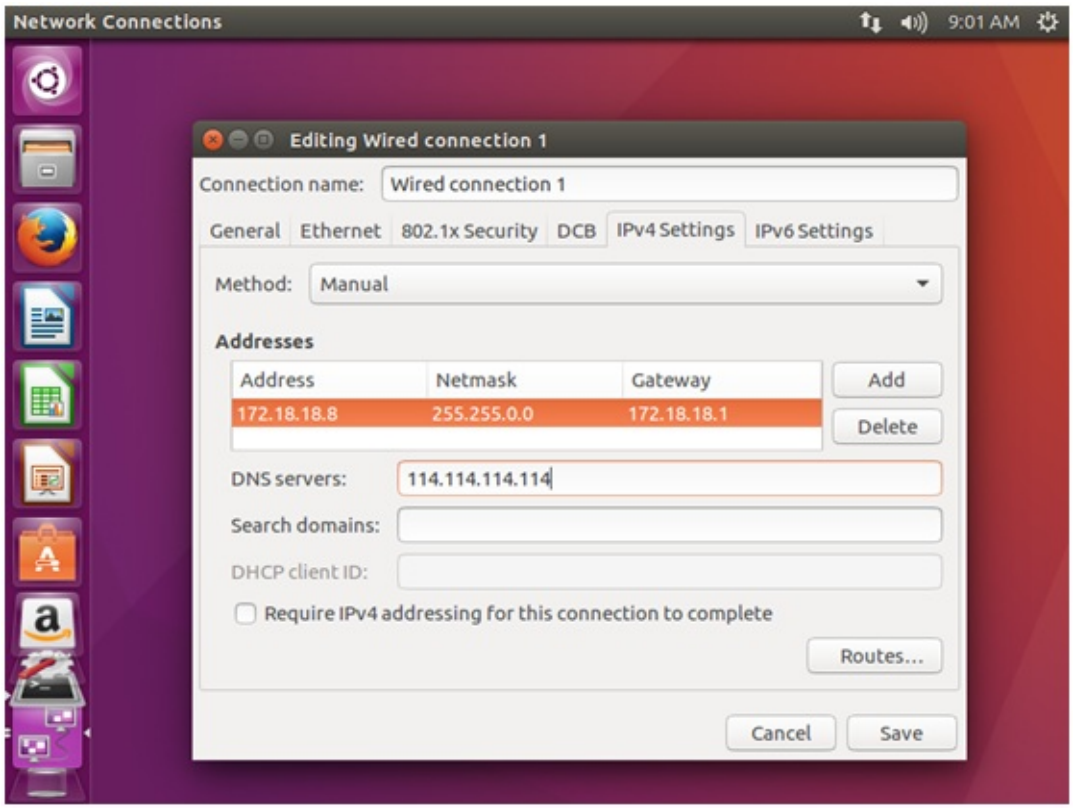

图2.23 IP配置为静态方式

重启虚拟机后，使用ifconfig命令可查询其IP地址，如图2.24所示。

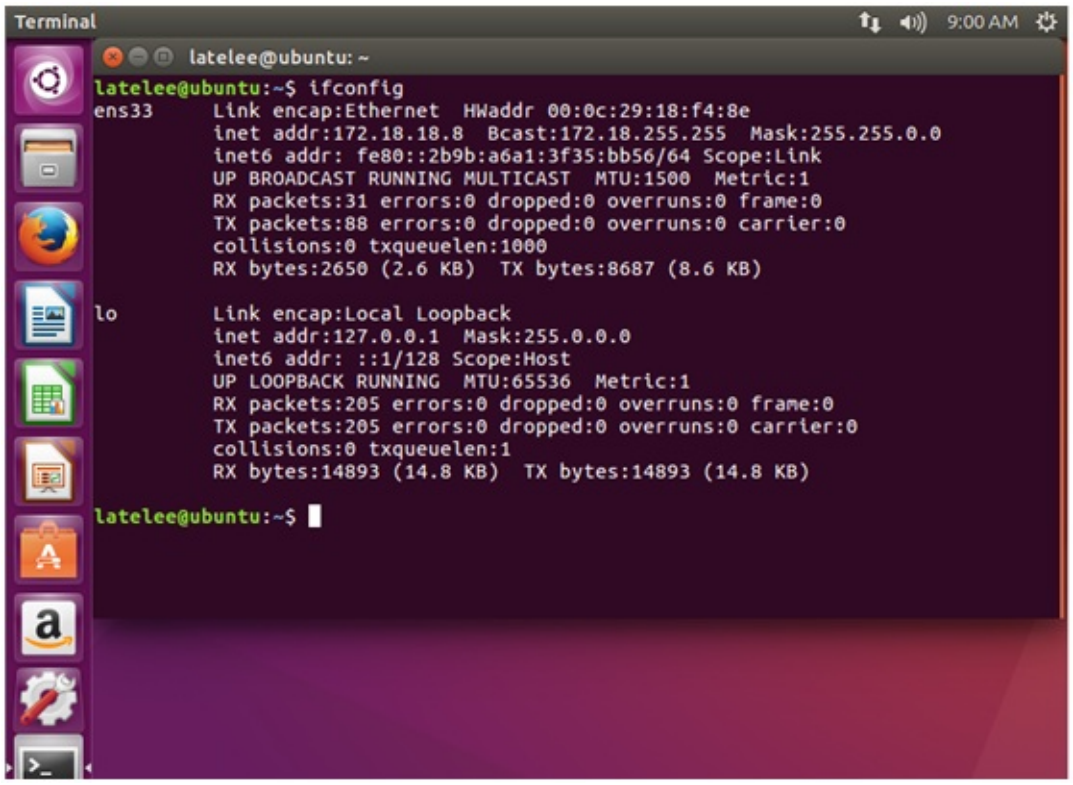

图2.24静态IP生效

这里演示的是网络设置最简单、最方便的方法。还有另一种方法，那就是通过修改相应的配置文件，具体我们在第4章节中进行介绍。

2.2.2 设置源

Ubuntu默认使用国外服务器，有时下载会比较慢。为了加快下载速度，将软件更新源设置为国内。
当然，不进行这个步操作也是可以的，只是影响下载速度而已，读者有兴趣的话，可以使用国内源和国外源进行速度的对比，然后选择最快的源。
要注意的是，设置更新源必须与当前使用的系统版本保持一致。本书使用的16.04版本，开发代号为 `xenial` ，因此，选择的源必须带有 `xenial` 字样。

在虚拟机Ubuntu中，用鼠标单击右键，选择“Open Terminal”，输入以下命令

```
$ sudo cp /etc/apt/sources.list /etc/apt/sources.list.bak # 先对source.list备份
$ sudo gedit /etc/apt/sources.list  # 使用gedit编辑器
```

提示：输入sudo后，表明使用root权限执行命令，按回车键之后，终端界面会有如下提示：
`[sudo] password for latelee:`
此时输入安装虚拟机时设置的密码即可。注意，为了保密性，Linux输入密码过程并没有任何输出。

然后可选择以下任意源地址。
清华大学源

```
deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial main restricted
deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-updates main restricted
deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial universe
deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-updates universe
deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial multiverse
deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-updates multiverse
deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-backports main restricted universe multiverse
deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-security main restricted
deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-security universe
deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-security multiverse
deb http://mirrors.ustc.edu.cn/ubuntu/ xenial main restricted universe multiverse
deb http://mirrors.ustc.edu.cn/ubuntu/ xenial-security main restricted universe multiverse
deb http://mirrors.ustc.edu.cn/ubuntu/ xenial-updates main restricted universe multiverse
deb http://mirrors.ustc.edu.cn/ubuntu/ xenial-proposed main restricted universe multiverse
deb http://mirrors.ustc.edu.cn/ubuntu/ xenial-backports main restricted universe multiverse
deb-src http://mirrors.ustc.edu.cn/ubuntu/ xenial main restricted universe multiverse
deb-src http://mirrors.ustc.edu.cn/ubuntu/ xenial-security main restricted universe multiverse
deb-src http://mirrors.ustc.edu.cn/ubuntu/ xenial-updates main restricted universe multiverse
deb-src http://mirrors.ustc.edu.cn/ubuntu/ xenial-proposed main restricted universe multiverse
deb-src http://mirrors.ustc.edu.cn/ubuntu/ xenial-backports main restricted universe multiverse
```

东北大学源

```
deb-src http://mirror.neu.edu.cn/ubuntu/ xenial main restricted #Added by software-properties
deb http://mirror.neu.edu.cn/ubuntu/ xenial main restricted
deb-src http://mirror.neu.edu.cn/ubuntu/ xenial restricted multiverse universe #Added by software-properties
deb http://mirror.neu.edu.cn/ubuntu/ xenial-updates main restricted
deb-src http://mirror.neu.edu.cn/ubuntu/ xenial-updates main restricted multiverse universe #Added by software-
properties
deb http://mirror.neu.edu.cn/ubuntu/ xenial universe
deb http://mirror.neu.edu.cn/ubuntu/ xenial-updates universe
deb http://mirror.neu.edu.cn/ubuntu/ xenial multiverse
deb http://mirror.neu.edu.cn/ubuntu/ xenial-updates multiverse
deb http://mirror.neu.edu.cn/ubuntu/ xenial-backports main restricted universe multiverse
deb-src http://mirror.neu.edu.cn/ubuntu/ xenial-backports main restricted universe multiverse #Added by softwar
e-properties
deb http://archive.canonical.com/ubuntu xenial partner
deb-src http://archive.canonical.com/ubuntu xenial partner
deb http://mirror.neu.edu.cn/ubuntu/ xenial-security main restricted
deb-src http://mirror.neu.edu.cn/ubuntu/ xenial-security main restricted multiverse universe #Added by software
-properties
deb http://mirror.neu.edu.cn/ubuntu/ xenial-security universe
```

```
deb http://mirror.neu.edu.cn/ubuntu/ xenial-security multiverse
```

网易源

```
deb http://mirrors.163.com/ubuntu/ xenial main restricted universe multiverse
deb http://mirrors.163.com/ubuntu/ xenial-security main restricted universe multiverse
deb http://mirrors.163.com/ubuntu/ xenial-updates main restricted universe multiverse
deb http://mirrors.163.com/ubuntu/ xenial-proposed main restricted universe multiverse
deb http://mirrors.163.com/ubuntu/ xenial-backports main restricted universe multiverse
deb-src http://mirrors.163.com/ubuntu/ xenial main restricted universe multiverse
deb-src http://mirrors.163.com/ubuntu/ xenial-security main restricted universe multiverse
deb-src http://mirrors.163.com/ubuntu/ xenial-updates main restricted universe multiverse
deb-src http://mirrors.163.com/ubuntu/ xenial-proposed main restricted universe multiverse
deb-src http://mirrors.163.com/ubuntu/ xenial-backports main restricted universe multiverse
```

本书选用网易源，编辑过程如图2.25所示。

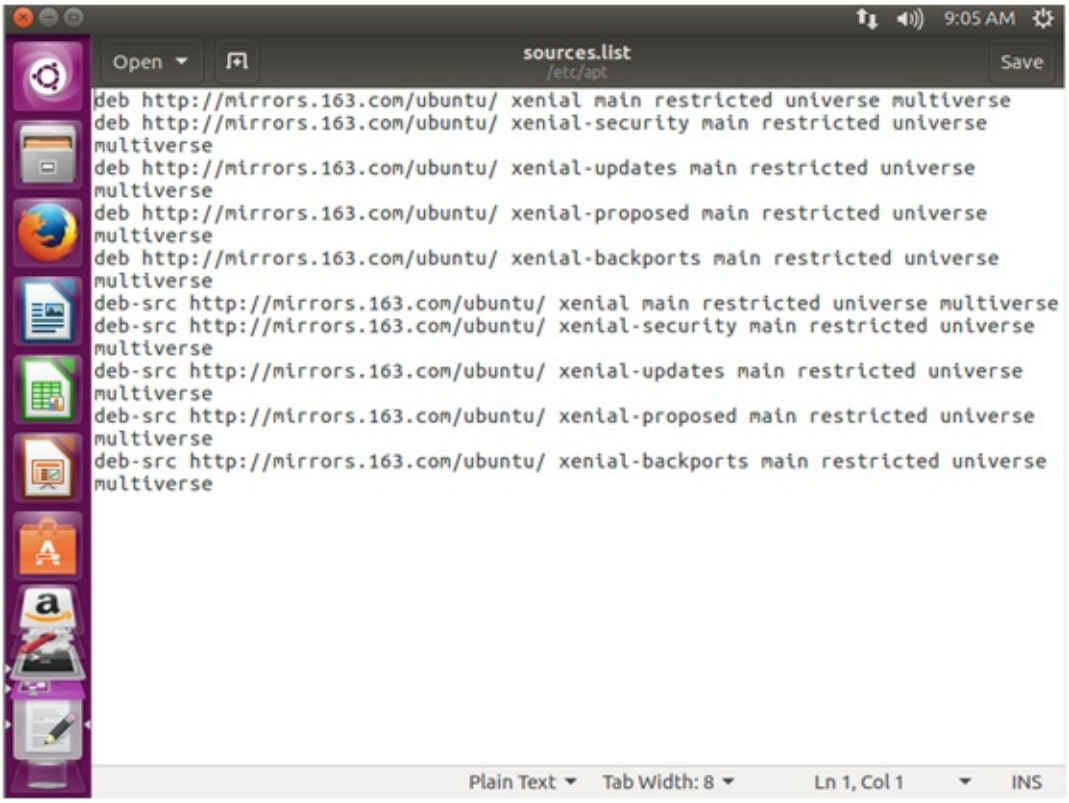

图2.25 设置软件更新源

按“Ctrl+s”保存，然后关闭gedit界面，输入以下命令更新源：

```
$ sudo apt-get update
```

输出信息如图2.26所示：

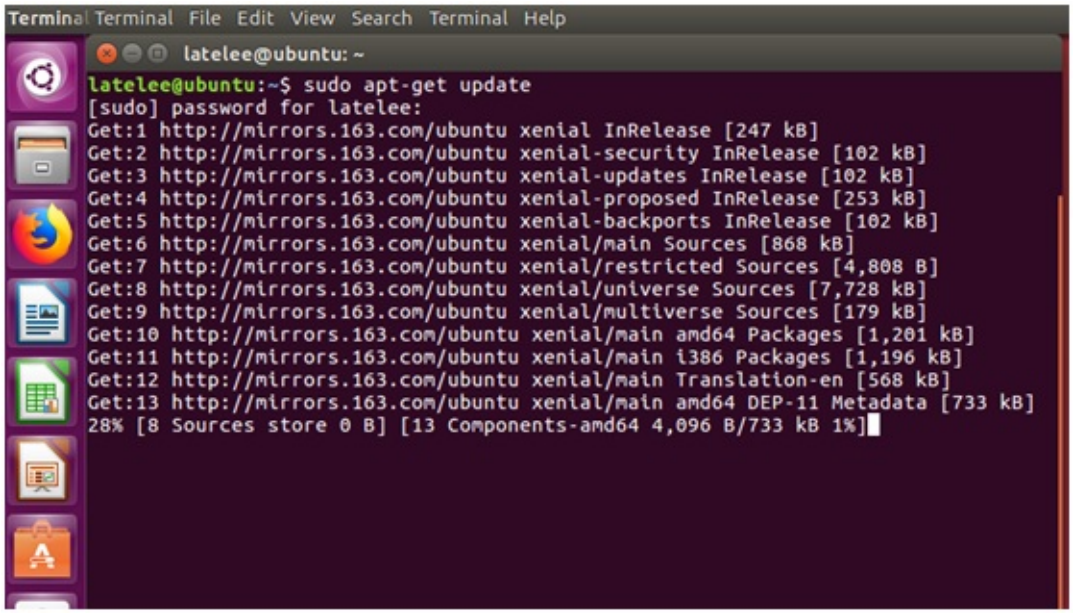

图2.26 更新源过程

2.2.3 安装必备软件

为了减少不必要的搜索工作，本节将大部分后续开发工作中使用到的工具、库的安装列出来，并加上简单说明。在Ubuntu系统安装完毕后，强烈建议安装本小节提到的所有工具和库。工具（或库）与工具（库）之间可能存在依赖关系，如果不安装前置依赖条件，那么，在用apt-get或deb方式安装时，系统会提示缺少的工具（或库）。

以下安装命令中，均使用sudo前缀，表明此命令使用root权限执行，因此需要输入用户名密码（在安装系统时，默认登陆用户名拥有切换root的权限）。如果不使用sudo，则会提示如下错误信息：

```
E: Could not open lock file /var/lib/dpkg/lock - open (13: Permission denied)
E: Unable to lock the administration directory (/var/lib/dpkg/), are you root?
```

当然，也可以直接使用root用户登陆，这样不再需要输入 `sudo` ，但笔者不建议直接使用root用户登陆系统。后面行文为了简洁，有时会省略掉 `sudo` 。

安装开发类软件，编译器、调试器

```
$ sudo apt-get -y install build-essential #GCC、G++等
$ sudo apt-get -y install libncurses5-dev libncurses5 ## 编译内核需要
$ sudo apt-get -y install bc #bc工具，编译内核需要
$ sudo apt-get -y install device-tree-compiler # 设备树编译器
$ sudo apt-get -y install libssl-dev # ssl开发库，编译内核需要
$ sudo apt-get -y install gdb # gdb调试器
$ sudo apt-get -y install flex bison # 词法分析器、语法分析器
$ sudo apt-get -y install autoconf # 自动化编译所需
$ sudo apt-get -y install pkg-config # 编译  查找库路径需要
$ sudo apt-get -y install cmake # 编译工具
```

安装其它工具：

```
$ sudo apt-get -y install git  # 版本控制
$ sudo apt-get -y install unzip # zip解压工具
$ sudo apt-get -y install vim # vim编辑器
$ sudo apt-get -y install dos2unix # 格式转换工具
$ sudo apt-transport-https ca-certificates # https认证
```

安装服务类软件

```
$ sudo apt-get -y install openssh-server # ssh服务器
$ sudo apt-get -y install xinetd telnetd  # telnet服务器
$ sudo apt-get -y install Samba # Samba服务器
$ sudo apt-get -y install nfs-kernel-server # nfs服务器
```

提示：由于作者环境使用的场合有限，不可能将所有的工具（或库）都列举出来。因此，在读者实际操作中就根据自己的情况进行安装。原则就是：缺少什么就安装什么，直到所有依赖条件满足。注意指出的是，一些软件在安装时要进行确认，因此上面的安装命令均加上 `-y` 表示默认同意安装。

2.3 虚拟机Ubuntu的访问

对虚拟机Ubuntu进行操作时，需要切换到虚拟机内部，当需要使用物理机时，还需要切换到外面来。本书使用物理机Windows+虚拟机Ubuntu双系统的环境进行实验。为了提高开发效率，实际操作时，在Ubuntu设置并开启Samba服务，这样Ubuntu的Samba共享目录就成为Windows系统的一个磁盘，从而按Windows的操作习惯来使用Linux的文件系统，比如下载Ubuntu所用的工具，或编写代码、脚本。至于命令的操作，则使用SecrueCRT软件通过ssh协议连接到Ubuntu。SecrueCRT连接如图2.27所示:

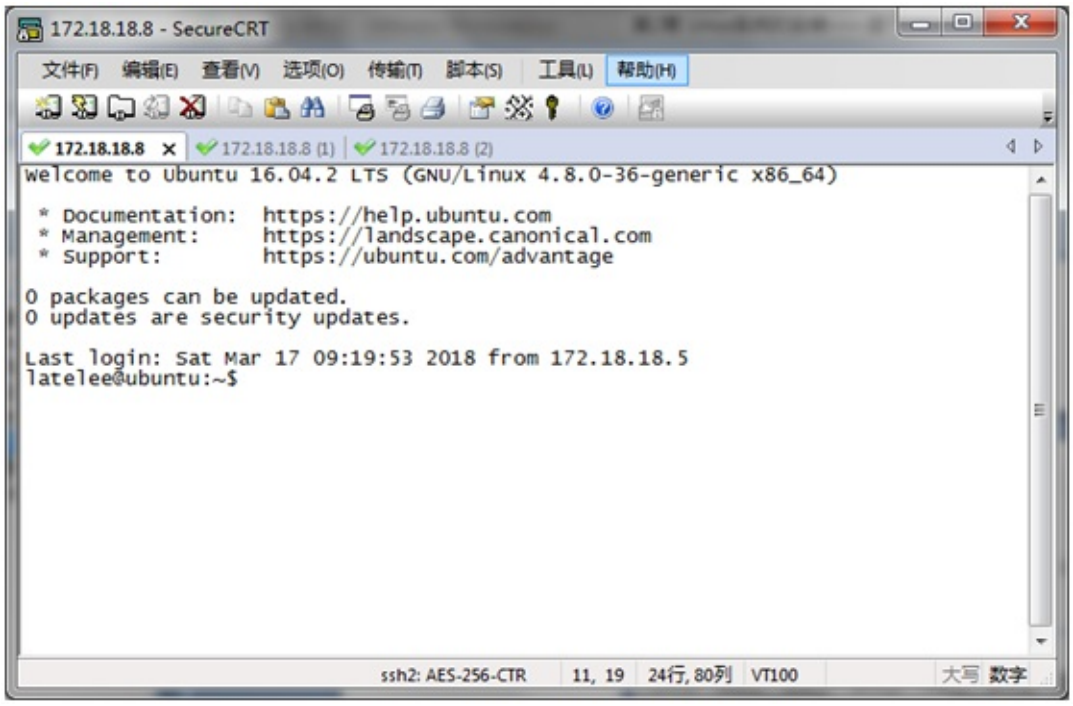

图2.27 SecureCRT同时连接Ubuntu系统多个终端

Samba的安装将在第4章介绍，这里先给出访问示例，如图2.28所示：

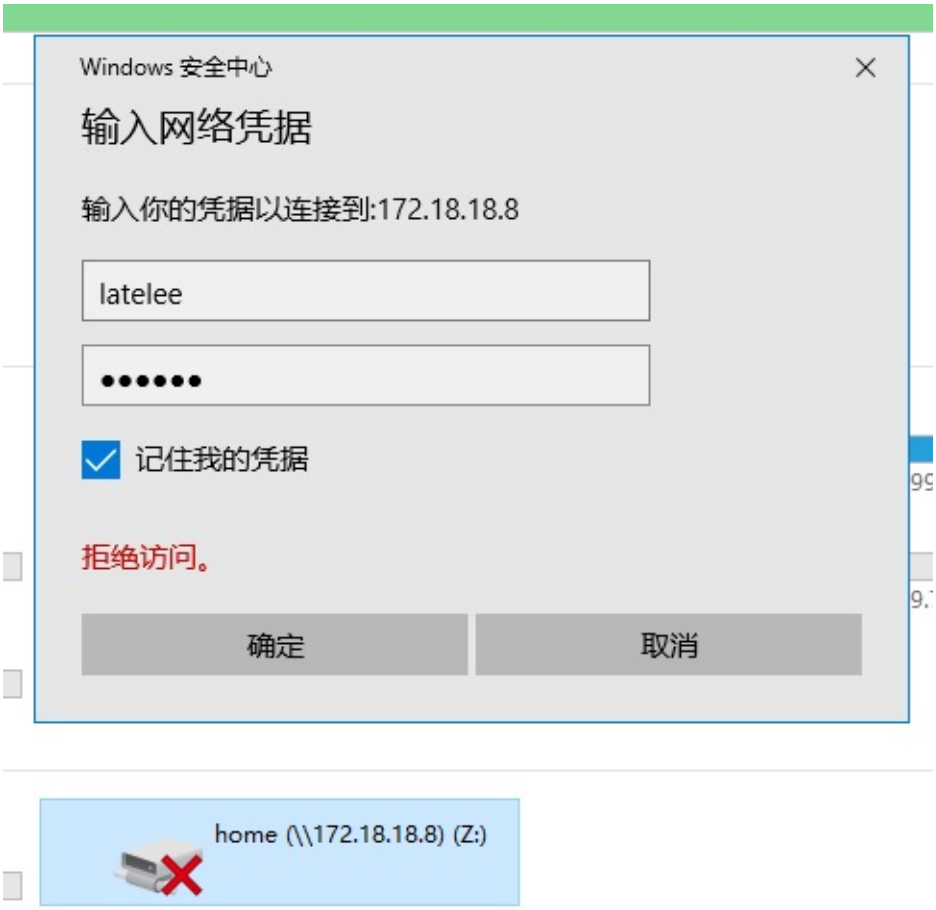

图2.28 Samba访问凭据

输入正确的用户名和密码后，可访问Samba共享目录，图2.29将共享目录挂载到“Z”盘。

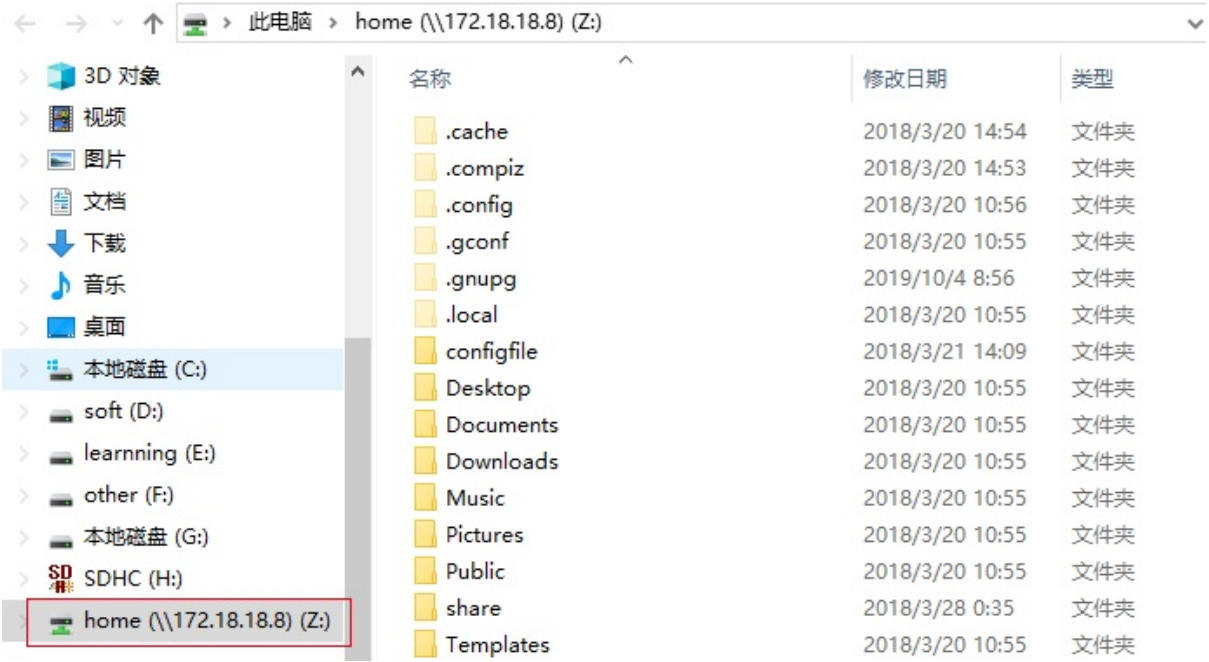

图2.29 “Z”磁盘目录

系统安装成功后，还需要进行一些设置，这些设置将在后续章节中进行说明。

说明：为了更好地实验，作者另外还创建了一个虚拟机。即一共有2台Linux(Ubuntu)系统。一台IP为 `172.18.18.8` ，另一台为 `172.18.18.18` 。在书中会使用这2个IP地址，希望读者在遇到这些IP地址，能自动替换成实际的IP地址（当然，与书中IP地址相同也没问题，起码看起来，这些IP地址的数字比较好看^_^）。

2.4 拓展思考

1、本节的Ubuntu安装于虚拟机中，主要考虑到便捷性，因为这样，一台电脑即能拥有2种系统。如果经济允许情况下，读者可以考虑使用物理机安装Ubuntu系统。

2、虚拟机的网络模式有主机模式、桥接模式和NAT（网络地址转换）模式，为了方便起见，本节使用桥接模式，其它的模式，请读者自行尝试。只要确保虚拟机中的Ubuntu系统能连通互联网、Windows主机即可，方式可自由选择。（待改：如果是NAT，怎么连接硬件板子？）

3、虚拟机有“挂起”功能，能够将Ubuntu系统的当前工作状态“挂起”，等下次启动时，可立刻恢复之前工作状态。类似于“睡眠”功能。

4、虚拟机本身提供了共享机制。有兴趣的读者请自行试验。

5、在开发过程中，虚拟机使用得越多，文件就会越来越多。可通过创建虚拟机快照来解决。

2.5 本章小结

在这一章中，我们以图文的形式对在虚拟机VMware中Ubuntu 16.04的安装过程进行讲解，由于Ubuntu系统使用比较广泛，资源较多，社区建设也较好，因此就选择该系统。另外介绍了安装后进行的一些必要配置，同时列出了很多必装的软件，希望读者能在开始时就安装这些软件。

如果出现此行文字，则说明本书本章节还没有完稿，诸多地方还在编写、修改、审校中。敬请注意。

第3章 Linux系统使用

上一章我们在虚拟机中安装了Ubuntu系统，本章介绍Linux系统的使用。包括Linux系统目录介绍、常见命令、环境变量、软件安装以及内核更新等内容。

一般通常意义的Linux系统包括了内核以及根文件系统里的各种工具、库、文件，还有图形界面等。狭义的Linux只是一个操作系统的内核，而GNU则提供了大量的自由软件供用户使用。Linux系统上大部分工具都来自GNU，如gcc、makefile、emacs、X窗口gnome等。Linux与GNU绝妙的合作造就了现在我们看到的Linux系统，但两者命名上发生了一些争议[注1]，不过我们可以抛却这些争议，关注如何在工作上使用这个系统。

初学者对Linux的第一印象可能是一个黑色的屏幕，上面密密麻麻布满了英文字符，并且快速滚动，这种情况基本上都是命令终端操作界面。现今主流的Linux发行版本，图形界面都比较友好，一些场合的使用媲美于Windows系统。如果读者不习惯命令终端，可以使用图形界面，不过，就笔者的经验来看，大部分情况命令终端更加有利于工作效率的提高。建议读者慢慢从图形界面转变为命令终端界面。实际上，本书大部分的操作，都是在Windows上用SecureCRT工具登陆到虚拟机Linux系统，在命令终端上进行的。关于Linux如何与Windows配合进行工作，后续章节将介绍。

另外，从本章开始，一些操作截图会减少，使用文字形式代替，图示能证明操作的实效，但文字更方便读者的操作。

> 注1. 参见 Wikipedia：http://zh.wikipedia.org/wiki/GNU/Linux%E5%91%BD%E5%90%8D%E7%88%AD%E8%AD%B0。 ↩

3.1 UNIX哲学准则

Linux与UNIX有很大不同，但有些思想却是一脉相传的，在正式开始使用Linux之前，允许笔者花一点篇幅介绍UNIX的哲学，内容来自《Linux/Unix设计思想》[注1]。另外加上笔者的一些小小的备注。读者可稍微了解一下，在后续开发中体会。

（1）小即是美。
相对于同类庞然大物，小巧的事物有着其无可比拟的巨大优势。其中一点就是它们能够以独特有效的方式结合其他小事务，而且这种方式往往是最初的设计者没能预见的。（笔者注：功能设计、函数设计，也遵循这个原则。）

（2）让每一个程序只做好一件事情。
通过集中精力应对单一任务，程序可以减少冗余代码，从而避免过高的开销、不必要的复杂性和缺乏灵活性。（笔者注：功能设计、函数设计，也遵循这个原则。）

(3)尽快建立原型。
大多数人认同“建立原型”是任何项目的一个重要组成部分。在其他方法论中，建立原型只是设计中一个不太重要的组成部分，然而，在Unix环境下它却是达成完美设计的主要工具。（笔者注：建立原型有利于快速掌握项目整体，先见森林，再见树木。）

（4）舍高效率而取可移植性。
当Unix作为第一个可移植系统而开创先河时，它曾经掀起过轩然大波。今天，可移植性早被视作现代软件设计中一个理所当然的特性，这更加充分说明这条Unix准则早就在Unix之外的系统中获得了广泛认可。（笔者注：Linux广泛应用于各种芯片平台就是一个很好的佐证。）

（5）使用纯文本文件来存储数据。
舍高效率而取可移植性强调了可移植代码的重要性。其实可移植性数据的重要性绝不亚于可移植代码。在关于可移植性的准则中，人们往往容易忽视可移植性数据。（笔者注：这里强制的是数据可移植性，实际应用中，不一定限于“纯文本”。）

（6）充分利用软件的杠杆效应。
很多程序员对可重用代码模块的重要性只有一些肤浅认识。代码重用能帮助人们充分利用软件的杠杆效应。一些Unix的开发人员正是遵循这个强大的理念，在相对较短的时间内编写出了大量应用程序。（笔者注：实际上是代码的重复使用，建立在模块化的基础上。）

（7）使用shell脚本来提高杠杆效应和可移植性。
shell脚本在软件设计中可谓是一把双刃剑，它可以加强软件的可重用性和可移植性。无论什么时候，只要有可能，编写shell脚本来替代C语言程序都不失为一个良好的选择。（笔者注：这里强调的是提高工具的效率和移植性，即一套工具通用于其它平台，如果SHELL脚本适合则使用之，如果其它方法合适亦使用之。）

（8）避免强制性的用户界面。
Unix开发人员非常了解，有一些命令用户界面为什么会被称为是“强制性的”用户界面。这些命令在运行的时候会阻止用户去运行其他命令，这样用户就会成为这些系统的囚徒。在图形用户界面中，这样的界面被称为“模态”。（笔者注：也是强调效率。）

（9）让每一个程序都成为过滤器。
所有软件程序共有的最基本特性就是，它们只是修改而从不创造数据。因此，基于软件的过滤器本质，人们就应该把它们编写成执行过滤器任务的程序。（笔者注：数据流应该是：输入-处理-输出）

> 注1. 《Linux/Unix设计思想》的原版为《Linux and the Unix Philosophy》，笔者强烈推荐给各位读者。 ↩

3.2 “一切皆文件”

Linux有一个设计哲学：“一切皆文件”。即Linux中所有内容都是以文件的形式存在的，也是通过文件的形式管理的，文件包括但不限于普通文件、资源文件（如图片、视频、音频，等）、目录（目录也是文件的一种，后面行文中在不影响理解情况下，将“目录”列入“文件”范围）、硬件资源（键盘、鼠标、硬盘、打印机、摄像头、屏幕，等）、网络套接字(socket)。在介绍之前，先引入Linux文件权限的小知识。

文件权限

Linux系统所有文件都有“可读”(r)、“可写”(w)、“可执行”(x)三种权限，分别对应数字4、2、1，如果没有权限，则标记为“-”，每个文件又分“文件拥有者”(u)、“同组用户”(g)、“其它组用户”(o)三种用户等级，每组用户有三种权限，一共使用9个权限位来表示，三种权限的数字可合并为一个数字，如“可读写”合并为6（4+2），“可读可执行”合并为5（4+1）。所以实际编程只有3个数字，如“777”表示所有人都拥有可读、可写、可执行权限，“700”表示文件拥有者具备可读可写可执行权限，其他人没有任何权限，以此类推。另外，还有文件拥有者、创建时间等信息。描述似乎有点复杂，下面是使用 `ls -l` 命令列出的文件的解析：

```
 -   rw-r--r--   1   latelee  latelee  4 Sep 8 13:20    foo.txt
文件   权限位   文件    用户ID   组ID      文件创建的       文件名称
类型           链接数                      日期时间
```

这个文件的属性有：这个foo.txt文件是一个普通文件文件，创建者（即latelee）拥有读写权限，同组用户和其它组用户拥有读权限，换成数字表示为“644”。注意，Linux中的可执行文件，其属性一定要带x，否则无法执行。

（1） 普通文件
常见的txt、html、pdf等后续的文件，图片、音视频文件等资源文件，其实都可以归属为普通文件，Linux并不区别对待它们。文件类型为“-”，表示普通文件。

（2） 目录
文件类型为“d”，表示目录（directory）。目录文件均有可执行权限(x权限)，其功能可理解为“可访问权限”。另外，注意2个特殊的目录，“./”表示当前目录，“../”表示上一层目录。

（3） 硬件资源
硬件设备文件位于/dev/目录下，通过该目录的设备文件，就可以对外设进行操作。文件类型为“c”或“b”，分别表示字符设备(character)、块设备（block）。
（4） 网络套接字(socket)

套接字文件一般隐藏在 /var/run/ 目录下，用于进程间的网络通信。
文件类型为“s”，表示是socket套接字。

（5） 符号链接文件
Linux符号链接文件类似Windows系统的快捷方式，通过 `ln` 命令可以创建链接文件。
文件类型为“l”，表示链接文件（link)。

（6） 管道文件
文件类型为“p”，表示管道文件（pipe)。

命名规则

Linux 系统中，文件和目录的命名规则如下：
（1）理论上，除“/”外，其它字符均可使用。“/”为目录的分隔符，如“foo/bar”被理解为“foo”目录的“bar”目录或文件。但在实际中，特殊字符如“<”、“>”、“?”、“*”、空格等，不建议使用，这会加大程序处理的难度。如果一定要使用，则需要使用引号（""）或“\”转义。比如创建“foo bar”文件（中间有2个空格键），需要使用 `touch foo\ \ bar` 或 `touch "foo bar"` 来创建。
（2） 文件（或目录）名称的长度不能超过255个字符。
（3） Linux系统的文件名称区分大小写，相同字母，但大小写不同，被理解为不同的文件，如foo、FOO、Foo是三个文件。在实际中不建议这样使用。
（4） Linux系统不以文件后缀名来区分文件类型，这点与Windows系统不同。比如“foo.exe”在Windows下为可执行文件，但在Linux，却不一定是可执行文件（当然，也可以将Linux的可执行文件改名为foo.exe，只是如此一来，似乎有点不伦不类）。
（5） 以点号（“.”）开头的文件（或目录）为隐藏文件（在Windows系统亦如是），默认不列出，如果要列出，需要使用 `ls -a` 命令。

标准输入、标准输出、标准错误

当Linux执行程序的时候，默认会自动创建三个流，分别是：标准输入(standard input，缩写为stdin)、标准输出(standard output，缩写为stdout)、标准错误(standard error，缩写为stderr)。标准输出就是正常的信息，如打印信息，输出结果等等。而标准错误一定是错误信息，如不存在的命令、目录、文件的提示信息，等等。

项 目	功能	文件描述符	介质	重定向符号
标准输入	输入源	0	默认键盘	使用"<"
标准输出	输出信息	1	屏幕（默认）、终端、串口	使用">"、">>"(追加)，也可用"| tee"
标准错误	输出错误信息	2	同标准输出	同标准输出

学习过C语言的读者，相信对 `printf("hello world.\n");` 这个打印语句不会陌生，该语句的作用是打印出字符串 `hello world` 。在哪里打印？——就是在执行程序的地方打印。如果是在Linux远程登陆（如用ssh或telnet），则打印到终端；如果是嵌入式设备，则打印到串口终端。标准输出的实体形式不同，但对 `printf` 函数而言，并无差别。因为系统层已经抽象了硬件的细节。

上表的文件描述符，在编程语言和命令中都会使用到。对于某个进程来说，虽然默认总是打开这三个流，但会根据需要使用或不使用。有兴趣的读者可以查看自己系统 `/proc/<进程PID>/fd/` 目录，该目录就是进程的文件描述符，除了0、1、2之外，还会有其它文件描述符。[注1]

管道

管道可以把一个进程的标准输出流与另一个进程的标准输入流连接在一起，用于多命令之间的接连达到复杂的功能，也用于进程间通信。很多UNIX和Linux系统的命令被设计为过滤器，从标准输入中读取输入，将内容输出到标准输出。shell命令用“|”符号创建管道，连接两个或多个命令，即前一个命令的输出作为后一个命令的输入。

下面结合三种流和管道来看看如何使用。整理如下表所示。

项 目	说明
cmd > file	cmd执行结果重定向到文件file。文件不存在自动创建，如已存在则覆盖原有内容

cmd >> file	cmd执行结果重定向到文件file。文件不存在自动创建，如已存在则在原有内容后追加
cmd > file 2>&1	将标准输出和标准错误重定向到文件中。2>&1表示使用后台模式将标准错误重定向到标准输出中
cmd >> file 2>&1	同上，追加到原文件
cmd 2> file	只将错误信息重定向到文件，常用于只记录错误信息的场合。注意，`2`可能是cmd的参数，所以一定注意写法
cmd >& file	同上
cmd 2>> file	同上，追加到原文件
cmd 2>file1 > file2	将错误信息重定向到file1，标准输出重定向到file2
cmd < file	将file的内容作为cmd的输入（可理解为读取文件内容）
cmd < file1 > file2	将file1的内容作为cmd的输入，以file2作为标准输出

下面是一些的实例。

项 目	命令	说明
将`ls`结果输出到foo.txt文件	$ ls > foo.txt	最简单的数据重定向
同上，但使用管道方式	$ ls | tee foo.txt	管道的话，需要使用tee命令
计算文件行、词、字符总数	$ cat < a.txt | wc	直接用`wc a.txt`即可，此处仅作演示
查找指定进程名称	$ ps aux | grep a.out	ps列出所有的进程，再用grep进行查找
查找指定进程文件句柄最大值	$ cat /proc/17835/limits | grep "files"	17835为进程PID
查找指定进程占用的文件句柄	$ ls -l /proc/17835/fd | wc -l	17835为进程PID
查找指定文件的指定字符	$ cat foobar.txt | grep '08:10:50' -n	查找时间为08:10:50的字符串，同时输出行号（如果有特殊字符，最好使用引号）

下面单独整理与标准错误有关的实例。

项 目	命令	说明
过滤掉错误信息	cmd 2 > /dev/null	将cmd输出的错误信息重定向到/dev/null，终端不会再输出
重定向错误信息到文件	cmdnotfound > a.txt	无法得到预期结果，因为`>`默认是重定向标准输出，此时终端会打印错误信息，但a.txt无内容
重定向错误信息到文件	cmdnotfound > a.txt 2>&1	可达到预期结果，此时终端无输出，但a.txt有错误信息，因为命令明确指定将标准错误重定向标准输出（而标准输出默认重定向到文件）
重定向错误信息到文件	cmdnotfound 2>&1 | tee a.txt	可达到预期结果，此时终端有输出，a.txt也有错误信息，因为命令明确指定将标准错误重定向标准输出，然后用管道将标准输出重定向到文件
重定向错误信息到文件	ls 2> a.txt	因为ls输出正确，故终端显示ls的结果，而a.txt无错误内容。

重定向错误信息到文件	ls 2 > a.txt	无法得到预期结果。该命令是列出文件 2 并将输出结果重定向到a.txt，而不是将标准错误 2 的信息重定向到文件。

重定向有重要的作用。

- 终端输出的信息有时候非常多，屏幕无法显示完整内容，此时要将信息保存下来。
- 将错误信息和正常信息（标准输出）分开处理，方便分类和查看。
- 如果明确知道要丢弃错误信息，则将标准错误重定向到/dev/null即可。
- 有些系统命令执行时可能没有合适的终端或还没有输出终端的机会，此时可保存信息到文件中。

Linux与Windows在文件机制上的差别

项 目	Linux	Windows
大小写	大小写敏感	大小写不敏感
文件后缀	无后缀名概念	有后缀名概念
隐藏文件	隐藏文件前带“.”	同Linux
权限	区分root权限和普通用户权限	有管理员权限和普通用户权限
可执行属性	显式的可执行属性，可手动去掉	不明显
链接属性	使用ln命令创建	相当于快捷方式
保留文件	权限允许即可创建	不能创建包括“aux”、“com1”、“com2”、“comN”、“con”以及“nul”等名称的文件

针对大小写问题，再说明一下，Linux内核源码有部分文件名称是以大小写区分不同文件的，如果在Windows系统解压，这些“同名文件”会被覆盖掉，只保留一个文件，因此，Linux内核开发者需要在Linux系统中进行解压、管理内核文件。

> 注1. 如果该进程不用到三个默认的流，会将它们链接到 `/dev/null` 文件。另外，如果程序只打开文件而不关闭的话，该目录下会有大量文件描述符，如果超过系统限制值，则会导致资源不可用，这也是排查问题的一个参考。 ↩

3.3 Linux系统目录

Linux系统的目录遵循FHS(Filesystem Hierarchy Standard，文件系统层次结构标准)标准[注1]。FHS采用树形结构组织文件，根为“/”，也称为根目录——即所有文件、目录都位于根目录“/”下面。早期各个Linux厂家各自定义自己的Linux目录结构，比较混乱，因此FHS在较高层面上定义了各个目录及其作用，但是还是有很多目录是由各个厂家自定义的，如果在根目录下列出目录，可以看到大部分Linux系统几乎都一样，但实际上其中的子目录及各目录用途还是有很大差别的。

下面根据FHS标准列出必须存在的目录：

目录	说明
/	根目录
bin	基本的命令文件
boot	启动所需文件，如内核镜像文件，启动配置文件
dev	设备文件
etc	系统配置文件
lib	静态/动态库，以及内核模块文件
media	媒体文件挂载
mnt	文件系统挂载
opt	存放用户程序文件
sbin	基本的系统命令
srv	存放系统服务数据
tmp	临时文件目录，重启后内容会被清除
usr	UNIX系统资源目录，十分复杂和庞大
var	包含可变的数据

在FHS中，home目录和root目录是可选的目录，但基础上所有发行版本都会带有这两个目录。下面使用 `tree / -L 1` 列出本书所用的Ubuntu系统根目录下的目录和文件：

```
/
├── bin
├── boot
├── cdrom
├── dev
```

```
├── etc
├── home
├── initrd.img -> boot/initrd.img-4.8.0-36-generic
├── lib
├── lib32
├── lib64
├── lost+found
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin
├── snap
├── srv
├── sys
├── tmp
├── usr
├── var
└── vmlinuz -> boot/vmlinuz-4.8.0-36-generic

23 directories, 2 files
```

最后一行所述的2个文件分别是vmlinuz和initrd.img，其它为目录。

下面介绍一些重要的常用的目录。

/dev目录

/dev主要是硬件设备目录，与硬件打交道都是通过该目录的文件，我们编写的驱动产生的硬件文件也是在此目录。简介如下：

文件	说明
/dev/hd[a-t]	IDE设备
/dev/sd[a-z]	SCSI设备
/dev/md[0-31]	软raid设备
/dev/loop[0-7]	本地回环设备
/dev/ram[0-15]	内存文件
/dev/null	无限数据接收设备，可将任何内容写到该文件，而不会有影响
/dev/zero	无限零资源，可藉此文件创建指定大小的文件、磁盘镜像
/dev/tty[0-63]	虚拟终端，当用户登录时，就使用虚拟终端
/dev/ttyS[0-N]	串口设备
/dev/ttyUSB[0-N]	USB串口设备
/dev/pts/[0-N]	虚拟终端
/dev/console	Linux系统控制台
/dev/fb[0-31]	framebuffer
/dev/random	随机数设备
/dev/urandom	随机数设备 加解密程序会使用这个设备产生随机数（如SSH、SSL）

注：表格的数字仅作示例，不同系统，其值范围可能不同。

不同的芯片平台生成的串口终端设备名称会有不同，PC领域中，大部分使用8250驱动，其名称为“ `ttyS*` ”，三星生产的部分芯片（S3C24XX）的串口终端设备名称为“ `ttySAC*` ”，而TI公司生产的Omap系列芯片则命名为“ `ttyO*` ”，飞思卡尔（NXP已收购）的imx系列芯片命名为“ `ttymxc*` ”，本书使用的vexpress v9则是“ `ttyAMA*` ”。虽然名称千变万化，但其本质未变。

console为Linux系统控制台，需要在内核启动参数在将其映射到真正的终端上，PC上一般设置为“console=ttyS0”，至于ARM平台，则根据厂家不同从而设置为不同的名称，如“console=ttySAC0”、“console=ttyO0”、“console=ttyAMA0”，等等。

/etc目录

系统大部分的配置文件都存储于该目录。包括系统用户账号密码、启动脚本、服务配置，等等。这个目录只有root权限才可修改，但基本上都允许系统用户查看。其中一些目录介绍如下：

文件/目录	说明
/etc/init.d/	系统启动或注销脚本
/etc/rc[0~6].d	不同等级的启动文件，其目录均是链接文件，真实文件位于/etc/init.d/。“S”开头表示启动时执行的脚本，“K”表示注销时执行的脚本
/etc/rc.local	本地启动脚本，在/etc/rc[0~6].d内的所有脚本执行之后，此文件才会被执行
/etc/passwd	系统用户的配置文件，存储了系统中所有用户的基本信息，除了可以登陆的用户外，还有如ssh、telnet、www、daemon、nobody等
/etc/shadow	密码文件
/etc/samba/	samba服务配置文件目录
/etc/exports	NFS服务配置文件

/lib目录

/lib目录为库文件大本营。有些64位发行版会额外创建/lib32/和/lib64/目录，分别存放32位系统和64位系统的库文件。

目录	说明
/lib/i386-linux-gnu/	32位系统库目录
/lib/x86_64-linux-gnu/	64位系统库目录
modules/	驱动模块（ko文件）目录
ld-linux.so.2	链接器文件

库文件有其独特的命名方式，一般形式为 `lib<库名称><版本号>.so` ，同时还有对应的链接文件，形式为 `lib<库名称>.so.<版本号>` 。比如C语言库文件为libc-2.23.so，其对应的链接文件为libc.so.6，链接器文件为ld-2.23.so，对应的链接文件为ld-linux.so.2。

/usr目录

/usr是UNIX System Resource（UNIX系统资源）的缩写，但很多人理解为 `user` （用户）的缩写，笔者认为这样理解亦无不可，当成一个“美丽的错误” 。因为在/usr目录下，有很多内容都是使用这个系统的“用户” 的。目录简述如下：

目录	说明
/usr/bin/	存放系统命令，普通用户和超级用户都可以执行。这些命令和系统启动无关，在单用户模式下不能执行
/usr/sbin/	存放根文件系统不必要的系统管理命令，如多数服务程序，只有root可以使用。
/usr/local/	手工安装的软件保存位置。
/usr/src	源码包保存位置。
/usr/include	C/C++ 等编程语言头文件的放置目录
/usr/lib/	应用程序调用的函数库保存位置
/usr/share/	应用程序的资源文件保存位置，如帮助文档、说明文档和字体目录
/usr/share/zoneinfo	时区文件
/usr/share/man	man帮助文档目录
/usr/share/fonts/	字体目录

/var目录

/var目录一般存放程序的数据和产生的日志。

目录	说明
/var/lib/	存放系统命令，普通用户和超级用户都可以执行。这些命令和系统启动无关，在单用户模式下不能执行
/var/log/	日志目录
/var/run/	进程数据
/var/www/	web服务目录，如Apache会使用此目录
/var/cache/	软件安装缓存数据目录

/proc目录

该目录为虚拟文件系统。一些硬件信息，如内核、进程、外设、网络状态等信息，可以通过这个目录对应的文件获取。

文件	说明
/proc/	以进程PDI名称目录，保存该进程的执行信息
/proc/cpuinfo	CPU信息，如核心数，频率，指令集等
/proc/meminfo	内存信息，如总内存数，空闲内存数等
/proc/vmstat	虚拟内存信息
/proc/interrupts	系统中断信息
/proc/devices	系统外设列表
/proc/cmdline	系统启动参数
/proc/version	内核版本信息，另外可用uname命令查看
/proc/uptime	系统启动后运行的秒数以及空闲的秒数，可算出CPU负载，uptime命令即读取此文件信息

/proc目录在实际工作中应用十分广泛，通过/proc/cpuinfo查看CPU是否支持浮点数，是否支持特定指令（如neon、mmx、sse)；通过/proc/interrupts查看网络、串口是否有中断发生；通过/proc/net下面的文件了解网络设备名称、连接状态，等等，不一而足。

/sys目录

该目录也是虚拟文件系统。和/proc/目录类似。用于将系统中的设备组织成层次结构，在用户空间可以对外设进行读取或通信等操作。

文件	说明
/sys/block	块设备
/sys/bus	总线信息，按总线类型将设备分类
/sys/class	系统设备信息
/sys/dev	储存设备号文件，包括主、次设备号。
/sys/devices	设备信息
/sys/firmware	系统固件
/sys/fs	文件系统相关信息
/sys/kernel	存储内核可调整的参数，或者驱动的调试信息
/sys/module	系统的模块信息（注：不存储ko文件）
/sys/power	电源相关信息

关于/sys/devices目录，一般来说，所有的物理设备都按其在总线上的拓扑结构来显示，但有两个例外即platform设备和system设备，前者是挂到platform总线的外设，后者是芯片内部模块，如CPU、定时器等。

/home目录

普通用户的HOME目录在/home目录下，以以用户登陆名为名称的目录，如用户`latelee`的HOME目录，是`/home/latelee`，**root**用户比较特殊，其HOME目录为/root。系统默认创建了桌面、音乐、下载、图片、视频等目录，不过这些目录在开发中较少用到。另外，也存储用户的配置信息，如ssh、git、环境变量等配置文件，就在用户的HOME目录。注意，不同登陆用户，HOME目录不同，权限不同，配置的信息亦不同。HOME目录可使用`~`表示。

几个bin目录

/sbin、/bin、/usr/sbin、/usr/bin、/usr/local/sbin、/usr/local/bin这几个目录，重要等级程度是不同的，带`s`(`system`)表示为系统级别命令，路径越短越重要，/usr/local/sbin和/usr/local/bin基本上很少命令了，因为其它目录已经差不多完备了。
/sbin是系统必要的命令所在目录，如重启(reboot)、磁盘修复（fsck）、磁盘分区（fdisk）。
/usr/sbin则存放不那么十分重要的系统命令（但也是必不可缺少的），如添加用户（useradd）、删除用户（userdel）、grub等。
/bin是用户级别的命令存储目录，常用的命令几乎都在这个目录中，如ls、echo、cp、mkdir、pwd、rm、mount，等等。
/usr/bin是/bin的一个重要补充，同样包含大量命令，如编辑器（vim）、编译器（gcc)、其它语言（nodejs、python）、find，等等。

这些目录都已经设置到PATH环境变量中。理论上将可执行二进制文件放到任何的目录，都可以直接使用而不需要路径。但实际上一般按FHS规定的实行。另外，我们也添加自定义目录到PATH中，比如笔者一般将交叉编译器、自编译的测试工具放到个人HOME目录的bin目录，这样管理起来比较方便。

绝对路径与相对路径

本节开头处提到“/”为根目录，Linux所有目录都位于其下，各目录使用“/”隔离。如果从路径角度看，“/”为绝对路径的起点。凡是以“/”开始的路径均是绝对路径，不以“/”开始的路径就是相对路径，相对路径，顾名思义，是当前路径的相对位置。这个概念要结合下面提到的特殊目录来理解才会比较深刻。

Linux系统有几个特殊目录，如下表所示：

目录	说明
./	本目录
../	上一层目录，可多次组合使用
~	用户的HOME目录
-	上一工作目录

假设当前目录为“/home/latelee/linux_ebook/foobar”，“./a.out”表示当前目录的a.out可执行二进制文件（绝对路径为“/home/latelee/linux_ebook/foobar/a.out”），而“/a.out”表示根目录“/”下的a.out文件（绝对路径为“/a.out”），“../a.out”表示当前目录的上一层目录的a.out文件（绝对路径为“/home/latelee/linux_ebook/a.out”）。

操作当前目录的文件（含目录）可以省略“./”，比如 `ls a.out` 就是查看当前目录a.out文件信息，`cd foobar` 是切换到当前目录的子目录foobar，等。但是执行二进制文件却不适用，`ls` 是执行/bin目录的ls程序（等效于执行 `/bin/ls`），而 `./ls` 是执行当前目录的ls程序（当前目录无ls文件的话，将会出错）。

绝对路径安全可靠，但要输入的字符太多，而相对路径灵活好用，建议启动脚本中使用绝对路径，在日常操作中常相对路径。

使用cd命令切换目录时，要记得当前的目录，如果不知道，则使用pwd命令查看。

后续我们会根据FHS构建我们自己的根文件系统，届时可回头看本节，并且与虚拟机的Linux发行版目录对比一下，找找其中的异同。这里预先说明一下，嵌入式中秉承的是简约原则，不需要的东西，一概不使用。

> 注1. 更多关于FHS的介绍，参考http://www.pathname.com/fhs/pub/fhs-2.3.html。 ↩

3.4 Linux命令

Linux命令非常多，每个命令都有很多选项参数，有些命令常用，而有些却不常用，如果都进行详细解释，篇幅过长且不必要，本节根据笔者经验，将日常开发所用到的命令进行介绍，并结合实例说明。
本节不会列出所有的命令，如果需要了解更多，请自行学习。对于每个命令的完整帮助信息，请使用 `man <命令名称>` 查看。

本小节命令形式中的“<>”表示必填内容，“[]”表示选填内容。

3.4.1 基本命令

ls：列出文件

功能

ls命令主要用于列出当前目录（默认选项）或指定目录的文件，根据选项不同，所得结果亦不同。

语法

```
ls [目录] <参数>
```

或

```
ls <参数> [目录]
```

参数

参数	说明
-a	列出所有目录、文件，包括隐藏文件，以及本目录“.”、上一级目录“..”
-A	同-a，但不列出本目录和上一级目录
-d	仅列出目录
-f	直接列出结果，而不进行排序 (注：ls默认以文件名称排序)

-h	将容量以人类易读(human-readable)的方式列出来，如KB、MB、GB等
-l	长数据串行出，包含文件的属性等内容，主流发行版本将“ls -l”设置别名“ll”
-r	将排序结果反向输出，与-S、-t连用
-R	连同子目录内容一起列出来
-S	按文件容量大小排序，容量大在前面
-t	按时间排序，新的时间在前面

实例

按文件大小顺序列出文件：`ls -lSh`，示例：

```
$ ls -lSh
total 74M
-rwxrw-r-- 1 latelee latelee  54M Apr 23 12:35 qemu-4.0.0.tar.xz
-rwxr--r-- 1 latelee latelee  21M Apr 14 23:56 h264cut.mp4
-rwxr-xr-x 1 latelee latelee  30K Sep 13 09:37 a.out
-rwxr--r-- 1 latelee latelee  28K Sep 13 08:57 img2.29.png
drwxrwxr-x 2 latelee latelee 4.0K Sep 13 09:40 helloworld
-rwxr--r-- 1 latelee latelee  456 Sep 13 09:13 note.txt
```

按文件大小倒序列出文件：`ls -lShr`，示例：

```
$ ls -lShr
total 74M
-rwxr--r-- 1 latelee latelee  456 Sep 13 09:13 note.txt
drwxrwxr-x 2 latelee latelee 4.0K Sep 13 09:40 helloworld
-rwxr--r-- 1 latelee latelee  28K Sep 13 08:57 img2.29.png
-rwxr-xr-x 1 latelee latelee  30K Sep 13 09:37 a.out
-rwxr--r-- 1 latelee latelee  21M Apr 14 23:56 h264cut.mp4
-rwxrw-r-- 1 latelee latelee  54M Apr 23 12:35 qemu-4.0.0.tar.xz
```

按时间由新到旧排列：`ls -lt`，示例：

```
$ ls -lt
total 74940
drwxrwxr-x 2 latelee latelee     4096 Sep 13 09:40 helloworld
-rwxr-xr-x 1 latelee latelee    30668 Sep 13 09:37 a.out
-rwxr--r-- 1 latelee latelee      456 Sep 13 09:13 note.txt
-rwxr--r-- 1 latelee latelee    28573 Sep 13 08:57 img2.29.png
-rwxrw-r-- 1 latelee latelee 55628624 Apr 23 12:35 qemu-4.0.0.tar.xz
-rwxr--r-- 1 latelee latelee 21035750 Apr 14 23:56 h264cut.mp4
```

按时间由旧到新排列：`ls -ltr`，示例：

```
$ ls -ltr
total 74940
-rwxr--r-- 1 latelee latelee 21035750 Apr 14 23:56 h264cut.mp4
-rwxrw-r-- 1 latelee latelee 55628624 Apr 23 12:35 qemu-4.0.0.tar.xz
-rwxr--r-- 1 latelee latelee    28573 Sep 13 08:57 img2.29.png
-rwxr--r-- 1 latelee latelee      456 Sep 13 09:13 note.txt
-rwxr-xr-x 1 latelee latelee    30668 Sep 13 09:37 a.out
drwxrwxr-x 2 latelee latelee     4096 Sep 13 09:40 helloworld
```

备注

下面汇总一下用ls可以查看的信息。

参数	说明
r	读取权限，数字代号为“4”
w	写入权限，数字代号为“2”
x	执行或切换权限，数字代号为“1”
-	不具任何权限，数字代号为“0”
l	链接文件
d	目录

cd：切换目录

功能

cd为切换目录（change directory)的命令，其后加所要切换的目录路径即可。

语法

```
cd <目录>
```

实例

`-` 为上一个目录，实际是 `$OLDPWD` 的值，无论目录路径有多深，只需要使用 `cd -` 就能返回上一次目录。注意，当系统初次登陆时，`$OLDPWD` 还未设置，所以无法使用切换到“上一个目录”，错误信息如下：

```
$ cd -
-bash: cd: OLDPWD not set
```

pwd：显示当前目录

功能

pwd作用是打印当前工作目录(print working directory)，直接输入pwd即可显示当前目录。

语法

```
pwd
```

备注

在脚本中也会常常使用 `$PWD` 环境变量来表示当前目录。

mkdir：创建目录

功能

创建目录。

语法

```
mkdir <目录>
```

实例

默认情况下，mkdir只能创建不存在的目录，如果目录已存在，会提示出错，错误信息如下：

```
$ mkdir helloworld/
mkdir: cannot create directory ‘helloworld/’: File exists
```

在实际中，常常使用 `-p` 参数，加上该参数后，不管目录存在不存在，都不会报错。使用 `mkdir -p` 还可以一次性创建多个目录。

```
$ mkdir helloworld/hello/world  ## 此命令会提示目录不存在
mkdir: cannot create directory ‘helloworld/hello/world’: No such file or directory
$ mkdir -p helloworld/hello/world ## 添加-p选项后，可一次性创建2个不存在的目录
```

rmdir：删除空目录

功能

删除空目录。

语法

```
mkdir <空目录>
```

备注

注意，当目录不为空时，无法使用rmdir删除，但可以使用 `rm -r` 命令删除。

rm：删除文件

功能

删除目录或文件

语法

其命令形式为

```
rm [-rf] <目录或文件>
```

参数

参数说明如下：

参数	说明
-f	强制删除，不管是否存在，也不会提示
-r	递归删除子目录或子目录文件
-i	删除前均提示（注：有些服务器为安全起见，使用此参数）
-d	删除空目录

实例

如果删除当前目录所有内容，直接使用" `*` "匹配，即执行命令 `rm -rf *` 。rm命令有风险，要按下回车键之前，一定要再三确认清楚，否则数据被删除了，恢复会十分困难，所以要格外小心。著名开源项目bumblebee曾经因为一个空格键的笔误导致系统无法正常运行[注1]。

注意，特殊目录“./”和“../”是无法删除，提示如下：

```
rm: refusing to remove '.' or '..' directory: skipping '.'
rm: refusing to remove '.' or '..' directory: skipping '..'
```

mv：移动文件

功能

移动文件（或重命名）

语法

```
mv <文件1> <文件2>
```

参数

参数	说明
-f	强制移动，不管目标文件存在不存在
-i	若目标文件已经存在，就会询问是否覆盖
-u	若目标文件已经存在，且比目标文件新，才会更新

实例

备注

注意，特殊目录“./”和“../”是无法移动，提示如下：

```
mv: cannot move 'tmp/.' to './.': Device or resource busy
mv: cannot move 'tmp/..' to './..': Device or resource busy
```

touch：创建文件

功能

touch命令有2种功能：创建空文件，更新文件修改时间。

语法

```
touch <已存在文件或不存在文件>
```

参数

实例

备注

有些脚本需要创建一个空文件来表示进程已被占用，就可以用touch命令来创建。使用Makefile编译源码时，假如没有修改到main函数所在文件，可能没有将所有的修改编译生成可执行二进制文件，此情况下，可以使用touch命令更新main函数所在文件的时间，从而保证main函数会被重新编译。

ln：链接文件

功能

语法

参数

实例

备注

cp：复制文件

功能

语法

参数

实例

备注

echo：显示文本

功能

语法

参数

实例

备注

还可以使用echo命令显示一些特殊的环境变量，比如上一次命令执行结果保存在 `$?` 中，可以用 `echo $?` 来查看。

```
echo $?
```

cat：显示文件内容

功能

cat用于将文件内容显示在终端上，实际上，cat（concatenate）命令原意是将多个文件按顺序显示在终端上，因此，可以将多个文件通过cat命令合并成一个文件。

语法

参数

参数	说明
无参数或带 '-'	读取标准输入的内容并显示到标准输出中
-n	输出的信息带有行号
-b	输出行号，但除外空行（可统计实际有用行数）
-T	以'^I'形式显示Tab键
-E	在行尾显示结束符'$'
-v	使用'^'或'M-'显示不可打印字符，除Tab、换行（LF）外
-A	等效'-vET'

实例

备注

more less

功能

语法

参数

实例

备注

head tail

功能

语法

参数

实例

备注

time

功能

用于统计程序、脚本运行所耗时间，包括用户空间、系统时间等。

语法

参数

实例

sleep

功能

休眠（延时）指定时间。

语法

```
sleep 时间数值<时间单位>
```

参数

参数	说明
s	秒，默认值
m	分
h	时

d	天

实例

```
$ sleep 5    ## 休眠5秒
$ sleep 5m   ## 休眠5分钟
```

备注

在脚本中会使用到sleep，有些操作需要等一段时间重试。

file：检测文件类型

功能

该命令可以检测目录文件类型，包括链接文件、文件文本、编程语言，甚至编码格式。

语法

```
file <目标文件>
```

实例

下面演示其用法：

```
$ file main.c   # 查看C语言文件
main.c: C source, ASCII text, with CRLF line terminators
$ file a.out    # 查看可执行二进制文件
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/l, for GNU/L
inux 2.6.32, BuildID[sha1]=fef357a07fb96881e38fae239260bcb578f18c26, not stripped
$ file /lib/ld-linux.so.2  # 查看解析器文件，实际是链接文件
/lib/ld-linux.so.2: symbolic link to i386-linux-gnu/ld-2.23.so
$ file /lib/i386-linux-gnu/ld-2.23.so # 查看解析器
/lib/i386-linux-gnu/ld-2.23.so: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV), dynamically linked
, BuildID[sha1]=784628319b1eebed400e7d74f5a14ed2c656b538, stripped
```

file命令在查看二进制可执行文件、动态库（共享库）时十分有用，可以知道该文件是何种平台，以及是否“瘦身”（即减少文件体积）。如示例中，a.out文件为64位X86系统的可执行文件，非瘦身，/lib/i386-linux-gnu/ld-2.23.so为32位X86系统的共享库，已瘦身。

tree：显示树形目录

功能

tree命令用于将指定目录以树形展开出来，默认为当前目录，并列出所有的子目录。注意，tree并非Linux的原生命令，需要使用 `$ sudo apt-get install tree` 命令来安装。

语法

```
file [目录] [参数]
```

参数

参数	说明
-L	后跟数字，指定要显示的目录的层级
-h	以人类易读的方法显示目录/文件大小

-U	显示时不排序（默认按字母顺序排序）

实例

显示当前目录，层级为1：

```
$ tree -L 1
.
├── a.out
├── h264cut.mp4
├── helloworld
├── img2.29.png
├── note.txt
└── qemu-4.0.0.tar.xz

1 directory, 5 files
```

date hwclock：日期时区

功能

本小节有3个知识点：
date：设置/获取当前时间。
hwclock：写入/获取硬件时钟。
时区：时区信息以文件形式存在，保存在目录 `/usr/share/zoneinfo` 中。

语法

参数

date命令参数说明如下：

参数	说明
-s	设置时间，需用root权限操作
+%U	本年度的周数（即当前时间为本年度第几周）
+%u	周数

%Y表示年，%m表示月，%d表示日，%H表示小时，%M表示分钟，%S表示秒，%s，1970-01-01 00:00:00 UTC

hwclock命令参数说明如下：

参数	说明
-r	读取并打印硬件时间
-s	将硬件时钟设置为系统时间
-w	将系统时间保存为硬件时钟
-c	对比系统时间、硬件时钟的误差，按Ctrl+C退出
-u	硬件时钟保持为UTC时间

实例

设置时间有很多种形式，下面给出具体示例：

```
$ sudo date -s 12:00:00 #设置具体时间，不会对日期做更改
Sun Sep 15 12:00:01 CST 2019
```

```
$ sudo date -s 20190501 #设置成20190501，这样会把具体时间设置成空00:00:00
Wed May  1 00:00:01 CST 2019
$ sudo date -s "12:00:00 2019-05-01" #这样可以设置全部时间
Wed May  1 12:00:00 CST 2019
$ sudo date -s "12:00:00 20190501"   #这样可以设置全部时间
Wed May  1 12:00:00 CST 2019
$ sudo date -s "2019-05-01 12:00:00" #这样可以设置全部时间
Wed May  1 12:00:00 CST 2019
$ sudo date -s "20190501 12:00:00"   #这样可以设置全部时间
Wed May  1 12:00:00 CST 2019
```

查询并设置系统时间和硬件时间：

```
$ date
Sun Sep 15 00:23:10 CST 2019
$ sudo hwclock
Sun 15 Sep 2019 12:23:11 AM CST  .493808 seconds

$ sudo date -s "09/10/19 12:20" # 修改时间
Tue Sep 10 12:20:00 CST 2019
$ sudo hwclock -w # 将修改后的时间同步到硬件时钟
$ date
Tue Sep 10 12:20:07 CST 2019
$ sudo hwclock
Tue 10 Sep 2019 12:20:13 PM CST  .196113 seconds
```

备注

结合命令 `ls` 和 `date` 可判断出编译的可执行二进制是否为最新版本。

whereis

功能

语法

参数

实例

备注

tee

功能

从标准输入读取，并写到标准输出或文件（可同时指定多个文件）。实际应该中，tee一般从管道获取内容。

语法

```
tee [参数] [文件]
```

参数

参数	说明
-a	追加到文件，非覆盖
-i	忽略中断信号

实例

将a.txt文件内容写到foo.txt和bar.txt文件中：

```
$ cat a.txt | tee foo.txt bar.txt
```

3.4.2 查找替换相关命令

find

功能

语法

参数

实例

备注

仅查看当前目录所有.cpp和.c文件：

```
find ./ -maxdepth 1 -name '*.cpp' -or -name '*.c'
```

注：使用-maxdepth 1指定搜索的目录深度。去掉的话，则是递归搜索所有子目录。

grep

功能

语法

参数

实例

grep结合cat可以查询文件指定的文本，多用于过滤字符串。

```
cat output.txt | grep "3030303030303030303030300500 (2)"
```

cat output.txt | grep "3030303030303030303030300500 (2)" > a.txt

sed

功能

语法

参数

实例

备注

```
sed -n '100, 200p' foo.txt >> output.txt
```

```
sed -i 's/intMAX/INT_MAX /g' `grep MAX_INt ./ -rl`
```

注：可以修改grep查找路径来指定目录。

awk

功能

语法

参数

实例

备注

dos2unix unix2dos

功能

语法

参数

实例

备注

tee

功能

语法

参数

实例

备注

3.4.3 压缩解压命令

ar

功能

语法

参数

实例

备注

tar

功能

压缩或解压缩文件。

语法

```
tar [参数] [文件或目录]
```

参数

参数	说明
-c	新建打包文件
-t	查看打包文件的内容含有哪些文件名
-x	解打包或解压缩的功能，可以搭配-C（大写）指定解压的目录，注意-c,-t,-x不能同时出现在同一条命令中
-j	通过bzip2的支持进行压缩/解压缩
-z	通过gzip的支持进行压缩/解压缩
-v	在压缩/解压缩过程中，将正在处理的文件名显示出来
-f	指定要处理的文件名称
-C	指定压缩/解压缩的目录

实例

```
压缩：$ tar jcf arm-unknown-linux-gnueabihf.tar.bz2 ./arm-unknown-linux-gnueabihf/
查询：tar -jtv -f filename.tar.bz2
解压：tar -jxv -f filename.tar.bz2 -C ./new
```

备注

zip

功能

语法

参数

实例

备注

gzip

功能

压缩文件或目录为.gz文件。

语法

```
$ gzip [参数] [文件或目录]
```

参数

参数	说明
-a	使用ASCII文字模式。
-c	把压缩后的文件输出到标准输出设备，不去更动原始文件。
-d	解开压缩文件。
-f	强行压缩文件。不理会文件名称 or 硬连接是否存在以及该文件是否为符号连接。
-h	在线帮助。
-l	列出压缩文件的相关信息。
-L	显示版本与版权信息。
-n	压缩文件时，不保存原来的文件名称及时间戳记。
-N	压缩文件时，保存原来的文件名称及时间戳记。
-q	不显示警告信息。
-r	递归处理，将指定目录下的所有文件及子目录一并处理。
-t	测试压缩文件是否正确无误。
-v	显示指令执行过程。
-num	用指定的数字num调整压缩的速度，-1(--fast)表示最快压缩，-9(--best)表示高压缩比。系统缺省值为6。

实例

备注

gunzip

功能

语法

参数

实例

备注

磁盘管理命令

df

功能

语法

参数

实例

备注

dd

功能

语法

参数

实例

备注

du

功能

语法

参数

参数	说明

实例

备注

fdisk

功能

语法

参数

参数	说明

实例

备注

fsck

功能

语法

参数

参数	说明

实例

备注

mkfs

功能

语法

参数

参数	说明

实例

备注

sync

功能

语法

参数

参数	说明

实例

备注

3.4.4 系统相关命令

top

功能

语法

参数

参数	说明

实例

备注

ps

功能

显示运行的进程及相应信息，如PID、CPU和内存使用情况。

语法

```
$ ps [参数]
```

参数

参数	说明
-A	所有的进程均显示出来
-a	不与terminal有关的所有进程
-u	有效用户的相关进程
-x	一般与a参数一起使用，可列出较完整的信息
-l	详细地将PID的信息列出

实例

```
$ ps
  PID TTY          TIME CMD
  761 pts/1    00:00:00 bash
 3697 pts/1    00:00:00 ps

$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2  37832  4284 ?        Ss   Jun30   0:35 /lib/systemd/systemd --system --deserialize 19
root         2  0.0  0.0      0     0 ?        S    Jun30   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    Jun30   3:19 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   Jun30   0:00 [kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S    Jun30  53:11 [rcu_sched]
root         8  0.0  0.0      0     0 ?        S    Jun30   0:00 [rcu_bh]
root         9  0.0  0.0      0     0 ?        S    Jun30   0:00 [migration/0]
root        10  0.0  0.0      0     0 ?        S    Jun30   0:32 [watchdog/0]
root        11  0.0  0.0      0     0 ?        S    Jun30   0:00 [kdevtmpfs]
root        12  0.0  0.0      0     0 ?        S<   Jun30   0:00 [netns]
root        13  0.0  0.0      0     0 ?        S<   Jun30   0:00 [perf]
root        14  0.0  0.0      0     0 ?        S    Jun30   0:04 [khungtaskd]
root        15  0.0  0.0      0     0 ?        S<   Jun30   0:00 [writeback]
```

备注

由于系统进程较多，一般使用管道方式与grep命令使用。

kill

功能

终止进程。

语法

```
$ kill [-s 信号名称或编号] PID
```

参数

参数	说明
-s	指定信号名称或编号编号
-l	不加参数列出所有信号名称，传递具体编号，转换为对应的名称

实例

```
终止进程： $ kill 12345
```

```
强制终止进程：$ kill -KILL 123456 或 $ kill -9 123456
终止允许的所有进程（慎用）： $ kill -9 -1
转换信号编号为信号名称：$ kill -l 1 # 结果为1对应的信号名称HUP
发送指定信号： $ kill -HUP pid
```

备注

列出所有信号名称：

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

killall

功能

终止进程。

语法

```
$ killall [参数] <程序名称>
```

参数

参数	说明
-i	交互式，终止时会询问用户
-e	与后面指定的程序名称完全一致，但名称不能超过15个字符
-I	程序名称忽略大小写
-l	列出所有信号名称

实例

```
$ killall a.out # 终止a.out程序
```

备注

free

功能

语法

参数

参数	说明

实例

备注

exit

功能

用于退出目前的shell。在脚本中调用exit表示退出脚本，回到命令行。也用于退出系统(如ssh登陆远程系统)、退出其它用户的shell。

语法

```
exit
```

备注

exit有状态值，一般0表示正常，其它表示异常。

login

功能

用于登陆系统。已登陆系统情况下，可以用login切换到其它用户登陆，用于多用户的系统。

语法

```
login
```

logout

功能

用于退出系统。

语法

```
logout
```

reboot

功能

语法

参数

实例

备注

reboot用于重启系统，也可以在图形界面上点击“重启”按钮。

halt

功能

语法

参数

实例

备注

poweroff

功能

语法

参数

实例

备注

shutdown

功能

shutdown用于关闭系统，也可以用该命令重启系统。

语法

```
shutdown [参数]
```

参数

参数	说明
-k	关不进行关机，只是
-r	关机后，再重新开机
-h	关机后停电
-c	取消关电动作

实例

```
$ shutdown -h now
```

备注

嵌入式中关机的操作较少，较多为重启系统。

who

功能

用于显示系统此时登陆的用户。包括使用的终端、登陆时间、IP地址等信息

语法

```
who
```

实例

```
# who
root     pts/0        2019-09-17 20:23 (113.13.12.24)
root     pts/1        2019-09-17 20:05 (113.13.12.24)
root     pts/3        2019-09-17 20:25 (113.13.12.24)
```

备注

一般在服务器方面使用较多，嵌入式应用中该少用到。

whoami

功能

用于显示登陆用户自己的名称。

语法

```
whoami
```

实例

```
$ whoami
latelee
$ whoami
root
```

备注

在脚本中的如果需要执行root权限的操作，要使用whoami判断是否为root。

uname

功能

用于显示系统信息。包括内核版本，发行版，架构等。

语法

```
uname
```

参数

参数	说明
-a	打印所有信息
-s	打印内核名称
-n	打印主机节点名称
-r	打印内核版本号
-v	打印内核编译版本

-m	打印机器
-o	打印操作系统

实例

```
$ uname -a
Linux latelee 4.4.0-154-generic #181-Ubuntu SMP Tue Jun 25 05:29:03 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
```

mount umount

功能

语法

参数

实例

备注

ulimit

功能

语法

参数

实例

备注

nohup

功能

不挂断执行命令。

语法

```
nohup <命令或程序>
```

实例

执行命令，并将信息重定向到文件中： `nohup cmd > file &` 。 同上，将所有信息（包括错误）重定向到文件中： `nohup cmd > file 2>&1 &` 。

备注

如果我们使用ssh登陆Linux宿主机，在终端上执行程序，在程序执行过程中如果切断连接，则程序会被中断（即使用 `&` 亦会中断）。那么，就可以使用nohup来运行程序。

用户、权限相关命令

chmod

功能

语法

参数

实例

备注

chown

功能

语法

参数

实例

备注

sudo

功能

在命令前添加sudo表示以root权限来执行命令，多常于非root用户登陆系统，但又需要使用root权限来执行命令的场合。本书使用普通用户登陆，当需要用到root权限执行命令，则在命令前添加sudo，并需要密码。参见本节的命令示例。

passwd

功能

语法

参数

实例

备注

驱动相关命令

modprobe

功能

语法

参数

实例

备注

lsmod

功能

语法

参数

实例

备注

insmod

功能

语法

参数

实例

备注

rmmod

功能

语法

参数

实例

备注

dmesg

功能

语法

参数

实例

备注

网络相关命令

ssh scp nfs telnet（测试端口是否开放） ftp 提一下，后面再继续写

scp

功能

语法

参数

实例

备注

ping

功能

语法

参数

实例

备注

iperf

功能

语法

参数

实例

备注

netstat

功能

语法

参数

实例

备注

telnet

功能

语法

参数

实例

备注

telnet还有另外一个用法，可以测试服务器端口是否开放。示例如下：

```
$ telnet baidu.com 79 # 没有开放端口，无法连接
Trying 39.156.69.79...
telnet: connect to address 39.156.69.79: Attempt to connect timed out without establishing a connection

$ telnet baidu.com 80   ## 开放了端口，该端口也运行了服务，但协议不合法
Trying 39.156.69.79...
Connected to baidu.com.
Escape character is '^]'.
Connection closed by foreign host.

$ telnet 172.18.18.8 8080 # 开放了端口，但没有运行服务，被拒绝
Trying 172.18.18.8...
telnet: Unable to connect to remote host: Connection refused
```

scp、nfs等命令在后续章节中会派上用场。

3.4.5 联机帮助命令

man：帮助手册

功能

很多人将man理解为英文man（男人），但实际它是`manual`的缩写，遇到不知道用法的函数、命令，就可以使用man命令寻求帮助，这也是Linux系统一大利器。曾一度流行“有问题，man一下”的说法。

语法

```
man [参数] [类别]] <函数名称或命令名称>
```

参数

实例

Linux的帮助文档有很多类别，列举如下：

分类	说明
1	可执行程序或Shell命令
2	系统调用（由内核提供的函数，如open、write、read、socket等）
3	库函数（由库提供，如C库的printf、memcpy、strcpy等）
4	特殊文件（一般位于/dev）
5	文件格式（如配置文件格式）
6	游戏
7	杂项（如man 7 man）
8	系统管理工具（一般限于root）
9	其它

一般情况下，查询很多命令（或函数）无须指定类别。但也有一些命令（或函数）比较特殊，不同类别下存在相同名称，此情况下如果不指定类别的话，默认使用类别1（即命令）。

比如open，它既是一个命令的名称（实际是openvt的别名），也是一个函数的名称，如果不指定分类的话，默认显示命令，如下：

```
$ man open
OPENVT(1)                          Linux 1.x                          OPENVT(1)

NAME
       openvt - start a program on a new virtual terminal (VT).

SYNOPSIS
       openvt [-c vtnumber] [OPTIONS] [--] command

DESCRIPTION
       openvt  will  find  the  first available VT, and run on it the given command
       with the given  command  options,  standard  input,  output  and  error  are
       directed  to  that terminal. The current search path ($PATH) is used to find
       the requested command. If no command is specified then the environment vari-
       able $SHELL is used.
```

如果要查看函数，则必须指定类别，如下：

```
$ man 2 open
OPEN(2)                     Linux Programmer's Manual                     OPEN(2)

NAME
       open, openat, creat - open and possibly create a file

SYNOPSIS
       #include <sys/types.h>
       #include <sys/stat.h>
       #include <fcntl.h>

       int open(const char *pathname, int flags);
       int open(const char *pathname, int flags, mode_t mode);

       int creat(const char *pathname, mode_t mode);

       int openat(int dirfd, const char *pathname, int flags);
       int openat(int dirfd, const char *pathname, int flags, mode_t mode);
```

类似的还有 `man printf` 和 `man 3 printf` 。

备注

如果要查看命令 `man` 的帮助信息，请使用 `man man` 。

命令自身参数

除了使用man命令外，也可以使用命令自身提供的帮助信息。Linux系统几乎每个命令都提供帮助信息，使用参数 `-h` 或 `--help` 可查看，有些命令参数 `-h` 有实际意思（一般表示human-readble)，如ls、tree等命令，此情况下只能使用 `--help` 参数。但也有个别是没有帮助信息选项的，如time命令，只能用man time来查看帮助信息。

编译相关命令

本节以表格形式列出编译相关的命令，这些命令不是日常使用的，但对于了解编译过程有很大帮助，后续章节会进行详解，在这里稍做了解即可。 gcc nm strip objdump另外介绍

命令	说明

开发库相关工具

有一些库，在日常工作中直接使用的机率很少，但却是不可或缺的，通常是由其它的程序（如make)来调用，各位只需要将其安装到系统上即可。无须研究太多原理。

cpp

cpp是C语言预编译器（C Preprocessor），并不是C++语言源码后缀。我们可以通过这个命令输出Linux系统的C语言宏定义。

m4

m4是宏处理命令，在编译源码时会使用到。

flex bison

flex是词法分析工具，是lex的代替者，bison是语法分析工具，是yacc的代替者，两者结合，可以做简单的C语言编译器前端，或计算器。有些源码需要这两个工具才能顺序编译，安装命令为 `$ sudo apt-get install flex bison` 。

3.4.7 其它命令

本小节介绍一些有趣的命令，用以调节一下枯燥的工作。安装命令形式为 `$ sudo apt-get install <命令名称>` ，将 `<命令名称>` 替换为下面的命令即可。

1、figlet和toilet：将文本转换成由其它字符组成的大的文本形式。 两个命令执行结果示例如下：

```
$ figlet CST studio
  ____ ____ _____       _             _ _
 / ___/ ___|_   _|  ___| |_ _   _  __| (_) ___
| |   \___ \ | |   / __| __| | | |/ _` | |/ _ \
| |___ ___) || |   \__ \ |_| |_| | (_| | | (_) |
 \____|____/ |_|   |___/\__|\__,_|\__,_|_|\___/

$ toilet CST studio

   mmm    mmmm mmmmmmm                  m               #    "
 m"   " #"   "   #            mmm   mm#mm  m   m   mmm#  mmm     mmm
 #      "#mmm    #           #   "    #    #   #  #" "#    #    #" "#
 #          "#   #            """m    #    #   #  #   #    #    #   #
  "mmm" "mmm#"   #           "mmm"    "mm  "mm"#  "#m##  mm#mm  "#m#"
```

2、sl：在屏幕中输出一辆奔跑的小火车 其执行效果如下：

```
$ sl
                      (@@) (  ) (@)  ( )  @@    ()    @     O     @     O      @
                 (   )
             (@@@@)
          (    )
        (@@@)
      ====        ________                ___________
  _D _|  |_______/        \__I_I_____===__|_________|
   |(_)---  |   H\________/ |   |        =|___ ___|      _________________
   /     |  |   H  |  |     |   |         ||_| |_||     _|                \_____A
  |      |  |   H  |__--------------------| [___] |   =|                      |
  | ________|___H__/__|_____/[][]~\_______|       |   -|                      |
  |/ |   |-----------I_____I [][] []  D   |=======|____|_____________________|_
```

```
  __/ =| o |=-~~\  /~~\  /~~\  /~~\ ____Y___________|__|__________________________|_
   |/-=|___|=O=====O=====O=====O   |_____/~\___/          |_D__D__D_|  |_D__D__D_|
    \_/      \__/  \__/  \__/  \__/      \_/               \_/   \_/    \_/   \_/
```

3、cmatrix：如《黑客帝国》电影里的字节雨。 其执行效果如图3.1所示。

图3.1 字节雨

按 `Ctrl+C` 即可退出。

3.4.8 命令快捷方式

Linux的命令非常多，有些也难记，或许Linux大牛们意识到此问题，在实践上提供了非常有用的功能：命令自动补齐，只要按Tab键，或一次或两次，就能得到命令。当已输入的字符能匹配到唯一的命令时，按一次Tab键，即可自动补齐余下的字符，无须再输入，当匹配到多个命令，按两次Tab键将出现所有候选命令。有了此功能，将大大提高命令的输入效率。在工作中，强烈建议读者使用Tab键。

除了自动补齐，Linux还有很多有用的快捷键，常见快捷键说明如下，这些快捷键，大部分可适用于Linux命令终端、u-boot命令终端、emacs编辑器——注意，这里的措词是“大部分”，说明并不适用所有的场合。

快捷键	说明
Ctrl + a	移到命令行首 （a：ahead）
Ctrl + e	移到命令行尾 （e：end）
Ctrl + f	按字符前移 （f：forward）
Ctrl + b	按字符后移 （b：backward）
Alt + f	按单词前移
Alt + b	按单词后移
Ctrl + p	上一个使用的历史命令（p：previous），相当于向上方向键
Ctrl + n	下一个使用的历史命令（n：next ），相当于向下方向键
Ctrl + r	逆向搜索命令历史（r：retrieve）
Ctrl + d	删除光标后一个字符 (d：delete）

Ctrl + h	删除光标前一个字符
Ctrl + w	从光标处剪切/删除至字首
Ctrl + u	从光标处剪切/删除至行首
Ctrl + k	从光标处剪切/删除至行尾
Ctrl + h	从光标处向前删除一个字符
Ctrl + d	从光标处向后删除一个字符
Alt + d	从光标处删除至字尾
Ctrl + y	粘贴文本（注：文本由Ctrl + u、Ctrl + k、Ctrl + w获得）
Ctrl + t	交换光标处和之前的字符
Alt + t	交换光标处和之前的单词
Alt + c	从光标处更改为首字母大写的单词
Alt + u	从光标处更改为全部大写的单词
Alt + l	从光标处更改为全部小写的单词
Ctrl + o	执行当前命令，并选择上一条命令
Ctrl + _	撤销操作（Ctrl+Shift+减号）
Ctrl + l	清除屏幕，作用与命令clear相同
Tab	自动补齐
Ctrl + c	强制退出命令
Ctrl + z	挂起命令
q	退出，适用于man、more、less等信息查询命令

本小节涉及到readline库，u-boot中也实现了部分功能，阅读其源码可了解原理细节。

3.4.9 管道的使用

管道在前面章节已经介绍过了，下面结合几个具体的实例，来加深管道的认识。这些实例都是从实际问题出发，利用管道和各种命令达成目标的。

统计代码数量

统计当前目录所有.h .cpp文件文件行数（其它类型类推）：

```
find . ! -name "." -name "*[.h|.hpp|.c|.cpp]"|xargs cat|grep -v ^$|wc -l
或
find . ! -name "." -name "*[.h|.hpp|.c|.cpp]" -type f | xargs cat | wc -l
```

统计当前目录.cpp文件个数

```
find . ! -name "." -name "*[.h|.hpp|.c|.cpp]"|wc -l
```

遗留：.h会匹配.sh文件，.c会匹配.cpp文件。

查找文件指定字符串出现第一次或最后一次位置

查找指定文件某字符串第一次出现位置：
命令：`cat <文件名称> | grep -n "<字符串>" | head -n 1`
示例：

```
cat log.txt | grep -n "MISCONF Redis is configured to save RDB snapshots" | head -n 1
```

查找指定文件某字符串最后一次出现位置：
命令：`cat <文件名称> | grep -n "<字符串>" | tail -n 1`
示例：

```
cat log.txt | grep -n "MISCONF Redis is configured to save RDB snapshots" | tail -n 1
```

解释：
先用cat命令查看文件内容，接着用grep搜索指定的字符串，注意要添加"-n"选项以便显示匹配字符串在文件中的行号，最后用head或tail显示查找到的内容的开头部分或结尾部分字符串，"-n 1"表示只显示一行，故能实现显示第一次或最后一次字符串。打开文件（notepad++或vs code使用ctrl+g快捷键）定位到行号即可。综上，使用管道可实现目的。

3.4.10 命令使用经验

学习Linux的命令是十分枯燥的，但却是必须学习和死记的，熟悉缩写字母背后的含义有助于快速记忆。无他，唯手熟尔，只要在工作中经常使用，多敲命令有利于肌肉记忆，慢慢会熟悉起来。解决工作中的问题往往需要多个命令、多个知识结合分析，下面给出一些使用心得。

Windows系统创建的文件，默认行尾为回车换行（CR，0x0D，LF，0x0A），在shell脚本中是非法的，使用 `cat -A` 查看行尾带有'^M'，使用 `dos2unix` 更改换行符号即可解决。

> 注1. 酷壳上有一篇文章《一个空格引发的惨剧》介绍此事：https://coolshell.cn/articles/4875.html。有兴趣的读者可以到项目仓库围观：https://github.com/MrMEEE/bumblebee-Old-and-abbandoned/issues/123。调侃归调侃，但细节有时真的很重要。 ↩

3.5 shell脚本

shell脚本及属性

shell简单编程

3.6 系统配置

系统配置可以通过/etc目录的配置文件、用户HOME目录的配置文件，以及部分命令来实现。所涉及内容较多，本小节列出一些常用的配置。

终端

Unix/Linux终端类型很多，深究起来也复杂[注1]。此处本着应用的目的来了解终端的一些知识，终端名称可用 `tty` 命令查看。为行文方便，这里假定有2种终端：

- tty终端
  操作真实Linux系统（或虚拟机Linux系统）机器的，默认启动的是图形界面，使用Ctrl+Alt+F1~F6切换到字符界面。其中，Ctrl+Alt+F1切换到第1个字符界面，终端为tty1，Ctrl+Alt+F2为第2个，终端为tty2，依此类推。注意，图形界面亦有终端概念，其终端为tty7（Ctrl+Alt+F7切换）。但是，如果在图形界面中打开终端（Ubuntu系统默认为GNOME终端），其终端名为pts。
- pts终端 使用telnet或者ssh远程登录Linux系统，其终端为pts终端。

（待写：串口终端也要带一带）

我们使用SecureCRT连接2次Linux，打开2个字符界面（Ctrl+Alt+F1/F2），再在打开图形界面的终端（Ctrl+Alt+F7切换）。接着使用命令 `who` 查看一下，结果如下：

```
$ who
latelee  pts/8        2019-10-04 18:56 (172.18.18.1)
latelee  pts/9        2019-10-04 18:57 (172.18.18.1)
latelee  tty1         2019-10-04 19:01
latelee  tty7         2019-10-04 19:04 (:0)
latelee  tty2         2019-10-04 19:02
```

可以看到，所有的行都有用户名，终端名，登陆时间，使用ssh连接，还有带IP地址信息。另外，即使打开GNOME终端，也无法看到终端名，因为GNOME归属到tty7中。

命令提示符

登录系统后，第一眼看到的内容是：

```
[latelee@localhost ~]$
```

如果使用root用户登陆，则变成：

```
[root@localhost ~]#
```

同时光标在闪烁，等待用户输入命令。

这就是Linux系统的命令提示符。下面分析出现的字符：

字符	说明
[]	提示符的分隔符号，以便与用户输入的信息区分开
root或latelee	当前登陆用户
@	用户与主机名的分隔符号
localhost	主机名
~	当前目录，首次进入的目录为~（即HOME目录），如切换其它目录，则显示相应的目录
#或$	命令提示符，root权限（包括root用户以及sudo切换的root权限）为#，普通用户为$

命令提示符的格式由环境变量 `$PS1` 确定，不同系统默认值不同，如Ubuntu为 `${debian_chroot:+($debian_chroot)}\u@\h:\w\$` ，CentOS为 `[\u@\h \W]\$` 。可通过修改其值自定义显示的格式[注2]。

值	说明
\d	显示日期
\h	显示主机名，在/etc/hostname定义，默认为localhost
\t	显示24小时制时间，格式为HH:MM:SS
\T	显示12小时制时间，格式同上
\A	显示24小时时间，格式为HH:MM
\u	显示用户名
\w	显示当前所在完整路径
\W	显示当前目录（仅目录）
\#	显示执行了多少条命令，注意，这不是root权限的提示符 `#`
\$	提示符，注意，如果为root权限，则自动变为 `#`

读者可以通过 `$ export PS1="[\u@\h:\w]\$ "` 命令观察效果，该命令即时生效，由于是临时设置 `PS1` 的值，所以不会对原有值产生影响。下面给出部分演示效果：

```
$ export PS1="[\u@\h:\W]\$ "
--> [latelee@ubuntu:Etc]$

export PS1="[\u@\h:\W]\$ my shell test >"
--> [latelee@ubuntu:Etc]$ my shell test >

$ export PS1="[\u@\h:\w]\$ "
--> [latelee@localhost:/usr/share/zoneinfo/Etc]$

$ export PS1="[\d \A \# \u@\H:\W]\$ "
--> [Fri Oct 04 16:43 22 latelee@localhost:Etc]$
```

环境变量

Linux有很多环境变量，很多程序都会使用这些环境变量。用命令 `env` 可显示出系统当前的环境变量。示例如下：

环境变量及值	说明
TERM=xterm-256	终端名称

TERM=vt100	终端名称
SHELL=/bin/bash	shell名称
SSH_CLIENT=172.18.18.2 63903 22	ssh客户端信息
SSH_CONNECTION=172.18.18.2 63903 172.18.18.8 22	ssh连接信息
SSH_TTY=/dev/pts/8	ssh连接使用的伪终端
USER=latelee	当前用户名称
LOGNAME=latelee	登陆用户名称
PWD=/home/latelee	当前所在目录，与命令 `pwd` 等效
OLDPWD=/home	上一次所在目录，可用 `cd -` 返回该目录
LANG=en_US.UTF-8	系统字符编码
_=/usr/bin/env	当前执行的命令名称，可用 `echo $_` 查看
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin	可执行程序路径
PS1	命令提示符环境变量

环境变量可直接在shell或其它脚本语言中使用，在环境变量前面添加 `$` 即可，如 `$PWD` 、 `$PATH` 等等。

最常见的环境变量为 `$PATH` ，该变量指定了程序执行的路径，系统默认
将/sbin、/bin、/usr/sbin、/usr/bin、/usr/local/sbin、/usr/local/bin这几个目录都设置到 `$PATH` 中，所以，这些目录的可执行文件可以在任意目录中执行。我们也可以将自己的目录添加到该环境变量中。笔者喜欢在自己的HOME目录下创建一个bin目录，用于存放各种临时的小工具。其目录绝对路径为 `/home/latelee/bin` ，下面看看有哪些添加的方式。

- 临时方式
  方式如下：

  ```
  $ export PATH="$PATH:/home/latelee/bin"
  ```

  注意，只能在当前终端生效，切换终端、退出终端或重启系统均失效。适用于临时测试所用。

- 个人用户
  编辑文件 `~/.bashrc` ，在文件末尾添加：

  ```
  export PATH=/home/latelee/bin:$PATH
  ```

  该方式适用于个人用户，能永久生效且不影响其它用户。

- 全局生效
  编辑文件 `/etc/profile` ，在文件末尾添加：

  ```
  export PATH=/home/latelee/bin:$PATH
  ```

  该方式针对所有的系统用户，如果设置不正确可能会影响他人使用，所以不建议使用。

`$PATH` 的设置建议逐个目录添加，方便管理。 注意，环境变量的 `$` 千万不可少，否则可能会导致严重后果。以临时方式为例，如果将命令修改为 `$ export PATH="PATH:/home/latelee/bin"` （各位读者仔细观察细小差异）。那么，已存在的 `$PATH` 目录将被覆盖，则无法正常执行命令，以 `$ ls` 为例，错误信息如下：

```
Command 'ls' is available in '/bin/ls'
The command could not be located because '/bin' is not included in the PATH environment variable.
ls: command not found
```

时区

Linux时间使用的是UTC时间，再加上时区，才是我们看到的时间，可以在系统运行时变更时区，从而得到真实的本地时间。时区文件位于目录 `/usr/share/zoneinfo` 中，该目录或按大洲分类（如亚洲、非洲），或按国家地区分类。还有一个 `Etc` 目录，大部分文件按时区分类[注3]。而文件 `/etc/localtime` 存储的是时区信息，将其内容替换为所需的时区文件内容即可。下面是一个示例：

```
$ date    # 当前为PDT时区（即太平洋夏令时）
Sat Sep 14 09:10:00 PDT 2019
$ sudo cp /usr/share/zoneinfo/Etc/GMT-8 /etc/localtime   # 时区文件为GMT-8
$ date    # 当前为东8时区
Sun Sep 15 00:11:10 +08 2019
$ sudo cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime   # 时区文件为Shanghai
$ date    # 当前为CST时区（即中国标准时区，东8区）
Sun Sep 15 00:11:30 CST 2019
```

下面介绍一下GMT时间和UTC时间。
GMT(Greenwich Mean Time，格林威治标准时间)是指位于英国伦敦郊区的皇家格林尼治天文台的标准时间。因为本初子午线被定义为在通过那里的经线。
UTC（英文Coordinated Universal Time和法语Temp Universelle Coordinee的混合体。通常记作Universal Time Coordinated。中文为协调世界时，又称世界统一时间，世界标准时间，国际协调时间）是经过平均太阳时(以格林威治时间GMT为准)、地轴运动修正后的新时标以及以秒为单位的国际原子时所综合精算而成的时间，计算过程相当严谨精密，GMT天文上的概念，UTC基于一个原子钟，因此若以世界标准时间的角度来说，UTC比GMT来得更加精准。
在日常交流中，称UTC或GMT还有一定争议，但如果不是严格使用精确的时间的话，笔者认为不必纠结于这些称呼。一般认为GMT和UTC是一样的，而且都和英国伦敦的本地时间相同。

（待写：有个配置文件为UTC时间设置，/etc/default/rcS将UTC=yes改成UTC=no，但ubuntu新版本没有了）

字符集

所谓字符，就是各种文字和符号的总称，包括各国的文字、标点符号、图形符号、数字等等。字符集（Character set）是多个字符的集合，字符集种类较多，每个字符集包含的字符个数不同，常见字符集名称：ASCII字符集（最通用的单字节编码系统）、GB2312字符集（1981年5月1日实施，很多生僻字没有收录）、BIG5字符集（1984年创立，繁体中文）、 GB18030字符集（2000年3月17日发布，最新的汉字编码国家标准，兼容GB2312）、Unicode字符集（1994年正式公布，UTF-32、UTF-16和 UTF-8 是Unicode的字符编码方案）等。

在Linux中，使用 `locale -a` 命令查看系统所有的字符集：

```
$ locale -a
C
C.UTF-8
en_AG
en_AG.utf8
en_AU.utf8
en_BW.utf8
en_CA.utf8
en_DK.utf8
en_GB.utf8
en_HK.utf8
en_IE.utf8
en_IN
en_IN.utf8
en_NG
en_NG.utf8
en_NZ.utf8
en_PH.utf8
```

```
en_SG.utf8
en_US.utf8
en_ZA.utf8
en_ZM
en_ZM.utf8
en_ZW.utf8
POSIX
```

使用 `locale` 命令查看当前系统的字符集：

```
$ locale
LANG=en_US.UTF-8
LANGUAGE=
LC_CTYPE="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
LC_TIME="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
LC_PAPER="en_US.UTF-8"
LC_NAME="en_US.UTF-8"
LC_ADDRESS="en_US.UTF-8"
LC_TELEPHONE="en_US.UTF-8"
LC_MEASUREMENT="en_US.UTF-8"
LC_IDENTIFICATION="en_US.UTF-8"
LC_ALL=
```

上面显示了许多环境变量，它们的优先级关系是：`LC_ALL` > `LC_*` > `LANG`。为了保证系统字符集一致，这些环境变量最好保持一致。

从字符集结果可以看到，这台Linux系统只有英文UTF8字符集，因为在第二章节中，我们采用了最简安装，所以默认是英文环境。这表示，系统界面信息、帮助信息等均为英文，但我们可以使用中文。示例如下：

```
$ ls 中文文件
ls: cannot access '中文文件': No such file or directory
latelee@ubuntu:~$ echo "中文字符" > 中文文件
latelee@ubuntu:~$ ls 中文文件
中文文件
latelee@ubuntu:~$ cat 中文文件
中文字符
```

从示例看到，我们可以创建中文文件，也可以编辑中文文本。

下面安装中文字符集：

```
$ sudo apt-get install -y language-pack-zh-hans # 安装简体中文包zh_CN
$ sudo apt-get install -y language-pack-zh-hant # 安装繁体中文包zh_HK、zh_TW
$ sudo /usr/share/locales/install-language-pack zh_CN # 安装zh_CN字符集，也可安装zh_TW
```

安装后，使用 `locale -a` 查看发现新加如下的字符集：

```
zh_CN.utf8
zh_HK.utf8
zh_SG.utf8
zh_TW.utf8
```

将当前字符集更改为简体中文或繁体中文。如下：

```
$ export LANG="zh_CN.utf8" LC_ALL="zh_CN.utf8" LANGUAGE="zh_CN"
$ export LANG="zh_TW.utf8" LC_ALL="zh_TW.utf8" LANGUAGE="zh_TW"
```

为不影响系统，此处仅针对当前终端进行测试。

安装了中文字符集后，系统就变成中文版本了，即界面信息，提示信息，帮助信息均为中文。但是，Linux系统本身为英文，其它国家或地区语言需要靠语言文件才能正常显示相应的语言，否则使用默认的英文信息[注5]。
笔者在测试时发现，设置简体中文后，`man man` 可出现中文帮助信息，但 `man fopen` 依旧是英文。而设置为繁体中文，两者均为英文。

下面看一下不同字符集的提示信息[注6]：

```
$ export LC_ALL="POSIX" ## POSIX环境，中文无法显示，提示语为英文
$ 中文命令
\udce4\udcb8\udcad\udce6\udc96\udc87\udce5\udc91\udcbd\udce4\udcbb\udca4: command not found

$ export LC_ALL="en_US.utf8" ## utf8英文环境，中文正常显示，提示语为英文
$ 中文命令
中文命令: command not found

$ export LC_ALL="zh_CN.utf8" ## utf8简体中文环境，中文正常显示，提示语为简体中文
$ 中文命令
中文命令：未找到命令

$ export LC_ALL="zh_TW.utf8" ## utf8繁体中文环境，中文正常显示，提示语为繁体中文
$ 中文命令
中文命令：無此指令
```

> 中文乱码问题
> 在编程中可能会遇到中文乱码问题，此处按源码端、显示端，连接端三部分进行介绍，只要保证这三者的字符集一致，就不会出现乱码。以UTF8为例。
> 1、源码端。需要将源码文件的编码格式设置为UTF8。在VS Code或Notepad++中可修改。
> 2、显示端。Linux终端需要设置字符为UTF8。en_US.UTF-8或zh_US.UTF-8均可。
> 3、连接端。以SecureCRT为例，在选项->会话选项->终端->外观->字符编码处，选择这字符编码为“UTF-8”。

登陆信息

Linux系统登陆前后（无论哪一种方式）都会显示一些信息，如上次登陆时间，系统版本号，是否可升级等等，这些信息通过一些配置文件来实现，至于由哪些工具以及何种机制触发，我们暂时不进行研究。

- /etc/issue
  该文件的信息显示于登陆前，在字符界面中生效。
- /etc/issue.net
  该文件的信息显示于登陆前，在远程登陆（如Telnet、ssh）生效。注意，ssh默认不显示，可以编辑 `/etc/ssh/sshd_config` 文件，去掉 `Banner /etc/issue.net` 前的注释符号，再重启ssh服务即可。
- /etc/update-motd.d/
  Ubuntu新版本（16.04及以上）中，启用了动态motd，登陆显示的信息由该目录下的脚本文件控制，如 `00-header` 文件为第一行信息，内容为 `Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic x86_64)` ，`10-help-text` 文件为帮助文档和技术支持的网络地址，其它类似。我们可以更改或删除这些脚本达到我们的目的[注4]。

值得注意的事，这些文件记录了操作系统的名称和版本号，可能会泄漏系统的信息，为了安全起见，在公共环境（如公网中）中建议将系统相关信息去除。

HOME目录的隐藏文件

用户的HOME目录有许多隐藏的目录或文件，其有较重要作用，这里作为扩展知识稍微介绍一下。

.ssh配置

熟悉ssh在日常使用中有很大的便利，我们可以生成公钥、私钥，与服务进行免密访问或免密拷贝文件。
使用命令 `ssh-keygen -t rsa -C "li@latelee.org"` 生成的文件位于 `~/.ssh` 目录，其中 `~/.ssh/id_rsa` 为私钥文件， `~/.ssh/id_rsa.pub` 为公钥文件。 `~/.ssh/authorized_keys` 为允许访问的机器的公钥。

.git配置

如果在Linux上使用git进行版本管理，则必须设置git的用户与邮箱（可全局设置，也可以针对仓库设置，这里演示全局设置），设置示例如下：

```
git config --global user.name "Late Lee"
git config --global user.email "li@latelee.org"
```

执行后，将在HOME目录生成 `.gitconfig` 文件，其内容如下：

```
$ cat ~/.gitconfig
[user]
        email = li@latelee.org
        name = 李迟
```

其它

下面再罗列一下其它的配置目录及作用。

docker（一个应用容器引擎）的配置文件为 `~/.docker/config.json` ，该文件可指定加速器地址。

pm2（一个nodejs管理工具）的配置目录为 `~/.pm2` ，里面包括了pm2的日志、安装的模块。

pip（Python 包管理工具）的配置目录 `~/.pip` ，里面包括了包源地址。

> 注1. 本文后续会在代码层面跟踪串口终端。 ↩

> 注2. 不同系统的PS1值是不同的，有的系统没有 `[]` ，有的系统显示完整目录，有的显示当前目录，为保证命令示例的通用性，本书命令示例从 `$` 或 `#` 开始。另外，还有PS2、PS3、PS4等环境变量，有兴趣的读者可自行了解。 ↩

> 注3. Etc目录的时区文件名称，与我们理解的稍有不同，如 `GMT-8` 是指东八区时区，名称格式为“GMT加上与UTC的偏移值(时间差)”，如果时间差为正数，则时区位于本初子午线(Prime Meridian)之西(即西几区)，如果是负数，则时区位于本初子午线之东(即东几区)，详细解释可以参阅 `man tzset` 的说明。 ↩

> 注4. 如团队中共用一台服务，可将通知信息、工具位置使用帮助等信息在各成员每次登陆时均显示出来。 ↩

> 注5. 可查阅gettext工具，以及po、pot、mo格式文件了解更多。 ↩

> 注6. 设置为中文字符集会出现警告信息“-bash: warning: setlocale: LC_ALL: cannot change locale (zh_CN.utf8)”，但不影响使用。测试使用SecureCRT软件并且设置了UTF8字符集。 ↩

3.7 软件安装

dpkg方式安装

apt方式安装

源码安装

源码安装相对来说较麻烦一些，总的来说有如下几个步骤：

- 下载源码
- 手动解压
- 编译安装

详情我们在第5章节再介绍。

系统级别或Linux发行版官方提供的软件，最好是使用官方的渠道来安装。只有官方提供的版本不符合（如使用指定版本）或未提供（如小众软件）的情况下才考虑源码安装。

3.8 更新Linux内核

使用官方的内核镜像文件

查看本机当前内核版本：

```
$ uname -a
Linux ubuntu 4.8.0-36-generic #36~16.04.1-Ubuntu SMP Sun Feb 5 09:39:57 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
```

搜索内核镜像：

```
$ sudo apt-cache search linux-image
```

输出镜像列表：

```


linux-image-4.4.0-21-generic - Linux kernel image for version 4.4.0 on 64 bit x86 SMP
linux-image-4.4.0-21-lowlatency - Linux kernel image for version 4.4.0 on 64 bit x86 SMP
linux-image-extra-4.4.0-21-generic - Linux kernel extra modules for version 4.4.0 on 64 bit x86 SMP
...
linux-image-4.10.0-14-generic - Linux kernel image for version 4.10.0 on 64 bit x86 SMP
linux-image-4.10.0-14-lowlatency - Linux kernel image for version 4.10.0 on 64 bit x86 SMP
...
linux-image-extra-4.4.0-1031-gke - Linux kernel extra modules for version 4.4.0 on 64 bit x86 SMP
linux-image-extra-4.4.0-1032-gke - Linux kernel extra modules for version 4.4.0 on 64 bit x86 SMP
linux-image-extra-4.4.0-1033-gke - Linux kernel extra modules for version 4.4.0 on 64 bit x86 SMP
linux-image-extra-4.4.0-1034-gke - Linux kernel extra modules for version 4.4.0 on 64 bit x86 SMP
...
```

选择一个版本测试：

```
$ sudo apt-get install linux-image-4.10.0-14-generic
```

提示信息如下：

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  fdutils linux-tools linux-headers-4.10.0-14-generic
The following NEW packages will be installed:
  linux-image-4.10.0-14-generic
0 upgraded, 1 newly installed, 0 to remove and 282 not upgraded.
Need to get 23.3 MB of archives.
After this operation, 70.0 MB of additional disk space will be used.
Get:1 http://mirrors.tuna.tsinghua.edu.cn/ubuntu xenial-updates/main amd64 linux-image-4.10.0-14-generic amd64
4.10.0-14.16~16.04.1 [23.3 MB]
Fetched 23.3 MB in 1min 58s (196 kB/s)
Selecting previously unselected package linux-image-4.10.0-14-generic.
(Reading database ... 202294 files and directories currently installed.)
Preparing to unpack .../linux-image-4.10.0-14-generic_4.10.0-14.16~16.04.1_amd64.deb ...
Done.
Unpacking linux-image-4.10.0-14-generic (4.10.0-14.16~16.04.1) ...
Setting up linux-image-4.10.0-14-generic (4.10.0-14.16~16.04.1) ...
Running depmod.
update-initramfs: deferring update (hook will be called later)
Examining /etc/kernel/postinst.d.
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 4.10.0-14-generic /boot/vmlinuz-4.10.0-14-generic
```

```
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 4.10.0-14-generic /boot/vmlinuz-4.10.0-14-generic
update-initramfs: Generating /boot/initrd.img-4.10.0-14-generic
run-parts: executing /etc/kernel/postinst.d/pm-utils 4.10.0-14-generic /boot/vmlinuz-4.10.0-14-generic
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 4.10.0-14-generic /boot/vmlinuz-4.10.0-14-gener
ic
run-parts: executing /etc/kernel/postinst.d/update-notifier 4.10.0-14-generic /boot/vmlinuz-4.10.0-14-generic
run-parts: executing /etc/kernel/postinst.d/zz-runlilo 4.10.0-14-generic /boot/vmlinuz-4.10.0-14-generic
Warning: Not updating LILO; /etc/lilo.conf not found!
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 4.10.0-14-generic /boot/vmlinuz-4.10.0-14-generic
Generating grub configuration file ...
Warning: Setting GRUB_TIMEOUT to a non-zero value when GRUB_HIDDEN_TIMEOUT is set is no longer supported.
Found linux image: /boot/vmlinuz-4.10.0-14-generic
Found initrd image: /boot/initrd.img-4.10.0-14-generic
Found linux image: /boot/vmlinuz-4.8.0-36-generic
Found initrd image: /boot/initrd.img-4.8.0-36-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
Found Ubuntu 16.04 LTS (16.04) on /dev/sdb2
done
```

重启系统，默认使用新的内核版本：

```
$ uname -a
Linux ubuntu 4.10.0-14-generic #16~16.04.1-Ubuntu SMP Fri Mar 17 17:28:20 UTC 2017 x86_64 x86_64 x86_64 GNU/Lin
ux
```

自定义编译内核

3.9 自定义构建Linux系统

自定义构建Linux系统有很多方式，如使用debootstrap、LFS、busybox，等等。本节介绍前2种方式，

debootstrap

debootstrap是debian/ubuntu下的一个工具，用来构建一套基本的系统(根文件系统)。生成的目录符合Linux文件系统标准(FHS)，即包含了/boot、/etc、/bin、/usr等等目录，但它比发行版本的Linux体积小很多，当然功能也没那么强大，因此，只能说是“基本的系统”。 fedora下(centos亦可用)有类似功能的工具：febootstrap。观察这两个工具名称，可以看到debootstrap使用debian前缀“de”，而febootstrap使用fedora前缀“fe”。

ubuntu默认没有安装debootstrap，安装十分简单，执行下列命令即可：

```
# sudo apt-get install debootstrap
```

使用也十分简单，命令格式为：

```
sudo debootstrap --arch [平台] [发行版本代号] [目录]
```

比如下面的命令

```
sudo debootstrap --arch i386 trusty /mnt
```

即是构建x86(32位)平台ubuntu最新发行版14.04(代号为trusty)的基本系统，存放到/mnt目录下。
当前debootstrap支持的发行版本可以在/usr/share/debootstrap/scripts查看，而各发行版代号，可以到http://en.wikipedia.org/wiki/List_of_Ubuntu_releases查看。比如gutsy是7.10的代号，precise是12.04的代号，等等。
输入上述命令后，就会从网络下载相关的文件，当看到

```
I: Configuring python-central...
I: Configuring ubuntu-minimal...
I: Configuring libc-bin...
I: Configuring initramfs-tools...
I: Base system installed successfully.
```

即表示成功。如果看到

```
E: Failed getting release file [
http://archive.ubuntu.com/ubuntu/dists/trusty/Release](http://archive.ubuntu.com/ubuntu/dists/trusty/Release)
```

或卡在

```
I: Retrieving Release
```

则可能是网络原因。
下载的文件在/mnt/var下，如：

```
$ tree
.
├── debootstrap
│   ├── debootstrap.log
│   └── debpaths
└── var
```

```
├── cache
│   └── apt
│       └── archives
│           ├── adduser_3.113+nmu3ubuntu3_all.deb
│           ├── apt_1.0.1ubuntu2_i386.deb
```

其中adduser_3.113是14.04对应的adduser。从这里也可以确认其下载的是哪一发行版的软件。
下面使用chroot进入/mnt目录，并查看linux版本。

```
latelee@ubuntu:~$ cd /mnt/
latelee@ubuntu: /mnt$ ls
bin  boot  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
latelee@ubuntu: /mnt$ sudo -s
[sudo] password for latelee:
root@ubuntu: /mnt# chroot .
root@ubuntu:/# ls
bin  boot  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@ubuntu:/# cat etc/issue
Ubuntu 14.04 LTS \n \l
root@ubuntu:/# ls /proc/ -l
total 0
root@ubuntu:/# ifconfig
Warning: cannot open /proc/net/dev (No such file or directory). Limited output.
root@ubuntu:/# uname -a
Linux ubuntu 3.13.0-32-generic #57-Ubuntu SMP Tue Jul 15 03:51:12 UTC 2014 i686 i686 i686 GNU/Linux
```

因为当然系统使用的并不这个新的系统，因此/proc并没有内容，而内核依然是当前系统所用的版本。
使用光盘不成功

```
W: Failure trying to run: chroot /home/latelee/test_sys mount -t proc proc /proc
W: See /home/latelee/test_sys/debootstrap/debootstrap.log for details
```

注： 本文并没有过多技术含量，仅是在学习过程中碰见了debootstrap而写点笔记。本文所用环境均是虚拟机vmware。

附录：
未完事宜：
限于时间，目前还没有实际启动新的系统。
一些涉及到ubuntu根文件系统构建的资源：
http://www.virtuatopia.com/index.php/Building_a_Debian_or_Ubuntu_Xen_Guest_Root_Filesystem_using_debootstrap
https://wiki.ubuntu.com/DebootstrapChroot https://help.ubuntu.com/10.04/installation-guide/i386/linux-upgrade.html
http://www.thegeekstuff.com/2010/01/debootstrap-minimal-debian-ubuntu-installation/
http://askubuntu.com/questions/442610/debootstrap-warning-during-installation-12-04-lts-server-vmware-virtual-mach
https://help.ubuntu.com/lts/installation-guide/i386/index.html
查看ubuntu各发行版本wiki：
http://en.wikipedia.org/wiki/List_of_Ubuntu_releases
查看ubuntu安装包： http://packages.ubuntu.com/
后记：本想写稍有点技术含量的文章，把过程所涉及到的知识点都提及，但发现自己文笔不复如前，还是按流水账那样写出来比较畅快些。

LFS

LFS是Linux From Scratch的简称，即从头开始构建一个Linux系统。

官方网站为：http://www.linuxfromscratch.org/

通过LFS，我们可以了解Linux系统内部的实现细节。比工具包分类，编译方法，grub、系统配置文件。

另外，基于ubuntu系统的构建方法有debootstrap，使用该方法比较简单。

此项仅对于有兴趣研究的人，不了解LFS并不影响Linux的日常使用。

3.10 本章小结

通过本章的介绍，希望读者对Linux系统有一个比较初步的认识，本章的介绍偏向Linux系统的初步认识，有很多知识没有涉及，比如Linux系统服务启动、日志管理、多用户管理，等等，这些大部分属于Linux运维方面的，故本章不涉及。熟练了解掌握Linux的使用是一个漫长过程，需要在日常工作中重复、积累，这样才能牢固掌握。

Linux的使用涉及面广，大家可以逐个知识点学习，对于命令的熟悉，其实日常的使用积累已经可以达到，不必集中时间专门学习，在需要使用时可搜索其使用方法，并做笔记。而对于Linux系统与Windows的差异，则需要注意，比如在Windows下用svn检出的库文件可能没有链接属性，可执行文件没有可执行属性，等等)。 (各个知识点归纳一下)

第4章 嵌入式Linux开发环境搭建

本章我们提前介绍嵌入式Linux开发环境的搭建。虽然在本书第二部分才接触真正的“嵌入式”，但笔者坚信“工欲善其事，必先利其器”，嵌入式Linux开发环境不仅仅只是设置交叉编译器路径，还包括其它各种环境的搭建：目录共享，源码编写，连接工具等等。

4.1 嵌入式Linux开发概述

开发嵌入式需要大量辅助软件，一般来说，我们是在电脑上开发好程序，然后通过某些方式将程序下载到开发板上运行。

Linux系统

主要作为程序开发的系统，选择用户较多且资源丰富的发行版本，如Ubuntu、CentOS、RedHat、Fedora等，可以安装于虚拟机或物理机上。本书即使用虚拟机安装Ubuntu发行版本。

虚拟机

有条件的推荐使用物理机安装，但一般经常使用VMware虚拟机软件，通过这个软件安装Linux系统，然后设置共享。然后在这个系统上交叉编译。

共享方式

有的人喜欢用VMware自带共享功能，有的人喜欢用samba共享。个人建议在Linux系统中设置samba共享，这样可以在Windows上将Linux共享目录映射成为其中一个盘符，这样做，就可以在windows下操作linux系统的目录或文件了。另一种常见的共享方式是nfs，多用于主机和开发板之间的文件传输。

开发IDE

有的人建议在Linux系统中用vim或emacs，但作为初学者入门，不必如此，使用vim、emacs的学习成本高，且会打击积极性。在samba共享情况下，建议使用Notepad++、source insight或者VS Code进行代码编辑。

交叉编译

交叉编译是嵌入式一个很重要的概念。由于我们编译的程序是在开发板（开发板又称目标板）上运行的，但开发板又没有环境进行编译，所以带出“交叉编译”概念。即在一台linux主机系统上使用交叉编译器对代码进行编译，但编译得到的二进制文件无法在该主机运行，只能在开发板上运行。不同的板子使用的交叉编译器不同。一般使用商家自带的交叉编译器。

下载方式

根据应用场合，可以用jtag烧录器下载程序(适用如u-boot开发)。可以使用tftp方式下载程序(适用kernel开发)。在开发板系统启动后且网络正常情况下，可以使用tftp下载、nfs拷贝等方式进行调试(适用于应用层程序开发)。

开发流程如图4.1所示。

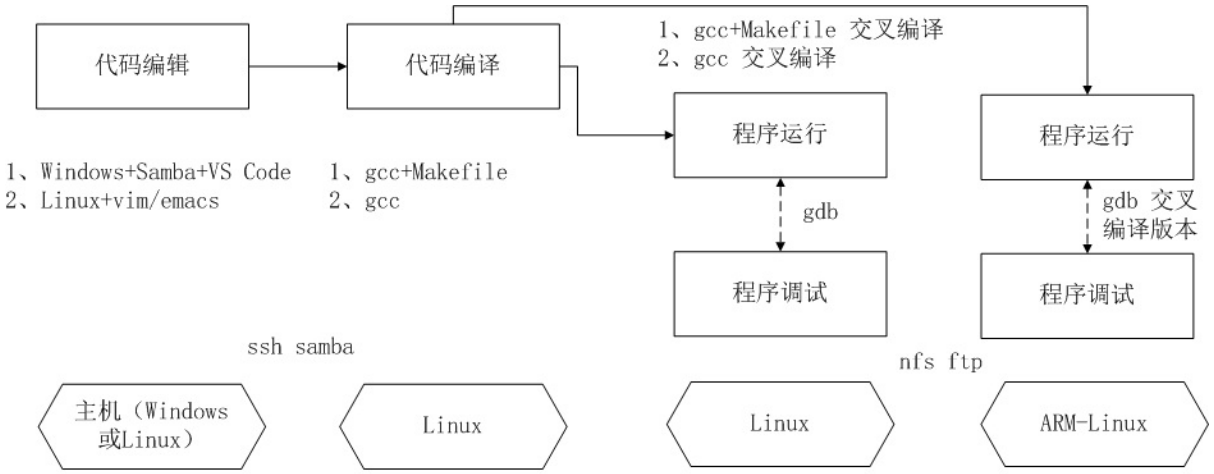

图4.1 开发流程图

系统环境使用熟悉程度越高，越能提高开发速度。笔者曾经供职的单位里，有同事调试硬件设备应用程序，其步骤是：编码、编译、制作软件包、将升级软件烧写设备上，然后重启设备查看结果。这一系列的步骤中，制作包、升级软件包耗时很长，如果能使用NFS，时间能节省三分之二以上。所以，环境的熟悉是十分重要和必要的。

不同人使用的开发环境不尽相同，没有固定的模式。如果一定要定一个原则，那就是：使用自己熟手的工具、方法即可，以提高开发效率为准则。

一开始可能无法找到适合自己的模式，多尝试，慢慢摸索，最终总会有熟练的一天。

4.2 Windows系统所需工具

4.2.1 综合类

思维导图

思维导图是一种图像式思维的工具以及一种利用图像式思考辅助工具。在头脑风暴、会议管理及项目管理工具等应用场合中均有应用。可以进行高效沟通，提高工作、学习效率。

VMware Workstation

虚拟机软件。

云笔记

印象笔记、有道笔记、为知笔记。

版本控制

tortoisegit、tortoisesvn。

4.2.2 编辑类

source insight

代码阅读利器。可跟踪函数调用过程。

VS Code

微软出品的代码编辑器，功能强大。

notepad++

编辑利器，支持列模式。

UltraEdit

编辑利器，支持列模式，支持十六进制显示、修改。

Beyond Compare

对比工具，可用于图片、二进制、代码等文件的比较。

4.2.3 连接工具类

SecureCRT

支持多种协议的连接。如串口、ssh、telnet，等等。

SSH Secure Shell Client

限于ssh，但有文件传输。

tftpd32

一个小巧的tftp服务器(含tftp客户端、dhcp服务器等功能)。

串口调试助手

进行串口功能接收、发送测试，十分有用。另外还可以用来测试串口线是否正常（开启自动发送功能，然后短接2、3脚，就能自发自收）。

4.3 Linux系统所需服务

4.3.1 SSH

在Linux宿主机（或服务器）开启SSH，使用如ssh secure shell client等客户端工具连接、登陆，找到对应目录，可实现相互拷贝。一般地，在嵌入式Linux中只使用SSH客户端功能即可，无法安装SSH服务器。

1、安装：

```
$ sudo apt-get install openssh-server
```

2、重启：

```
$ sudo/etc/init.d/ssh restart
```

3、修改端口

SSH默认服务端口为22，可修改为其它端口，如220，修改配置文件/etc/ssh/sshd_config，将 `Port 22` 改为 `Port 220` 即可。

附SSH Secure Shell Client无法连接ubuntu解决方法

1、编辑**/etc/ssh/sshd_config**配置文件

允许**root**登陆

把 `PermitRootLogin prohibit-password` 改为 `PermitRootLogin yes`

最后添加

```
Ciphers aes128-cbc,aes192-cbc,aes256-cbc,aes128-ctr,aes192-ctr,aes256-ctr,3des-cbc,arcfour128,arcfour256,arcfou
r,blowfish-cbc,cast128-cbc MACs hmac-md5,hmac-sha1,umac-64@openssh.com,hmac-ripemd160,hmac-sha1-96,hmac-md5-96
KexAlgorithms diffie-hellman-group1-sha1,diffie-hellman-group14-sha1,diffie-hellman-group-exchange-sha1,diffie-
hellman-group-exchange-sha256,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group1-sh
a1,curve25519-sha256@libssh.org
```

2、重启ssh服务

```
$ sudo service ssh restart
```

再次使用SSH Secure Shell Client即可连接。

4.3.2 Samba

Linux主机搭建samba服务器后，可以使用windows连接Linux，并建立硬盘映射，这样，就可以将主机当成Windows一个硬盘使用。

1、安装Samba

```
sudo apt-get install samba
sudo apt-get install smbclient (客户端，可选)
```

注：如果安装有错误，则要更新源：

```
sudo apt-get update
```

2、创建共享目录(可选):

```
mkdir /home/latelee/share
sudo chmod 777 /home/latelee/share
```

注意：可以不创建共享目录，直接使用个人的HOME目录。

3、创建Samba配置文件:

3.1、保存现有的配置文件(可选)

```
sudo cp /etc/samba/smb.conf /etc/samba/smb.conf.bak
```

3.2、 修改现配置文件

```
sudo vim /etc/samba/smb.conf
```

在smb.conf最后添加

```
[home] # 用于显示在windows的名称
    comment = samba home # 注释，不要也可以
    path = /home/latelee # 共享目录路径，直接使用登陆用户HOME目录
    writable = yes # 可写
    browseable = yes # 可看
    guest ok = no # 不允许guest
```

注：格式如上，路径根据实际情况改

4、创建samba帐户

```
  sudo touch /etc/samba/smbpasswd (此步不要也行)
  sudo smbpasswd -a latelee(用户名)

New SMB password:(此处密码，建议与登陆密码相同)
Retype new SMB password:(此处密码，建议与登陆密码相同)
```

注：如果没有第四步，登录时会提示 `session setup failed: NT_STATUS_LOGON_FAILURE`

5、重启samba服务器

```
sudo /etc/init.d/samba restart
sudo /etc/init.d/smbd restart (此步可不要)
```

注：不同版本路径、名称可能不同，根据实际情况执行。

6、测试 (不要也行)

```
$ smbclient -L //localhost/共享目录
```

7、使用windows连接

在windows地址栏中输入： `\\<虚拟机IP地址>\home` ，如： `\\172.18.18.8\home` ，然后输入账号密码即可，如图4.2所示。

图4.2 访问Samba

正确打开Sabma共享目录如图4.3所示。

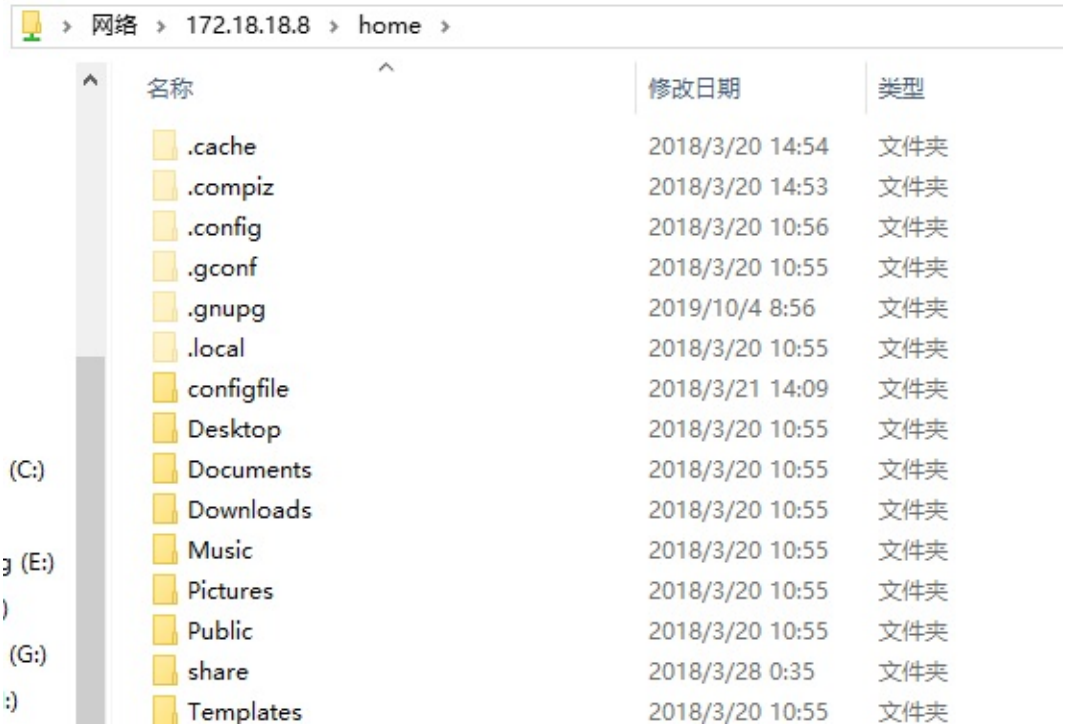

图4.3 共享目录

为方便管理，需要进行磁盘驱动，点击此电脑 -> 映射网络驱动器**(N)...**，选择驱动器（即盘符，此处为Z盘），输入完整的地址，如图4.4所示。

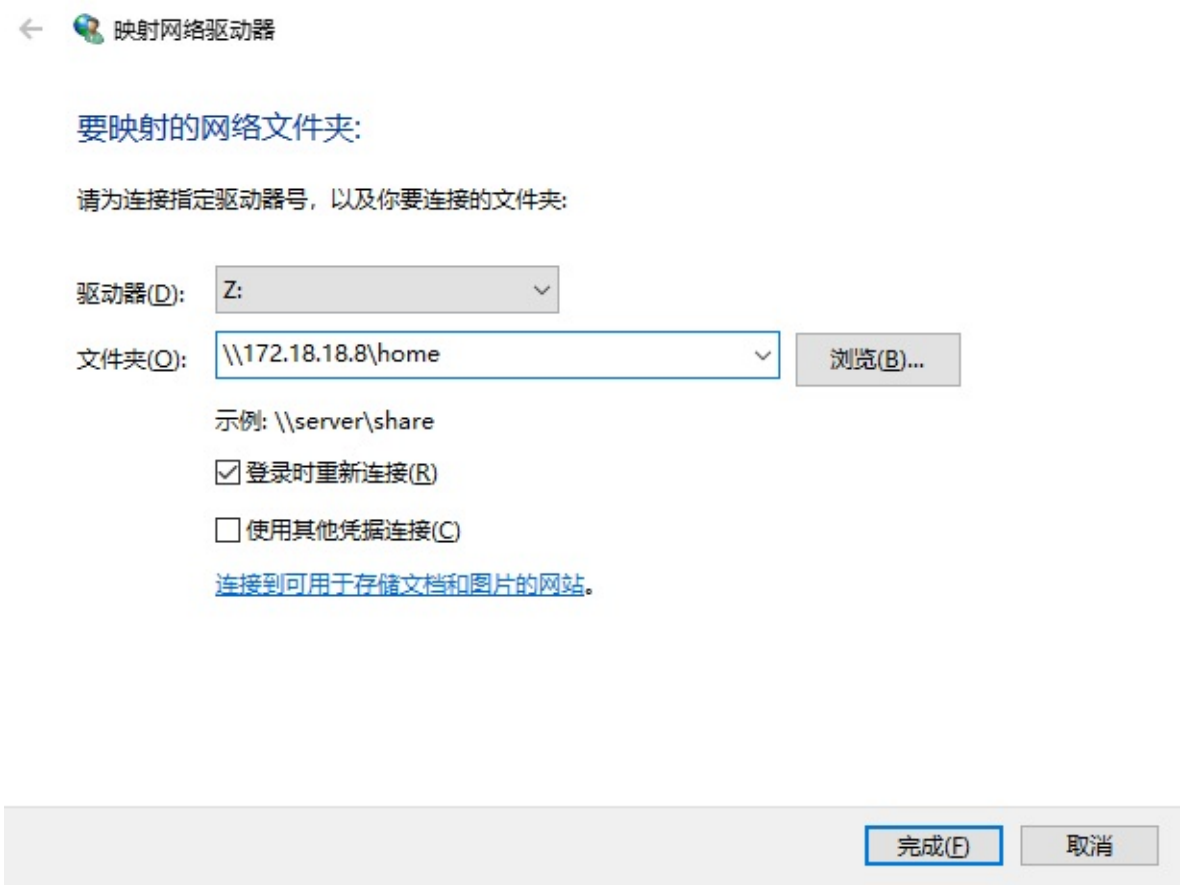

图4.4 映射磁盘

成功后打开共享目录如图4.5所示（注意如图4.3略有不同）。

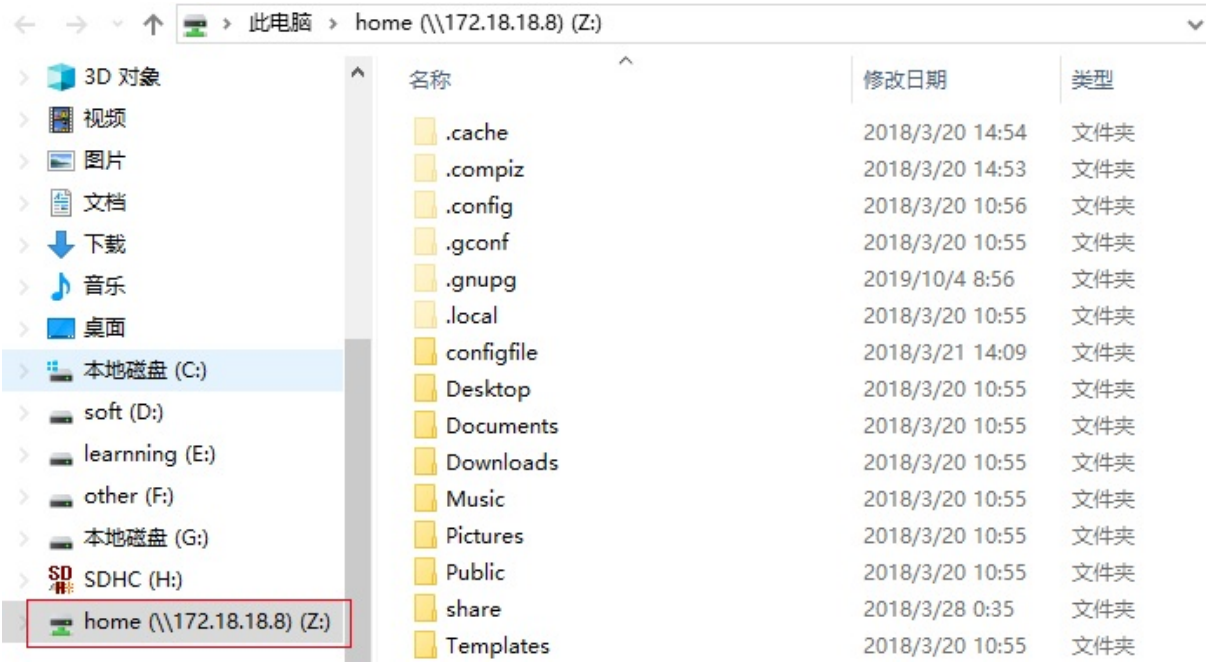

图4.5 共享目录

4.3.3 NFS

Linux宿主机安装了NFS服务后，使用NFS挂载之后，可以将远程主机目录作为本地目录那样使用，十分方便。

1、安装

```
$ sudo apt-get install nfs-kernel-server
```

注：如果失败，可以再尝试一次，或者用命令：

```
$ sudo apt-get update
```

2、配置

编辑文件 `/etc/exports` ，加入下列语句：

```
[共享目录绝对路径] *(rw,no_root_squash,no_all_squash,sync)
```

例如：

```
/opt *(rw,no_root_squash,no_all_squash,sync)
```

注：可添加多个共享目录

3、启动NFS服务

```
$ sudo /etc/init.d/nfs-kernel-server restart
```

4、ARM-Linux挂载测试

命令示例：

```
# mount -t nfs -o nolock 172.18.18.8:/opt /mnt/nfs
```

注意，在嵌入式环境中建议添加 `-o nolock` 选项，否则可以挂载不成功。（待补：列出实际示例）

4.3.4 FTP

4.3.5 Telnet

1、安装：

默认源没有xinetd等软件包，所以要更新源：

```
$ sudo apt-get update
$ sudo apt-get install xinetd telnetd
$ sudo apt-get install inetutils-telnetd
```

2、配置

修改文件 `/etc/xinetd.conf` (xinetd默认为此文件)：

```
# new add by Late Lee
service telnet
{
disable = no
flags = REUSE
socket_type = stream
wait = no
user = root (存疑：root表示执行程序的权限还是登陆用户？从测试中看，是root权限，换其它用户名，会提示telnetd /usr/lib/telnet l
ogin permission denied)
server = /usr/sbin/in.telnetd
log_on_failure += USERID
}
```

3、修改端口

修改 `/etc/services` ，将telnet的23改为其它的不冲突的端口号，如250。

4、重启

```
$ sudo /etc/init.d/xinetd restart
```

5、登陆

命令：

```
telnet ip 端口号
```

6、允许root登陆

ubuntu不允许root用户用telnet来登陆，但可以使用非root用户，与ssh类似。
如果一定要用root登陆，方法有2种：

6.1、将/etc/securetty文件改名

6.2、在/etc/securetty文件最后添加

```
# add pts by Late Lee
pts/0
pts/1
pts/2
pts/3
pts/4
pts/5
pts/6
pts/7
pts/8
pts/9
```

说明： securetty文件规定了root可以从哪些终端登陆，像ssh、telnet等需要用伪终端pts，另外该文件还有如标准终端的 `tty*` (按Ctrl+Alt+数字登陆)、串口 `ttyS*` 、 `ttyUSB*` （USB串口名称）、 `ttyO*` （OMAP系列平台的串口名称），等等。

本节提到的修改端口号，在内部局域网中可以使用默认的，延展一步，如果在公共网络（如云平台），建议将默认的端口改为其它的端口号，以减少服务暴露的风险。

4.4 本章小结

本章节介绍了嵌入式Linux开发环境的搭建，其中Windows系统的工具有思维导图、版本控制软件、编辑器以及连接Linux系统的工具，这些并没有详细介绍，有待各位在日常中积累。Linux系统方面，因为要操作的步骤多，因而详细介绍了各种服务的步骤。需要说明的是，工具的使用，我们不必要深入研究其原因，知道什么工具可以做什么事不能做什么事即可，毕竟工具是为了方便开发的。熟用工具，能够事半功倍。

我们将在下一章节地介绍交叉编译器的安装和使用。

第5章 自动化编译和Makefile

自动化编译是GNU/Linux一大特色，许多开源项目以源码包形式提供用户下载、编译，这些源码包使用Autotools系统构建了编译所需脚本。有的项目除了Linux外，还提供MacOS和Windows的编译脚本。拥有源码前提下，方便我们根据自己的实际情况编译得到库文件或可执行二进制文件。我们可以编译在PC机上使用的库或二进制文件，也可以编译出在嵌入式ARM（或其它芯片）上应用的库或二进制文件。
当然，自行编译源码有一定门槛，总体来说，涉及到编译“三部曲”、交叉编译器以及Makefile的知识。——这就是本章节的内容。

5.1 Linux系统源码编译方法

开源项目除了提供编译好的二进制或库文件外，还会提供源码供用户自行编译，源码的构建工作一般分几个步骤：下载源码、解压源码、编译安装[注1]。

5.1.1 下载源码

源码一般为压缩包形式，或在官网提供，或在其它第三方托管平台下载。下面列出一些源码的下载地址。

项目	下载说明
U-Boot	http://www.denx.de/wiki/U-Boot/，主页左侧"Source Code"，找到"Released Versions"
busybox	https://busybox.net/，主页左侧"Download Binaries"，选择版本
kernel	https://www.kernel.org/，选择tarball下载。也可点击"HTTP"处链接逐步选择
zlib	http://www.zlib.net/，找到"he current release is publicly available here:"处下载

通过这些实例，希望读者知道如何快速找到开源项目的下载地址。

5.1.2 手动解压

压缩包大部分可用tar工具解压，后缀名为 `tar.bz2` 或 `tar.gz` 。也有一些以 `zip` 为后缀，使用 `gunzip` 工具解压。

5.1.3 编译安装

5.1.3.1 配置

配置命令：

```
$ ./configure
```

解压后的源码目录，一般带有configure脚本文件，该文件就是配置所用的脚本。如果没有可执行属性，会有如下提示信息：

```
$ ./configure
-bash: ./configure: Permission denied
```

此时执行 `$ chmod +x configure` 添加可执行属性即可。

configure有很多选项，下面列出一些常用的选。至于源码具体支持的选项，则可以通过"`./configure --help`"查看详细信息。

参数	说明
CC	C编译器
CFLAGS	C编译选项
LDFLAGS	链接选项
LIBS	传递给链接器的库路径
CPPFLAGS	C++编译选项
CPP	C预编译器
-D	自定义宏
--prefix	安装目录，如果不指定，默认使用/usr/local目录
--host	指定主机，一般用于交叉编译场合
--includedir	所需依赖库头文件
--libdir	依赖静态库目录
--sharedlibdir	依赖动态库目录
--static	指定编译为静态库
--disable-XXX	禁止某些特性
--enable-XXX	使能某些特性
--with-PACKAGE	使用某个包
--without-PACKAGE	不使用某个包

成功后会生成Makefile。 总的来说，`./configure` 就是根据我们的需求配置好各种依赖条件，编译选项，如果Makefile还达不到我们的目的，则要手动修改。另外有些开源项目源码包已带有Makefile，那么就不需要进行配置了。

5.1.3.2 编译

编译命令：

```
$ make
```

如系统是多核的，一般可以使用 `make -j<线程数>` ，如：

```
$ make -j4
```

表示使用4线程编译，类似的可用8线程编译。这样可以加快编译速度。

5.1.3.3 安装

安装命令：

```
$ make install
```

将所得结果安装到前面配置时使用 `--prefix` 指定的目录。该目录下一般有下列的子目录：

参数	说明
bin	二进制文件所在目录
include	该库头文件
lib	库文件
share	man帮助信息

如果我们只需要二进制文件，取bin目录的文件即可。如果是库文件，需要include目录和lib目录，share目录的作用不是很大。

5.1.4 编译实例

- openssl-0.9.8e

```
$ ./configure linux-elf-arm -DB_ENDIAN linux:'arm-inux-gcc' \
--prefix=/home/latelee/bin/sip_new/ssl
$ make
$ make install
```

- libosip2-3.6.0

```
$ ./configure --prefix=/home/latelee/bin/sip_new/libosip2 \
CC=arm-inux-gcc --host=arm-linux -enable-static
$ make
$ make install
```

> 注1. 这背后的原理涉及到GNU构建系统和Autotools（ Autoconf、Automake、Libtool），有兴趣的读者可了解其概念，但当前阶段不建议深入研究。 ↩

5.2 安装交叉编译器

5.2.1 获取

获取交叉编译器的方式有很多种。
如果以个人名义购买开发板，一般会提供光盘或网盘压缩包，里面有交叉编译器，如果没有，可以到官方网站下载（国内有的厂家权限管理较严，必须购买板子才提供账号）。
如果以公司名义签署NDA拿到软件开发包，都会附带交叉编译器，这些编译器有可能是原厂制作的，也可能是从第三方获取的。

如果上两种途径没有，也可以在网上搜索得到。下面是一些提供下载的地址：

- http://ftp.arm.linux.org.uk/pub/armlinux/toolchain/
- https://releases.linaro.org/components/toolchain/binaries//
- https://releases.linaro.org/components/toolchain/binaries//
- https://www.mentor.com/embedded-software/sourcery-tools/support

建议读者首先厂家提供的编译器，其次使用第三方做好的交叉编译器。

5.2.2 安装

一般厂家或第三方提供的交叉编译器为压缩包形式，需要自行进行解压到自定义目录。接着设置环境变量PATH的值，即将交叉编译的bin目录设置到PATH中。但是目前有很多芯片的开发环境比较复杂（或者将很多库、功能集中在一直，或者依赖工具、库太多），需要专门的脚本才能设置好。这里暂时不涉及。
设置PATH的格式是：

```
export PATH=<交叉编译器bin目录的绝对路径>:$PATH
```

一般在 `~/.bashrc` 文件最后添加即可。添加完成后，需要执行：

```
source ~/.bashrc
```

命令以便更新环境变量。另外，如果是团队协作，可以考虑在 `/etc/profile` 中添加，这样所有登陆用户都可使用。

下面是笔者的设置示例（在 `~/.bashrc` 文件末尾，笔者有2个交叉编译器，所以设置了2行）：

```
export PATH=$HOME/x-tools/arm-unknown-linux-gnueabihf/bin:$PATH

export PATH=~/bin/gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12/fsl-linaro-toolchain/bin/:$PATH
```

注： `~` 表示用户的HOME目录，比如在笔者虚拟机上就表示 `/home/latelee` ，路径目录可使用绝对路径，也可使用 `~` 或 `$HOME` 。

现在，主流的Linux系统发行版本可直接安装交叉编译器了，以ubuntu为例，直接使用 `apt-get` 来安装：

```
sudo apt-get install -y gcc-arm-linux-gnueabi g++-arm-linux-gnueabi      # 普通版本
sudo apt-get install -y gcc-arm-linux-gnueabihf  g++-arm-linux-gnueabihf # 硬件浮点版本
```

这样安装的交叉编译器版本与系统源有关。比如笔者安装的 `arm-linux-gnueabi-gcc` 版本为5.4.0，与 `gcc` 版本相同。

5.2.3 验证

在命令行输入 `<交叉编译器名称> -v[或--version]` 即可查看编译器版本，也可验证是否正确安装。
直接输入 `arm` ，再按2次 `Tab` 键，输入 `y` 即可查看所有以 `arm` 开头的命令。 笔者在虚拟机上安装了2个，如下：

```
$ arm-fsl-linux-gnueabi-gcc --version
arm-fsl-linux-gnueabi-gcc (Freescale MAD -- Linaro 2011.07 -- Built at 2011/08/10 09:20) 4.6.2 20110630 (prerel
ease)
$ arm-unknown-linux-gnueabihf-gcc --version
arm-unknown-linux-gnueabihf-gcc (crosstool-NG crosstool-ng-1.23.0) 6.3.0
```

`arm-fsl-linux-gnueabi-gcc` 是由 `Linaro` 出品的，而 `arm-unknown-linux-gnueabihf-gcc` 则是由笔者使用 `crosstool-ng` 编译出来的。

5.2.4 使用

交叉编译器的使用与其它编译器没有什么不同。以Makefile为例，可以在Makefile文件中添加或修改CROSS_COMPILE选项，也可以在make的时候手动指定CROSS_COMPILE的值。下面是编译内核的一个例子：

```
make zImage CROSS_COMPILE=arm-fsl-linux-gnueabi- ARCH=arm
```

注：CROSS_COMPILE是默认的交叉编译器前缀。

5.3 自制交叉编译器

本节我们自行制作一个属于自己的交叉编译器，使用的制作工具为crosstool-ng，其全称是crosstool Next Generation，即下一代crosstool。crosstool-ng的特点如下：

- 支持menuconfig（类似于Linux内核配置）
- 支持众多的架构
- 可选多种不同的C库等模块
- 提供示例sample配置
- 支持多种主机编译环境

5.3.1 安装必备工具

安装依赖工具：

```
$ sudo apt-get install -y bison flex gperf texinfo gawk libtool automake libncurses5-dev help2man
```

5.3.2 下载源码包

进入任意目录，使用如下命令下载crosstool-ng源码压缩包（版本为1.23.0）：

```
$ wget http://crosstool-ng.org/download/crosstool-ng/crosstool-ng-1.23.0.tar.xz
```

也可以打开http://crosstool-ng.org/download/crosstool-ng/选择压缩包下载。
下载成功后，解压并进入crosstool-ng目录：

```
$ tar xf crosstool-ng-1.23.0.tar.xz
$ cd crosstool-ng-1.23.0/
```

5.3.3 编译安装

由于crosstool-ng没有提供现成的可执行二进制文件，所以需要手动编译源码。我们将crosstool-ng安装到另外的目录中。因为

```
$ sudo mkdir -p /opt/ellp/crosstool-ng  # 创建目录
$ sudo chmod 777 /opt/ellp/crosstool-ng # 设置权限
$ ./configure --prefix=/opt/ellp/crosstool-ng # 指定目录
$ make # 编译
$ make install # 安装
$ sudo cp /opt/ellp/crosstool-ng/bin/ct-ng /usr/local/bin/ # 拷贝到系统目录
```

安装的ct-ng工具位于 `/opt/ellp/crosstool-ng/bin/` 目录，为方便使用，将其拷贝到系统$PATH目录中。输入命令 `ct-ng --help` 验证是否安装成功。

5.3.4 配置交叉编译器

进入目录 `/opt/ellp/crosstool-ng` [注1]。

```
$ cd /opt/ellp/crosstool-ng/
$ ls
bin  lib  share
```

拷贝已有的配置模板：

```
$ cp lib/crosstool-ng-1.23.0/samples/arm-unknown-linux-gnueabi/crosstool.config .config
```

对交叉编译器进行配置：

```
$ ct-ng menuconfig
```

之后将出现menuconfig界面，该界面使用ncurses实现，在Linux开发使用比较广。下面简单解析主要的配置选项。注意，我们使用的是ARM芯片配置文件arm-unknown-linux-gnueabi，该配置已经非常完备——包括所需要依赖软件包以及对应的版本，只需要修改个别地方即可。以下均保留原始提示信息，`###` 后为需要修改的地方，`#` 表示一般注释。

- 路径相关

```
Paths and misc options  --->
  *** crosstool-NG behavior ***
 [ ] Use obsolete features (NEW)
 [ ] Try features marked as EXPERIMENTAL (NEW)
 [ ] Debug crosstool-NG (NEW)
     *** Paths ***
 (${HOME}/src) Local tarballs directory (NEW) # 这是依赖包所在目录，在编译时自动下载
 [*]   Save new tarballs (NEW)                # 默认会保存依赖包，方便下次再编译时使用
 (${CT_TOP_DIR}/.build) Working directory (NEW) # 编译目录，无须理会
 (${CT_PREFIX:-${HOME}/x-tools}/${CT_HOST:+HOST-${CT_HOST}/}${CT_TARGET}) Prefix directory (NEW) # 编译后的安
装工具，默认即可
 [*]   Remove the prefix dir prior to building (NEW)
 [ ] Remove documentation
 [ ]   Build the manuals (NEW)
 [*] Render the toolchain read-only (NEW)
 [*] Strip host toolchain executables (NEW)
 [ ] Strip target toolchain executables (NEW)
     *** Downloading ***
     Download agent (wget)  --->
 [ ] Forbid downloads (NEW)
 [ ]   Force downloads (NEW)
 (10)  Connection timeout (NEW)
 (--passive-ftp --tries=3 -nc --progress=dot:binary) Extra options to wget (NEW)
 [ ]   Stop after downloading tarballs (NEW)
 [ ]   Use a mirror (NEW)
     *** Extracting ***
 [ ] Force extractions (NEW)
 [*] Override config.{guess,sub} (NEW)
 [ ] Stop after extracting tarballs (NEW)
     Patches origin (Bundled only)  --->
     *** Build behavior ***
 (0) Number of parallel jobs (NEW) ### 并行数量，一般为CPU核心数，可添加编译速度，如果在虚拟机，建议改为0或2
 ()  Maximum allowed load (NEW)
 [*] Use -pipe (NEW)
 ()  Extra build compiler flags (NEW)
```

```
 ()  Extra build linker flags (NEW)
 ()  Extra host compiler flags (NEW)
 ()  Extra host linker flags (NEW)
     Shell to use as CONFIG_SHELL (bash)  --->
     *** Logging ***
     Maximum log level to see: (EXTRA)  --->
 [ ] Warnings from the tools' builds (NEW)
 [*] Progress bar (NEW)
 [*] Log to a file (NEW)
 [*]   Compress the log file (NEW)
```

- 目标选项

```
Target options  --->
  Target Architecture (arm)  ---> # 目标硬件平台，默认为arm，其它还有alpha、avr、mips、powerpc、x86
()  Suffix to the arch-part (NEW)
    *** Generic target options ***
[ ] Build a multilib toolchain (READ HELP!!!) (NEW)
[*]   Attempt to combine libraries into a single directory (NEW)
[*] Use the MMU (NEW)
    Endianness: (Little endian)  ---> # 默认为小端模式，有些芯片为大端模式
    Bitness: (32-bit)  ---> # 默认为32位，可改为64位
    *** Target optimisations ***
()  Architecture level (NEW)
()  Emit assembly for CPU (NEW)
()  Tune for CPU (NEW)
()  Use specific FPU (NEW)  ### 按回车，输入neon，这里假定硬件芯片支持neon指令
    Floating point: (software (no FPU))  ---> ### 按回车，选择hardware (FPU)，假定支持硬浮点
()  Target CFLAGS (NEW)
()  Target LDFLAGS (NEW)
    *** arm other options ***
    Default instruction set mode (arm)  ---> #默认指令集为arm，另有thumb指令集
[ ] Use Thumb-interworking (READ HELP) (NEW)
-*- Use EABI
```

- 交叉工具链选项

```
Toolchain options  --->
  *** General toolchain options ***
-*- Use sysroot'ed toolchain
(sysroot) sysroot directory name (NEW)
()    sysroot prefix dir (READ HELP) (NEW)
[ ] Build Static Toolchain (NEW)
()  Toolchain ID string (NEW)
()  Toolchain bug URL (NEW)
    *** Tuple completion and aliasing ***
(unknown) Tuple's vendor string (NEW)  # 芯片字符串，默认即可
()  Tuple's sed transform (NEW)
()  Tuple's alias (NEW) ### 按回车，输入arm-linux。编译器各工具的别名，如arm-linux-gcc
    *** Toolchain type ***
    Type (Cross)  --->
    *** Build system ***
()    Tuple        (READ HELP!) (NEW)
()    Tools prefix (READ HELP!) (NEW)
()    Tools suffix (READ HELP!) (NEW)
    *** Misc options ***
[ ] Enable nls (NEW)
```

- 操作系统

```
Operating System  --->
   Target OS (linux)  ---> # 默认为linux，表示程序在跑linux内核上，另有Bare metal表示跑在无内核的系统
[ ] custom tarball or directory  # 自定义内核（压缩包或目录），这里不选择
      Linux kernel version (4.10.8)  ---> # 选择内核版本，本版本支持的内核最高为4.10.8
```

```
    *** Common kernel options ***
[*] Build shared libraries
    *** linux other options ***
    Kernel verbosity: (Simplified)  --->
[*] Check installed headers
```

- 二进制工具

```
Binary utilities  --->
  Binary format: (ELF)  ---> # 二进制格式，Linux下为ELF格式
Binutils (binutils)  --->
*** GNU binutils ***
[ ] Show Linaro versions
binutils version (2.28)  --->
Linkers to enable (ld, gold)  --->
[*]   Enable threaded gold
[*]   Add ld wrapper
[*] Enable support for plugins
()  binutils extra config
[ ] binutils libraries for the target
*** binutils other options ***
```

- C库

```
C-library  --->
  C library (glibc)  --->
[ ] Show Linaro versions
    glibc version (2.25)  ---> # glibc版本
    *** Common C library options ***
    Threading implementation to use: (native)  --->
[ ] Create /etc/ld.so.conf file
[*] Install a cross ldd-like helper
    *** glibc other options ***
()  extra config
()  Extra config params (READ HELP)
()  extra target CFLAGS
[ ] Disable symbols versioning
()  Oldest supported ABI
[*] Force unwind support (READ HELP!)
()  Extra addons
[ ] Build and install locales
    Minimum supported kernel version (Same as kernel headers (default))  --->
```

- C编译器

```
C compiler  --->
 C compiler (gcc)  --->
[ ] Show Linaro versions
    gcc version (6.3.0)  ---> # gcc版本为6.3.0
()  Flags to pass to --enable-cxx-flags
()  Core gcc extra config
()  gcc extra config
[*] Link libstdc++ statically into the gcc binary
[ ] Use system zlib
<M> Configure TLS (Thread Local Storage)
    *** Optimisation features ***
[*] Enable GRAPHITE loop optimisations
[*] Enable LTO
    *** Settings for libraries running on target ***
[*] Optimize gcc libs for size
[ ] Compile libmudflap
[ ] Compile libgomp
[ ] Compile libssp
[ ] Compile libquadmath
```

```
[ ] Compile libsanitizer
    *** Misc. obscure options. ***
[*] Use __cxa_atexit
[ ] Do not build PCH
< > Use sjlj for exceptions
<M> Enable 128-bit long doubles
[ ] Enable build-id
    linker hash style (Default)  --->
    Decimal floats (auto)  --->
    *** Additional supported languages: ***
[*] C++ # 只使用C++，其它语言不需要
[ ] Fortran
[ ] Java
```

- 其它
  余下的保留默认值即可。

```
Debug facilities  ---> # 调试相关，gdb、ltrace、strace
Companion libraries  --->   # 依赖库，libiconv、GMP、ncurses
Companion tools  ---> # 依赖工具，autoconf、automake、libtool
```

5.3.5 构建交叉编译器

```
$ time ct-ng build
```

该步骤编译耗时较久，因为要从网络下载软件包，还需要解压、编译、安装，下面是构建的部分输出信息[注2]：

```
[INFO ]  Performing some trivial sanity checks
[INFO ]  Build started 20191011.181638
[INFO ]  Building environment variables
[EXTRA]  Preparing working directories
[EXTRA]  Installing user-supplied crosstool-NG configuration
[EXTRA]  ===================================================================
[EXTRA]  Dumping internal crosstool-NG configuration
[EXTRA]    Building a toolchain for:
[EXTRA]      build  = x86_64-pc-linux-gnu
[EXTRA]      host   = x86_64-pc-linux-gnu
[EXTRA]      target = arm-unknown-linux-gnueabihf
// ...
[INFO ]  ===================================================================
[INFO ]  Installing strace
[EXTRA]    Configuring strace
[EXTRA]    Building strace
[EXTRA]    Installing strace
[INFO ]  Installing strace: done in 75.96s (at 107:50)
[INFO ]  ===================================================================
[INFO ]  Finalizing the toolchain's directory
[INFO ]    Stripping all toolchain executables
[EXTRA]    Installing the populate helper
[EXTRA]    Installing a cross-ldd helper
[EXTRA]    Creating toolchain aliases
[INFO ]  Finalizing the toolchain's directory: done in 7.59s (at 107:57)
[INFO ]  Build completed at 20191011.200429
[INFO ]  (elapsed: 107:51.25)
[INFO ]  Finishing installation (may take a few seconds)...
[107:57] /
real    108m13.378s
user    140m17.464s
sys     20m22.204s
```

最终生成的文件位于 `~/x-tools` 目录。

5.3.6 设置PATH环境变量

- 1、将 `export PATH=$HOME/x-tools/arm-unknown-linux-gnueabihf/bin:$PATH` 写到 `~/.bashrc` 文件末尾。
- 2、执行 `source ~/.bashrc` 或重启机器使交叉编译器生效。
- 3、交叉编译器版本如下：

```
$ arm-linux-gcc -v
Using built-in specs.
COLLECT_GCC=./bin/arm-linux-gcc
COLLECT_LTO_WRAPPER=/home/latelee/x-tools/arm-unknown-linux-gnueabihf/libexec/gcc/arm-unknown-linux-gnueabi
hf/6.3.0/lto-wrapper
Target: arm-unknown-linux-gnueabihf
Configured with: /opt/ellp/crosstool-ng/.build/src/gcc-6.3.0/configure --build=x86_64-build_pc-linux-gnu --
host=x86_64-build_pc-linux-gnu --target=arm-unknown-linux-gnueabihf --prefix=/home/latelee/x-tools/arm-unkn
own-linux-gnueabihf --with-sysroot=/home/latelee/x-tools/arm-unknown-linux-gnueabihf/arm-unknown-linux-gnue
abihf/sysroot --enable-languages=c,c++ --with-fpu=neon --with-float=hard --with-pkgversion='crosstool-NG cr
osstool-ng-1.23.0' --disable-sjlj-exceptions --enable-__cxa_atexit --disable-libmudflap --disable-libgomp -
-disable-libssp --disable-libquadmath --disable-libquadmath-support --disable-libsanitizer --disable-libmpx
 --with-gmp=/opt/ellp/crosstool-ng/.build/arm-unknown-linux-gnueabihf/buildtools --with-mpfr=/opt/ellp/cros
stool-ng/.build/arm-unknown-linux-gnueabihf/buildtools --with-mpc=/opt/ellp/crosstool-ng/.build/arm-unknown
-linux-gnueabihf/buildtools --with-isl=/opt/ellp/crosstool-ng/.build/arm-unknown-linux-gnueabihf/buildtools
 --enable-lto --with-host-libstdcxx='-static-libgcc -Wl,-Bstatic,-lstdc++,-Bdynamic -lm' --enable-threads=p
osix --enable-target-optspace --enable-plugin --enable-gold --disable-nls --disable-multilib --with-local-p
refix=/home/latelee/x-tools/arm-unknown-linux-gnueabihf/arm-unknown-linux-gnueabihf/sysroot --enable-long-l
ong
Thread model: posix
gcc version 6.3.0 (crosstool-NG crosstool-ng-1.23.0)
```

5.3.7 交叉编译器小知识

每个交叉编译器的名称都会有所不同，而且名称都比较长。具体可到交叉编译器的bin目录查看。但是，一般情况，都会提供 `arm-linux-gcc` 这类简洁版本的文件——其实它们是链接文件。

交叉编译器的名称规则为 `<前缀>-<工具名称>` 。简述如下：

- 前缀
  `<架构>-<CPU核心/或芯片厂家>-<所运行的操作系统>-<编译器库和目标镜像规范>`
  如 `arm-fsl-linux-gnueabi-` 分别表示：arm芯片，飞思尔卡平台（fsl），生成的文件运行在linux系统，接口规范为 `gnueabi` [注3]。
- 工具名称
  工具名称表示 `binary utils` ，比如编译器为 `gcc` ，调试器为 `gdb` ，链接器为 `ld` ，等等。

综合起来就构成很多不同的工具，列举如下：

```
$ arm-fsl-linux-gnueabi-  # 按2次Tab键
arm-fsl-linux-gnueabi-addr2line     arm-fsl-linux-gnueabi-gcc           arm-fsl-linux-gnueabi-objcopy
arm-fsl-linux-gnueabi-ar            arm-fsl-linux-gnueabi-gcc-4.6.2     arm-fsl-linux-gnueabi-objdump
arm-fsl-linux-gnueabi-as            arm-fsl-linux-gnueabi-gcov          arm-fsl-linux-gnueabi-populate
arm-fsl-linux-gnueabi-c++           arm-fsl-linux-gnueabi-gdb           arm-fsl-linux-gnueabi-ranlib
arm-fsl-linux-gnueabi-cc            arm-fsl-linux-gnueabi-gdbtui        arm-fsl-linux-gnueabi-readelf
arm-fsl-linux-gnueabi-c++filt       arm-fsl-linux-gnueabi-gprof         arm-fsl-linux-gnueabi-run
arm-fsl-linux-gnueabi-cpp           arm-fsl-linux-gnueabi-ld            arm-fsl-linux-gnueabi-size
arm-fsl-linux-gnueabi-ct-ng.config  arm-fsl-linux-gnueabi-ld.bfd        arm-fsl-linux-gnueabi-strings
arm-fsl-linux-gnueabi-elfedit       arm-fsl-linux-gnueabi-ldd           arm-fsl-linux-gnueabi-strip
arm-fsl-linux-gnueabi-g++           arm-fsl-linux-gnueabi-nm

$ arm-unknown-linux-gnueabihf-  # 按2次Tab键
arm-unknown-linux-gnueabihf-addr2line     arm-unknown-linux-gnueabihf-gcov-tool
arm-unknown-linux-gnueabihf-ar            arm-unknown-linux-gnueabihf-gdb
arm-unknown-linux-gnueabihf-as            arm-unknown-linux-gnueabihf-gprof
arm-unknown-linux-gnueabihf-c++           arm-unknown-linux-gnueabihf-ld
```

```
arm-unknown-linux-gnueabihf-cc             arm-unknown-linux-gnueabihf-ld.bfd
arm-unknown-linux-gnueabihf-c++filt        arm-unknown-linux-gnueabihf-ldd
arm-unknown-linux-gnueabihf-cpp            arm-unknown-linux-gnueabihf-ld.gold
arm-unknown-linux-gnueabihf-ct-ng.config   arm-unknown-linux-gnueabihf-nm
arm-unknown-linux-gnueabihf-dwp            arm-unknown-linux-gnueabihf-objcopy
arm-unknown-linux-gnueabihf-elfedit        arm-unknown-linux-gnueabihf-objdump
arm-unknown-linux-gnueabihf-g++            arm-unknown-linux-gnueabihf-populate
arm-unknown-linux-gnueabihf-gcc            arm-unknown-linux-gnueabihf-ranlib
arm-unknown-linux-gnueabihf-gcc-6.3.0      arm-unknown-linux-gnueabihf-readelf
arm-unknown-linux-gnueabihf-gcc-ar         arm-unknown-linux-gnueabihf-size
arm-unknown-linux-gnueabihf-gcc-nm         arm-unknown-linux-gnueabihf-strings
arm-unknown-linux-gnueabihf-gcc-ranlib     arm-unknown-linux-gnueabihf-strip
arm-unknown-linux-gnueabihf-gcov
```

> 注1. 因为我们前面指定了 `/opt/ellp/crosstool-ng` 目录，所以必须进入该目录，读者可选择其它目录。也因为如此，`ct-ng` 不必要拷贝到系统目录，这样在执行 `ct-ng` 时使用 `./bin/ct-ng` 即可。 ↩

> 注2. 这部分的输出信息，实际是笔者第二次编译所得的，使用已下载的软件包，因为在虚拟机中编译，所以耗时非常大，超过100分钟。 ↩

> 注3. 关于abi、eabi、gnueabi、gnueabihf等的知识，请自行了解。 ↩

5.4 Makefile

要进行Linux开发，不得不接触Makefile，如果看到u-boot或Linux内核源码的Makefile，会顿时觉得Makefile太复杂太难学了，这是因为复杂的源码需要复杂的Makefile才能处理得了。本节我们学习简单的Makefile知识，最后给出Makefile模板。Makefile属于工具类的知识点，在已有基础上能修改并扩展到实际项目中即可，无须研究内部原理。包括：

- 在大型开源项目（如kernel）中如何通过Makefile找到所需源码文件。
- 如何在内核中仿照Makefile添加自己的驱动。
- 一个全新项目，如何构造项目源码树，如何使用Makefile管理。

5.4.1 Makefile基础知识

Makefile是一个文件，有自己的一套规则、语法[注1]，包括编译顺序，编译、链接、安装规则，等等。 Makefile规则有三个要素，分别是：

- 目标
- 依赖
- 命令

格式如下：

```
目标: 依赖
tab 命令
```

命令前面一定是**tab**，不能是空格[注2]。

下面以一个简单C工程为例对Makefile进行讲解：

```
.
├── bar.c
├── bar.h
├── foo.c
├── foo.h
├── main.c
└── Makefile
```

该工程设有头文件和实现文件，其中main.c文件包含主函数main，最终编译生成可执行二进制文件a.out。

5.4.1.1 基本形式

最简单的Makefile如下：

```
a.out: main.c foo.c bar.c
    gcc main.c foo.c bar.c -o a.out
```

目标为可执行二进制a.out，依赖为实现源码文件，命令是一条完整的编译（及链接）命令。

如果将编译和链接分开，变成如下形式：

```
a.out: main.o foo.o bar.o
    gcc main.o foo.o bar.o -o a.out

main.o: main.c
    gcc -c main.c

foo.o: foo.c
    gcc -c foo.c

bar.o: bar.c
    gcc -c bar.c
```

前一部分是将.o文件链接生成二进制文件，后一部分是依次对源码进行编译。

5.4.1.2 规则初现

- 自定义变量

自定义变量使用 `$` 标识，形式为 `$(var)` ，变量名称可随意定义，但一般有一些约定的变量，有些为内置变量，但这些变量也可覆盖赋值。列举如下：

变量	说明
target	最后的目标文件，可以是库或二进制文件
CROSS_COMPILE	交叉编译器前缀，非交叉编译情况保留为空
CC	内置变量，默认为gcc或cc，C编译器， 很多开源项目使用 `$(CROSS_COMPILE)gcc` 形式
CXX	内置变量，默认为g++，C++编译器
AR	内置变量，默认为ar，打包工具，生成静态库
RM	内置变量，默认为rm -f，强制删除
CFLAGS	C编译选项
CXXFLAGS	C++编译选项
LDFLAGS	链接选项
LIBS	库文件
SRCS	源文件
OBJS	目标文件（.o文件）
其它	其它变量根据实际情况定义

变量的赋值有几种形式，说明如下：

赋值运算符	说明
=	最基本的赋值
?=	如未赋值，则赋值之，适用于设置默认值

?=	如未赋值，则赋值之，适用于设置默认值
:=	覆盖之前赋的值
+=	追加赋值

- 自动变量

除自定义变量外，Makefile还有一些内置的自动变量，这些自动变量无须赋值，直接使用即可：

变量	说明
$@	目标集合
$%	当目标是函数库文件时, 表示其中的目标文件名
$<	第一个依赖目标. 如果依赖目标是多个, 逐个表示依赖目标
$?	比目标新的依赖目标的集合，，以空格分隔
$^	所有依赖目标的集合, 会去除重复的依赖目标
$+	所有依赖目标的集合, 不会去除重复的依赖目标
$*	这个是GNU make特有的, 其它的make程序不一定支持

- 隐含规则

为简化编译流程，Makefile引入了隐含规则，隐含规则就是Makefile在背后根据一些约定的机制，自动帮我们完成某些事，如把.c文件编译成.o文件，编译的命令为 `gcc -c xx.c` ，这个过程就是自动完成的。比如：

```
a.out: foo.o bar.o
    gcc –o a.out foo.o bar.o $(CFLAGS) $(LDFLAGS)
```

这里并没有指定如何得到foo.o和bar.o文件，但有了隐含规则，在执行make时会自动帮我们推导出如何得到（编译）这些文件。不过，默认的推导规则并不一定适合所有的应用场合，因此，实际工作中，还是需要手动指定一些规则的。

了解这些知识后，Makefile变成：

```
target=a.out
SRCS=main.c foo.c bar.c
OBJS=main.o foo.o bar.o

$(target):$(OBJS)
    $(CC) $^ -o $@

# 注意，此处不指定如何得到$(OBJS)文件
```

5.4.1.3 条件判断

Makefile支持条件判断，可以根据不同的变量值执行不同的操作。以ifeq为例，语法如下：

```
ifeq (arg1, arg2)
    text-if-true
else
    text-if-false
endif
```

比较arg1和arg2两个参数是否相同，如是为真，执行text-if-true部分语句，否则为假，执行text-if-false部分语句。
下表列出四种条件判断语句的关键字，每一种均有else和endif，判断逻辑亦相似。如下：

变量	说明
ifeq (arg1, arg2)	比较arg1和arg2参数的值是否相同。如是为真，否则为假。另外还有 `ifeq 'arg1' 'arg2'` ， `ifeq "arg1" "arg2"` 等形式，此处只取一种，下同
ifneq (arg1, arg2)	比较arg1和arg2参数的值是不相同。如是为真，否则为假。
ifdef variable-name	如果定义variable-name（即值为非空），则为真，否则为假。
ifndef variable-name	如果没有定义variable-name（值为空），则为真，否则为假。

注意，条件判断依然属于赋值阶段，语句前面可使用空格键，但不能使用**Tab**键，否则会被认为是命令。

举例如下：

```
# 例1：判断$(CC)变量是否“gcc”，如是则使用GNU库。
libs_for_gcc = -lgnu
normal_libs =
ifeq ($(CC),gcc)
  libs=$(libs_for_gcc)
else
  libs=$(normal_libs)
endif
foo: $(objects)
    $(CC) -o foo $(objects) $(libs)

# 例2：如果foo被赋值，则aa为yes，如未赋值，则为no
foo = helloworld
ifdef foo
aa = yes
else
aa = no
endif

# 例3：如果foo为空，则aa为yes，如被赋值，则为no
foo =
ifndef foo
aa = yes
else
aa = no
endif

# 例4：此为特殊示例，由于foo已经被赋值了（即使bar为空），所以aa结果为yes
bar =
foo = $(bar)
ifdef foo
aa = yes
else
aa = no
endif
```

5.4.1.4 内置函数

Makefile支持很多有用的函数，这些函数同样是为了简化处理的流程。与变量类似，也是使用 `$` 标识。语法如下：

```
$(<函数名> <参数>)
${<函数名> <参数>}
```

参数间以英文逗号“,”隔开，而函数名和参数之间以“空格”分隔。下面介绍一些常见函数。

- 扩展通配符函数：wildcard
  语法： `$(wildcard <pattern...>)` 。
  功能：查找符合pattern的对象。
  返回：匹配的对象。
  示例： `SRCS=$(wildcard ./*.c)` 查找当前目录所有的C文件，并赋值为SRCS。

- 模式字符串替换函数：patsubst 语法： `$(patsubst <pattern>,<replacement>,<text>)` 。
  功能：查找text中的单词（单词以“空格”、“Tab”或“回车”“换行”分隔）是否符合模式pattern，如果匹配的话，则以replacement替换。
  返回：被替换过后的字符串。
  示例： `OBJS=$(patsubst ./%.c, ./%.o, $(SRCS))` 将SRCS目录的.c文件替换成.o文件：

- 字符串替换函数：subst
  语法： `$(subst <from>,<to>,<text>)` 。 功能：把字串text中的from字符串替换成to。
  返回：被替换过后的字符串。
  示例： `$(subst foo,bar,foot on the street)` 将foot替换为bart。

- 过滤函数：filter
  语法： `$(filter <pattern...>,<text>)` 。
  功能：以pattern模式过滤text字符串中的单词，保留符合模式pattern的单词。可以有多个模式。
  返回：符合模式的字串。

- 获取目录函数：dir
  语法： `$(dir <names...>)` 。
  功能：从文件names中取出目录部分。
  返回：目录。如果参数不带目录，则返回当前目录。
  示例： `$(dir src/foo.c bar.c)` 返回值是“src/ ./”。

- 取文件函数：notdir
  语法： `$(notdir <names...>)` 。
  功能：从文件names中取出非目录部分，亦即文件名。
  返回：文件名称。 示例： `$(notdir src/foo.c bar.c)` 返回值是“foo.c bar.c”。

- 取后缀函数：suffix
  语法： `$(suffix <names...>)` 。
  功能：从文件names中取出各个文件名的后缀名。
  返回：文件后缀名，如无返回为空。
  示例： `$(suffix src/foo.c bar.o myfile)` 返回值是“.c .o”。

- 取前缀函数：basename
  语法： `$(basename <names...>)` 。
  功能：从文件names中取出各个文件名的前缀部分。
  返回：文件names的前缀（去年后缀名的所有字符，含目录追）。
  示例： `$(basename src/foo.c bar.o myfile)` 返回值是“src/foo bar myfile”。

- 追加后缀函数：addsuffix
  语法： `$(addsuffix <suffix>,<names...>)` 。
  功能：把后缀suffix加到names中每个变量后面。
  返回：添加了后缀的文件。
  示例： `$(addsuffix .c, src/foo bar myfile)` 返回值是“src/foo.c bar.c myfile.c”。

- 加前缀函数——addprefix 语法： `$(addprefix <prefix>,<names...>)` 。
  功能：把前缀prefix加到names中的每个变量前面。
  返回：加过前缀的文件。
  示例： `$(addprefix src/, foo bar myfile)` 返回值是“src/foo src/bar src/myfile”。

- 循环函数：foreach
  语法：`$(foreach var,list,text)`。
  功能：类型for语句，循环遍历。
  返回：文件names的前缀序列。
  示例：

  ```
  names = a b c d
  files = $(foreach n,$(names),$(n).o)
  ```

  返回值为“a.o b.o c.o d.o”。

- 判断函数：if 语法：`$(if condition,then-part[,else-part])`
  功能：类似if语句，如果符合条件，执行then-part部分，否则执行else-part（该部分可没有）。
  示例：

  ```
  SRC = $(if $(SRC_DIR) $(SRC_DIR),/usr/src)
  ```

  如果定义了SRC_DIR则返回之，否则返回“/usr/src”。

- 调用函数：call 语法：`$(call variable,param,param,…)`
  功能：根据定义的变量（可理解为某种形式的函数）和参数（及位置）来调用。
  示例：

  ```
  reverse = $(2) $(1) # 注意参数位置
  foo = $(call reverse,a,b)
  ```

  返回值foo为“b a”。

5.4.1.5 终极版本

最终示例如下：

```
target=a.out

OBJS=$(patsubst ./%.c, ./%.o, $(SRCS))

$(target):$(OBJS)
    $(CC) $^ -o $@

%.o: %.c
    $(CC) -c $< -o $@
```

可以看到，这个Makefile没有出现具体的源码文件，但只要符合了要求（如包含了main函数的文件，实现文件，头文件），就能编译出最终的可执行二进制a.out文件。

5.4.2 在Makefile中执行shell命令

在Makefile中可以使用shell所提供的任何命令来完成想要的工作。Makefile中的shell命令可以加两种前缀：`@`和`-`，当然，也可以不使用前缀。它们区别如下：

前缀	说明
无	输出执行的命令及其结果，如出错则停止执行
@	只输出命令执行的结果，如出错则停止执行

-	输出执行的命令及其结果，如出错亦继续执行

在应用中，一般命令都使用`@`作为前缀，因为重点是看命令执行的结果。当对Makefile进行调试时，才会关注命令本身和其执行结果。

查看当前目录下所有的.cpp文件和.c文件，并赋值给SRCS变量：

```
SRCS := $(shell find $(SRC_DIRS) -name '*.cpp' -or -name '*.c')
```

删除当前目录（及子目录）下所有.o文件和.d文件：

```
@find . -iname '*.o' -o -iname '*.d' | xargs rm -f
```

5.4.3 使用make编译Makefile

Makefile由make命令进行读取、解析、执行。make的参数有很多, 常用的有：

参数	说明
-f FILE, --file=FILE	指定Makefile文件，如不指定，默认为当前目录的Makefile文件
-j --jobs	同时运行的命令的个数, 也就是多线程执行Makefile
-r --no-builtin-rules	禁止使用任何隐含规则
-R --no-builtin-variabes	禁止使用任何作用于变量上的隐含规则
-B --always-make	假设所有目标都有更新, 即强制重编译

在Makefile中使用的变量，也可以在执行make时将参数传递到Makefile中。如下命令指定了交叉编译器前缀，并指定为非调试版本：

```
$ make CROSS_COMPILE=arm-unknown-linux-gnueabihf- debug=n
```

make传递的参数会覆盖Makefile的变量。如此一来，Makefile的通常性将会增大，即同一份Makefile，传递不同参数达到不同目标，而无须修改Makefile文件。

待写：Makefile要手动更新main所在文件？ 依赖文件

5.4.4 Makefile模板实例

利用隐含规则，结合条件判断，内置函数和shell命令，再加上make传递参数，我们可以写出比较通用的实用Makefile模板了[注3]。

Makefile模板可分几大部分：

- 1、自定义变量的指定或赋值，包括编译器和链接器及其标志，如CC、CXX、CFLAGS、LDFLAGS等等。
- 2、指定头文件路径INCDIRS、源码路径SRC_DIRS。此部分要结合项目源码目录结构。我们在编码规范章节中涉及。
  - 手动指定各个目录。
  - 只指定项目根目录。
  - 使用find命令查找目录。
- 3、指定源码文件SRCS。
  - 使用内置wildcard函数匹配。

    - 使用find命令查找匹配源码后缀。
- 4、指定目标文件OBJS（.o文件），使用内置patsubst匹配。
- 5、确定编译规则。
    - .o文件：根据源码不同，使用CC或CXX编译。
    - 二进制：使用CC或CXX生成。如是C、C++混合编程，则使用CXX。
    - 动态库：使用CXX生成。
    - 静态库：使用AR归档。
- 6、清除项目。删除中间生成文件、依赖文件，最终目标文件等等。

5.4.4.1 应用程序模板

```
# (C) Copyleft 2011 2012 2013 2014 2015 2016 2017 2018
# Late Lee(li@latelee.org) from http://www.latelee.org
#
# A simple Makefile for *ONE* project(c or/and cpp file) in *ONE* or *MORE* directory
#
# note:
# you can put head file(s) in 'include' directory, so it looks
# a little neat.
#
# usage:
#  $ make
#  $ make V=1  # verbose ouput
#  $ make CROSS_COMPILE=arm-arago-linux-gnueabi-  # cross compile for ARM, etc.
#  $ make debug=y # debug
#
# log
# 2013-05-14 sth about debug...
# 2016-02-29 sth for c/c++ multi diretory
# 2017-04-17 -s for .a/.so if no debug
# 2017-05-05 Add V for verbose ouput
################################################################################

# !!!=== cross compile...
CROSS_COMPILE ?=

MKDIR_P ?= mkdir -p

CC  = $(CROSS_COMPILE)gcc
CXX = $(CROSS_COMPILE)g++
AR  = $(CROSS_COMPILE)ar

# !!!===
# in case all .c/.cpp need g++, specify CC=CXX...
# CC = $(CXX)

ARFLAGS = -cr
RM  = -rm -rf
MAKE= make

# !!!===
# target executable file or .a or .so
target = a.out

# !!!===
# compile flags
CFLAGS += -Wall -MMD

# !!!=== pkg-config here
#CFLAGS += $(shell pkg-config --cflags --libs glib-2.0 gattlib)
#LDFLAGS += $(shell pkg-config --cflags --libs glib-2.0 gattlib)

#*******************************************************************************
# debug can be set to y to include debugging info, or n otherwise
```

```
54  debug  = y
55
56  #*********************************************************************
57
58  ifeq ($(debug), y)
59   CFLAGS += -ggdb -rdynamic
60  else
61   CFLAGS += -O2 -s
62  endif
63
64  # !!!===
65  # Macros define here
66  DEFS += -DJIMKENT
67
68  # !!! compile flags define here...
69  CFLAGS  += $(DEFS)
70  # !!! c++ flags
71  CXXFLAGS = $(CFLAGS)
72  # !!! library here...
73  LIBS +=
74  # !!! gcc/g++ link flags here
75  LDFLAGS += $(LIBS) -lpthread -lrt
76
77  # !!!===
78  # include head file directory here
79  INC = ./ ./inc
80  # or try this
81  #INC := $(shell find $(INC) -type d)
82
83  # !!!===
84  # build directory
85  BUILD_DIR ?= #./build/
86
87  # !!!===
88  # source file(s), including ALL c file(s) or cpp file(s)
89  # just need the directory.
90  SRC_DIRS = .
91  # or try this
92  #SRC_DIRS = . ../outbox
93
94  # !!!===
95  # gcc/g++ compile flags
96  CFLAGS += $(INCDIRS)
97  CXXFLAGS += -std=c++11
98
99  # dynamic library build flags
100 DYNC_FLAGS += -fpic -shared
101
102 # include directory(s)
103 INCDIRS := $(addprefix -I, $(INC))
104
105 # source file(s)
106 SRCS := $(shell find $(SRC_DIRS) -maxdepth 1 -name '*.cpp' -or -name '*.c')
107 # or try this
108 #SRCS := $(shell find $(SRC_DIRS) -name '*.cpp' -or -name '*.c')
109
110 OBJS = $(patsubst %.c,$(BUILD_DIR)%.o, $(patsubst %.cpp,$(BUILD_DIR)%.o, $(SRCS)))
111
112 # depend files(.d)
113 DEPS := $(OBJS:.o=.d)
114
115 ifeq ($(V),1)
116 Q=
117 NQ=true
118 else
119 Q=@
120 NQ=echo
121 endif
122
```

```
123  ###############################################################################
124
125  all: $(BUILD_DIR)$(target)
126
127  $(BUILD_DIR)$(target): $(OBJS)
128
129  ifeq ($(suffix $(target)), .so)
130      @$(NQ) "Generating dynamic lib file..." $(notdir $(target))
131      $(Q)$(CXX) $(CXXFLAGS) $^ -o $(target) $(LDFLAGS) $(DYNC_FLAGS)
132  else ifeq ($(suffix $(target)), .a)
133      @$(NQ) "Generating static lib file..." $(notdir $(target))
134      $(Q)$(AR) $(ARFLAGS) -o $(target) $^
135  else
136      @$(NQ) "Generating executable file..." $(notdir $(target))
137      $(Q)$(CXX) $(CXXFLAGS) $^ -o $(target) $(LDFLAGS)
138  endif
139
140  # make all .c or .cpp
141  $(BUILD_DIR)%.o: %.c
142      @$(MKDIR_P) $(dir $@)
143      @$(NQ) "Compiling: " $(basename $(notdir $@)).c
144      $(Q)$(CC) $(CFLAGS) -c $< -o $@
145
146  $(BUILD_DIR)%.o: %.cpp
147      @$(MKDIR_P) $(dir $@)
148      @$(NQ) "Compiling: " $(basename $(notdir $@)).cpp
149      $(Q)$(CXX) $(CXXFLAGS) -c $< -o $@
150
151  clean:
152      @$(NQ) "Cleaning..."
153      $(Q)$(RM) $(OBJS) $(target) $(DEPS)
154  # delete build directory if needed
155  ifneq ($(BUILD_DIR),)
156      $(Q)$(RM) $(BUILD_DIR)
157  endif
158  # use 'grep -v soapC.o' to skip the file
159      @find . -iname '*.o' -o -iname '*.bak' -o -iname '*.d' | xargs rm -f
160
161  .PHONY: all clean
162
163  -include $(DEPS)
```

该模板适用于单目录多文件的项目，能编译C、C++源码，无须修改即可使用。主要内容说明如下：

- L24指定了交叉编译器前缀，如非交叉编译，保留为空即可。
- L28~L38为编译器、链接器等的定义。
- L42指定可执行二进制文件名称，默认为a.out，可修改为其它名称。
- L46的CFLAGS，指定编译选项，编译选项需要根据实际情况增删。
- L53~L62为调试、非调试版本的选择和判断。
- L73~L75指定静态库和动态库，此处只使用最基本的线程库（-lrt）和运行时库（-lrt）。根据实际情况增删。
- L79~L81指定头文件路径，可手动分别指定目录，也可通过find命令将所有的目录作为头文件路径。
- L90指定源码路径，默认为当前目录，在L105~L108通过find的参数来查找源码。可用-name过滤源码类型，用-maxdepth指定目录深度。
- L96指定了C编译器选项，L97指定C++编译器选项。
- L110通过通配符来获取目标文件（.o文件）。
- L113指定了依赖文件，这样当某一文件被修改时，所以依赖该文件的文件都会被编译。(待写：不修改main文件会如何？)
- L127~L138为目标文件的编译规则，此处通过目标文件后缀名自适应了静态库、动态库和可执行二进制文件三种类型。
- L140~L149为C、C++源码的编译规则。
- L151~L160为清除项目规则[注4]。

5.4.4.2 驱动模板

```
1    ###################################################################
2    # (C) Copyleft 2010 2011 2013 2015 2016 2017 2018
3    # by Late Lee<li@latelee.org> from www.latelee.org
4    #
5    # A simple Makefile for linux driver
6    #
7    # make CROSS_COMPILE=arm-linux-
8    # usage:
9    #  $ make
10   #  $ make V=1  # verbose ouput
11   #  $ make CROSS_COMPILE=arm-linux-  # cross compile for ARM, etc.
12   #  $ make debug=y # debug
13   ###################################################################
14
15   CLEAN_BEGIN=@echo "Cleaning up ..."
16   CLEAN_END=@echo "[Done.]"
17
18   MAKE_BEGIN=@echo "Compiling ..."
19   MAKE_DONE="  [Job done!]";
20   MAKE_ERR="  [Oops!Error occurred]";
21   ### nothing end
22
23   # !!!=== for cross compile
24   CROSS_COMPILE = arm-linux-
25
26   CC := $(CROSS_COMPILE)gcc
27   LD := $(CROSS_COMPILE)ld
28
29
30   # !!!=== for debug
31   #DEBUG = y
32
33   ifeq ($(DEBUG), y)
34       DEBFLAGS = -O -g
35   else
36       DEBFLAGS = -O1
37   endif
38
39   # CFLAGS += $(DEBFLAGS) -I$(LDDINCDIR)
40
41   # !!!=== module name here
42   MODULE = GotoHell
43
44   SRC = $(wildcard *.c *.cpp)
45   OBJ = $(patsubst %.c,%.o, $(patsubst %.cpp,%.o, $(SRC)))
46
47   ifeq ($(V),1)
48   Q=
49   NQ=true
50   else
51   Q=
52   NQ=echo
53   endif
54
55   # obj-m = module
56   # obj-y = into kernel
57   # foo.o -> foo.ko
58   ifneq ($(KERNELRELEASE), )
59       obj-m := $(MODULE).o
60
61   # !!!=== your obj file(s) here
62       $(MODULE)-objs := come.o on.o
63
64   else
65    # !!!=== change to your linux kernel path
66       #KERNELDIR ?= /lib/modules/$(shell uname -r)/build
```

```
67        KERNELDIR ?= /home/latelee/linux_ebook/linux
68        PWD := $(shell pwd)
69
70
71    all:
72        $(MAKE_BEGIN)
73        @echo
74        @if \
75        $(MAKE) -C $(KERNELDIR) M=$(PWD) modules;\
76        then echo $(MAKE_DONE)\
77        else \
78        echo  $(MAKE_ERR)\
79        exit 1; \
80        fi
81    endif
82
83    clean:
84        $(CLEAN_BEGIN)
85        rm -rf *.cmd *.o *.ko *.mod.c *.symvers *.order *.markers \
86        .tmp_versions .*.cmd *~
87        $(CLEAN_END)
88
89    install:
90        @echo " Note:"
91        @echo "To install or not install,that is a question."
92
93    modules:
94        @echo "Do not need to do this."
95
96    modules_install:
97        @echo "Do not need to do this."
98    love:
99        @echo "To make or not to make, that is also a question."
100
101   .PHONY:all clean install love modules modules_install
```

该模板适用自定义内核驱动项目，即脱离内核源码目录开发驱动，由于需要内核头文件及其它实现函数，因此，必须手动指定内核源码目录。主要内容说明如下：

- L17~L21为提示信息定义。
- L24指定了交叉编译器前缀，不同交叉编译器使用前缀区分。如非交叉编译，保留为空即可。
- L31~L37为调试、正式版本的选择和判断。
- L42指定了驱动模块名称。
- L62指定了驱动源码的目标文件（*.o文件）。
- L67指定了内核源码目录。注意，该内核必须与目标板使用的一致。

> 注1. Makefile语法请参考官网说明：http://www.gnu.org/software/make/manual/make.html。 ↩

> 注2. 有一些网络页面，因为tab被解析为一个空格，为美化版面，将自动转换为4个空格，这点需要注意。 ↩

> 注3. 此处的Makefile模板依然为初级的，对多目录复杂的大型项目来说，还无法适用。 ↩

> 注4. 如果目标文件（.o文件）无须修改，但又必须编译且编译较耗时，可以使用 `grep -v XXX.o` 过滤掉不删除，典型的有ONVIF协议文件 `soapC.o` 。 ↩

5.5 本章小结

本章介绍了Linux系统编译源码的一般方法，以及交叉编译器的安装。作为编译源码的实践，同时介绍了如何从头构建一个属于自己的交叉编译器。最好，学习了Makefile的知识。

第6章 程序的编译、链接和运行

本章节内容有一定抽象，甚至可能会认为无用。首先，xxxxxxxxxxx，所以日常我们不去主动关注。 知其然，也知其所以然。

在这里写概述

探究helloworld——程序编译

预编译

编译

汇编

链接

Linux二进制工具(Binutils)

ar

nm

strings

strip

objcopy

objdump

readelf

size

ld

目标文件格式

目标文件格式分类

ELF格式

符号修饰

静态库

动态库

探究**helloworld**——程序运行

进程空间

栈

堆

Linux装载**ELF**文件过程

探究**helloworld**——库到内核

glibc

Linux系统调用

硬件输出

拓展思考

本章小结

6.1 探究helloworld——程序编译

涉及的工具，说明一下命令选项。

编程语言有编译型、解释型语言，前者必须将源码使用编译器进行编译方可生成可执行文件，后者则， 考虑到嵌入式资源的限制以及效率问题，大部分应用软件使用C/C++实现。但如树莓派（Raspberry Pi）则使用Python语言。

在Windows系统中，我们有大量编程IDE（集成开发环境，Integrated Development Environment）可选择，如Visual Studio（微软出品）、NetBeans（免费开源的Java开发环境）、Eclipse（免费开源，支持多种语言）、IntelliJ IDEA（Jet Brains，主要针对Java开发）、Code:: Block（支持定制且跨平台的IDE），等等。这些IDE使用方便，建立工程后，只需要点击几个按钮或使用快捷键即可完成编译工作，并且支持可视化调试。

但是，Linux系统可用的IDE却少得可怜，可能是Linux文化的传统，很多软件源码都使用Makefile或automake来管理编译，在前面章节中，我们也看到一些实例，这些管理机制要花一定时间学习，从而加大了门槛，但这是必由之路，针对日常工作开发，所需要掌握的东西并不多。了解编译的过程，了解可执行二进制文件是如何产生的，对我们编程有很大作用。

gcc编译选项

先说明gcc编译有很多个步骤，会用到很多命令，但一般只用gcc来操作即可，将“编译”的适用范围扩大，即包含了预编译、编译、链接等步骤。再介绍编译选项。下面使用最简单的例程进行演示，不搞复杂的。

介绍

此处写gcc编译的选项？优化的O1 O2 O3？概览，再细化每一个阶段？

警告（几个而言） 库包含 -Wl,--start-group $^ -Wl,--end-group -v 编译细节。 -I -L. -ltestlib -L. -static -ltestlib（同时有动态静态库，指定静态） -D 定义宏 --prefix 指定前缀 和交叉编译有关的--host、CC等

参考：https://www.cnblogs.com/fnlingnzb-learner/p/8119854.html

预编译

查看宏定义

编译

汇编

链接

6.2 Linux二进制工具(Binutils)

在了解程序编译之后，我们再了几个二进制工具，这些工具用途不同，综合起来，能让我们更加了解二进制文件的面目。

https://blog.csdn.net/whatday/article/details/88542258

ar

nm

strings

strip

objcopy

objdump

readelf

size

ld

6.3 目标文件格式

目标文件格式分类

ELF格式

符号修饰

要不要涉及弱符号?

静态库

动态库

要不要涉及so找不到？

6.4 探究helloworld——程序运行

进程空间

栈

堆

Linux装载ELF文件过程

ld内容？？

6.5 探究helloworld——库到内核

本节应该放到后面的章节的，因为了解系统底层的知识之后再阅读会更加理解，但为保持章节连续性，所以提前了。

glibc

Linux系统调用

硬件输出

6.6 本章小结

第7章 应用程序开发

编程规范

编码风格

编程原则

C标准库

C++标准库

Linux应用层模块分类

文件IO编程

多进程编程

多线程编程

信号处理编程

串口编程

本章小结

7.1 编程规范

编码风格

适当空格

编程原则

7.2 C标准库

7.3 C++标准库

7.4 Linux应用层模块分类

文件IO编程

多进程编程

多线程编程

信号处理编程

串口编程

7.5 本章小结

第8章 调试方法

在嵌入式Linux中，限于各种条件，调试往往是十分困难的。 不能使用gdb单步调试，网络阻塞，可以得不到预期效果。。。 是一个大工程，不能只依靠工具，规范化，等。 调试经验？

内核打印？

8.1 关于“调试”的说明

调试阶段

“调试”所指的范围比较模糊，当编码完成后，就会进入调试阶段（也有称测试阶段），当调试（测试）无问题后，就可以正式使用。调试的结果是要确保功能是否正常，调试的过程不一定会遇到问题。狭义上讲，调试是解决已发现的问题的过程，即已经出现问题了，再寻找方法解决问题。

出现问题再解决，属于事后补救，所谓防患于未然，如果在事前做一些防范措施，以减少出现问题的机率，是不是更好呢？（要删除）~~但是，在调试（测试）之前将低级错误解决掉比调试（测试）时发现好，在调试时解决问题比上线或移交客户后发现问题好。~~

代码质量

调试有时比编码更耗时，利用工具减少低级错误，比如提高代码质量，消除代码编译警告，降低可能带来的隐患，这些工作，可以节省大量时间。

借用工具

在不得已情况下，只能借助工具来排查问题。如日志打印（最原始的方式），gdb，kgdb，coredump，等。需要注意的是，有时候使用调试工具单步执行不一定完全等同于实际的运行环境，如单步调试socket收发的程序，如果停留片刻，socoket数据可能已经失效了。

从容应付

写程序不可能没有bug，只是bug易不易被发现，bug的危害大不大。调试是程序员的一项基本能力，随着项目的参加、见识的增长，只要用心留意并做总结，相信调试的能力会越来越好。调试的经验要靠平时积累，别人无法帮忙。但可以阅读别人的经验，吸取别人的经验，从而提升自己的能力。

本章先从代码层面规避风险，再介绍具体的调试手段，希望读者能建立规范化的概念。

8.2 消除gcc编译警告

8.2.1 gcc编译警告选项

gcc编译器自带很多编译警告选项[注1]，开启这些选项可以输出编译警告消息，然后分析、修正，在代码层面上消除潜在风险。一般地使用-Wall 或-Wextra即可满足大部分场合。下面分别介绍警告选项。

-w
禁止编译警告的打印。这个警告不建议使用。

-Wall
开启“所有”的警告。强烈建议加上，并推荐该选项成为共识。如case语句没有default处理，有符号、无符号处理，未使用变量(特别是函数内有大量未使用的数组，会占用栈空间)，用 `%d` 来打印地址或用 `%s` 打印int值，等，都会发出警告。-Wall包括：

```
-Waddress
-Warray-bounds=1 (only with -O2)
-Wc++11-compat  -Wc++14-compat
-Wchar-subscripts
-Wenum-compare (in C/ObjC; this is on by default in C++)
-Wimplicit-int (C and Objective-C only)
-Wimplicit-function-declaration (C and Objective-C only)
-Wbool-compare
-Wduplicated-cond
-Wcomment
-Wformat
-Wmain (only for C/ObjC and unless -ffreestanding)
-Wmaybe-uninitialized
-Wmissing-braces (only for C/ObjC)
-Wnonnull
-Wopenmp-simd
```

```
-Wparentheses
-Wpointer-sign
-Wreorder
-Wreturn-type
-Wsequence-point
-Wsign-compare (only in C++)
-Wstrict-aliasing
-Wstrict-overflow=1
-Wswitch
-Wtautological-compare
-Wtrigraphs
-Wuninitialized
-Wunknown-pragmas
-Wunused-function
-Wunused-label
-Wunused-value
-Wunused-variable
-Wvolatile-register-var
```

-Wextra
除-Wall外，还有一些其它的警告选项，是使用-Wextra打开的，它包括：

```
-Wclobbered
-Wempty-body
-Wignored-qualifiers
-Wmissing-field-initializers
-Wmissing-parameter-type (C only)
-Wold-style-declaration (C only)
-Woverride-init
-Wsign-compare
-Wtype-limits
-Wuninitialized
-Wshift-negative-value
-Wunused-parameter (only with -Wunused or -Wall)
-Wunused-but-set-parameter (only with -Wunused or -Wall)
```

需要指出的是，不是所有的警告选项都是必要的，各位读者需要根据个人或团队习惯而制定。举个例子，`-Wfatal-errors` 不一定适合所有项目，如果编译警告实在太多，使用此选项反而消耗时间，还不如一次性将所有警告都输出，再逐个修正。

-Werror
将所有的警告当成错误处理。此选项谨慎使用。有的开源库警告很多(大名鼎鼎的ffmpeg也有很多警告呢)，一一改掉耗时耗人力，必要性也不大。

-Wfatal-errors
遇到第一个错误就停止，减少查找错误时间。此选项谨慎使用。

-Wchar-subscripts
使用char类型变量作为数组下标(因为char可能是有符号数)。

-Wcomment
注释使用不规范。如 `/* */` 注释中还包括 `/*` 。

-Wmissing-braces
括号不匹配。在多维数组的初始化或赋值中经常出现。

-Wparentheses
括号不匹配，在运算符操作或if分支语句中，可能会出现此警告。这类bug隐藏得较深，建议显式地加上括号。

-Wsequence-point
如出现i=i++这类代码，则报警告。-Wall默认有该警告。

-Wswitch-defaultcase
没有default时，报警告。

-Wunused-but-set-parameter
设置了但未使用的参数警告。

-Wunused-but-set-variable
设置了但未使用的变量警告。

-Wunused-function
声明但未使用函数。

-Wunused-label
未使用的标签，比如用goto会使用label，但在删除goto语句时，忘了删除label。

-Wunused-variable
未使用的变量。

-Wmaybe-uninitialized
变量可能没有被初始化。特别是在有if语句或switch语句中，最好在声明变量时加上初始化。

-Wfloat-equal
对浮点数使用等号，这是不安全的。

-Wreturn-type
函数有返回值，但函数体个别地方没有返回值(特别是有if判断，可能忘记在else添加返回值)。

-Wpointer-sign
指针有符号和无符号的错误传参。如函数使用unsigned char，但传入*char*指针。

-Wsign-compare
有符号和无符号比较。

-Wconversion-null -Wsizeof-pointer-memaccess
在sizeof中经常出现，误用指针求大小。

-Wreorder
C++出现，构造函数中成员变量初始化与声明的顺序不一致。

-Woverflow
范围溢出。

-Wshadow
局部变量覆盖参数、全局变量，会报警告。

8.2.2 gcc常见编译警告

下面列出一些常用的警告信息及示例。[注2]

未使用变量或函数

警告信息

```
warning: unused variable ‘ret’ [-Wunused-variable]
  int ret;
```

原因及解决
将不使用的变量删除即可。其它还有类似的：未使用的参数（包括赋值和函数参数）、未使用的函数、未使用的标签，等等。

变量未初始化

警告信息

```
warning: 'mode' may be used uninitialized in this function
```

示例代码

```
// 片段1
int mode;
if (idx > 1)
{
    mode = 1;
}
int set = mode;

// 片段2
{
int x;
switch (y)
  {
  case 1: x = 1;
    break;
  case 2: x = 4;
    break;
  case 3: x = 5;
  }
foo (x);
}
```

原因及解决
片段1中，mode在声明时没有初始化，但后面有条件地被赋值，但不满足所有情况，编译器报警告。示例中假如idx小于1，mode就不会被赋值，所以set的值就是未知的了。片段2中，当y不是1、2、3时，x没有被明确的赋值，其值是不确定的。
没有被正确赋值的变量危害比较大，并且不容易发现。

非空函数无返回值

警告信息

```
warning: control reaches end of non-void function [-Wreturn-type]
或
warning: no return statement in function returning non-void [-Wreturn-type]
```

示例代码

```
// 片段1
int foo()
{
  if(a==1)
  {
    return ok;
  }
  // no return here
}
```

```
// 片段2
int open(const char* file)
{
  // no return here
}
```

原因及解决
有返回值的函数，有的分支没有返回，如片段1。或者没有返回值，如片段2。加上返回值即可，

空字符串

警告信息

```
warning: zero-length gnu_printf format string [-Wformat-zero-length]
         sprintf(buffer, "");
```

原因及解决
`""` 为空字符串，如果要初始化buffer，则在声明时初始化，或使用memset函数初始化。

常量字符串

警告信息

```
warning: deprecated conversion from string constant to ‘CHAR* {aka char*}’ [-Wwrite-strings]
```

原因及解决
声明字符串时，在前面加const。

格式化：类型不一致

警告信息

```
warning: format ‘%s’ expects argument of type ‘char *’, but argument 2 has type ‘int’ [-Wformat=]
warning: format ‘%d’ expects argument of type ‘int’, but argument 3 has type ‘short int *’ [-Wformat=]
warning: format ‘%d’ expects argument of type ‘int *’, but argument 3 has type ‘short int *’ [-Wformat=]
```

示例代码

```
int mode = 0;
short int* idx = 1;
printf("bbb: %s %d\n", mode, idx);

short int uword = 0;
sscanf("test: 250", "test: %d", &uword);
```

原因及解决
这类警告常出现在输入、输出格式函数中（如scanf、printf系列函数）。不同类型的格式化符号不同，示例代码中，混用了这些符号。如下是类型与格式化符号对应关系：

```
short: %hd
unsigned short: %hu
int: %u
unsigned int: %u
long long: %lld
unsigned long long: %llu
```

```
字符串: %s
```

对于函数参数类型的检查，如果是自实现类似printf的函数，那么，要使用“ `__attribute__((format(printf,N,M)))` ”的形式进行修饰。其中N表示第几个参数是格式化字符串，M指明从第几个参数开始做检查。否则，gcc无法在编译阶段检测类型不一致问题。

格式化：参数过多

警告信息

```
warning: too many arguments for format [-Wformat-extra-args]
```

示例代码

```
printf("bbb: %d\n", mode, idx, set);
```

原因及解决
格式化打印符号，比实际传递的参数少，即参数过多，补充打印符号即可。类似的有参数过少。

有符号和无符号混用

警告信息

```
warning: pointer targets in passing argument 2 of ‘ll_foobar’ differ in signedness [-Wpointer-sign]
expected ‘unsigned char *’ but argument is of type ‘char *’

warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
```

示例代码

```
// 片段1 函数参数与实参不一致
void foo(char *p)
{

}
unsigned char* p = "hello";
foo(p);

// 片段2 有符号和无符号比较
int d = 0;
unsigned int e = 1;
if (d == e)
{
  // do sth
}
```

原因及解决
片段1：有2个相似警告，字符串默认是“ `char *` ”类型，故语句“ `unsigned char* p = "hello";` ”会出现警告。传参时亦有警告，因为foo函数参数为有符号，而传参为无符号。其中一方强制转换类型即可。
片段2：如果要比较有符号数和无符号数，要将其中一方转换再进行比较。注意，无符号数没有负数。

括号与优先级

警告信息

```
warning: suggest parentheses around arithmetic in operand of '|' [-Wparentheses]
```

示例代码

```
    int reg = 1;
    reg = reg&(~(1<<4))|(1<<4);
```

原因及解决
各种运算有其优先级，如同时使用多个运算符号，建议添加括号以明确它们之间的关系。正确示例如下：

```
reg = (reg&(~(1<<4))) | (1<<4);
```

类似的例子还有：

```
    // 多个函数调用并列，其中关系无法直观知道
    return id == foo::Get() || foo::Get() != bar::Get() && bar::Go();

    // && 与 || 要明确
    int e = a && b || c ^ d;

    // else就近原则，是if (b)的另一分支，但不是if (a)的—哪怕写法上似乎如此
    {
        a = 0;
        if (a)
            if (b)
                foo();
        else
            bar();
    }
```

初始化不完整

警告信息

```
warning: missing braces around initializer [-Wmissing-braces]
warning: (near initialization for 'a[0]') [-Wmissing-braces]
```

示例代码

```
int a[2][2] = { 0, 1, 2, 3 };
int b[2][2] = { { 0, 1 }, { 2, 3 } }; // OK
```

原因及解决
二维（或多维）数组，初始化时显式加上括号。如示例代码数组b。

switch省略部分处理

警告信息

```
warning: enumeration value ‘LL_BAR’ not handled in switch [-Wswitch]
```

示例代码

```
    enum foobar {LL_FOO, LL_BAR};
```

```
    enum foobar e = LL_FOO;
    switch (e)
    {
    case LL_FOO:
        break;

    // default: // 不使用default出现警告
    //     break;
    }
```

原因及解决
枚举类型没有完全被处理，或者没有default语句，会出现此警告，根据需要将所有类型都进行判断处理，或者添加default进行默认处理。

浮点数比较

警告信息

```
warning: comparing floating point with == or != is unsafe [-Wfloat-equal]
```

示例代码

```
float d = 2.0;
int i = 2;
if (d == i) {}
```

原因及解决
浮点数的比较，不能直接使用 `==` 。此警告需要额外加上 `-Wfloat-equal` 选项。

未使用的值计算，语句无效

警告信息

```
warning: value computed is not used [-Wunused-value]
warning: statement has no effect [-Wunused-value]
```

示例代码

```
// 片段1
    char* p = 1;
    *p++;

// 片段2
#if 0
#define LL_DEBUG printf
#else
#define LL_DEBUG
#endif
```

原因及解决
值未使用或语句执行没有效果。修正后代码如下：

```
// 片段1
    char* p = 1;
    q = *p++; // 将*p++的结果返回给另一变量

// 片段2
```

```
#if 0
#define LL_DEBUG(fmt, ...) printf(fmt, ##__VA_ARGS__)
#else
#define LL_DEBUG(fmt, ...)
#endif
```

变量覆盖

警告信息

```
warning: declaration of 'g_var' shadows a global declaration [-Wshadow]
warning: shadowed declaration is here [-Wshadow]
```

示例代码

```
int g_var;
void test4()
{
    double g_var;
}
```

原因及解决
局部变量g_var遮盖了全局变量g_var。良好的编程习惯可避免该问题。此警告需要额外添加 `-Wshadow` 选项。

值范围判断误用

警告信息

```
warning: comparison is always true due to limited range of data type [-Wtype-limits]
```

示例代码

```
int test5()
{
    unsigned char c;
    if (c < 0xff)
        return 0;
    else
        return -1;
}
```

原因及解决
c为char类型，其值永远小于0xff，故if(c < 0xff)始终为true，无法进入else分支。数值比较时，应注意其范围。

获取大小

警告信息

```
warning: argument to ‘sizeof’ in ‘char* strncpy(char*, const char*, size_t)’ call is the same expression as the
 destination; did you mean to provide an explicit length? [-Wsizeof-pointer-memaccess]
```

示例代码

```
// 片段1
char buffer[64];
char buffer2[64];
```

```
char* ptr = buffer2;
strncpy(ptr, buffer, sizeof(ptr));

// 片段2
memset(this, 0, sizeof(this));
```

原因及解决
ptr或this为指针，用sizeof得到的是指针的长度（32位系统为4，64位系统为8），此时指针长度不一定为缓冲区或类的的大小（如果缓冲区恰好为4无异常，但只是幸运而已）。故一定要使用正确的长度值。

新版本带来的警告

警告信息

```
error: macro "__DATE__" might prevent reproducible builds [-Werror=date-time]
error: macro "__TIME__" might prevent reproducible builds [-Werror=date-time]
```

原因及解决
gcc4.9及以上的版本，添加了 `-Werror=date-time` 警告选项，所以不能在代码中使用 `__DATE__` ， `__TIME__` ，否则会出现错误（gcc说法是会导致编译的不确定性），如果一定要用，则在编译时添加 `-Wno-error=date-time` ，但依然会有警告打印。

类构造函数初始化顺序不一致

警告信息

```
warning: CFoobar::m_nInit will be initialized after [-Wreorder]
  int m_nInit;
      ^
warning:   ‘int CFoobar::m_nTime’ [-Wreorder]
  int m_nTime’;
```

原因及解决
在初始化时，按变量声明的顺序排列即可。

其它

警告信息

```
warning: backslash and newline separated by space
```

原因及解决
宏后面使用“\”来连接多行，但“\”后面多了空格，删除空格即可。

警告信息

```
warning: "/*" within comment [-Wcomment]
```

原因及解决
注释中还有注释符号，即 `/***/` 中还有 `/*` 。删除 `/*` 即可。

警告信息

```
error: function declaration isn’t a prototype
```

原因及解决
函数参数不能为空，如函数没有参数，要加上void。

警告信息

```
warning: integer overflow in expression [-Woverflow]
```

原因及解决
整数溢出，定义的整数超出其范围了，如`#define BIG_SIZE 800*1024*1024*1024`。减少值即可。

警告信息

```
warning: passing NULL to non-pointer argument 2 of ‘void* memset(void*, int, size_t)’ [-Wconversion-null]
  memset(foobar, NULL, sizeof(foobar));
```

原因及解决
C++中，NULL是指针，不是数值，而memset函数要求的是数值0，故传递0，或将NULL强转为整数0。

> 注1. 参见gcc官方文档http://gcc.gnu.org/onlinedocs/gcc/Warning-Options.html ↩

> 注2. 使用gcc 4.6.0或5.4.0测试，部分示例根据需要，手动添加警告选项，读者可结合本章节内容验证，同时总结出自己的编译选项。 ↩

8.3 C/C++代码静态检查

pclint 有些编译器无法检测的警告，示例说明。

访问非法内存？ format示例

多级指针连续调用？

结合上一小节和本小节，我们知道可以通过消除编译警告以及检查代码来规避潜在的风险，

8.4 使用printf打印跟踪流程

printf大法有时十分有用，对于跟踪代码流程、函数调用，帮助很大。在出现问题时，我们使用printf来打印变量地址、变量值，对于出现的bug，会有一些帮助。当然，使用printf本身也要注意，我就曾经遇到过形如 `printf("%s", int_value);` 这类形式的代码，即用%s来打印一个整型变量，这个结果一般都是非法指针。在嵌入式上，非法指针多半会死机——现在强大的芯片应该少出现死机了。这些情况，就要十分注意函数参考的使用了。

打印等级定义 include/linux/kern_levels.h 0 KERN_EMERG 1 KERN_ALERT 2 KERN_CRIT 3 KERN_ERR 4 KERN_WARNNING 5 KERN_NOTICE 6 KERN_INFO 7 KERN_DEBUG

2.4.2 控制台打印等级修改 方法1：echo 8 > /proc/sys/kernel/printk 方法2：dmesg –n 8

打印往往是最常用的调试技巧。

调试内核和驱动都可以采用printk。在Kernel.h (include\linux)中定义了log的等级。

未指定日志级别的 printk() 采用的默认级别是 DEFAULT_MESSAGE_LOGLEVEL，这个宏在kernel/printk.c 中被定义为整数 4，即对应KERN_WARNING。在 /proc/sys/kernel/printk 会显示4个数值（可由 echo 修改），分别表示当前控制台日志级别、未明确指定日志级别的默认消息日志级别、最小（最高）允许设置的控制台日志级别、引导时默认的日志级别。当 printk() 中的消息日志级别小于当前控制台日志级别时，printk 的信息（要有\n符）就会在控制台上显示。但无论当前控制台日志级别是何值，通过 /proc/kmsg （或使用dmesg）总能查看。另外如果配置好并运行了 syslogd 或 klogd，没有在控制台上显示的 printk 的信息也会追加到 /var/log/messages.log 中。

define KERN_EMERG "" / system is unusable /

define KERN_ALERT "" / action must be taken immediately /

define KERN_CRIT "" / critical conditions /

define KERN_ERR "" / error conditions /

define KERN_WARNING "" / warning conditions /

define KERN_NOTICE "" / normal but significant condition /

define KERN_INFO "" / informational /

define KERN_DEBUG "" /* debug-level messages

2、打印的东西在哪输出？

uboot中设置了命令行参数：

set bootargs console=ttySAC0,115200 root=/dev/mtdblock3
正是console环境决定了printk的输出地方。

3、如何利用printk调试？

在程序中合适的地方加入printk（）函数，当无法正常输出的时候，认定此处可能存在bug。

printk(KERN*DEBUG"%s %s %d\n", **FILE**, **FUNCTION**, _LINE*);

打印跟踪优点和缺点

自定义打印日志系统

8.5 gdb调试

8.6 利用coredump文件调试

coredump常常作为事后分析的手段。对于一些不容易出现的bug，则可以使用coredump来调试——比如有些软件在设备运行几天后崩溃。但这种方式有几个前提：一是程序必须是debug版本，这样才能保留调试信息，方便调试。二是必须设置coredump文件大小为unlimited。三是有足够空间存储产生的coredump文件。在生成coredump后，就可以用gdb(或其交叉编译版本)来调试了。

导致coredump的原因有： 非法内存访问，如数组越界，内存指针使用错误。 多线程读写的数据未加锁保护 非法指针 // 空指针、野指针 堆栈溢出 // 函数内使用大的数组、大的局部变量

生成coredump文件前置条件

查看文件格式：cat /proc/sys/kernel/core_pattern 设置文件格式：echo ./core.%e.%p> /proc/sys/kernel/core_pattern Core dump 文件格式： %p 所dump进程的进程ID %u 所dump进程的实际用户ID %g 所dump进程的实际组ID %s 导致本次core dump 的信号 %t core dump的时间 (由1970年1月1日计起的秒数) %h 主机名 %e 程序文件名

段错误调试实例

嵌入式设备上coredump调试经验

第9章 嵌入式Linux移植总览

嵌入式Linux移植

bootloader移植

Linux内核移植

根文件系统移植

宿主机与目标板共享的几种方式

NFS

SSH

FTP

硬件介质：U盘和SD卡

qemu模拟环境

qemu介绍

qemu选项介绍

嵌入式Linux开发实践

嵌入式Linux移植实践经验

嵌入式Linux应用开发实践经验

如何阅读芯片手册

如何阅读开源代码

几种嵌入式Linux平台的开发环境介绍

三星s3c2440(samsung)

德州仪器dm8127(TI)

瑞芯微rk8188(rockchip)

赛灵思zed board(Xilinx)

x86工控机

本章小结

9.1 嵌入式Linux移植

bootloader移植

Linux内核移植

根文件系统移植

9.2 宿主机与目标板共享的几种方式

NFS

SSH

FTP

硬件介质：U盘和SD卡

9.3 qemu模拟环境

qemu介绍

qemu安装

qemu选项介绍

qemu启动实例

9.4 嵌入式Linux开发实践

嵌入式Linux移植实践经验

嵌入式Linux应用开发实践经验

如何阅读芯片手册

如何阅读开源代码

9.5 几种嵌入式Linux平台的开发环境介绍

如何补充呢？

三星s3c2440(samsung)

德州仪器dm8127(TI)

瑞芯微rk8188(rockchip)

赛灵思zed board(Xilinx)

x86工控机

9.6 本章小结

第10章 bootloader移植

u-boot

u-boot概述

u-boot目录说明

u-boot编译

u-boot在qemu环境的启动

u-boot启动流程

在u-boot中新加命令

u-boot进程空间

coreboot

coreboot概述

coreboot目录说明

coreboot编译

coreboot在qemu环境的启动

coreboot启动u-boot

coreboot启动流程

本章小结

10.1 u-boot

u-boot概述

u-boot目录说明

u-boot编译

u-boot在qemu环境的启动

u-boot启动流程

在u-boot中新加命令

u-boot进程空间

10.2 coreboot

coreboot概述

coreboot目录说明

coreboot编译

coreboot在qemu环境的启动

coreboot启动u-boot

coreboot启动流程

10.3 本章小结

第11章 Linux内核移植

本章介绍Linux内核的移植知识，包括。。。。。。 如何下载内核？

内核目录说明

内核配置编译步骤

内核配置(menuconfig)选项说明

常规设置

CPU设置

网络协议设置

驱动设置

添加自定义驱动

helloworld设备驱动

将驱动编译进内核

将驱动编译为ko文件：独立目录

将驱动编译为**ko**文件：使用内核目录

驱动的加载、卸载过程

内核启动过程

内核在**qemu**环境的启动

过程分析

拓展思考

本章小结

要不要添加一些小经验？如何入到驱动的入口函数？如何确定使用哪个驱动？

11.1 内核目录说明

下载

主页为https://www.kernel.org/。点击主页https://www.kernel.org/pub/链接进入。。。

linux->kernel，在页面选择所需的内核版本即可[注1]。本书采用的是4.15.9版本，位于https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/目录

HTTP

内核Makefile与Kconfig文件

> 注1. 截至写稿过程中，v5.x版本已经成了主流，而v6.x版本即将面世。 ↩

11.2 内核配置

(menuconfig)选项说明

常规设置

CPU设置

网络协议设置

驱动设置

在不在这里介绍Makefile和Kconfig？添加自定义驱动

11.3 内核配置编译

包括内核编译常见错误

11.4 helloworld设备驱动

驱动实例

将驱动编译进内核

将驱动编译为**ko**文件：独立目录

将驱动编译为**ko**文件：使用内核目录

驱动的加载、卸载过程

11.5 内核启动过程

内核在qemu环境的启动

过程分析

11.6 本章小结

第12章 根文件系统移植

要说明有很多构建的工具，同时说明为什么用busybox构建 要不要画一个图？左侧busybox得到的二进制，右侧为配置文件、脚本，下面是制作镜像，再下面是烧写？ 要不要介绍一些小经验？如可执行属性x，

busybox

busybox介绍

busybox编译

构建Linux根文件系统

构建/bin目录

构建/etc目录

构建/lib目录

其它构建方法

根文件系统启动流程

系统级别启动过程

用户级别启动过程

制作ramdisk镜像文件

在qemu挂载rootfs

拓展思考

本章小结

12.1 busybox

busybox介绍

busybox编译

12.2 构建Linux根文件系统

构建/bin目录

说明哪些busybox没有的，要单独编译。

构建/lib目录

有些库要从交叉编译器中获取，有些要自己编译。注意版本一致性

构建/etc目录

启动脚本

其它目录

其它目录也有脚本，或程序

12.3 其它构建方法

本章节介绍其它的构建方式，这些方式实现手法不同，但最终目的是一样的。现在很多嵌入式系统较复杂，单纯靠手动创建、编写脚本已不太现象，借助工具可大大减轻工作量。

12.3.1 Buildroot构建

Buildroot是在Linux系统中构建完整的嵌入式Linux系统的框架。构建方式由Makefile和Kconfig组成，与Linux内核编译过程类似。这里的“完整”，是指通过Buildroot，可以编译生成bootloader、kernel、rootfs以及rootfs所需要的各种库、配置文件、启动脚本，等等。
Buildroot官方网站：https://buildroot.org/。
Buildroot配置选项说明文档：https://buildroot.org/downloads/manual/manual.html。
Buildroot源码下载地址：https://buildroot.org/download.html，

12.3.1.1 下载

这里使用的版本为buildroot-2019.02.7。下载文件名称为buildroot-2019.02.7.tar.bz2。

12.3.1.2 环境准备

宿主机所依赖的编译工具和库，参考本书第二章节。如果有额外的工具，根据实际提示安装。
这里列出笔者遇到的额外的依赖库：

```
$ sudo apt-get install hgsubversion
$ sudo apt install whois # mkpasswd
```

解压Buildroot源码包：

```
$ sudo tar jxf buildroot-2019.02.7.tar.bz2
$ pwd
/home/latelee/linux_ebook/buildroot-2019.02.7
```

此处假定Buildroot目录为 `/home/latelee/linux_ebook/buildroot-2019.02.7` 。
交叉编译器所在目录为 `$HOME/x-tools/arm-unknown-linux-gnueabihf/bin` 。
全路径：/home/latelee/x-tools/arm-unknown-linux-gnueabihf/ configs目录有许多不同板子的默认配置文件，我们选定 `qemu_arm_vexpress_defconfig` 进行实验。其它

12.3.1.3 配置

cp configs/qemu_arm_vexpress_defconfig .config make menuconfig

12.3.1.4 编译

```
make
```

附 目录

```
.
├── arch
├── board
├── boot
├── CHANGES
├── Config.in
├── Config.in.legacy
├── configs
├── COPYING
├── DEVELOPERS
├── docs
├── fs
├── linux
├── Makefile
├── Makefile.legacy
├── package
├── README
├── support
├── system
├── toolchain
└── utils
```

配置选项说明

12.4 根文件系统启动流程

先介绍流程，然后在后面验证

系统级别启动过程

用户级别启动过程

12.5 制作ramdisk镜像文件

12.6 在qemu挂载rootfs

要不要画一个流程图？包括各个目录如何得到（构建），启动流程如何走，创建镜像，下载，挂载

12.7 本章小结

第13章 Linux驱动开发

说明先从实例入手，感性认识。再对一些经典驱动进行分析。

驱动分类概述

驱动实例

platform设备驱动实例

异步通信驱动实例

用户空间与内核空间之间的交互方式

sys文件系统

proc文件系统

自定义驱动

驱动分析

platform设备驱动分析

字符设备驱动驱动分析

看门狗驱动分析

实时时钟驱动分析

串口驱动分析

Linux驱动常见错误

Linux驱动学习建议

本章小结

13.1 驱动分类概述

13.2 驱动实例

platform设备驱动实例

异步通信驱动实例

还要添加其它的

13.3 用户空间与内核空间的交互方式

sys文件系统

proc文件系统

自定义驱动

13.4 驱动分析

platform设备驱动分析

字符设备驱动驱动分析

看门狗驱动分析

实时时钟驱动分析

串口驱动分析

是否添加其它驱动的分析？ 如LED，或者socket？

13.5 Linux驱动常见错误

本节介绍什么内容？ 内核崩溃？如何找bug？ 标题改为经验好些？

13.6 Linux驱动学习建议

这节要不要？

13.7 本章小结

本章节列出《嵌入式Linux入门与实践》参考文献。

网络资源

ubuntu发行版本下载：http://old-releases.ubuntu.com/releases/
crosstool-ng：http://crosstool-ng.github.io/
GNU大本营：http://www.gnu.org/
LFS：http://www.linuxfromscratch.org
uboot：http://www.denx.de/wiki/U-Boot/
coreboot：https://www.coreboot.org/
Linux内核官网：https://www.kernel.org/
busybox：https://busybox.net/
c/c++参考手册：http://www.cplusplus.com/

书籍

《Linux C 编程一站式学习》
《程序员自我修养——链接、装载与库》
《嵌入式linux应用开发完全手册》 韦东山

《loader and linker》
《The Linux Programmer's Toolbox》
《Linux and the Unix Philosophy》，中文版本《Linux/Unix设计思想》
《Linux设备驱动开发详解》
《Linux设备驱动程序》第三版

源码

ftp://ftp.denx.de/pub/u-boot/u-boot-2018.03.tar.bz2
https://busybox.net/downloads/busybox-1.28.1.tar.bz2
https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/linux-4.15.tar.xz

一直不敢奢望自己会写书，怕能力不够。工作多年，对于一些技术，自认为有点经验、心得，偶然机会，出版社责任编辑找到我，联系出书事宜。这是个难得的机会，虽然会很辛苦，但值得一试。本文记录写作的时间表。

本书的章节是由易到难，由简到繁，按学习的流程编写，但却不是按章节顺序来写的，因为笔者先制定了大纲，再细分章节，对于某些知识点，因为内容较多，会先写一部分，再写其它章节，再回头继续编写。另外，起初确定的大纲可能有部分逻辑不通，行文时会再进行调整。

备忘

电子书正文：
按书籍风格来编写。

配套实验笔记（随笔形式）

Qemu+arm模拟。

过程技术解决

先确定大纲，用word编写，再转换为gitbook，一度转为hexo，后再转回gitbook，解决gitbook一些问题：如锚点，页面跳转，生成pdf，图片显示等。

时间表

- 2019.9上旬：决定继续编写。
- 2019.4~2019.8：因工作、家庭原因，又中断计划。
- 2019年4月中旬：重构博客之际，决定重启计划，主要解决gitbook的一些问题。
- 此后一年中：因各种大事、小事中断计划。
- 3月中旬、下旬： 安装虚拟机（截图）、qemu、交叉编译器确定及测试、示例代码确认及编写
  编译uboot、内核、在qemu中跑，rootfs后续再做。
  动手写第1章、第2章内容（后面再补充）
- 2018.3.12：选题通过，着手编写。
- 2018.3.2~3.7：报选题
- 2月初~3月初：确定、编写大纲
- 2018.1.31：编辑李博通过微信联系，然后通过邮件交流