Linux 教程

Ubuntu 版

零基础入门系列教程

作者: 魏明择

2025 年版

https://weimingze.com

目录

第一章、Linux系统入门

前言

本课程是 针对零基础学员的 Linux 教程。本课程将从安装Linux 操作系统开始，逐渐深入讲解 Linux 操作系统的使用及开发。

本课程是 Linux 运维和 Linux 开发人员的基础教程。如果你是想把梦想变成现实的人，推荐学习此课程。

本课程会实时更新。本人保证本站的过时的内容及时清理，现存的所有内容都有效且可操作。

为什么要学习 Linux?

1. Linux 在服务器市场占据了 90% 以上的市场份额，腾讯游戏、网易游戏大部分后端都是Linux 服务器。
2. 移动终端市场占有率高。目前的安卓手机、原生 HarmonyOS 5.0 系统手机 的底层都是 Linux 操作系统。
3. 运维和开发人员必须精通 Linux 操作系统。进大厂企业 Linux 是必经之路。

为什么要学习 Ubuntu Linux?

1. 图形化界面、完善的驱动支持，对新手最友好。
2. 软件生态丰富。
3. 企业级支持与稳定性。
4. 70% Linux 开发人员的首选操作系统。

让我们先来了解一下 Linux 操作系统。

Linux 操作系统是仿照 Unix 设计，广泛应用于服务器、嵌入式设备、超级计算机以及个人电脑的多任务操作系统。

它的特点是可裁剪、稳定、安全，并支持多用户的操作系统。

其实我们常说的 Linux 应当是指 Linux 内核。Linux 内核，由 Linus Torvalds 于 1991 年首次发布，后来经过几次比较大的框架行的改动，最近的一次大的框架性的改动是2003年12月发布的 Linux 2.6 的 内核，后续的版本都是在2.6 内核的基础上进行的扩充功能的调整，没有做大的框架性的改动。目前的最新的版本已经是6.14.6。

Linux 创始人 Linus：

Linux 内核是开源的系统，你可以去Linux 内核的官方网站：https://kernel.org/ 下载和编译内核以运行在你自己的硬件系统上。如图

Linux 内核是由 C语言结合汇编语言编写的。他的主要作用是管理CPU、内存、以及外设（蓝牙、网络、键盘鼠标等），同时会对应用软件的启动，进程调度，运行权限、进程间通信等进行管理。他是运行于 **“内核态”** 的程序。是最重要的操作程序。

相比于 Windows 和 MacOS 系统，Linux 的优缺点

优点

1. 开源与免费，无需支付授权费用。
2. 内核可裁剪，很容易运行在资源有限的嵌入式系统中。
3. 效率高，驱动程序等都运行在同一个单体内核中，可以通过共享内存进行通信，无需复制数据，减少次数，运行速度快。
4. 安全性高，用户没有获取 root 权限，对系统的攻击难度极高，保证了系统的稳定性和安全性。
5. 开发容易，各种开发工具，如 gcc、g++、python、docker 等都支持Linux 系统且都免费。

缺点

1. 单体内核。所有的驱动程序，进程调度，内存管理都运行在一个内核进程中，一旦有驱动程序等内核态程序含有破坏性代码，则对整个系统造成安全隐私泄露，甚至宕机。
2. 硬件兼容性差，很多显卡、打印机等都没有对应的驱动程序，用户体验比较差。

Linux 的发行版

现在很多人说的 Linux 操作系统其实是在 Linux 内核的基础上，各个发行版厂商自己开发了桌面系统和应用程序管理系统的总称。如 Ubuntu、Debian、CentOS、Redhat、Fedora等。这些发行版有些是免费的，如： Ubuntu 和 CentOS，还有 Radhat 和 Debian 是收费的。

目前我在工作中了解到，服务器部署常用 CentOS，软件开发常用 Ubuntu。因为 Ubuntu 的生态相对做的比较好些。

本课程将以 Ubuntu 系统为起点，让大家尽快了解 Linux 操作系统。

1. Ubuntu Linux 简介

Ubuntu Linux 是由 Canonical 公司主导开发一个基于 Debian 的开源操作系统。

Ubuntu Linux以用户友好和易用性著称。它是全球最流行的 Linux 发行版之一，广泛应用于个人电脑、服务器和云计算环境。

Ubuntu 的 Logo：

Ubuntu 这个系统的中文名我们一般读作 "乌班图"。

Ubuntu的主要特点

1. 免费且开源：Ubuntu 遵循开源协议，用户可以自由使用、修改和分发。
2. 长期支持： Ubuntu 每两年发布一个 LTS（Long-Term Support） 版本，提供 5 年 的安全更新和维护（如 Ubuntu 22.04 LTS）。非 LTS 版本每 6 个月发布一次，支持期 9 个月，不建议在部署时使用。
3. 用户友好：默认使用 GNOME 桌面环境。提供图形化安装工具和软件中心，适合新手。
4. 强大的软件生态：支持 apt 包管理，提供数万个开源软件。并兼容 snap 等通用包管理。
5. 多平台支持：支持 x86_64（Intel/AMD）、ARM（树莓派、香橙派等硬件）、云服务器（AWS/Azure） 等架构。
6. 安全性高：内置防火墙、定期安全更新，适合企业级应用。

对于Linux 开发者，建议使用 Ubuntu Linux 进行开发。最终在软件发布或部署的时候可以再移植到其他平台如 CentOS 等。这样可以大大提高开发效率。

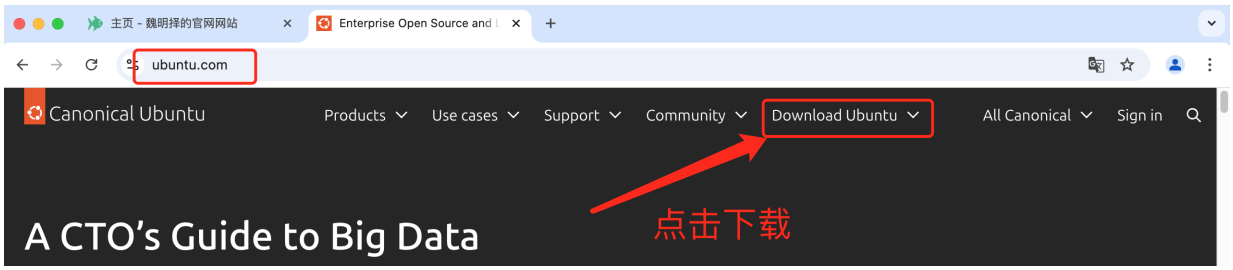
本课程使用最新的 Ubuntu 24.04 LTS(2024年4月发布) 为教学平台，遗弃旧的系统的不用的内容，以便爱好者能走最快的学习路径。避免浪费时间。

Ubuntu Linux 的官网：

- https://ubuntu.com/

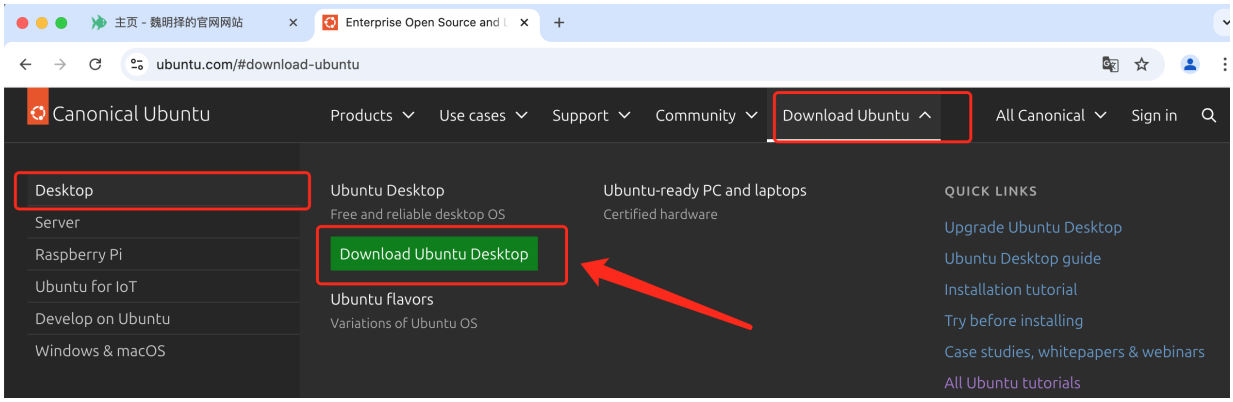
在官网，你可以获取免费的安装光盘镜像的 `ISO` 文件，将此 ISO 文件用 Nero 或 UltraISO 等软件刻录成光盘后就可以在你的电脑上进行安装了。一般我是将此 ISO 文件刻录到 U 盘中，然后用 U 盘进行安装（因为我的电脑没有光盘驱动器）。

ISO文件的下载地址:

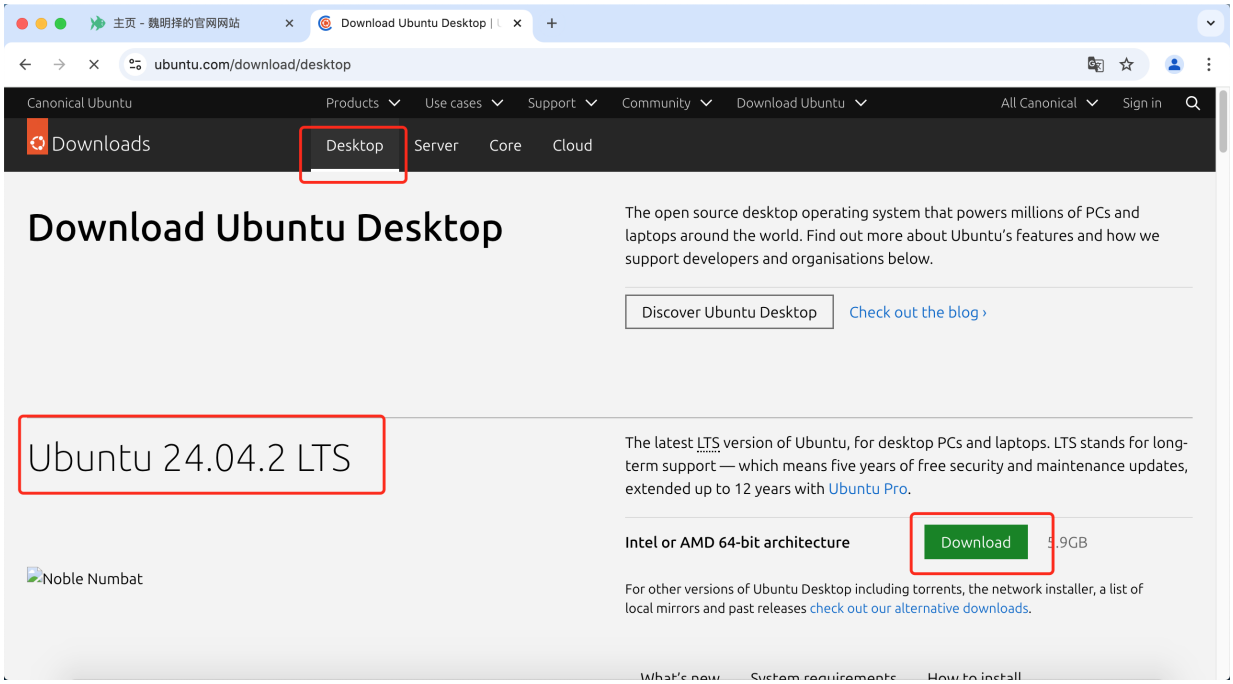


点击 "Download Ubuntu" 后选择 Desktop 菜单内的 "Download Ubuntu Desktop", 就进入下载页面。位置如下:

从左侧的菜单当中，我们能看到有桌面版(Desktop)、服务器版(Server)、树莓派版(Raspberry Pi)、还有物联网版(Ubuntu for IoT)以及 Windows 和 MacOS 版的下载，在这里我们学习开发，所以选择桌面版的下载入口。



下载对应的版本的 下载镜像



点击 "Download" 则提示保存路径，默认的文件名为: `ubuntu-24.04.2-desktop-amd64.iso` 保存即可。等下载完毕后准备安装。

> 注意: 直接刻录成光盘在你的电脑上安装可能会引起你电脑上所有的文件丢失，建议你做好备份工作。

本课程推荐使用虚拟机进行安装，这样可以不破坏你的原有系统。

为什么要安装虚拟机

虚拟机软件可以将你的电脑的硬件资源，比如CPU或者是内存的一部分分离出来形成一个单独的机器，在这个单独的机器上，我们就可以安装其他的操作系统。这个虚拟的操作系统是在原系统上运行，因此不会破坏原系统的内容。

下一节我们将介绍如何使用 VMware 虚拟机来安装 Ubuntu Linux 操作系统。

推荐使用的虚拟机软件

- Windows 系统建议安装 VMware Workstation Pro 16.0 或更高的版本。
- MacOS 系统建议先安装 VMWare Fusion 12.1.0 或更高的版本或 Parallels Desktop 虚拟机 。

因以上软件不是免费软件，因此建议同学们自己去下载安装上述软件。

安装步骤

1. 先装虚拟机软件。
2. 使用虚拟机软件虚拟一台电脑。

3. 在这台虚拟的电脑上安装 Ubuntu Linux 操作系统。

如果你寂静安装了上述虚拟机软件，则可以直接下载 Ubuntu 24.04 的 VMware 虚拟机的镜像，直接使用上述软件打开就可以运行 Ubuntu Linux 了，

Ubuntu24.04 VMware镜像:

- 百度网盘下载地址:
  - https://pan.baidu.com/s/1jYAdvByctakhRVrDIteoLA 提取码: m8ee
- 用法:
  1. 解压缩 `Ubuntu24.04LTS_IntelX86CPU.zip`
  2. 使用 VMware workstation pro v16.0 或 VMware fusion v12.1 及以上版本打开 `Ubuntu24.04LTS_IntelX86CPU/Ubuntu24.04LTS.vmx` 文件并运行。
- 登录信息: 此虚拟机的登录用户名是：weimingze，密码：weimingze，root 用户的密码也是：weimingze。

> 成功登陆以后你就可以创建一个自己的账户了。

如果你要从头自己尝试安装 Ubuntu Linux，详见下一节内容。

2. MacOS 安装 VMware Fusion虚拟机

这节课我们来讲解在 MacOS 操作系统上安装 VMware Fusion 虚拟机软件的详细步骤以及可能遇到的问题。

硬件需求

因虚拟机要使用一部分内存和CPU作为虚拟机器的硬件部分，因此要求安装 VMware 软件的机器硬件内存要大，至少 8G，建议是 16G 或以上。

如果你的电脑是 Windows 操作系统，请略过本节内容。

安装软件和操作的基本步骤

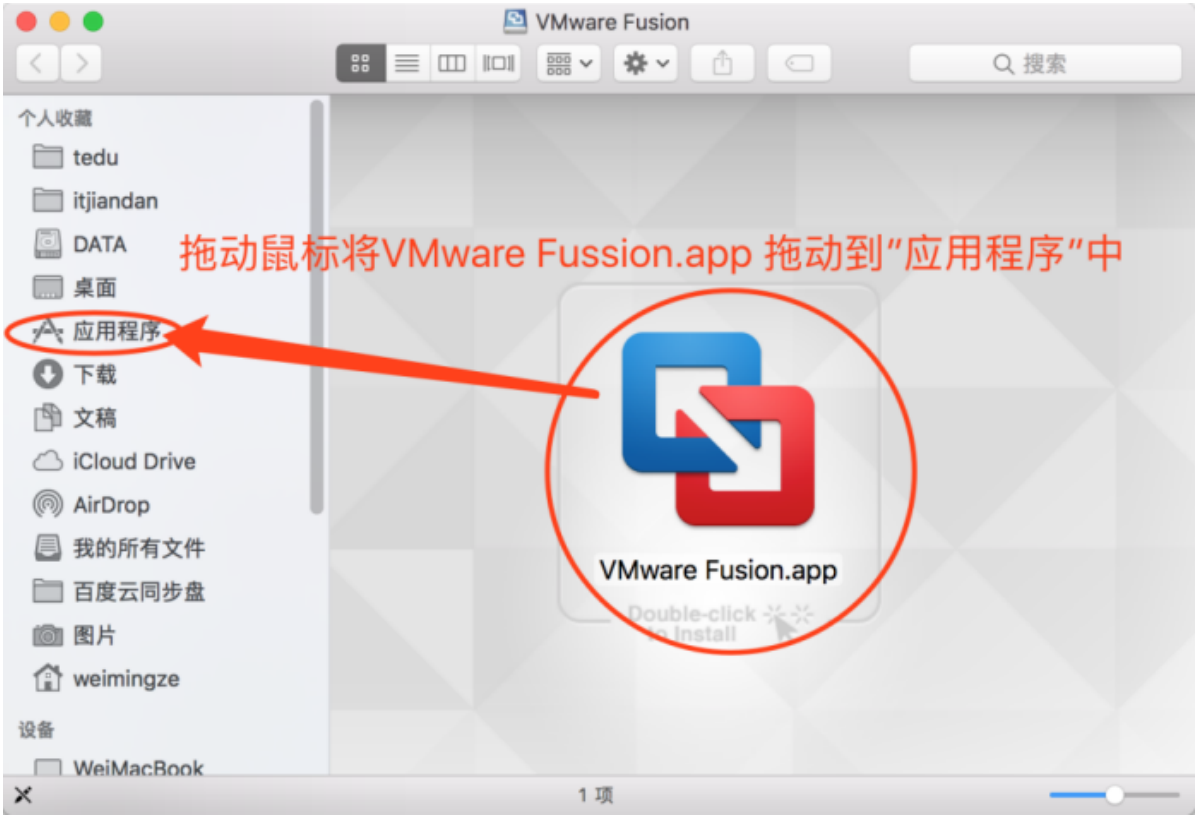
软件准备：你需要下载 VMware Fusion的虚拟机软件，文件名一般为:

`VMware-Fusion-xx.x.x-xxxxxxxx.dmg`

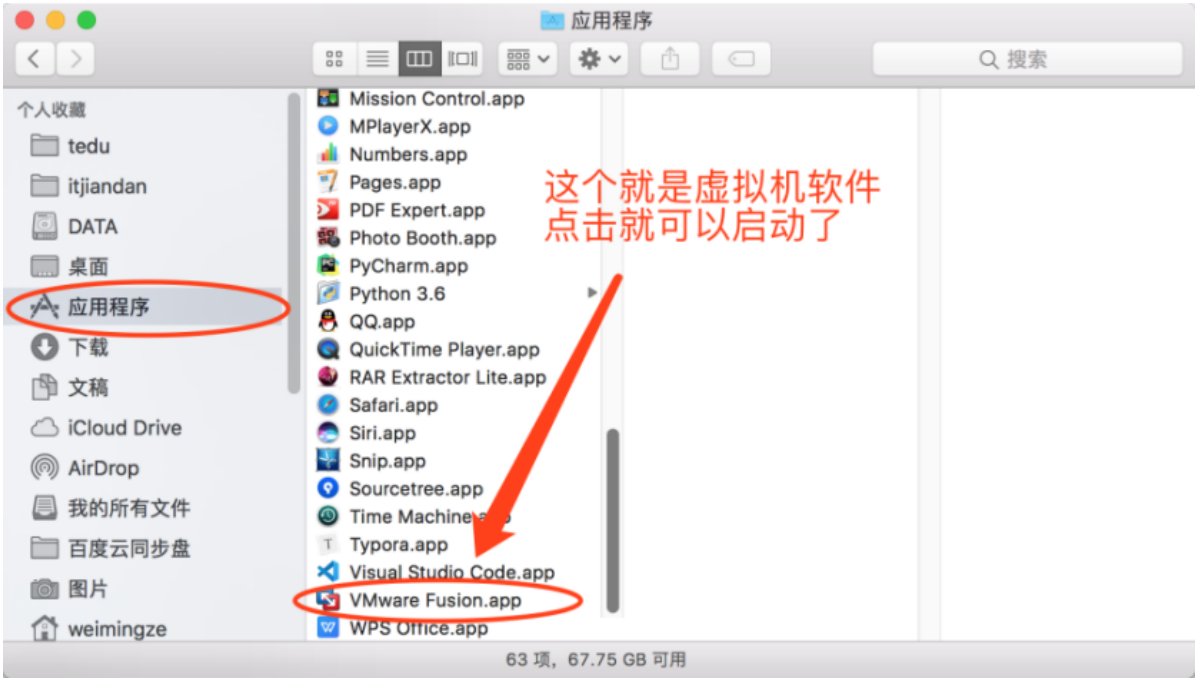
> 注: x 一般为数字

安装步骤：

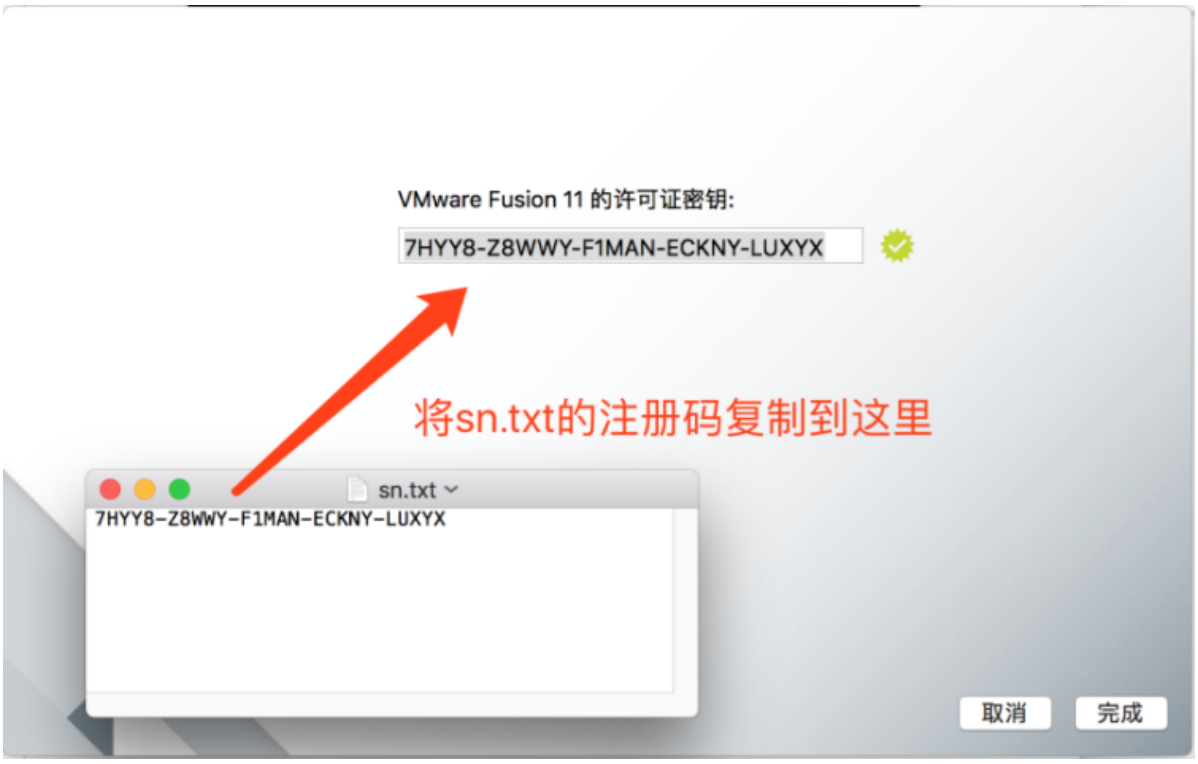
1. 双击下载的`VMware-Fusion-xx.x.x-xxxxxxxx.dmg`文件，进入如下界面，将 VMware Fusion.app 拖动到应用程序（Application）中就完成了安装，如下图：



2. 启动 VMware 就完成注册，先启动 VMware Fusion。

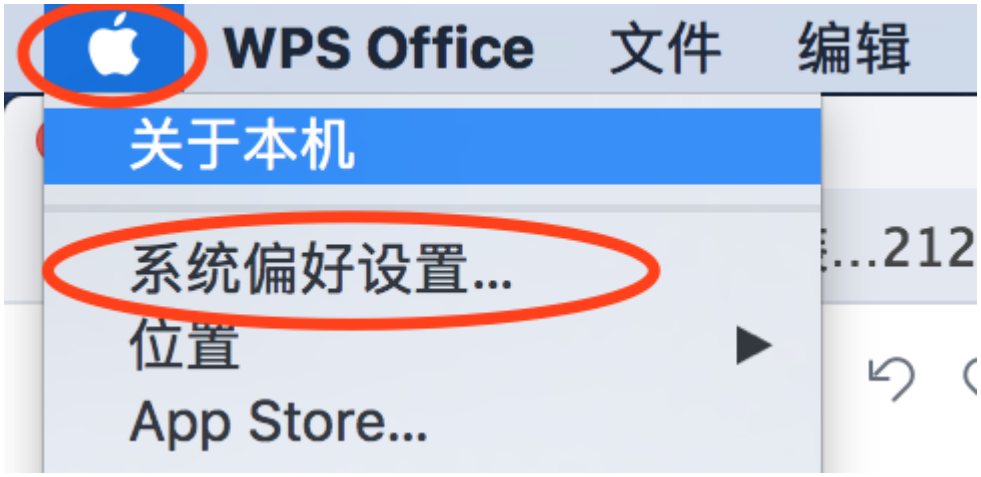


3. 输入注册码完成注册。



打开失败的处理方法

1. 弹出系统扩展被阻止提示，选择左上角苹果图标里的“系统偏好设置”。



2. 进入”安全性与隐私”。

3. 点击 来自开发者"VMware inc" 的系统软件已被阻止载入 右侧的 允许 按钮。

至此，Mac 下 VMware Fusion 安装结束。下一步我们将用 VMware Fusion 创建一台虚拟的电脑，准备安装 Ubuntu Linux!

跳转至Ubuntu Linux安装文档：

3. Windows 安装 VMware Workstation

这节课我们来讲解在 Windows 操作系统上安装 VMware Workstation 虚拟机软件的详细步骤以及可能遇到的问题。

硬件需求

1. 内存要大，至少8G，建议是16G或以上。
2. CPU处理器需要支持虚拟化

如果你是 MacOS 操作系统，请看上一节的内容。

安装软件和操作的基本步骤

软件准备：你需要下载 VMware Fusion 的虚拟机软件，文件名一般为:

`VMware-workstation-full-xx.x.x-xxxxxxxx.exe`

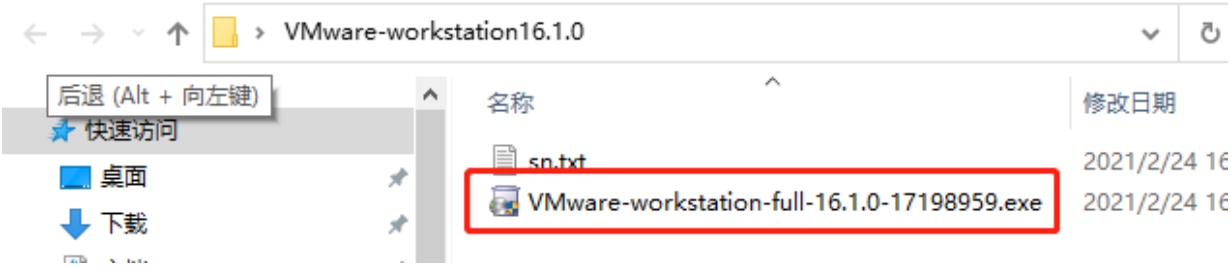
> 注: x 一般为数字

VMware workstation 安装步骤

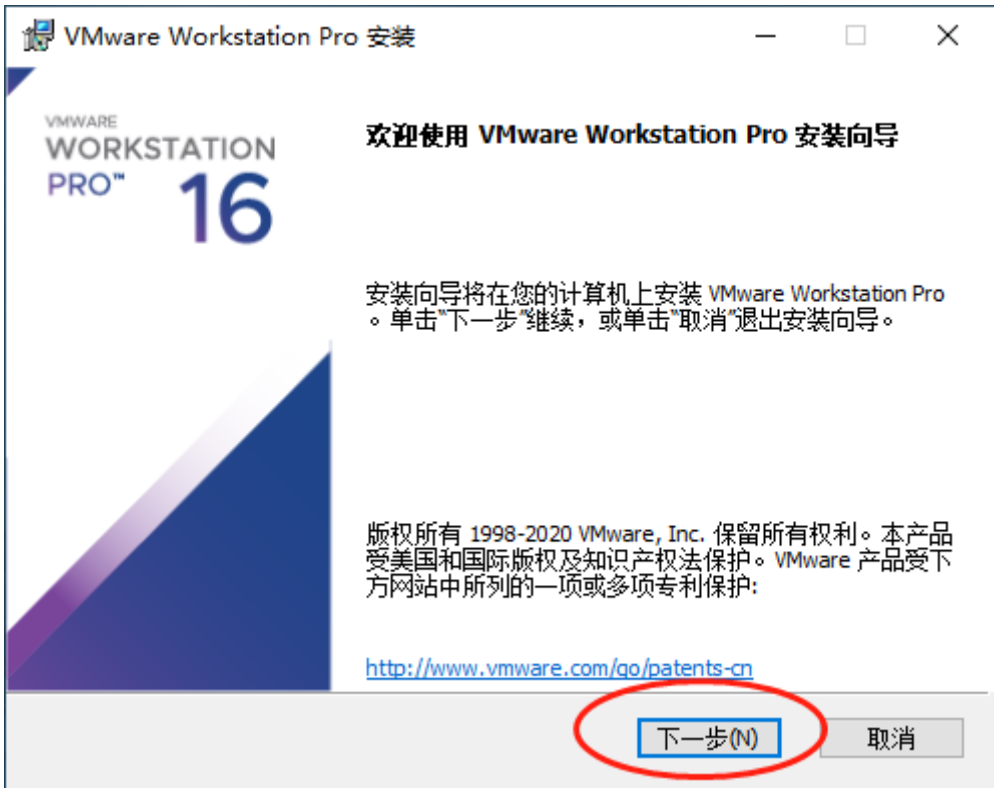
> （以下仅适用于Windows 64 位系统）

如果您已经安装 VMware Workstation 16及更高版本的软件包，可以跳过此步骤。

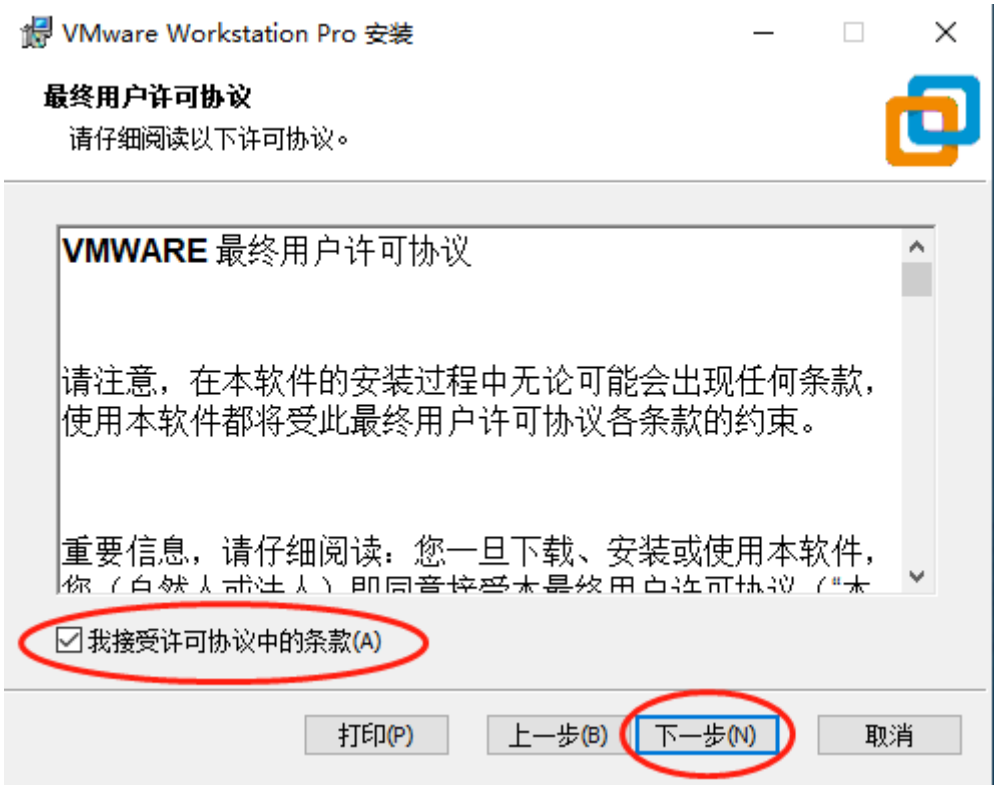
双击下载的 “VMware-workstation-full-xx.x.x-xxxxxxxx.exe”并运行安装向导



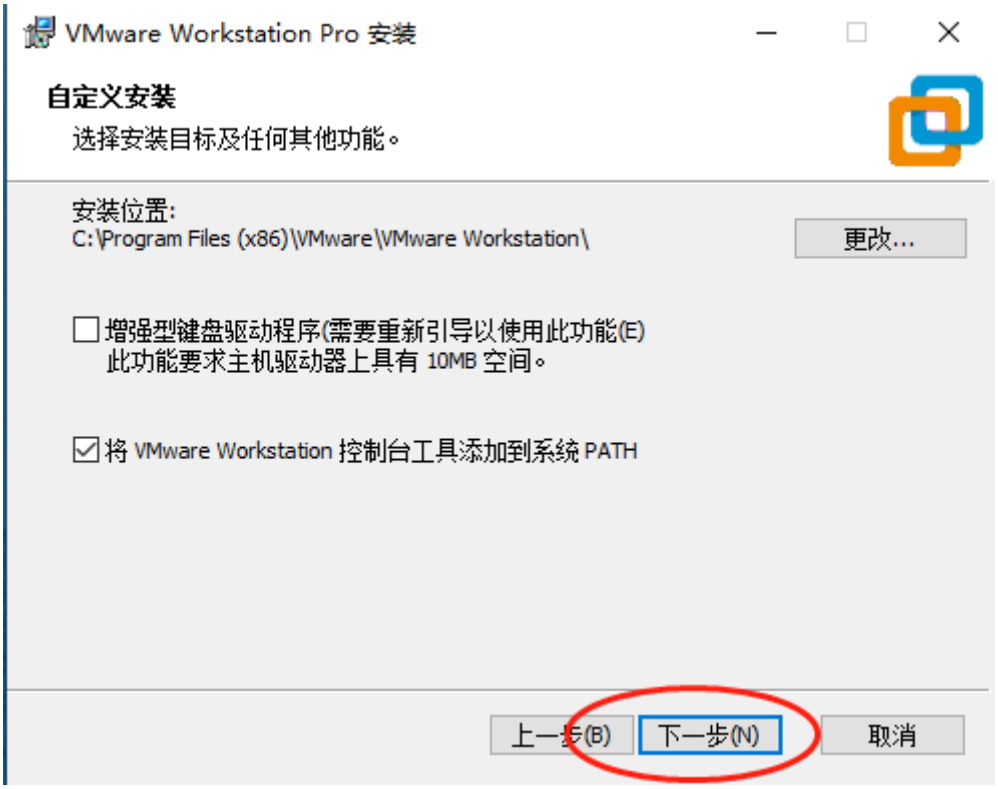
一路点击“下一步”，直至看到“完成”



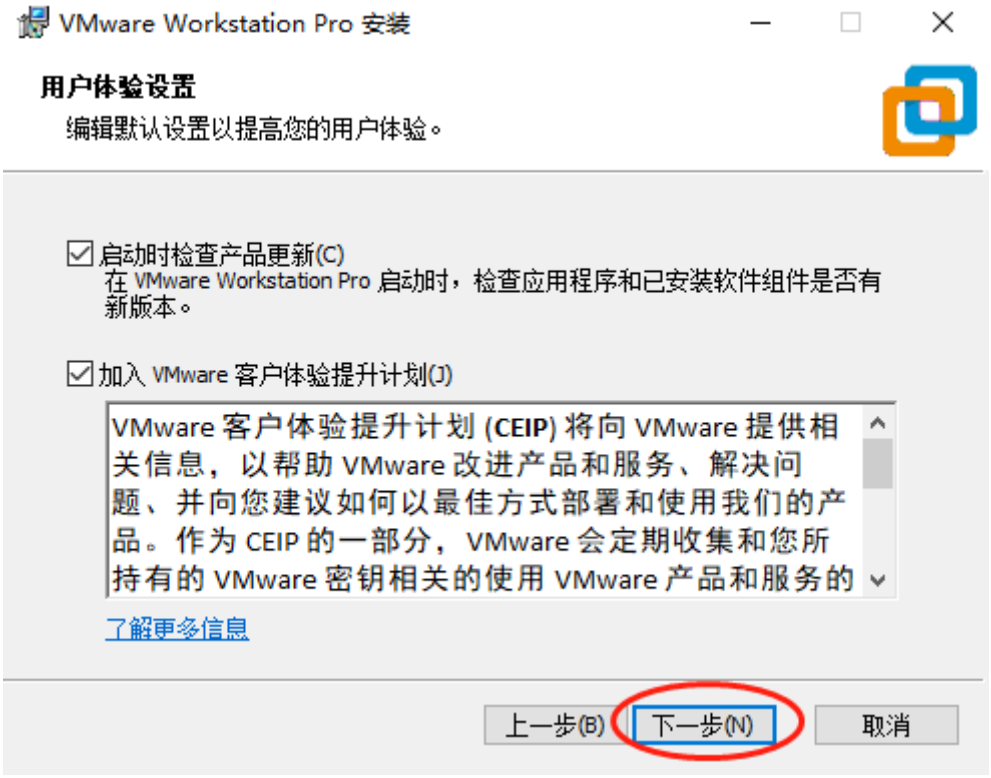
进入用户许可协议签署页面。



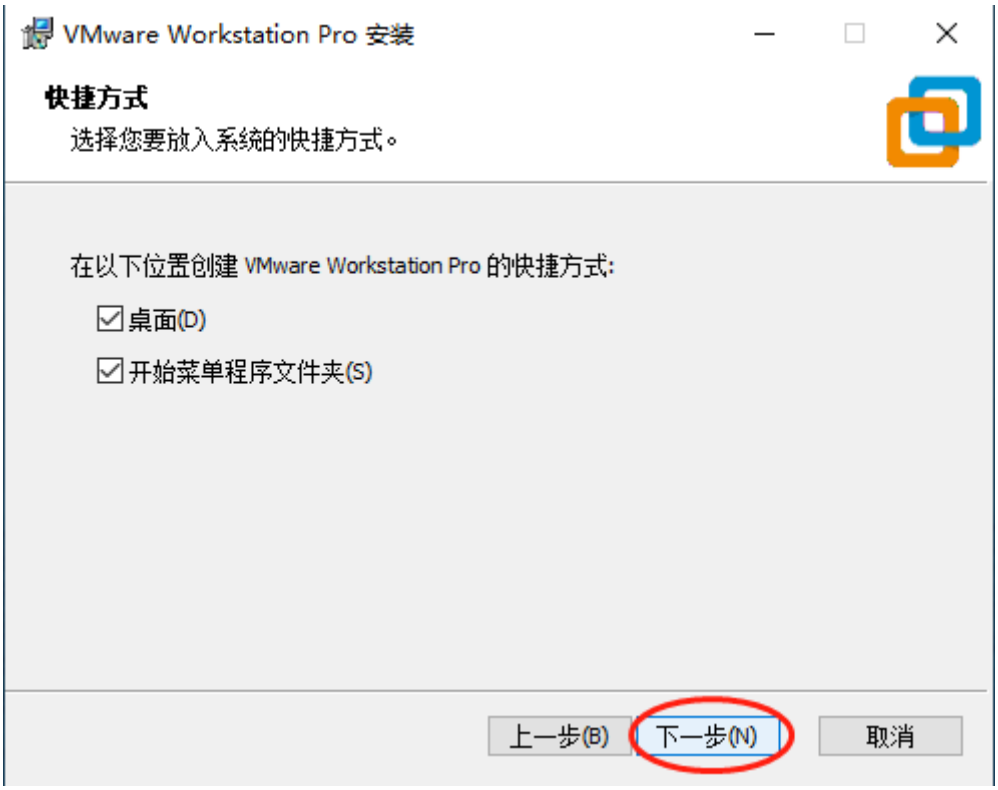
进入自定义安装选项页面。



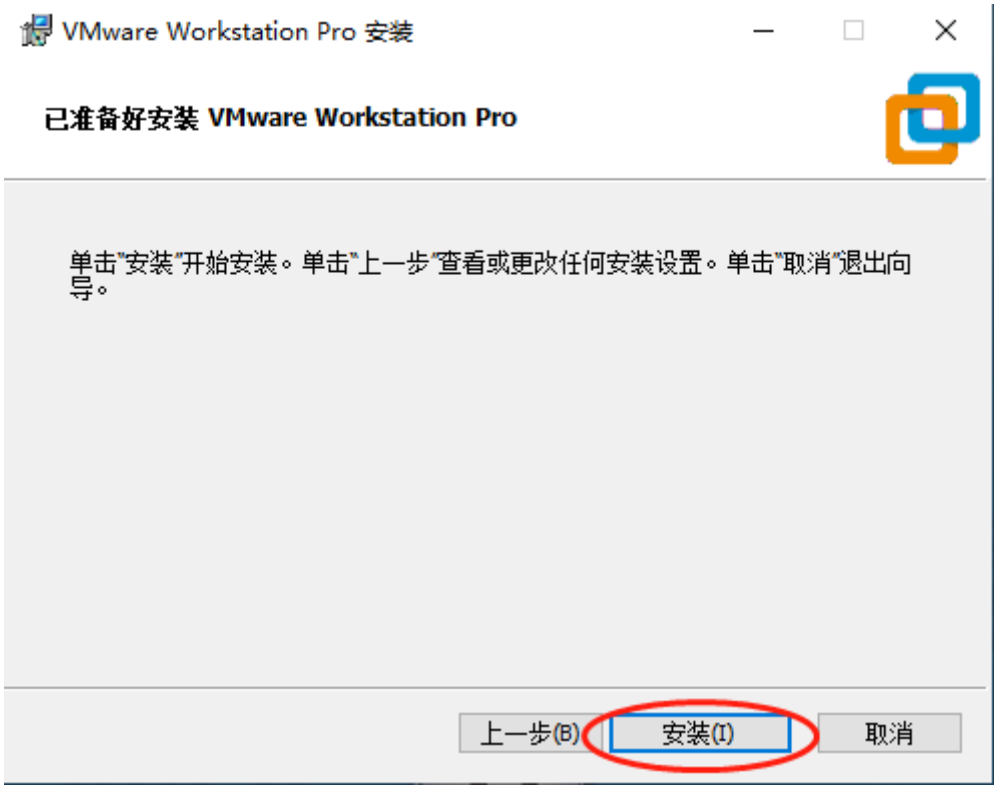
进入用户体验设置选项页面。



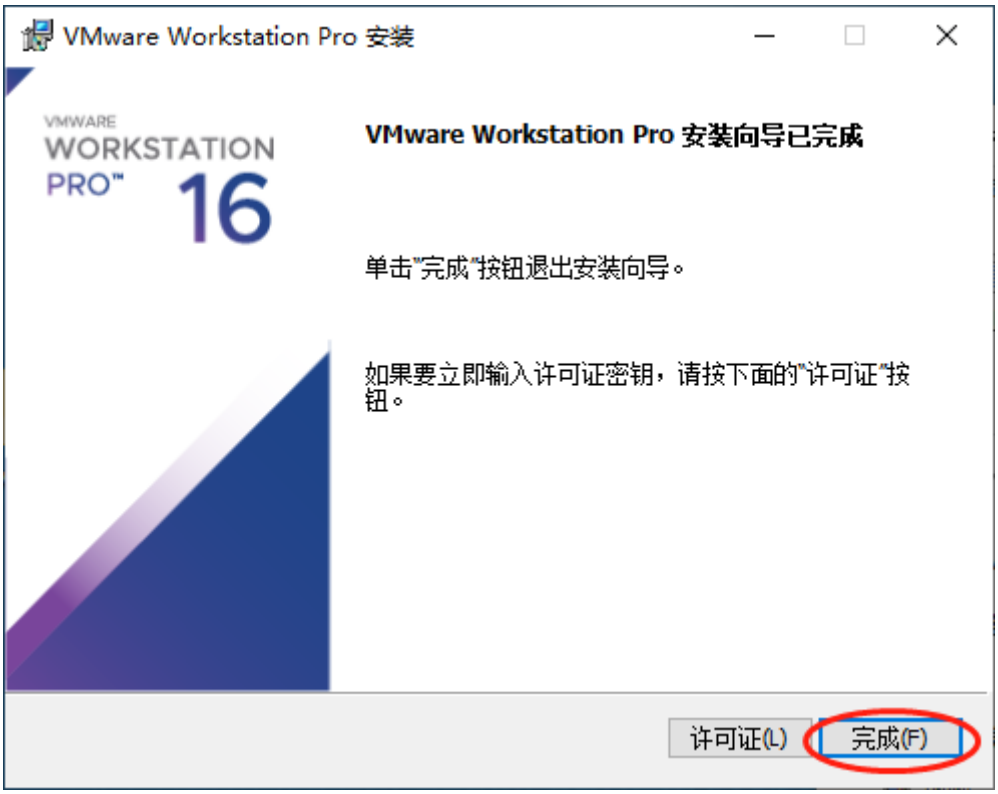
进入创建快捷方式设置选项页面。



进入确认进行安装页面。



确认安装完成页面。

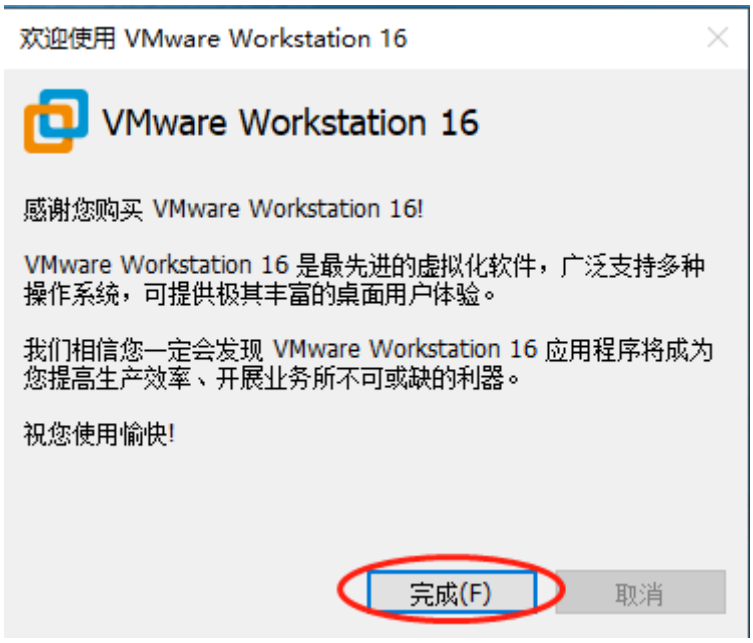


打开安装完成的虚拟机完成注册

双点桌面图标

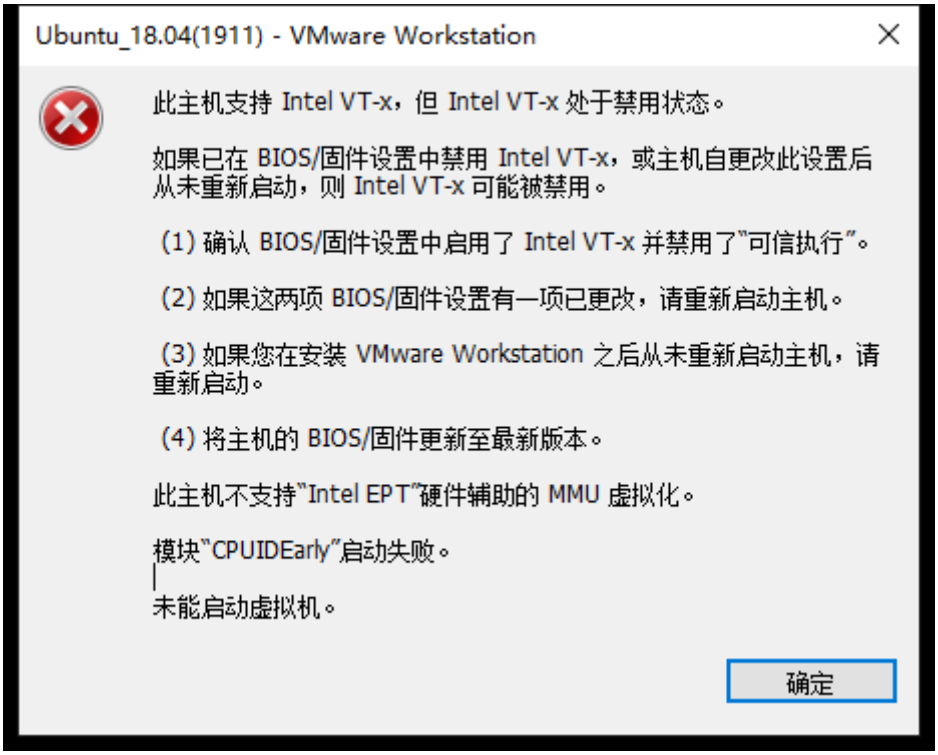
运行 VMware Workstation Pro，第一次运行时会有如下提示，请按下图提示完成注册。

注册成功后进入完成注册页面。



现在你可以使用虚拟机软件了。

- 如果出现如下界面，说明你的电脑 CPU 的 VT 指令没有打开。



这个问题是因为 CPU 没有打开 VT 指令的开关导致的。需要进入BIOS 打开Virtualization Technology 开关，具体操作如下：

- 在电脑开机时按住 `DEL` 键（或者 `F2`、`ESC`、`F12` 键） 进入BIOS设置程序，打开 VT 虚拟化开关即可。机器主板型号不同进 BIOS 的方法可能会有差异，详情咨询电脑厂商或者自行使用 DeepSeek 解决。

- 另可参见如下文章：
  - https://jingyan.baidu.com/article/4b52d702f9c25cfc5d774b5a.html
  - https://blog.csdn.net/QQ2119459337/article/details/78874267

如果自己依旧无法搞定，请去 **明择工作室** 找魏老师！

4. VMware Fusion 创建虚拟机器

本节课我们来讲述如何使用 VMware Fusion 软件创建虚拟机器。

用 VMware 安装 Ubuntu Linux 的步骤大致分为两步:

1. 创建虚拟机器（相当于买了一台电脑）。
2. 安装 Ubuntu 操作系统。

如果你是 Windows 操作系统，那你只要使用 VMware Workstation 创建虚拟机器即可，第一步创建虚拟机操作界面不同，但原理都是一样的。第二步安装 Ubuntu 的操作就和 MacOS 完全一样了。 此种安装方法 100% 成功且安全。

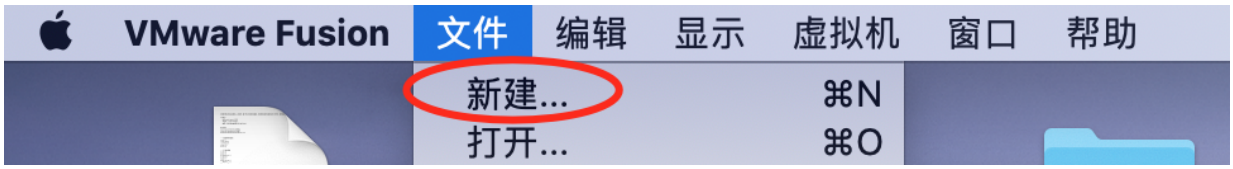
下面我们用 VMware 创建一个虚拟机。

创建虚拟机器

- 先在 MacOS 的启动台  ，然后找到 VMware Fusion 

并启动它。

在菜单栏找到 `文件`，点击新建，如下图：



在 选择安装方法 界面中选择 创建自定义虚拟机，后点击 继续，如图所示：

> 此时 Windows 下的 VMware Workstation 的界面与苹果电脑的界面略有不同，请注意观察找到对应功能的位置进行操作。

在 选择操作系统 界面中选择 Linux 的 Linux 64 位，后点击 继续，如图下图所示：

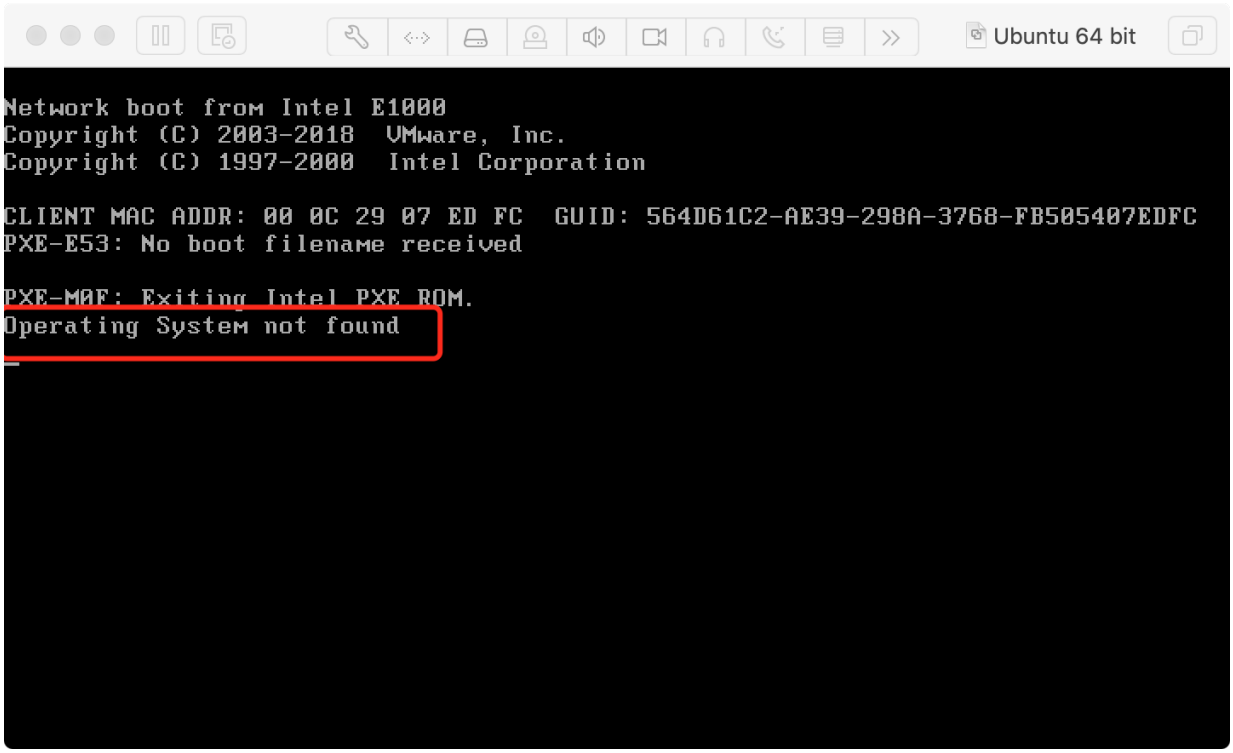
在 选择固件类型 界面中选择 传统 BIOS，后点击 继续，如图下图所示：

在 选择虚拟磁盘 界面中选择 新建虚拟磁盘，后点击 继续，如图下图所示：

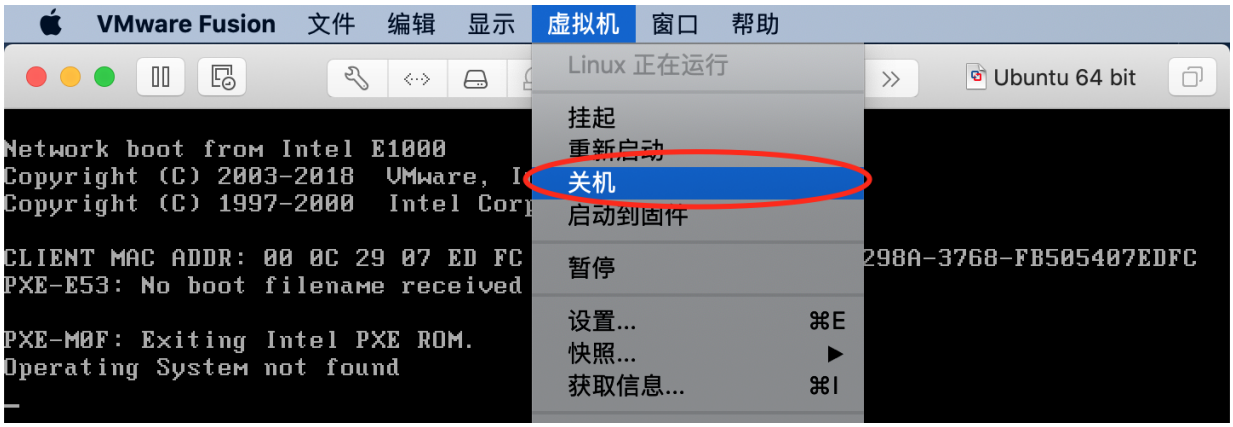
在 完成 界面中点击 完成，如图下图所示：

选择 虚拟机器文件的保存位置（名称最好用英文），然后点击 存储，如图下图所示：

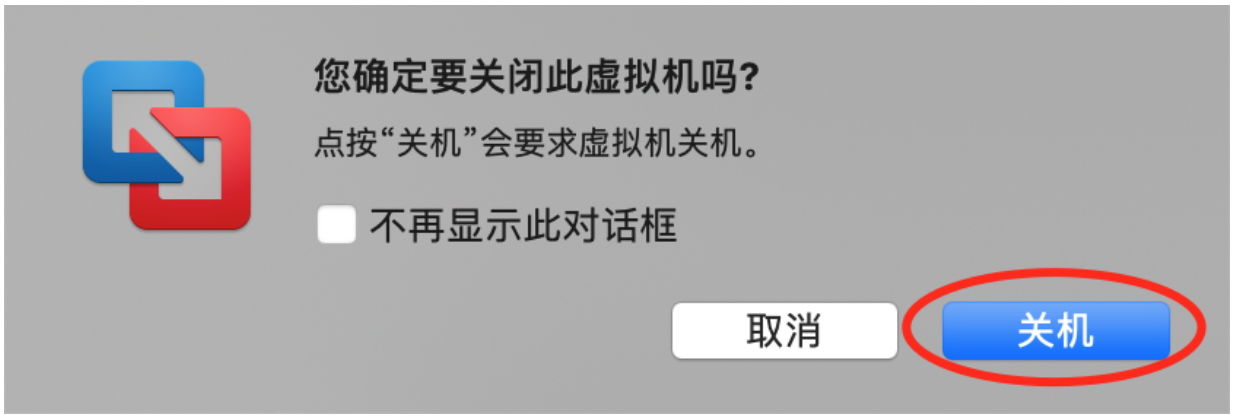
新创建的虚拟机在创建完成后会直接运行，因为没有安装操作系统，所以会显示操作系统没有找到这样的信息。如下图所示：



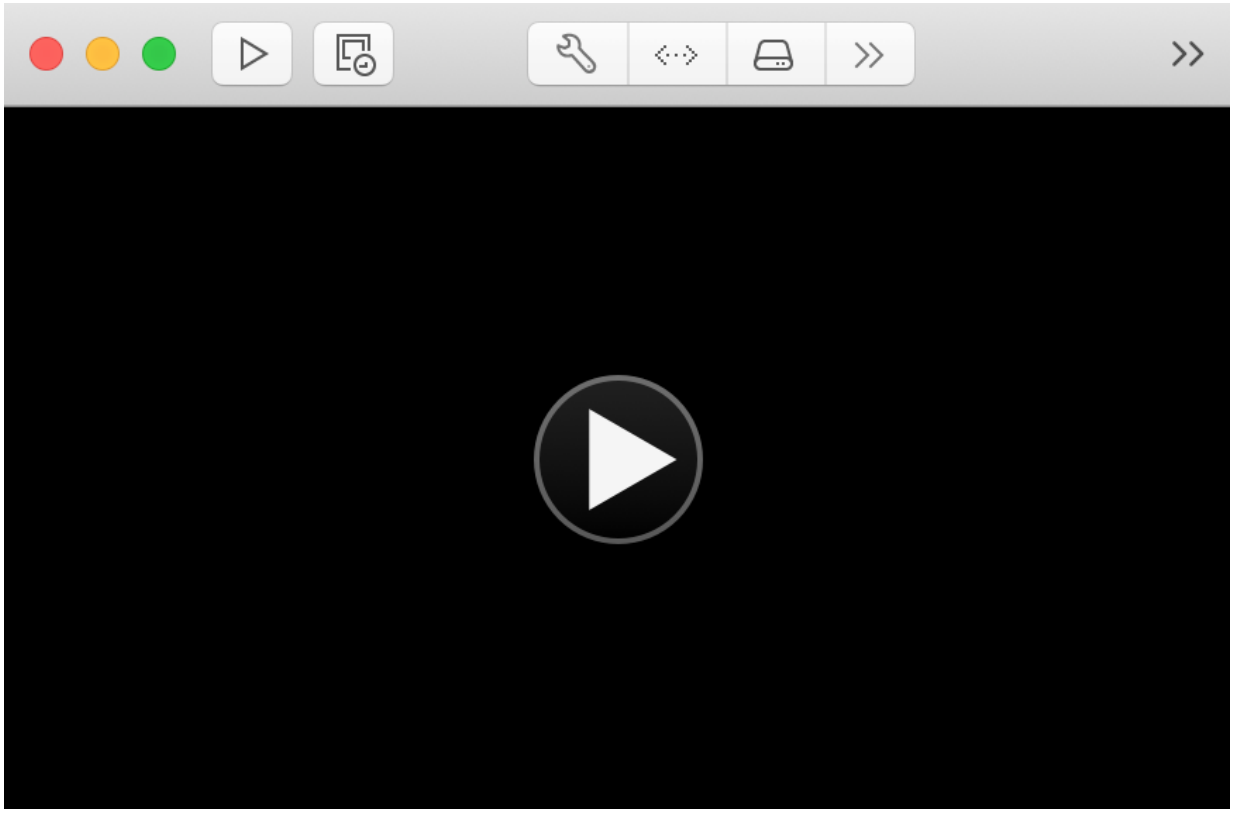
此时先关闭这个正在运行的虚拟机选择 菜单中的 虚拟机 下的 关机。这种关机方式相当于按一下电源键强行关机。如下图所示：



点击关机完成关机操作。

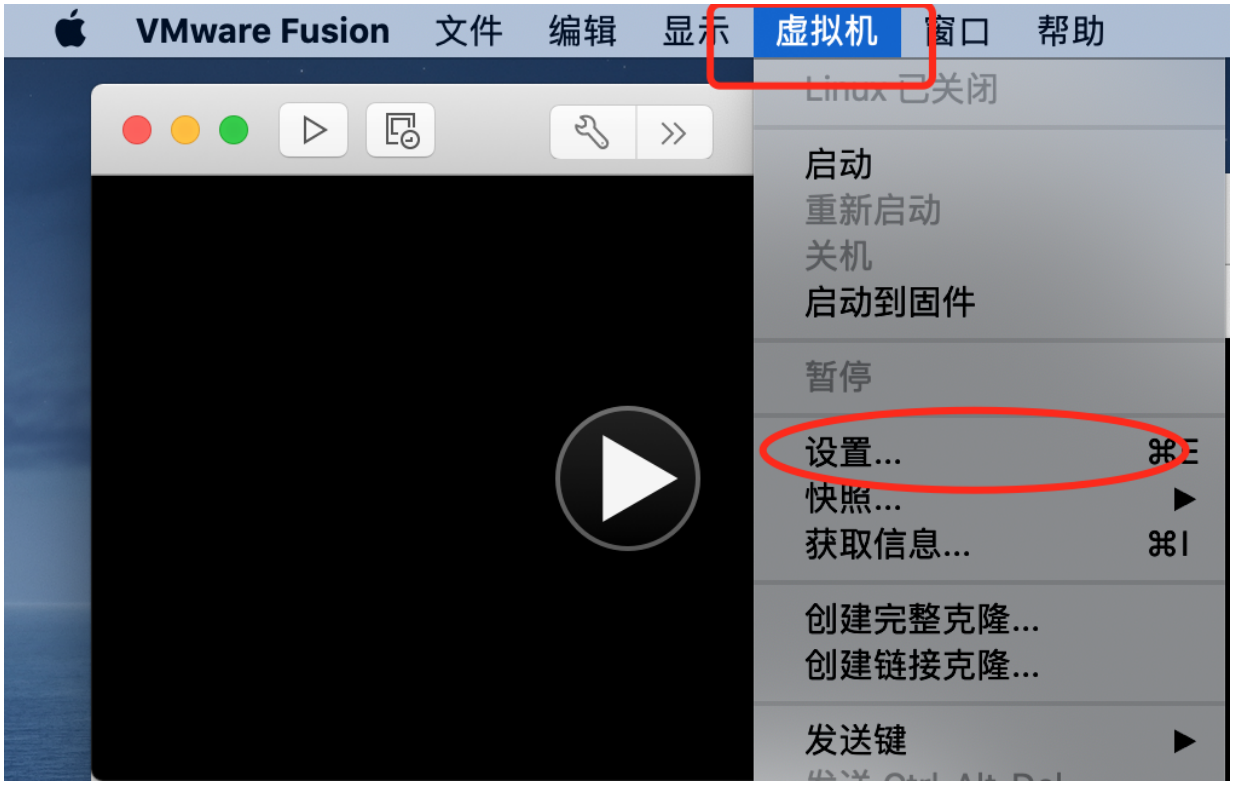


关机后的效果，如下图所示：



接下来我们要修改一下硬件配置，新创建的虚拟机默认硬盘容量是 20G，内存是 4G，这个时候我们可以根据自己的情况来修改，这个虚拟机的配置如下：

进入设置菜单位置:



点击进入设置界面如下：

在这里，我们可以修改虚拟机的各种硬件配置：我们将内存改为 8G，将硬盘由 20G 改成 500G。注意这里面的硬盘改成 500G 并非真实的占用 500G 硬盘，而是自动根据需要占用真是硬盘，这个 500G 是虚拟硬盘的最大容量。

修改内存为 8G，如下图：

修改硬盘为 500G，后点击应用， 如下图：

好了，我们已将有了一台虚拟的机器了。下节课我们再安装操作系统。

5. 安装 Ubuntu 24.04 LTS 操作系统

本节程来讲述如何在 MacOS 系统下使用 VMware Fusion 虚拟机安装最新的 Ubuntu 24.04 LTS 操作系统。

第2节课我们已经下载了 Ubuntu 的 ISO 的镜像文件 `ubuntu-24.04.2-desktop-amd64.iso` 在这里我们需要用到了。如果你还没有下载，那么去下面的网址下载吧：

- 下载地址：https://www.ubuntu.com/download/desktop

下面我们开始安装系统了。

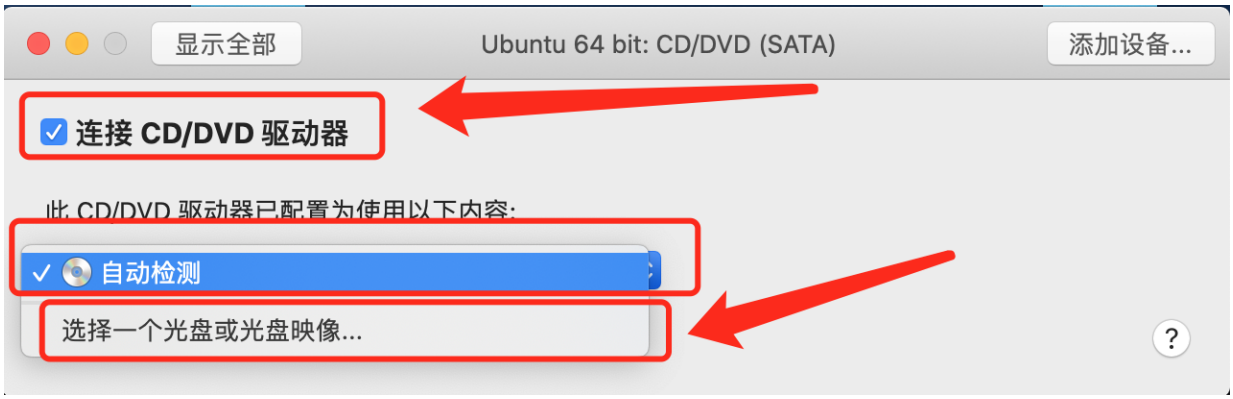
2. 安装 Ubuntu 操作系统。

1. 将 光盘 插入虚拟机。

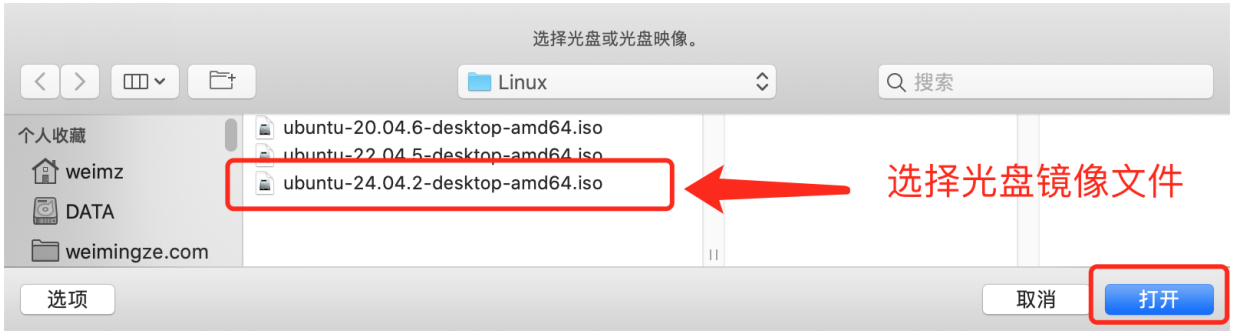
此时我们可以不用将 iso 文件刻录成光盘。虚拟机直接使用 iso 文件即可。

进入虚拟机设置界面如下：

点击 CD/DVD 进入到虚拟机的光盘管理界面，选择连接CD/DVD驱动器，然后点击选择一个光盘或光盘映像。

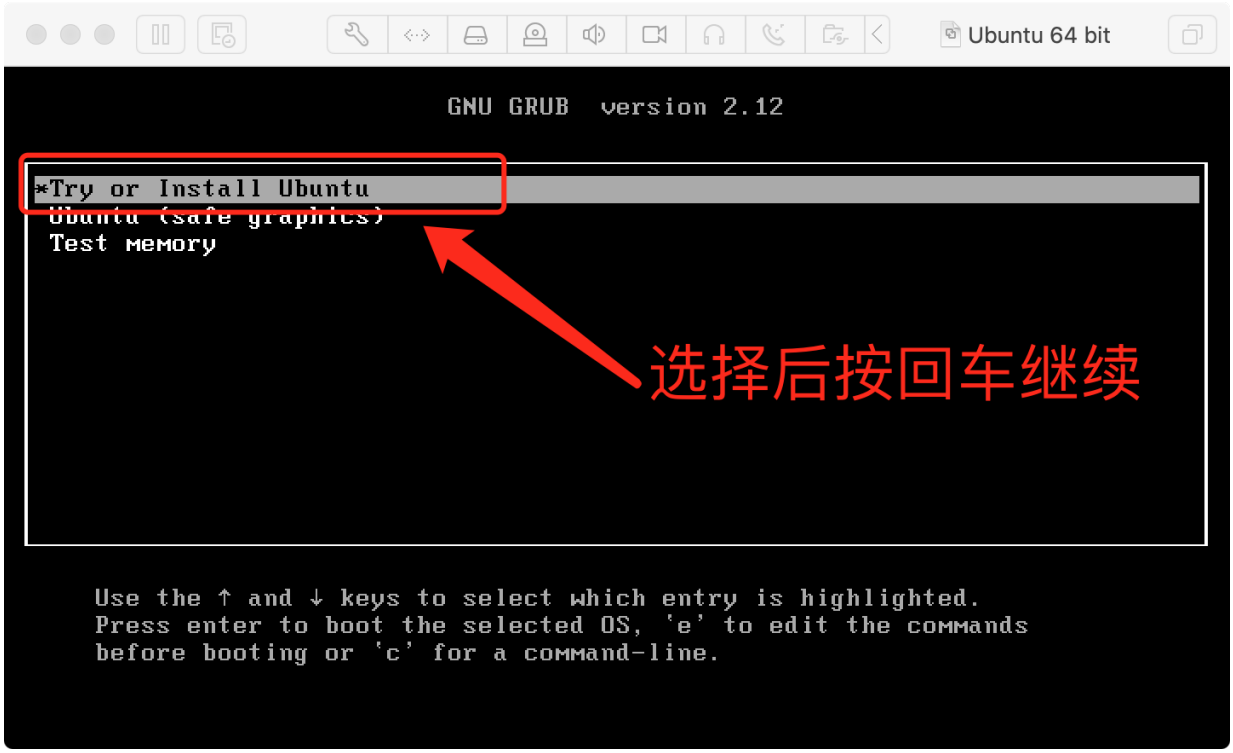


选择已经下载的 iso 镜像文件，然后退出。

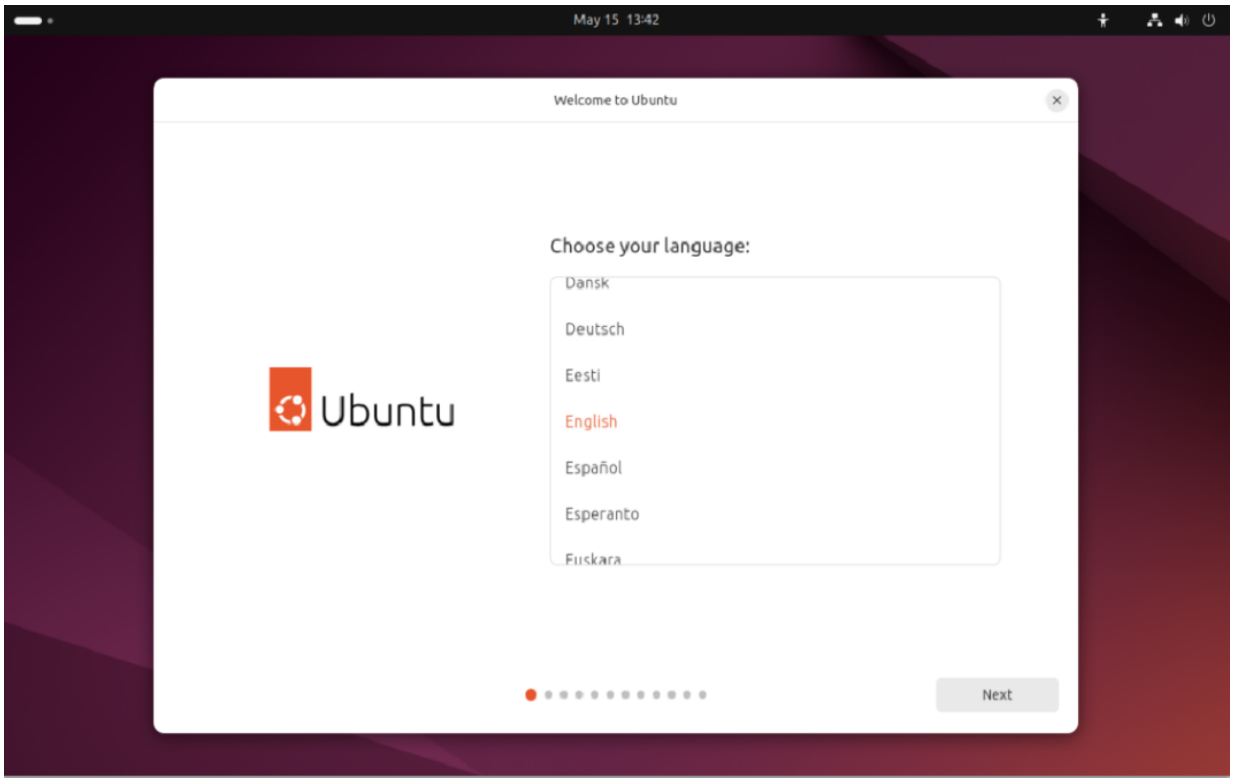


接下来点击 菜单 虚拟机 中的启动 来开启虚拟机。

然后将进入启动选项界面，选择 `Try Or Install Ubuntu` 就可以在光盘上运行 Ubuntu 了。



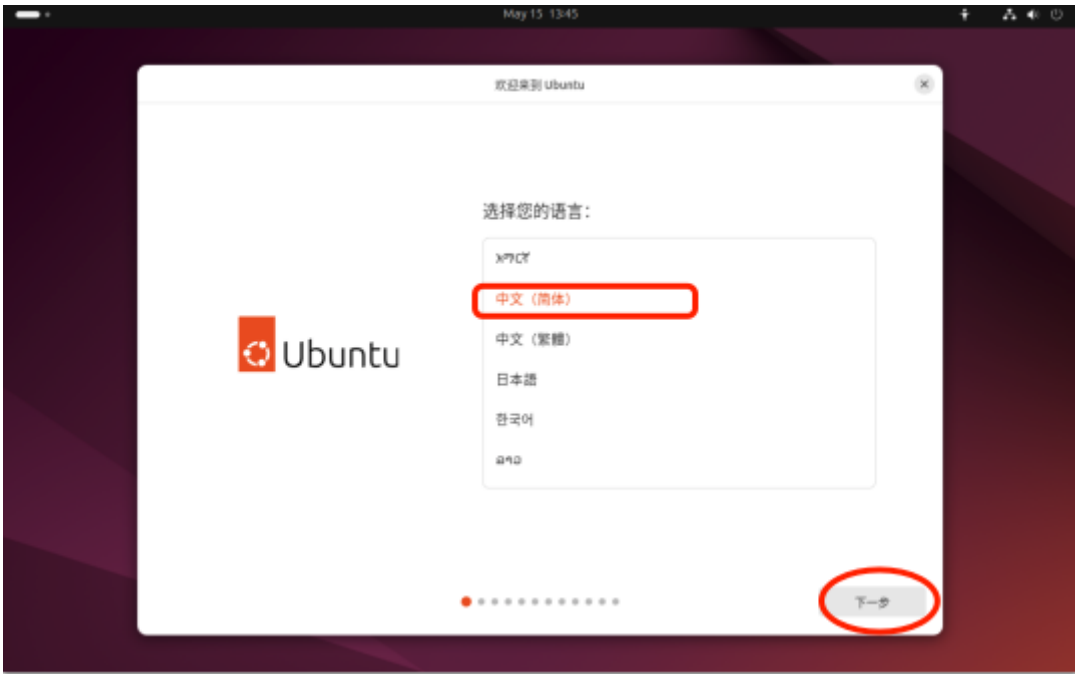
运行后的 Ubuntu 的效果是这样的。详见下图：



此时我们根据提示试用 Ubuntu 系统或者安装Ubuntu系统。

下面我们选择将 Ubuntu 安装到虚拟机的硬盘上，以后从虚拟机的硬盘启动。

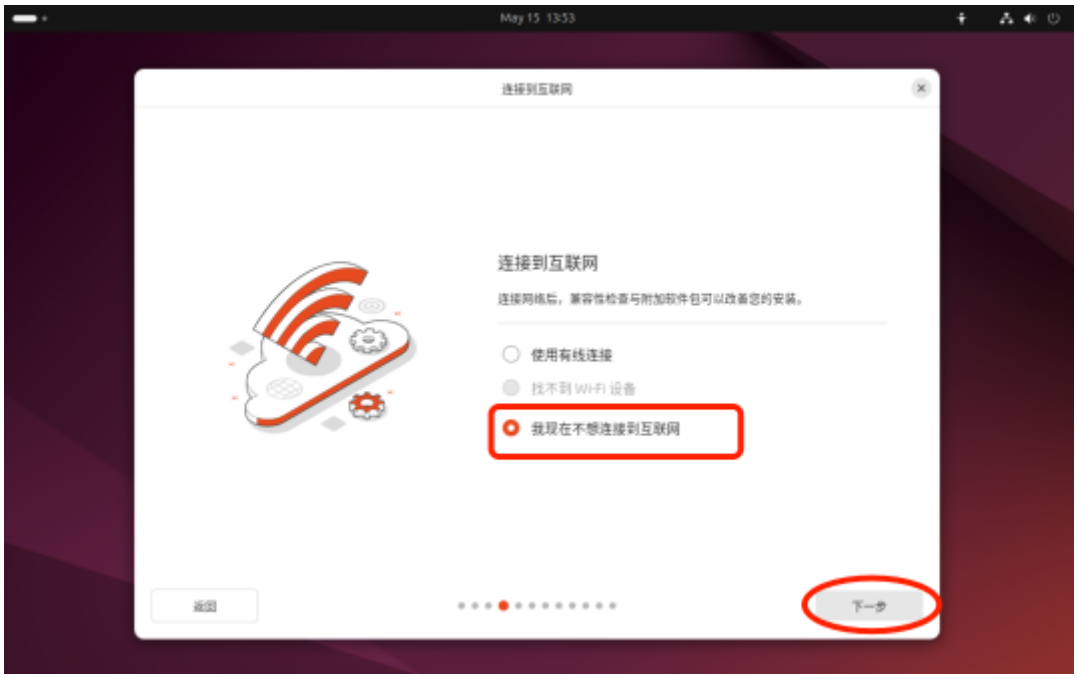
选择安装语言，后续默认使用此语言运行。



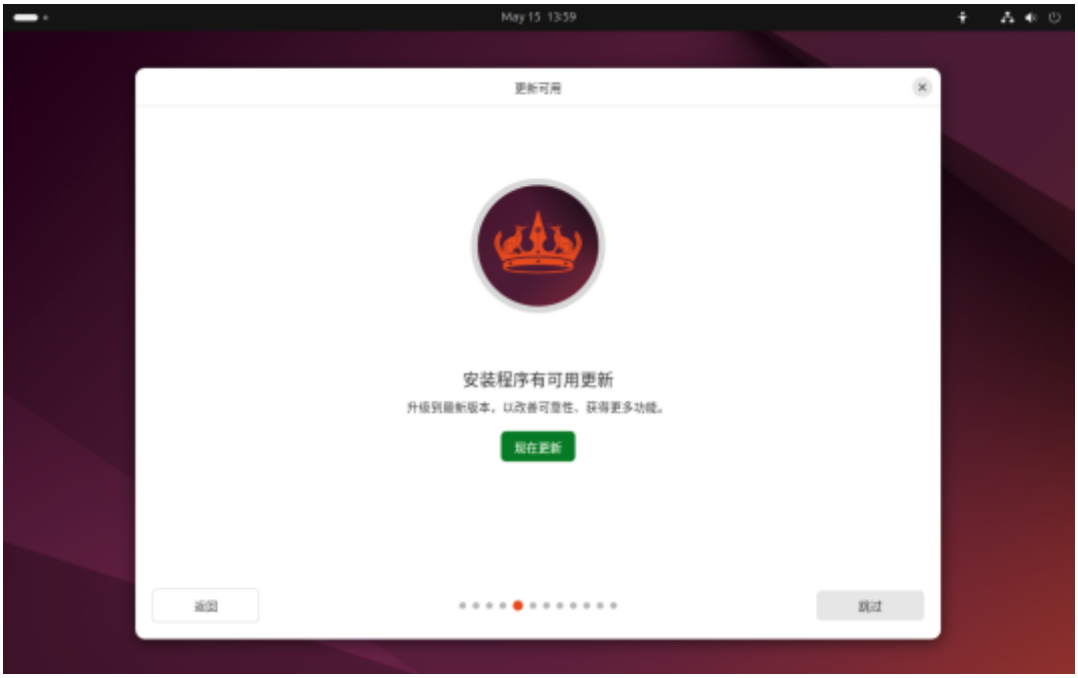
可访问性 界面都使用默认，点击 下一步 继续。

键盘布局 界面选择您的键盘布局，一般我们选择默认即可，点击 下一步 继续。

连接到互联网 界面选择 我现在不想连接到互联网 ，否则会自动升级很长时间，影响安装，这些升级可以后续再进行，点击 下一步 继续。



更新可用 界面选择跳过此步骤， 点击 跳过 继续。

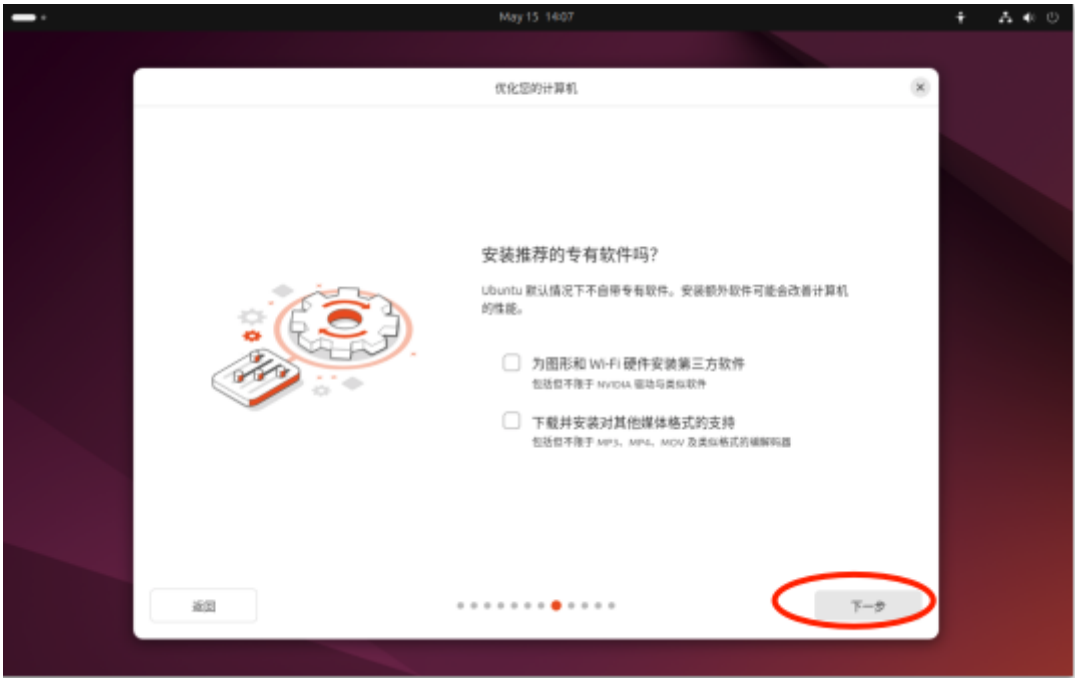


使用或安装 Ubuntu 界面选择 安装 Ubuntu， 点击 下一步 继续。

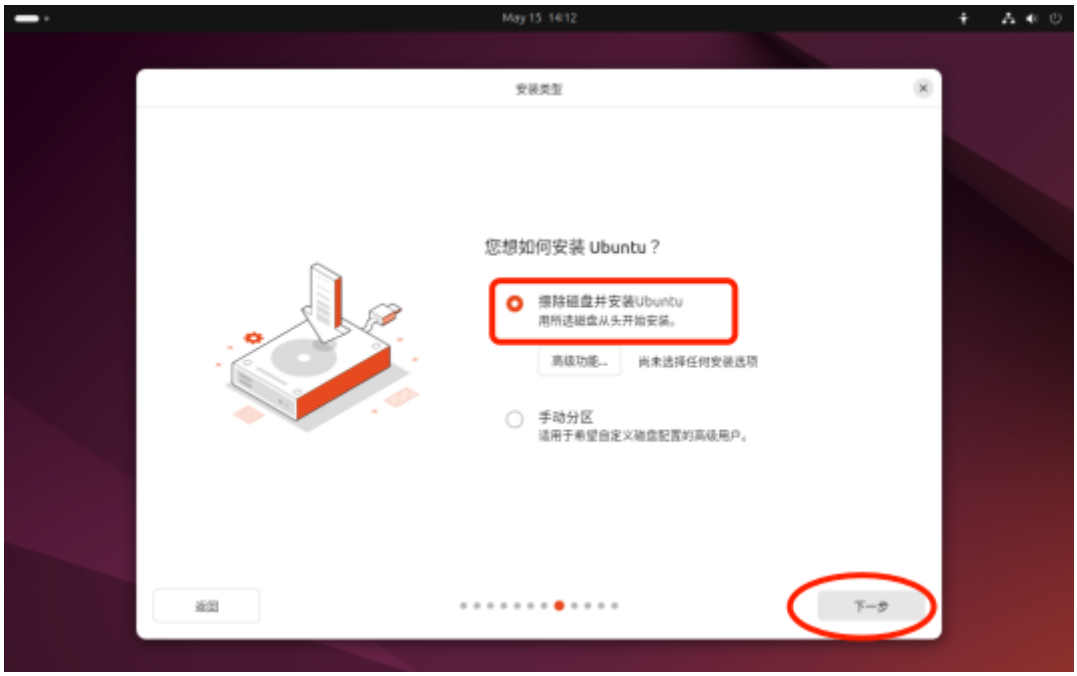
安装类型 界面选择 交互安装， 点击 下一步 继续。

应用程序和更新 界面选择 默认集合， 点击 下一步 继续。

优化您的计算机 界面选择 默认，以后需要啥软件再安装， 点击 下一步 继续。



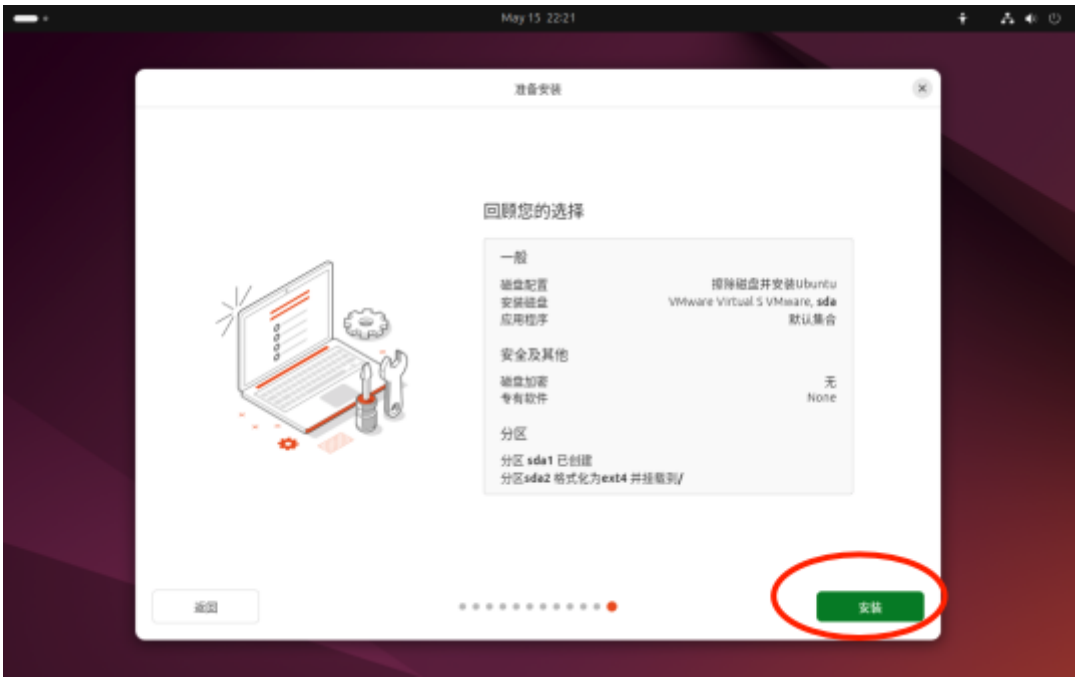
进入 安装类型 界面，这一步很重要，如果你是真机安装要选择手动分区并要认真操作，否则数据容易丢失。我们现在是虚拟机，默认的500G硬盘也是虚拟机，因此直接擦除虚拟磁盘就好，选择 擦除磁盘并安装Ubuntu， 点击 下一步 继续。



进入 设置您的账户 界面，这一步将为您设置机器名称，在新的操作系统上创建一个账户，同时需要输入用户名和密码，点击 下一步 继续。

选择您的时区 界面，选择 中国上海 即可，点击 下一步 继续。

准备安装 界面是确认你的安装信息，如果你确认无误，点击 安装 则进入安装进度界面。

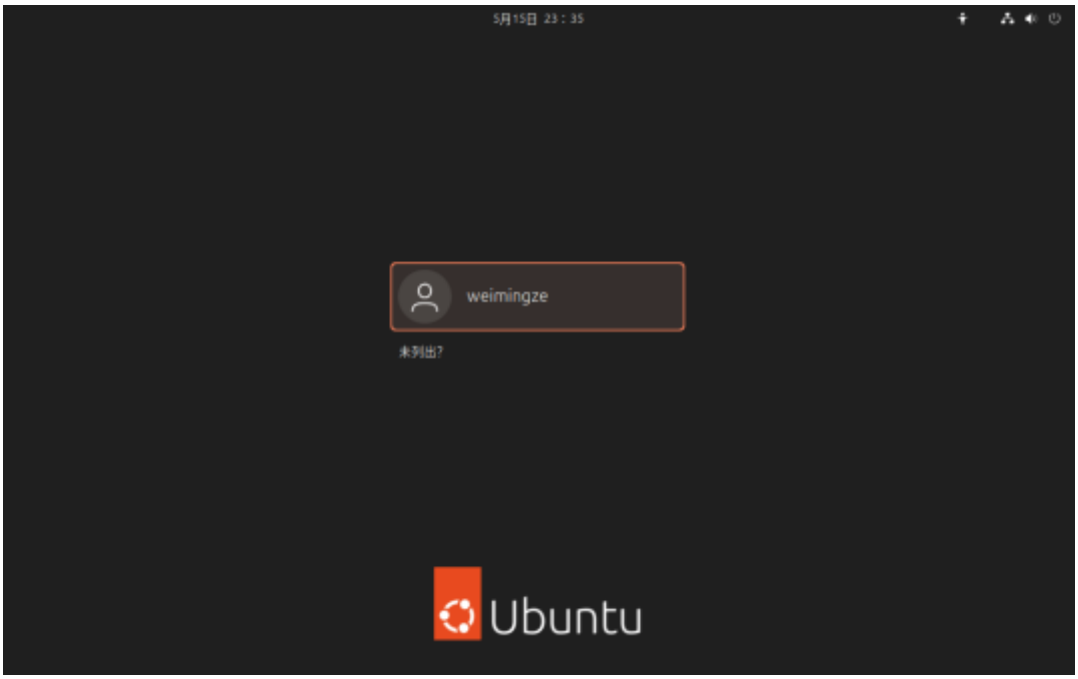


接下来就是复制文件阶段，此阶段需要很长的时间，请耐心等待！等安装完成后会提示重新启动计算机。如下图所示：

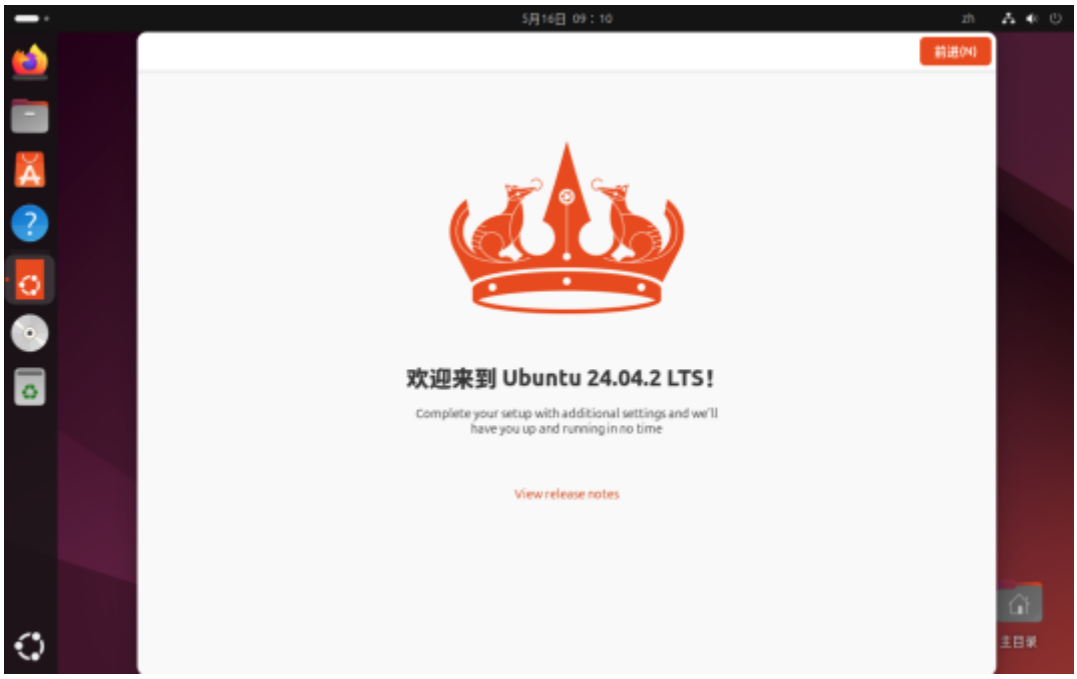
此时可以弹出光盘后点击重新启动。

重新启动以后，就可以正常使用 Ubuntu 操作系统了。

每次登陆桌面前需要输入登录信息，如下图所示，此时的用户名和密码就是你在安装时设定的，不要忘记哦。



当我们第一次运行 Ubuntu 的时候会进入欢迎界面如下图所示；一路点击前进，内容自己看一下就好。



点击桌面左下角的 显示应用 按钮，就可以列出本机安装的所有程序，这里面我们最常用的是 终端。

启动终端。终端的界面如下所示：

通过终端窗口，我们可以用文字的方式来操作计算机，我们把这些文字称之为命令和参数，每一个命令都对应着一个计算机内部的应用程序。

通过终端窗口输入命令是 Linux 和 UNIX 系统开发人员和运维人员最常用的运行方式。

下一章我们将详细的讲解，通过命令来操作计算机的方法。

Ubuntu 操作系统安装完毕，快来体验一下吧！

6. Ubuntu 24.04 开发体验

Ubuntu 操作系统已经自带了 Python 的解释器 3.12.3 的版本。可谓是目前最新的系统了。

Ubuntu 已经不再预装 Python2。Python2 已经成为过去了！

这节课我们先来使用终端编写一个 Python3 的 `hello world` 程序。

通过这个示例，我们来学习在 Ubuntu Linux 上面编写和运行 Python 程序的方法。

用 nano 编辑器编写 Python 程序

打开终端，在终端内输入命令 `nano hello.py` 然后 `回车` 进入编辑界面如下图所示:

输入如下一行文字

```
print("hello world!")
```

如下图所示：

注意下面是操作菜单，`^X` 表示按键 `Control + x` 表示退出 nano 编辑器。如果提示操作是 `M-U` 则表示按键 `Alt + u` 或 表示先按一下 `ESC` 按键，再按一下 `u` 按键表示撤销之前操作。非常简单易用的编辑器。其他操作你是不是也都会了呢？后面我会详细讲解 nano 的用法。

我们输入 `Control + x` 提示 `保存已修改的缓冲区`，选择 `Y` 是。保存后退出 nano 界面回到终端中。

接下来在终端中输入 `python3 hello.py` 用 Python 来运行 `hello.py`这个 Python 程序，如下图所示:

我们可以在 Python上编辑和运行 Python 程序了。

第二章、文件操作

1. Linux 文件系统简介

文件系统(File System)是操作系统用于存储数据、挂载设备而设计的一种数据结构，目的是存储数据和管理设备。如硬盘、U盘等上的数据文件的组织和存储都是用文件系统来实现的。

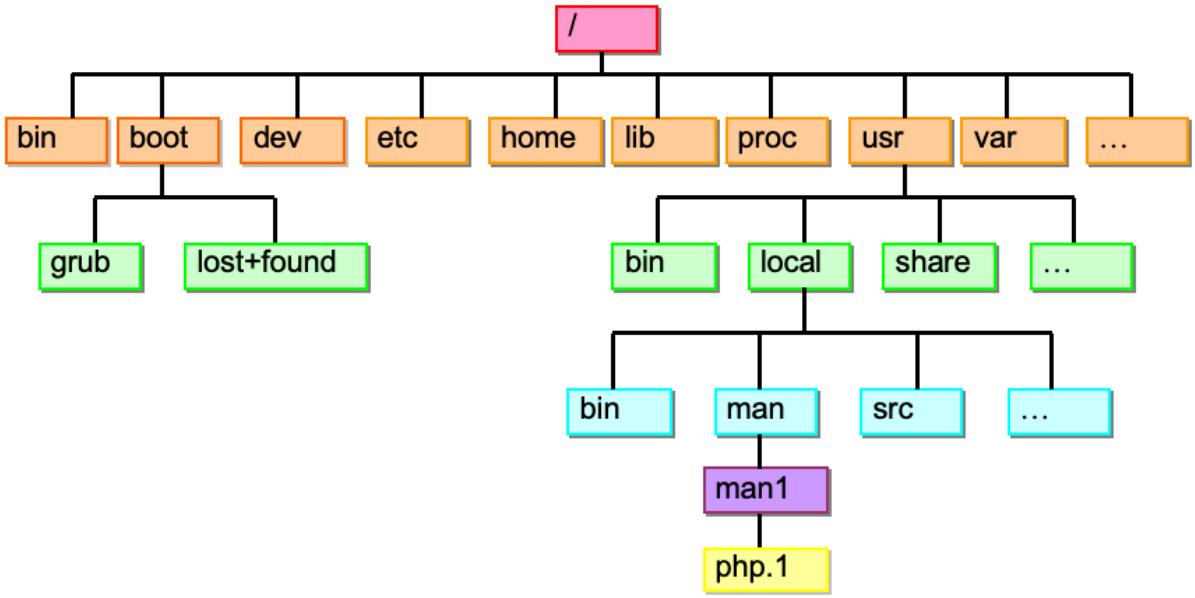
一般现代操作系统的文件系统都是树形结构。就是一个文件夹内可以存放多个文件或文件夹，文件夹内部还可以存储其他文件和文件夹，这就形成了一种树形结构，也称作目录树。

所谓 `目录` 你就理解成 `文件夹` 就行了。

在 UNIX 和 Linux 操作系统中，一切文件系统的起点都是 `"根"`（你可以理解成大树的根，向上可以有分支和叶子），用 字符 `/` 来表示。

`根` 是最顶层的文件夹，是访问一切文件或文件夹的起点位置。

MacOS 系统源于 UNIX 系统，因此 MacOS 的文件系统的目录结构也是和 Linux 一样，如下图所示。



我们要想到达 `"share"` 这个文件夹要从 `根` 开始，然后进入 `usr` 文件夹，才能进入到 `share` 这个文件名。我们把从根开始，经过各个文件夹到达 `share` 的过程叫做"路径"。

路径

路径是描述一个文件或文件夹位置的字符串，如上图中 "share" 文件夹的路径就是 `/usr/share`，第一个 `/` 代表根，后面再出现 `/` 都代表路径的分隔符。

> 需要注意的是 Windows 的路径分隔符是 反斜杠 `\`，而 Linux 和 UNIX 的路径分隔符是斜杠 `/`。

路径分为两种:

- 绝对路径：指从 根 `/` 开始的路径，如：`/usr/share`。
- 相对路径：指从 根 以外的其他文件夹开始的路径，如：`man1/php.1`，相对路径要根据 `当前工作目录` 来确定具体位置（具体内容我们后面再讲）。

当我们向一个计算机内插入U盘或光盘时，这是通常在某个文件夹下多出一个文件夹，这个多出来的文件夹，我们称之为挂载点。而U盘的文件系统将作为一个子分支挂载在这个挂载点上。在 Ubuntu 24.04 系统下插入 U 盘通常会挂载到 `/media/你的用户名/` 这个文件下。你自己找个 U 盘试一下吧。

以下是根下的文件夹的说明和作用：

文件夹	内容
`/`	根，UNIX/Linux 文件系统最顶层的文件夹。
`/bin/`	用来存储用户命令的文件夹。文件夹 `/usr/bin` 也被用来存储用户命令。
`/sbin/`	用于存储系统命令（例如 shutdown）的文件夹。
`/home/`	默认是存储每个用户自己数据的文件夹（称为用户主目录），每个用户都会在这个文件夹下有一个子文件夹，用于存储此用户的数据（root用户除外）。
`/root/`	根用户（超级用户）的用户主目录。
`/mnt/`	该文件夹中通常包括系统引导后被挂载的文件系统的挂载点。如，光盘挂载点可以选择 `/mnt/cdrom`。
`/boot/`	存放 Linux 内核和系统启动期间使用的文件的文件夹。
`/lost+found/`	被 fsck 用来放置零散文件（没有名称的文件）
`/lib/`	用来存放系统动态连接共享库的文件夹。
`/dev/`	存储设备文件的文件夹。
`/etc/`	存放了系统管理时要用到的各种配置文件的文件夹 。
`/var/`	用于存储 variable（或不断改变的）文件，例如日志文件和打印机假脱机文件的文件夹。
`/usr/`	这是存储 UNIX 系统命令的文件夹。用户的很多应用程序和文件都存放在这个文件夹下。
`/proc/`	一个虚拟的系统文件夹，可以在这个文件夹下获取系统信息。这些信息是在内存中，由系统运行时自己产生的。
`/tmp/`	用户和程序的临时文件夹。用来存放不同程序执行时产生的临时文件。
`/opt/`	可选文件和程序的存储文件夹。该文件夹主要被第三方开发者用来简易地安装和卸装他们的软件包。

在上述安装的Ubuntu 24.04 系统当中，当我们使用 `weimingze` 这个用户登录的时。我们只能对 `/home/weimingze/` 这个文件进行修改。上述 根 `/` 下的文件夹有些只能查看（如：/etc/），甚至有

些连看都不能看（如：/root），这源于 Linux 的权限管理。 `weimingze` 是普通用户，要管理上述文件夹需要使用 `root` 用户登录或切换到 `root` 用户。 这也是很多苹果手机或安卓手机在刷机之前必须要想办法 `root` 的原因之一，只有拿到 `root` 用户的权限才能对操作系统的核心进行修改。

Linux/UNIX 的文件名是区分大小写的（和 Windows 不同），例如 `/home/weimingze/`、`/home/WEIMINGZE/` 和 `/home/Weimingze/` 是三个不同的文件夹。文件名也是如此。在工作时，如果没有特殊要求，建议要把文件名的英文部分用小写编写。

下面我们来学习一下文件系统常用的操作命令。

2. 文件的操作命令

这一节的主要目标是学习文件操作的命令。我们将学习创建、删除文件，修改文件的名称以及复制文件等操作。

用到的命令有：`ls`、`touch`、`rm`、`cp`、`mv`。

本节讲解的命令如下：

- ls 命令
- touch 命令
- rm 命令
- cp 命令
- mv 命令

通常一个命令对应一个独立功能的程序。命令一定有命令名称，后面跟选项或参数，这些选项和参数通常用一个空格或多个空格分隔。

我们先学习命令的格式。

命令的格式如下:

```
命令名 [选项] [参数]
```

> 注：选项和参数用 中括号 [] 括起来表示其中的内容可选。 通常选项在前，参数在后。具体要根据命令内部的实现来决定。

如：

```
weimingze@mzstudio:~$ ls -a /
.                 cdrom  lib.usr-is-merged  opt   sbin.usr-is-merged  tmp
..                dev    lib64              proc  snap                usr
```

```
bin                etc    lost+found         root  srv                  var
bin.usr-is-merged  home   media              run   swap.img
boot               lib    mnt                sbin  sys
```

上述终端运行中，`ls` 是命令，`-a` 是选项， `/` 是参数，组合起来就形成了上述结果。

- 选项通常以 减号（`-`） 或 两个减号（`--`） 开头。选项通常指示此命令执行那种类型的操作；一个命令有哪些选项由该命令来决定。
- 参数通常是要传递给命令的数据。

2.1 ls 命令

ls 命令用于列出当前文件夹或其他的某个文件夹下有哪些文件和文件夹。

命令格式:

```
ls [选项1] [选项2 ...] [参数1] [参数2 ...]
```

示例:

列出当前文件夹下有哪些文件或文件夹。

```
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
```

列出 根 `/` 下有哪些文件或文件夹。

```
weimingze@mzstudio:~$ ls /
bin                home               mnt   sbin.usr-is-merged  usr
bin.usr-is-merged  lib                opt   snap                var
boot               lib.usr-is-merged  proc  srv
cdrom              lib64              root  swap.img
dev                lost+found         run   sys
etc                media              sbin  tmp
```

列出 `/usr/` 下有哪些文件或文件夹。

```
weimingze@mzstudio:~$ ls /usr/
bin  games  include  lib  lib64  libexec  local  sbin  share  src
```

列出当前文件夹下的文件和文件夹的详细信息。

```
weimingze@mzstudio:~$ ls -l
total 40
```

```
-rw-rw-r-- 1 weimingze weimingze   23 May 16 13:02 hello.py
drwx------ 3 weimingze weimingze 4096 May 16 12:09 snap
drwxr-xr-x 2 weimingze weimingze 4096 May 16 12:09 下载
drwxr-xr-x 2 weimingze weimingze 4096 May 16 12:09 公共
drwxr-xr-x 2 weimingze weimingze 4096 May 16 12:09 图片
drwxr-xr-x 2 weimingze weimingze 4096 May 16 12:09 文档
drwxr-xr-x 2 weimingze weimingze 4096 May 16 12:09 桌面
drwxr-xr-x 2 weimingze weimingze 4096 May 16 12:09 模板
drwxr-xr-x 2 weimingze weimingze 4096 May 16 12:09 视频
drwxr-xr-x 2 weimingze weimingze 4096 May 16 12:09 音乐
```

> `-l` 选项是列出文件或文件夹的详细信息。下面我们来说一下详细信息.

文件的详细信息说明

每一行是一个文件信息。该信息用空格分隔成八列。

1. 第一列：文件类型和权限信息
   - 每一行的第一个字母代表文件的类型：`d` 代表文件夹，`-` 代表普通文件，`l` 代表软链接文件，`c` 代表字符设备文件（比如键盘），`b` 代表块设备文件（比如 U 盘），`s` 代表套接字文件，`p` 代表管道文件等。
   - 每一行的第2~10个字符为文件或文件夹的读、写和执行权限：`-`代表无权限，`r`代表有读权限，`w`代表有写权限，`x`代表有执行权限。权限分为三组，分别是拥有者（属主）权限，拥有者所在的组（属组）权限和其他权限。如 `rwxr-xr--` 前三个 `rwx`代表拥有者可以读写和执行，中间三个`r-x` 代表 同组内的用户可以读和执行，但是不可以写。第8~10个`r--` 代表 不是拥有者，也不是同组内的其他用户可以读，但是不可以写和执行。

     > 假设现实当中我在力学实验室的组中，我的一本书，他的属主是我，他的属组是力学实验室，书的权限是`rw-r-----`，那么我可以读和修改，力学实验室中的老师都可以读，其他人则不能执行任何操作。

2. 第二列：代表该文件的硬链接数。你可以理解成其他文件或文件夹关联到此文件的个数，文件一般都是1。
3. 第三列：代表该文件或文件夹的拥有者用户，就是属主。上述文件的拥有者都是 weimingze。
4. 第四列：代表该文件或文件夹的所在的组，就是属组，weimingze这个组内的所有成员都遵循第 5-7 个字符所指示的权限。
5. 第五列：该文件或文件夹所占用的空间。文件为内容的长度，文件夹一般显示 4096 是索引节点 (inode) 所占用的空间。
6. 第六、七、八列：分别是此文件或文件夹的最后修改日期（月 日 时间）信息。

7. 第九列：该文件或文件夹的名称。

ls命令的常用选项

选项	说明
`-l`	显示详细信息
`-a`	显示以 `.` 开头的隐藏文件
`-t`	按修改时间排序（最新修改的排前面）
`-S`	按文件大小排序（从大到小）
`-r`	反向排序（配合 -t、-S 等使用）

> 注：Linux 和 UNIX 系统中，默认以点开头的文件或文件夹为隐藏文件。它真实存在，但通常不显示，但在 Windows 系统中会正常显示。
>
> 这就是很多安卓手机的存储卡拿到 Windows 电脑上读取文件的时候，会发现很多以点开头的文件就是这个道理。

在 UNIX 和 Linux 操作系统的命令当中，有些选项是可以任意组合的，比如说下列用法是相同的，如：

列出当前文件夹下所有的文件和文件夹的详细信息。

```
weimingze@mzstudio:~$ ls -l -a
# 或
weimingze@mzstudio:~$ ls -a -l
# 或
weimingze@mzstudio:~$ ls -la
# 或
weimingze@mzstudio:~$ ls -al
```

上述命令的运行结果相同。

2.2 touch 命令

touch 是 Linux/Unix 系统中用于 创建空文件 和 修改文件时间戳 的命令。

touch 命令会在没有文件时创建此空文件，如果有此文件时会修改此文件的访问时间和修改时间。

示例

```
weimingze@mzstudio:~$ touch a.py
weimingze@mzstudio:~$ ls -l a.py hello.py
-rw-rw-r-- 1 weimingze weimingze    0 May 16 14:10 a.py
-rw-rw-r-- 1 weimingze weimingze   23 May 16 13:02 hello.py
weimingze@mzstudio:~$ touch hello.py
weimingze@mzstudio:~$ ls -l a.py hello.py
-rw-rw-r-- 1 weimingze weimingze    0 May 16 14:10 a.py
-rw-rw-r-- 1 weimingze weimingze   23 May 16 14:11 hello.py
```

可见 使用 `touch a.py` 命令，在没有 `a.py` 这个文件的时候会创建这个文件。

使用 `touch hello.py` 对 `hello.py` 这个文件的修改时间进行的修改。

touch 的常用选项

选项	说明
`-a`	**仅更新访问时间（atime）**
`-m`	**仅更新修改时间（mtime）**
`-c`	**不创建新文件**（仅当文件存在时才更新时间戳）
`-d`	**指定时间**（而不是当前时间）
`-t`	**使用自定义时间格式**（`[[CC]YY]MMDDhhmm[.ss]`）
`-r`	**参照另一个文件的时间**（复制时间戳）

特殊文件名的处理

当我们要创建一个含有空格等的特殊文件名是，需要将文件名或整个路径名用英文的双引号（`"`）括起来。

如我要创建一个 `hello world.py` 文件时，这个文件名中有空格，如果我们运行 `touch hello world.py` 这个命令，就创建了 `hello` 和 `world.py` 两个文件。

正确的做法是：

```
touch "hello world.py"
```

还有另外一种做法是将空格进行转义（类似于Python中字符的转义），就是用反斜杠加一个空格('\ ') 代替文件名中的一个空格(`' '`)。

如上述创建也可以使用如下命令完成:

```
touch hello\ world.py
```

2.3 rm 命令

rm（remove）命令是 Linux/Unix 系统中用于删除文件或文件夹的命令。

rm 命令是永久删除文件或文件夹（不进入回收站），需谨慎使用！

命令格式

```
rm [选项] 文件或文件夹名
```

示例

```
weimingze@mzstudio:~$ ls
a.py  hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ rm a.py
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
```

a.py 被删除了。

rm 的常用选项

选项	说明
`-i`	交互式删除（避免误删）。
`-r`	递归删除文件夹内部的全部子文件，即删除当前文件夹。
`-f`	强制删除（不提示）。

> 注意，请不要运行 `rm -rf /` 这是致命的。

2.4 cp 命令

cp 命令用于复制当前的文件或者是文件夹。

命令格式

```
cp [选项] 源文件或文件夹名 目标文件或文件夹名
```

示例:

复制当前的 `hello.py` 到桌面下，重新命名为 `b.py`。

```
weimingze@mzstudio:~$ cp hello.py 桌面/b.py
weimingze@mzstudio:~$ ls 桌面/
b.py
```

复制当前的 `hello.py` 到桌面下，名称不变。

```
weimingze@mzstudio:~$ cp hello.py 桌面/
weimingze@mzstudio:~$ ls 桌面/
b.py  hello.py
```

复制当前的 `hello.py` 到当前文件夹下，重新命名为 `c.py`。

```
weimingze@mzstudio:~$ cp hello.py c.py
weimingze@mzstudio:~$ ls
c.py  hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
```

cp 的常用选项

选项	说明	示例
无选项	复制文件	`cp file1.txt file2.txt`
`-r` 或 `-R`	**递归复制文件夹**（包括子文件夹和文件）	`cp -r dir1/ dir2/`
`-v`	**显示复制过程**（verbose）	`cp -v file.txt dir/`
`-n`	**不覆盖已存在文件**（no-clobber）	`cp -n file.txt backup/`
`-u`	**仅当源文件更新时才复制**（update）	`cp -u file.txt backup/`
`-a`	**归档模式**（保留所有属性，相当于 `-rp`）	`cp -a dir1/ dir2/`
`-p`	**保留权限、时间戳等属性**	`cp -p file.txt backup/`

2.5 mv 命令

mv 命令常用于移动文件或文件夹到另外的地址，或者为文件或文件夹改名字。

命令格式

```
mv [选项] 源文件或文件夹名 目标文件或文件夹名
```

示例:

将 `c.py` 改名为 `newname.py`

```
weimingze@mzstudio:~$ ls
c.py  hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ mv c.py newname.py
weimingze@mzstudio:~$ ls
hello.py  newname.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$
```

将 `newname.py` 移动到桌面并改名为 `d.py`

```
weimingze@mzstudio:~$ ls
hello.py  newname.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ ls 桌面/
b.py  hello.py
weimingze@mzstudio:~$ mv newname.py 桌面/d.py
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ ls 桌面/
b.py  d.py  hello.py
```

2.6 查看命令帮助信息

一个命令他的格式是什么？有几个选项？我们去哪里去查呢？

我常用两种方法:

1. `--help` 选项查看。格式: `命令 --help`
2. 使用 `man` 命令查看此命令的手册。格式 `man 命令`

使用 `--help` 选项查看 `ls` 命令都有哪些用法?

```
weimingze@mzstudio:~$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
  -a, --all                  do not ignore entries starting with .
  -A, --almost-all           do not list implied . and ..
      --author               with -l, print the author of each file
  ...... 以下省略
```

使用 `man` 命令查看 `ls` 命令都有哪些用法?

```
weimingze@mzstudio:~$ man ls
LS(1)
  User Commands
           LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List information about the FILEs (the current directory by default).  Sor
t entries alphabetically if none of -cftuvSUX nor --sort is specified.
```

```
       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
    ....... 以下省略
```

> 使用上下方向键可以翻页，按 `q` 键可以退出帮助手册。

命令自动补全

当我们使用终端来输入命令的时候，如果命令或参数过长，我们可以先输入几个字符，然后按 `<tab>` 键(Q左侧的按键)去补全剩余的字符。这样可以大大提高我们输入的效率，避免出错。

比如，我们要输入 `touch` 命令，当我们输入 `tou` 时按下 `<tab>` 键，则会补全剩余字符。

```
weimingze@mzstudio:~$ tou
```

关于文件操作相关命令我们就先讲到这里了。

3. 文件内容查看命令

本节课我们将学习在 Linux 下使用命令来查看文件中的内容。下面我们来讲解 `cat`、`tail`、`head`、`more`、`less` 这几个命令的用法。这些命令在 UNIX 系统中也一样可用。

本节讲解的命令如下：

- cat 命令
- head 命令
- tail 命令
- more 命令
- less 命令

我们先来介绍一个很重要的文件 `/etc/passwd`， 这个文件是 Linux/UNIX 系统中用于保存用户名和用户信息的文件。他是黑客们为了攻破你的系统一直惦记的文件。

我们先来看一下 `/etc/passwd` 文件的部分内容:


```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
... # 此处省略了很对行。
weimz:x:1000:1000:weimingze:/home/weimz:/bin/bash
sshd:x:122:65534::/run/sshd:/usr/sbin/nologin
postfix:x:123:125::/var/spool/postfix:/usr/sbin/nologin
weimingze:x:1001:1001:weimingze,,,:/home/weimingze:/bin/bash
```

此文件使用英文的冒号（`:`） 将文件分隔成了七列。下面我们详细说一下这七列都代表什么？

1. 第一列是用户名：用于登录（如:root、weimingze）。
2. 第二列是密码占位符：以前版本此处存放密码的哈希加密信息，现在用 `x` 代替。密码的哈希加密信息现在已经放在了文件 `/etc/shadow` 中。
3. 第三列是用户ID（UID）：0 一定是 root用户(超级用户)，1~999 为系统用户，1000及以上为普通用户（如: weimz 和 weimingze）。
4. 第四列是主组ID（GID）：在 Linux 中，一个用户可以在多个组中。但一个用户一定要有一个主组，而此整数代表主组的ID(即主组号)。
5. 第五列是用户全名或备注（用于登录时显示等，相当于很多论坛的昵称）。
6. 第六列是用户主文件夹（也称作主目录或家目录）的位置：用于存放此用户的数据文件，一般都放在 `/home/` 文件夹下。
7. 第七列是用户登录后所使用的 Shell，如：root 和 weimz 这几个用户都使用 `/bin/bash` 作为默认的 Shell。

> 关于 Shell 我们后面再讲。

3.1 cat 命令

作用：将文件的全部内容输出到屏幕终端上。

命令格式

```
cat 文件1 [文件2]
```

示例

显示 `/etc/passwd` 的全部内容。

```
weimingze@mzstudio:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
...
```

3.2 head/tail 命令

tail 命令

作用：将文件的末尾几行（默认是10行）内容输出到屏幕终端上。

命令格式

```
tail [-行数] 文件1
```

> 此处的行数必须是10进制的整数

示例

显示 `/etc/passwd` 的末尾 10 行。

```
weimingze@mzstudio:~$ tail /etc/passwd
polkitd:x:987:987:User for polkitd:/:/usr/sbin/nologin
rtkit:x:117:119:RealtimeKit,,,:/proc:/usr/sbin/nologin
colord:x:118:120:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/
nologin
gnome-initial-setup:x:119:65534::/run/gnome-initial-setup/:/bin/false
gdm:x:120:121:Gnome Display Manager:/var/lib/gdm3:/bin/false
nm-openvpn:x:121:122:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/
sbin/nologin
weimz:x:1000:1000:weimingze:/home/weimz:/bin/bash
sshd:x:122:65534::/run/sshd:/usr/sbin/nologin
postfix:x:123:125::/var/spool/postfix:/usr/sbin/nologin
weimingze:x:1001:1001:weimingze,,,:/home/weimingze:/bin/bash
```

显示 `/etc/passwd` 的末尾 5 行。

```
weimingze@mzstudio:~$ tail -5 /etc/passwd
nm-openvpn:x:121:122:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/
sbin/nologin
weimz:x:1000:1000:weimingze:/home/weimz:/bin/bash
sshd:x:122:65534::/run/sshd:/usr/sbin/nologin
postfix:x:123:125::/var/spool/postfix:/usr/sbin/nologin
weimingze:x:1001:1001:weimingze,,,:/home/weimingze:/bin/bash
```

head 命令

作用：将文件的前面几行（默认是10行）内容输出到屏幕终端上。

用法同 tail 一样。

命令格式

```
head [-行数] 文件1
```

> 此处的行数必须是10进制的整数

示例

显示 `/etc/passwd` 的前面 10 行。

```
weimingze@mzstudio:~$ head  /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
```

显示 `/etc/passwd` 的前面 5 行。

```
weimingze@mzstudio:~$ head -5 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
```

3.3 more 命令

作用：在终端中分页查看文件内容，用于较大文件的查看。

命令格式

```
more 文件1
```

示例

查看 `/etc/passwd` 的内容。

```
weimingze@mzstudio:~$ more /etc/passwd
...
```

此时如果文件比较小，就会一次性显示完成。如果文件比较大，会显示当前页面所占的百分比。

操作的快捷方式如下：

按键	说明
回车	向下翻一行。
空格	向下翻一页。
q	退出显示。

然后使用 `回车` 键可以向下翻一行，使用 `空格` 可以翻一页。使用 `q` 键可以退出显示。

3.4 less 命令

less 命令

less 命令用于在终端中分页查看文件内容，用于浏览较大文件内容。它支持 向前/向后翻页、搜索、跳转等操作，且不会一次性加载整个文件到内存中。

命令格式：

```
less 文件1
```

如:

```
weimingze@mzstudio:~$ less /etc/passwd
...
```

此时如果文件比较小，就会一次性显示完成。如果文件比较大，会显示当前页面所占的百分比。

操作的快捷方式如下：

按键	说明
空格 或 Page Down	向下翻一页。
b 或 Page Up	向上翻一页。
上、下 方向键	上下翻一行。
回车 键	向下翻一行。
g	跳转到第一行。
G	跳转到文件末尾。
q	退出显示。

动手试一试吧！

4. 文件夹的操作命令

文件夹（Folder）是文件系统中用于将文件分类存储的容器，他的内部可以存放文件和文件夹。

文件夹的作用就是用来管理文件，方便查找。

本节课我们将学习在 Linux 下用来操作文件夹的命令的用法。

本节讲解的命令如下：

- pwd 命令
- cd 命令
- mkdir 命令
- rmdir 命令
- du 命令

，如： pwd、cd、mkdir、rmdir、du 命令

这些命令在 UNIX 系统中也一样可用。

前面我们已经讲过 Linux 文件系统是一个树形结构。当然我们打开一个终端，要对 Linux 进行操作时，这个终端一定在 文件系统中的某个文件夹内。我们把这个文件夹叫做 `当前工作目录(Current Working Directory)`，起初这个目录通常是此用户的家目录，如 weimingze 这个用户的家目录通常在 `/home/weimingze`。

4.1 pwd 命令

pwd 命令用来打印当前的工作路径（ Print Working Directory），即当前所在的位置文件夹的绝对路径。

示例

打印当前的工作路径。

```
weimingze@mzstudio:~$ pwd
/home/weimingze
```

4.2 cd 命令

cd（Change Directory）命令的作用是用于在终端中切换当前工作路径。

> 几乎所有操作系统（Linux、macOS、Windows）都支持此命令。

命令格式

```
cd [目标文件夹的路径]
```

如果不给出 `目标文件夹的路径` 默认回到此用户的用户主目录（`~`）。

如:

切换到 根 文件夹：

```
weimingze@mzstudio:~$ pwd
/home/weimingze
weimingze@mzstudio:~$ cd /
weimingze@mzstudio:/$ pwd
/
weimingze@mzstudio:/$ cd
weimingze@mzstudio:~$ pwd
/home/weimingze
```

路径是描述一个文件或文件夹位置的字符串。

路径分为两种:

- 绝对路径：指从 根 `/` 开始的路径，如：`/home/weimingze`。
- 相对路径：指从 根 以外的其他文件夹开始的路径，相对路径的第一个字符不是 `/`, 如：`home`，相对路径要表示的具体位置要根据 `当前工作目录` 来确定。

比如要再次进入 `/home` 文件夹，然后进入 `/home/weimingze` 这个文件夹。

```
weimingze@mzstudio:/$ pwd
/
weimingze@mzstudio:/$ cd home
weimingze@mzstudio:/home$ pwd
/home
weimingze@mzstudio:/home$ ls
weimingze  weimz
weimingze@mzstudio:/home$ cd weimingze
weimingze@mzstudio:~$ pwd
/home/weimingze
```

特殊路径表示法

路径符号	说明
`.`	用于表示当前文件夹。
`..`	用于表示上一级文件夹。
`~`	在终端中表示当前用户的 `主目录`。
`-`	表示切换到此工作路径前的那个工作路径。

示例

```
weimingze@mzstudio:~$ cd
weimingze@mzstudio:~$ cd /etc  # 进入到 /etc 文件夹
weimingze@mzstudio:/etc$ pwd
/etc
weimingze@mzstudio:/etc$ cd ..  # 进入到 / 文件夹
weimingze@mzstudio:/$ pwd
/
weimingze@mzstudio:/$ cd -  # 回到 /etc 文件夹
/etc
weimingze@mzstudio:/etc$ pwd
/etc
weimingze@mzstudio:/etc$ cd ~  # 回到 /home/weimingze 这个用户主文件夹
weimingze@mzstudio:~$ pwd
/home/weimingze
weimingze@mzstudio:~$ cd ../../etc  # 又回到 /etc 文件夹
weimingze@mzstudio:/etc$ pwd
```

```
/etc
weimingze@mzstudio:/etc$ cd .  # 还是 /etc 文件夹
weimingze@mzstudio:/etc$ cd ../home/./././weimingze  # 回到用户主目录 /home/
weimingze ,多个 "./" 也不影响位置定位。
weimingze@mzstudio:~$ pwd
/home/weimingze
```

> 注意 井号 （#） 开头一直到命令行末尾的内容是注释，会被忽略，不影响运行。

4.3 mkdir/rmdir 命令

mkdir 命令

mkdir（Make Directory）是用于在终端中创建新文件夹的命令。

命令格式

```
mkdir 新文件夹路径名1 [新文件夹路径名2 ...]
```

例如：在用户主目录（主文件夹）下创建 `aaa` 和 `bbb` 两个文件夹。

```
weimingze@mzstudio:~$ cd ~
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ mkdir aaa bbb
weimingze@mzstudio:~$ ls
aaa  bbb  hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$
```

常用选项

选项	说明
`-p`	递归创建：自动创建上一级文件夹（当上一级文件不存在时）

示例

在用户主目录 `/home/weimingze` 下创建一个 `/home/weimingze/ccc/ddd/eee/fff` 文件夹。

```
weimingze@mzstudio:~$ cd ~
weimingze@mzstudio:~$ pwd
/home/weimingze
weimingze@mzstudio:~$ mkdir -p ~/ccc/ddd/eee/fff
```

```
weimingze@mzstudio:~$ ls
aaa  bbb  ccc  hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ cd ccc
weimingze@mzstudio:~/ccc$ ls
ddd
weimingze@mzstudio:~/ccc$ cd ddd
weimingze@mzstudio:~/ccc/ddd$ ls
eee
weimingze@mzstudio:~/ccc/ddd$ cd eee/
weimingze@mzstudio:~/ccc/ddd/eee$ ls
fff
weimingze@mzstudio:~/ccc/ddd/eee$ cd fff
weimingze@mzstudio:~/ccc/ddd/eee/fff$ pwd
/home/weimingze/ccc/ddd/eee/fff
```shell

需要注意的是你在创建新文件夹时，新文件夹的位置一定是你有权限操作的位置，否则会提示：`Permission denied` 错误!

**rmdir 命令**

rmdir（Remove Directory）命令是用来删除空文件夹的命令。与 rm -r 不同，rmdir 仅能删除空文件夹，安全性更高，可防止误删非空文件夹。

**命令格式**

rmdir 文件夹路径名1 [文件夹路径名2 ...]

**例如：**

删除用户主目录下创建 `aaa` 和 `bbb` 两个空文件夹（我们刚创建的两个文件夹）。

```shell
weimingze@mzstudio:~$ ls
aaa  bbb  ccc  hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ rmdir ~/aaa ~/bbb
weimingze@mzstudio:~$ ls
ccc  hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
```

常用选项

选项	说明
`-p`	递归删除：如果上一级文件夹为空也删除

示例

删除用户主文件夹 `/home/weimingze` 下的 `/home/weimingze/ccc/ddd/eee/fff` 文件夹，同时删除它的上层空文件。

```
weimingze@mzstudio:~$ ls
ccc  hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ rmdir -p ccc/ddd/eee/fff/
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
```

可见 文件夹 `ccc/ddd/eee/fff/` 连同他的上级文件夹一同删除了。

如果你删除非空文件夹 可以使用 `rm -r [文件夹]` 命令来删除。

4.4 du 命令

du（Disk Usage）命令是一个用于显示文件或文件夹磁盘使用情况的命令。它可以帮助你列出此文件夹及内部文件共同占用磁盘空间，从而进行存储优化。

命令格式

```
du [选项] 文件夹路径名1
```

示例

显示用户主目录下每一个文件的磁盘占用情况。

```
weimingze@mzstudio:~$ pwd
/home/weimingze
weimingze@mzstudio:~$ du .
4   ./下载
8   ./.local/state/wireplumber
12  ./.local/state
4   ./.local/share/icc
16  ./.local/share/Trash/info
16  ./.local/share/Trash/files
36  ./.local/share/Trash
4   ./.local/share/flatpak/db
8   ./.local/share/flatpak
8   ./.local/share/gnome-shell
...
```

上述显示中，第一列数字是文件的磁盘占用情况，单位是kByte。 第二列是文件名。

常用选项

选项	说明
`-s`	仅显示总大小（不递归子文件夹）
`-m`	单位改为 mbyte
`-k`	单位改为 kbyte(默认)

示例

看一下当前用户主目录一共占用的磁盘空间是几兆字节(常用)。

```
weimingze@mzstudio:~$ du -sm ~
13  /home/weimingze
```

看一下当前用户主目录一共占用的磁盘空间是几K字节。

```
weimingze@mzstudio:~$ du -sk ~
13144   /home/weimingze
```

关于文件夹常用的命令我们就先讲到这里。

5. 文件查找命令

我们在计算机上工作时间长了就会积累大量的数据，这些数据大多都是以文件的形式进行存储。由于时间久了，我们经常会忘记这些文件保存在哪里，忘记有些内容存在了哪个文件当中。

这时候我们可以使用 find 和 grep 这两个常用的查找命令。

- find 命令：用于根据给定的文件名信息进行查找。
- grep 命令：用于根据给定的内容信息进行查找。

5.1 find 命令

find 命令用于在文件夹中递归搜索文件和内部的文件夹，基于文件名、路径、类型、大小、时间等属性进行查找。

他将直接操作文件系统，支持复杂的条件组合。

命令格式:

```
find 查找文件夹位置 [选项1 参数1] [选项2 参数2]
```

例如:

我们先创建一些测试文件。

```
weimingze@mzstudio:~$ mkdir -p myfiles/c
weimingze@mzstudio:~$ mkdir -p myfiles/cpp
weimingze@mzstudio:~$ mkdir -p myfiles/python
weimingze@mzstudio:~$ touch myfiles/c/main.c
weimingze@mzstudio:~$ touch myfiles/cpp/main.Cpp
weimingze@mzstudio:~$ touch myfiles/cpp/hello.cpp
weimingze@mzstudio:~$ touch myfiles/python/main_test.py
```

通过上述命令，我们形成这样一个文件结构。

```
myfiles/
├── c
│   └── main.c
├── cpp
│   ├── hello.cpp
│   └── main.Cpp
└── python
    └── main_test.py
```

示例

在当前文件夹下查找 `main.c` 的位置。

```
weimingze@mzstudio:~$ find . -name "main.c"
./myfiles/c/main.c
```

- `.` 代表查找当前文件夹
- `-name` 选项是根据文件名信息查找，后跟文件名信息，文件名信息建议使用英文的双引号（`"`）括起来。

`*` 和 `?` 号通配符

- `*` 代表 0个、1个或多个**任意**字符。
- `?` 代表 1个**任意**字符。

示例

```
weimingze@mzstudio:~$ find myfiles -name "*ai*"  # 查找文件名包含 "ai" 的文件，前后是什么都行
myfiles/cpp/main.Cpp
myfiles/c/main.c
myfiles/python/main_test.py
```

```
weimingze@mzstudio:~$ find myfiles -name "main*"  # 查找文件名以 "main" 开头的文件，后面是什么都行。
myfiles/cpp/main.Cpp
myfiles/c/main.c
myfiles/python/main_test.py
weimingze@mzstudio:~$ find myfiles -name "*.py"  # 查找文件名以 ".py" 结尾的文件。
myfiles/python/main_test.py
weimingze@mzstudio:~$ find myfiles -name "main.?"  # 查找文件名以 "main." 开头，文件名含有6个字符的文件。
myfiles/c/main.c
weimingze@mzstudio:~$ find myfiles -name "main.???"  # 查找文件名以 "main." 开头，文件名含有8个字符的文件。
myfiles/cpp/main.Cpp
```

另外文件名中还可以包含正则表达式, 正则表达式教程见：https://weimingze.com/re/，如:

查找所有 以 `.cpp`、`.CPP`等人不区分大小写的 `.cpp` 结尾的文件

```
weimingze@mzstudio:~$ find myfiles -name "*.[Cc][pP][pP]"
myfiles/cpp/main.Cpp
myfiles/cpp/hello.cpp
```

常用选项

选项	说明
`-name <文件名信息>`	根据文件名查找。
`-iname <文件名信息>`	根据文件名查找（不区分大小写）。
`-type <类型>`	类型: `d`是文件夹、`f` 普通文件、`l` 符号链接。
`-path <路径信息>`	如 `-path *lib*` 表示路径中包含"lib"的文件。
`-mtime <数字>`、`-atime <数字>`、`-ctime <数字>`	按修改时间（m）、访问时间（a）、状态变更时间（c）查找（单位：天）
`-size <大小信息>`	根据文件大小查找，（单位：c=字节，k=KB，M=MB，G=GB）

示例

```
weimingze@mzstudio:~$ find myfiles -type d  # 查找文件夹
myfiles
myfiles/cpp
myfiles/c
myfiles/python
```

```
weimingze@mzstudio:~$ find myfiles -type f  # 查找文件
myfiles/cpp/main.Cpp
myfiles/cpp/hello.cpp
myfiles/c/main.c
myfiles/python/main_test.py
weimingze@mzstudio:~$ find myfiles -type d -name "c*"  # 查找 以 c 开头的文件夹
myfiles/cpp
myfiles/c
weimingze@mzstudio:~$ find myfiles -iname "*.cpp"  # 查找所有以 '.cpp" 结尾的文件（不区分大小写）
./myfiles/cpp/main.Cpp
./myfiles/cpp/hello.cpp
weimingze@mzstudio:~$ find myfiles/c/ -mtime -7  # 查找 7天内修改的文件
myfiles/c/
myfiles/c/main.c
weimingze@mzstudio:~$ find myfiles/c/ -mtime +365  # 查找 一年前修改的文件
weimingze@mzstudio:~$ find myfiles/ -size -10c  # 查找文件长度小于10字节的文件
myfiles/cpp/main.Cpp
myfiles/cpp/hello.cpp
myfiles/c/main.c
myfiles/python/main_test.py
weimingze@mzstudio:~$ find . -size +10M        # 大于10MB的文件
```

5.2 grep 命令

grep 这个命令是英文 `Global Regular Expression Print` 的缩写。 用于在文件内容中搜索匹配的文本模式（字符串或正则表达式），并输出包含匹配模式的行。

命令格式:

```
grep [选项] "搜索模式" 文件名
```

常用选项

选项	说明	示例
`-n`	显示匹配行的**行号**	`grep -n "root" /etc/passwd`
`-r` / `-R`	**递归搜索**文件夹下的所有文件	`grep -r "hello" ~/`
`-i`	忽略大小写	`grep -i "hello" ~/hello.py`
`-l`	只输出**包含匹配模式的文件名**（不显示具体行）	`grep -l "main" *.c`
`-c`	统计匹配到的**行数**（不是次数）	`grep -c "warning" /var/log.txt`
`-w`	匹配**完整单词**（避免部分匹配）	`grep -w "world" ~/hello.py`

示例

```
weimingze@mzstudio:~$ grep "root" /etc/passwd  # 打印出 /etc/passwd 文件 含有 "root"的行
root:x:0:0:root:/root:/bin/bash
nm-openvpn:x:121:122:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
weimingze@mzstudio:~$ grep -n "root" /etc/passwd
# 打印出 /etc/passwd 文件 含有 "root"的行在行首现实行号。
1:root:x:0:0:root:/root:/bin/bash
48:nm-openvpn:x:121:122:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
weimingze@mzstudio:~$ grep "root" /etc/*  # 打印出 /etc下所有文件包含 "root" 的所有行.
grep: /etc/ModemManager: Is a directory
grep: /etc/NetworkManager: Is a directory
grep: /etc/PackageKit: Is a directory
grep: /etc/UPower: Is a directory
grep: /etc/X11: Is a directory
/etc/aliases:postmaster:    root
...
```

上述程序中 `grep: /etc/ModemManager: Is a directory` 是非正常的错误输出（也叫标准错误输出），这些错误输出没有使用价值。我们可以在末尾加 `2> /dev/null` 将标准错误重定向到 `/dev/null` 这个 `黑洞` 文件将其丢掉。这样就丢掉了错误信息。如:

```
weimingze@mzstudio:~$ grep "root" /etc/* 2> /dev/null  # 错误重定向后打印查找内容
/etc/aliases:postmaster:    root
/etc/anacrontab:HOME=/root
/etc/anacrontab:LOGNAME=root
/etc/bash.bashrc:# set variable identifying the chroot you work in (used in the
prompt below)
/etc/bash.bashrc:if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ]; th
en
...
```

我个人常用的是 `-nr` 选项，如：

我要查找我用户主文件夹下 所有包含 "hello world" 的文件内容，如下：

```
weimingze@mzstudio:~$ grep "hello world" -nr  ~/*
/home/weimingze/hello.py:1:print("hello world!")
```

这样我就可以找到 要查找的内容 在位于 `/home/weimingze/hello.py` 这个文件的第一行。

第三章、软件包管理

Ubuntu 系统为了减小安装后的磁盘体积，有些软件是默认没有安装，这些软件需要使用相应的命令去安装。如（nginx、openssh-server、tree等）。

1. apt 命令

apt 命令是 Ubuntu 系统 **高级打包工具**（Advanced Package Tool）。他是用于管理软件包的工具，如：安装、更新、卸载软件包等操作。它是 apt-get 和 apt-cache 等命令的现代化替代品，提供了更友好的交互体验（如进度条、颜色提示等）。

常用 apt 的子命令及参数

以下是 `apt` 的主要子命令和常用参数：

命令	功能描述
`apt update`	更新软件包索引（从服务器获取最新软件包列表）
`apt upgrade`	升级所有可升级的软件包（不删除旧包）
`apt full-upgrade`	升级并自动处理依赖冲突（可能删除旧包）
`apt install <包名>`	安装指定软件包
`apt remove <包名>`	卸载软件包（保留配置文件）
`apt purge <包名>`	彻底卸载软件包（删除配置文件）
`apt autoremove`	自动删除不再需要的依赖包
`apt search <关键词>`	搜索软件包
`apt show <包名>`	显示软件包的详细信息（版本、依赖等）
`apt list`	列出所有可用软件包
`apt list --installed`	列出已安装的软件包
`apt edit-sources`	编辑软件源配置文件（`/etc/apt/sources.list`）

常用选项

参数	说明
`-y` 或 `--yes`	自动回答“是”（用于脚本中跳过确认提示）
`--no-upgrade`	安装时不升级已存在的包
`--only-upgrade`	仅升级已安装的包，不安装新包
`-q` 或 `--quiet`	静默模式（减少输出信息）

示例

1. 安装 nginx 服务：

```
weimingze@mzstudio:~$ sudo apt install nginx
[sudo] password for weimingze:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  nginx-common
Suggested packages:
  fcgiwrap nginx-doc
The following NEW packages will be installed:
  nginx nginx-common
0 upgraded, 2 newly installed, 0 to remove and 117 not upgraded.
Need to get 551 kB of archives.
After this operation, 1596 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
...
```

2. 彻底卸载 nginx 服务：

```
sudo apt purge nginx
```

apt 换源

apt 安装软件时，软件的存储位置默认是 Ubuntu 官方的网站，网速比较慢。这种情况可以使用 apt 换源来解决。

详见附录 Ubuntu 换源

2. tree 命令

tree 命令用于以将一个文件夹以树状图形的方式更直观地展示出来的命令。

Ubuntu 24.04 默认已经不在安装这个命令，需要使用 apt 命令进行安装。

tree 命令安装命令

```
sudo apt install tree
```

命令格式

```
tree [选项] [路径]
```

命令说明

- 不给出 **路径** 默认显示当前工作路径。

示例:

```
weimingze@mzstudio:~$ tree
.
├── snap
│   ├── firefox
│   │   ├── 5751
│   │   ├── common
│   │   └── current -> 5751
│   ├── firmware-updater
│   │   ├── 167
│   │   ├── common
│   │   └── current -> 167
│   └── snapd-desktop-integration
│       ├── 253
│       │   ├── Desktop
│       │   ├── Documents
│       │   ├── Downloads
│       │   ├── Music
│       │   ├── Pictures
│       │   ├── Public
│       │   ├── Templates
│       │   └── Videos
│       ├── common
│       └── current -> 253
├── 下载
├── 公共
├── 图片
├── 文档
├── 桌面
├── 模板
├── 视频
└── 音乐
```

常用选项：

选项	说明
`-a`	显示所有文件，包括隐藏文件（以点开头的文件）
`-d`	只显示目录
`-f`	显示文件的完整路径
`-L <n>`	限制目录树的显示深度为 n 层
`-P <pattern>`	只显示匹配指定模式的文件名
`-I <pattern>`	排除匹配指定模式的文件名
`-o <filename>`	将结果输出到文件

示例

显示根目录下所有的文件夹，最大深度为 1

```
weimingze@mzstudio:~$ tree -L 1 -d /
/
├── bin -> usr/bin
├── bin.usr-is-merged
├── boot
├── cdrom
├── dev
├── etc
├── home
├── lib -> usr/lib
├── lib.usr-is-merged
├── lib64 -> usr/lib64
├── lost+found
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin -> usr/sbin
├── sbin.usr-is-merged
├── snap
├── srv
├── sys
├── tmp
├── usr
└── var
```

第四章、文件归档

归档（Archiving） 指将 不再频繁使用但仍需保留的数据、文件或软件包 转移到专门的存储位置。

归档的目的如下：

1. 释放主系统资源（如硬盘空间等）。
2. 长期保存（满足合规性或历史记录需求）。
3. 分类管理（区分活跃数据和历史数据等）。
4. 方便分发（传递给其他计算机）。

归档通常使用数据压缩算法，常用的数据压缩算法按是否能够还原成原始数据大致可以分为如下两类：

1. 有损压缩：压缩后内容会丢失，无法还原成原始数据，如：`.mp3`、`.jpg`、`.mp4` 等文件的压缩。
2. 无损压缩：数据压缩后还可以还原成原始数据，如：`.png`、`.zip`、`.gz`、`.xz`等文件的压缩。

本章我们主要讲解**无损压缩**算法相关的命令。

Linux 下常用的归档相关的命令如下：

- gzip/gunzip 命令
- zip/unzip 命令
- xz/unxz 命令
- tar 命令

先来说一下数据压缩。在 Linux/UNIX 在常用的 zip 压缩有两种，一种就是 GNU 的 zip 压缩算法(文件后缀名 `.gz`)，另一种是普通的 zip 压缩算法(文件后缀名 `.zip`)。两种压缩算法同为 zip 却不兼容。

> GUN 是一个项目，是由 Richard Stallman 于 1983 年发起的，目标是创建一个完全自由的类 Unix 操作系统。 GNU 全称：GNU's Not Unix（GUN是递归拼写，表示“GNU 不是 Unix”）。这个项目开发了大量的工具软件，如: gcc、g++编译器，emacs编辑器以及我们这节课学的 gzip/gunzip等。

GUN zip 压缩算法相关的命令有 gzip 和 gunzip。

普通 zip 压缩算法相关的命令有 zip 和 unzip。

1. gzip/gunzip 命令

这两个命令主要适用于 GUN zip 压缩算法的压缩和解压缩的命令，通常只用于 Linux/UNIX 系统中。

gzip 和 gunzip 都只能对文件进行压缩和解压缩，不能对文件夹进行操作。gzip 用于压缩文件，gunzip 用于解压缩文件。如果需要将文件夹打包再压缩，需要结合 tar 命令一起使用。

gzip命令

作用

使用 GNU Zip 算法压缩文件（生成 .gz 文件），这个命令只能对文件进行压缩，不能对文件夹进行操作。

命令格式:

```
gzip [选项] 文件名
```

压缩后，原始文件会被替换为 文件名.gz（默认删除原文件）。

例如:

```
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ gzip hello.py
weimingze@mzstudio:~$ ls
hello.py.gz  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
```

hello.py.gz 是 压缩 hello.py 后生成的文件，原来的 hello.py 被删除了

> 需要注意的是对于比较小的文件压缩后文件的总体体积变大了，因此较小的单个文件没有必要压缩。

gzip常用选项

选项	说明
`-d`	解压文件（等同于 `gunzip`）
`-k`	保留原始文件（不删除）
`-v`	显示压缩/解压的详细信息
`-1` 到 `-9`	压缩级别（`-1` 最快，`-9` 最高压缩率）
`-c`	输出到标准输出（不修改原文件）
`-t`	测试压缩文件的完整性

解压缩可以使用 `gzip -d` 如：

```
weimingze@mzstudio:~$ ls
hello.py.gz  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ gzip -d hello.py.gz
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
```

gunzip命令

作用

对使用 gzip 命令压缩的 `.gz` 文件进行解压缩操作。其实此命令等同于 `gzip -d` 命令和参数的组合。

命令格式:

```
gunzip [选项] 文件名.gz
```

解压缩后，`xxx.gz` 文件的文件名 xxx 会被作为生成的文件，原文件 `xxx.gz`被删除。

如:

```
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ gzip hello.py
weimingze@mzstudio:~$ ls
hello.py.gz  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ gunzip hello.py.gz
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
```

2. zip/unzip 命令

zip 和 unzip 命令用于将多个文件或文件夹归档打包成一个文件并用普通 zip 压缩算法进行压缩和解压缩。

这种 zip 压缩算法比较通用，使用 普通zip 压缩算法打包的 `.zip` 压缩文件几乎各种操作系统（包括 Windows、Linux 和 MacOS）都能够压缩了解压缩。是非常通用的压缩格式。

zip命令

作用

将文件或文件夹用普通的 zip 压缩算法进行打包并压缩，生成后缀名为 `.zip` 的归档文件。zip 命令支持将文件和文件夹一起打包压缩。

命令格式:

```
zip [选项] 压缩包名.zip 文件或文件夹1 [文件或文件夹2]
```

例如:

```
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ zip myfiles.zip hello.py snap/
  adding: hello.py (stored 0%)
  adding: snap/ (stored 0%)
weimingze@mzstudio:~$ ls
hello.py  myfiles.zip  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
```

此时的 myfiles.zip 文件是一个文件 hello.py 和一个文件夹 snap(不包含文件夹内部的文件和文件夹) 打包并压缩的归档文件。

常用选项

选项	说明
`-r`	递归压缩文件夹（包含子文件夹）
`-q`	静默模式（不显示输出信息）
`-e`	加密压缩（会提示输入密码）
`-m`	压缩后删除原文件
`-x <文件>`	排除指定文件不压缩
`-<压缩级别>`	1-9（1最快压缩，9最佳压缩，默认6）

如:

```
weimingze@mzstudio:~$ rm myfiles.zip
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ zip -r myfiles.zip hello.py snap/
... 此处省略了压缩过程的打印输出。
weimingze@mzstudio:~$ ls
hello.py  myfiles.zip  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
```

此时的 myfiles.zip 文件是 一个文件 hello.py 和 文件夹 snap 打包并压缩的归档文件，snap文件夹内部的文件和文件夹也一同打包了。

unzip命令

作用

用于解压缩用普通 zip 压缩算法进行压缩的压缩包。

命令格式:

```
unzip [选项] 压缩包名.zip
```

示例

```
weimingze@mzstudio:~$ ls
hello.py  myfiles.zip  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ cp myfiles.zip 桌面/
weimingze@mzstudio:~$ cd 桌面/
weimingze@mzstudio:~/桌面$ ls
myfiles.zip
weimingze@mzstudio:~/桌面$ unzip myfiles.zip
```

```
Archive:  myfiles.zip
 extracting: hello.py
   creating: snap/
   creating: snap/firmware-updater/weimingze@mzstudio:~/桌面$ ls
   ... 此处省略了很多行。
weimingze@mzstudio:~/桌面$ ls
hello.py  myfiles.zip  snap
```

常用选项

选项	说明
`-d <路径>`	指定解压到目标路径
`-l`	仅列出压缩包内容（不解压）
`-o`	强制覆盖已存在的文件（不提示）
`-q`	静默解压（不显示输出）
`-P <密码>`	直接指定密码（不安全，建议交互输入）

通用 zip 和 GUN zip 算法对比

Linux命令	`zip`	`gzip`
压缩格式	`.zip`	`.gz`
来源	Windows/Linux 通用	Linux/Unix 原生工具
是否支持文件夹	是（自动递归）	否（需先用 `tar` 打包）
多文件处理	支持（直接压缩多个文件）	需结合 `tar`
是否删除原文件	保留原文件	默认删除（`-k` 可保留）
压缩率	较低	中等（比 `.zip` 略高）
常用场景	跨平台文件交换	Linux 系统日志压缩

3. xz/unxz 命令

xz 命令是数据压缩的命令，是基于 LZMA（Lempel-Ziv-Markov Chain Algorithm） 算法的数据压缩格式进行数据压缩的命令。xz命令以其高压缩比和相对较慢的压缩/解压速度著称。

特点：

1. 对文本、日志等冗余数据，压缩比通常为比价高。对已压缩文件（如 ZIP、JPEG），压缩效果有限，甚至可能略微膨胀。
2. 高压缩比代价：压缩速度显著慢于 gzip 等算法
3. 常用于 Linux 软件包（如 .deb、.rpm）的的数据压缩。

xz 是用于数据压缩的命令，unxz 是用于解压缩的命令。

xz 和 unxz 都只能对文件进行压缩和解压缩，不能对文件夹进行操作压缩文件。如果需要将文件夹打包再压缩，需要结合 tar 命令一起使用。

xz命令

命令格式:

```
xz [选项] 文件名1 [文件名2]...
```

压缩后，原始文件会被替换为 `文件名.xz`（默认删除原文件）。

例如:

```
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ xz hello.py
weimingze@mzstudio:~$ ls
hello.py.xz  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
```

hello.py.xz 是 压缩 hello.py 后生成的文件，原来的 hello.py 被删除了

xz 常用选项

选项	说明
`-d`	解压文件（等同于 `unxz`）
`-k`	保留原始文件（不删除）
`-v`	显示压缩/解压的详细信息
`-1` 到 `-9`	压缩级别（`-1` 最快，`-9` 最高压缩率，默认 `-6`）
`-c`	输出到标准输出（不修改原文件）
`-t`	测试压缩文件的完整性

解压缩可以使用 `xz -d` 如：

```
weimingze@mzstudio:~$ ls
hello.py.xz  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ unxz hello.py.xz
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
```

unxz命令

unxz 命令对使用 xz 命令压缩的 `.xz` 文件进行解压缩操作。其实此命令等同于 `xz -d` 命令和参数的组合。

命令格式:

```
unxz [选项] 文件名1.xz [文件名2.xz]...
```

解压缩后，`xxx.xz` 文件的文件名 xxx 会被作为生成的文件，原文件 `xxx.xz`被删除。

如:

```
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ xz hello.py
weimingze@mzstudio:~$ ls
hello.py.xz  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ unxz hello.py.xz
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
```

4. tar 命令

tar 是 Linux/Unix 系统中用于 打包（归档） 和 压缩/解压 文件的命令。tar 命令可以将多个文件或文件夹打包成为一个单独的 `.tar` 文件，但并不压缩。

tar 命令可以使用选项结合其他压缩工具（如 gzip、bzip2、xz等）在归档的同时生成压缩归档文件（如 `.tar.gz`、`.tar.bz2`、`.tar.xz`等格式）。

tar 命令可以通过选项解压缩用 tar 命令归档的包，提取包内的数据文件。

命令格式:

```
tar [选项] [输出文件名] [输入文件/文件夹]
```

常用选项

选项	作用
`-c`	**创建**新的归档文件
`-x`	**解压**归档文件
`-f <文件名>`	指定归档文件名（必须放在最后）
`-v`	显示详细过程（verbose）
`-z`	使用 `gzip` 压缩/解压（`.tar.gz` 或 `.tgz`）
`-j`	使用 `bzip2` 压缩/解压（`.tar.bz2`）
`-J`	使用 `xz` 压缩/解压（`.tar.xz`）
`-t`	查看归档文件内容（不解压）
`-r`	向归档文件追加文件
`-C` 路径	解压到指定文件夹

示例:

使用 tar 命令将文件 hello.py 和 snap 文件夹打包为 `.tar` 文件。但并不压缩。

```
weimingze@mzstudio:~$ ls
hello.py  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ tar -cf myfiles.tar hello.py snap/
weimingze@mzstudio:~$ ls
```

```
hello.py  myfiles.tar  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ ls -l myfiles.tar
-rw-rw-r-- 1 weimingze weimingze 583680 May 19 16:23 myfiles.tar
```

使用 tar命令将 文件 hello.py 和 snap 文件夹打包为 `.tar.gz` 文件。并使用 gzip 压缩。

```
weimingze@mzstudio:~$ ls
hello.py  myfiles.tar  snap  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ tar -czf myfiles.tar.gz hello.py snap/
weimingze@mzstudio:~$ ls
hello.py     myfiles.tar.gz  下载  图片  桌面  视频
myfiles.tar  snap            公共  文档  模板  音乐
weimingze@mzstudio:~$ ls -l myfiles.tar.gz myfiles.tar
-rw-rw-r-- 1 weimingze weimingze 583680 May 19 16:23 myfiles.tar
-rw-rw-r-- 1 weimingze weimingze  40091 May 19 16:25 myfiles.tar.gz
```

从上述程序中可以看出 myfiles.tar.gz 要比 myfiles.tar 小很多。但是他们的内容是一样的。

使用 tar 命令将上述生成的 `.tar.gz` 文件解压缩到 `桌面` 文件夹下。

```
weimingze@mzstudio:~$ ls 桌面/
weimingze@mzstudio:~$ tar -xzf myfiles.tar.gz -C 桌面/
weimingze@mzstudio:~$ ls 桌面/
hello.py  snap
```

> 注意: 如果不写 -C 选项，则默认会将文件解压缩到当前文件夹。

下面我们尝试一下各种压缩算法

同上述打包一样，使用 tar 命令将 文件 hello.py 和 snap 文件夹打包为归档文件。

使用 xz 压缩为 `.tar.xz`。

```
weimingze@mzstudio:~$ tar -cJf myfiles.tar.xz hello.py snap/
```

使用 bzip2 压缩为 `.tar.bz2`。

```
weimingze@mzstudio:~$ tar -cjf myfiles.tar.bz2 hello.py snap/
/bin/sh: 1: bzip2: not found
tar: myfiles.tar.bz2: Cannot write: Broken pipe
tar: Child returned status 127
tar: Error is not recoverable: exiting now
weimingze@mzstudio:~$ sudo apt install bzip2
[sudo] password for weimingze:
weimingze@mzstudio:~$ sudo apt install bzip2
[sudo] weimingze 的密码:
正在读取软件包列表... 完成
```

```
正在分析软件包的依赖关系树... 完成
正在读取状态信息... 完成
...  #  此处省略了安装过程。
weimingze@mzstudio:~$ tar -cjf myfiles.tar.bz2 hello.py snap/
weimingze@mzstudio:~$ ls -l myfiles.tar myfiles.tar.gz myfiles.tar.bz2 myfiles.t
ar.xz
-rw-rw-r-- 1 weimingze weimingze 583680 May 19 16:23 myfiles.tar
-rw-rw-r-- 1 weimingze weimingze  25199 May 19 16:50 myfiles.tar.bz2
-rw-rw-r-- 1 weimingze weimingze  40091 May 19 16:25 myfiles.tar.gz
-rw-rw-r-- 1 weimingze weimingze  25032 May 19 16:39 myfiles.tar.xz
weimingze@mzstudio:~$ ls -l myfiles.tar*
-rw-rw-r-- 1 weimingze weimingze 583680 May 19 16:23 myfiles.tar
-rw-rw-r-- 1 weimingze weimingze  25199 May 19 16:50 myfiles.tar.bz2
-rw-rw-r-- 1 weimingze weimingze  40091 May 19 16:25 myfiles.tar.gz
-rw-rw-r-- 1 weimingze weimingze  25032 May 19 16:39 myfiles.tar.xz
```

> 此处会报告 bzip2 这个命令没有找到这样一个错误。如果你想尝试使用这个命令，需要你使用 `sudo apt install bzip2` 这个命令安装 `bzip2` 这个命令。Ubuntu 24.04 已经不再预安装这个高不成低不就 的 bzip2 命令了。

上述我们在使用 tar 命令打包是分别使用了 `-z`、`-j` 和 `-J` 压缩成了不同格式的文件。通过 `ls -l` 我们可以看出 没有压缩的 `myfiles.tar` 最大，使用 `-J` 选择压缩的 `myfiles.tar.xz` 文件最小。

在使用 tar 命令打包时如果打包使用了 `-z` 选项，那么解包时也必须使用 `-z` 选项。

`.tar.xz` 使用 xz 压缩算法压缩的文件。这是我目前看到的最优秀的无损压缩算法。现在 Linux 内核 普遍使用 xz 算法压缩。Linux 6.14.7 内核源码的压缩文件包 `linux-6.14.7.tar.xz` 才 149.5M 大小，解压缩后为 1622M，压缩比超过了 10比1。 超赞！

上述示例中 我在命令行中使用了 星号 `*` 号通配符，即 `*` 可以代表 0 或多个任意字母。所以 `ls -l myfiles.tar*` 和 `ls -l myfiles.tar myfiles.tar.gz myfiles.tar.bz2 myfiles.tar.xz` 得到了相同的效果。

tar压缩格式对比

格式	命令选项	压缩率	速度	常见扩展名
不压缩	`-cvf`	无	最快	`.tar`
gzip	`-czvf`	中等	快	`.tar.gz` / `.tgz`
bzip2	`-cjvf`	较高	较慢	`.tar.bz2`
xz	`-cJvf`	最高	最慢	`.tar.xz`

第五章、文本编辑器

文本编辑器是指用来编写纯文字信息的文本文件编辑器。文本编辑器是开发人员和运维人员最常用的编辑器，通常我们用文本编辑起来书写各种编程语言的代码和配置文件。

从文本文件编辑器的显示类型来看，可以大致分为两种：

1. 基于图形用户界面的编辑器(需要桌面 GUI 运行环境)，如：gedit、Visual Studio Code、PyCharm等。
2. 基于终端的文本编辑器（在终端中运行，可以远程编辑文件），如：nano、vi/vim、emacs等。

这里我们主要学习基于终端的文本编辑器的用法。

1. nano 编辑器

nano 编辑器是终端下的简易编辑器，他的程序占用空间小，容易上手。nano适合用于嵌入式开发和运维等领域。

启动命令：nano

命令格式:

```
nano [选项] [文件名]
```

一般在运行时我们不用任何选项，后面跟了文件名则编辑此文件，后面不跟随文件名，则先编辑，编辑完后在保存。

例如

```
weimingze@mzstudio:~$ nano hello.py  # 编辑 hello.py 这个文件
```

界面如下:

nano 编辑器最下面的两行是操作菜单，菜单都要通过组合键来进行操作。

- `^` 等同于 `Control` 键。
- `M` 等同于 `Alt` 键(Mac上对应 `Option` 键)，对于 MacOS 用户来说，M 也等同于 `Esc` 键。

`^X` 表示按键 `Control + x` 表示退出 nano 编辑器。

`M-U` 则表示按键 `Alt + u` 表示 撤销 之前操作，在 MacOS 系统下也可以先按一下 `ESC` 按键，再按一下 `u` 按键来执行该菜单命令。

常用快捷键如下：

快捷操作	说明
^G	帮助
^X	离开
^O	写入
^R	读档
^W	搜索
^\	替换
^K	剪切
^U	粘贴
^T	执行命令
^J	对齐
^C	位置
^/	跳行
M-U	撤销
M-E	重做
M-A	设置标记
M-6	复制
M-]	至括号
^Q	向前搜索
M-Q	上一个
M-W	下一个
^B	向后
^F	向前
^◂	前一个字
^▸	后一个字
^A	顶端
^E	尾端

`^P`	上行
`^N`	下行

如果上述快捷键没有你要的操作，你也可以使用 `Control + g` 来查看帮助文档来了解所有的操作。

执行了上述菜单命令后，根据提示就可以完成相应的操作了。

2. vi/vim 编辑器

vi 是 Unix/Linux 系统中基于终端的最经典的文本编辑器，诞生于 1976 年（由 Bill Joy 开发）。

Vim（Vi IMproved）是 vi 的增强版， 于 1991 年发布（由 Bram Moolenaar 开发）。

Vim 完全兼容 vi 编辑器，在 vi 的基础上扩充了功能。

特点：

- vi 体积小，功能强大，几乎所有的系统都预装 vi编辑器，是编辑配置文件，编写 Shell 脚本的首选，尤其在服务器环境中普遍存在。
- Vim 功能更强，在 vi 的基础上还支持语法高亮，代码折叠，多重撤销等功能。
- Vim 支持各种插件来扩展功能。如：代码补全。Git 集成等。
- Vim 在 Linux/UNIX（MacOS）系统中可用，在 Windows 系统中也可以安装基于图形用户界面(GUI)的版本(`GVim`)。
- vi / Vim 学习难度比 nano 高，但他是嵌入式开发、服务器运维，Linux/UNIX 下 C/C++ 开发人员的首选。

Ubuntu 24.04 没有预装 Vim 编辑器，需要使用 apt 命令安装，安装命令如下:

```
sudo apt install vim
```

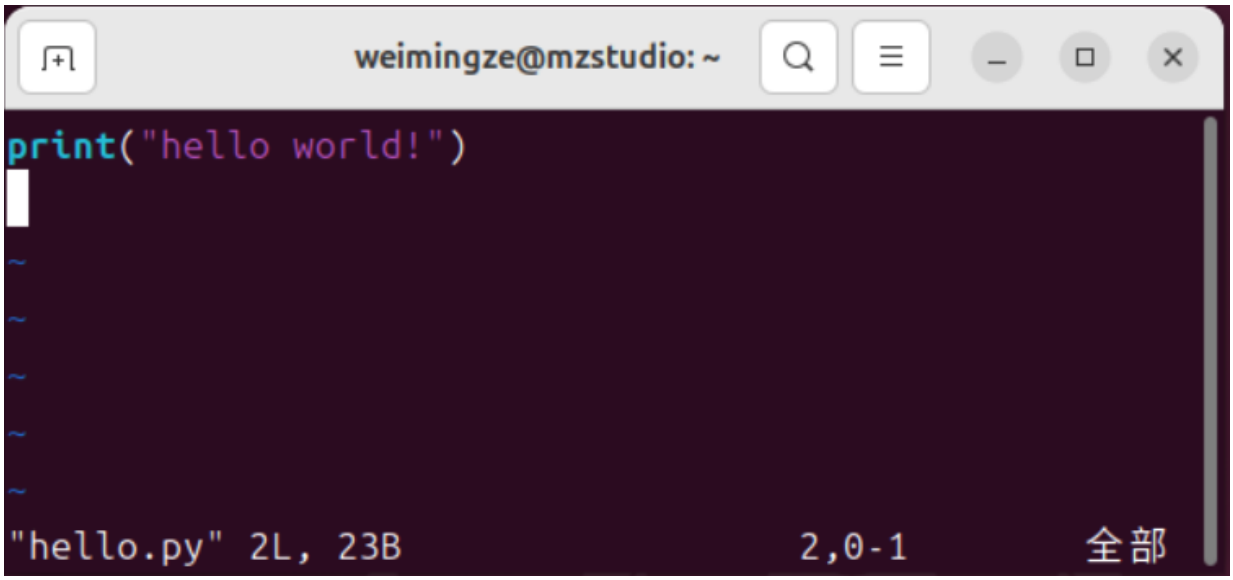
vi/Vim 启动命令:

```
vi [文件名]
vim [文件名]
```

如:

```
weimingze@mzstudio:~$ vim hello.py
```


vi 编辑器界面如下:



左下角的 `"hello.py"` 是正在编辑的文件的名字。

文件名后的 `2L,23B` 是指你正在编辑的这个文件共 2 行，共 23 字节。

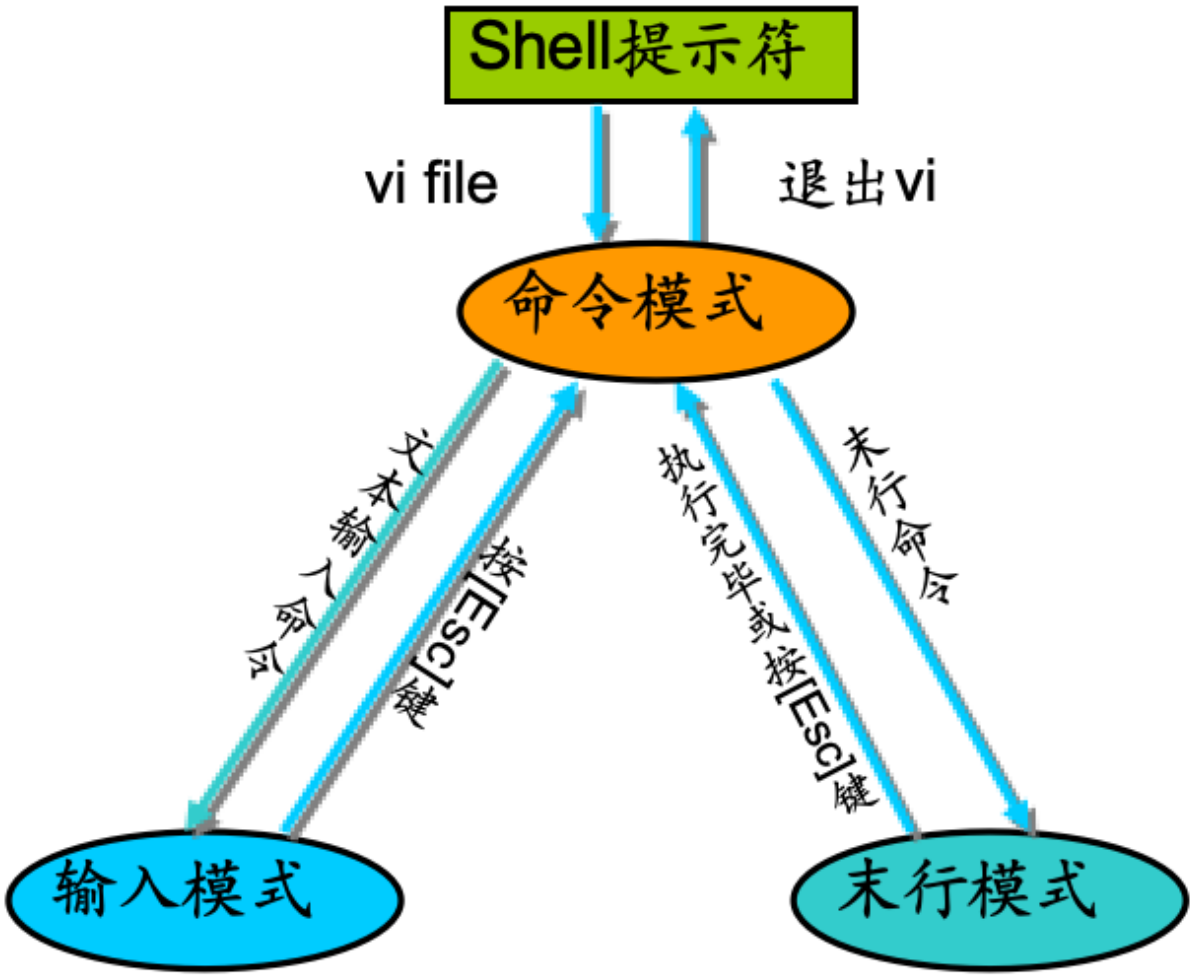
后面的 `2,0-1` 是指示当然光标所在的位置，

右下角的 `全部` 指当前向内容已经全部显示。

vi/Vim 编辑器的三种模式

- 命令模式
- 输入模式
- 末行模式

刚进入 vi 时是在 `命令模式` 下，如图:



在 命令模式 下，你的任何按键都是命令，不能输入内容。

我们先来学习如何退出 vi/Vim。

在命令模式下输入英文的冒号(:)，这个冒号会出现在左下角的最后一行，此时进入末行模式:

```
print("hello world!")

~
~
~
:
```

此时输入 `:q!` 后 回车 则不保存退出 vi。输入 `:wq` 后 回车 则保存后退出 vi，如果在末行模式下 按下 ESC 键则回到命令模式。

重新进入 vi，我们再来说一下 编辑模式。在 命令模式 下，输入 `i`、`a`、`o`中的任意一个按键则进入 编辑模式，在编辑模式下，你输入的任何按键都是文档的内容，当你编辑完成后需要按 ESC 键返回到 命令模式 才能进行保存退出。

尝试一下吧！至此你已经学会了 vi/Vim 的基本用法。

下面我们来学习他的高级用法。

进入编辑模式的命令

命令	说明
`i`	在当前光标处进入插入状态
`a`	在当前光标后进入插入状态
`A`	将光标移动到当前行的行末，并进入插入状态
`o`	在当前行的下面插入新行，光标移动到新行的行首，进入插入状态
`O`	在当前行的上面插入新行，光标移动到新行的行首，进入插入状态
`cw`	删除当前光标到所在单词尾部的字符，并进入插入状态
`c$`	删除当前光标到行尾的字符，并进入插入状态
`c^`	命令删除当前光标之前（不包括光标上的字符）到行首的字符，并进入插入状态

进入插入状态后，左下角会显示 `-- INSERT --` 提示，如:

```
print("hello world!")

~
~
-- INSERT --              1,22          All
```

> 在编辑模式下编辑 C 语言程序时，使用快捷键 `Control + n` 可以调出自动补全功能。这个是我用了很多年的功能，咱们可以试试。

我常用的末行命令

在命令模式下，输入英文的冒号就进入末行模式了。末行模式需要用回车键进行确认。按 `ESC` 则取消末行模式，返回到命令模式。以下是我用过的末行模式。

命令	说明
`:w`	保存当前编辑。
`:w 文件路径名`	另存为新文件。
`:wq`	保存修改后退出。
`:q`	退出vi（如果文档没有修改）。
`:q!`	不保存修改，强制退出。
`:e 其他文件路径名`	在当前编辑器打开其他文件。
`:e! 其他文件路径名`	在当前编辑器打开其他文件，放弃之前的修改。
`:e#`	回到上次打开的文件。
`:r 其他文件路径名`	读取文件内容到当前vi编辑器中。
`:! 命令`	执行命令并显示结果。回车键返回到 vi。
`:r! 命令`	执行命令并将结果添加到当前vi编辑器中。
`:set nu`	显示行号。
`:set nonu`	取消显示行号。
`:数字`	光标跳转到指定的行，如 `:20` 跳转到第 20 行。
`:sp` 或 `:sp 其他文件路径名`	水平分割当前窗口，在命令模式下使用 `Control + w + w` 可以在窗口内切换光标。
`:vs` 或 `:vs 其他文件路径名`	垂直分割当前窗口，同上，可以使用 `Control + w + w` 切换光标。

> `:sp` 和 `:vs` 可以将窗口无限细分，如果你的电脑屏幕足够大，那真是太爽了。

我就想起这么多，以后想起来的再补充。

下面我来说一下命令模式下的命令。

在任何模式下，只要按下 `ESC` 键就会回到命令模式。

命令模式下常用的命令

命令模式下光标相关的命令：

命令	说明
h 或 左方向键(←)	向左移动光标
l 或 右方向键(→)	向右移动光标
k 或 上方向键(↑)	向上移动光标
j 或 下方向键(↓)	向下移动光标
Control + f 或 Page Up	向前翻整页
Control + b 或 Page Down	向后翻整页
Control + u	向前翻半页
Control + d	向后翻半页
^ 或 Home	快速定位光标到行首
$ 或 End	快速定位光标到行尾
w	将光标快速跳转到当前光标所在位置的后一个单词的首字母
b	将光标快速跳转到当前光标所在位置的前一个单词的首字母
e	将光标快速跳转到当前光标所在位置的后一个单词的尾字母
gg 或 1G	跳转到文件的首行
G	跳转到文件的末尾行

命令模式下删除、复制、粘贴相关的命令：

命令	说明
`x`	删除光标处的单个字符。
`dd`	删除光标所在行。
`dw`	删除当前字符到单词尾（包括空格）的所有字符。
`de`	删除当前字符到单词尾（不包括单词尾部的空格）的所有字符。
`d$`	删除当前字符到行尾的所有字符。
`d^`	删除当前字符到行首的所有字符。
`J`	删除光标所在行行尾的换行符，相当于合并当前行和下一行的内容。
`yy`	复制当前行整行的内容到vi缓冲区。
`yw`	复制当前光标到单词尾字符的内容到vi缓冲区。
`y$`	复制当前光标到行尾的内容到vi缓冲区。
`y^`	复制当前光标到行首的内容到vi缓冲区。
`p`	读取vi缓冲区中的内容，并粘贴到光标当前的位置（不覆盖文件已有的内容）。

> 删除命令删除的内容都会进入`vi缓冲区`，可以使用 `p` 命令进行粘贴。

命令模式下撤销/重做操作

命令	说明
`u`	取消最近一次的操作，并恢复操作结果可以多次使用u命令恢复已进行的多步操作。
`U`	取消对当前行进行的所有操作。
`Control + r`	对使用u命令撤销的操作进行恢复。

字符串查找操作

在命令模式下用 `/` 或 `?` 进入末行模式进行查找。

命令	说明
/查找字符串	从上而下在文件中查找字符串"查找字符串"。
?查找字符串	从下而上在文件中查找字符串"查找字符串"。
n	定位下一个匹配的被查找字符串。
N	定位上一个匹配的被查找字符串。

字符串替换操作

在命令模式下用 : 进入末行模式进行查找。

命令	说明
:%s/old/new/gc	在整个文件范围内替换所有的字符串"old"为"new"，提示：(y/n/a/q/l/^E/^Y)?，y替换当前，n不替换（跳过），a替换所有，q退出替换。l替换当前匹配后退出，^E(Control+E)向下滚动屏幕，^Y(Control+Y)向上滚动屏幕。
以下供了解	我不常用
:s/old/new	将当前行中查找到的第一个字符"old" 串替换为"new"。
:s/old/new/g	将当前行中查找到的所有字符串"old" 替换为"new"。
:#,#s/old/new/g	在行号"#,#"范围内替换所有的字符串"old"为"new"。
:%s/old/new/g	在整个文件范围内替换所有的字符串"old"为"new"。
:s/old/new/c	在替换命令末尾加入c命令，将对每个替换动作提示用户进行确认。

块操作

Vim 在命令模式下使用 ^V(Control + V) 可以进行块操作。下面介绍一个我常用的块操作。在 Python 或 Shell 中都没有多行注释，我们要在每一行的行首插入井号（#）我们可以使用块操作

组合键实现上述功能，方法是 `Control + v` 再用`j`/`k`上下移动方向键，在输入大写的`I`(shift + i)，然后输入一个井号(`#`)，再按下 `ESC` 结束输入回到命令模式。此时所有块光标前都多了一个井号。

使用 Control + V 进入块模式还可以剪切、复制和粘贴，挺方便的。

关于 vi/Vim 的操作远不止这些。如果你想了解更多，请使用 末行命令 `:help` 查看帮助。

关于 vi/Vim 编辑器的笔记我先记下到这里，虽然有很多常用的命令要记住，但他确实很强大，如何你是码农，你会爱上他。当使用 vi/Vim 时，你会发现你的手根本不用离开主键盘就可以快速完成一切任务，简直是码农必备之神器。

如果你想玩点 Linux/UNIX 下更高级的编辑器，请尝试 emacs。

第六章、用户权限管理

Linux 是多用户的操作系统，他允许多个用户同时登录一台电脑，并进行不同的操作。Linux 内部使用用户ID(即UID，是一个从 0 开始的整数)来区分各个用户。每一个UID对应一个字符串，这个字符串就是用户名，且用户名不能重复。

root用户简介

在 Linux 系统中，root 用户是系统的超级管理员账户（UID 为 0），拥有最高权限，可以执行所有操作（包括修改内核、系统文件、加载驱动、安装软件、管理用户权限等）。

root与其他用户的区别

特性	root用户	普通用户
用户ID（UID）	0	通常≥1000（Linux默认）
权限范围	无限制（可读写所有文件、终止任意进程）	仅限自身文件和授权操作
系统修改权	可安装/删除软件、修改系统配置	仅影响用户主目录（如`/home/username`）
危险操作	可直接执行`rm -rf /`等破坏性命令	受权限限制，无法删除系统文件
登录限制	默认禁止SSH直接登录（安全最佳实践）	允许正常登录

本章我们来学习用户权限管理相关的命令，内容如下:

- sudo 命令
- chmod 命令
- chgrp 命令
- chown 命令

1. sudo 命令

sudo 命令可以让普通用户临时使用 root 用户权限来执行命令，你可以认为是让普通用户如皇帝授旨，拥有大权！

命令格式

```
sudo 命令 [选项] [参数]
```

示例

Linux 的 根(/) 下不允许普通用户创建文件，但 root 用户就可以创建文件和文件夹。

```
weimingze@mzstudio:~$ touch /mylinux.py
touch: cannot touch '/mylinux.py': Permission denied
weimingze@mzstudio:~$ sudo touch /mylinux.py
[sudo] password for weimingze:
weimingze@mzstudio:~$ ls -l /mylinux.py
-rw-r--r-- 1 root root 0 May 22 13:57 /mylinux.py
weimingze@mzstudio:~$
```

可见普通用户 weimingze 创建 `/mylinux.py` 文件失败了，但使用 `sudo touch /mylinux.py` 并输入 `root` 用户的密码后文件被创建了，并且他的属主是 root，属组是 root。

sudo 常用选项

选项	说明
`-i`	切换到 root 用户

如:

```
weimingze@mzstudio:~$ pwd
/home/weimingze
weimingze@mzstudio:~$ sudo -i
[sudo] password for weimingze:
root@mzstudio:~# pwd
/root
root@mzstudio:~#
```

一开始 weimingze 用户的当前工作路径是 `/home/weimingze`，当执行 `sudo -i` 命令并输入密码后，用户的当前工作路径是 `/root`。

我们再来看一下 命令提示符前面的这段字符串: `root@mzstudio:~#` 这个是 `bash` 解析器（后面再学bash）下的特有提示符。我们来解释下每一段的含义

1. `root` 代表当前的用户名。
2. `mzstudio` 是当前的主机名（在安装系统是设定的，保存在 `/etc/hostname` 文件中，可以修改）。
3. `@` 是用户名和主机名的分隔符。
4. `~` 代表你的当前工作路径是用户主目录`~`。
5. `:` 是分隔符。
6. `#` 是命令提示符，超级用户的提示符是 `#`, 普通用户提示符是 `$`（普通用户都向钱看，而超级用户看到的是格局）。

有了 root 用户的密码，你就拿到了操作系统的最高权限，你自己试试吧。

2. chmod 命令

chmod（Change Mode）是 Linux/Unix 系统中用于修改 `文件或文件夹权限` 的命令，通过调整权限位来控制用户对文件的访问级别（读、写、执行）。其核心作用是 管理文件系统的安全性。

Linux 中每个文件/文件夹的权限分为三类用户：

1. 用户（User）：文件拥有者（属主）。
2. 所属组（Group）：文件所属的用户组（属组）。
3. 其他用户（Others）：既非拥有者也不在所属组的用户。

每种用户可分配以下权限：

1. `r`（读）：查看文件内容 / 列出文件夹内容。
2. `w`（写）：修改文件 / 在文件夹中创建/删除文件。
3. `x`（执行）：运行文件（如python或shell脚本） 或 进入文件夹（如 cd）。

每个用户的权限用一个二进制的位(1bit)表示。如果该位置 0 则表示没有权限，置 1 则表示有权限。如下图所示:

权限项	读	写	执行	读	写	执行	读	写	执行
字符表示	(r)	(w)	(x)	(r)	(w)	(x)	(r)	(w)	(x)
数字表示	4	2	1	4	2	1	4	2	1
权限分配	文件的用户			文件所属组用户			其他用户		

如：`hello.py` 文件的权限如下:

```
weimingze@mzstudio:~$ ls -l hello.py
-rw-rw-r-- 1 weimingze weimingze 23 May 22 11:57 hello.py
```

`hello.py`的权限就是二进制的 `110`(属主)、`110`(属组)和`100`(其他)，用十进制的数字表示就是`6`、`6`、`4`。

命令格式

```
chmod 权限 文件/文件夹
```

权限有两种表示方法：

1. 使用十进制的数字，如上述 `hello.py` 的权限就是 `664`。
2. 使用符号模式（直观），即: `[用户类别][操作符][权限]`
   - 用户类别:
     - u：所有者
     - g：所属组
     - o：其他用户
     - a：所有用户（默认）
   - 操作符:
     - +：添加权限
     - -：移除权限
     - =：直接设置权限（覆盖原有）
   - 权限:
     - r：读权限
     - w：写权限
     - x：执行权限

示例:

```
weimingze@mzstudio:~$ ls -l hello.py
-rw-rw-r-- 1 weimingze weimingze 23 May 22 11:57 hello.py
weimingze@mzstudio:~$ chmod 700 hello.py  # 属主可以读写执行，属组和其他无权限。
weimingze@mzstudio:~$ ls -l hello.py
-rwx------ 1 weimingze weimingze 23 May 22 11:57 hello.py
weimingze@mzstudio:~$ chmod 664 hello.py
weimingze@mzstudio:~$ ls -l hello.py
-rw-rw-r-- 1 weimingze weimingze 23 May 22 11:57 hello.py
weimingze@mzstudio:~$ chmod u+x hello.py  # 属主加执行权限
weimingze@mzstudio:~$ ls -l hello.py
-rwxrw-r-- 1 weimingze weimingze 23 May 22 11:57 hello.py
weimingze@mzstudio:~$ chmod u-x hello.py  # 属主去掉执行权限
weimingze@mzstudio:~$ ls -l hello.py
-rw-rw-r-- 1 weimingze weimingze 23 May 22 11:57 hello.py
weimingze@mzstudio:~$ chmod a+x hello.py  # 所有用户加执行权限
weimingze@mzstudio:~$ ls -l hello.py
-rwxrwxr-x 1 weimingze weimingze 23 May 22 11:57 hello.py
weimingze@mzstudio:~$ chmod a=r hello.py  # 所用用户只读权限
weimingze@mzstudio:~$ ls -l hello.py
-r--r--r-- 1 weimingze weimingze 23 May 22 11:57 hello.py
weimingze@mzstudio:~$ chmod a=rw,g=rw,o=r hello.py  # 等同于 chmod 664 hello.py
weimingze@mzstudio:~$ ls -l hello.py
-rw-rw-r-- 1 weimingze weimingze 23 May 22 11:57 hello.py
```

> 注意: chmod 设置的权限对 root 用户无效。

3. chgrp 命令

chgrp（Change Group）是 Linux/Unix 系统中用于修改 文件或文件夹的**所属组**的命令。它的作用是 调整文件/文件夹的组归属，从而控制哪些用户组可以访问该文件。

组

组（Group）是 Linux/UNIX中用于权限管理的工具。

组是一种用于管理用户权限的逻辑统一操作机制。它通过将多个用户归类到同一个组，简化了对文件、文件夹和系统资源的权限分配。

组的作用

1. 权限共享：组内的用户可以共享文件或文件夹的组权限（如读、写、执行）。
2. 精细控制：通过为组分配特定权限，避免为每个用户单独设置权限的繁琐操作。
3. 系统管理：服务进程（如 www-data、docker）通常以特定组运行，确保安全性。

组相关文件

`/etc/group` 是用于存储组信息的文件，他的格式是 `组名:密码占位符:GID:成员列表`。如下所示：

```
weimingze@mzstudio:~$ cat /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:syslog,weimingze
...
gnome-initial-setup:x:985:
weimingze:x:1000:
```

冒号分隔

1. 第一列：组名
2. 第二列：密码占位符
3. 第三列：组ID(GID)
4. 第四列：组内的用户，以英文的逗号分隔。

命令格式

```
chgrp [选项] 新组名 文件或文件夹
```

示例:

```
weimingze@mzstudio:~$ ls -l hello.py
-rw-rw-r-- 1 weimingze weimingze 23 May 22 11:57 hello.py
weimingze@mzstudio:~$ sudo chgrp root hello.py  # 将 hello.py 的属组改为 root
[sudo] password for weimingze:
weimingze@mzstudio:~$ ls -l hello.py
-rw-rw-r-- 1 weimingze root 23 May 22 11:57 hello.py
weimingze@mzstudio:~$ sudo chgrp 1000 hello.py  # 将 hello.py 的属组改为
weimingze(weimingze的组ID为1000)
weimingze@mzstudio:~$ ls -l hello.py
-rw-rw-r-- 1 weimingze weimingze 23 May 22 11:57 hello.py
```

常用选项

选项	说明
`-R`	递归修改文件夹及其子文件/文件夹的组归属。
`--reference=参照文件`	将目标文件的组设为与参照文件相同。
`-v`	显示详细操作信息（verbose）。
`-c`	仅显示发生更改的文件信息。

3. chown 命令

chown（Change Owner）是 Linux/Unix 系统中用于修改文件或文件夹的所有者和所属组的命令，其核心作用是管理文件的归属关系，确保正确的用户或组拥有访问权限。

命令格式

```
chown [选项] [新所有者][:新所属组] 文件或文件夹
```

示例:

```
weimingze@mzstudio:~$ ls -l hello.py
-rw-rw-r-- 1 weimingze weimingze 23 May 22 11:57 hello.py
weimingze@mzstudio:~$ sudo chown root hello.py  # 将拥有者改改成 root
[sudo] password for weimingze:
weimingze@mzstudio:~$ ls -l hello.py
-rw-rw-r-- 1 root weimingze 23 May 22 11:57 hello.py
weimingze@mzstudio:~$ sudo chown weimingze hello.py  # 将拥有者改成 weimingze
weimingze@mzstudio:~$ ls -l hello.py
-rw-rw-r-- 1 weimingze weimingze 23 May 22 11:57 hello.py
weimingze@mzstudio:~$ sudo chown root:root  hello.py    # 同时修改拥有者和属组
weimingze@mzstudio:~$ ls -l hello.py
-rw-rw-r-- 1 root root 23 May 22 11:57 hello.py
weimingze@mzstudio:~$ sudo chown 1000:1000  hello.py  # 使用 UID和GID 来代替用户名和组名
weimingze@mzstudio:~$ ls -l hello.py
-rw-rw-r-- 1 weimingze weimingze 23 May 22 11:57 hello.py
```

常用选项

选项	说明
`-R`	递归修改文件夹及其子文件夹/文件的所有权。
`--reference=参照文件`	使目标文件的所有权与参照文件相同。
`-v`	显示详细操作信息（verbose）。
`-c`	仅显示发生更改的文件信息。

第七章、用户和组管理

Linux 是一个多用户操作系统，用户（User） 和 组（Group） 是权限管理的核心机制，用于控制文件和系统资源的访问权限。

本章我们将讲解如下几个命令:

1. groups 命令
2. groupadd 命令
3. groupdel 命令
4. useradd 命令
5. usermod 命令
6. userdel 命令
7. passwd 命令
8. id 命令
9. who 命令

用户

Linux/UNIX 系统下，每个用户拥有独立的账户，用于登录系统、运行程序和管理文件。

每个用户有唯一的 用户ID（User ID 简写：UID），系统通过 UID 识别用户。

Linux/UNIX 用户存储 用户信息的文件是 `/etc/passwd`

前面我们已经讲过这个文件。

部分内容如下:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
...
```

此文件使用英文的冒号（`:`） 将文件分隔成了七列。

1. 第一列是用户名：用于登录（如:root、weimingze）。
2. 第二列是密码占位符：以前版本此处存放密码的哈希加密信息，现在用 `x` 代替。密码的哈希加密信息现在已经放在了文件 `/etc/shadow` 中。
3. 第三列是用户ID（UID）：0 一定是 root用户(超级用户)，1~999 为系统用户，1000及以上为普通用户（如: weimz 和 weimingze）。

4. 第四列是主组ID（GID）：在 Linux 中，一个用户可以在多个组中。但一个用户一定要有一个主组，而此整数代表主组的ID(即主组号)。
5. 第五列是用户全名或备注（用于登录时显示等，相当于很多论坛的昵称）。
6. 第六列是用户主文件夹（也称作主目录或家目录）的位置：用于存放此用户的数据文件，一般都放在 `/home/` 文件夹下。
7. 第七列是用户登录后所使用的 Shell，如：root 和 weimz 这几个用户都使用 `/bin/bash` 作为默认的 Shell。

组

组是将用户归类，用户批量管理的工具，方便批量分配权限（如共享文件和设备给某个组）。

每个组有唯一的 组ID（Group ID 简写：GID）。

Linux/UNIX 用户存储 组 信息的文件是 `/etc/group`

部分内容如下：

```
root:x:0:
daemon:x:1:
...
```

此文件同样使用英文的冒号（`:`） 将文件分隔成了四列。格式是 `组名:密码占位符:GID:成员列表`。

1. 第一列：组名
2. 第二列：密码占位符
3. 第三列：组ID(GID)
4. 第四列：组内的用户，以英文的逗号分隔。，以英文的逗号分隔。

主组

主组（Primary Group）：用户创建文件或文件夹时默认继承的组（每个用户必须属于一个主组）。

附加组

Linux/UNIX 用户可加入多个组，附加组是指用户加入的其他组，用于获取额外权限。

下面我们来介绍 用户和组相关的命令。先从组入手比较简单。

1. groups 命令

groups 命令用于显示当前用户属于哪几个组中。

命令格式:

```
$ groups [用户名]
主组名 附加组1 附加组2 ...
```

如果不给出 用户名 默认是获取当前用户的组。

如:

```
weimingze@mzstudio:~$ groups
weimingze adm cdrom sudo dip plugdev users lpadmin
```

weimingze 是 weimingze 用户的主组，adm、cdrom 等是 weimingze 加入的其附加组，比如 weimingze 加入的 cdrom 这个组，则这个用户就有 cdrom 这个组权限了。

2. groupadd/groupdel 命令

在 Linux/UNIX 下默认只有 root 用户才能对用户和组进行操作。或者用户通过 sudo 权限授权给特定用户root 权限来执行相应的命令。

常用的组管理命令有: 添加组 `groupadd`命令和删除组 `groupdel` 俩命令。

groupadd 命令

在系统中创建一个新的用户组。同时为新组分配唯一的 GID（Group ID）。组主要用于权限管理，方便多用户共享文件或文件夹的访问权限。

命令格式:

```
groupadd [选项] 组名
```

例如:

创建开发部(develop) 和 人力资源部（hr）两个组。

```
weimingze@mzstudio:~$ sudo groupadd develop
[sudo] password for weimingze:
weimingze@mzstudio:~$ sudo groupadd hr
weimingze@mzstudio:~$ tail -2 /etc/group
develop:x:1001:
hr:x:1002:
```

创建完成后使用 tail 命令查看 `/etc/group` 的最后两行，显示 develop 组ID 分别是 1001，hr 组ID 是 1002，说明创建成功了。

groupadd 常用选项

选项	说明
`-g GID`	指定新组的 GID（默认自动分配）
`-f`	如果组已存在，强制成功退出（不报错）

groupdel 命令

groupdel 命令的用于删除系统中已存在的用户组。此命令将从 `/etc/group` 和 `/etc/gshadow` 中移除该组的记录行。

命令格式:

```
groupdel [选项] 组名
```

示例:

删除 hr 这个组。

```
weimingze@mzstudio:~$ tail -2 /etc/group
develop:x:1001:
hr:x:1002:
weimingze@mzstudio:~$ sudo groupdel hr
[sudo] password for weimingze:
weimingze@mzstudio:~$ tail -2 /etc/group
weimingze:x:1000:
develop:x:1001:
weimingze@mzstudio:~$
```

可见这个组已经不存在了。

注意事项:

groupdel 命令不会删除 属于该组的用户，但用户会失去该组的附加权限。

groupdel 不能删除 某个用户的 主组，除非先修改该用户的主组。

groupdel 常用选项

选项	说明
`-f`	强制删除，可能导致用户无法登录。

3. useradd 命令

useradd 命令是用于创建新用户的基本命令。它会根据命令行参数和系统默认设置创建新的用户账户。

作用

1. 在系统中创建新的用户账户。
2. 设置用户的基本属性（如用户ID、主目录、登录 Shell 等）
3. 为新用户创建主目录（如果指定了相应选项）
4. 为新用户设置初始组和附加组

命令格式:

```
useradd [选项] 登录用户名
```

示例:

添加一个叫张三(`zhang3`)的用户

```
weimingze@mzstudio:~$ sudo useradd zhang3
[sudo] password for weimingze:
weimingze@mzstudio:~$ tail -2 /etc/passwd
sshd:x:122:65534::/run/sshd:/usr/sbin/nologin
zhang3:x:1001:1002::/home/zhang3:/bin/sh
weimingze@mzstudio:~$ tail -2 /etc/group
develop:x:1001:
zhang3:x:1002:
weimingze@mzstudio:~$ sudo tail -2 /etc/shadow
sshd:!:20230::::::
zhang3:!:20230:0:99999:7:::
weimingze@mzstudio:~$ ls /home/
weimingze
```

如上述可见在 `/etc/passwd` 的最后一行多了一条记录 `zhang3:x:1001:1002::/home/zhang3:/bin/sh`。这说明 zhang3 的用户 ID 是 1001，主组 ID 是 1003， 没有别名，用户主文件夹是 `/home/zhang3`，但并没有创建这个文件夹。用户默认使用的 Shell 是 `/bin/sh`，默认 Shell 可以通过选项进行更改。

可见 上述命令会为这个用户单独创建了一个组，在 `/etc/group` 下多了一行 `zhang3:x:1002:`，这说明组名是 `zhang3`，组 ID 是 1003。这个是 zhang3 用户的主组。

在文件 `/etc/shadow` 也多了一行 `zhang3:!:20230:0:99999:7:::`，此处用于保存 `zhang3` 的密码信息。需要使用 `passwd` 命令修改密码。

> Shell 是用户用于解释执行命令的程序。在终端中，用于解释我们输入的命令的程序就是 Shell的一种具体实现。如果我们不需要这个用户能够登录系统，则可以使用 `/usr/sbin/nologin` 作为 Shell。

常用选项

选项	描述	示例
`-m`	创建用户主目录（通常与`-k`一起使用）	`useradd -m zhangsan`
`-s <路径>`	指定用户登录 Shell	`useradd -s /bin/bash zhangsan`
`-c <别名>`	为用户添加备注/描述信息	`useradd -c "Developer" zhangsan`
`-d <路径>`	指定用户主目录路径	`useradd -d /home/zhangsanhome zhangsan`
`-g <组名或组ID>`	指定用户主组的组ID或组名	`useradd -g developer zhangsan`
`-e <时间字符串>`	设置账户过期日期（YYYY-MM-DD）	`useradd -e 2023-12-31 zhangsan`
`-f <数字>`	密码过期后账户被禁用的天数	`useradd -f 30 zhangsan`
`-G <组名或组ID>`	指定用户附加组的组ID或组名（逗号分隔）	`useradd -G develop,weimingze,zhangsan`
`-M`	不创建用户主目录	`useradd -M zhangsan`
`-N`	不创建与用户同名的组	`useradd -N zhangsan`
`-p <密码加密串>`	设置加密后的用户密码（不推荐直接使用）	`useradd -p encrypted_pass zhangsan`
`-r`	创建系统账户（无主目录，UID<1000）	`useradd -r systemuser`
`-u <数字>`	指定用户ID	`useradd -u 5000 zhangsan`

`-k <路径>`	指定骨架文件夹（与`-m`一起使用），默认会把骨架文件夹的内容复制到用户主目录	`useradd -m -k /etc/skel zhangsan`

示例:

给开发部的 李四 创建一个新的用户 `li4`，将他的主组指定到开发组 `develop`；在创建用户的时候，创建自己的用户主文件夹：`/home/lisi`；将 li3 的备注名指定为 `develop_lisi`，指定他的默认 Shell 为 `/bin/bash`

```
weimingze@mzstudio:~$ sudo useradd -g develop -m -d /home/lisi -c develop_lisi -
s /bin/bash li4
weimingze@mzstudio:~$ tail -4 /etc/passwd
weimingze:x:1000:1000:weimingze:/home/weimingze:/bin/bash
sshd:x:122:65534::/run/sshd:/usr/sbin/nologin
zhang3:x:1001:1002::/home/zhang3:/bin/sh
li4:x:1002:1001:develop_lisi:/home/lisi:/bin/bash
weimingze@mzstudio:~$ tail -3 /etc/group
weimingze:x:1000:
develop:x:1001:
zhang3:x:1002:
weimingze@mzstudio:~$ sudo tail -4 /etc/shadow
weimingze:$6$4cfDY79VgwUp70yB$vvN6o6.ldMV1doonvd7tppRtK.NLW3FI6Cg5rvFmwOvlhqYerJ
Dj3iiZekxLeeI79SozxG5ce8GJfyRlmBocJ/:20230:0:99999:7:::
sshd:!:20230::::::
zhang3:!:20230:0:99999:7:::
li4:!:20230:0:99999:7:::
weimingze@mzstudio:~$ ls /home/
lisi  weimingze
weimingze@mzstudio:~$
```

可见 `/etc/passwd` 下多了一行 `li4:x:1002:1001:develop_lisi:/home/lisi:/bin/bash`，文件 `/etc/group` 还是原来的样子，没有为 `li4` 创建新的组。在 `/home/` 多了一个文件夹 `lisi` 是 `li4` 用户的主目录。

4. passwd 命令

passwd 命令用于 在Linux/UNIX 系统中修改已经存在的用户的密码。

命令格式

```
passwd [选项] [用户名]
```

> 如果不给出用户名，默认是更改自己的密码。

示例

修改weimingze用户的密码。

```
weimingze@mzstudio:~$ passwd
Changing password for weimingze.
Current password:
New password:
Retype new password:
passwd: password updated successfully
```

先输入以前的密码，再两次输入新密码。

修改用户 `li4` 的密码。

```
weimingze@mzstudio:~$ sudo passwd li4
[sudo] password for weimingze:
New password:
Retype new password:
passwd: password updated successfully
weimingze@mzstudio:~$
```

先输入管理员密码，在输入 `li4` 的新密码。修改密码后 `/etc/shadow` 文件的变化

```
weimingze@mzstudio:~$ sudo tail -4 /etc/shadow
[sudo] password for weimingze:
weimingze:$6$4cfDY79VgwUp70yB$vvN6o6.ldMV1doonvd7tppRtK.NLW3FI6Cg5rvFmwOvlhqYerJ
Dj3iiZekxLeeI79SozxG5ce8GJfyRlmBocJ/:20230:0:99999:7:::
sshd:!:20230::::::
zhang3:!:20230:0:99999:7:::
li4:$y$j9T$tMMhKiIFx.bZrl9okX8gI1$71O/ie4nl4ud223.7wZAuJed9/
GzkpbiofwFVyJxKU5:20230:0:99999:7:::
```

第二列 `$y$j9T$tMMhKiIFx.bZrl9okX8gI1$71O/ie4nl4ud223.7wZAuJed9/GzkpbiofwFVyJxKU5` 就是 `li4` 的密码。这个密码是计算出来的，很难反推回之前输入的密码（不可逆）。

常用选项

选项	说明
`-l`, `--lock`	锁定用户账户（禁止登录）
`-e`, `--expire`	强制用户下次登录时修改密码
`-d`, `--delete`	删除用户密码（使其无密码）
`-k`, `--keep-tokens`	只更新过期的认证令牌
`-S`, `--status`	显示密码状态信息

5. usermod 命令

usermod 是 用于修改用户属性信息的命令，由 root 或具有 sudo 权限的用户执行。它可以调整用户的用户名、主目录、用户组、登录 Shell 等多种属性。

命令格式

```
usermod [选项] [用户名]
```

> 如果不给出用户名，默认是更改自己属性信息。

常用选项

选项	说明
`-s <Shell路径>`	修改用户的默认 Shell（如：`usermod -s /bin/bash li4`）
`-l <新用户名>`	修改用户名（如：`usermod -l newuser olduser`）
`-d <新主目录>`	修改用户主目录（需配合 `-m` 移动文件）
`-m`	将旧主目录内容移动到新目录（与 `-d` 联用）
`-g <主组>`	修改用户的主组（如：`usermod -g developer li4`）
`-G <附加组>`	修改用户的附加组（覆盖原有附加组，用逗号分隔）
`-aG <附加组>`	**追加**附加组（不覆盖原有组，常与 `-G` 联用）
`-L`	锁定用户账户（禁用登录）
`-U`	解锁用户账户
`-e <YYYY-MM-DD>`	设置账户过期时间（如：`usermod -e 2025-5-23 li4`）
`-u <UID>`	修改用户 UID（需确保唯一性）

示例

锁定/解锁 `li4` 账户

```
weimingze@mzstudio:~$ sudo usermod -L li4  # 锁定
[sudo] password for weimingze:
weimingze@mzstudio:~$ sudo usermod -U li4  # 解锁
```

修改 `li4` 账户的默认 Shell 为 `/bin/sh`。

```
weimingze@mzstudio:~$ sudo usermod -s /bin/sh li4
[sudo] password for weimingze:
weimingze@mzstudio:~$ tail -2 /etc/passwd
zhang3:x:1001:1002::/home/zhang3:/bin/sh
li4:x:1002:1001:develop_lisi:/home/lisi:/bin/sh
```

将用户添加到附加组：`adm`,`sudo`,`users`,`lpadmin`中（保留原组）

```
weimingze@mzstudio:~$ groups
weimingze adm cdrom sudo dip plugdev users lpadmin
weimingze@mzstudio:~$ sudo groups li4
li4 : develop
weimingze@mzstudio:~$ sudo usermod -aG adm,sudo,users,lpadmin li4
weimingze@mzstudio:~$ sudo groups li4
li4 : develop adm sudo users lpadmin
```

```
weimingze@mzstudio:~$ grep "li4" /etc/group
adm:x:4:syslog,weimingze,li4
sudo:x:27:weimingze,li4
users:x:100:weimingze,li4
lpadmin:x:114:weimingze,li4
```

> 可见 li4 用户已经加入了组：`adm`,`sudo`,`users`,`lpadmin` 中。

删除上述添加的所有的组。

```
weimingze@mzstudio:~$ sudo groups li4
li4 : develop adm sudo users lpadmin
weimingze@mzstudio:~$ sudo usermod -G "" li4
weimingze@mzstudio:~$ sudo groups li4
li4 : develop
weimingze@mzstudio:~$ grep "li4" /etc/group
weimingze@mzstudio:~$
```

6. userdel 命令

userdel 是 Linux 系统中用于删除用户账户的命令，由 root 或具有 sudo 权限的用户执行。它不仅可以删除用户账户，还可以选择是否同时删除用户的主目录和邮件池（mail spool）。

作用

1. 删除用户账户（从 `/etc/passwd` 和 `/etc/shadow` 中移除记录）。
2. 可选删除用户的主目录（`/home/用户名`）。
3. 可选删除用户的邮件池（`/var/mail/用户名`）。

命令格式

```
userdel [选项] [用户名]
```

常用选项

选项	说明
`-r`	删除用户的同时，删除其主目录和邮件池（推荐使用）。
`-f`	强制删除用户，即使该用户仍处于登录状态或存在未退出的进程（谨慎使用）。
`-Z`	同时删除用户的 SELinux 上下文（仅适用于 SELinux 系统）。

示例

删除 li4 账户的所有信息

```
weimingze@mzstudio:~$ ls /home/
lisi  weimingze
weimingze@mzstudio:~$ ls /var/mail/
weimingze@mzstudio:~$ tail -2 /etc/passwd
zhang3:x:1001:1002::/home/zhang3:/bin/sh
li4:x:1002:1001:develop_lisi:/home/lisi:/bin/sh
weimingze@mzstudio:~$ sudo tail -2 /etc/shadow
[sudo] password for weimingze:
zhang3:!:20230:0:99999:7:::
li4:$y$j9T$tMMhKiIFx.bZrl9okX8gI1$71O/ie4nl4ud223.7wZAuJed9/
GzkpbiofwFVyJxKU5:20230:0:99999:7:::
weimingze@mzstudio:~$ sudo userdel -r li4
userdel: li4 mail spool (/var/mail/li4) not found
weimingze@mzstudio:~$ ls /home/
weimingze
weimingze@mzstudio:~$ tail -2 /etc/passwd
sshd:x:122:65534::/run/sshd:/usr/sbin/nologin
zhang3:x:1001:1002::/home/zhang3:/bin/sh
weimingze@mzstudio:~$ sudo tail -2 /etc/shadow
sshd:!:20230::::::
zhang3:!:20230:0:99999:7:::
weimingze@mzstudio:~$
```

删除 `li4` 用户的信息并保留主文件夹（主目录）则需要用 `sudo userdel li4` 命令。

7. id 命令

Linux/UNIX 下的id 命令用于 查看用户和组的身份信息 的命令。

作用

1. 显示当前用户（或指定用户）的 UID（用户ID）和 GID（主组ID）。
2. 列出用户所属的所有组（包括主组和附加组）。

3. 检查用户是否存在，并返回其权限相关信息。

命令格式

```
id [选项] [用户名]
```

常用选项

选项	作用	示例
`-u`	仅显示 `UID`（用户ID）	`id -u`
`-g`	仅显示 `GID`（主组ID）	`id -g`
`-G`	显示用户所属的 `所有组ID`（包括附加组）	`id -G`
`-n`	显示 `名称（Name）` 而非数字ID（需搭配 `-u`/`-g`/`-G` 使用）	`id -un`
`-r`	显示 `真实ID`（Real ID，而非有效ID，适用于 setuid/setgid 场景）	`id -r -u`

示例

```
weimingze@mzstudio:~$ id
uid=1000(weimingze) gid=1000(weimingze) groups=1000(weimingze),4(adm),24(cdrom),
27(sudo),30(dip),46(plugdev),100(users),114(lpadmin)
weimingze@mzstudio:~$ id zhang3
uid=1001(zhang3) gid=1002(zhang3) groups=1002(zhang3)
weimingze@mzstudio:~$ id -u
1000
weimingze@mzstudio:~$ id -g
1000
weimingze@mzstudio:~$ id -G
1000 4 24 27 30 46 100 114
weimingze@mzstudio:~$ id -n -u
weimingze
weimingze@mzstudio:~$ id -r -u
1000
```

8. who 命令

Linux/UNIX 中的 who 命令是用于 查看当前登录系统的用户信息 的命令。

作用

1. 显示当前登录的所有用户（用户名、终端、登录时间、IP 地址等）。
2. 查看系统的启动时间（uptime）。
3. 检查谁正在使用系统（常用于系统管理和故障排查）。

命令格式

```
who [选项]... [ 文件或参数 ]
```

示例

```
weimingze@mzstudio:~$ who
weimingze pts/0        2025-05-23 11:35 (192.168.33.1)
weimingze seat0        2025-05-23 12:13 (login screen)
weimingze tty2         2025-05-23 12:13 (tty2)
```

从上述结果来看，内容共分成 4 列：

- 第一列是用户名，如：`weimingze`。
- 第二列是终端类型，如：`tty1`、`tty2`是本地物理终端、`pts/0`、`pts/1`是远程终端（如 ssh 等）、`seat0` 是图形用户界面（桌面）
- 第三列是登录时间，如：`2025-05-23 12:13`。
- 第四列是登录来源，如：`192.168.33.1`。

上述结果说明有三个用户登录了。

常用选项

选项	作用	示例
`-a`	显示 所有信息（相当于 `-b -d --login -p -r -t -T -u`）	`who -a`
`-b`	显示 系统最近启动时间（boot time）	`who -b`
`-H`	显示 表头（Header）	`who -H`
`-u`	显示 空闲时间（IDLE）和进程ID（PID）	`who -u`
`-q`	快速模式，仅显示用户名和登录用户数	`who -q`
`-r`	显示 系统运行级别（runlevel）	`who -r`
`-T`	显示 用户的消息状态（+ 可接收，- 不可接收）	`who -T`

第八章、进程管理

进程

进程（Process）是操作系统中正在运行的程序，可以理解成为进行中的程序。

我们安装的程序文件（如：`/usr/bin/tar` 文件就是 `tar` 命令对应的程序文件）是计算机 CPU 指令和数据的打包文件。程序文件保存在磁盘上，只占用磁盘空间，并不会占用 CPU、内存等系统资源。当我们运行 tar 命令（如：`tar czvf myhome.tar.gz ~/*`）时，此时你就开始了 `tar` 进程。当 tar 命令运行完毕，则此进程结束。

进程是程序的一次执行实例，是操作系统进行资源分配和调度的基本单位。Linux/UNIX 系统中每个进程拥有独立的虚拟地址空间、内存、文件描述符等系统资源，确保程序运行时相互隔离。

本章我们来学习 **进程** 相关的命令，内容如下:

- ps 命令
- pstree 命令
- top/htop 命令
- kill 命令
- killall 命令
- jobs/fg/bg/nohup 命令
- systemctl/who 命令

Ubuntu Linux 操作系统的启动步骤

1. 当计算机自动是会运行 BIOS/UEFI 中的程序来初始化 CPU（基础运行速度：主频）、内存（真实地址空间）和引导设备（如 U 盘、硬盘）。从固定位置加载自动引导程序（Bootloader）到内存中并开始运行。
2. Linux 下的 Bootloader 通常使用 GRUB(一个引导程序)，在此可以指定你安装的多操作系统(Windows 或 Linux 系统)中的一个，然后根据选择在磁盘上找到操作系统内核程序（Ubuntu Linux 的内核文件保存在 `/boot/vmlinuz`）加载到内存中并开始运行。
3. Ubuntu Linux 内核重新初始化CPU，初始化中断向量表、虚拟内存表、磁盘/网络等外设，上述初始化工作完成后，就会挂在根 `/` 文件系统，然后启动第一个用户级的进程 systemd（位于 `/usr/bin/systemd`）并开始进程调度。
4. systemd 完成后续各个用户进程的创建（如：桌面系统进行、账户管理进程等）并等待用户登录进行操作。systemd 不会退出，他一直停留在后台运行，监控所有的子进程运行。

内核程序和用户程序

- 内核程序：运行于 CPU 内核态的程序，内核模式下有运行 CPU 所有指令的权限（包括开关中断、切换模式等特殊指令），Linux内核程序就运行与内核模式下。
- 用户程序：运行于 CPU 用户态的程序，用户模式下仅限运行 CPU 普通指令（如：加减乘除等指令）。
  - 当用户需要操作磁盘等系统资源时（比如使用 `cp` 命令复制文件），需要将操作的数据（如源文件和目标文件路径）放入固定的位置，然后调用软中断指令切换到内核程序，内核程序完成操作后将操作结果在放在固定位置供本地系统调用的用户程序读取，然后继续运行此用户程序。
  - 用户程序的执行、挂起和终止完全由内核程序控制。

进程ID（PID）

Linux/UNIX 系统下，每一个用户进程都有一个自己的编号，这个编号唯一的标识了一个正在运行的用户进程，这就是进程ID，也叫做 `PID`。

Linux 下的进程 ID 是从 1 开始的，后面每启用一个进程此数字自动加 1。如果超过了系统能够表达的最大数值则会从 1 开始找到闲置不用的数字给此用户程序赋予此 ID 值。

Ubuntu Linux 系统的 systemd 进程的 ID 永远是 1，这个进程也是系统最重要的一个进程，他相当于所有其他用户进程的 **祖进程**。

父进程ID（PPID）

在 Ubuntu Linux 系统下，除 `systemd` 由内核创建外，所有其他的进程都是由 systemd 进程直接或间接的创建。这样用户进程的创建就形成一个树形结构。我们把创建此进程的进程称之为父进程，每个进程都会标识一个父进程的ID，即：（PPID）。

`systemd` 进程没有父进程。

systemd 进程

systemd 进程是现代 Linux 系统中启动的第一个进程，用于进程管理。早期 linux 2.6 内核(及之前版本) 最先启动的进程是 init 进程。 init 进程在单 CPU 时代非常流行，后由 systemd 取代。systemd 的优点是启动速度更快。为了兼容旧的用户程序，现在 Linux 操作系统上的 init 进程实际是 systemd 进程的别名。

1. ps 命令

ps（Process Status）命令用于查看当前 Linux/UNIX 系统的进程状态，他可以显示进程的详细信息。

命令格式

```
ps [选项]
```

例如:

```
weimingze@mzstudio:~$ ps
    PID TTY          TIME CMD
   2429 pts/0    00:00:00 bash
   3704 pts/0    00:00:00 ps
weimingze@mzstudio:~$
```

ps 命令如果不加任何选项，它会默认显示当前终端（TTY）关联的进程的基本信息（不显示其他用户或后台进程）。

输出的列说明

列名	含义
`PID`	进程 ID。
`TTY`	进程关联的终端（如 `pts/0` 表示当前终端，`?` 表示无终端）。
`TIME`	进程累计 CPU 占用时间。
`CMD`	进程的启动命令。

常用选项

选项	说明
`-aux`	显示所有进程及资源占用（CPU、内存等）。
`-ejH`	缩进显示进程树形关系。
`--forest`	树状结构显示父子进程关系
`-e`	显示所有进程（包括其他用户）。
`-f`	完整格式（显示命令行、父进程PPID等）。
`-l`	长格式显示
`-ef`	查看所有进程的完整信息（常用）。
`-u <用户>`	显示指定用户（如：`ps -u root`）的进程。
`-p <进程ID>`	显示指定 PID（如：`ps -p 1234`）的进程。
`-C <命令名>`	显示指定命令（如：`ps -C nginx`）的进程。
`-t <终端>`	显示指定终端（如：`ps -t pts/0`）的进程。

示例

显示 Linux 操作系统中所有进程及资源占用。

```
weimingze@mzstudio:~$ ps -aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.3  23348 14224 ?        Ss   12:23   0:05 /sbin/init sp
lash
root           2  0.0  0.0      0     0 ?        S    12:23   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        S    12:23   0:00 [pool_workque
ue_release]
root           4  0.0  0.0      0     0 ?        I<   12:23   0:00 [kworker/R-
rcu_gp]
root           5  0.0  0.0      0     0 ?        I<   12:23   0:00 [kworker/R-
sync_wq]
...
```

各个列说明

列名	含义
USER	启动此进程的用户。
PID	进程 ID。
PPID	父进程 ID。
%CPU	CPU 占用率。
%MEM	内存占用率。
VSZ	虚拟内存大小（KB）。
RSS	实际内存占用（KB）。
TTY	进程关联的终端（如 pts/0 表示当前终端，? 表示无终端）。
STAT	进程状态（如 S=睡眠, R=运行, Z=僵尸）。
START	进程启动时间。
TIME	进程累计 CPU 占用时间。
CMD	进程的启动命令。

进程的状态

在 Linux 内核中，常见有 6 种进程状态，在 进程状态（STAT）这一列的第一个字符标识进程的状态，如下：

状态代码	英文说明	说明
R	(Running)	正在运行状态（在 运行 调度队列中），占用CPU。
S	(Interruptible Sleeping)	可中断的睡眠（等待事件完成），不占用CPU。
D	(Uninterruptible Sleep)	不可中断的睡眠（通常与 I/O 相关），不占用CPU。
Z	(Zombie)	僵尸进程（已终止占用CPU，但占用资源未被父进程回收）。
T	(Stopped)	进程被信号停止（如 Control+Z），不占用CPU（需要信号恢复运行）。
I	(Idle)	内核线程（某些系统显示为 I）。

示例2

显示进程树形结构。

```
weimingze@mzstudio:~$ ps -ejH --forest
    PID    PGID     SID TTY          TIME CMD
      2       0       0 ?        00:00:00 kthreadd
      3       0       0 ?        00:00:00  \_ pool_workqueue_release
      4       0       0 ?        00:00:00  \_ kworker/R-rcu_gp
      5       0       0 ?        00:00:00  \_ kworker/R-sync_wq
    ...
   2298    2298    2298 ?        00:00:00 sshd
   2300    2300    2300 ?        00:00:00  \_ sshd
   2426    2300    2300 ?        00:00:00      \_ sshd
   2429    2429    2429 pts/0    00:00:00          \_ bash
   4041    4041    2429 pts/0    00:00:00              \_ ps
   2307    2307    2307 ?        00:00:00 systemd
   2311    2307    2307 ?        00:00:00  \_ (sd-pam)
   2320    2320    2320 ?        00:00:00  \_ pipewire
   2321    2321    2321 ?        00:00:00  \_ pipewire
   2328    2328    2328 ?        00:00:01  \_ snapd-desktop-i
   2488    2328    2328 ?        00:00:00  |   \_ snapd-desktop-i
   2335    2335    2335 ?        00:00:00  \_ wireplumber
   2340    2340    2340 ?        00:00:00  \_ pipewire-pulse
   2360    2360    2360 ?        00:00:00  \_ dbus-daemon
   2430    2430    2430 ?        00:00:00  \_ xdg-document-po
   2452    2452    2452 ?        00:00:00  |   \_ fusermount3
   2437    2437    2437 ?        00:00:00  \_ xdg-permission-
   4022    4022    4022 ?        00:00:00 fwupd
```

输出的列说明

列名	含义
PID	同上
PGID	进程组 ID（Process Group ID），同一组的进程共享相同的 PGID。
SID	会话 ID（Session ID），关联到终端的进程会话。
TTY	同上
TIME	同上
CMD	同上

2. pstree 命令

pstree 命令以树状结构直观显示系统中的进程层级关系，清晰展示父子进程的依赖关系。

命令格式

```
pstree [选项] [用户名]
```

不给出 用户名 默认显示所有进程。

示例1

显示所有进程的进程树

```
weimingze@mzstudio:~$ pstree
systemd─┬─ModemManager───3*[{ModemManager}]
        ├─NetworkManager───3*[{NetworkManager}]
        ├─at-spi-bus-laun─┬─dbus-daemon
        │                 └─4*[{at-spi-bus-laun}]
        ├─at-spi2-registr───3*[{at-spi2-registr}]
        ├─sshd───sshd───sshd───bash───pstree
        ├─systemd─┬─(sd-pam)
        │         ├─dbus-daemon
        │         ├─2*[pipewire───2*[{pipewire}]]
        │         ├─pipewire-pulse───2*[{pipewire-pulse}]
        │         ├─wireplumber───5*[{wireplumber}]
        │         ├─xdg-document-po─┬─fusermount3
        │         │                 └─6*[{xdg-document-po}]
        │         └─xdg-permission-───3*[{xdg-permission-}]
        ├─systemd-journal
```

```
        ├─systemd-logind
        ...
```

常用选项

选项	说明
`-p`	显示 PID（进程ID）。
`-u`	显示用户名（标识进程所有者）。
`-a`	显示完整命令行（包括参数）。
`-n`	按 PID 数字排序（默认按进程名排序）。
`-h`	高亮当前进程及其祖先。

显示用户 `weimingze` 创建的进程的进程树。

```
weimingze@mzstudio:~$ pstree weimingze
sshd───bash───pstree

systemd─┬─(sd-pam)
        ├─dbus-daemon
        ├─2*[pipewire───2*[{pipewire}]]
        ├─pipewire-pulse───2*[{pipewire-pulse}]
        ├─snapd-desktop-i───snapd-desktop-i───3*[{snapd-desktop-i}]
        ├─wireplumber───5*[{wireplumber}]
        ├─xdg-document-po─┬─fusermount3
        │                 └─6*[{xdg-document-po}]
        └─xdg-permission-───3*[{xdg-permission-}]
```

3. top/htop 命令

top 命令

top 命令是 Linux 中用于实时监控系统进程和资源占用的交互式工具。他提供内存、CPU等的实时占用情况。

命令格式

```
top [选项]
```

示例:

```
weimingze@mzstudio:~$ top
top - 16:55:22 up  4:02,  2 users,  load average: 0.01, 0.11, 0.15
Tasks: 282 total,   1 running, 281 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.3 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.2 si,  0.0 st
MiB Mem :   3866.7 total,   1982.1 free,    855.7 used,   1286.0 buff/cache
MiB Swap:   3866.0 total,   3866.0 free,      0.0 used.   3011.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   5447 weiming+  20   0   23604   5748   3572 R   1.0   0.1   0:00.07 top
   4845 root      20   0       0      0      0 I   0.3   0.0   0:01.46 kworker+
   4863 root      20   0       0      0      0 I   0.3   0.0   0:01.88 kworker+
      1 root      20   0   23348  14352   9488 S   0.0   0.4   0:06.69 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.06 kthreadd
      3 root      20   0       0      0      0 S   0.0   0.0   0:00.00 pool_wo+
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker+
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker+
    ...
```

当输入 `q` 键时退出 top 监控。

上述显示的顶部 5 行信息的内容如下：

信息	说明
`16:55:22`	当前系统时间（24小时制）。
`up 4:02`	系统已运行时间（up 表示运行时长，这里是 4 小时 2 分钟）。
`2 users`	当前登录系统的用户会话数（通过 who 命令可查看具体用户）。
`load average: 0.01, 0.11, 0.15`	系统负载平均值（1分钟、5分钟、15分钟的平均值）：数值表示平均活跃进程数，若数值接近或超过CPU核心数，表示系统过载。
Tasks	进程数统计（总数、运行、睡眠、停止、僵尸状态的进程数等）。
%Cpu(s)	CPU 使用情况（用户态 us、内核态 sy、低优先级进程 ni、空闲 id 等、I/O 等待时间 wa、硬件中断处理时间 hi、软件中断处理时间 si、被虚拟化层偷走的时间 st）。
Mem/Swap	物理内存和交换分区使用情况。

进程列表信息部分的内容如下：

信息	说明
PID	进程 ID。
USER	进程所有者。
PR	进程的**优先级**（Priority，内核动态调整）。
NI	**Nice 值**（用户态优先级调整）。
VIRT	进程占用的虚拟内存总量（单位 KB/MB）。
RES	进程实际使用的物理内存（单位 KB/MB）。
SHR	进程共享内存（Shared Memory，如库文件）。
%CPU	进程占用的 CPU 百分比。
%MEM	进程占用的 物理内存百分比（相对于总内存）。
S	进程状态（R=运行, S=睡眠, Z=僵尸等）。
TIME+	累计 CPU 占用时间。

htop 命令

htop 命令是 top 命令的增强版。他能够更直观显示 CPU 每个核的状态，并提供彩色的界面显示。

安装 htop 命令（Ubuntu Linux 没有预装 htop 命令，需要使用 apt 命令安装）。

```
weimingze@mzstudio:~$ sudo apt install htop
...
```

命令格式

```
htop [选项]
```

示例

```
weimingze@mzstudio:~$ htop
    0[||                          3.2%] Tasks: 95, 231 thr, 184 kthr; 1 runnin
    1[                            0.0%] Load average: 0.09 0.04 0.07
  Mem[|||||||||||||||||     610M/3.78G] Uptime: 04:35:54
  Swp[                        0K/3.78G]

  [Main] [I/O]
```

```
    PID USER       PRI  NI  VIRT   RES   SHR S  CPU%▽MEM%   TIME+  Command
   5520 weimingze   20   0 20184  4588  3436 R   3.9  0.1  0:00.50 htop
      1 root        20   0 23348 14352  9488 S   0.0  0.4  0:06.79 /sbin/init sp
    380 root        19  -1 67020 18524 17244 S   0.0  0.5  0:01.53 /usr/lib/syst
    ...
```

4. IPC信号

信号（Signals）

信号（Signals）是 Linux/UNIX 进程间通信（IPC）的一种方式（共七种方式）。信号由一个用户进程或者操作系统内核产生，通过 Linux 或 UNIX 操作系统内核传递给另外一个进程。信号在经过操作系统时，操作系统可以根据信号的意义做出相应的操作。

信号是一个 1~64 的整数，每一个整数代表不同的含义，这个含义由操作系统和用户定义。

我们可以使用 `kill -l` 命令列出所有的信号值和对应的类型，如下所示：

```
weimingze@mzstudio:~$ 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
```

其中 1~31 由早期 UNIX 系统定义并延续至 Linux 系统，每个信号的用途已经明确定义，是标准信号。

其中 34 ~ 64 是 Linux 扩展的信号，用于用户自定义用途。

标准信号（1-31）

以下是传统的 UNIX 信号，具有明确的用途，如下表所示：

编号	信号名称	默认行为	说明
1	SIGHUP	终止	**挂起信号**，终端断开或控制进程终止时发送（常用于重新加载配置，如 `nginx -s reload`）。
2	SIGINT	终止	**中断信号**（`Control+C`），由终端发送，请求进程终止。
3	SIGQUIT	终止+核心	**退出信号**（`Control+\`），类似 `SIGINT`，但会生成核心转储（core dump）。
4	SIGILL	终止+核心	**非法指令**，进程尝试执行无效的 CPU 指令。
5	SIGTRAP	终止+核心	**陷阱信号**，调试器（如 `gdb`）用于断点调试。
6	SIGABRT	终止+核心	**中止信号**，由 `abort()` 触发，表示严重错误（如 `assert` 失败）。
7	SIGBUS	终止+核心	**总线错误**，非法内存访问（如未对齐的内存访问）。
8	SIGFPE	终止+核心	**浮点异常**，如除以零。

9	`SIGKILL`	终止	**强制终止**（不可捕获、阻塞或忽略），确保进程结束。
10	`SIGUSR1`	终止	**用户自定义信号 1**，用途由应用程序定义（如 `nginx` 日志重载）。
11	`SIGSEGV`	终止+核心	**段错误**，非法内存访问（如访问 `NULL` 指针）。
12	`SIGUSR2`	终止	**用户自定义信号 2**，用途由应用程序定义。
13	`SIGPIPE`	终止	**管道破裂**，写入无读取端的管道
14	`SIGALRM`	终止	**定时器信号**，由 `alarm()` 或 `setitimer()` 触发（如 `timeout` 命令）。
15	`SIGTERM`	终止	**终止信号**（默认 `kill`），请求进程正常退出（可捕获）。
16	`SIGSTKFLT`	终止	**协处理器栈错误**（极少使用）。
17	`SIGCHLD`	忽略	**子进程状态变更**（子进程终止时发送给父进程）。
18	`SIGCONT`	继续	**继续执行**，恢复被 `SIGSTOP`/`SIGTSTP` 停止的进程（如 `fg` 命令）。
19	`SIGSTOP`	停止	**强制停止**（不可捕获、阻塞或忽略）。
20	`SIGTSTP`	停止	**停止进程**（`Control+Z`），可捕获（如 `vim` 临时挂起）。
21	`SIGTTIN`	停止	**后台进程尝试读取终端**（如 `cat &` 后尝试输入）。
22	`SIGTTOU`	停止	**后台进程尝试写入终端**（如 `echo "test" &` 后尝试输出）。

23	`SIGURG`	忽略	**紧急数据**（如带外数据 `OOB` 到达 socket）。
24	`SIGXCPU`	终止+核心	**CPU 时间超限**（超过 `ulimit -t` 限制）。
25	`SIGXFSZ`	终止+核心	**文件大小超限**（超过 `ulimit -f` 限制）。
26	`SIGVTALRM`	终止	**虚拟定时器超时**（由 `setitimer(ITIMER_VIRTUAL)` 触发）。
27	`SIGPROF`	终止	**性能分析定时器超时**（由 `setitimer(ITIMER_PROF)` 触发）。
28	`SIGWINCH`	忽略	**窗口大小变化**（终端调整大小时发送，如 `vim` 自适应）。
29	`SIGIO`	终止	**异步 I/O 事件**（文件描述符准备就绪）。
30	`SIGPWR`	终止	**电源故障**（UPS 电池即将耗尽时发送）。
31	`SIGSYS`	终止+核心	**无效系统调用**（进程执行了不存在的系统调用）。

常用信号列表

编号	信号名称	说明
1	SIGHUP	挂起信号
2	SIGINT	中断信号
3	SIGQUIT	退出信号
9	SIGKILL	强制终止
11	SIGSEGV	段错误
13	SIGPIPE	管道破裂
14	SIGALRM	定时器信号
15	SIGTERM	终止信号
17	SIGCHLD	子进程状态变更
18	SIGCONT	继续执行
19	SIGSTOP	强制停止
20	SIGTSTP	停止进程

常见终端信号

快捷键	编号	信号名称	说明
Control + C	2	SIGINT	中断信号
Control + \	3	SIGQUIT	退出信号
Control + Z	20	SIGTSTP	停止进程

测试终端信号

写一个 Python 程序 `always_run.py`，内容如下:

```python
import os

print("PID:", os.getpid())
while True:
    pass
```

> 注意: pass 前面要保留 4 个空格其他行前面不要留有空格。

> 具体 Python 的语法详见我的 《python教程》

此程序先打印当前进程的 PID ，然后进入死循环，一直运行不退出。

运行 `always_run.py` 程序

```
weimingze@mzstudio:~$ python3 always_run.py
PID: 8217
```

此时程序一直执行，一直占用终端，没有退出。因此在终端中不会看见 `weimingze@mzstudio:~$` 这样的命令提示符。

再打开另外一个终端查看 启动命令为 `python3`（使用 `-C python3` 选项）的进程的详细信息（使用 `-l` 选项），如下：

```
weimingze@mzstudio:~$ ps -l -C python3
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R  1000    8217    5889 99  80   0 -  7336 -      pts/0    00:04:42 python3
```

可见第二列（S 列）对应的状态是 `R` 说明程序在运行状态。

此时使用 top 命令查看，你会发现此进程将一个 CPU 占用 99.7% 的资源。 如下:

```
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   8217 weiming+  20   0   29344   9116   6300 R  99.7   0.2  11:07.93 python3
```

在第一个终端中输入 `Control + C` 向程序发送 `SIGINT` 信号，则 `always_run.py` 程序**停止运行**，提示如下：

```
weimingze@mzstudio:~$ python3 always_run.py
PID: 8217
^CTraceback (most recent call last):
  File "/home/weimingze/always_run.py", line 4, in <module>
    while True:
          ^^^^
KeyboardInterrupt

weimingze@mzstudio:~$
```

此时程序停止，命令提示符 `weimingze@mzstudio:~$` 出现，CPU 占有率降低。再用 ps 命令也查不到此进程。

再次运行 `always_run.py` 程序。

```
weimingze@mzstudio:~$ python3 always_run.py
PID: 8358
```

然后在终端中输入 `Control + Z` 向程序发送 `SIGTSTP` 信号，则 `always_run.py` 程序 **停止运行**，提示如下：

```
weimingze@mzstudio:~$ python3 always_run.py
PID: 8358
^Z
[1]+  Stopped                 python3 always_run.py
```

使用 `ps` 命令查看进程情况：

```
weimingze@mzstudio:~$ ps -l -C python3
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 T  1000    8358    5889 92  80   0 -  7336 do_sig pts/0    00:03:10 python3
```

此时 此进程的状态是 `T`（停止状态），此时 `top` 命令发现 CPU 占有率下降。

使用 `fg` 命令(后面再讲) 回复此进程为运行状态，如下：，

```
weimingze@mzstudio:~$ fg
python3 always_run.py
```

使用 `ps` 命令查看进程情况：

```
weimingze@mzstudio:~$ ps -l -C python3
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R  1000    8358    5889 93  80   0 -  7336 -      pts/0    00:07:54 python3
```

状态为 `R` 运行状态。

然后再终端中输入 `Control + \` 向程序发送 `SIGQUIT` 信号，则 `always_run.py` 程序 **退出运行**，提示如下：

```
weimingze@mzstudio:~$ fg
python3 always_run.py
^\Quit (core dumped)
weimingze@mzstudio:~$ ps -l -C python3
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
```

5. kill 命令

kill 命令

在 Linux 系统中，kill 命令用于向运行中的进程发送信号（signal），以控制进程的行为（如终止、停止、恢复等）。

它的作用是向另外一个进程发送信号（通过进程 ID 确定目标进程），默认情况下会发送`SIGTERM(15)`信号来结束进程。

命令格式

```
kill [选型] 进程ID(PID)
```

常用选项

选项	说明
`-<信号名字>` 或 `-s <信号名字>`	给出发送信号的名字。
`-<信号值>` 或 `-n <信号值>`	给出发送信号的名字。
`-l`	列出所有信号。

示例

如：运行上一节我们写的一个 Python 程序`always_run.py`，内容如下:

```
import os

print("PID:", os.getpid())
while True:
    pass
```

运行`always_run.py`程序

```
weimingze@mzstudio:~$ python3 always_run.py
PID: 8553
```

打开另外一个终端执行下列操作。

- 向进程ID 为 8553 的进程发送值为 20(SIGTSTP) 的信号。

```
weimingze@mzstudio:~$ ps -l -C python3
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R  1000    8553    5889 98  80   0 -  7336 -      pts/0    00:02:08 python3
weimingze@mzstudio:~$ kill -SIGTSTP 8553
weimingze@mzstudio:~$ ps -l -C python3
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 T  1000    8553    5889 93  80   0 -  7336 do_sig pts/0    00:02:39 python3
```

源终端显示

```
weimingze@mzstudio:~$ python3 always_run.py
PID: 8553

[1]+  Stopped                 python3 always_run.py
weimingze@mzstudio:~$
```

- 向进程ID 为 8553 的进程发送值为 18(SIGCONT) 的信号。

```
weimingze@mzstudio:~$ kill -18 8553
weimingze@mzstudio:~$ ps -l -C python3
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R  1000    8553    5889 47  80   0 -  7336 -      pts/0    00:02:46 python3
```

- 向进程ID 为 8553 的进程发送值为 9(SIGKILL) 的信号。

```
weimingze@mzstudio:~$ kill -9 8553
weimingze@mzstudio:~$ ps -l -C python3
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
```

-9 SIGKILL 信号

SIGKILL(值为 9) 的信号是杀死进程的信号，是比较常用的信号，通常我们用 `kill -9` 关闭一个 CPU 占用率比较高或可疑的进程。

SIGKILL 信号在进程收到此信号后也无法终止被关闭进程的命运。是一个可靠关闭进程的信号，但此信号可能会引起进程数据的丢失。

最好还是要正常退出进程。

权限

普通用户只能终止自己的进程，root 用户可终止所有进程。

6. killall 命令

killall 命令用于通过进程名（而非 PID）终止进程的命令。它会向所有匹配指定名称的进程发送信号（默认 SIGTERM），他比使用 kill 命令 更方便进行批量管理进程。

命令格式

```
killall [选项] 进程名
```

常用选项

选项	说明
`-s <信号名字>`	给出发送信号的名字。
`-<信号值>` 或 `-n <信号值>`	给出发送信号的值。
`-r <进程名正则表达式>`	通过正则表达式匹配进程名。
`-u <用户名>`	仅终止指定用户的进程。

示例

```
weimingze@mzstudio:~$ killall -s SIGTSTP python3
weimingze@mzstudio:~$ killall -s SIGCONT python3
weimingze@mzstudio:~$ killall -n 18 python3
weimingze@mzstudio:~$ killall -18 python3
weimingze@mzstudio:~$ killall python3
weimingze@mzstudio:~$ killall -9 python3
weimingze@mzstudio:~$ killall -9 -r "pyth*"  # 关闭所有命令名以 pyth 开头的进程。
weimingze@mzstudio:~$ killall -u weimingze python3  # 关闭weimingze 用户下所有命令名为 python3的进程。
```

7. 作业控制

作业控制（Job Control）是一种允许用户通过一个终端同时管理多个进程（作业）的机制。

它使得用户可以在一个终端会话中启动、停止、恢复、前后台切换以及终止作业，从而实现对多任务的灵活控制。

本节课我们要学习如下四个命令:

1. **jobs 命令**
2. **fg 命令**
3. **bg 命令**
4. **nohup 命令**

> 以上命令是作业控制（Job Control）相关的命令，主要用于管理进程的前后台运行、终端断开后的持久化以及作业查询等功能。

作业控制中的几个概念。

1. 作业（Job）：
   - 一个作业通常对应一个进程或多个进程通过管道（|）连接的进程组。(管道我们后面再讲)
2. 前台作业（Foreground Job）：
   - 前台作业：独占终端输入/输出，用户需等待其完成才能输入新命令。
3. 后台作业（Background Job）：
   - 在后台运行，不占用终端，用户可继续输入其他命令。

细心的学者会发现，当我们打开一个终端并在终端运行程序时，如果终端关闭了，那么在终端中使用命令启动的进程也停止的运行。为什么呢？终端又有什么作用呢?

先来观察一个现象，我们在终端中运行 `python3 always_run.py`，如下：

```
weimingze@mzstudio:~$ python3 always_run.py
PID: 4359
```

在打来另外一个终端，运行 `ps -ejH --forest` 查看进程树。我们看到如下:

```
weimingze@mzstudio:~$ ps -ejH --forest
    PID    PGID     SID TTY          TIME CMD
      2       0       0 ?        00:00:00 kthreadd
    ...
      4       0       0 ?        00:00:00  \_ kworker/R-rcu_gp
   3485    2650    2650 ?        00:00:00  \_ gnome-terminal
   3486    2650    2650 ?        00:00:00  |   \_ gnome-terminal.
   3492    3492    3492 ?        00:00:02  \_ gnome-terminal-
   3502    3502    3502 pts/1    00:00:00      \_ bash
   4359    4359    3502 pts/1    00:03:38          \_ python3
```

我们会发现 终端进程(`gnome-terminal`) 启动了 `bash` 进程，`bash` 进程启动了 `python3 always_run.py` 进程。

当终端(`gnome-terminal`)关闭，则终端进程会向所有的子进程发送 `SIGHUP`(值为 `1`) 的信号，子进程收到此信号会执行相应的操作（包括向他的子进程发送SIGHUP信号）然后退出运行。这就是为什么子进程会关闭的原因。

那 `bash` 又是什么呢?

bash 是命令的解释执行的程序。bash 接管我们键盘输入，解释执行我们输入的信息并启动进程，并能够让我们能看进程输出的结果。（后面章节会系统全面的讲解 bash）。

如下面这一行提示符就是 bash 的输出。

```
weimingze@mzstudio:~$
```

任何一个程序都有不变的三个通道:

1. 标准输入：bash 接管键盘，由 bash 交给前台进程，作为前台进程的输入。
2. 标准输出：由程序产生的正确信息，交给 bash（默认 bash 会打印在屏幕上）。
3. 标准错误输出：由程序产生的错误提示信息，交给 bash（默认 bash 会打印在屏幕上）

前面讲述了基础知识，下面我们来讲解作业控制。

启动前台作业

在终端中正常是启动程序是启动前台作业。前台作业会独占标准输入。如:

```
weimingze@mzstudio:~$ python3 always_run.py
```

使用 `Control + c` 向前台作业发送 SIGINT 信号可以终止前台作业。

启动后台作业

在终端中 输入启动程序的命令后，在后面加一个 `&` 符号则是启动后台作业。后台作业不占标准输入，但默认的标准输出和标准错误输出依旧会在终端显示。如:

```
weimingze@mzstudio:~$ python3 always_run.py &
[1] 4484
weimingze@mzstudio:~$ PID: 4484

weimingze@mzstudio:~$
```

再次回车后会显示终端提示符 `weimingze@mzstudio:~$` 此时你可以继续输入命令，刚运行的命令启动的进程就是后台作业。此时你的输入是交给 `bash`，而不是 `always_run.py` 这个程序。

在运行几次 `python3 always_run.py &`。

查看作业列表

jobs 命令

使用 jobs 命令可以查看当前 bash 启动的所有作业。如:

```
weimingze@mzstudio:~$ jobs
[1]   Running                 python3 always_run.py &
[2]-  Running                 python3 always_run.py &
[3]+  Running                 python3 always_run.py &
weimingze@mzstudio:~$ jobs -l
[1]   4484 Running                 python3 always_run.py &
[2]-  4486 Running                 python3 always_run.py &
[3]+  4487 Running                 python3 always_run.py &
weimingze@mzstudio:~$
```

`-l` 选项会增加 PID 的显示列。

- 第一列：[1], [2], [3] 是 作业的编号。
- 第二列：PID
- 第三列：作业状态，Running：运行中、Stopped：已停止、Terminated：已终止。
- 第四列：命令行。

停止作业

在前台作业输入 `Control + z` 快捷键将向进程发送 SIGTSTP 信号。当前进程进入 停止状态（`T`）并成为后台作业。如:

```
weimingze@mzstudio:~$ python3 always_run.py
PID: 4514
^Z
[4]+  Stopped                 python3 always_run.py
weimingze@mzstudio:~$ jobs -l
[1]   4484 Running                 python3 always_run.py &
[2]   4486 Running                 python3 always_run.py &
[3]-  4487 Running                 python3 always_run.py &
[4]+  4514 Stopped                 python3 always_run.py
weimingze@mzstudio:~$
```

恢复作业

使用 `fg 命令` 和 `bg 命令` 可以将进程修改为前台作业或后台作业。

fg命令

将后台作业变为前台作业，如果作业处于 **停止状态** 则发送 SIGCONT 信号使其进入运行状态。

命令格式

```
bg %<作业编号>
```

如

```
weimingze@mzstudio:~$ fg %4
python3 always_run.py
```

PID 为 4514 的进程变成前台作业。再次 `Control + z` 让其变为后台作业。

bg命令

将作业变为后台作业，向处于 **停止状态** 的作业发送 SIGCONT 信号使其进入运行状态。

命令格式

```
fg %<作业编号>
```

如

```
weimingze@mzstudio:~$ jobs -l
[1]   4484 Running                 python3 always_run.py &
[2]   4486 Running                 python3 always_run.py &
[3]-  4487 Running                 python3 always_run.py &
[4]+  4514 Stopped                 python3 always_run.py
weimingze@mzstudio:~$ bg %4
[4]+ python3 always_run.py &
weimingze@mzstudio:~$ jobs -l
[1]   4484 Running                 python3 always_run.py &
[2]   4486 Running                 python3 always_run.py &
[3]-  4487 Running                 python3 always_run.py &
[4]+  4514 Running                 python3 always_run.py &
weimingze@mzstudio:~$
```

PID 为 4514 的进程依旧是后台作业。但他处于运行状态了。

强制终止作业

使用 `kill %<作业编号>` 可以向一个后台作业发送 **信号** 让其退出。如：

```
weimingze@mzstudio:~$ kill %4
weimingze@mzstudio:~$ jobs -l
[1]   4484 Running                 python3 always_run.py &
[2]   4486 Running                 python3 always_run.py &
[3]-  4487 Running                 python3 always_run.py &
[4]+  4514 Terminated              python3 always_run.py
```

PID 为 4514 的进程已经终止了，使用 ps 命令也查不到此进程了。

nohup 命令

nohup 用于让 启动的进程 忽略挂断信号（SIGHUP）的操作，让启动的进程编程一个真正的后台进程，即使终端关闭，进程仍会继续执行。

命令格式

```
nohup 命令 [选项]  &
```

> `&` 表示推向后台运行。

使用 nohup 执行的程序，他将忽略标准输入，同时将标准输出和标准错误输出不在指向终端，而是保存在文件 `'nohup.out'` 中。当然我们可以通过输出重定向将其指向其他文件或打印设备。

示例：

```
weimingze@mzstudio:~$ nohup python3 always_run.py &
[1] 4945
weimingze@mzstudio:~$ nohup: ignoring input and appending output to 'nohup.out'

weimingze@mzstudio:~$ ps -ejH --forest
    PID    PGID     SID TTY          TIME CMD
      2       0       0 ?        00:00:00 kthreadd
      3       0       0 ?        00:00:00  \_ pool_workqueue_release
      ...
   3399    3399    3399 ?        00:00:00  \_ gvfsd-metadata
   3485    2650    2650 ?        00:00:00  \_ gnome-terminal
   3486    2650    2650 ?        00:00:00  |   \_ gnome-terminal.
   3492    3492    3492 ?        00:00:02  \_ gnome-terminal-
   3502    3502    3502 pts/1    00:00:00      \_ bash
   4945    4945    3502 pts/1    00:01:06          \_ python3
weimingze@mzstudio:~$ jobs
[1]+  Running                 python3 always_run.py &
weimingze@mzstudio:~$
```

可见此时 PID 为 4945 的进程是 bash 进程的子进程。然后我们关闭终端。再查看进程。

```
weimingze@mzstudio:~$ ps -ejH --forest
    PID    PGID     SID TTY          TIME CMD
      2       0       0 ?        00:00:00 kthreadd
      3       0       0 ?        00:00:00  \_ pool_workqueue_release
      ...
   3399    3399    3399 ?        00:00:00  \_ gvfsd-metadata
   4945    4945    3502 ?        00:05:27  \_ python3
```

此时 PID 为 4945 的进程依旧存在，但他的父进程已经不在是 `bash` 进程，而是 `systemd` 进程了。

此时 PID 为 4945 的进程已经不在依赖任何终端，他变成了 **守护进程**。

守护进程

守护进程是指不依附于某个终端而独立运行的程序。他是一个后台作业，通常提供某种服务，比如 网站前端服务 nginx、ssh 远程连接、Django 或 Flask 后端服务程序等。

8. systemctl 命令

服务

服务（Service）是指一类在后台持续运行的程序（守护进程，Daemon），通常用于提供系统或网络功能。如：Web服务(nginx 等)、远程控制(ssh、ftpd等)、数据库(mysqld等)、日志管理等。服务通常在系统启动时自动运行，无需用户交互，并通过特定方式（如命令行或系统工具）进行管理。

服务由 Linux 的初始化系统管理，Ubuntu 24.04 使用 systemd 对服务进行管理。

systemctl 命令

systemctl 命令是最新 Linux 系统中用于 **查询** 和 **发送控制命令** 的系统管理工具命令。

主要功能

1. 查询服务的状态。
2. 管理系统服务：启动、停止、重启服务等。

单元

单元（UNIT）是 systemd 管理系统资源的基本单位，代表系统中被 systemd 管理的各种实体（如服务、设备、挂载点等）。每个单元对应一个配置文件，定义了资源的属性、依赖关系和行为。

为了演示服务的用法。我们需要运行 `sudo apt install openssh-server` 命令安装一个 ssh 服务。命令如下:

```
weimingze@mzstudio:~$ sudo apt install openssh-server
```

如果你使用的是本教材提供的 `Ubuntu24.04LTS_IntelX86CPU.zip` 虚拟机镜像则无需安装（已经安装过了）。

命令格式

```
systemctl [选项] [参数]
```

如:

列出所有的单元。

```
weimingze@mzstudio:~$ systemctl
  UNIT                          LOAD   ACTIVE     SUB       DESCRIPTION
  sys-module-configfs.device    loaded active     plugged   /sys/module/configfs
  sys-module-fuse.device        loaded active     plugged   /sys/module/fuse
  run-snapd-ns.mount            loaded active     mounted   /run/snapd/ns
  run-user-1000-doc.mount       loaded active     mounted   /run/user/1000/doc
  NetworkManager.service        loaded active     running   Network Manager
  openvpn.service               loaded active     exited    OpenVPN service
  rsyslog.service               loaded active     running   System Logging Service
  setvtrgb.service              loaded active     exited    Set console scheme
  snapd.service                 loaded active     running   Snap Daemon
  ssh.service                   loaded active     running   OpenBSD Secure Shell
```

这里列出的所有 **活动** 单元（UNIT），如 `ssh` 服务的 单元（UNIT）是 `ssh.service`。

每列具体内容如下：

- **UNIT**：单元名称（如服务:`.service`、挂载点`.mount`、套接字`.socket`、设备`.device`等）。
- **LOAD**：单元加载状态（loaded 表示已加载、error：配置有错误、masked：单元被禁用）。
- **ACTIVE**：单元的高层级状态（如 active 运行中，inactive 未运行，failed 尝试启动但失败）。
- **SUB**：更详细的子状态（如 running 表示服务正在运行，exited 表示已退出）。
  - 服务（.service）：running（运行中）、exited（已退出）、dead（未运行）。
  - 挂载点（.mount）：mounted（已挂载）、waiting（等待挂载）。
  - 设备（.device）： plugged（设备插入）、listening（套接字监听）等。
- **DESCRIPTION**：单元的简要描述。

单元状态查询命令和选项

命令和选项	说明
`systemctl` 或 `systemctl list-units`	列出所有 **活动** 的单元（默认过滤掉未运行的）。
`systemctl list-units --all`	列出 **所有单元**（包括未运行的）。
`systemctl list-units -t <类型名>`	按类型名筛选（如 `service`, `mount`, `socket`）。
`systemctl status <UNIT>`	查看单元的详细状态（如 `ssh.service`）。
`systemctl is-active <UNIT>`	检查单元是否运行（返回 `active`/`inactive`）。
`systemctl --failed`	显示启动失败的单元。

如:

查询 ssh.service 的服务状态（单元名可以使用 `ssh.service`、也可以是 `ssh`）：

```
weimingze@mzstudio:~$ systemctl status ssh.service
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/usr/lib/systemd/system/ssh.service; disabled; preset: enab
led)
     Active: active (running) since Sun 2025-05-25 09:28:12 CST; 6h ago
TriggeredBy: ● ssh.socket
       Docs: man:sshd(8)
             man:sshd_config(5)
   Main PID: 1880 (sshd)
      Tasks: 1 (limit: 4551)
     Memory: 4.1M (peak: 5.1M)
        CPU: 513ms
     CGroup: /system.slice/ssh.service
             └─1880 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"
...
```

可见守护进程已经运行（running），PID 是 1880，单元配置文件是 `/usr/lib/systemd/system/ssh.service`。

生命周期管理命令和选项

命令/选项	说明
`systemctl start <UNIT>`	启动单元。
`systemctl stop <UNIT>`	停止单元。
`systemctl restart <UNIT>`	重启单元。
`systemctl reload <UNIT>`	重新加载配置（不重启服务）。
`systemctl reload-or-restart <UNIT>`	尝试 reload，失败则 restart。
`systemctl kill <UNIT>`	强制终止单元（默认发送 SIGTERM，可用 `-s <信号值>` 指定信号）。

如:

```
weimingze@mzstudio:~$ sudo systemctl stop ssh.service
weimingze@mzstudio:~$ sudo systemctl start ssh.service
weimingze@mzstudio:~$ sudo systemctl restart ssh.service
weimingze@mzstudio:~$ sudo systemctl kill ssh.service
weimingze@mzstudio:~$ sudo systemctl reload ssh.service
weimingze@mzstudio:~$ sudo systemctl restart ssh.service
```

开机自启管理

命令/选项	说明
`systemctl enable <UNIT>`	启用开机自启。
`systemctl disable <UNIT>`	禁用开机自启。
`systemctl is-enabled <UNIT>`	检查单元是否开机自启（返回 `enabled`/`disabled`）。
`systemctl mask <UNIT>`	**彻底禁用** 单元（创建符号链接到 `/dev/null`，防止手动或依赖启动）。
`systemctl unmask <UNIT>`	取消禁用。

9. 自制 Python 服务实验

这节课我们将自己写的 Python 程序设置为系统服务并能够使用 systemctl 进行管理。这是一节实验课，他能让我们更深入理解服务概念和管理服务。

这节课还是使用我们写的一个 Python 程序 `always_run.py`，内容如下:

```
import os

print("PID:", os.getpid())
while True:
    pass
```

这个程序在程序启动时打印一下自己的 PID，然后进入无限的死循环，什么都没有做，但他会将一个 CPU 的资源占满。当你使用 top 或 htop 命令查看 CPU 使用率情况就能看出此进程是否运行了。

将这个程序放置于用户主文件夹。我的这个程序的路径是：`/home/weimingze/always_run.py`

which 命令

which 命令能够帮我们打印出命令所在文件的位置，我们通过这个命令找到 `python3` 这个命令的路径，如下:

```
weimingze@mzstudio:~$ which python3
/usr/bin/python3
```

python3 对应的文件位于 `/usr/bin/python3`。

创建服务单元文件

我们创建一个服务单元，服务名为 `myfirstd`。

使用 vim 或 nano 编写文本文件 `/usr/lib/systemd/system/myfirstd.service`。

执行命令:

```
weimingze@mzstudio:~$ sudo vi /usr/lib/systemd/system/myfirstd.service
[sudo] password for weimingze:
```

写入如下内容:

```
[Unit]
Description=My Custom Service
After=network.target
```

```
[Service]
Type=simple
User=root
ExecStart=/usr/bin/python3 /home/weimingze/always_run.py
Restart=on-failure
RestartSec=5s

[Install]
WantedBy=multi-user.target
```

`ExecStart=/usr/bin/python3 /home/weimingze/always_run.py` 这一行内容需要根据自己的实际路径进行修改。

参数说明：

- Description：服务描述。
- After：定义服务启动的依赖（如网络就绪后启动）。
- Type：
  - simple（默认）：假设服务立即启动，不通知完成。
  - forking：服务派生后台进程，需指定 PIDFile。
  - 其他类型：oneshot（一次性任务）、notify（通过通知告知启动完成）等。
- User：运行服务的用户（如 root 或普通用户）。
- ExecStart：服务启动时执行的命令或脚本路径（需有可执行权限）。
- Restart：定义何时重启服务（如 on-failure、always）。
- WantedBy：指定服务所属的 target（通常为 multi-user.target）。

保存上述文件。

重载 systemd 配置

每次修改服务文件后，需让 systemd 重新加载配置文件的内容，使用 `systemctl daemon-reload` 来重新加载配置，如下所示。

```
weimingze@mzstudio:~$ sudo systemctl daemon-reload
```

启动服务

```
weimingze@mzstudio:~$ sudo systemctl start myfirstd
weimingze@mzstudio:~$ ps -l -C python3
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 R     0    3777       1 99  80   0 -  7246 -      ?        00:00:20 python3
```

使用 top 或 ps 命令 查看服务进程是否存在。

停止服务

```
weimingze@mzstudio:~$ sudo systemctl stop myfirstd
weimingze@mzstudio:~$ ps -l -C python3
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
```

服务进程退出了。

设置开机自启服务

```
weimingze@mzstudio:~$ sudo systemctl enable myfirstd
Created symlink /etc/systemd/system/multi-user.target.wants/myfirstd.service → /
usr/lib/systemd/system/myfirstd.service.
```

重启 Ubuntu Linux 你会发现使用 `/usr/bin/python3 /home/weimingze/always_run.py` 命令启动的这个进程一直存在，一个 CPU 占有率 近乎达到 100%。

取消开机自启服务

```
weimingze@mzstudio:~$ sudo systemctl disable myfirstd
Removed "/etc/systemd/system/multi-user.target.wants/myfirstd.service".
```

停止服务，重启电脑，你的电脑不在运行这个 `always_run.py` 程序，CPU 占有率下降了。

第九章、网络管理

网络管理

网络管理是指对计算机网络进行规划、配置、监控、维护和优化的过程，以确保网络高效、安全、稳定地运行。它涵盖硬件设备（如路由器、交换机）、软件配置（如IP地址、防火墙）、连接状态监控以及故障排除等方面。

网络管理的目标

1. 保证连通性。
2. 性能优化、减少延迟、提高带宽利用率。
3. 安全性（防御攻击）。
4. 可靠性（保障网络高可用性）。
5. 故障处理，快速诊断和修复问题。

Linux 常用的网络管理命令如下：

1. ping 命令
2. ip 命令
3. ssh 命令
4. scp 命令
5. wget 命令

1. ping 命令

ping 命令用于测试与目标主机的网络连通性（ICMP 协议）。

命令格式

```
ping [选项] 目标IP地址或DNS域名
```

示例

测试是否能连通 `weimingze.com` 这个网络主机地址，有如下提示说明成功连接。（按 `Control + c` 停止）。

```
weimingze@mzstudio:~$ ping weimingze.com # 或 ping 192.144.206.112
PING weimingze.com (192.144.206.112) 56(84) bytes of data.
64 bytes from 192.144.206.112: icmp_seq=1 ttl=128 time=9.08 ms
64 bytes from 192.144.206.112: icmp_seq=2 ttl=128 time=7.76 ms
64 bytes from 192.144.206.112: icmp_seq=3 ttl=128 time=7.96 ms
...
```

上述信息解释如下：

1. `64 bytes` 表示接收到的 ICMP 回应报文的大小（单位是字节）。
2. `from 192.144.206.112` 表示回应报文的来源 IP 地址（即目标主机的地址）。
3. `icmp_seq=1` 表示 ICMP 报文的序列号（从 1 开始递增）。
4. `ttl=128` 即：`Time To Live`（生存时间）：表示数据包在网络中的最大跳数（经过的路由器数量）。每经过一个路由器，TTL 值减 1，当 TTL 为 0 时，数据包被丢弃。此处 默认值是 128。
5. `time=9.08 ms` 表示数据包的往返时间，即从发送请求到收到回应的时间（单位：毫秒），时间越小说明速度越快。

如果连接失败则提示失败信息，如下：

```
weimingze@mzstudio:~$ ping weimingze.com
ping: weimingze.com: 域名解析出现暂时性错误
```

常用选项

选项	说明
`-c <次数>`	指定发送的 ICMP 请求次数，然后自动停止（如 `ping -c 4 example.com`）。
`-i <秒数>`	设置发送报文的时间间隔（默认 1 秒，root 用户可设为 <0.2 秒）。
`-s <字节数>`	指定发送的 ICMP 数据包大小（默认 56 字节 + 8 字节 ICMP 头 = 64 字节）。
`-t <TTL>`	设置 ICMP 包的生存时间（TTL），限制经过的路由器跳数。
`-W <秒数>`	设置等待回应的超时时间（默认无限制，单位：秒）。
`-q`	安静模式，只显示统计信息（不显示每次 ping 的结果）。
`-v`	详细模式，显示更多信息（如错误报文）。
`-D`	在时间戳前显示 Unix 时间（方便记录日志）。
`-n`	禁用 DNS 反向解析，直接显示 IP 地址（加快响应速度）。
`-4`	强制使用 IPv4（默认行为）。
`-6`	强制使用 IPv6。
`-I <网卡>`	指定使用的网络接口（如 `ping -I eth0 example.com`）。

如测试 `192.144.206.112` 能否连通，发送三次数据包，间隔 2秒发送一次，然后给出统计信息:

```
weimingze@mzstudio:~$ ping -c 3 -i 2 192.144.206.112
PING 192.144.206.112 (192.144.206.112) 56(84) bytes of data.
64 bytes from 192.144.206.112: icmp_seq=1 ttl=128 time=8.84 ms
64 bytes from 192.144.206.112: icmp_seq=2 ttl=128 time=7.74 ms
64 bytes from 192.144.206.112: icmp_seq=3 ttl=128 time=8.07 ms

--- 192.144.206.112 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 7.743/8.218/8.842/0.460 ms
```

2. ip 命令

ip 命令是 Linux 系统中功能强大的网络配置工具，用于替代传统的 ifconfig、route 等命令。它能够管理网络接口、IP 地址、路由表、ARP 缓存、隧道等网络相关功能。

作用

1. 管理网络接口（查看、启用/禁用、配置参数）。
2. 配置 IP 地址（IPv4/IPv6）。
3. 管理路由表（添加/删除路由、策略路由）。
4. 管理邻居表（ARP/NDP）。
5. 管理隧道、VLAN、VPN 等虚拟设备。

命令个格式

```
ip [选项] [操作对象] 子命令
```

常用操作对象

操作对象	说明
`address`、 `addr` 或 `a`	管理 IP 地址
`link`	管理网络接口（如网卡状态）
`route`	管理路由表
`neigh`	管理邻居表（ARP）
`tunnel`	配置隧道
`maddr`	管理多播地址

如：

查看本机的IP地址: `ip a`

```
weimingze@mzstudio:~$ ip a # 或命令全称: ip address show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP gro
up default qlen 1000
    link/ether 00:0c:29:c4:a8:57 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.33.148/24 brd 192.168.33.255 scope global dynamic
noprefixroute ens33
```

```
       valid_lft 1606sec preferred_lft 1606sec
    inet6 fe80::20c:29ff:fec4:a857/64 scope link
       valid_lft forever preferred_lft forever
```

- 其中有两个网络设备 `lo` 和 `ens33`。`lo` 是本地环回，`ens33` 是硬件网卡。
- `ens33` 的 IPv4 地址是 192.168.33.148，24位用于标记地址，8（32-24）位用于标记网络。
- `ens33` 的 IPv6 地址是 fe80::20c:29ff:fec4:a857，64位用于标记地址。

查看本机的IP地址(简洁显示 IP 信息):`ip -br address`

```
weimingze@mzstudio:~$ ip -br address
lo               UNKNOWN        127.0.0.1/8 ::1/128
ens33            UP             192.168.33.148/24 fe80::20c:29ff:fec4:a857/64
```

常用选项

操作对象	说明
`-br[ief]`	简洁显示（默认详细显示）
`-4`	指定 IPv4
`-6`	指定 IPv6
`-s`	显示详细统计信息（如 ip -s link）
`-c`	彩色输出

IP 地址管理（ip address）

1. 修改 IP 地址

```
weimingze@mzstudio:~$ sudo ip addr change 192.168.33.149/24 dev ens33
[sudo] password for weimingze:
```

2. 删除 IP 地址

```
weimingze@mzstudio:~$ sudo ip addr del 192.168.33.148/24 dev ens33
```

3. 添加临时 IP 地址

```
weimingze@mzstudio:~$ sudo ip addr add 192.168.33.148/24 dev ens33
```

接口管理（ip link）

1. 查看接口状态：

```
weimingze@mzstudio:~$ ip link show  # 或简写 ip l
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode
DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mod
e DEFAULT group default qlen 1000
    link/ether 00:0c:29:c4:a8:57 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
weimingze@mzstudio:~$ ip -br l
lo               UNKNOWN        00:00:00:00:00:00 <LOOPBACK,UP,LOWER_UP>
ens33            UP             00:0c:29:c4:a8:57 <BROADCAST,MULTICAST,UP,LOWER_
UP>
```

2. 启用接口 `ip link set ens33 up`

```
weimingze@mzstudio:~$ ip -br l
lo               UNKNOWN        00:00:00:00:00:00 <LOOPBACK,UP,LOWER_UP>
ens33            UP             00:0c:29:c4:a8:57 <BROADCAST,MULTICAST>
weimingze@mzstudio:~$ sudo ip link set ens33 up
weimingze@mzstudio:~$ ip -br l
lo               UNKNOWN        00:00:00:00:00:00 <LOOPBACK,UP,LOWER_UP>
ens33            UP             00:0c:29:c4:a8:57 <BROADCAST,MULTICAST,UP,LOWER_
UP>
```

3. 禁用接口 `ip link set ens33 down`

```
weimingze@mzstudio:~$ ip -br l
lo               UNKNOWN        00:00:00:00:00:00 <LOOPBACK,UP,LOWER_UP>
ens33            UP             00:0c:29:c4:a8:57 <BROADCAST,MULTICAST,UP,LOWER_
UP>
weimingze@mzstudio:~$ sudo ip link set ens33 down
weimingze@mzstudio:~$ ip -br l
lo               UNKNOWN        00:00:00:00:00:00 <LOOPBACK,UP,LOWER_UP>
ens33            UP             00:0c:29:c4:a8:57 <BROADCAST,MULTICAST>
```

4. 修改 MAC 地址： `ip link set ens33 address 00:11:22:33:44:55`

```
weimingze@mzstudio:~$ ip -br l
lo               UNKNOWN        00:00:00:00:00:00 <LOOPBACK,UP,LOWER_UP>
ens33            UP             00:0c:29:c4:a8:57 <BROADCAST,MULTICAST,UP,LOWER_
```

```
UP>
weimingze@mzstudio:~$
weimingze@mzstudio:~$ sudo ip link set ens33 address 00:11:22:33:44:55
weimingze@mzstudio:~$ ip -br l
lo               UNKNOWN        00:00:00:00:00:00 <LOOPBACK,UP,LOWER_UP>
ens33            UP             00:11:22:33:44:55 <BROADCAST,MULTICAST,UP,LOWER_
UP>
```

路由管理（ip route）

1. 查看路由表：`ip route show`

```
weimingze@mzstudio:~$ ip route show  # 或简写: ip r
default via 192.168.33.2 dev ens33 proto dhcp src 192.168.33.148 metric 100
192.168.33.0/24 dev ens33 proto kernel scope link src 192.168.33.148 metric 100
```

2. 添加默认网关：`ip route add`

```
weimingze@mzstudio:~$ ip r
default via 192.168.33.2 dev ens33 proto dhcp src 192.168.33.148 metric 100
192.168.33.0/24 dev ens33 proto kernel scope link src 192.168.33.148 metric 100
weimingze@mzstudio:~$ sudo ip route add default via 192.168.33.2
weimingze@mzstudio:~$ ip r
default via 192.168.33.2 dev ens33
default via 192.168.33.2 dev ens33 proto dhcp src 192.168.33.148 metric 100
192.168.33.0/24 dev ens33 proto kernel scope link src 192.168.33.148 metric 100
```

加静态路由：`ip route add`

```
weimingze@mzstudio:~$ ip r
default via 192.168.33.2 dev ens33
default via 192.168.33.2 dev ens33 proto dhcp src 192.168.33.148 metric 100
192.168.33.0/24 dev ens33 proto kernel scope link src 192.168.33.148 metric 100
weimingze@mzstudio:~$ ip route add 10.0.0.0/24 via 192.168.33.2
RTNETLINK answers: Operation not permitted
weimingze@mzstudio:~$ sudo ip route add 10.0.0.0/24 via 192.168.33.2
weimingze@mzstudio:~$ ip r
default via 192.168.33.2 dev ens33
default via 192.168.33.2 dev ens33 proto dhcp src 192.168.33.148 metric 100
10.0.0.0/24 via 192.168.33.2 dev ens33
192.168.33.0/24 dev ens33 proto kernel scope link src 192.168.33.148 metric 100
```

删除路由：`ip route del`

```
weimingze@mzstudio:~$ ip r
default via 192.168.33.2 dev ens33
```

```
default via 192.168.33.2 dev ens33 proto dhcp src 192.168.33.148 metric 100
10.0.0.0/24 via 192.168.33.2 dev ens33
192.168.33.0/24 dev ens33 proto kernel scope link src 192.168.33.148 metric 100
weimingze@mzstudio:~$ sudo ip route del 10.0.0.0/24
weimingze@mzstudio:~$ ip r
default via 192.168.33.2 dev ens33
default via 192.168.33.2 dev ens33 proto dhcp src 192.168.33.148 metric 100
192.168.33.0/24 dev ens33 proto kernel scope link src 192.168.33.148 metric 100
weimingze@mzstudio:~$ sudo ip route del default via 192.168.33.2
weimingze@mzstudio:~$ ip r
default via 192.168.33.2 dev ens33 proto dhcp src 192.168.33.148 metric 100
192.168.33.0/24 dev ens33 proto kernel scope link src 192.168.33.148 metric 100
```

详细用法请通过 `man ip` 查看 ip 命令手册。

3. ssh 命令

ssh 命令是 Linux/Unix 系统中用于安全远程登录和管理其他计算机的命令。

ssh 命令是基于 SSH（Secure Shell）协议开发的命令。它通过加密的通道实现安全的远程访问、文件传输和命令执行。

作用

1. 远程连接到另一台 **已运行 SSH 服务** 的 Linux/UNIX 系统（如服务器）。
2. 通过终端（Linux、UNIX 和 Windows 终端）在远程计算机上执行命令，对登录的计算机进行操作。

使用 ssh 管理的远程主机需要安装 ssh 服务。我们前面已经讲过，该服务是一个守护进程，用于远程使用 ssh 命令进行连接。

ssh 服务的安装方法:

```
weimingze@mzstudio:~$ sudo apt install openssh-server
```

> 需要确保你的 远程主机的 ssh 服务的处于运行状态，可以使用：`systemctl status ssh.service` 来确认运行状态。

ssh 常用的命令格式

```
ssh 用户名@远程IP地址或域名
```

如：此网站域名为 `weimingze.com`，用户名为 `weimingze`，使用ssh 登录命令如下：

```
weimz@wo-deMacBook-Pro ~ % ssh weimingze@weimingze.com
The authenticity of host '192.144.206.112 (192.144.206.112)' can't be
established.
ECDSA key fingerprint is SHA256:LxVUHuBKH/Z5mi2gZMNNTT+UgEIhVm5g8VXIHzy93ls.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.144.206.112' (ECDSA) to the list of known hosts.
weimingze@weimingze.com's password:
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.11.0-26-generic x86_64)
...
weimingze@web-server:~$
```

> 第一次连接会确认 **指纹** 信息，输入 `yes` 确认即可。此时出现远程主机的命令提示符：`weimingze@web-server:~$`。说明连接成功。

ssh 常用的选项和参数

选项/参数	说明
`ssh user@hostname`	基本用法，以 `user` 身份登录 `hostname` 主机。
`-p <port>`	指定 SSH 端口（默认 22）。
`-i <私钥文件>`	指定身份认证的私钥文件（默认用 `~/.ssh/id_rsa`）。
`-X`	启用 X11 转发（支持图形界面应用）。
`-C`	压缩数据传输，提升慢速网络的效率。
`-L <本地端口:目标主机:目标端口>`	本地端口转发（将本地端口映射到远程主机端口）。
`-R <远程端口:本地主机:本地端口>`	远程端口转发（将远程端口映射到本地主机端口）。
`-N`	不执行远程命令，仅用于端口转发。
`-T`	不分配伪终端（适用于非交互式任务）。
`-4`	强制使用 IPv4 地址
`-6`	强制使用 IPv6 地址
`-v` / `-vv` / `-vvv`	显示详细调试信息（用于排查连接问题）。

其他平台的 ssh 命令

ssh 命令是 Linux/UNIX 预装的命令，用法也基本一致，这里不在赘述。

Windows 操作系统也是由 ssh 命令的，你需要启动 **PowerShell** 才能使用他。虽然不如 Linux 原生的 ssh 好用，但也可以应急。

用法如下：

1. 先启动 Windows 的终端 **命令提示符**
2. 在命令提示符内输入 `powershell` 进入 **PowerShell**。
3. 使用 `ssh 命令登录`。

如下所示：

```
Microsoft Windows [版本 10.0.19045.5011]
(c) Microsoft Corporation。保留所有权利。

C:\Users\weimz>powershell
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS C:\Users\weimz> ssh weimingze@weimingze.com
The authenticity of host 'weimingze.com (192.144.206.112)' can't be established.
ED25519 key fingerprint is SHA256:3bMWotfoAuAMhs5ZQMBV1lwWlbPHe27lJamadg+Gdko.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'weimingze.com' (ED25519) to the list of known hosts.
weimingze@weimingze.com's password:
```

如下图所示：

4. scp 命令

scp（Secure Copy）命令是基于 SSH 协议的安全文件远程传输命令，用于在本地与远程主机之间加密传输文件或文件夹，提供与 cp 命令类似的用法，但支持跨网络操作。

作用：

1. 本地 向 远程主机 复制文件或文件夹。
2. 远程主机 向 本地 复制文件或文件夹。

命令格式:

```
scp [选项] 本地路径 远程路径
# 或
scp [选项] 远程路径 本地路径
```

远程路径的表示方式:

```
远程主机用户名@远程主机域名或IP地址:路径
```

常用选项

选项	说明
`-P <端口>`	指定远程主机的 SSH 端口（默认为 22）
`-r`	递归复制整个目录（用于传输目录）
`-C`	启用压缩传输（节省带宽）
`-p`	保留文件的权限、修改时间等属性
`-q`	静默模式（不显示传输进度）
`-i <密钥文件>`	指定用于认证的私钥文件
`-l <限速>`	限制带宽使用（单位：Kbit/s，如 `-l 1000` 限制为 1Mbps）
`-v`	显示详细调试信息（排错时有用）

示例

将我的用户主目录 下的 `hello.py` 文件复制到 远程主机 `weimingze.com`（用户名：weimingze）的 `/home/weimingze` 文件夹下。

```
weimingze@mzstudio:~$ scp ~/hello.py weimingze@weimingze.com:/home/weimingze/
weimingze@weimingze.com's password:
```

再复制回来

```
weimingze@mzstudio:~$ scp weimingze@weimingze.com:/home/weimingze/hello.py ~/
```

将我的用户主目录 下的 `mywebsite` 文件夹复制到 远程主机 `weimingze.com`（用户名：weimingze） 的 `/home/weimingze` 文件夹下。

```
weimingze@mzstudio:~$ scp -r ~/mywebsite weimingze@weimingze.com:/home/
weimingze/
```

5. wget 命令

wget 命令是 Linux 下常用的非交互式命令行下载工具，他支持 HTTP、HTTPS 和 FTP 协议，具有断点续传、递归下载、后台运行等特性，适合在脚本或服务器环境中使用。

命令格式:

```
wget [选项] URL链接
```

常用选项

选项	说明
`-O <文件名>`	指定下载文件的保存名称
`-P <目录>`	指定下载文件的存储目录（默认当前目录）
`-c`	断点续传（继续未完成的下载）
`-b`	后台静默下载（日志写入 `wget-log`）
`-r`	递归下载（用于下载整个网站）
`-np`	不追溯父目录（配合 `-r` 使用）
`-nd`	不创建目录结构（所有文件下载到当前目录）
`-l <深度>`	限制递归下载深度（如 `-l 2` 只下载两层）
`-A <扩展名>`	只下载指定扩展名的文件（如 `-A "*.jpg"`）
`-R <扩展名>`	排除指定扩展名的文件（如 `-R "*.mp4"`）
`-i <文件>`	从文件读取下载链接（批量下载）
`-q`	静默模式（不显示下载信息）
`--limit-rate=<速度>`	限速下载（如 `--limit-rate=500k` 限制 500KB/s）
`-U <User-Agent>`	设置 User-Agent（模拟浏览器）
`--no-check-certificate`	跳过 SSL 证书验证（不安全，慎用）

示例

下载图片文件

下载上节课用到的图片 https://weimingze.com/linux/images/chapter09/win_ssh.png

```
weimingze@mzstudio:~$ wget https://weimingze.com/linux/images/chapter09/
win_ssh.png
```

图片放在了当前路径下的 `win_ssh.png`(名称不变)。

指定文件名和保存路径

```
weimingze@mzstudio:~$ wget -O /tmp/1.png https://weimingze.com/linux/images/
chapter09/win_ssh.png
```

断点续传

```
weimingze@mzstudio:~$ wget  -c https://mirrors.neusoft.edu.cn/ubuntu-releases/
24.04.2/ubuntu-24.04.2-desktop-amd64.iso
```

递归下载整个网站（镜像）

```
weimingze@mzstudio:~$ wget -r -l https://weimingze.com/linux/
```

限速下载（避免占用带宽）

```
weimingze@mzstudio:~$ wget --limit-rate=1m https://mirrors.neusoft.edu.cn/
ubuntu-releases/24.04.2/ubuntu-24.04.2-desktop-amd64.iso
```

> 限制下载速度为 1MB/s。

批量下载（从文件读取文件中的 URL）

```
weimingze@mzstudio:~$ wget -i urls.txt
```

> urls.txt 每行包含一个下载链接。

模拟浏览器访问

```
wget -U "Mozilla/5.0" https://example.com
```

> `-U` 设置浏览器的 `User-Agent`

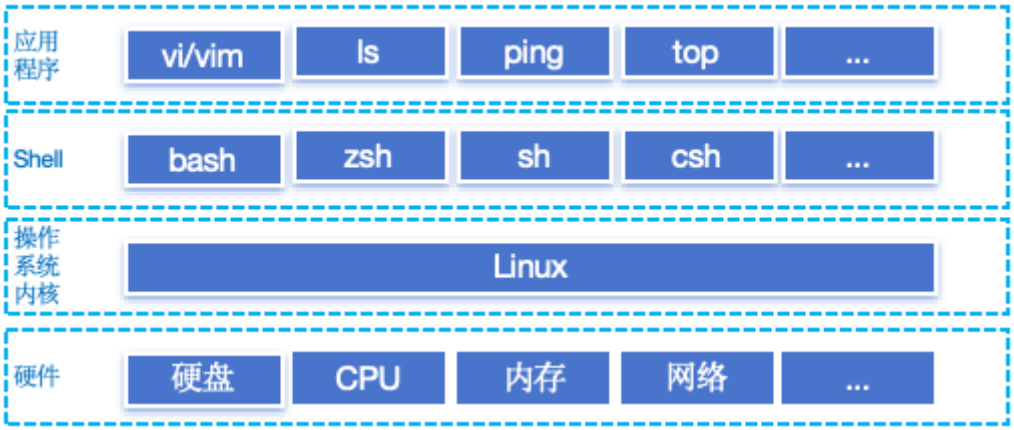
第十章、Shell 编程

1. Shell 简介

Shell 是 Linux/UNIX 操作系统中用于用户与内核（Kernel）交互的接口，既是命令行解释器，也是一种脚本编程语言。它的核心作用是充当用户与操作系统硬件/服务之间的翻译，同时具备自动化任务的能力。

Linux 操作系统的机构

计算器在现代计算机系统中，硬件是最根本的存在。为了能更好的管理硬件，让硬件协同工作，出现了运行与内核态的操作系统软件（如：Windows、Linux、MacOS等）。但操作系统并不提同各种应用程序，处于内核态运行的操作系统只提供一些可以让应用程序调用的接口供应用程序访问硬件。Linux 操作系统结构如下图所示：



Shell（壳）软件是调用用户管理应用程序，解析用户输入，并能够批量运行用户命令的工具。他允许你讲操作的命令写入文件中，然后批量的运行。

Shell 的种类

1. `sh`： 位于 `/bin/sh` 是早期标准 Shell，较简洁。
2. `bash`： 位于 `/bin/bash` 是 Linux/macOS(10.15及以下) 默认 Shell，兼容 `sh`，功能丰富。
3. `zsh`： 位于 `/bin/zsh` Linux 需要 `sudo apt install zsh` 安装,是 macOS(11及以上) 默认 Shell。
4. `powershell`：Windows 上最新比较好用的 Shell。

Linux 下 每个用户在登录后都会自定默认使用的 Shell ，他是在创建用户时指定的，见 `/etc/passwd` 最后一列，如:


```
root:x:0:0:root:/root:/bin/bash
...
weimingze:x:1000:1000:weimingze:/home/weimingze:/bin/bash
```

查看你的系统可用的 Shell

Linux 系统可用登录的 Shell 保存在文件 `/etc/shells` 内部。可以使用 `cat /etc/shells` 查看所有可用的 Shell。如：

```
weimingze@mzstudio:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/usr/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/usr/bin/dash
```

切换 Shell

用户可以在终端内切换到其他的 Shell 对计算机进行操作：

- 进入 Shell 需要执行 Shell 命令，如：`/bin/sh`。
- 退出 Shell 使用 `exit` 命令。

示例:

```
weimingze@mzstudio:~$ sh  # 进入 sh Shell
$ pwd
/home/weimingze
$ exit  # 退出 sh Shell
weimingze@mzstudio:~$ zsh  # 进入 zsh Shell(需要用 sudo apt install zsh 安装)
mzstudio% pwd
/home/weimingze
mzstudio% exit  # 退出 zsh
weimingze@mzstudio:~$
```

> 你会发现 不同的 Shell 命令提示符有所图同，`bash` 的提示符是 `weimingze@mzstudio:~$`, `sh` 的提示符是 `$`, `zsh` 的提示符是 `mzstudio%`
>
> 除了提示符不同外，各种 Shell 的语法也会略有不同。

使用 `sh` Shell 来查看进行示例

我们先进入 `sh`, 然后在 `sh` 内运行 `ps -ejH --forest` 命令，查看进程树

```
weimingze@mzstudio:~$ sh  # 进入 sh Shell
$ ps -efH --forest
...
weiming+    3945    2880  0 12:14 ?        00:00:06  \_ /usr/libexec/gnome-
terminal-server
weiming+    3953    3945  0 12:14 pts/0    00:00:00  |   \_ bash
weiming+    5500    3953  0 14:24 pts/0    00:00:00  |       \_ sh
weiming+    5585    5500 99 14:24 pts/0    00:00:00  |           \_ ps -efH --
forest
weiming+    5510    2880 20 14:24 ?        00:00:03  \_ /usr/bin/nautilus --
gapplication-service
$ exit
weimingze@mzstudio:~$
```

可见，终端默认运行的是 `bash` 进程，`bash` 启动了 `sh`, 最后由 `sh` 进程启动了 `ps` 进程。

下面我们来学习 `bash` 的语法。

2. echo 命令

echo 命令用于显示一行字符串到当前的标准输出。

命令格式

```
echo [选项] 字符串
```

> 字符串建议使用英文的双引号（"）括起来。

常用选项

选项	说明
`-n`	不输出末尾的换行符`"\n"`
`-e`	解释反斜杠 `\` 转义字符
`-E`	不解释反斜杠 `\` 转义字符（默认）

转义字符

转义字符	说明
``	
\\	反斜杠
\a	响铃
\b	退格
\c	停止后续输出
\e	ESC 键
\f	换页
\n	新行
\r	回车
\t	水平制表符
\v	垂直制表符
\0NNN	NNN 是1~3位八进制值表示的一个字节
\xHH	HH 是1~2位十六进制值表示的一个字节

示例

显示一行 `weimingze.com`

```
weimingze@mzstudio:~$ echo "weimingze.com"
weimingze.com
weimingze@mzstudio:~$
```

显示一行 `weimingze.com` 末尾不换行。

```
weimingze@mzstudio:~$ echo -n "weimingze.com"
weimingze.comweimingze@mzstudio:~$
```

> 可见 程序打印完 "weimingze.com" 后并没有换行操作，接下来 bash 打印的命令提示符紧跟其后打印了。这就连在一起了。

显示一行 `aaa\nbbb\nccc`

```
weimingze@mzstudio:~$ echo "aaa\nbbb\nccc"
aaanbbbnccc
weimingze@mzstudio:~$
```

显示三行文字 第一行是 `aaa`、第二行是 `bbb`、第三行是 `ccc`。

```
weimingze@mzstudio:~$ echo -e "aaa\nbbb\nccc"
aaa
bbb
ccc
weimingze@mzstudio:~$
```

3. 注释

作用

注释的作用是让 Shell 解释执行器忽略注释部分的内容。不让其执行。

语法规则

注释以英文的井号 （#） 开头，在本行末尾截止。

例如

```
weimingze@mzstudio:~$ ls  # 我是注释哈哈哈!
```

> 注: Shell 的只有单行注释，没有多行注释（这个语法同 Python 一样）。

4. 第一个 hello world程序。

这节课我们来说明 Shell 脚本程序的编写和运行方法。

脚本程序

所谓脚本程序是指依赖解释执行器来运行的程序，脚本程序一般都是文本文件而非二进制的机器指令。

编写 Shell 脚本程序

先编写一个文本文件名为：`myhello.sh`，内容如下：

```
echo "hello Shell!"
```

> Shell 文件一般以 `.sh` 作为后缀名。上面这个文件内部只有一行 `echo` 命令，但你可以书写多条 Linux 命令，每个命令占用一行即可。

运行 Shell 程序

shell 脚本程序有两种运行方法：

1. 在当前 Shell 进程内部运行 Shell 程序（使用 `source` 或 `.` 命令运行）。
2. 另外开始一个 Shell 进程，在新进程内部运行 Shell 程序（使用 `bash` 或 `./myhello.sh` 命令运行）。

> 两种方式的不同在于：第一中方式有可能改变当前 Shell 的状态（如环境变量的值），第二中运行则不会改变当前 Shell 的运行状态。

1. 在当前 Shell 进程内部运行 Shell 程序

命令格式

```
source 脚本文件
# 或
. 脚本文件
```

如:

```
weimingze@mzstudio:~$ source myhello.sh
hello Shell!
weimingze@mzstudio:~$ . myhello.sh
hello Shell!
weimingze@mzstudio:~$
```

> `source` 和 `.` 都是 bash 的内部命令，在 MacOS 的 zsh 下只能用 `source` 命令（没有 `.` 命令）。

2. 在新 Shell 进程内部运行 Shell 程序

此种方法也有两种运行方是：

2.1 显示启动 *bash* 命令

命令格式

```
bash 脚本文件
```

如:

```
weimingze@mzstudio:~$ bash myhello.sh
hello Shell!
weimingze@mzstudio:~$
```

2.2 在文件内部支持调用 *bash*

命令格式

```
./脚本文件
```

> 使用脚本文件当做命令直接执行，此种做法要求脚本文件必须有读取和执行权限。

如:

```
weimingze@mzstudio:~$ chmod +rx myhello.sh
weimingze@mzstudio:~$ ls -l myhello.sh
-rwxrwxr-x 1 weimingze weimingze 19 May 27 21:36 myhello.sh
weimingze@mzstudio:~$ ./myhello.sh
hello Shell!
weimingze@mzstudio:~$
```

首行解释器注释

在 Linux/UNIX 系统中，脚本文件的第一行以 `#!` 开头的注释是用来告诉 Shell 此脚本文件用那个解释器解释执行。英文名：`sha-bang`。

由于这个 `sha-bang` 语法没有中文命名，魏老师在这里给他取个中文名叫：**首行解释器注释**

语法格式

```
#!解释器路径
```

> 要求：脚本文件的前两个字符必须是 `#!`

bash 脚本文件的 首行解释器注释

重新编写 `myhello.sh`，在第一行插入 `#! /bin/bash`(如果不加入这一行。默认也是 `/bin/bash` 来解释执行)，内容如下：

```
#!/bin/bash
echo "hello Shell!"
```

运行

```
weimingze@mzstudio:~$ /home/weimingze/myhello.sh
hello Shell!
weimingze@mzstudio:~$
```

python 脚本文件的 首行解释器注释

重新编写 `myhello.py`，在第一行插入 `#!/usr/bin/python3`，内容如下：

```
#!/usr/bin/python3
print("hello Python3!")
```

运行

```
weimingze@mzstudio:~$ vim myhello.py
weimingze@mzstudio:~$ chmod +rx myhello.py
weimingze@mzstudio:~$ /home/weimingze/myhello.py
hello Python3!
weimingze@mzstudio:~$
```

所有的脚本语言的文件都可以使用 **首行解释器注释** 的语法直接运行（如：python、perl、node、ruby等）。

第十一章、Shell 变量

1. Shell 变量

Shell 变量在 Shell 中用一个英文名字来绑定一个数据（通常是字符串），供后续使用。

创建和修改变量的语法

```
变量名=值
```

语法说明：

1. 等号（=）两边不能有空格。
2. 变量名必须以英文字母或下划线开头，后跟英文字母、下划线或数字（同C语言、Python变量的命名规则）。
3. 变量区分大小写，即：`A` 和 `a` 是两个不同的变量。
4. Shell 的保留字（又称作关键字），含有特殊的含义，不能作为变量名。
5. Shell 变量绑定的值的默认类型是字符串。

Shell 中的保留字（关键字）

```
!       do      esac    in
{       done    fi      then
}       elif    for     until
case    else    if      while
```

如:

```
name="张三"
age=18
```

> name 和 age 绑定的都是字符串

变量取值

变量取值是根据变量名来获取变量绑定的值。

取值语法

```
$变量名
# 或
${变量名}
```

如:

```
weimingze@mzstudio:~$ name="张三"
weimingze@mzstudio:~$ age=18
weimingze@mzstudio:~$ echo $name
张三
weimingze@mzstudio:~$ echo ${name}
张三
weimingze@mzstudio:~$ echo $age
18
weimingze@mzstudio:~$ echo ${age}
18
weimingze@mzstudio:~$ echo "姓名:${name}, 年龄:$age"
姓名:张三, 年龄:18
weimingze@mzstudio:~$ echo $age+1
18+1
weimingze@mzstudio:~$
```

双引号字符串和单引号字符串

- 双引号：解析变量和特殊字符（如 `$` 等）并保护空白字符。
- 无引号：解析变量和特殊字符（如 `$` 等），不保护空白字符。
- 单引号：原样输出，不解析变量。

如:

```
weimingze@mzstudio:~$ name="张三"
weimingze@mzstudio:~$ age=18
weimingze@mzstudio:~$ echo "姓名:${name}, 年龄:$age"
姓名:张三, 年龄:18
weimingze@mzstudio:~$ echo 姓名:${name}, 年龄:$age
姓名:张三, 年龄:18
weimingze@mzstudio:~$ echo '姓名:${name}, 年龄:$age'
姓名:${name}, 年龄:$age
```

双引号字符串保护空白字符

如:

```
weimingze@mzstudio:~$ ABCD="a b  c    d"
weimingze@mzstudio:~$ echo "$ABCD"
a b  c    d
weimingze@mzstudio:~$ echo "${ABCD}"
a b  c    d
weimingze@mzstudio:~$ echo ${ABCD}
a b c d
weimingze@mzstudio:~$ echo '${ABCD}'
${ABCD}
weimingze@mzstudio:~$
```

删除变量

用 `unset` 命令移除变量（不能删除只读变量）：

```
weimingze@mzstudio:~$ name="张三"
weimingze@mzstudio:~$ age=18
weimingze@mzstudio:~$ echo "姓名:${name}, 年龄:$age"
姓名:张三, 年龄:18
weimingze@mzstudio:~$ unset name
weimingze@mzstudio:~$ echo "姓名:${name}, 年龄:$age"
姓名:, 年龄:18
```

> ${name} 显示为空。

只读变量

使用 readonly 命令设置的变量为只读变量，只读变量不能修改，不能删除。

创建只读变量命令

```
readonly 变量名=值
```

如:

```
weimingze@mzstudio:~$ readonly pi=3.1415
weimingze@mzstudio:~$ echo $pi
3.1415
weimingze@mzstudio:~$ pi=3.14
-bash: pi: readonly variable
weimingze@mzstudio:~$ unset pi
-bash: unset: pi: cannot unset: readonly variable
weimingze@mzstudio:~$ echo $pi
3.1415
```

使用 `readonly -p` 可以列出所有的只读变量

```
weimingze@mzstudio:~$ readonly -p
declare -r BASHOPTS="checkwinsize:cmdhist:complete_fullquote:expand_aliases:extg
lob:extquote:force_fignore:globasciiranges:globskipdots:histappend:interactive_c
omments:login_shell:patsub_replacement:progcomp:promptvars:sourcepath"
declare -ar BASH_VERSINFO=([0]="5" [1]="2" [2]="21" [3]="1" [4]="release" [5]="x
86_64-pc-linux-gnu")
declare -ir EUID="1000"
declare -ir PPID="9022"
declare -r SHELLOPTS="braceexpand:emacs:hashall:histexpand:history:interactive-
comments:monitor"
declare -ir UID="1000"
declare -r pi="3.1415"
```

默认值处理

在变量取值时，如果没有此变量，取值可以使用默认值。

语法

```
${变量:-默认值}
```

如:

```
weimingze@mzstudio:~$ echo ${myundef_var}

weimingze@mzstudio:~$ echo ${myundef_var:-"mydefault_value"}
mydefault_value
weimingze@mzstudio:~$
```

2. read 命令

read 命令用于从**标准输入**或**文件描述符**中读取一行数据，并将其分割后赋值给指定的变量。

命令格式

```
read [选项] [变量名1] [变量名2] ...
```

常用选项

选项	说明。
-a	将读取的单词依次赋值给数组 array 的成员。
-d	用指定的 `delimiter` 作为输入行的结束标志，而不是换行符。
-e	使用 readline 库来获取输入（提供行编辑功能）。
-i	在使用 readline 时，将 text 作为初始输入文本。
-n	读取 nchars 个字符后返回，而不是等待整行输入。
-N	准确读取 nchars 个字符（忽略分隔符）。
-p	显示提示信息 prompt，不换行。
-r	原始读取，不解释反斜杠转义字符。
-s	静默模式，不显示输入的内容（如密码输入）。
-t	设置超时秒数，超时后返回非零状态码。
-u	从文件描述符 fd 中读取，而不是标准输入。

示例:

```
weimingze@mzstudio:~$ read name
weimingze
weimingze@mzstudio:~$ echo "${name}"
weimingze
weimingze@mzstudio:~$ read name age
laowei      35
weimingze@mzstudio:~$ echo "${name} 的年龄是 ${age}"
laowei 的年龄是 35
weimingze@mzstudio:~$ read -p "请输入您的名字: " name
请输入您的名字: weimingze
weimingze@mzstudio:~$ echo $name
weimingze
weimingze@mzstudio:~$ read -t 3 -p "请3秒内输入您的名字: " name
请3秒内输入您的名字: weimingze@mzstudio:~$ echo $name

weimingze@mzstudio:~$
```

3. 局部变量和环境变量

局部变量

局部变量是只能在当前 Shell 内使用的变量，此变量不能传递给自己的子进程。

环境变量（全局变量）

环境变量也称全局变量，是可以被自己的子进程继承和使用的变量。

创建和修改局部变量的语法

```
变量名=值
```

创建和修改环境变量的语法

- 方法1，使用 export 命令直接创建环境变量。

```
export 变量名=值
```

- 方法2，先创建局部变量，再使用 export 命令修改为环境变量。

```
变量名=值

export 变量名
```

如：

```
weimingze@mzstudio:~$ lvar="我是局部变量"
weimingze@mzstudio:~$ export gvar="我是环境变量"
weimingze@mzstudio:~$ gvar2="我是环境变量2"
weimingze@mzstudio:~$ export gvar2
weimingze@mzstudio:~$ echo "lvar=${lvar} gvar=${gvar} gvar2=${gvar2}"
lvar=我是局部变量 gvar=我是环境变量 gvar2=我是环境变量2
```

测试局部变量和全局变量的取值

编写 `test_var.sh` 文件并写入如下内容:

```
#!/bin/bash

echo "lvar=${lvar}"
echo "gvar=${gvar}"
echo "gvar2=${gvar2}"
```

添加读取和执行权限

```
weimingze@mzstudio:~$ chmod +rx test_var.sh
```

在此 *bash* 内执行

```
weimingze@mzstudio:~$ source test_var.sh
lvar=我是局部变量
gvar=我是环境变量
gvar2=我是环境变量2
weimingze@mzstudio:~$ . test_var.sh
lvar=我是局部变量
gvar=我是环境变量
gvar2=我是环境变量2
```

在新的 bash 进程内执行

```
weimingze@mzstudio:~$ bash test_var.sh
lvar=
gvar=我是环境变量
gvar2=我是环境变量2
weimingze@mzstudio:~$ ./test_var.sh
lvar=
gvar=我是环境变量
gvar2=我是环境变量2
```

可见局部变量没有传递给子进程。

查看所有环境变量

`printenv` 命令用来打印所有的环境变量。如：

```
weimingze@mzstudio:~$ printenv
SHELL=/bin/bash
PWD=/home/weimingze
LOGNAME=weimingze
HOME=/home/weimingze
LANG=C.UTF-8
USER=weimingze
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/
usr/local/games:/snap/bin
gvar=我是环境变量
gvar2=我是环境变量2
...
```

说明

环境变量	值说明
SHELL	当前使用的 Shell。
PWD	当前工作路径。
LOGNAME	当前登录用户名。
HOME	当前的用户主目录。
LANG	当前使用的语言。
USER	用户名。
PATH	命令的搜索路径列表，用英文的 冒号 : 分隔。
gvar	自定义环境变量。
...	其他(略)。

4. 特殊 Shell 变量

在 Shell 脚本中，有一些特殊变量用于存储特定的信息，比如脚本参数、进程状态等。

Shell 中的特殊变量

变量	值说明
$0	脚本文件名。
$1 ~$9	第 1-9 个参数。
${10} ~${nn}	第 10 到 nn 个参数（nn是大于 10 的整数）。
$#	传递给脚本或函数的参数个数（数字）。
$*	所有参数（合并成一个字符串）。
$@	所有参数（独立字符串-数组）。
$?	上一条命令的退出状态。
$$	当前 shell 的 PID。
$!	最后一个后台进程的 PID。
$_	上一个命令的最后一个参数。

示例

写一个 Shell 脚本文件: `echo_special_var.sh` 写入如下信息:

```
#!/bin/bash
echo "参数的个数是\$#: $#"
echo "脚本文件名是\$0: $0"
echo "参数\$1~\${14}: $1,$2,$3,$4,$5,$6,$7,$8,$9,${10},${11},${12},${13},${14}"
echo "所有参数合并成字符串\$*: $*"
echo "所有参数合并成字符串\$@: $@"

mkdir /wei  # 在根文件夹下创建文件夹
echo "上一个命令的退出值\$?: $?"
mkdir ~/wei   # 在我的用户主目录下创建文件夹
echo "上一个命令的退出值\$?: $?"
echo "当前Shell 的PID:\$\$: $$"

/usr/bin/python3 always_run.py &
echo "最后一个后台进程的PID \$!: $!"
touch mynote.txt
echo "上一个命令 touch mynote.txt 的最后一个参数\$_: $_"
```

在当前 *bash* 进程内运行结果

```
weimingze@mzstudio:~$ source  echo_special_var.sh aaa bbb ccc ddd eee fff ggg
hhh iii jjj kkk lll mmm nnn
参数的个数是$#: 14
脚本文件名是$0: -bash
参数$1~${14}: aaa,bbb,ccc,ddd,eee,fff,ggg,hhh,iii,jjj,kkk,lll,mmm,nnn
所有参数合并成字符串$*: aaa bbb ccc ddd eee fff ggg hhh iii jjj kkk lll mmm nnn
所有参数合并成字符串$@: aaa bbb ccc ddd eee fff ggg hhh iii jjj kkk lll mmm nnn
mkdir: cannot create directory ‘/wei’: Permission denied
上一个命令的退出值$?: 1
上一个命令的退出值$?: 0
当前Shell 的PID:$$: 10127
最后一个后台进程的PID $!: 10263
上一个命令 touch mynote.txt 的最后一个参数$_: mynote.txt
weimingze@mzstudio:~$ PID: 10263

weimingze@mzstudio:~$
```

在新的 *bash* 进程内内运行结果

```
weimingze@mzstudio:~$ bash  echo_special_var.sh aaa bbb ccc ddd eee fff ggg hhh
iii jjj kkk lll mmm nnn
参数的个数是$#: 14
脚本文件名是$0: echo_special_var.sh
参数$1~${14}: aaa,bbb,ccc,ddd,eee,fff,ggg,hhh,iii,jjj,kkk,lll,mmm,nnn
所有参数合并成字符串$*: aaa bbb ccc ddd eee fff ggg hhh iii jjj kkk lll mmm nnn
```

```
所有参数合并成字符串$@: aaa bbb ccc ddd eee fff ggg hhh iii jjj kkk lll mmm nnn
mkdir: cannot create directory ‘/wei’: Permission denied
上一个命令的退出值$?: 1
上一个命令的退出值$?: 0
当前Shell 的PID:$$: 10284
最后一个后台进程的PID $!: 10289
上一个命令 touch mynote.txt 的最后一个参数$_: mynote.txt
weimingze@mzstudio:~$ PID: 10289

weimingze@mzstudio:~$ ./echo_special_var.sh aaa bbb ccc ddd eee fff ggg hhh iii
jjj kkk lll mmm nnn
参数的个数是$#: 14
脚本文件名是$0: ./echo_special_var.sh
参数$1~${14}: aaa,bbb,ccc,ddd,eee,fff,ggg,hhh,iii,jjj,kkk,lll,mmm,nnn
所有参数合并成字符串$*: aaa bbb ccc ddd eee fff ggg hhh iii jjj kkk lll mmm nnn
所有参数合并成字符串$@: aaa bbb ccc ddd eee fff ggg hhh iii jjj kkk lll mmm nnn
mkdir: cannot create directory ‘/wei’: Permission denied
上一个命令的退出值$?: 1
上一个命令的退出值$?: 0
当前Shell 的PID:$$: 10301
最后一个后台进程的PID $!: 10305
上一个命令 touch mynote.txt 的最后一个参数$_: mynote.txt
weimingze@mzstudio:~$ PID: 10305

weimingze@mzstudio:~$
```

命令的退出状态

在 Linux/UNIX 中，任何一个进程在退出时都有一个返回值，通常这个是 0 ~ 255 的的范围(8bit表示)，通常 0 代表成功（真值），其他值代表失败（假值）。

在 Shell 中，使用特殊变量 `$?` 可以 返回上一条命令的进程退出时的值。

第十二章、标准输入输出

任何一个程序都有不变的三个通道: 标准输入、标准输出 和 标准错误输出。 默认 标准输入 是键盘，标准输出 和 标准错误输出 是打印在终端上。

1. printf 命令

printf 命令是 Shell 中一个用于数据格式化成标准输出的内置命令。 printf 命令实现了类似于 C 语言中的 printf() 函数的功能。

printf 命令比 echo 命令具有更灵活的格式控制。

命令格式

```
printf 格式字符串 [参数 ]
```

示例:

```
weimingze@mzstudio:~$ printf "hello world\n"
hello world
weimingze@mzstudio:~$ printf "name:%s, age:%d\n" "weimingze" 35
name:weimingze, age:35
```

其中 printf 将第二个参数 "weimingze" 放在了 %s 占位符处，将第三个参数 35 放在了 %d 占位符处。并进行输出。

printf 命令将内容打印成标准输出，但在末尾并不会打印换行符（\n）。如果需要换行时需要在格式字符串内添加换行符。

printf 格式化字符串内可以使用的转义字符

转义字符	说明
`\"`	双引号
`\\`	反斜杠
`\a`	响铃
`\b`	退格
`\c`	停止后续输出
`\e`	ESC 键
`\f`	换页
`\n`	新行
`\r`	回车
`\t`	水平制表符
`\v`	垂直制表符
`\NNN`	NNN 是1~3位八进制值表示的一个字节
`\xHH`	HH 是1~2位十六进制值表示的一个字节
`\uHHHH`	HHHH 是4位十六进制值表示的一个Unicode (ISO/IEC 10646) 字符
`\UHHHHHHHH`	HHHHHHHH 是8位十六进制值表示的一个Unicode字符

格式化字符串中的 `%s` 和 `%d` 是占位符，`s` 和 `d` 转换符

占位符和转换符

占位符和转换符	说明
%s	转换为字符串。
%d	转换为有符号十进制整数。
%o	转换为有符号八进制数。
%x	转换为有符号十六进制数（小写）。
%X	转换为有符号十六进制数（大写）。
%e	转换为浮点指数格式（小写）。
%E	转换为浮点指数格式（大写）。
%f	转换为浮点十进制格式。
%F	转换为浮点十进制格式。
%g	浮点格式。 如果指数小于 -4 或不小于精度则使用小写指数格式，否则使用十进制格式。
%G	浮点格式。 如果指数小于 -4 或不小于精度则使用大写指数格式，否则使用十进制格式。
%%	不转换参数，在结果中输出一个 '%' 字符。

示例:

```
weimingze@mzstudio:~$ printf "https://%s.%s\n" "weimingze" "com"
https://weimingze.com
weimingze@mzstudio:~$ printf "%d\n" 1000
1000
weimingze@mzstudio:~$ printf "%o\n" 1000
1750
weimingze@mzstudio:~$ printf "%x\n" 1000
3e8
weimingze@mzstudio:~$ printf "%X\n" 1000
3E8
weimingze@mzstudio:~$ printf "%e\n" 12345.67890123
1.234568e+04
weimingze@mzstudio:~$ printf "%E\n" 12345.67890123
1.234568E+04
weimingze@mzstudio:~$ printf "%f\n" 12345.67890123
12345.678901
weimingze@mzstudio:~$ printf "%F\n" 12345.67890123
12345.678901
weimingze@mzstudio:~$ printf "%g\n" 12345.67890123
12345.7
```

```
weimingze@mzstudio:~$ printf "%g\n" 123456789.67890123
1.23457e+08
weimingze@mzstudio:~$ printf "%G\n" 123456789.67890123
1.23457E+08
weimingze@mzstudio:~$ printf "百分比：%d%%\n" 99
百分比：99%
```

% 和 转换符 之间的旗标语法

```
%[- + 0 宽度.精度]转换符
```

说明

- `-`：左对齐(默认是右对齐)；
- `+`：显示正号；
- `0`：左侧空白位置补零；
- 宽度：整个数据输出的宽度；
- 精度：保留小数点后多少位(默认6位)。

示例:

```
weimingze@mzstudio:~$ printf "|%d|\n" 99
|99|
weimingze@mzstudio:~$ printf "|%5d|\n" 99
|   99|
weimingze@mzstudio:~$ printf "|%8d|\n" 99
|      99|
weimingze@mzstudio:~$ printf "|%-8d|\n" 99
|99      |
weimingze@mzstudio:~$ printf "|%-+8d|\n" 99
|+99     |
weimingze@mzstudio:~$ printf "|%-+8d|\n" -99
|-99     |
weimingze@mzstudio:~$ printf "|%-+8d|\n" 99
|+99     |
weimingze@mzstudio:~$ printf "|%08d|\n" 99
|00000099|
weimingze@mzstudio:~$ printf "|%f|\n" 3.1415926897
|3.141593|
weimingze@mzstudio:~$ printf "|%5.2f|\n" 3.1415926897
| 3.14|
weimingze@mzstudio:~$ printf "|%5.3f|\n" 3.1415926897
|3.142|
```

综合示例-打印表格

编写文件`print_table.sh`，写入如下内容：

```
FMT="| %10s | %3d |  %3d  |\n"
printf "+------------+-----+-------+\n"
printf "|    name    | age | score |\n"
printf "+------------+-----+-------+\n"
printf "$FMT" zhang3 18 100
printf "$FMT" li4 18 88
printf "$FMT" laowei 35 60
printf "+------------+-----+-------+\n"
```

运行结果如下：

```
weimingze@mzstudio:~$ vi print_table.sh
weimingze@mzstudio:~$ source print_table.sh
+------------+-----+-------+
|    name    | age | score |
+------------+-----+-------+
|     zhang3 |  18 |  100  |
|        li4 |  18 |   88  |
|     laowei |  35 |   60  |
+------------+-----+-------+
```

2. 输入输出重定向

标准输入重定向 是指用一个文件的内容代替键盘输入。

输出重定向 是指将 `标准输出` 、 `标准错误输出` 输出到文件来代替打印到屏幕上。

标准输入重定向

语法格式

```
命令 ... < 文件路径
```

> ... 代表选项和参数。

示例：

`head -2` 命令默认是将键盘输入的内容的前两行打印到屏幕上，当输入文件结束符（`ctrl + d`）时表示输入完成。如：

```
weimingze@mzstudio:~$ tail -2
this is first line.  # 这一行是键盘输入
this is second line. # 这一行是键盘输入
```

```
this is third line.  # 这一行是键盘输入，回车后输人 `ctrl + d`
this is second line.  # 这两行是打印输如的内容
this is third line.
```

使用文件 `/etc/passwd` 作为键盘输入。

```
weimingze@mzstudio:~$ tail -2 < /etc/passwd
weimingze:x:1000:1000:weimingze:/home/weimingze:/bin/bash
sshd:x:122:65534::/run/sshd:/usr/sbin/nologin
```

> 用法：当有程序需要复杂的输入时，可以将这些输入内容按格式写入文件中，然后再使用输入重定向来代替键盘输入以节省时间。

标准输出重定向

语法格式

```
命令 ... > 文件路径
# 或
命令 ... >> 文件路径
```

标准错误输出重定向

语法格式

```
命令 ... 2> 文件路径
# 或
命令 ... 2>> 文件路径
```

重定向操作符

操作符	说明
`< 文件路径`	标准输入重定向，用指定文件内容代替输入。
`> 文件路径`	标准输出重定向，将输出重定向到指定文件（清空原有内容）。
`>> 文件路径`	标准输出重定向，将输出重定向到指定文件（追加新内容）。
`2> 文件路径`	标准错误输出重定向，将输出重定向到指定文件（清空原有内容）。
`2>> 文件路径`	标准错误输出重定向，将输出重定向到指定文件（追加新内容）。
`&> 文件路径`	将标准输出与错误输出的内容全部重定向到指定文件。

示例

将 `grep` 和 `tail -2` 命令的输出重定向到文件 `result.txt`

```
weimingze@mzstudio:~$ grep "root" /etc/passwd > result.txt
weimingze@mzstudio:~$ cat result.txt
root:x:0:0:root:/root:/bin/bash
nm-openvpn:x:121:122:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/
sbin/nologin
weimingze@mzstudio:~$ grep "root" /etc/passwd > result.txt
weimingze@mzstudio:~$ cat result.txt
root:x:0:0:root:/root:/bin/bash
nm-openvpn:x:121:122:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/
sbin/nologin
weimingze@mzstudio:~$ tail -2 /etc/group >> result.txt
weimingze@mzstudio:~$ cat result.txt
root:x:0:0:root:/root:/bin/bash
nm-openvpn:x:121:122:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/
sbin/nologin
gnome-initial-setup:x:985:
weimingze:x:1000:
```

将 `find /etc/ -name "passwd"` 命令的标准输出重定向到文件 `find.txt`, 将标准错误输出重定向到文件 `error.txt`。

```
weimingze@mzstudio:~$ find /etc/ -name "passwd"
/etc/pam.d/passwd
find: ‘/etc/sssd’: Permission denied
find: ‘/etc/polkit-1/rules.d’: Permission denied
find: ‘/etc/credstore’: Permission denied
find: ‘/etc/vmware-tools/GuestProxyData/trusted’: Permission denied
/etc/passwd
find: ‘/etc/cups/ssl’: Permission denied
find: ‘/etc/ssl/private’: Permission denied
```

```
find: ‘/etc/credstore.encrypted’: Permission denied
weimingze@mzstudio:~$ find /etc/ -name "passwd" > find.txt 2> error.txt
weimingze@mzstudio:~$ cat find.txt
/etc/pam.d/passwd
/etc/passwd
weimingze@mzstudio:~$ cat error.txt
find: ‘/etc/sssd’: Permission denied
find: ‘/etc/polkit-1/rules.d’: Permission denied
find: ‘/etc/credstore’: Permission denied
find: ‘/etc/vmware-tools/GuestProxyData/trusted’: Permission denied
find: ‘/etc/cups/ssl’: Permission denied
find: ‘/etc/ssl/private’: Permission denied
find: ‘/etc/credstore.encrypted’: Permission denied
```

可见标准输出只有如下两行（即正确结果）：

```
/etc/pam.d/passwd
/etc/passwd
```

将 `find /etc/ -name "passwd"` 命令的标准输出和标准错误输入重定向到一个文件 `all.txt`。

```
weimingze@mzstudio:~$ find /etc/ -name "passwd" &> all.txt
weimingze@mzstudio:~$ cat all.txt
/etc/pam.d/passwd
find: ‘/etc/sssd’: Permission denied
find: ‘/etc/polkit-1/rules.d’: Permission denied
find: ‘/etc/credstore’: Permission denied
find: ‘/etc/vmware-tools/GuestProxyData/trusted’: Permission denied
/etc/passwd
find: ‘/etc/cups/ssl’: Permission denied
find: ‘/etc/ssl/private’: Permission denied
find: ‘/etc/credstore.encrypted’: Permission denied
weimingze@mzstudio:~$
```

`/dev/null` 文件

`/dev/null` 文件是一个特殊的字符设备文件。通常被称为 `空设备`或`黑洞` 文件。向此文件写入的所有数据会被丢弃（不会占用存储空间），从此文件读取数据会立即提示文件结束并返回。

如：

创建 `1.txt` 或清空 文件的内容：

```
weimingze@mzstudio:~$ cat /dev/null > null.txt
weimingze@mzstudio:~$ cat null.txt
weimingze@mzstudio:~$
```

将 `find /etc/ -name "passwd"` 命令的标准输出重定向到文件 `find.txt`, 将标准错误输出丢弃。

```
weimingze@mzstudio:~$ find /etc/ -name "passwd" > find.txt 2> /dev/null
weimingze@mzstudio:~$ cat find.txt
/etc/pam.d/passwd
/etc/passwd
```

特殊的字符设备文件

设备文件	说明
`/dev/null`	丢弃所有数据，读取返回空。
`/dev/zero`	读取时返回无限的零字节（`\x00`）。
`/dev/random`	读取时返回随机数的字节（阻塞式）。

特殊输出重定向

操作符	说明
`>&2`	**标准输出** 重定向为 **标准错误输出**，如：`echo "error!" >&2`。
`2>&1`	**标准错误输出** 重定向到 **标准输出**，如：`mkdir /weimingze 2>&1`。

3. 命令替换

命令替换是指执行一个命令，返回此命令的标准输出的字符串。将此字符串替换到当前位置。

字符串替换可以用来替换当前位置的命令、参数以及创建变量时的值等。

命令替换语法

```
`命令 ...`
# 或
$(命令 ...)
```

> `...` 代表选项和参数
>
> `$(命令 ...)` 是比较新的语法，支持嵌套调用

示例

```
weimingze@mzstudio:~$ CUR_PATH=`pwd`
weimingze@mzstudio:~$ echo $CUR_PATH
/home/weimingze
weimingze@mzstudio:~$ CUR_PATH2=$(pwd)
weimingze@mzstudio:~$ echo ${CUR_PATH2}
/home/weimingze
weimingze@mzstudio:~$
```

找到 `/etc/` 文件夹下所有名字叫 `passwd` 的文件(包含子文件夹)，然后详细列出这些文件的信息

```
weimingze@mzstudio:~$ ls -l `find /etc/ -name "passwd" 2> /dev/null`
-rw-r--r-- 1 root root   92 Feb 22  2024 /etc/pam.d/passwd
-rw-r--r-- 1 root root 2919 May 22 11:44 /etc/passwd
weimingze@mzstudio:~$ ls -l $(find /etc/ -name "passwd" 2> /dev/null)
-rw-r--r-- 1 root root   92 Feb 22  2024 /etc/pam.d/passwd
-rw-r--r-- 1 root root 2919 May 22 11:44 /etc/passwd
```

> 使用 `命令 ...` 和 `$(命令 ...)` 都能达到同样的效果。

`$(命令 ...)` 内部还可以嵌套命令，比如 `$(命令 $(命令2 ...))` 是合法的。

注意事项：

命令替换可以用于双引号（`"`）字符串中，但不能用于单引号（`'`）字符串中，如:

```
weimingze@mzstudio:~$ MYSHELL="ls -l `tail -1 /etc/shells`"
weimingze@mzstudio:~$ echo ${MYSHELL}
ls -l /usr/bin/dash
weimingze@mzstudio:~$ MYSHELL="ls -l $(tail -1 /etc/shells)"
weimingze@mzstudio:~$ echo ${MYSHELL}
ls -l /usr/bin/dash
weimingze@mzstudio:~$ MYSHELL='ls -l `tail -1 /etc/shells`'
weimingze@mzstudio:~$ echo ${MYSHELL}
ls -l `tail -1 /etc/shells`
weimingze@mzstudio:~$ MYSHELL='ls -l $(tail -1 /etc/shells)'
weimingze@mzstudio:~$ echo ${MYSHELL}
ls -l $(tail -1 /etc/shells)
weimingze@mzstudio:~$
```

4. 管道

管道（Pipe）是 Shell 中能够将一个命令的输出直接作为另一个命令的输入的强大的功能，它允许多个命令的串联协同操作。

管道的符号是竖线（`|`）。

作用：

1. 将一个进程的标准输出作为下一个命令的标准输入，实现数据流的连续处理。
2. 避免创建临时文件，提高效率。
3. 将多个简单的命令通过数据流组合在一起，完成复杂的功能，多个进程组合在一起形成一个作业。

语法格式

```
命令1 ... | 命令2 ... | 命令3 ...
```

如：

取 `/etc/passwd` 文件中，含有 `root` 的所有行中的最后一行。

```
weimingze@mzstudio:~$ cat /etc/passwd | grep "root" | tail -1
nm-openvpn:x:121:122:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/
sbin/nologin
```

第十三章、Shell 逻辑运算

在 Shell 中，任何一条命令的退出都有 **真** 或 **假** 两种布尔状态。 Shell 中用一个命令返回 `0` 代表真，返回其他值(1~255) 代表假。

在 Shell 中，可以 在命令执行后使用 特殊变量 `$?` 来获取最后一个命令的返回值。

true 命令和 false 命令

在 Shell 中有两个内置的命令 true 和 false 用来确定的返回真值和假值。

- `true` 返回真值(`0`)。
- `false` 返回假值(`1`)。

如:

```
weimingze@mzstudio:~$ true
weimingze@mzstudio:~$ echo $?
0
weimingze@mzstudio:~$ false
weimingze@mzstudio:~$ echo $?
1
```

1. test 命令

test 命令的作用是 **检测文件的类型** 或 **比较两个值** 并返回检测的布尔状态。

命令的语法

```
test 表达式
# 或
[ 表达式 ]
```

语法说明

- `[ 表达式 ]` 是 test 命令的另一种形式，作用相同。
- 使用 `[ 表达式 ]` 作为测试命令时，左括号后和有括号前必须留有空格（第一个左括号是命令，最后一个右括号是参数）。

常用文件表达式

表达式	说明
`-f <FILE>`	文件存在并且是一个普通文件
`-d <FILE>`	文件存在并且是一个文件夹
`-e <FILE>`	文件存在（不区分文件和文件夹）
`-r <FILE>`	文件存在并且用户有 **读** 权限
`-w <FILE>`	文件存在并且用户有 **写** 权限
`-x <FILE>`	文件存在并且用户有 **执行** 权限或进入权限（文件夹）
`-s <FILE>`	文件存在并且文件长度大于零
`-b <FILE>`	文件存在并且是块设备文件
`-c <FILE>`	文件存在并且是字符设备文件
`-L <FILE>` 或 `-h <FILE>`	文件存在并且是一个符号链接文件
`-p <FILE>`	文件存在并且是一个命名管道文件
`<FILE1> -ot <FILE2>`	文件1 比 文件2 旧（修改时间）

示例

```
weimingze@mzstudio:~$ test -f /etc/passwd
weimingze@mzstudio:~$ echo $?
0
weimingze@mzstudio:~$ [ -f /etc/passwd ]
weimingze@mzstudio:~$ echo $?
0
weimingze@mzstudio:~$ [ -d /etc/passwd ]
weimingze@mzstudio:~$ echo $?
1
weimingze@mzstudio:~$ [ -d /etc/ ]
weimingze@mzstudio:~$ echo $?
0
weimingze@mzstudio:~$ test -r /etc/shadow  # shadow 文件不允许普通用户读取
weimingze@mzstudio:~$ echo $?
1
```

字符串和整数比较表达式

表达式	说明
-n <字符串>	字符串的长度不是零（非空字符串）
-z <字符串>	字符串的长度是零（""）
<字符串1> = <字符串2>	判断两个字符串相等
<字符串1> != <字符串2>	判断两个字符串不等
<整数1> -eq <整数2>	整数1 等于 整数2
<整数1> -ge <整数2>	整数1 大于等于 整数2
<整数1> -gt <整数2>	整数1 大于 整数2
<整数1> -le <整数2>	整数1 小于等于 整数2
<整数1> -lt <整数2>	整数1 小于 整数2
<整数1> -ne <整数2>	整数1 不等于 整数2

> 注意字符和整数两侧都要留有空格，因为这些都是参数（需要用空格分开）。

示例

```
weimingze@mzstudio:~$ [ -n "hello" ]
weimingze@mzstudio:~$ echo $?
0
weimingze@mzstudio:~$ [ -n "" ]
weimingze@mzstudio:~$ echo $?
1
weimingze@mzstudio:~$ [ -z "" ]
weimingze@mzstudio:~$ echo $?
0
weimingze@mzstudio:~$ [ 'ABC' == 'abc' ]
weimingze@mzstudio:~$ echo $?
1
weimingze@mzstudio:~$ [ 100 -gt 200 ]
weimingze@mzstudio:~$ echo $?
1
weimingze@mzstudio:~$ [ 100 -lt 200 ]
weimingze@mzstudio:~$ echo $?
0
```

常用逻辑运算表达式

表达式	说明
<表达式1> -a <表达式2>	表达式1 和 表达式2 两者都为真，结果才为真，否则结果为假
<表达式1> -o <表达式2>	表达式1 和 表达式2 只要有一个为真，结果旧为真，两者都为假结果才为假

示例

```
weimingze@mzstudio:~$ [ 200 -gt  100 -a "ABC" != "abc" ]
weimingze@mzstudio:~$ echo $?
0
weimingze@mzstudio:~$ [ 200 -gt  100 -a "ABC" == "abc" ]
weimingze@mzstudio:~$ echo $?
1
```

> 要了解更多的表达式请参见手册 `man test`。

2. expr 命令

expr 命令是用于表达式求值的命令，主要用于执行基本的数学运算、字符串操作和逻辑比较。

expr 命令用来求表达式的值，并将最终结果打印至标准输出。

命令的语法

```
expr 表达式
```

常用表达式

表达式	说明
整数比较	
<整数1> `<` <整数2>	整数1 小于 整数2(`<` 需要转义写成 `\<`)
<整数1> `<=` <整数2>	整数1 小于等于 整数2(`<=` 需要转义写成 `\<=`)
<整数1> `=` <整数2>	整数1 等于 整数2
<整数1> `!=` <整数2>	整数1 不等于 整数2
<整数1> `>=` <整数2>	整数1 大于等于 整数2(`>=` 需要转义写成 `\>=`)
<整数1> `>` <整数2>	整数1 大于 整数2(`>` 需要转义写成 `\>`)
整数算数运算	
<整数1> `+` <整数2>	计算 整数1 加上 整数2 的和
<整数1> `-` <整数2>	计算 整数1 减去 整数2 的差
<整数1> `*` <整数2>	计算 整数1 乘以 整数2 的积
<整数1> `/` <整数2>	计算 整数1 除以 整数2 的商（整数）
<整数1> `%` <整数2>	求余数，求 整数1 除以 整数2 的余数
字符串运算	
<字符串> `:` <正则表达式>	返回 正则表达式 在 字符串 中匹配到的子串
`match` <字符串> <正则表达式>	返回 正则表达式 在 字符串 中匹配到的子串（同上）
`substr` <字符串> <起始位置整数> <长度整数>	返回 字符串 的子串，起始位置是从 1 开始的整数。
`index` <字符串> <子字符串>	返回 子字符串 在 字符串 中出现的起始位置（从1开始），失败返回 0。
`length` <字符串>	返回 字符串 长度。

说明

- 在进行比较运算时返回布尔值，真值（true）用 `1` 表示，假值（false）用 `0` 表示。
- expr 只能进行整数算数运算，运算的返回值也是整数（用字符串表示）。

示例

```
weimingze@mzstudio:~$ expr 100 \< 200
1
weimingze@mzstudio:~$ expr 100 \<= 200
1
weimingze@mzstudio:~$ expr 100 = 200
0
weimingze@mzstudio:~$ expr 100 != 200
1
weimingze@mzstudio:~$ expr 100 + 200
300
weimingze@mzstudio:~$ expr 100 - 200
-100
weimingze@mzstudio:~$ expr 100 \* 200
20000
weimingze@mzstudio:~$ expr 14 / 3
4
weimingze@mzstudio:~$ expr 14 % 3
2
weimingze@mzstudio:~$ expr match "hello.py" "\w*.py"
8
weimingze@mzstudio:~$ expr match "hello.sh" "\w*.py"
0
weimingze@mzstudio:~$ expr length "hello"
5
weimingze@mzstudio:~$ expr index "hello" "e"
2
weimingze@mzstudio:~$ expr substr "hello" 2 3
ell
```

3. 逻辑运算

逻辑运算

逻辑运算又叫布尔运算，是一种基于布尔值（true或false）进行的运算，它们用于根据条件的逻辑关系来评估表达式。

逻辑运算有三种运算：

- 逻辑与运算，运算符：（&&）
- 逻辑或运算，运算符：（||）
- 逻辑非运算，命令：（!）

逻辑与运算

逻辑与运算是两个参与运算的布尔值都为真(true) 时结果才为真(true)，否则结果为假(false)。

语法格式:

```
命令1 && 命令2
```

短路运算

短路运算是指一旦结果确定，将放弃后续的求值，直接返回结果。

> Shell 中的逻辑与运算是短路运算，一旦 `命令1` 的结果为假值，则放弃 `命令2` 的执行，直接返回 **假值**。否则继续运行 `命令2` 并返回 `命令2` 的结果。

示例：

如果创建文件夹成功就复制文件，然后查看命令运行的返回结果。

```
weimingze@mzstudio:~$ mkdir /weimingze && cp /etc/passwd /weimingze  # cp 命令没有执行
mkdir: cannot create directory ‘/weimingze’: Permission denied
weimingze@mzstudio:~$ echo $?
1
weimingze@mzstudio:~$ mkdir ~/test && cp /etc/passwd ~/test  # cp 命令执行并复制的文件
weimingze@mzstudio:~$ echo $?
0
weimingze@mzstudio:~$ ls ~/test
passwd
```

逻辑或运算

逻辑或运算是两个参与运算的布尔值都为假(false) 时结果才为假(false)，否则结果为真(true)。

语法格式:

```
命令1 || 命令2
```

Shell 中的逻辑或运算是短路运算，一旦 `命令1` 的结果为真值，则放弃 `命令2` 的执行，直接返回 **真值**。否则继续运行 `命令2` 并返回 `命令2` 的结果。

示例：

创建文件夹 `/weimingze` 如果不成功就在当前工作路径创建文件夹 `weimingze`，并返回命令执行的最终状态值

```
weimingze@mzstudio:~$ mkdir /weimingze || mkdir weimingze
mkdir: cannot create directory ‘/weimingze’: Permission denied
weimingze@mzstudio:~$ echo $?
0
weimingze@mzstudio:~$ ls
snap  weimingze  下载  公共  图片  文档  桌面  模板  视频  音乐
```

> 前面的 `mkdir /weimingze` 命令失败，后面的 `mkdir weimingze` 命令执行了，否则就不会执行 `mkdir weimingze` 命令了。

逻辑非运算

逻辑非运算 是将执行的命令的结果取 **逻辑非** 操作（即**真**变为**假**，**假**变为**真**）。

Shell 逻辑非 是通过 叹号（英文 `!`）命令实现的。

语法格式:

```
! 命令 ...
```

示例

如果当前不存在文件 `mynote.txt` 则创建此文件，并将 `/etc/passwd` 的最后两行写入该文件。并查看最终程序返回的结果（`$?` 的值）。

```
weimingze@mzstudio:~/test$ ls
weimingze@mzstudio:~/test$ ! [ -f mynote.txt ] && tail -2 /etc/passwd > mynote.t
xt
weimingze@mzstudio:~/test$ echo $?
0
weimingze@mzstudio:~/test$ ls
mynote.txt
weimingze@mzstudio:~/test$ cat mynote.txt
weimingze:x:1000:1000:weimingze:/home/weimingze:/bin/bash
sshd:x:122:65534::/run/sshd:/usr/sbin/nologin
```

逻辑与、逻辑或、逻辑非混合运算

逻辑与、逻辑或和逻辑非运算可以任意组合。大致语法如下：

语法格式

```
命令1 && 命令 && 命令3 || 命令4 || ! 命令5 && ....
```

> 可以根据业务逻辑需要任意组合。

4. 顺序执行

在 Shell 中一个命令或一组命令组成的作业任务通常写在一行内，后续命令需要另起一行。如果要在一行内写入顺序执行的多个命令则需要使用 命令终止符（英文的分号`;`） 来分隔各个命令。

语法格式:

```
命令1; 命令2; 命令3 ...
```

如:

```
weimingze@mzstudio:~$ mkdir mydoc; cd mydoc; touch a.txt; ls -l a.txt
-rw-rw-r-- 1 weimingze weimingze 0 May 30 16:27 a.txt
```

> 上述命令会顺序执行，当一个命令结束并让出终端后，后续命令才开始执行。同写入多行有一样的效果。

第十四章、Shell 的分支结构

分支命令

分支命令是可以根据不同命令的返回结果来执行不同的命令块（命令组）的命令。

分支命令允许 Shell 程序根据条件判断来做出决策，从而实现更复杂和灵活的逻辑。

Shell 中的分支命令:

- if 命令
- case 命令

1. if 命令

if 命令用于基于条件执行不同的代码块。它是控制流程的基本结构之一，允许脚本根据测试条件的结果做出决策。

if 命令让 Shell 程序根据条件选择性的执行其中的某一个命令块。

官方语法

```
if COMMANDS; then COMMANDS; [ elif COMMANDS; then COMMANDS; ]... [ else
COMMANDS; ] fi
```

上述 中括号（`[ ]`）内部的数据是可选的。后面跟 `...` 说明前面的可选项可以有零次、一次或多次。if 命令可以写在一行内，也可以写在多行便于编写者阅读程序。

将上述一行整理成多行，并用中文表达需要填充的部分，即成为如下语法：

```
if 命令1; then
    命令块1
elif 命令2; then
    命令块2
... # 此处可以有多个 elif-then 子命令.
elif 命令n; then
    命令块n
else
    命令块（其他）
fi
```

if 命令的语法中，if 主命令和 elif 子命令中的 then 如果写在一行则需要再命令后面加一个 命令终止符 `;`，如:`if 命令1; then`。如果将 then 写到下一行，则不需要在命令后加命令终止符。

即 if 语法，也可以写成

```
if 命令1
then
    命令块1
elif 命令2
then
    命令块2
... # 此处可以有多个 elif-then 子命令.
elif 命令n
then
    命令块n
else
    命令块（其他）
fi
```

> 这里的 **命令** 通常是 测试命令 `test` （`[ ]`）。

语法说明

1. `if`、`then`、`elif`、`else`、`fi` 是关键字。
2. `if - then` 主命令必须存在。
3. `elif - then` 是子命令，可以有0个、1个或多个。
4. `else` 子命令只能有一个或者没有，且只能放在最后。
5. `fi` 是 if 命令的结束标志。
6. 命令块是由一个或多个命令组成，多个命令可以写在一行用命令终止符（`;`）分开，也可以写在多行内。
7. 上述语法每行前的空格（或水平制表符）缩进仅是便于阅读代码，没有特殊含义（不同于 Python 编程语言的缩进规则）。

执行顺序说明

它自上而下通过对 `then` 前的命令逐个运行并检查返回值状态，直至找到一个真值，然后找到 then 后面唯一匹配的一个命令块，然后执行该命令块。

如果上述的测试命令都为假值，则执行 `else` 子句中的命令块（如果有 else 子句）。

示例

- 判断你的电脑有没有 `/home/weimingze` 这个文件夹。如果有则在 `result.txt` 中写入 `存在用户主目录 /home/weimingze`

编写一个文件 `test_if.sh`，写入如下内容：

```
if [ -d "/home/weimingze" ]; then
    echo "存在用户主目录 /home/weimingze" > result.txt
fi
```

我的运行结果如下：

```
weimingze@mzstudio:~$ vim test_if.sh
weimingze@mzstudio:~$ bash test_if.sh
weimingze@mzstudio:~$ cat result.txt
存在用户主目录 /home/weimingze
```

- 判断你的电脑有没有 `/home/weimz` 这个文件夹。如果有则在 `result.txt` 中写入 `存在用户主目录 /home/weimz`，如果没有则写入 `不存在用户主目录 /home/weimz`

修改文件 `test_if.sh`，写入如下内容：

```
#!/bin/bash

if [ -d "/home/wemz" ]; then
    echo "存在用户主目录 /home/weimz" > result.txt
else
    echo "不存在用户主目录 /home/weimz" > result.txt
fi
```

我的运行结果如下：

```
weimingze@mzstudio:~$ vim test_if.sh
weimingze@mzstudio:~$ bash test_if.sh
weimingze@mzstudio:~$ cat result.txt
不存在用户主目录 /home/weimz
```

- 判断你的用户主目录下是否有 `mybackup` 和 `mydocs` 两个文件夹。如果存在 `mybackup` 则将 `result.txt` 复制其中，如果不存在则再测试是否有 `muydocs` 文件夹，如果存在 `muydocs` 则将 `result.txt` 复制其中，否则打印 `备份失败。`

修改文件 `test_if.sh`，写入如下内容：

```
if test -d "~/mybackup" ; then
    cp "result.txt" "~/mybackup"
    echo "备份成功: ~/mybackup"
elif [ -d "~/mydocs" ]; then
    cp "result.txt" "~/mydocs"
    echo "备份成功: ~/mydocs"
else
    echo "备份失败"
fi
```

我的运行结果如下：

```
weimingze@mzstudio:~$ ls
result.txt  snap  test_if.sh  下载  公共  图片  文档  桌面  模板  视频  音乐
weimingze@mzstudio:~$ bash test_if.sh
备份失败
```

退出状态

if 命令是一条可以写成多行的命令，他的退出状态是最后执行的命令的退出状态。

2. case 命令

case 命令是根据 **值** 来匹配不同的多个样式中的一个，如果匹配成功，则执行样式对应的命令块。他主要用来简化 **if 命令** 的结构，同样可以实现多分支的执行路径。

case 命令似于 Python 语言中的 `match` 语句 和 C 语言中的 `switch` 语句。

官方语法

```
case WORD in [PATTERN [| PATTERN]...) COMMANDS ;;]... esac
```

上述 中括号（`[ ]`）内部的数据是可选的。后面跟 `...` 说明前面的可选项可以有零次、一次或多次。case 命令可以写在一行内，也可以写在多行便于编写者阅读程序。

将上述一行整理成多行，并用中文表达需要填充的部分，即成为如下语法：

语法格式:

```
case 字符串值 in
    样式1[|样式2]... )
        命令块 ;;
    样式3[|样式4]... )
```

```
        命令块 ;;
    样式1[|样式2]... )
        命令块 ;;
esac
```

语法说明:

- `case`、`in`、`esac` 是关键字。
- `esac` 是 `case` 命令结束的标记。
- 右括号 `)` 是 **样式** 结束的分隔符，后面是命令块。
- 两个分号 `;;` 是命令块的终止符。
- 样式中的 `|` 表示 **或** 关系，即 `样式1 | 样式2` 表示 `样式1` 或 `样式2` 中的一个匹配即可满足匹配条件。

执行顺序说明

case 命令会根据 字符串值 去下面样式列表中找寻匹配的样式，如果样式匹配成功，则执行此样式对应的命令块后结束此 case 命令的执行。

示例:

写一个 Shell 程序 `test_case.sh`，此程序输入一年中的月份（1~12），打印这个月是四季：春、夏、秋、冬中的那个季节。

写入程序如下：

```
#!/bin/bash

read -p "请输入月份: " SEASON

case $SEASON in
    1 | 2 | 3)
        echo "春季!"
        ;;
    4 | 5 | 6)
        echo "夏季!"
        ;;
    7 | 8 | 9)
        echo "秋季!"
        ;;
    10 | 11 | 12)
        echo "冬季!"
        ;;
    *)
        echo "您输入的月份有误!"
        ;;
esac
```

运行结果:

```
weimingze@mzstudio:~$ bash test_case.sh
请输入月份: 12
冬季!
weimingze@mzstudio:~$ bash test_case.sh
请输入月份: 2
春季!
weimingze@mzstudio:~$ bash test_case.sh
请输入月份: 999
您输入的月份有误!
```

退出状态

case 命令的退出状态是最后执行的命令的退出状态。

第十五章、Shell 的循环结构

1. for 命令

for 命令在 shell 脚本中用于遍历给定的字符串列表，绑定变量后再执行命令块。

for 命令又叫迭代循环命令，他只能对优先的数据集合进行遍历，比如字符串列表等。

名词解释

- **迭代** 是指重复的做某件事。
- **遍历** 是指所有的数据都经历且只经历一遍。

for 命令的作用

遍历一组给定的字符串列表（使用空白字符分隔，如：空格，换行等）。

官方语法

```
for NAME [in WORDS ... ] ; do COMMANDS; done
```

上述 中括号（`[ ]`）内部的数据是可选的。`WORDS` 后跟 `...` 说明前面的 字符串 `WORDS` 可以有零个、一个或多个。 `WORDS ...` 后面的分号 `;` 表示 字符串列表的结束，如果后面 `do` 写在新的一行内，则 分号`;` 可以省略。

将上述一行整理成多行，并用中文表达需要填充的部分，即成为如下语法：

语法格式:

```
for 变量名 [in 字符串1 字符串2 ... ; ] do
    命令块;
done
```

或

```
for 变量名 [in 字符串1 字符串2 ...]
do
    命令块;
done
```

语法说明:

- `for`、`in`、`do`、`done` 是关键字。
- in 后面的字符串是选项，用空白字符分开。
- for 命令默认所有数据遍历完毕后才结束循环。可以使用 `break` 命令可以终止当前 for 命令的执行。

示例:

写一个 Shell 程序 `test_for.sh`，实现遍历给定的字符串:

```
#!/bin/bash
for STRING in AAA BBB "CCC ccc" DDD
do
    echo ${STRING}
done
```

执行结果

```
weimingze@mzstudio:~$ bash test_for.sh
AAA
BBB
CCC ccc
DDD
```

示例2

写一个 Shell 程序 `test_for2.sh`，遍历 `/etc/` 文件夹下所有名为 `passwd` 的文件，并将其内容合并到文件当前文件夹下的 `passwd.txt` 文件。

```
#!/bin/bash
for PATHNAME in $(find /etc/ -name "passwd" 2> /dev/null)
do
    echo "find: ${PATHNAME}"
    cat ${PATHNAME} >> passwd.txt
done
```

执行结果

```
weimingze@mzstudio:~$ bash test_for2.sh
find: /etc/pam.d/passwd
find: /etc/passwd
weimingze@mzstudio:~$ cat passwd.txt
#
# The PAM configuration file for the Shadow `passwd' service
#
```

```
@include common-password

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
```

2. while 命令

while 命令用户创建一个循环运行一组命令（命令块）的结构，会根据执行 `测试命令` 的测试结果（即退出状态）来决定是否重复执行 `命令块`。如果测试结果为真则继续执行命令块，当测试结果为假（即退出状态为非零值）时终止循环。

官方语法

```
while COMMANDS; do COMMANDS-2; done
```

`COMMANDS` 后面的分号 `;` 表示 命令的终止符，如果后面 `do` 写在新的一行内，则 分号`;` 可以省略。

将上述一行整理成多行，并用中文表达需要填充的部分，即成为如下语法：

语法格式:

```
while 测试命令; do
    命令块;
done
```

或

```
while 测试命令
do
    命令块;
done
```

语法说明:

- `while`、`do`、`done` 是关键字。
- `do` 用于标记命令块的开始，`done` 用于标记命令块的结束。

while命令执行过程说明:

1. 先执行测试命令，根据测试命令的退出状态来决定是否执行命令块。
2. 如果测试命令的退出状态为假值（非零值）则直接退出此 while 命令，while 循环执行结束。
3. 如果测试命令的退出状态为真值（零 `0`）则执行命令块部分的命令。然后回到第一步，再执行测试命令。

示例:

循环打印 1 到 10 的整数。每一行打印一个数字。

写一个 Shell 程序 `test_while.sh`，内容如下：

```
#!/bin/bash

count=1

while [ $count -le 10 ]
do
    echo "$count"
    count=$(expr $count + 1)
done
```

运行结果

```
weimingze@mzstudio:~$ bash test_while.sh
1
2
3
4
5
6
7
8
9
10
```

示例2:

循环读取键盘输入的内容。当直接回车时结束输入，打印输入文字的行数并打印每一行输入的内容。

编写一个 Shell 程序 `test_while2.sh`，内容如下：

```
#!/bin/bash

count=0
content=""
```

```
while read -p '请输入一行文字，直接回车结束输入: ' line && [ `expr length "$line"` -
ne 0 ]
do
    count=$(expr $count + 1)
    content=${content}${line}
done
echo "你输入了 $count 行文字!"
echo "内容是:" $content
```

运行结果：

```
weimingze@mzstudio:~$ bash test_while2.sh
请输入一行文字，直接回车结束输入: hello shell
请输入一行文字，直接回车结束输入: bye
请输入一行文字，直接回车结束输入:
你输入了 2 行文字!
内容是: hello shellbye
```

退出状态

while 命令的退出状态是内部最后执行的一条命令的退出状态。

3. until 命令

until 命令的作用与 while 循环相反，即当条件为**假**时执行循环体，直到条件为**真**时停止（while 命令是：当条件为**真**时执行循环体，直到条件为**假**时停止）。

换句话说，until 会一直循环，直到给定的条件返回成功（退出状态为 0）时停止循环。

官方语法

```
until COMMANDS; do COMMANDS-2; done
```

`COMMANDS` 后面的分号 `;` 表示 命令的终止符，如果后面 `do` 写在新的一行内，则 分号`;` 可以省略。

将上述一行整理成多行，并用中文表达需要填充的部分，即成为如下语法：

语法格式:

```
until 测试命令; do
    命令块;
done
```

或

```
until 测试命令
do
    命令块;
done
```

语法说明:

- `until`、`do`、`done` 是关键字。
- `do` 用于标记命令块的开始，`done` 用于标记命令块的结束。

until命令执行过程说明:

1. 先执行测试命令，根据测试命令的退出状态来决定是否执行命令块。
2. 如果测试命令的退出状态为真值（零值 `0`）则直接退出此 until 命令，until 循环执行结束。
3. 如果测试命令的退出状态为假值（非零值 ）则执行命令块部分的命令。然后回到第一步，再执行测试命令。

示例:

循环打印 1 到 10 的整数。每一行打印一个数字。

写一个 Shell 程序 `test_until.sh`，内容如下:

```
#!/bin/bash

count=1

until [ $count -gt 10 ]
do
    echo "$count"
    count=$(expr $count + 1)
done
```

运行结果

```
weimingze@mzstudio:~$ bash test_until.sh
1
2
3
4
5
6
7
8
```

```
9
10
```

退出状态

until 命令的退出状态是内部最后执行的一条命令的退出状态。

4. select 命令

select 命令提供选择性输入的命令，他提供一种交互式菜单和并循环让用户选择输入功能，它允许用户从一组选项中进行选择其中的一个，并将选择对应的值交给变量，并做后续的工作。

作用

1. 创建一个简单的文本菜单界面。
2. 自动显示带编号的选项列表。
3. 等待用户输入选择。
4. 将用户选择的值赋给变量。

官方语法

```
select NAME [in WORDS ... ;] do COMMANDS; done
```

上述 中括号（`[ ]`）内部的数据是可选的。`WORDS` 后跟 `...` 说明前面的 字符串 `WORDS` 可以有零个、一个或多个。 `WORDS ...` 后面的分号 `;` 表示 字符串列表的结束，如果后面 `do` 写在新的一行内，则 分号`;` 可以省略。

将上述一行整理成多行，并用中文表达需要填充的部分，即成为如下语法：

语法格式:

```
select 变量名 [in 字符串1 字符串2 ... ; ] do
    命令块;
done
```

或

```
select 变量名 [in 字符串1 字符串2 ...]
do

```

```
    命令块;
done
```

语法说明:

- `select`、`in`、`do`、`done` 是关键字。
- in 后面的字符串是选项，用空白字符分开。
- select 命令默认循环输入信息，使用 `break` 命令可以终止当前 select 命令的执行。
- select 命令的输入提示语用 `PS3` 变量值，默认是 `#?`。

示例:

写一个 Shell 程序 `test_select.sh` ，实现如下的四个功能:

1. 创建 `select.txt` 文件
2. 删除 `select.txt` 文件
3. 查看当前的所有文件
4. 退出此功能

写入程序如下:

```
#!/bin/bash

PS3="请输入你的选项项: "   # 设置选择提示符

select option in "创建文件" "删除文件" "列出所有文件" "退出"
do
    case $option in
        "创建文件")
            touch select.txt && echo "成功创建文件 select.txt"
            ;;
        "删除文件")
            rm select.txt && echo "成功删除文件 select.txt"
            ;;
        "列出所有文件")
            ls
            ;;
        "退出")
            echo "退出..."
            break
            ;;
        *)
            echo "不可用的选项"
            ;;
    esac
done
```

运行效果如下：

```
weimingze@mzstudio:~$ bash test_select.sh
1) 创建文件
2) 删除文件
3) 列出所有文件
4) 退出
请输入你的选项项: 1
成功创建文件 select.txt
请输入你的选项项: 2
成功删除文件 select.txt
请输入你的选项项: 2
rm: cannot remove 'select.txt': No such file or directory
请输入你的选项项: 1
成功创建文件 select.txt
请输入你的选项项: 3
select.txt  snap  test_select.sh  下载  公共  图片  文档  桌面  模板  视频  音乐
请输入你的选项项: 4
退出...
weimingze@mzstudio:~$
```

5. break 命令

break 命令用于 终止循环命令(for、while、until 命令) 或 select 命令的执行。当 break 命令执行时，他会终止包含他的当前命令执行并退出。

命令格式

```
break [n]
```

语法说明

- break 命令只能用于 for 命令、while 命令、until 命令 或 select 命令的内部。
- 参数 n 是 break 命令跳出循环的层数，默认为1。当有循环嵌套时可以使用 2 或以上的层数（依据需求来定）。

示例

写一个程序 `test_break.sh`，此程序输入一系列整数，当输入 0 时结束输入。打印这些数的个数。这些数的总和。

```
#!/bin/bash
```

```
count=0
total=0

while read -p "请输入一个整数: " number
do
    if [ "$number" -eq 0 ]; then
        break
    fi
    count=`expr $count + 1`
    total=`expr $total + "$number"`
done

echo "数字个数: $count"
echo "数字的和: $total"
```

执行结果

```
weimingze@mzstudio:~$ bash test_break.sh
请输入一个整数: 1
请输入一个整数: 2
请输入一个整数: 3
请输入一个整数: 4
请输入一个整数: 0
数字个数: 4
数字的和: 10
```

6. continue 命令

continue 命令用于循环命令(for、while、until 命令) 或 select 命令的内部。他用于终止当前循环体内部，剩余部分的命令的执行，重新开始一次新的循环。

命令格式

```
continue [n]
```

语法说明

- continue 命令只能用于 for 命令、while 命令、until 命令 或 select 命令的内部。
- 参数 n 是 continue 命令外层循环嵌套的层数，默认为1。。

示例

写一个程序 `test_continue.sh`，此程序输入一系列整数代表学生的成绩，当输入 `-1` 时结束输入。统计成绩在 60 ~ 100 分的人数，打印这些数的人数。

```
#!/bin/bash

count=0

while read -p "请输入一个整数: " score
do
    if [ "$score" -le -1 ]; then
        break
    fi
    if [ "$score" -lt 60 ]; then
        continue
    fi
    if [ "$score" -gt 100 ]; then
        continue
    fi
    count=`expr $count + 1`
done

echo "成绩合格的人数: $count"
```

执行结果

```
weimingze@mzstudio:~$ bash test_continue.sh
请输入一个整数: 90
请输入一个整数: 20
请输入一个整数: 80
请输入一个整数: 30
请输入一个整数: -1
成绩合格的人数: 2
```

第十六章、Shell 函数

什么是函数

函数是一段具有特定功能的命令块，它允许你为这段命令块定义一个名称（即函数名），并通过这个名称来调用这段命令。

函数一次定义就可以重复调用。

1. 函数的定义和调用

函数定义的官方语法

```
function name { COMMANDS ; }
# 或者
name () { COMMANDS ; }
```

将上述一行整理成多行，即成为如下语法：

语法格式:

```
function 自定义函数名 {
    命令块
}
```

或

```
自定义函数名 {
    命令块
}
```

语法说明:

- 函数定义是将命令打包在一起，给这些命令取一个名字，供后续使用。
- 函数名的定义方法同变量名一致。
- 函数定义时，内部的命令并不会执行，只有函数在调用时函数内部的命令才会执行。

函数调用的语法

```
函数名 [参数1] [参数2] ...
```

语法调用语法说明:

- 函数调用时，程序会跳转到函数定义处，执行函数名对应的函数内部的命令块，当命令块执行完毕后返回到函数调用处继续执行。
- 函数调用时可以向函数传递参数，这些参数需要使用空白字符分隔，在函数内部可以使用位置变量 （`$1`、`$2`、`$3`等）来获取这些参数值。
- 在函数内部 可是使用 `$#` 变量来获取函数参数的个数。
- 函数内部的 变量: `$#`、`$1`、`$2`、`$3`等位置变量时函数内部的局部变量，他不会影响函数外面同名的位置变量的值。

示例:

定义一个控制全自动洗衣机洗衣服流程的函数 washing_machine，用它来提示洗衣服的全过程（这里我们使用 echo 命令代替控制过程）。

```
washing_machine() {
    echo "放入衣服"
    echo "注水"
    echo "洗涤20分钟"
    echo "排水"
    echo "甩干"
    echo "报警提示完成"
}
```

函数调用

```
washing_machine
```

执行结果

```
放入衣服
注水
洗涤20分钟
排水
甩干
报警提示完成
```

上述的洗衣机程序只能洗涤 `衣服`, 我们可以重新定义 washing_machine 函数，让其能接收一个参数（洗涤的物品）。

改写上述程序，加入一个形参 $1，让此函数能够洗涤不同的衣物。

```
washing_machine() {
    echo "放入 $1"
    echo "注水"
    echo "洗涤20分钟"
    echo "排水"
    echo "甩干"
    echo "报警提示完成"
}

#  调用函数 washing_machine 并传入对应的参数
washing_machine "羽绒服"
washing_machine "床单"
```

执行结果

```
放入 羽绒服
注水
洗涤20分钟
排水
甩干
报警提示完成
放入 床单
注水
洗涤20分钟
排水
甩干
报警提示完成
```

这样我们的函数就能根据不同的参数执行不同的操作了。

2. return 命令

return 命令用于函数的内部，他的作用是终止当前函数的执行，不再执行函数中 return 后面的代码，返回到调用此函数的地方，同时返回一个状态值（0~255的值）。

命令格式

```
return [n]
```

命令说明

- return 后面不跟参数，则返回return 之前一条命令执行后的状态值（即 `$?`）。

- return 后跟参数必须是 0~255 之间的数来表示此函数的执行状态（如果 n 的值不在 0~255 之间，则求 n 对 256 求余数的值）
- return 命令返回的值可以在函数调用处用特殊变量 `$?` 来获取。
- 当函数调用需要返回更多信息时，可以向标准输出打印字符串。在调用此函数的地方使用 **命令替换** 就可以获取此字符串了。

示例

测试 由参数的 return 命令的执行和返回值。

```
function myfunc() {
    echo "函数开始"
    return 20
    echo "函数结束"
}

myfunc
echo "函数的返回值是 $?"
```

执行结果

```
函数开始
函数的返回值是 20
```

测试没有参数的 return 命令的执行和返回值。

```
function myfunc() {
    echo "函数开始"
    test 100 = 200
    return
    echo "函数结束"
}

myfunc
echo "函数的返回值是 $?"
```

执行结果

```
函数开始
函数的返回值是 1
```

使用 命令替换 返回字符串信息示例

```
function myadd() {
    temp=`expr $1 + $2`
    echo $temp
}

result=$(myadd 10000 20000)
echo $result
```

执行结果

```
30000
```

3. exit 命令

exit 命令使用来终止当前脚本程序的（结束当前进程），并返回一个退出状态码给调用者（终端或父进程）。

普通 shell 程序执行完毕最后一条命令才能退出，使用 exit 命令可以随时结束执行并退出。

命令格式

```
exit [n]
```

命令说明

- 无论 exit 命令是在函数中、循环中，都会结束整个程序的运行。
- exit 后面不跟参数，则返回 exit 之前一条命令执行后的状态值（即 `$?`）。
- exit 命令 后跟参数必须是 表示此脚本程序的退出状态，通常 `0` 表示成功，其他值表示失败。此状态值 n 必须是 0~255 之间的数值（如果 n 的值不在 0~255 之间，则求 n 对 256 求余数的值）。
- 父进程可以通过特殊变量 `$?` 来获取该状态值。

示例:

编写脚本文件 `test_exit.sh` 测试 exit 命令中途退出 Shell 程序，内容如下：

```
echo "程序开始"
exit 10
echo "程序结束"
```

执行结果

```
weimingze@mzstudio:~$ bash test_exit.sh
程序开始
weimingze@mzstudio:~$ echo $?
10
```

示例2

编写文件 `make_chapter_dir.sh`，此程序有一个整数的参数表示章节，创建一系列文件夹，如 `./make_chapter_dir.sh 15`，程序则创建 `chapter_15/video`、`chapter_15/mp4`、`chapter_15/code` 三个文件夹。

```
if [ $# -ne 1 ]; then
    echo -e "USAGE:\n\t./make_chapter_dir.sh xx"
    exit 1;
fi
# 使用printf 命令格式化字符串
chapter_dir=`printf "chapter_%02d\n" $1`
echo "mkdir" $chapter_dir
mkdir -p ${chapter_dir}/video
mkdir -p ${chapter_dir}/mp4
mkdir -p ${chapter_dir}/code
exit 0
```

执行结果

```
weimingze@mzstudio:~$ chmod +x make_chapter_dir.sh
weimingze@mzstudio:~$ ./make_chapter_dir.sh
USAGE:
    ./make_chapter_dir.sh xx
weimingze@mzstudio:~$ ./make_chapter_dir.sh 14
mkdir chapter_14
weimingze@mzstudio:~$ ./make_chapter_dir.sh 15
mkdir chapter_15
weimingze@mzstudio:~$ echo $?
0
weimingze@mzstudio:~$ tree .
.
├── chapter_14
│   ├── code
│   ├── mp4
│   └── video
├── chapter_15
│   ├── code
│   ├── mp4
│   └── video
├── make_chapter_dir.sh
...
└── 音乐
```

附录

Ubuntu 换源

什么是源

Ubuntu 的 源（Software Sources） 是指系统用于获取软件包（包括应用程序、库、系统更新等）的服务器地址连接。这些源服务器存储了 Ubuntu 官方及其社区维护的软件包，用户通过配置不同的源来安装、更新和管理软件。

在新安装完软件后 Ubuntu Linux 系统后，使用 apt 命令安装软件时，下载速度可能很慢；通常新安装的系统的下载源是国外的服务器。其实我们可以使用国内的镜像服务器下载，这时我们就需要更换源。

为什么要换源?

Ubuntu 默认连接国外的源服务器，我们在使用 apt 命令安装软件时速度比较慢，这时我们可以更换成国内的软件源服务器。更换国内源服务器后我们使用 apt 命令安装软件的时候，速度就会大大增加，节省时间。

配置软件包源的位置：

Ubuntu 18.04 ~ 22.04 的源配置文件的位置为:

```
/etc/apt/sources.list
```

Ubuntu 24.04 及以后版本的位置为:

```
/etc/apt/sources.list.d/ubuntu.sources
```

以下我们说一下各个版本换源的方法。

1. Ubuntu 24.04 换源

Ubuntu 24.04 将源的地址从 `/etc/apt/sources.list` 更换成 `/etc/apt/sources.list.d/ubuntu.sources`

还原步骤如下：

1. 备份原有的源

```
cd /etc/apt/sources.list.d
sudo cp ubuntu.sources ubuntu.sources.bak
```

2. 使用 vim 编辑源文件

```
sudo vim /etc/apt/sources.list.d/ubuntu.sources
```

在 vim 中删除原来所有的内容，粘帖如下文本内容(任选其一)。

阿里云的源：

```
Types: deb
URIs: http://mirrors.aliyun.com/ubuntu/
Suites: noble noble-updates noble-security
Components: main restricted universe multiverse
Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
```

清华的源：

```
Types: deb
URIs: http://mirrors.tuna.tsinghua.edu.cn/ubuntu/
Suites: noble noble-updates noble-security
Components: main restricted universe multiverse
Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
```

中科大的源：

```
Types: deb
URIs: http://mirrors.ustc.edu.cn/ubuntu/
Suites: noble noble-updates noble-security
Components: main restricted universe multiverse
Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
```

网易 163 的源：

```
Types: deb
URIs: http://mirrors.163.com/ubuntu/
Suites: noble noble-updates noble-security
Components: main restricted universe multiverse
Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
```

换源后执行如下命令，来更新 Ubuntu 软件包的索引（元数据）

```
sudo apt update
```

准备就绪，可以使用 apt 命令来安装软件了，速度那叫一个快！

2. Ubuntu 22.04 换源

还原步骤如下：

1. 备份原有的源

```
cd /etc/apt/
sudo cp sources.list sources.list.bak
```

2. 使用 vim 编辑源文件

```
sudo vim /etc/apt/sources.list
```

在 vim 中删除原来所有的内容，粘帖如下文本内容(下列源任选其一)。

阿里云的源：

```
deb http://mirrors.aliyun.com/ubuntu/ jammy main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ jammy-updates main restricted universe mul
tiverse
deb http://mirrors.aliyun.com/ubuntu/ jammy-backports main restricted universe m
ultiverse
deb http://mirrors.aliyun.com/ubuntu/ jammy-security main restricted universe mu
ltiverse
deb http://mirrors.aliyun.com/ubuntu/ jammy-proposed main restricted universe mu
ltiverse
```

清华的源：

```
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy main restricted universe
multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-updates main restricted u
niverse multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-backports main
restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-security main restricted
universe multiverse
```

华为云镜像源:

```
deb https://mirrors.huaweicloud.com/ubuntu/ jammy main restricted universe
multiverse
deb https://mirrors.huaweicloud.com/ubuntu/ jammy-security main restricted unive
rse multiverse
deb https://mirrors.huaweicloud.com/ubuntu/ jammy-updates main restricted univer
se multiverse
deb https://mirrors.huaweicloud.com/ubuntu/ jammy-backports main restricted univ
erse multiverse
```

中科大镜像源

```
deb https://mirrors.ustc.edu.cn/ubuntu/ jammy main restricted universe
multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ jammy-security main restricted universe
multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ jammy-updates main restricted universe m
ultiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ jammy-backports main restricted
universe multiverse
```

换源后执行如下命令，来更新 Ubuntu 软件包的索引（元数据）

```
sudo apt update
```

准备就绪，可以使用 apt 命令来安装软件了！

3. Ubuntu 20.04 换源

还原步骤如下：

1. 备份原有的源

```
cd /etc/apt/
sudo cp sources.list sources.list.bak
```

2. 使用 vim 编辑源文件

```
sudo vim /etc/apt/sources.list
```

在 vim 中删除原来所有的内容，粘帖如下文本内容(下列源任选其一)。

阿里云的源：

```
deb http://mirrors.aliyun.com/ubuntu/ focal main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ focal-updates main restricted universe mul
tiverse
deb http://mirrors.aliyun.com/ubuntu/ focal-backports main restricted universe m
ultiverse
deb http://mirrors.aliyun.com/ubuntu/ focal-security main restricted universe mu
ltiverse
deb http://mirrors.aliyun.com/ubuntu/ focal-proposed main restricted universe mu
ltiverse
```

清华的源：

```
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal main restricted universe
multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-updates main restricted u
niverse multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-backports main
restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ focal-security main restricted
universe multiverse
```

华为云镜像源:

```
deb https://mirrors.huaweicloud.com/ubuntu/ focal main restricted universe
multiverse
deb https://mirrors.huaweicloud.com/ubuntu/ focal-security main restricted unive
rse multiverse
deb https://mirrors.huaweicloud.com/ubuntu/ focal-updates main restricted univer
se multiverse
deb https://mirrors.huaweicloud.com/ubuntu/ focal-backports main restricted univ
erse multiverse
```

中科大镜像源

```
deb https://mirrors.ustc.edu.cn/ubuntu/ focal main restricted universe
multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ focal-security main restricted universe
multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ focal-updates main restricted universe m
ultiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ focal-backports main restricted
universe multiverse
```

换源后执行如下命令，来更新 Ubuntu 软件包的索引（元数据）

```
sudo apt update
```

准备就绪，可以使用 apt 命令来安装软件了！