

计算程序与数学算法在化学中的应用

学习笔记

何尹铭

2025 年 8 月 5 日

简介

本笔记在课程教学 PPT 的基础上，结合个人的学习经验，适当调整了一些顺序，整合了较分散的内容，每部分内容都形成了“总结”部分。在此基础上，稍微拓展了一些有用的功能。同时，笔记里面包含了大量的代码示例，可以跟着笔记进行实践操作，加深记忆、理解。

水平所限，难免疏漏，如有错误，恳请指正。

目录

0 写在前面	1
0.1 章节结构	1
0.2 字体	1
0.3 内容	2
1 Linux 入门	3
1.1 Linux 简介	3
1.1.1 Linux 的起源	3
1.2 Linux 的特点	3
1.2.1 Linux 优点	3
1.2.2 Linux 操作特点	4
1.3 Linux 的安装	4
1.3.1 虚拟机	4
1.3.2 子系统	5
1.4 Linux 的文件结构与目录相关操作	5
1.4.1 Linux 文件系统结构	5
1.4.2 切换、创建、删除目录	7
1.4.3 文件和目录的复制、移动、重命名以及列出目录里的文件	9
1.5 文件操作命令	14
1.5.1 输入和输出、输出重定向	14
1.5.2 查看文件内容	16
1.5.3 查找文件	18
1.6 用户与文件权限管理	20
1.6.1 用户和用户组	20
1.6.2 文件权限管理	21
1.7 为系统安装软件	23
1.7.1 软件包管理器与软件源	23
1.7.2 安装软件	24
1.7.3 卸载软件	25
1.7.4 课程所需软件	25
1.8 文本处理工具	26

1.8.1	grep 命令	26
1.8.2	awk 命令	28
1.8.3	sed 命令	28
1.9	Vim 文本编辑器	29
1.9.1	Vim 简介与三个模式	29
1.9.2	编辑模式	31
1.9.3	命令、查找模式	31
1.10	与 window 宿主机共享文件	32
2	Fortran 语言简介	33
2.1	编程语言与 Fortran 的发展	33
2.2	Fortran 与打孔卡片	33
2.3	Fortran 语言的编译	36
3	Fortran 基础	38
3.1	Fortran 变量类型声明与转化	38
3.1.1	Fortran 变量类型声明	38
3.1.2	Fortran 变量类型转化	38
3.2	Fortran 程序的输入与输出	39
3.2.1	输入语句	39
3.2.2	输出语句	40
3.3	格式化输出	40
3.3.1	格式化输出语句	40
3.3.2	I 型	41
3.3.3	F 型	42
3.3.4	E 型	42
3.4	数学语句	43
3.4.1	数字与运算符	43
3.4.2	数学函数	43
3.4.3	赋值语句	44
3.4.4	算术运算	44
3.4.5	大型算术表达式	45
3.5	Fortran 程序构成	45

目录	III
A 更换软件源为清华大学镜像站	47
A.1 Ubuntu 24.04.2 LTS 软件源配置	47
A.2 更早或更新版本软件源配置	49
B 与 Window 宿主机共享文件夹	50
B.1 WSL 子系统	50
B.2 虚拟机	50

0 写在前面

0.1 章节结构

本笔记层级为章节、小节、小小节，例如 1、1.1、1.1.1 等。

笔记中代码块以小节为基础，向下编号，与小小节并行，与小小节编号无关。如：

代码 0.1.1 text

1 这是第 0 节第 1 小节第 1 个代码块。

代码 0.1.2 text

1 这是第 0 节第 1 小节第 2 个代码块。

笔记中总结以章节为基础，向下编号，与小节并行，与小节编号无关。如：

总结 0.1 这是第 0 节第 1 小节第 1 个总结。

总结 0.2 这是第 0 节第 1 小节第 2 个总结。

0.2 字体

本笔记正文中文字体为宋体，英文字体为 Times New Roman。

标题字体为思源宋体，页眉中文为楷体，页眉英文为 Times New Roman。

代码块中字体为 Maple 字体。Maple 字体中，字符等宽、显示清晰，需要注意的是 @ 为 “@” 符号。

自己需要在 PPT 或 Word 等软件中展示代码时，推荐使用电脑自带的 Consolas 字体，是英文等宽字体。本笔记为了实现中英文均等宽（一个中文字符严格占两个英文字符空间），使用了 Maple 字体。

代码 0.2.1 text

1 一 二 三 四 五 六 七 八 九 十 百 千 万

2 AbCdEfGhIjKlMnOpQrStUvWxYz

0.3 内容

本笔记基于课堂 PPT 内容整理。对于超出课程范围的内容，笔记中会标出，并且可能会有错误或不准确之处。

课程范围内的内容，笔记中力求准确、清晰、易懂，但水平所限，难免有疏漏或错误之处，注意甄别，并请不吝指出。

1 Linux 入门

1.1 Linux 简介

1.1.1 Linux 的起源

1965 年，美国麻省理工学院（MIT），通用电气公司（GE）以及 AT&T 的贝尔实验室联合开发 Multics 工程计划。

1969 年，贝尔实验室的肯·汤姆森（Kenneth Thompson）在小型机上开发出了 UNIX 系统，1970 年发布。

1971 年，丹尼斯·里奇（Dennis Ritchie）发明了 C 语言，

1973 年，UNIX 系统的绝大部分源代码用 C 语言重写，提高了 UNIX 系统的可移植性。

1991 年，李纳斯·托尔德（Linus Torvalds）开发了 Linux。

目前，Linux 内核已经发展到 6.13.1 版本（主版本号. 次版本号. 末版本号），是 Linux 操作系统的核心。负责管理系统的进程、内存、设备驱动程序、文件和网络系统。

仅内核和工具链对普通用户来说难以安装和使用。为了解决这个问题，发行版 (Distributions) 开始出现。

最早的发行版包括：Softlanding Linux System (SLS)、Slackware (1993)；基于 SLS，至今仍在维护的最古老发行版之一。Debian (1993)：以其严格的自由软件原则、社区治理和庞大的软件仓库而闻名。

1994 年 3 月，Linux 1.0 内核发布，标志着它被认为足够稳定和功能完备。

目前常用的发行版有 Ubuntu、Debian、Fedora、CentOS、Red Hat Enterprise Linux (RHEL)、Arch Linux 等。这份笔记全部基于 Ubuntu24.04.2 LTS 版本。

1.2 Linux 的特点

1.2.1 Linux 优点

Linux 的分层权限体系是其核心优势之一，显著提升了系统安全性与管理效率。

系统用户明确分为两类：

root 用户：拥有最高权限，可执行所有系统操作和管理任务。

普通用户：权限受限，默认仅能访问自身文件，有效防止误操作或恶意软件破坏系统关键部分（需 `sudo` 授权提升权限）。

此外，用户组机制提供了便捷的文件协作方式。管理员可将用户加入特定组，并设置组权限，实现组内成员对共享资源的稳定访问，简化了团队协作中的权限管理。

这种严谨的权限隔离机制，是 Linux 系统高稳定性和安全性的重要基石。

1.2.2 Linux 操作特点

Linux 秉承“一切皆文件”的核心设计理念。整个系统的操作实质上是对文件的管理，例如：硬件设备抽象为 `/dev` 目录下的设备文件；用户空间映射为 `/home` 下的个人目录。这种统一通过文件接口管理软硬件资源的机制，是 Linux 操作的基础特点和强大优势。

1.3 Linux 的安装

现在我们常用的是 Windows 操作系统或 MacOS 操作系统，这时要使用 Linux 的相关功能，就要安装 Linux 操作系统。安装方式有三种：双系统、虚拟机、子系统，其中双系统操作复杂、稳定性差，这里不作介绍。

1.3.1 虚拟机

虚拟机软件是一种特殊的软件，可以模拟具有完整硬件系统功能的、运行在一个完全隔离环境中的完整计算机系统，可使用其所在物理机（即主机系统）的物理资源。

在 Windows 和 MacOS 系统中，我们常用 VMware Workstation Pro / VMware Fusion。这原本是收费软件，被博通收购后逐步开放，到现在已经完全免费了，在官网可下载安装。现在任何收费的都是盗版软件。下载安装流程比较复杂，可以参考[这篇知乎文章](#)。

另外，要下载系统的镜像文件，在虚拟机软件 VMware 中创建虚拟机时需要用到。镜像软件可在发行版官网或者开源镜像网站找到，比如 Ubuntu24.04.2 LTS 版本，可以在[清华大学开源软件镜像站](#)下载，打开后选择“ubuntu-24.04.2-desktop-amd64.iso”下载即可。

都下载完成后，虚拟机具体安装流程可参考[这一篇知乎文章](#)。以上两篇文章建议先通读一遍，了解整体流程，再逐步安装。上面提到的 VMware Workstation Pro 和 VMware Fusion 还有 Ubuntu24.04.2 LTS 镜像文件，我打包了一个压缩包，放在了[学习通网盘](#)，可以直接下载解压后使用。

安装之后，右击桌面，选择“在终端中打开”。运行如下指令（需要输入的是 \$ 之后的部分）：

代码 1.3.1 Bash

```
1 user@host:~$ sudo passwd root
```

首先输入当前用户的密码，然后输入两次要为 root 用户设置的密码，即可设置 root 用户的密码。如果密码过于简单，系统会提示密码不安全，此时不用管它，原样再输入一遍，就能设置成功。

1.3.2 子系统

子系统是指在 Windows 操作系统中运行的 Linux 环境，通常通过 Windows Subsystem for Linux (WSL) 的终端来实现，通常没有桌面环境，只提供命令行界面，如需桌面环境，需要额外配置。

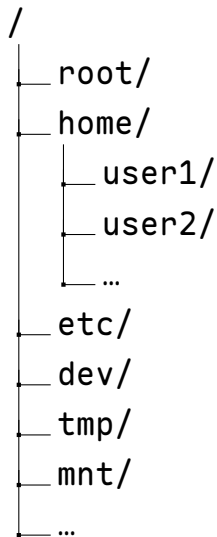
在 Windows 自带的 Microsoft Store 中搜索“Ubuntu”，可以找到多个版本的 Ubuntu 子系统，如 Ubuntu 24.04 LTS 等，选择合适的版本进行安装即可。

具体安装细节可参考[这一篇知乎文章](#)。安装相较于虚拟机简单很多。

1.4 Linux 的文件结构与目录相关操作

1.4.1 Linux 文件系统结构

Linux 的文件系统采用树形结构，所有文件和目录都从根目录（/）开始，根目录下有多个子目录。每个目录下可以包含文件和其他子目录，形成一个层次分明的目录结构。以下是较重要的目录。



其中：

`/`：根目录，与开机、还原、系统修复等操作有关；

`/root`：管理员的主目录；

`/home`：系统默认的用户主目录；

`/bin`：二进制可执行文件，常用命令；

`/sbin`：管理员才能使用的目录，存放系统执行文件；

`/usr`：用户级别目录，存放用户自己的指令，与软件安装执行有关；

`/var`：存放经常变动、与系统运作过程有关的文件，如日志、网页、缓存等；

`/etc`：系统管理所需要的配置文件和子目录；

`/dev`：外部设备文件；

`/lib` 和 `/lib64`：32 位和 64 位软件安装位置，函数库；

`/boot`：启动 Linux 时使用的文件；

`/opt`：放置第三方软件，是默认的软件安装目录

`/tmp`：临时文件目录；

`/mnt`：临时挂载的文件系统，挂载光盘、U 盘等；

我们可以使用 `pwd` 命令（print working directory）查看当前所在目录，如：

代码 1.4.1 Bash

```
1 user1@host:~$ pwd
2 /home/user1
```

运行 `pwd` 给出的结果就是当前所在目录的绝对路径 `/home/user1`。

与绝对路径相对的还有相对路径，绝对路径是从根目录开始的完整路径，而相对路径是相对于当前目录的路径。例如，如果当前目录是 `/home`，则相对路径 `user1` 或 `./user1` 表示 `/home/user1`，其中 `.` 表示当前目录。而绝对路径则是 `/home/user1/test`。

1.4.2 切换、创建、删除目录

如果现在登陆的用户是 `user1`，则主文件夹就是 `/home/user1`，可以通过如下指令进入：

代码 1.4.2 Bash

```
1 user1@host:~$ cd /home/user1
```

或

代码 1.4.3 Bash

```
1 user1@host:~$ cd ~
```

其中 `cd` 就是“change directory”的缩写，表示切换目录；`~` 表示当前用户主文件夹。

创建文件夹用 `mkdir` 命令（make directory），如在当前目录下创建一个名为 `test` 的文件夹，则可以执行：

代码 1.4.4 Bash

```
1 user1@host:~$ mkdir test
```

在 `test` 文件夹中创建一个名为 `test1` 的子目录，在 `test1` 中创建 `test2`、`test3` 目录，则可以执行（可同时创建多个）：

代码 1.4.5 Bash

```
1 user1@host:~$ mkdir ~/test/test1
```

```
2 user1@host:~$ mkdir ~/test/test1/test2 ~/test/test1/test3
```

或直接执行：

代码 1.4.6 Bash

```
1 user1@host:~$ mkdir -p ~/test/test1/test2
```

其中 `-p` 表示递归创建多层级目录。

可以用 `rmdir` 命令（remove directory）删除空目录。

代码 1.4.7 Bash

```
1 user1@host:~$ rmdir ~/test/test1/test3
```

此时 `user1` 目录下有一个名为 `test` 的子目录，其下又有 `test1` 的目录，则可以通过如下指令进入：

代码 1.4.8 Bash

```
1 user1@host:~$ cd ~/test/test1
```

我们采用 `touch` 命令创建文件，如在当前目录下创建一个名为 `1.txt` 的文件，则可以执行：

代码 1.4.9 Bash

```
1 user1@host:~/test/test1$ touch 1.txt
```

也可以直接创建多个文件，如在当前目录下创建 `1.txt` 和 `2.txt` 两个文件，则可以执行：

代码 1.4.10 Bash

```
1 user1@host:~/test/test1$ touch 1.txt 2.txt
```

此时这个文件夹中有 `1.txt` `2.txt` 文件以及 `test2` 目录，进入此文件夹后，则可以通过如下指令查看当前目录下的文件：

代码 1.4.11 Bash

```
1 user1@host:~/test/test1$ ls
2 1.txt 2.txt test2
```

其中 **ls** 就是“list”的缩写，表示列出当前目录下的文件和子目录；运行结果就是当前目录下的文件和子目录列表，在这里就显示为 **1.txt 2.txt test2**。

目录中，**.** 表示当前目录，**..** 表示上一级目录。如目前的状态，要进入 **test2** 目录，可执行：

代码 1.4.12 Bash

```
1 user1@host:~/test/test1$ cd ./test2
```

其中 **.** 即当前目录。如果要返回上一级目录，则可以执行：

代码 1.4.13 Bash

```
1 user1@host:~/test/test1/test2$ cd ..
```

1.4.3 文件和目录的复制、移动、重命名以及列出目录里的文件

我们可以用 **cp** 命令（copy）复制文件，或用 **mv** 命令（move）移动文件或重命名文件。如将 **1.txt** 复制到 **test2** 目录下，则可以执行：

代码 1.4.14 Bash

```
1 user1@host:~/test/test1$ cp 1.txt test2/
```

如把 **1.txt** 复制到 **test2** 目录下，并重命名为 **3.txt**，则可以执行：

代码 1.4.15 Bash

```
1 user1@host:~/test/test1$ cp 1.txt test2/3.txt
```

如给 **1.txt** 重命名为 **.4.txt**，则可以执行：

代码 1.4.16 Bash

```
1 user1@host:~/test/test1$ mv test2/1.txt test2/.4.txt
```

其中 `.4.txt` 是一个隐藏文件，前面有 `.`，在 Linux 中，前面带 `.` 的文件或目录都是隐藏的。

`cp` 命令有 `-r` 选项（recursive），与 `mkdir -r` 类似，表示递归复制目录及其内容。`cp` 命令和 `mv` 命令都可以使用 `-i` 选项（interactive），表示在覆盖文件时进行提示。`cp` 命令和 `mv` 命令也可以复制、移动多个文件，基本结构为：

代码 1.4.17 Bash

```
1 cp [选项] 源文件 1 源文件 2 ... 目标目录
2 mv [选项] 源文件 1 源文件 2 ... 目标目录
```

上文提到，`ls` 命令可以列出当前目录下的文件和子目录，但如果要查看更详细的信息，则可以使用 `ls -l` 命令。其中 `-l` 表示“long”，即长格式，显示更详细的信息，如文件权限、所有者、大小和修改时间等。

查看隐藏文件或文件夹，可用 `ls -a` 命令（all）。展示子文件夹下文件和目录，可用 `ls -R` 命令（注意大写）。

`ls` 除了可以查看当前目录下的文件和子目录，还可以查看其他目录下的文件和子目录。

总结而言，`ls` 命令的基本结构为：

代码 1.4.18 Bash

```
1 ls [选项] [目录]
```

以上命令可用组合使用，例子如下：

代码 1.4.19 Bash

```
1 user1@host:~/test/test1$ ls -a ~/test
2 .  ..  1.txt  2.txt  test2
3
4 user1@host:~/test/test1$ ls -l
5 总计 4
6 -rw-rw-r-- 1 user1 user1    0  7 月  1 12:48 1.txt
7 -rw-rw-r-- 1 user1 user1    0  7 月  1 12:48 2.txt
8 drwxrwxr-x 2 user1 user1 4096  7 月  1 12:47 test2
9
```

```
10 user1@host:~/test/test1$ ls -a
11 .  ..  1.txt  2.txt  test2
12
13 user1@host:~/test/test1$ ls -R
14 .:
15 1.txt  2.txt  test2
16
17 ./test2:
18 1.txt
19
20 user1@host:~/test/test1$ ls -aR
21 .:
22 .  ..  1.txt  2.txt  test2
23
24 ./test2:
25 .  ..  1.txt  .4.txt
26
27 user1@host:~/test/test1$ ls -alR
28 .:
29 总计 12
30 drwxrwxr-x 3 user1 user1 4096 7 月 1 12:48 .
31 drwxrwxr-x 3 user1 user1 4096 7 月 1 12:47 ..
32 -rw-rw-r-- 1 user1 user1 0 7 月 1 12:48 1.txt
33 -rw-rw-r-- 1 user1 user1 0 7 月 1 12:48 2.txt
34 drwxrwxr-x 2 user1 user1 4096 7 月 1 12:53 test2
35
36 ./test2:
37 总计 8
38 drwxrwxr-x 2 user1 user1 4096 7 月 1 12:53 .
39 drwxrwxr-x 3 user1 user1 4096 7 月 1 12:48 ..
40 -rw-rw-r-- 1 user1 user1 0 7 月 1 12:53 1.txt
41 -rw-rw-r-- 1 user1 user1 0 7 月 1 12:53 .4.txt
```

其中长格式显示的内容包括：

- 第一列：文件类型和权限，第一个字符为类型（如 `-` 为一般文件、`d` 为目录），之后三个字符为一组，分别为所有者、用户组、所有用户的权限，`r` 表示可读、`w` 表示可写、`x` 表示可执行。如 `-rwxrw-r--` 表示一般文件，所有者可读可写可执行，用户组内用户可读可写，所有人可读；

- 第二列：硬链接数（指向该文件的链接数量）；
- 第三列：文件所有者（如 `user1`）；
- 第四列：文件所属组（如 `user1`）；
- 第五列：文件大小（以字节为单位）；
- 第六列：最后修改时间（如 `7 月 1 12:48`）；
- 第七列：文件名（如 `1.txt`）。

最后，我们可用使用 `rm` 命令（remove）删除文件或目录，也可以同时删除多个文件或目录。如果要删除 `1.txt` 文件，则可以执行：

代码 1.4.20 Bash

```
1 user1@host:~/test/test1$ rm 1.txt
```

`rm` 命令也有一些可选项，如 `-i` 表示在删除前进行确认，`-r` 表示递归删除目录及其内容，`-f` 表示强制删除，忽略不存在的文件。

如果要删除 `test1` 目录及其内容，则可以执行：

代码 1.4.21 Bash

```
1 user1@host:~/test/test1$ cd ..
2 user1@host:~/test$ rm -r test1
```

总结 1.1 目录和文件相关命令用法

绝对路径是从根目录开始的完整路径，而相对路径是相对于当前目录的路径。

`pwd` 命令用于显示当前工作目录的绝对路径。

`/home/username` 为当前用户的主文件夹，可以用 `~` 表示。

`cd` 命令用于切换目录；`.` 表示当前目录；`..` 表示上一级目录。

`mkdir` 命令用于创建目录；`rmdir` 命令用于删除空目录；`touch` 命令用于创建文件，这三个命令都可以同时操作多个文件或目录。`mkdir` 命令有可选项 `-p`，表示递归创建

多级目录。其命令结构为：

代码 1.4.22 Bash

```
1  mkdir [-p] 目录 1 目录 2 ...
2  rmdir 目录 1 目录 2 ...
3  touch 文件 1 文件 2 ...
```

`cp` 命令用于复制文件或目录；`mv` 命令用于移动文件或目录或重命名文件或目录，这两个命令都可以同时操作多个文件或目录。`cp` 命令有可选项 `-r`，表示递归复制目录及其内容；`cp` 命令和 `mv` 命令有可选项 `-i`，表示在覆盖文件时进行提示。其命令结构为：

代码 1.4.23 Bash

```
1  cp [-r,-i] 源文件 1 源文件 2 ... 目标目录
2  mv [-i] 源文件 1 源文件 2 ... 目标目录
```

`ls` 命令用于列出当前目录下的文件和子目录；`-l` 用于列出当前目录下的文件和子目录的详细信息；`-a` 用于列出当前目录下的所有文件和子目录，包括隐藏文件和目录；`-R` 命令用于递归列出当前目录下的所有文件和子目录。其命令结构为：

代码 1.4.24 Bash

```
1  ls [-l,-a,-R] [目录]
```

`rm` 命令用于删除文件或目录；`-i` 表示在删除前进行确认，`-r` 表示递归删除目录及其内容，`-f` 表示强制删除，忽略不存在的文件。其命令结构为：

代码 1.4.25 Bash

```
1  rm [-i,-r,-f] 文件 1 文件 2 ... 目录 1 目录 2 ...
```

以上命令的选项可组合使用。

注意 1.1 重点

Linux 对大小写敏感，不可混用。

1.5 文件操作命令

前面我们已经用到了 `touch` 命令创建文件，接下来我们介绍一些常用的文件操作命令。

1.5.1 输入和输出、输出重定向

前面我们使用的 `pwd` 命令和 `ls` 命令都是直接输出到终端的，而有时我们希望将输出结果保存到文件中，这时就可以使用输出重定向。

在 Linux 中，可以使用 `>` 符号将命令的输出重定向到文件中，例如：

代码 1.5.1 Bash

```
1 user1@host:~$ ls > list.txt
```

上述命令会将当前目录下的文件和子目录列表保存到 `list.txt` 文件中。如果 `list.txt` 文件已存在，则会被覆盖。

如果希望将输出追加到文件末尾而不是覆盖，可以使用 `>>` 符号，例如：

代码 1.5.2 Bash

```
1 user1@host:~$ ls >> list.txt
```

此外，还可以使用 `2>` 将错误输出重定向到文件，例如：

代码 1.5.3 Bash

```
1 user1@host:~$ ls non_existing_file 2> error.log
```

上述命令会将错误信息保存到 `error.log` 文件中。

通过输出重定向，我们可以方便地将命令的输出结果保存到文件中，以便后续查看和处理。

`echo` 命令能够将文本输出到终端中，可以使用输出重定向将文本保存到文件中，例如：

代码 1.5.4 Bash

```
1 user1@host:~$ echo "Hello, World!"
2 Hello, World!
3 user1@host:~$ echo "Hello, World!" > test/hello.txt
```

如第一条命令会将 `Hello, World!` 输出到终端，而第二条命令会将 `Hello, World!` 保存到 `test/hello.txt` 文件中。如果 `test/hello.txt` 文件不存在，它会被创建；如果 `test/hello.txt` 文件已存在，则会被覆盖。

代码 1.5.5 Bash

```
1 user1@host:~$ echo "I'm Linux." >> test/hello.txt
2 user1@host:~$ echo "I'm also Ubuntu." >> test/hello.txt
```

上述命令会将 `I'm Linux.`、`I'm also Ubuntu.` 追加到 `test/hello.txt` 文件的末尾。此时 `test` 目录下的 `hello.txt` 文件内容为：

代码 1.5.6 Text

```
1 Hello, World!
2 I'm Linux.
3 I'm also Ubuntu.
```

管道符 `|` 用于将一个命令的输出作为另一个命令的输入，这样可以将多个命令组合起来，实现更复杂的操作。一个简单的例子：`pwd` 能给出当前的目录，而 `ls` 能列出当前目录下的文件和子目录，我们可以将这两个命令组合起来，查看当前目录下的文件和子目录列表：

代码 1.5.7 Bash

```
1 user1@host:~$ pwd | ls
```

上述命令会将 `pwd` 的输出作为 `ls` 的输入，列出当前目录下的文件和子目录列表。事实上，上面的命令与直接使用 `ls` 没有区别，因为 `ls` 默认列出当前目录下的文件和子目录，这里只是借此介绍管道符的用法。之后会给出一些实用的例子。

除此之外，还有 `<` `<<` 符号用于将文件或文本内容作为命令的输入。例子在之后给出。

通配符是 Linux 中用于匹配文件名或目录名的特殊字符，可以用来简化命令输入。常用的通配符包括：

`*`：匹配零个或多个字符。例如，`*.txt` 匹配所有以 `.txt` 结尾的文件。

`?`：匹配一个字符。例如，`file?.txt` 匹配 `file1.txt`、`file2.txt` 等。

`[]`：匹配括号内的任意一个字符。例如，`file[1-3].txt` 匹配 `file1.txt`、`file2.txt` 等。

`{ }`: 匹配多个选项。例如, `file{1,2,3}.txt` 匹配 `file1.txt`、`file2.txt`、`file3.txt`。

`[!]`: 匹配不在括号内的字符。例如, `file[!1-3].txt` 匹配 `file4.txt`、`file5.txt` 等。

如:

代码 1.5.8 Bash

```
1 user1@host:~$ rm *.txt
```

上述命令会删除当前目录下所有以 `.txt` 结尾的文件。

1.5.2 查看文件内容

查看文件内容我们可以使用 `cat` 命令 (concatenate), 它可以将文件内容输出到终端。`cat` 命令有 `-n` 选项, 可显示行号。例如:

代码 1.5.9 Bash

```
1 user1@host:~$ cat test/hello.txt
2 Hello, World!
3 I'm Linux.
4 I'm also Ubuntu.
5 user1@host:~$ cat -n test/hello.txt
6 1 Hello, World!
7 2 I'm Linux.
8 3 I'm also Ubuntu.
```

我们使用随机数命令 `shuf` 生成随机数, 这是课外内容, 但在之后非常有用, 可以快速生成测试用的数据。

代码 1.5.10 Bash

```
1 user1@host:~$ shuf -r -n 300 -i 1-100 > test/numbers.txt
```

其中 `-r` 表示允许重复, `-n 300` 表示生成 300 个随机数, `-i 1-100` 表示随机数范围在 1 到 100 之间。上述命令会生成 300 个 1 到 100 之间的随机数, 并将结果保存到 `test/numbers.txt` 文件中。

可以预见，这个文件中会有 300 行，此时使用 `cat` 命令输出到终端查看会非常不方便，我们可以使用 `less` 命令（或 `more` 命令）分页查看文件内容。

代码 1.5.11 Bash

```
1 user1@host:~$ less test/numbers.txt
```

上述命令会打开 `test/numbers.txt` 文件，并在终端分页显示内容。可以使用空格键或 PageDown 向下翻页，用 PageUp 向上翻页（`more` 命令不行）。

还可以使用以下命令查询里面的内容：

`/` 字符串：向下查询字符串

`?` 字符串：向上查询字符串

`n` 重复前一个查询

`q` 退出

如 `/5` 表示向下查询数字 5，会查询到包含 5 的数字，如 15、51、45 等，查询到一个后，可以按 `n` 重复查询。

我们还可以使用 `head` 命令和 `tail` 命令查看文件的前几行或后几行内容，默认为 10 行。例如：

代码 1.5.12 Bash

```
1 user1@host:~$ head test/numbers.txt
```

上述命令会显示 `test/numbers.txt` 文件的前 10 行内容。如果要显示前 20 行内容，可以使用 `-n` 选项：

代码 1.5.13 Bash

```
1 user1@host:~$ head -n 20 test/numbers.txt
```

类似地，使用 `tail` 命令可以查看文件的后几行内容：

另外，这两个命令可以利用管道符 `|`、输出重定向等组合使用，查看文件特定的几行内容。例如：

代码 1.5.14 Bash

```
1 user1@host:~$ head -n 20 test/numbers.txt | tail -n 10
2 user1@host:~$ head -n 20 test/numbers.txt | tail -n 5 >
   test/numbers16-20.txt
```

第一条命令会显示 `test/numbers.txt` 文件的第 11 到 20 行内容；第二条命令会将 `test/numbers.txt` 文件的第 16 到 20 行内容保存到 `test/numbers16-20.txt` 文件中。

1.5.3 查找文件

查找文件可以使用 `find` 命令，它可以在指定目录下查找符合条件的文件或目录。例如，要在 `test` 目录下查找所有以 `.txt` 结尾的文件，可以使用以下命令：

代码 1.5.15 Bash

```
1 user1@host:~$ find ./test -name "*.txt"
2 ./test/numbers.txt
3 ./test/hello.txt
```

上述命令会在 `test` 目录下查找所有以 `.txt` 结尾的文件，并输出其相对路径。

还可以使用 `-user` 选项查找指定用户的文件，例如：

代码 1.5.16 Bash

```
1 user1@host:~$ find ./test -user user1
2 ./test/numbers.txt
3 ./test/hello.txt
```

上述命令会查找 `test` 目录下所有属于用户 `user1` 的文件。

还有其它选项不一一介绍。

查找可执行文件可以使用 `which` 命令，它会显示指定命令的绝对路径。例如：

代码 1.5.17 Bash

```
1 user1@host:~$ which ls
2 /bin/ls
3 user1@host:~$ which find
```

4 /usr/bin/find

ls 和 **find** 命令实际上都是程序，即可执行文件，上述命令会显示 **ls** 和 **find** 命令的绝对路径。

查找命令的二进制可执行文件、源代码和手册页位置，可以使用 **whereis** 命令。它会显示指定命令的二进制程序路径、源代码文件路径（如果存在）和帮助手册页路径。

代码 1.5.18 Bash

```
1 user1@host:~$ whereis ls
2 ls: /bin/ls /usr/share/man/man1/ls.1.gz
3 user1@host:~$ whereis find
4 find: /usr/bin/find /usr/share/man/man1/find.1.gz
   ↪ /usr/share/info/find.info.gz
```

whereis 命令不仅会显示它们的绝对路径，还会一并显示关联的手册页和源代码（若有安装）的路径。

总结 1.2 文件操作命令

> 符号用于将命令的输出重定向到文件中，覆盖原有内容；>> 符号用于将命令的输出追加到文件末尾。2> 用于将错误输出重定向到文件中。| 管道符用于将一个命令的输出作为另一个命令的输入。< 符号用于将文件内容作为命令的输入；<< 符号用于将多行文本作为命令的输入。

、?、[]、{ }、[!] 等符号用于匹配文件名或目录名的特殊字符，称为通配符。 可以匹配零个或多个字符；? 可以匹配一个字符；[] 可以匹配括号内的任意一个字符；{ } 可以匹配多个选项；[!] 可以匹配不在括号内的字符。

cat 命令用于查看文件内容；cat -n 用于显示行号。less 命令用于分页查看文件内容；more 命令也可以分页查看文件内容，不能向前翻页。/string 用于向下查询字符串“string”；?string 用于向上查询字符串“string”；n 重复前一个查询；q 退出。

head -n 用于查看文件的前 n 行内容；tail -n 用于查看文件的后 n 行内容。只用 head 或 tail 则默认查看前 10 行或后 10 行内容。可以使用管道符 | 和输出重定向等组合使用。

find 命令用于查找指定目录下符合条件的文件或目录；-name 选项用于按名称查

找文件或目录；`-user` 选项用于按所有者查找文件或目录。

`which` 命令用于查找命令的绝对路径；`whereis` 命令用于查找命令的二进制可执行文件、源代码和手册页位置。

1.6 用户与文件权限管理

1.6.1 用户和用户组

Linux 系统中的用户和用户组是管理系统资源和权限的基本单位。每个用户都有一个唯一的用户名和用户 ID (UID)，用户组则是由多个用户组成的集合，每个用户组也有一个唯一的组名和组 ID (GID)。

涉及到用户和用户组的管理，通常需要使用 `root` 用户或具有 `sudo` 权限的用户来执行相关操作。`root` 用户是系统的超级用户，拥有最高权限，可以执行所有操作。以下是相关命令：

`su`：切换用户。例如 `su user2` 切换到 `user2` 用户，只运行 `su` 命令则切换到 `root` 用户。

`sudo`：以超级用户权限执行命令。注意，使用 `sudo` 命令需要输入当前用户的密码，而不是 `root` 用户的密码。

用户和用户组的管理可以通过命令行工具来完成，常用的命令包括：

`useradd`：添加新用户；

`userdel`：删除用户；`userdel -r` 表示删除用户及其主目录；

`groupadd`：添加新用户组；

`groupdel`：删除用户组；

`id`：查看当前用户的 UID 和 GID；

`groups`：查看当前用户所属的用户组。

`passwd`：修改用户密码。例如 `passwd user2` 修改 `user2` 用户的密码。

`gpasswd`：修改用户组密码。例如 `gpasswd group1` 修改 `group1` 用户组的密码。`-a` 表示把用户加入该组；`-d` 表示把用户从该组删除

这些命令通常都需要管理员权限才能执行，但我们一般不直接使用 `root` 用户，而是使用普通用户并通过 `sudo` 命令来执行需要管理员权限的操作。直接在 `root` 用户下操作不当可能会给系统带来致命错误。

注意 1.2 sudo 命令相关

在安装并初始化系统时，你会根据指引创建一个用户，这个用户会被赋予 `sudo` 权限。`sudo` 命令允许普通用户以超级用户权限执行命令。

而用 `useradd` 命令添加用户时，默认情况下新用户不会被赋予 `sudo` 权限。

要给新用户赋予 `sudo` 权限，可以将该用户添加到 `sudo` 用户组中。在有 `sudo` 权限的用户下执行以下命令：

代码 1.6.1 Bash

```
1 user1@host:~$ sudo usermod -aG sudo user2
```

上述命令会将 `user2` 用户添加到 `sudo` 用户组中，从而赋予其 `sudo` 权限。

`sudo` 命令例如：

代码 1.6.2 Bash

```
1 user1@host:~$ sudo useradd user2
2 user1@host:~$ sudo passwd user2
```

输入命令后的操作很简单，根据提示操作即可。

1.6.2 文件权限管理

Linux 系统中的文件权限管理是通过用户、用户组和其他用户的权限来控制对文件和目录的访问。

在 `ls -l` 命令的输出中，第一列显示了文件或目录的权限信息，这里详细介绍一下。

每个文件和目录都有三种权限：读（`r`）、写（`w`）和执行（`x`），分别对应数字权限表示法的 `4`、`2`、`1`。权限的组合可以用数字相加表示，如 `7`（`4+2+1`）表示读、写、执行（`rwX`），`6`（`4+2`）表示读、写（`rw-`），`5`（`4+1`）表示读、执行（`r-X`），`4` 表示读（`r--`），`3` 表示写、执行（`-wX`），`2` 表示写（`-w-`），`1` 表示执行（`--X`），`0` 表示无权限（`---`）。

对于目录而言，执行权限表示可以进入该目录，读权限表示可以列出该目录下的文件和子目录，写权限表示可以在该目录下创建、删除或重命名文件和子目录。

`chown` 命令用于修改文件或目录的所有者，`-R` 表示递归修改子目录和文件的所有者。例如：

代码 1.6.3 Bash

```
1 user1@host:~$ sudo chown user2 test/numbers.txt
2 user1@host:~$ sudo chown -R user2 test/
```

上述命令会将 `test/numbers.txt` 文件的所有者修改为 `user2` 用户，第二条命令会将 `test` 目录及其子目录和文件的所有者修改为 `user2` 用户。

`chgrp` 命令用于修改文件或目录的所属组，`-R` 表示递归修改子目录和文件的所属组。例如：

代码 1.6.4 Bash

```
1 user1@host:~$ sudo chgrp group1 test/numbers.txt
2 user1@host:~$ sudo chgrp -R group1 test/
```

上述命令会将 `test/numbers.txt` 文件的所属组修改为 `group1`，第二条命令会将 `test` 目录及其子目录和文件的所属组修改为 `group1`。

`chmod` 命令用于修改文件或目录的权限，可以使用数字表示法或符号表示法来设置权限，`-R` 表示递归修改子目录和文件的权限。数字表示法是将权限用数字表示，如 `chmod 755 file.txt` 表示设置文件 `file.txt` 的权限为 `rwxr-xr-x`（所有者可读、写、执行，用户组和其他用户可读、执行）。符号表示法是使用字母表示权限，如 `chmod u=rwx file.txt` 表示设置文件 `file.txt` 的所有者权限为 `rwx`（可读、写、执行），用户组和其他用户的权限不变。例如以下两条命令等价：

代码 1.6.5 Bash

```
1 user1@host:~$ sudo chmod 755 test/numbers.txt
2 user1@host:~$ sudo chmod u=rwx,g=rw,o=rw test/numbers.txt
```

此外，还能使用 `+` 和 `-` 符号来添加或删除权限，如 `chmod u+x file.txt` 表示给文件 `file.txt` 的所有者添加执行权限，`chmod g-w file.txt` 表示给文件 `file.txt` 的用户组删除写权限。例如：

代码 1.6.6 Bash

```
1 user1@host:~$ sudo chmod u+x test/numbers.txt
2 user1@host:~$ sudo chmod g-w test/numbers.txt
```

总结 1.3 用户与文件权限管理

用户和用户组：每个用户有唯一的用户名和 UID，用户组有唯一的组名和 GID。root 用户拥有最高权限。

`su` 命令用于切换用户；`sudo` 命令以超级用户权限执行命令（需要输入当前用户密码）。

用户管理命令：`useradd` 添加用户；`userdel` 删除用户（`-r` 同时删除主目录）；`passwd` 修改用户密码。

用户组管理命令：`groupadd` 添加用户组；`groupdel` 删除用户组；`gpasswd` 修改用户组密码（`-a` 添加用户到组，`-d` 从组中删除用户）。

`id` 查看当前用户的 UID 和 GID；`groups` 查看当前用户所属组。

给用户添加 `sudo` 权限：使用 `sudo usermod -aG sudo 用户名`。

文件权限管理：文件权限分为读（`r`）、写（`w`）、执行（`x`），对应数字 4、2、1，分为所有者、用户组、其他用户三组。

目录权限：执行权限（进入目录）、读权限（列出内容）、写权限（创建/删除/重命名）。

`chown` 命令修改文件所有者（`-R` 递归修改）；`chgrp` 命令修改文件所属组（`-R` 递归修改）。

`chmod` 命令修改文件权限（`-R` 递归修改），支持数字表示法（如 `chmod 755 file`）和符号表示法（如 `chmod u=rwx,g=rx,o=rx file`），以及使用 `+`、`-` 添加或删除权限（如 `chmod u+x file`）。

1.7 为系统安装软件

1.7.1 软件包管理器与软件源

在 Linux 系统中，软件包管理器是用于安装、升级和删除软件包的工具。常见的软件包管理器有：

- APT（Advanced Package Tool）：用于 Debian 及其衍生版（如 Ubuntu）的软件包管理。
- YUM（Yellowdog Updater Modified）：用于 Red Hat 及其衍生版（如 CentOS）的软件包管理。
- DNF（Dandified YUM）：YUM 的下一代版本，提供更好的性能和依赖关系管理。
- Zypper：用于 openSUSE 的命令行软件包管理器。

- **Pacman**: 用于 Arch Linux 的包管理器。

APT 是 Debian 及其衍生版（如 Ubuntu）中最常用的软件包管理器，它使用 **.deb** 格式的软件包。APT 提供了简单易用的命令行工具，如 **apt-get** 和 **apt** 命令，用于安装、升级和删除软件包。YUM 是 Red Hat 及其衍生版（如 CentOS）中最常用的软件包管理器，它使用 **.rpm** 格式的软件包。

我们在使用的 Ubuntu 系统中，APT 是默认的软件包管理器。其软件仓库在国外，速度较慢，但可以通过修改软件源来加快下载速度，新手不建议修改软件源，若版本有误，会带来致命错误。有需要修改软件源的可以参考附录A。

1.7.2 安装软件

首先运行以下命令更新软件源：

代码 1.7.1 Bash

```
1 user1@host:~$ sudo apt-get update
```

我们可以尝试安装一个小工具 **fortune**，它会随机显示一段有趣的文字。安装命令如下：

代码 1.7.2 Bash

```
1 user1@host:~$ sudo apt-get install fortune
```

按照提示输入当前用户密码，输入 **y** 确认后，安装完成，可以运行 **fortune** 命令来查看随机的有趣文字：

代码 1.7.3 Bash

```
1 user1@host:~$ fortune
```

若正常安装，则会显示一段随机的有趣文字。

你还可以安装 **sl**（Steam Locomotive），它会在终端中显示一辆火车，运行命令如下：

代码 1.7.4 Bash

```
1 user1@host:~$ sudo apt-get install sl
```

安装完成后，可以运行 **sl** 命令：

代码 1.7.5 Bash

```
1 user1@host:~$ sl
```

若终端上跑过一辆火车，则表示安装成功。

1.7.3 卸载软件

如果要卸载软件，可以使用 `apt-get remove` 命令，例如：

代码 1.7.6 Bash

```
1 user1@host:~$ sudo apt-get remove fortune
```

1.7.4 课程所需软件

本课程中，我们会用到 Vim 编辑器、Python 编程语言和 Fortran 编程语言。Vim 和 Python3 通常在 Ubuntu 系统中预装，但 Fortran 编译器需要手动安装。

安装 Fortran 编译器可以使用以下命令：

代码 1.7.7 Bash

```
1 user1@host:~$ sudo apt-get install gfortran
```

安装完成后，可以使用 `gfortran --version` 命令查看 Fortran 编译器的版本信息：

代码 1.7.8 Bash

```
1 user1@host:~$ gfortran --version
2 GNU Fortran (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
3 Copyright (C) 2023 Free Software Foundation, Inc.
4 This is free software; see the source for copying conditions.
   ↪ There is NO
5 warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
   ↪ PURPOSE.
```

出现以上输出则表示 Fortran 编译器安装成功。

1.8 文本处理工具

这里介绍三个命令：`grep`、`awk` 和 `sed`，它们是 Linux 中文本处理工具。此部分要求不高，看见命令能明白意思、会写简单的 `grep` 命令即可。

1.8.1 `grep` 命令

`grep` 命令用于在文件中查找匹配指定模式的行，并输出这些行。它的基本语法为：

代码 1.8.1 Bash

```
1 grep [选项] PATTERN [FILE...]
```

其中，`PATTERN` 是要匹配的模式，可以是字符串或正则表达式，`FILE` 是要搜索的文件名。常用选项包括：

- `-i`：忽略大小写
- `-v`：反向匹配，输出不匹配的行
- `-w`：匹配整个单词
- `-n`：显示匹配行的行号
- `-An`：显示匹配行及其后 `n` 行
- `-Bn`：显示匹配行及其前 `n` 行
- `-Cn`：显示匹配行及其前后各 `n` 行
- `-G`：使用基本正则表达式
- `-E`：使用扩展正则表达式

例如，要在 `test/numbers.txt` 文件中查找包含数字 `5` 的行，并显示行号，可以使用以下命令：

代码 1.8.2 Bash

```
1 user1@host:~$ grep -n 5 test/numbers.txt
```

上述命令会输出所有包含数字 `5` 的行及其行号。如果只想找到数字 `55`，可以使用以下命令：

代码 1.8.3 Bash

```
1 user1@host:~$ grep -nw 15 test/numbers.txt
```

上述命令会输出所有包含数字 **15** 的行及其行号。不包括包含 **15** 的其他数字，如 **155**、**515** 等。

这里通配符的使用与之前匹配文件的通配符类似：

- **^**：匹配行的开头
- **\$**：匹配行的结尾
- **.**：匹配任意单个字符
- *****：匹配零个或多个任意字符
- **[]**：匹配括号内的任意单个字符
- **[!]**：匹配不在括号内的任意单个字符
- **{n,m}**：匹配前一个字符或表达式至少 **n** 次，至多 **m** 次
- ****：转义字符，用于匹配特殊字符
- **|**：逻辑或，用于匹配多个模式

组合使用如：

- **^\$**：匹配空行
- **.***：匹配非空行
- **^.***：匹配以任意字符开头的行
- **.*\$**：匹配以任意字符结尾的行
- **[0-9]**：匹配数字
- **[A-Za-z]**：匹配字母

例如我想找到所有三位数的数字，可以使用以下命令：

代码 1.8.4 Bash

```
1 user1@host:~$ grep -G '[0-9]\{3\}$' test/numbers.txt
```

1.8.2 awk 命令

我们先准备一个文本文件 `test/grades.txt` 以供操作使用，按如下步骤生成：

代码 1.8.5 Bash

```
1 user1@host:~$ sudo apt-get install rs
2 user1@host:~$ seq 1 100 > test/numbers-2.txt
3 user1@host:~$ shuf -r -n 300 -i 60-100 >> test/numbers-2.txt
4 user1@host:~$ rs 4 100 < test/numbers-2.txt | rs -T >
   test/grades.txt
```

我们安装了 `rs` 命令（`reformat`），它可以格式化文本文件。上述命令会生成一个名为 `test/grades.txt` 的文件，共 400 个数字，每行 4 个数字，共 100 行，第一列为序号（`seq` 生成的顺序序列），第二至第四列为随机生成的 60 到 100 之间的数字（`shuf` 生成的随机数），可认为是 100 个人三次考试的成绩。

我们现在用 `awk` 命令来处理这个文件。

例如我们可以计算每个人的平均成绩，并输出到终端：

代码 1.8.6 Bash

```
1 user1@host:~$ awk '{print $1, ($2 + $3 + $4) / 3}' test/grades.txt
```

或可将结果保存到文件中：

代码 1.8.7 Bash

```
1 user1@host:~$ awk '{print $1, $2, $3, $4, ($2 + $3 + $4) / 3}'
   test/grades.txt > test/grades-averages.txt
```

上述命令会将每个人的序号、三次成绩和平均成绩输出到 `test/grades-averages.txt` 文件中。其中 `$n` 表示第 `n` 列的内容。另外，`$NF` 表示最后一列的内容，`$0` 表示整行内容。

同时，还有一些可选参数，如 `-F`：指定字段分隔符，默认为空格或制表符（`tab`），可以使用 `-F,` 指定逗号为分隔符。

1.8.3 sed 命令

`sed` 命令用于对文本进行流编辑，可以对输入的文本进行替换、删除、插入等操作。它的

基本语法为：

代码 1.8.8 Bash

```
1 sed [选项] '命令' [文件...]
```

常用选项包括：

- **-n**：只输出被“命令”处理过的行
- **-i**：直接修改文件

常用命令有：

- **s/匹配的文本/替换文本/g**：替换匹配模式的文本为替换文本，**g** 表示全局替换
- **d/匹配的文本/**：删除匹配模式的行
- **a/匹配的文本/**：在匹配模式的行后添加文本
- **i/匹配的文本/**：在匹配模式的行前添加文本

例如，要将 `test/grades.txt` 文件中的所有 `60` 替换为 `70`，可以使用以下命令：

代码 1.8.9 Bash

```
1 user1@host:~$ sed 's/60/70/g' test/grades.txt
```

上述命令会输出替换后的内容，但不会修改原文件。如果要直接修改原文件，可以使用 **-i** 选项：

代码 1.8.10 Bash

```
1 user1@host:~$ sed -i 's/60/70/g' test/grades.txt
```

1.9 Vim 文本编辑器

1.9.1 Vim 简介与三个模式

在 Windows 中，我们用 Word 或记事本编辑文本文件，而在 Linux 中，我们通常使用 Vim 编辑器，相比于 Word 或记事本，支持多种编程语言的语法高亮、自动缩进等功能。

使用命令 `vim` 打开文件：

代码 1.9.1 Bash

```
1 user1@host:~$ vim test/grades.txt
```

Vim 编辑器有三种模式：一般模式、编辑模式和命令、查找模式。进入 Vim 编辑器后，默认处于一般模式。

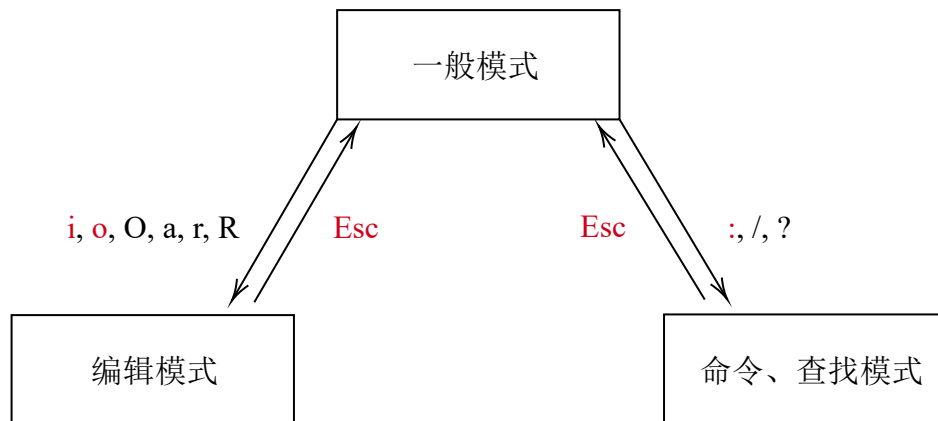


图 1: Vim 文本编辑器的三种模式

在一般模式下，有以下常用操作：

- **h**：光标左移
- **j**：光标下移
- **k**：光标上移
- **l**：光标右移
- **0**：光标移动到行首
- **gg**：光标移动到文件开头
- **G**：光标移动到文件结尾
- **x**：删除光标所在字符
- **dd**：剪切光标所在行
- **yy**：复制光标所在行
- **p**：粘贴复制的行到光标后面
- **u**：撤销上一步操作
- **Ctrl+r**：重做上一步操作

用 **i**、**o**、**O**、**a**、**r**、**R** 进入编辑模式；

用 **:**、**/**、**?** 进入底线命令模式。在非一般模式下，按 **Esc** 键可以返回一般模式。

如按 **:** 进入命令模式，输入 **wq** 并回车即可保存并退出 Vim 编辑器。

下面详细介绍编辑模式和底线命令（即命令或查找，命令输入在屏幕最下方一行输入，故称底线命令）模式。

1.9.2 编辑模式

在编辑模式下，我们可以直接编辑文本，输入文本、删除文本、复制粘贴文本等。常用的进入编辑模式命令有：

- **i**：在光标前插入文本
- **I**：在行首插入文本
- **a**：在光标后插入文本
- **A**：在行尾插入文本
- **o**：在光标下方新建一行并插入文本
- **O**：在光标上方新建一行并插入文本
- **r**：替换光标所在字符
- **R**：进入替换模式，替换光标所在位置及后续字符

其中主要掌握 **i** 的用法即可

注意 1.3 提示

在 Ubuntu 终端中，包括终端里进入的 Vim 编辑器，按 **Ctrl+Shift+V** 可以粘贴文本。

1.9.3 命令、查找模式

在命令、查找模式下，我们可以执行各种命令和查找操作。常用的命令有：

- **:w**：保存文件
- **:q**：退出 Vim 编辑器
- **:wq**：保存并退出 Vim 编辑器

- `:q!`：强制退出 Vim 编辑器，不保存修改

比如在一般模式下按 `:` 进入命令模式，输入 `wq` 并回车即可保存并退出 Vim 编辑器。

查找操作有：

- `/string`：向下查找字符串 “string”
- `?string`：向上查找字符串 “string”
- `n`：重复前一个查找

这里不再举例，自行体会即可。

1.10 与 window 宿主机共享文件

这在后面的课程中会用到，但不是考察内容，本笔记正文中也不多介绍。

在 Linux 系统中，我们可以通过 Samba 服务或 FTP 与 Windows 宿主机共享文件。主要原理是通过网络协议实现文件的共享和传输。有需要可自行搜索教程。

这里介绍另一种方法，设置完成后比 Samba 服务或 FTP 方便，具体方法参见附录B。

2 Fortran 语言简介

2.1 编程语言与 Fortran 的发展

计算机只认识机器语言，机器语言是由二进制数（0 和 1）组成的指令序列。

机器语言难以理解和编写，因此需要更高级的编程语言来简化编程过程。其它语言需要经过编译器翻译成机器语言才能被计算机执行。

为了方便编写程序，发展出了汇编语言，汇编语言只是特定功能的二进制数的助记符，而汇编语言的编译器实际上更像“文本替换器”，只是把特定字母和符号替换成机器语言指令。

随后便发展出了更高级的编程语言，如 Fortran、C、C++、Java 等，这些语言更接近人类的自然语言，使得编程更加直观和易于理解。

Fortran 是最古老的高级编程语言之一，最初设计用于科学计算和数值分析。目前看来有些过时，但出于其对硬件的高效利用和对数值计算的优化，以及多年积累的丰富代码库，Fortran 仍然在科学计算领域广泛使用。

早期的 Fortran 版本主要有 Fortran 77、Fortran 90。Fortran 也在不断发展，新版本的 Fortran 语言增加很多功能，如动态数组等，Fortran 2003、Fortran 2008 等版本引入了面向对象编程和并行计算等新特性，Fortran modern 是对 Fortran 90 及其后续版本的统称。

出于稳定性以及兼容性考虑，我们课程主要使用 Fortran 77，会用到一些 Fortran 90 的特性。文件扩展名统一用 `.f`（即文件名为 `*.f`），避免出现语言版本与文件扩展名不一致的情况。

2.2 Fortran 与打孔卡片

Fortran 语言创始的时候，计算机还没有现代计算机键盘、屏幕等输入输出设备，主要通过打孔卡片输入程序以及输出结果。

打孔卡片格式统一，共 12 行（0-9 共 10 行，上面空白处 2 行），80 列，每一列表示一个字符，每一张打孔卡片可以表示一行语句。

如

代码 2.2.1 Text

```
1 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ,.;'"+=!@#$%^&*()_</>
```

在打孔卡片上就如图2显示，注意，打孔卡片不分大小写。



图 2: 打孔卡片

对于 Fortran 语言，打孔卡片的格式有一些特殊要求，称为 Fortran 的固定格式，每一行的要求如下：

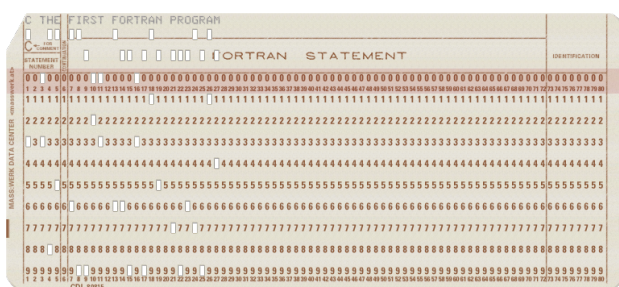
第 1 个字符	如果是字母 C 或 *，这一行就是注释，不会被编译
第 1~5 个字符	如果是数字，就是给这一行代码取一个代号，不然只能是空格
第 6 个字符	如果是“0”以外的任何字符，表示这一行程序接上一行
第 7~72 个字符	Fortran 程序代码的编写区域
第 73 个字符及之后	保留区域，留空

例如如下代码片段：

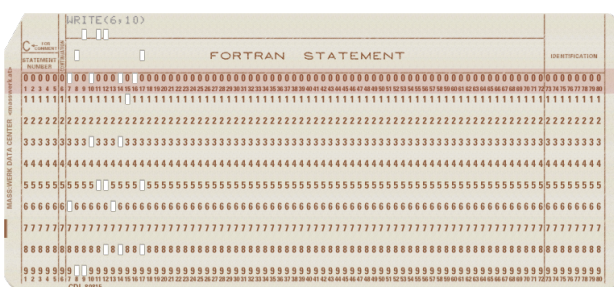
```
代码 2.2.2 Fortran
1 C THE FIRST FORTRAN PROGRAM
2     WRITE(6,10)
3     10 FORMAT(1X, 'HELLO, WORLD.')
4     END
```

每一行在打孔卡片上就如图3所示，每一行占一张卡片，共 4 张卡片。（现在还不需要知道代码各个部分是什么意思，这里只是为了说明 Fortran 的固定格式和打孔卡片的概念。）

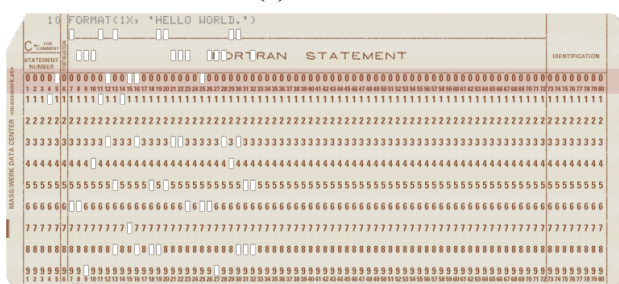
在向机器输入程序时，依次读取四张卡片即可。



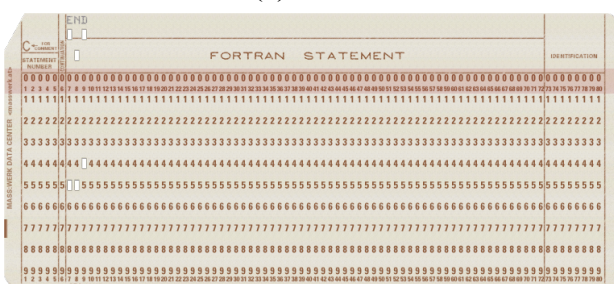
(a) 第 1 行



(b) 第 2 行



(c) 第 3 行



(d) 第 4 行

图 3: The first Fortran program

整个程序编译后运行的输出结果为:

代码 2.2.3 Text

```
1 HELLO, WORLD.
```

在打孔卡片上如图所示:



图 4: 程序输出结果

当然, 现在已经没有人使用打孔卡片了, Fortran 程序的编写和运行都可以在现代计算机

上完成。但是，Fortran 的固定格式仍然保留了下来，Fortran 程序的编写仍然需要遵循固定格式。

同时卡片的概念在 Fortran 中也保留了下来，Fortran 程序的每一行代码都可以看作是一张卡片，读取的每一行数据也可以看作是一张卡片。

另外，打孔卡片是不区分大小写的，Fortran 程序的代码也不区分大小写。

2.3 Fortran 语言的编译

Fortran 程序需要经过编译器编译成机器语言才能被计算机执行。

在本笔记 Linux 部分已经提到了“gfortran”编译器的安装，如果还没有安装，可以使用以下命令安装：

代码 2.3.1 Bash

```
1 user1@host:~$ sudo apt-get install gfortran
```

以上面的例子为例，假设我们将代码保存在 `learn` 文件夹的 `hello.f`：

代码 2.3.2 Bash

```
1 user1@host:~$ mkdir learn
2 user1@host:~$ cd learn
3 user1@host:~/learn$ vim hello.f
```

Vim 编辑器用法不再重复，在文件中写入上面例子的代码，注意按照固定格式确认好空格数：

代码 2.3.3 Fortran

```
1 C THE FIRST FORTRAN PROGRAM
2     WRITE(6,10)
3     10 FORMAT(1X, 'HELLO, WORLD.')
4     END
```

保存并退出编辑器后，使用以下命令编译程序：

代码 2.3.4 Bash

```
1 user1@host:~/learn$ gfortran -c hello.f
2 user1@host:~/learn$ gfortran -o hello hello.o
3 user1@host:~/learn$ ls
4 hello hello.f hello.o
```

第一条命令会生成一个目标文件 `hello.o`，第二条命令会将目标文件链接成可执行文件 `hello`。运行程序使用以下命令：

代码 2.3.5 Bash

```
1 user1@host:~/learn$ ./hello
2 HELLO, WORLD.
```

如果是像这样只有一个代码文件的简单程序，或者不想生成目标文件，可以直接使用以下命令编译并运行程序：

代码 2.3.6 Bash

```
1 user1@host:~/learn$ gfortran -o hello hello.f
2 user1@host:~/learn$ ./hello
3 HELLO, WORLD.
```

这样不会生成目标文件 `hello.o`，直接生成可执行文件 `hello`。

3 Fortran 基础

3.1 Fortran 变量类型声明与转化

3.1.1 Fortran 变量类型声明

Fortran 语言中，变量的类型和声明是非常重要的概念。Fortran 支持多种数据类型，包括整数、实数（与数学上的实数定义不同）、字符等。

变量名可以包含字母、数字和下划线，但要以字母开头，同时不超过 6 个字符。

Fortran 有隐式声明和显式声明两种方式。

隐式声明是指在使用变量之前不需要显式地声明变量的类型，Fortran 会根据变量名的首字母自动推断类型，例如以 **I、J、K、L、M、N** 开头的变量默认为整数类型，其它变量默认为实数类型。

显式声明是指在使用变量之前需要显式地声明变量的类型，使用类型声明语句，例如：

代码 3.1.1 Fortran

```
1      INTEGER :: I
2      REAL :: X
3      DOUBLE PRECISION :: Y
```

上面第一行声明了一个整数类型的变量 **I**，第二行声明了一个单精度实数类型的变量 **X**，第三行声明了一个双精度实数类型的变量 **Y**。

一般我们遵循隐式声明的规则，除非有特殊需要，否则不使用显式声明。

3.1.2 Fortran 变量类型转化

Fortran 语言中，变量类型的转化是一个重要的概念。Fortran 提供了多种方式进行变量类型的转化。

使用类型转换函数，例如：

代码 3.1.2 Fortran

```
1      M = INT(A)
2      N = IFIX(B)
```

```
3      I = IDINT(C)
```

前两个语句将实数 **A** 和 **B** 转换为整数类型，第三个语句将双精度实数 **C** 转换为整数类型。可统一使用 **INT** 函数进行转换。

代码 3.1.3 Fortran

```
1      X = REAL(I)
2      Y = FLOAT(J)
3      Z = DBLE(K)
```

前两个语句作用相同，将整数 **I** 和 **J** 转换为单精度实数类型，第三个语句将整数 **K** 转换为双精度实数类型。

以上可统一记为 **INT** 转化为整数，**REAL** 转化为单精度实数，**DBLE** 转化为双精度实数，其它的都能用这三个代替。

3.2 Fortran 程序的输入与输出

Fortran 程序的输入与输出主要通过以下几个语句实现：

- **READ**：用于从标准输入设备（通常是键盘）读取数据。
- **WRITE**：用于向标准输出设备（通常是屏幕）输出数据。
- **PRINT**：用于向标准输出设备写入数据，语法更简单。

我们主要使用 **READ** 和 **WRITE** 语句来进行输入和输出。

3.2.1 输入语句

Fortran 中的输入语句主要有两种形式：**READ** 和 **READ(*,*)**。

READ 语句的基本形式为：

代码 3.2.1 Fortran

```
1      READ, 变量
2      READ(单位, 格式) 变量1 变量2 ...
```

其中，单位表示输入设备，其中 **5** 表示读卡器（在现代的电脑上就是用键盘输入），格式表示数据的格式，变量列表表示要读取的变量。

READ(5,*) 表示从标准输入设备读取数据，格式由编译器自动推断，适用于简单的输入场景。

3.2.2 输出语句

Fortran 中的输出语句主要有两种形式：**WRITE** 和 **PRINT**。

WRITE 语句的基本形式为：

代码 3.2.2 Fortran

```
1      WRITE(单位, 格式) 变量1 变量2 ...
```

其中，单位表示输出设备，其中 **6** 表示行式打印机（在现代的电脑上就是用屏幕输出），格式表示数据的格式，变量列表表示要输出的变量。

PRINT 语句的基本形式为：

代码 3.2.3 Fortran

```
1      PRINT(格式) 变量1 变量2 ...
```

其中，格式表示数据的格式，变量列表表示要输出的变量。

3.3 格式化输出

Fortran 支持格式化输入输出，可以通过格式字符串来控制输出的格式。一般情况下，输入数据来源复杂，格式多样，我们会使用 ***** 表示自动推断格式；输出时为了获取整齐美观的结果，可以指定格式。下面只介绍输出的格式化，输入格式化类似，不再赘述。

3.3.1 格式化输出语句

上节命令如 **READ** 和 **WRITE** 中都有“格式”参数，一般读取时我们用 ***** 表示自动推断格式，输出时我们可以指定格式。

指定格式有两种方式：

第一种是使用格式字符串，例如：

代码 3.3.1 Fortran

```
1      WRITE(6, '(A, I5, F10.2, E10.2)') '结果是: ', I, X
```

其中, `'(A, I5, F10.2, E10.2)'` 表示输出一个字符串 (A), 一个整数 (I5, 宽度为 5), 一个单精度实数 (F10.2, 宽度为 10, 小数点后保留 2 位), 一个双精度实数 (E10.2, 宽度为 10, 小数点后保留 2 位)。

第二种是使用格式标签, 例如:

代码 3.3.2 Fortran

```
1      WRITE(6, 100) I, X
2      100 FORMAT('结果是: ', I5, F10.2, E10.2)
```

其中, `100` 是标签, 其指定的行中的 `FORMAT` 语句定义了格式。如果格式字符串较长, 可以使用标签来分割, 避免超出固定格式行长度限制。

3.3.2 I 型

I 型格式用于输出整数, 格式为 `Iw`, 其中 `w` 表示输出的宽度, 可以理解为给这个数据 5 个字符的空间, 例如:

代码 3.3.3 Fortran

```
1      M=12345
2      N=123
3      K=123456
4      WRITE(6, '(I5)') M
5      WRITE(6, '(I5)') N
6      WRITE(6, '(I5)') K
7      END
```

编译后运行输出结果为:

代码 3.3.4 Text

```
1 12345
2 123
```

```
3 *****
```

其中，`*****` 表示输出的整数超过了指定的宽度，Fortran 会用星号填充。

3.3.3 F 型

F 型格式用于输出单精度实数，格式为 `Fw.d`，其中 `w` 表示输出的宽度，`d` 表示小数点后保留的位数，例如：

代码 3.3.5 Fortran

```
1      K=1234567890
2      X=123.456
3      WRITE(6, '(I10)') K
4      WRITE(6, '(F10.2)') X
5      END
```

编译后运行输出结果为：

代码 3.3.6 Text

```
1 1234567890
2    123.46
```

（这里第一行的整数用来标记位置，便于直观观察 F 型所占空间）其中，`123.46` 表示输出的单精度实数，Fortran 会根据指定的格式进行填充和舍入。

3.3.4 E 型

E 型格式用于输出双精度实数，格式为 `Ew.d`，其中 `w` 表示输出的宽度，`d` 表示小数点后保留的位数，例如：

代码 3.3.7 Fortran

```
1      K=1234567890
2      Y=-123.456
3      Z=123.456
4      WRITE(6, '(I10)') K
```

```
5      WRITE(6, '(E10.3)') Y
6      WRITE(6, '(E10.3)') Z
7      END
```

编译运行后输出结果为：

代码 3.3.8 Text

```
1  1234567890
2  -0.123E+03
3   0.123E+03
```

（这里第一行的整数用来标记位置，便于直观观察 E 型所占空间）其中，`0.123E+03` 表示输出的双精度实数。

Fortran 中标准的 E 型格式需要注意，一个数用科学计数法表示为 $A \times 10^N$ ，表示为 E 型为 `AEN`，标准化 E 型中，A 的整数部分为 0。如 `123.456`，则表示为 `0.123456E+03` 而不是 `1.23456E+02`，再根据格式进行填充和舍入。

以上内容可见，E 型占用的空间较大，使用时注意前后留足空间，如使用 `E20.5`。

3.4 数学语句

3.4.1 数字与运算符

在编写 Fortran 程序时，我们需要使用数字和运算符来进行计算。整数如 `123`、`-456` 等，直接表示即可。

实数如 `3.14`、`-2.718` 等，直接表示即可；`0.123`、`456.0`，可以省略 0，直接写为 `.123`、`456.`。

Fortran 常用的运算符有加（`+`）、减（`-`）、乘（`*`）、除（`/`）和幂（`**`）等。可直接对数字或变量进行运算。运算优先级从高到低依次为：幂、乘除、加减。如果有括号，则括号内的运算优先级最高。

3.4.2 数学函数

Fortran 提供了多种数学函数，用于对数字进行处理和计算。例如：

代码 3.4.1 Fortran

```
1      A = ABS(X)      ! 绝对值
2      B = SQRT(Y)     ! 平方根
3      C = EXP(Z)      ! 指数函数
4      D = LOG(W)      ! 自然对数
5      E = LOG10(V)    ! 常用对数
6      F = SIN(T)      ! 正弦函数
7      G = COS(U)      ! 余弦函数
8      M = MAX(A, B)   ! 最大值
9      N = MIN(C, D)   ! 最小值
10     O = MOD(P, Q)   ! 取模
```

其中自然对数函数在课堂上介绍的是 `ALOG`，这是 Fortran 77 的函数，用于处理单精度实数；另外还有 `DLOG` 用于处理双精度实数、`CLOG` 用于处理复数，比较复杂。

在 Fortran 90 中，增加了 `LOG` 函数，包括了上面三种用法，兼容性强，推荐使用。

3.4.3 赋值语句

赋值语句用于将一个值赋给一个变量，例如：

代码 3.4.2 Fortran

```
1      I = 5
2      X = 3.14
3      Y = 2.718281828459045
```

3.4.4 算术运算

Fortran 支持基本的算术运算，包括加法、减法、乘法、除法等，例如：

代码 3.4.3 Fortran

```
1      Z = X + Y
2      A = B - C
3      D = E * F
4      G = H / I
5      J = K ** L
```


3.4.5 大型算术表达式

Fortran 支持大型算术表达式，可通过嵌套多层括号、合理利用运算符的优先级来实现复杂的计算。例如：

$$z = \frac{e^{x+y} + \sin x + s^t}{|x| + \sqrt{y} - \lg(w)}$$

用 Fortran 的数学语句表示为：

代码 3.4.4 Fortran

```
1      Z = (EXP(X+Y) + SIN(X) + S**T) / (ABS(X) + SQRT(Y) -  
      ↪ LOG10(W))
```

为了防止语句过长违反固定格式，可拆开表示：

代码 3.4.5 Fortran

```
1      P = EXP(X+Y) + SIN(X) + S**T  
2      Q = ABS(X) + SQRT(Y) - LOG10(W)  
3      Z = P / Q
```

3.5 Fortran 程序构成

Fortran 程序一般由以下几个部分构成：

- 程序头：包括程序名、变量声明等。
- 主程序：包含程序的主要逻辑和计算部分。
- 子程序：可选部分，用于实现特定功能的代码块。
- 结束语句：标志程序的结束。

例如：

代码 3.5.1 Fortran

```
1      PROGRAM HelloWorld  
2      INTEGER :: I  
3      REAL :: X, Y  
4  
5      WRITE(6, '(A)') 'Hello, World!'
```

```
6
7      I = 5
8      X = 3.14
9      Y = 2.718281828459045
10
11     WRITE(6, '(I5, F10.2, E10.3)') I, X, Y
12
13     END PROGRAM HelloWorld
```

其中第一行 `PROGRAM HelloWorld` 表示程序名为 `HelloWorld`，非必须，可以不写。

接下来是变量声明部分，使用 `INTEGER` 和 `REAL` 声明了整数和实数类型的变量，这里的内容实际上包含在隐式声明内了，可以不再显式声明。

`END` 语句标志程序的结束，后面的 `PROGRAM HelloWorld` 表示程序名的结束，可以直接写 `END`。

A 更换软件源为清华大学镜像站

A.1 Ubuntu 24.04.2 LTS 软件源配置

如果是按照文章开始的方式安装的 Ubuntu 系统，安装的版本应当是 24.04.2 LTS（长期支持版），请确保版本是 24.04.2。可以通过以下命令查看当前系统版本：

代码 A.1.1 Bash

```
1 user1@host:~$ lsb_release -a
2 No LSB modules are available.
3 Distributor ID: Ubuntu
4 Description:    Ubuntu 24.04.2 LTS
5 Release:        24.04
6 Codename:       noble
```

运行如下命令，打开软件源配置文件：

代码 A.1.2 Bash

```
1 user1@host:~$ sudo vim /etc/apt/sources.list.d/ubuntu.sources
```

不断按 **dd** 删除原有内容，直到清空文件。

然后按 **i** 进入插入模式，粘贴以下内容：

代码 A.1.3 Text

```
1 Types: deb
2 URIs: https://mirrors.tuna.tsinghua.edu.cn/ubuntu
3 Suites: noble noble-updates noble-backports
4 Components: main restricted universe multiverse
5 Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
6
7 # 默认注释了源码镜像以提高 apt update 速度，如有需要可自行取消注释
8 # Types: deb-src
9 # URIs: https://mirrors.tuna.tsinghua.edu.cn/ubuntu
10 # Suites: noble noble-updates noble-backports
```

```
11 # Components: main restricted universe multiverse
12 # Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
13
14 # 以下安全更新软件源包含了官方源与镜像站配置，如有需要可自行修改
    ↪ 注释切换
15 Types: deb
16 URIs: http://security.ubuntu.com/ubuntu/
17 Suites: noble-security
18 Components: main restricted universe multiverse
19 Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
20
21 # Types: deb-src
22 # URIs: http://security.ubuntu.com/ubuntu/
23 # Suites: noble-security
24 # Components: main restricted universe multiverse
25 # Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
26
27 # 预发布软件源，不建议启用
28
29 # Types: deb
30 # URIs: https://mirrors.tuna.tsinghua.edu.cn/ubuntu
31 # Suites: noble-proposed
32 # Components: main restricted universe multiverse
33 # Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
34
35 # # Types: deb-src
36 # # URIs: https://mirrors.tuna.tsinghua.edu.cn/ubuntu
37 # # Suites: noble-proposed
38 # # Components: main restricted universe multiverse
39 # # Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
```

按“Esc”键退出插入模式，然后输入 `:wq` 后回车保存并退出。

接下来，运行以下命令更新软件源列表：

代码 A.1.4 Bash

```
1 user1@host:~$ sudo apt-get update
```

A.2 更早或更新版本软件源配置

清华大学开源软件镜像站一般只有 LTS 版本的 Ubuntu 软件源，首先请确保自己系统为 LTS 版本。

访问[清华大学开源软件镜像站](#)，仔细阅读网页指导内容，仿照上例更改软件源配置。或自行查找相关教程，这里不作介绍。

B 与 Window 宿主机共享文件夹

B.1 WSL 子系统

作为 Windows 下的子系统，WSL（Windows Subsystem for Linux）可以直接访问 Windows 宿主机的文件系统。

在 WSL 中，Windows 的文件系统通常挂载在 `/mnt/c` 目录下。你可以通过以下命令访问 Windows 的 C 盘、D 盘：

代码 B.1.1 Bash

```
1 user1@host:~$ cd /mnt/c
2 user1@host:/mnt/c$ ls
3 Users  Program Files  Program Files (x86)  Windows ...
4 user1@host:~$ cd /mnt/d
```

注意大小写。

复制、移动操作方式与 Linux 系统相同，可以使用 `cp`、`mv` 等命令。

B.2 虚拟机

如果按照文章开始的方法，正确配置虚拟机 VMware，可以通过虚拟机的共享文件夹功能来实现与宿主机的文件共享。

按照下图所示，在虚拟机设置中启用共享文件夹功能，并添加想要共享的文件夹：

在“添加共享文件夹向导”的这一步时，点击“浏览”选择需要共享的文件夹，名称可不填：

以上设置完成后，打开终端，运行以下命令：

代码 B.2.1 Bash

```
1 user1@host:~$ vmware-hgfsclient
```

若列出了所有共享的文件夹名称，则表示共享文件夹设置成功。

继续运行以下命令：

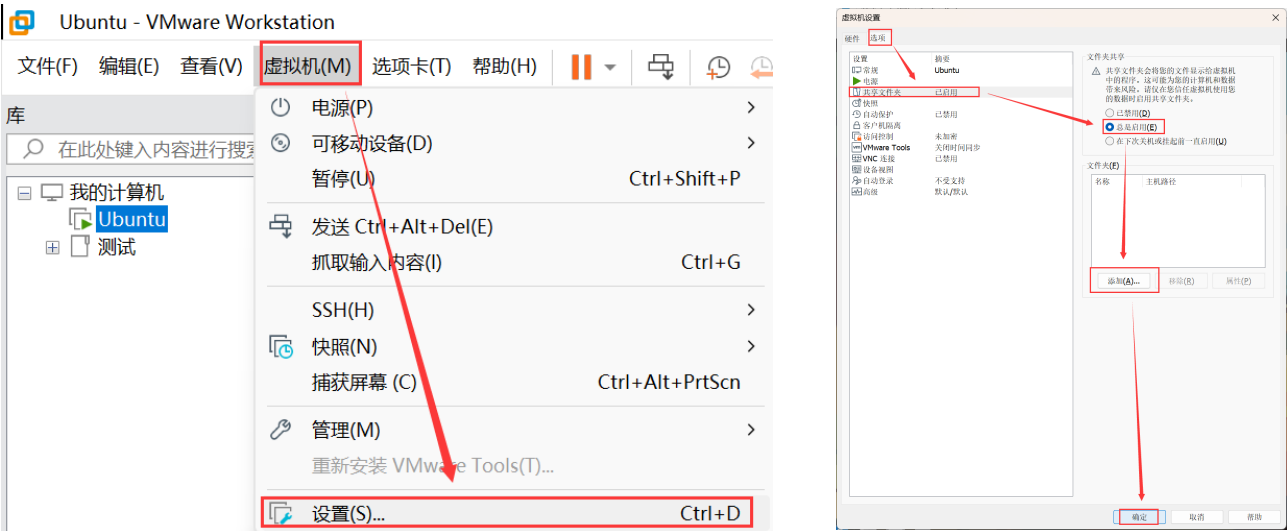


图 5: 设置共享文件夹

代码 B.2.2 Bash

```
1 user1@host:~$ sudo /usr/bin/vmhgfs-fuse .host:/ /mnt/hgfs -o
   allow_other -o uid=1000 -o gid=1000 -o umask=022
```

这将把共享文件夹挂载到 `/mnt/hgfs` 目录下。

现在，你可以通过以下命令访问共享文件夹：

代码 B.2.3 Bash

```
1 user1@host:~$ cd /mnt/hgfs
2 user1@host:/mnt/hgfs$ ls
3 SharedFolder1 SharedFolder2 ...
```

若显示了共享文件夹，则表示共享文件夹设置成功。

接下来设置开机自动挂载，编辑 `/etc/fstab` 文件：

代码 B.2.4 Bash

```
1 user1@host:~$ sudo vim /etc/fstab
```

按 “i” 进入插入模式，移动光标到最后一行，在文件末尾新建一行添加以下内容：



图 6: 共享文件夹向导

代码 B.2.5 Text

```
1 .host:/ /mnt/hgfs fuse.vmhgfs-fuse  
   ↪ allow_other,uid=1000,gid=1000,umask=022 0 0
```

按“Esc”键退出插入模式，然后输入 `:wq` 后回车保存并退出。

重启虚拟机，进入 `/mnt/hgfs` 目录，如果可以看到共享的文件夹，则说明开机自动挂载设置成功。

如果需要访问共享文件夹，只需进入 `/mnt/hgfs` 目录即可。

如果之后需要添加新的共享文件夹，重复第一步，添加新的共享文件夹即可，不必重复设置开机自动挂载。