金融计算机第二周作业

2025 年 3 月 14 日

一、ls 命令，检查计算机文件系统路径

1.目录树（检查桌面、下载、文档路径）

计算机在管理资源的时候使用的是树形结构，树形结构有很多节点，是一种用于表示文件系统中目录结构的树形数据结构

它以根目录为起点，就像树的主干，然后分支形成各个子目录，子目录又可以继续包含更多的子目录和文件，类似于树的枝叶，从而形成一种层次化的结构。

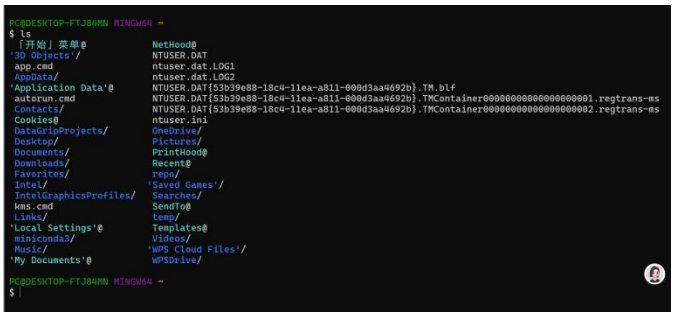

图 1

如图 1，在终端中输入 **“ls”** 命令可以输出一个列表，看到当前工作目录下有什么东西。

图 2

如图 2，输入 **“pwd”** 命令可输出当前的工作目录意思是在 C 盘中的 Users 文件夹下的 PC 文件夹中。

如果看不到列表中的文件夹，说明可能被隐藏了，所以可以通过查看隐藏项目来查看到文件夹。

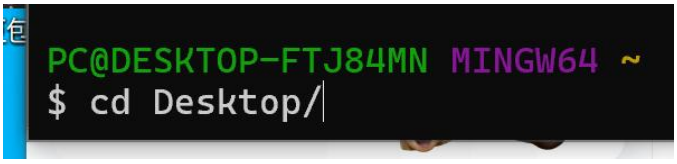

图 3

如图 3，可以在终端输入 cd des 后按 tap 键即可补全所有代码。这个补足之可以用于在当前目录下只有一个 des 开头的文件夹名称，如果有 2 个 des 开头的文件夹名，就无法进行补全，可以连按 2 下 tap 键，或多增加代码中输入的字母。

图 4

如图 4 所示，显示的就是

图 5

如图 5 所示，我们可以看到用 **pwd** 可以确定当前工作目录是在 deskbook，通过使用 **“ls”** 代码即可看到在桌面目录下的文件或文件夹都有什么。再通过“cat 文件名.格式”即可抓取文件或文件夹。

！！！Control＋S：用于保存的

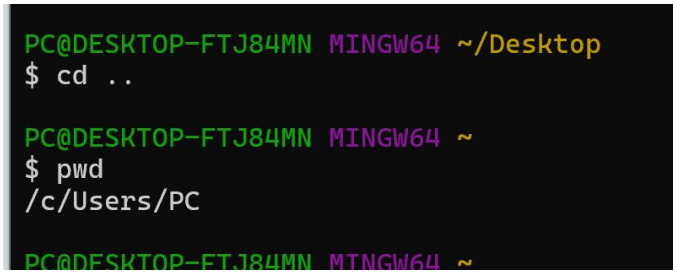

图 6

代码 **“cd ..”** 的含义是返回上一个目录，所以从 PC 下的 Deskbook 目录，回到了 PC 目录

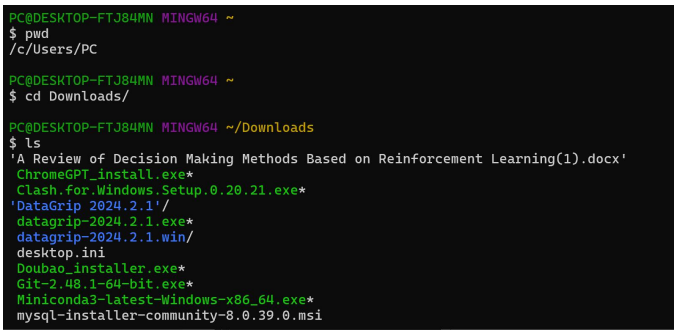

图 7

想要进入到 Download 文件夹，与进入 deskbook 相似，首先确定当前目录位置（pwd），再定位 download 的文件夹位置（cd download），最后通过使用“ls”代码，查看当前目录下的文件夹中的内容，即 download 中的文件内容。

图 8

通过使用代码“cd ../Document/”即可到达上一个目录下的另外一个文件夹中，再通过使用“ls”查看文件夹下的文件

2.根目录

根目录是文件系统中最顶层的目录，是整个文件系统的起点。在不同操作系统中，根目录有不同的表示形式，比如在 Linux 系统中根目录用“/”表示 ，在 Windows 系统中通常以磁盘盘符（如 C:\ 、D:\等）作为根目录的标识。它就像一棵树的根部，其他所有的子目录和文件都从根目录分支展开。

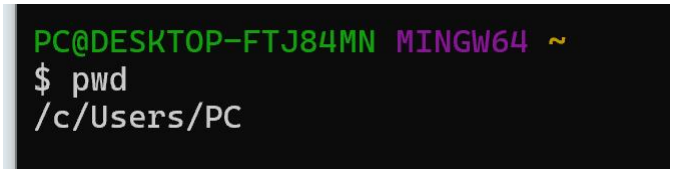

图 9

例如在 git bash 中的跟目录是/，如图 9 所示

图 10

在 windows 的文件夹中显示根目录是盘符

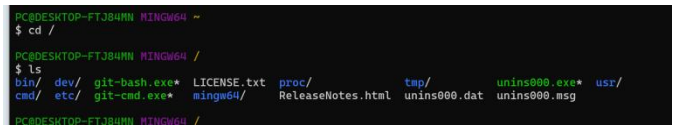

图 11

代码“cd /”可以返回到根目录下，再使用代码“ls”，就可以查看在根目录下的文件有哪些

如果有外接 U 盘，则可能要在根目录下的别文件夹下去查找它。

3.路径

“路径”指的是从一个地方到另一个地方所经过的路线。在不同领域有不同含义，比如在计算机领域，路径是指文件或文件夹在存储系统中的位置标识，用于准确找到相应资源；在地理方面，路径可以是行人、车辆等移动所经过的路线 。

路径有两种写法：绝对路径和相对路径

- **绝对路径**是从目录树的根目录开始写一只写到文件名

绝对路径是指在文件系统或网络中，从根目录开始到目标文件或目录的完整、明确的路径。它不依赖于当前工作目录，无论在什么环境下，都能唯一确定一个文件或目录的位置。例如在 Windows 系统中，“C:\Program Files\Adobe\Photoshop.exe” 就是一个绝对路径，从根盘符 “C:” 开始，依次通过各级文件夹直到目标文件；在 Linux 系统中，“/home/user/Documents/file.txt” 也是绝对路径，从根目录 “/” 开始描述路径到目标文件。

- **相对路径**不是从/开始写的，就都是相对路径。

其中可以直接用代码“cat +相对路径”这里输入的就是相对路径。

相对路径是指相对于当前工作目录或某个基准目录的文件或目录的路径表示方式。它不使用完整的绝对路径（从根目录开始的完整路径），而是基于当前位置来描述目标位置。例如，当前在 C:\project\src 目录下，要访问同一目录下的 data.txt 文件，相对路径可以简单写成 data.txt；若要访问 src 目录的上级目录中的 config.ini 文件，相对路径可以写成..\config.ini ，其中.. 表示上级目录。相对路径在编程、文件管理等场景中广泛应用，能使路径表示更简洁、灵活，便于在不同系统或环境中移动和部署项目时保持路径的有效性。

3./：其实就是在路径里文件之间的分隔符，和\是不一样的

/和\的区别是什么

- 在计算机领域，常用于路径表示，在 URL（网址）中，用“/”来分隔不同层级的目录或资源，如 https://www.example.com/page/1

- 在 Windows 系统的文件路径中常作为路径分隔符使用，例如“C:\Program Files\Python”。

4.当前工作目录与“pwd”

当前工作目录指的是在操作系统或编程语言环境中，程序或用户当前正在操作的目录位置。在这个目录下，程序默认进行文件的读取、写入等操作。

“pwd”代码可以显示当前工作目录是什么

5.Unix 路径与 Windows 路径的标准写法与区别

- **Unix 路径：**

以斜杠（/）作为路径分隔符。

路径从根目录（/）开始，根目录是整个文件系统层次结构的顶级目录。例如，一个文件在 Unix 系统中的路径可能是/home/user/Documents/file.txt，其中/home 是根目录下的一个目录，user 是/home 目录下的用户主目录，Documents 是 user 目录下的文件夹，file.txt 是 Documents 文件夹中的文件。

相对路径是相对于当前工作目录的路径。例如，当前工作目录是/home/user，要访问 Documents 目录下的 file.txt，相对路径可以写成 Documents/file.txt。

- **Windows 路径：**

以反斜杠（\）作为路径分隔符。不过在编程语言中，由于反斜杠在字符串中有转义作用，所以通常会使用双反斜杠（\\）来表示路径分隔符，或者使用正斜杠（/）也可以被识别。

路径通常以驱动器号（如 C:、D:等）开始，后跟一个冒号和反斜杠。例如，C:\Users\username\Documents\file.txt，其中 C:是驱动器号，\Users 是 C 盘下的一个目录，username 是\Users 目录下的用户文件夹，Documents 是 username 文件夹中的文件夹，file.txt 是 Documents 文件夹中的文件。

相对路径同样是相对于当前工作目录。例如，当前工作目录是 C:\Users\username，要访问 Documents 目录下的 file.txt，相对路径可以写成 Documents\file.txt 。

• **区别**

分隔符：Unix 系统使用正斜杠“/”作为路径分隔符，例如“/home/user/Documents”；而 Windows 系统传统上使用反斜杠“\”作为路径分隔符，如“C:\Users\user\Documents”。不过在 Windows 系统中，现在也支持使用正斜杠。

盘符表示：Unix 系统没有盘符的概念，根目录用“/”表示，所有的文件和目录都在这个根目录下的树形结构中。Windows 系统则通过盘符（如 C:、D:等）来区分不同的存储设备，每个盘符都有自己独立的根目录。

大小写敏感性：Unix 系统路径是大小写敏感的，“/home/user”和“/Home/User”被视为不同的路径。Windows 系统默认情况下路径不区分大小写，“C:\Users\user”和“C:\USERS\USER”通常被认为是同一个路径。

路径表示方式：Unix 系统中有相对路径和绝对路径，绝对路径从根目录“/”开始，相对路径基于当前工作目录。Windows 系统同样有相对路径和绝对路径，绝对路径以盘符开头，相对路径也是基于当前工作目录。

特殊目录表示：Unix 系统中，“.”表示当前目录，“..”表示父目录。在 Windows 系统中也有类似概念，但在命令行等操作中使用方式略有不同。例如在 Windows 命令提示符中也可以用“.”和“..”来表示当前目录和父目录，但在资源管理器地址栏中较少这样使用。

6.Shell（命令行）的基本语法结构

早期出现的是 sh 后面出现了 Bash，随着时间的发展发现也不够用了因此出现了 Zsh，是在 Bash 的基础上发展产生的。

问 AI：Bash 是什么？Zsh 是什么？二者是什么关系

Bash 是 Bourne Again SHell 的缩写，它是一种 Unix shell，是 GNU 计划的一部分。Bash 是许多 Linux 发行版以及 macOS 默认的 shell，它为用户提供了与操作系统进行交互的命令行界面，用户可以通过在 Bash 中输入命令来执行各种系统操作、运行程序等，并且支持脚本编程，能将一系列命令组合成脚本文件来自动执行复杂任务。

Zsh 是 Z shell 的缩写，它也是一种 Unix shell，是 Bash 的一个扩展和增强版本。Zsh 在保留 Bash 基本功能的基础上，增加了许多强大的功能和特性，例如更强大的命令行自动补全、主题化、插件系统等，能极大提高用户在命令行环境下的操作效率和体验，受到很多高级用户和开发者的喜爱。

它们的关系：Zsh 是在 Bash 基础上发展而来的，Zsh 兼容 Bash 的大部分语法，这意味着大部分为 Bash 编写的脚本在 Zsh 环境中也能正常运行。但 Zsh 又在功能上进行了大量扩展和改进，提供了更多高级特性，用户可以根据自己的需求和偏好选择使用 Bash 或 Zsh。

基本语法结构

• **空格分隔**

在 Shell（如 Bash、Zsh）中，空格用于分隔命令、选项和参数。例如命令 ls -l /home，ls 是命令，-l 是选项，/home 是参数，它们之间通过空格分隔。这使得 Shell 能够准确识别命令的各个部分并正确执行。

例如在 cat 代码后的空格、ls 后的空格。

• **短选项**

短选项是用于修改命令行为的一种简洁方式，通常以单个连字符 - 开头，后跟一个或多个字母。例如 ls -l 中的 -l，-l 选项让 ls 命令以长格式列出文件和目录的详细信息，包括文件权限、所有者、大小、修改时间等。多个短选项可以组合在一起，如 ls -la，等同于 ls -l -a，-a 选项表示显示隐藏文件。

• **长选项**

长选项用于更清晰地指定命令的行为，以两个连字符 -- 开头，后跟一个完整的单词。例如 ls --all，--all 选项和短选项 -a 功能相同，都是显示隐藏文件。长选项更具描述性，在某些复杂命令中有助于提高可读性。

• **参数**

参数是传递给命令的数据。它可以是文件名、目录名、数字或其他特定于命令的信息。例如在 cp file1.txt /tmp 命令中，file1.txt 是源文件参数，/tmp 是目标目录参数，cp 命令会将 file1.txt 复制到 /tmp 目录下。参数可以有多个，并且顺序通常很重要，不同的命令对参数的要求和解释也不同。

Shell（Bash、Zsh）通过这些基本语法结构（空格分隔、短选项、长选项、参数），用户能够灵活地调用各种命令并指定其行为，以完成各种系统管理和操作任务。

7.阅读在线手册-ls

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

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

列出文件（默认为当前目录）的信息。如果没有指定-cftuvSUX 或--sort，则按字母顺序排序条目。

```
PC@DESKTOP-FTJ84MN MINGW64 /
$ ls -lh
total 5.3M
drwxr-xr-x 1 PC 197121    0  3月 11 19:14 bin/
drwxr-xr-x 1 PC 197121    0  3月 11 19:14 cmd/
drwxr-xr-x 1 PC 197121    0  3月 11 19:14 dev/
drwxr-xr-x 1 PC 197121    0  3月 11 19:14 etc/
-rwxr-xr-x 1 PC 197121 136K  2月 13 11:00 git-bash.exe*
-rwxr-xr-x 1 PC 197121 135K  2月 13 11:00 git-cmd.exe*
-rw-r--r-- 1 PC 197121  19K  2月 13 11:12 LICENSE.txt
drwxr-xr-x 1 PC 197121    0  3月 11 19:14 mingw64/
dr-xr-xr-x 8 PC 197121    0  3月 16 09:50 proc/
-rw-r--r-- 1 PC 197121 271K  2月 13 11:12 ReleaseNotes.html
drwxr-xr-x 1 PC 197121    0  3月 16 09:49 tmp/
-rw-r--r-- 1 PC 197121 1.3M  3月 11 19:14 unins000.dat
-rwxr-xr-x 1 PC 197121 3.3M  3月 11 18:50 unins000.exe*
-rw-r--r-- 1 PC 197121  24K  3月 11 19:14 unins000.msg
drwxr-xr-x 1 PC 197121    0  3月 11 19:14 usr/
```

图 12

“ls -lh”代码的用途：**-l** —— use a long listing format

-h —— --human-readable

with -l and -s, print sizes like 1K 234M 2G etc.

“ls”是“list”的缩写，意为列出目录内容；“-l”参数表示以长格式显示文件和目录的详细信息，包括文件权限、所有者、文件大小、修改时间等；“-h”参数通常与“-l”一起使用，它的作用是将文件大小以人类可读的格式显示。

所有手册上的字母符号可以叠加，例如 l 和 h 可以叠加在一起

二、CP 命令与 MV 命令

1.CP 命令（copy）

```
PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ cat abc.txt
abc
hello world
你好

PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ cp abc.txt ../Downloads/

PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
```

图 13

利用 cat 命令，抓取 abc.txt 文件，再利用 cp 命令把在桌面上的 abc.txt 文件复制进下载中。

```
PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ cp abc.txt ../Downloads/xyz.txt
```

图 14

在图 13 的第 2 行代码后面添加一个新的文件名，如图 14 所示，意思是把文件复制过来并更改文件名为 xyz.txt（在这里 target 的意思是，前往桌面的上一级，下的下载文件夹，复制的文件名是 xyz.txt）

其中：abc.txt 是 copy the source 就是复制来源，而../Downloads/xyz.txt 就是 target 即目标路径，如果加上了文件名，那就是把前面的内容复制为后面的 target，没有给文件名就是保留原先的文件名。

```
PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ ls -l store
total 0

PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ ls -l store
total 1
-rw-r--r-- 1 PC 197121 26  3月 16 10:09 abc.txt

PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ cp store ../Downloads/
cp: -r not specified; omitting directory 'store'

PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
```

图 15

图 15 适用于将位于桌面上的文件夹 store 复制到 Downloads 中，并且

其中-r 的意思是你需要指定一个选项

图 16

-r 和-l 对于 ls 的意义是相似的，-r 也是 cp 命令行的一个小命令，可以通过在搜索引擎中搜索 man cp（manual cp）查找 cp 命令行下的小命令，如图 16，-r, --recursive（copy directories recursively 递归复制目录）并且这是一个最常用的命令，就是把文件夹中的文件夹/文件/文件夹下的文件/文件夹...全都复制进 target 中，所以称为递归的思想去复制。

```
PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ cp -r store ../Downloads/
```

图 17

如图 17，这样就可以把文件夹中的内容也都复制到 target 中了

图 18

结果如图 18 所示，在下载中是有复制好的文件的。

2.MV 命令（move）

```
PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ mv ../Downloads/xyz.txt ./

PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
```

图 19

图 19 代码的含义是：当前目录是在 desktop（桌面）上，把在当前目录下的上一级目录的 downloads 目录下的 xyz.txt 文件，移动到当前目录（./）

图 20

```
PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ cp -r store ../Downloads/store2

PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ mv ../Downloads/store2 ./
```

图 21

如图 21 所示将文件夹复制进下载中斌进行重命名。

第 2 行代码是将复制在下载中的文档移动到当前目录（桌面），其结果如图 22 所示，已移动到桌面了

图 22

为什么 mv 命令行没有-r 这个递归的小命令呢

因为 mv 命令只是更改了路径的名字，没有使用递归的算法，实际上文件文档的数据都没有发生变化。

三、mkdir 命令、ls 命令、rm 命令

1.mkdir 命令、创建文件夹

```
PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ mkdir myproject
```

图 23

图 23 这就是一个相对路径的典型案例，同时 mkdir 是用于创建新文件夹的终端命令。

2.ls 命令、查看复制后的文件夹内容

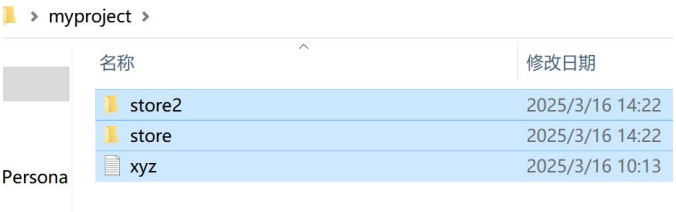

图 24

按照要求复制些文件夹进去（myproject 文件夹）

```
PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ ls -al myproject/
total 9
drwxr-xr-x 1 PC 197121  0  3月 16 14:22 ./
drwxr-xr-x 1 PC 197121  0  3月 16 11:23 ../
drwxr-xr-x 1 PC 197121  0  3月 16 14:22 store/
drwxr-xr-x 1 PC 197121  0  3月 16 14:22 store2/
-rw-r--r-- 1 PC 197121 26  3月 16 10:13 xyz.txt
```

图 25

使用代码“ls -al myproject/”或代码“ls -alh myproject/”对文件夹中的所有文件及修改时间进行列表，其中 a 代表 all，l 代表 long list，h 代表显示文件大小。

3.rm 命令（remove）

```
PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ rm xyz.txt

PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ rm -r myproject/

PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ rm store
rm: cannot remove 'store': Is a directory

PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ rm -r store
```

图 26

如图 26 所示，可以直接用 rm 命令删除一个文件，但是没有办法删除一个文件夹，例如“rm store”运行后显示，不能删除因为是文件夹，所以 rm 的用法和 cp 相似也需要使用**递归算法**，在删除文件夹的时候执行-r 的小命令完整命令如下：“rm -r store”。

！！！注意：所以在终端执行的删除是无法撤回的，不是开玩笑的，在回收里找不到的！！！

```
PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ rm -rf store2
```

图 27

有的时候会出现权限问题，所以会加一个小命令-rf，其中 f 代表 false，也就是强制删除。

四、df 命令、du 命令

1.Df 命令（disk free）

“df 命令”是 Unix 和类 Unix 操作系统中用于**显示文件系统磁盘空间使用情况**的命令。例如在 Linux 系统中，在终端输入“df -h”

```
PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ df
Filesystem          1K-blocks     Used Available Use% Mounted on
C:/Program Files/Git  97655804 53596860  44058944  55% /
```

图 28

如图 28 所示，可以看到当前文件系统磁盘空间的使用情况，其中查看根目录如图 29 所示，查找到 Git 下的根目录内容，用代码“ls -al /”实现。

```
PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ ls -al /
total 5428
drwxr-xr-x 1 PC 197121       0  3月 11 19:14 ./
drwxr-xr-x 1 PC 197121       0  3月 11 19:14 ../
drwxr-xr-x 1 PC 197121       0  3月 11 19:14 bin/
drwxr-xr-x 1 PC 197121       0  3月 11 19:14 cmd/
drwxr-xr-x 1 PC 197121       0  3月 11 19:14 dev/
drwxr-xr-x 1 PC 197121       0  3月 11 19:14 etc/
-rwxr-xr-x 1 PC 197121  138640  2月 13 11:00 git-bash.exe*
-rwxr-xr-x 1 PC 197121  138112  2月 13 11:00 git-cmd.exe*
-rw-r--r-- 1 PC 197121   18765  2月 13 11:12 LICENSE.txt
drwxr-xr-x 1 PC 197121       0  3月 11 19:14 mingw64/
dr-xr-xr-x 8 PC 197121       0  3月 16 14:45 proc/
-rw-r--r-- 1 PC 197121  276810  2月 13 11:12 ReleaseNotes.html
drwxr-xr-x 1 PC 197121       0  3月 16 14:44 tmp/
-rw-r--r-- 1 PC 197121 1336706  3月 11 19:14 unins000.dat
-rwxr-xr-x 1 PC 197121 3384048  3月 11 18:50 unins000.exe*
-rw-r--r-- 1 PC 197121   24183  3月 11 19:14 unins000.msg
drwxr-xr-x 1 PC 197121       0  3月 11 19:14 usr/
```

图 29

```
PC@DESKTOP-FTJ84MN MINGW64 ~/Desktop
$ df -h
Filesystem            Size  Used Avail Use% Mounted on
C:/Program Files/Git   94G   52G   43G  55% /
```

图 30

如图 30 所示，“df -h”的代码使得最后的输出具有可读性，输出为兆、G 等等。

剩余磁盘空间还是很重要的，要关注一下，不然可能文件就无法安装了等等。

2.du 命令

“du 命令”是 Unix 和类 Unix 操作系统中用于估计文件系统磁盘空间使用情况的命令。“du”是“disk usage”（磁盘使用情况）的缩写。该命令可以显示指定文件或目录所占用的磁盘空间大小，帮助用户了解磁盘空间的使用分布情况。例如，在终端中输入“du -h”，“-h”参数用于以人类可读的格式显示文件大小，它会列出当前目录下各个文件和子目录占用的磁盘空间大小。

```
PC@DESKTOP-FTJ84MN MINGW64 ~
$ cd miniconda3/

PC@DESKTOP-FTJ84MN MINGW64 ~/miniconda3
$ du .
864     ./conda-meta
23      ./condabin
3560    ./DLLs
0       ./envs
```

图 31

首先将目录定位至 miniconda3 中

！！！在定位过程中我遇到了一点点小问题：在输入的时候输入的是 miniconda，而不是 miniconda3，但是我并不知道自己输入的是错误的，因此我使用了 ls 命令，查看当前目录下的所有内容，由此确定了是因为我的文件名称输入有误，而不是工作路径有误。

其次使用“du .”的代码，让终端跑出在 miniconda3 中安装的所有文件程序及其大小，但这个过程过于漫长且不是我们所需要的，因此我们可以通过快捷键“control+C”打断程序的运行。

```
  -d, --max-depth=N
        print the total for a directory (or file, with --all) only
        if it is N or fewer levels below the command line argument;
        --max-depth=0 is the same as --summarize
```

图 32

```
PC@DESKTOP-FTJ84MN MINGW64 ~/miniconda3
$ du -d 1 .
864     ./conda-meta
23      ./condabin
3560    ./DLLs
0       ./envs
18      ./etc
1718    ./include
```

图 33

代码“du -d 1 .”的意思如图 32 所示，是指跑出最大深度，这里的 1 表示只显示当前目录下一级子目录的磁盘使用情况。“.”表示当前目录。

```
PC@DESKTOP-FTJ84MN MINGW64 ~/miniconda3
$ du -d 1 .
864     ./conda-meta
23      ./condabin
3560    ./DLLs
0       ./envs
18      ./etc
1718    ./include
104241  ./Lib
201756  ./Library
560     ./libs
108     ./Menu
982     ./Scripts
1690    ./share
9       ./shell
248     ./Tools
366367  .

PC@DESKTOP-FTJ84MN MINGW64 ~/miniconda3
$
```

图 34

跑一遍的结果如图 34 所示。

为了缩小跑程序的范围，我们挑选了其中一个部分来进行跑取。

五、使用 AI 解释 Bash 命令

1.查询代码

`du -s * | sort -nr > ~/report.txt`

- **`du -s *`：**

  `du`是“disk usage”的缩写，查看磁盘使用情况。

  `-s`选项表示显示每个指定文件或目录的总大小（summarize），不是显示每个子目录和文件的详细大小。

  `*`是通配符，表示当前目录下的所有文件和目录。所以这部分命令会列出当前目录下所有文件和目录占用磁盘空间的大小。

- `|`：这是管道符号，用于将前一个命令的输出作为后一个命令的输入。

- **`sort -nr`：**

  `sort`命令用于对输入进行排序。

  `-n`选项表示按照数字大小进行排序（numeric sort）。

  `-r`选项表示逆序排序（reverse sort），即从大到小排序。

  所以`sort -nr`会将`du -s *`输出的结果按照磁盘占用空间大小从大到小进行排序。

- **`> ~/report.txt`：**

  `>`是重定向符号，用于将命令的输出重定向到指定的文件中。

  `~/report.txt`表示将输出结果保存到用户主目录下的`report.txt`文件中。如果该文件不存在，会创建一个新文件；如果文件已存在，则会覆盖原有内容。

- **综上所述**，这条命令的作用是统计当前目录下所有文件和目录的磁盘使用情况，按照占用空间大小从大到小排序，并将结果保存到用户主目录下的`report.txt`文件中。

（以上为豆包 AI 询问出来的结果）

2.运用代码

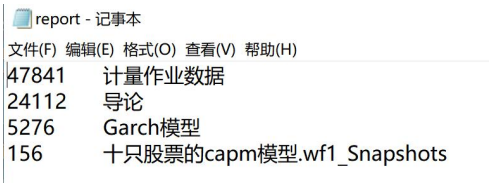

图 35

为方便运行，我找了桌面上的其中一个文件夹进行模拟命令行的用途，这个文件夹名称是“计量作业”运行结果后产生了一个文件夹为“report.txt”，所谓用户主目录在本机电脑上的目录如图 36 所示，因此在 PC 目录下找到了输出文件。

图 36

3.改编代码

想让最后输出的文档直接保存在当前所查看的目录中，因此改编代码如图 37 所示，并运行

图 37

与原代码的区别在于将“~/”改为“./”，其含义是，将文档保存到当前目录下（“./”）。

图 38

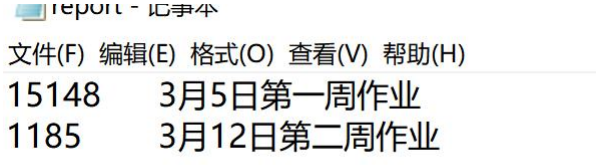

图 39

得到如图 38 与图 39 的结果，图 38 为输出文档的保存位置，图 39 为文件 report.txt 中的内容

！！！问题：在根目录下无法运行代码，只能是挑选一个较小的文件夹进行运行。猜测是因为无权限/内容太多

六、创建私密代码仓库

图 40

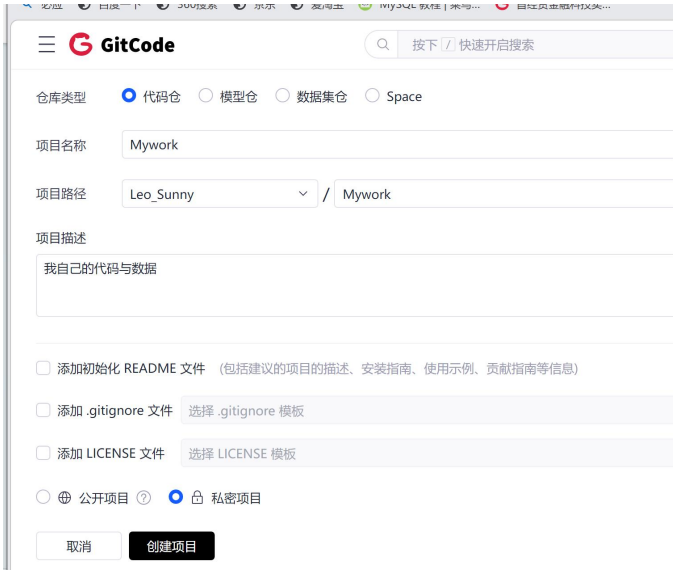

图 41

从图 40 中的新建项目中新建一个代码仓库，图 41 为具体选项，由于代码最常用，其他基本用不到因此我们创建为代码仓，并给项目名称进行命名为 Mywork，撰写项目描述，设置为私密项目后即可创建项目。

图 42

而后复制 SSH 域名，克隆到文件夹中。（现在 PC 下创建一个文件夹 repo 再进行 clone 如图 43 所示）

图 43

！！！在 clone 过程中出现了如图 43 所示的问题，出现了 warning，并且没有出现老师的需要输入 yes 的情况。Q:因为在上周的过程中已经输如果 yes，后续无需再输入，因此和老师的运行结果不是很一样。是新建且还未添加内容的仓库，该警告无需处理。

```
PC@DESKTOP-FTJ84MN MINGW64 ~/repo
$ ls -l
total 8
drwxr-xr-x 1 PC 197121   0  3月 18 09:28 Mywork/
-rw-r--r-- 1 PC 197121 981  3月 12 07:45 script1.py
drwxr-xr-x 1 PC 197121   0  3月 13 18:12 week01/

PC@DESKTOP-FTJ84MN MINGW64 ~/repo
$ cd Mywork/

PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ ls -l
total 0

PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ git log
fatal: your current branch 'main' does not have any commits yet
```

图 44

查看 repo 文件夹下的内容大小写作一个 long list。并**查看提交日志（git log）**，输出结果为没有提交过内容。（因为我们仓库还为空）

```
PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ cp ~/.gitconfig ./

PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ ls -l
total 0

PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ pwd
/c/Users/PC/repo/Mywork

PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ ls -al
total 5
drwxr-xr-x 1 PC 197121   0  3月 18 09:37 ./
drwxr-xr-x 1 PC 197121   0  3月 18 09:28 ../
drwxr-xr-x 1 PC 197121   0  3月 18 09:28 .git/
-rw-r--r-- 1 PC 197121 171  3月 18 09:37 .gitconfig
```

图 45

```
PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ cp ~/miniconda3/Scripts/tkconch-script.py ./
cp: cannot stat '/c/Users/PC/miniconda3/Scripts/tkconch-script.py': No such file or directory
```

图 46

！！！问题：和老师同样的步骤，发现没有这个文件，因此后续挑选了另外的.py 文件。

```
PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ cp ~/miniconda3/Scripts/cph-script.py ./
```

图 47

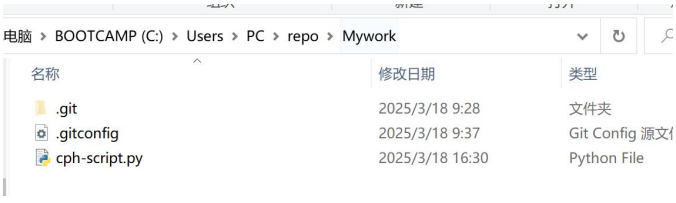

图 48

在修改文件名称，更换复制文件后，可以看到图 47 的运行结果为图 48，Mywork 文件夹中出现了对应的文件。

```
$ ls -al
total 6
drwxr-xr-x 1 PC 197121   0  3月 18 16:30 ./
drwxr-xr-x 1 PC 197121   0  3月 18 09:28 ../
drwxr-xr-x 1 PC 197121   0  3月 18 09:28 .git/
-rw-r--r-- 1 PC 197121 171  3月 18 09:37 .gitconfig
-rw-r--r-- 1 PC 197121 220  3月 18 16:30 cph-script.py

PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitconfig
        cph-script.py

nothing added to commit but untracked files present (use "git add" to track)
```

图 49

如图 49 可以看到使用“**git status**”代码可以查看当前状态，发现后面添加的两个文件都处于游离状态。

```
PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ git add .
warning: in the working copy of '.gitconfig', LF will be replaced by CRLF the next time Git touches it

PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitconfig
        new file:   cph-script.py
```

图 50

使用“**git add .**”代码，将两个处于游离状态的文件添加进当前目录，再使用“**git status**”查看状态，发现，两个文件已经添加进仓库中。

```
PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ git commit -m "add"
[main (root-commit) 1708050] add
 2 files changed, 16 insertions(+)
 create mode 100644 .gitconfig
 create mode 100644 cph-script.py

PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ git config --global user.name "Leo_Sunny"

PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ git config --global user.email "zhuyj_8219@163.com"
```

图 51

可以使用 git commit -m “...”代码进行提交，并进行备注，而后进行全局配置，只需要进行这一次的全局配置即可。

```
PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ git log
commit 17080505d43a2e2bbf0f29c9e51178bb9ada0496 (HEAD -> main)
Author: Leo_Sunny <Leo_Sunny@noreply.gitcode.com>
Date:   Tue Mar 18 16:39:29 2025 +0800
```

图 52

每次提交都会生成一份哈吉马，即如图 52 中划红色线的数字。

```
PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 560 bytes | 280.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Start Git Hooks Checking                                    [PASSED]
To gitcode.com:Leo_Sunny/Mywork.git
 * [new branch]      main -> main

PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ rm .gitconfig

PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    .gitconfig

no changes added to commit (use "git add" and/or "git commit -a")
```

图 53

如图 53 所示删除了文件.gitconfig，并查看状态

！补充：git 命令中的 stage 可以翻译为暂存区

```
PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ git log
commit 17080505d43a2e2bbf0f29c9e51178bb9ada0496 (HEAD -> main, origin/main)
Author: Leo_Sunny <Leo_Sunny@noreply.gitcode.com>
Date:   Tue Mar 18 16:39:29 2025 +0800

    add

PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ git commit -m "在 .gitconfig里添加了用户名和邮箱"
[main 3f64b3e] 在 .gitconfig里添加了用户名和邮箱
 1 file changed, 6 deletions(-)
 delete mode 100644 .gitconfig

PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ git log
commit 3f64b3e2a95e20cf2cbcdf310eb298ba61edc0d8 (HEAD -> main)
Author: Leo_Sunny <zhuyj_8219@163.com>
Date:   Tue Mar 18 17:04:22 2025 +0800

    在 .gitconfig里添加了用户名和邮箱

commit 17080505d43a2e2bbf0f29c9e51178bb9ada0496 (origin/main)
Author: Leo_Sunny <Leo_Sunny@noreply.gitcode.com>
Date:   Tue Mar 18 16:39:29 2025 +0800

    add
```

图 54

而后在图 54 中查看提交日志，发现目前仍然只是 1 笔提交，我们再次对.gitconfig 文件进行提交，并进行说明，用 **git commit -m“ ”** 代码，再次查看提交日志，这样就有了两笔提交。

其中也有区别，第一笔提交是在远程的（显示“origin/main”）第二笔提交是最新状态（显示“Head”），说明我们比远程多走了一步，因此要使用 **git push** 代码进行远程的更新，如图 55 所示。

```
PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ git push
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (1/1), done.
Writing objects: 100% (2/2), 293 bytes | 293.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Start Git Hooks Checking                                    [PASSED]
To gitcode.com:Leo_Sunny/Mywork.git
   1708050..3f64b3e  main -> main

PC@DESKTOP-FTJ84MN MINGW64 ~/repo/Mywork (main)
$ git log
commit 3f64b3e2a95e20cf2cbcdf310eb298ba61edc0d8 (HEAD -> main, origin/main)
Author: Leo_Sunny <zhuyj_8219@163.com>
Date:   Tue Mar 18 17:04:22 2025 +0800

    在 .gitconfig里添加了用户名和邮箱

commit 17080505d43a2e2bbf0f29c9e51178bb9ada0496
Author: Leo_Sunny <Leo_Sunny@noreply.gitcode.com>
Date:   Tue Mar 18 16:39:29 2025 +0800

    add
```

图 55

再查看提交日志，就可以看到远程和最新状态都是标注在最新一次提交的日志位置了，并在平台上也可以看到两笔提交了。

所以 gitcode 不只是可以托管线上的内容，电脑本机的内容也可以进行托管。