实践报告

高鹏宇
学号：2302007025

一、练习内容与结果

（一）shell 脚本

1. 使用 ls 命令列出所有文件（包括隐藏文件），并以人类可读的格式、按最近访问顺序排序，并以彩色文本显示输出结果。

   - 命令：

     ```
     ls -la --time-style=+%h %d %n --color=auto
     ```

   - 说明：
     - -la：显示所有文件和目录，包括隐藏文件（以. 开头的文件）。
     - –time-style=+%h %d %n：按最近访问时间排序。
     - –color=auto：以彩色文本显示输出结果。

2. 编写两个 bash 函数 marco 和 polo

   - 命令：

     ```
     #!/usr/bin/env bash

     # marco 函数：保存当前目录
     marco() {
         marco_dir=$(pwd)
         echo "Marco: Current directory saved as $marco_dir"
     }
     ```

```
# polo 函数：无论当前目录是什么，都返回 marco 保存的目录
polo() {
    echo "Polo: Returning to $marco_dir"
    cd "$marco_dir"
}
```

- 说明：
  - 将上述代码保存到 marco.sh 文件中。
  - 在终端中运行 source marco.sh 来加载函数。
  - 运行 marco 来保存当前目录。
  - 无论当前目录是什么，运行 polo 都会返回到 marco 保存的目录。

3. **随机数脚本**

- 命令：

```
#!/usr/bin/env bash

# 生成一个0到99之间的随机数
n=$(( RANDOM % 100 ))

# 检查随机数是否等于42
if [[ n -eq 42 ]]; then
    # 如果等于42，则打印错误信息到标准输出
    echo "Something went wrong"
    # 打印错误信息到标准错误输出
    >&2 echo "The error was using magic numbers"
    # 退出脚本并返回错误码1
    exit 1
fi
```

- 说明：

  - 使用 RANDOM 变量生成一个 0 到 99 之间的随机数，并将其赋值给变量 n。
  - 使用 if 语句检查变量 n 是否等于 42。
  - 如果 n 等于 42，脚本会执行两个 echo 命令：
    * 第一个 echo 将错误信息打印到标准输出（通常是终端）。
    * 第二个 echo 将错误信息重定向到标准错误输出（通常是终端）。
  - 然后，脚本通过 exit 1 命令退出，并返回一个非零的错误码，表示发生了错误。

4. **编写脚本，运行直到出错并记录输出**

- 命令：

```
#!/usr/bin/env bash

# 定义要运行的命令
command_to_run="your_command_here"

# 初始化尝试次数
attempts=0

# 循环运行命令，直到出错
while ! $command_to_run; do
    # 增加尝试次数
    ((attempts++))

    # 捕获标准输出和标准错误
    # 将标准输出重定向到文件
    $command_to_run > output.log 2>&1
done

# 检查是否出错
if [ $? -ne 0 ]; then
    # 将错误信息重定向到错误日志文件
```

```
    cat output.log > error.log

    echo "The command failed after $attempts attempts." >> error.log

    echo "The output was:"

    cat error.log
else
    echo "The command did not fail after $attempts attempts."
fi
```

- 说明：
  - 请将代码替换为你想要运行的实际命令。
  - 使用 while 循环运行命令，直到命令出错（即命令返回非零退出状态）。
  - 每次命令运行时，都会将标准输出和标准错误重定向到 output.log 文件。
  - 一旦命令出错，将 output.log 文件的内容复制到 error.log 文件，并记录尝试次数。
  - 最后，脚本会输出错误日志文件的内容。

5. **编写命令或脚本递归查找最近使用的文件**

- 命令：

```
find . -type f -printf "%T+ %p\n" | sort -r
```

- 说明：
  - find . -type f -printf ”
  - sort -r：按时间倒序排序。

6. **创建目录**

- 命令：

```
mkdir new_directory
```

- 说明：
  - mkdir：创建新目录。

7. **统计当前目录下文件数目**

- 命令：

```
find . -type f | wc -l
```

- 说明：
  - find . -type f：查找当前目录下的所有文件。
  - wc -l：计算行数，即文件数量。

8. **复制文件**

- 命令：

```
cp /path/to/source.txt /path/to/destination.txt
```

- 说明：
  - cp：复制文件。
  - /path/to/source.txt：源文件路径。
  - /path/to/destination.txt：目标文件路径。

9. **递归查找并删除特定扩展名的文件**

- 命令：

```
find . -type f -name "*.log" -exec rm -f {} \;
```

- 说明：
  - find . -type f -name ”*.log”：递归查找所有扩展名为.log 的文件。
  - -exec rm -f ：对找到的每个文件执行 rm -f 命令进行删除。

10. **将当前目录下的所有.jpg 图片文件压缩为一个 ZIP 文件**

    - 命令：

      ```
      zip -r images.zip *.jpg
      ```

    - 说明：
      - zip -r images.zip：创建一个名为 images.zip 的压缩文件。
      - *.jpg：将当前目录下所有.jpg 文件添加到压缩文件中。

（二）Vim 使用

1. **打开文件**

   ```
   vim filename
   ```

2. **进入插入模式**

   - i 进入插入模式，在光标前插入。
   - I 进入插入模式，在行首插入。
   - a 进入插入模式，在光标后插入。
   - A 进入插入模式，在行尾插入。

3. **退出插入模式**

   `Esc` 退出插入模式，返回普通模式。

4. **保存文件**

   `:w` 写入文件。
   `:w filename` 将文件保存为另一个名字。

5. **退出 Vim**

:q 退出Vim。
:q! 强制退出，不保存更改。

6. 保存并退出

:wq 保存更改并退出。
:x 同 :wq。

7. 撤销和重做

u 撤销。
Ctrl + r 重做。

8. 复制和粘贴

yy 复制整行。
3yy 复制三行。
p 粘贴。
d 删除（剪切）。
dd 删除（剪切）整行。

9. 查找和替换

/pattern 查找“pattern”。
n 跳转到下一个匹配项。
N 跳转到上一个匹配项。
:%s/old/new/g 全局替换“old”为“new”。

10. 多文件编辑

:e filename 打开另一个文件。
:bn 切换到下一个文件。

`:bp` 切换到上一个文件。

11. **撤销历史**

`:u` 撤销。
`:undolist` 列出撤销历史。

12. **全局命令**

`:g` 执行全局命令，例如 `:g/^$/d` 删除所有空行。

二、解题感悟

在使用过程中，我发现熟练使用 Shell 脚本可以极大地提高自动化任务的效率。我们编写好合适的 Shell 脚本，可以通过命令行执行复杂的任务，这对于批量处理文件和自动化工作流程非常有用。而 Vim 作为一个强大的文本编辑器，通过熟练掌握其快捷键和命令，可以非常快速地编辑文本。除此之外，Vim 在文本处理方面非常强大，特别是对于大文件的编辑，它的性能和效率往往超过其他文本编辑器。无论是 Shell 还是 Vim，我们都需要持续学习和实践来提高个人能力。

Github 地址：`https://github.com/Gao-py/class_homework`