

系统开发工具基础实验报告（一）

姓名：陈怡冰 学号：23020007007

September 5, 2024

目录

1	shell 工具和脚本	2
1.1	实验内容	2
1.2	实验结果	3
1.3	实验代码及关键步骤	10
1.4	实验中遇到的问题及解决方案	14
1.4.1	实例 9: 使用 kill 终止程序失败	14
1.4.2	实例 14: 重复运行某脚本在运行失败前只运行了一次	14
2	编辑器 Vim	17
2.1	实验内容	17
2.2	实验结果	17
2.3	实验代码及关键步骤	23
3	数据整理	25
3.1	实验内容	25
3.2	实验结果	25
3.3	实验代码及关键步骤	28
3.4	实验中遇到的问题	31
3.4.1	开机时间计算错误	31
4	实验心得	32
4.0.1	Shell 脚本编写	32
4.0.2	Vim 的定制化操作	32
4.0.3	数据查找与整理	32
5	我的 Github 仓库网址	33

实验部分 1 shell 工具和脚本

1.1 实验内容

1. 查找当前目录下，后缀为.txt 的文件。
2. 查找文件内容包含 “hello” 的文件。
3. 查找所有使用foo变量的代码文件。
4. 编写hello world脚本。
5. 使用grep命令从文件中查找包含”error”或”fail”字样的行，并统计这些行的总数。
6. 编写一个脚本，使用 mv 命令将当前目录下所有的.txt 文件的扩展名改为.bak。
7. 使用 tar 命令将当前目录下的所有.txt 文件压缩为一个名为 archive.tar.gz 的文件，然后解压缩该文件到一个新的目录 backup/中。
8. 组合命令与重定向：编写一个命令，列出当前目录下所有以.sh 结尾的文件，将其内容按字母顺序排序并输出到一个名为 sorted_scripts.txt 的文件中。
9. 使用 ps 命令查找所有运行中的 python 进程，并使用 kill 命令终止其中的一个进程。
10. 编写一个脚本，在指定目录及其所有子目录中递归查找并删除所有大小为 0 的文件。
11. 编写一个单行命令，先创建一个名为 output.txt 的文件，将当前日期写入文件，然后在文件末尾追加当前用户的登录名。
12. 阅读 man ls ，然后使用 ls 命令进行如下操作：
 - 所有文件（包括隐藏文件）
 - 文件打印以人类可以理解的格式输出（例如，使用 454M 而不是 454279954）
 - 文件以最近访问顺序排序
 - 以彩色文本显示输出结果

典型输出如下：

```
1  -rw-r--r--    1 user group 1.1M Jan 14 09:53 baz
2  drwxr-xr-x    5 user group 160 Jan 14 09:53 .
3  -rw-r--r--    1 user group 514 Jan 14 06:42 bar
4  -rw-r--r--    1 user group 106M Jan 13 12:12 foo
5  drwx-----+ 47 user group 1.5K Jan 12 18:08 ..
```

13. 编写两个 bash 函数 marco 和 polo 执行下面的操作。每当你执行 marco 时，当前的工作目录应当以某种形式保存，当执行 polo 时，无论现在处在什么目录下，都应当 cd 回到当时执行 marco 的目录。为了方便 debug，你可以把代码写在单独的文件 marco.sh 中，并通过 source marco.sh 命令，（重新）加载函数。
14. 假设您有一个命令，它很少出错。因此为了在出错时能够对其进行调试，需要花费大量的时间重现错误并捕获输出。编写一段 bash 脚本，运行如下的脚本直到它出错，将它的标准输出和标准错误流记录到文件，并在最后输出所有内容。加分项：报告脚本在失败前共运行了多少次。

```

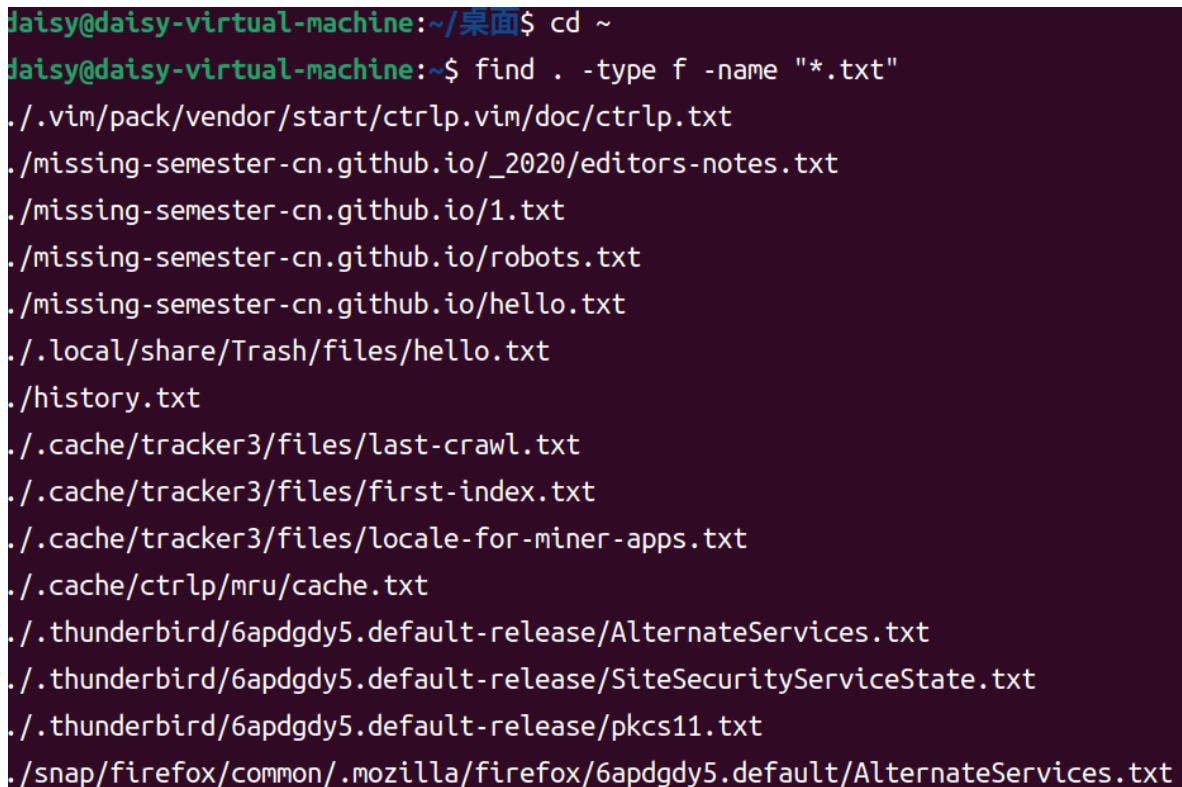
1  #!/usr/bin/env bash
2
3  n=$(( RANDOM % 100 ))
4
5  if [[ n -eq 42 ]]; then
6      echo "Something went wrong"
7      >&2 echo "The error was using magic numbers"
8      exit 1
9  fi
10
11 echo "Everything went according to plan"

```

1.2 实验结果

1. 实例 1

当前目录下后缀为.txt的文件。



```

daisy@daisy-virtual-machine:~/桌面$ cd ~
daisy@daisy-virtual-machine:~$ find . -type f -name "*.txt"
./vin/pack/vendor/start/ctrlp.vim/doc/ctrlp.txt
./missing-semester-cn.github.io/_2020/editors-notes.txt
./missing-semester-cn.github.io/1.txt
./missing-semester-cn.github.io/robots.txt
./missing-semester-cn.github.io/hello.txt
./.local/share/Trash/files/hello.txt
./history.txt
./.cache/tracker3/files/last-crawl.txt
./.cache/tracker3/files/first-index.txt
./.cache/tracker3/files/locale-for-miner-apps.txt
./.cache/ctrlp/mru/cache.txt
./.thunderbird/6apdgd5.default-release/AlternateServices.txt
./.thunderbird/6apdgd5.default-release/SiteSecurityServiceState.txt
./.thunderbird/6apdgd5.default-release/pkcs11.txt
./snap/firefox/common/.mozilla/firefox/6apdgd5.default/AlternateServices.txt

```

图 1.1: 实例 1 结果图

2. 实例 2

使用递归查找包含hello的文件

```
daisy@daisy-virtual-machine:~$ grep -rI "hello" .
./bash_history
./missing-semester-cn.github.io/_2020/version-control.md
./missing-semester-cn.github.io/_2020/security.md
./missing-semester-cn.github.io/_2020/course-shell.md
./missing-semester-cn.github.io/.git/index
./missing-semester-cn.github.io/.git/COMMIT_EDITMS
./missing-semester-cn.github.io/.git/logs/refs/heads/master
./missing-semester-cn.github.io/.git/logs/HEAD
./missing-semester-cn.github.io/static/files/subtitles/2020/shell-tools.sbv
./missing-semester-cn.github.io/_2019/shell.md
./local/share/Trash/info/hello.txt.trashinfo
./cache/gstreamer-1.0/registry.x86_64.bin
./viminfo
./snap/snap-store/common/.cache/gnome-software/odrs/ratings.json
./snap/snap-store/common/.cache/gnome-software/appstream/components.xmlb
./snap/firefox/common/.mozilla/firefox/6apdgd5.default/saved-telemetry-pings/1
```

图 1.2: 实例 2 结果图

3. 实例 3

查找所有使用`foo`变量的代码文件。

```
daisy@daisy-virtual-machine:~$ grep -rI "\bfoo\b" .
./vim/pack/vendor/start/ctrlp.vim/doc/ctrlp.cnx
./vim/pack/vendor/start/ctrlp.vim/doc/ctrlp.txt
./missing-semester-cn.github.io/_2020/debugging-profiling.md
./missing-semester-cn.github.io/_2020/files/vimrc
./missing-semester-cn.github.io/_2020/command-line.md
./missing-semester-cn.github.io/_2020/version-control.md
./missing-semester-cn.github.io/_2020/editors.md
./missing-semester-cn.github.io/_2020/metaprogramming.md
./missing-semester-cn.github.io/_2020/shell-tools.md
./missing-semester-cn.github.io/_2020/potpourri.md
./missing-semester-cn.github.io/.git/objects/pack/pack-8cc6c265d474233494b8658f17875a7bfff42f30.pack
```

图 1.3: 实例 3 结果图

4. 实例 4

编写`hello world`脚本。运行，打印出`Hello, World!`。

```
daisy@daisy-virtual-machine:~$ vim hello.sh
daisy@daisy-virtual-machine:~$ source hello.sh
Hello, World!
```

图 1.4: 实例 4 结果图

5. 实例 5

使用 `grep` 命令从文件中查找包含“error”或“fail”字样的行，并统计这些行的总数为1。

```
daisy@daisy-virtual-machine:~$ grep -Ei "error|fail" ~/vimrc | wc -l
1
```

图 1.5: 实例 5 结果图

6. 实例 6

编写一个脚本，使用 `mv` 命令将当前目录下所有的.txt 文件的扩展名改为.bak。

我的当前目录下只有 `hello.txt`。但是我的命令中使用了 `*.txt`，详情见“实验步骤”

```
daisy@daisy-virtual-machine:~$ cd missing-semester-cn.github.io
daisy@daisy-virtual-machine:~/missing-semester-cn.github.io$ ls
1.txt      apple-touch-icon.png  favicon.ico  index.md    robots.txt
_2019      CNAME                 Gemfile      _layouts    static
_2020      _config.yml           Gemfile.lock lectures.html
404.html   favicon-16x16.png     hello.txt    license.md
about.md   favicon-32x32.png     _includes    README.md
```

图 1.6: 实例 6 结果图：使用脚本前仍是 `hello.txt`

```
daisy@daisy-virtual-machine:~/missing-semester-cn.github.io$ vim bak.sh
daisy@daisy-virtual-machine:~/missing-semester-cn.github.io$ source bak.sh
daisy@daisy-virtual-machine:~/missing-semester-cn.github.io$ chmod +x bak.sh ./bak.sh
daisy@daisy-virtual-machine:~/missing-semester-cn.github.io$ ls
1.bak      apple-touch-icon.png  favicon-32x32.png  _includes    README.md
_2019      bak.sh               favicon.ico        index.md     robots.bak
_2020      CNAME               Gemfile            _layouts     static
404.html   _config.yml          Gemfile.lock       lectures.html
about.md   favicon-16x16.png    hello.bak          license.md
```

图 1.7: 实例 6 结果图：使用脚本后 `hello.txt` 被改为 `hello.bak`

7. 实例 7

将主目录中的 `history.txt` 文件压缩为 `archive.tar.gz` 文件，并在新创建的 `backup` 目录中解压缩。

```

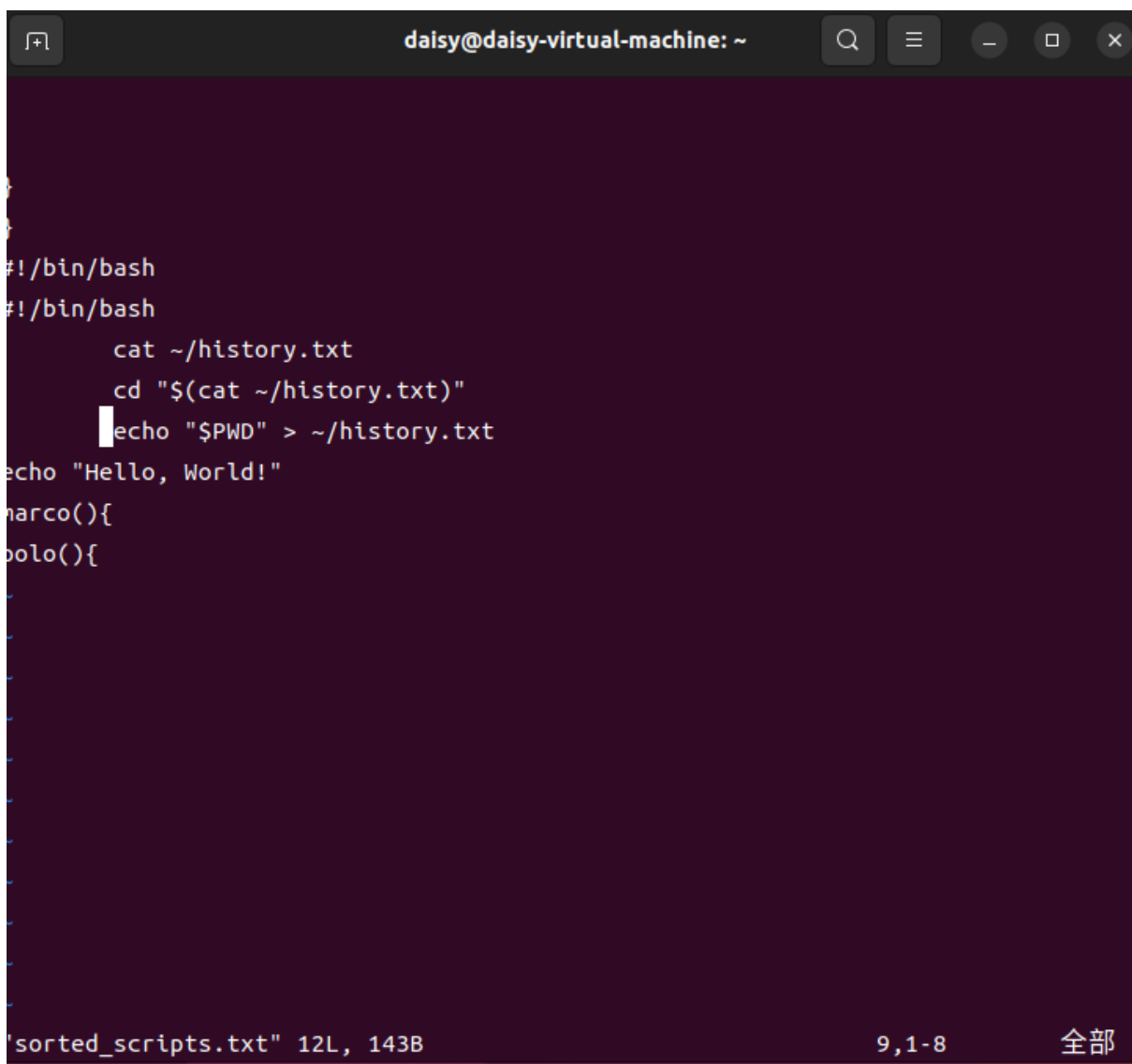
daisy@daisy-virtual-machine:~$ ls
公共的  图片  音乐          backup      marco.sh          try
模板    文档  桌面          hello.sh     missing-semester-cn.github.io vimrc
视频    下载  archive.tar.gz history.txt  snap
daisy@daisy-virtual-machine:~$ cd -
/home/daisy/backup
daisy@daisy-virtual-machine:~/backup$ ls
history.txt

```

图 1.8: 实例 7 结果图

8. 实例 8

是按照内容的首字母逐行输出的，结果不是很理想。



```

daisy@daisy-virtual-machine: ~
#!/bin/bash
#!/bin/bash
    cat ~/history.txt
    cd "$(cat ~/history.txt)"
    echo "$PWD" > ~/history.txt
echo "Hello, World!"
marco(){
bolo(){

```

'sorted_scripts.txt" 12L, 143B 9,1-8 全部

图 1.9: 实例 8 结果图

9. 实例 9

使用 kill 命令通过 PID 结束进程但是操作不被允许。

```

daisy@daisy-virtual-machine:~$ ps aux | grep python
root          704  0.0  0.5 44100 21248 ?        Ss   10:05   0:00 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
root          860  0.0  0.5 121224 23680 ?        Ssl  10:05   0:00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-signal
root         3373 14.2  5.4 386940 215772 ?        Sl   10:07   1:00 /usr/bin/python3 /usr/bin/unattended-upgrade
daisy        11336  0.0  0.4  38984 19200 ?        S    10:10   0:00 /usr/bin/python3 /usr/bin/gnome-terminal --wait
root        27142  6.7  3.4 386940 136916 ?        S    10:14   0:00 /usr/bin/python3 /usr/bin/unattended-upgrade
root        27177  0.0  2.0 386940  81456 ?        S    10:14   0:00 /usr/bin/python3 /usr/bin/unattended-upgrade
root        27201 35.0  4.4 201760 176120 ?        SN   10:14   0:01 /usr/bin/python3 /usr/lib/update-notifier/apt-check --human-readable
daisy       27316  0.0  0.0  12324  2560 pts/0    S+   10:14   0:00 grep --color=auto python

```

图 1.10: 实例 9 结果图

10. 实例 10

编写一个脚本，在指定目录及其所有子目录中递归查找并删除所有大小为 0 的文件。

```

daisy@daisy-virtual-machine:~$ vim delete.sh
daisy@daisy-virtual-machine:~$ source delete.sh
daisy@daisy-virtual-machine:~$ delete .
./.local/share/nautilus/tracker2-migration-complete
./.local/share/gnome-shell/gnome-overrides-migrated
./.local/share/gnome-settings-daemon/input-sources-converted
./.config/.gsd-keyboard.settings-ported
./.config/enchant/en_US.dic
./.config/enchant/en_US.exc
./.sudo_as_admin_successful
./.cache/thunderbird/6apdgd5.default-release/.startup-incomplete
./.cache/tracker3/files/.meta.isrunning
./.thunderbird/6apdgd5.default-release/.parentlock
./snap/firefox/common/.mozilla/firefox/6apdgd5.default/datareporting/archived/2024-08/1724748012110.99fe5d33-641a-438a-baa5-b08bd80b235f.health.jsonlz4

```

图 1.11: 实例 10 结果图

11. 实例 11

编写一个单行命令，先创建一个名为 output.txt 的文件，将当前日期写入文件，然后在文件末尾追加当前用户的登录名。


```
daisy@daisy-virtual-machine: ~
2024年 09月 03日 星期二 10:46:21 CST
daisy
~
~
~
```

图 1.12: 实例 11 结果图

12. 实例 12

所有文件（包括隐藏文件）做到：

1. 以人类可以理解的格式输出（例如，使用 454M 而不是 454279954）
2. 以最近访问顺序排序
3. 以彩色文本显示输出结果

```
daisy@daisy-virtual-machine:~$ ls --all -lht --color=auto
总计 164K
drwxr-x--- 21 daisy daisy 4.0K 9月 3 10:47 .
-rw----- 1 daisy daisy 11K 9月 3 10:47 .viminfo
-rw-rw-r-- 1 daisy daisy 49 9月 3 10:46 output.txt
-rw-rw-r-- 1 daisy daisy 67 9月 3 10:46 delete.sh
drwx----- 13 daisy daisy 4.0K 9月 3 10:33 .config
-rw-rw-r-- 1 daisy daisy 18 9月 3 10:11 sorted_scripts.txt
-rw----- 1 daisy daisy 8.8K 9月 3 10:10 .bash_history
drwxrwxr-x 2 daisy daisy 4.0K 9月 2 22:49 backup
-rw-rw-r-- 1 daisy daisy 140 9月 2 22:48 archive.tar.gz
drwxrwxr-x 9 daisy daisy 4.0K 9月 2 22:45 missing-semester-cn.github.io
-rw-rw-r-- 1 daisy daisy 33 9月 2 22:26 hello.sh
drwxrwxr-x 3 daisy daisy 4.0K 8月 30 11:37 try
-rw-rw-r-- 1 daisy daisy 16 8月 30 11:34 history.txt
-rw----- 1 daisy daisy 20 8月 30 11:32 .lessht
-rw----- 1 daisy daisy 3.4K 8月 30 11:23 vimrc
-rwxrwxr-x 1 daisy daisy 110 8月 30 11:03 marco.sh
```

图 1.13: 实例 12 结果图

13. 实例 13

使用 marco 和 polo 命令，实现从/try 目录到主目录。

```
daisy@daisy-virtual-machine:~$ vim marco.sh
daisy@daisy-virtual-machine:~$ source marco.sh
daisy@daisy-virtual-machine:~$ cd try
daisy@daisy-virtual-machine:~/try$ marco
/home/daisy/try
daisy@daisy-virtual-machine:~/try$ cd -
/home/daisy
daisy@daisy-virtual-machine:~$ polo
daisy@daisy-virtual-machine:~/try$
```

图 1.14: 实例 13 结果图

14. 实例 14

运行题目提供的脚本一直未出错，所以再判断条件里写了一些提示语，并在运行完 100 次以后停止运行。

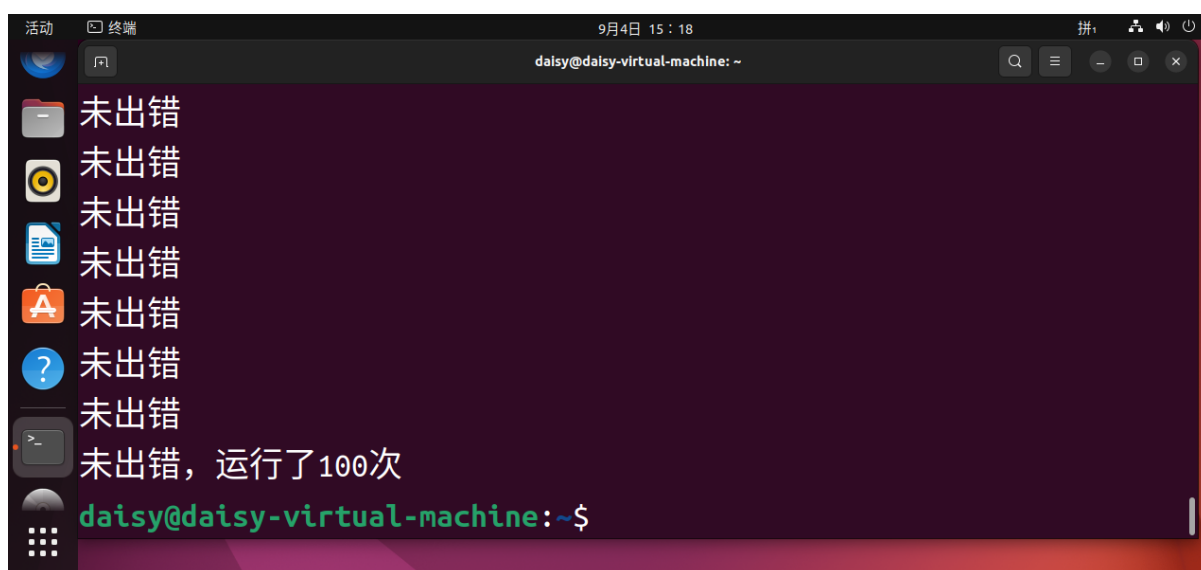


图 1.15: 实例 14 结果图

1.3 实验代码及关键步骤

1. 使用

```
1 find . -type f -name "*.txt"
```

`-name "*.txt"`: 匹配所有以.txt 结尾的文件。

2. 使用

```
1 grep -rl "hello"
```

`-r`: 表示递归搜索，即搜索当前目录及其所有子目录。`-l`: 表示只显示匹配的文件名，而不是显示具体的匹配内容。

3. 使用

```
1 grep -rl "\bfoo\b"
```

`-r`: 表示递归搜索，查找当前目录及其所有子目录中的文件。`-l`: 表示只显示匹配的文件名，而不是显示具体的匹配内容。`\bfoo\b`: 使用 `\b` 是为了匹配整个单词 `foo`，防止匹配到类似 `foobar` 或 `foot` 等不完整的变量名。

4. 实例 4 的 Bash 脚本

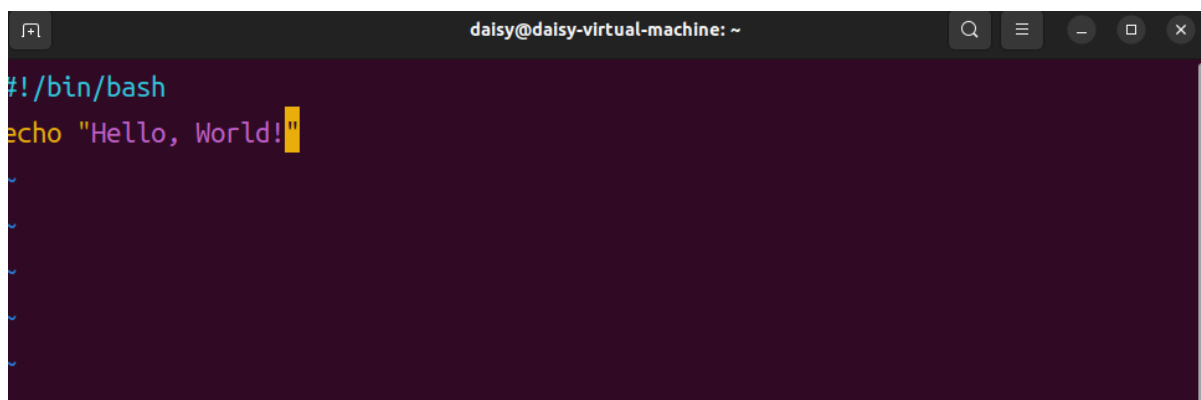


图 1.16: Bash 脚本

5. 使用

```
1 grep -Ei "error|fail" filename.txt | wc -l
```

`-E`: 启用扩展正则表达式，以便使用 `|` 来匹配多个模式。
`-i`: 忽略大小写匹配，因此它会匹配 `error`, `Error`, `FAIL`, `Fail` 等等。
`"error|fail"`: 搜索模式，匹配包含 `"error"` 或 `"fail"` 的行。
`filename.txt`: 这是要搜索的文件名。替换为你的实际文件名。
`| wc -l`: 将匹配的行传递给 `wc -l`，该命令会统计行数。

6. 使用

```
1 chmod +x bak.sh ./bak.sh
```

改变文件权限，使其变成可以直接执行的文件。

```
daisy@daisy-virtual-machine: ~/missing-semester-cn.github.io
#!/bin/bash

for file in *.txt;
do
    if [ -e "$file" ]; then
        mv "$file" "${file%.txt}.bak"
    fi
done
```

图 1.17: 实例 6 的 Bash 脚本内容

7. (a) 使用

```
1 tar -czvf archive.tar.gz *.txt
```

压缩所有后缀为.txt到archive.tar.gz。

(b) 使用

```
1 tar -xzf archive.tar.gz -C backup/
```

使用gzip解压指定的归档文件archive.tar.gz。到指定的目录 backup 目录中，并显示处理的文件

8. 使用

```
1 ls *.sh | sort > sorted_scripts.txt
```

将后缀为.sh 的文件排序后输出到指定文件。

9. 编写脚本，传入文件目录删除当前目录下的大小为 0 的文件，find 命令默认递归。

```
#!/bin/bash

delete(){
    find "$1" -type f -size 0 -print -delete
}

~
~
```

图 1.18: 脚本内容

10. 使用

```
1 date > output.txt && echo $USER >> output.txt
```

并把环境变量用户名追加写入 output.txt

11. `ls --all -lht --color=auto`

-all: 显示所有文件，也可以用-a 代替

-l: 输出详细信息

-h: 易读模式，-human-readable

-t: 按照时间顺序输出

-color[=WHEN]:colorize the output; WHEN can be 'always' (default if omitted),'auto', or 'never'; more info below.

12. marco 负责把当前文件名输入到 history.txt 文件并保存，polo 负责获得 history.txt 中的目录名并进入目录。



```
#!/bin/bash

marco(){
    cat ~/history.txt
    echo "$PWD" > ~/history.txt
}

polo(){
    cd "$(cat ~/history.txt)"
}
```

图 1.19: 实例 13 脚本内容

13. 实例 14 的代码，如下所示。

因为一直运行未出错，所以写了一些提示语，并在运行够 100 次时结束脚本。

```
daisy@daisy-virtual-machine: ~  
#!/usr/bin/env bash  
  
count=0  
output="output.txt"  
  
> "$output"  
  
while true  
do  
    let count+=1  
    {  
        chmod +x /home/daisy/subject.sh  
        status=$?  
    } >> "$output" 2>&1  
  
    if [[ $status -ne 0 || $count -eq 100 ]]  
    then  
        break  
    else  
        echo "未出错"  
    fi  
done  
  
if [[ $count -eq 100 ]]  
then  
    echo "未出错, 运行了100次"  
else  
    echo "在错误之前, 运行了 $count 次" | tee -a "$output"  
fi
```

2,0-1 全部

图 1.20: 实例 14 代码图

1.4 实验中遇到的问题及解决方案

1.4.1 实例 9: 使用 kill 终止程序失败

- grep 操作使用 kill 命令，显示没有那个进程。

```
daisy@daisy-virtual-machine:~$ kill 27316
bash: kill: (27316) - 没有那个进程
daisy@daisy-virtual-machine:~$ kill -9 27316
bash: kill: (27316) - 没有那个进程
daisy@daisy-virtual-machine:~$ kill 27316
bash: kill: (27316) - 没有那个进程
daisy@daisy-virtual-machine:~$ kill 12324
bash: kill: (12324) - 没有那个进程
daisy@daisy-virtual-machine:~$ kill 2560
bash: kill: (2560) - 没有那个进程
```

图 1.21: 实例 9: 没有那个进程

- 解决方案
是因为 grep 进程在命令执行后立即退出，所以使用 kill 命令时，这个进程已经不存在了，所以会提示“没有那个进程”。

1.4.2 实例 14: 重复运行某脚本在运行失败前只运行了一次

- 编写error.sh脚本重复运行subject.sh脚本直至出错，并输出出现错误前运行了多少次，结果输出在运行失败前只运行了一次。

```
daisy@daisy-virtual-machine:~$ bash error.sh
在错误之前，运行了 1 次
daisy@daisy-virtual-machine:~$ source error.sh
在错误之前，运行了 1 次
```

图 1.22: 实例 14 运行图

- 解决方案
– 打开日志看了一下，结果是没有运行权限。

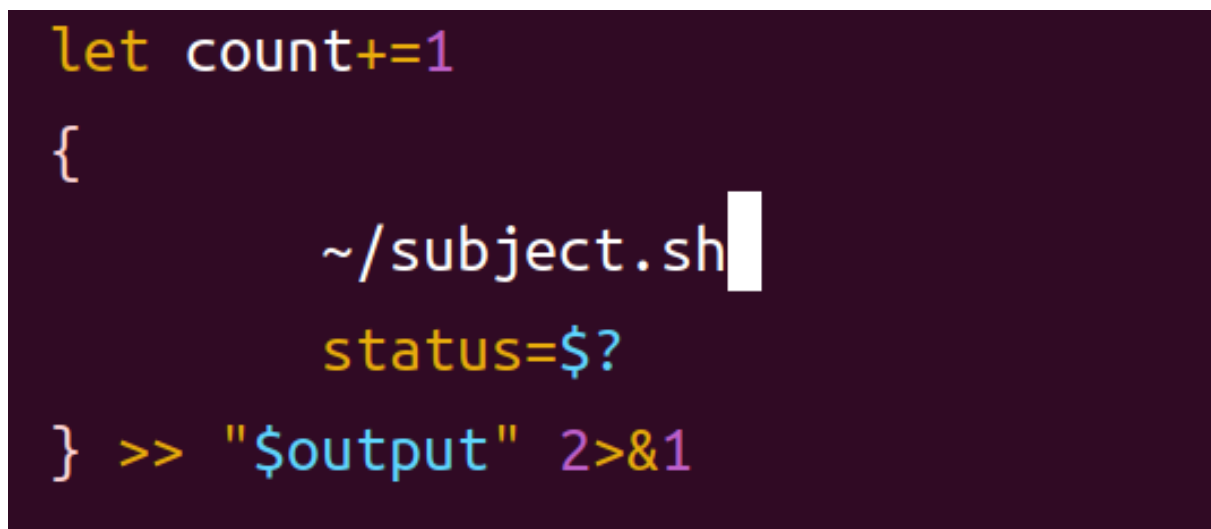


```
bash: /home/daisy/subject.sh: 权限不够
在错误之前, 运行了 1 次

output.txt" 2L, 77B 1,1 全部
```

图 1.23: output.txt 日志

- 使用 `chmod` 命令赋予脚本运行权限。



```
let count+=1
{
    ~/subject.sh
    status=$?
} >> "$output" 2>&1
```

图 1.24: 修改前程序命令


```
let count+=1
{
    chmod +x /home/daisy/subject.sh
    status=$?
} >> "$output" 2>&1
```

图 1.25: 修改后程序命令

实验部分 2 编辑器 Vim

2.1 实验内容

1. 使用 Vim 中的替换命令，将文件中所有的”foo” 替换为”bar”，并仅替换当前行的”foo” 为”bar”。
2. 在 Vim 中同时打开两个或多个文件，并切换不同文件的编辑窗口。
3. 使用 Vim 的可视模式选中文本，并对选中的文本进行以下操作：
 - 复制
 - 删除
 - 替换
4. 在 Vim 的配置文件 `/.vimrc` 中添加自定义快捷键，使得按下 `<F2>` 时能够快速注释掉当前行。
5. 在 Vim 中启用拼写检查，查找并纠正文本文件中的拼写错误。
6. 在 Vim 中打开多个文件，并将它们分别放置在不同的分屏和标签页中。
7. 在 Vim 中编辑文件时，执行一个外部 shell 命令（如 `ls` 或 `grep`），并将结果插入到当前文件中。
8. 在 Vim 中录制一个宏，用于在多行文本中重复执行一组操作，然后回放该宏。
9. 安装和配置一个插件：`ctrlp.vim`.
 - 用 `mkdir -p ~/.vim/pack/vendor/start` 创建插件文件夹
 - 下载这个插件：`cd ~/.vim/pack/vendor/start; git clone https://github.com/ctrlpvim/ctrlp.vim`
 - 阅读这个插件的文档。尝试用 `CtrlP` 来在一个工程文件夹里定位一个文件，打开 Vim，然后用 Vim 命令控制行开始:CtrlP.
 - 自定义 CtrlP：添加 configuration 到你的 `/.vimrc` 来用按 `Ctrl-P` 打开 CtrlP

2.2 实验结果

1. 实例 1
使用

```
1  :%s/foo/bar/g
```

将 foo 替换成 bar。

```
daisy@daisy-virtual-machine: ~  
bar bar  
bar  
bar  
bar  
bar  
  bar  
~  
~  
~  
$ld 行 $ld 次替换
```

图 2.1: 实例 1 结果图

2. 实例 2

使用

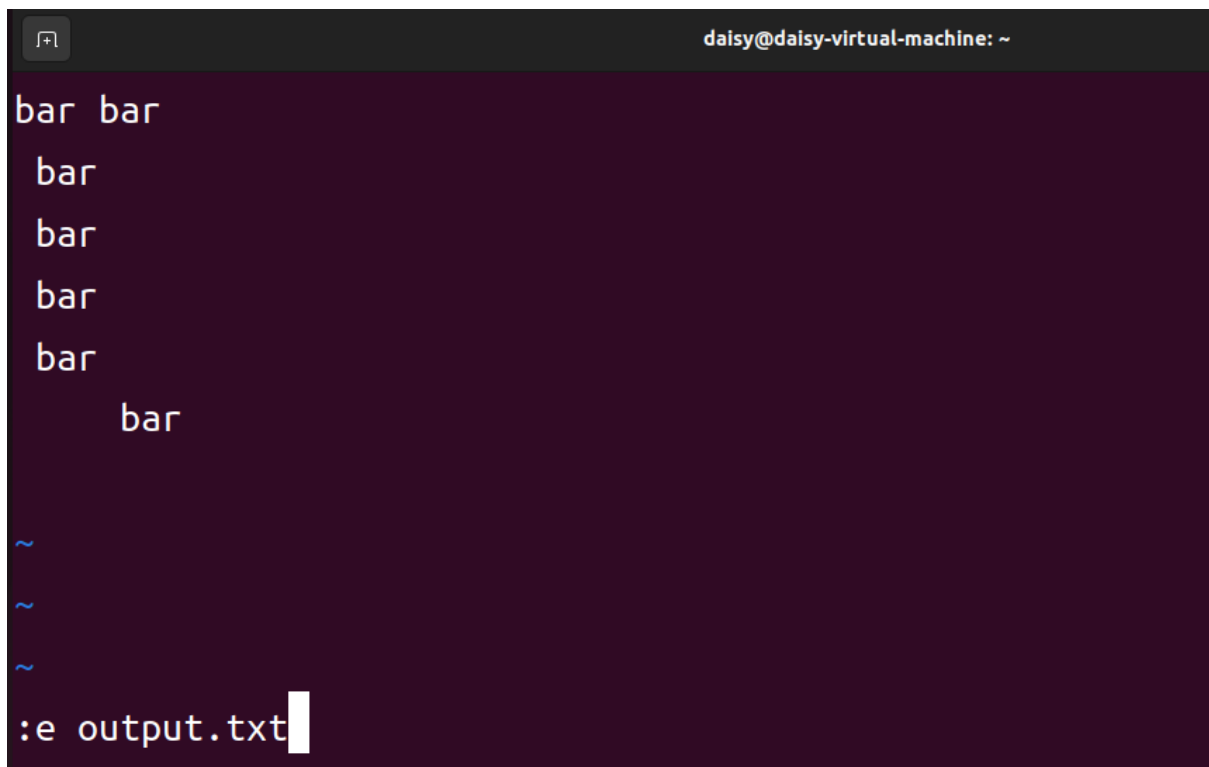
```
1 vim output.txt foo.txt
```

同时打开多个文件。

使用

```
1 :e output.txt
```

跳转到 output.txt。

A terminal window with a dark purple background. The text 'bar bar' is on the first line, 'bar' on the second, 'bar' on the third, 'bar' on the fourth, 'bar' on the fifth, and 'bar' on the sixth line. Below these are three tilde '~' characters. At the bottom, the command ':e output.txt' is entered, followed by a white cursor block.

```
daisy@daisy-virtual-machine: ~  
bar bar  
bar  
bar  
bar  
bar  
bar  
~  
~  
~  
:e output.txt
```

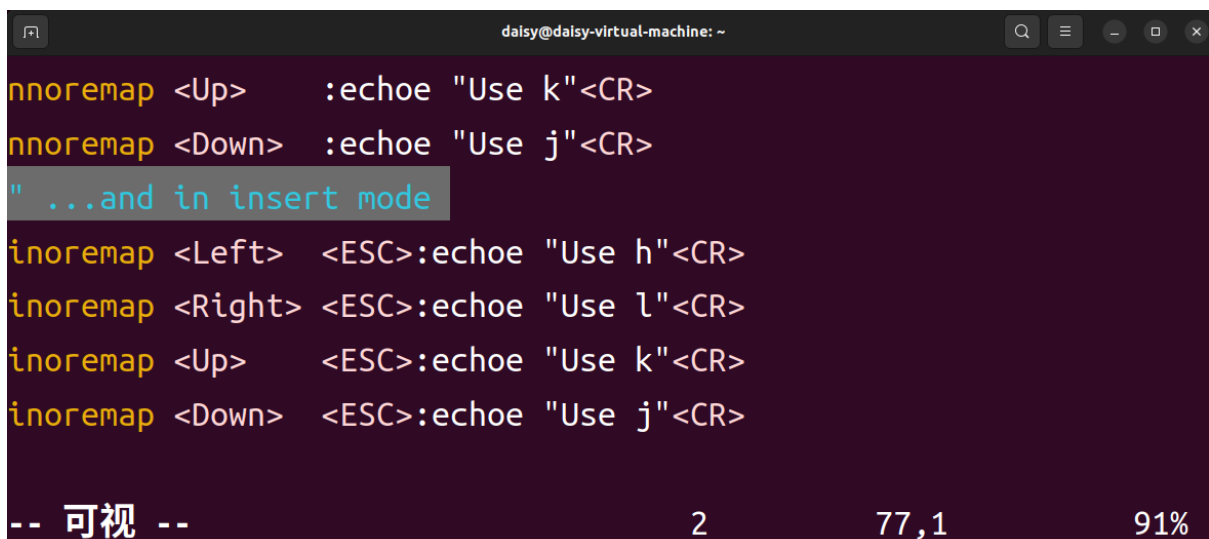
图 2.2: 实例 2 结果图

3. 实例 3

按 **v** 进入字符可视模式，然后使用箭头键或 **h**、**j**、**k**、**l** 键来移动光标以选中文本。

进入可视模式并选中想要复制的文本后，按 **y** ("**y**ank") 复制选中的文本。可以在可视模式下按 **y** 复制文本，也可以退出可视模式后按 **p** 粘贴复制的内容。

进入可视模式并选中想要删除的文本后，按 **d** 删除选中的文本。删除文本后，文本会被自动放入 Vim 的寄存器中，你可以使用 **p** 或 **P** 在需要的地方粘贴删除的内容。

A terminal window with a dark purple background. It shows Vim configuration lines for mapping arrow keys to echo commands. A line is highlighted in light blue. At the bottom, there is a status line with '2', '77,1', and '91%'.

```
daisy@daisy-virtual-machine: ~  
nnoremap <Up>      :echoe "Use k"<CR>  
nnoremap <Down>    :echoe "Use j"<CR>  
" ...and in insert mode  
inoremap <Left>    <ESC>:echoe "Use h"<CR>  
inoremap <Right>   <ESC>:echoe "Use l"<CR>  
inoremap <Up>      <ESC>:echoe "Use k"<CR>  
inoremap <Down>    <ESC>:echoe "Use j"<CR>  
-- 可视 --                2                77,1                91%
```

图 2.3: 实例 3 进入可视模式

4. 实例 4

设置自定义快捷键，按下 **F2** 后，注释到该行。


```

daisy@daisy-virtual-machine: ~
8月 29 23:02:16 daisy-virtual-machine systemd[988]: Startup finished in 4.805s
.
8月 29 23:02:32 daisy-virtual-machine systemd[1]: Startup finished in 3.467s (kernel) + 22.661s (userspace) = 26.129s
.
8月 29 23:04:40 daisy-virtual-machine
@@@
starttime.txt      20,50-49      95% foo.txt      6,6      全部

```

图 2.6: 实例 6 垂直分屏

- 使用 `vim -o starttime.txt foo.txt` 形成水平分屏。

```

daisy@daisy-virtual-machine: ~
9月 02 21:18:50 daisy-virtual-machine systemd[1]: Startup finished in 3.599s (kernel) + 22.504s (userspace) = 26.103s.
8月 30 10:34:08 daisy-virtual-machine systemd[1023]: Startup finished in 5.622s
.
starttime.txt      9,50-49      40%
bar
bar
bar
foo.txt            6,6      底端

```

图 2.7: 实例 6 水平分屏

7. 实例 7

使用 `r!` 可以将其他指令的内容插入到文件中。

```

1 :r!  ls -lt

```

```
daisy@daisy-virtual-machine: ~  
-rw-rw-r-- 1 daisy daisy 140 9月 2 22:48 archive.tar.gz  
drwxrwxr-x 9 daisy daisy 4096 9月 2 22:45 missing-semester-cn.github.io  
-rw-rw-r-- 1 daisy daisy 33 9月 2 22:26 hello.sh  
-rwxrwxr-x 1 daisy daisy 110 8月 30 11:03 marco.sh  
drwxr-xr-x 3 daisy daisy 4096 8月 25 14:09 下载  
drwx----- 5 daisy daisy 4096 8月 25 14:09 snap  
drwxr-xr-x 2 daisy daisy 4096 8月 25 12:38 桌面  
drwxr-xr-x 2 daisy daisy 4096 8月 25 11:41 公共的  
drwxr-xr-x 2 daisy daisy 4096 8月 25 11:41 视频  
drwxr-xr-x 2 daisy daisy 4096 8月 25 11:41 图片  
27,1 70%
```

图 2.8: 实例 7 将当前目录下文件列表插入到文件中

8. 实例 8

使用录制一个将改行注释掉的宏，然后在可视模式下回放宏。

```
daisy@daisy-virtual-machine: ~  
#!/bin/bash  
  
#for file in *.txt;  
#do  
#  if [ -e "$file" ]; then  
#    mv "$file" "${file%.txt}.bak"  
#  fi  
#done  
#
```

图 2.9: 实例 8: 将 bak.sh 所有行注释掉

9. 实例 9

按照题目所给步骤，就可以使用自定义的快捷键<CtrlP>进入模糊搜索模式。

```
daisy@daisy-virtual-machine: ~
for file in *.txt;
do
    if [ -e "$file" ]; then
        mv "$file" "${file%.txt}.bak"
    fi
done

bak.sh 3,1 底端
> 404.html
> index.md
> Gemfile
> you.txt
> CNAME
prt path <mru>={ files }=<buf> <-> <sing-semester-cn.github.io
```

图 2.10: 实例 9 结果图

2.3 实验代码及关键步骤

1. 实例 4 设置快捷键，注释掉选中行。在 vimrc 文件中设置快捷键。nnoremap <F2> I<Esc>

```
daisy@daisy-virtual-machine: ~
inoremap <Right> <ESC>:echoe "Use l"<CR>
inoremap <Up>    <ESC>:echoe "Use k"<CR>
inoremap <Down>  <ESC>:echoe "Use j"<CR>
nnoremap <F2> I#<Esc>

let g:ctrlp_map = '<c-p>'
let g:ctrlp_cmd = 'CtrlP'
-- 插入 -- 81,1 93%
```

图 2.11: 实例 4 设置快捷键

2. 实例 8

- 使用 qa 将宏操作录制到寄存器 a 中。

- 然后录制一个将改行注释掉的宏，并按`q`保存该宏。
- 按`v`，进入可视模式，然后按`j`选中所有行。

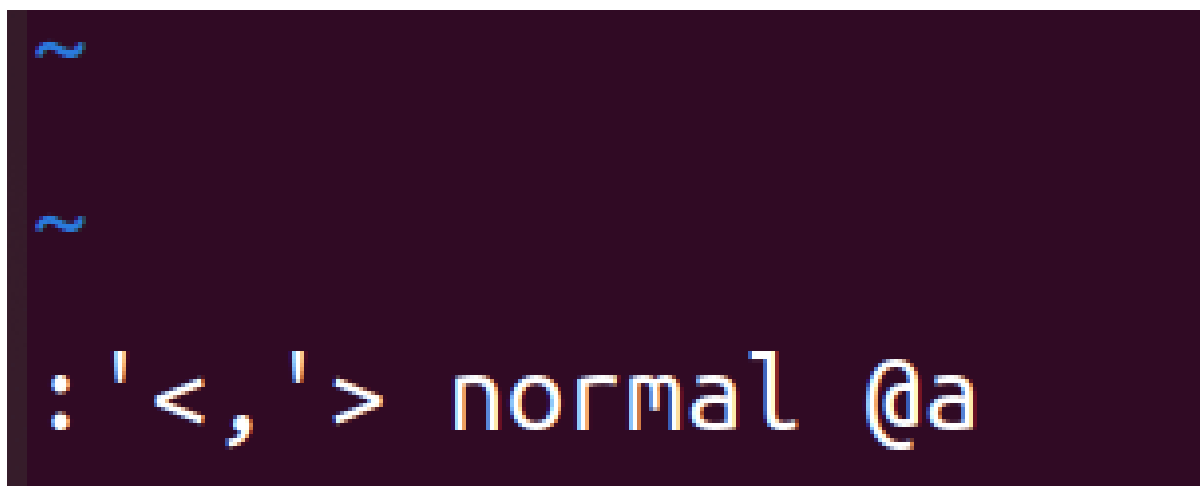


A terminal window with a dark purple background. The title bar shows 'daisy@daisy-virtual-machine: ~'. The prompt is '#!/bin/bash'. A bash script is being edited, and the entire content is selected in visual mode, indicated by a grey highlight. The script text is as follows:

```
#!/bin/bash  
  
for file in *.txt;  
do  
    if [ -e "$file" ]; then  
        mv "$file" "${file%.txt}.bak"  
    fi  
done
```

图 2.12: 可视模式下，选中所有行

- 按下`:`，会自动出现`'<,'> normal @a`回放宏。



The same terminal window as in Figure 2.12. The visual mode selection has been cleared. The prompt is now '~'. The user has entered the command `: '<,'> normal @a` to record a macro that will replay the previous visual selection.

图 2.13: 回放宏

实验部分 3 数据整理

3.1 实验内容

- 统计 words 文件 (/usr/share/dict/words) 中包含至少三个 a 且不以's 结尾的单词个数。
 - 这些单词中，出现频率前三的末尾两个字母是什么？sed 的 y 命令，或者 tr 程序也许可以帮你解决大小写的问题。
 - 共存在多少种词尾两字母组合？
- 进行原地替换听上去很有诱惑力，例如：sed s/REGEX/SUBSTITUTION/ input.txt > input.txt。但是这并不是一个明智的做法，为什么呢？还是说只有 sed 是这样的？查看 man sed 来完成这个问题
- 找出您最近十次开机的开机时间平均数、中位数和最长时间。在 Linux 上需要用到 journalctl，而在 macOS 上使用 log show。找到每次起到开始和结束时的时间戳。在 Linux 上类似这样操作：
Logs begin at ...
和
systemd[577]: Startup finished in ...

3.2 实验结果

- 实例 1**
 可以得到结果为**854**种。
 使用正则来找到这些单词中，出现频率前三的末尾两个字母是：**am、ce、ca**

```
daisy@daisy-virtual-machine:~$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^(^[a]*a){3}.*$" | grep -v "'s$" | wc -l
854
daisy@daisy-virtual-machine:~$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^(^[a]*a){3}.*$" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq -c | sort | tail -n3
      8 am
      8 ce
      9 ca
```

图 3.1: 实例 1(1) 结果图

共存在**112**种词尾两字母组合

```
daisy@daisy-virtual-machine:~$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^([a]*a){3}.*$" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq | wc -l
112
daisy@daisy-virtual-machine:~$
```

图 3.2: 实例 1(2) 结果图

共有564种组合未出现过

```
daisy@daisy-virtual-machine:~$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^([a]*a){3}.*$" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq > occurrence.txt
daisy@daisy-virtual-machine:~$ diff --unchanged-group-format='' <(cat occurrence.txt) <(cat char.txt) | wc -l
564
```

图 3.3: 实例 1(3) 结果图

2. 实例 2

不仅仅是 sed，对于任何命令都不应该在原地替换时使用重定向来覆盖原始文件。这是因为在大多数 Linux 系统中，重定向会在命令执行之前清空目标文件，这可能导致数据丢失。

当执行类似的命令时，操作系统会首先打开 input.txt 文件以进行读取，然后将其清空，然后再将 sed 命令的输出重定向到同一个文件。这会导致在 sed 命令执行之前就已经清空了文件，结果是文件内容丢失。

```
daisy@daisy-virtual-machine: ~
SED(1) User Commands SED(1)

NAME
    sed - stream editor for filtering and transforming text

SYNOPSIS
    sed [OPTION]... {script-only-if-no-other-script} [input-file]...

DESCRIPTION
    Sed is a stream editor. A stream editor is used to perform basic text
    transformations on an input stream (a file or input from a pipeline).
    While in some ways similar to an editor which permits scripted edits
    (such as ed), sed works by making only one pass over the input(s), and
    is consequently more efficient. But it is sed's ability to filter text
    in a pipeline which particularly distinguishes it from other types of
    editors.

Manual page sed(1) line 1 (press h for help or q to quit)
```

图 3.4: 实例 2:sed 的使用说明

3. 实例 3

最近十次开机

最长开机时间: 28.082

最短开机时间: 28.082

开机时间平均数: 24.0754

开机时间中位数: 26.6085

```
daisy@daisy-virtual-machine:~$ cat starttime.txt | grep "systemd\[1\]" | sed -E
"s/.*=\ (.*?)s\./\1/" | sort | tail -n1
28.082
daisy@daisy-virtual-machine:~$ cat starttime.txt | grep "systemd\[1\]" | sed -E
"s/.*=\ (.*?)s\./\1/" | sort -r | tail -n1
26.063
daisy@daisy-virtual-machine:~$ cat starttime.txt | grep "systemd\[1\]" | sed -E
"s/.*=\ (.*?)s\./\1/" | paste -sd+ | bc -l | awk '{print $1/10}'
24.0754
daisy@daisy-virtual-machine:~$ cat starttime.txt | grep "systemd\[1\]" | sed -E
"s/.*=\ (.*?)s\./\1/" | sort | paste -sd\ | awk '{print ($5+$6)/2}'
26.6085
```

图 3.5: 实例 3 结果图

3.3 实验代码及关键步骤

1. 实例 1

- 统计 words 文件中包含至少三个 a 且不以's 结尾的单词个数。使用

```
1 cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "
  ^([a]*a){3}.*$" | grep -v "'s$" | wc -l
```

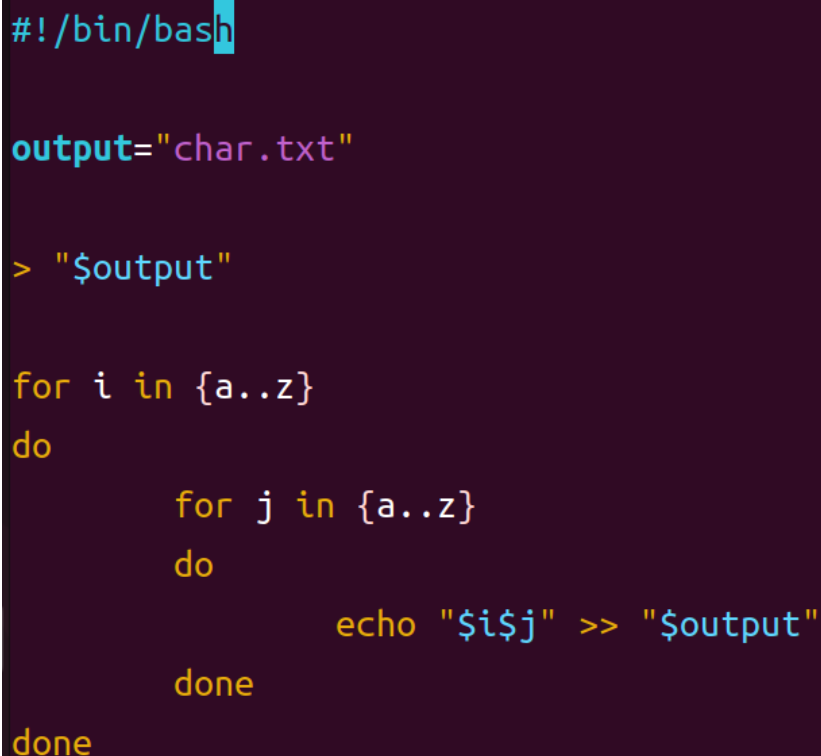
- 使用正则来找到这些单词中，出现频率前三的末尾两个字母那么得到它们是：am、ce、ca

```
1 cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "
  ^([a]*a){3}.*$" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/"
  | sort | uniq -c | sort | tail -n3
```

- 词尾两字母组合个数。

```
1 cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "
  ^([a]*a){3}.*$" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/"
  | sort | uniq | wc -l
```

- 先生成一个包含所有组合的列表char.txt。



```
#!/bin/bash

output="char.txt"

> "$output"

for i in {a..z}
do
    for j in {a..z}
    do
        echo "$i$j" >> "$output"
    done
done
```

图 3.6: 生成 char.txt 的脚本 char.sh

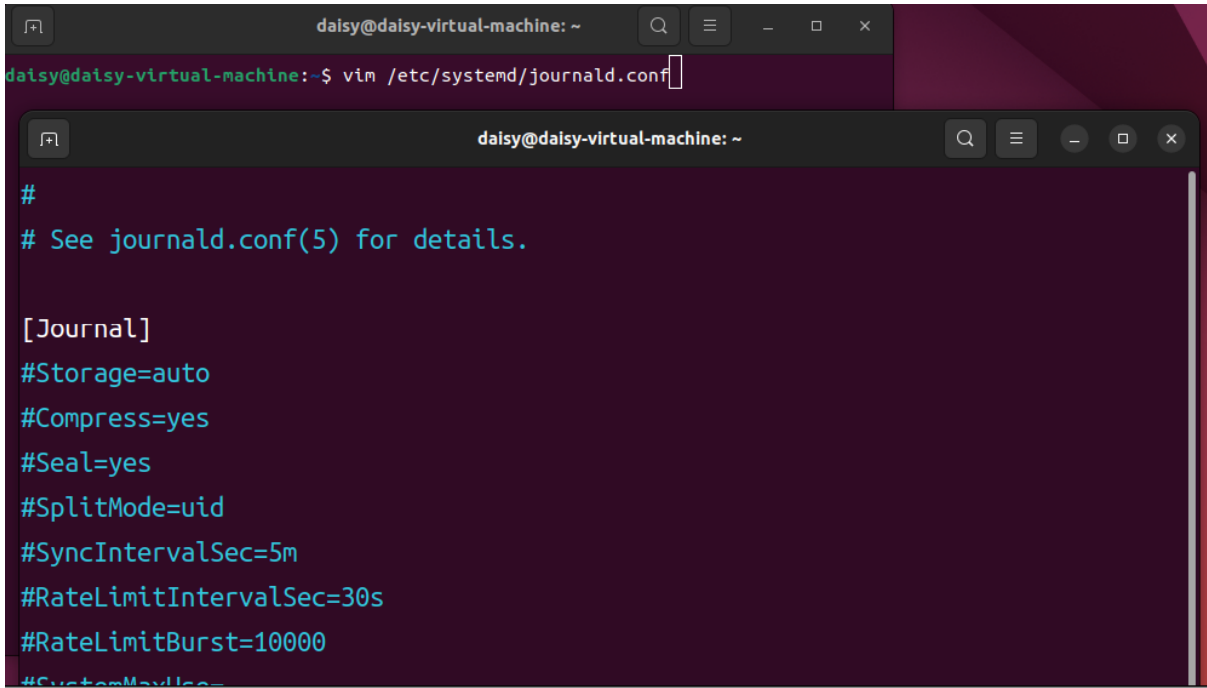
- 然后使用上面得到的出现的 112 种组合，比较二者不同。

```
1 cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "
  ^([a]*a){3}.*$" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$
  /\1/" | sort | uniq > occurrence.txt
```

```
1 diff --unchanged-group-format='' <(cat occurrence.txt) <(cat all.
  txt) | wc -l
```

2. 实例 3

- 首先需要首先允许 journalctl 记录多次开机的日志，否则我们看到的始终都只有本次启动的日志。

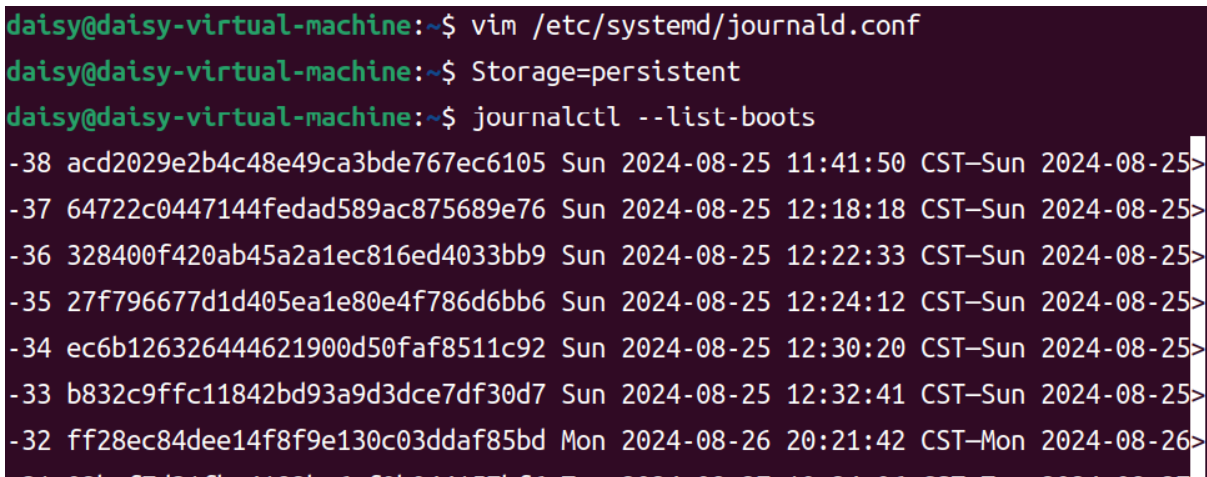


```
daisy@daisy-virtual-machine: ~  
daisy@daisy-virtual-machine:~$ vim /etc/systemd/journald.conf  
#  
# See journald.conf(5) for details.  
  
[Journal]  
#Storage=auto  
#Compress=yes  
#Seal=yes  
#SplitMode=uid  
#SyncIntervalSec=5m  
#RateLimitIntervalSec=30s  
#RateLimitBurst=10000  
#SystemMaxUse=
```

图 3.7: 开机日志

- 设置 Storage=persistent 执行上述命令后，重启

```
1 journalctl --list-boots
```



```
daisy@daisy-virtual-machine:~$ vim /etc/systemd/journald.conf  
daisy@daisy-virtual-machine:~$ Storage=persistent  
daisy@daisy-virtual-machine:~$ journalctl --list-boots  
-38 acd2029e2b4c48e49ca3bde767ec6105 Sun 2024-08-25 11:41:50 CST-Sun 2024-08-25>  
-37 64722c0447144fedad589ac875689e76 Sun 2024-08-25 12:18:18 CST-Sun 2024-08-25>  
-36 328400f420ab45a2a1ec816ed4033bb9 Sun 2024-08-25 12:22:33 CST-Sun 2024-08-25>  
-35 27f796677d1d405ea1e80e4f786d6bb6 Sun 2024-08-25 12:24:12 CST-Sun 2024-08-25>  
-34 ec6b126326444621900d50faf8511c92 Sun 2024-08-25 12:30:20 CST-Sun 2024-08-25>  
-33 b832c9ffc11842bd93a9d3dce7df30d7 Sun 2024-08-25 12:32:41 CST-Sun 2024-08-25>  
-32 ff28ec84dee14f8f9e130c03ddaf85bd Mon 2024-08-26 20:21:42 CST-Mon 2024-08-26>
```

图 3.8: 重启信息

- 使用 systemd-analyze 工具看一下启动时间都花在哪里:

```
1 systemd-analyze  
2 systemd-analyze blame
```

```
daisy@daisy-virtual-machine:~$ systemd-analyze
Startup finished in 3.918s (kernel) + 23.170s (userspace) = 27.088s
graphical.target reached after 23.137s in userspace
daisy@daisy-virtual-machine:~$ systemd-analyze blame
47.718s apt-daily-upgrade.service
20.710s plymouth-quit-wait.service
 1.202s dev-sda3.device
 1.190s dev-loop9.device
 1.190s dev-loop8.device
 1.189s dev-loop11.device
 1.189s dev-loop10.device
 1.189s dev-loop12.device
```

图 3.9: 启动时间

- 接下来，编写脚本 `getlog.sh` 来获取最近十次的启动时间数据并输入到 `starttime.txt`

```
#!/bin/bash

for i in {0..9}
do
    journalctl -b-$i | grep "Startup finished in"
done
```

图 3.10: `getlog.sh` 脚本

- 得到结论
最长开机时间: 28.082
最短开机时间: 28.082
开机时间平均数: 24.0754
开机时间中位数: 26.6085

```
daisy@daisy-virtual-machine:~$ cat starttime.txt | grep "systemd\[1\]" | sed -E
"s/.*=\ (.*?)s\.$/\1/" | sort | tail -n1
28.082
daisy@daisy-virtual-machine:~$ cat starttime.txt | grep "systemd\[1\]" | sed -E
"s/.*=\ (.*?)s\.$/\1/" | sort -r | tail -n1
26.063
daisy@daisy-virtual-machine:~$ cat starttime.txt | grep "systemd\[1\]" | sed -E
"s/.*=\ (.*?)s\.$/\1/" | paste -sd+ | bc -l | awk '{print $1/10}'
24.0754
daisy@daisy-virtual-machine:~$ cat starttime.txt | grep "systemd\[1\]" | sed -E
"s/.*=\ (.*?)s\.$/\1/" | sort | paste -sd\ | awk '{print ($5+$6)/2}'
26.6085
```

图 3.11: 开机时间数据

3.4 实验中遇到的问题

3.4.1 开机时间计算错误

1. 最后的实验，**开机最短时间**大于**开机时间平均值**。
2. 可能是**使用的数据不同**，但是我通过命令只选择starttime.txt文件中最近十次的开机时间。目前不知道如何解决 TT。

实验部分 4 实验心得

这一星期的学习主要集中在三个方面：Shell 脚本编写、Vim 的定制化操作以及数据的查找与整理。通过这些操作，我深刻体会到了 Unix/Linux 环境下的高效工作方式，也加深了对工具与编程语言灵活运用

4.0.1 Shell 脚本编写

Shell 脚本作为自动化的利器，能够大幅度减少重复劳动，提升工作效率。本周通过练习，我掌握了如何在脚本中循环执行命令、捕获错误信息并生成报告。例如，利用 `find` 命令递归查找文件、结合 `grep` 过滤文件内容等操作，帮助我在数据整理中快速获取目标文件。Shell 的强大在于它能结合各种命令进行批量处理，大大减少了手动操作的成本。

4.0.2 Vim 的定制化操作

Vim 是一款功能强大的文本编辑器，尤其适合在命令行环境下工作。通过配置 `.vimrc` 文件，定义快捷键、自定义补全、自动扩展代码片段等操作，我发现了 Vim 的无限可能。配置 Vim 让我意识到，定制化操作不仅能提升工作效率，还能让开发体验更加顺畅。

4.0.3 数据查找与整理

无论是使用 `grep` 进行文本查找，还是使用 `find` 查找特定类型的文件，这些命令工具能够在海量数据中快速定位目标。我特别体验到，结合 Shell 和 Vim 进行数据处理，能够使复杂的操作变得更加简洁。比如，Vim 的可视模式可以选中并快速替换文本，而 Shell 则能够批量处理和操作文件内容。

附录 1

我的 Github 仓库网址

我的 Github 仓库 [需下载 pdf 点击]

或者直接通过地址寻找<https://github.com/ChenFirstIce/classwork>