

# 实验二：Shell 工具和脚本、编辑器（Vim）、数据整理学习实验报告

邓林 23020007014

中国海洋大学 23软件工程

## 摘要

本实验报告主要记录了作者通过课程网站及B站学习Shell 工具和脚本、编辑器（Vim）、数据整理的学习过程以及心得。

## 1 实验内容

主要学习了1.Shell语法和简单的使用；2.vim编辑器的使用方法以及光标移动快捷键等；3.数据整理及正则表达式。

Shell中的特殊符号	
lt(less than)	小于
le(less than or equal to)	小于等于
gt(greater than)	大于
ge(greater than or equal to)	大于等于
eq(equal to)	等于
ne(not equal to)	不等于
#	注释
\$#	传递给脚本或函数的位置参数的个数
\$?	上一命令的退出状态码。0通常表示没有错误，非0值表示有错误
\$*	传递给脚本或函数的位置参数，双引号包围时作为一个整体
\$@	传递给脚本或函数的位置参数
\$\$	当前Shell进程的进程ID (PID)
\$_	最后一个后台命令的进程ID
\$0	当前脚本的名称
\$1-n	脚本或函数的位置参数

编辑器(vim)使用指令		
模式选择	vim 文件名	编辑文件
	i	编辑模式
	esc	正常模式
	:	命令行模式
	q	仅退出
	wq	保存并退出
	q!	不保存退出
导航与编辑	h j k l	左下上右
	i	插前
	a	附后
	o	新增下一行
	I/shift+i	移到本行最前
	A/shift+a	移到本行最后
	O/shift+o	新增上一行
导航与编辑	G	到最后一行
	gg	到第一行
	yy	复制当前行
	dd	删除当前行
	.	重复前次操作
	u	撤销前此操作
	ctrl+r	恢复前次操作
	dw	删除单词
	cw	改变单词
	w	下个单词尾部
替换与视觉模式	/	搜索
	:%s/旧/新/g	全局替换
	yw	复制单词
	p	粘贴
	ci{}	删除内容
	ctrl+v	可视化块

Shell语法	
if [[ 条件 ]]; then	if 语句
else if [[ 条件 ]]; then	
else	
fi	
for (( 条件 ))	for 循环
while ture	while 循环
do	
done	
export	配置临时环境变量
name=名字	变量赋值
read	读入变量
echo	输出语句

正则表达式的功能元素		
基本字符	字母和数字	直接表示它们自身
	.	匹配任何单个字符（通常不包括换行符）
量词	*	表示匹配前面的表达式零次或多次
	+	表示匹配前面的表达式一次或多次
	?	表示匹配前面的表达式零次或一次
	{n}	表示前面的表达式恰好出现 n 次
	{n,}	表示前面的表达式至少出现 n 次
	{n,m}	表示前面的表达式出现 n 到 m 次
组合与引用	()	用于创建捕获组，可以对匹配的部分进行引用
	(?:)	非捕获组，不会保存匹配结果，用于组合
	\1, \2, ...	用于引用前面的捕获组
字符类	[abc]	匹配 a 或 b 或 c
	[a-z]	匹配 a 到 z 之间的任何字符
	[^abc]	匹配除了 a、b 和 c 之外的任何字符
	\d	匹配任何数字
	\D	匹配任何非数字
	\w	匹配任何字母数字字符（相当于 [a-zA-Z0-9_]）

Continued on next page

(Continued)

	\W	匹配任何非字母数字字符
	\s	匹配任何空白字符
	\S	匹配任何非空白字符
断言	^	表示匹配字符串的开始
	\$	表示匹配字符串的结尾
	(?=...)	正向前瞻断言，匹配后面跟着指定模式的位置
	(?!...)	负向前瞻断言，匹配后面不跟指定模式的位置
	(?=...)	正向后瞻断言，匹配前面有指定模式的位置
	(?!...)	负向后瞻断言，匹配前面没有指定模式的位置

2 课后习题

2.1 Shell工具和脚本

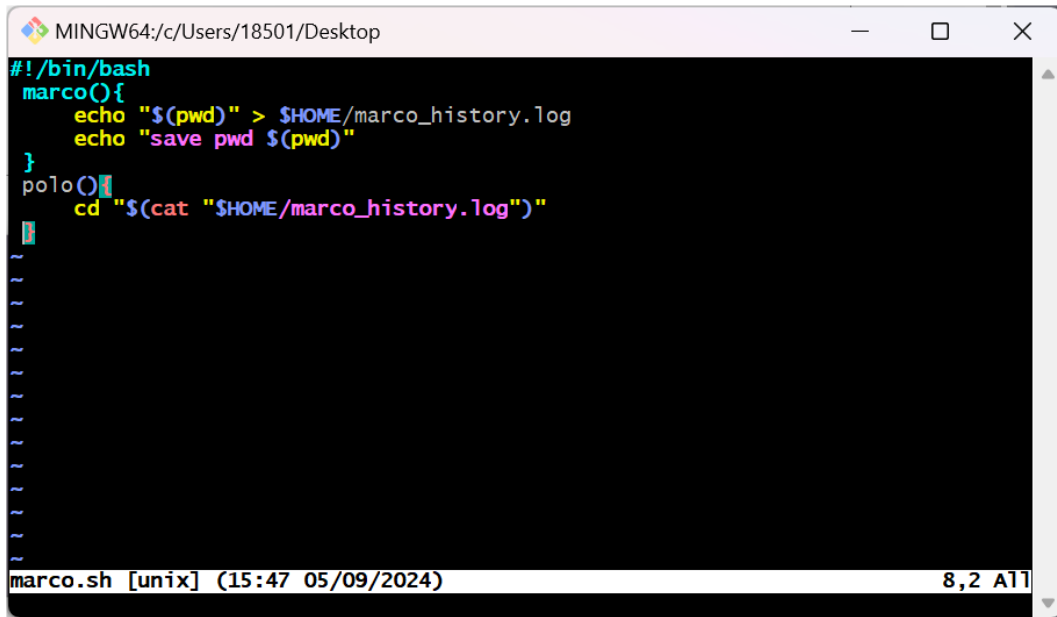
1. 阅读 man ls，然后使用 ls 命令进行操作

```
ls501@: I MINGW64 ~/Desktop
$ ls -alh --color=auto --time=atime
total 196M
drwxr-xr-x 1 18501 197609 0 Sep 5 15:33 ./
drwxr-xr-x 1 18501 197609 0 Sep 5 15:33 ../
-rw-r--r-- 1 18501 197609 33K Sep 5 00:20 14.软件工程.xlsx
drwxr-xr-x 1 18501 197609 0 Sep 5 13:31 2023-2024综测材料/
drwxr-xr-x 1 18501 197609 0 Sep 5 13:30 23计算机4班/
-rwxr-xr-x 1 18501 197609 1.1K Sep 5 15:04 Dev-C++.lnk*
-rwxr-xr-x 1 18501 197609 2.4K Sep 5 15:04 'GitHub Desktop.lnk'*
-rwxr-xr-x 1 18501 197609 2.4K Sep 5 15:04 'Google Chrome.lnk'*
-rw-r--r-- 1 18501 197609 16K Sep 5 13:36 Latex指令.docx
drwxr-xr-x 1 18501 197609 0 Sep 5 12:14 MUD-game/
-rwxr-xr-x 1 18501 197609 2.1K Sep 5 15:04 MuMu多开器12.lnk*
-rwxr-xr-x 1 18501 197609 2.1K Sep 5 15:04 MuMu模拟器12.lnk*
drwxr-xr-x 1 18501 197609 0 Sep 5 12:14 Ncm转mp3拖一拖/
-rwxr-xr-x 1 18501 197609 2.9K Sep 5 15:04 PPT超级市场.lnk*
drwxr-xr-x 1 18501 197609 0 Sep 5 12:14 SimpleMUD/
-rwxr-xr-x 1 18501 197609 2.1K Sep 5 15:04 SumatraPDF.lnk*
-rwxr-xr-x 1 18501 197609 1.8K Sep 5 15:04 'Visual Studio 2022.lnk'*
-rwxr-xr-x 1 18501 197609 1.4K Sep 5 15:04 'Visual Studio Code.lnk'*
-rwxr-xr-x 1 18501 197609 186M Sep 5 12:09 'aTrustInstaller[https@otrust.ouc.edu.cn@443].exe'*
-rw-r--r-- 1 18501 197609 282 Sep 5 15:30 desktop.ini
-rw-r--r-- 1 18501 197609 6 Sep 5 14:05 hello.txt
-rw-r--r-- 1 18501 197609 6 Sep 5 14:05 hello2.txt
drwxr-xr-x 1 18501 197609 0 Sep 5 12:14 missing-semester-cn.github.io/
-rw-r--r-- 1 18501 197609 162 Sep 5 13:36 '~$latex指令.docx'
-rw-r--r-- 1 18501 197609 49K Sep 3 16:08 【2023秋】计算机类平均学绩.xlsx
drwxr-xr-x 1 18501 197609 0 Sep 5 12:14 参考书籍/
drwxr-xr-x 1 18501 197609 0 Sep 5 12:14 国赛模板/
-rwxr-xr-x 1 18501 197609 1.4K Sep 5 15:04 夸克网盘.lnk*
drwxr-xr-x 1 18501 197609 0 Sep 5 13:31 我的/
-rwxr-xr-x 1 18501 197609 794 Sep 5 15:04 沙漏验机.lnk*
-rwxr-xr-x 1 18501 197609 2.1K Sep 5 15:04 爱奇艺.lnk*
-rwxr-xr-x 1 18501 197609 1.1K Sep 5 15:04 百度网盘.lnk*
-rw-r--r-- 1 18501 197609 836K Sep 3 17:39 程序设计基础实践-实验大纲-2024年夏.pdf
-rw-r--r-- 1 18501 197609 1.8M Sep 4 15:21 程序设计基础实践-实验要求-2024年夏.pptx
-rw-r--r-- 1 18501 197609 6.6M Sep 5 12:11 '系统开发工具基础(1).pdf'
-rw-r--r-- 1 18501 197609 603K Sep 4 09:44 系统开发工具基础_实验一.docx
-rw-r--r-- 1 18501 197609 222 Sep 5 15:04 饥荒联机版.url
```

图 1: 第一题

2. 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。



```
#!/bin/bash
marco(){
    echo "$(pwd)" > $HOME/marco_history.log
    echo "save pwd $(pwd)"
}
polo(){
    cd "$(cat "$HOME/marco_history.log")"
}

marco.sh [unix] (15:47 05/09/2024) 8,2 All
```

图 2: 编辑marco.sh文件中内容

```

18501@MINGW64 ~/Desktop
$ vim marco.sh

18501@MINGW64 ~/Desktop
$ ls
14. 软件工程.xlsx
2023-2024综测材料/
23计算机4班/
Dev-C++.lnk*
'GitHub Desktop.lnk'*
'Google Chrome.lnk'*
Latex指令.docx
MUD-game/
MuMu多开器12.lnk*
MuMu模拟器12.lnk*
Ncm转mp3拖一拖/
PPT超级市场.lnk*
SimpleMUD/
SumatraPDF.lnk*
'Visual Studio 2022.lnk'*
'Visual Studio Code.lnk'*
'aTrustInstaller[https@otrust.ouc.edu.cn@443].exe'*
desktop.ini
hello.txt
hello2.txt
marco.sh*
missing-semester-cn.github.io/
~$atex指令.docx'
【2023秋】计算机类平均学分绩.xlsx
参考书籍/
国赛模板/
夸克网盘.lnk*
我的/
沙漏验机.lnk*
爱奇艺.lnk*
百度网盘.lnk*
程序设计基础实践-实验大纲-2024年夏.pdf
程序设计基础实践-实验要求-2024年夏.pptx
'系统开发工具基础(1).pdf'
系统开发工具基础_实验一.docx
饥荒联机版.url

18501@MINGW64 ~/Desktop
$ source marco.sh

18501@MINGW64 ~/Desktop
$ marco
save pwd /c/Users/18501/Desktop

18501@MINGW64 ~/Desktop
$ cd

18501@MINGW64 ~
$ po1o

18501@MINGW64 ~/Desktop
$

```

图 3: 测试代码, 成功运行

3. 编写一段bash脚本, 运行脚本直到它出错, 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。

首先, 创建一个test.sh文件编写以下脚本:

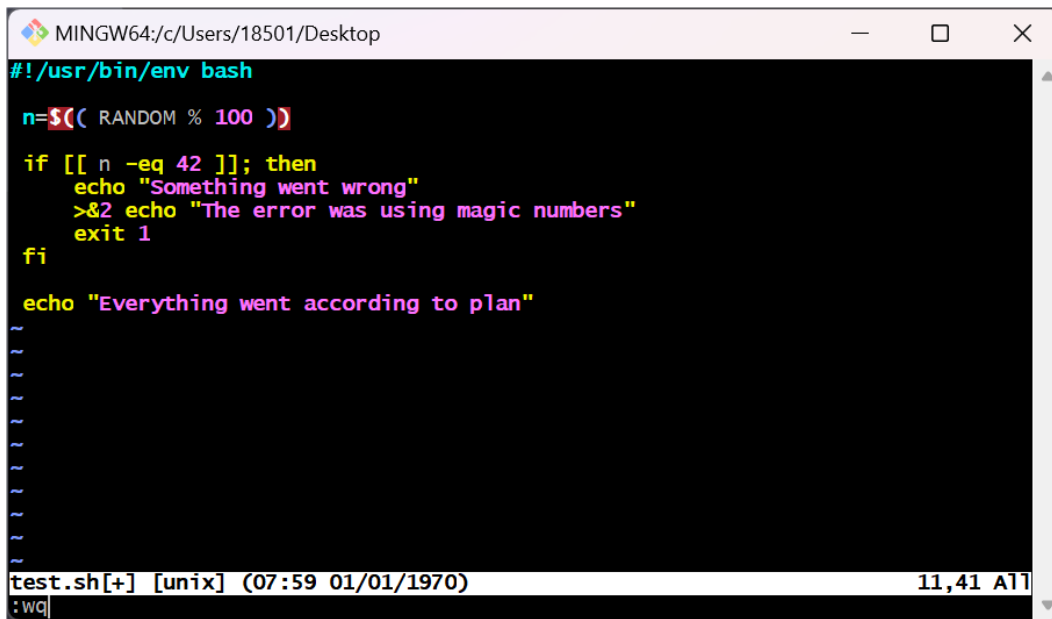


图 4: 编写test.sh脚本

然后，编写bash脚本，使用while循环完成：

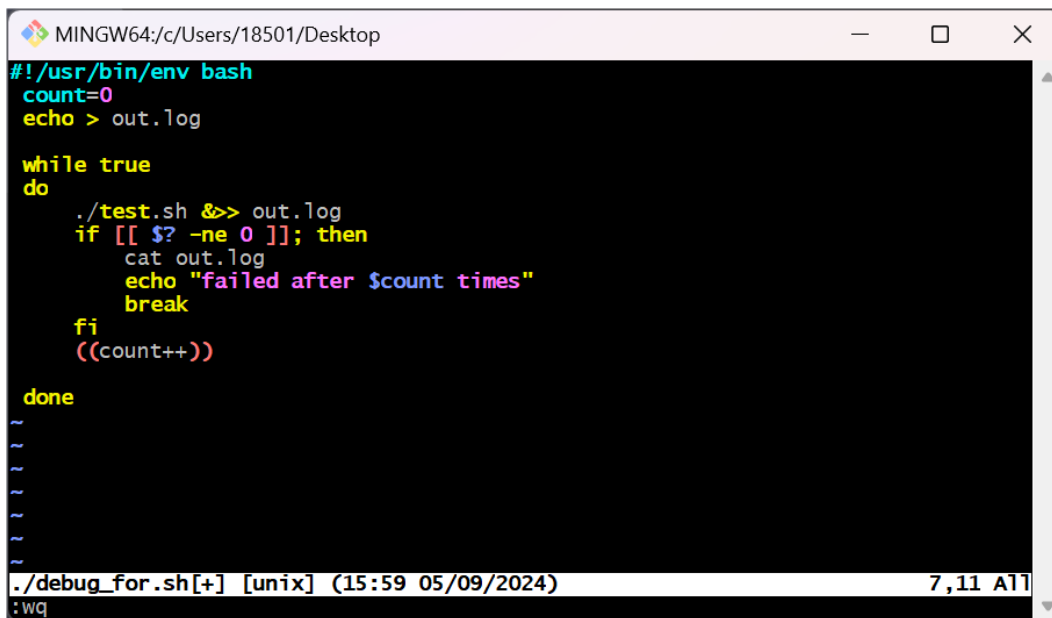


图 5: 编写debug\_for.sh脚本

最后，执行测试脚本`debug_for.sh`,并验证脚本结果的正确性：

```
18501@msys64: C:\Users\user\I-4 MINGW64 ~/Desktop
$ vim test.sh

18501@msys64: C:\Users\user\I-4 MINGW64 ~/Desktop
$ vim ./debug_for.sh

18501@msys64: C:\Users\user\I-4 MINGW64 ~/Desktop
$ AC

18501@msys64: C:\Users\user\I-4 MINGW64 ~/Desktop
$ ./debug_for.sh
```

[illegible]

图 6: 执行脚本

```
18501@C:\Users\18501\OneDrive\Desktop MINGW64 ~/Desktop
$ cat out.log | grep Everything | wc -l
89
```

图 7: 执行脚本



## 2.2 数据整理

1. 统计word文件中包含至少三个a 且不以's 结尾的单词个数  
大小写转换:

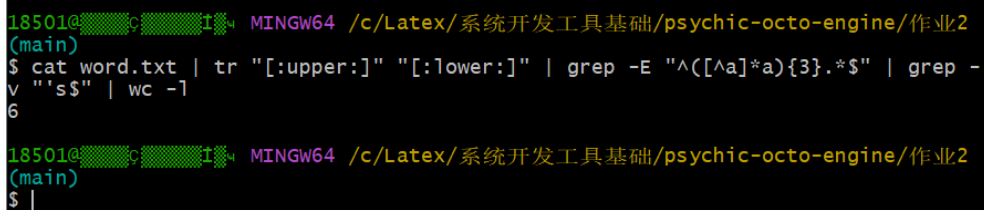
```
tr "[:upper:]" "[:lower:]"
```

查找一个以 a 结尾的字符串三次:

```
^([\^a]a){3}.\[^'s]$
```

匹配结尾为's 的结果, 然后取反:

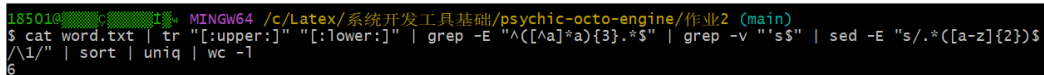
```
grep -v "'s$"
```



```
18501@MINGW64 /c/Latex/系统开发工具基础/psychic-octo-engine/作业2 (main)
$ cat word.txt | tr "[:upper:]" "[:lower:]" | grep -E "^[^a]*a{3}.*$" | grep -v "'s$" | wc -l
6
18501@MINGW64 /c/Latex/系统开发工具基础/psychic-octo-engine/作业2 (main)
$ |
```

图 8: 至少三个a 且不以's 结尾的单词个数

2. 统计word文件中是否存在多少种词尾两字母组合



```
18501@MINGW64 /c/Latex/系统开发工具基础/psychic-octo-engine/作业2 (main)
$ cat word.txt | tr "[:upper:]" "[:lower:]" | grep -E "^[^a]*a{3}.*$" | grep -v "'s$" | sed -E 's/.*([a-z]{2})$/\1/' | sort | uniq | wc -l
6
```

图 9: 存在多少种词尾两字母组合

3. 备份文件, 自动创建一个后缀为 .bak 的备份文件



```
18501@MINGW64 /c/Latex/系统开发工具基础/psychic-octo-engine/作业2 (main)
$ sed -i.bak 's/REGEX/SUBSTITUTION/g' word.txt
```

图 10: 备份文件

## 3 练习实例

### 3.1 Shell工具和脚本

1. 初步使用Shell

```
18501@C:\Users\18501: MINGW64 ~/Desktop
$ date
Thu Sep  5 13:42:13 2024

18501@C:\Users\18501: MINGW64 ~/Desktop
$ echo hello
hello

18501@C:\Users\18501: MINGW64 ~/Desktop
$ echo $PATH
/c/Users/18501/bin:/mingw64/bin:/usr/local/bin:/usr/bin:/bin:/mingw64/bin:/usr/bin:/c/Users/18501/bin:/c/Program Files (x86)/VMware/VMware Player/bin:/c/windows/system32:/c/windows:/c/windows/System32/Wbem:/c/windows/System32/WindowsPowerShell/v1.0:/c/windows/System32/OpenSSH:/c/Program Files (x86)/NVIDIA Corporation/PhysX/Common:/c/Program Files/NVIDIA Corporation/NVIDIA NvDLISR:/c/Users/Administrator/AppData/Local/Microsoft/WindowsApps:/c/Program Files/HP/OMEN-Broadcast/Common:/c/Program Files (x86)/Windows Kits/10/Windows Performance Toolkit:/c/Strawberry/bin:/c/Strawberry/perl/site/bin:/c/Strawberry/perl/bin:/cmd:/c/texlive/2024/bin/windows:/c/Users/18501/AppData/Local/Microsoft/WindowsApps:/c/Program Files (x86)/Tencent/QQGameTempest/Hall.57986:/c/Users/18501/AppData/Local/GitHubDesktop/bin:/c/Users/18501/AppData/Local/Programs/Microsoft VS Code/bin:/c/texlive/2024/bin/windows:/usr/bin/vendor_perl:/usr/bin/core_perl

18501@C:\Users\18501: MINGW64 ~/Desktop
$ which echo
/usr/bin/echo

18501@C:\Users\18501: MINGW64 ~/Desktop
$ /usr/bin/echo $PATH
/c/Users/18501/bin:/mingw64/bin:/usr/local/bin:/usr/bin:/bin:/mingw64/bin:/usr/bin:/c/Users/18501/bin:/c/Program Files (x86)/VMware/VMware Player/bin:/c/windows/system32:/c/windows:/c/windows/System32/Wbem:/c/windows/System32/WindowsPowerShell/v1.0:/c/windows/System32/OpenSSH:/c/Program Files (x86)/NVIDIA Corporation/PhysX/Common:/c/Program Files/NVIDIA Corporation/NVIDIA NvDLISR:/c/Users/Administrator/AppData/Local/Microsoft/WindowsApps:/c/Program Files/HP/OMEN-Broadcast/Common:/c/Program Files (x86)/Windows Kits/10/Windows Performance Toolkit:/c/Strawberry/bin:/c/Strawberry/perl/site/bin:/c/Strawberry/perl/bin:/cmd:/c/texlive/2024/bin/windows:/c/Users/18501/AppData/Local/Microsoft/WindowsApps:/c/Program Files (x86)/Tencent/QQGameTempest/Hall.57986:/c/Users/18501/AppData/Local/GitHubDesktop/bin:/c/Users/18501/AppData/Local/Programs/Microsoft VS Code/bin:/c/texlive/2024/bin/windows:/usr/bin/vendor_perl:/usr/bin/core_perl

18501@C:\Users\18501: MINGW64 ~/Desktop
$
```

图 11: 初步使用Shell

## 2. pwd: 获取当前工作目录

```
18501@C:\Users\18501: MINGW64 ~/Desktop
$ pwd
/c/Users/18501/Desktop

18501@C:\Users\18501: MINGW64 ~/Desktop
$ cd /c

18501@C:\Users\18501: MINGW64 /c
$ pwd
/c

18501@C:\Users\18501: MINGW64 /c
$ cd ..

18501@C:\Users\18501: MINGW64 /
$ cd ./c

18501@C:\Users\18501: MINGW64 /c
$ cd Desktop
bash: cd: Desktop: No such file or directory

18501@C:\Users\18501: MINGW64 /c
$ cd Users

18501@C:\Users\18501: MINGW64 /c/Users
$ pwd
/c/Users

18501@C:\Users\18501: MINGW64 /c/Users
$ ../../bin/echo hello
hello

18501@C:\Users\18501: MINGW64 /c/Users
$ |
```

图 12: pwd学习与使用

## 3. 在程序间创建连接

```
18501@MINGW64 ~/Desktop
$ echo hello > hello.txt

18501@MINGW64 ~/Desktop
$ cat hello.txt
hello

18501@MINGW64 ~/Desktop
$ cat < hello.txt
hello

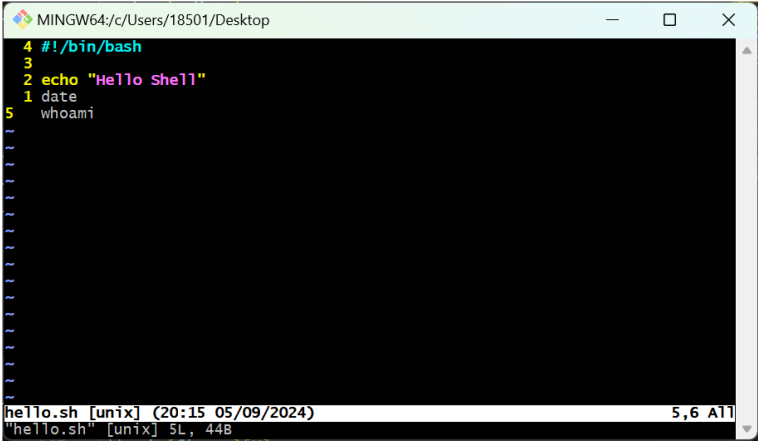
18501@MINGW64 ~/Desktop
$ cat <hello.txt > hello2.txt

18501@MINGW64 ~/Desktop
$ cat hello2.txt
hello

18501@MINGW64 ~/Desktop
$ |
```

图 13:

4. Shell写一个简单代码

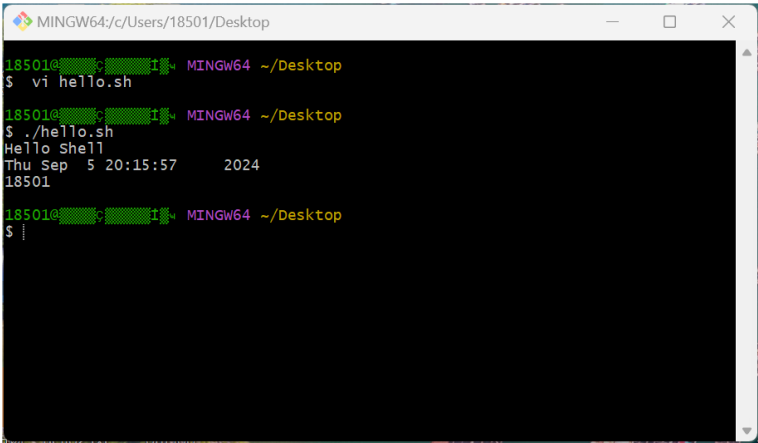


The screenshot shows a text editor window titled "MINGW64:/c/Users/18501/Desktop". The editor contains a shell script with the following content:

```
4 #!/bin/bash
3
2 echo "Hello Shell"
1 date
5 whoami
```

The status bar at the bottom indicates the file is "hello.sh [unix] (20:15 05/09/2024)" and is 5,6 A11 characters long.

图 14: 编辑代码



The screenshot shows a terminal window titled "MINGW64:/c/Users/18501/Desktop". The user has executed the following commands:

```
18501@MINGW64 ~/Desktop
$ vi hello.sh

18501@MINGW64 ~/Desktop
$ ./hello.sh
Hello Shell
Thu Sep 5 20:15:57 2024
18501
```

The output of the script is displayed in the terminal.

图 15: 运行效果

5. Shell写一个猜数字游戏

```

MINGW64:/c/Latex/系统开发工具基础/psychic-octo-engine/作业2
26 #!/bin/bash
25
24 name=嘻嘻要哈哈
23 channel=猜数字
22 echo "您好, $name, 欢迎来到$channel!"
21 #number=$((RANDOM % 10 + 1))
20 #echo $number
19 while true
18 do
17 number=$((RANDOM % 10 + 1))
16 #echo $number
15 echo "请输入一个1-10之间的数字"
14 read guess
13 if [[ $guess -eq $number ]]; then
12     echo "恭喜你猜对了! 是否继续? (y/n) : "
11     read choice
10     if [[ $choice = "y" ]] || [[ $choice = "Y" ]]; then
9         continue
8     else
7         break
6     fi
5 elif [[ $guess -lt $number ]]; then
4     echo "小了"
3 else
2     echo "大了"
1 fi
27 done
~
~

```

图 16: 编辑代码

```

18501@MINGW64: c:\mingw64\ I MINGW64 /c/Latex/系统开发工具基础/psychic-octo-engine/作业2
(main)
$ vim game.sh

18501@MINGW64: c:\mingw64\ I MINGW64 /c/Latex/系统开发工具基础/psychic-octo-engine/作业2
(main)
$ ./game.sh
您好, 嘻嘻要哈哈, 欢迎来到猜数字!
请输入一个1-10之间的数字
5
小了
请输入一个1-10之间的数字
6
小了
请输入一个1-10之间的数字
7
大了
请输入一个1-10之间的数字
8
大了
请输入一个1-10之间的数字
9
大了
请输入一个1-10之间的数字
恭喜你猜对了! 是否继续? (y/n) :
y
请输入一个1-10之间的数字
5
大了
请输入一个1-10之间的数字
6
大了
请输入一个1-10之间的数字
7
恭喜你猜对了! 是否继续? (y/n) :
n
18501@MINGW64: c:\mingw64\ I MINGW64 /c/Latex/系统开发工具基础/psychic-octo-engine/作业2 (main)
$ |

```

图 17: 运行效果

## 3.2 编辑器 (Vim)

### 1. 显示编辑器行号与相对行号

首先, 检查vim版本

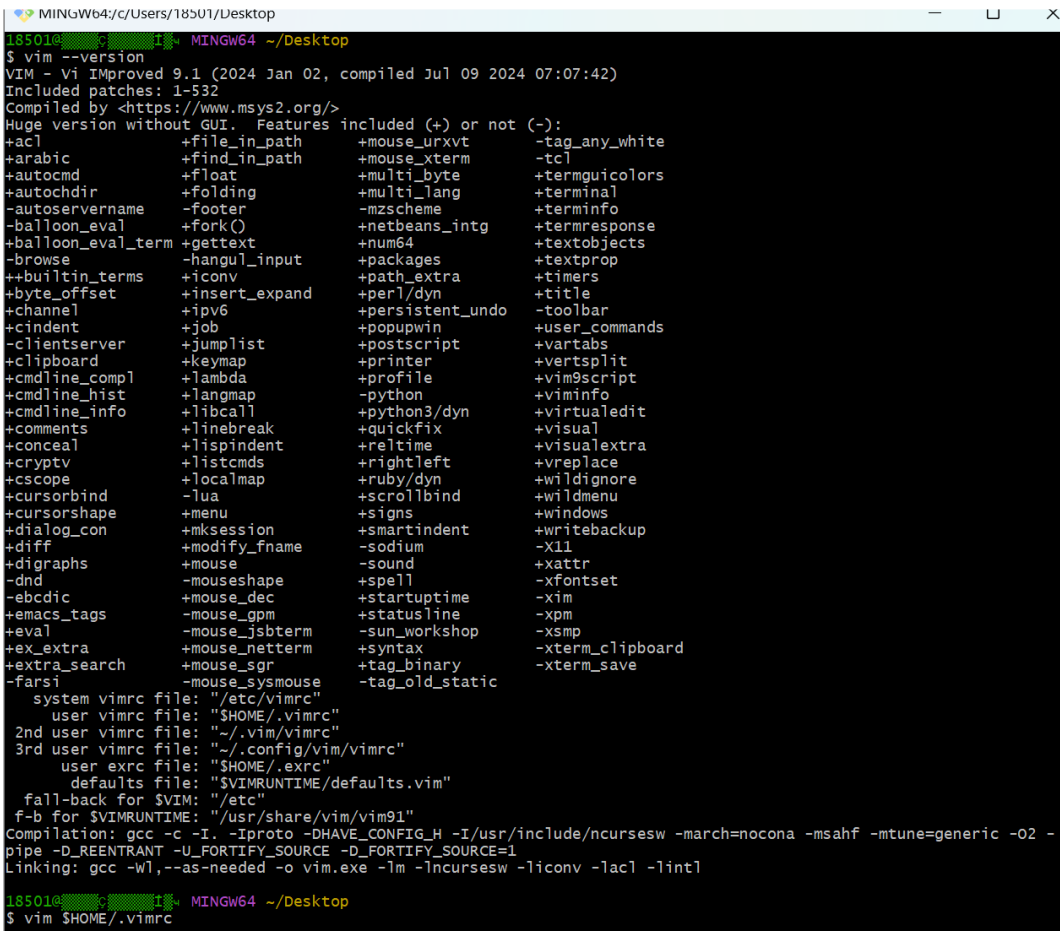


图 18: 检查vim版本

接着，找到并打开 \$HOME/.vimrc，添加如下代码：

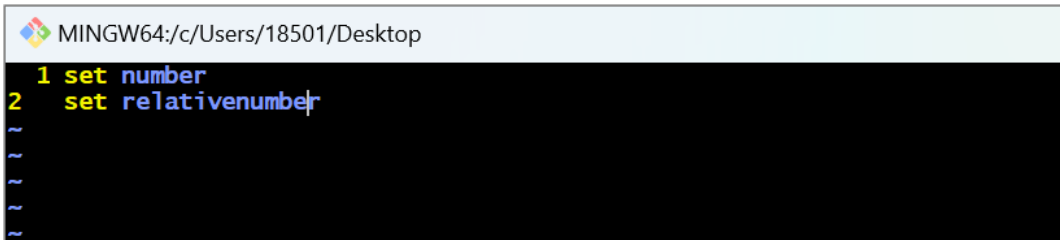


图 19: 显示行号，相对行号的代码

效果如下：

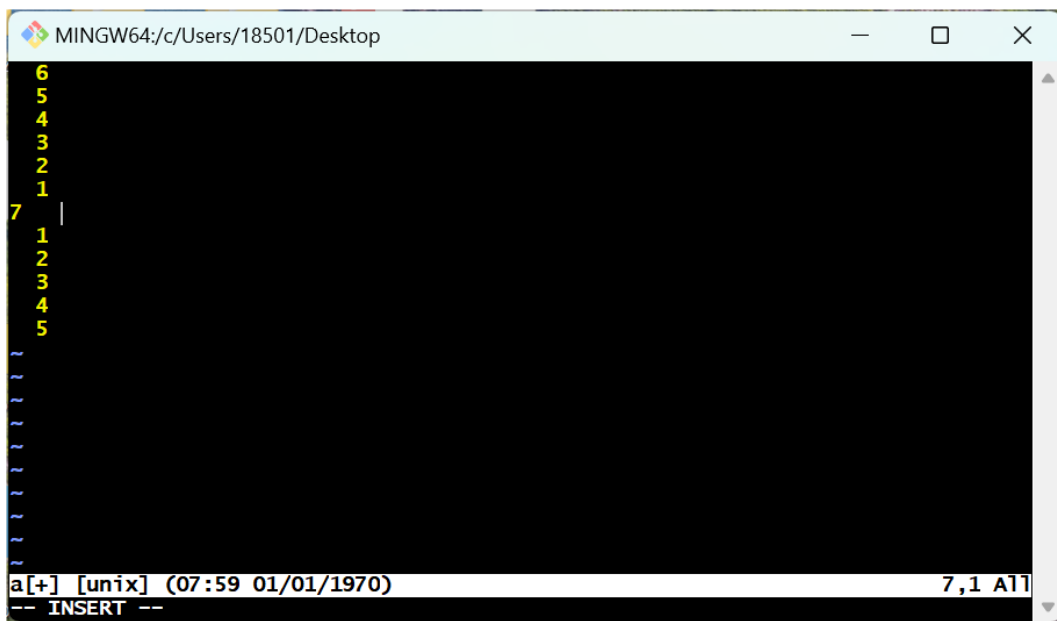


图 20: 效果如图

### 3.3 数据整理

一些由正则表达式的基本元素组合而成的例子

#### 1. 匹配电子邮件地址

$\text{^[a-zA-Z0-9. \_ \% + - ] + @ [a-zA-Z0-9. - ] + \. [a-zA-Z] \{ 2 , \} \$}$

#### 2. 匹配一个美国格式的电话号码（区号-三位数-四位数）

$\text{^[a-zA-Z0-9. \_ \% + - ] + @ [a-zA-Z0-9. - ] + \. [a-zA-Z] \{ 2 , \} \$}$

#### 3. 匹配电子邮件地址

$\text{^\d{3} - \d{3} - \d{4} \$}$

#### 4. 匹配URL

$\text{^ ( http | https ) : / / [a-zA-Z0-9. - ] + ( : \d + ) ? ( / [a-zA-Z0-9. \_ \% \& = - ] * ) ? \$}$

#### 5. 匹配中文字符

$\text{^\ [ \ u4e00 - \ u9fa5 ] + \$}$

#### 6. 匹配一个 YYYY-MM-DD 格式的日期

$\text{^\d{4} - \d{2} - \d{2} \$}$

## 4 实验心得

学习 Shell 语法、Vim 编辑器的使用以及正则表达式的掌握对于我们这种从事编程、系统管理或日常脚本开发的人来说是非常重要的技能，对于我们在未来无论是科研竞赛抑或是工作上都是有不小的帮助的。这学习的过程中我也体会颇多。

### 4.1 Shell

- 自动化能力：掌握了 Shell 脚本的基础之后，我们便可以编写简单的自动化任务，如文件备份、定时任务、数据处理等。
- 环境控制：Shell 脚本可以让我们更好地控制系统环境，包括启动服务、停止进程、监控资源使用等。
- 命令行效率：熟悉 Shell 命令和管道 (`|`)、重定向 (`<`, `>`) 等概念，可以提高我们在命令行界面工作的效率。

很重要的一点是，shell 对于格式的要求很严格，对于空格等有非常明确的判定规则，对于我们初学者时常会在这个方面犯错而久久找不到问题所在。

### 4.2 Vim 编辑器

- 高效编辑：Vim 的模式切换（普通模式、插入模式、命令行模式）使得文本编辑更加高效，尤其是通过键盘快捷键进行光标移动、文本选择、删除和复制等操作。
- 跨平台性：Vim 几乎在所有操作系统上都可以使用，这对于需要在不同平台上进行开发的人来说非常有用。

我以后在日常使用中尽量使用 Vim，逐渐习惯其操作方式。

### 4.3 数据整理

- 精确匹配：正则表达式可以让你非常精确地匹配和处理文本中的模式，无论是查找、替换还是提取信息都非常方便。
- 灵活性：正则表达式的灵活性非常高，几乎可以用来匹配任何文本模式。

## 5 Github仓库ssh链接

```
git@github.com:xixiyhaha/psychic-octo-engine.git
```