

第二次实验报告

时伟杰

2024 年 8 月 30 日

目录

1	shell 简介	1
2	vim 简介	1
3	实例演示与截图	2
4	心得体会	3
5	附带图片	3

1 shell 简介

Shell 是一个 命令解释器，它在操作系统的最外层负责直接与用户对话，把用户的输入解释给操作系统，并且处理操作系统的输出结果，输出到屏幕返回给用户。这种对话方式可以是交互的方式。Shell 是用 C 语言编写的程序，它是用户使用 Linux 的桥梁。Shell 既是一种命令语言，又是一种程序设计语言。简单来说，Shell 是指一种应用程序，这个应用程序提供了一个界面，用户通过这个界面访问操作系统内核的服务。也可以这样认为，Linux 中的 shell 就是 Linux 内核的一个外层保护工具，并负责完成用户与内核之间的交互。Shell 脚本是一种专门使用 shell 编写的脚本程序。Shell 脚本编程如同其他编程语言一样，只要有一个能编写代码的文本编辑器和一个能解释执行的脚本解释器就可以运行了。其中，Bash（Bourne Again Shell）因为其能提供环境变量、支持历史记录、内置算数功能、支持通配符表达式等高效性能，被广泛应用于 Debian 系列的 Linux 发行版中。

2 vim 简介

Vim 是一个高度可定制的文本编辑器，它是 Vi 编辑器的改进版，增加了许多特性并提高了易用性。Vim 在各种操作系统上运行，包括 Linux、macOS、Windows 等，这使得它成为跨平台的文本编辑器。Vim 支持多种字符集和语言，包括 UTF-8 和 Unicode，并且提供了丰富的编辑功能，如搜索和替换、宏录制与回放、多光标编辑等，这些功能有助于高效地编辑和处理各种文本文件。Vim 的设计理念是减少对鼠标的依赖，通过高效的键盘操作来提高工作效率。它拥有多种模式，包括普通模式、插入模式和可视模式等，每种模式下都可以进行特定的操作，这使得 Vim 的使用非常灵活且功能强大。Vim 的高度可定制性是其另一个重要特点，用户可以根据自己的需要配置 Vim，通过修改配置文件或安装插件来扩展其功能，满足不同的编程环境和需求。此外，Vim 还拥有一个庞大且活跃的社区，提供了丰富的资源和支持，无论是教程、插件还是问题解答，用户都可以轻松找到所需的帮助。这种社区支持对于学习和使用 Vim 的用户来说是非常宝贵的资源。

3 实例演示与截图

shell 操作	实例	介绍
1	date	输出现在的日期
2	echo hello	输出 hello 字符串
3	which echo	确定某个程序名代表的是哪个具体的程序
4	pwd	获取当前工作目录
5	cd /bin	切换到 bin 目录
6	cd ..	切换到上级目录
7	../bin/echo hello	另一种输出 hello 的方式
8	ls	查看指定目录下包含哪些文件
9	echo debug > debug.txt	将 debug 输入文本文档
10	cat debug.txt	输入文档中内容
11	cat < debug.txt	另一种输出的方式
12	cat < debug.txt > debug2.txt	将 debug.txt 中的文字输出到 debug2.txt
13	foo=bar echo "\$foo" echo '\$foo'	单引号与双引号的区别
14	与、或的实例	与操作符、或操作符
vim 操作	实例	介绍
15	vim+ 文件路径	打开指定文件
16	wq	退出文件
17	h 或左方向键	光标向左移动一个字符
18	l 或右方向键	光标向右移动一个字符
19	k 或上方向键	光标向上移动一个字符
20	j 或下方向键	光标向下移动一个字符
21	0 或 Ctrl+6 或 ome 键	光标移动到行首
22	Ctrl+4 或 end 键	光标移动到行尾
23	Ctrl+f	屏幕向下移动一页
24	Ctrl+b	屏幕向上移动一页
25	gg	移动到文件的第一行
26	G	移动到文件的最后一行
27	x	删除后一个字符
28	X	删除前一个字符

vim 操作	实例	介绍
29	dd	剪切一行
30	数字 dd	剪切多行
31	yy	复制一行
32	数字 yy	复制多行
33	p	粘贴到下一行
34	P	粘贴到上一行
35	u	撤回操作
36	Ctrl + r	撤回刚才的撤回操作
37	i	输入模式，进入后显示，INSTER
38	o	输入模式，在光标下一行插入新行，INSTER
39	R	取代模式，输入的值会取代光标所在的内容，REPLACE
40	esc 键	退出编辑模式（输入模式）

4 心得体会

经过老师和助教的帮助，以及配套资料和 B 站视频的学习，我对 shell 和 vim 有了初步的了解，明白了学习 Shell 脚本让我能够高效地自动化重复性任务，比如文件处理、系统监控和数据备份。通过掌握变量、条件语句和函数，我能够编写出更灵活和复杂的脚本，提高了工作效率。同时，Vim 的学习改变了我对文本编辑的理解。其模式化编辑方式虽然初看复杂，但极大提升了编辑速度。通过使用快捷键和命令，我能更快速地进行文本操作，而插件系统进一步增强了 Vim 的功能，使得编辑过程更加顺畅。但是学习过程是一波三折的，比如虚拟机的安装和了解更是消耗了不少时间，这也锻炼了我学习的能力和对计算机的进一步了解，综合这两者的使用，我的开发和系统管理能力得到了显著提升，使得面对各种任务时更加得心应手。

5 附带图片

```
m1585@Ü MINGW64 / (master)
$ date
Sat Aug 31 16:03:32      2024
```

图 1: 第一个实例演示

```
m1585@Ü MINGW64 / (master)
$ echo hello
hello
```

图 2: 第二个实例演示

```
m1585@Ü MINGW64 / (master)
$ which echo
/usr/bin/echo
```

图 3: 第三个实例演示

```
m1585@Ü MINGW64 / (master)
$ pwd
/
```

图 4: 第四个实例演示

```
m1585@Ü MINGW64 / (master)
$ cd /bin

m1585@Ü MINGW64 /bin (master)
$ pwd
/bin
```

图 5: 第五个实例演示

```
m1585@Ü MINGW64 /bin (master)
$ cd ..

m1585@Ü MINGW64 / (master)
$ pwd
/
```

图 6: 第六个实例演示

```
m1585@Ü MINGW64 / (master)
$ ../bin/echo hello
hello
```

图 7: 第七个实例演示

```
m1585@Ü MINGW64 / (master)
$ ls
LICENSE.txt      bin/  debug.txt  etc/  git-cmd.exe*  proc/  usr/
README.portable cmd/  dev/       git-bash.exe* mingw64/  tmp/
```

图 8: 第八个实例演示

```
m1585@Ü MINGW64 / (master)
$ echo debug > debug.txt

m1585@Ü MINGW64 / (master)
$ cat debug.txt
debug
```

图 9: 第九、十个实例演示

```
m1585@Ü MINGW64 / (master)
$ cat < debug.txt
debug
```

图 10: 第十一个实例演示

```
m1585@Ü MINGW64 / (master)
$ cat < debug.txt >debug2.txt

m1585@Ü MINGW64 / (master)
$ cat debug2.txt
debug
```

图 11: 第十二个实例演示

```
m1585@Ü MINGW64 / (master)
$ foo=bar
echo "$foo"
echo '$foo'
bar
$foo
```

图 12: 第十三个实例演示

```
m1585@Ü MINGW64 / (master)
$ false || echo "Oops, fail"
Oops, fail

m1585@Ü MINGW64 / (master)
$ true || echo "will not be printed"

m1585@Ü MINGW64 / (master)
$ true && echo "Things went well"
Things went well

m1585@Ü MINGW64 / (master)
$ false && echo "will not be printed"

m1585@Ü MINGW64 / (master)
$ false ; echo "This will always run"
This will always run
```

图 13: 第十四个实例演示