

实验报告二：Vim 和 Shell 工具脚本的学习

23020007011 崔涛

2024 年 9 月 10 日

1 实验目的

学习使用 Vim 编辑器和 Shell 工具和脚本，熟悉一些基本操作，方便以后的学习和工作需要。

2 介绍

2.1 Vim 和 Shell 简介

Vim 是一个强大的文本编辑器，从 vi 发展而来，被程序员广泛使用。它不仅支持代码补全、编译及错误跳转等功能，还通过语法高亮等方式方便编程

Shell 是 Linux 系统中与操作系统内核交互的接口，充当命令解释器的角色。最常用的 shell 环境是 bash。Shell 脚本能够批量执行一系列命令，非常适用于自动化任务

Vim 和 Shell 在 Linux 使用中各有其独特之处，它们提供了从文本编辑到系统管理、自动化任务处理的全面功能。掌握它们对于每个 Linux 用户来说都是必要的，不仅能提高工作效率，还能深入理解 Linux 系统的运作方式。

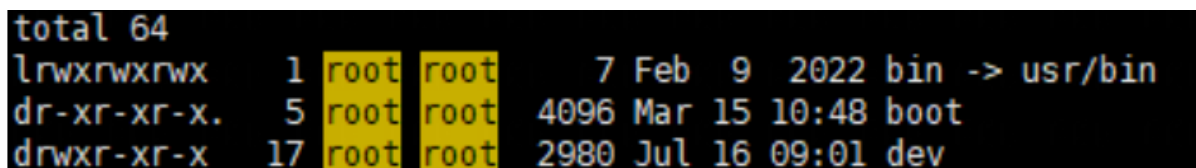
3 练习内容

3.1 Vim 学习样例

1. 在默认设置下，Vim 会在左下角显示当前的模式。Vim 启动时的默认模式是正常模式。

可以按下 <ESC>（退出键）从任何其他模式返回正常模式。在正常模式，键入 i 进入插入模式，R 进入替换模式，v 进入可视（一般）模式，V 进入可视（行）模式，<C-v>（Ctrl 进入可视（块）模式，: 进入命令模式。

2. vim +/关键字 文件路径可以高亮显示关键字，如图1所示



```
total 64
lrwxrwxrwx  1 root root    7 Feb  9 2022 bin -> usr/bin
dr-xr-xr-x.  5 root root 4096 Mar 15 10:48 boot
drwxr-xr-x 17 root root 2980 Jul 16 09:01 dev
```

图 1: 这是第二个例子的图片。

3. 在正常模式下键入 `:` 进入命令行模式。在键入 `:` 后, 你的光标会立即跳到屏幕下方的命令行。这个模式有很多功能, 包括打开, 保存, 关闭文件, 以及退出 Vim

- `:q` 退出 (关闭窗口)
- `:w` 保存 (写)
- `:wq` 保存然后退出
- `:e {文件名}` 打开要编辑的文件
- `:ls` 显示打开的缓存
- `:help {标题}` 打开帮助文档
- `:help :w` 打开 `:w` 命令的帮助文档
- `:help w` 打开 `w` 移动的帮助用户文档

4.

- `d{移动命令}` 删除 {移动命令} 例如, `dw` 删除词, `d$` 删除到行尾, `d0` 删除到行头。
- `c{移动命令}` 改变 {移动命令}
 - 例如, `cw` 改变词
 - 比如 `d{移动命令}` 再 `i`
- `x` 删除字符 (等同于 `dl`)
- `s` 替换字符 (等同于 `xi`)
- 可视化模式 + 操作选中文字, `d` 删除或者 `c` 改变
- `u` 撤销, `<C-r>` 重做
- `y` 复制 / “yank” (其他一些命令比如 `d` 也会复制)
- `p` 粘贴
- `~` 改变字符的大小写

5 `:s` (替换) 命令。

- `%s/foo/bar/g` 在整个文件中将 `foo` 全局替换成 `bar`
- `%s/\[.*\](\(.*)\)/\1/g` 将有命名的 Markdown 链接替换成简单 URLs

6. Vim 命令 / 宏

- `ggdd`, `Gdd` 删除第一行和最后一行
- 格式化最后一个元素的宏 (寄存器 `e`)

- 跳转到有 `<name>` 的行
- `qe^r"f>s": "<ESC>f<C"<ESC>q`
- 格式化一个的宏
 - 跳转到有 `<person>` 的行
 - `qpS{<ESC>j@eA,<ESC>j@ejS},<ESC>q`
- 格式化一个标签然后转到另外一个的宏
 - 跳转到有 `<person>` 的行
 - `qq@pjq`
- 执行宏到文件尾 `999@q`
- 手动移除最后的 `,` 然后加上 `[` 和 `]` 分隔符

7. 安装和配置插件 `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`
下载后需要在`./vimrc`中添加如下设置,

```
\textbf{set} runtimepath^=~/.vim/pack/vendor/start/ctrlp.\textbf{vim}
```

8. 安装 `vim-plug`

- 输入这串指令
`curl -fLo ~/.vim/autoload/plug.vim --create-dirs https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim`
- 修改 `./vimrc`
`call plugbegin()`
Plug 'preservim/NERDTree' 需要安装的插件 NERDTree
Plug 'wikitopian/hardmode' 安装 hardmode
`call plugend()`
- 在 `vim` 命令行中执行 `:PlugInstall`

9. 打开 `vim` 内置的教程, 输入命令 `vimtutor`。跟随教程可以得到一些基本的 `vim` 指导。

10. 输入 `vimdiff` 加上两个文件名即可轻松实现两个文档的对比!

3.2 Shell 学习样例 10 个

1. 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

```
19726@MINGW64 /d/崔涛/学习
$ ls -a
./ ../ PR/ 学习资料/ 毕业相关/ 活动竞赛/ 课程/ 选课/

19726@MINGW64 /d/崔涛/学习
$ ls -t
毕业相关/ 选课/ 课程/ PR/ 学习资料/ 活动竞赛/

19726@MINGW64 /d/崔涛/学习
$ ls -h
PR/ 学习资料/ 毕业相关/ 活动竞赛/ 课程/ 选课/

19726@MINGW64 /d/崔涛/学习
$ ls --color=auto
PR/ 学习资料/ 毕业相关/ 活动竞赛/ 课程/ 选课/
```

图 2: `ls` 指令查看文件

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。

解答：新建一个 `.sh` 文件，向文件中写入如下函数，然后执行 `source` 指令，随后再运行 `marco` 和 `polo` 即可

```
#!/bin/bash
```

```
marco()
```

```
echo "$(pwd)" >HOME/marco_history.log
```

```
echo "savepwd(pwd)"
```

```
polo()
```

```
cd "$(cat"HOME/marco_history.log")"
```

```
cuitao@ubuntu:~$ cd Desktop
cuitao@ubuntu:~/Desktop$ \source marco.sh
cuitao@ubuntu:~/Desktop$ source marco.sh
cuitao@ubuntu:~/Desktop$ marco
cuitao@ubuntu:~/Desktop$ cd ..
cuitao@ubuntu:~$ polo
cuitao@ubuntu:~/Desktop$
```

图 3: 编写简单脚本

3. 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。

解答：编写如下 `bash` 测试脚本，然后执行 `source` 指令，随后再运行即可。

```

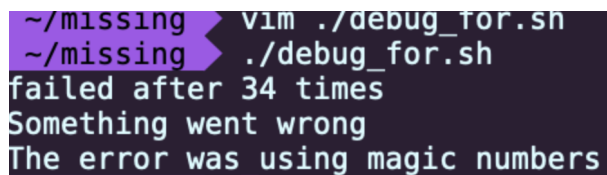
if [[ n -eq 42 ]]; then
echo "Something went wrong"
>&2 echo "The error was using magic numbers"
exit 1
fi

```

```

echo "Everything went according to plan"

```



```

~/missing$ vim ./debug_for.sh
~/missing$ ./debug_for.sh
failed after 34 times
Something went wrong
The error was using magic numbers

```

图 4: 测试脚本的编写

4. 编写一个命令，它可以递归地查找文件夹中所有的 HTML 文件，并将它们压缩成 zip 文件。注意，即使文件名中包含空格，您的命令也应该能够正确执行

首先创建所需的文件

```

mkdir html_root
cd html_root
touch \textbf{{{1..10}\textbf{}}}.html
mkdir html
cd html
touch xxxx.html

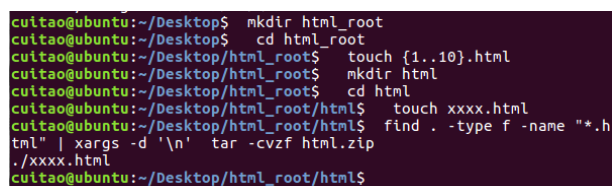
```

随后执行如下的 find 指令查看所有 HTML 文件并将其压缩为 ZIP 文件。

```

find . -type f -name "*.html" | xargs -d '\n' tar -cvzf html.zip

```



```

cuitao@ubuntu:~/Desktop$ mkdir html_root
cuitao@ubuntu:~/Desktop$ cd html_root
cuitao@ubuntu:~/Desktop/html_root$ touch {1..10}.html
cuitao@ubuntu:~/Desktop/html_root$ mkdir html
cuitao@ubuntu:~/Desktop/html_root$ cd html
cuitao@ubuntu:~/Desktop/html_root/html$ touch xxxx.html
cuitao@ubuntu:~/Desktop/html_root/html$ find . -type f -name "*.html" | xargs -d '\n' tar -cvzf html.zip
./xxxx.html
cuitao@ubuntu:~/Desktop/html_root/html$

```

图 5: 查看指定格式并压缩文件

5. 编写一个命令或脚本递归的查找文件夹中最近使用的文件

解答: 输入以下命令即可

```

find . -type f -print0 | xargs -0 ls -lt | head -1

```

```

cuitao@ubuntu:~/Desktop$ mkdir html_root
cuitao@ubuntu:~/Desktop$ cd html_root
cuitao@ubuntu:~/Desktop/html_root$ touch {1..10}.html
cuitao@ubuntu:~/Desktop/html_root$ mkdir html
cuitao@ubuntu:~/Desktop/html_root$ cd html
cuitao@ubuntu:~/Desktop/html_root/html$ touch xxxx.html
cuitao@ubuntu:~/Desktop/html_root/html$ find . -type f -name "*.html" | xargs -d '\n' tar -cvzf html.zip
/xxxx.html
cuitao@ubuntu:~/Desktop/html_root/html$ find . -type f -print0 | xargs -0 ls -lt | head -1
-rw-rw-r-- 1 cuitao cuitao 119 Sep 10 19:30 ./html.zip
cuitao@ubuntu:~/Desktop/html_root/html$

```

图 6: 查找最近使用文件

6. 当执行脚本时，我们经常需要提供形式类似的参数。bash 使我们可以轻松的实现这一操作，它可以基于文件扩展名展开表达式。这一技术被称为 shell 的通配

- 通配符当想要利用通配符进行匹配时，你可以分别使用 `?` 和 `*` 来匹配一个或任意个字符。例如，对于文件 `foo`, `foo1`, `foo2`, `foo10` 和 `bar`, `rm foo?` 这条命令会删除 `foo1` 和 `foo2`，而 `rm foo*` 则会删除除了 `bar` 之外的所有文件。
- 花括号 `{}` - 当有一系列的指令，其中包含一段公共子串时，可以用花括号来自动展开这些命令。这在批量移动或转换文件时非常方便。

```

convert image.{png,jpg}
# 会展开为
convert image.png image.jpg

cp /path/to/project/{foo,bar,baz}.sh /newpath
# 会展开为
cp /path/to/project/foo.sh /path/to/project/bar.sh /path/to/project/baz.sh

# 也可以结合通配使用
mv *.py,*.sh folder
# 会移动所有 *.py 和 *.sh 文件

mkdir foo bar

# 下面命令会创建 foo/a, foo/b, ... foo/h, bar/a, bar/b, ... bar/h
touch {foo,bar}/{a..h}
touch foo/x bar/y
# 比较文件夹 foo 和 bar 中包含文件的不同
diff <(ls foo) <(ls bar)
# 输出
# < x
# ---
# > y

```

图 7: Shell 命令中通配符的使用

7. 所有的类 UNIX 系统都包含一个名为 `find` 的工具，它是 shell 上用于查找文件的绝佳工具。`find` 命令会递归地搜索符合条件的文件，例如查找前一天修改过的文件。

```

cuitao@ubuntu:~/Desktop/html_root/html$ find . -mtime -1
./html.zip
/xxxx.html
cuitao@ubuntu:~/Desktop/html_root/html$

```

图 8: 利用 find 来查找前一天修改的文件

8. `history` 命令允许您以程序员的方式来访问 shell 中输入的历史命令。这个命令会在标准输出中打印 shell 中的历史命令。如果我们要搜索历史记录，则可以利用管道将输出结果传递给 `grep` 进行模式搜索。`history | grep find` 会打印包含 `find` 子串的命令。

```

cuitao@ubuntu:~/Desktop/html_root/html$ history|grep find
333 find cuitao
334 find ~ cuitao
335 find / -name cuitao
346 find . -cuitao cuitao
347 find . -name cuitao
348 find . -name cuitao.txt
349 find . -name cuitao.txt12
380 find ./ -name cuitao
381 find ./ -name *tao
389 sudo find / -size + 5K
390 sudo find / -size + 5k
391 sudo find / -size +5k
392 sudo find / -size + 5G
393 sudo find / -size +5G
394 sudo find / -size + 5
395 sudo find / -size +5
459 ls | find cuitao.txt | cat
797 find . -mtime -1
798 history|grep find

```

图 9: 查找以往命令记录

9.alias 是 Bash 内建命令，用来设置命令的别名。

alias [-p] [NAME[=VALUE] ...]

10.

chmod 命令

- `ls -lh` 显示当前目录所有文件的权限
- `chmod 777 文件名` 修改文件权限（最高权限）

4 解题感悟

在初步接触 Vim 和 Shell 后，我不禁对这两个工具的强大功能和灵活性感到惊讶。作为计算机科学的初学者，我原以为图形界面的编辑器和操作系统是现代编程的标配。然而，通过学习 Vim 和 Shell，我发现命令行界面同样能够提供高效、直观的操作环境。

在学习 Vim 的过程中，最初我对它的模式切换感到困惑，为何一个简单的文本编辑器需要如此复杂的操作模式？但当我逐渐习惯了这种模式之后，我意识到这是 Vim 设计哲学的一部分，旨在提高编辑效率。我可以不用离开键盘就能完成大部分文本编辑任务，这显著加快了我的工作速度。使用 hjkl 键进行移动，以及各种快捷命令，如 dd 和 yy，让我感受到了 Vim 的威力。更不用说，Vim 脚本和插件的可扩展性为我定制自己的工作环境提供了无限可能。

转向 Shell 的学习，我开始了解到命令行。Shell 不仅仅是一个简单的命令执行器，它是一种强大的脚本语言，允许我自动化日常任务、处理文件和数据流、甚至编写复杂的程序。学习如何使用管道、重定向、和正则表达式，让我能够以前所未有的方式处理文本数据。此外，Shell 脚本让我能够将常用的命令序列保存为脚本，实现一键执行，极大提升了我的工作效率。总的来说，虽然 Vim 和 Shell 的学习对我比较困难，但一旦掌握，它们就会成为了我的工具箱中不可或缺的工具。它们不仅提高了我的编程效率，还让我对计算机的工作方式有了更深入的理解。我期待着继续探索它们的高级功能，并将这些知识应用到更广泛的计算机科学领域中去。

github 路径您可以在这里查看我的源代码：

<https://github.com/cuitao223/myhomework>