

Shell 和 Vim 学习

23020007160 张绍延

2024 年 9 月 12 日

1 实验目的

本次课程主要讲授了 Shell 工具和脚本，用编辑器 Vim 进行数据整理。

2 介绍

2.1 两大工具的优点

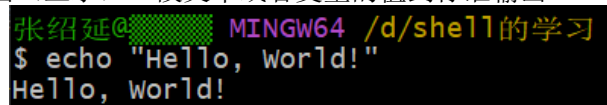
1.Shell 的优点：Shell 脚本可以将多个命令组合在一起，自动执行重复性的任务，大大提高工作效率。Shell 可以很容易地调用其他编程语言（如 Python、Perl 等）编写的脚本或程序。

2.Vim 有一个非常高效的编辑模型，熟练使用后可以大大提高文本编辑速度。Vim 可以在命令行中直接使用，非常适合在服务器或远程环境中进行文本编辑。

3 练习内容

3.1 Shell 学习例子 10 个

1.echo "Hello, World!" 用 Shell 打印 hello world echo 是一个在命令行界面（shell）中常用的命令，它的主要作用是输出（显示）一段文本或者变量的值到标准输出

A terminal window screenshot with a black background. The prompt is '张绍延@MINGW64 /d/shell的学习'. The command '\$ echo "Hello, world!"' is entered, and the output 'Hello, world!' is displayed on the next line.

```
张绍延@MINGW64 /d/shell的学习
$ echo "Hello, world!"
Hello, world!
```

图 1: echo 用来显示文本或变量

2. 变量赋值以及变量的使用

```
word="Hello, Git Bash!"
```

```
echo $word
```

#向word变量赋值，再通过echo \$输出变量

```
张绍延@MINGW64 /d/shell的学习
$ word="Hello, Git Bash!"

张绍延@MINGW64 /d/shell的学习
$ echo $word
Hello, Git Bash!
```

图 2: 变量的赋值和使用

3. 条件语句的判断

```
if [ "$word" = "Hello, Git Bash!" ];then
    echo "Yes."
else
    echo "No."
fi
```

#[] 里是条件判断, then后面是条件正确的结果, else是条件错误的结果

```
张绍延@MINGW64 /d/shell的学习
$ if [ "$word" = "Hello, Git Bash!" ]; then
    echo "Yes."
else
    echo "No."
fi
Yes.
```

图 3: 条件语句的判断

4. 循环的使用

```
for i in {1..5}; do
    echo "number: $i"
done
```

#in {1..5} 指定了循环的范围。

在这个例子中, {1..5} 是一个序列表达式, 表示从1到5的数字序列, 包括1和5。

done是结束这个循环

```
张绍延@MINGW64 /d/shell的学习
$ if [ "$word" = "Hello, Git Bash!" ]; then
    echo "Yes."
else
    echo "No."
fi
Yes.

张绍延@MINGW64 /d/shell的学习
$ for i in {1..5}; do
    echo "number: $i"
done
number: 1
number: 2
number: 3
number: 4
number: 5
```

图 4: 循环语句

5. 文件的创立与写入

创建一个新文件

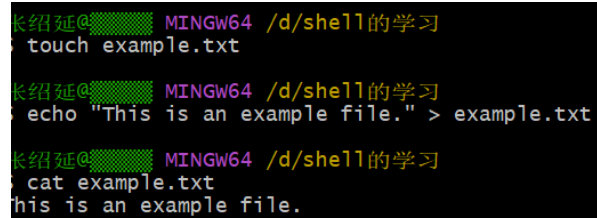
```
touch example.txt
```

写入内容到文件

```
echo "This is an example file." > example.txt
```

读取文件内容

```
cat example.txt
```

A terminal window with a black background and green text. The prompt is '张绍延@MINGW64 /d/shell的学习'. The user enters 'touch example.txt', then 'echo "This is an example file." > example.txt', and finally 'cat example.txt'. The output of the last command is 'his is an example file.'.

```
张绍延@MINGW64 /d/shell的学习
$ touch example.txt

张绍延@MINGW64 /d/shell的学习
$ echo "This is an example file." > example.txt

张绍延@MINGW64 /d/shell的学习
$ cat example.txt
his is an example file.
```

图 5: 创建写入读取文件

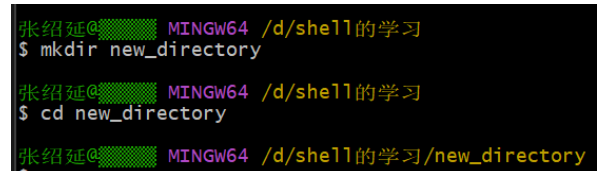
6. 创建目录与切换

创建一个新目录

```
mkdir new_directory
```

切换到新目录

```
cd new_directory
```

A terminal window with a black background and green text. The prompt is '张绍延@MINGW64 /d/shell的学习'. The user enters 'mkdir new_directory', then 'cd new_directory', and the prompt changes to '张绍延@MINGW64 /d/shell的学习/new_directory'.

```
张绍延@MINGW64 /d/shell的学习
$ mkdir new_directory

张绍延@MINGW64 /d/shell的学习
$ cd new_directory

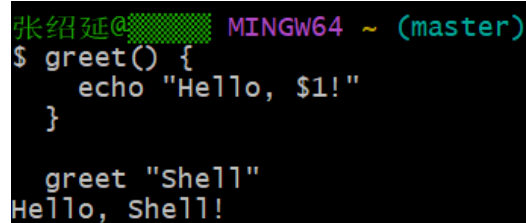
张绍延@MINGW64 /d/shell的学习/new_directory
$
```

图 6: 创建与切换新目录

7. 函数的创建和使用

```
greet() {
    echo "Hello, $1!"
}
```

```
greet "Shell"
```

A terminal window with a black background and green text. The prompt is '张绍延@MINGW64 ~ (master)'. The user enters 'greet() { echo "Hello, \$1!" }' and then 'greet "Shell"'. The output is 'Hello, Shell!'.

```
张绍延@MINGW64 ~ (master)
$ greet() {
    echo "Hello, $1!"
}

$ greet "Shell"
Hello, Shell!
```

图 7: 定义函数与使用

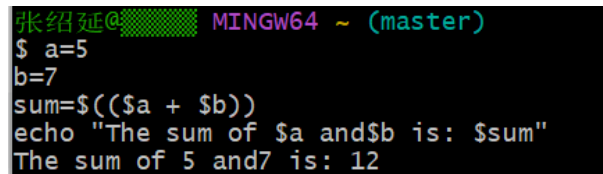
8. 加法运算

```
a=5
```

```

b=7
sum=$((a + b))
echo "The sum of $a and$b is: $sum"

```



```

张绍延@MINGW64 ~ (master)
$ a=5
b=7
sum=$((a + b))
echo "The sum of $a and$b is: $sum"
The sum of 5 and7 is: 12

```

图 8: 进行加法运算

9. 获取工作目录，并列出工作目录

```

# 获取当前工作目录
pwd
# 列出当前目录下的所有文件和目录
ls -l

```



```

张绍延@MINGW64 /d/shell的学习
$ pwd
/d/shell的学习

张绍延@MINGW64 /d/shell的学习
$ ls -l
example.txt
new_directory/

```

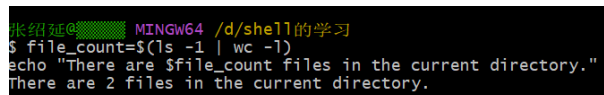
图 9: 获取目录并列出当前目录

10. 查看当前目录文件数目

```

file_count=$(ls -l | wc -l)
echo "There are $file_count files in the current directory."

```



```


张绍延@MINGW64 /d/shell的学习
$ file_count=$(ls -l | wc -l)
echo "There are $file_count files in the current directory."
There are 2 files in the current directory.

```

图 10: 查看当前目录文件数目

3.2 Vim 学习例子 10 个

1.Vim 文件的创立，在终端中输入 vim example.txt



```

"example.txt" [新]
0,0-1 全部

```

图 11: Vim 文件的创立

2. i 键为插入，开始输入内容 hello,vim!

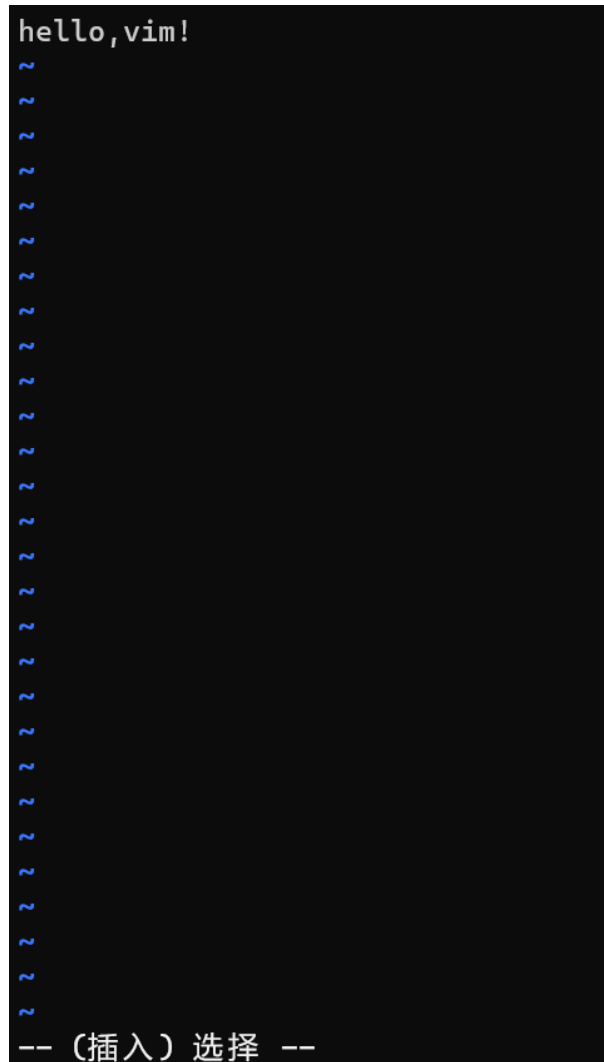


图 12: i 插入内容

3. 编辑的退出与保存

Esc是回到正常模式

: **w**带编者保存

: **q**代表退出

如果不按**w**的话则文件为**SWP**文件，用于临时储存数据

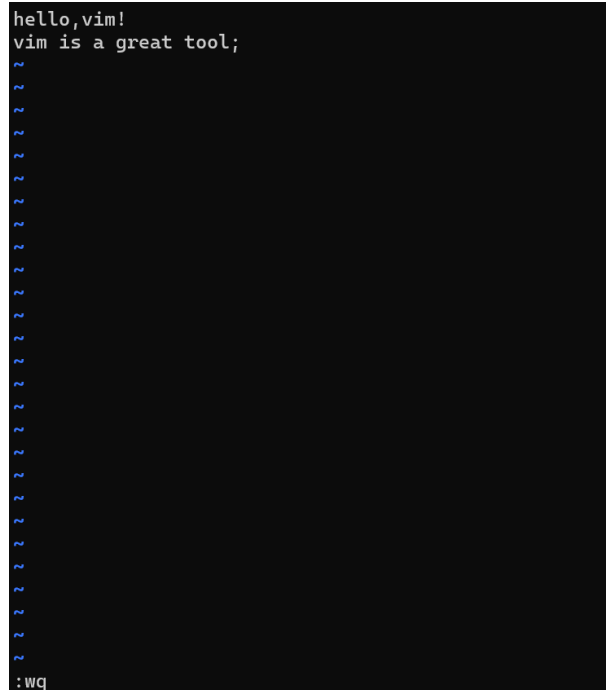


图 13: 编辑的退出与保存

4.dd 表示删除当前行

在vim is a greet tool前输入dd删除后的结果

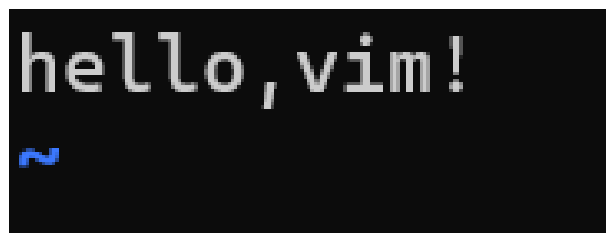


图 14: dd 删除当前行后的结果

5.dw: 删除光标后的单词。

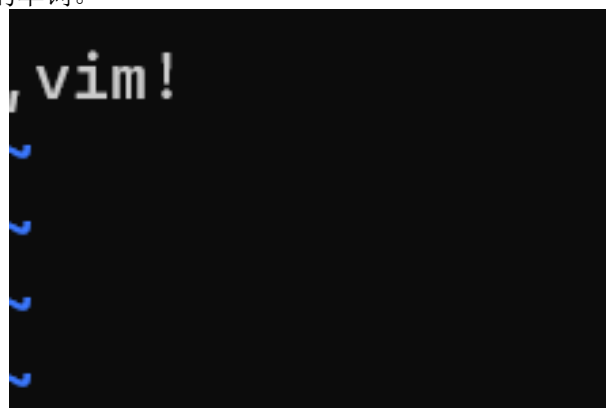


图 15: dw 删除光标后的单词的结果

6.x: 删除光标下的字符。

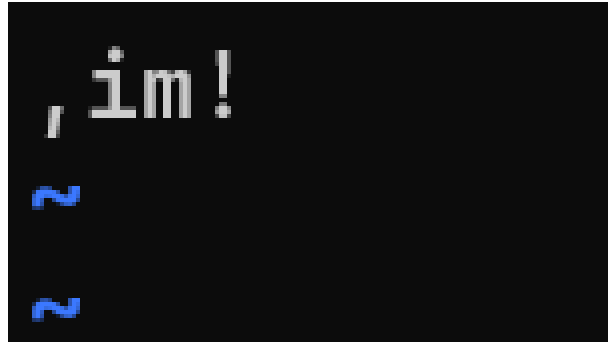


图 16: 删除光标下的字符的结果

7.r: 替换光标下的单个字符。R: 进入替换模式，连续替换多个字符直到按下 Esc。

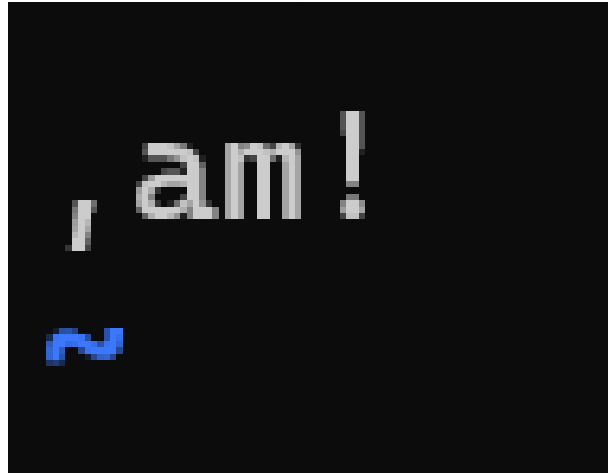


图 17: 用 r 替换光标下的单个字符

[illegible]

图 18: 在 R 模式未改变前

```
hello,vim!  
vim is a gwood tool.
```

图 19: 在 R 模式改变后

8. u: 撤销最后一次更改。

Ctrl + r: 重做最后一次撤销, 即反撤销。

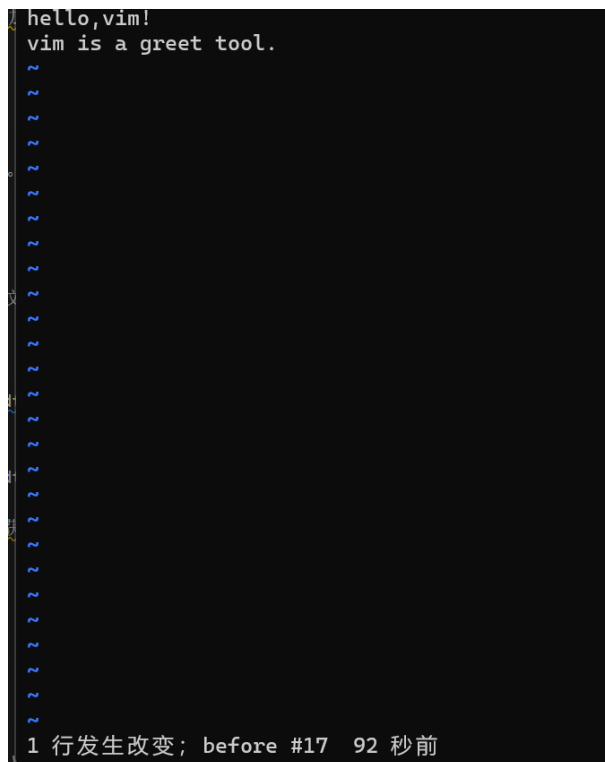


图 20: 撤销

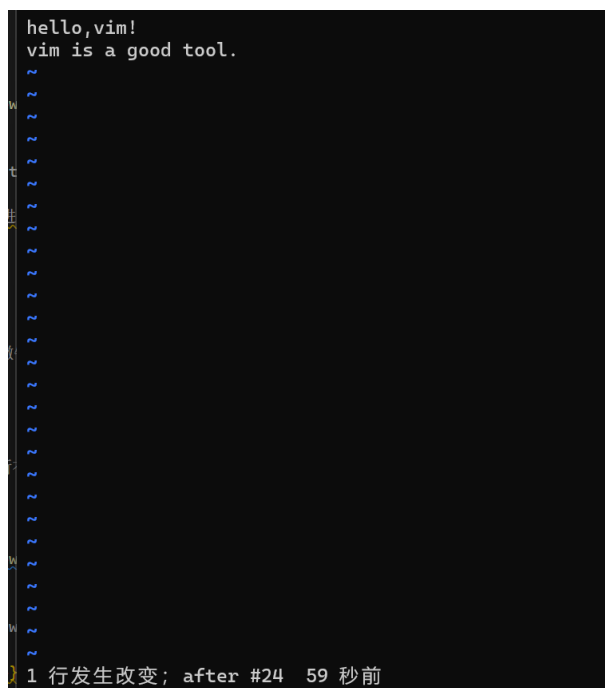


图 21: 反撤销

9. 复制的学习:

复制单行: 将光标移动到要复制的行的任意位置。输入 `yy` 来复制整行。

复制部分行: 将光标移动到要开始复制的位置。进入可视化模式, 可以按 `v` (用于字符模式) 或 `V` (用于行模式)。使用光标键 `h``j``k``l` 选择要复制的文本。一旦选择了文本, 按 `y` 来复制选中的文本。

复制多个行: 将光标移动到要复制的第一行的任意位置。输入数字 `yy`, 其中“数字”是您想要

复制的行数。例如，3yy 会复制光标所在的当前行以及下面两行，总共三行。

```
hello,vim!  
vim is a good tool.  
  
vim is a good tool.  
~
```

图 22: 复制一行后的结果

```
hello,vim!  
vim is a good tool.  
  
vim is a good tool.  
vim is a good tool.  
  
vim is a good tool  
~
```

图 23: 选择性复制后的结果

10. p: 在光标后粘贴。P: 在光标前粘贴。下面是粘贴后的结果

```
hello,vim!  
vim is a good tool.  
  
vim is a good tool.  
~
```

图 24: 粘贴后的结果

4 解题感悟

Shell 是一种强大的自动化工具。通过编写脚本，可以将日常繁琐的任务变得简单高效，这让我对编程产生了浓厚的兴趣，也锻炼了我的逻辑思维能力。与此同时，Shell 的学习让我更加深入地理解了操作系统的内部工作原理，我开始意识到，每一个命令背后都隐藏着复杂而精巧的设计，这激发了我对技术的探索欲。

Vim 完全改变了我对文本编辑的看法。起初，Vim 的模式系统和快捷键让我感到困惑，但随着时间的推移，熟练使用 Vim 后，我编辑文本的速度和准确性都有了显著提升。Vim 的学习过程也是对个人习惯的一次挑战。它要求我放弃一些固有的操作习惯，转而接受一种更高效的工作方式。这个过程虽然艰难，但也让我认识到，改变习惯需要时间和耐心，而一旦新的习惯形成，它将带来巨大的收益。

总的来说，Shell 和 Vim 的学习不仅仅提升了我的技术能力，更让我在解决问题的思维方式上

有了新的认识。

github 路径您可以在查看项目的源代码:

<https://github.com/AmbitiousLight/AmbitiousLight/tree/ff751aec5aaa0eb9b01d6326244bc747e2921>

Latex%E8%AE%BA%E6%96%87