

第二次实验报告

——Shell 工具和脚本；编译器（Vim）；数 据整理

杨昕昱 23020007142

September 2024

目录

1	Shell 工具和脚本	3
1.1	实例 1: 下载 Windows Subsystem for Linux	3
1.2	实例 2: 在 shell 中导航	3
1.3	实例 3: 在程序间创建连接	4
1.4	实例 4: bash 脚本中为变量赋值	5
1.5	实例 5: 查找文件	5
1.6	实例 6: 返回码	5
1.7	实例 7: 函数	6
1.8	实例 8: ls 命令	6
1.9	实例 9: 编写两个 bash 函数	8
2	编译器 (Vim)	10
2.1	实例 1: 完成 vintutor	10
2.2	实例 2: 安装和配置一个插件: ctrlp.vim	11
2.3	实例 3: CtrlP	11
2.4	实例 4: 自定义 CtrlP	12
2.5	实例 5: 移动	13
2.6	实例 6: 选择可视化模式	13
2.7	实例 7: 编辑	13
2.8	实例 8: 计数	14
2.9	实例 9: 修饰语	14
3	数据整理, 正则表达式	15
3.1	实例 1: 查找	15
3.2	实例 2: 替换	15

1 Shell 工具和脚本

1.1 实例 1：下载 Windows Subsystem for Linux

访问 Windows Subsystem for Linux 官方网址，根据指示安装 WSL。打开 PowerShell，通过右键单击并选择“以管理员身份运行”，输入 `wsl -install` 命令，然后重新启动电脑。

安装WSL命令

现在，您可以用一个命令安装运行WSL所需的一切。在中打开PowerShell或Windows命令提示符**管理人员**通过右键单击并选择“以管理员身份运行”，输入`wsl -install`命令，然后重新启动您的机器。

```
PowerShell Copy  
  
wsl --install
```

此时打开 WSL，安装后界面如下。

```
Installing, this may take a few minutes...  
Please create a default UNIX user account. The username does not need to match your Windows username.  
For more information visit: https://aka.ms/wslusers  
Enter new UNIX username: yxy  
New password:  
Retype new password:  
passwd: password updated successfully  
Installation successful!  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
  
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.153.1-microsoft-standard-WSL2 x86_64)  
  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:        https://ubuntu.com/advantage  
  
This message is shown once a day. To disable it please create the  
/home/yxy/.hushlogin file.
```

1.2 实例 2：在 shell 中导航

如果某个路径以 `/` 开头，那么它是一个绝对路径，其他的都是相对路径。相对路径是指相对于当前工作目录的路径，当前工作目录可以使用 `pwd` 命令来获取。此外，切换目录需要使用 `cd` 命令。在路径中，`.` 表示的是当前目录，而`..` 表示上级目录。

```

yxy@DESKTOP-6MF70N6:/$ pwd
/
yxy@DESKTOP-6MF70N6:/$ cd /home/yxy
yxy@DESKTOP-6MF70N6:~$ pwd
/home/yxy
yxy@DESKTOP-6MF70N6:~$ cd /home
yxy@DESKTOP-6MF70N6:/home$ pwd
/home
yxy@DESKTOP-6MF70N6:/home$ cd ..
yxy@DESKTOP-6MF70N6:/$ pwd
/
yxy@DESKTOP-6MF70N6:/$ cd ./home
yxy@DESKTOP-6MF70N6:/home$ pwd
/home
yxy@DESKTOP-6MF70N6:/home$ cd yxy
yxy@DESKTOP-6MF70N6:~$ pwd
/home/yxy

```

1.3 实例 3：在程序间创建连接

在 shell 中，程序有两个主要的“流”：它们的输入流和输出流。当程序尝试读取信息时，它们会从输入流中进行读取，当程序打印信息时，它们会将信息输出到输出流中。通常，一个程序的输入输出流都是您的终端。也就是，您的键盘作为输入，显示器作为输出。但是，我们也可以重定向这些流！最简单的重定向是 `< file` 和 `> file`。这两个命令可以将程序的输入输出流分别重定向到文件。

```

yxy@DESKTOP-6MF70N6:~$ echo hello > hello.txt
yxy@DESKTOP-6MF70N6:~$ cat hello.txt
hello
yxy@DESKTOP-6MF70N6:~$ cat < hello.txt
hello
yxy@DESKTOP-6MF70N6:~$ cat < hello.txt > hello2.txt
yxy@DESKTOP-6MF70N6:~$ cat hello2.txt
hello

```

1.4 实例 4: bash 脚本中为变量赋值

在 bash 中为变量赋值的语法是 `foo=bar`，访问变量中存储的数值，其语法为 `$foo`。Bash 中的字符串通过 `'` 和 `"` 分隔符来定义，但是它们的含义并不相同。以 `'` 定义的字符串为原义字符串，其中的变量不会被转义，而 `"` 定义的字符串会将变量值进行替换。

```
yxy@DESKTOP-6MF70N6:~$ foo=bar
yxy@DESKTOP-6MF70N6:~$ echo "$foo"
bar
yxy@DESKTOP-6MF70N6:~$ echo '$foo'
$foo
```

1.5 实例 5: 查找文件

程序员们面对的最常见的重复任务就是查找文件或目录。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：

```
yxy@DESKTOP-6MF70N6:~$ find . -name src -type d
yxy@DESKTOP-6MF70N6:~$ find . -path '*/test/*.py' -type f
yxy@DESKTOP-6MF70N6:~$ find . -mtime -1
.
./snap
./snap/ubuntu-desktop-installer
./snap/ubuntu-desktop-installer/current
./snap/ubuntu-desktop-installer/common
./snap/ubuntu-desktop-installer/common/.cache
./snap/ubuntu-desktop-installer/common/.cache/gdk-pixbuf-loaders.cache
./snap/ubuntu-desktop-installer/1286
./snap/ubuntu-desktop-installer/1286/.last_revision
./motd_shown
yxy@DESKTOP-6MF70N6:~$ find . -size +500k -size -10M -name '*.tar.gz'
```

1.6 实例 6: 返回码

命令通常使用 `STDOUT` 来返回输出值，使用 `STDERR` 来返回错误及错误码，便于脚本以更加友好的方式报告错误。返回码或退出状态是脚本/命令之间交流执行状态的方式。返回值 0 表示正常执行，其他所有非 0

的返回值都表示有错误发生。退出码可以搭配 `&&`（与操作符）和 `||`（或操作符）使用，用来进行条件判断，决定是否执行其他程序。它们都属于短路运算符（short-circuiting）同一行的多个命令可以用 `;` 分隔。程序 `true` 的返回码永远是 0，`false` 的返回码永远是 1。

```
yxy@DESKTOP-6MF70N6:~$ false || echo "Oops, fail"
Oops, fail
yxy@DESKTOP-6MF70N6:~$ true || echo "Will not be printed"
yxy@DESKTOP-6MF70N6:~$ true && echo "Things went well"
Things went well
yxy@DESKTOP-6MF70N6:~$ false && echo "Will not be printed"
yxy@DESKTOP-6MF70N6:~$ false ; echo "This will always run"
This will always run
```

1.7 实例 7：函数

`bash` 也支持函数，它可以接受参数并基于参数进行操作。下面这个函数是一个例子，它会创建一个文件夹并使用 `cd` 进入该文件夹。这里 `$1` 是脚本的第一个参数。

```
yxy@DESKTOP-6MF70N6:~$ mcd () {
    mkdir -p "$1"
    cd "$1"
}
yxy@DESKTOP-6MF70N6:~$ mcd new
yxy@DESKTOP-6MF70N6:~/new$ pwd
/home/yxy/new
yxy@DESKTOP-6MF70N6:~/new$ |
```

1.8 实例 8：ls 命令

题目介绍 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

解题 首先输入 `man ls`，查看 `ls` 命令的说明。

```
LS(1) User Commands LS(1)
NAME
  ls - list directory contents

SYNOPSIS
  ls [OPTION]... [FILE]...

DESCRIPTION
  List information about the FILES (the current directory by default). Sort entries alphabetically if none of
  -cftuvSUX nor --sort is specified.

  Mandatory arguments to long options are mandatory for short options too.

  -a, --all
    do not ignore entries starting with .

  -A, --almost-all
    do not list implied . and ..

  --author
    with -l, print the author of each file

  -b, --escape
    print C-style escapes for nongraphic characters

  --block-size=SIZE
    with -l, scale sizes by SIZE when printing them; e.g., '--block-size=M'; see SIZE format below

Manual page ls(1) line 1/223 11% (press h for help or q to quit)
```

找到题目中操作相应的命令。

```
-a, --all
    do not ignore entries starting with .

-h, --human-readable
    with -l and -s, print sizes like 1K 234M 2G etc.

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

-t    sort by time, newest first; see --time
```

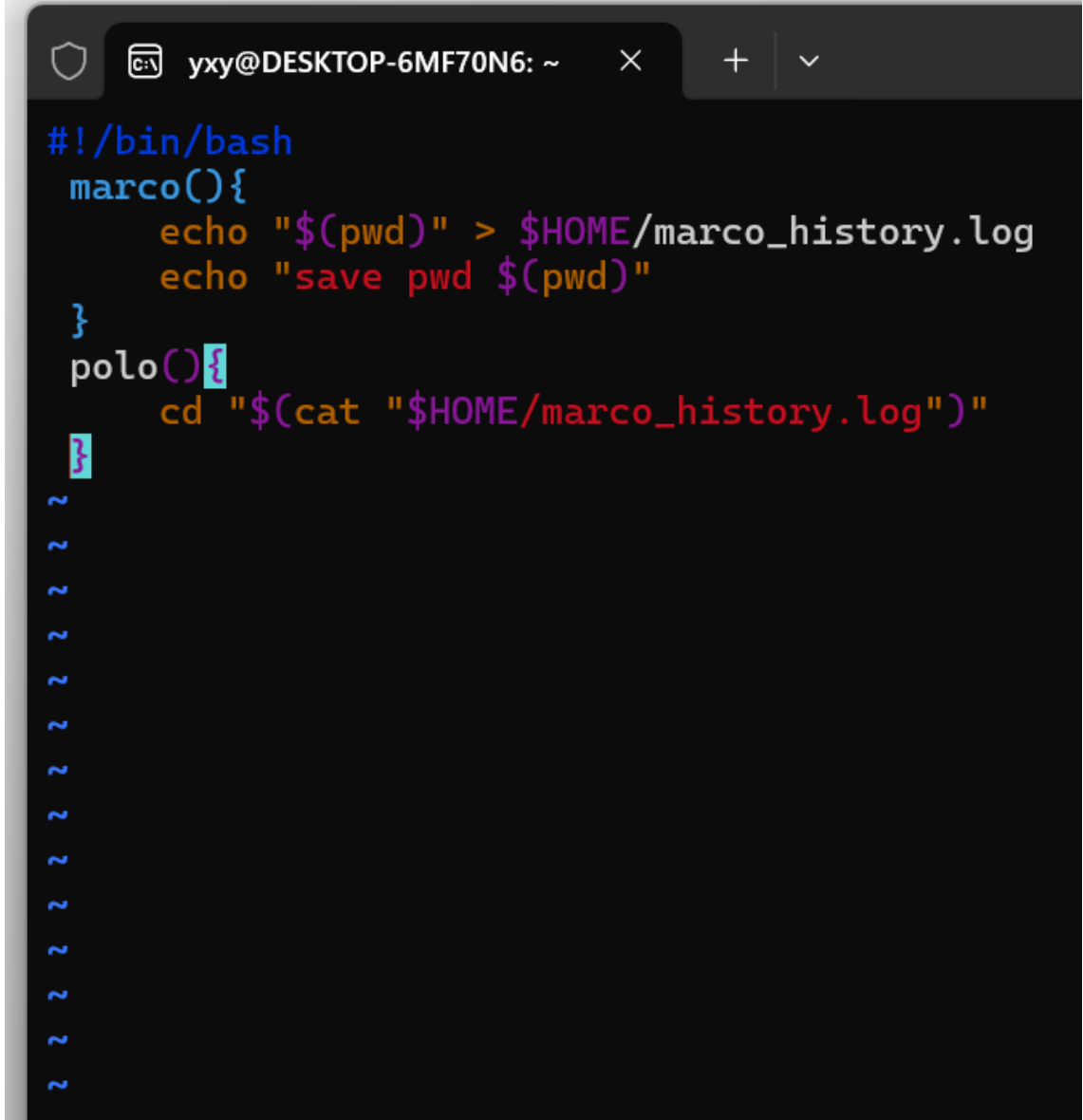
输入这些命令，输出结果如下：

```
xyx@DESKTOP-6MF70N6:~$ man ls
xyx@DESKTOP-6MF70N6:~$ ls -a
.  ..  .bash_logout  .bashrc  .cache  .lessshst  .motd_shown  .profile  hello.txt  hello2.txt
xyx@DESKTOP-6MF70N6:~$ ls -t
hello2.txt  hello.txt
xyx@DESKTOP-6MF70N6:~$ ls -h
hello.txt  hello2.txt
xyx@DESKTOP-6MF70N6:~$ ls -color=auto
ls: invalid option -- '='
Try 'ls --help' for more information.
xyx@DESKTOP-6MF70N6:~$ ls --color=auto
hello.txt  hello2.txt
xyx@DESKTOP-6MF70N6:~$ ls -l /home
total 4
drwxr-x--- 3 yxy yxy 4096 Aug 30 11:26 yxy
```

1.9 实例 9：编写两个 bash 函数

题目介绍 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。

解题 输入 `vim marco.sh`，使用 `vim` 文本编辑器，插入编写的函数。

A screenshot of a terminal window with a dark background. The window title bar shows a shield icon, a file icon, and the text 'yxy@DESKTOP-6MF70N6: ~'. The terminal content shows a bash script with two functions: 'marco()' and 'polo()'. The 'marco()' function uses 'echo' to write the current directory to a log file. The 'polo()' function uses 'cd' to change to the directory stored in the log file. The script ends with a closing brace. The terminal prompt is '~'.

输入 `source marco.sh` 来加载这些函数。这时就可以通过运行 `marco` 来保存当前目录。当你想要回到最近一次保存的目录时，只需运行 `polo`。

```

yxy@DESKTOP-6MF70N6:~$ vim marco.sh
yxy@DESKTOP-6MF70N6:~$ source marco.sh
yxy@DESKTOP-6MF70N6:~$ ls
hello.txt  hello2.txt  marco.sh
yxy@DESKTOP-6MF70N6:~$ marco
save pwd /home/yxy
yxy@DESKTOP-6MF70N6:~$ polo
yxy@DESKTOP-6MF70N6:~$ cd
yxy@DESKTOP-6MF70N6:~$ pwd
/home/yxy
yxy@DESKTOP-6MF70N6:~$ marco
save pwd /home/yxy
yxy@DESKTOP-6MF70N6:~$ cd ..
yxy@DESKTOP-6MF70N6:/home$ polo
yxy@DESKTOP-6MF70N6:~$ |

```

2 编译器 (Vim)

2.1 实例 1: 完成 vimtutor

直接输入命令 `vimtutor` 打开。可根据提示完成学习相关操作。

```

=====
=  Welcome to the VIM Tutor - Version 1.7  =
=====

Vim is a very powerful editor that has many commands, too many to
explain in a tutor such as this. This tutor is designed to describe
enough of the commands that you will be able to easily use Vim as
an all-purpose editor.

The approximate time required to complete the tutor is 30 minutes,
depending upon how much time is spent with experimentation.

ATTENTION:
The commands in the lessons will modify the text. Make a copy of this
file to practice on (if you started "vimtutor" this is already a copy).

It is important to remember that this tutor is set up to teach by
use. That means that you need to execute the commands to learn them
properly. If you only read the text, you will forget the commands!

Now, make sure that your Caps-Lock key is NOT depressed and press
the  j  key enough times to move the cursor so that lesson 1.1
completely fills the screen.
=====
Lesson 1.1:  MOVING THE CURSOR

** To move the cursor, press the h,j,k,l keys as indicated. **
^
"/tmp/tutorSsNSFd" 972 lines, 33583 bytes

```

2.2 实例 2：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。再用 `cd ~/.vim/pack/vendor/start` 打开文件夹。用 `git clone https://github.com/ctrlpvim/ctrlp.vim` 下载插件。

```

yxy@DESKTOP-6MF70N6:~$ mkdir -p ~/.vim/pack/vendor/start
yxy@DESKTOP-6MF70N6:~$ cd ~/.vim/pack/vendor/start
yxy@DESKTOP-6MF70N6:~/.vim/pack/vendor/start$ git clone https://github.com/ctrlpvim/ctrlp.vim
Cloning into 'ctrlp.vim'...
remote: Enumerating objects: 4299, done.
remote: Counting objects: 100% (168/168), done.
remote: Compressing objects: 100% (105/105), done.
remote: Total 4299 (delta 71), reused 147 (delta 62), pack-reused 4131 (from 1)
Receiving objects: 100% (4299/4299), 1.70 MiB | 2.48 MiB/s, done.
Resolving deltas: 100% (1661/1661), done.

```

2.3 实例 3: CtrlP

用 Vim 打开这个插件。

[illegible]

然后用 Vim 命令控制行开始:CtrlP。用 CtrlP 来在一个工程文件夹里定位一个文件。

```
~
~
~
:CtrlP|
```

```
doc/ctrlp.cnx 1,1 Top
> autoload/ctrlp/mixed.vim
> autoload/ctrlp.vim
> plugin/ctrlp.vim
> readme.md
> autoload/ctrlp/line.vim
> autoload/ctrlp/undo.vim
> autoload/ctrlp/tag.vim
> autoload/ctrlp/dir.vim
> doc/ctrlp.txt
> LICENSE
```

2.4 实例 4：自定义 CtrlP

要通过按 Ctrl-P 打开 CtrlP，需要在 `/.vimrc` 文件中添加一些配置。打开 `/.vimrc` 文件，并添加以下内容：

```
packadd! ctrlp.vim

nnoremap <C-p> :CtrlP<CR>

inoremap <C-p> <Esc>:CtrlP<CR>

let g:ctrlp_working_path_mode = 'ra'
~
~
```

添加完这些配置后，保存 `/.vimrc` 文件，并在 Vim 中重新加载配置，可以通过 `:source ~/.vimrc` 命令在 Vim 内部完成。

```
~
~
~
:source ~/.vimrc|
```

现在，在 Vim 中按下 Ctrl-P 时，就能够打开 CtrlP 插件来搜索和打开文件了。

```
[No Name] 0,0-1 ALL
> ctrlp.vim/autoload/ctrlp.vim
> ctrlp.vim/plugin/ctrlp.vim
> ctrlp.vim/doc/ctrlp.txt
> ctrlp.vim/doc/ctrlp.cnx
> ctrlp.vim/readme.md
> ctrlp.vim/autoload/ctrlp/line.vim
> ctrlp.vim/autoload/ctrlp/undo.vim
> ctrlp.vim/autoload/ctrlp/tag.vim
> ctrlp.vim/autoload/ctrlp/dir.vim
> ctrlp.vim/LICENSE
prt path <mru>=f files !=<buf> <-> /home/yxy/.vim/pack/vendor/start
>>> _
```

2.5 实例 5：移动

基本移动: 按键 hjkl (左, 下, 上, 右)

2.6 实例 6：选择可视化模式

选择可视化模式:

-可视化: v

-可视化行: V

-可视化块: Ctrl+v

```
-- VISUAL --
-- VISUAL LINE --
```

2.7 实例 7：编辑

用键盘替代鼠标: 采用编辑命令和移动命令的组合来完成。

-i 进入插入模式

但是对于操纵/编辑文本, 不单想用退格键完成

-O / o 在之上/之下插入行

-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



2.8 实例 8: 计数

用一个计数来结合“名词”和“动词”，这会执行指定操作若干次。

-3w 向后移动三个词

-5j 向下移动 5 行

-7dw 删除 7 个词

2.9 实例 9: 修饰语

可以用修饰语改变“名词”的意义。修饰语有 i, 表示“内部”或者“在内”，和 a, 表示“周围”。

-ci(改变当前括号内的内容

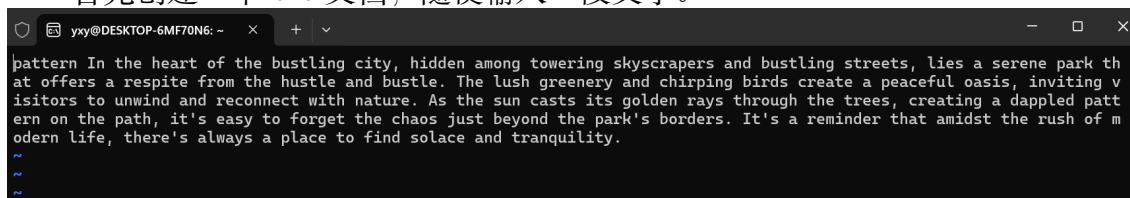
-ci[改变当前方括号内的内容

-da' 删除一个单引号字符串，包括周围的单引号

3 数据整理，正则表达式

3.1 实例 1：查找

首先创建一个 txt 文档，随便输入一段文字。



```
xy@DESKTOP-6MF70N6: ~  
pattern In the heart of the bustling city, hidden among towering skyscrapers and bustling streets, lies a serene park th  
at offers a respite from the hustle and bustle. The lush greenery and chirping birds create a peaceful oasis, inviting v  
isitors to unwind and reconnect with nature. As the sun casts its golden rays through the trees, creating a dappled patt  
ern on the path, it's easy to forget the chaos just beyond the park's borders. It's a reminder that amidst the rush of m  
odern life, there's always a place to find solace and tranquility.  
~  
~  
~
```

然后在 Vim 中输入正则表达式：使用命令： `/pattern`



```
~  
~  
:/pattern|
```

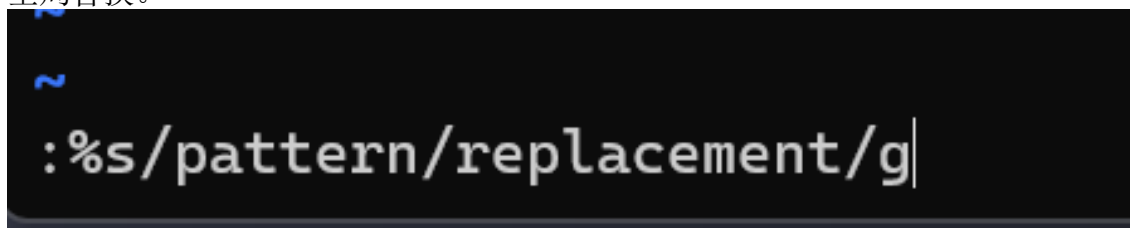
查找结果如下：



```
~  
~  
search hit BOTTOM, continuing at TOP 1,1 All
```

3.2 实例 2：替换

在 Vim 中输入正则表达式：`%s/pattern/replacement/g`。其中，`pattern` 是你要替换的正则表达式模式，`replacement` 是你要替换成的文本，`g` 表示全局替换。



```
~  
~  
:%s/pattern/replacement/g|
```

可看到 `pattern` 被替换成了 `replacement`, 结果如下：

```
replacement In the heart of the bustling city, hidden among towering skyscrapers and bustling streets, lies a serene park that offers a respite from the hustle and bustle. The lush greenery and chirping birds create a peaceful oasis, inviting visitors to unwind and reconnect with nature. As the sun casts its golden rays through the trees, creating a dappled replacement on the path, it's easy to forget the chaos just beyond the park's borders. It's a reminder that amidst the rush of modern life, there's always a place to find solace and tranquility.

:~s/pattern/replacement/g 1,1 All
```