

第二次实验报告

22130011068 郑传奇

2024 年 9 月 5 日

1 shell 实例

1.1 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

ls -a

ls -lh

ls -t

ls --color=auto

组合使用 ls -aht --color=auto

```
takuto@takuto-virtual-machine:~/Desktop$ ls -aht --color=auto
..          test_with_nx  test_fortify3      test_with_protection  last5.py
test        test_no_nx   test_fortify2      test_full_protection  last4.py
.           test_default test_fortify1      test_no_protection.obj last3.py
test_ple2   test1        test_with_protection.obj test_no_protection    last2.py
test_ple    test1.c      test_full_protection.obj test.c                last1.py
takuto@takuto-virtual-machine:~/Desktop$
```

加上 l

```
takuto@takuto-virtual-machine:~/Desktop$ ls -laht --color=auto
total 280K
drwxr-x--- 15 takuto takuto 4.0K 8月 30 09:46 ..
-rwxrwxr-x 1 takuto takuto 16K 6月 22 11:39 test
drwxr-xr-x 2 takuto takuto 4.0K 6月 22 11:39 .
-rwxrwxr-x 1 takuto takuto 16K 6月 22 11:31 test_pie2
-rwxrwxr-x 1 takuto takuto 16K 6月 22 11:31 test_pie
-rwxrwxr-x 1 takuto takuto 16K 6月 22 11:28 test_with_nx
-rwxrwxr-x 1 takuto takuto 16K 6月 22 11:28 test_no_nx
-rwxrwxr-x 1 takuto takuto 16K 6月 22 11:27 test_default
-rwxrwxr-x 1 takuto takuto 16K 6月 22 11:26 test1
-rw-rw-r-- 1 takuto takuto 86 6月 22 11:26 test1.c
-rwxrwxr-x 1 takuto takuto 16K 6月 22 11:18 test_fortify3
-rwxrwxr-x 1 takuto takuto 16K 6月 22 11:17 test_fortify2
-rwxrwxr-x 1 takuto takuto 16K 6月 22 11:16 test_fortify1
-rw-rw-r-- 1 takuto takuto 8.9K 6月 22 11:06 test_with_protection.obj
-rw-rw-r-- 1 takuto takuto 9.4K 6月 22 11:06 test_full_protection.obj
-rwxrwxr-x 1 takuto takuto 16K 6月 22 11:06 test_with_protection
-rwxrwxr-x 1 takuto takuto 16K 6月 22 11:04 test_full_protection
-rw-rw-r-- 1 takuto takuto 8.1K 6月 22 11:02 test_no_protection.obj
-rwxrwxr-x 1 takuto takuto 16K 6月 22 11:02 test_no_protection
```

1.2 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。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。通过 source 来加载函数，随后可以在 bash 中直接使用

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

将 pwd 路径写到 history.log 中

```
takuto@takuto-virtual-machine:~/Desktop$ vim marco.sh
takuto@takuto-virtual-machine:~/Desktop$ source marco.sh
takuto@takuto-virtual-machine:~/Desktop$ marco
saved /home/takuto/Desktop
takuto@takuto-virtual-machine:~/Desktop$ cd
takuto@takuto-virtual-machine:~$ polo
takuto@takuto-virtual-machine:~/Desktop$
```

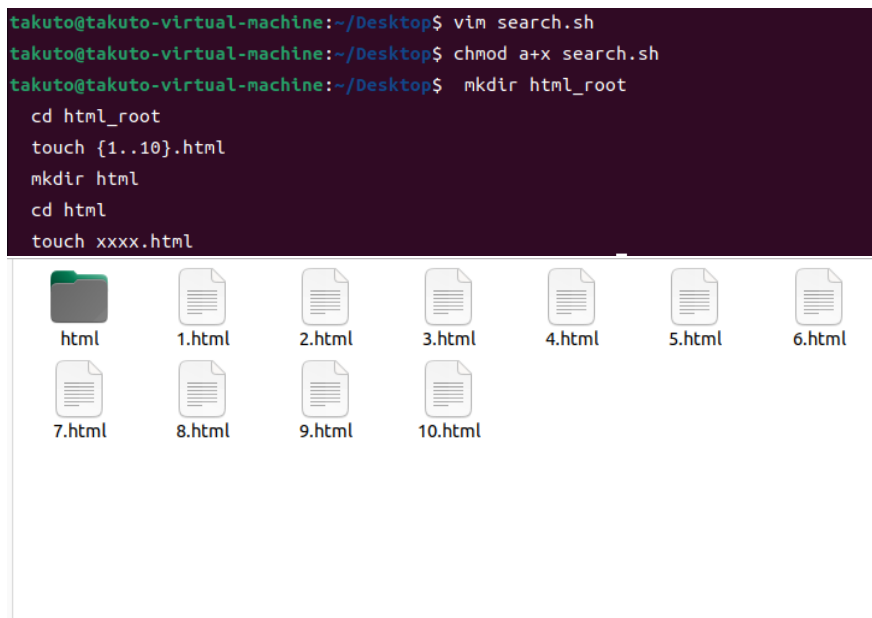
- 1.3 假设您有一个命令，它很少出错。因此为了在出错时能够对其进行调试，需要花费大量的时间重现错误并捕获输出。编写一段 bash 脚本，运行如下的脚本直到它出错，将它的标准输出和标准错误流记录到文件，并在最后输出所有内容。

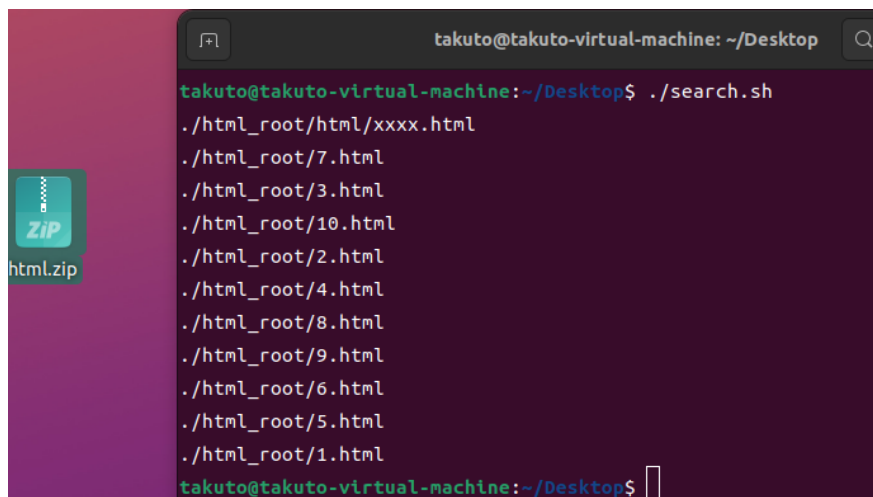
```
#!/usr/bin/env bash
echo > out.log
for ((count=0;;count++))
do
    ./buggy.sh &>> out.log
    if [[ $? -ne 0 ]]; then
        echo "failed after $count times"
        break
    fi
done
```

- 1.4 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

```
find . -type f -name "*.html" | xargs tar -cvzf html.zip
```

使用测试文件

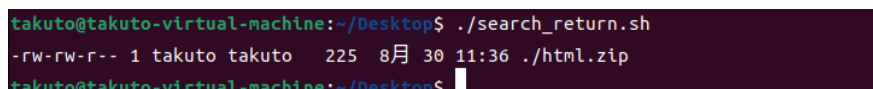




```
takuto@takuto-virtual-machine: ~/Desktop
takuto@takuto-virtual-machine:~/Desktop$ ./search.sh
./html_root/html/xxxx.html
./html_root/7.html
./html_root/3.html
./html_root/10.html
./html_root/2.html
./html_root/4.html
./html_root/8.html
./html_root/9.html
./html_root/6.html
./html_root/5.html
./html_root/1.html
takuto@takuto-virtual-machine:~/Desktop$
```

1.5 (进阶) 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。更通用的做法，你可以按照最近的使用时间列出文件吗？

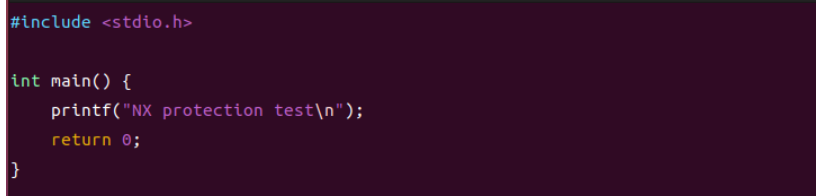
```
find . -type f -print0 | xargs -0 ls -lt | head -1
```



```
takuto@takuto-virtual-machine:~/Desktop$ ./search_return.sh
-rw-rw-r-- 1 takuto takuto 225 8月 30 11:36 ./html.zip
takuto@takuto-virtual-machine:~/Desktop$
```

2 vim 实例

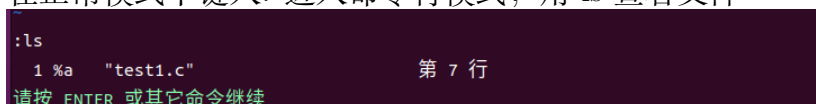
1. 在正常模式，键入 i 进入插入模式。现在 Vim 跟很多其他的编辑器一样，直到你键入 <ESC> 返回正常模式



```
#include <stdio.h>

int main() {
    printf("NX protection test\n");
    return 0;
}
```

2. 在正常模式下键入: 进入命令行模式，用 ls 查看文件



```
:ls
1 %a "test1.c" 第 7 行
请按 ENTER 或其它命令继续
```

3. w (下一个词), b (词初), e (词尾)
4. r: 进入替换模式, v: 进入一般可视化模式, 可以选择文本
5. % (找到配对, 比如括号或者 /* */ 之类的注释对)

在 {使用查找}

```
#include <stdio.h>

int main() {
    printf("NX protection test\n");
    return 0;
}
```

6. 计数来结合“名词”和“动词”

3w 向后移动三个词

```
#include <stdio.h>

int main() {
    printf("NX protection test\n");
    return 0;
}
```

7. 用修饰语改变“名词”的意义

ci(改变当前括号内的内容)

```
#include <stdio.h>

int main() {
    printf();
    return 0;
}
```

8. 搜索和替换

%s/foo/bar/g 在整个文件中将 foo 全局替换成 bar

```
#include <stdio.h>

int main() {
    printf();
    int foo;
    foo=5;
    printf("%d",foo);
    return 0;
}
```

```
#include <stdio.h>

int main() {
    printf();
    int bar;
    bar=5;
    printf("%d",bar);
    return 0;
}
```

9. 多窗口

用:sp / :vsp 来分割窗口, 同一个缓存可以在多窗口中显示。

```
#include <stdio.h>

int main() {
    printf();
    int bar;
    bar=5;
    printf("%d",bar);
    return 0;
}

~
test1.c [+] 7,19 全部
#include <stdio.h>

int main() {
    printf();
    int bar;
    bar=5;
    printf("%d",bar);
    return 0;
}

test1.c [+] 7,19 全部
:sp
```

3 数据整理实例

1. 统计 words 文件 (/usr/share/dict/words) 中包含至少三个 a 且不以's 结尾的单词个数。这些单词中，出现频率前三的末尾两个字母是什么？sed 的 y 命令，或者 tr 程序也许可以帮你解决大小写的问题。共存在多少种词尾两字母组合？

```
cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E ".*(.a)3.*$" | grep -v "'s$" | wc -l
```

```
sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq -c | sort -g | tail -3
```

```
sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq | wc -l
```

```
takuto@takuto-virtual-machine:~/Desktop$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E ".*(.a){3}.*$" | grep -v "'s$" | wc -l
854
```

2. 进行原地替换听上去很有诱惑力，例如：sed s/REGEX/SUBSTITUTION/ input.txt > input.txt。但是这并不是一个明智的做法，为什么呢？还是说只有 sed 是这样的？查看 man sed 来完成这个问题

```
SED(1)                                User Commands                                SED(1)

NAME
    sed - stream editor for filtering and transforming text

SYNOPSIS
    sed [OPTION]... {script-only-if-no-other-script} [input-file]...

DESCRIPTION
    Sed is a stream editor. A stream editor is used to perform basic text
    transformations on an input stream (a file or input from a pipeline).
    While in some ways similar to an editor which permits scripted edits
    (such as ed), sed works by making only one pass over the input(s), and
    is consequently more efficient. But it is sed's ability to filter text
    in a pipeline which particularly distinguishes it from other types of
    editors.

    -n, --quiet, --silent

    suppress automatic printing of pattern space

    --debug
```

> 重定向会将原始的文件覆盖，将 input.txt 的内容清空掉，sed 的替换就没有用了。

3. 找出您最近十次开机的开机时间平均数、中位数和最长时间。在 Linux 上需要用到 journalctl，而在 macOS 上使用 log show。找到每次起到开始和结束时的时间戳

```
takuto@takuto-virtual-machine:~/Desktop$ journalctl -q | grep "Startup finished
in" | sed -E 's/(.*)s\./\1/' | tail -10 | gawk '{print $NF}' | sort -n | paste
-s -d "+" | bc | gawk '{print $1 /10}'
18.6569
takuto@takuto-virtual-machine:~/Desktop$ journalctl -q | grep "Startup finished
in" | sed -E 's/(.*)s\./\1/' | tail -10 | gawk '{print $NF}' | sort -n | paste
-s | gawk '{print ($5 + $6) / 2}'
22.0735
takuto@takuto-virtual-machine:~/Desktop$ journalctl -q | grep "Startup finished
in" | tail -10 | gawk '{print $NF}' | sort -n | tail -1
33.954s.
```