系统开发工具基础 | Shell & Vim & Data wrangling

梁子毅 22090001041 **Instructor:** 周小伟，范浩, **2024 Summer** | 截止时间：Sep 5, 2024

一、 shell

1. 课后练习

问题 **1**： 阅读 **man ls** ，然后使用 **ls** 命令进行如下操作：

1. 所有文件（包括隐藏文件）
2. 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)**
3. 文件以最近访问顺序排序
4. 以彩色文本显示输出结果

1. `ls -a`
2. `ls -l -h`
3. `ls -l -t`
4. `ls --color=auto`

运行如图 1

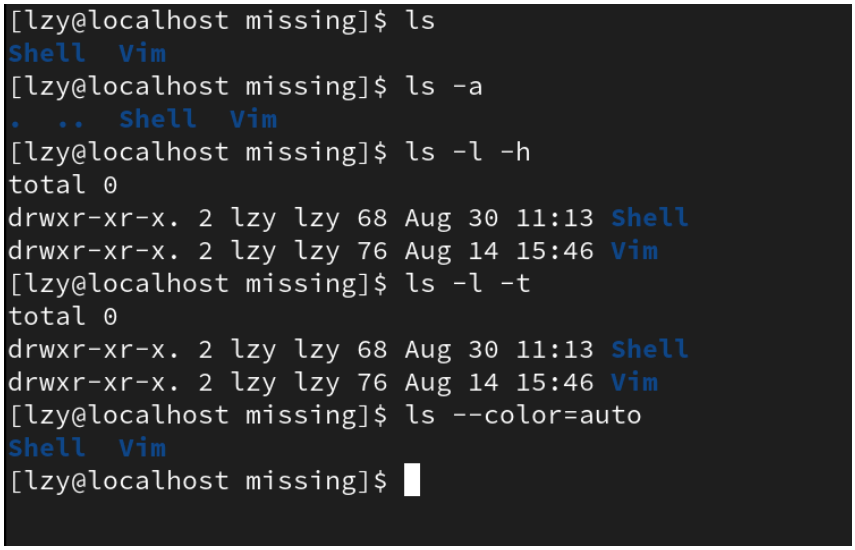

图 1　分别对应上面四个实例

问题 **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。

```bash
#!bin/bash

marco() {
  # 使用 pwd 命令获取当前工作目录的完整路径
  echo "$(pwd)" > $HOME/marco_history.log  # 将路径写入日志文件，覆盖旧内容
  echo "the path $(pwd) has saved"         # 输出保存成功的消息
```

```
}

polo() {
  # 使用 cat 命令显示日志文件中保存的路径
  cat $HOME/marco_history.log
  # 使用 cd 命令切换到日志文件中显示的路径
  # 注意：这里使用了反引号 ` 来执行命令替换，获取日志文件中的路径
  cd `cat $HOME/marco_history.log`
  echo "Done!"  # 输出完成的消息
}
```

运行结果如图 2

```
[lzy@localhost Shell]$ cat marco.sh
#!bin/bash

marco(){
        echo "$(pwd)" > $HOME/marco_history.log
        echo "the path $(pwd) has saved"
}


polo(){
        cat $HOME/marco_history.log
        cd `cat $HOME/marco_history.log`
        echo "Done!"
}
[lzy@localhost Shell]$ source marco.sh
[lzy@localhost Shell]$ marco
the path /home/lzy/Course/missing/Shell has saved
[lzy@localhost Shell]$ cd //
[lzy@localhost //]$ polo
/home/lzy/Course/missing/Shell
Done!
[lzy@localhost Shell]$
```

图 2　对应一个 marco 实例

问题 **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。

```
#!/bin/bash
# 初始化计数器变量
count=0
# 清空 out.log 文件，用于记录输出
echo > out.log
# 无限循环，直到找到错误或手动停止
while true
do
    # 运行 buggy.sh 脚本，并将输出追加到 out.log 文件
    ./buggy.sh &>> out.log

    # 检查上一个命令的退出状态
    if [[ $? -ne 0 ]]; then
        # 如果命令失败，则打印日志文件内容
        cat out.log
        # 打印失败次数
```

```
        echo "failed after $count times"
        # 退出循环
        break
    fi
    # 如果命令成功，则增加计数器
    ((count++)) # 双括号比较醒目
done
```

运行结果如图 3

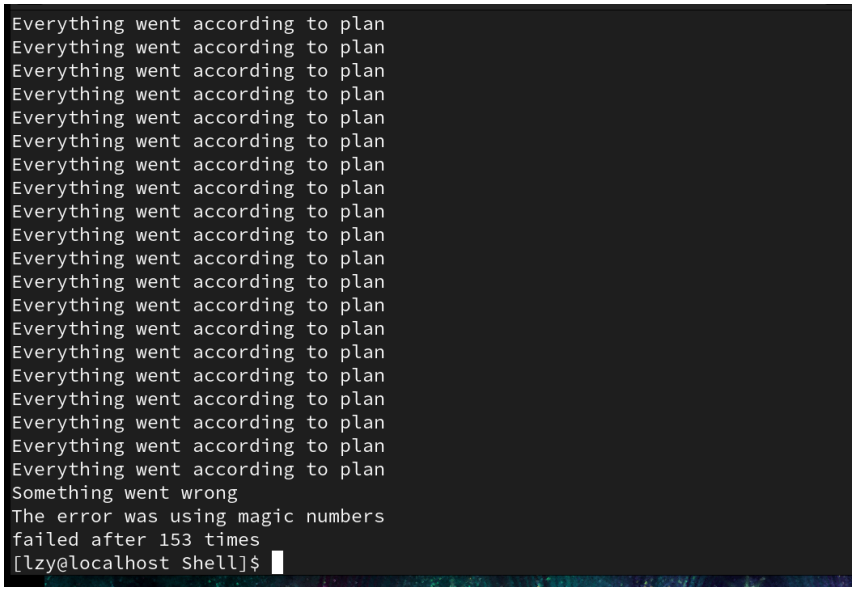

图 3　对应一个 test 实例

问题 **4**：

- 本节课我们讲解的 **find** 命令中的 **-exec** 参数非常强大，它可以对我们查找的文件进行操作。 如果我们要对所有文件进行操作呢？例如创建一个 **zip** 压缩文件？我们已经知道，命令行可以从参数或标准输入接受输入。在用管道连接命令时，我们将标准输出和标准输入连接起来，但是有些命令，例如 **tar** 则需要从参数接受输入。这里我们可以使用 **xargs** 命令，它可以使用标准输入中的内容作为参数。 例如 **ls | xargs rm** 会删除当前目录中的所有文件。您的任务是编写一个命令，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（提示：查看 **xargs** 的参数**-d**）译注：**MacOS** 上的 **xargs** 没有**-d**，查看这个 **issue**
- 如果您使用的是 **MacOS**，请注意默认的 **BSD find** 与 **GNU coreutils** 中的是不一样的。你可以为 **find** 添加**-print0** 选项，并为 **xargs** 添加**-0** 选项。作为 **Mac** 用户，您需要注意 **mac** 系统自带的命令行工具和 **GNU** 中对应的工具是有区别的；如果你想使用 **GNU** 版本的工具，也可以使用 **brew** 来安装。

1. 通过如图 4 代码创建所需文件
2. 在输入下面的一行命令

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

- 使用 `find` 命令从当前目录开始搜索所有 `.test` 文件
- 用 `xargs` 命令处理管道来的 `find` 命令的输出
- `-d '\n'` 选项告诉 `xargs` 使用换行符作为分隔符，这样每个文件名都会被单独处理

- 调用 `tar` 命令创建一个名为 `test.zip` 的压缩文件
  - `-c` 选项表示创建一个新的压缩文件
  - `-v` 选项表示在压缩过程中显示详细信息
  - `-z` 选项表示使用 `gzip` 压缩

```
[lzy@localhost Shell]$ mkdir folder
[lzy@localhost Shell]$ cd folder/
[lzy@localhost folder]$ touch {1..6}.test
[lzy@localhost folder]$ mkdir test
[lzy@localhost folder]$ cd test
[lzy@localhost test]$ touch test.test
```

图 4　操作实例 1

```
[lzy@localhost folder]$ find . -type f -name "*.test" | xargs -d '\n' tar -cvzf
test.zip
./1.test
./2.test
./3.test
./4.test
./5.test
./6.test
./test/test.test
```

图 5　操作实例 2

二、 Vim

问题 **5**： 下载课程提供的 **vimrc**，然后把它保存到 `~/.vimrc`。 通读这个注释详细的文件 （用 **Vim!**）， 然后观察 **Vim** 在这个新的设置下看起来和使用起来有哪些细微的区别。

- 文档如下

```
" Comments in Vimscript start with a `"`.

" If you open this file in Vim, it'll be syntax highlighted for you.

" Vim is based on Vi. Setting `nocompatible` switches from the default
" Vi-compatibility mode and enables useful Vim functionality. This
" configuration option turns out not to be necessary for the file named
" '~/.vimrc', because Vim automatically enters nocompatible mode if that file
" is present. But we're including it here just in case this config file is
" loaded some other way (e.g. saved as `foo`, and then Vim started with
" `vim -u foo`).
set nocompatible

" Turn on syntax highlighting.
syntax on

" Disable the default Vim startup message.
set shortmess+=I

" Show line numbers.
set number

" This enables relative line numbering mode. With both number and
" relativenumber enabled, the current line shows the true line number, while
" all other lines (above and below) are numbered relative to the current line.
" This is useful because you can tell, at a glance, what count is needed to
" jump up or down to a particular line, by {count}k to go up or {count}j to go
```

```
" down.
set relativenumber

" Always show the status line at the bottom, even if you only have one window open.
set laststatus=2

" The backspace key has slightly unintuitive behavior by default. For example,
" by default, you can't backspace before the insertion point set with 'i'.
" This configuration makes backspace behave more reasonably, in that you can
" backspace over anything.
set backspace=indent,eol,start

" By default, Vim doesn't let you hide a buffer (i.e. have a buffer that isn't
" shown in any window) that has unsaved changes. This is to prevent you from "
" forgetting about unsaved changes and then quitting e.g. via `:qa!`. We find
" hidden buffers helpful enough to disable this protection. See `:help hidden`
" for more information on this.
set hidden

" This setting makes search case-insensitive when all characters in the string
" being searched are lowercase. However, the search becomes case-sensitive if
" it contains any capital letters. This makes searching more convenient.
set ignorecase
set smartcase

" Enable searching as you type, rather than waiting till you press enter.
set incsearch

" Unbind some useless/annoying default key bindings.
nmap Q <Nop> " 'Q' in normal mode enters Ex mode. You almost never want this.

" Disable audible bell because it's annoying.
set noerrorbells visualbell t_vb=

" Enable mouse support. You should avoid relying on this too much, but it can
" sometimes be convenient.
set mouse+=a

" Try to prevent bad habits like using the arrow keys for movement. This is
" not the only possible bad habit. For example, holding down the h/j/k/l keys
" for movement, rather than using more efficient movement commands, is also a
" bad habit. The former is enforceable through a .vimrc, while we don't know
" how to prevent the latter.
" Do this in normal mode...
nnoremap <Left>  :echoe "Use h"<CR>
nnoremap <Right> :echoe "Use l"<CR>
nnoremap <Up>    :echoe "Use k"<CR>
nnoremap <Down>  :echoe "Use j"<CR>
" ...and in insert mode
inoremap <Left>  <ESC>:echoe "Use h"<CR>
inoremap <Right> <ESC>:echoe "Use l"<CR>
inoremap <Up>    <ESC>:echoe "Use k"<CR>
inoremap <Down>  <ESC>:echoe "Use j"<CR>

```

问题 **6**：

- 安装和配置一个插件： **ctrlp.vim.**
  1. 用 **mkdir -p /.vim/pack/vendor/start** 创建插件文件夹
  2. 下载这个插件： **cd /.vim/pack/vendor/start; git clone https://github.com/ctrlpvim/ctrlp.vim**

3. 阅读这个插件的 文档。 尝试用 **CtrlP** 来在一个工程文件夹里定位一个文件，打开 **Vim**, 然后用 **Vim** 命令控制行开始 `:CtrlP`.
4. 自定义 `CtrlP:` 添加 **configuration** 到你的 `~/.vimrc` 来用按 **Ctrl-P** 打开 **CtrlP**

- 运行结果如图 6

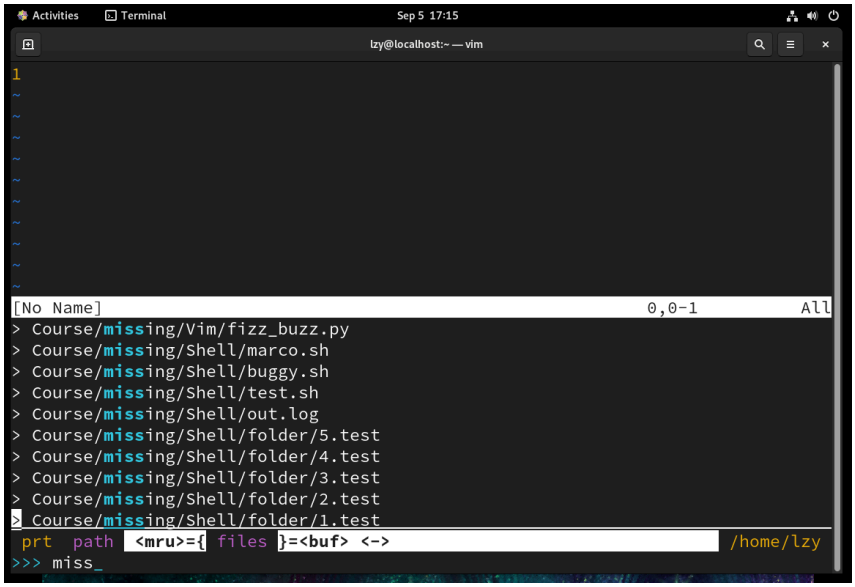

图 6　操作实例

问题 **7**：如何在 **Vim** 中保存文件？

- 按 `:w` 保存文件。
- 按 `:wq` 保存并退出 Vim。
- 按 `:x` 也是保存并退出的快捷方式。
- 按 `:w <filename>` 可以将文件另存为指定的文件名。

问题 **8**：如何在 **Vim** 中复制和粘贴文本？

- 按 `v` 进入字符可视模式，选择文本，然后按 `y` 复制。
- 将光标移动到目标位置，按 `p` 粘贴到光标后，或按 `shift+p` 粘贴到光标前。

问题 **9**：如何在 **Vim** 中搜索和替换文本？

- 按 / 后输入搜索的文本，按 Enter 进行搜索。
- 按 :%s/old/new/g 替换文件中所有的 “old” 为 “new"。
- 按 :%s/old/new/gc 替换文件中所有的 “old” 为 “new"，并且每次替换前会进行确认。
- 替换之前图 7
- 替换之后图 8

图 7　操作实例

图 8　操作实例

问题 **10**：如何在 **Vim** 中分割窗口？

- 按 :split 或 :sp 垂直分割窗口。
- 按 :vsplit 或 :vsp 水平分割窗口。

问题 **11**：如何在 **Vim** 中撤销分割的窗口？

- 按 :close 或 :q 关闭当前窗口

三、 Data wrangling

1. 课后练习

问题 **12**：

- 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？
- 还有一个很 有挑战性的问题：哪个组合从未出现过？

1. `cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^([^a]?*a){3}.*$" | grep -v "'s$"|wc -l`
   - tr xxx 用于将大写字母转换为小写字母
   - `grep -E "^([^a]?*a){3}.*$"` 意为：匹配前面是不是 a 都可以含有 a 的组三次
   - grep -v

2. 自行发挥

问题 **13**： 如何删除文件中的空行？

- 使用 `sed` 命令： `sed '/^$/d' filename > newfile` 会删除文件中的所有空行，并将结果输出到 newfile。如果要直接修改原文件，可以使用 `sed -i '/^$/d' filename` 。
- 使用 `awk` 命令： `awk NF filename > newfile` 也会删除空行，并将结果输出到 newfile。
- 使用 `grep` 命令： `grep -v '^$' filename > newfile` 删除空行，并将结果输出到 newfile。

问题 **14**： 如何删除文件中的重复行

- 使用 uniq 命令：sort filename | uniq > newfile 首先对文件进行排序，然后使用 uniq 删除重复行，结果输出到 newfile。

问题 **15**： 如何统计文件中特定单词的出现次数？

- 使用 grep 和 wc 命令：grep -o “word” filename | wc -l 可以统计特定单词出现的次数。

问题 **16**： 如何合并多个 **CSV** 文件？

- 使用 `cat` 命令： `cat file1.csv file2.csv > merged.csv` 可以将两个 CSV 文件合并成一个。

四、 实例计算

- 16 个问题，其中第一个问题 4 个实例，一共 20 个实例

五、 总结

1. Shell
   1. 通过编写简单的脚本来自动化日常任务，能够处理文件和目录。
   2. 学会了使用 ls 命令的高级选项来显示文件的详细信息
   3. 使用 find 和 xargs 组合来查找和压缩特定类型的文件。
2. Vim
   1. 学了如何自定义 Vim 的配置文件 /.vimrc
   2. 使用 Vim 插件如 ctrlp.vim 来增强编辑体验
   3. 使用 Vim 的快捷键进行文本编辑
3. 数据整理
   1. 使用 sed、awk、grep 和 uniq 来处理文本数据，包括删除空行、去除重复行、统计单词出现次数等
4. 这次比上次要难（很有可能是因为三节课被合成了一节课），基本上最难的应该是后面数据处理的内容，正则化表达式挺复杂，比较生涩
5. vim 主要是熟练度的问题，平时多使用就能有所长进，多想想如何简化操作流程
6. shell 是一些语法要记住，可以用 shell 来简化一些流程从而让工作过程变得简单，比如这次的检测多少次运行之后出现漏洞的问题

7. 在此附上 GitHub 链接和 commit 记录（上次忘了，这次一起附上）如图 9

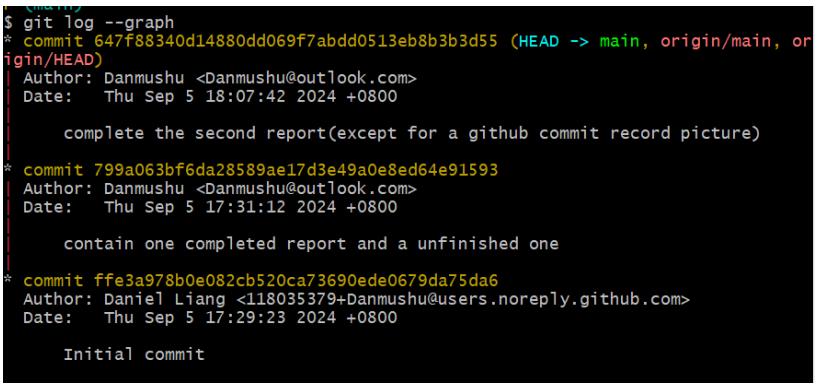

图 9　操作实例