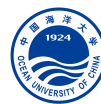


海纳百川  
取则行远



中国海洋大学  
OCEAN UNIVERSITY OF CHINA

# Week2

左昊天

2024-08-30

Github 仓库地址

# 1 Shell

## 1.1 新建文件

### 1.1.1 创建任意类型的文件：

```
1 touch 文件名
```

### 1.1.2 创建可写入内容的文件：

```
1 echo "要写入的内容">test.txt
```

```
19355@-:~$ cd /mnt/c/Users/19355/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ touch create.txt

19355@-:~$ cd /mnt/c/Users/19355/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ echo "test">create2.txt
```

create.txt	2024-08-30 10:21	文本文档	0 KB
create2.txt	2024-08-30 10:21	文本文档	1 KB

## 1.2 新建文件夹

### 1.2.1 新建单个文件夹：

```
1 mkdir 文件夹名
```

### 1.2.2 新建多个文件夹：

```
1 mkdir 文件夹名1 文件夹名2 文件夹名3
```

```
19355@-:~$ cd /mnt/c/Users/19355/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ mkdir 1 2 3
```

... 系统开发工具基础 > 1 > tool\_class\_2024\_sum > week2 > 在 week2 中搜索

排序

查看

名称	修改日期	类型	大小
1	2024-08-30 10:48	文件夹	
2	2024-08-30 10:48	文件夹	
3	2024-08-30 10:48	文件夹	

1

### 1.2.3 新建多级文件夹:

```
1 mkdir -p 文件夹名1/文件夹名2/文件夹名3
```

```
19355@- 9 MINGW64 ~/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2/1 (main)
$ mkdir -p 1/2/3

week2 > 1 > 1 > 2 > 3
```

### 1.3 将命令一行一行写入脚本文件并执行

用 echo 逐行添加命令，再用 ./semester 执行文件。

注: > 会覆盖文件内容，» 用于在文件末尾追加内容

```
19355@- 9 MINGW64 ~/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ echo '#!/bin/sh' > semester

19355@- 9 MINGW64 ~/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ echo curl --head --silent https://missing.csail.mit.edu >> semester

19355@- 9 MINGW64 ~/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ ./semester
HTTP/2 200
server: GitHub.com
content-type: text/html; charset=utf-8
last-modified: Thu, 08 Aug 2024 20:16:01 GMT
access-control-allow-origin: *
etag: "66b52781-205d"
expires: Mon, 26 Aug 2024 19:04:44 GMT
cache-control: max-age=600
x-proxy-cache: MISS
x-github-request-id: 4A7C:81A35:72A77:769D8:66CCCF74
accept-ranges: bytes
age: 0
date: Fri, 30 Aug 2024 03:10:17 GMT
via: 1.1 varnish
x-served-by: cache-nrt-rjtf7700053-NRT
x-cache: HIT
x-cache-hits: 0
```

### 1.4 使用 | 和 >，将 semester 文件输出的最后更改日期信息，写入当前目录下的 last-modified.txt 的文件中

#### 1.4.1 管道操作符 |

| 用于将左边的命令的输出内容传递给右边的命令

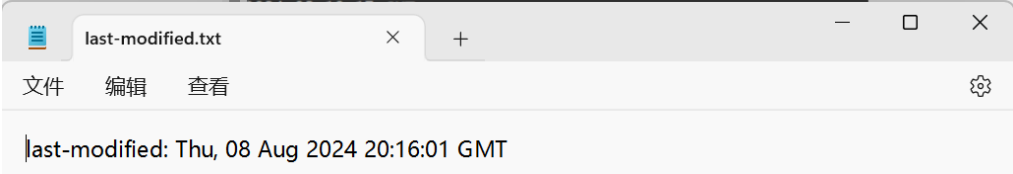
#### 1.4.2 行过滤工具 grep

grep 用于查找符合条件的行并输出

grep last-modified 可以查找有 last-modified 字样的行并输出

### 1.4.3 实现

```
19355@Mingw64: ~/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ ./semester | grep last-modified > last-modified.txt
```



## 1.5 输入输出

read 变量名 可将用户输入内容赋值给变量

echo 输出的字符串中用\$变量名表示变量

```
#!/bin/bash
echo "请输入用户名:"
read name
echo "您好, $name"
```

echo -n 为不换行输出

示例:

```
1      #!/bin/bash
2
3      echo "请输入用户名:"
4      read name
5      echo -n "您好, $name"
6      echo "!!!"
```

```
$ ./test.sh
请输入用户名:
eric
您好, eric!!!
```

遇到的问题:

一开始会报错

```
$ ./test.sh
请输入用户名:
eric
./test.sh: line 5: echo您好, eric: command not found
```

原因: shell 脚本中, 命令和参数之间必须有空格。即 echo 和"输出的内容"间需有空格。

## 1.6 循环

### 1.6.1 for 循环

写法一:

```
1      for (( 表达式1;表达式2;表达式3 )) ; do
2          命令
3      done
```

写法二:

```
1  for i in {1..10}
2  do echo $i
3  done
```

```
$ ./for.sh
0
1
2
3
4
5
6
7
8
9
10
```

### 1.6.2 while 循环

```
1      while [[ $变量1 -lt $变量2 ]] do
2          命令
3      done
```

shell 中比较数值的操作符:

操作符	含义
-eq	等于
-ne	不等于
-ge	大于等于
-le	小于等于
-gt	大于
-lt	小于

## 1.7 猜字游戏

```
1      #!/bin/bash
2
3      number=$(shuf -i 1-10 -n 1)
```

```
4     while [[ $num -ne $number ]] do
5         echo "请输入1-10之间的数字"
6         read num
7         if [[ $num -eq $number ]];then
8             echo "恭喜你猜对了"
9         elif [[ $num -lt $number ]];then
10            echo "猜小了"
11        else
12            echo "猜大了"
13        fi
14    done
```

```
$ ./game.sh
请输入1-10之间的数字
5
猜大了
请输入1-10之间的数字
2
猜小了
请输入1-10之间的数字
4
恭喜你猜对了
```

1.8 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。

代码：

```
1     #!/bin/bash
2
3     marco() {
4         echo "$(pwd)" > C:/Users/19355/Desktop/大二暑假/
5             系统开发工具基础/1/tool_class_2024_sum/week2/
6             history.log
7         echo "已保存目录"
8     }
9
10    polo() {
11        cd "$(cat C:/Users/19355/Desktop/大二暑假/系统
12            开发工具基础/1/tool_class_2024_sum/week2/
```

```
        history.log)"
10 }
```

解释：

- 1、\$( )为命令替换符，可将内容替换为括号中命令执行后的内容。
- 2、cat 文件地址 用于打开文件并输出文件内容

运行截图：



2 Vim

2.1 vim 中的不同模式以及进入方式

模式	进入方式
normal	vim 打开文件后默认进入的模式
insert	在 normal 下有多重方式进入。如输入 i, 在光标之前开始输入
	a 光标之后开始输入
	o 下方插入一行开始输入
	s 删除当前光标字符，开始输入
	I 从本行开头输入
	A 从本行结尾输入
	O 上方插入一行开始输入
command	S 删除当前行并输入
command	在 normal 下输入: 进入
visual	在 normal 下输入 v 进入

## 2.2 退出 vim 的方式

在 normal 模式下输入:wq 为保存并退出

:q为直接退出

:q!为强制退出

## 2.3 搜索

在命令行模式下，用 /要搜索的单词 可以使光标跳到该单词前面

## 2.4 cw

在 normal 模式下，输入cw可以删除光标后的一个单词并进入 insert 模式。

## 2.5 替换

(在命令行模式下输入)

格式:

```
1 {作用范围}s/{目标}/{替换}/{替换标志}
```

如:

```
1 %s/sort/array/g
```

作用范围:

输入内容	替换方式
%	全局替换
. 或不输入	当前行替换
行号 1, 行号 2	指定行替换 (从行号 1 到行号 2)

注: 在 visual 模式下输入: 后会自动输入 '<,>' 它表示作用范围是当前选定的内容

## 2.6 跳转行

行数j 用于光标向下多少行, 如 5j 表示光标向下 5 行

行数k 用于光标向上多少行

## 2.7 重复操作命令与撤销操作命令

重复操作: 在 normal 模式下输入 .

撤销操作: 在 normal 模式下输入 u



## 2.8 通过 vimrc 文件自定义 vim

```
1 syntax on %打开语法高亮
2 set number %显示行号
3 set relativenumber %开启相对行号
```

```
3 Life is too short to spend time with people who suck the happiness out of you. If someone wants you in their life, they'll make room for you. You should n't have to fight for a spot. Never, ever insist yourself to someone who continuously overlooks your worth.
2 And remember, it's not the people that stand by your side when you're at your best, but the ones who stand beside you when you're at your worst that are your true friends.
1 In the flood of darkness, hope is the light. It brings comfort, faith, and confidence. It gives us guidance when we are lost, and gives support when we are afraid. And the moment we give up hope, we give up our lives. The world we live in is disintegrating into a place of malice and hatred, where we need hope and find it harder. In this world of fear, hope to find better, but easier said than done, the more meaningful life of faith will make life meaningful.
4 Only when you understand the true meaning of life can you live truly. Bitter sweet as life is, it's still wonderful, and it's fascinating even in tragedy. If you're just alive, try harder and try to live wonderfully.
1 I believe there is a person who brings sunshine into your life. That person may have enough to spread around. But if you really have to wait for someone to bring you the sun and give you a good feeling, then you may have to wait a long time.
```

开启相对行号后，vim 会自动计算其他行与当前行的相对行差，方便在跳转时直接输入移动的行数。

## 3 数据整理

### 3.1 流编辑器 sed

（行过滤工具 grep 已在 1.4.2 涉及）

sed 常用于替换文字

```
1 echo 'aba' | sed 's/[ab]//'
```

表示将 aba 传给 sed，并且 sed 将第一个是 a 或 b 的字母替换成空。

```
$ echo 'aba' | sed 's/[ab]//'  
ba
```

注：默认只匹配第一个符合条件的字符。如果要匹配全部字符，需要在最后加上 g

```
1 echo 'abacd' | sed 's/[ab]//g'
```

```
$ echo 'abacd' | sed 's/[ab]//g'  
cd
```

### 3.2 () 和 |

```
1      echo 'abacbcab' | sed 's/(ab|bc)//g'
```

()中的内容为捕获组，将括号内视为一个整体处理。

| 表示“或”

所以上述代码中 sed 进行的操作是匹配所有的 ab, bc 并将其改为空。

遇到的问题：无法识别 sed 中这个正则表达式的语句

原因：sed 过于老旧，所以只支持简单的正则表达式，如果想要它支持别的正则表达式，需要在 sed 后加上 -E

```
19355@MINGW64 ~/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ echo 'abacbcab' | sed 's/(ab|bc)//g'
abacbcab

19355@MINGW64 ~/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ echo 'abacbcab' | sed -E 's/(ab|bc)//g'
ac
```

### 3.3 \*（星号）和 +（加号）

#### 3.3.1 \*（星号）

\* 表示前面的元素可以重复 0 次或多次, (ab)\* 表示可以是 ab, 也可以是 ababab。

但也只匹配一次!

```
19355@MINGW64 ~/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ echo 'ababbcabbcab' | sed -E 's/(ab)*/g'
bcbc

19355@MINGW64 ~/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ ^C

19355@MINGW64 ~/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ echo 'ababbcabbcab' | sed -E 's/(ab)//g'
bcbc

19355@MINGW64 ~/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ echo 'ababbcabbcab' | sed -E 's/(ab)*/g'
bcabbcab
```

#### 3.3.2 +（加号）

+ 表示前面的元素可以重复 1 次或多次。

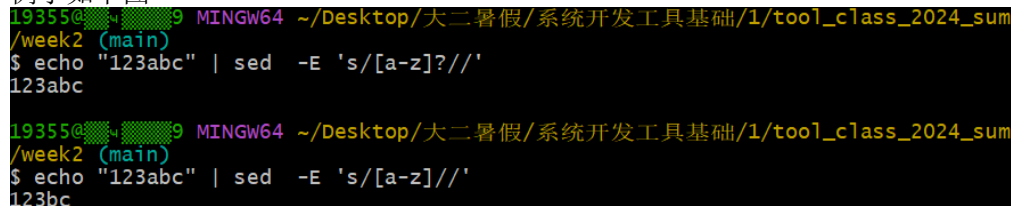
### 3.4 .（点）

. 在正则表达式中代表任意字符, a.b 表示可以是 axb,aab,abb,acb 以此类推。

注：如果真的想表达“点”这个标点符号，需要用\.

### 3.5 ? (问号)

? 表示前面的元素可以出现 0 次或 1 次，与其他正则表达式符号共用可起到非贪婪量词的作用。如 `[a-z]?` 只匹配一个或 0 个字母，而 `[a-z]` 会匹配一个字母。具体例子如下图



```
19355@MINGW64 ~/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ echo "123abc" | sed -E 's/[a-z]?//'
123abc

19355@MINGW64 ~/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ echo "123abc" | sed -E 's/[a-z]//'
123bc
```

### 3.6 统计 words 文件 (使用 AI 生成了 300 个左右的单词) 中包含至少两个 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 1 个字母是什么？

#### 3.6.1 第一问：

```
1 cat words.txt | grep -E "^(^[a]*a){2}.*$" | grep -v "s$" | wc -l
```

解释：

- 1、第一个 `grep` 中的第一个 `^` 表示字符串的开始，`$` 表示字符串的结束。
- 2、`[]` 用于指定字符集合，如果 `[]` 中第一个字符是 `^` 则表示排除这个字符集合。
- 3、`(^[a]*a){2}` 表示开头不是 a 可以重复 0 或多次并且后面跟着 a，且这个捕获组需要重复 2 次。
- 4、`grep -v "s$"` 表示筛选出末尾不是's 的单词，`-v` 表示取反，即将后面的筛选逻辑取反。
- 5、`wc -l` 为计算行数。

```
$ cat words.txt | grep -E "^[^a]*a){2}.*$" | grep -v "'s$" | wc -l
27

19355@- 9 MINGW64 ~/Desktop/大二暑假/系统开发工具基础/1/tool_class_2024_sum/week2 (main)
$ cat words.txt | grep -E "^[^a]*a){2}.*$" | grep -v "'s$"
paradise
basketball
camera
kangaroo
language
kangaroo
language
kangaroo
basketball
kangaroo
basketball
paradise
imagination
karma
nostalgia
aura
sarcasm
alacrity
annoyance
agitation
exasperation
embarrassment
awkwardness
alarm
agitation
heartache
heartbreak
```

### 3.6.2 第二问:

```
1      cat words.txt | grep -E "^[^a]*a){2}.*$" | grep -v
      "'s$" | sed -E "s/.*([a-z])$/\1/" | sort | uniq -
      c | sort | tail -n3
```

解释:

- 1、sed -E "s/.\*([a-z])\$/\1/"表示将以下内容替换为捕获组: 前面是任意字符, 最后是 26 个字母。捕获组为单词结尾的字母。
- 2、第一个 sort 表示对字母进行排序, 为后续使用uniq -c 删除在一起的重复行做准备。
- 3、第二个 sort 对字母出现次数进行排序。
- 4、tail -n3 表示显示最后三行内容 (sort 默认升序)。

```
$ cat words.txt | grep -E "^[^a]*a){2}.*$" | grep -v "'s$" | sed -E "s/.*([a-z])$/\1/" | sort | uniq -c | sort | tail -n3
4 n
4 o
6 e
```

### 3.7 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?

不只是 `sed` 是这样的, 是所有这样操作的都会产生问题。

原因是 `input.txt` 会首先被清空, 再执行相关的操作

**解决方法:**

在 `sed` 后添加 `-i` 可以将命令改为直接修改文件, 而有些系统需要使用 `-i.bak` 先添加原文件的备份, 再进行修改。

```
1      sed -i.bak s/REGEX/SUBSTITUTION/ input.txt > input.
      txt
```