

实验报告

实验名称 : 实验 6

实验日期 : 2023/11/21

姓 名 : 吴泽同

学 号 : 084622109

班 级 : 计算机类 221

人工智能与信息技术学院

南京中医药大学

实验目的:

在 **Shell** 编程中, 掌握流程控制语句和函数的创建与调用是非常重要的。通过使用条件判断、循环控制和错误处理等流程控制语句, 我们可以根据不同的条件和异常情况来执行相应的操作。而函数的创建与调用则能帮助我们封装重复代码, 提高代码的复用性和可维护性。

条件判断语句可以通过测试特定的条件来决定执行的代码块。例如, **if** 语句可以根据条件是否为真来执行相应的代码块。循环控制语句则可以用于重复执行一段代码, 直到满足特定的条件。**while** 语句和 **for** 语句是常见的循环控制语句, 它们可以在满足条件时重复执行代码块, 直到条件不再满足。错误处理语句则能帮助我们处理脚本运行中可能出现的异常情况, 例如 **trap** 语句可以在脚本运行期间捕获指定的信号并执行相应的操作。

在 **Shell** 中, 函数可以使用 **function** 关键字进行定义, 例如: ``function myFunction() { echo "Hello, World!"; }``。定义函数时需要注意函数名称、参数列表和函数体之间的匹配。在 **Shell** 中, 可以使用 ``$1``、``$2`` 等变量来获取函数的参数值。调用函数时可以使用函数名称加上参数列表的方式, 例如: ``myFunction "arg1" "arg2"``。

除了定义和调用函数, 我们还可以在 **Shell** 中使用变量的作用域和命令替换等技术来增强脚本的功能。变量的作用域决定了变量在脚本中的可见性和可访问性。局部变量在函数内部定义, 只能在函数内部使用; 全局变量则在脚本中任何位置都可以定义和使用。命令替换是通过反引号或 ``$()`` 来执行命令并将其输出保存到变量中的技术。例如, ``myVar=$(ls)`` 将执行 **ls** 命令并将其输出保存到 **myVar** 变量中。

实验内容和要求

完成 ppt 最后的 8 个流程控制的练习题

综合练习:

使用 **root** 权限, 新建一个用户 **newuser**

创建一个 **Linux** 用户管理脚本, 输入一个用户名, 脚本判断系统是否存在该用户, 如果存在则打印用户名, 如果不存在, 则新建这个用户, 并新建对应的密码例如, **sh UserCheck.sh newuser**, 打印 **"user exist"**

例如, **sh UserCheck.sh otheruser**, 打印 **"user not exist, add this user"**, 并输入密码补全代码, 完成脚本

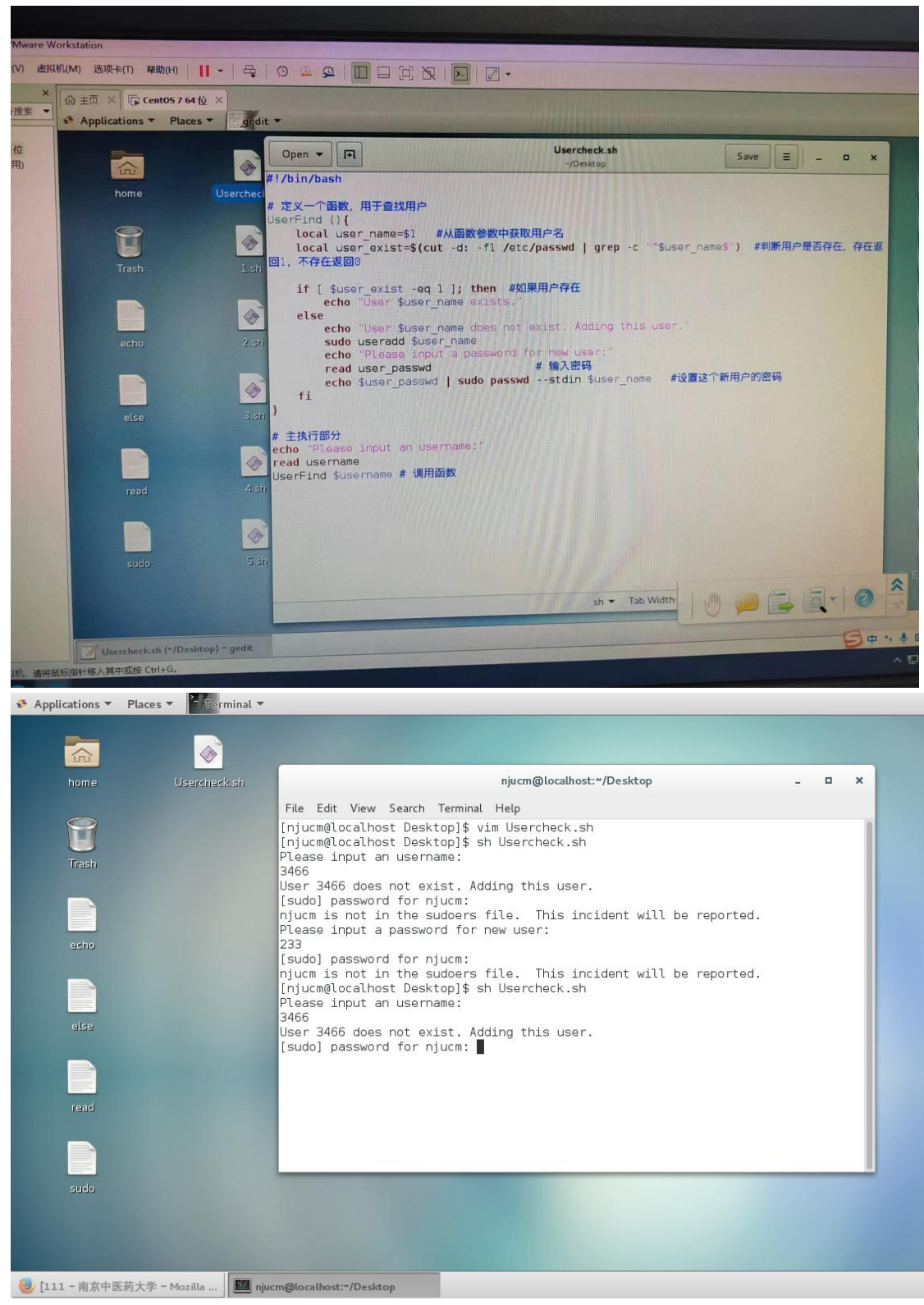
实验结果 (含关键步骤截图)

关键步骤截图可能包括：

编辑脚本文件 **UserCheck.sh**

运行脚本并输入已存在用户的名称，观察输出结果

运行脚本并输入新用户的名称，观察输出结果并输入密码（可以使用 **passwd** 命令）



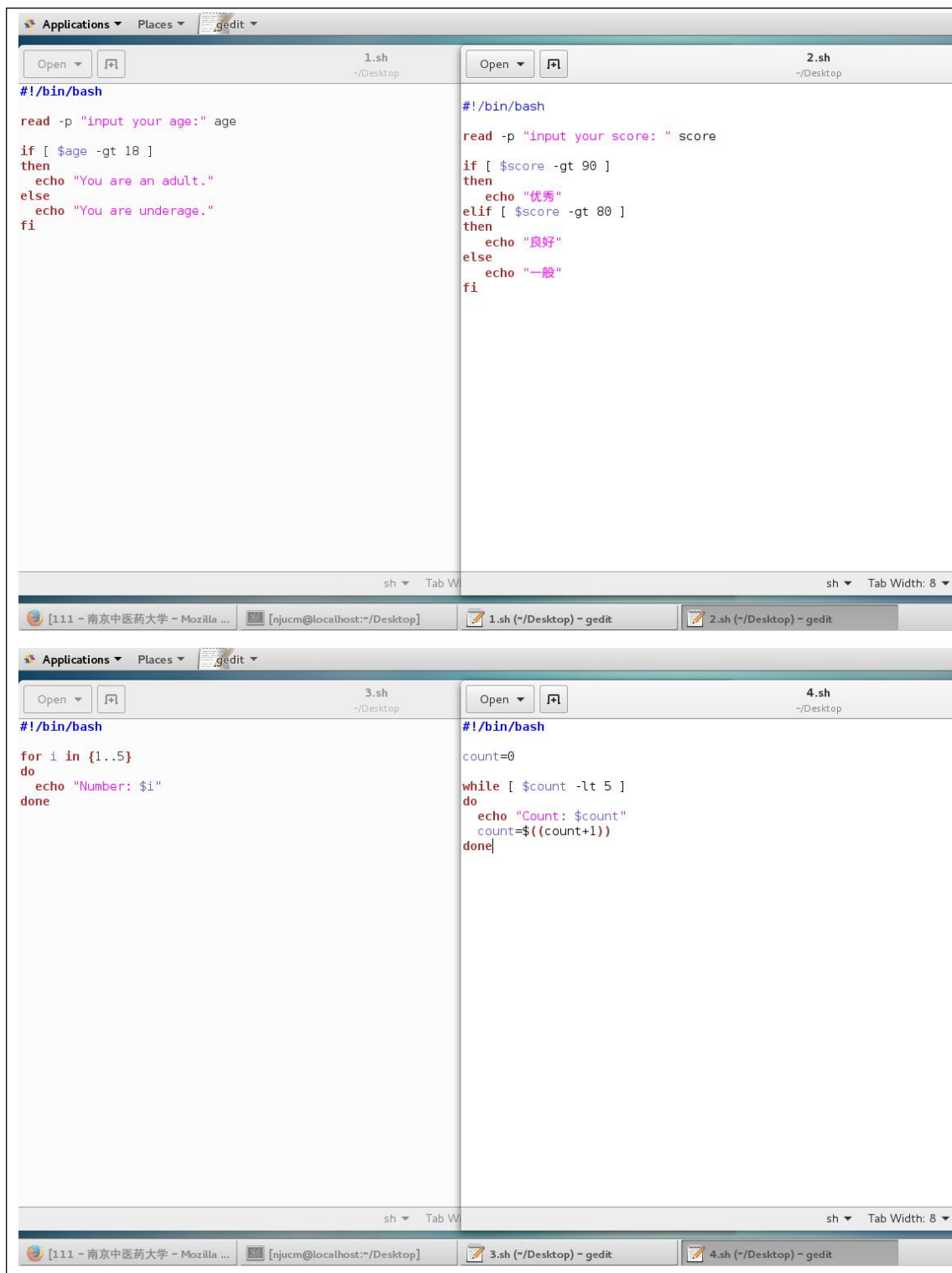
```
Applications ▾ Places ▾ Terminal ▾
njucm@localhost:~/Desktop

File Edit View Search Terminal Help
[njucm@localhost Desktop]$ vim 1.sh
[njucm@localhost Desktop]$ sh 1.sh
input your age:33
1.sh: line 5: syntax error near unexpected token `then'
1.sh: line 5: `if[$age -ge 18];then'
[njucm@localhost Desktop]$ vim 1.sh
[njucm@localhost Desktop]$ vim 1.sh
[njucm@localhost Desktop]$ vim 1.sh
[njucm@localhost Desktop]$ sh 1.sh
input your age:6
1.sh: line 5: if[6 -ge 18]: command not found
1.sh: line 6: syntax error near unexpected token `then'
1.sh: line 6: `then'
[njucm@localhost Desktop]$ vim 1.sh
[njucm@localhost Desktop]$ sh 1.sh
input your age:8
1.sh: line 5: syntax error near unexpected token `then'
1.sh: line 5: `if[ $age -gt 18 ]; then'
[njucm@localhost Desktop]$ vim 1.sh
[njucm@localhost Desktop]$ sh 1.sh
input your age:6
You are underage.
[njucm@localhost Desktop]$ sh 1.sh
input your age:78
You are an adult.
[njucm@localhost Desktop]$ vim 2.sh
[njucm@localhost Desktop]$ sh 2.sh
input your score: 78
一般
[njucm@localhost Desktop]$ vim 3.sh
[njucm@localhost Desktop]$ sh 3.sh
Number: 1
Number: 2
Number: 3


111 - 南京中医药大学 - Mozilla F... njucm@localhost:~/Desktop

Applications ▾ Places ▾ Terminal ▾
njucm@localhost:~/Desktop

File Edit View Search Terminal Help
Count: 2
Count: 3
Count: 4
[njucm@localhost Desktop]$ vim 5.sh
[njucm@localhost Desktop]$ sh 5.sh
输入 1到4之间的数字 :
3
你选择了 3
[njucm@localhost Desktop]$ vim 6.sh
[njucm@localhost Desktop]$ sh 6.sh
Count: 0
Count: 1
Count: 2
Count: 3
Count: 4
[njucm@localhost Desktop]$ vim 7.sh
[njucm@localhost Desktop]$ sh 7.sh
Number:1
Number:2
Number:3
Number:4
[njucm@localhost Desktop]$ vim 8.sh
[njucm@localhost Desktop]$ sh 8.sh
Number:1
Number:2
Number:3
Number:4
Number:5
Number:6
Number:7
Number:8
Number:9
Number:10
[njucm@localhost Desktop]$
```



Applications ▾ Places ▾ gedit ▾

Open ▾ 

5.sh
~/Desktop


```
#!/bin/bash

echo "输入1到4之间的数字："
read number

case $number in
  1) echo "你选择了1" ;;
  2) echo "你选择了2" ;;
  3) echo "你选择了3" ;;
  4) echo "你选择了4" ;;
  *) echo "你没有输入1到4之间的数字" ;;
esac
```

sh ▾ Tab Width: 8 ▾

Applications ▾ Places ▾ gedit ▾

Open ▾ 

6.sh
~/Desktop


```
#!/bin/bash

count=0

until [ $count -gt 4 ]
do
  echo "Count: $count"
  count=$((count + 1))
done
```

sh ▾ Tab Width: 8 ▾

Applications ▾ Places ▾ gedit ▾

Open ▾ 


7.sh
~/Desktop

```
#!/bin/bash

for i in {1..10};
do
  if [ $i -le 4 ]; then
    echo "Number:$i"
  fi
done
```

sh ▾ Tab Width: 8 ▾

Applications ▾ Places ▾ gedit ▾

Open ▾ 

8.sh
~/Desktop

```
for i in {1..10}; do
  echo "Number:$i"
done
```

sh ▾ Tab Width: 8 ▾

实验中出现的问题汇总

1.权限问题：确保脚本文件有执行权限，可以使用 `chmod +x UserCheck.sh` 赋予执行权限。

2.用户名输入错误：注意确认输入的用户名是否正确。

3.添加用户时的密码设置：可以通过 `passwd` 命令手动为新用户设置密码，或者在脚本中调用 `passwd` 命令。

实验总结

通过这次实验，我们深入了解了如何在 **Shell** 脚本中使用条件判断语句来检测用户是否存在，并根据实际情况执行相应的操作。同时，我们还学会了如何创建简单的 **Linux** 用户管理脚本，并掌握了一些常见的问题和解决方法。这次实验对于提高我们的 **Shell** 编程能力非常有帮助。

在 **Shell** 脚本中，条件判断语句是一种非常重要的控制结构，它可以根据条件的结果来执行不同的操作。我们使用了 `if` 语句来实现条件判断，`if` 语句通常包括三个部分：条件表达式、执行语句 1 和执行语句 2。如果条件表达式的结果为真，则执行语句 1 会被执行；如果条件表达式的结果为假，则执行语句 2 会被执行。在 `if` 语句中，我们使用了双括号 `(())` 来包裹条件表达式，这样可以更方便地计算表达式的值。

在检查用户是否存在时，我们使用了 **Linux** 中的 `useradd` 命令来创建用户，并使用 `id` 命令来检查用户是否存在。如果用户不存在，则 `id` 命令会返回一个非零的退出状态码，我们可以利用这个特点在 **Shell** 脚本中进行条件判断。如果用户存在，则 `id` 命令会返回一个零的退出状态码，我们可以使用 `if` 语句来实现根据用户是否存在来执行不同的操作。

在创建 **Linux** 用户管理脚本时，我们使用了 **Linux** 中的一些常用命令来实现用户的创建、删除、密码设置等操作。我们还介绍了一些常见的问题和解决方法，例如在创建用户时遇到的问题、密码设置时遇到的问题等。这些问题的解决方法对于我们在实际工作中解决类似的问题非常有帮助。

通过这次实验，我们不仅学会了在 **Shell** 脚本中使用条件判断语句来检测用户是否存在，还学会了如何创建简单的 **Linux** 用户管理脚本，并掌握了一些常见的问题和解决方法。这对于提高我们的 **Shell** 编程能力非常有帮助，同时也为我们今后在实际工作中解决类似的问题提供了宝贵的经验。