

一、实验目的

熟练使用文本编辑器编写 Shell 脚本，掌握 Shell 脚本的基本语法（包括变量定义、条件判断、循环结构、函数定义与调用等），能够独立完成系统时间判断、数值比较、列表最小值查找、可执行文件计数及质数判断等实操任务，并成功在 Linux 系统中运行脚本，分析运行结果及可能出现的问题。

二、实验要求

1. 编写 Shell 脚本获取系统时间，判断当前时段为早上、下午或晚上并输出对应提示；
2. 编写 Shell 脚本实现输入两个整数，比较其大小并输出比较结果；
3. 编写 Shell 脚本查找给定列表中的最小值并输出；
4. 编写 Shell 脚本统计当前目录下可执行文件的数量并输出；
5. 编写 Shell 脚本，通过自定义函数判断给定数字是否为质数，调用该函数并输出判断结果。

三、实验内容与说明

（一）获取系统时间并判断时段

1. 实验目标：通过脚本获取系统当前小时（24 小时制），依据小时范围判断所属时段（早上、下午、晚上）并输出对应问候语。
2. 脚本设计与核心逻辑：脚本以 `#!/bin/bash` 指定解释器，通过 `hour=`date +%H`` 获取当前小时数（%H 表示 24 小时制格式）；采用 `case` 条件判断语句划分时段：01-11 点（匹配规则 `0[1-9]|1[01]`）输出“Good morning !!”，12-17 点（匹配规则 `1[2-7]`）输出“Good afternoon !!”，18-23 点及 00 点（匹配规则*）输出“Good evening !!”，覆盖全天所有时段。
3. 执行步骤与结果：使用 `gedit` 编辑器创建脚本文件 `time_check.sh`，通过 `chmod +x time_check.sh` 命令赋予脚本可执行权限，执行 `./time_check.sh` 运行脚本。从实操结果可知，脚本成功输出“Good morning !!”，说明运行时系统时间处于早上时段，脚本逻辑执行正常。

相关截图与脚本内容：

```
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ gedit
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ gedit time_check.sh
```



The screenshot shows a gedit editor window titled '*time_check.sh' with a subtitle '~/.桌面'. The window has a menu bar with '打开(O)', a dropdown arrow, a file icon, and buttons for '保存(S)', a hamburger menu, and window controls (minimize, maximize, close). The editor contains the following shell script:

```
1 #!/bin/bash
2 # 获取当前小时 (24小时制)
3 hour=`date +%H`
4
5 # 判断时段
6 case $hour in
7     0[1-9] | 1[01]) # 01~11点 → 早上
8         echo "Good morning !!"
9         ;;
10    1[2-7])         # 12~17点 → 下午
11        echo "Good afternoon !!"
12        ;;
13    *)              # 18~23/00点 → 晚上
14        echo "Good evening !!"
15        ;;
16 esac
```

```
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ gedit
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ gedit time_check.sh
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ chmod +x time_check.sh
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ ./time_check.sh
Good morning !!
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$
```

```
#!/bin/bash
# 获取当前小时 (24小时制)
hour=`date +%H`

# 判断时段
case $hour in
    0[1-9] | 1[01]) # 01~11点 → 早上
        echo "Good morning !!"
        ;;
    1[2-7]) # 12~17点 → 下午
        echo "Good afternoon !!"
        ;;
    *) # 18~23/00点 → 晚上
        echo "Good evening !!"
        ;;
esac
```

(二) 输入两个整数并比较大小

1. 实验目标：编写脚本实现接收用户输入的两个整数，通过条件判断比较两者大小，输出“大于”“小于”或“等于”的比较结果。
2. 脚本设计与核心逻辑：脚本以 `#!/bin/sh` 指定解释器，通过 `echo` 命令提示用户输入两个整数，使用 `read` 命令接收输入并赋值给变量 `first` 和 `second`；采用 `if-elif-else` 条件判断结构，通过 `-gt`（大于）、`-lt`（小于）运算符比较两个变量大小。需注意修正原始要求中变量大小写不一致的问题（将 `FIRST` 改为 `first`），确保变量引用准确。
3. 执行步骤与结果：使用 `gedit` 创建 `number_check.sh` 脚本，赋予可执行权限后运行。用户依次输入 13 和 19，脚本成功输出“13 is less than 19”，准确反映了两个整数的大小关系，验证了脚本逻辑的正确性。

相关截图与脚本内容：

```
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ gedit number_check.sh
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ chmod +x number_check.sh
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ ./number_check.sh
Enter the first integer:
13
Enter the second integer:
19
13 is less than 19
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$
```

```
#!/bin/sh
# 提示输入第一个整数
echo "Enter the first integer:"
read first

# 提示输入第二个整数
echo "Enter the second integer:"
read second

# 比较两个数的大小
if [ "$first" -gt "$second" ]
then
    echo "$first is greater than $second"
elif [ "$first" -lt "$second" ]
then
    echo "$first is less than $second"
else
    echo "$first is equal to $second"
fi
```

(三) 查找给定列表中的最小值

1. 实验目标：编写脚本遍历指定的整数列表，通过循环比较找到列表中的最小值并输出。
2. 脚本设计与核心逻辑：脚本指定 bash 解释器，初始化变量 `smallest` 为 10000（选取一个远大于列表中所有元素的数值，确保列表中最小元素能覆盖该初始值）；使用 `for` 循环遍历列表[8,2,18,0,-3,87]中的每个元素，通过 `test $i -lt $smallest` 判断当前元素 `i` 是否小于当前最小值 `smallest`，若成立则更新 `smallest` 为 `i`；循环结束后，

通过 `echo` 命令输出最终的最小值。

3. 执行步骤与结果：创建 `find_min.sh` 脚本并赋予可执行权限，运行后脚本输出“The minimal value in the list is:-3”。列表中元素-3 为最小数值，与输出结果一致，说明脚本的循环比较逻辑完全正确。

相关截图与脚本内容：

```
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ gedit find_min.sh
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ chmod +x find_min.sh
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ ./find_min.sh
The minimal value in the list is: -3
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$
```

```
#!/bin/bash
# 初始化smallest为一个较大数值（确保小于列表中所有元素）
smallest=10000

# 遍历列表中的每个元素
for i in 8 2 18 0 -3 87
do
    # 比较当前元素是否小于当前最小值
    if test $i -lt $smallest
    then
        # 若更小，则更新最小值
        smallest=$i
    fi
done

# 输出结果
echo "The minimal value in the list is: $smallest"
```

（四）统计当前目录下可执行文件数量

1. 实验目标：编写脚本遍历当前目录下的所有非隐藏文件，判断文件是否具有可执行权限，统计可执行文件的总数并输出。

2. 脚本设计与核心逻辑：脚本指定 `bash` 解释器，初始化计数器 `count` 为 0；使用 `for i in *` 遍历当前目录下所有非隐藏文件（*匹配所有非隐藏文件），通过 `test -x $i`

判断文件 `i` 是否具有可执行权限，若具有则通过 `count=`expr $count +1`` 实现计数器自增；循环结束后输出可执行文件总数。首次编写的 `count_exefile.sh` 脚本出现“第 7 行: `test`: 参数太多”的报错，推测是遍历过程中遇到名称含特殊字符的文件导致，后续修改为 `count_exefile1.sh` 脚本后解决该问题。

3. 执行步骤与结果：先后创建 `count_exefile.sh` 和 `count_exefile1.sh` 两个脚本，均赋予可执行权限并运行。`count_exefile.sh` 运行报错，`count_exefile1.sh` 运行后成功输出“Total of 8 files executable”，说明修正后的脚本能够正常遍历文件并统计可执行文件数量，最终统计结果准确。

相关截图与脚本内容：

```
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ gedit count_exefile.sh
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ chmod +x count_exefile.sh
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ ./count_exefile.sh
./count_exefile.sh: 第 7 行: test: 参数太多
Total of 8 files executable
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ S
```

```
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ gedit count_exefile1.sh
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ chmod +x count_exefile1.sh
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ ./count_exefile1.sh
Total of 8 files executable
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ S
```

```
#!/bin/bash
count=0
# 遍历当前目录下的所有非隐藏文件
for i in *
do
    # 判断文件是否具有可执行权限
    if test -x $i
    then
        # 计数器加1
        count=`expr $count + 1`
    fi
done
# 输出可执行文件总数
echo Total of $count files executable
```

（五）自定义函数判断给定数字是否为质数

1. 实验目标：编写自定义函数实现质数判断功能，脚本接收用户输入的数字，调用该函数后输出该数字是否为质数的结果。
2. 脚本设计与核心逻辑：脚本指定 `bash` 解释器，自定义 `prime` 函数处理质数判断：
 - ①接收参数 `num`（待判断数字），先进行边界处理：若 `num` 小于 2（1、0、负数），直接返回 0（表示非质数）；
 - ②初始化 `flag` 为 1（标记为质数），循环变量 `j` 为 2；
 - ③通过 `while [$j -le `expr $num / 2`]` 循环（优化逻辑：只需判断到 `num` 的 1/2 即可，无需遍历至 `num-1`），若 `num` 能被 `j` 整除（`expr $num % $j -eq 0`），则将 `flag` 设为 0（标记为非质数）并跳出循环；
 - ④循环结束后，根据 `flag` 值返回对应状态（1 表示质数，0 表示非质数）。脚本接收用户输入的数字，调用 `prime` 函数后，通过 `$?` 获取函数返回状态，输出最终判断结果。
3. 执行步骤与结果：创建 `prime_check.sh` 脚本并赋予可执行权限，多次运行脚本进行测试：输入 17（质数），输出“17 is a prime!”；输入 2（质数），输出“2 is a prime!”；输入 140（非质数），输出“140 is not a prime!”。多次测试结果均准确，说明自定义函

数逻辑严谨，脚本运行稳定。

相关截图与脚本内容：

```
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ gedit prime_check.sh
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ chmod +x prime_check.sh
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ ./prime_check.sh
Enter a number:
17
17 is a prime!
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ ./prime_check.sh
Enter a number:
2
2 is a prime!
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$ ./prime_check.sh
Enter a number:
140
140 is not a prime!
weizhang@weizhang-VMware-Virtual-Platform:~/桌面$
```

打开(O) 保存(S) prime_check.sh ~/桌面/linux code count_exefile1.sh

makefile

```
#!/bin/bash
# 定义判断质数的函数 (Shell函数需用规范格式)
prime() {
    num=$1
    # 边界处理: 小于2的数 (1、0、负数) 不是质数
    if [ $num -lt 2 ]; then
        return 0 # 非质数返回退出状态0
    fi
    flag=1
    j=2
    # 循环到数字的1/2 (优化: 也可循环到平方根, 效率更高)
    while [ $j -le `expr $num / 2` ]; do
        # 判断当前数字是否能被整除
        if [ `expr $num % $j` -eq 0 ]; then
            flag=0 # 能整除则标记为非质数
            break
        fi
        j=`expr $j + 1` # 循环变量自增
    done
    # 根据标记返回对应状态
    if [ $flag -eq 1 ]; then
        return 1 # 是质数返回退出状态1
    else
        return 0 # 非质数返回退出状态0
    fi
}
```

四、实验总结

本次实验聚焦 Shell 脚本的编写与运行，系统掌握了 Shell 脚本的核心语法与实操技巧。通过五个实验任务，分别掌握了：时段判断的 **case** 条件语句用法、数值比较的 **if-elif-else** 结构、列表遍历的 **for** 循环逻辑、文件权限判断与计数方法，以及自定义函数的定义、调用与返回值处理。

实验过程中，遇到了部分问题并成功解决：如数值比较脚本中变量大小写不一致的语法问题、可执行文件计数脚本中因文件名称特殊字符导致的报错问题，通过修正语法、优化脚本逻辑等方式完成调试。同时，在质数判断脚本中采用了循环范围优化策略，提升了脚本执行效率。本次实验不仅强化了 Shell 脚本的编写能力，也培养了问题排查与逻辑优化的思维，为后续复杂 Shell 脚本的开发奠定了坚实基础。