任务4 Shell编程

要求：

- 7选3完成即可
- 鼓励自选SHELL程序

参考资料：

- 知乎课堂笔记：第5篇 Linux登堂入室：shell编程 - 知乎 (zhihu.com)
- Bash 脚本教程

例题1：探测服务器的信息。

源码：example7.sh （点击右键另存为可以下载源码文件）

```
#第一个例子，探测主机信息
[zhangsan@localhost ~]$ vi example7.sh
#!/bin/bash
name=`hostname`
ip=`ifconfig ens160|awk /netmask/'{print $2}'`
os=`cat /etc/redhat-release`
kernel=`uname -r`
cpu=`lscpu|grep 厂商|tr -s " "|cut -d " " -f1 --complement`
mem=`free -hm |awk /Mem/'{print $2}'`

#-----------------------------------------------------------
# 这里是计算硬盘的总大小
#-----------------------------------------------------------
space=`lsblk |awk /disk/'{print $4}'|awk -F G '{print $1}'`
G="\e[1;32m"
B="\e[1;34m"
E="\e[0m"
#如果有多块硬盘的话，就循环计算所有的硬盘的总容量的大小
for i in $space
do
        let sum=$[$sum+$i]
done

echo -e "$B主机名：$E$G\t$name\t$E"
echo -e "${B}ip地址:$E$G\t$ip\t$E"
echo -e "$B系统版本：$E$G\t$os\t$E"
echo -e "$B内核版本：$E$G\t$kernel\t$E"
echo -e "${B}cpu名称：$E$G\t$cpu\t$E"
echo -e "$B内存总大小：$E$G\t$mem\t$E"
echo -e "$B硬盘总大小：$E$G\t${sum}G\t$E"
[zhangsan@localhost ~]$chmod u+x example7.sh
[zhangsan@localhost ~]$ ./example7.sh
主机名：     localhost.localdomain
ip地址：    192.168.14.140
系统版本：     Red Hat Enterprise Linux release 9.1 (Plow)
内核版本：     5.14.0-162.23.1.el9_1.aarch64
cpu名称：     ID:  Apple
内存总大小：     1.7Gi
硬盘总大小：     38G
```

例题2：多线程扫描主机

源码：example8.sh

```
#!/bin/bash
#编写脚本测试192.168.14.*整个网段中哪些主机处于开机状态，哪些主机处于关机状态。
#example8.sh这个程序是一个多线程的，速度较快
#example8-2.sh时一个单线程的，速度较慢

#清空当前目录下up.txt文件和down.txt的内容
 > up.txt
 > down.txt
#记录程序开始执行的时间
start_time=`date +%s`
for i in {1..10}
do
        #每隔0.3秒ping一次，一共ping2次，并以1毫秒为单位设置ping的超时时间
        ping -c 2 -i 0.3 -W 1 172.21.3.$i &>/dev/null

        (if [ $? -eq 0 ]
        then
                echo "172.21.3.$i is up" >> up.txt    #能够ping通的记录在up.txt文件中
        else
                echo "172.21.3.$i is down" >> down.txt  #不能够ping通记录在down.txt文件中
        fi)&   #把一个命令放进后台运行，相当于启动了一个独立的线程，这种方式是修改为并行多线程执行，调高效率。

done
        wait   #每个线程执行的时间不一样，这里等待所有的线程执行完毕
stop_time=`date +%s`    #所有线程执行完毕以后，记录结束时间
echo "时间：`expr $stop_time - $start_time`秒"  #计算一共花费了多长时间

#输出有多少线上的机器，多少线下的机器。
echo `cat up.txt | wc -l`hosts are up.
echo `cat down.txt | wc -l`hosts are down.
```

例题3：单线程扫描网段

源码：example8-2.sh

```
#!/bin/bash
#编写脚本测试192.168.14.*整个网段中哪些主机处于开机状态，哪些主机处于关机状态。
#example8.sh这个程序是一个多线程的，速度较快
#example8-2.sh时一个单线程的，速度较慢

#清空当前目录下up.txt文件和down.txt的内容
 > up.txt
 > down.txt
#记录程序开始执行的时间
start_time=`date +%s`
for i in {1..10}
do
        #每隔0.3秒ping一次，一共ping2次，并以1毫秒为单位设置ping的超时时间
        ping -c 2 -i 0.3 -W 1 172.21.3.$i &>/dev/null
        if [ $? -eq 0 ]
        then
                echo "172.21.3.$i is up" >> up.txt
        else
                echo "172.21.3.$i is down" >> down.txt
        fi
                ping -c 2 -i 0.3 -W 1 172.21.3.$i &>/dev/null

done

stop_time=`date +%s`    #记录结束时间
echo "时间：`expr $stop_time - $start_time`秒"  #计算一共花费了多长时间

#输出有多少线上的机器，多少线下的机器。
echo `cat up.txt | wc -l`hosts are up.
echo `cat down.txt | wc -l`hosts are down.
```

例题4：从网络上获取天气预报

源码：example32.sh

```
#第三个例子
[root@localhost examplesh]# cat example32.sh
#!/bin/bash
#从天气预报网获取该城市的天气json数据
json=`curl -s http://www.weather.com.cn/data/sk/101010100.html`
echo $json
#提取城市、温度、几级风、风向四个数据
city=`echo $json | sed 's/.*city":"//g' | sed 's/","cityid.*$//g'`
temp=`echo $json | sed 's/.*temp":"//g' | sed 's/","WD.*$//g'`
wd=`echo $json | sed 's/.*WD":"//g' | sed 's/","WS.*$//g'`
ws=`echo $json | sed 's/.*WS":"//g' | sed 's/","SD.*$//g'`

#提取时间，判断早中晚
tm=$(date +%H)
if [ $tm -gt 12 ]
then
    msg="Good Morning $USER"
elif [ $tm -gt 12 -a $tm -le 18 ]
then
    msg="Good Afternoon $USER"
else
    msg="Good Night $USER"
fi

#格式化输出时间和天气信息
echo "当前的时间是：$(date)"
echo -e "\033[34m$msg\033[0m"
echo '你现在位于'$city', '$temp'℃, '$ws $wd'.'
[root@localhost examplesh]# bash example32.sh
当前的时间是：2024年 03月 22日 星期五 22:30:32 CST
Good Morning root
你现在位于北京, 18℃, 1级 东南风.
```

例题5：批量创建用户

> 运行说明：
>
> 例题5：sh07.sh 批量创建用户
>
> 例题6： sh08.sh 批量删除用户
>
> 这两个程序运行都需要再当前目录下创建一个文件user1.txt，该文件保存的是想要批量创建或者批量删除的用户的用户名和密码。
>
> ```
> # 新建一个文件user1.txt
> [root@bogon ~]# touch user1.txt
> # 使用nano编辑器编辑其内容
> [root@bogon ~]# nano user1.txt
> # 该文件内容如下所示，每一行都包含两个字段，第一个是用户名，第二个字段是密码，以空格作为分隔符
> [root@bogon ~]# cat user1.txt
> zhangsan 123456
> lisi 654321
>
> 运行方式为：
> # 批量创建user1.txt文件中指定的用户
> [root@bogon ~]#bash sh07.sh user1.txt
> # 批量删除user1.txt文件中
> [root@bogon ~]#bash sh08.sh user1.txt
> ```

源码：

sh07.sh

sh08.sh

user1.txt

```
[root@localhost temp]# cat sh07.sh

#!/bin/bash
#作用：把需要批量的创建的用户的名称和密码保存在当前目录下的user1.txt中，、
#然后从脚本运行时读入该文件，例如：bash sh07.sh user1.txt
#系统循环读入该文件中的每一行，并自动创建用户。
#特别注意的是：第二次运行程序的时候，需要使用userdel -r 用户名的方式彻底从系统中清除该用户的文件。然后才能再次成功运行。
#运行完成后，最好把创建的用户从系统清除。

#判断是否有参数
#用户名和密码保存在user1.txt文件中
if [ $# -eq 0 ]
then
    echo "输入文件：`basename $0` file"
    exit
fi

#判断是否是一个文件
if [ ! -f $1 ]
then
    echo "error file"
    exit
fi

#考虑到特殊情况，如果变量是空行，其解决办法是重新定义分隔符
#for处理文件按回车分隔，而不是按空格或者TAB
#重新定义分隔符
#IFS内部字段分隔符
IFS=$'\n'
for line in `cat $1`
do
    if [ ${#line} -eq 0 ]
    then
        echo "nothing to do"
        continue
    fi

# 在Linux系统中，管道无法给变量直接赋值，例如下面的句子
#  user=`echo "$line"`| awk -F " " '{print $1}这里直接用管道把两个命令连接起来，无法给user变量赋值
#  只有使用echo 把整个管道处理后的结果打印出来以后，才能给变量赋值，例如下面的示例：

    user=`echo "$line" | awk -F " " '{print $1}'`
    pass=`echo "$line" | awk -F " " '{print $2}'`

    echo $user
    echo $pass

    id $user &> /dev/null

    if [ $? -eq 0 ]
    then
        echo "user $user already exists"
    else
        echo "success"
        useradd $user
        echo "$pass" | passwd --stdin $user &> /dev/null


        if [ $? -eq 0 ]
        then
            echo "$user is created."
        fi
    fi
done
#批量创建用户和密码
```

例题6：批量删除用户的程序

```
#-------------------------------------------------------------------------------------------------
#-------------------------------------------------------------------------------------------------
[root@localhost temp]# cat sh08.sh
#!/bin/bash
#作用：该程序的作用是和sh07.sh程序配套的，用来进行把用户从系统里面干净的清除。
#每次运行：bash sh07.sh user1.txt 以后
#可以使用：bash sh08.sh user1.txt 来清除sh07脚本创建的用户。


#判断是否有参数
#用户名和密码保存在user1.txt文件中
if [ $# -eq 0 ]
then
    echo "输入文件：`basename $0` file"
    exit
```

```bash
fi

#判断是否是一个文件
if [ ! -f $1 ]
then
    echo "error file"
    exit
fi

#考虑到特殊情况，如果变量是空行，其解决办法是重新定义分隔符
#for处理文件按回车分隔，而不是按空格或者TAB
#重新定义分隔符
#IFS内部字段分隔符
IFS=$'\n'
for line in `cat $1`
do
    if [ ${#line} -eq 0 ]
    then
        echo "nothing to do"
        continue
    fi

#在Linux系统中，管道无法给变量直接赋值，例如下面的句子
#  user=`echo "$line"`| awk -F " " '{print $1} 这里直接用管道把两个命令连接起来，无法给user变量赋值
#  只有使用echo 把整个管道处理后的结果打印出来以后，才能给变量赋值，例如下面的示例：
    user=`echo "$line" | awk -F " " '{print $1}'`

    id $user &> /dev/null

    if [ $? -eq 0 ]
    then
        echo "$user is already deleted!"
        userdel -r $user
    else
        echo "没有$user用户"
    fi
done
#批量创建用户和密码
```

如何运行这两个程序：

```
#----------------------------------------------------------------------------------------------------------
运行示例
#----------------------------------------------------------------------------------------------------------
[root@localhost temp]# bash sh07.sh user1.txt
test_user1
123456
success
test_user1 is created.
[root@localhost temp]# bash sh08.sh user1.txt
test_user1 is already deleted!
```

例题7：打字小游戏

源码：example33.sh

```bash
#!/bin/bash

#------------------------------------------------------------
# 练习：单词打字练习，本程序主要练习函数的使用方式
#------------------------------------------------------------

# 随机单词列表
word_list=("hello" "world" "programming" "practice" "typing")

# 总单词数
total_words=$((${#word_list[@]}))

# 当前单词索引
current_word_index=0

# 显示当前单词
function display_word() {
  local word=${word_list[$current_word_index]}
  echo "请输入单词: $word"
}

# 检查输入是否正确
function check_input() {
  local input_word=$1
  local correct_word=${word_list[$current_word_index]}

  if [ "$input_word" = "$correct_word" ]; then
    echo "正确! 下一个单词..."
    ((current_word_index++))
  else
    echo "错误! 正确的单词是: $correct_word, 请重新输入。"
  fi

  # 检查是否完成所有单词
  if [ $current_word_index -eq $total_words ]; then
    echo "恭喜您完成了所有单词的打字练习! "
    exit 0
  fi
}

# 主循环
while true; do
  display_word # 显示当前单词
  read -p "请输入您的答案并按回车键: " user_input
  # 检查输入是否正确,调用函数来实现
  check_input "$user_input"
done
```