* 1. shell

shell is a command line interpreter which provides user with an interface to send requests to Linux Kernel.
shell is between the user application level and Linux Kernel level.
Many different shells are available in Linux. However, the most widely-used is bash.

* 2. annotations in shell

annotation for one line: #
annotation for multiple lines:
:<<!
annotation content...
!

* 3. format of shell script

vim hello.sh

#!/bin/bash
output the string hello, world!
echo hello, world!

how to execute the file?
First way: ./FileName or using absolute path (need to grant execution permission in advance)
Second way: using command sh FileName, in this case which is sh hello.sh (no need for granting execution permission in advance)

* 4. variables in shell

two types of variables in Linux. One is system variables which have been pre-defined in the system and the other one is user-defined variables.
Examples: $HOME, $PWD, $USER, $SHELL ... (echo $HOME ...)
command "set" can be used to display all the current variables in shell.

vim var1.sh

#!/bin/bash
#define variable A (it is crucial that no space between A and 100)
A=100
#two ways to output the variable defined and note that $ cannot be omitted(the output result

```
are both A=100)
echo A=$A
echo "A=$A"
#discard variables (the output result is that A= )
unset A
echo A=$A
#declare static variable B, which cannot be discarded (the output result is B=200)
readonly B=200
echo B=$B
#try to unset readonly variable, which will fail.
unset B
```

rules: there are some rules of defining variables in shell
variable name can consist of letters, numbers and underlines (cannot start with a number)
no SPACE on either sides of the equal sign
variable names are generally uppercases

vim var2.sh

```
#!/bin/bash
#the output is A=date
A=date
echo A=$A
#the output is the result of the command "date", which return the time
A=`date`   #or use A=$(date)
echo A=$A
```

* 5. set environment variables

all the shell scripts can use it after the environment variable is defined.

the grammar used to define environment variables:
export varname=value
source filename #refresh the file (must do this before using the variable)
echo $varname #check whether taking effect

* 6. positional parameter variable

basic knowledge: $n -- n is a number, $0 represents the command itself and $1-$9 represent the first to the ninth parameter

the parameter larger than 10 need to be surrounded by the braces, e.g. ${10}

$* -- represent all the parameters in the shell. It regards all the parameters as an entirety

$@ -- same as $*. But it regards the parameters separately

$# -- represent the number of the parameters in the shell

vim position.sh

```
#!/bin/bash
#the quotes will not be output (quotes after key word echo means output the content between the quotes)
echo "0=$0 1=$1 2=$2"
echo "all the parameters=$*"
echo "$@"
echo "the number of parameters=$#"
```

sh position.sh 100 200

the output result is:

```
0=position.sh 1=100 2=200
all the parameters=100 200
100 200
the number of parameters=2
```

* 7. pre-defined variable

pre-defined variables are pre-defined by the shell developers, which can be used directly

basic grammar: $$ -- show the id of current process

$! -- show the id of the last process running in the background

$? -- return the status of command executed in the final time

the return value is 0, which means executed correctly

otherwise, executed incorrectly

vim prevar.sh

```
#!/bin/bash
echo "id of current process=$$"
~/Desktop/sh/position.sh &
echo "id of the last process running in the background=$!"
echo "the execution result=$?"
```

* 8. operator

how to do some arithmetic operation in shell?
basic grammar:
$((expression)) or $[expression] or expr m + * / % n
note that if using expr, there must be spaces between values and operators

vim operator.sh
#calculate the value of (2+3)*4
#!/bin/bash
A=$[(2+3)*4]
echo A=$A

:<<!
or:
A=$(((2+3)*4))
echo A=$A

or:
TEMP=`expr 2 + 3`
RES=`expr $TEMP \* 4`
echo result=$RES
!

#calculate the sum of parameter1 + parameter2
SUM=$1+$2
echo SUM

chmod u+x operator.sh
./operator.sh 20 50

* 9. conditional statement

9.1 flow control (if--)
basic grammar: [condition]
note that the there must be one space between the condition and square brackets
condition not empty, then return true. Otherwise, return false.
e.g. [james] -- return true
[ ] -- return false

frequently-used:
(1) string comparison: =
(2) integer comparison: -lt(<) -le(<=) -eq(=) -gt(>) -ge(>=) -ne(not equal)
(3) file permission comparison: -r(read) -w(write) -x(execute)
(4) file type: -f(file exits and is a regular file) -e(file exists) -d(the file is a directory)

vim ifdemon.sh

```
#!/bin/bash
#case1: "ok" =? "ok"
if [ "ok" = "ok" ]
then
        echo equal
fi
#case2: 23 >=? 22
if [ 23 -ge 22 ]
then
        echo 23 >= 22
fi
#case3: /etc/hostid    whether the file hostid exists
if [ -f /etc/hostid ]
then
        echo the path of file hostid is /etc/hostid
fi
#case4: more cases
if [ ]
then
        echo "true"
fi
```

a concise version of if-elif:
[condition] && command1 || command2 #if condition is true, execute command1, otherwise command2
e.g.

9.2 flow control (elif--)

vim ifcase.sh

```
#!/bin/bash
#requirement: input a score, if the value >= 60 then output "pass". Otherwise, output "fail"
if [ $1 -ge 60 ]
then
        echo pass
elif [ $1 -lt 60 ]
then
        echo fail
fi
```

chmod u+x ifcase.sh
./ifcase.sh 90

9.3 flow control (case--)
basic grammar:
case $varname in
"value1")
#if varname=value1, execute here
;;
"value2")
#if varname=value2, execute here
;;
"value3")
#if varname=value3, execute here
;;
*)
#if varname doesn't equal to any values above, execute here
;;
esac

e.g.
case $1 in
"1")
echo Monday
;;
"2")
echo Tuesday
;;
"3")
echo Wednesday
;;
*)
echo others
;;
esac

9.4 loop (for--)
basic grammar:
the fisrt way--
for var in value1 value2 value3
do
#content here...
done

vim for1.sh

```
#see the difference of $* and $@ in section6.
#!/bin/bash
for i in "$*"    #note the quotes are necessary here
do
        echo num is $i
done
# ---------------------
for i in "$@"    #note the quotes are necessary here
do
        echo num is $i
done
```

chmod u+x for1.sh
./for1.sh 1 2 3 4 5   #1 2 3 4 5 is just an example

the second way--

```
for((initial value;loop control condition;variable change))
do
#content here
done
```

vim for2.sh

```
#output the value of 1+2+3+...+n
#!/bin/bash
SUM=0
for((i=0;i<=$1;i++))
do
        SUM=$[$SUM+$i]
done
echo sum=$SUM
```

chmod u+x for2.sh 10   #10 is just an example

9.5 loop (while--)
basic grammar:

```
while [ condition ]
do
#content here
done
```

vim while.sh

```
#output the value of 1+2+3+...+n
#!/bin/bash
SUM=0
i=0
while [ $i -le $1 ]
do
        SUM=$[$SUM+$i]
        i=$[$i+1]
done
```

```
chmod u+x while.sh
./while.sh 10    #here 10 is just an example
```

* 10. read input from command line

basic grammar:
read option argument
option: -p specify the prompt message of the reading input
        -t specify the input waiting time
argument: variable

vim read.sh

```
#!/bin/bash
#read command line and input to variable NUM1
read -p "the input value is: " NUM1
echo "the input value is $NUM1"
#read command line and input to variable NUM2 (input in 10s)
read -t 10 -p "input value(please input in 10s): " NUM2
echo "the input value is $NUM2"
```

* 11. function

11.1 system function
basename path #it will return the file name
dirname path #contrary to the command basename, it will return the path
e.g.

```
basename /etc/firefox/syspref.js    #it will return syspref.js
dirname /etc/firefox/syspref.js    #it will return /etc/firefox
```

11.2 custom function
basic grammar:

```
function funname(){
        Action
        return #optional
}
funname para1 para2 ... paraN
```

vim getsum.sh

```
#!/bin/bash
#construct a function capable of calculating the sum of two numbers
function getsum() {
        echo "sum = $[$NUM1+$NUM2]"
}
read -p "input first value: " NUM1
read -p "input second value: " NUM2
getsum $NUM1 $NUM2
```

* 12. case study--database backup

requirement:
(1)backup the database xxx to the directory /data/backup/db at every morning 02:30
(2)show some correspoding prompt messages at the beginning and the end of backup process
(3)take the backup time as the name of the backup file and package it to .tar.gz, like: 2022-10-05_023012.tar.gz
(4)check if there are still backup files 10 days ago when backup, delete them if existing

#----------
first make sure MySQL is installed on the VM
use command to start MySQL and reset the password:

```
        systemctl start mysqld.service
        mysql -u root -p    #log in using root with its password
        set password for 'root'@'localhost' =password('input password here'); #in centos I set it "123456789"
        flush privileges; #this command ensures the password to take effect
```

#----------

vim mysql_db_backup.sh

```
#!/bin/bash
#the directory for backup files
```

```
BACKUP=/data/backup/db
#current time
DATETIME=$(date +%Y-%m-%d_%H%M%S)
#the address of database
HOST="localhost"
#database user
DB_USER=root
#user password
DB_PW=123456789
#the name of the database created in MySQL
DATABASE=james

#create the backup directory if it does not exist
[ ! -d ${BACKUP}/${DATETIME} ] && mkdir -p ${BACKUP}/${DATETIME}

#dump the specific database created in MySQL
mysqldump -u${DB_USER} -P${DB_PW} --host=${HOST} --databases ${DATABASE} | gzip >
${BACKUP}/${DATETIME}/$DATETIME.sql.gz
#or mysqldump -h${IP} -u ${DB_USER} -p${DB_PW} ${DATABASE} | gzip >
${BACKUP}/${DATETIME}/$DATETIME.sql.gz
#first declare variable IP="127.0.0.1"

#compress the file to tar.gz
cd ${BACKUP}
tar -zcvf $DATETIME.tar.gz ${DATETIME}
#delete corresponding backup files
rm -rf ${BACKUP}/${DATETIME}

#delete backup files 10 days ago
find ${BACKUP} -atime +10 -name "*.tar.gz" -exec rm -rf {} \; #here -exec is followed by
command and {} means the files found by "find"
echo "database $DATABASE backup completed!"

crontab -e
30 2 * * * /usr/sbin/mysql_db_backup.sh
#crontab -l | -r | -e
# * * * * * ...
# minite hour day month week

#then everything is completed!
```