

# Shell概述

---

本次课程主要包含内容:

1. Shell脚本入门
2. Shell变量
3. Shell内置命令
4. Shell运算符与执行运算命令
5. 流程控制语句
6. Shell函数
7. Shell重定向
8. Shell好用的工具, cut sed awk sort
9. 大厂常见企业面试题

## Shell脚本入门：介绍

---

### 目标

理解Shell是什么

理解Shell脚本是什么

理解为什么学习Shell脚本(Shell脚本程序的作用)

linux系统默认的Shell解析器

### 疑问

linux系统是如何操作计算机硬件CPU,内存,磁盘,显示器等?

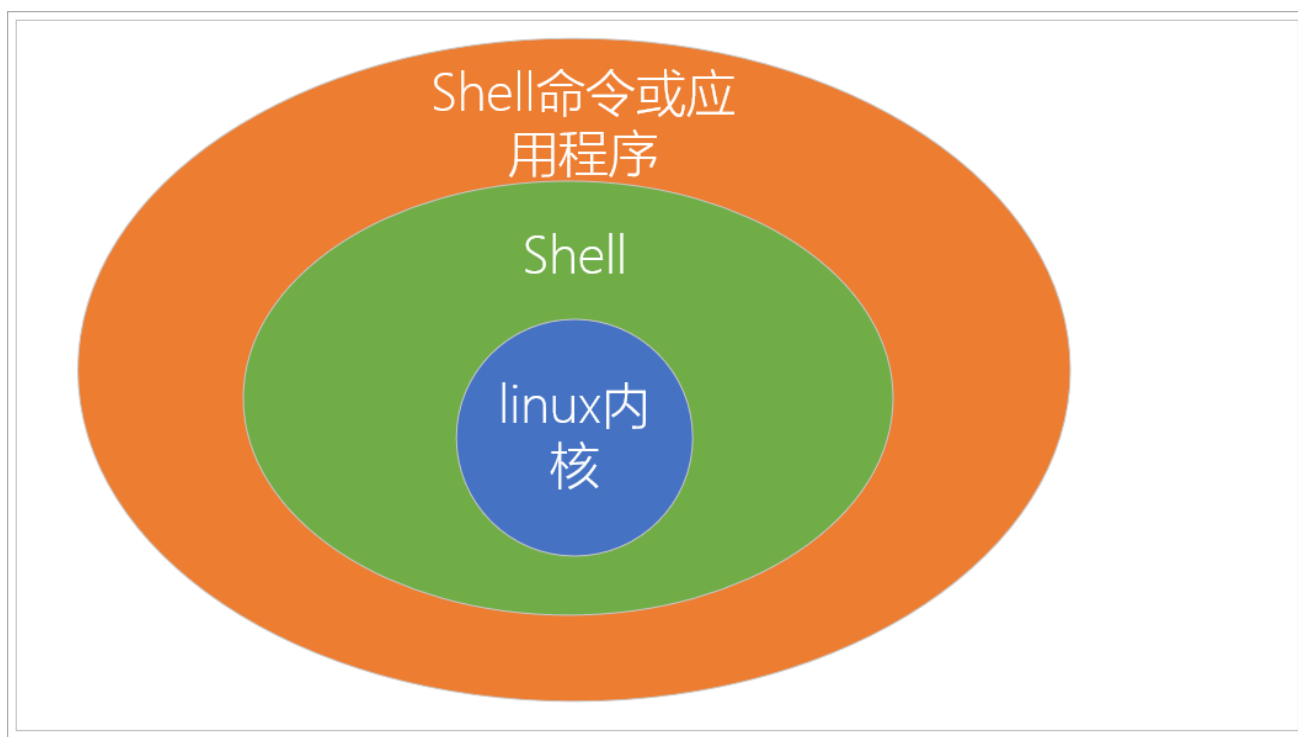
答: 使用linux的内核操作计算机的硬件

### Shell介绍

通过编写Shell命令发送给linux内核去执行, 操作就是计算机硬件. 所以Shell命令是用户操作计算机硬件的桥梁,

Shell是命令, 类似于windows系统Dos命令

Shell是一个门程序设计语言, Shell里面含有变量, 函数, 逻辑控制语句等等



## Shell脚本

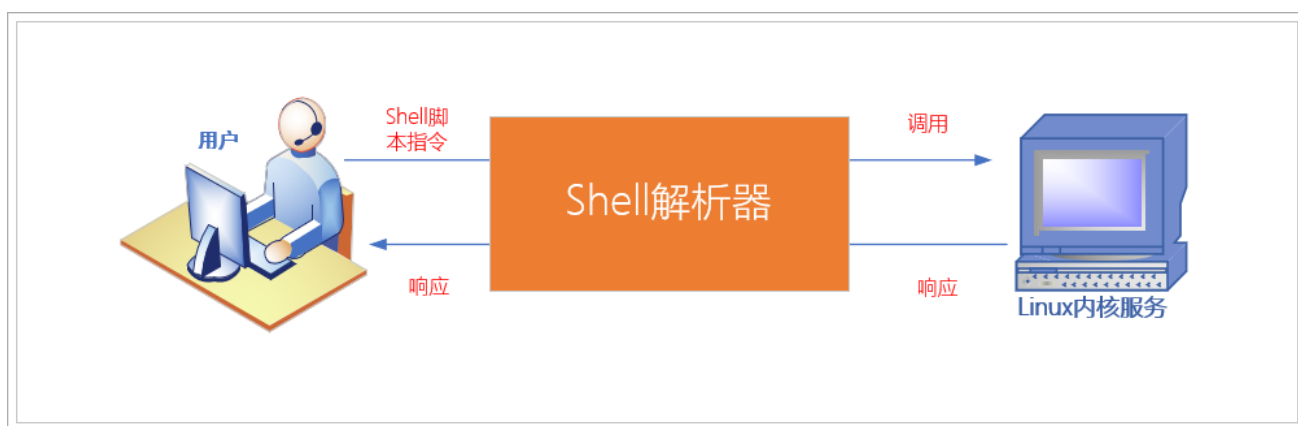
通过Shell命令或程序编程语言编写的Shell文本文件, 这就是Shell脚本, 也叫Shell程序

## 为什么学习Shell脚本?

通过Shell命令与编程语言来提高linux系统的管理工作效率

## Shell的运行过程

当用户下达指令给该操作系统的时候, 实际上是把指令告诉shell, 经过shell解释, 处理后让内核做出相应的动作。系统的回应和输出的信息也由shell处理, 然后显示在用户的屏幕上。



## Shell解析器

查看linux系统centos支持的shell解析器

```
cat /etc/shells
```

效果

```
[root@itcast ~]# cat /etc/shells
/bin/sh
/bin/bash
/sbin/nologin
/usr/bin/sh
/usr/bin/bash
/usr/sbin/nologin
/bin/tcsh
/bin/csh
[root@itcast ~]# █
```

介绍解析器类型

解析器类型	介绍
/bin/sh	Bourne Shell,是UNIX最初使用的shell;
/bin/bash	<b>Bourne Again Shell它是Bourne Shell的扩展，简称bash，是LinuxOS默认shell,有灵活和强大的编辑接口，同时又很友好的用户界面，交互性很强；</b>
/sbin/nologin	未登录解析器，shell设置为/sbin/nologin 是用于控制用户禁止登陆系统的, 有时候有些服务，比如邮件服务，大部分都是用来接收主机的邮件而已，并不需要登陆
/bin/dash	dash（Debian Almquist Shell），也是一种 Unix shell。它比 Bash 小，只需要较少的磁盘空间，但是它的对话性功能也较少，交互性较差。
/bin/csh	C Shell是C语言风格Shell
/bin/tcsh	是C Shell的一个扩展版本。

## Centos默认的解析器是bash

语法

```
echo $SHELL
```

含义: 打印输出当前系统环境使用的Shell解析器类型

echo 用于打印输出数据到终端

`$SHELL` 是全局共享的读取解析器类型环境变量, 全局环境变量时所有的Shell程序都可以读取的变量,

效果

```
[root@itcast ~]# echo $SHELL
/bin/bash
[root@itcast ~]# █
```

## 小结

### 1、Shell是什么

是命令，类似windows的dos命令  
又是一门程序设计语言，里面含有变量，函数，逻辑控制语句等

### 2、Shell脚本是什么

是一个文本文件，里面可以编写Shell命令或进行编程，形成一个可重用执行的脚本文件

### 3、shell脚本的作用

通过shell编程提高对linux系统管理工作效率

### 4、linux系统默认的shell解析器

/bin/bash

# Shell脚本入门：编写格式与执行方式

## 目标

- 1、掌握shell脚本编写规范
- 2、掌握执行shell脚本文件的3种方式与区别

## Shell脚本文件编写规范

### 脚本文件后缀名规范

shell脚本文件就是一个文本文件, 后缀名建议使用 `.sh` 结尾

## 首行格式规范

首行需要设置Shell解析器的类型, 语法

```
#!/bin/bash
```

含义: 设置当前shell脚本文件采用bash解析器运行脚本代码

## 注释格式

单行注释, 语法

```
# 注释内容
```

多行注释, 语法

```
:<<!  
# 注释内容1  
# 注释内容2  
!
```

## shell脚本HelloWord入门案例

### 需求

创建一个Shell脚本文件helloworld.sh, 输出hello world字符串

### 效果

```
[root@itcast ~]# sh helloworld.sh  
hello world  
[root@itcast ~]#
```

### 实现步骤

1、创建shell脚本文件

```
touch helloworld.sh
```

2、编辑文件

```
vim helloworld.sh
```

3、增加shell脚本文件内容如下，并保存退出

```
#!/bin/bash
echo "hello world"
```

4、执行脚本

```
sh helloworld.sh
```

运行效果

```
[root@itcast ~]# touch helloworld.sh
[root@itcast ~]# vim helloworld.sh
[root@itcast ~]# sh helloworld.sh
hello world
[root@itcast ~]# █
```

## 脚本文件的常用执行3种方式

### 介绍

#### 1. sh解析器执行方式

语法: `sh 脚本文件`

介绍: 就是利用sh命令执行脚本文件, 本质就是使用Shell解析器运行脚本文件

#### 2. bash解析器执行方式

语法: `bash 脚本文件`

介绍: 就是利用bash命令执行脚本文件, 本质就是使用Shell解析器运行脚本文件

#### 3. 仅路径执行方式

语法: `./脚本文件`

介绍: 执行当前目录下的脚本文件

注意: 脚本文件自己执行需要具有可执行权限, 否则无法执行

### 3种方式的区别

sh或bash执行脚本文件方式是直接使用Shell解析器运行脚本文件, 不需要可执行权限

仅路径方式是执行脚本文件自己, 需要可执行权限

### 执行脚本方式1-sh命令执行

相对路径效果

```
sh helloworld.sh
```

```
[root@itheima ~]# sh helloworld.sh
hello world
[root@itheima ~]# █
```

绝对路径效果

```
sh /root/helloworld.sh
```

```
[root@itheima ~]# sh /root/helloworld.sh
hello world
[root@itheima ~]# █
```

## 执行脚本方式2-bash命令执行

相对路径效果

```
[root@itheima ~]# bash helloworld.sh
hello world
[root@itheima ~]# █
```

绝对路径效果

```
[root@itheima ~]# bash /root/helloworld.sh
hello world
[root@itheima ~]# █
```

## 执行脚本方式3-仅路径执行

语法

步骤1：设置所有用户对此脚本文件增加可执行性权限

```
chmod a+x 脚本文件
```

步骤2：执行脚本语法

脚本文件的相对路径或绝对路径

**示例：使用仅路径方式执行helloworld.sh脚本文件**

添加执行权限

```
chmod a+x helloworld.sh
```

```
[root@itheima ~]# chmod a+x helloworld.sh
[root@itheima ~]#
```

相对路径执行命令

```
./helloworld.sh
```

相对路径执行效果

```
[root@itheima ~]# ./helloworld.sh
hello world
[root@itheima ~]#
```

绝对路径执行命令

```
/root/helloworld.sh
```

绝对路径执行效果

```
[root@itheima ~]# /root/helloworld.sh
hello world
[root@itheima ~]#
```

## 小结

1、shell脚本文件编写规范？

文件的后缀名: 建议使用 `.sh` 扩展名

首行需要设置解析器类型 `#!/bin/bash`

脚本文件中的注释

单行注释, `# 注释内容`

多行注释 `:<<!`

`# 注释内容`

!

2、执行shell脚本文件有哪3种方式，并说明他们的区别？

sh执行脚本文件

bash执行脚本文件

仅路径执行脚本文件

区别: 前2种是解析器直接执行不需要可执行权限, 最后一种是脚本文件自己执行需要可执行权限



# Shell脚本入门：多命令处理

## 目标

掌握shell脚本文件中执行多命令处理

## 多命令处理介绍

就是在Shell脚本文件中编写多个Shell命令

## 案例需求

已知目录/root/itheima目录，执行batch.sh脚本，实现在/root/itheima/目录下创建一个one.txt,在one.txt文件中增加内容“Hello Shell”。

## 步骤分析

1. 使用mkdir创建/root/itheima目录
2. 创建脚本文件batch.sh
3. 编辑脚本文件
  - 3.1 命令1: touch创建文件, 文件名 `/root/itheima/one.txt`
  - 3.2 命令2: 输出数据"Hello Shell"到one.txt文件中

输出数据到文件中的命令：  
`数据 >> 文件`
4. 执行脚本文件

## 实现步骤

- 1、进入root目录，执行创建/root/itheima目录命令

```
mkdir /root/itheima
```

- 2、创建/root/batch.sh文件

```
touch batch.sh
```

```
[root@itheima ~]# pwd
/root
[root@itheima ~]# touch batch.sh
[root@itheima ~]# ll
总用量 8
-rw----- 1 root root 1287 3月  7 10:02 anaconda-ks.cfg
-rw-r--r-- 1 root root    0 4月  2 20:53 batch.sh
-rwxr-xr-x 1 root root   31 4月  1 12:03 helloworld.sh
[root@itheima ~]#
```

## 2、编辑batch.sh文件，编写shell命令

```
vim batch.sh
```

## 3、编写命令

命令1：创建/root/itheima/one.txt文件

命令2：输出“I love Shell”字符串数据到one.txt文件中

```
#!/bin/bash
cd itheima      # 切换到itheima目录
touch one.txt   # 创建文件one.txt
echo "Hello Shell">>/root/itheima/one.txt #输出数据到one.txt文件中
```

```
#!/bin/bash
cd itheima
touch one.txt
echo "Hello Shell">>/root/itheima/one.txt
```

## 运行脚本效果

运行batch.sh脚本文件

```
sh batch.sh
```

查看one.txt文件内容

```
cat itheima/one.txt
```

```
[root@itheima ~]# sh batch.sh
[root@itheima ~]# cat itheima/one.txt
Hello Shell
[root@itheima ~]#
```

## 小结

shell脚本文件中是否可以执行多命令处理？

可以

# Shell变量：环境变量

---

## 目标

---

- 1、理解什么是系统环境变量？
- 2、掌握常用的系统环境变量都有哪些？

## Shell变量的介绍

---

变量用于存储管理临时的数据, 这些数据都是在运行内存中的.

## 变量类型

---

1. 系统环境变量
2. 自定义变量
3. 特殊符号变量

## 系统环境变量

---

### 介绍

是系统提供的共享变量.是linux系统加载Shell的配置文件中定义的变量共享给所有的Shell程序使用

## Shell的配置文件分类

- 1.全局配置文件 /etc/profile /etc/profile.d/\*.sh /etc/bashrc
- 2.个人配置文件 当前用户/.bash\_profile 当前用户/.bashrc

一般情况下，我们都是直接针对全局配置进行操作。

## 环境变量分类

在Linux系统中，环境变量按照其作用范围不同大致可以分为系统级环境变量和用户级环境变量。

系统级环境变量：Shell环境加载全局配置文件中的变量共享给所有用户所有Shell程序使用, 全局共享 用户级环境变量：Shell环境加载个人配置文件中的变量共享给当前用户的Shell程序使用, 登录用户使用

## 查看当前Shell系统环境变量

查看命令

env

效果

```
[root@itheima ~]# env
XDG_SESSION_ID=71
HOSTNAME=itheima
SELINUX_ROLE_REQUESTED=
TERM=xterm
SHELL=/bin/bash
HISTSIZE=1000
SSH_CLIENT=192.168.56.1 55720 22
SELINUX_USE_CURRENT_RANGE=
SSH_TTY=/dev/pts/1
USER=root
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:bd=40;33:01:cd=40;33;01:or=40;31;01:mi=01;05;37;41:su=37;41:sg=30;4
3:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.
lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01
;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:
*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.
jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.
tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:
*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.
rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.e
mf=01;35:*.axv=01;35:*.anx=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=01;36:*.au=01;36:*.flac=01;36:*.mid=01;36:*.midi=01;36:*.mka=01;36:*.m
p3=01;36:*.mpc=01;36:*.ogg=01;36:*.ra=01;36:*.wav=01;36:*.axa=01;36:*.oga=01;36:*.spx=01;36:*.xspf=01;36:
MAIL=/var/spool/mail/root
PATH=/usr/local/jdk1.8.0_162/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
PWD=/root
JAVA_HOME=/usr/local/jdk1.8.0_162
LANG=zh_CN.UTF-8
SELINUX_LEVEL_REQUESTED=
HISTCONTROL=ignoredups
SHLVL=1
HOME=/root
LOGNAME=root
CLASSPATH=.:usr/local/jdk1.8.0_162/lib
SSH_CONNECTION=192.168.56.1 55720 192.168.56.102 22
LESSOPEN=||/usr/bin/lesspipe.sh %s
XDG_RUNTIME_DIR=/run/user/0
_=usr/bin/env
[root@itheima ~]#
```

## 查看Shell变量(系统环境变量+自定义变量+函数)

命令

set

效果

```
[root@itheima ~]# set
ABRT_DEBUG_LOG=/dev/null
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:expand_aliases:extglob:extquote:forc
omments:login_shell:progcomp:promptvars:sourcepath
BASHRC_VARI=/etc/bashrc
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_COMPLETION_COMPAT_DIR=/etc/bash_completion.d
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSION0=([0]="4" [1]="2" [2]="46" [3]="2" [4]="release" [5]="x86_64-redhat-linux-gnu")
BASH_VERSION='4.2.46(2)-release'
CLASSPATH=./usr/local/jdk1.8.0_162/lib
COLUMNS=100
COMP_WORDBREAKS=$' \t\n\"'><=;|&(:'
DIRSTACK=()
EUID=0
GROUPS=()
HISTCONTROL=ignoredups
HISTFILE=/root/.bash_history
HISTFILESIZE=1000
HISTSIZE=1000
HOME=/root
HOSTNAME=itheima
HOSTTYPE=x86_64
ID=0
IFS=$' \t\n'
JAVA_HOME=/usr/local/jdk1.8.0_162
LANG=zh_CN.UTF-8
```

## 常用系统环境变量

变量名称	含义
==PATH==	与windows环境变量PATH功能一样，设置命令的搜索路径，以冒号为分割
HOME	当前用户主目录：/root
SHELL	当前shell解析器类型：/bin/bash
==HISTFILE==	显示当前用户执行命令的历史列表文件：/root/.bash_history
PWD	显示当前所在路径：/root
OLDPWD	显示之前的路径
HOSTNAME	显示当前主机名：itheima
HOSTTYPE	显示主机的架构，是i386、i686、还是x86、x64等：x86_64
==LANG==	设置当前系统语言环境：zh_CN.UTF-8

## 环境变量输出演示

```
[root@itheima ~]# echo $PATH
/usr/local/jdk1.8.0_162/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
[root@itheima ~]# echo $HOME
/root
[root@itheima ~]# echo $SHELL
/bin/bash
```

```
[root@itheima ~]# echo $HISTFILE      ← 查看存储执行命令历史文件
/root/.bash_history
[root@itheima ~]# cat /root/.bash_history ← 查看历史执行命令
clear
ssh-copy-id itcast
ssh itcast
clear
clear
rpm -qa
cd /soft
clear
ll
```

```
[root@itheima ~]# echo $PWD
/root
[root@itheima ~]# echo $OLDPWD

[root@itheima ~]# echo $HOSTNAME
itheima
[root@itheima ~]# echo $HOSTTYPE
x86_64
[root@itheima ~]# echo $LANG
zh_CN.UTF-8
[root@itheima ~]# █
```

## 小结

### 1.系统环境变量是什么？

是系统提供的环境变量, 通过加载Shell配置文件中变量数据共享给Shell程序使用

### 2.环境变量的分类？

系统级环境变量, Shell环境加载全局配置文件中定义的变量

用户级环境变量, Shell环境加载个人配置文件中定义的变量

### 3.env与set区别

env用于查看系统环境变量

set用于查看系统环境变量+自定义变量+函数

### 4.常用环境变量

变量名称	含义
PATH	命令搜索的目录路径, 与windows的环境变量PATH功能一样
LANG	查询系统的字符集
HISTFILE	查询当前用户执行命令的历史列表

# Shell变量：自定义变量

## 目标

理解自定义变量的分类

能够自定义变量进行增\删\改\查

## 自定义变量介绍

就是自己定义的变量

## 分类

1. 自定义局部变量
2. 自定义常量
3. 自定义全局变量

## 自定义局部变量

### 介绍

就是定义在一个脚本文件中的变量, 只能在这个脚本文件中使用的变量, 就是局部变量

## 定义与使用

定义语法

```
var_name=value
```

变量定义规则

1. 变量名称可以有字母,数字和下划线组成, 但是不能以数字开头
2. 等号两侧不能有空格
3. 在bash环境中, 变量的默认类型都是字符串类型, 无法直接进行数值运算
4. 变量的值如果有空格, 必须使用双引号括起来
5. 不能使用Shell的关键字作为变量名称

演示

```
[root@itheima ~]# var1=播仔  
[root@itheima ~]# var2=播妞
```

## 查询变量值语法

```
# 语法1: 直接使用变量名查询
$var_name
# 语法2: 使用花括号
${var_name}
# 区别: 花括号方式适合拼接字符串
```

### 演示

```
[root@itheima ~]# echo $var1
播仔
[root@itheima ~]# echo $var2
播妞
```

```
[root@itheima ~]# echo "My name is ${var2}Style"
My name is 播妞Style
[root@itheima ~]#
```

注意: 如果 "My name is \${var2}Style" 中 `$var2` 不带花括号, 系统会认为获取 `$var2Style` 变量数据, 这个变量不存在就获取不到数据, 执行效果如下

```
[root@itheima ~]# echo "My name is $var2Style"
My name is
[root@itheima ~]#
```

结论: 推荐大家使用花括号才是编程好习惯

## 变量删除

### 语法

```
unset var_name
```

### 演示

```
[root@itheima ~]# var1=播仔
[root@itheima ~]# echo $var1
播仔
[root@itheima ~]# unset var1
[root@itheima ~]# echo $var1
[root@itheima ~]#
```

← 删除变量

## 自定义常量

### 介绍

就是变量设置值以后不可以修改的变量叫常量, 也叫只读变量

### 语法



```
readonly var_name
```

演示

```
[root@itheima ~]# var3=itcast
[root@itheima ~]# readonly var3
[root@itheima ~]# var3=itheima
bash: var3: 只读变量
[root@itheima ~]#
```

设置var3为只读变量

修改只读变量报错

## 自定义全局变量

### 父子Shell环境介绍

例如: 有2个Shell脚本文件 A.sh 和 B.sh

如果 在A.sh脚本文件中执行了B.sh脚本文件, 那么A.sh就是父Shell环境, B.sh就是子Shell环境

### 自定义全局变量介绍

就是在当前脚本文件中定义全局变量, 这个全局变量可以在当前Shell环境与子Shell环境中都可以使用

### 自定义全局变量语法

```
export var_name1 var_name2
```

### 案例需求

测试全局变量在子Shell中是否可用, 在父Shell中是否可用

### 案例实现步骤

1. 创建2个脚本文件 demo2.sh 和 demo3.sh
2. 编辑demo2.sh
  - 命令1: 定义全局变量VAR4
  - 命令2: 执行demo3.sh脚本文件
3. 编辑demo3.sh
  - 输出全局变量VAR4
4. 执行demo2.sh脚本文件

### 案例演示

1. 创建demo2.sh和demo3.sh文件

```
[root@itheima ~]# touch demo2.sh
[root@itheima ~]# touch demo3.sh
[root@itheima ~]#
```

2. 编辑demo2.sh, 里面定义变量VAR4并设置为全局, 并里面执行demo3.sh脚本文件

```
vim demo2.sh
```

```
#!/bin/bash
VAR4=itheima
export VAR4
echo "demo2.sh中输出VAR4变量:${VAR4}"
bash demo3.sh
```

3. 编辑demo3.sh, 里面打印VAR4

```
vim demo3.sh
```

```
#!/bin/bash
echo "在demo3.sh中打印VAR4变量:${VAR4}"
```

4. 执行脚本文件demo2.sh, 观察打印VAR4效果

```
[root@itheima ~]# sh demo2.sh
demo2.sh中输出VAR4变量:itheima
在demo3.sh中打印VAR4变量:itheima
[root@itheima ~]#
```

5. 执行脚本文件后, 在交互式Shell环境打印VAR4, 观察打印VAR4效果

```
[root@itheima ~]# echo "在父Shell环境中使用VAR4变量:${VAR4}"
在父Shell环境中使用VAR4变量:
[root@itheima ~]#
```

## 结论

全局变量在当前Shell环境与子Shell环境中可用, 父Shell环境中不可用

## 小结

### 自定义变量的分类

自定义局部变量: 就是在脚本文件内部使用 `var_name=value`

自定义常量: 不可以修改值的变量, `readonly var_name`

自定义全局变量: 设置变量在当前脚本文件中与子Shell环境可以使用的变量, `export var_name`

自定义变量进行增\删\改\查

定义和修改: `var_name=value`

查询: `${var_name}` 或 `$var_name`

删除: `unset var_name`

## Shell变量：特殊变量

### 目标

能够说出常用的特殊变量有哪些

### 特殊变量：\$n

#### 语法

`$n`

#### 含义

用于接收脚本文件执行时传入的参数

`$0` 用于获取当前脚本文件名称的

`$1~$9`，代表获取第一输入参数到第9个输入参数

第10个以上的输入参数获取参数的格式：`${数字}`，否则无法获取

#### 执行脚本文件传入参数语法

`sh` 脚本文件 输入参数1 输入参数2 ...

### 案例需求

创建脚本文件demo4.sh文件, 并在脚本文件内部执行打印脚本文件名字, 第一个输入参数, 第二个输入参数

#### 实现步骤

1. 创建脚本文件demo4.sh
2. 编辑demo4.sh的文件内容

```
# 命令1: 打印当前脚本文件名字
# 命令2: 打印第1个输入参数
# 命令3: 打印第2个输入参数
# 命令4: 打印第10个输入参数
```

3. 执行脚本文件demo4.sh

## 演示

1. 创建demo4.sh文件

```
[root@itheima ~]# touch demo4.sh
[root@itheima ~]#
```

2. 编辑demo4.sh文件, 输出脚本文件名称\第一个输入参数\第二个输入参数

```
[root@itheima ~]# vim demo4.sh
```

```
#!/bin/bash
echo "脚本文件名:${0}"
echo "第一个输入参数:${1}"
echo "第二个输入参数:${2}"
```

3. 执行demo4.sh文件,输入输出参数itcast itheima的2个输入参数, 观察效果

```
[root@itheima ~]# sh demo4.sh itcast itheima
脚本文件名:demo4.sh
第一个输入参数:itcast
第二个输入参数:itheima
```

## 特殊变量: \$#

### 语法

```
$#
```

### 含义

获取所有输入参数的个数

### 案例需求

在demo4.sh中输出输入参数个数

### 演示

编辑demo4.sh, 输出输入参数个数

```
[root@itheima ~]# vim demo4.sh
```

```
#!/bin/bash
echo "脚本文件名:${0}"
echo "第一个输入参数:${1}"
echo "第二个输入参数:${2}"
echo "输入参数个数:${#}"
```

执行demo4.sh传入参数itcast, itheima, 播仔 看效果

```
[root@itheima ~]# sh demo4.sh itcast itheima 播仔
脚本文件名:demo4.sh
第一个输入参数:itcast
第二个输入参数:itheima
输入参数个数:3
[root@itheima ~]#
```

## 特殊变量： `$*`、 `$@`

### 语法

```
$*
$@
# 含义都是获取所有输入参数，用于以后输出所有参数
```

`$*` 与 `$@` 区别

1. 不使用双引号括起来，功能一样  
`$*`和`$@`获取所有输入参数，格式为：`$1 $2 ... $n`
2. 使用双引号括起来  
`"$"`获取的所有参数拼接为一个字符串，格式为：`"$1 $2 ... $n"`  
`"$@"`获取一组参数列表对象，格式为：`"$1" "$2" ... "$n"`  
使用循环打印所有输入参数可以看出区别

循环语法

```
for var in 列表变量
do      # 循环开始
    命令 # 循环体
done    # 循环结束
```

## 案例需求

在demo4.sh中循环打印输出所有输入参数, 体验 `$*` 与 `$@` 的区别

## 实现步骤

编辑demo4.sh脚本文件

```
# 增加命令：实现直接输出所有输入后参数
# 增加命令：使用循环打印输出所有输入参数
```

## 演示

### 1. 编辑demo4.sh文件

```
[root@itheima ~]# vim demo4.sh
```

### 2. 直接输出所有输入参数, 与循环方式输出所有输入参数(使用双引号包含 `$*` 与 `$@` )

```
#!/bin/bash
# 命令1：打印当前脚本文件名字
echo "当前脚本文件名称:$0"

# 命令2：打印第1个输入参数
echo "第一个输入参数:$1"

# 命令3：打印第2个输入参数
echo "第二个输入参数:$2"

# 命令4：打印第10个输入参数
echo "第十个输入参数不带花括号获取:$10"
echo "第十个输入参数带花括号获取:${10}"

# 命令5 打印所有输入参数的个数
echo "所有输入参数个数:${#}"

# 增加命令：实现直接输出所有输入后参数
echo '使用$*直接输出:$*'
echo '使用$@直接输出:$@'

# 增加命令：使用循环打印输出所有输入参数
echo '循环遍历输出$*所有参数'
for item in "$*"
do
    echo $item
done
echo '循环遍历输出$@所有参数'
for item in "$@"
do
    echo $item
done
```

### 3. 运行观察区别

```
[root@itheima ~]# sh demo4.sh itheima itcast Shell
当前脚本文件名称:demo4.sh
第一个输入参数:itheima
第二个输入参数:itcast
第十个输入参数不带花括号获取:itheima0
第十个输入参数带花括号获取:
所有输入参数个数:3
使用$*直接输出:itheima itcast Shell
使用$@直接输出:itheima itcast Shell
循环遍历输出$*所有参数
itheima itcast Shell
循环遍历输出$@所有参数
itheima
itcast
Shell
```

## 特殊变量：\$?

### 语法

\$?

### 含义

用于获取上一个Shell命令的退出状态码, 或者是函数的返回值

每个Shell命令的执行都有一个返回值, 这个返回值用于说明命令执行是否成功

一般来说, 返回0代表命令执行成功, 非0代表执行失败

### 演示

输入一个正确命令, 再输出 \$?

```
[root@itheima ~]# echo "hello"
hello
[root@itheima ~]# echo $?
0
```

输入一个错误命令, 在输出 \$?

```
[root@itheima ~]# ee
-bash: ee: 未找到命令
[root@itheima ~]# echo $?
127
```

## 特殊变量：\$

### 语法

\$\$

### 含义

用于获取当前Shell环境的进程ID号

## 演示

查看当前Shell环境进程编号

```
ps -aux|grep bash
```

```
[root@itheima ~]# ps -aux | grep bash
root      1990  0.0  0.3 116452  3136 pts/0    Ss   09:26   0:00 -bash
root      2230  0.0  0.0 112676   976 pts/0    R+   11:47   0:00 grep --color=auto bash
[root@itheima ~]#
```

输出 \$\$ 显示当前shell环境进程编号

```
[root@itheima ~]# echo $$
1990
```

## 小结

常用的特殊符号变量如下

特殊变量	含义
\$n	获取输入参数的 \$0，获取当前Shell脚本文件名字 \$1~\$9，获取第一个输入参数到第九个输入参数 \${10} 获取10和10以上的参数需要使用花括号
\$#	获取所有输入参数的个数
* 与 @	获取所有输入参数数据 区别: 如果不使用双引号, 功能一样, 获取所有参数数据为一个字符串, 如果使用了双引号, @ 获取的就是参数列表对象, 每个参数都是一个独立字符串,
?	获取上一个命令的退出状态码, 一般来说0代表命令成功, 非0代表执行失败
\$\$	获取当前shell环境进程的ID号

# Shell环境变量深入：自定义系统环境变量

## 目标

能够自定义系统级环境变量



# 全局配置文件/etc/profile应用场景

当前用户进入Shell环境初始化的时候会加载全局配置文件/etc/profile里面的环境变量, 供给所有Shell程序使用  
以后只要是所有Shell程序或命令使用的变量, 就可以定义在这个文件中

## 案例演示

### 需求

/etc/profile定义存储自定义系统级环境变量数据

### 创建环境变量步骤

1. 编辑/etc/profile全局配置文件

```
# 增加命令: 定义变量VAR1=VAR1 并导出为环境变量  
# 扩展: vim里面的命令模式使用G快速定位到文件末尾位置, 使用gg定位到文件首行位置
```

2. 重载配置文件/etc/profile, 因为配置文件修改后要立刻加载里面的数据就需要重载, 语法

```
source /etc/profile
```

3. 在Shell环境中读取系统级环境变量VAR1

### 创建环境变量演示

编辑/etc/profile文件

```
vim /etc/profile
```

添加设置变量VAR1=VAR1并导出成为环境变量, 在/etc/profile文件末尾添加如下命令

```
# 创建环境变量  
VAR1=VAR1  
export VAR1
```

```
export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE HISTCONTROL

# By default, we want umask to get set. This sets it for login shell
# Current threshold for system reserved uid/gids is 200
# You could check uidgid reservation validity in
# /usr/share/doc/setup-*/uidgid file
if [ $UID -gt 199 ] && [ "`/usr/bin/id -gn`" = "`/usr/bin/id -un`" ]; then
    umask 002
else
    umask 022
fi

for i in /etc/profile.d/*.sh ; do
    if [ -r "$i" ]; then
        if [ "${-#*i}" != "$-" ]; then
            . "$i"
        else
            . "$i" >/dev/null
        fi
    fi
done

unset i
unset -f pathmunge
#set java environment
JAVA_HOME=/usr/local/jdk1.8.0_162
CLASSPATH=.:$JAVA_HOME/lib
PATH=$JAVA_HOME/bin:$PATH
export JAVA_HOME CLASSPATH PATH
# 创建环境变量
VAR1=VAR1
export VAR1
```

3、保存/etc/profile退出

4、重新加载/etc/profile文件数据更新系统环境变量

```
source /etc/profile
```

注意：如果这一步不执行，无法读取更新的环境变量

3、输出环境变量VAR1

```
echo $VAR1
```

```
[root@itheima ~]# echo $VAR1
VAR1
[root@itheima ~]#
```

## 小结

如何自定义系统级环境变量

1. 系统级全局配置文件: /etc/profile
2. 设置环境变量: `export var_name=value`, 注意环境变量建议变量名全部大写
3. 修改了/etc/profile文件后, 要立刻加载修改的数据需要重载配置文件: `source /etc/profile`

# Shell环境变量深入：加载流程原理介绍

## 目标

1. 能够说出交互式Shell与非交互式Shell
2. 能够说出登录Shell与非登录Shell环境

## Shell工作环境介绍

用户进入linux系统就会初始化Shell环境, 这个环境会加载全局配置文件和用户个人配置文件中环境变量. 每个脚本文件都有自己的Shell环境

## shell工作环境分类

### 交互式与非交互式shell

交互式Shell

与用户进行交互, 互动. 效果就是用户输入一个命令, Shell环境立刻反馈响应.

非交互式Shell

不需要用户参与就可以执行多个命令. 比如一个脚本文件含有多个命令, 直接执行并给出结果

### 登录Shell与非登录Shell环境

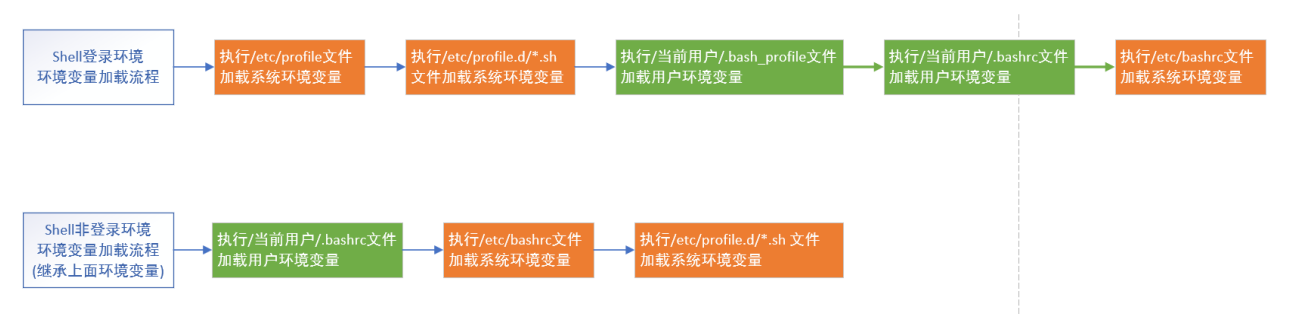
类型名称	含义
shell登录环境	需要用户名\密码登录的Shell环境
shell非登录环境	不需要用户名,密码进入的Shell环境 或 执行脚本文件

注意：不同的工作环境加载环境变量流程不一样

## 环境变量初始化流程

- 1.全局配置文件 /etc/profile /etc/profile.d/\*.sh /etc/bashrc
- 2.个人配置文件 当前用户/.bash\_profile 当前用户/.bashrc

环境变量加载初始化过程



小结

1. 能够说出交互式Shell与非交互式Shell

- 交互式Shell: 就是需要用户参与互动的Shell环境, 效果用户输入一个命令, 环境就立刻响应结果
- 非交互式Shell: 只执行命令, 不需要用户的参与

2. 能够说出登录Shell与非登录Shell环境

- 登录Shell环境: 要以用户名与密码登录到系统默认采用登录Shell环境
- 非登录Shell环境: 不使用用户名与密码进入linux系统的Shell环境

Shell环境变量深入：加载流程测试

目标

- 理解Shell环境变量的加载流程测试
- 能够知道环境变量应该配置在哪里

切换Shell环境执行脚本文件介绍

在执行一个脚本文件时可以指定具体Shell环境进行执行脚本文件, 这个就是切换Shell环境执行脚本

Shell登录环境执行脚本文件语法

```
sh/bash -l/--login 脚本文件
```

含义: 先加载Shell登录环境流程初始化环境变量, 再执行脚本文件

Shell非登录环境变量执行脚本文件语法

```
bash # 加载Shell非登录环境
sh/bash 脚本文件 # 直接执行脚本文件
```

含义: 先执行加载Shell非登录环境流程初始化环境变量, 再执行脚本文件

## 测试案例

### 需求

Shell登录环境会运行/etc/profile

Shell非登录环境会运行/.bashrc

在/etc/profile与/当前用户/.bashrc文件分别设置环境变量数据, 然后在shell脚本文件中输出环境变量数据, 最后切换不同环境执行shell脚本文件观察验证上面的流程运行

### 分析

1. 清理工作, 清理/etc/profile文件中VAR1环境变量进行删除, 并且重载这个配置文件
2. 编辑/etc/profile, 增加环境变量VAR1=VAR1
3. 编辑/root/.bashrc, 增加环境变量VAR2=VAR2
4. 创建demo1.sh文件, 读取环境变量数据进行打印

```
# 输出环境变量VAR1
# 输出环境变量VAR2
```

5. 以Shell非登录环境执行demo1.sh脚本文件, 观察只会输出VAR2, 不会输出VAR1
6. 以Shell登录环境执行demo1.sh脚本文件, 观察会输出VAR2和VAR1

### 演示

编辑/etc/profile文件

```
vim /etc/profile
```

编辑添加如下内容, 保存退出

```
VAR1=VAR1
export VAR1
```

在root目录下,编辑.bashrc文件

```
vim .bashrc
```

```
[root@itheima ~]# vim .bashrc
```

编辑添加如下最后2行内容, 保存退出

```
# .bashrc

# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
VAR2=VAR2
export VAR2
```

创建文件demo1.sh

```
touch demo1.sh
```

编辑文件demo1.sh, 添加如下内容

```
#!/bin/bash
echo $VAR1
echo $VAR2
~
~
```

直接执行脚本文件

```
bash demo1.sh
```

```
[root@itheima ~]# bash demo1.sh

[root@itheima ~]#
```

直接执行脚本文件, 即没有加载登录Shell环境变量, 也没有加载非登录Shell环境变量

先加载非登录Shell环境变量, 然后执行脚本文件

```
bash
bash demo1.sh
```

```
[root@itheima ~]# bash demo1.sh
VAR2
[root@itheima ~]#
```

Shell非登录环境会加载文件 `当前用户/.bashrc` 的环境变量数据

所以这里会输出VAR2的环境变量数据

先加载登录Shell环境变量, 然后执行脚本文件

```
bash -l demo1.sh
```

```
[root@itheima ~]# bash -l demo1.sh
VAR1
VAR2
[root@itheima ~]#
```

Shell登录环境会加载文件 `etc/profile` 和 当前用户 `/.bashrc` 的环境变量数据

所以这里会输出VAR1和VAR2的环境变量数据

## 小结

### 1、Shell环境变量初始化加载原理过程

分类	初始化环境变量过程执行文件顺序
shell登录环境初始化过程	<code>/etc/profile--&gt; /etc/profile.d/*.sh--&gt; ~/.bash_profile--&gt; ~/.bashrc--&gt; /etc/bashrc</code>
shell非登录环境初始化过程	<code>~/.bashrc--&gt; /etc/bashrc--&gt; /etc/profile.d/*.sh</code>

### 2、那么以到底将环境变量定义到哪里呢？`/etc/profile`与`/etc/bashrc`的区别？

需要登录的执行的shell脚本读取的环境变量配置在：`/etc/profile`、当前用户`/.bash_profile`

不需要登录的用户执行的shell脚本读取的环境变量配置在：当前用户`/.bashrc`、`/etc/bashrc`

# Shell环境变量深入：识别Shell环境类型

## 目标

理解如何识别shell登录环境与非登录环境

## 语法

使用`$0`识别环境语法

```
echo $0
```

输出 `-bash` 代表：shell登录环境

输出 `bash` 代表：shell非登录环境

注意：这个 `$0` 环境变量如果用在子shell中(shell脚本文件)输出Shell脚本本身的文件名

bash命令语法

```
bash
```

bash命令：用于切换为Shell非登录环境

## 分析

- 1、直接登录系统为shell登录环境输出 \$0 观察输出信息效果
- 2、使用 bash 命令切换为shell非登录环境输出 \$0 观察输出信息效果
- 3、创建test.sh脚本文件，编辑添加输出 \$0 ,编程保存完成后执行test.sh脚本文件观察输出信息效果

## 演示

直接登录linux系统使用如下命令效果

```
[root@itheima ~]# echo $0
-bash
[root@itheima ~]# bash
[root@itheima ~]# echo $0
bash
[root@itheima ~]#
```

输出-bash代表是【shell登录环境】

输出bash代表是【shell非登录环境】

bash命令将当前环境转换为Shell非登录环境

## 小结

- 1、如何识别shell登录环境与非登录环境？

\$0 用于获取当前Shell环境的类型， bash代表Shell非登录环境， -bash 代表Shell登录环境  
# \$0不能在脚本文件中使用， 因为代表获取脚本文件名字

# Shell环境变量深入：详细切换Shell环境

## 目标

理解切换shell环境的命令

## 切换shell环境命令介绍

1. 直接登录加载shell登录环境
2. su切换用户加载Shell登录与Shell非登录环境
3. bash加载Shell登录与Shell非登录环境



# 切换Shell环境命令演示

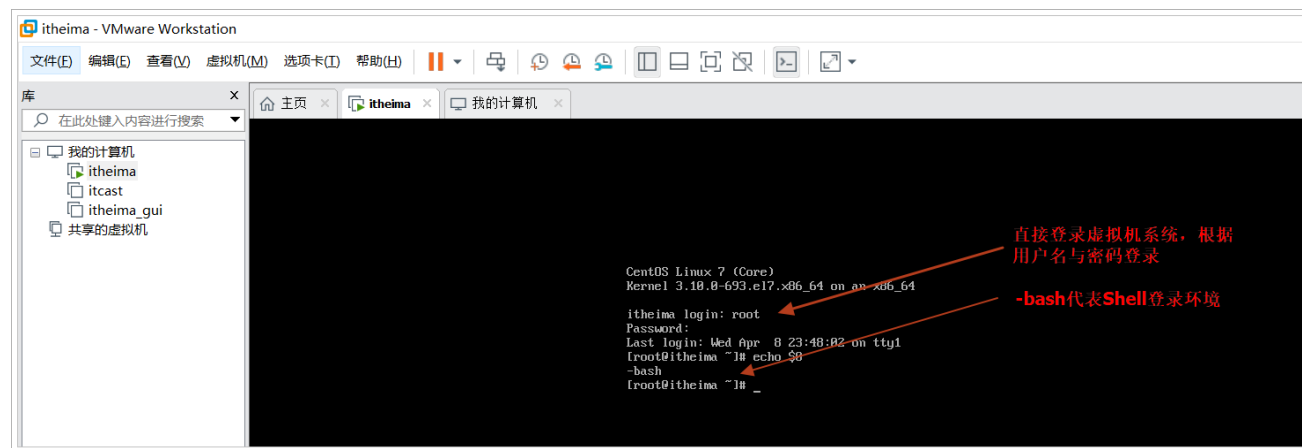
## 切换环境方式1：直接登录系统

### 介绍

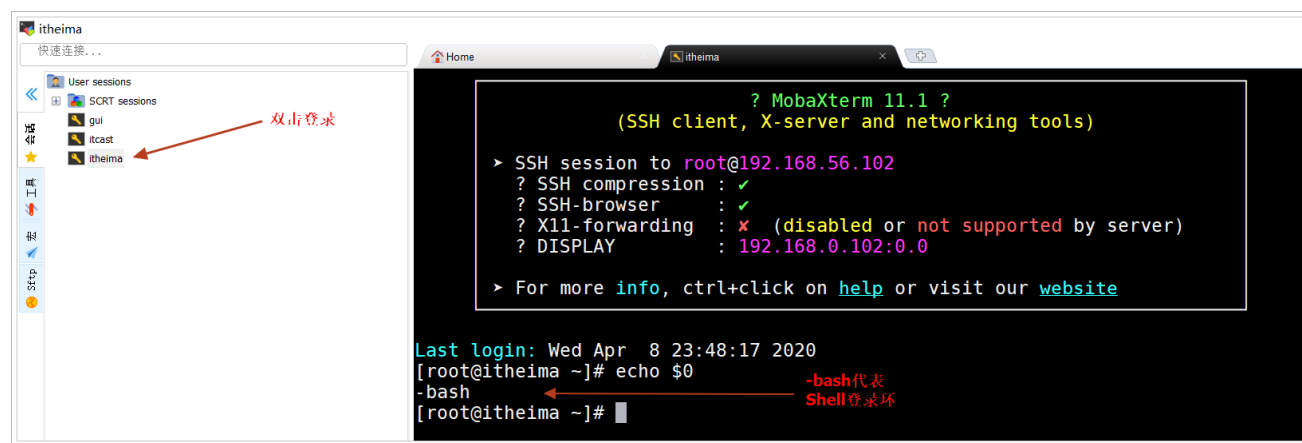
直接在虚拟机上使用用户名与密码登录linux系统或使用客户端直接连接远程linux系统

### 演示

虚拟机本地直接登录演示



客户端远程采用SSH登录演示



## 切换环境方式2：su切换用户登录

### 命令

#### 语法1

```
su 用户名 --login
或
su 用户名 -l
# 切换到指定用户，加载Shell登录环境变量
```

#### 语法2

```
su 用户名
# 切换到指定用户， 加Shell非登录环境变量
```

## 分析步骤

- 1、创建普通用户userA
- 2、切换到用户userA，使用-l加载Shell登录环境变量，输出环境变量\$0，观察输出-bash
- 4、使用exit退出userA
- 5、切换到用户userA，加载Shell非登录环境变量，输出环境变量\$0，观察输出bash

## 演示

创建普通用户userA

```
useradd -m userA
```

```
[root@itheima ~]# useradd -m userA
[root@itheima ~]#
```

以Shell登录环境执行切换到用户userA，输出环境变量\$0，输出 -bash 说明当前为Shell登录环境

```
[root@itheima ~]# su userA --login
[userA@itheima ~]$ echo $0
-bash
[userA@itheima ~]$
```

使用exit退出userA

```
[userA@itheima ~]$ exit
登出
```

以Shell非登录环境执行切换到用户userA，输出环境变量\$0，输出 bash 说明当前为Shell非登录环境

```
[root@itheima ~]# su userA
[userA@itheima root]$ echo $0
bash
[userA@itheima root]$
```

## 切换环境方式3：bash切换

### 命令

语法1:

```
bash # 加载【Shell非登录环境】
```

语法2:

```
bash -l shell脚本文件 / bash --login shell脚本文件
sh -l shell脚本文件 / sh --login shell脚本文件
# 先加载【Shell登录环境】然后运行指定Shell脚本文件
```

## 分析

使用bash执行test.sh脚本文件，发生错误说明当前环境为Shell非登录环境

## 演示

```
[userA@itheima root]$ bash test.sh
test.sh
test.sh: 第 3 行:logout: 不是登录shell: 使用 `exit'
[userA@itheima root]$
```

#####