# Shell环境及特性

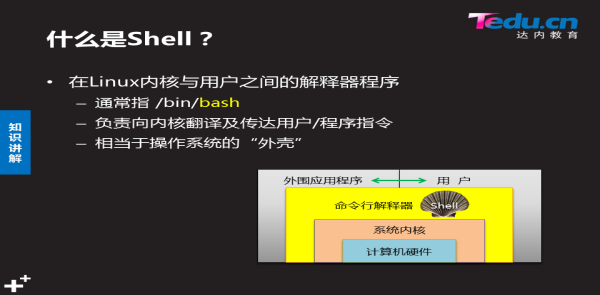
## 什么是shell

在Linux内核与用户之间的解释器程序

--通常指/bin/bash

--负责向内核翻译及传达用户/程序指令

--相当与操作系统的外壳



## Shell的使用方式

交互式 ----命令行

--人工干预,智能化程度高

--逐条解释执行,效率低

非交互式 ----脚本

--需要提前设计,智能化难度大

--批量执行,效率高

--方便在后台悄悄进行

## 常见的shell程序种类

如何切换shell环境

--通过usermod,chsh更换登录shell

--手动执行目标shell程序

查看解释器配置文件 : /etc/shells

## Bash基本特性

命令行环境回顾

--快捷键,tab补全

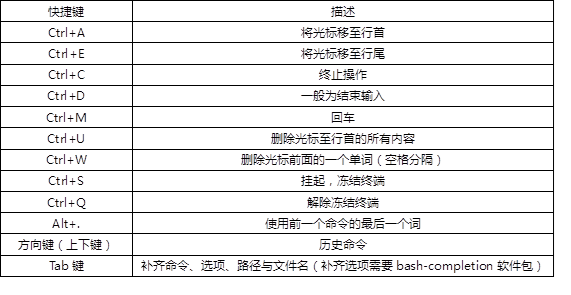
--命令历史

--命令别名

--标准输入输出

--重定向

--管道操作



# 历史命令

## 存放位置及数量

默认记录1000条

--保存位置:~/.bash\_history

--控制历史命令的数量:/etc/profile

## 操作历史命令

History工具

--history:查看历史记录命令列表

--history -c : 清空历史命令

调用历史命令

--!78 : 执行历史记录中的第78条命令

--!str : 执行最近一次以str开头的历史命令

# I/O设备与重定向

## 交互式硬件设备

标准输入:从该设备接收用户输入的数据

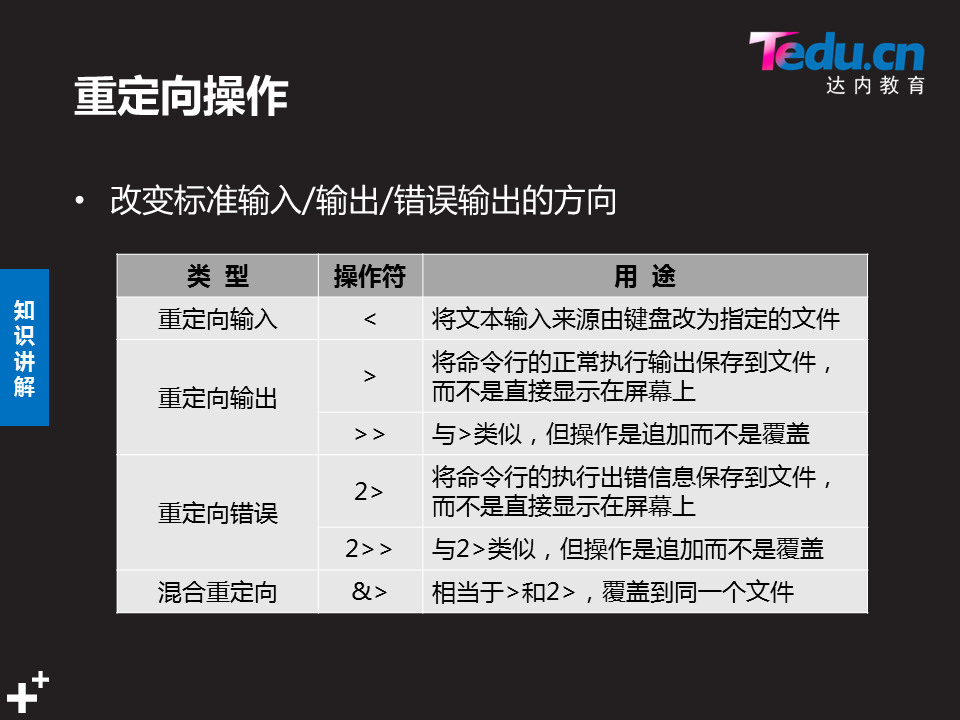
标准输出:通过该设备向用户输入数据

标准错误:通过该设备报告执行中的出错信息



## 重定向操作

改变标准输出/输入/错误输出的方向



# 第一个shell脚本

## 什么是shell脚本

提前写好可执行语句,能够完成特定任务的文件

--顺序,批量化处理

--解释型程序

## Shell版HelloWord的诞生

脚本创建三步骤

--1.新建文本文件

--2.添加可执行文件的脚本语句(命令行)

--3.添加x执行权限

# 脚本构成及执行

## 规范的脚本构成

#!脚本声明(使用哪种解释器)

#注释信息(步骤,思路,用途,变量含义等)

可执行语句

## 脚本的执行方式

方法一,作为命令字

--指定脚本文件的路径,前提是有x权限

方法二,作为参数

--sh 脚本文件路径

--source 脚本文件路径

--. 脚本路径

## 调试shell脚本

主要途径

--直接观察执行中的输出,报错信息

--通过sh -x 开启调试模式

--在可能出错的地方设置echo

# 变量的设置和取消

## 什么是变量

以固定名称存放,可能会变化的值

--提高脚本对人物需求,运行环境变化的适应能力

--方便在脚本中中重复使用

## 定义/赋值/查看变量

定义/赋值变量

-变量名=变量值

变量名不可以数字开头 不可使用特殊字符 只可以字母,数字,下划线

查看变量

--引用变量值: $变量名

--查看变量值: echo $变量名 , echo ${变量名}

取消变量

变量的失效

--退出定义变量的shell环境时,变量会自动失效

--也可手动取消:unset 变量名

# 变量的种类

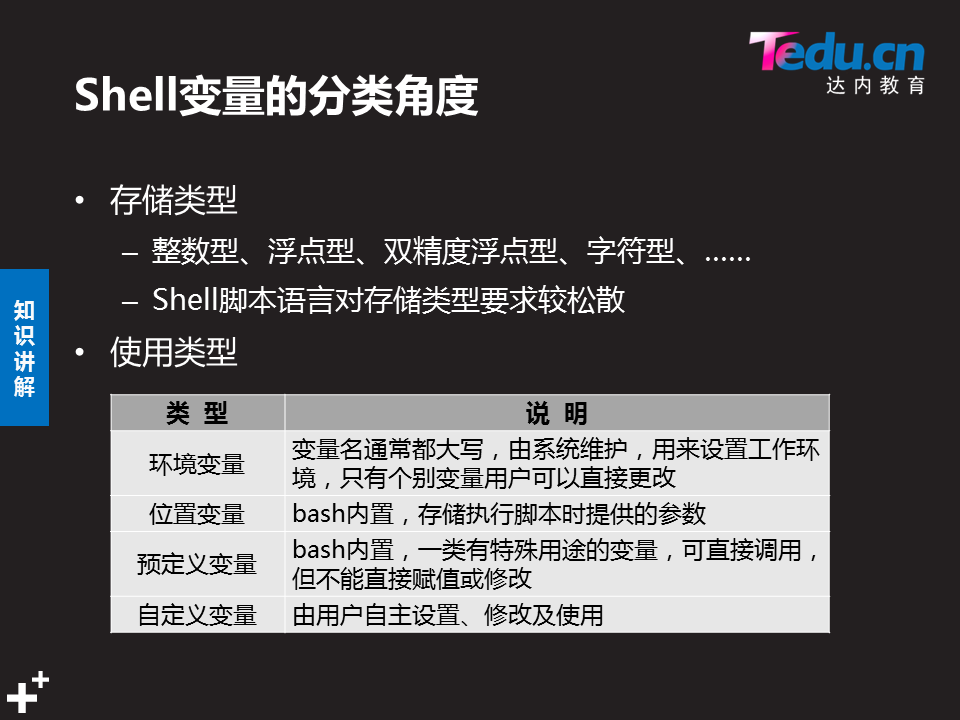
## Shell变量的分类角度

存储类型

--整数型,浮点型,双精度型,字符型

--shell脚本语言对储存类型要求较松散

使用类型



## 环境变量

配置文件

--/etc/profile ~/.bash\_profile

相关操作

--env:列出所有的环境变量

--set:列出所有变量

常见的环境变量

--PWD,PATH,USER,LOGNAME,UID

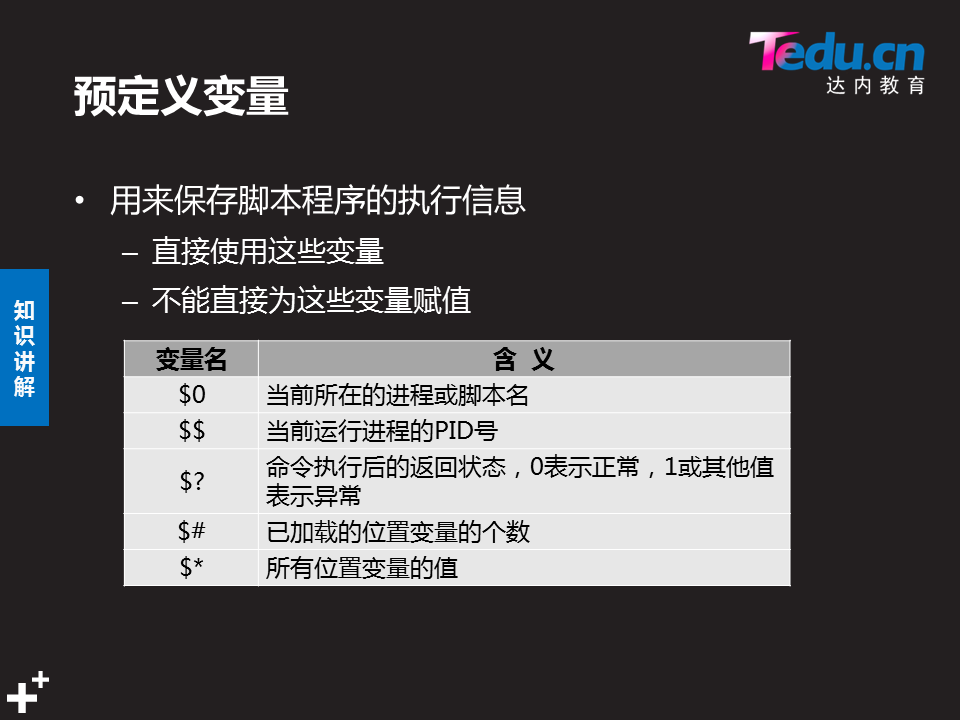
--SHELL,HOME,PS1,PS2.....

## 预定义变量

用来保存脚本程序的执行信息

--直接使用这些变量

--不能直接为这些变量赋值



## 位置变量

在执行脚本是提供的命令行参数

--表示为$n , n为序号

--$1,$2,....${10},${11},...

# 变量值及范围控制

## 扩展赋值操作

区分三种定界符

--双引号 ” ” : 允许扩展,以$引用其他变量

--单引号 ‘ ‘ : 禁用扩展,即便$也视为普通字符

--反撇号 ` ` : 将命令的执行输出作为变量值与$()相同

## Read标准输出取值

Read从键盘读入变量值完成赋值

--格式 : read [ -p “提示信息” ] 变量名

-- -p可选,-t可指定超时秒数

终端显示控制

-- stty -echo : 关闭终端输出(无显示)

-- stty echo : 恢复终端输出(显示)

-- -s的用处和stty一样

## 变量的作用范围

局部变量

--新定义的变量默认只在当前shell环境中有效

--无法在子shell环境中使用

全局变量

--全局变量在前shell及子shell环境中均有效

--使用export可将局部变量声明为全局变量

export 局部变量名[=变量名].... : 为局部变量添加全局属性

export -n 全局变量名... ... : 取消指定全局变量

# 整数运算

## Expr运算工具

计算指定的表达式,并输出结果

--格式:expr 整数1 运算符 整数2...

--乘法操作应采用 \\* 转义,避免被视作shell通配符

## $[]算式替换

使用$[]或$(())表达式

--格式: $[整数1 运算符 整数2 ...]

--乘法操作\*无需转义,运算符两侧可以无空格

--引用变量可省略$符号

--计算结果替换表达式本身,可结合echo命令输出

## let命令

可以进行运算覆盖赋值

let 赋值变量=变量/常量的运算

let的结果不会显示

# 小数运算

## 整数运算的局限性

Bash内建机制仅支持整数值运算

--expr命令,$[]算式替换 不支持有小数的运算

## 使用bc实现小数运算

多数Linux系统默认安装此工具

--支持高精度的数值运算

--直接运行bc可以进入交互式运算页面,quit退出

--设置 scale=n 可约束小数位

结合管道向bc发送表达式

--多个表达式以分号分隔

--通过echo命令+管道传递要计算的表达式

## 小数值的比较

基本用法

--echo “数值1 比较符 数值2” | bc

--如果表达式成立,则返回的计算结果为1,否则返回0

--常见比较操作: >,>=,<,<=,==,!=

# 测试操作概述

## Shell脚本的智能化

使shell脚本获得识别能力

为命令的执行提供最直接的识别依据

--文件或目录的读/写等状态

--数值的大小

--字符串是否匹配

--多条件组合

## Test测试操作

语法格式

--test 选项 参数

--[ 选项 参数 ]

# 常见的测试操作

## 字符串比较

[ 操作符 字符串 ]

-z 字符串的值为空

-n 字符串的值不为空

== 两个字符相同

!= 两个字符不相同

## 整数比较

[ 整数值1 操作符 整数值2 ]

-eq 等于(Equal)

-ne 不等于(Not Equal)

-ge 大于或等于(Greater or Equal)

-le 大于或等于(Lesser or Equal)

-gt 大于(Greater Then)

-lt 小于(Lesser Then)

## 文件状态测试

[ 操作符 文件或目录 ]

-e 判断对象是否存在(Exist),若存在则结果为真

-d 判断对象是否为目录(Directory),是则结果为真

-f 判断对象是否为一般文件(File),是则结果为真

-r 判断对象是否有可读(Read)权限,是则结果为真

-w 判断对象是否有可写(Write)权限,是则结果为真

-x 判断对象是否有可执行(eXcute)权限,是则结果为真

# 组合多个条件

## 逻辑分隔操作

主要用法:

--命令1 操作符 命令2

--[ 条件1 ] 操作符 [ 条件2 ]

&& 给定条件必须都成立,整个测试结果才为真

|| 只要其中一个条件成立,则整个测试结果为真

# If选择结构

## 单分支结构

语法结构及特点

当条件成立时执行命令序列

否则,不执行任何操作

if [ 条件 ] ; then

命令

fi

## 双分支结构

if [ 条件 ] ; then

命令

else

命令

fi

## 多分支结构

if [ 条件1 ] ; then

命令

elif [ 条件2 ] ; then

命令

.

.

else

命令

fi

# For循环

for 变量名 in 值列表

do

命令

done

C语言风格

for((变量;条件;步长)){

命令

}

# While循环

while [ 条件 ]

do

命令

done

# Case语句

case 变量值 in

模式一)

命令1;;

模式二)

命令2;;

......

\*)

默认命令

esac

# Shell函数

## 关于函数

--在shell环境中,将一些需要重复使用的操作,定义为公共的语句快,即可称为函数

--使脚本代码更简洁,增强易读性

--提高脚本的执行效率

## 函数的定义与调用

function 函数名 {

命令

}

函数名(){

命令

}

调用已定义函数

--格式: 函数名

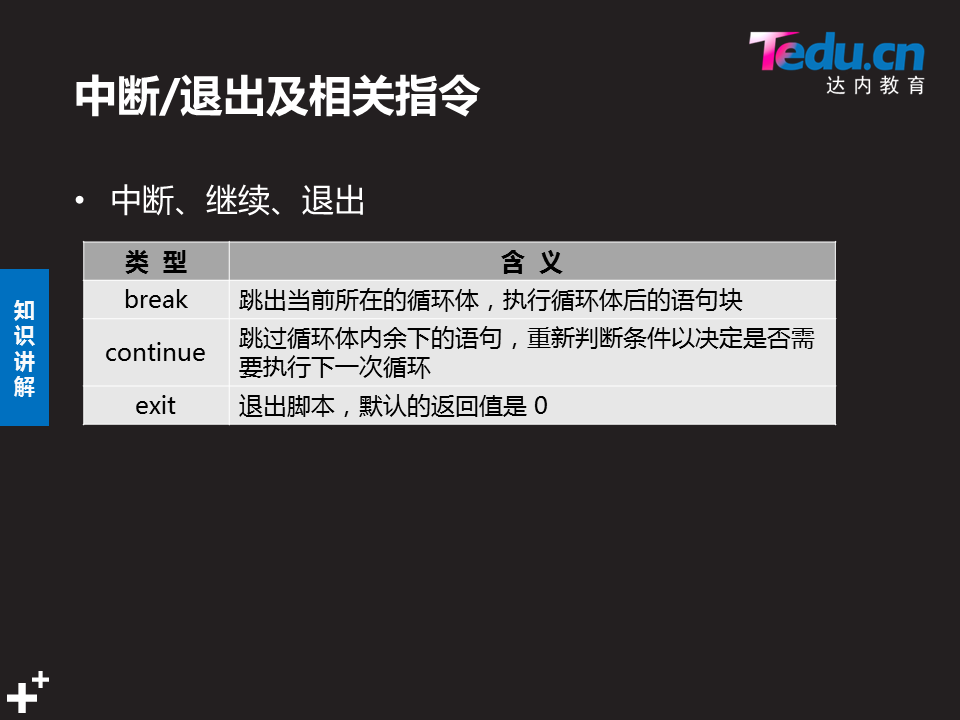
--先定义了才能调用

函数传值

--格式: 函数名 值1 值2

--传递的值作为函数的”位置参数”

## 中断/退出及相关指令



# 字符串处理

## 子串截取

方法一,使用${}表达式

--格式:${var:起始位置:长度}

方法二,使用expr substr

--格式:expr substr “$var” 起始位置 长度

方法三,使用cut工具

--格式:echo $var | cut -b 起始位置-结束位置

## 子串替换

只替换第一个匹配结果

--格式:${var/old/new}

替换全部匹配结果

--格式:${var//old/new}

## 字符串掐头

从左向右,最短匹配删除

--格式: ${变量#\*关键词}

从左向右,最长匹配删除

--格式:${变量名##\*关键词}

## 字符串去尾

从右向左,最短匹配删除

--格式: ${变量%关键词\*}

从右向左,最长匹配删除

--格式:${变量名%%关键词\*}