

压缩：

指通过某些算法，将文件尺寸进行相应的缩小，同时不损失文件的内容。

打包：

指将多个文件（或目录）合并成一个文件，方便传递或部署。

在Linux系统中，文件的后缀名不重要，但是针对于压缩文件的后缀名是必须的，因为可以让其他的程序员根据文件的后缀名使用对应的算法进行解压。

Linux常见的压缩文件后缀名：

*.gz	gzip程序压缩的文件
*.bz2	bzip2 程序压缩的文件
*.tar	tar命令打包的数据,并没有压缩过
*.tar.gz	tar程序打包的文件,并且经过 gzip 的压缩
*.tar.bz2	tar程序打包的文件,并且经过 bzip2 的压缩

gzip:

压缩/解压命令

选项:

-c :	将压缩的数据输出到标准输出 (stdout) 上
-d :	解压缩
-t :	可以用来检验一个压缩文件的一致性,看看文件有无错误
-v :	可以显示出原文件/压缩文件的压缩比等信息
-(1,2,...,9):	压缩等级,1最快,但是压缩比最差; 9最慢,但是压缩比最好, 默认是6。
-l :	查看压缩文件的压缩比: <code>gzip -l *.gz</code>

案例:

```
cp /root/install.log /home/gzip
```

```
1, gzip -c install.log //将压缩的数据输出到标准输出
```

```
2, gzip -v install.log //压缩完显示
```

这时发现源文件不在了, 如果想保留源文件, 可以用数据重导向技术

```
3, gzip -d install.log.gz //解压
```

```
4, gzip -c install.log > install.log.gz
```

```
5, gzip -t install.log.gz //检查文件是否有误
```

```
6, gzip -c9v install.log //提高压缩比 (文件如果本身很小可能体现不出来)
```

练习:

在/tmp文件夹下创建part1/gzip

将/root/anaconda-ks.cfg文件拷贝到/tmp/part1/gzip

将拷贝后的文件进行gzip压缩,并显示压缩信息。

将压缩后文件的名称改为mygzip01.gz

bzip2:

压缩/解压命令:

选项:

-c :	将压缩的过程产生的数据输出到标准输出 (stdout)
------	-----------------------------

-d :	解压缩的参数
-k :	保留源文件,而不会删除原始的文件
-v :	可以显示出原文件/压缩文件案的压缩比等信息;
-(1,2,...,9):	与gzip同样的,都是在计算压缩比的参数,-9最佳,-1最快

gzip拥有更快的压缩性能。

bzip2拥有更高的压缩比。

单纯从压缩比方面来说, 那么bzip2 > gzip > compress

查看压缩文件中的内容:

cat: 可以用来查看文本文件中的内容。

zcat: 可以用来查看gzip算法压缩的压缩文件内容。

bzcat: 可以用来查看bzip2算法压缩的压缩文件内容。

☑️⭐️! 打包、解包命令

10:01

tar:

可以将一个文件/夹打包成一个文件。可以结合gzip、bzip2的算法对包文件进行相应的压缩和解压。

语法:

压缩: `tar [选项] newFileName.tar.gz sourceFileName`

解压: `tar [选项] fileName.tar.gz [-C /path]`

选项:

-c :	建立打包文件,
-t :	查看打包文件的内容含有哪些文件
-x :	解打包或解压缩的功能,可以搭配-C(大写)在指定目录解开
-j :	通过bzip2的支持进行压缩/解压缩:此时文件最好为 *.tar.bz2
-z :	通过gzip的支持进行压缩/解压缩:此时文件最好为 *.tar.gz
-v :	在压缩/解压缩的过程中,将正在处理的文件名显示出来
-f filename:	-f 后面跟处理文件的全名称 (路径+文件名+后缀名)
-C 目录:	这个选项用在 解压的时候 ,若要在特定目录解压,可以使用这个选项

注:

使用命令进行打包、压缩的时候,使用了什么算法,文件后缀名就一定要与其对应。

案例:

压缩:

1、使用gzip的算法进行打包压缩。

```
# bash
```

```
tar -zcvf install.log.tar.gz install.log
```

注意tar的语法, tar -zcvf newFile sourceFile

2、使用bzip2的算法进行打包压缩。

```
# bash
```

```
tar -jcvf install.log.tar.bz2 install.log
```

3、如果想要压缩指定目录中的内容是，可以考虑使用绝对路径。

```
# bash
```

```
tar -zcvf [path]/newFileName.tar.gz [path]/sourceFile
```

解压：

1、将一个压缩包文件解压到当前目录下

```
# bash
```

```
tar -zxvf install.log.tar.gz
```

执行完成之后，文件会在当前的目录下。

2、将一个压缩包文件解压到指定目录下

```
# bash
```

```
tar -zxvf install.log.tar.gz -C /
```

3、只解压包中的某个文件

```
# bash
```

```
tar -zxvf etc.tar.gz etc/shells
```

4、配置jdk环境变量：

```
# bash
```

```
tar -zxvf jdk-8u131-linux-x64.tar.gz
```

```
cd jdk1.8.0_131
```

```
pwd # 复制路径
```

```
vim /etc/profile # profile文件是系统环境变量的配置文件
```

在该文件的最后一行添加内容：

```
export JAVA_HOME=/home/software/jdk1.8.0_131
```

```
export PATH=$JAVA_HOME/bin:$PATH
```

保存退出

```
source /etc/profile
```

使环境变量生效

最初只有.tar.gz的打包文件，用户必须编译每个他想在Linux上运行的软件。用户们普遍认为系统很有必要提供一种方法来管理这些安装在机器上的软件包，当Debian诞生时，这样一个管理工具也就应运而生，它被命名为dpkg。稍后RedHat才决定开发自己的“rpm”包管理系统。

优点：

- 自带编译后的文件，免除用户对软件编译的过程
- 可以自动检测文件系统(硬盘)的容量、系统的版本。避免软件被错误的安装。
- 自带软件的版本信息、帮助文档、用途说明等信息。

缺点：

- 无论安装还是卸载，RPM都有一个恶心人的依赖关系。
- 安装的软件需要依赖，那么优先安装依赖。
- 卸载的软件存在依赖，那么优先卸载依赖。

默认路径：

/etc	一些配置文件放置的目录,例如/etc/crontab
/usr/bin	一些可执行文件
/usr/lib	一些程序使用的动态链接库
/usr/share/doc	一些基本的软件使用手册与说明文件
/usr/share/man	一些man page（Linux命令的随机帮助说明）文件

安装：

语法：rpm -ivh packageName.rpm

选项：

i	表示安装
v	表示处理过程
h	显示处理进度(进度条)

案例：

软件包在资料中提供：

X:\课件V4.0提供的资料\rpm\

安装软件：

单个安装：

```
# bash
rpm -ivh pack1.rpm
```

多个安装：

```
# bash
```

```
rpm -ivh pack1.rpm pack2.rpm *.rpm
安装网络上的RPM包
# bash
rpm -ivh "https://网络地址/package.rpm"
```

查询：

rpm -[选项]

选项：

-q :	仅查询,后面接的软件名称是否有安装
-qa :	列出所有的,已经安装在本机Linux系统上面的所有软件名称 !!!
-ql :	列出该软件所有的文件与目录所在完整文件名 !!
-qc :	列出该软件的所有配置文件 !
-qd :	列出该软件的所有说明文件
-qR :	列出和该软件有关的相依软件所含的文件

案例1: 查找是否安装jdk

```
# rpm -qa |grep jdk
```

案例2: 查找所有系统已经安装的包, 并只查看前3个

```
# rpm -qa |head -n 3
```

案例3: 查询lrzsz所包含的文件及目录

```
# rpm -ql lrzsz
```

案例4: 查看lrzsz包的相关说明

```
# rpm -qi lrzsz
```

列出iptables的配置文件

```
# rpm -qc iptables
```

案例7: 查看apr需要的依赖

```
# rpm -qR apr
```

卸载：

rpm -e package_Name # package_Name需要通过qa的选项来查询出来。

RPM 升级与更新

rpm -Uvh <package_name> (不管有没有都安装最新版)

-Uvh后面接的软件如果没有安装过, 系统会直接安装,若后面接的软件安装过但版本较旧,则更新至新版

```
[root@localhost soft]# rpm -Uvh jdk-8u111-linux-x64.rpm
```

```
Preparing... #####
```

```
[100%]
```


package jdk1.8.0_111-2000:1.8.0_111-fcs.x86_64 is already installed

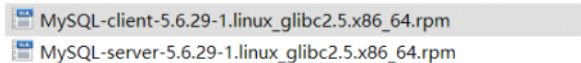
rpm -Fvh <package_name> (只有安装才更新)

-Fvh如果后面接的软件并未安装到Linux系统上,则该软件不会被安装,只有已安装的软件才会被升级

通过RPM安装mysql

2019年3月18日 17:36

1. 下载mysql安装包



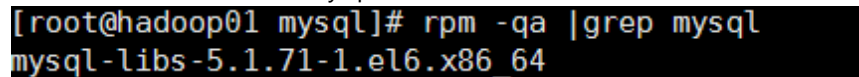
```
MySQL-client-5.6.29-1.linux_glibc2.5.x86_64.rpm
MySQL-server-5.6.29-1.linux_glibc2.5.x86_64.rpm
```

2. 确认当前虚拟机之前是否有安装过mysql

执行: `rpm -qa` 查看linux安装过的所有rpm包

执行: `rpm -qa | grep mysql`

如果出现下图, 证明已经安装了mysql, 需要删除



```
[root@hadoop01 mysql]# rpm -qa | grep mysql
mysql-libs-5.1.71-1.el6.x86_64
```

3. 删除mysql

执行: `rpm -ev --nodeps mysql-libs-5.1.71-1.el6.x86_64`

此时, 再执行: `rpm -qa | grep mysql` 发现没有相关信息了

4. 新增mysql用户组, 并创建mysql用户

`groupadd mysql`

`useradd -r -g mysql mysql`

5. 安装mysql server rpm包和client包, 执行:

`rpm -ivh MySQL-server-5.6.29-1.linux_glibc2.5.x86_64.rpm`

`rpm -ivh MySQL-client-5.6.29-1.linux_glibc2.5.x86_64.rpm`

6. 安装后, mysql文件所在的目录

Directory	Contents of Directory
/usr/bin	Client programs and scripts
/usr/sbin	The mysqld server
/var/lib/mysql	Log files, databases
/usr/share/info	MySQL manual in Info format
/usr/share/man	Unix manual pages
/usr/include/mysql	Include (header) files
/usr/lib/mysql	Libraries
/usr/share/mysql	Miscellaneous support files, including error messages, character set files, sample configuration files, SQL for database installation
/usr/share/sql-bench	Benchmarks

7. 修改my.cnf,默认在/usr/my.cnf, 执行: vim /usr/my.cnf, 添加如下内容:

```
[client]

default-character-set=utf8

[mysql]
```

```
default-character-set=utf8
```

```
[mysqld]
```

```
character_set_server=utf8
```

8. 将mysqld加入系统服务，并随机启动

执行: `cp /usr/share/mysql/mysql.server /etc/init.d/mysqld`

说明: /etc/init.d 是linux的一个特殊目录，放在这个目录的命令会随linux开机而启动。

9. 启动mysqld，执行: `service mysqld start`

```
[root@hadoop01 mysql]# cp /usr/share/mysql/mysql.server /etc/init.d/mysqld
[root@hadoop01 mysql]# service mysqld start
Starting MySQL... SUCCESS!
```

10. 查看初始生成的密码，执行: `vim /root/.mysql_secret`。这个密码随机生成的

```
# The random password set for
: UCgUjqWmTcBNctCJ
```

11. 修改初始密码

第一次安装完mysql后，需要指定登录密码

执行: `mysqladmin -u root -p password root` 此时，提示要输入初始生成的密码，拷贝过来即可

12. 进入mysql数据库

执行: `mysql -u root -p`

输入: root进入

执行: \s查看mysql数据配置信息

13.mysql卸载

```
[root@Demo02 usr]# service mysqld stop
Shutting down MySQL..
[root@Demo02 usr]# service mysqld status
MySQL is not running
[root@Demo02 usr]# rpm -qa|grep -i mysql
MySQL-client-5.6.29-1.linux_glibc2.5.x86_64
MySQL-server-5.6.29-1.linux_glibc2.5.x86_64
[root@Demo02 usr]# rpm -e MySQL-client-5.6.29-1.linux_glibc2.5.x86_64
[root@Demo02 usr]# rpm -e MySQL-server-5.6.29-1.linux_glibc2.5.x86_64
[root@Demo02 usr]# rpm -qa|grep -i mysql
[root@Demo02 usr]# find / -name mysql
/var/lib/mysql
/var/lib/mysql/mysql
/usr/lib64/mysql
```

[确定]

[失败]

1.把显示出来的所有文件用rpm -e 文件名删除

2.在find 查询一遍看是否还存在

3.删除user目录下的my.conf文件

14.虚拟机数据库连接本机的可视化工具

```

mysql> update user set host='%' where user='root' limit 1;
ERROR 1046 (3D000): No database selected
mysql> use mysql; 1
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed 1
mysql> update user set host='%' where user='root' limit 1;
Query OK, 1 row affected (0.09 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> flush privileges; 2
Query OK, 0 rows affected (0.00 sec)

mysql> exit
Bye
[root@Demo02 ~]# service iptables restart 4
iptables: 将链设置为政策 ACCEPT: filter
iptables: 清除防火墙规则:
iptables: 正在卸载模块:
iptables: 应用防火墙规则:
[root@Demo02 ~]# vim /etc/sysconfig/iptables 3
# Firewall configuration written by system-config-firewall
# Manual customization of this file is not recommended.
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 3306 -j ACCEPT
-A INPUT -i REJECT --reject-with icmp-host-prohibited

```

- 1.进入数据库修改
- 2.冲刷
- 3.退出数据库修改配置文件
- 4.重启防火墙

[确定]
[确定]
[确定]
[确定]



软件管理-YUM

14:33

yum的由来，是因为rpm的缺点所导致，因为rpm无论安装还是卸载都需要解决依赖关系，并且比较繁琐，所以诞生yum的技术。

yum通过分析rpm的信息来进行软件的安装、升级、卸载。

优点：	可以一键解决rpm的依赖关系。
缺点：	yum的所有执行操作全都都需要repo文件(YUM源)。使用yum安装软件，中招几率高达90%。

所有的yum源都存放在/etc/yum.repos.d/目录下。

工作环境中，一般都会屏蔽系统自带的yum源，而选择权威机构的yum源。

yum的查询：

search	查询某个软件名称或者是描述的关键字
list	列出目前yum所管理的所有的软件名称与版本,有点类似 rpm -qa

yum的安装：

```
yum install package_Name
```

案例：

```
# bash
```

```
yum install lrzsz
```

期间会提示y/N

输入y即可。

yum的卸载：

```
yum remove package_Name
```

案例：

```
# bash
```

```
yum remove lrzsz
```

YUM安装tomcat：

1、查询可以安装的tomcat安装包

```
Yum search tomcat
```

2、找到tomcat6-admin-webapps.noarch这个rpm安装包，通过yum进行安装

```
Yum install -y (忽略所有yes 的提醒) tomcat6-admin-webapps.noarch
```

3、安装完的文件目录在 /usr/share 目录下

yum的更新：

```
yum update package_Name
```

yum安装、卸载、更新的过程中出现的y/N，可以通过在命令的结尾出 -y，表示全部过执行yes操作。

yum客户端运行机制

客户端每次使用yum调用 install或者search的时候，都会去解析/etc/yum.repos.d/下面所有以.repo结尾的文件，这些配置文件指定了yum服务器的地址。

yum需要定期去“更新”yum服务器上的rpm“清单”，然后把“清单”下载保存到yum自己的cache里面，根据/etc/yum.conf里配置(默认是在/var/cache/yum/\$basearch/\$releasever下、即/var/cache/yum/x86_64/6)，每次调用yum安装包的时候都会去这个cache目录下去找“清单”，根据“清单”里的rpm包描述从而来确定安装包的名字，版本号，所需要的依赖包等，如果rpm包的cache不存在，就去yum服务器下载rpm包安装。

3.清理yum缓存，并生成新的缓存

```
yum clean all
```

```
yum makecache
```

加入hadoop组件相关yum源

1, 查看当前系统中yum支持的所有软件包中是否存在hadoop

```
[root@tedu yum.repos.d]# yum list|grep hadoop #发现没有
```

2, 如果想要当前系统的yum支持hadoop软件包，需要本地/etc/yum.repos.d下创建cloudera-cdh5.repo文件

http://archive.cloudera.com/cdh5/redhat/6/x86_64/cdh

centos6系列更换阿里yum源

1.首先备份原来的cent os官方yum源

```
cp /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.bak
```

2.获取阿里的yum源覆盖本地官方yum源

```
wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-6.repo
```

RPM和YUM的取舍

15:29

如果安装、卸载、更新的软件是单个独立的离线安装包，那么建议使用RPM的方式进行安装、卸载、更新。

如果安装一个软件时，发现此软件有众多的依赖环境，那么首选就是yum的方式进行处理。

例如：

安装一个jdk，那么首选rpm的方式。

安装tomcat的话就可以考虑使用yum。

数据重定向

16:49

- 数据重定向

- 数据重定向就是将某个命令执行后应该要出现在屏幕上的数据, 给他传输到其他地方,
- 通常执行一条命令的时候会有标准输出和标准错误输出

- 标准输出是指命令执行之后, 传回正确信息的输出目标 •

```
[root@localhost ~]# ll ./media •
```

```
total0 •
```

- 标准错误输出是命令执行失败后, 所传回错误信息的输出目标

- [root@localhost ~]# ll • m •

```
ls: can • not • access • m • : No • such • file • or • directory • •
```

标准输入 (stdin) : 编号为0 使用<或<<

标准输出 (stdout) : 编号为1 使用>或>>

标准错误输出 (stderr) : 编号为2 使用>或>>

1> : 以覆盖的方法, 将正确的数据输出到文件;

1>> : 以累加的方法, 将正确的数据输出到文件;

2> : 以覆盖的方法, 将错误输出的数据输出到文件;

2>> : 以累加的方法, 将错误输出的数据输出到文件; •

案例:

某一条命令执行后会有标准输出和标准错误输出, 将标准输出的内容输出到文件中。

```
# bash
```

```
ll /root /roo 1> fileName
```

某一条命令执行后会有标准输出和标准错误输出, 将标准错误输出的内容输出到文件中。

```
# bash
```

```
ll /root /roo 2> fileName
```


某一条命令执行后会有标准输出和标准错误输出，将标准出和标准错误输出的内容输出到文件中。

```
# bash
```

```
ll /root /roo > fileName 2>&1
```

还是上面的案例，只不过要求结果文件不保存

```
# bash
```

```
ll /root /roo > /dev/null 2>&1
```

>>用法同上

标准输入案例

打印文本中的行数

```
wc -l < 文本
```

并且可以将打印出来的重导向到新的文件中

```
wc -l< 文本 > count
```

利用标准输入编写文件

```
cat >> demo.txt<< "abc"
```

命令执行判断

17:30

命令执行判断

\$?:命令回传值

命令回传值\$?的两种用法：与 && 或 ||

&&:

cmd1 && cmd2 若cmd1运行完毕且正确运行 (\$? =0) , 则开始运行cmd2; 若cmd1运行完毕且为错误 (\$? !=0) , 则cmd2不运行;

|| :

cmd1 || cmd2 若cmd1进行完毕且正确运行 (\$?=0) , 则cmd2不运行; 若cmd1运行完毕且为错误 (\$?!=0) ,则开始运行cmd2;

不管与还是或, 运行正确回传值均为0,不同的是与的时候运行cmd2,而或的时候不运行cmd2; 若运行错误, 则回传值均为非0,但与的时候不运行cmd2,而或的时候运行cmd2。。

举例:

如果/tmp/test存在, 则创建/tmp/test/demo

```
# bash
```

```
mkdir -p /tmp/test && touch /tmp/test/demo
```

如果/tmp/test1不存在, 则删除/tmp

```
# bash
```

```
p/test/demo
```

```
cd /tmp/test1 || rm -rf /tmp/test/demo
```

! ★ Shell基础

15:33

shell的概念:

Shell 是一个用 C 语言编写的程序，它是用户使用 Linux 的桥梁。Shell 既是一种命令语言，又是一种程序设计语言。即是shell程序，也是一门编程语言。

Shell 是指一种应用程序，这个应用程序提供了一个界面，用户通过这个界面访问操作系统内核的服务。

简单的说就是用户和内核之间进行通信/沟通的翻译官

Shell 环境

Shell 编程跟 JavaScript、php 编程一样，只要有一个能编写代码的文本编辑器和一个能解释执行的脚本解释器就可以了。

- 1、脚本解释器
- 2、文本编辑工具

系统提供多种shell程序：cat /etc/shells

```
[root@localhost ~]# cat /etc/shells
/bin/sh
/bin/bash
/sbin/nologin
/bin/dash
/bin/tcsh
/bin/csh
[root@localhost ~]# _
```

初见shell脚本

打开文本编辑器(可以使用 vi/vim 命令来创建文件)，新建一个文件 test.sh

实例：

```
#!/bin/bash
echo "Hello World !"
```

运行shell脚本的两种方法：

1	chmod +x ./test.sh #使脚本具有执行权限 ./test.sh #执行脚本
---	--

shell中变量:

定义一个变量:

```
your_name="chenpingping"
```

注意，变量名和等号之间不能有空格，这可能和你熟悉的所有编程语言都不一样。同时，变量名的命名须遵循如下规则：

- 命名只能使用英文字母，数字和下划线，首个字符不能以数字开头。
- 中间不能有空格，可以使用下划线（_）。
- 不能使用标点符号。

使用变量:

```
echo $your_name  
echo ${your_name}
```

变量名外面的花括号是可选的，加不加都行，加花括号是为了帮助解释器识别变量的边界

例如：

```
boy="good"  
echo "you are my $boyfriend" // $boyfriend值为空  
改为  
echo "you are my ${boy}friend"
```

另外：已经定义的变量可以重新定义，但是如果是只读的则不可以

```
your_name="tom"  
echo $your_name  
your_name="alibaba"  
echo $your_name
```

readonly + 变量名 // 表示这个变量只读，不能重新赋值

删除变量

unset[空格] 变量名 （不能删除只读变量）

shell字符串

- 1、单引号中的内容会被当作普通字符串(java的String)来处理。

案例：

```
# bash
x='$LANG'
echo $x
# $LANG
```

2、双引号中的内容会按照其原本的属性进行输出。

案例：

```
# bash
x="$LANG"
echo $x
# zh_CN.UTF-8
```

3、可用转义字符 “\” 将特殊符号（如\$、\、！等）变为一般字符；

```
your_name='chenzhe'
str="Hello, I know you are \" $your_name \"! "
echo -e $str
```

4、拼接字符串

```
your_name="piaolaoshi"
# 使用双引号拼接
greeting="hello, "$your_name
greeting_1="hello, ${your_name}"
echo $greeting $greeting_1
# 使用单引号拼接
greeting_2='hello, '$your_name
greeting_3='hello, ${your_name}'
echo $greeting_2 $greeting_3
```

5、获取字符串长度

```
string="abcd"
echo ${#string} #输出 4
```

6、获取子字符串

```
string="mayun is a great man"
```

```
echo ${string:1:4} # 输出ayun
```

shell数组

定义数组： 数组名= (值1 值2 值3 值4)

例如：

```
array=(1 2 3 4)
```

```
echo ${array[1]} #2
```

利用@可以获取元素当中的所有元素

例如

```
echo ${array[@]} #1 2 3 4
```

可以给数组单独赋值

```
array[1]=89
```

获取数组的长度

取得数组元素的个数

```
length=${#array[@]}
```

或者

```
length=${#array[*]}
```

取得数组单个元素的长度

```
array=(mayun mahuateng leijun liuqiangdong)
```

```
length=${#array[n]}
```


shell传递参数

0:03

我们可以在执行 Shell 脚本时，向脚本传递参数，脚本内获取参数的格式为：**\$n**。**n** 代表一个数字，其中\$0为执行的文件名，1 为执行脚本的第一个参数，2 为执行脚本的第二个参数，以此类推.....

举例：

以下实例我们向脚本传递三个参数，并分别输出

`#!/bin/bash`

```
echo "Shell 传递参数实例！ ";
echo "执行的文件名：$0";
echo "第一个参数为：$1";
echo "第二个参数为：$2";
echo "第三个参数为：$3";
```

为脚本设置可执行权限，并执行脚本，输出结果如下所示：

```
$ chmod +x test1.sh
$ ./test.sh 1 2 3
```

另外，还有几个特殊字符用来处理参数：

<code>\$#</code>	传递到脚本的参数个数
<code>\$*</code>	以一个单字符串显示所有向脚本传递的参数。 如" <code>\$*</code> "用「 <code>"</code> 」括起来的情况、以" <code>\$1 \$2 ... \$n</code> "的形式输出所有参数。
<code>\$\$</code>	脚本运行的当前进程ID号
<code>\$_</code>	后台运行的最后一个进程的ID号
<code>\$@</code>	与 <code>\$*</code> 相同，但是使用时加引号，并在引号中返回每个参数。 如" <code>\$@</code> "用「 <code>"</code> 」括起来的情况、以" <code>\$1" "\$2" ... "\$n</code> " 的形式输出所有参数。
<code>\$?</code>	显示最后命令的退出状态。0表示没有错误，其他任何值表明有错误。

`#!/bin/bash`

```
echo "Shell 传递参数实例！ ";
echo "第一个参数为：$1";

echo "参数个数为：$#";
echo "传递的参数作为一个字符串显示：$*";
```

`$*` 与 `$@` 区别：


```
#!/bin/bash
echo "-- \${*} 演示 ---"
for i in "${*}"; do
    echo $i
done

echo "-- \${@} 演示 ---"
for i in "${@}"; do
    echo $i
done
```