**Shell 脚本实战**

1. **shell基础知识回顾**

**linux中变量的定义与调用**

命名规则：

只能包括：<字母>、<数字>和<\_下划线>，并且，<变量名>不能以<数字>开头。

（最好具有实际意义，见名知意 。）

定义方式：

变量名=“变量值”

var01="123" #定义一个变量叫var01，并赋值为123

var02="${var01}456" #引用其他变量来赋值

var03="`date`" #将命令的输出结果赋值给变量 （`命令`用点号括起来）

还有一种类似的写法：

var04="$(date)" #命令的输出结果赋值给变量 ( $(命令) )

#数组变量

数组：就是一组同姓不同名的<变量集合>，彼此用<索引下标>加以区分。

（用来存放多个值）

定义方式：

array01=() #只定义，不赋值

array02=("a" "b" "c") #既定义，又赋值（注意：需要用<空格>隔开）

输出数组：

语法：echo ${数组名[下标]}

下标：值的编号，从0开始 例：下标2表示第三个值

#删除变量

语法：unset 变量名

例：unset aaa #删除变量aaa

**#执行脚本的方式**

方式1.独立子进程执行方式（常用）不影响父进程

bash 脚本名 或者 sh 脚本名 例：sh 1.sh 或者 bash 1.sh

/路径/脚本名 #需要脚本具有执行权限 例：/root/1.sh 或者 ./1.sh

方式2.共享父进程执行法（用于刷新或者立即生效时） 会影响父进程

source 脚本名 例：source 1.sh

方式3：置换父进程执行法 （很少用）会替换父进程

exec 脚本名 例： exec ./1.sh

**#shell内置变量** （命令行参数）shell自带的变量，无需定义直接使用

$0 :shell脚本自身包括路径

$1-$n :脚本的命令行参数，$1 是第 1 参数、$2 是第 2 参数，以此类推

$\* ：输出所有的参数

$#：输出参数的总个数

$?：上一条命令的返回值

$$：shell脚本自身的pid进程号

**#引号的使用**

双引号：弱引，部分引用，可以识别特殊符号

单引号：强引，完全引用，显示原内容

#$(命令) 、`命令` 可以把命令的运行结果赋值给变量

也可以直接在命令行替换：

例：touch `cat 1.txt`.txt

touch `date +%F`.txt

**#变量的作用域**（变量的生效范围）

分类：本地变量和环境变量

环境变量作用域：当前进程及子进程

定义方法：export 变量名="变量值"

env #该命令可以查看所有的环境变量

常用的：$PATH $HOSTNAME $HISTSIZE(历史记录的条数) $LANG（语言环境）

本地变量作用域：当前进程

定义方法：变量名="变量值"

**# 通配符**

\*：0个或者多个任意字符

例：rm -rf /tmp/\*

?：单个任意字符

例：?.txt 只可以表示类似1.txt，不能表示11.txt

[ ]：范围内单个字符

例：sd[bcd] #可以表示 sdb sdc sdd

**# 正则表达式**

作用：用字符去实现匹配的内容

正则表达式分类：基本正则和扩展正则

常用的正则元字符

基本正则元字符： （基本正则支持的扩展正则也支持）

^ 表示以什么开头的行

$ 表示以什么结尾的行

. 表示任意单个字符

\* 表示前一项出现0次或多次

[] 表示范围内任意一个字符 # [a-d] [1-9] [dhp]

\s 表示空白字符（空格）

^$:可以用来表示空白行

扩展正则元字符：

? 表示前一项出现0次或1次

+ 表示前一项出现1次或多次

| 逻辑或

() 正则分组

{}：

{n} 表示前一项出现n次，n可以为0

{n,} 表示前一项至少出现n次

{,m} 表示前一项最多出现m次

{n,m} 表示前一项至少出现n次，最多出现m次

**# shell常用辅助命令** （sort uniq wc cut read）

1.sort 排序 （默认按照字典顺序）

选项：

-g：使用数值大小去排序

-r ：倒序

-k：按指定字段排序

例：cat 1.txt | sort -k 4 #按照第4个字段去排序

ps aux | sort -k 6 -g #第6个字段按数值大小去排序

2.uniq 去重 （分组统计）

只能对连续的行去重，因此去重前一般需要先排序

例：cat 3.txt | sort | uniq

选项 -c 统计重复的次数

例：cat 3.txt | sort | uniq -c

3.wc 统计

选项 -l ：统计行数

例： cat /etc/passwd | wc -l #统计当前的用户数量

4.cut 截取内容

根据需要截取每行匹配的内容

选项：

-c ：按照字符位置进行截取

-f ：输出分隔后的字段编号（需要与-d选项连用）

-d ：指定分隔符，默认以tab制表符分隔，自定义分隔符需要用引号引起来

例：cat 3.txt | cut -c 2 #截取每行的第2个字符

-c 5-7 ：表示第5个到第7个字符

-c -7 ：表示从开始到第7个字符

-c 7- ：表示从第7个字符到行尾

例：cat 1.txt | cut -d ":" -f 1 #按照冒号分隔，分隔后取第1个字段

-f 1,3,5 #可以取多个字段

5.tr 替换

语法：tr 【选项】 字符集1 【字符集2】

例：cat 3.txt | tr 1 a #将3.txt输出结果中的数字1替换成字母a

cat 4.txt | tr a-z A-Z #将4.txt输出结果中小写字母换成大写字母

注意：只是替换本次输出结果，不会修改原文件

选项 -d ：删除匹配字符集的内容

例： cat 4.txt | tr -d 0-9 #删除所有数字

cat 4.txt | tr -d a-z #删除所有小写字母

6.read命令 （第4种变量赋值的方式）

选项：

-p：设置提示信息

-e：用户输入的时候允许回退

-s：不显示输入的内容 （密码输入的时候使用）

#!/bin/bash

read -e -p "请输入你的用户名：" name

echo "你的用户名是：$name"

read -e -s -p "请输入你的密码：" passwd

echo "你的密码是：$passwd"

7.sleep和wait（了解，多进程脚本下使用）

例：sleep 10 #睡眠10秒，默认单位是秒

使用场景：有些命令执行会比较耗时，可以起缓冲作用

**# 脚本三剑客：grep sed awk**

#sed命令

sed 是一个用于：过滤并转换文本行的流编辑器

sed和vi编辑器的区别：两者都是编辑器，vi主要用于交互式编辑

sed是非交互式，更适合脚本

文本处理工具对比：

grep：快速的横向过滤（过滤行）

awk：快速的纵向过滤 （过滤字段或者列）

sed：横向纵向都行，更倾向于对文件内容的修改

sed特点：

1. 逐行输入、逐行处理、逐行输出 （低内存消耗，处理大文件）

2. 过程：在模式空间，暂存数据、处理数据、输出数据、清空数据

3. 默认不会修改源文件（可以彩排一次）

sed基本运用

语法1：sed 【选项】 动作脚本 文件名 （主要的）

语法2：sed 【选项】 -f 动作文件 文件名

（把处理动作写在文件里）

动作脚本：匹配地址+子命令

实用的语法：sed 【选项】 '匹配地址+子命令' 文件名

哪些行+怎么处理

常用选项：

-n：默认所有行都会输出到屏幕，抑制标准输出

一般和p（输出子命令）连用

-r：启用扩展正则 类似grep 的 -E选项

-e：用来执行多个动作 # sed -n -e '3p' -e '5p' 1.txt

-i：回源（修改源文件） （警告：不要与-n一起用）

#匹配地址的方法 （哪些行）

1.直接使用行号

例：

sed -n '3p' 1.txt #输出指定行

sed -n '1,3p' 1.txt #输出第1行到第3行

sed -n '$p' 1.txt #输出文件最后一行

'n,+m' #从n行开始，往后数m行

sed -n '1,+2p' 1.txt #从第一行开始，往后数两行

'n~m' 从n行开始每m行输出一次

sed -n '1~2p' 1.txt #从第一行开始，每两行输出一次（奇数行）

2.通过正则或关键字匹配

/正则表达式或关键字/ ：输出满足该表表达式的行

例：sed -n '/root/p' 1.txt #关键字匹配：输出含有root的行

sed -n '/^root/p' 1.txt #正则匹配：输出root开头的行

正则范围匹配：/正则表达式1或关键字1/，/正则表达式2或关键字2/

例：

sed -n '/^root/,/^lp/p' 1.txt #输出从root开头的行到以lp开头的行

#子命令（怎么处理） p d i a c s

1. p 打印 （需要-n选项）

例：sed -n '1,3p' 1.txt #打印指定行

2. d 删除子命令

例：sed '1,5d' 1.txt #删除1到5行

注意：并不是真正的删除，不会影响源文件，如果需要修改源文件

需要使用-i选项

sed -i '1,5d' 1.txt #-i选项会删除原文的1到5行

3. a 追加 子命令 （追加一行内容）

语法：'匹配地址a\追加的内容'

例：

sed '1a\这是第二行' 1.txt #在第1行后面追加指定内容

4. i 插入 子命令 （插入一行新内容）

语法：'匹配地址i\追加的内容'

例：

sed '1i\这是第一行' 1.txt #在第1行前面插入指定内容

5.c 替换 子命令 （替换指定行）

语法：'匹配地址c\替换后的内容'

例：

sed '1c\这是第一行' 1.txt #替换指定行内容

6.s 字符替换子命令 （重点）

语法：'匹配地址s/原内容/新内容/【修饰符】'

's///' 如果冲突，分隔符可以更换

例：sed '1s/root/ROOT/' 2.txt #将第一行的root替换成ROOT，默认只会替 换第一个

sed '1s/root/ROOT/g' 2.txt #第一行全部替换

sed 's/root/ROOT/g' 2.txt #全文全部替换 （不需要匹配地址）

修饰符：

g：全局替换

n：n是一个数字，替换第n个匹配到的

i : 替换时不区分大小写

例：

sed '1s/root/ROOT/3' 2.txt #只替换第一行的第3个root

--------------------------------------------------------------------

#awk 命令 （编程语言）被设计用来专门处理文本数据

awk由三个部分组成：开始（begin）、主体、结束（end）

开始（begin）、结束（end）这两部份可选，一般用于格式化输出

begin：在主体前运行，只运行一次

end ：在主体结束后运行，只执行一次

#主体 （重点）（哪些行、进行什么样的处理）

匹配地址 + 动作

语法：awk 【选项】 '地址{子命令}' 文件名

选项：

-F ：指定分割符 类似cut命令的 -d （默认用空白分割）

子命令：print 打印 +内置变量

$1~$n:表示分割后的第几个字段，如果没有指定分隔符，默认用空白分割

$0或者不写，表示输出整行

例：print $1 :打印第1字段 print $1,$3 : 打印第1字段和第3字段

#基本操作：（类似cut）

1.取出指定的字段

例：

cat 1.txt | awk -F ":" '{print $1}' #用：分割，取第1字段

cat 1.txt | awk -F ":" '{print $1,$7}' #用：分割，取第1和7字段

2.取出的字段也可以添加描述信息

'{print "额外信息" 内置变量}' #额外信息需要双引号引起来

cat 1.txt | awk -F ":" '{print "用户名："$1}'

3.取出的字段也可以进行计算（字段必须是数字）

cat 3.txt |awk '{print $2+$3+$4}' #对第2、3、4字段进行求和计算

#地址匹配：

不支持直接使用行号

1. 正则匹配，类似sed 整行去匹配正则

例：cat 1.txt | cat 1.txt | awk '/^root/{print}' #打印root开头的行

cat 1.txt | cat 1.txt | awk '/^root/，/^lp/{print}' #打印root开头的行到lp开头的行

2.正则匹配：某个字段去匹配

字段~/正则表达式/

例：cat 4.txt |awk '$2~/^a/{print $0}' #匹配第2字段是a开头的行

3.关系表达式（字段的值为数字）

例：cat test.txt | awk '$4>87{print}' #用第4个字段进行比较，然后输出整行

cat test.txt | awk '$4>87{print $2}' #用第4个字段进行比较，然后输出第2字段

常用的比较符号： > < ==(等号需要写两个)

#多个表达式使用逻辑符号

&& ：逻辑与

|| ： 逻辑或

！ ： 取反

例：

cat test.txt | awk '$4>87&&$4<95{print $2}' #两个条件同时满足

cat test.txt | awk '$4>89||$4<85{print}' #两个条件满足一个即可

cat test.txt | awk '!($4==87){print}' #取相反的条件

#常用的内置变量

$0:整行输出

$1-$n :分隔后的字段序号

NR变量：表示行号

例：cat 1.txt | awk 'NR==2{print}' #输出文件的第2行

cat 1.txt | awk 'NR>=3 && NR<=5{print}' #输出第3行到第5行

**# 算数计算**

一类：整数计算

1. ((计算表达式))

例：n=1; ((n=n+1)); echo $n

((n=3+4));echo $n

2.let 命令

例：let n=(1+2)\*3; echo $n

let n=3+4; echo $n

3.expr 命令

例：expr 3 + 4 (数字和操作符之间有空格)

expr 3 \\* 4 （乘号需要转义）

第二类：小数计算（浮点数）

yum install -y bc #先安装bc命令

echo "10/3" | bc #整数计算

echo "scale=2;10/3" | bc #小数计算 scale用来指定保留小数的位数

#计算操作符（运算符）

+ - \* / 加减乘除

% 取余数 (可以 用来取指定范围的值)

id++ id-- 自增、自减 （先赋值后自增、自减） （用于循环）

++id --id 自增、自减 （先自增、自减再赋值）

**# shell条件判断**

1.数值条件判断表达式 （用来判断数字之间的关系）

以整数为主：

-eq 等于

-ne 不等于

-gt 大于

-ge 大于等于

-lt 小于

-le 小于等于

2.字符串条件判断表达式

-z 字符串 ：判断字符串是否为空

-n 字符串 ：判断字符串是否为非空

string1 == string2 ：判断两个字符串的内容是否相同

string1 != string2 ：判断是否不相同

3.正则条件判断表达式

=~ 正则表达式 #判断指定能容能否用该正则表达式表示出来

例：a="qwer"; [[ $a =~ ^q ]]

#条件判断命令 test [ ] （判断一个表达式的真假）

test 3 -gt 4 等价 [ 3 -gt 4 ]

可以判断整数直接的关系、字段串之间的关系

还有和文件相关的判断：

-e 文件名 ：判断文件是否存在

-s 文件名 ：判断文件存在且是否非空

-f 文件名 ：判断文件存在且为普通文件

-d 文件名 ：判断文件存在且为目录文件（目录是否存在）

-b 文件名 ：判断文件存在且为块设备文件

-r 文件名 ：判断当前用户对文件是否可读

-w 文件名 ：判断当前用户对文件是否可写

-x 文件名 ：判断当前用户对文件是否可执行

**# 逻辑操作**

！条件表达式 #逻辑取反

-a #逻辑与 例：表达式1 -a 表达式2 (所有条件为真，则为真)

-o #逻辑或 例：表达式1 -o 表达式2 （至少一个为真，则为真）

[ -d /tmp -a -d /etc ] #判断两个目录是否都存在

#[ ] 与 [[ ]]的区别

[ ]是一个判断命令：[ 条件表达式 ] 注意条件表达式两边有空格，用来判断表达式的真假。

[[ 条件表达式 ]] 用来判断表示式的真假，一般与if连用，作为if语句后面的条件判断

[[ ]]结构比[ ]结构更加通用

#组合多个条件判断表达式

组合符： &&（逻辑与） || （逻辑或） #用来连接多个判断表达式

[ -d /tmp ] && [ -d /etc ]

#命令连接符 （；）#用来连接多个命令

&& ：前面的命令返回值是0，则执行后面的命令

|| ：前面的命令返回值非0，则执行后面的命令

用法：命令1 && 命令2 || 命令3 #命令1的返回值是0时执行命令2，反之执行命令3

**# if条件判断语句**

语法1: （单条件）

if 命令1；

then

命令块1 （命令1的返回值是0就执行）

else

命令块2 （命令1的返回值非0的时候执行）

fi

语法2： （多条件）

if 命令1；

then

命令块1 （命令1的返回值是0就执行）

elif 命令2；

then

命令块2 （命令2的返回值是0就执行）

........

elif 命令n

then

命令块n （命令n的返回值是0就执行）

else

命令块 （前面条件都不满足的时候剩下的情况）

**# case 条件判断语句** （case分支）（主要用途：编写菜单）

语法：

case 关键字(变量) in

匹配模式1）

命令1

;;

匹配模式2）

命令2

;;

.......

匹配模式n）

命令n

;;

\*) #前面都未匹配到时执行

命令

;;

Esac

**# shell循环语句**

一、for循环语句

1.算术条件判断循环 （一般用于控制循环次数）

语法：for ((计算表达式1；条件表达式；计算表达式2))；

# 初始值 循环的条件 变量每次的变化（一般为自增或自减）

do

命令块 （每次循环时执行的命令）

done

例：

for ((i=1;i<=10;i++)); .

do

echo "这是第$i 次循环"

done

2.遍历单词序列循环

语法：for 变量 in 单词序列 ； #变量每次会去序列中取一个值， #每个单词用空

格隔开，取完为止

do

命令块 ；

done

**# while循环和until循环**

1.while循环

语法：

while 命令； #通过命令的返回值来判断循环是否继续，如果是0就继续，

非0则终止循环

do

命令块；

done

例：

n=1

while ((n<=10)); do

read -e -p "请输入一个数字：" n

done

while 死循环常用的写法：

while ture；

do

命令块

done

**2.until循环** （直到条件满足，就终止循环）

语法：

until 命令； #通过命令的返回值来判断循环是否继续，如果是0就终止，

非0则循环继续

do

命令块；

done

例：

n=11

until ((n<=10)); do

read -e -p "请输入一个数字：" n

done

until 死循环常用的写法：

until false；

do

命令块

done

**# 循环语句中常用的控制命令**

1.continue ：终止本次循环，直接下一次循环 （用于嵌套循环）

for i in $a

do

for j in $b;

do

命令

done

done

2.break ：终止整个循环

for i in {1..100};

do

echo $i

if [ $i -eq 10 ];

then

break

fi

done

1. **shell脚本实战案例**
2. **网络主机连通性测试工具**

**2）机器巡检脚本**

#!/bin/bash

##系统信息##

sys\_check(){

os\_type=`uname`

echo "操作系统类型是:$os\_type"

os\_banben=`cat /etc/redhat-release`

echo "操作系统版本号是:$os\_banben"

os\_neihe=`uname -r`

echo "操作系统的内核是:$os\_neihe"

os\_time=`date +%F\_%T`

echo "操作系统当前时间是:$os\_time"

os\_uptime=`uptime | awk '{print $3}'|awk -F , '{print $1}'`

echo "操作系统最后重启时间为:$os\_uptime"

os\_hostname=`hostname`

echo "操作系统主机名称为:$os\_hostname"

}

##网络信息##

net\_check(){

net\_ip=`/sbin/ifconfig -a|grep inet|grep -v 127.0.0.1|grep -v inet6|awk '{print $2}'|tr -d "addr:"`

echo "操作系统的ip是:$net\_ip"

ping -c1 www.baidu.com >/dev/null

if [ $? -eq 0 ];then

echo "外网可以连通"

else

echo "外网连不通，请检查"

fi

}

cpu\_check(){

physical\_id=`cat /proc/cpuinfo | grep "physical id"|sort|uniq|wc -l`

echo "操作系统cpu物理个数是:$physical\_id"

cpu\_core=`cat /proc/cpuinfo | grep "cpu cores"|sort|uniq|awk -F ':' '{print $2}'`

echo "操作系统的cpu核心数是:$cpu\_core"

cpu\_type=`cat /proc/cpuinfo | grep "model name"|sort|uniq|awk -F ':' '{print $2}'`

echo "操作系统的cpu型号是:$cpu\_type"

free\_total=`free -m | grep Mem|awk '{printf $2}'`

echo "操作系统的内存总大小为:$free\_total M"

free\_used=`free -m | grep Mem|awk '{printf $3}'`

echo "操作系统已使用内存为:$free\_used M"

free\_shengyu=`free -m | grep Mem|awk '{printf $4}'`

echo "操作系统剩余内存为:$free\_shengyu M"

used\_baifen=`echo "scale=2;$free\_used/$free\_total\*100"|bc`

echo "已使用内存百分比是:$used\_baifen"%

shengyu\_baifen=`echo "scale=2;$free\_shengyu/$free\_total\*100"|bc`

echo "未使用内存百分比是:$shengyu\_baifen"%

}

disk\_check(){

disk\_size=`lsblk | grep -w sda |awk '{print $4}'`

echo "磁盘总量为:$disk\_size"

a=($(df -m | grep -v "tmpfs" | egrep -A 1 "mapper|sd" | awk 'NF>1{print $(NF-2)}'))

sum=0

for i in ${a[\*]}

do

let sum=sum+$i

done

shengfree=$[$sum/1024]

echo "剩余磁盘总量为:$shengfree" G

}

sys\_check

net\_check

cpu\_check

disk\_check

1. **nginx日志切割**

#!/bin/bash

dir=/usr/local/nginx

mv $dir/logs/access.log $dir/logs/`date +%F`.access.log

mv $dir/logs/error.log $dir/logs/`date +%F`.error.log

$dir/sbin/nginx -s reopen

find $dir/logs/ -iname "\*log" -mtime +3 -exec rm -rf {} \;

编辑周期性计划任务定期执行该脚本：

crontab -e

59 23 \* \* \* /bin/bash /root/log.sh

1. **数据库定时备份脚本**

参考1：

#!/bin/bash

DB\_USER="your\_username"

DB\_PASSWORD="your\_password"

DB\_NAME="your\_database\_name"

BACKUP\_DIR="/path/to/your/backup/directory"

TIMESTAMP=$(date +"%F")

BACKUP\_FILE="${BACKUP\_DIR}/${DB\_NAME}-${TIMESTAMP}.sql.gz"

mysqldump -u ${DB\_USER} -p${DB\_PASSWORD} ${DB\_NAME} | gzip > ${BACKUP\_FILE}

If [ $? -eq 0]

then

echo "Database backup successful: ${BACKUP\_FILE}"

参考2：

#!/bin/bash

mysql\_user="username"

mysql\_password="yourpassword"

mysql\_host="serverip"

mysql\_port="3306"

# 备份文件存放地址(根据实际情况填写)

backup\_location=/data/mysql\_backup

# 是否删除过期数据

expire\_backup\_delete="OFF"

# 过期时间设置

expire\_days=15

backup\_time=`date +%Y%m%d%H%M`

backup\_dir=$backup\_location

welcome\_msg="Welcome to use MySQL backup tools!"

# 判断mysql实例是否正常运行

mysql\_ps=`ps -ef |grep mysql |wc -l`

mysql\_listen=`netstat -an |grep LISTEN |grep $mysql\_port|wc -l`

if [ [$mysql\_ps == 0] -o [$mysql\_listen == 0] ]; then

echo "ERROR:MySQL is not running! backup stop!"

exit

else

echo $welcome\_msg

fi

# 备份指定数据库中数据(此处假设数据库是xxx)

mysqldump -h$mysql\_host -P$mysql\_port -u$mysql\_user -p$mysql\_password --single-transaction --master-data=2 --default-character-set=utf8mb4 --databases xxx > $backup\_dir/xxx-$backup\_time.sql

flag=`echo $?`

if [ $flag == "0" ];then

echo "database success backup to $backup\_dir"

else

echo "database backup fail!"

fi

# 删除过期数据

`find $backup\_location/ -type f -mtime +$expire\_days | xargs rm -rf` &&

echo "Expired backup data delete complete!"

1. **lnmp平台部署脚本**

#!/bin/bash

#function: 部署lnmp

#author: lcn

##################判断是否为root用户################

if

[ "$USER" != "root" ]

then

echo "错误：非root用户，权限不足！"

exit 0

fi

#####################防火墙与高级权限###################

systemctl stop firewalld && systemctl disable firewalld && echo "防火墙已经关闭"

sed -i 's/SELINUX=\*/SELINUX=disabled/g' /etc/selinux/config && echo "关闭selinux"

#########下载epel扩展库############

wget -O /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-7.repo

################清除repo链接，才可以自动加载epel的库#############

yum clean all

if

[ $? -eq 0 ];

then

echo "清除成功"

else

echo "清除失败"

exit 1

fi

##################安装ngnix的依赖包####################

yum -y install zlib-devel openssl-devel pcre-devel gcc gcc-c++ make cmake nginx

if

[ $? = 0 ];

then

echo "下载成功"

else

echo "下载失败"

exit 1

fi

##########源码编译nginx1.18、创建ngnix用户############

useradd -s /sbin/nologin -c "Web Nginx" -M nginx

NGINX="/root/nginx-1.18.0.tar.gz"

if

[ ! -e $NGINX ]

then

echo "安装包不存在，请上传安装文件到/root/,上传完成再重新运行该脚本"

echo "或wget http://nginx.org/download/nginx-1.18.0.tar.gz下载安装包"

exit 1

fi

tar -zxf $NGINX && cd /root/nginx-1.18.0

./configure \

--prefix=/usr/local/nginx \

--user=nginx \

--group=nginx \

--with-http\_ssl\_module \

--with-http\_realip\_module \

--with-http\_gzip\_static\_module \

--with-pcre \

--with-http\_stub\_status\_module \

--with-http\_dav\_module \

--with-http\_addition\_module \

--with-http\_sub\_module \

--with-http\_flv\_module \

--with-http\_mp4\_module

make && make install

if

[ $? = 0 ];

then

echo "编译安装已成功"

else

echo "编译安装失败"

exit 1

fi

sleep 2

###############赋予所属用户和所属用户组###############

chown -R nginx:nginx /usr/local/nginx/

################配置nginx支持php################

cp /usr/local/nginx/conf/nginx.conf /usr/local/nginx/conf/nginx.conf.bak

cat > /usr/local/nginx/conf/nginx.conf <<EOF

user nginx;

worker\_processes 1;

#error\_log logs/error.log;

#error\_log logs/error.log notice;

#error\_log logs/error.log info;

#pid logs/nginx.pid;

events {

worker\_connections 1024;

}

http {

include mime.types;

default\_type application/octet-stream;

#log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

# '$status $body\_bytes\_sent "$http\_referer" '

# '"$http\_user\_agent" "$http\_x\_forwarded\_for"';

#access\_log logs/access.log main;

sendfile on;

#tcp\_nopush on;

#keepalive\_timeout 0;

keepalive\_timeout 65;

#gzip on;

server {

listen 80;

server\_name localhost;

#charset koi8-r;

#access\_log logs/host.access.log main;

location / {

root html;

index index.php index.html index.htm;

}

location ~ \.php$ {

root html;

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index index.php;

fastcgi\_param SCRIPT\_FILENAME /usr/local/nginx/html\$fastcgi\_script\_name;

include fastcgi\_params;

}

#error\_page 404 /404.html;

# redirect server error pages to the static page /50x.html

#

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root html;

}

# proxy the PHP scripts to Apache listening on 127.0.0.1:80

#

#location ~ \.php$ {

# proxy\_pass http://127.0.0.1;

#}

# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000

#

#location ~ \.php$ {

# root html;

# fastcgi\_pass 127.0.0.1:9000;

# fastcgi\_index index.php;

# fastcgi\_param SCRIPT\_FILENAME /scripts$fastcgi\_script\_name;

# include fastcgi\_params;

#}

# deny access to .htaccess files, if Apache's document root

# concurs with nginx's one

#

#location ~ /\.ht {

# deny all;

#}

}

# another virtual host using mix of IP-, name-, and port-based configuration

#

#server {

# listen 8000;

# listen somename:8080;

# server\_name somename alias another.alias;

# location / {

# root html;

# index index.html index.htm;

# }

#}

# HTTPS server

#

#server {

# listen 443 ssl;

# server\_name localhost;

# ssl\_certificate cert.pem;

# ssl\_certificate\_key cert.key;

# ssl\_session\_cache shared:SSL:1m;

# ssl\_session\_timeout 5m;

# ssl\_ciphers HIGH:!aNULL:!MD5;

# ssl\_prefer\_server\_ciphers on;

# location / {

# root html;

# index index.html index.htm;

# }

#}

}

EOF

####检查配置文件的是否有错误####

nginx -t

sleep 5

#####编辑nginx.sh文件，设置nginx环境变量、生成环境变量、启动nginx服务#######

cat > /etc/profile.d/nginx.sh <<EOF

export PATH=/usr/local/nginx/sbin:$PATH

EOF

source /etc/profile.d/nginx.sh

nginx

#############设置nginx开机自动启动、赋予权限################

cat >> /etc/rc.d/rc.local <<EOF

/usr/local/nginx/sbin/nginx

EOF

chmod +x /etc/rc.d/rc.local

##########安装MySQL########

##########安装php########

##########测试完成部署########