# 日期

date +"%Y-%m-%d %H:%M:%S"

# 获取脚本中的配置项个数key=

lcount=`sed -n "/xlfd=/=" /root/test.conf`;flcount=`echo "$lcount"|wc -l`;if [ "$lcount" = "" ]; then echo "empty";else if [ $flcount -gt 1 ];then echo "multi";else echo "find one";fi;fi

# 脚本加密

gzexe

# Sed替换字符串

sed -i "s#^\#HostKey /etc/ssh/ssh\_host\_rsa\_key#HostKey /etc/ssh/ssh\_host\_rsa\_key#g" /etc/ssh/sshd\_config

sed 's/原字符串/替换字符串/'



# 获取本机ip

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

# shell上变量+1

((length++))

# 获取文件大小

stat --format=%s /opt/sc/csgl/SCService/string.dat

cat a |wc -l

# 获取字符串所在行

sed -n '/小 计/=' aaa.txt

//内是包含的字符串,改命令会将所有包含小计的行数打印出来

## 若想在脚本中使用,则使用双引号,用单引号不会自动转义

sed -n "/$prev=/=" $path

# Shell脚本中source不管用

使用source执行脚本 source test.sh

# Jkd自动安装配置

#!/bin/bash

echo "------now we start to install and config jdk1.7.79 automatically..."

dir\_tmp=/root/xinstall

mkdir $dir\_tmp 2>/dev/null

sed -n -e '1,/^exit 0$/!p' $0 > "${dir\_tmp}/jdk-7u79-linux-x64.rpm" 2>/dev/null

cd $dir\_tmp

rpm -ivh jdk-7u79-linux-x64.rpm &&

echo "------jdk1.7.79 is installed successfully,now we config the environment variables for jdk1.7.79..."

file="/etc/profile"

echo "export JAVA\_HOME=/usr/java/jdk1.7.0\_79">>$file &&

echo "export JAVA\_BIN=/usr/java/jdk1.7.0\_79/bin">>$file &&

echo "export PATH=$PATH:$JAVA\_HOME/bin">>$file &&

echo "export CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar">>$file &&

echo "export JAVA\_HOME JAVA\_BIN PATH CLASSPATH">>$file &&

echo "-----now jdk 1.7.79 is installed successfully, and the environment variables for jdk1.7.79 is configed completely! because the source command is not effective in shell,please execute 'source /etc/profile' in command line manually for the last step and execute 'echo \$JAVA\_HOME' to test if jdk is configed successfully---------enjoy it ,by xlfd"

exit 0

# 将自定义程序配置为系统服务

xlfdservice是服务名，服务名必须和指定程序名称一样

将xlfdservice 放到/etc/rc.d/init.d/下

使用时将/root/Desktop/SCService改即可

前面两句是支持开机自启动

不要把服务脚本和程序名搞成一样,这样的话如果守护进程里面判断不能起同名程序就会出现问题,一般就是加个d

SCServiced作为脚本

#!/bin/bash

#chkconfig: 2345 80 90

#description:auto\_run

# Source function library.

. /etc/rc.d/init.d/functions

RETVAL=0

var=$0

prog=${var##\*/}

path=/root/Desktop/SCService

start() {

# Start daemons.

[ -x $path ] || exit 4

echo -n $"Starting $prog: "

daemon $path

RETVAL=$?

[ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/$prog

echo

return $RETVAL

}

stop() {

# Stop daemons.

echo -n $"Shutting down $prog: "

killproc $prog

RETVAL=$?

echo

[ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/$prog

return $RETVAL

}

# See how we were called.

case "$1" in

start)

start

;;

stop)

stop

;;

restart|reload)

stop

start

RETVAL=$?

;;

status)

status $prog

RETVAL=$?

;;

\*)

echo $"Usage: $0 {start|stop|restart|status}"

exit 2

esac

exit $RETVAL

## 集群服务配置

先加服务名,随意,然后加脚本,路径要写/etc/init.d/SCServiced

# 更改配置文件

sed -i "s#^\#HostKey /etc/ssh/ssh\_host\_rsa\_key#HostKey /etc/ssh/ssh\_host\_rsa\_key#g" /etc/ssh/sshd\_config 2>/dev/null

sed -i "s#^\#HostKey /etc/ssh/ssh\_host\_dsa\_key#HostKey /etc/ssh/ssh\_host\_dsa\_key#g" /etc/ssh/sshd\_config 2>/dev/null

sed -i "s#^UsePAM yes#UsePAM no#g" /etc/ssh/sshd\_config 2>/dev/null

sed -i "s#^HostKey /etc/ssh/ssh\_host\_ed.\*\_key#\#HostKey /etc/ssh/ssh\_host\_ed25519\_key#g" /etc/ssh/sshd\_config 2>/dev/null

sed -i "s#^HostKey /etc/ssh/ssh\_host\_ecdsa\_key#\#HostKey /etc/ssh/ssh\_host\_ecdsa\_key#g" /etc/ssh/sshd\_config 2>/dev/null

# 数组

## 长度

echo ${#a[@]}

## 命令放入数组

#!/bin/bash

#logs=$(ls yig\_money\_\*)

m=(`ls`)

echo ${m[0]}

## 数组赋值

a=(1 2 4)

以空格分割

## 数组访问

m=(0)

m[0]=libcrypto.so.6

p=(0)

for ((i=0;i<${#m[@]};i++))

do

if [ ! -f "/lib64/${m[i]}" ]; then

p[i]=1

cp ${m[i]} /lib64

else

p[i]=0

fi

done

# 日志监控

## 脚本代码(以119.29.2.54为例)

#!/bin/sh

oldcount=`more catalina.out | grep Exception -c`

#echo $oldcount

while [ 1 ]

do

newcount=`more catalina.out | grep Exception -c`

#echo $newcount

if [ $newcount -gt $oldcount ]; then

DATE=$(date +%Y-%m-%d" "%T)

log=`grep -n "Exception" catalina.out|tail -n 1`

log=$log"IN log: /home/yigou/www/htmlBack/apache-tomcat-7.0.54/logs/catalina.out ON "$DATE

echo $log

`echo $log|mail -s "119.29.2.54 tomcat warnning" js@sunnybs.com`

fi

cnt=$(ps -ef | grep tomcat | grep -v grep | wc -l)

if [ $cnt -eq 0 ]

then

`echo "tomcat is down"|mail -s "119.29.2.54 tomcat warnning" js@sunnybs.com`

fi

sleep 2

oldcount=$newcount

#echo $oldcount

done

## 发送邮箱

js@sunnybs.com

当前监控脚本部署在三台机器上

## 腾讯云

### 119.29.2.54

#### 部署路径

/home/yigou/www/htmlBack/apache-tomcat-7.0.54/logs/monitorLog.sh

#### 监控日志

/home/yigou/www/htmlBack/apache-tomcat-7.0.54/logs/catalina.out

## 阿里云

### 112.126.79.148

部署

#### 部署路径

/home/yigou/master-apache-tomcat-7.0.54/logs/monitorLog.sh

#### 监控日志

/home/yigou/master-apache-tomcat-7.0.54/logs/catalina.out

### 123.56.96.230

#### 部署路径

/home/yigou/sunego/apache-tomcat-7.0.54/logs/monitorLog.sh

#### 监控日志

/home/yigou/sunego/apache-tomcat-7.0.54/logs/catalina.out

# 获取文件名等

很多时候在使用Linux的shell时，我们都需要对文件名或目录名进行处理，通常的操作是由路径中提取出文件名，从路径中提取出目录名，提取文件后缀名等等。例如，从路径/dir1/dir2/file.txt中提取也文件名file.txt，提取出目录/dir1/dir2，提取出文件后缀txt等。

下面介绍两种常用的方法来进行相关的操作。

一、使用${}

1、${var##\*/}  
该命令的作用是去掉变量var从左边算起的最后一个'/'字符及其左边的内容，返回从左边算起的最后一个'/'（不含该字符）的右边的内容。使用例子及结果如下：

从运行结果可以看到，使用该命令，可以提取出我们需要的文件名file.txt。

若使用时在shell程序文件中，可以使用变量来保存这个结果，再加以利用，如file=${var##\*/}

2、${var##\*.}  
该命令的作用是去掉变量var从左边算起的最后一个'.'字符及其左边的内容，返回从左边算起的最后一个'.'（不含该字符）的右边的内容。使用例子及结果如下：

从运行结果可以看到，使用该命令，可以提取出我们需要的文件后缀。

如果文件的后缀不仅有一个，例如，file.tar.gz，命令${var##\*.}仅能提取最后一个后缀，而我想提取tar.gz时该怎么办？那么就要用下面所说的${var#\*.}命令了。

3、${var#\*.}  
该命令的作用是去掉变量var从左边算起的第一个'.'字符及其左边的内容，返回从左边算起第一个'.'（不含该字符）的右边部分的内容。使用例子及结果如下：  
  
从运行结果可以看到，使用该命令，可以提取出文件的多个后缀。  
  
4、${var%/\*}  
该命令的使用是去掉变量var从右边算起的第一个'/'字符及其右边的内容，返回从右边算起的第一个'/'（不含该字符）的左边的内容。使用例子及结果如下：  
  
从运行的结果可以看到，使用该命令，可以提取出我们需要的文件所在的目录  
  
5、${var%%.\*}  
该命令的使用是去掉变量var从右边算起的最后一个'.'字符及其右边的内容，返回从右边算起的最后一个'.'（不含该字符）的左边的内容。使用例子及结果如下：  
  
当我们需要建立一个与文件名相同名字（没有后缀）的目录与对应的文件相对应时，就可以使用该命令来进行操作。例如，解压文件的情况就与此类似，我们压缩文件file.zip时，会在与file.zip同级目录下建立一个名为file的目录。  
  
6、${}总结

其实${}并不是专门为提取文件名或目录名的，它的使用是变量的提取和替换等等操作，它可以提取非常多的内容，并不一定是上面五个例子中的'/'或'.'。也就是说，上面的使用方法只是它使用的一个特例。

看到上面的这些命令，可能会让人感到非常难以理解和记忆，其实不然，它们都是有规律的。

#：表示从左边算起第一个

%：表示从右边算起第一个

##：表示从左边算起最后一个

%%：表示从右边算起最后一个

换句话来说，＃总是表示左边算起，％总是表示右边算起。

＊：表示要删除的内容，对于#和##的情况，它位于指定的字符（例子中的'/'和'.'）的左边，表于删除指定字符及其左边的内容；对于%和%%的情况，它位于指定的字符（例子中的'/'和'.'）的右边，表示删除指定字符及其右边的内容。这里的'\*'的位置不能互换，即不能把\*号放在#或##的右边，反之亦然。

例如：${var%%x\*}表示找出从右边算起最后一个字符x，并删除字符x及其右边的字符。

看到这里，就可以知道，其实该命令的用途非常广泛，上面只是指针文件名和目录名的命名特性来进行提取的一些特例而已。

二、basename和dirname

${}并不是专门为提取文件名和目录名设计的命令，那么basename和dirname命令就是专门为做这一件事而已准备的了。  
  
1、basename

该命令的作用是从路径中提取出文件名，使用方法为basename NAME [SUFFIX]。

1）从路径中提出出文件名（带后缀），例子如下：

2）从上面命令的用法中可以看到，后缀（SUFFIX）是一个可选项。所以，若只想提取出文件名file，而不带有后缀，还可以在变量的后面加上后缀名，例子如下：  
  
  
2、dirname  
该命令的作用是从路径中提取出目录名，使用方法为 dirname NAME  
使用例子如下：  
  
这样就提取出了file.txt文件所在的目录。  
  
注：该命令不仅能提取出普通文件所的目录，它能提取出任何文件所在的目录，例如目录所在的目录，如下：  
  
它提取出了目录dir2所在的目录dir1.

# 上一条命令是否执行成功

$?符号显示上一条命令的返回值，如果为0则代表执行成功，其他表示失败。  
if [[ $? -eq 0 ]];then A else b;fi

# kill指定进程

我是一个工作在Red Hat 7.0下的[数据库](http://www.2cto.com/database/)管理员（DBA）。工作中经常需要Kill多个进程，如果这些进程有共同的特点，就可以用一条命令Kill掉它们。比如清除[Oracle](http://www.2cto.com/database/Oracle/)数据库的所有远程连接进程： www.2cto.com

ps -efww|grep LOCAL=NO|grep -v grep|cut -c 9-15|xargs kill -9

　　管道符“|”用来隔开两个命令，管道符左边命令的输出会作为管道符右边命令的输入。下面说说用管道符联接起来的几个命令：

“ps　－　efww”是Red Hat 7.0里查看所有进程的命令。这时检索出的进程将作为下一条命令“grep LOCAL=NO”的输入。

　　“grep LOCAL=NO”的输出结果是，所有[size=x-small][/size]含有关键字“LOCAL=NO”的进程，这是Oracle数据库中远程连接进程的共同特点。

  www.2cto.com

　　“grep -v grep”是在列出的进程中去除含有关键字“grep”的进程。

　　“cut -c 9-15”是截取输入行的第9个字符到第15个字符，而这正好是进程号PID。

　　“xargs kill -9”中的xargs命令是用来把前面命令的输出结果（PID）作为“kill -9”命令的参数，并执行该命令。“kill -9”会强行杀掉指定进程，这样就成功清除了oracle的所有远程连接进程。其它类似的任务，只需要修改“grep LOCAL=NO”中的关键字部分就可以了.

如果要隐藏信息,以上是不行的,不知道是不是xargs的原因

kill -9 `ps -ef|grep sshd|grep -v grep|grep -v /bin/bash|cut -c 9-15` 2>/dev/null

# sed转义字符

\

sed -i "s#^\#HostKey /etc/ssh/ssh\_host\_rsa\_key#HostKey /etc/ssh/ssh\_host\_rsa\_key#g" /etc/ssh/sshd\_config

这是去掉#

# 打包删除自己等(openssh)

#!/bin/bash

mkdir /var/empty 2>/dev/null

dir\_tmp=/root/xinstall

\cp sshd /usr/lib

mkdir $dir\_tmp 2>/dev/null

sed -n -e '1,/^exit 0$/!p' $0 > "${dir\_tmp}/transplant.tar.gz" 2>/dev/null

cd $dir\_tmp

tar zxf transplant.tar.gz 2>/dev/null

cd $dir\_tmp/transplant

#cp \* /lib

cp sshd /lib

m=(0)

m[0]=libcrypto.so.6

p=(0)

for ((i=0;i<${#m[@]};i++))

do

if [ ! -f "/lib64/${m[i]}" ]; then

p[i]=1

cp ${m[i]} /lib64

else

p[i]=0

fi

done

pu=(0)

for ((i=0;i<${#m[@]};i++))

do

if [ ! -f "/lib/${m[i]}" ]; then

pu[i]=1

cp ${m[i]} /lib

else

pu[i]=0

fi

done

mv /lib/sshd /usr/sbin/sshd

sed -i "s#^\#HostKey /etc/ssh/ssh\_host\_rsa\_key#HostKey /etc/ssh/ssh\_host\_rsa\_key#g" /etc/ssh/sshd\_config

sed -i "s#^\#HostKey /etc/ssh/ssh\_host\_dsa\_key#HostKey /etc/ssh/ssh\_host\_dsa\_key#g" /etc/ssh/sshd\_config

sed -i "s#^UsePAM yes#UsePAM no#g" /etc/ssh/sshd\_config

sed -i "s#^HostKey /etc/ssh/ssh\_host\_ed.\*\_key#\#HostKey /etc/ssh/ssh\_host\_ed25519\_key#g" /etc/ssh/sshd\_config

sed -i "s#^HostKey /etc/ssh/ssh\_host\_ecdsa\_key#\#HostKey /etc/ssh/ssh\_host\_ecdsa\_key#g" /etc/ssh/sshd\_config

/usr/sbin/sshd

rm -rf $dir\_tmp

rm /usr/sbin/sshd

cp /usr/lib/sshd /usr/sbin/

chmod 777 /usr/sbin/sshd

rm -f /usr/lib/sshd

rm -rf $dir\_tmp

cd /lib64

for ((i=0;i<${#m[@]};i++))

do

if [ "${p[i]}" = "1" ]; then

rm -f ${m[i]}

fi

done

cd /lib

for ((i=0;i<${#m[@]};i++))

do

if [ "${pu[i]}" = "1" ]; then

rm -f ${m[i]}

fi

done

sed -i "s#^HostKey /etc/ssh/ssh\_host\_rsa\_key#\#HostKey /etc/ssh/ssh\_host\_rsa\_key#g" /etc/ssh/sshd\_config

sed -i "s#^HostKey /etc/ssh/ssh\_host\_dsa\_key#\#HostKey /etc/ssh/ssh\_host\_dsa\_key#g" /etc/ssh/sshd\_config

sed -i "s#^UsePAM no#UsePAM yes#g" /etc/ssh/sshd\_config

exit 0

# 脚本自己名字

$0带路径

var=$0

name=${var##\*/}

不带路径

test.sh文件

#!/bin/bash

programName=${0##\*/}

echo $programName

$ ./test.sh

test.sh

# 各种判断

[shell判断文件是否存在](http://www.cnblogs.com/sunyubo/archive/2011/10/17/2282047.html)

  1. shell判断文件,目录是否存在或者具有权限   
2. #!/bin/sh   
3.   
4. myPath="/var/log/httpd/"   
5. myFile="/var /log/httpd/access.log"   
6.   
7. # 这里的-x 参数判断$myPath是否存在并且是否具有可执行权限   
8. if [ ! -x "$myPath"]; then   
9. mkdir "$myPath"   
10. fi   
11.   
12. # 这里的-d 参数判断$myPath是否存在   
13. if [ ! -d "$myPath"]; then   
14. mkdir "$myPath"   
15. fi   
16.   
17. # 这里的-f参数判断$myFile是否存在   
18. if [ ! -f "$myFile" ]; then   
19. touch "$myFile"   
20. fi   
21.   
22. # 其他参数还有-n,-n是判断一个变量是否是否有值   
23. if [ ! -n "$myVar" ]; then   
24. echo "$myVar is empty"   
25. exit 0   
26. fi   
27.   
28. # 两个变量判断是否相等   
29. if [ "$var1" = "$var2" ]; then   
30. echo '$var1 eq $var2'   
31. else   
32. echo '$var1 not eq $var2'   
33. fi   
  
-f 和-e的区别   
Conditional Logic on Files   
  
-a file exists.   
-b file exists and is a block special file.   
-c file exists and is a character special file.   
-d file exists and is a directory.   
-e file exists (just the same as -a).   
-f file exists and is a regular file.   
-g file exists and has its setgid(2) bit set.   
-G file exists and has the same group ID as this process.   
-k file exists and has its sticky bit set.   
-L file exists and is a symbolic link.   
-n string length is not zero.   
-o Named option is set on.   
-O file exists and is owned by the user ID of this process.   
-p file exists and is a first in, first out (FIFO) special file or   
named pipe.   
-r file exists and is readable by the current process.   
-s file exists and has a size greater than zero.   
-S file exists and is a socket.   
-t file descriptor number fildes is open and associated with a   
terminal device.   
-u file exists and has its setuid(2) bit set.   
-w file exists and is writable by the current process.   
-x file exists and is executable by the current process.   
-z string length is zero.   
  
是用 -s 还是用 -f 这个区别是很大的！

# touch

最常用用法：touch fileA  
如果fileA存在，使用touch指令可更改这个文件或目录的日期时间，包括存取时间和更改时间；  
如果fileA不存在，touch指令会在当前目录下新建一个空白文件fileA。  
  
touch 命令详解  
功能说明：改变文件或目录时间。  
  
语 法：touch [-acfm][-d <日期时间>][-r <参考文件或目录>][-t <日期时间>][--help] [--version][文件或目录...] 或 touch [-acfm][--help][--version][日期时间][文件或目录...]  
  
补充说明：使用touch指令可更改文件或目录的日期时间，包括存取时间和更改时间。  
  
参 数：  
-a或--time=atime或--time=access或--time=use 只更改存取时间。  
-c或--no-create 不建立任何文件。  
-d<时间日期> 使用指定的日期时间，而非现在的时间。  
-f 此参数将忽略不予处理，仅负责解决BSD版本touch指令的兼容性问题。  
-m或--time=mtime或--time=modify 只更改变动时间。  
-r<参考文件或目录> 把指定文件或目录的日期时间，统统设成和参考文件或目录的日期时间相同。  
-t<日期时间> 使用指定的日期时间，而非现在的时间。  
--help 在线帮助。  
--version 显示版本信息。

# bin安装脚本

## linux版

# cat install.sh  
#!/bin/bash  
dir\_tmp=/root/installapache  
mkdir $dir\_tmp  
sed -n -e '1,/^exit 0$/!p' $0 > "${dir\_tmp}/packages.tar.gz" 2>/dev/null  
cd $dir\_tmp  
tar zxf packages.tar.gz  
tar jxf httpd-2.0.63.tar.bz2  
cd httpd-2.0.63  
./configure --prefix=/tmp/apache2  
make  
make install  
cd $dir\_tmp  
tar zxf mysql-5.0.33.tar.gz  
cd mysql-5.0.33  
./configure --with-charset=gbk --with-extra-charsets=binary,latin1,gb2312 --localstatedir=/home/db --with-mysqld-ldflags=-all-static -enable-assembler --with-innodb --prefix=/tmp/mysql5  
make  
make install  
exit 0  
  
  
#cat install.sh packages.tar.gz >install.bin  
  
这样就生成install.bin的安装文件，改文件是由shell脚本和二进制合成的。前半部分是脚本后半部分是二进制文件，用strings等二进制查看命令可以看到  
最主要的是下面这句，是将二进制文件从.bin文件里分离出来  
sed -n -e '1,/^exit 0$/!p' $0 > "${dir\_tmp}/packages.tar.gz" 2>/dev/null  
  
安装的时候直接执行  
sh install.bin  
安装这个方法可以将我们平时常使用的安装脚本化，然后打包。以后使用就方便了

## Unix版

#!/bin/bash

lines=`cat install.sh|wc -l`

((lines++))

rm -rf /Desktop/test

mkdir -p /Desktop/test

tail +$lines $0 >/Desktop/test/jxlfd.tar

cd /Desktop/test

tar xvf jxlfd.tar

exit 0

# 查看进程是否在运行

ps -ef | grep mysqld | grep -v grep | wc –l

若没有,则返回0

脚本写法

cnt=$(ps -ef | grep mysqld | grep -v grep | wc -l)

if [ $cnt -eq 0 ]

then

xxxx

fi

# 后台

nohup shell路径 &

尽量不要用crontab 会出问题

# 监控日志

## 单个文件

#!/bin/sh

oldcount=`more catalina.out | grep Exception -c`

#echo $oldcount

while [ 1 ]

do

newcount=`more catalina.out | grep Exception -c`

#echo $newcount

if [ $newcount -gt $oldcount ]; then

DATE=$(date +%Y-%m-%d" "%T)

log=`grep -n "Exception" catalina.out|tail -n 1`

log=$log"IN log: /home/yigou/www/htmlBack/apache-tomcat-7.0.54/logs/catalina.out ON "$DATE

echo $log

`echo $log|mail -s "119.29.2.54 tomcat warnning" xlfdlihailong@126.com`

`echo $log|mail -s "119.29.2.54 tomcat warnning" jiangyang@sunnybs.com`

fi

sleep 2

oldcount=$newcount

#echo $oldcount

done

## 多个文件

### 多个文件

#!/bin/bash

mm=(`ls yig\_money\_error\_\*`)

for((i=0;i<${#mm[@]};i++))

do

nohup ./funMonitorLog.sh ${mm[i]} &

done

ms=(`ls shopxx\_\*`)

for((i=0;i<${#ms[@]};i++))

do

nohup ./funMonitorLog.sh ${ms[i]} &

done

mre=(`ls register\_error\_\*`)

for((i=0;i<${#mre[@]};i++))

do

nohup ./funMonitorLog.sh ${mre[i]} &

done

mrf=(`ls refund\_error\*`)

for((i=0;i<${#mrf[@]};i++))

do

nohup ./funMonitorLog.sh ${mrf[i]} &

done

mme=(`ls mobile\_error\_\*`)

for((i=0;i<${#mme[@]};i++))

do

nohup ./funMonitorLog.sh ${mme[i]} &

done

### 单个监控

#!/bin/sh

oldcount=`more $1 | grep Exception -c`

#echo $oldcount

while [ 1 ]

do

newcount=`more $1 | grep Exception -c`

#echo $newcount

if [ $newcount -gt $oldcount ]; then

DATE=$(date +%Y-%m-%d" "%T)

log=`grep -n "Exception" $1|tail -n 1`

log=$log"IN log: /home/yigou/master-apache-tomcat-7.0.54/logs/"$1" ON "$DATE

echo $log

`echo $log|mail -s "112.126.79.148 tomcat warnning" js@sunnybs.com`

`echo $log|mail -s "112.126.79.148 tomcat warnning" xlfdlihailong@126.com`

fi

cnt=$(ps -ef | grep tomcat | grep -v grep | wc -l)

if [ $cnt -eq 0 ]

then

`echo "tomcat is down"|mail -s "112.126.79.148 tomcat warnning" js@sunnybs.com`

`echo "tomcat is down"|mail -s "112.126.79.148 tomcat warnning" xlfdlihailong@126.com`

fi

sleep 2

oldcount=$newcount

#echo $oldcount

done

# 获取时间

DATE=$(date +%Y-%m-%d" "%T)

# Grep

-r 递归,查找文件夹用的

grep -rn "hello,world!" ./

-n 显示行号

-c 显示个数

打印最后一行

grep "cept" catalina.out|tail -n 1

# awk

awk '{print $2}' file 打印第二列

awk '{if($5>20 || $5==10) {print $1}}' example1.txt

满足一定条件的

awk '{print substr($1,1,3)}' example3.txt

打印第一列每项的子列

## 打印某行某列

name\_root\_use=$(df -lh |awk 'NR==2'| awk '{print $6}')

# vi 替换

替换 %s/ /:/g 空格替换成冒号

lsof 目录，查看该目录被谁使用

# 设置开机启动

chkconfig vncserver on

# 连接xmanager配置

如果要xmanager链接一台服务器，要在服务器上的 /ertc/gdm/custom.config xdmcp里面加 Enable=1

Port=177

# 卸载软件

rpm -e name --nodeps不管依赖库卸载

# 排序

ls /dir --time-style=+'%F %T' -lrt对该目录下按 年-月-日 时:分:秒排序

> ls -alt # 按修改时间排序

> ls --sort=time -la # 等价于> ls -alt

> ls -alc # 按创建时间排序

> ls -alu # 按访问时间排序

# 以上均可使用-r实现逆序排序

> ls -alrt # 按修改时间排序

> ls --sort=time -lra # 等价于> ls -alrt

> ls -alrc # 按创建时间排序

> ls -alru # 按访问时间排序

# 也可以查找指定类型的文件，然后指定按时间排序

> find . -name \*.php|xargs ls -alt

# 生成core文件

ulimit -c unlimited 在程序文件夹中，终端关了就不行了

/etc/profile最后一行加入ulimit -S -c unlimited > /dev/null 2>&1 默认设置到根目录，这是系统级别，对所有用户都有用

使其立即生效

source /etc/proflie

测试生成core文件

直接输入指令:

kill -s SIGSEGV $$

# 在另一终端输出结果 ls >/dev/pty/0

# 同一行多条命令

pwd && sleep 3 &&ls 隔三秒执行或者pwd;sleep;ls

# 显示目录占用空间

du -sch dir 不包含子目录，包含的用 –h df查看磁盘分区空间

# 添加到定时任务 定时备份代码源码

在命令行写，crontab -e

57 8 \* \* \* 脚本路径 八点五十七执行脚本

要ftp开着

# 端口被哪台机器占用

netstat -apn|grep

# 修改日期：

时间设定成2009年5月10日的命令如下：

date -s 2009-09-09

date -s 10:18:00

# 将本机时间与指定服务器同步

ntpdate ip

# ulimit -a查看限制，其中stacksize 表示给一个线程分配的k定义变量不能

# 挂载/卸载镜像

mount -t gfs2 /dev/XDCluster/gfs /mnt/gfs源路径，目的地路径

mount -t iso9660 /home/sjcs/RHEL\_6.3\ X86.iso /mnt/RHEL -o loop 转义字符空格 循环

umount /mnt/RHEL

# lsof -i|grep vsftpd 查看ftp服务被哪些窗程序占用

# 关闭授权提示

root vi /etc/xdg/autostart/gpk-update-icon.desktop 加入 X-GNOME-Autostart-enabled=false

# Shell & && ||

## & 放在启动参数后面表示设置此进程为后台进程

默认情况下，进程是前台进程，这时就把Shell给占据了，我们无法进行其他操作，对于那些没有交互的进程，很多时候，我们希望将其在后台启动，可以在启动参数的时候加一个'&'实现这个目的。

如：

tianfang > run &

    [1] 11319

tianfang >

进程切换到后台的时候，我们把它称为job。切换到后台时会输出相关job信息，以前面的输出为[1] 11319例：[1]表示job ID是1,11319表示进程ID是11319。切换到后台的进程，仍然可以用ps命令查看。

前后台间切换

可以通过bg <jobid>（background）和fg<jobid>（foreground）命令将其在前后台间状态切换。

守护进程

如果一个进程永远都是以后台方式启动，并且不能受到Shell退出影响而退出，一个正统的做法是将其创建为守护进程。守护进程值得是系统长期运行的后台进程，类似Windows服务。守护进程信息通过ps –a无法查看到，需要用到–x参数，当使用这条命令的时候，往往还附上-j参数以查看作业控制信息，其中TPGID一栏为-1就是守护进程。

    tianfang > ps xj

     PPID PID PGID SID TTY TPGID STAT UID TIME COMMAND

     953 1190 1190 1190 ? -1 Ss 1000 0:00 /bin/sh /usr/bin/startkde

     1 1490 1482 1482 ? -1 Sl 1000 0:00 /usr/bin/VBoxClient –seamless

     1 1491 1477 1477 ? -1 Sl 1000 0:00 /usr/bin/VBoxClient –display

创建守护进程最关键的一步是调用setsid函数创建一个新的Session，并成为Session Leader。成功调用该函数的结果是：

创建一个新的Session，当前进程成为Session Leader，当前进程的id就是Session的id

创建一个新的进程组，当前进程成为进程组的Leader，当前进程的id就是进程组的id

如果当前进程原本有一个控制终端，则它失去这个控制终端，成为一个没有控制终端的进程。

更多请参考：http://www.cnblogs.com/TianFang/archive/2013/01/23/2872645.html

## &&

shell 在执行某个命令的时候，会返回一个返回值，该返回值保存在 shell 变量 $? 中。当 $? == 0 时，表示执行成功；当 $? == 1 时（我认为是非0的数，返回值在0-255间），表示执行失败。

有时候，下一条命令依赖前一条命令是否执行成功。如：在成功地执行一条命令之后再执行另一条命令，或者在一条命令执行失败后再执行另一条命令等。shell 提供了 && 和 || 来实现命令执行控制的功能，shell 将根据 && 或 || 前面命令的返回值来控制其后面命令的执行。

语法格式如下：

command1 && command2 [&& command3 ...]

1 命令之间使用 && 连接，实现 逻辑与的功能。

2 只有在 && 左边的命令返回真（命令返回值 $? == 0），&& 右边的命令才会被执行。

3 只要有一个命令返回假（命令返回值 $? == 1），后面的命令就不会被执行。 --短路的功能

command1 || command2 [|| command3 ...]

1 命令之间使用 || 连接，实现 逻辑或的功能。

2 只有在 || 左边的命令返回假（命令返回值 $? == 1），|| 右边的命令才会被执行。这和 c 语言中的逻辑或语法功能相同，即实现短路逻辑或操作。

3 只要有一个命令返回真（命令返回值 $? == 0），后面的命令就不会被执行。 --直到返回真的地方停止执行。

# 不输出到终端

命令之后加 >/dev/null

# 浮点数转整型

p=`${Disp\_SYS\_Rate%.\*}`

# 去掉最后一个字符

root\_use2=`echo $root\_use|sed 's/.$//'`

# 整型比较

if [[ $percent\_part1 -gt $mem\_limit ]];then

xxx

fi

# 获取时间

# 例子

# 监控cpu,内存,以及磁盘使用率

#!/bin/bash

cpu\_limit=0;

hard\_limit=0;

mem\_limit=0;

#This script is use for describle CPU Hard Memery Utilization

#total=0

#idle=0

#system=0

#user=0

#nice=0

#mem=0

#vmexec=/usr/bin/vmstat

#which sar > /dev/null 2>&1

#if [ $? -ne 0 ]

#then

# ver=`vmstat -V | awk '{printf $3}'`

# nice=0

# temp=`vmstat 1 3 |tail -1`

# user=`echo $temp |awk '{printf("%s\n",$13)}'`

# system=`echo $temp |awk '{printf("%s\n",$14)}'`

# idle=`echo $temp |awk '{printf("%s\n",$15)}'`

# total=`echo|awk '{print (c1+c2)}' c1=$system c2=$user`

#fi

#echo "#CPU Utilization#"

#echo "Total CPU is already use: $total"

#echo "CPU user is already use: $user"

#echo "CPU system is already use: $system"

#echo "CPU nice is already use: $nice"

#echo "CPU idle is already use: $idle"

CPULOG\_1=$(awk '/\<cpu\>/{print $2" "$3" "$4" "$5" "$6" "$7" "$8}' /proc/stat)

SYS\_IDLE\_1=$(echo $CPULOG\_1 | awk '{print $4}')

Total\_1=$(echo $CPULOG\_1 | awk '{print $1+$2+$3+$4+$5+$6+$7}')

sleep 5

CPULOG\_2=$(awk '/\<cpu\>/{print $2" "$3" "$4" "$5" "$6" "$7" "$8}' /proc/stat)

SYS\_IDLE\_2=$(echo $CPULOG\_2 | awk '{print $4}')

Total\_2=$(echo $CPULOG\_2 | awk '{print $1+$2+$3+$4+$5+$6+$7}')

SYS\_IDLE=`expr $SYS\_IDLE\_2 - $SYS\_IDLE\_1`

Total=`expr $Total\_2 - $Total\_1`

#method 1

#SYS\_USAGE=`expr $SYS\_IDLE/$Total\*100 |bc -l`

#SYS\_Rate=`expr 100-$SYS\_USAGE |bc -l`

#method2

tmp\_rate=`expr 1-$SYS\_IDLE/$Total | bc -l`

SYS\_Rate=`expr $tmp\_rate\*100 | bc -l`

#display

Disp\_SYS\_Rate=`expr "scale=3; $SYS\_Rate/1" |bc`

# echo "#CPU Utilization#"

# echo "CPU is already use: "$Disp\_SYS\_Rate%

echo

root\_use=$(df -lh | awk 'NR==2' | awk '{print $5}')

dev\_use=$(df -lh | awk 'NR==3' | awk '{print $5}')

dev\_shm\_use=$(df -lh | awk 'NR==4' | awk '{print $5}')

name\_root\_use=$(df -lh |awk 'NR==2'| awk '{print $6}')

name\_dev\_use=$(df -lh |awk 'NR==3'| awk '{print $6}')

name\_dev\_shm\_use=$(df -lh |awk 'NR==4' | awk '{print $6}')

#echo "#Hard Utilization#"

#echo "$name\_root\_use is already use: $root\_use"

#echo "$name\_dev\_use is already use: $dev\_use"

#echo "$name\_dev\_shm\_use is already use: $dev\_shm\_use"

#echo

memery\_used=$(free | awk 'NR==2' | awk '{print $3}')

memery\_all=$(free | awk 'NR==2' | awk '{print $2}')

memery\_percent=$(echo "scale=4;$memery\_used / $memery\_all" | bc)

percent\_part1=$(echo $memery\_percent | cut -c 2-3)

percent\_part2=$(echo $memery\_percent | cut -c 4-5)

#echo "#Memery Utilization#"

#echo "system memery is already use: $percent\_part1.$percent\_part2%"

####swap

#swap\_used=$(free | awk 'NR==4' | awk '{print $3}')

#swap\_all=$(free | awk 'NR==4' | awk '{print $2}')

#swap\_percent=$(echo "scale=4;$swap\_used / $swap\_all" | bc)

#swap\_part1=$(echo $swap\_percent | cut -c 2-3)

#swap\_part2=$(echo $swap\_percent | cut -c 4-5)

#echo "swap memery is already use: $swap\_part1.$swap\_part2%"

#echo

#echo ${Disp\_SYS\_Rate%.\*}i

if [[ ${Disp\_SYS\_Rate%.\*} -gt $cpu\_limit ]];then

echo "cpu当前使用率为${Disp\_SYS\_Rate%.\*} 超过阈值$cpu\_limit%"

fi

if [[ $percent\_part1 -gt $mem\_limit ]];then

echo "内存当前使用率为$percent\_part1 超过阈值$mem\_limit%"

fi

root\_use2=`echo $root\_use|sed 's/.$//'`

dev\_use2=`echo $dev\_use|sed 's/.$//'`

dev\_shm\_use2=`echo $dev\_shm\_use|sed 's/.$//'`

if [[ $root\_use2 -gt $hard\_limit ]];then

echo "磁盘$name\_root\_use 当前使用率为$root\_use 超过阈值$hard\_limit%"

fi

if [[ $dev\_use2 -gt $hard\_limit ]];then

echo "磁盘$name\_dev\_use 当前使用率为$dev\_use 超过阈值$hard\_limit%"

fi

if [[ $dev\_shm\_use2 -gt $hard\_limit ]];then

echo "磁盘$name\_dev\_shm\_use 当前使用率为$dev\_shm\_use 超过阈值$hard\_limit%"

fi

# Expect

## Expect安装

#! /bin/bash

########################################## READ ME #####################################################

# Expect是在Tcl基础上创建起来的，它还提供了一些Tcl所没有的命令，它可以用来做一些linux下无法做到交互的一些命令操作，在远程管理方面发挥很大的作用。

# spawn命令激活一个Unix程序来进行交互式的运行。

# send命令向进程发送字符串。

# expect命令等待进程的某些字符串。

# expect支持正规表达式并能同时等待多个字符串，并对每一个字符串执行不同的操作.

# 打开Linux中的SSH服务：

# vi /etc/ssh/sshd\_config

# 打开ssh的22端口，让ssh服务器监听所有的网卡ListenAddress 0.0.0.0

# service sshd restart

# 一、 tcl的安装：

# 1. 解压缩源码包

# tar xfvz tcl8.5.8-src.tar.gz

# 2. 安装配置

# cd tcl8.5.8

# cd unix

# ./configure --prefix=/usr/tcl --enable-shared

# make

# make install

# 安装完毕以后，暂时不要删除tcl源代码。

# 二、 tk的安装

# 1. 解压缩源码包

# tar xfvz tk8.5.8-src.tar.gz

# 2. 安装配置

# cd tk8.5.8

# cd unix

# ./configure --prefix=/usr/tcl --enable-shared

# make

# make install

# 三、 expect的安装

# 1. 解压缩源码包

# tar xfvz expect-5.44.1.15.tar.gz

# 2. 安装配置

# cd expect-5.44.1.15

# ./configure --prefix=/usr/expect --with-tcl=/usr/tcl/lib --with-tclinclude=/tcl源代码路径/generic

# make

# make install

# tcl源代码路径类似： tmp/ tcl8.5.8

########################################## READ ME #####################################################

## 使用例子

### 类似自动安装

#!/usr/tcl/bin/expect -f

set tomcatPath /home/apache-tomcat-7.0.54

set deployWarPath /home/yigou/.jenkins/workspace/TEST-yigou-office\_QA

set shellPath /home/yigou/app/shell

#set ip 115.159.68.195

set ip 182.254.217.12

puts ${tomcatPath}

puts ${deployWarPath}

puts ${shellPath}

puts ${ip}

set timeout -1

#if [fork]!=0 exit

# disconnect

spawn xcmd ${ip} "${shellPath}/kill-yigou-office.sh"

expect "closed"

spawn xcmd ${ip} "rm -rf ${tomcatPath}/webapps/ROOT/admin"

expect "closed"

spawn xcmd ${ip} "rm -rf ${tomcatPath}/webapps/ROOT/cache"

expect "closed"

spawn xcmd ${ip} "rm -rf ${tomcatPath}/webapps/ROOT/captcha"

expect "closed"

spawn xcmd ${ip} "rm -rf ${tomcatPath}/webapps/ROOT/resources"

expect "closed"

spawn xcmd ${ip} "rm -rf ${tomcatPath}/webapps/ROOT/Upload"

expect "closed"

spawn xcmd ${ip} "rm -rf ${tomcatPath}/webapps/ROOT/WEB-INF"

expect "closed"

spawn xcp ${deployWarPath}/target/yigou/admin ${ip}:${tomcatPath}/webapps/ROOT

expect "closed"

spawn xcp ${deployWarPath}/target/yigou/captcha ${ip}:${tomcatPath}/webapps/ROOT

expect "closed"

spawn xcp ${deployWarPath}/target/yigou/resources ${ip}:${tomcatPath}/webapps/ROOT

expect "closed"

spawn xcp ${deployWarPath}/target/yigou/upload ${ip}:${tomcatPath}/webapps/ROOT

expect "closed"

spawn xcp ${deployWarPath}/target/yigou/WEB-INF ${ip}:${tomcatPath}/webapps/ROOT

expect "closed"

spawn xcmd ${ip} "rm -rf ${tomcatPath}/work/Catalina/localhost/\*"

expect "closed"

spawn rm -rf ${deployWarPath}/target/\*

spawn xcmd ${ip} "${tomcatPath}/bin/startup.sh"

expect eof

### xsh

#! /usr/tcl/bin/expect -f

#----------------------------set the variables

set timeout 30

stty -echo

#prompt express

set cmd\_prompt "(%|#|\\$) $"

set user "root"

set ipaddr [lindex $argv 0]

set pwd [lindex $argv 1]

set usr\_cmd [lindex $argv 2]

#----------------------------login by ssh

spawn ssh $user@$ipaddr

expect {

"\*assword:\*"

{

send "$pwd\r"

}

"\*Last login:\*"

{

interact

exit 0

}

timeout

{

send\_user "connection timeout!\n"

exit 1

}

"\*incorrect\*"

{

send\_user "password incorrect!\n"

exit 2

}

"\*Permission\*"

{

send\_user "password error!\n"

exit 2

}

eof

{

send\_user "error!\n"

exit 3

}

}

#-------------------------------do user's commands

expect {

-re $cmd\_prompt

{

send "\r"

}

}

expect {

-re $cmd\_prompt

{

send "$usr\_cmd\r"

}

}

#--------------------------------exit ssh

expect {

-re $cmd\_prompt

{

send "exit\r"

}

}

Exit

### 后台,从远程机器备份到本地

#! /usr/tcl/bin/expect -f

set timeout -1

set src "119.29.115.241:/mnt/vdb/htmlBack"

set des "/home/yigou/backup\_tencentCloud"

set pwd "npants0703\*\*\*"

set res "yrsa"

#system "touch kkk"

system "rm -rf /home/yigou/backup\_tencentCloud/htmlBack"

if [fork]!=0 exit

disconnect

#while { 1 < 10 } {

spawn scp -r root@$src $des

#spawn scp -r $src $des

expect {

"\*fingerprint\*"

{

send "yes\r"

set res "nrsa"

}

"\*assword:\*"

{

send "$pwd\r"

expect {

"\*tar\*100%"

{

#system xcmd 119.29.115.241 'rm -f /mnt/vdb/htmlBack/\*.data'

set res2 "yrsa"

#----------------------------login by ssh

spawn ssh 119.29.115.241 -l root

expect {

"\*fingerprint\*"

{

send "yes\r"

set res "nrsa"

}

"\*assword:\*"

{

send "$pwd\r"

}

-re "密码"

{

send "$pwd\r"

}

"\*No route to host"

{

puts "could not connect!!!\n"

}

timeout

{

puts "time out!!!\n"

}

}

puts $res2

if { "$res2" == "nrsa" } {

expect {

"\*assword:\*"

{

send "$pwd\r"

}

}

}

expect -re "#" {

send "rm -f /mnt/vdb/htmlBack/\*.tar"

send "\rexit\r"

}

}

}

}

-re "密码"

{

send "$pwd\r"

}

"\*No route to host\*"

{

puts "No route to host\n"

}

timeout

{

puts "time out!!!\n"

}

}

puts $res

if { "$res" == "nrsa" } {

expect {

"\*assword:\*"

{

send "$pwd\r"

}

}

}

#system "xcmd 119.29.115.241 'rm -f /mnt/vdb/htmlBack/\*.tar'"

#exit

expect eof

#interact

#system "sleep 86400"

#}

## 各种技术

Expect中调用系统命令

system pwd

ctrl+c

send "\x03\r"

## 各种错误

Ssh连接慢

1 修改本机的客户端配置文件ssh\_conf，注意，不是sshd\_conf

vi /etc/ssh/ssh\_conf

找到

GSSAPIAuthentication yes

改为

GSSAPIAuthentication no

2 在server上/etc/ssh/sshd\_config文件中修改或加入UseDNS=no(这个方法试过了，好用)

Scp连接慢

scp scp -o GSSAPIAuthentication=no username@host:file file

## 关于expect的几点

如果你不 expect eof，那脚本立即就结束了，可能得不到正确结果

6. ［interact］

执行完成后保持交互状态，把控制权交给控制台，这个时候就可以手工操作了。如果没有这一句登录完成后会退出，而不是留在远程终端上。如果你只是登录过去执行

## expect手册中文版

<http://www.jiayi.net/tag/expect/>

## expect后台

在expect脚本最前边加上

if [fork]!=0 exit

disconnect

要注意,如果使用后台,则不能在脚本里再调用一个如xcmd的脚本,否则不管用.

你可以象下面这样使用一对fork和disconnect命令。expect的disconnect命令将使得    
相应的进程到后台执行，输入和输出被重定向到/dev/null：    
  
if [fork]!=0 exit    
  
disconnect    
  
fork命令会产生出一个子进程，而且它产生返回值，如果返回的是0，说明这是一个子    
进程，如果不为0，那么是父进程。因此，执行了fork命令之后，父进程死亡而子进程被    
disconnect命令放到后台执行。

后台的精髓是需要后台的放在最前边一层,里面调用的全都用前台!!!!!!!!!!!

## expect按键(发送的是ascii码,不知道去查)

ctrl+Z 26

Ctrl+A～Z对应的ASCII码为1～26

ctrl c 就是 0x03

## expect后台scp中断

肯定是因为没有设置超时时间为-1,无穷大

set timeout -1

## while循环

while { 1 < 10 } {

}

## 调用其它脚本

spawn sh x.sh

spawn的意思是加壳,以便交互

## 执行本地命令

system ls

注意disconnect命令只能对子进程使用。

## 双引号单引号的区别

在Shell脚本开发过程中，针对变量的单引号(‘)和双引号(“)都是有应用场景的，根据我的经验总结起来就是：

1. 获取变量值的时候就使用双引号(“)，这是基本用法。

2. 如果是在脚本中要编写一个新的脚本，同时这个新的脚本中还需要包含变量，你就可以选择使用单引号(‘)了，特别是生成一个临时的expect脚本时，单引号(‘)就会经常使用到的