Tedu.cn
达内教育

Shell脚本调试技术

Shell

Shell脚本调试技术

知识讲解

- shell 编程在 unix/linux 世界中使用得非常广泛，熟练掌握 shell 编程也是成为一名优秀的 unix/linux 系统管理员的必经之路。

- 什么是调试技术？
  - 脚本调试的主要工作就是发现引发脚本错误的原因以及在脚本源代码中定位发生错误的行，常用的手段包括分析输出的错误信息，通过在脚本中加入调试语句，输出调试信息来辅助诊断错误，利用调试工具等。

Shell脚本调试技术

- shell 脚本调试技术的问题
  - 但与其它高级语言相比，shell解释器缺乏相应的调试机制和调试工具的支持，其输出的错误信息又往往很不明确，初学者在调试脚本时，除了知道用 echo 语句输出一些信息外，别无它法，而仅仅依赖于大量的加入 echo 语句来诊断错误，确实令人不胜其繁，故常见初学者抱怨shell脚本太难调试了。

知识讲解

在脚本中输出调试信息

Shell脚本调试技术

- 基本调试技术 echo
  - 把 id 数组里面的奇数都删除

```
#!/bin/bash
id=(1 2 3 4 5 6 7 8 9)
for ((i=0;i<${#id[@]};i++));do
    if (($((id[i]%2))));then
        unset id[i]
    fi
done
echo ${id[@]}
```

知识讲解

Shell脚本调试技术

- 基本调试技术 echo
  - 我们在脚本中增加 echo 语句进行调试

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

    echo   "[${i}, ${id[@]}]"

    if (($((id[i]%2))));then
       unset id[i]
    fi
done
```

知识讲解

使用 tee 调试

Shell脚本调试技术

知识讲解

- 在 shell 脚本中管道以及输入输出重定向使用得非常多，在管道的作用下，一些命令的执行结果直接成为了下一条命令的输入。如果我们发现由管道连接起来的一批命令的执行结果并非如预期的那样，就需要逐步检查各条命令的执行结果来判断问题出在哪儿，但因为使用了管道，这些中间结果并不会显示在屏幕上，给调试带来了困难，此时我们就可以借助于tee命令了。tee命令会从标准输入读取数据，将其内容输出到标准输出设备,同时又可将内容保存成文件。

Shell脚本调试技术

- tee 调试
  - 例如获取本机的ip地址

```
ipaddr=$(ifconfig eth0 |grep inet |grep -Po '[0-9.]+')
```

  - 由于使用了很多管道，我们很难定位问题，这时候可以在管道中加入 tee 来把一个阶段的数据获取出来

```
ipaddr=$(ifconfig eth0 |grep inet|tee dump.txt |grep -Po
'[0-9.]+')
```

知识讲解

Tedu.cn
达内教育

使用调试钩子

Shell脚本调试技术

知识讲解

- 基本调试技术 echo
  - 如果能知道执行到第几行程序出错，对我们很有帮助
  - LINENO
  - 代表shell脚本的当前行号，类似于C语言中的内置宏__LINE__，调用方法 ${LINENO}
  - 我们在 echo 语句中加入该变量即可准确知道哪里发生的错误

```
echo  "ERR_LINE: ${LINENO} ${i} ${id}"
```

Shell脚本调试技术

- 基本调试技术 echo
  - 这样每次增加完都要在调试完成后删掉，非常麻烦
  - 我们可以使用调试钩子

```
DEBUG=0
… …

if ((DEBUG));then
    echo   “[${i}, ${id[@]}]”
fi
```

  - 虽然可以解决删除的问题，但需要在要调试的点预埋很多代码来实现，复杂代码调试困难，而且增加了很多判断，无形中牺牲了脚本的性能。

知识讲解

Shell脚本调试技术

- 我们可以使用调试钩子
  - debug 钩子函数
  - 定义统一函数，不用每次都写判断脚本，减少代码冗余度

```
function DEBUG(){ false && $@; }
… …

DEBUG echo    “[${i}, ${id[@]}]”
```

知识讲解

使用 trap 陷阱

Shell脚本调试技术

知识讲解

- trap 命令用于捕获指定的信号并执行预定义的命令。
- 其基本的语法是:
- trap 'command' signal
- 其中 signal 是要捕获的信号，command 是捕获到指定的信号之后，所要执行的命令。可以用 trap -l 命令看到系统中全部可用的信号名，捕获信号后所执行的命令可以是任何一条或多条合法的 shell 语句，也可以是一个函数名。

Shell脚本调试技术

知识讲解

- shell 脚本在执行时，会产生四个所谓的“伪信号”，(之所以称之为“伪信号”是因为这三个信号是由shell产生的，而其它的信号是由操作系统产生的)，通过使用 trap 命令捕获这四个“伪信号”并输出相关信息对调试非常有帮助
- 信号何时产生　　信号含义
  - EXIT　　整个脚本执行完毕
  - RETURN　　函数退出（必须有 trace 状态）
  - ERR　　当一条命令返回非零状态时
  - DEBUG　　脚本中每一条命令执行之前

Shell脚本调试技术

知识讲解

- 通过捕获 EXIT 信号
  - 我们可以在 shell 脚本中止执行或从函数中退出时，输出某些想要跟踪的变量的值，并由此来判断脚本的执行状态以及出错原因,其使用方法是：
  - trap 'command' EXIT　或　trap 'command' 0
- 通过捕获ERR信号
  - 我们可以方便的追踪执行不成功的命令或函数，并输出相关的调试信息，以下是一个捕获 ERR 信号的示例程序，其中的 $LINENO 是一个shell的内置变量，代表 shell 脚本的当前行号。

Shell脚本调试技术

- 捕获 EXIT 信号

```
#!/bin/bash
function EXITTRAP(){
    echo   “$0 exit status $?"
}
function foo(){
    echo is foo
    return 0;
}
trap 'EXITTRAP' EXIT
foo

echo  THE END
```

知识讲解

Shell脚本调试技术

- 捕获 ERR 信号

```
#!/bin/bash
function ERRTRAP(){
    echo "[LINE:$1] Error: function exit status $?"
}
function foo(){
    echo is foo
    return 0;
}
function bar(){
    echo is bar
    return 1;
}
```

知识讲解

Shell脚本调试技术

- 捕获 ERR 信号，续上页

```
... ...

trap 'ERRTRAP $LINENO' ERR

abc
foo
bar

echo THE END
```

知识讲解

Shell脚本调试技术

- 捕获 DEBUG 信号
  - 在调试过程中，为了跟踪某些变量的值，我们常常需要在 shell 脚本的许多地方插入相同的 echo 语句来打印相关变量的值，这种做法显得烦琐而笨拙。而通过捕获 DEBUG 信号，我们只需要一条 trap 语句就可以完成对相关变量的全程跟踪。

知识讲解

Shell脚本调试技术

- DEBUG 信号调试
  - 脚本样例

```
#!/bin/bash
trap 'echo "before execute line:$LINENO, a=$a,b=$b,c=$c"'
DEBUG
a=1
if [ "$a" -eq 1 ];then
   b=2
else
   b=1
fi
c=3
echo "the end"
```

知识讲解

Shell脚本调试技术

知识讲解

- 其他信号调试
  - 在脚本执行过程中，我们也可以预埋陷阱
  - 可以在需要的时候向脚本发送信号触发陷阱从而获得脚本在运行时的当前状态

```
function  status(){
    获取程序执行状态等相关参数
}

trap 'status ${args}' SIGXXX
```

Shell脚本调试技术

- 给猜数字游戏增加陷阱

```
#!/bin/bash
_N=$((RANDOM%100))
str="You lost !!!"
for((i=0;i<5;i++));do
    read -p "guest id number: " n
    (( n == _N )) && { str="You win !!!"; break; }
    if (( n > _N ));then
       echo "too great"
    else
       echo "too little"
    fi
done
echo "$str id is ${_N}"
```

课堂练习

Shell脚本调试技术

知识讲解

- 多进程陷阱调试
  - 后台进程往往关闭了 stdin,stdout,stderr 这给我们调试代码带来了困难，即使没有关闭，他的输出只是到进程所在的终端中，有时一个程序往往运行一周或更长的时间，终端有可能已经断开，或我们在另一个地方或另一台机器上调试怎么办？
  - 在多进程中我们可以使用管道来解决
  - 进程间通信方式：
  - 信号、管道、队列、共享内存、套接字