Bash源码分析(1)

周荣华

作者简介：10年通讯底层研发经验，熟悉linux/vxworks等实时操作系统的内核原理和实现，在虚拟化的openstack，kubernetes，docker等领域也初有涉猎。

摘要：本文讲述当下留下的linux的bash的源代码，通过代码分析和单步调试解析bash的运行流程，适合喜欢研究linux原理的高级用户。分析的源代码来自gnu的开源项目<https://git.savannah.gnu.org/git/bash.git>，例子是作者自己编写，可以随意引用。

# 1 引言

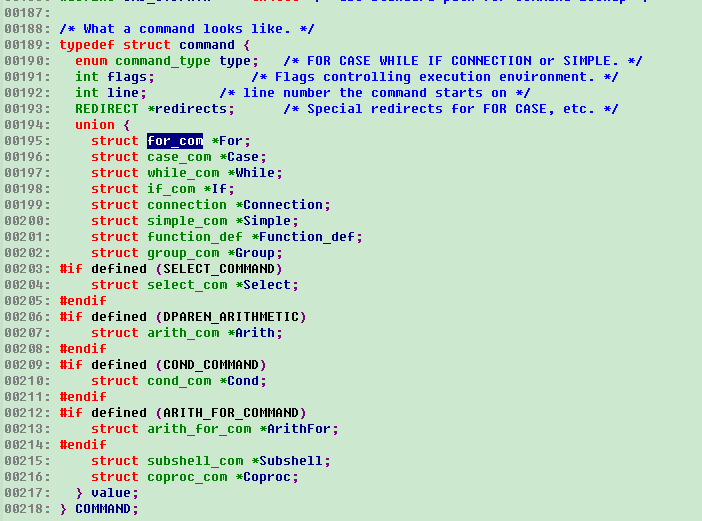
Bash这个程序作为一个linux的用户，用的实在太频繁了，但一般局限于会用就结束了，一直没机会研究bash本身的原理。因工作需要，调试一个bash的cpu冲高问题，趁此机会对bash的源码做了一些研究，希望能对大家有点帮助。

# 2 linux的各种主流shell介绍

现在一般使用的shell有sh，bash和csh这几种，我们这里主要说的是bash，其他shell的源代码逻辑也差不多。

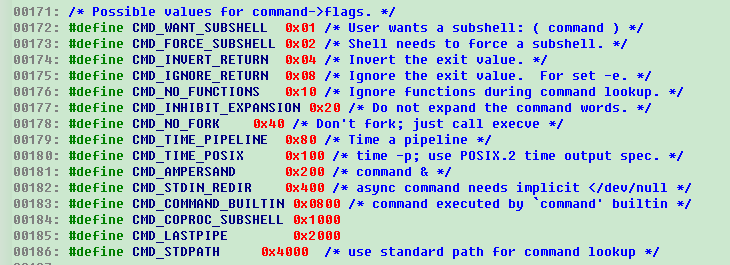
# 3 bash使用到的主要数据结构介绍

## 3.1 COMMAND

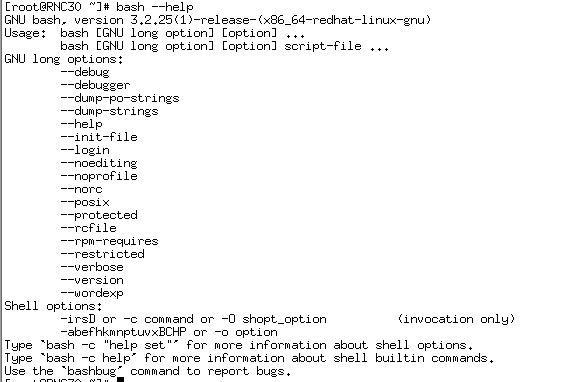


COMMAND是所有数据结构的纲，从这里可以看出一个bash实际能执行的语句有14种，分别位for，case，while，fi，connection，simple\_com，function，group，select，arith，cond，arith\_for，subshell，coproc，其中select，arith，cond，arith\_for这4个命令需要打开对应的编译开关之后才能执行。

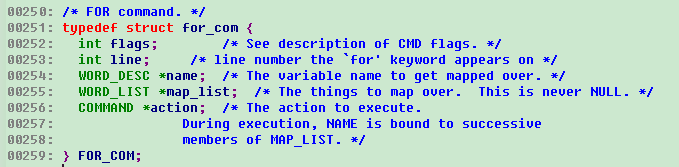
除了下面的这个union外，另外几个属性分别对应命令类型，行号和执行环境控制参数。其中控制参数有很多，每个控制参数占用一个bit位，包括是否启动子shell，是否忽略exit值等。



这些flag可以在bash启动shell脚本时设置，或者在shell脚本内部调用set指令来设置，一般用户不怎么关注，高阶用户可以看看：



## 3.2 FOR\_COM

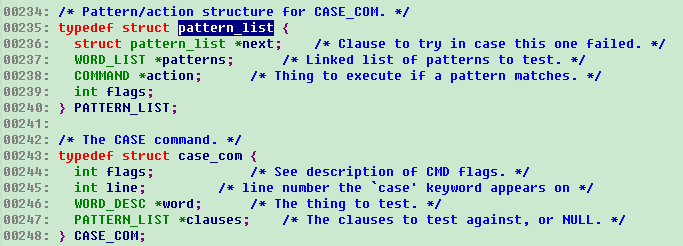


FOR\_COM对应的shell语句是for name in map\_list; do action; done

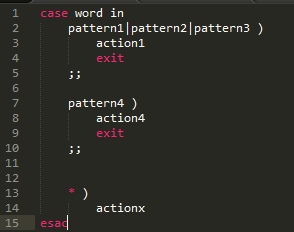
从结构体定义可以看出，除了和COMMAND相同的flags和行号外，for语句是有一个变量名，一个列表和一个递归的COMMAND组成的，实际for循环执行过程中也是将列表中的每个元素拿出来赋值给变量名，并执行action中的脚本段。

从这里的flags，可以看出，每条命令的flags是可以单独设置的，本条命令设置的控制参数可以不影响其他命令的控制参数。

## 3.3 CASE\_COM



对照下面的脚本，可以看出，先判断一个变量，变量判断晚走到复合语句clauses，注意clause最终实现的时候是一个单向链表，链表中每个元素由一个样式的列表和一个执行体action来组成。



# 4 bash脚本的执行过程分析

一个环境上多个sh的cpu占用达到99%，但实际通过ps看，这个sh并没有带任何参数，如果想要知道这个sh在干什么活，为何会一直冲高，还是gdb调试一下比较靠谱（还有一种可选的方法是不断的敲 cat /proc/\*/stack 来反复查看堆栈，多敲敲之后总能抓到几次上下文，其中\*换成对应进程的pid）。

通过下面的调试，可以看到当前执行是一个简单命令(cm\_simple)，通过p \*command-＞value-＞Simple-＞words-＞word 看到当前执行的简单命令在字符串是true。

Breakpoint 1, execute\_command (command=0x10946f0) at execute\_cmd.c:386

386       result = execute\_command\_internal (command, 0, NO\_PIPE, NO\_PIPE, bitmap);

(gdb) p \*command

$9 = {type = cm\_simple, flags = 8, line = 0, redirects = 0x0, value = {For = 0x1094760, Case = 0x1094760, While = 0x1094760,

    If = 0x1094760, Connection = 0x1094760, Simple = 0x1094760, Function\_def = 0x1094760, Group = 0x1094760, Select = 0x1094760,

    Arith = 0x1094760, Cond = 0x1094760, ArithFor = 0x1094760, Subshell = 0x1094760, Coproc = 0x1094760}}

(gdb) p \*command-＞value-＞Simple

$10 = {flags = 8, line = 3, words = 0x1094740, redirects = 0x0}

(gdb) p \*command-＞value-＞Simple-＞words-＞word

$11 = {word = 0x10946d0 "true", flags = 0}

(gdb) c

Continuing.

重新运行，可以看到又跑到了一个简单命令，其命令是"i=i+1"（汗）。

Breakpoint 1, execute\_command (command=0x10947b0) at execute\_cmd.c:386

386       result = execute\_command\_internal (command, 0, NO\_PIPE, NO\_PIPE, bitmap);

(gdb) p \*command

$12 = {type = cm\_simple, flags = 0, line = 0, redirects = 0x0, value = {For = 0x1094ad0, Case = 0x1094ad0, While = 0x1094ad0,

    If = 0x1094ad0, Connection = 0x1094ad0, Simple = 0x1094ad0, Function\_def = 0x1094ad0, Group = 0x1094ad0, Select = 0x1094ad0,

    Arith = 0x1094ad0, Cond = 0x1094ad0, ArithFor = 0x1094ad0, Subshell = 0x1094ad0, Coproc = 0x1094ad0}}

(gdb) p \*command-＞value-＞Simple-＞words-＞word

$13 = {word = 0x1094ab0 "i=i+1", flags = 20}

(gdb) c

Continuing.

通过shell的进程号，查询进程的上下文，发现是从另外一个虚机链接过来的ssh，咨询环境负责人，该虚机是跑测试用例的，之前跑的测试用例不知道为何没有正常停止，测试用例确实就是简单的一行命令：



cat /proc/27538/environ

XDG\_SESSION\_ID=24620SHELL=/bin/bashSSH\_CLIENT=192.168.9.13 45494 22USER=rootPATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/binMAIL=/var/mail/rootPWD=/rootHOME=/rootSHLVL=2LOGNAME=rootSSH\_CONNECTION=192.168.9.13 45494 192.168.9.196 22XDG\_RUNTIME\_DIR=/run/user/0\_=/bin/sh-bash-4.2#