


学习 > Linux

内容

Linux 技巧：让进程在后台可靠运行

概览



申毅

/setsid/&

2008 年 5 月 29 日发布

disown

screen

我们经常会碰到这样的问题，用 telnet/ssh 登录了远程的 Linux 服务器，运行了一些耗时较长的任务，导致任务中途失败。如何让命令提交后不受本地关闭终端窗口/网络断开连接的干扰呢？下面举几个场景选择不同的方式来处理这个问题。

相关主题

评论

nohup/setsid/&

场景：

如果只是临时有一个命令需要长时间运行，什么方法能最简便的保证它在后台稳定运行呢？

解决方法：

我们知道，当用户注销（logout）或者网络断开时，终端会收到 HUP（hangup）信号从而关闭其所有子进程。因此，我们的解决办法就有两种途径：要么让进程忽略 HUP 信号，要么让进程运行在新的会话里从而成为不属于此终端的子进程。

1. nohup

nohup 无疑是我们首先想到的办法。顾名思义，nohup 的用途就是让提交的命令忽略 hangup 1 号。让我们先来看一下 nohup 的帮助信息：

1	NOHUP(1)	User Commands	NOHUP(1)
2			
3	NAME		

```

4      nohup - run a command immune to hangups, with output to a non-tty
5
6  SYNOPSIS
7      nohup COMMAND [ARG]...
8      nohup OPTION
9
10 DESCRIPTION
11      Run COMMAND, ignoring hangup signals.
12
13      --help display this help and exit
14
15      --version
16          output version information and exit

```

概览

可见, nohup 的使用是十分方便的, 只需在要处理的命令前加上 nohup 即可, 标准输出和标准文件中。一般我们可在结尾加上 "&" 来将命令同时放入后台运行, 也可用 ">filename 2>&1" 来

disown

nohup 示例

screen

```

1 [root@pvcent107 ~]# nohup ping www.ibm.com &
2 [1] 3059
3 nohup: appending output to `nohup.out'
4 [root@pvcent107 ~]# ps -ef |grep 3059
5 root      3059   984   0 21:06 pts/3    00:00:00 ping www.ibm.com
6 root      3067   984   0 21:06 pts/3    00:00:00 grep 3059
7 [root@pvcent107 ~]#

```

2. setsid

nohup 无疑能通过忽略 HUP 信号来使我们的进程避免中途被中断, 但如果我们换个角度思考, 信号的终端的子进程, 那么自然也就不会受到 HUP 信号的影响了。setsid 就能帮助我们做到这的帮助信息:

```

1 SETSID(8)                                Linux Programmer's Manual                                SETSID(8)
2
3 NAME
4      setsid - run a program in a new session
5
6 SYNOPSIS
7      setsid program [ arg ... ]
8
9 DESCRIPTION
10     setsid runs a program in a new session.

```

可见 setsid 的使用也是非常方便的, 也只需在要处理的命令前加上 setsid 即可。

setsid 示例

```

1 [root@pvcent107 ~]# setsid ping www.ibm.com
2 [root@pvcent107 ~]# ps -ef |grep www.ibm.com
3 root      31094     1   0 07:28 ?        00:00:00 ping www.ibm.com
4 root      31102 29217   0 07:29 pts/4    00:00:00 grep www.ibm.com

```

```
5 | [root@pvcent107 ~]#
```

值得注意的是，上例中我们的进程 ID(PID)为31094，而它的父 ID (PPID) 为1（即为 init 进程的 ID）。请将此例与 [nohup 例](#) 中的父 ID 做比较。

3. &

这里还有一个关于 subshell 的小技巧。我们知道，将一个或多个命名包含在“()”中就能让这些命令展现出很多有趣的功能，我们现在要讨论的就是其中之一。

概览

当我们将"&"也放入“()”内之后，我们就会发现所提交的作业并不在作业列表中，也就是说，是 [nohup/setsid/&](#) 看看为什么这样就能躲过 HUP 信号的影响吧。

disown

subshell 示例

screen

```
1 | [root@pvcent107 ~]# (ping www.ibm.com &)
2 | [root@pvcent107 ~]# ps -ef |grep www.ibm.com
3 | root      16270      1  0 14:13 pts/4      00:00:00 ping www.ibm.com
4 | root      16278 15362  0 14:13 pts/4      00:00:00 grep www.ibm.com
5 | [root@pvcent107 ~]#
```

对比

从上例中可以看出，新提交的进程的父 ID (PPID) 为1（init 进程的 PID），并不是当前终端的子进程，从而也就不会受到当前终端的 HUP 信号的影响了。

disown

场景：

我们已经知道，如果事先在命令前加上 nohup 或者 setsid 就可以避免 HUP 信号的影响。但是，如果已经运行了命令，该如何补救才能让它避免 HUP 信号的影响呢？

解决方法：

这时想加 nohup 或者 setsid 已经为时已晚，只能通过作业调度和 disown 来解决这个问题了。

```
1 | disown [-ar] [-h] [jobspec ...]
2 |     Without options, each jobspec is removed from the table of
3 |     active jobs.  If the -h option is given, each jobspec is not
4 |     removed from the table, but is marked so that SIGHUP is not
```

```

5      sent to the job if the shell receives a SIGHUP. If no jobspec
6      is present, and neither the -a nor the -r option is supplied,
7      the current job is used. If no jobspec is supplied, the -a
8      option means to remove or mark all jobs; the -r option without
9      a jobspec argument restricts operation to running jobs. The
10     return value is 0 unless a jobspec does not specify a valid
11     job.

```

可以看出，我们可以用如下方式来达成我们的目的。

内容

- 用 `disown -h jobspec` 来使**某个作业**忽略HUP信号。

概览

- 用 `disown -ah` 来使**所有的作业**都忽略HUP信号。

- 用 `setsid /&` 来使**正在运行的作业**忽略HUP信号。

disown

需要注意的是，当使用过 `disown` 之后，会将把目标作业从作业列表中移除，我们将不能再使用 `jobs` 来查看它，但是依然能够用 `ps -ef` 查找到它。

总结

但是还有一个问题，这种方法的操作对象是作业，如果我们在运行命令时在结尾加了 `"&"` 来使它成为一个作业并在后台运行，那么就万事大吉了，我们可以通过 `jobs` 命令来得到所有作业列表。但是如果并没有把当前命令作为作业来运行，如何才能得到它的作业号呢？答案就是用 `CTRL-z`（按住 `Ctrl` 键的同时按住 `z` 键）了！

`CTRL-z` 的用途就是将当前进程挂起（Suspend），然后我们就可以用 `jobs` 命令来查询它的作业号，再用 `bg jobspec` 来将它放入后台并继续运行。需要注意的是，如果挂起会影响当前进程的运行结果，请慎用此方法。

`disown` 示例1（如果提交命令时已经用 `"&"` 将命令放入后台运行，则可以直接使用 `"disown"`）

```

1  [root@pvcent107 build]# cp -r testLargeFile largeFile &
2  [1] 4825
3  [root@pvcent107 build]# jobs
4  [1]+  Running                  cp -i -r testLargeFile largeFile &
5  [root@pvcent107 build]# disown -h %1
6  [root@pvcent107 build]# ps -ef |grep largeFile
7  root      4825    968  1 09:46 pts/4      00:00:00 cp -i -r testLargeFile largeFi
8  root      4853    968  0 09:46 pts/4      00:00:00 grep largeFile
9  [root@pvcent107 build]# logout

```

`disown` 示例2（如果提交命令时未使用 `"&"` 将命令放入后台运行，可使用 `CTRL-z` 和 `"bg"` 将其放入后台

```

1  [root@pvcent107 build]# cp -r testLargeFile largeFile2
2
3  [1]+  Stopped                  cp -i -r testLargeFile largeFile2
4  [root@pvcent107 build]# bg %1
5  [1]+  cp -i -r testLargeFile largeFile2 &
6  [root@pvcent107 build]# jobs
7  [1]+  Running                  cp -i -r testLargeFile largeFile2 &
8  [root@pvcent107 build]# disown -h %1

```

```

9 [root@pvcent107 build]# ps -ef |grep largeFile2
10 root      5790  5577  1 10:04 pts/3    00:00:00 cp -i -r testLargeFile largeF
11 root      5824  5577  0 10:05 pts/3    00:00:00 grep largeFile2
12 [root@pvcent107 build]#

```

screen

内容

场景： 概览

我们已经知道了如何让进程免受 HUP 信号的影响，但是如果有大量这种命令需要在稳定的后台这样的操作呢？
disown

screen

解决方法： 总结

此时最方便的方法就是 screen 了。简单的说，screen 提供了 ANSI/VT100 的终端模拟器，使全屏的伪终端。screen 的参数很多，具有很强大的功能，我们在此仅介绍其常用功能以及简要避免 HUP 信号的影响。我们先看一下 screen 的帮助信息：

```

1 SCREEN(1) SCREEN(1)
2
3 NAME
4     screen - screen manager with VT100/ANSI terminal emulation
5
6 SYNOPSIS
7     screen [ -options ] [ cmd [ args ] ]
8     screen -r [[pid.]tty[.host]]
9     screen -r sessionowner/[[pid.]tty[.host]]
10
11 DESCRIPTION
12     Screen is a full-screen window manager that multiplexes a physical
13     terminal between several processes (typically interactive shells).
14     Each virtual terminal provides the functions of a DEC VT100 terminal
15     and, in addition, several control functions from the ISO 6429 (ECMA
16     48, ANSI X3.64) and ISO 2022 standards (e.g. insert/delete line and
17     support for multiple character sets). There is a scrollback history
18     buffer for each virtual terminal and a copy-and-paste mechanism that
19     allows moving text regions between windows.

```

使用 screen 很方便，有以下几个常用选项：

- 用 `screen -dmS session name` 来建立一个处于断开模式下的会话（并指定其会话名）。
- 用 `screen -list` 来列出所有会话。
- 用 `screen -r session name` 来重新连接指定会话。

- 用快捷键CTRL-a d 来暂时断开当前会话。

screen 示例

```

1 [root@pvcent107 ~]# screen -dmS Urumchi
2 [root@pvcent107 ~]# screen -list
3 There is a screen on:
4      12842.Urumchi      (Detached)
5 1 Socket in /tmp/screens/S-root.
6
7 [root@pvcent107 ~]# screen -r Urumchi

```

概览

当我们用“-r”连接到 screen 会话后，我们就可以在这个伪终端里面为所欲为，再也不用担心 HUP 信号，也不用给每个命令前都加上“nohup”或者“setsid”了。这是为什么呢？让我来看一下下面两

disown

1. 未使用 screen 时新进程的进程树

```

screen
1 [root@pvcent107 ~]# ping www.google.com &
2 [1] 9499
3 [root@pvcent107 ~]# pstree -H 9499
4 init--Xvnc
5      |--acpid
6      |--atd
7      |--2*[sendmail]
8      |--sshd--sshd--bash--pstree
9      |      |--sshd--bash--ping

```

我们可以看出，未使用 screen 时我们所处的 bash 是 sshd 的子进程，当 ssh 断开连接时，HUP 子进程（包括我们新建立的 ping 进程）。

2. 使用了 screen 后新进程的进程树

```

1 [root@pvcent107 ~]# screen -r Urumchi
2 [root@pvcent107 ~]# ping www.ibm.com &
3 [1] 9488
4 [root@pvcent107 ~]# pstree -H 9488
5 init--Xvnc
6      |--acpid
7      |--atd
8      |--screen--bash--ping
9      |--2*[sendmail]

```

而使用了 screen 后就不同了，此时 bash 是 screen 的子进程，而 screen 是 init（PID为1）的 HUP 信号自然不会影响到 screen 下面的子进程了。

总结