

27 前后台进程，灵活切换

更新时间：2019-07-20 17:23:36



“

受苦的人，没有悲观的权利。

——尼采

”

内容简介

1. 前言
2. & 符号和 nohup 命令：后台运行进程
3. Ctrl + Z, jobs, bg 和 fg 命令：控制进程的前后台切换
4. 总结

1. 前言

上一课进程操作和系统重启中，我们简单介绍了进程，也学习了如何列出系统中的进程，如何过滤列表结果，如何结束进程。

这一课我们继续乘胜追击，“一路向北”，来学习进程的后台运行。

我们使用的终端让我们难免有一种感觉：我们每次只能在一个终端中运行一个进程。但其实这是大错特错的。

终端还可以运行后台进程。要使一个进程在后台运行，有几种方法，我们都将学习一下。

2. & 符号和 nohup 命令：后台运行进程

我们到目前为止用终端做的事情都是目所能及的，也就是说：我们运行的命令都是在前台可见的。

这样的一个是好处是我们可以看到命令运行的过程，有什么问题可以及时发现。但是也有缺陷，例如有的命令运行耗时良久，我们又不想无所事事，怎么办呢？难道我开一个终端专门执行一个耗时命令，然后为了能做其他事情，我再启动一个终端，那也很不方便。

而且，这样的规避方法在非图形界面的终端（还记得我们的 `tty2 ~ tty6` 吗？注：新版 Ubuntu 是这样的，旧版的 Ubuntu 是 `tty1 ~ tty6`）中是难以实现的，因为只有一个终端窗口。

所以这课显得尤为重要。

事实上，我们可以在同一个终端中同时运行好几个命令。怎么做呢？就需要用到后台进程的概念。

前台进程和后台进程

默认情况下，用户创建的进程都是前台进程。前台进程从键盘读取数据，并把处理结果输出到显示器。

我们可以看到前台进程的运行过程。例如，使用 `ls` 命令来遍历当前目录下的文件。

```
ls
```

这个程序就运行在前台，它会直接把结果输出到显示器。如果 `ls` 命令需要数据（实际上不需要），那么它会等待用户从键盘输入。

当程序运行在前台时，由于命令提示符（`$`）还未出现，用户不能输入其他命令；即使程序需要运行很长时间，也必须等待程序运行结束才能输入其他命令。

后台进程与键盘没有必然的关系。当然，后台进程也可能会等待键盘输入。

后台进程的优点是不必等待程序运行结束就可以输入其他命令。

那么怎么使一个进程（程序的实例）运行在后台呢？

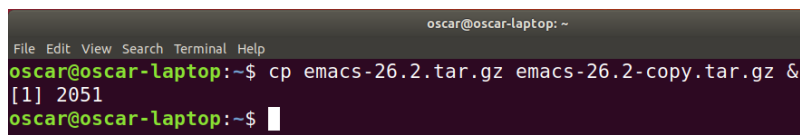
& 符号：在后台运行进程

前面说过，让一个进程在后台运行有几种方法。

我们带大家来学习第一种，很简单：就是在你要运行的命令最后加上 `&` 这个符号。

我们可以用熟悉的 `cp` 命令做例子。例如，我运行 `cp` 命令来拷贝文件：`emacs` 的软件包。当然了，你可以用其他文件来测试。

```
cp emacs-26.2.tar.gz emacs-26.2-copy.tar.gz &
```



```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ cp emacs-26.2.tar.gz emacs-26.2-copy.tar.gz &  
[1] 2051  
oscar@oscar-laptop:~$
```

上图中，因为命令最后加了 `&` 符号，运行时此进程就成为了后台进程。终端输出了一些信息：

```
[1] 2051
```

`[1]`：这是此终端的后台进程的标号。因为这是第一个后台进程，所以标号为 1。

`2051`：这是进程号（PID），如果你想要结束这个后台进程，你可以用我们上一课学习的 `kill` 命令：

我们虽然看不到这个拷贝进程的“所作所为”，但它确实在后台默默进行着文件的拷贝。

因为这个 emacs 的软件包不大，就几十 MB，所以几乎是瞬间就拷贝完毕了。可以用 ls 命令来看看：

```

oscar@oscar-laptop: ~
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ cp emacs-26.2.tar.gz emacs-26.2-copy.tar.gz &
[1] 2051
oscar@oscar-laptop:~$ ls
chinese.txt      file.txt          one_copy          share
Desktop          Music             Pictures          sorted_name.txt
Documents        name.txt          Public            Templates
Downloads        new_file          redirect          test
emacs-26.2-copy.tar.gz  newly_created_file  renamed_file      unique.txt
emacs-26.2.tar.gz  number.txt        repeat.txt        Videos
errors.log        one               results.txt
[1]+  Done                  cp emacs-26.2.tar.gz emacs-26.2-copy.tar.gz
oscar@oscar-laptop:~$

```

可以看到，终端显示了：

```
[1]+  Done  cp emacs-26.2.tar.gz emacs-26.2-copy.tar.gz
```

Done 是英语“完成的”的意思，表示这个后台进程的任务已经完成了。

如果我们用其他命令试一下，例如 find 命令，你也许会比较吃惊。例如，我们运行：

```
sudo find / -name "*log" &
```

意思是：以 root 身份在根目录 / 下查找所有以 log 结尾的文件名的文件，并且在后台运行此进程。在前面加 sudo 是为了防止出现太多 Permission denied 的警告。

```

oscar@oscar-laptop: ~
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ sudo find / -name "*log" &
[1] 2060
oscar@oscar-laptop:~$ find: '/run/user/1000/gvfs': Permission denied
/run/log
/run/systemd/journal/syslog
/run/systemd/journal/dev-log
/run/initramfs/fsck.log
/usr/share/perl5/Parse/DebianChangelog
/usr/share/perl5/Debconf/Element/Dialog
/usr/share/perl5/Dpkg/Changelog
/usr/share/lintian/overrides/rsyslog
/usr/share/alsa/ucm/PandaBoardES/FMAnalog
/usr/share/alsa/ucm/PandaBoard/FMAnalog
/usr/share/alsa/ucm/SDP4430/FMAnalog
/usr/share/doc/ca-certificates/examples/ca-certificates-local/debian/changelog
/usr/share/doc/rsyslog
/usr/share/doc/python3-debian/examples/changelog
/usr/share/doc/python3-debian/examples/changelog/simple_changelog
/usr/share/doc/ppp/examples/scripts/plog
/usr/share/yelp/dtd/catalog
/usr/share/bash-completion/completions/faillog
/usr/share/bash-completion/completions/rlog
/usr/share/bash-completion/completions/lastlog

```

可以看到，这次 find 命令的这个后台进程的进程号是 2060。

你会发现，find 命令虽然在后台运行了，但是终端还是会不断显示所有找到的内容或错误信息。虽然我们还可以在终端中输入其他命令，但是一直会跳出 find 搜索的结果，还是很让人感到厌烦的。最后我不得不把它停止（可以用 Ctrl + C 组合键。或者你用 `kill 2060` 也可以）。

```
File Edit View Search Terminal Help
oscar@oscar-laptop: ~
/var/log/auth.log
/var/log/apt/term.log
/var/log/apt/history.log
/var/log/lastlog
/var/log/unattended-upgrades/unattended-upgrades-dpkg.log
/var/log/unattended-upgrades/unattended-upgrades-shutdown.log
/var/log/unattended-upgrades/unattended-upgrades.log
/var/log/fontconfig.log
/home/oscar/errors.log
/home/oscar/.cache/mozilla/firefox/jmlm5hs9.default/cache2/index.log
/home/oscar/.local/share/gvfs-metadata/home-1a210e73.log
/home/oscar/.local/share/gvfs-metadata/root-c2bdd0b3.log
/home/oscar/.local/share/xorg/Xorg.0.log
/sys/kernel/debug/tracing/events/syscalls/sys_enter_syslog
/sys/kernel/debug/tracing/events/syscalls/sys_exit_syslog
/sys/class/misc/mcelog
/sys/devices/virtual/misc/mcelog
/sys/module/vboxguest/parameters/log
/dev/log
/dev/mcelog
/opt/VBoxGuestAdditions-5.2.28/src/vboxguest-5.2.28/vboxguest/common/log
^C
[1]+  Exit 1                  sudo find / -name "*log"
oscar@oscar-laptop:~$
```

幸好，我们之前学过重定向，我们可以把 find 的输出结果重定向到文件里，就不会再来烦我们了。

```
sudo find / -name "**log" > output_find &
```

这样就不会一直有信息输出了。

当然了，我们还可以更保险一些，将标准错误输出也重定向到同一个文件，这样就不会有任何输出了。

```
sudo find / -name "**log" > output_find 2>&1 &
```

但现在有一个问题：虽然我们的进程是被放到后台了，在终端貌似看不到它的运行过程了。但是此进程还是与此终端相关联的，假如我们把终端关闭，那么这个进程也会结束。

nohup 命令：使进程与终端分离

& 符号虽然常用，但却有一个不可忽视的缺点：后台进程与终端相关联。一旦终端关闭或者用户登出，进程就自动结束。

如果我们想让进程在以上情况下仍然继续在后台运行，那么我们须要用到 nohup 命令。

当用户注销（logout）或者网络断开时，终端会收到 HUP（是 hangup 的缩写，英语“挂断”的意思）信号从而关闭其所有子进程；终端被关闭时也会关闭其子进程。

我们可以用 nohup 命令使命令不受 HUP 信号影响。

我们用 man 来看一下 nohup 命令的解释：

```
File Edit View Search Terminal Help
NOHUP(1) User Commands NOHUP(1)

NAME
    nohup - run a command immune to hangups, with output to a non-tty

SYNOPSIS
    nohup COMMAND [ARG]...
    nohup OPTION

DESCRIPTION
    Run COMMAND, ignoring hangup signals.

    --help display this help and exit

    --version
        output version information and exit

    If standard input is a terminal, redirect it from an unreadable file.
    If standard output is a terminal, append output to 'nohup.out' if possible,
    '$HOME/nohup.out' otherwise. If standard error is a terminal, redirect
    it to standard output. To save output to FILE, use 'nohup COMMAND > FILE'.

Manual page nohup(1) line 1 (press h for help or q to quit)
```

可以看到，nohup 命令的简单描述如下：

```
run a command immune to hangups, with output to a non-tty
```

翻译出来大致就是：“使得运行的命令不受 hangup 信号影响，而且输出会存放到一个非 tty 中”。

nohup 命令的用法很简单：在 nohup 命令之后接要运行的命令。例如，我们可以用 nohup 配合 cp 命令来实现文件的拷贝（这次拷贝的是 node.js 的源码。当然了，你可以用其他文件来测试）：

```
nohup cp node-v10.15.3.tar.gz node-v10.15.3-copy.tar.gz
```

```
File Edit View Search Terminal Help
oscar@oscar-laptop: ~
oscar@oscar-laptop:~$ nohup cp node-v10.15.3.tar.gz node-v10.15.3-copy.tar.gz
nohup: ignoring input and appending output to 'nohup.out'
oscar@oscar-laptop:~$
```

可以看到这次的输出信息是：“ignoring input and appending output to nohup.out”。

大致意思是：“忽略输入，把输出追加到 nohup.out 文件中”。

使用 nohup 命令后，输出会被默认地追加写入到一个叫 nohup.out 的文件里。

现在，我们的进程已经不受终端关闭或者用户断开连接的影响了，会一直运行。当然了，用 kill 命令还是可以结束此进程的。要获知进程号，可以用我们之前学过的 ps 命令配合 grep 来查找。

```
ps -ax | grep command
```

上面命令里的 command 指代 nohup 后面跟着的命令。

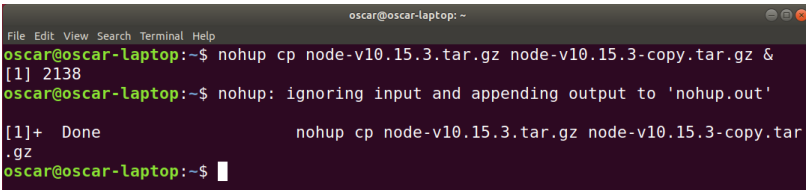
nohup 命令相当有用。想象以下场景：

我登录远程服务器，然后运行了一个耗时命令，或者一个需要一直运行的命令，例如一个游戏的服务器程序。这时假如我掉线了，或者我不小心用 exit 命令退出了登录。那么这个耗时命令也会中止运行。那就很麻烦了。而且，如果这个程序本应该一直运行很久的，我也不可能一直保持登录状态等它结束啊。我家里还有老婆孩子呢，不能不去做饭啊，我要下班... 开个玩笑。

幸好，nohup 命令解决了这样的难题。

一般我们也会把 nohup 和 & 一起使用，例如：

```
nohup cp node-v10.15.3.tar.gz node-v10.15.3-copy.tar.gz &
```



```
[1]+ Done nohup cp node-v10.15.3.tar.gz node-v10.15.3-copy.tar.gz
```

这是在我按下了回车之后显示的。

3. Ctrl + Z, jobs, bg 和 fg 命令：控制进程的前后台切换

我们来考虑一种情况：假如你要将进程转到后台运行，但是执行命令时忘记了在最后加上 & 符号。

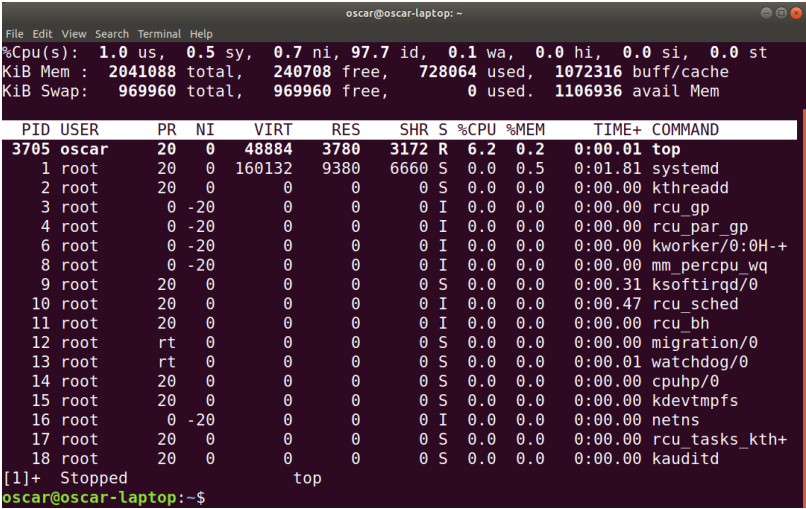
如何再使此进程转为后台进程呢？有几种方法。我们一一来学习。

Ctrl + Z：转到后台，并暂停运行

我们用 top 命令来演示。运行：

```
top
```

因为 top 命令的作用是实时地显示各种系统信息和进程列表。这时，我们按下 Ctrl + Z 这个组合键：



可以看到终端显示了：

```
[1]+ Stopped top
```

这行信息。

stopped 是英语“停止的”的意思，我们又看到 [1] 这个熟悉的信息，表示这是此终端第一个后台进程。

所以表示 top 命令被放到了后台，此进程还是驻留在内存中，但是被暂停运行了。这个时候命令提示符又出现了，我们可以做其他事情了。

bg 命令：使进程转到后台

经过上面的 Ctrl + Z 操作，我们可怜的 top 进程已经被“打入冷宫”（转入后台，并且被暂停运行了）。

但是皇后不甘心啊：“臣妾虽然做不到，但即使在冷宫中，我也要工于心计、运筹帷幄，以期早日打败甄嬛。”

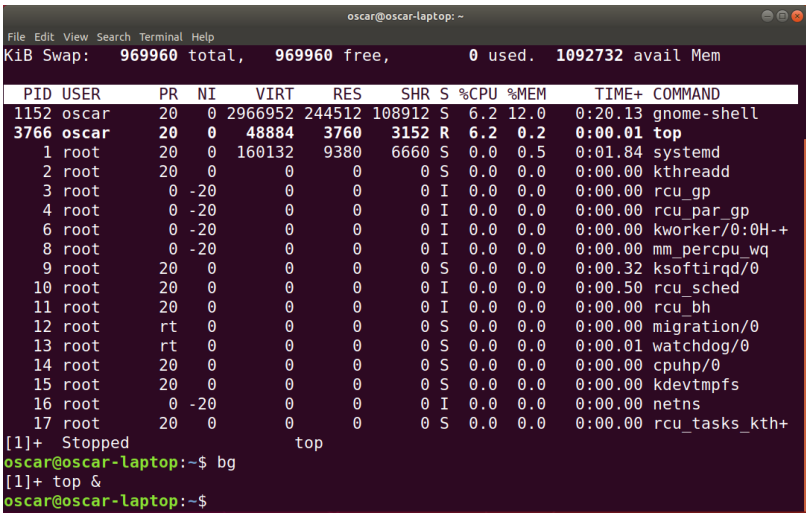
那怎么办呢？可以运行 bg 命令。

就是很简单地输入 bg，然后回车。bg 是英语 background 的缩写，表示“后台”。

bg 命令的作用是将命令转入后台运行。假如命令已经在后台，并且暂停着，那么 bg 命令会将其状态改为运行。

不加任何参数，bg 命令会默认作用于最近的一个后台进程，也就是刚才被 Ctrl + Z 暂停的 top 进程。如果后面加 %1, %2 这样的参数（不带 %，直接 1, 2 这样也可以），则是作用于指定标号的进程。因为进程转入后台之后，会显示它在当前终端下的后台进程编号。例如目前 top 进程转入了后台，它的进程编号是 1（可以由 [1]+ 推断）。依次类推，bg %2 就是作用于编号为 2 的后台进程。

我们输入 bg，然后回车。看到如下输出：



```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
KiB Swap: 969960 total, 969960 free, 0 used, 1092732 avail Mem  


| PID  | USER  | PR | NI  | VIRT    | RES    | SHR    | S | %CPU | %MEM | TIME+   | COMMAND        |
|------|-------|----|-----|---------|--------|--------|---|------|------|---------|----------------|
| 1152 | oscar | 20 | 0   | 2966952 | 244512 | 108912 | S | 6.2  | 12.0 | 0:20.13 | gnome-shell    |
| 3766 | oscar | 20 | 0   | 48884   | 3760   | 3152   | R | 6.2  | 0.2  | 0:00.01 | top            |
| 1    | root  | 20 | 0   | 160132  | 9380   | 6660   | S | 0.0  | 0.5  | 0:01.84 | systemd        |
| 2    | root  | 20 | 0   | 0       | 0      | 0      | S | 0.0  | 0.0  | 0:00.00 | kthreadd       |
| 3    | root  | 0  | -20 | 0       | 0      | 0      | I | 0.0  | 0.0  | 0:00.00 | rcu_gp         |
| 4    | root  | 0  | -20 | 0       | 0      | 0      | I | 0.0  | 0.0  | 0:00.00 | rcu_par_gp     |
| 6    | root  | 0  | -20 | 0       | 0      | 0      | I | 0.0  | 0.0  | 0:00.00 | kworker/0:0H-+ |
| 8    | root  | 0  | -20 | 0       | 0      | 0      | I | 0.0  | 0.0  | 0:00.00 | mm_percpu_wq   |
| 9    | root  | 20 | 0   | 0       | 0      | 0      | S | 0.0  | 0.0  | 0:00.32 | ksoftirqd/0    |
| 10   | root  | 20 | 0   | 0       | 0      | 0      | I | 0.0  | 0.0  | 0:00.50 | rcu_sched      |
| 11   | root  | 20 | 0   | 0       | 0      | 0      | I | 0.0  | 0.0  | 0:00.00 | rcu_bh         |
| 12   | root  | rt | 0   | 0       | 0      | 0      | S | 0.0  | 0.0  | 0:00.00 | migration/0    |
| 13   | root  | rt | 0   | 0       | 0      | 0      | S | 0.0  | 0.0  | 0:00.01 | watchdog/0     |
| 14   | root  | 20 | 0   | 0       | 0      | 0      | S | 0.0  | 0.0  | 0:00.00 | cpuhp/0        |
| 15   | root  | 20 | 0   | 0       | 0      | 0      | S | 0.0  | 0.0  | 0:00.00 | kdevtmpfs      |
| 16   | root  | 0  | -20 | 0       | 0      | 0      | I | 0.0  | 0.0  | 0:00.00 | netns          |
| 17   | root  | 20 | 0   | 0       | 0      | 0      | S | 0.0  | 0.0  | 0:00.00 | rcu_tasks_kth+ |

  
[1]+ Stopped top  
oscar@oscar-laptop:~$ bg  
[1]+ top &  
oscar@oscar-laptop:~$
```

上图中，终端显示了：

```
[1]+ top &
```

表示 top 命令被转到了后台。因为 bg 命令会把在后台暂停的进程重新唤醒，使之在后台重新运行。

不过，此时你的光标会跑到命令行提示符最前面。如果此时按下回车，会显示：

```
[1]+ Stopped top
```

```

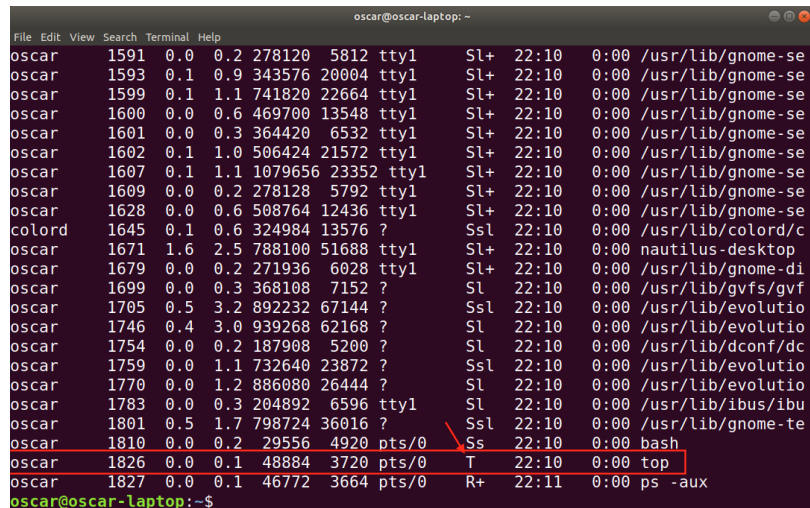
17 root      20   0      0      0      0 S  0.0  0.0   0:00.00 rcu_tasks_kth+
18 root      20   0      0      0      0 S  0.0  0.0   0:00.00 kauditd
[1]+  Stopped                  top
oscar@oscar-laptop:~$ bg
[1]+  top &
oscar@oscar-laptop:~$

[1]+  Stopped                  top
oscar@oscar-laptop:~$

```

我们用 ps 命令来查看一下进程信息：

```
ps -aux
```



```

oscar@oscar-laptop: ~
File Edit View Search Terminal Help
oscar 1591 0.0 0.2 278120 5812 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-se
oscar 1593 0.1 0.9 343576 20004 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-se
oscar 1599 0.1 1.1 741820 22664 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-se
oscar 1600 0.0 0.6 469700 13548 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-se
oscar 1601 0.0 0.3 364420 6532 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-se
oscar 1602 0.1 1.0 506424 21572 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-se
oscar 1607 0.1 1.1 1079656 23352 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-se
oscar 1609 0.0 0.2 278128 5792 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-se
oscar 1628 0.0 0.6 508764 12436 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-se
colord 1645 0.1 0.6 324984 13576 ? Ssl 22:10 0:00 /usr/lib/colord/c
oscar 1671 1.6 2.5 788100 51688 tty1 Sl+ 22:10 0:00 nautilus-desktop
oscar 1679 0.0 0.2 271936 6028 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-di
oscar 1699 0.0 0.3 368108 7152 ? Sl 22:10 0:00 /usr/lib/gvfs/gvf
oscar 1705 0.5 3.2 892232 67144 ? Ssl 22:10 0:00 /usr/lib/evolutio
oscar 1746 0.4 3.0 939268 62168 ? Sl 22:10 0:00 /usr/lib/evolutio
oscar 1754 0.0 0.2 187908 5200 ? Sl 22:10 0:00 /usr/lib/dconf/dc
oscar 1759 0.0 1.1 732640 23872 ? Ssl 22:10 0:00 /usr/lib/evolutio
oscar 1770 0.0 1.2 886080 26444 ? Sl 22:10 0:00 /usr/lib/evolutio
oscar 1783 0.0 0.3 204892 6596 tty1 Sl 22:10 0:00 /usr/lib/ibus/ibu
oscar 1801 0.5 1.7 798724 36016 ? Ssl 22:10 0:00 /usr/lib/gnome-te
oscar 1810 0.0 0.2 29556 4920 pts/0 Ss 22:10 0:00 bash
oscar 1826 0.0 0.1 48884 3720 pts/0 T 22:10 0:00 top
oscar 1827 0.0 0.1 46772 3664 pts/0 R+ 22:11 0:00 ps -aux
oscar@oscar-laptop:~$

```

在上图中可以看到，top 这个进程的进程号（PID）是 1826，状态是 T。

首先补充一些知识：

Linux 中，进程有 5 种状态：

1. 运行 (正在运行或在运行队列中等待)
2. 中断 (休眠中, 受阻, 在等待某个条件的形成或接受到信号)
3. 不可中断 (收到信号不唤醒和不可运行, 进程必须等待直到有中断发生)
4. 僵死 (进程已终止, 但进程描述符存在, 直到父进程使用 wait4() 系统调用后释放)
5. 停止 (进程收到 SIGSTOP, SIGSTP, SIGTIN, SIGTOU 信号后停止运行)

ps 命令标识进程的 5 种状态码如下：

1. D 不可中断 uninterruptible sleep (usually IO)
2. R 运行 runnable (on run queue)
3. S 中断 sleeping
4. T 停止 traced or stopped
5. Z 僵死 a defunct ("zombie") process

因此，我们的 top 进程的状态还是 T，也就是停止 (stopped) 的状态。很奇怪是吧？

我们用其他的命令来测试看看，我们测试 grep 命令。首先运行：


```
grep -r "log" / > grep_log 2>&1
```

上面这个命令的作用是：在根目录 / 下查找包含 log 的行，将标准输出和标准错误输出都重定向到 grep_log 文件中。

因此，虽然上述命令在运行，但终端中看不到任何信息。

我们用 Ctrl + Z 来暂停此进程，并将其转到后台。然后再运行 bg 命令，使其重新在后台运行。

```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ grep -r "log" / > grep_log 2>&1  
^Z  
[1]+  Stopped                  grep --color=auto -r "log" / > grep_log 2>&1  
oscar@oscar-laptop:~$ bg  
[1]+  grep --color=auto -r "log" / > grep_log 2>&1 &  
oscar@oscar-laptop:~$
```

为什么 bg 作用于暂停的 grep 命令后，没有像刚才 top 命令一样仍然显示 Stopped 呢？

我们再用 ps -aux 看一下：

```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar 1601 0.0 0.3 364420 6532 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-se  
oscar 1602 0.0 1.0 506424 21572 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-se  
oscar 1607 0.0 1.1 1079656 23352 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-se  
oscar 1609 0.0 0.2 278128 5792 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-se  
oscar 1628 0.0 0.6 508764 12436 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-se  
colord 1645 0.1 0.6 324984 13576 ? Ssl 22:10 0:00 /usr/lib/colord/c  
oscar 1671 1.0 2.5 788100 51688 tty1 Sl+ 22:10 0:00 nautilus-desktop  
oscar 1679 0.0 0.2 271936 6028 tty1 Sl+ 22:10 0:00 /usr/lib/gnome-di  
oscar 1699 0.0 0.3 368108 7152 ? Sl 22:10 0:00 /usr/lib/gvfs/gvf  
oscar 1705 0.3 3.2 892232 67144 ? Ssl 22:10 0:00 /usr/lib/evolutio  
oscar 1746 0.3 3.0 939268 62168 ? Sl 22:10 0:00 /usr/lib/evolutio  
oscar 1754 0.0 0.2 187908 5200 ? Sl 22:10 0:00 /usr/lib/dconf/dc  
oscar 1759 0.0 1.1 732640 23872 ? Ssl 22:10 0:00 /usr/lib/evolutio  
oscar 1770 0.0 1.2 886080 26444 ? Sl 22:10 0:00 /usr/lib/evolutio  
oscar 1783 0.0 0.3 204892 6596 tty1 Sl 22:10 0:00 /usr/lib/ibus/ibu  
oscar 1801 0.4 1.7 799172 36136 ? Ssl 22:10 0:00 /usr/lib/gnome-te  
oscar 1810 0.0 0.2 29556 4924 pts/0 Ss 22:10 0:00 bash  
oscar 1826 0.0 0.1 48884 3720 pts/0 T 22:10 0:00 top  
oscar 1828 0.2 1.0 664144 21808 tty1 Sl+ 22:11 0:00 update-notifier  
oscar 1830 8.7 5.4 1061160 111720 tty1 SLl+ 22:11 0:02 /usr/bin/gnome-so  
root 1847 0.3 1.0 555676 21600 ? Ssl 22:11 0:00 /usr/lib/fwupd/fw  
oscar 1911 18.4 0.1 22188 3116 pts/0 D 22:11 0:01 grep --color=auto  
oscar 1912 0.0 0.1 46772 3476 pts/0 R+ 22:11 0:00 ps -aux  
oscar@oscar-laptop:~$
```

可以看到，top 命令的状态是 T，也就是停止（Stopped）。而 grep 命令的状态则是 D，也就是不可中断的睡眠（但其实是在运行，等我们会看到）。

疑问：我也不太清楚为什么对普通的命令（例如 grep），bg 命令是起作用的，会将其转成后台运行。

但是对于 top 命令，bg 为什么不能将其转成后台运行，可能是因为 top 命令本身比较特殊吧。

也许是因为 top 命令是前台交互式命令，因此不能被置于后台运行。如有知道的，欢迎留言，我可以修改。谢谢

小结一下：

如果你原本想要使一个命令运行在后台，成为后台进程，但是忘记加 & 符号了。那么可以按下面的顺序使此进程转为后台运行：

- Ctrl + Z：使进程转为后台暂停。
- bg：使进程转为后台运行。

那你也也许要问：为什么不直接用 bg 命令一步到位呢？

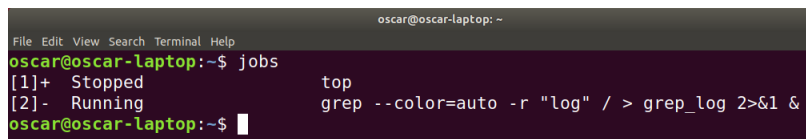
因为，如果不先用 `Ctrl + Z` 将此进程暂停，此进程就一直在前台运行，你没法在命令提示符后面输入啊。

jobs 命令：显示后台进程状态

这个命令很强大，毕竟和乔布斯老爷子（乔布斯的英文就是 `jobs`，全名是 `Steve Jobs`。`job` 是英语“工作”的意思，`jobs` 是复数形式）一样名字么。

`jobs` 命令的作用是显示当前终端里的后台进程状态。虽然我们可以用 `ps` 命令来查看进程状态，但是 `ps` 命令输出的进程列表太长了。

聪明如你一定想到了，我们可以用 `jobs` 命令来显示刚才那两个进程的状态：`top` 进程和 `grep` 进程。



```
oscar@oscar-laptop: ~
File Edit View Search Terminal Help
oscar@oscar-laptop:~$ jobs
[1]+  Stopped                  top
[2]-  Running                  grep --color=auto -r "log" / > grep_log 2>&1 &
oscar@oscar-laptop:~$
```

`jobs` 命令的输出共分三列，我们逐列来说明：

1. 显示后台进程标号：比如上例中 `top` 进程的标号是 1，`grep` 进程的标号是 2，如果还有其他后台进程，那么就会有 [3]，[4]等等。这个标号和 PID（进程号）是不一样的。这个标号只是显示当前终端下的后台进程的一个编号；
2. 显示后台进程状态：比如 `Stopped` 是“停止的”的意思，`Running` 是“运行的”的意思。还有其他状态；
3. 命令本身。

可以看到，我们的 `top` 进程确实是在后台暂停了，因为显示 `Stopped`。`grep` 进程在后台运行，因为显示 `Running`。

fg 命令：使进程转到前台

`fg` 是英语 `foreground` 的意思，表示“前台”。

与 `bg` 命令相反，`fg` 命令的作用是：使进程转为前台运行。

用法也很简单，和 `bg` 一样，如果不加参数，那么 `fg` 命令作用于最近的一个后台进程；如果加参数，如 `%2`，那么表示作用于本终端中第二个后台进程。

如果我们现在运行 `fg` 命令，就会把 `top` 命令转到前台运行。

```
fg
```

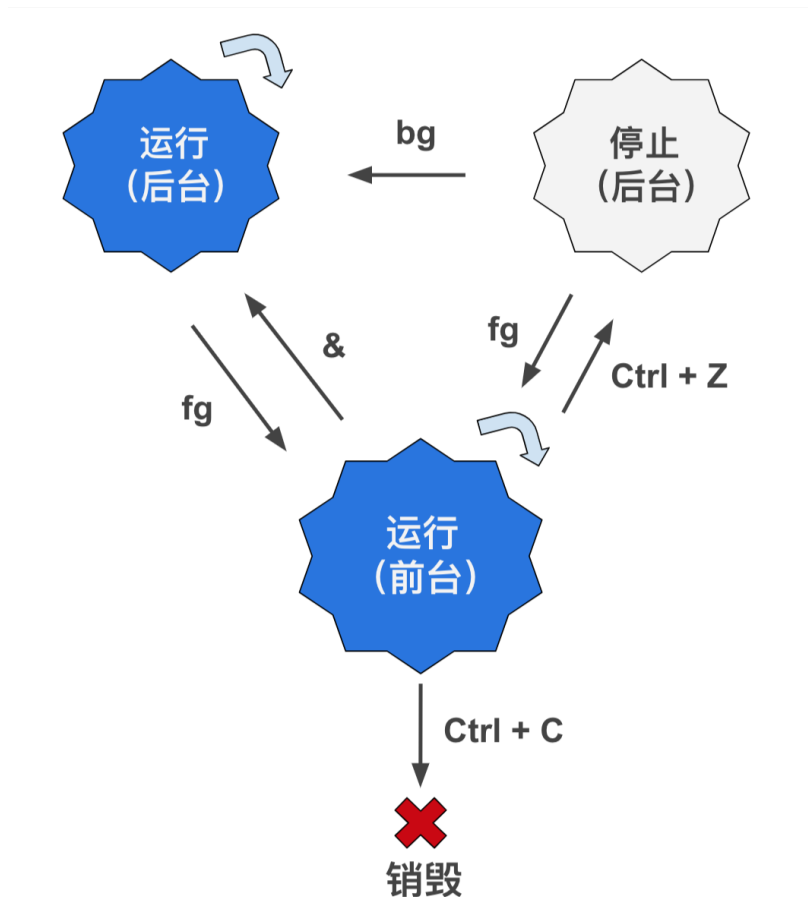
```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
KiB Mem : 2041088 total, 72684 free, 661488 used, 1306916 buff/cache  
KiB Swap: 969960 total, 969960 free, 0 used, 1168948 avail Mem  


| PID  | USER  | PR | NI  | VIRT    | RES    | SHR    | S | %CPU | %MEM | TIME+    | COMMAND        |
|------|-------|----|-----|---------|--------|--------|---|------|------|----------|----------------|
| 1911 | oscar | 20 | 0   | 27964   | 8956   | 2500   | R | 98.7 | 0.4  | 13:03.70 | grep           |
| 1152 | oscar | 20 | 0   | 2985188 | 265072 | 107048 | S | 0.3  | 13.0 | 0:06.06  | gnome-shell    |
| 1826 | oscar | 20 | 0   | 48884   | 3720   | 3100   | R | 0.3  | 0.2  | 0:00.07  | top            |
| 1    | root  | 20 | 0   | 159832  | 9024   | 6636   | S | 0.0  | 0.4  | 0:01.05  | systemd        |
| 2    | root  | 20 | 0   | 0       | 0      | 0      | S | 0.0  | 0.0  | 0:00.00  | kthreadd       |
| 3    | root  | 0  | -20 | 0       | 0      | 0      | I | 0.0  | 0.0  | 0:00.00  | rcu_gp         |
| 4    | root  | 0  | -20 | 0       | 0      | 0      | I | 0.0  | 0.0  | 0:00.00  | rcu_par_gp     |
| 6    | root  | 0  | -20 | 0       | 0      | 0      | I | 0.0  | 0.0  | 0:00.00  | kworker/0:0H-+ |
| 7    | root  | 20 | 0   | 0       | 0      | 0      | I | 0.0  | 0.0  | 0:00.07  | kworker/u2:0-+ |
| 8    | root  | 0  | -20 | 0       | 0      | 0      | I | 0.0  | 0.0  | 0:00.00  | mm_percpu_wq   |
| 9    | root  | 20 | 0   | 0       | 0      | 0      | S | 0.0  | 0.0  | 0:00.27  | ksoftirqd/0    |
| 10   | root  | 20 | 0   | 0       | 0      | 0      | I | 0.0  | 0.0  | 0:00.25  | rcu_sched      |
| 11   | root  | 20 | 0   | 0       | 0      | 0      | I | 0.0  | 0.0  | 0:00.00  | rcu_bh         |
| 12   | root  | rt | 0   | 0       | 0      | 0      | S | 0.0  | 0.0  | 0:00.00  | migration/0    |
| 13   | root  | rt | 0   | 0       | 0      | 0      | S | 0.0  | 0.0  | 0:00.00  | watchdog/0     |
| 14   | root  | 20 | 0   | 0       | 0      | 0      | S | 0.0  | 0.0  | 0:00.00  | cpuhp/0        |
| 15   | root  | 20 | 0   | 0       | 0      | 0      | S | 0.0  | 0.0  | 0:00.00  | kdevtmpfs      |

  
oscar@oscar-laptop:~$ jobs  
[2]+  Running                  grep --color=auto -r "log" / > grep_log 2>&1 &  
oscar@oscar-laptop:~$
```

上图中，top 进程被 fg 命令从后台 Stopped 的状态转到了前台运行。接着，我用 Ctrl + C 终止了 top 进程的运行。再次运行 jobs 命令，可以看到这会只有 grep 进程还在后台运行了，因为 top 进程已经被销毁了。

好了，讲了这么多知识点，是不是有点晕呢？没关系。我们用下面这个状态图来做个总结，应该就很清楚了：



解释一下上图：

1. 如果我们运行一个程序，默认情况下，它会成为一个前台运行的进程。我们可以按组合键 Ctrl + C 来销毁此进程。
2. 我们也可以使此进程在后台运行。假如运行程序时就用 & 放在命令最后，那么进程就会在后台运行。
3. 假如在进程运行起来后，按 Ctrl + Z，则进程会转到后台，并且停止。此时如果运行 bg 命令，则进程重新运行，并继续在后台。
4. fg 命令可以使进程转到前台，并且运行。

花点时间好好理解一下这个状态图。这个图很重要，几乎概括了后台前台进程切换的所有情况。

小结

1. 我们可以使程序在后台运行，成为后台进程。这样在当前终端中我们就可以做其他事了，而不必等待此进程运行结束。
2. 为了使一个程序在后台运行，可以在命令的最后加上 `&` 这个符号。但是，如果你关闭终端或退出登录，此后台进程还是会结束。为了将后台进程与本终端分离，可以使用 `nohup` 命令，使得进程不再受终端关闭或用户登出的影响。
3. 如果你运行了一个前台进程，但是想要将其转为后台运行进程。你可以先用 `Ctrl + Z` 组合键将其转为后台暂停，然后运行 `bg` 命令使其在后台重新运行。如果你要将一个后台命令（不管它是后台运行还是后台暂停）重新转为前台运行，只要用 `fg` 命令就可以了。

今天的课就到这里，一起加油吧！

← 26 进程操作和系统重启

28 多个终端和Terminator软件 →

精选留言 1

欢迎在这里发表留言，作者筛选后可公开显示

向往那片天空

学习了

👍 0 回复

2天前