使用 Linux 系统的开发者，很多人都有自己喜欢的系统命令，下面这个几个命令令是我平常用的比较多的，分享一下。

我不会教科书般的罗列每个指令的详细用法，只是把日常开发过程中的一些场景下，经常使用的命令常见用法进行演示。

希望对您有小小的帮助~~也非常欢迎各位大侠，在留言区分享您的私房命令。

No.1 grep

grep、awk、sed 这三个指令，作为 Linux 系统中文本处理的三大法宝，我最喜欢、最常用的就是 grep 指令，没有之一！

它的基本用法是：

```
grep [OPTIONS] PATTERN [FILE...]
grep [OPTIONS] [-e PATTERN]...  [-f FILE]...  [FILE...]
```

看起来有那么的选项，我最常用的是这 2 个场景：

1. 在一个文件或者文件夹中，查找指定的字符串：

```
grep -rni "pthread" *
```

> -r: 递归查找;
> -n：打印行号;
> -i: 不区分大小写;

2. 查看某个进程的相关信息，例如：进程 ID

```
$ ps -aux | grep bash
root  4681  0.0  0.1  24892  5912 pts/3    Ss   10:10   0:00 bash
root 18052  0.0  0.0  15968   960 pts/3    S+   13:38   0:00 grep --color=auto bash
```

可以看到，结果中出现了 grep 这个指令自身的进程信息，可以通过 -v 选项过滤掉它：

```
$ ps -aux | grep bash | grep -v grep
root  4681  0.0  0.1  24892  5912 pts/3    Ss   10:10   0:00 bash
```

最后，再结合 awk 命令，就可以把进程ID 4681 提取出来了：

```
$ ps -aux | grep bash | grep -v grep | awk  '{print $2}'
4681
```

在一些脚本工具中，这样的用法还是很常见的。

例如：在一些守护进程的启动脚本中，都会利用这条指令来判断：当前系统中是否已经有一个实例正在运行了。

No.2 q

看到这个指令，您一定会疑惑：仅仅一个字母 q，这是何方神圣?

Linux 系统中压根就没有这个命令！

是的，这个字母仅仅是一个 alias(别名)。

我有很强的强迫症，在终端窗口执行一条命令的时候，我经常会需要确认指令是否执行正确。

在 Linux 系统中，$? 用来表示最后命令的退出状态: 0 表示没有错误，其他表示有错误。

因此，在执行完一条命令后，可以执行下面的这条命令来确认：刚才执行的那条命令是否成功了。

```
echo $?
```

强迫症的问题是解决了，但是由于这条指令使用的太频繁了，需要敲那么多的字符，还要结合 shift 按键。

于是我就给它设置了一个 alias(别名)。

设置 alias 的方法估计都知道啊，就是在个人家目录下的 .bashrc 中修改。

我的 alias 设置如下：

```
alias ll='ls -lF'
alias la='ls -A'
alias l='ls -CF'
alias q='echo $?'
```

这样的话，每次执行完一条系统命令之后，随手敲一个字母 q 就可以检查执行结果了，省时省力！

No.3 pwd

可能有些人会奇：pwd 怎么会是常用命令呢？它的作用是打印当前路径，在命令行窗口中，路径是一直显示出来的啊！

没错，在默认的情况下，当前所处的路径信息，是直接显示出来的，如下：

```
root@ubuntu:~/OpenSource/linux-4.15/samples/watchdog$
```

但是这里有一个小小的问题：如果终端窗口的大小并不是全屏的，如果目录层次比较深，那么显示的路径信息就会特别的长，这样的话，本来就不太宽的终端窗口就显得很挤，输入命令的时候很可能要折返到下一行去。

于是，我就喜欢把这个显示的路径给它缩短：只显示最后一个文件目录，如下：

```
root@ubuntu:watchdog$
```

也就是把前面的 ~/OpenSource/linux-4.15/samples 路径信息都丢掉，这样的话，终端窗口中就有足够的空间来输入了。

如果某个时候，我想看一下当前目录的全路径，那么就执行一下 pwd 这个指令就可以了。

这就是我为什么经常使用 pwd 命令的原因。

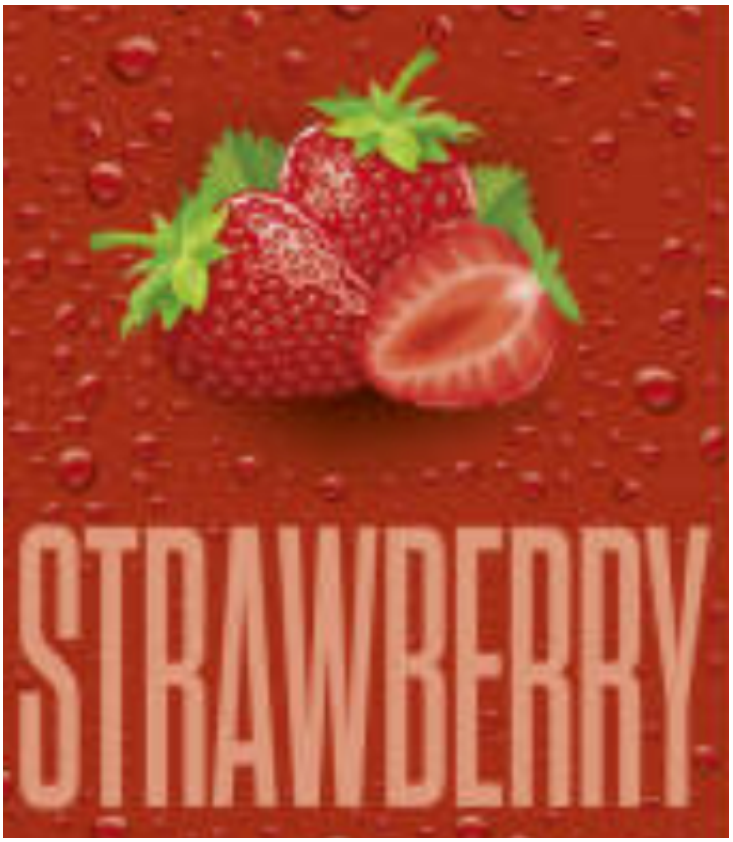
那么，应该怎么样来去掉显示路径中的目录信息呢?

还是修改家目录下的 .bashrc 文件：

```
if [ "$color_prompt" = yes ]; then
    PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\
[\033[01;34m\]\w\[\033[00m\]\$ '
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
fi
```

找到上面这几行内容，把最后面的 \w 修改成 \W 即可，也就是小写的 w 改成大写的 W。

如果你正在测试，请不要忘记使用 source .bashrc 或者 . .bashrc 命令来重新加载哦！



No.4 find

`find` 命令用来查找符合指定条件的文件。

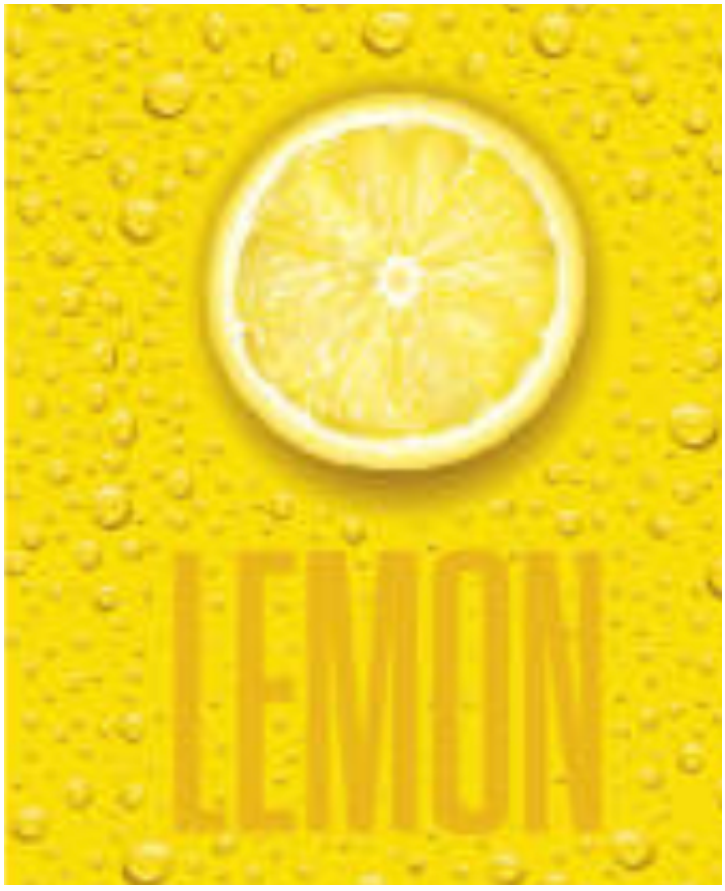
我最常用的场景就是：查找指定名称或类型的文件了。

特别是在写 `Makefile` 的时候，经常遇到找不到头文件的错误，于是就可以这样查找：

```
find ./ -name xxx.h
```

或者按照后缀名来查找文件：

```
find ./ -name *.txt
```



No.5 history

`history` 用来记录执行过的命令，如果您很少使用这条命令，那说明您的记忆力很好！

但是对于我这样忘性比较好的人来说，`history` 命令简直太有用了！

我在命令行窗口中测试某段代码，经常需要反复的做这样的排错过程：修改代码-编译-执行-查看结果。

如果编译指令比较长，我相信没有谁乐意一个字符一个字符的敲键盘，大部分是使用 `history` 列出最近使用的命令，然后复制、粘贴一下。

查看历史命令的时候，由于输出太多，可以结合 grep 指令，只显示我们感兴趣的命令记录，例如：

```
history | grep gcc
```

那么，结果中将只会显示带有 gcc 字眼的那些命令。

另外，还有一个小技巧：在不查看 history 的情况下，快速的输入之前执行过的某条指令(有一个前提条件：你必须能记住那条指令中刚开始的几个字符)。

比如，之前已经执行过这条指令：

```
gcc -m32 -Wl,--export-dynamic -o main main.c -ldl
```

几分钟之后，我想再次执行这条命令，可以这么做：

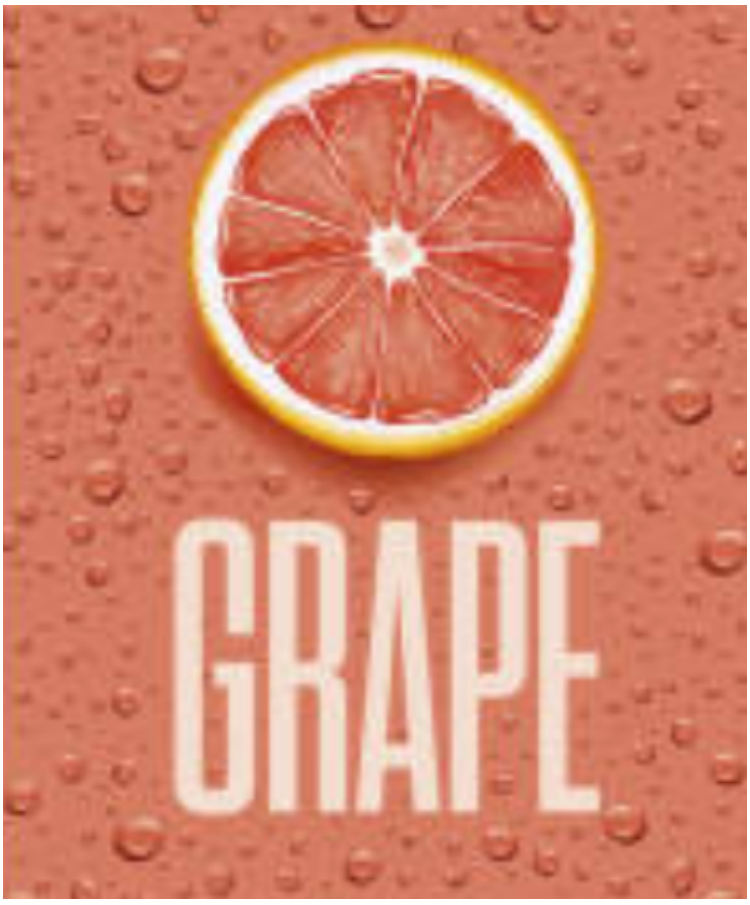
同时按下 control 和 r 这两个按键，此时输入光标处就会变成这样：

```
root@ubuntu:tmp$
(reverse-i-search)`':
```

这时，可以输入命令最前面的几个字母：gcc -m，此时终端就会到历史命令记录中去查找，输入的字符越多，匹配就越精确。

如果输入的字符，精准的匹配到了某个历史命令记录，它就立刻把这条命令完整的显示出来。

这个小技巧真的很好用，推荐您试一下！



No.6 od

od 命令用来输出给定文件的内容。

输入文件内容的指令有很多了：cat、head、tail等等。但是 od 命令主要用来查看文件的二进制编码，显示的时候可以以指定的进制进行显示。

在之前的一篇拆解 ELF 格式的文章中：Linux系统中编译、链接的基石-ELF文件：扒开它的层层外衣，从字节码的粒度来探索，我就大量的使用了 od 指令，在一个 ELF 格式的文件中，从任意地址开始、读取任意长度的字节码。

例如下面这条指令：读取 main 文件中最开始的 52 个字节的内容：

```
od –Ax –t x1 –N 52 main
```

main 是 Linux 系统中的可执行程序，当然也就是 ELF 格式了。

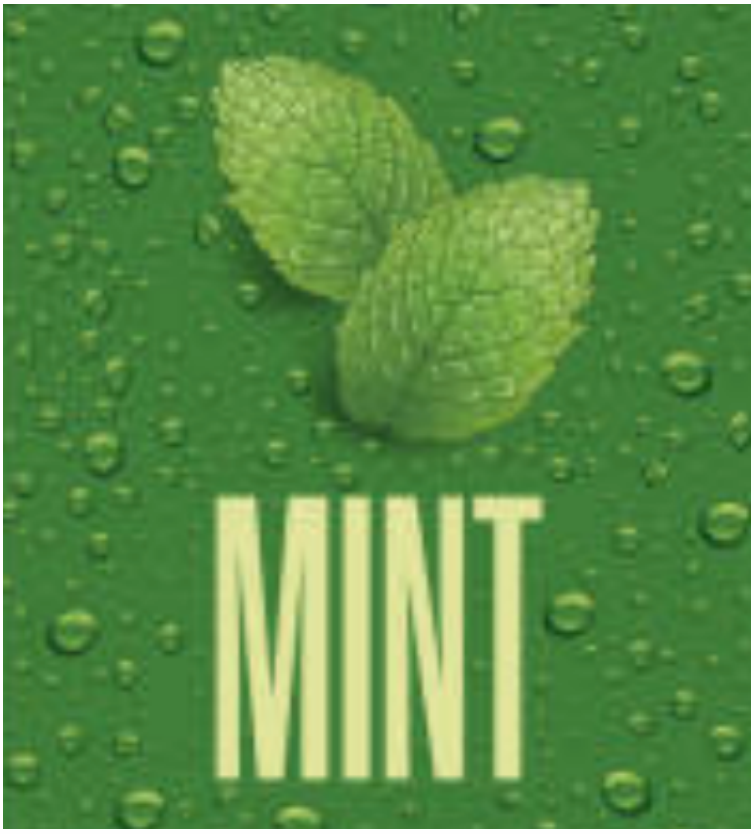
od 指令中使用到了下面这几个选项：

> -Ax: 显示地址的时候，用十六进制来表示。如果使用 -Ad，意思就是用十进制来显示地址;
>
> -t -x1: 显示字节码内容的时候，使用十六进制(x)，每次显示一个字节(1);
>
> -N 52：只需要读取 52 个字节;

```
000000 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
000010 02 00 03 00 01 00 00 00 70 84 04 08 34 00 00 00
000020 f8 17 00 00 00 00 00 00 34 00 20 00 09 00 28 00
000030 1f 00 1c 00
```

可以看出 main 文件最开始的四个字节：7f 是 ELF 文件的魔数，45 4c 46 是 "ELF" 3个字母。

因此，使用 od 命令来分析二进制文件的内容，还是很有威力的！



No.7 for

for 这个命令，常常出现在脚本文件中，用来处理循环的情况，比如：遍历文件、计数，例如：

```
#!/bin/bash

for file in /tmp/*;
do
echo $file;
done
```

我在使用 for 的时候，最常用的场景是给很多相同后缀的文件，按顺序进行重命名：

```
i=0;for x in *.mp4; do n=$(printf "%02d" "$i"); mv $x $n.mp4; let i=i+1; done
```

这里是按照纯数字来重命名的，也可以根据需要加上前缀等字符串。

这里还有一个小问题需要注意一下：如果文件名中存在空格，mv 指令就会提示错误：

```
mv: target 'xxx' is not a directory
```

解决方法是：在终端窗口中，先执行一下这个命令：

```
IFS='

'
```

然后，再执行批量重命名命令，就不会出现错误了！

当然，更好的方式是，把这几个命令写成一个脚本文件，实现对任意类型的文件进行批量重命名功能，然后放在自己的私有 bin 目录下，随取随用。

别担心，我已经帮你写好了，如下所示(file_rename.sh)：

```
#!/bin/bash

if [ $# -eq 0 ];then
sufix=mp4
else
sufix=$1
fi

IFS='

'

i=0;for x in *.$sufix; do n=$(printf "%02d" "$i"); mv $x $n.$sufix; let i=i+1; done
```

只要执行 ./file_rename.sh，就会默认把当前目录下所有 mp4 文件进行重命名。

如果是其他类型的文件，那就传递一个参数进去。

比如：如果要批量对 png 格式的图片进行重命名，那就执行 ./file_rename.sh png，最后的 png 是传入的参数，对应于脚本文件中的 $1 变量。

------ End ------

期待您在留言区分享自己的私房指令，任何技术问题可以探讨！

推荐阅读

【1】C语言指针-从底层原理到花式技巧，用图文和代码帮你讲解透彻
【2】一步步分析-如何用C实现面向对象编程
【3】原来gdb的底层调试原理这么简单
【4】内联汇编很可怕吗？看完这篇文章，终结它！
【5】都说软件架构要分层、分模块，具体应该怎么做

星标公众号，能更快找到我！

C/C++、物联网、嵌入式、Lua语言

Linux 操作系统、应用程序开发设计

扫码关注公众号

道哥 个人微信

喜欢请分享，满意点个赞，最后点在看。