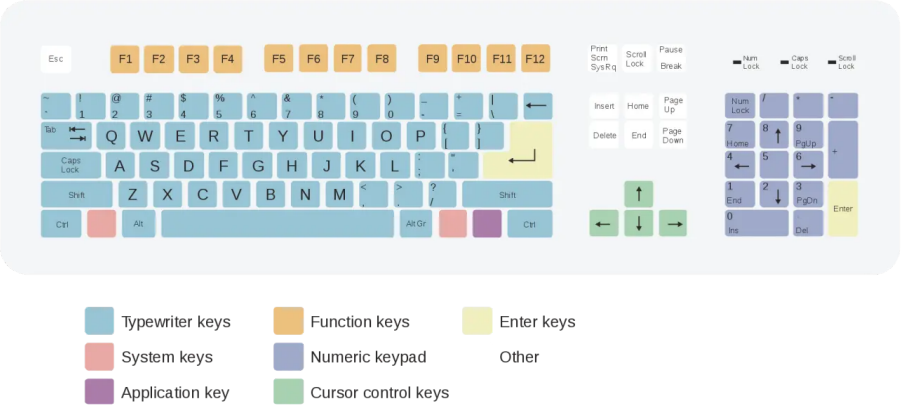
一些常见的解决方案（极不推荐，有缺陷，可跳过）

（具体产生原因的推测会在下篇详述）

Emacs需要用高频使用Ctrl键，Vim需要高频使用Esc键。这两个按键在ANSI机械布局的键盘上一个在左上角，一个在左下角，都是极其反人类的位置。



IBM/Windows键盘（美式布局）
QWERTY布局，System key为Win键，在Linux下常称为“Super”键或“Mod4”键

为了拯救小拇指，常用的解决方案有：

1. Emacs用户：

   软件：交换LeftCtrl和leftAlt，或交换LeftCtrl和CapsLock，包括Evil-mode在内的各种插件

   硬件：脚踏板，HHKB或其他的可编程键盘，用手掌按（机械键盘还行，笔记本键盘就拉倒吧），以及一些其他的奇妙方法……

2. Vim用户：

   软件：交换Esc和CapsLock，自己映射一堆快捷键

在macOS上使用Emacs和Vim的用户可以使用macOS自带的修改方法，或者买日版Macbook或键盘。

在Linux上除了插件和硬件方法之外的就是修改键盘映射了，常见且常用的操作方法有：

1. X11：使用xmodmap修改（最常见）
2. X11：使用setxkbmap或者localectl提供的各种option来修改。（上一章内容）
3. TTY：修改/usr/share/kbd/keymaps/i386/qwerty/us.map.gz文件中keycode的定义。（上一章内容）

这三种方法都有自己的局限性，因为它们都工作在**应用软件层**。

1. 首先他们都不能在X11和TTY下通用。（我们说过X11和TTY的有各自的转译软件）
2. X11提供的这两种方法，都有这样那样的缺陷。包括个别选项不起作用注[1]，Esc和CapsLock交换之后大写锁定灯会错乱注[2]，并且它们对于**带图形界面**的VirtualBox或者KVM虚拟机来说有一个致命缺陷（VMware未测试）注[3]。

注[1]：交换LeftCtrl和LeftAlt不起作用（这里不知道是不是我的姿势有问题）。

注[2]：按照正常逻辑，交换这两个键之后按下Esc键会切换大写锁定，但是按这两个键大写锁定灯都会亮。

注[3]：主机使用这些方法之一**交换Esc和CapsLock**。虚拟机内捕获键盘后，按下Esc会**同时产生这两个键的作用**。比如Win10重命名一个文件，要修改大写锁定，按了一个键会**既修改大写锁定，也退出重命名的输入框**。Vbox和KVM实测都会这样，VMware未测试。

个人**推测**是虚拟机软件要运行在X11环境下，但是也要和内核交互，所以会同时接收内核和X11传来的按键事件，所以一个键触发了两个操作。说推测是因为我没看过这些虚拟化软件的源码，但根据具体行为来看，这个推测为真的可能性接近99%，在后面会默认这个猜测为真。

我们使用一个方便通用，还不会有BUG的方法

直接修改scancode -> keycode映射表

回顾一下，上一章中介绍了Linux中一个按键从按下到得到最终效果需要三次处理：

KEYBOARD --> scancode --> keycode --> keysymbol

①键盘电路 ②内核处理 ③软件设置

①键盘硬件把你的按键行为转换成计算机能识别的 scancode

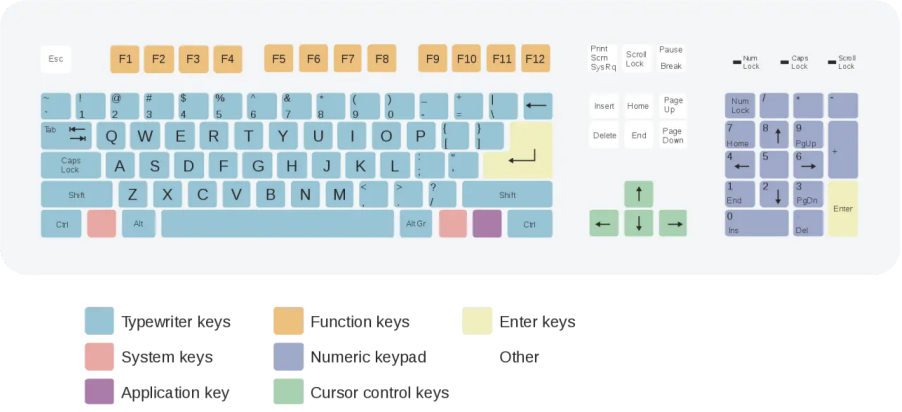
②系统内核把scancode转换成TTY或X11需要使用的 keycode

③TTY或X11的设置把keycode转换成具体的按键目的 keysymbol

厂商制造的键盘，要根据标准设置好每个物理位置的按键发出的scancode。操作系统根据最常见和最通用的功能布局：QWERTY，把scancode和keycode对应起来。（而且由于极其常见且通用，所以**US-QWERTY**也是通常情况的默认功能布局）

我们要达到自己的需求，交换LeftCtrl和leftAlt、或交换LeftCtrl和CapsLock、或交换Esc和CapsLock等功能，其实只需要把内置的scancode->keycode mapping修改成我们需要的样子就可以了。

并且因为这个修改对内核以上的软件透明，所以不会对上层应用的按键处理逻辑产生任何的额外影响。也就不会产生那些奇奇怪怪的问题，这正是我们所需要的。



IBM/Windows键盘（美式布局）
QWERTY布局，System key为Win键，在Linux下常称为“Super”键或“Mod4”键

修改原理简述：默认的scancode->keycode mapping由hwdb维护，hwdb除此之外还包括并维护了其他各类硬件的信息。详见/usr/lib/udev/hwdb.d/目录及man Page hwdb(7)-ZH hwdb(7)-EN

- 所需工具：

evtest（查看某个按键的具体scancode）

udevadm（更新自定义配置，查看配置内容，集成在systemd中，但是部分不使用systemd的发行版也会有udevadm的包提供，详见各发行版wiki）

任何文本编辑器（编辑你的自定义配置……）

- 所需权限：

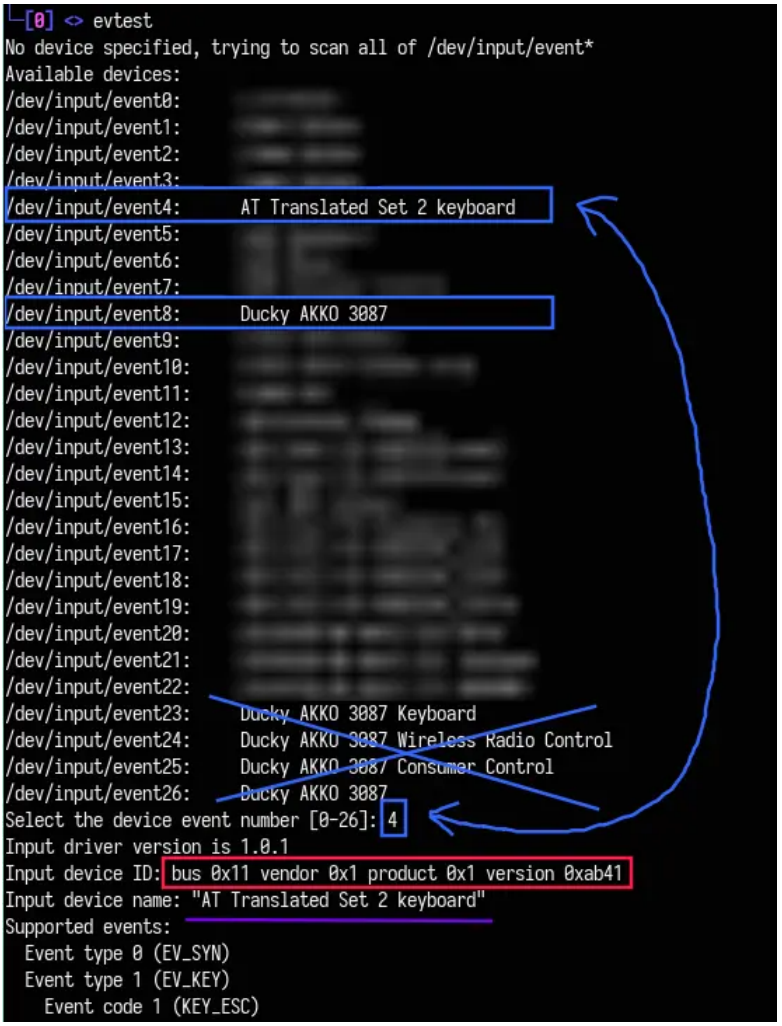
root（建议全程在root环境操作，**不要使用sudo**，可能会一些奇妙的问题）

1.首先获得你需要特定键盘上的某个键的scancode：

见Archwiki - Keyboard input #USE evtest

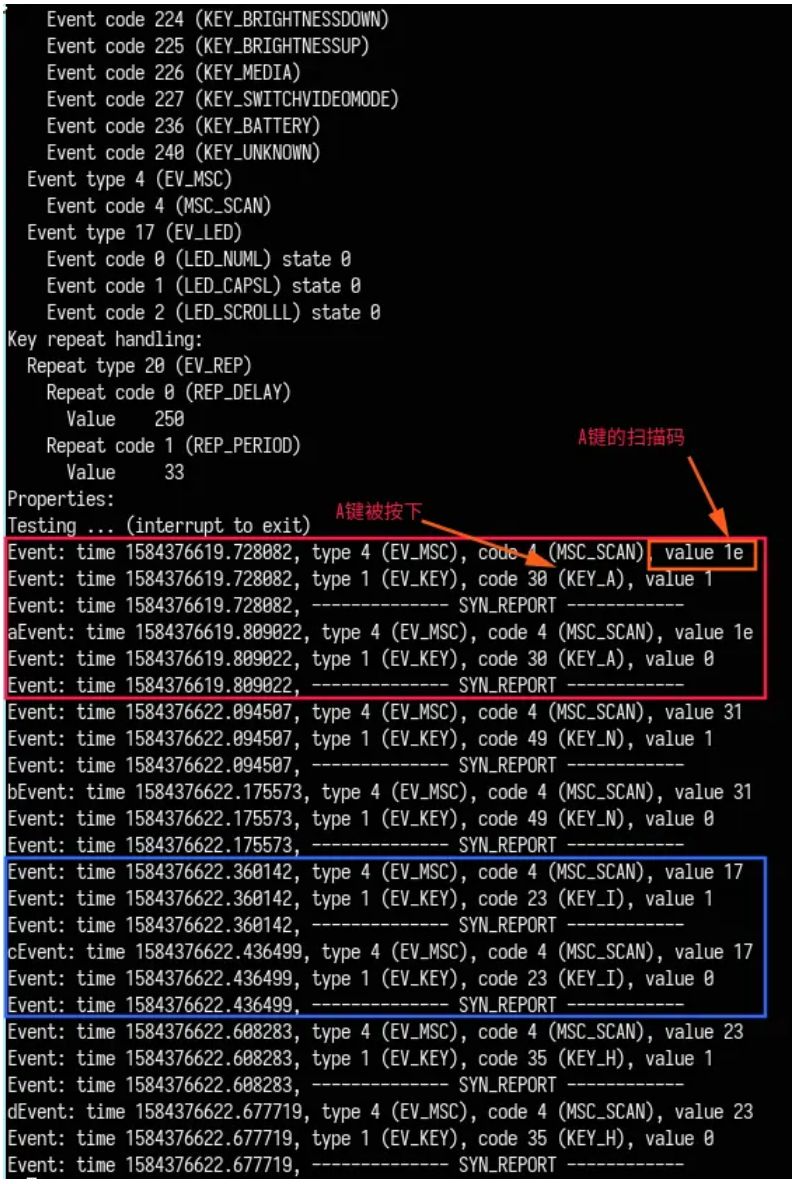
切换到root身份（非sudo）运行evtest，你可以看到像下图一样的输出

- /dev/input/event*就是输入设备，右边会跟着每个输入设备的名字，像蓝色框中的AT Translated Set 2 keyboard这样的一般是笔记本电脑自带的键盘，Ducky AKKO 3087这样的一般就是USB外接键盘。
- 注意如果是USB键盘，同时会显示有多个输入设备。比如我的Ducky AKKO 3087就显示了多个相关设备（event8、event23～event26），它们是由于USB键盘使用了一个叫**HID**的规范而产生的。我们只关心第一个，也就是/dev/input/event8，其他的不需要理会。
- **HID详见cv1480188-by元圈七**



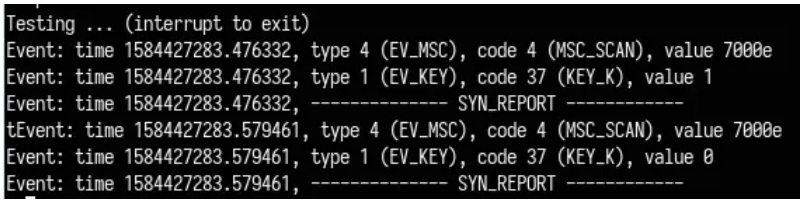
evtest输出图一

- 输入一个数字来选择你的键盘设备，一次只能获取一个键盘的扫描码，如果你有多个键盘需要多次运行evtest。这里输入了4来获取笔记本内置键盘的扫描码。**同理，再运行一次evtest并输入8来获取USB键盘上某个键的扫描码。**
- 在你选择的设备和你预期的键盘设备匹配后，**屏幕上的输出会直接滚动到下图的样子。如果你选错了就不会下图中红框和蓝框中这样的输出内容，请重新选择输入设备的设备号。**



evtest输出图二。这里说的A键指的是键盘上标了A的物理按键

- 图二红框中的是键盘上的A键被按下和释放的过程。扫描码为1e。红框中一共6行，第二行、第五行的value 1和value 0就代表键被按下和释放。蓝框中表示键盘上I键按下和释放的过程。
- **USB键盘同理，不过扫描码应该是一个5位16进制数。**

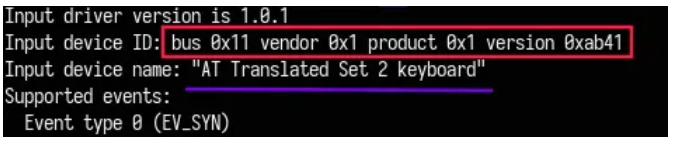


USB键盘的扫描码

- **按下你所要修改的几个按键，记录下它们的扫描码。明确这些扫描码所对应的物理位置。不要搞混，得到你需要的信息之后Ctrl-C结束。**

2.确认你要修改的输入设备的设备信息

- 上一步得到了几个特殊按键的scancode。现在把终端上的输出往回翻，找到你刚刚输入数字选择键盘设备的地方。**（就是evtest输出图一的那个部分）**



键盘输入设备的设备号和设备名

- **记录下红框中的内容或紫色划线的内容。bus是4位16进制的总线id，对于USB键盘通常显示0x3。vendor、product、version：分别是4位的16进制供应商id，产品id和版本id。所有的16进制数需要补0到4位。紫色划线内容就是设备名。**

3.编写自定义的scancode->keycode规则

- scancode->keycode的规则编写在**硬件数据库**中，分为两部分：

  硬件厂商和操作系统共同维护的**硬件数据库**，包括了可查看的具体数据库内容，位于/usr/lib/udev/hwdb.d/目录下，键盘部分存放在60-keyboard.hwdb中，这些都是文本文件，**极其不建议修改这些hwdb文件或在该目录下新建自己的规则**。系统运行时使用编译好的二进制文件/usr/lib/udev/hwdb.bin。

  本机管理员维护的**自定义硬件数据库**，位于/etc/udev/hwdb.d/目录，目前你这里应该还是空的。系统运行时使用编译好的二进制文件/etc/udev/hwdb.bin。

- **重要内容！！！**

  /usr/lib/udev/hwdb.d/目录下规则为默认规则，/etc/udev/hwdb.d/目录下对设备定义的规则如果和默认规则有冲突，会覆盖默认规则。

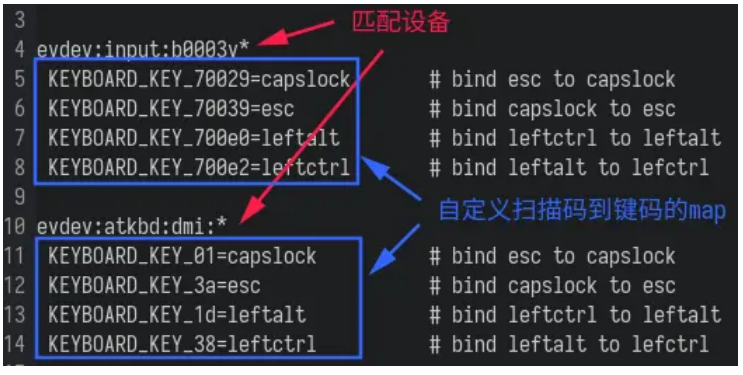
  系统数据库中的文件和自定义硬件数据库中的文件，命名都应该以数字开头，数字大小代表加载顺序，小的先加载，大的后加载。如果不同文件中对同一个设备的规则做了定义，后加载的（数字大的文件定义）会覆盖先加载的（数字小的文件定义）。

- 在/etc/udev/hwdb.d/下**以root身份**新建一个文本文件99-personal-kbd.hwdb。

  具体文件名需要遵守“<num>-<word>.hwdb”规则，即**“数字-单词.hwdb”**，如果这里本来就是空的，写成01-personal-kbd.hwdb或许也没什么问题。

- **文件格式：**

  “#”为注释，其他如图所示



99-personal-kbd.hwdb

本图表示在我的机器上交换Esc和Capslock、LeftAlt和LeftCtrl，并对所有的USB键盘和内置键盘生效。

经过测试，使用USB接收器的2.4Ghz无线键盘也是可以完美运行的（毕竟接收器是接在USB总线上的）。我手边没有蓝牙键盘，不清楚可行与否（理论上也是可以的，因为蓝牙也是接在USB总线上的）。欢迎在评论区报告问题。

- ## 多把USB键盘，每一把都需要有不同设置的用户：

- **精确匹配USB键盘**：

  evdev:input:b<bus_id>v<vendor_id>p<product_id>v<version_id>　　(根据id精确匹配)

  evdev:name:<input_device_name>:dmi:bvn*:bvr*:bd*:svn<vendor>:pn*　(根据设备名匹配)

  其中的<bus_id>、<vendor_id>、<product_id>、<vension_id>需要替换成刚刚记录的设备信息。<vendor>需要另一个软件包evemu提供的工具evemu-describe得到。注意**不够4位的数需要补“0”到4位，16进制数的a～f需要小写**，例如：evdev:input:b0003v0C45p7638*

  这是我的Ducky AKKO 3087 USB键盘，最后面使用*通配就可以了，加不加version_id都可以，只要能匹配到目标设备就可以了。<input_device_name>需要替换成设备名。

- **精确匹配笔记本电脑内置的AT键盘**：

  evdev:atkbd:dmi:bvn*:bvr*:bd*:svn<vendor>:pn<product>:pvr*　（根据id精确匹配）

  同样需要使用evemu-describe工具得到<vendor>和<product>。

- ## 一般用户：

- **匹配USB键盘：**

  evdev:input:b0003*　（匹配所有USB总线上的input类型的设备）

  虽然这种方法不止可以匹配到键盘，但是事实上其他设备也不会产生和键盘一样的扫描码输入KEYBOARD_KEY_<scancode>，所以一点问题也没有。

- **匹配笔记本电脑内置键盘：**

  evdev:atkdb:dim*　（匹配所有dmi总线上的at键盘）

  一般情况笔记本电脑也只有一把内置键盘，所以更不可能出问题。

- ## 自定义scancode->keycode mapping

- **KEYCODE_KEY_<scancode>=<keycode>**

  <scancode>就是最开始记录下来的那个目标物理按键的扫描码。<keycode>是键码，需要小写。可用的键码值见该表Keymap-list

- ## 保存退出

- ## 更新数据库内容

  就是把自定义的内容编译到hwdb.bin中，让自定义内容开始生效。

  以root身份运行：

  # udevadm hwdb --update

```
# udevadm trigger
```

第一条命令是重新编译二进制的数据库内容，第二条命令表示立刻重新触发所有输入设备，让更改立刻生效。

注意：这只能添加或修改数据库内容，如果你删除了某个规则，直到你重启原来的内容都会留在内核当中，不会失效。

- **如果你的操作没有问题，没有字符输入错误，那现在你的预期目标应该已经达成了。**

总结以及一些技巧

之前在群里教另一个小伙伴用修改hwdb的方法修改键映射。他需要使用Colemak键盘映射。但是并没有使用上一章结尾所说的方法去更改键盘的功能布局。而是直接在hwdb中硬编码了Colemak的scancode->keycode mapping，

他的操作给我提供了一个思路：如果你想使用Dvorak或Colemak映射。但是其他习惯于使用QWERTY映射的人需要在你的电脑上进行操作。由于协作问题的顾虑，你不能够随意更改自己的使用习惯。

那这个时候，实际上你可以在hwdb数据库中对自己私人使用的键盘进行硬编码。其他的键盘就用默认的mapping，当别人需要使用你的电脑，让TA带着TA自己的键盘来（狗头）

或者找某宝，最便宜的键盘20块，二手的可能更便宜。自己的键盘使用自己的映射，只用对这一个键盘硬编码成常规的QWERTY映射，反正是给别人用的，而且也是只用那么一会，不需要多好的键盘（狗头）

总之在Linux上，可自定义的地方太多了。多找文档，多搜索，总能有解决办法。当有一天你发现没有地方可以找了，只能自己写程序解决，那么你就已经是大神了。

参考内容：

Archwiki - Keyboard input

Archwiki - Map scancodes to keycodes