如何写作科技文档

科技文档相对特殊，把它写得通俗易懂而又不失简洁，几乎是一件不可能完成的任务。

尽管如此，通过有意识的使用一些技巧，还是能够尽可能的接近这个目标。 我这里有一些个人的经验，请看下面的三种表述:

- *第一个例子*[1]

  ```
  就像前面提到的，所有USE标记都声明在USE变量里面。为了让用户能方便地查找和选择USE标记，我们提供了一份默认的USE设定。这
  ```

- *第二个例子*

  ```
  /etc/make.profile/目录是一个符号链接，里面包含一些make.defaults文件，实际指向的是以下文件，可以在这些文件中设置
  ```

- *第三个例子*

  ```
  /etc/make.profile/目录是一个符号链接，里面包含一些make.defaults文件，放置开发者设置的USE标记：
  ```

目的明确. 手把手的操作步骤与宏观的概念描述、面向新手与面向老手，肯定会有很大的不同。要根据需要决定你的方向

第一个例子是给忠实用户使用的权威官方文档，后面例子中是向新手介绍的简要说明，目的不一样，所以介绍的详细程度不同

简明清晰. 在有些时候，要改变一下思路，从另一个角度叙述(在翻译文档时，这种技巧比较有用)。

第二个例子是最常见的介绍方式，它暴露了过于复杂的细节：实际指向、不可更改，而没有说明关键细节：开发者设置，还有一些冗余信息：可以在这些文件中设置

第三个例子中，在介绍了USE的概念之后，说明这些USE是开发者设置的，就可以暗示很多事情。如果说"系统默认、不可更改……"你最好说明为什么，因为"不可更改"这种声音，对于Linux用户来说无疑有些刺耳……

控制复杂度. 描述的复杂度必须控制，不要假定用户对Linux很熟悉……如果所有的文档都使用这样的假定，那么大多数用户很难熟悉Linux

在分析了自己的目的之后，你要有一个取舍。你要确定读者是否需要明白什么是"profile"

针对"profile"，如果不深入解释，有点虎头蛇尾，而科技文档的一个很重要的特点是"完整"；如果解释，则增加了一个不必要的分支

不要过度隐喻. 对于软件，只要不是在源代码的层面上描述，都算作隐喻。只是有的隐喻不明显，如启动、运行(它们的喻体是符号化的"系统")；而有些隐喻则比较形象，例如电视机、遥控器

如果不是难以描述的概念，不要用隐喻

如果确实需要使用形象的隐喻，特别是在短时间内多次使用，请确保喻体的相关性。电视机:系统、遥控器:终端 这样的比喻容易理解，而电视机:系统、手机:终端 的隐喻，多少令人费解

限制术语的使用. 术语本身拥有需要解释的特点。 像第一个例子中的"并集"，这是集合中的术语，并不是每个读者都熟悉集合的概念；而"并集"指多个集合相加，熟悉集合概念的读者，可能会深究被加在一起的集合……如果不引入这个术语也可以解释清楚，那么不要引入它

基础概念. 不要认为介绍基础概念是吃力不讨好的工作，尽管读者可以从操作步骤中得出基础概念

既然大多数的描述都是隐喻，在描述基础概念的时候，不要顾忌使用隐喻。读者得到一个基本概念之后，可以更容易的理解接下来的描述。("解释学之弧")

雕琢实例. 不要试图通过一个实例阐释许多个概念，不要随手抓一个什么东西当作实例。应该构建最简化的实例，每个实例只讲一件事(K&R 《The C Programming Language》)

充分利用文档工具的特性. 脚注，calloutlist

多数文档工具都可以使用脚注，使用脚注作简短的说明，可以确保在不增加分支的情况下，对概念作必要的说明

calloutlist用来作密集注释。虽然你可以把这部分内容分成多段描述，但是使用calloutlist，可以清晰的列出主干，更容易阅读[2]

厚积薄发。控制表达的冲动。很多文档作者都容易犯这样的错误，旁征博引、滔滔不绝……(《鸟哥私房菜》)可能你知道的确实挺详细，但请记住，你不是自说自话，而是写给读者。请考虑一下读者希望看到什么内容，而不是你想写什么内容

---

[1] 本例来自**gentoo中文手册**

[2] 另一方面，只有 DocBook 支持 calloutlist，这意味着，如果从 DocBook 转换到其它格式，calloutlist 将会制造一些麻烦
在代码环境中使用的注释，很大程度上可以替代 calloutlist的作用