怎样尊重一个程序员

得知一位久违的同学来到了旧金山湾区，然而我见到他时，这人正处于一生中最痛苦的时期。他告诉我，自己任职的公司在他加入之前和之后，判若两人。录取的时候公司对他说，我们对你在实习期间的表现和学术背景非常满意，你不用面试，甚至不用毕业拿学位，直接就可以加入我们公司成为正式员工。然而短短一年后的今天，这位同学已经完全感觉不到公司对自己技能的尊重。Manager 让他做一些乱七八糟没技术含量的事情，还抱怨说他做事太慢，并且在他的 evaluation 上很是写了一笔。在人格尊严和工作安全感的双重打击之下，这位同学压力非常大，周末经常偷偷地加班，仍然无法让 manager 满意。

我很了解这位同学的能力，在任何一流公司任职，肯定是绰绰有余了。他的名字我当然保密，然而他所任职的公司因为太过嚣张，我不得不直接指出来——这就是被很多人向往得像天堂一样的地方，Google。这位同学所描述的遭遇，跟我几年前在 Google 的实习经历如出一辙。我仍然记得，Google 的队友在旁边看着我用 Emacs，用小学老师似的口气对我说：“按 ctrl-k！”我仍然记得，在提交队友完全无法写出来的高质量代码时，被指责和嘲笑不会用 Perforce。我仍然记得，吃饭时同事们对所谓“Google 牛人”眉飞色舞的艳羡。我仍然记得，最后我一个人做出整个团队做梦都做不出来的项目的时候，有人发出沉闷的咆哮：“快——写——测——试！”……

我的这位同学也算得上本领域顶尖的专家了。如此的践踏一个专家的价值，用肤浅的标准来评判和对待他们，Google 并不是唯一一个这样的公司。我之前任职的好几个公司，或多或少都存在类似的问题。很多时候也不一定是公司管理层无端施加压力，而是程序员之间互斗的厉害，互相评判，伤害自尊。从最近 Linus Torvalds 在演讲现场公然对观众无理，你可以看出这种只关心技术，不尊重人的思潮，在程序员的社区里是非常普遍的。

后来我发现，并不是程序员故意想要藐视对方或者互相攻击，而是他们真的不明白什么叫做“尊重”，他们不知道如何说话才可以不刺伤别人。尊重他人其实是一个“技术问题”，并不是有心就可以做到的。由于这个原因，我想从心理和技术角度出发，指出这类不尊重人现象的起源，同时提出几点建议，告诉人们如何真正的尊重一个程序员。我希望这些建议对公司的管理层有借鉴意义，也希望它们能给与正在经受同样痛苦的程序员们一些精神上的鼓励。

为了建设一个互相尊重的公司文化，我认为应该注意以下几个要点。

认识和承认技术领域的历史遗留糟粕

很多不尊重人现象的起源，都是因为某些人偏执的相信某种技术就是世界上最好的，每个人都必须知道这些东西，否则他就不是一个合格的程序员。

这种现象在Unix（Linux）的世界尤为普遍。Unix 系统的鼓吹者们（我曾经是其中之一）喜欢到处布道，告诉你其它系统的设计有多蠢，你应该遵从 Unix 的“哲学”。他们认为 Unix 就是终极的操作系统，然而事实却是，Unix 是一个设计非常糟糕的系统。它似乎故意被设计为难学难用，容易犯错，却美其名曰“强大”，“灵活”。

眼界开阔一点的程序员都知道，Unix 的设计者其实基本不懂设计，他们并不是世界上最好的程序员，却有一点做得很成功，那就是他们很会制造宗教，煽动人们的盲从心理。Unix 设计者把自己的设计失误推在用户身上，让用户觉得学不会或者搞错了都是自己的错。

如果你对计算机科学理解到一定程度，就会发现我们其实仍然生活在计算机的石器时代。特别是软件系统，建立在一堆历史遗留的糟糕设计之上。各种蹩脚脑残的操作系统（比如 Unix，Linux），程序语言（比如 C++，JavaScript，PHP，Go），数据库，编辑器，版本控制工具，…… 时常困扰着我们，这就是为什么你需要那么多的所谓“经验”和“知识”。然而，很多IT公司不喜欢承认这一点，他们一向以来的作风是“一切都是程序员的错！”，“作为程序员，你应该知道这些！”这就造成了一种“皇帝的新装现象”——大家都不喜欢用一些设计恶劣的工具，却都怕别人嘲笑或者怀疑自己的能力，所以总是喜欢显示自己“会用”，“能学”，而没有人敢说它难用，敢指出设计者的失误。

我这个人呢，就是这种“黑客文化”的一个反例。我所受到的多元化教育，让我从这些偏激盲从，教条主义的心理里面跳了出来。每当有人因为不会某种工具或者语言来请教我时，我总是很轻松的调侃这工具的设计者，然后告诉他，你没理由知道这些破玩意儿，但其实它就是这么回事。然后我一针见血的告诉他这东西怎么回事，怎么用，是哪些设计缺陷导致了我们现在的诡异用法…… 我觉得所有的IT从业人员对于这些工具，都应该是这样的调侃态度。只有这样，软件行业才会得到实质性的进步，而不是被一些自虐的设计所困扰，造成思维枷锁。

总之，这是一个非常重要的“态度问题”。虽然在现阶段，我们有必要知道如何绕过一些蹩脚的工具，利用它们来完成自己的任务。然而在此同时，我们必须正视和承认这些工具的恶劣本质，而不能拿它们当教条，把什么事都怪罪于程序员。只有分清工具设计者的失误和程序员自己的失误，不把工具的设计失误怪罪于程序员，我们才能有效地尊重程序员们的智商，鼓励他们做出简单，优雅，完善的产品。

分清精髓知识和表面知识，不要太拿经验当回事

在任何领域，都只有少数知识是精髓的，另外大部分都是表面的，肤浅的，是从精髓知识衍生出来的。精髓知识和表面知识都是有用的，然而它们的分量和重要性却是不一样的。所以必须区分精髓知识和表面知识，不能混为一谈，对待它们的态度应该是不一样的。由于表面知识基本是死的，而且很容易从精髓知识推导衍生出来。我们不应该因为自己知道很多表面知识，就自以为比掌握了精髓知识的人还要强。不应该因为别人不知道某些表面知识，就以为自己高人一等。

IT公司经常有这样的人，以为精通一些看似复杂的命令行，或者某些难用的程序语言就很了不起似的。他们如果听说你不知道某个命令的用法，那简直就像法国人不知道拿破仑，美国人不知道华盛顿一样。这些人没有发现，自己身边有些同事其实掌握着精髓的知识，他们完全有能力从自己已有的知识，衍生制造出所有这些工具，而不只是使用它们，甚至设计得更加完善和方便易用。这种能够设计制造出更好工具的人，往往身负更加重要的任务，所以他们往往会在被现有工具的用法迷惑的时候，非常谦虚的请同事帮助解决，大胆的承认自己的糊涂。

如果你是这个精通工具用法的人，切不可以把同事的谦虚请求当成可以显摆自己“资历”的时候。这同事往往真的是在“不耻下问”。他并不是搞不懂，而是根本不屑于，也没有时间去考虑这种低级问题。他的迷惑，往往来源于工具设计者的失误。他很清楚这一点，他也知道自己的技术水平其实是高于这工具的设计者的。然而为了礼貌，他经常不直接批评这工具的设计，而是谦虚的责怪自己。所以同事向你“虚心请教”，完全是为了制造一种友好融洽的气氛，这样可以节省下时间来干真正重要的事情。这种虚心并不等于他在膜拜你，承认自己的技术能力不如你。

所以正确的对待方式应该是诚恳的表示对这种迷惑的理解，并且坦率的承认工具设计上的不合理，蹩脚之处。如果你能够以这种谦和的态度，而不是自以为专家的态度，同事会高兴地从你这里“学到”他需要的，肤浅的死知识，并且记住它，避免下次再为这种无聊事来打扰你。如果你做出一副“天下只有我知道这奇技淫巧”的态度，同事往往会对你，连同这工具一起产生鄙视的情绪。他下次会照样记不住这东西的用法，然而他却再也不会来找你帮忙，而是一拖再拖。

不要自以为聪明，不要评判别人的智商和能力

在IT公司里，总是有很多人觉得自己聪明，想显示自己比别人聪明。这种人似乎随时都在评判（judge）别人，你说的任何话，不管认真的还是开玩笑的，都会被他们拿去作为评估你智商和能力的依据。

有时候你写了一些代码，自己知道时间不够，可是当时有更重要的事情要做，所以打算以后再改进。如果你提交代码时被这种人看到了，他们就会坚定地认为你一辈子只能写出那样的代码。这就是所谓“wishful thinking”，人只能看到他希望看到的东西。这种人随时都在希望自己比别人聪明，所以他们随时都在监听别人显得不如他聪明的时候，而对别人比他高明的时候视而不见。他们只能看到别人疏忽的时候，因为那是可以证明他们高人一等的有利证据。

当然，谁会喜欢这样的人呢，可是他们在IT公司里相当的普遍。你不敢跟他们说话，特别是不敢开玩笑，因为他们会把你稀里糊涂的玩笑话全部作为你智商低下或者经验不足的证据。你不敢问他们问题，因为他们会认为你问问题，说明你不懂！我发现具有这种心理的人，一般潜意识里都存在着自卑。他们有某些方面（包括智力在内）不如别人，所以总是找机会显得高人一等。我还没有想出可以纠正这种心理问题的有效方法，但如我上节所说，意识到整个行业，包括你仰慕的鼻祖们，其实都不懂很多东西，都是混饭吃的，是一个有效的放松这种心理的手段。

有时候我喜欢自嘲，对人说：“我们这行业的祖先做了这么多BUG来让我们修补。现在你做了一坨屎，我也做了一坨屎，我的屎貌似比你的屎香一点。”这样一来，不但显示出心理的平等和尊重，而且避免了因为谦虚而让对方产生高人一等的情绪。说真的，做这行根本不需要很高的智力，所以最好是完全放弃对人智力的判断。你不比任何人更聪明，也不比他们笨。

解释高级意图，不要使用低级命令

随时都要记住，同事和下属是跟你智力相当的人。他们是通情达理的人，然而却不会简单地服从你的低级命令。像我在Google的队友的做法，就是一个很好的反面教材。其实这位Googler只是想告诉我：“删掉这行文本，然后改成这样……” 就是如此一个简单的事情，然而她却故弄玄虚，不直接告诉我这个“高级意图”，而是使用非常低级的指令：“按Ctrl-k！……” 语气像是在对一个不懂事的小学生说话，好像自己懂很多，别人什么都不知道似的。

有哪个Emacs用户不知道Ctrl-k是删掉一行字呢，况且你现在面对的其实是一个资深Emacs用户。我想大家都看出来这里的问题了吧。这样的低级命令不但逻辑不清楚，而且是对另一个人的智力的严重侮辱。你当我是什么啊？猴子？如果这位Googler表明自己的高级意图，就会很容易在心理上和逻辑上让人接受，比如她可以说：“配置文件的这行应该删掉，改成……”

在项目管理的时候也需要注意。在让人做某一件事之前，应该先解释为什么要做这件事，以及它的重要性。这样才能让人理解，才能尊重程序员的智商。

不要期望新人向自己学习

很多IT公司喜欢把新人当初学者，期望他们“从新的起跑线出发”，向自己“学习”。比如，Google把新员工叫

做“Noogler”（Newbie Googler的意思），甚至给他们发一种特殊的螺旋桨帽子，其寓意在于告诉他们，小屁孩要谦虚，要向伟大的Google学习，将来才可以飞黄腾达。

这其实是非常错误的作法，因为它完全不尊重新员工早已具备的背景知识，把自己的地位强加于他们头上。并不是你说“新的起跑线”就真的可以把人的过去都抹杀了的。新人不了解你们的代码结构和工程方式，并不等于你们的方式就会先进一些。Google里面真的有很多值得学习的东西吗？学校的教育真的一文不值吗？其实恰恰相反。我可以坦然的说，我从自己的教授身上学会了最精髓的知识，而从Google得到的，只是一些很肤浅的，死记硬背就可以掌握的技能，而且其中有挺多其实是糟粕。我在Google做出的所有创新成果，全都是从学校获得的精髓知识的衍生物。很多PhD学生鄙视Google，就是因为Google不但自己技术平庸，反倒喜欢把自己包装成最先进的，超越其它公司和学校的，并且嚣张的期望别人向他们“学习”。

一个真正尊重人才的公司会去了解，尊重和发挥新人从外界带来的特殊技能，施展他们特有的长处，而不是一味期望他们向自己“学习”。只有这样，我们才能保持这些锐利武器的棱角，在激烈的竞争中让自己立于不败之地。如果你一味的让新人“学习”，而无视他们特有的长处，最后就不免沦为平庸。

不要以老师自居，分清“学习”和“了解”

如上文所说，IT行业的很多所谓“知识”，只不过是一些奇技淫巧，用以绕过前人设计上的失误。所以遇到别人不知道一些东西的时候，请不要以为你“教会”了别人什么东西，不要以为自己可以当老师了。以老师自居，使用一些像“跟我学”一类的语言，其实是一种居高临下，不尊重人的行为。

人们很喜欢在获得了信息的时候用“学习”这个词，然而我觉得这个词被滥用了。我们应该分清两种情况：“学习”和“了解”。前者指你通过别人的指点和自己的理解，获得了精髓的，不能轻易制造出来的知识。后者只是指你“了解”了原来不知道的一些事情。举个例子，如果有人把一件物品放在了某个你不知道的地方，你找不到，问他，然后他告诉你了。这种信息的获取，显然不叫“学习”，这种信息也不叫做“知识”。

然而，IT行业很多时候所谓的“学习”，就是类似这种情况。比如，有人写了一些代码，设计了一些框架模块。有人不知道怎么用，然后有人告诉他了。很多人把这种情况称为“学习”，这其实是对人的不尊重。这跟有人告诉你他把东西放在哪里了，是同样性质的。这样的代码和设计，我也可以做，甚至做得更好，凭什么你说我在向你学习呢？我只是了解了一下而已。

所谓学习，必须是更加高级的知识和技能，必须有一种“有收获”，“有提高”的感觉。简单的信息获取不能叫做“学习”，只能叫做“了解”。分清“了解”和“学习”，不以老师自居，是尊重人的一个重要表现。

明确自己的要求，不要使用指责的语气

有些人很怪异，他根本没告诉过你他想要什么，有什么特别的要求，可他潜意识里假设已经告诉你了。到了后来，他发现你的作法不符合要求，于是严厉指责你没有按照他“心目中的要求”办事。这种现象不止限于程序员，而且包括日常生活中的普通人。举个例子，我妈就是这种人的典型，所以我以前在家生活经常很辛苦。她心目中有一套“正确”的做事方式，如果你没猜出来就会挨骂。你为了避免挨骂，干脆什么事都不要做，然后她又会说你懒，所以你就左右不是人 :)

IT公司里面也有挺多这样的人，他们假设有些信息他已经告诉你了，而其实根本没告诉你。到了后来，他们开始指责你没有按照要求做事。有些极其奇葩的公司，里面的程序员不但喜欢以老师自居，而且他们“传授”你“知识”的主要方式是指责。他们事先不告诉你任何规则，然后只在你违反的时候来责备你。我曾经在这样一个公司待过，名字就不提了。

现在举一个具体的场景例子：

A: 你push到master了？
B: 是啊？怎么了？
A: 不准push到master！只能用pull request！
B: 可是你们之前没告诉过我啊……
A: 现在你知道了？ ！

注意到了吗？这不是一个技术问题，而是一个礼节（etiquette）问题。你没有事先告诉别人一些规则，就不该用怪罪的语气来对人说话，况且你的规则还不一定总是对的。所以我现在提醒各位IT公司，在技术上的某些特殊要求必须事先提出来，确保程序员知道并且理解。如果没有事先提出，就不要怪别人没按要求做，因为这是非常伤害人自尊的作法。其实，在任何时候都不应该使用指责的语气，它不但对解决问题没有任何正面作用，而且会恶化人际关系，最终导致更加严重的后果。

程序员的工作量不可用时间衡量

很多IT公司管理层不懂得如何估算程序员的工作量，所以用他们坐在自己位置上工作的时间来估算。如果你能力很强，在很短的时间内把最困难的问题解决了，接下来他们不会让你闲着，而会让你做另外一些很低级的活。这是很不合理的作法。打个比方，能力强的员工就像一辆F1赛车，马力和速度是其他人的几十倍。当然，普通人需要很长时间才能解决，甚至根本没法解决的问题，到他手里很快就化解掉了。这就像一辆F1赛车，眨眼工夫就跑完了别人需要很久的路程。如果你用时间来衡量工作量，那么这辆赛车跑完全程只需要很短时间，所以你算出来的工作量就比普通车子小很多。你能因此说赛车工作不够努力，要他快马再加鞭吗？这显然是不对的。

物理定律是这样：能量 = 功率 x 时间。工作量也应该是同样的计算方法。英明的，真正理解程序员的公司，就不会指望高水平的程序员不停地工作。高水平程序员由于经常能够另辟蹊径，一个就可以抵好几个甚至几十个普通程序员。他们处理的问题比常人的困难很多，费脑力多很多，当然他们需要更好的休息，保养，娱乐，…… 如果你让高水平的程序员太忙了，一刻都不停着，有趣有挑战性的事情做完了就让他们做一些低级无聊的事情，他们悟出这个道理之后，就会故意放慢速度，有时候明明很快做完了也会说没做完。与其这样，不如只期望他们工作短一点的时间，把事情做完就可以。

当然这并不是说初级的程序员就应该过量工作。编程是一项艰苦的脑力活动，超时超量的工作再加上压力，只会带来效率的低下，质量的降低。

不要让其他人修补自己的BUG

这个我已经在一篇专门的文章里讨论过。让一个程序员修补另外一个程序员的BUG，不但是效率低下，而且是不尊重程序员个人价值的作法，应该尽量避免。

在软件行业，经常看到有的公司管理让一个人修补另一个人代码里的BUG。有时候有人写了一段代码，扔出来不管了，然后公司管理让其他工程师来修复它。我想告诉你们，这种方法会很失败。

首先，让一个人修复另一个人的BUG，是不尊重工程师个人技术的表现。久而久之会降低工程师的工作积极性，以至于失去有价值的员工。代码是人用心写出来的作品，就像艺术家的作品一样，它的质量牵挂着一个人的人格和尊严。如果一个人A写了代码，自己都不想修复里面的BUG，那说明A自己都认为他自己的代码是垃圾，不可救药。如果让另一个人B来修复A代码里的BUG，就相当于是让B来收拾其他人丢下的垃圾。可想而知，B在公司的眼里是什么样的地位，受到什么样的尊重。

其次，让一个人修复另一个人的BUG，是效率非常低下的作法。每个人都有自己写代码的风格和技巧，代码里面包含了一个人的思维方式。人很难不经解释理解别人的思想，所以不管这两人的编程技术高下，都会比较难理解。不能理解别人的代码，不能说明这人编程技术的任何方面。所以让一个人修补另一个人的BUG，无论这人技术多么高明，都会导致效率低下。有时候技术越是高的人，修补别人的BUG效率越是低，因为这人根本就写不出来如此糟糕的代码，所以他无法理解，觉得还不如推翻重写一遍。

当我在大学里做程序设计课程助教的时候，我发现如果学生的代码出了问题，你基本是没法简单的帮他们修复的。我的水平显然比学生的高出许多，然而我却经常根本看不懂，也不想看他们的代码，更不要说修复里面的BUG。就像上面提到的，有些人自己根本不知道自己在写什么，做出一堆垃圾来。看这样的代码跟吃屎的感觉差不多。对于这样的代码，你只能跟他们说这是不正确的。至于为什么不正确，你只能让他们自己去改，或者建议他们推翻重写。也许你能指出大致的方向和思路，然而深入到具体的细节却是不可能的，而且不应该是你的职责。这就是我的教授告诉我的做法：如果代码不能运行，直接打一个叉，不用解释，不用推敲，等他们自己把程序改好，或者实在没办法，来office hours找你，向你解释他们的思想。

如果你明白我在说什么，从今天起就对自己的代码负起责任来，不要再让其它人修补自己的BUG，不要再修补其他人的BUG。如果有人离开公司，必须要有人修补他遗留下来的BUG，那么说话应该特别特别的小心。你必须指出需要他帮忙的特殊原因，强调这件事本来不是他的错，本来是不应该他来做的，但是有人走了，没有办法，并且诚恳的为此类事情的发生表示歉意。只有这样，程序员才会心甘情愿的在这种特殊关头，修补另外一个人的BUG。

不要嚷着要别人写测试

在很多程序员的脑子里，所谓的“流程”和“测试”，比真正解决问题的代码还重要。他们跟你说起这些，那真的叫正儿八经，义正言辞啊！所以有时候你很迷惑，这些人除了遵守这些按部就班的规矩，还知道些什么。大概没有能力的人都喜欢追究各种规矩吧，这样可以显得自己“没有功劳有苦劳”。这些人自己写的代码很平庸，不知道如何简单有效地解决困难的问题，却喜欢在别人提交代码让他review的时候叫喊：“测试很重要！覆盖很重要！你要再加一些测试才能通过我的review！”

本来code review是让他们帮忙发现可能存在的问题，有些人却仿佛把它作为了评判（judge）其他人能力，经验，甚至智商的机会。他们根本不明白别人代码的实质价值，就知道以一些表面现象来判断。我在Google实习，最后提交了质量和难度都非常高的代码，然而一些完全没能力写出这样代码的人，不但没表示出最基本的肯定，反而发出沉闷的咆

哮：“快——写——测——试！”你觉得我会高兴吗?

我并不否认测试的用处，然而很多人提起这些事情时候，语气和态度是非常不尊重，让人反感的。这些人不但没有为解决问题作出任何实质贡献，当有人提交解决方案的时候，他们没有表达对真正做出贡献的人的尊重和肯定，反而指责别人没写测试。好像比他高明的人解决了问题，他反倒才是那个有发言权的，可以评判你的代码质量似的：“我管你代码写得多好，我完全没能力写出来，但你没写测试就是不够专业。你懂不懂测试的重要性啊，还做程序员！”

人际交往的问题经常不在于你说了什么，而在于你是怎么说的。所以我的意思并不是说你不该建议写测试，然而建议就该有建议的语气和态度。因为你没有做实际的工作，所以一些礼貌用语，比如“请”，“可不可以”……是必须的。经常有人说话不注意语气和态度，让人反感，却以自己是工程师，不善于跟人说话为借口。永远要记住，你没有做事，说话就应该委婉，切不可使用光秃秃的祈使句，说得好像这事别人非做不可，不做就是不懂规矩一样。

礼貌的语言，跟人的职业完全没有关系。身为工程师，完全不能作为说话不礼貌的借口。

关于Git的礼节

Git是现在最流行的代码版本控制工具。用外行话说，Git就是一个代码的“仓库”或者“保管”，这样很多人修改了代码之后，可以知道是谁改了哪一块。其实不管什么工具，不管是编辑器，程序语言，还是版本控制工具，比起程序员的核心思想来，都是次要的东西，都是起辅助作用的。可是Git这工具似乎特别惹人恼火。

Git并不像很多人吹嘘的那么好用，其中有明显的蹩脚设计。跟Unix的传统一脉相承，Git没有一个良好的包装，设计者把自己的内部实现细节无情地泄露给了用户，让用户需要琢磨者设计者内部到底怎么实现的，否则很多时候不知道该怎么办。用户被迫需要记住挺多稀奇古怪的命令，而且命令行的设计也不怎么合理，有时候你需要加-f之类的参数，各个参数的位置可能不一致，而且加了还不一定能起到你期望的效果。各种奇怪的现象，比如”head detached”，都强迫用户去了解它内部是怎么设计的。随着Git版本的更新，新的功能和命令不断地增加，后来你终于看到命令行里出现了foreach，才发现它的命令行就快变成一个（劣质的）程序语言。如果你了解ydiff的设计思想，就会发现Git之类基于文本的版本控制工具，其实属于古代的东西。然而很多人把Git奉为神圣，就因为它是Linus Torvalds设计的。

Git最让人恼火的地方并不是它用起来麻烦，而是它的“资深用户”们居高临下的态度给你造成的心理阴影。好些人因为自己“精通Git”就以为高人一等，摆出一副专家的态度。随着用户的增加，Git最初的设计越来越被发现不够用，所以一些约定俗成的规则似乎越来越多，可以写成一本书！跟Unix的传统一脉相承，Git给你很多可以把自己套牢的“机制”，到时候出了问题就怪你自己不知道。所以你就经常听有人煞有介事的说：“并不是Git允许你这么做，你就可以这么做的！Unix的哲学是不阻止傻人做傻事……”如果你提交代码时不知道Git用户一些约定俗成的规则，就会有人嚷嚷：“rebase了再提交！”“不要push到master！”“不要merge！”“squash commits！”如果你不会用git submodule之类的东西，有人可能还会鄙视你，说：“你应该知道这些！”

打个比方，这样的嚷嚷给人的感觉是，你得了奥运会金牌之后，把练习用的器材还回到器材保管科，结果管理员对你大吼：“这个放这边！那个放那边！懂不懂规矩啊你？”看出来问题了吗？程序员提交了有高价值的代码（奥运金牌），结果被一些自认为Git用的很熟的人（器材保管员）厉声呵斥。

一个尊重程序员的公司文化，就应该把程序员作为运动健将，把程序员的代码放在尊贵的地位。其它的工具，都应该像器材保管科一样。我们尊重这些器材保管员，然而如果运动员们不懂你制定的器材摆放规矩，也应该表示出尊重和理解，说话应该和气有礼貌，不应该骑到他们头上。所以，对于Git的一些命令和用法，我建议大家向新手介绍时，这样开场：“你本来不该知道这些的，可是现在我们没有更好的工具，所以得这样弄一下……”