使用Perl自动化UNIX系统管理 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/167/2021_2022__E4_BD_BF_E7_94_A8Perl_c103_167241.htm UNIX 系统管理总是一个棘手的问题，运用正确的工具会使这个问题变得容易。在这一部分中，Teodor 提出了关于使用 Perl 来简化和牢固系统管理的想法。在这种环境中，系统配置引擎 cfengine 是一个极其重要的工具。 要完成本文中的练习，系统中必须安装了 Perl 5.6.0。操作系统最好是主流 UNIX 安装（Linux、Solaris、BSD）的最近版本（2000 或更新）。在较早版本的 Perl 和 UNIX 以及其它操作系统上也可以使用本文中的示例，但应当将可能的功能故障作为练习来解决。 UNIX 管理具有挑战性的一大原因是每个 UNIX 供应商认为标准是针对低能傻瓜。所以，即使是同一供应商的操作系统（SunOS 4.x 和 Solaris 5.x）也可以是根本不同。在某些情况下，甚至根本没有供应商。例如，Linux 没有单独的供应商（虽然 Red Hat 目前是最大的 Linux 分发版），每一个版本的 Linux 都有其独到之处。如果 POSIX 标准化做得正确，那么它是解决这一问题的正确方向上的一个步骤。遗憾的是，它只能保证系统管理所需功能的一个小的子集。 正如我经常所说：了解您的工具。如果试图仅用一种工具、语言、或方法做每件事情，可能是一场噩梦。要具有灵活性。 如果存在一个系统管理公理，那就是：两次过后，没有系统管理任务是有趣的。如果您发现正在重复做单调而枯燥的事，那么自动化它。当然，有时很难自动化，但应该至少考虑这个问题，并且权衡其优势及自动化所花费的时间。cfengine 工具 如果您对自动化系统管理是认真的

，那么应该了解 cfengine 工具。仅当您宁愿把时间都花在 vi 编辑器时，可以不去了解 cfengine 。 cfengine 是一种系统配置引擎。它获取配置脚本作为输入，然后根据这些脚本来行动。目前版本是 1.6.3（非常稳定的发行版），而且版本 2.0 也呼之欲出。有关 cfengine 开发的更多信息，请访问 cfengine 网站（请参阅本文后面的 参考资料）。 不一定要用 cfengine 提供您的所有东西，而且您不可能立刻需要所有东西。一开始时，您的 cfengine 配置文件应该很简单，并且随着发现更多东西希望自动化而增长。 来自 cfengine 命令参考大全，这里有其最值得注意的特性： 可以监控和修改文件许可权和 ACL。例如，/etc/shadow 可以与 0400/root/sys 许可权保持一致，而且如果那些许可权发生变化，可以警告系统管理员或即刻纠正它们。 根据相应 fstab 变化，可自动安装和卸载 NFS 文件。可以通过单一文件来管理子网掩码、DNS 配置、缺省路由和主网络接口；文件和目录可以递归复制至另一位置，要么本地复制，要么从远程服务器复制。 可以编辑（这是一个 非常强大的特性，提供了正则表达式和全局查找／替换）、轮转（譬如，日志文件）或删除文件。 可以链接文件（单一的和／或目录下的所有文件或与正则表达式匹配的文件）和整个目录。 可以根据进程表中正则表达式的匹配来启动、杀死、重启进程或发送任意信号。 可以运行任意命令。 上述所有这些根据操作系统类型和修订版本、一天中的时间、任意用户定义的类、文件中文件、目录或数据的有无等等可以是有条件的。 即使用 Perl 可以做 cfengine 所做的所有事情，为什么要从头开始呢？例如，如果想用另一个词替换某个词，编辑文件可以是简单的一行程序。当开始允许系统的子类型、逻

辑系统部分以及所有其它杂项因素时，这一行程序会变成 300 行。为什么不在 cfengine 中做呢？它产生 100 行可读的配置代码。 根据我自己的经验，因为可以从最小配置文件开始，然后随着时间流逝逐步地向 cfengine 添加一些东西，所以将 cfengine 介绍给站点是很容易的。没有人喜欢突然的变化，所有系统管理员更是如此（因为如果任何事出错，他们理所当然地会受到责难）。 配置文件管理 管理配置文件是艰苦的。可以通过考虑 cfengine 是否胜任该任务开始。遗憾的是，cfengine 的编辑是面向行的，所以它可能不太适合复杂的配置文件。但对于如 TCP 包装器配置文件 /etc/hosts.allow 那样的简单文件 cfengine 是最适合的。 通常，希望保留配置文件的多个版本。譬如，可能需要在 /etc/resolv.conf 中有两组 DNS 配置设置，一组是用于外部机器，另一组是用于内部机器。很自然，外部 DNS resolv.conf 可以进入称为 "external" 的目录，而内部 resolv.conf 可以进入相应的 "internal" 目录。让我们假定这两个目录都在一个全局 "spec" 目录下，该目录是配置文件的一种根目录。 下列代码会遍历 spec 目录，搜索适合于给定机器的文件名。它将从 /usr/local/spec 开始，然后往下，寻找与请求相匹配的文件。而且，它将检查每个目录的名称是否与属于某些机器的类相同。因此，如果我们请求 locate_global(resolv.conf, wonka) ，该函数将在 /usr/local/spec 目录下查找 resolv.conf 文件，该文件要么在根目录下，要么在该根目录的子目录下，它的名称应与 "wonka" 机器所属的类相匹配。所以，如果 "wonka" 属于 "chocolate" 类，并且如果有 /usr/local/spec/chocolate/resolv.conf 文件，那么 locate_global() 将返回 "/usr/local/spec/chocolate/resolv.conf"。

http://127.0.0.1:8080/developerworks/cn/linux/sdk/perl\culture-5/index.shtml locate_global() 找到与文件相匹配的多个版本（譬如，/usr/local/spec/chocolate/resolv.conf 和 /usr/local/spec/resolv.conf），则它会放弃。这里假设没有配置比有两个错误之一要好。还有，请注意，机器可以属于不止一个类。 可以构建这样的结构。譬如，/usr/local/spec/external/chocolate/resolv.conf /usr/local/spec/internal/chocolate/resolv.conf /usr/local/spec/external/sugar/resolv.conf /usr/local/spec/internal/sugar 