UNIX 高手的 10 个习惯

克服不良的 UNIX 使用模式

级别：中级

采用 10 个能够提高您的 UNIX® 命令行效率的好习惯——并在此过程中摆脱不良的使用模式。本文循序渐进地指导您学习几项用于命令行操作的技术，这些技术非常好，但是通常被忽略。了解常见错误和克服它们的方法，以便您能够确切了解为何值得采用这些 UNIX 习惯。

胥家杰

2007-2-27

引言

当您经常使用某个系统时，往往会陷入某种固定的使用模式。有时，您没有养成以尽可能最好的方式做事的习惯。有时，您的不良习惯甚至会导致出现混乱。纠正此类缺点的最佳方法之一，就是有意识地采用抵制这些坏习惯的好习惯。本文提出了 10 个值得采用的 UNIX 命令行习惯——帮助您克服许多常见使用怪癖，并在该过程中提高命令行工作效率的好习惯。下面列出了这 10 个好习惯，之后对进行了更详细的描述。

采用 10 个好习惯

要采用的十个好习惯为：

1. 在单个命令中创建目录树。
2. 更改路径；不要移动存档。
3. 将命令与控制操作符组合使用。
4. 谨慎引用变量。
5. 使用转义序列来管理较长的输入。
6. 在列表中对命令分组。
7. 在 find 之外使用 xargs。
8. 了解何时 grep 应该执行计数——何时应该绕过。
9. 匹配输出中的某些字段，而不只是对行进行匹配。
10. 停止对 cat 使用管道。

在单个命令中创建目录树

清单 1 演示了最常见的 UNIX 坏习惯之一：一次定义一个目录树。

清单 1 坏习惯 1 的示例：单独定义每个目录树

```
~ $ mkdir tmp
~ $ cd tmp
~/tmp $ mkdir a
~/tmp $ cd a
~/tmp/a $ mkdir b
~/tmp/a $ cd b
~/tmp/a/b/ $ mkdir c
~/tmp/a/b/ $ cd c
~/tmp/a/b/c $
```

使用 mkdir 的 -p 选项并在单个命令中创建所有父目录及其子目录要容易得多。但是即使对于知道此选项的管理员，他们在命令行上创建子目录时也仍然束缚于逐步创建每级子目录。花时间有意识地养成这个好习惯是值得的：

清单 2 好习惯 1 的示例：使用一个命令来定义目录树

```
~ $ mkdir -p tmp/a/b/c
```

您可以使用此选项来创建整个复杂的目录树（在脚本中使用是非常理想的），而不只是创建简单的层次结构。例如：

清单 3 好习惯 1 的另一个示例：使用一个命令来定义复杂的目录树

```
~ $ mkdir -p project/{lib/ext,bin,src,doc/{html,info,pdf},demo/stat/a}
```

过去，单独定义目录的唯一借口是您的 mkdir 实现不支持此选项，但是在大多数系统上不再是这样了。IBM、AIX®、mkdir、GNU mkdir 和其他遵守单一 UNIX 规范 (Single UNIX Specification)的系统现在都具有此选项。

对于仍然缺乏该功能的少数系统，您可以使用 mkdirhier 脚本（请参见参考资料），此脚本是执行相同功能的 mkdir 的包装：

```
~ $ mkdirhier project/{lib/ext,bin,src,doc/{html,info,pdf},demo/stat/a}
```

更改路径；不要移动存档

另一个不良的使用模式是将 .tar 存档文件移动到某个目录，因为该目录恰好是您希望在其中提取 .tar 文件的目录。其实您根本不需要这样做。您可以随心所欲地将任何 .tar 存档文件解压缩到任何目录——这就是 -C 选项的用途。在解压缩某个存档文件时，使用 -C 选项来指定要在其中解压缩该文件的目录：

清单 4 好习惯 2 的示例：使用选项 -C 来解压缩 .tar 存档文件

```
~ $ tar xvf -C tmp/a/b/c newarc.tar.gz
```

相对于将存档文件移动到您希望在其中解压缩它的位置，切换到该目录，然后才解压缩它，养成使用 -C 的习惯则更加可取——当存档文件位于其他某个位置时尤其如此。

将命令与控制操作符组合使用

您可能已经知道，在大多数 Shell 中，您可以在单个命令行上通过在命令之间放置一个分号 (;) 来组合命令。该分号是 Shell *控制操作符*，虽然它对于在单个命令行上将离散的命令串联起来很有用，但它并不适用于所有情况。例如，假设您使用分号来组合两个命令，其中第二个命令的正确执行完全依赖于第一个命令的成功完成。如果第一个命令未按您预期的那样退出，第二个命令仍然会运行——结果会导致失败。相反，应该使用更适当的控制操作符（本文将描述其中的部分操作符）。只要您的 Shell 支持它们，就值得养成使用它们的习惯。

仅当另一个命令返回零退出状态时才运行某个命令

使用 && 控制操作符来组合两个命令，以便*仅当* 第一个命令返回零退出状态时才运行第二个命令。换句话说，如果第一个命令运行成功，则第二个命令将运行。如果第一个命令失败，则第二个命令根本就不运行。例如：

清单 5 好习惯 3 的示例：将命令与控制操作符组合使用

```
~ $ cd tmp/a/b/c && tar xvf ~/archive.tar
```

在此例中，存档的内容将提取到 ~/tmp/a/b/c 目录中，除非该目录不存在。如果该目录不存在，则 tar 命令不会运行，因此不会提取任何内容。

仅当另一个命令返回非零退出状态时才运行某个命令

类似地，|| 控制操作符分隔两个命令，并且仅当第一个命令返回非零退出状态时才运行第二个命令。换句话说，如果第一个命令*成功*，则第二个命令不会运行。如果第一个命令失败，则第二个命令*才会* 运行。在测试某个给定目录是否存在时，通常使用此操作符，如果该目录不存在，则创建它：

清单 6 好习惯 3 的另一个示例：将命令与控制操作符组合使用

```
~ $ cd tmp/a/b/c || mkdir -p tmp/a/b/c
```

您还可以组合使用本部分中描述的控制操作符。每个操作符都影响最后的命令运行：

清单 7 好习惯 3 的组合示例：将命令与控制操作符组合使用

```
~ $ cd tmp/a/b/c || mkdir -p tmp/a/b/c && tar xvf -C tmp/a/b/c ~/archive.tar
```

谨慎引用变量

始终要谨慎使用 Shell 扩展和变量名称。一般最好将变量调用包括在双引号中，除非您有不这样做的足够理由。类似地，如果您直接在字母数字文本后面使用变量名称，则还要确保将该变量名称包括在方括号 ([]) 中，以使其与周围的文本区分开来。否则，Shell 将把尾随文本解释为变量名称的一部分——并且很可能返回一个空值。清单 8 提供了变量的各种引用和非引用及其影响的示例。

清单 8 好习惯 4 的示例：引用（和非引用）变量

```
~ $ ls tmp/
a b
~ $ VAR="tmp/*"
~ $ echo $VAR
tmp/a tmp/b
~ $ echo "$VAR"
tmp/*
~ $ echo $VARa

~ $ echo "$VARa"

~ $ echo "${VAR}a"
tmp/*a
~ $ echo ${VAR}a
tmp/a
~ $
```

使用转义序列来管理较长的输入

您或许看到过使用反斜杠 (\) 来将较长的行延续到下一行的代码示例，并且您知道大多数 Shell 都将您通过反斜杠联接的后续行上键入的内容视为单个长行。然而，您可能没有在命令行中像通常那样利用此功能。如果您的终端无法正确处理多行回绕，或者您的命令行比通常小（例如在提示符下有长路经的时候），反斜杠就特别有用。反斜杠对于了解键入的长输入行的含义也非常有用，如以下示例所示：

清单 9 好习惯 5 的示例：将反斜杠用于长输入

```
~ $ cd tmp/a/b/c || \
> mkdir -p tmp/a/b/c && \
> tar xvf -C tmp/a/b/c ~/archive.tar
```

或者，也可以使用以下配置：

清单 10 好习惯 5 的替代示例：将反斜杠用于长输入

```
~ $ cd tmp/a/b/c \
>                   || \
> mkdir -p tmp/a/b/c \
>                      && \
> tar xvf -C tmp/a/b/c ~/archive.tar
```

然而，当您将输入行划分到多行上时，Shell 始终将其视为单个连续的行，因为它总是删除所有反斜杠和额外的空格。

注意：在大多数 Shell 中，当您按向上箭头键时，整个多行输入将重绘到单个长输入行上。

在列表中对命令分组

大多数 Shell 都具有在列表中对命令分组的方法，以便您能将它们的合计输出向下传递到某个管道，或者将其任何部分或全部流重定向到相同的地方。您一般可以通过在某个 Subshell 中运行一个命令列表或通过在当前 Shell 中运行一个命令列表来实现此目的。

在 Subshell 中运行命令列表

使用括号将命令列表包括在单个组中。这样做将在一个新的 Subshell 中运行命令，并允许您重定向或收集整组命令的输出，如以下示例所示：

清单 11 好习惯 6 的示例：在 Subshell 中运行命令列表

```
~ $ ( cd tmp/a/b/c/ || mkdir -p tmp/a/b/c && \
> VAR=$PWD; cd ~; tar xvf -C $VAR archive.tar ) \
> | mailx admin -S "Archive contents"
```

在此示例中，该存档的内容将提取到 tmp/a/b/c/ 目录中，同时将分组命令的输出（包括所提取文件的列表）通过邮件发送到地址 admin。

当您在命令列表中重新定义环境变量，并且您不希望将那些定义应用于当前 Shell 时，使用 Subshell 更可取。

在当前 Shell 中运行命令列表

将命令列表用大括号 ({}) 括起来，以在*当前* Shell 中运行。确保在括号与实际命令之间包括空格，否则 Shell 可能无法正确解释括号。此外，还要确保列表中的最后一个命令以分号结尾，如以下示例所示：

清单 12 好习惯 6 的另一个示例：在当前 Shell 中运行命令列表

```
~ $ { cp ${VAR}a . && chown -R guest.guest a && \
> tar cvf newarchive.tar a; } | mailx admin -S "New archive"
```

在 find 之外使用 xargs

使用 xargs 工具作为筛选器，以充分利用从 find 命令挑选的输出。find 运行通常提供与某些条件匹配的文件列表。此列表被传递到 xargs 上，后者然后使用该文件列表作为参数来运行其他某些有用的命令，如以下示例所示：

清单 13 xargs 工具的经典用法示例

```
~ $ find some-file-criteria some-file-path | \
> xargs some-great-command-that-needs-filename-arguments
```

然而，不要将 xargs 仅看作是 find 的辅助工具；它是一个未得到充分利用的工具之一，当您养成使用它的习惯时，将会希望进行所有试验，包括以下用法。

传递空格分隔的列表

在最简单的调用形式中，xargs 就像一个筛选器，它接受一个列表（每个成员分别在单独的行上）作为输入。该工具将那些成员放置在单个空格分隔的行上：

清单 14 xargs 工具产生的输出示例

```
~ $ xargs
a
b
c
Control-D
a b c
~ $
```

您可以发送通过 xargs 来输出文件名的任何工具的输出，以便为其他某些接受文件名作为参数的工具获得参数列表，如以下示例所示：

清单 15 xargs 工具的使用示例

```
~/tmp $ ls -1 | xargs
December_Report.pdf README a archive.tar mkdirhier.sh
~/tmp $ ls -1 | xargs file
December_Report.pdf: PDF document, version 1.3
README: ASCII text
a: directory
archive.tar: POSIX tar archive
mkdirhier.sh: Bourne shell script text executable
~/tmp $
```

xargs 命令不只用于传递文件名。您还可以在需要将文本筛选到单个行中的任何时候使用它：

清单 16 好习惯 7 的示例：使用 xargs 工具来将文本筛选到单个行中

```
~/tmp $ ls -l | xargs
-rw-r--r-- 7 joe joe 12043 Jan 27 20:36 December_Report.pdf -rw-r--r-- 1 \
root root 238 Dec 03 08:19 README drwxr-xr-x 38 joe joe 354082 Nov 02 \
16:07 a -rw-r--r-- 3 joe joe 5096 Dec 14 14:26 archive.tar -rwxr-xr-x 1 \
joe joe 3239 Sep 30 12:40 mkdirhier.sh
~/tmp $
```

谨慎使用 xargs

从技术上讲，使用 xargs 很少遇到麻烦。缺省情况下，文件结束字符串是下划线（_）；如果将该字符作为单个输入参数来发送，则它之后的所有内容将被忽略。为了防止这种情况发生，可以使用 -e 标志，它在不带参数的情况下完全禁用结束字符串。

了解何时 grep 应该执行计数——何时应该绕过

避免通过管道将 grep 发送到 wc -l 来对输出行数计数。grep 的 -c 选项提供了对与特定模式匹配的行的计数，并且一般要比通过管道发送到 wc 更快，如以下示例所示：

清单 17 好习惯 8 的示例：使用和不使用 grep 的行计数

```
~ $ time grep and tmp/a/longfile.txt | wc -l
2811


real    0m0.097s
user    0m0.006s
sys     0m0.032s
~ $ time grep -c and tmp/a/longfile.txt
2811


real    0m0.013s
user    0m0.006s
sys     0m0.005s
~ $
```

除了速度因素外，-c 选项还是执行计数的好方法。对于多个文件，带 -c 选项的 grep 返回每个文件的单独计数，每行一个计数，而针对 wc 的管道则提供所有文件的组合总计数。

然而，不管是否考虑速度，此示例都表明了另一个要避免地常见错误。这些计数方法仅提供*包含匹配模式的行数*——如果那就是您要查找的结果，这没什么问题。但是在行中具有某个特定模式的多个实例的情况下，这些方法无法为您提供实际匹配*实例数量*的真实计数。归根结底，若要对实例计数，您还是要使用 wc 来计数。首先，使用 -o 选项（如果您的版本支持它的话）来运行 grep 命令。此选项*仅*输出匹配的模式，每行一个模式，而不输出行本身。但是您不能将它与 -c 选项结合使用，因此要使用 wc -l 来对行计数，如以下示例所示：

清单 18 好习惯 8 的示例：使用 grep 对模式实例计数

```
~ $ grep -o and tmp/a/longfile.txt | wc -l
3402
~ $
```

在此例中，调用 wc 要比第二次调用 grep 并插入一个虚拟模式（例如 grep -c）来对行进行匹配和计数稍快一点。

匹配输出中的某些字段，而不只是对行进行匹配

当您只希望匹配输出行中*特定字段* 中的模式时，诸如 awk 等工具要优于 grep。

下面经过简化的示例演示了如何仅列出 12 月修改过的文件。

清单 19 坏习惯 9 的示例：使用 grep 来查找特定字段中的模式

```
~/tmp $ ls -l /tmp/a/b/c | grep Dec
-rw-r--r--   7 joe joe   12043 Jan 27 20:36 December_Report.pdf
-rw-r--r--   1 root root   238 Dec 03 08:19 README
-rw-r--r--   3 joe joe    5096 Dec 14 14:26 archive.tar
~/tmp $
```

在此示例中，grep 对行进行筛选，并输出其修改日期和名称中带 Dec 的所有文件。因此，诸如 December_Report.pdf 等文件是匹配的，即使它自从一月份以来还未修改过。这可能不是您希望的结果。为了匹配特定字段中的模式，最好使用 awk，其中的一个关系运算符对确切的字段进行匹配，如以下示例所示：

清单 20 好习惯 9 的示例：使用 awk 来查找特定字段中的模式

```
~/tmp $ ls -l | awk '$6 == "Dec"'
-rw-r--r--   3 joe joe    5096 Dec 14 14:26 archive.tar
-rw-r--r--   1 root root   238 Dec 03 08:19 README
~/tmp $
```

有关如何使用 awk 的更多详细信息，请参见参考资料。

停止对 cat 使用管道

grep 的一个常见的基本用法错误是通过管道将 cat 的输出发送到 grep 以搜索单个文件的内容。这绝对是不必要的，纯粹是浪费时间，因为诸如 grep 这样的工具接受文件名作为参数。您根本不需要在这种情况下使用 cat，如以下示例所示：

清单 21 好习惯和坏习惯 10 的示例：使用带和不带 cat 的 grep

```
~ $ time cat tmp/a/longfile.txt | grep and
2811

real    0m0.015s
user    0m0.003s
sys     0m0.013s
~ $ time grep and tmp/a/longfile.txt
2811
```

```
real    0m0.010s
user    0m0.006s
sys     0m0.004s
~ $
```

此错误存在于许多工具中。由于大多数工具都接受使用连字符 (-) 的标准输入作为一个参数，因此即使使用 cat 来分散 stdin 中的多个文件，参数也通常是无效的。仅当您使用带多个筛选选项之一的 cat 时，才真正有必要在管道前首先执行连接。

结束语：养成好习惯

最好检查一下您的命令行习惯中的任何不良的使用模式。不良的使用模式会降低您的速度，并且通常会导致意外错误。本文介绍了 10 个新习惯，它们可以帮助您摆脱许多最常见的使用错误。养成这些好习惯是加强您的 UNIX 命令行技能的积极步骤。

参考资料

学习

- "Use free software within commercial UNIX"（developerWorks, 2006 年 2 月）是一个有关在商业性专用 UNIX 实现上使用众多开放源代码软件的简短入门读物。
- "在 Bash shell 中工作"（developerWorks，2006 年 5 月）提供了关于流行的 Bash Shell 的介绍性教程。
- "GAWK 入门：AWK 语言基础"（developerWorks，2006 年 9 月）介绍了如何使用 AWK 语言来操作文本。
- "磨练构建正则表达式模式的技能"（developerWorks，2006 年 7 月）描述了使用 grep 的更有用的方法。
- 访问 developerWorks AIX and UNIX 专区以获取提高您的技能所需的资源。
- 您是 AIX 和 UNIX 新手吗？请访问"AIX and UNIX 新手入门"页以了解更多信息。
- 浏览技术书店，以了解有关这些技术主题及其他技术主题的相关书籍。

获得产品和技术

- 若要获得 mkdirhier 的副本，您可以从 Haskell compiler 下载某个版本。

讨论

- 播客：收听播客并与 IBM 技术专家保持同步。
- 访问 developerWorks 博客，从而加入到 developerWorks 社区中来。

关于作者

Michael Stutz 是 *The Linux Cookbook* 一书的作者，他仅使用开放源码软件对该书进行了设计和排版。他的研究兴趣包括数字出版和图书的发展未来。他使用各种 UNIX 操作系统已有 20 多年。您可以通过 stutz@dsl.org 与他联系。