# 命令行的艺术

gitter join chat



熟练使用命令行是一种常常被忽视，或被认为难以掌握的技能，但实际上，它会提高你作为工程师的灵活性以及生产力。本文是一份我在 Linux 上工作时，发现的一些命令行使用技巧的摘要。有些技巧非常基础，而另一些则相当复杂，甚至晦涩难懂。这篇文章并不长，但当你能够熟练掌握这里列出的所有技巧时，你就学会了很多关于命令行的东西了。

这篇文章是许多作者和译者共同的成果。
这里的部分内容
首次
出现
于 Quora，
但已经迁移到了 Github，并由众多高手做出了许多改进。
如果你在本文中发现了错误或者存在可以改善的地方，请贡献你的一份力量。

## 前言

涵盖范围：

- 这篇文章不仅能帮助刚接触命令行的新手，而且对具有经验的人也大有裨益。本文致力于做到*覆盖面广*（涉及所有重要的内容），*具体*（给出具体的最常用的例子），以及*简洁*（避免冗余的内容，或是可以在其他地方轻松查到的细枝末节）。在特定应用场景下，本文的内容属于基本功或者能帮助您节约大量的时间。
- 本文主要为 Linux 所写，但在仅限 OS X 系统章节和仅限 Windows 系统章节中也包含有对应操作系统的内容。除去这两个章节外，其它的内容大部分均可在其他类 Unix 系统或 OS X，甚至 Cygwin 中得到应用。
- 本文主要关注于交互式 Bash，但也有很多技巧可以应用于其他 shell 和 Bash 脚本当中。
- 除去“标准的”Unix 命令，本文还包括了一些依赖于特定软件包的命令（前提是它们具有足够的价值）。

注意事项：

- 为了能在一页内展示尽量多的东西，一些具体的信息可以在引用的页面中找到。我们相信机智的你知道如何使用 Google 或者其他搜索引擎来查阅到更多的详细信息。文中部分命令需要您使用 `apt-get`，`yum`，`dnf`，`pacman`，`pip` 或 `brew`（以及其它合适的包管理器）来安装依赖的程序。
- 遇到问题的话，请尝试使用 Explainshell 去获取相关命令、参数、管道等内容的解释。

# 基础

- 学习 Bash 的基础知识。具体地，在命令行中输入 `man bash` 并至少全文浏览一遍; 它理解起来很简单并且不冗长。其他的 shell 可能很好用，但 Bash 的功能已经足够强大并且到几乎总是可用的（如果你只学习 zsh，fish 或其他的 shell 的话，在你自己的设备上会显得很方便，但过度依赖这些功能会给您带来不便，例如当你需要在服务器上工作时）。
- 熟悉至少一个基于文本的编辑器。通常而言 Vim （`vi`） 会是你最好的选择，毕竟在终端中编辑文本时 Vim 是最好用的工具（甚至大部分情况下 Vim 要比 Emacs、大型 IDE 或是炫酷的编辑器更好用）。
- 学会如何使用 `man` 命令去阅读文档。学会使用 `apropos` 去查找文档。知道有些命令并不对应可执行文件，而是在 Bash 内置好的，此时可以使用 `help` 和 `help -d` 命令获取帮助信息。你可以用 `type` 命令 来判断这个命令到底是可执行文件、shell 内置命令还是别名。
- 学会使用 `>` 和 `<` 来重定向输出和输入，学会使用 `|` 来重定向管道。明白 `>` 会覆盖了输出文件而 `>>` 是在文件末添加。了解标准输出 stdout 和标准错误 stderr。
- 学会使用通配符 `*`（或许再算上 `?` 和 `[...]`） 和引用以及引用中 `'` 和 `"` 的区别（后文中有一些具体的例子）。
- 熟悉 Bash 中的任务管理工具：`&`，**ctrl-z**，**ctrl-c**，`jobs`，`fg`，`bg`，`kill` 等。
- 学会使用 `ssh` 进行远程命令行登录，最好知道如何使用 `ssh-agent`，`ssh-add` 等命令来实现基础的无密码认证登录。
- 学会基本的文件管理工具：`ls` 和 `ls -l` （了解 `ls -l` 中每一列代表的意义），`less`，`head`，`tail` 和 `tail -f` （甚至 `less +F`），`ln` 和 `ln -s` （了解硬链接与软链接的区别），`chown`，`chmod`，`du` （硬盘使用情况概述：`du -hs *`）。关于文件系统的管理，学习 `df`，`mount`，`fdisk`，`mkfs`，`lsblk`。知道 inode 是什么（与 `ls -i` 和 `df -i` 等命令相关）。
- 学习基本的网络管理工具：`ip` 或 `ifconfig`，`dig`。
- 学习并使用一种版本控制管理系统，例如 `git`。
- 熟悉正则表达式，学会使用 `grep` / `egrep`，它们的参数中 `-i`，`-o`，`-v`，`-A`，`-B` 和 `-C` 这些是很常用并值得认真学习的。
- 学会使用 `apt-get`，`yum`，`dnf` 或 `pacman` （具体使用哪个取决于你使用的 Linux 发行版）来查找和安装软件包。并确保你的环境中有 `pip` 来安装基于 Python 的命令行工具 （接下来提到的部分程序使用 `pip` 来安装会很方便）。

# 日常使用

- 在 Bash 中，可以通过按 **Tab** 键实现自动补全参数，使用 **ctrl-r** 搜索命令行历史记录（按下按键之后，输入关键字便可以搜索，重复按下 **ctrl-r** 会向后查找匹配项，按下 **Enter** 键会执行当前匹配的命令，而按下右方向键会将匹配项放入当前行中，不会直接执行，以便做出修改）。
- 在 Bash 中，可以按下 **ctrl-w** 删除你键入的最后一个单词，**ctrl-u** 可以删除行内光标所在位置之前的内容，**alt-b** 和 **alt-f** 可以以单词为单位移动光标，**ctrl-a** 可以将光标移至行首，**ctrl-e** 可以将光标移至行尾，**ctrl-k** 可以删除光标至行尾的所有内容，**ctrl-l** 可以清屏。键入 `man readline` 可以查看 Bash 中的默认快捷键。内容有很多，

例如 **alt-.** 循环地移向前一个参数，而 **alt-*** 可以展开通配符。

- 你喜欢的话，可以执行 `set -o vi` 来使用 vi 风格的快捷键，而执行 `set -o emacs` 可以把它改回来。
- 为了便于编辑长命令，在设置你的默认编辑器后（例如 `export EDITOR=vim`），**ctrl-x ctrl-e** 会打开一个编辑器来编辑当前输入的命令。在 vi 风格下快捷键则是 **escape-v**。
- 键入 `history` 查看命令行历史记录，再用 `!n`（n 是命令编号）就可以再次执行。其中有许多缩写，最有用的大概就是 `!$`，它用于指代上次键入的参数，而 `!!` 可以指代上次键入的命令了（参考 man 页面中的"HISTORY EXPANSION"）。不过这些功能，你也可以通过快捷键 **ctrl-r** 和 **alt-.** 来实现。
- `cd` 命令可以切换工作路径，输入 `cd ~` 可以进入 home 目录。要访问你的 home 目录中的文件，可以使用前缀 `~`（例如 `~/.bashrc`）。在 sh 脚本里则用环境变量 `$HOME` 指代 home 目录的路径。
- 回到前一个工作路径：`cd -`。
- 如果你输入命令的时候中途改了主意，按下 **alt-#** 在行首添加 # 把它当做注释再按下回车执行（或者依次按下 **ctrl-a**，`#`，**enter**）。这样做的话，之后借助命令行历史记录，你可以很方便恢复你刚才输入到一半的命令。
- 使用 `xargs`（或 `parallel`）。他们非常给力。注意到你可以控制每行参数个数（`-L`）和最大并行数（`-P`）。如果你不确定它们是否会按你想的那样工作，先使用 `xargs echo` 查看一下。此外，使用 `-I{}` 会很方便。例如：

```
find . -name '*.py' | xargs grep some_function
cat hosts | xargs -I{} ssh root@{} hostname
```

- `pstree -p` 以一种优雅的方式展示进程树。
- 使用 `pgrep` 和 `pkill` 根据名字查找进程或发送信号（`-f` 参数通常有用）。
- 了解你可以发往进程的信号的种类。比如，使用 `kill -STOP [pid]` 停止一个进程。使用 `man 7 signal` 查看详细列表。
- 使用 `nohup` 或 `disown` 使一个后台进程持续运行。
- 使用 `netstat -lntp` 或 `ss -plat` 检查哪些进程在监听端口（默认是检查 TCP 端口; 添加参数 `-u` 则检查 UDP 端口）或者 `lsof -iTCP -sTCP:LISTEN -P -n` (这也可以在 OS X 上运行)。
- `lsof` 来查看开启的套接字和文件。
- 使用 `uptime` 或 `w` 来查看系统已经运行多长时间。
- 使用 `alias` 来创建常用命令的快捷形式。例如：`alias ll='ls -latr'` 创建了一个新的命令别名 `ll`。
- 可以把别名、shell 选项和常用函数保存在 `~/.bashrc`，具体看下这篇文章。这样做的话你就可以在所有 shell 会话中使用你的设定。
- 把环境变量的设定以及登陆时要执行的命令保存在 `~/.bash_profile`。而对于从图形界面启动的 shell 和 `cron` 启动的 shell，则需要单独配置文件。
- 要想在几台电脑中同步你的配置文件（例如 `.bashrc` 和 `.bash_profile`），可以借助 Git。
- 当变量和文件名中包含空格的时候要格外小心。Bash 变量要用引号括起来，比如 `"$FOO"`。尽量使用 `-0` 或 `-print0` 选项以便用 NULL 来分隔文件名，例如 `locate -0 pattern | xargs -0 ls -al` 或 `find / -print0 -type d | xargs -0 ls -al`。如果 for 循环中循环访问的文件名含有空字符（空格、tab 等字符），只需用 `IFS=$'\n'` 把内部字段分隔符设为换行符。
- 在 Bash 脚本中，使用 `set -x` 去调试输出（或者使用它的变体 `set -v`，它会记录原始输入，包括多余的参数和注释）。尽可能地使用严格模式：使用 `set -e` 令脚本在发生错误时退出而不是继续运行；使用 `set -u` 来检查是否使用了未赋值的变量；试试 `set -o pipefail`，它可以监测管道中的错误。当牵扯到很多脚本时，使用 `trap` 来检测 ERR 和 EXIT。一个好的习惯是在脚本文件开头这样写，这会使它能够检测一些错误，并在错误发生时中断程序并输出信息：

```
set -euo pipefail
trap "echo 'error: Script failed: see failed command above'" ERR
```

- 在 Bash 脚本中，子 shell（使用括号 `(...)`）是一种组织参数的便捷方式。一个常见的例子是临时地移动工作路

径，代码如下：

```
# do something in current dir
(cd /some/other/dir && other-command)
# continue in original dir
```

- 在 Bash 中，变量有许多的扩展方式。`${name:?error message}` 用于检查变量是否存在。此外，当 Bash 脚本只需要一个参数时，可以使用这样的代码 `input_file=${1:?usage: $0 input_file}`。在变量为空时使用默认值：`${name:-default}`。如果你要在之前的例子中再加一个（可选的）参数，可以使用类似这样的代码 `output_file=${2:-logfile}`，如果省略了 `$2`，它的值就为空，于是 `output_file` 就会被设为 `logfile`。数学表达式：`i=$(( (i + 1) % 5 ))`。序列：`{1..10}`。截断字符串：`${var%suffix}` 和 `${var#prefix}`。例如，假设 `var=foo.pdf`，那么 `echo ${var%.pdf}.txt` 将输出 `foo.txt`。
- 使用括号扩展（{...}）来减少输入相似文本，并自动化文本组合。这在某些情况下会很有用，例如 `mv foo.{txt,pdf} some-dir`（同时移动两个文件），`cp somefile{,.bak}`（会被扩展成 `cp somefile somefile.bak`）或者 `mkdir -p test-{a,b,c}/subtest-{1,2,3}`（会被扩展成所有可能的组合，并创建一个目录树）。
- 通过使用 `<(some command)` 可以将输出视为文件。例如，对比本地文件 `/etc/hosts` 和一个远程文件：

```
diff /etc/hosts <(ssh somehost cat /etc/hosts)
```

- 编写脚本时，你可能会想要把代码都放在大括号里。缺少右括号的话，代码就会因为语法错误而无法执行。如果你的脚本是要放在网上分享供他人使用的，这样的写法就体现出它的好处了，因为这样可以防止下载不完全代码被执行。

```
{
    # 在这里写代码
}
```

- 了解 Bash 中的“here documents”，例如 `cat <<EOF ...`。
- 在 Bash 中，同时重定向标准输出和标准错误：`some-command >logfile 2>&1` 或者 `some-command &>logfile`。通常，为了保证命令不会在标准输入里残留一个未关闭的文件句柄捆绑在你当前所在的终端上，在命令后添加 `</dev/null` 是一个好习惯。
- 使用 `man ascii` 查看具有十六进制和十进制值的ASCII表。`man unicode`，`man utf-8`，以及 `man latin1` 有助于你去了解通用的编码信息。
- 使用 `screen` 或 `tmux` 来使用多份屏幕，当你在使用 ssh 时（保存 session 信息）将尤为有用。而 `byobu` 可以为它们提供更多的信息和易用的管理工具。另一个轻量级的 session 持久化解决方案是 `dtach`。
- ssh 中，了解如何使用 `-L` 或 `-D`（偶尔需要用 `-R`）开启隧道是非常有用的，比如当你需要从一台远程服务器上访问 web 页面。
- 对 ssh 设置做一些小优化可能是很有用的，例如这个 `~/.ssh/config` 文件包含了防止特定网络环境下连接断开、压缩数据、多通道等选项：

```
TCPKeepAlive=yes
ServerAliveInterval=15
ServerAliveCountMax=6
Compression=yes
ControlMaster auto
ControlPath /tmp/%r@%h:%p
ControlPersist yes
```

- 一些其他的关于 ssh 的选项是与安全相关的，应当小心翼翼的使用。例如你应当只能在可信任的网络中启用 `StrictHostKeyChecking=no`，`ForwardAgent=yes`。
- 考虑使用 `mosh` 作为 ssh 的替代品，它使用 UDP 协议。它可以避免连接被中断并且对带宽需求更小，但它需要在服务端做相应的配置。
- 获取八进制形式的文件访问权限（修改系统设置时通常需要，但 `ls` 的功能不那么好用并且通常会搞砸），可以使用类似如下的代码：

```
stat -c '%A %a %n' /etc/timezone
```

- 使用 `percol` 或者 `fzf` 可以交互式地从另一个命令输出中选取值。
- 使用 `fpp`（PathPicker）可以与基于另一个命令(例如 `git`）输出的文件交互。
- 将 web 服务器上当前目录下所有的文件（以及子目录）暴露给你所处网络的所有用户，使用：`python -m SimpleHTTPServer 7777`（使用端口 7777 和 Python 2）或`python -m http.server 7777`（使用端口 7777 和 Python 3）。
- 以其他用户的身份执行命令，使用 `sudo`。默认以 root 用户的身份执行；使用 `-u` 来指定其他用户。使用 `-i` 来以该用户登录（需要输入*你自己的*密码）。
- 将 shell 切换为其他用户，使用 `su username` 或者 `sudo - username`。加入 `-` 会使得切换后的环境与使用该用户登录后的环境相同。省略用户名则默认为 root。切换到哪个用户，就需要输入*哪个用户的*密码。
- 了解命令行的 128K 限制。使用通配符匹配大量文件名时，常会遇到“Argument list too long”的错误信息。（这种情况下换用 `find` 或 `xargs` 通常可以解决。）
- 当你需要一个基本的计算器时，可以使用 `python` 解释器（当然你要用 python 的时候也是这样）。例如：

```
>>> 2+3
5
```

# 文件及数据处理

- 在当前目录下通过文件名查找一个文件，使用类似于这样的命令：`find . -iname '*something*'`。在所有路径下通过文件名查找文件，使用 `locate something`（但注意到 `updatedb` 可能没有对最近新建的文件建立索引，所以你可能无法定位到这些未被索引的文件）。
- 使用 `ag` 在源代码或数据文件里检索（`grep -r` 同样可以做到，但相比之下 `ag` 更加先进）。
- 将 HTML 转为文本：`lynx -dump -stdin`。
- Markdown，HTML，以及所有文档格式之间的转换，试试 `pandoc`。
- 当你要处理棘手的 XML 时候，`xmlstarlet` 算是上古时代流传下来的神器。
- 使用 `jq` 处理 JSON。
- 使用 `shyaml` 处理 YAML。
- 要处理 Excel 或 CSV 文件的话，csvkit 提供了 `in2csv`，`csvcut`，`csvjoin`，`csvgrep` 等方便易用的工具。
- 当你要处理 Amazon S3 相关的工作的时候，`s3cmd` 是一个很方便的工具而 `s4cmd` 的效率更高。Amazon 官方提供的 `aws` 以及 `saws` 是其他 AWS 相关工作的基础，值得学习。
- 了解如何使用 `sort` 和 `uniq`，包括 uniq 的 `-u` 参数和 `-d` 参数，具体内容在后文单行脚本节中。另外可以了解一下 `comm`。
- 了解如何使用 `cut`，`paste` 和 `join` 来更改文件。很多人都会使用 `cut`，但遗忘了 `join`。
- 了解如何运用 `wc` 去计算新行数（`-l`），字符数（`-m`），单词数（`-w`）以及字节数（`-c`）。
- 了解如何使用 `tee` 将标准输入复制到文件甚至标准输出，例如 `ls -al | tee file.txt`。
- 要进行一些复杂的计算，比如分组、逆序和一些其他的统计分析，可以考虑使用 `datamash`。
- 注意到语言设置（中文或英文等）对许多命令行工具有一些微妙的影响，比如排序的顺序和性能。大多数 Linux

的安装过程会将 LANG 或其他有关的变量设置为符合本地的设置。要意识到当你改变语言设置时，排序的结果可能会改变。明白国际化可能会使 sort 或其他命令运行效率下降*许多倍*。某些情况下（例如集合运算）你可以放心的使用 `export LC_ALL=C` 来忽略掉国际化并按照字节来判断顺序。

- 你可以单独指定某一条命令的环境，只需在调用时把环境变量设定放在命令的前面，例如 `TZ=Pacific/Fiji date` 可以获取斐济的时间。
- 了解如何使用 awk 和 sed 来进行简单的数据处理。 参阅 One-liners 获取示例。
- 替换一个或多个文件中出现的字符串：

```
perl -pi.bak -e 's/old-string/new-string/g' my-files-*.txt
```

- 使用 repren 来批量重命名文件，或是在多个文件中搜索替换内容。（有些时候 rename 命令也可以批量重命名，但要注意，它在不同 Linux 发行版中的功能并不完全一样。）

```
# 将文件、目录和内容全部重命名 foo -> bar:
repren --full --preserve-case --from foo --to bar .
# 还原所有备份文件 whatever.bak -> whatever:
repren --renames --from '(.*)\.bak' --to '\1' *.bak
# 用 rename 实现上述功能（若可用）：
rename 's/\.bak$//' *.bak
```

- 根据 man 页面的描述，rsync 是一个快速且非常灵活的文件复制工具。它闻名于设备之间的文件同步，但其实它在本地情况下也同样有用。在安全设置允许下，用 rsync 代替 scp 可以实现文件续传，而不用重新从头开始。它同时也是删除大量文件的最快方法之一：

```
mkdir empty && rsync -r --delete empty/ some-dir && rmdir some-dir
```

- 若要在复制文件时获取当前进度，可使用 pv，pycp，progress，rsync --progress。若所执行的复制为block块拷贝，可以使用 dd status=progress。
- 使用 shuf 可以以行为单位来打乱文件的内容或从一个文件中随机选取多行。
- 了解 sort 的参数。显示数字时，使用 -n 或者 -h 来显示更易读的数（例如 du -h 的输出）。明白排序时关键字的工作原理（-t 和 -k）。例如，注意到你需要 -k1，1 来仅按第一个域来排序，而 -k1 意味着按整行排序。稳定排序（sort -s）在某些情况下很有用。例如，以第二个域为主关键字，第一个域为次关键字进行排序，你可以使用 `sort -k1, 1 | sort -s -k2, 2`。
- 如果你想在 Bash 命令行中写 tab 制表符，按下 **ctrl-v [Tab]** 或键入 `$'\t'`（后者可能更好，因为你可以复制粘贴它）。
- 标准的源代码对比及合并工具是 diff 和 patch。使用 diffstat 查看变更总览数据。注意到 diff -r 对整个文件夹有效。使用 `diff -r tree1 tree2 | diffstat` 查看变更的统计数据。vimdiff 用于比对并编辑文件。
- 对于二进制文件，使用 hd，hexdump 或者 xxd 使其以十六进制显示，使用 bvi，hexedit 或者 biew 来进行二进制编辑。
- 同样对于二进制文件，strings（包括 grep 等工具）可以帮助在二进制文件中查找特定比特。
- 制作二进制差分文件（Delta 压缩），使用 xdelta3。
- 使用 iconv 更改文本编码。需要更高级的功能，可以使用 uconv，它支持一些高级的 Unicode 功能。例如，这条命令移除了所有重音符号：

```
uconv -f utf-8 -t utf-8 -x '::Any-Lower; ::Any-NFD; [:Nonspacing Mark:] >; ::Any-NFC; ' < input.txt > output.txt
```

- 拆分文件可以使用 split（按大小拆分）和 csplit（按模式拆分）。

- 操作日期和时间表达式，可以用 dateutils 中的 dateadd、datediff、strptime 等工具。
- 使用 zless、zmore、zcat 和 zgrep 对压缩过的文件进行操作。
- 文件属性可以通过 chattr 进行设置，它比文件权限更加底层。例如，为了保护文件不被意外删除，可以使用不可修改标记：sudo chattr +i /critical/directory/or/file
- 使用 getfacl 和 setfacl 以保存和恢复文件权限。例如：

```
getfacl -R /some/path > permissions.txt
setfacl --restore=permissions.txt
```

- 为了高效地创建空文件，请使用 truncate（创建稀疏文件），fallocate（用于 ext4，xfs，btrf 和 ocfs2 文件系统），xfs_mkfile（适用于几乎所有的文件系统，包含在 xfsprogs 包中），mkfile（用于类 Unix 操作系统，比如 Solaris 和 Mac OS）。

## 系统调试

- curl 和 curl -I 可以被轻松地应用于 web 调试中，它们的好兄弟 wget 也是如此，或者也可以试试更潮的 httpie。
- 获取 CPU 和硬盘的使用状态，通常使用使用 top（htop 更佳），iostat 和 iotop。而 iostat -mxz 15 可以让你获悉 CPU 和每个硬盘分区的基本信息和性能表现。
- 使用 netstat 和 ss 查看网络连接的细节。
- dstat 在你想要对系统的现状有一个粗略的认识时是非常有用的。然而若要对系统有一个深度的总体认识，使用 glances，它会在一个终端窗口中向你提供一些系统级的数据。
- 若要了解内存状态，运行并理解 free 和 vmstat 的输出。值得留意的是“cached”的值，它指的是 Linux 内核用来作为文件缓存的内存大小，而与空闲内存无关。
- Java 系统调试则是一件截然不同的事，一个可以用于 Oracle 的 JVM 或其他 JVM 上的调试的技巧是你可以运行 kill -3 <pid> 同时一个完整的栈轨迹和堆概述（包括 GC 的细节）会被保存到标准错误或是日志文件。JDK 中的 jps，jstat，jstack，jmap 很有用。SJK tools 更高级。
- 使用 mtr 去跟踪路由，用于确定网络问题。
- 用 ncdu 来查看磁盘使用情况，它比寻常的命令，如 du -sh *，更节省时间。
- 查找正在使用带宽的套接字连接或进程，使用 iftop 或 nethogs。
- ab 工具（Apache 中自带）可以简单粗暴地检查 web 服务器的性能。对于更复杂的负载测试，使用 siege。
- wireshark，tshark 和 ngrep 可用于复杂的网络调试。
- 了解 strace 和 ltrace。这俩工具在你的程序运行失败、挂起甚至崩溃，而你却不知道为什么或你想对性能有个总体的认识的时候是非常有用的。注意 profile 参数（-c）和附加到一个运行的进程参数 （-p）。
- 了解使用 ldd 来检查共享库。但是永远不要在不信任的文件上运行。
- 了解如何运用 gdb 连接到一个运行着的进程并获取它的堆栈轨迹。
- 学会使用 /proc。它在调试正在出现的问题的时候有时会效果惊人。比如：/proc/cpuinfo，/proc/meminfo，/proc/cmdline，/proc/xxx/cwd，/proc/xxx/exe，/proc/xxx/fd/，/proc/xxx/smaps（这里的 xxx 表示进程的 id 或 pid）。
- 当调试一些之前出现的问题的时候，sar 非常有用。它展示了 cpu、内存以及网络等的历史数据。
- 关于更深层次的系统分析以及性能分析，看看 stap（SystemTap），perf，以及sysdig。
- 查看你当前使用的系统，使用 uname，uname -a（Unix / kernel 信息）或者 lsb_release -a（Linux 发行版信息）。
- 无论什么东西工作得很欢乐（可能是硬件或驱动问题）时可以试试 dmesg。
- 如果你删除了一个文件，但通过 du 发现没有释放预期的磁盘空间，请检查文件是否被进程占用：lsof | grep deleted | grep "filename-of-my-big-file"

# 单行脚本

一些命令组合的例子：

- 当你需要对文本文件做集合交、并、差运算时，`sort` 和 `uniq` 会是你的好帮手。具体例子请参照代码后面的，此处假设 `a` 与 `b` 是两内容不同的文件。这种方式效率很高，并且在小文件和上 G 的文件上都能运用（注意尽管在 `/tmp` 在一个小的根分区上时你可能需要 `-T` 参数，但是实际上 `sort` 并不被内存大小约束），参阅前文中关于 `LC_ALL` 和 `sort` 的 `-u` 参数的部分。

```
sort a b | uniq > c   # c 是 a 并 b
sort a b | uniq -d > c   # c 是 a 交 b
sort a b b | uniq -u > c   # c 是 a - b
```

- 使用 `grep . *`（每行都会附上文件名）或者 `head -100 *`（每个文件有一个标题）来阅读检查目录下所有文件的内容。这在检查一个充满配置文件的目录（如 `/sys`、`/proc`、`/etc`）时特别好用。
- 计算文本文件第三列中所有数的和（可能比同等作用的 Python 代码快三倍且代码量少三倍）：

```
awk '{ x += $3 } END { print x }' myfile
```

- 如果你想在文件树上查看大小/日期，这可能看起来像递归版的 `ls -l` 但比 `ls -lR` 更易于理解：

```
find . -type f -ls
```

- 假设你有一个类似于 web 服务器日志文件的文本文件，并且一个确定的值只会出现在某些行上，假设一个 `acct_id` 参数在 URI 中。如果你想计算出每个 `acct_id` 值有多少次请求，使用如下代码：

```
egrep -o 'acct_id=[0-9]+' access.log | cut -d= -f2 | sort | uniq -c | sort -rn
```

- 要持续监测文件改动，可以使用 `watch`，例如检查某个文件夹中文件的改变，可以用 `watch -d -n 2 'ls -rtlh | tail'`；或者在排查 WiFi 设置故障时要监测网络设置的更改，可以用 `watch -d -n 2 ifconfig`。
- 运行这个函数从这篇文档中随机获取一条技巧（解析 Markdown 文件并抽取项目）：

```
function taocl() {
  curl -s https://raw.githubusercontent.com/jlevy/the-art-of-command-
line/master/README-zh.md|
    pandoc -f markdown -t html |
    iconv -f 'utf-8' -t 'unicode' |
    xmlstarlet fo --html --dropdtd |
    xmlstarlet sel -t -v "(html/body/ul/li[count(p)>0])[$RANDOM mod last()+1]"
|
    xmlstarlet unesc | fmt -80
}
```

# 冷门但有用

- `expr`：计算表达式或正则匹配
- `m4`：简单的宏处理器
- `yes`：多次打印字符串
- `cal`：漂亮的日历
- `env`：执行一个命令（脚本文件中很有用）
- `printenv`：打印环境变量（调试时或在写脚本文件时很有用）
- `look`：查找以特定字符串开头的单词或行
- `cut`，`paste` 和 `join`：数据修改
- `fmt`：格式化文本段落
- `pr`：将文本格式化成页 / 列形式
- `fold`：包裹文本中的几行
- `column`：将文本格式化成多个对齐、定宽的列或表格
- `expand` 和 `unexpand`：制表符与空格之间转换
- `nl`：添加行号
- `seq`：打印数字
- `bc`：计算器
- `factor`：分解因数
- `gpg`：加密并签名文件
- `toe`：terminfo 入口列表
- `nc`：网络调试及数据传输
- `socat`：套接字代理，与 `netcat` 类似
- `slurm`：网络流量可视化
- `dd`：文件或设备间传输数据
- `file`：确定文件类型
- `tree`：以树的形式显示路径和文件，类似于递归的 `ls`
- `stat`：文件信息
- `time`：执行命令，并计算执行时间
- `timeout`：在指定时长范围内执行命令，并在规定时间结束后停止进程
- `lockfile`：使文件只能通过 `rm -f` 移除
- `logrotate`：切换、压缩以及发送日志文件
- `watch`：重复运行同一个命令，展示结果并 / 或高亮有更改的部分
- `when-changed`：当检测到文件更改时执行指定命令。参阅 `inotifywait` 和 `entr`。
- `tac`：反向输出文件
- `shuf`：文件中随机选取几行
- `comm`：一行一行的比较排序过的文件
- `strings`：从二进制文件中抽取文本
- `tr`：转换字母
- `iconv` 或 `uconv`：文本编码转换
- `split` 和 `csplit`：分割文件
- `sponge`：在写入前读取所有输入，在读取文件后再向同一文件写入时比较有用，例如 `grep -v something some-file | sponge some-file`
- `units`：将一种计量单位转换为另一种等效的计量单位（参阅 `/usr/share/units/definitions.units`）
- `apg`：随机生成密码
- `xz`：高比例的文件压缩
- `ldd`：动态库信息
- `nm`：提取 obj 文件中的符号
- `ab` 或 `wrk`：web 服务器性能分析
- `strace`：调试系统调用
- `mtr`：更好的网络调试跟踪工具

- `cssh`：可视化的并发 shell
- `rsync`：通过 ssh 或本地文件系统同步文件和文件夹
- `wireshark` 和 `tshark`：抓包和网络调试工具
- `ngrep`：网络层的 grep
- `host` 和 `dig`：DNS 查找
- `lsof`：列出当前系统打开文件的工具以及查看端口信息
- `dstat`：系统状态查看
- `glances`：高层次的多子系统总览
- `iostat`：硬盘使用状态
- `mpstat`：CPU 使用状态
- `vmstat`：内存使用状态
- `htop`：top 的加强版
- `last`：登入记录
- `w`：查看处于登录状态的用户
- `id`：用户/组 ID 信息
- `sar`：系统历史数据
- `iftop` 或 `nethogs`：套接字及进程的网络利用情况
- `ss`：套接字数据
- `dmesg`：引导及系统错误信息
- `sysctl`：在内核运行时动态地查看和修改内核的运行参数
- `hdparm`：SATA/ATA 磁盘更改及性能分析
- `lsblk`：列出块设备信息：以树形展示你的磁盘以及磁盘分区信息
- `lshw`，`lscpu`，`lspci`，`lsusb` 和 `dmidecode`：查看硬件信息，包括 CPU、BIOS、RAID、显卡、USB设备等
- `lsmod` 和 `modinfo`：列出内核模块，并显示其细节
- `fortune`，`ddate` 和 `sl`：额，这主要取决于你是否认为蒸汽火车和莫名其妙的名人名言是否“有用”

# 仅限 OS X 系统

以下是*仅限于* OS X 系统的技巧。

- 用 `brew`（Homebrew）或者 `port`（MacPorts）进行包管理。这些可以用来在 OS X 系统上安装以上的大多数命令。
- 用 `pbcopy` 复制任何命令的输出到桌面应用，用 `pbpaste` 粘贴输入。
- 若要在 OS X 终端中将 Option 键视为 alt 键（例如在上面介绍的 **alt-b**、**alt-f** 等命令中用到），打开 偏好设置 -> 描述文件 -> 键盘 并勾选“使用 Option 键作为 Meta 键”。
- 用 `open` 或者 `open -a /Applications/Whatever.app` 使用桌面应用打开文件。
- Spotlight：用 `mdfind` 搜索文件，用 `mdls` 列出元数据（例如照片的 EXIF 信息）。
- 注意 OS X 系统是基于 BSD UNIX 的，许多命令（例如 `ps`，`ls`，`tail`，`awk`，`sed`）都和 Linux 中有微妙的不同（Linux 很大程度上受到了 System V-style Unix 和 GNU 工具影响）。你可以通过标题为 "BSD General Commands Manual" 的 man 页面发现这些不同。在有些情况下 GNU 版本的命令也可能被安装（例如 `gawk` 和 `gsed` 对应 GNU 中的 awk 和 sed）。如果要写跨平台的 Bash 脚本，避免使用这些命令（例如，考虑 Python 或者 `perl`）或者经过仔细的测试。
- 用 `sw_vers` 获取 OS X 的版本信息。

# 仅限 Windows 系统

以下是*仅限于*Windows 系统的技巧。

## *在 Winodws 下获取 Unix 工具*

- 可以安装 Cygwin 允许你在 Microsoft Windows 中体验 Unix shell 的威力。这样的话，本文中介绍的大多数内容都将适用。
- 在 Windows 10 上，你可以使用 Bash on Ubuntu on Windows，它提供了一个熟悉的 Bash 环境，包含了不少 Unix 命令行工具。好处是它允许 Linux 上编写的程序在 Windows 上运行，而另一方面，Windows 上编写的程序却无法在 Bash 命令行中运行。
- 如果你在 Windows 上主要想用 GNU 开发者工具（例如 GCC），可以考虑 MinGW 以及它的 MSYS 包，这个包提供了例如 bash，gawk，make 和 grep 的工具。MSYS 并不包含所有可以与 Cygwin 媲美的特性。当制作 Unix 工具的原生 Windows 端口时 MinGW 将特别地有用。
- 另一个在 Windows 下实现接近 Unix 环境外观效果的选项是 Cash。注意在此环境下只有很少的 Unix 命令和命令行可用。

## *实用 Windows 命令行工具*

- 可以使用 `wmic` 在命令行环境下给大部分 Windows 系统管理任务编写脚本以及执行这些任务。
- Windows 实用的原生命令行网络工具包括 `ping`，`ipconfig`，`tracert`，和 `netstat`。
- 可以使用 `Rundll32` 命令来实现许多有用的 Windows 任务 。

## *Cygwin 技巧*

- 通过 Cygwin 的包管理器来安装额外的 Unix 程序。
- 使用 `mintty` 作为你的命令行窗口。
- 要访问 Windows 剪贴板，可以通过 `/dev/clipboard`。
- 运行 `cygstart` 以通过默认程序打开一个文件。
- 要访问 Windows 注册表，可以使用 `regtool`。
- 注意 Windows 驱动器路径 `C:\` 在 Cygwin 中用 `/cygdrive/c` 代表，而 Cygwin 的 `/` 代表 Windows 中的 `C:\cygwin`。要转换 Cygwin 和 Windows 风格的路径可以用 `cygpath`。这在需要调用 Windows 程序的脚本里很有用。
- 学会使用 `wmic`，你就可以从命令行执行大多数 Windows 系统管理任务，并编成脚本。
- 要在 Windows 下获得 Unix 的界面和体验，另一个办法是使用 Cash。需要注意的是，这个环境支持的 Unix 命令和命令行参数非常少。
- 要在 Windows 上获取 GNU 开发者工具（比如 GCC）的另一个办法是使用 MinGW 以及它的 MSYS 软件包，该软件包提供了 bash、gawk、make、grep 等工具。然而 MSYS 提供的功能没有 Cygwin 完善。MinGW 在创建 Unix 工具的 Windows 原生移植方面非常有用。

## 更多资源

- awesome-shell：一份精心组织的命令行工具及资源的列表。
- awesome-osx-command-line：一份针对 OS X 命令行的更深入的指南。
- Strict mode：为了编写更好的脚本文件。
- shellcheck：一个静态 shell 脚本分析工具，本质上是 bash / sh / zsh 的 lint。
- Filenames and Pathnames in Shell：有关如何在 shell 脚本里正确处理文件名的细枝末节。
- Data Science at the Command Line：用于数据科学的一些命令和工具，摘自同名书籍。

# 免责声明

除去特别小的工作，你编写的代码应当方便他人阅读。能力往往伴随着责任，你 *有能力* 在 Bash 中玩一些奇技淫巧并不意味着你应该去做！;)

# 授权条款