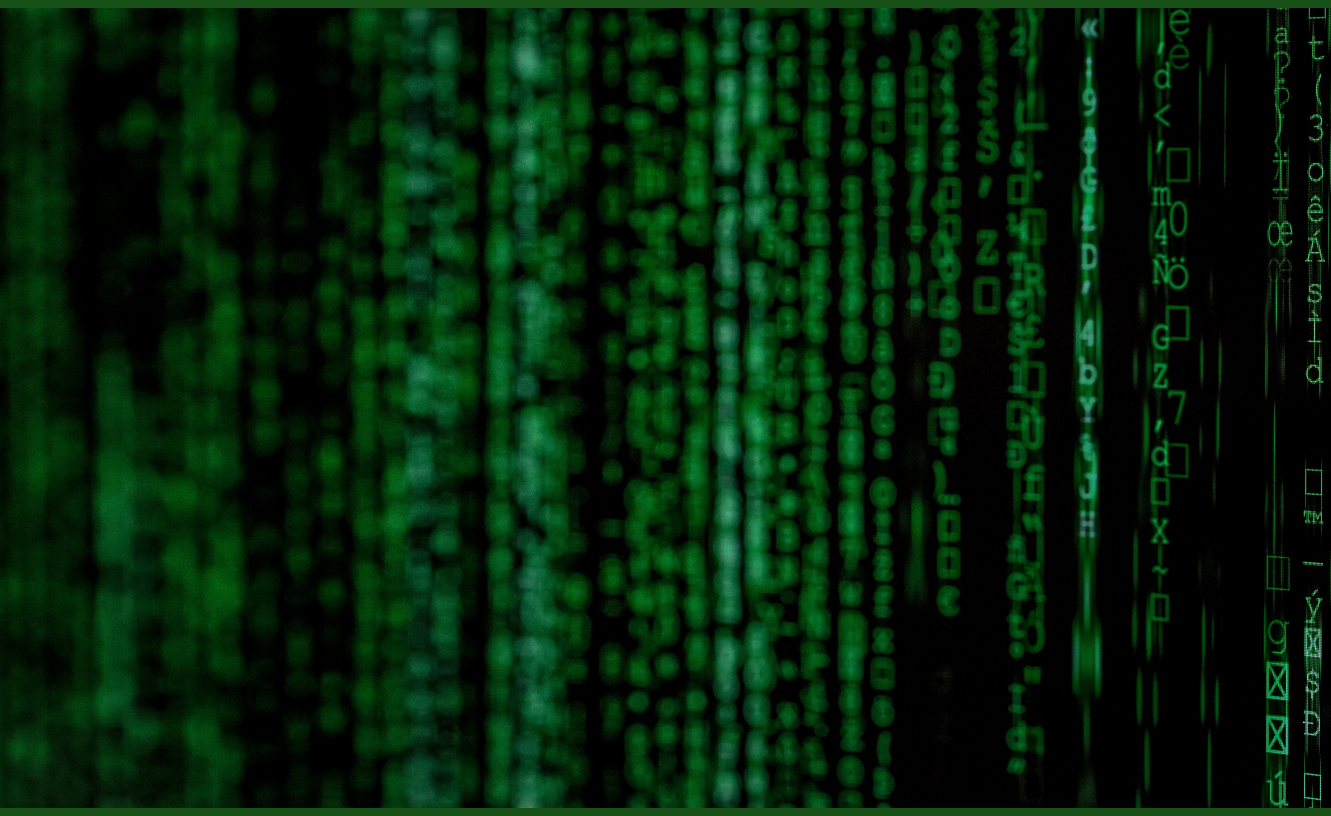
像黑客一样使用命令行

命令行高手修炼之道



徐小东

像黑客一样使用命令行

徐小东

献给

海燕和铭基

目录

表格

插图

致谢

感谢 Bash 及 Zsh 开源社区，你们永远是最棒的家伙！

更新

你可以从 https://selfhostedserver.com/usingcli-book 获取本书的更新版本。另外，本书也包括视频版本，请通过 https://selfhostedserver.com/usingcli 了解详情。

- Version 2019.3.17：初版
- Version 2019.8.24：修订版，增加 “高效查询 Shell 历史：HSTR” 小节。

作者简介

徐小东，网名 ~toy。GNU/Linux 爱好者，DevOps 践行者。喜技术，好分享。通过 https://linuxtoy.org 网站数年间原创及翻译文章达 3000 余篇。另著有《像黑客一样使用命令行》[1]、《容器化工具三剑客：Podman、Buildah 和 Skopeo》[2]、《Terraform：自动化管理云基础设施》[3]，译有《笨办法学 Git》[4]、《Perl 程序员应该知道的事》等图书。Twitter：https://twitter.com/linuxtoy，Mail：xuxiaodong@pm.me[5]。

[1] https://selfhostedserver.com/usingcli-book

[2] https://selfhostedserver.com/nextcontainer

[3] https://selfhostedserver.com/terraform

[4] https://selfhostedserver.com/learngit

[5] mailto:xuxiaodong@pm.me

第一章　入门指引

虽然如今计算机图形化界面大行其道，然而在计算机诞生之初却是命令行界面的天下。在图形化界面中，我们惯常使用鼠标来操作图标或窗口，从而完成各种任务。对于命令行界面来说，情况则有很大的不同。要在命令行界面下执行操作，我们需要更多的依赖键盘。那么，什么是命令行界面呢？在回答这个问题之前，不妨让我们先来谈谈控制台、终端、终端模拟器、以及 Shell 这几个基本概念。

1.1　控制台

控制台（Console），又称为系统控制台（System console）、计算机控制台（Computer console）、根控制台（Root console）、以及操作员控制台（Operator's console）。事实上，早先的控制台是一种用来操作计算机的硬件，如图 1.1 所示。[1]从这幅图片中，我们可以看到 IBM 1620 计算机的控制台由左边的操作前面板（参考图 1.2）和右边的打字机组成。通过控制台，操作员将文本数据或待执行的指令录入到计算机，并最终通过计算机读取或执行。

随着计算机的发展，控制台从硬件概念变成了一个软件概念。于是，控制台有了新的称呼：虚拟控制台。虚拟控制台正好与物理的控制台硬件相对。通过观察 Linux 系统的启动过程，我们不难发现：在经过计算机硬件自检之后，一旦由引导载入程序接管，不一会儿便会进入系统控制台。在这个过程中，通常会显示如图 1.3 所示的 Linux 系统引导信息。[2]

[1] https://en.wikipedia.org/wiki/System_console#/media/File:IBM_1620_Model_1.jpg

[2] https://en.wikipedia.org/wiki/Linux_console#/media/File:Knoppix-3.8-boot.png

图 1.1: IBM 1620 的控制台

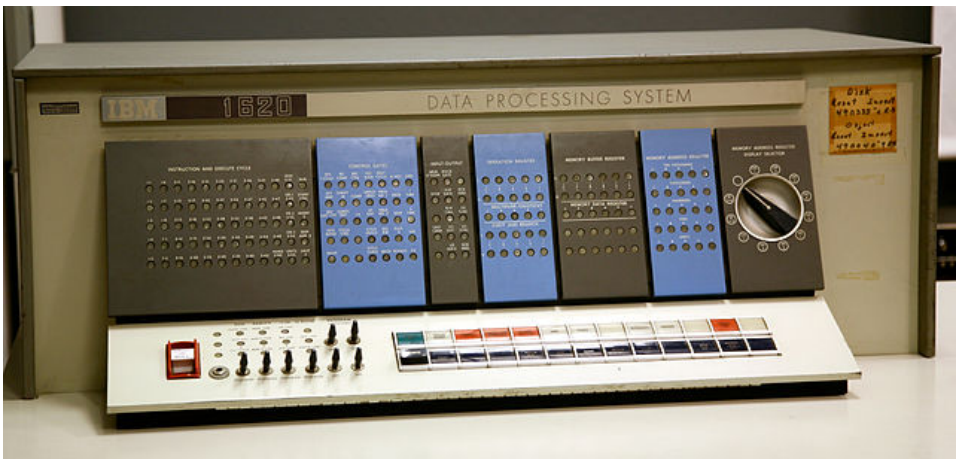


图 1.2: IBM 1620 控制台的操作前面板

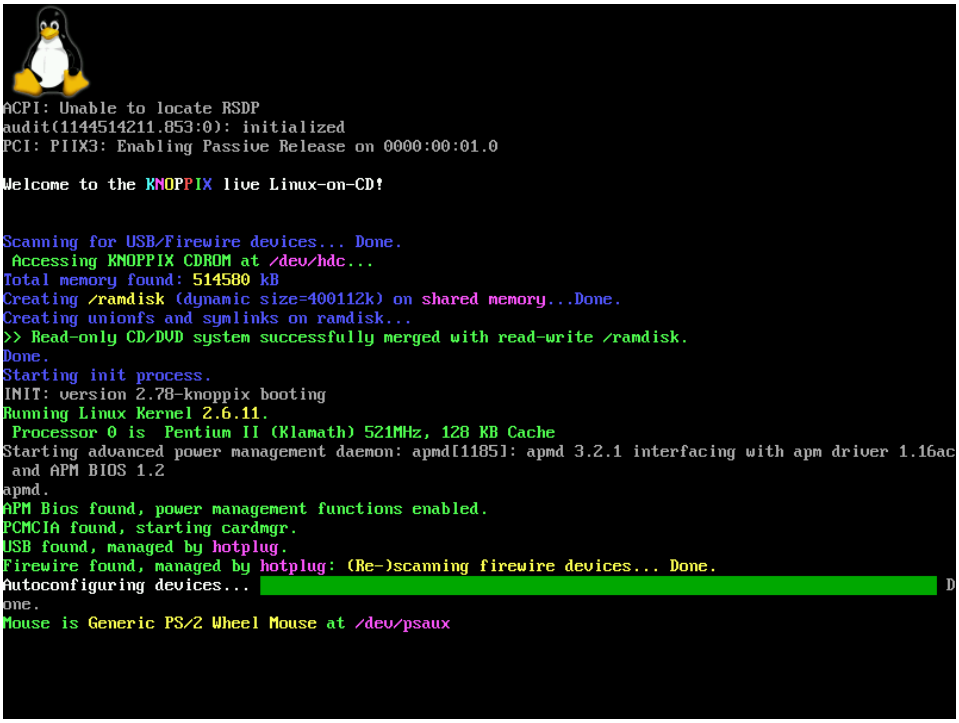


图 1.3: Linux 系统虚拟控制台

1.2 终端

跟控制台一样，起初的终端（Terminal）也是一种计算机硬件设备。从外形上看，终端类似于我们今天所看到的显示器和键盘的结合体。通过终端，用户将指令和数据输入到计算机。同时，终端也将计算机执行的结果展示给用户。图 1.4 中显示的是曾经广为流行的终端 DEC VT100。[3]

或许你会产生疑问，为什么会出现终端这种硬件设备呢？以今天的眼光来看，显得似乎有些难以理解。诞生之初的计算机造价相当昂贵，可不像现在人人都能拥有一台那么简单。除了大型商业组织或大学研究机构，很难在别处看到计算机的身影。为了能够共享计算机资源，终端应运而生。然而，伴随着科技的进步，终端最终掉进了历史的黑洞。不过，它后来却以新的形式重生，这就是终端模拟器（Terminal emulator），或称之为虚拟终端。

[3] https://en.wikipedia.org/wiki/Computer_terminal#/media/File:DEC_VT100_terminal.jpg

图 1.4: DEC VT100 终端

1.3 终端模拟器

终端模拟器，即用来模拟终端硬件设备的应用程序。在物理终端中存在的某些显示体系结构，比如用来控制色彩的转义序列、光标位置等在终端模拟器中也得到了支持。图 1.5 显示 Linux 中流行的终端程序之一 XTerm。

不管是 Linux 操作系统，还是 macOS 操作系统，乃至 Windows 操作系统，今天都有许多终端模拟器可以选择。以下罗列的是这三个操作系统中比较流行的终端模拟器。

图 1.5: XTerm 终端模拟器

1.3.1 Linux

- XTerm[4]：XTerm 是 X 窗口环境的默认终端。它提供了与 DEC VT102 和 Tektronix 4014 终端兼容的特性。此外，它也支持 ISO/ANSI 彩色模式。
- GNOME Terminal[5]：GNOME Terminal 是 GNOME 桌面环境的默认终端。它提供了与 XTerm 相似的特性。除此之外，它也包括支持多配置、标签页、鼠标事件等其它功能。
- Konsole[6]：Konsole 是 KDE 桌面环境的默认终端。它包括标签页、多配置、书签支持、搜索等特性。
- rxvt-unicode[7]：rxvt-unicode 原本克隆自 rxvt，但加入了 unicode 支持，具有很强的定制特性。另外，rxvt-unicode 还包含 Daemon 模式、嵌入了 Perl 编程语言等功能。本书作者使用的就是这款终端模拟器。

[4] https://invisible-island.net/xterm/

[5] https://gitlab.gnome.org/GNOME/gnome-terminal/

[6] https://kde.org/applications/system/konsole/

[7] http://software.schmorp.de/pkg/rxvt-unicode.html

1.3.2 macOS

- Terminal.app：Terminal.app 是 macOS 操作系统默认的终端。它的功能不多，除了提供设置 `TERM` 环境变量的选项外，还包括能够使用其搜索功能来查找 Man pages。
- iTerm2[8]：iTerm2 是 macOS 系统上针对默认终端的开源替代品。它非常流行，包含许多很棒的功能，比如窗口分割、自动补全、无鼠拷贝、粘贴历史等等。如果你在 macOS 上工作，那么不妨使用 iTerm2 这款终端模拟器，相信它所具有的功能一定不会让你失望。

1.3.3 Windows

- Mintty[9]：Mintty 是一个支持 Cygwin、MSYS、WSL 等多种环境的终端模拟器。它的功能与 XTerm 兼容，包括 256 色和真彩色、unicode、以及 Emoji 表情支持。
- ConEmu[10]：ConEmu 是 Windows 上一款相当流行的开源终端模拟器。它包含标签页、多种图形窗口模式、用户友好的文本块选择等功能。

1.4 Shell

Shell 是一种命令解释程序，它负责用户输入命令的读取、解析和执行。现代 Shell 除了具有与用户直接交互的特性之外，通常也包含编程功能，支持变量、数组、函数、循环、条件等编程基本要素。

Shell 之所以如此称呼，是由于它相对 Unix 及 Linux 的核心——内核（Kernel）而言，处于整个操作系统的最外层，就像乌龟的壳一样。也正因为如此，Shell 提供用来访问系统服务的用户界面，扮演着与内核交互的角色，如图 1.6 所示。

在 Unix 及 Linux 的发展过程中，出现了许多种 Shell，其中比较知名的包括：sh、csh、ksh、bash、zsh 等等。

[8] https://www.iterm2.com/
[9] https://mintty.github.io/
[10] https://conemu.github.io/

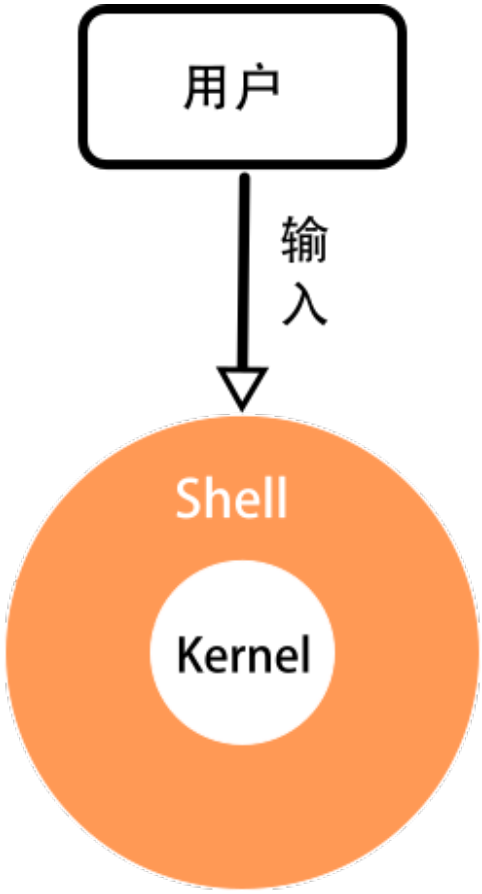


图 1.6: Shell 与内核

1.4.1 sh

sh，即 Bourne shell，它是 Unix 第 7 版的默认 Shell。Bourne shell 由贝尔实验室的 Stephen Bourne 开发，于 1979 年发布。随着《Unix 编程环境》（Brian Kernighan 与 Rob Pike 著）一书的出版，sh 变得大为流行。

Bourne shell 早已被后来的 Shell 所取代，现代 Linux 系统中的 sh 通常是符号链接的某个兼容 Shell。例如，本书作者所用的 Debian 9 里的 sh 为 dash。

```
root@toydroid:~# ls -l /bin/sh
lrwxrwxrwx 1 root root 4 Jan 24  2017 /bin/sh -> dash
```

而在作者的另一个系统 Arch Linux 上，sh 则为 bash。

```
root@codeland:~# ls -l /bin/sh
lrwxrwxrwx 1 root root 4 Feb  7 15:15 /bin/sh -> bash
```

1.4.2 csh

csh 是 C shell 的简称，它由 Bill Joy 开发，通过 BSD 得到了广泛的分发。在风格上，开发者将 csh 设计得像 C 编程语言一样，因而由此得名。同时，csh 具有很不错的交互使用体验。后来被其它 Shell 所吸收的诸如历史、别名、目录栈、文件名补全、作业控制等特性均出自 csh。

csh 有一个改进版本叫 tcsh，目前是 FreeBSD 的默认 Shell。

1.4.3 ksh

ksh 指 Korn Shell，其开发者为 David Korn，在 1983 年公布于世。ksh 遵循 POSIX 标准，能够向下兼容 Bourne shell，整合了来自 C shell 的诸多特性。ksh 的一大亮点是引入了 vi 和 Emacs 风格的命令行编辑模式，使用户完全可以按照自己的按键习惯操作。此外，在 ksh 中还增加了关联数组的特性。

由于 ksh 最初以私有软件的形式进行分发，从而被限制了传播。代之以出现的替代品包括 pdksh（public domain ksh，公有域的 ksh）、mksh（后成为 Android 的默认 Shell）等。

1.4.4 bash

bash 作为理查德 · 斯托曼 GNU 工程的一部分出现，从它诞生之初就是为了用来取代 Bourne shell（参考 1.4.1 节）。Brian Fox 开发了最初的 bash，首个版本发布于 1989 年。如今，bash 已变得十分流行，它是大多数 Linux 发行版以及 macOS 的默认 Shell。此外，通过 WSL（Windows Subsystem for Linux），在 Windows 10 中也可以安装并使用 bash。

bash 的名称来自于 **B**ourne-**a**gain **sh**ell，它也遵循 POSIX 标准，其特性吸收自 sh、csh、ksh 等多种 Shell。

1.4.5 zsh

zsh 是 Z shell 的简称，最初的版本由 Paul Falstad 所开发，发布于 1990 年。zsh 极大的扩展了 Bourne shell 的功能，并包含来自 tcsh、ksh、bash 等 Shell 的特性。

在交互用户体验上，zsh 尤其出彩。比如，它支持对命令的选项进行补全、可以设置右提示符等，如图 1.7 所示。

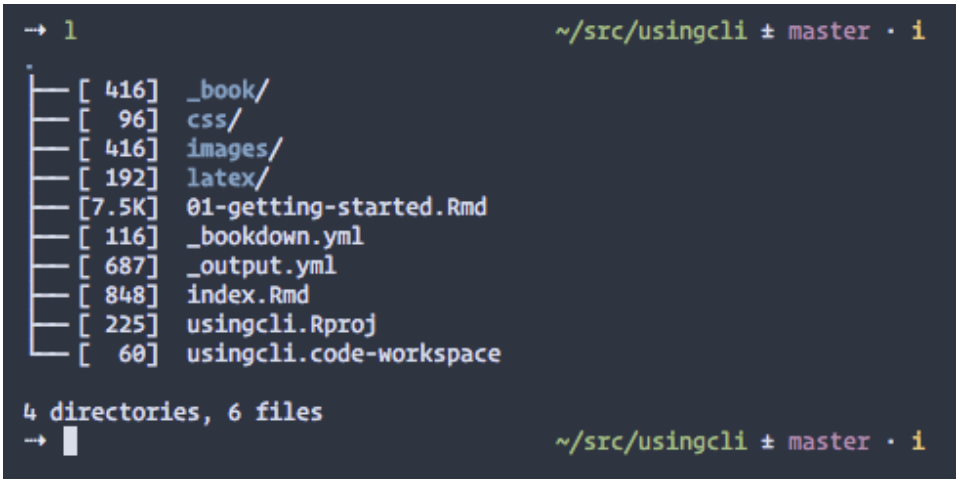


图 1.7: zsh 的右提示符

本书主要讨论 bash 和 zsh 这两种目前市面上最流行的 Shell。

1.5 命令行界面

命令行界面（Command-line interface），经常缩写为 CLI，亦即用户输入命令的地方。一旦用户将命令输入完毕并加以提交后，后续对命令的解析以及执行的任务都由 Shell 来完成。

与 CLI 相对的是 GUI，即 Graphical user interface，意为图形用户界面，它采用图形化的方式让用户与计算机进行交互。因其具有容易使用的优点，包括 Linux、macOS、Windows 等在内的现代操作系统无一例外都提供了图形用户界面。

既然图形用户界面要比命令行界面更加易用，那么是否说明可以完全抛弃命令

行界面呢？答案是并非如此。事实上，有经验的用户尤其擅长使用命令行界面，其理由至少包括以下几个方面。

1.5.1 功能强大

让我们先来看一个例子：

```
xiaodong@codeland:~$ history |
awk '{CMD[$2]++;count++;}END \
{ for (a in CMD)print CMD[a] " " \
CMD[a]/count*100 "% " a;}' |
grep -v "./" |
column -c3 -s " " -t |
sort -nr |
nl |
head -n10
```

在作者的 macOS 系统上执行这条命令后，其输出结果如下：

```
 1  1348  14.3771%  cd
 2  1034  11.0282%  l
 3  838   8.93771%  git
 4  569   6.06869%  ssh
 5  513   5.47142%  cat
 6  405   4.31954%  vim
 7  372   3.96758%  brew
 8  360   3.83959%  scp
 9  265   2.82637%  rm
10  264   2.8157%   grep
```

这条命令虽然看起来似乎有些“吓人”，因为它由 history、awk、grep、column、sort、nl、head 等 7 个命令组成，并通过管道符（|）串接在一起；然而其结果却颇为有趣。它将作者平时在命令行中执行的所有命令都进行了统计，最终展示出 10 个最常用的命令，并相应列出每个命令的使用次数和所占百分比。

管道符将前一命令的输出作为后一命令的输入，使这些表面上不相干的命令进行协同工作，犹如搭积木一般。这是命令行的真正威力所在。

1.5.2 灵活高效

再看另一个例子，假如我们打算从 photos 目录中找出今年三月份拍摄的照片，并将其文件名称保存到 mar_photos.txt 这个文本文件中。在图形用户界面中，首先，我们可能会打开一个文件管理器（在 Linux 下也许是 GNOME Files，macOS 中则是 Finder）。接着，导航到 photos 这个目录，同时切换成详细视图模式。然后，我们睁大双眼逐一找出符合要求的照片。可是，现在怎么把照片的文件名称写到文本文件中呢？我们当然可以直接输入，或者想省点力使用复制和粘贴也行。要是找出的文件数量比较多，那可绝对是体力活。

但是，如果在命令行下，那么我们只需通过执行一条命令即可达到目的：

```
xiaodong@codeland:~$ cd photos; \
ls -l | grep 'Mar' | awk '{ print $9 }' > mar_photos.txt
```

1.5.3 能自动化

使用命令行还有一个很棒的优势，那就是能够自动化各种操作。Shell 允许我们将所用的命令编写成函数（Function）或脚本（Script）。这样，我们不仅可以反复执行它们，而且函数或脚本比手动输入效率更高。由此，我们得以从重复的劳动中解放出来，从而能够腾出时间去做其它有意义的事情。

```
xiaodong@codeland:~$ ./script.sh
```

1.6 如何进入命令行

通过前面的描述，现在你应当了解：我们想要输入命令的界面是由 Shell 提供的。那么，如何执行 Shell 呢？我们可以通过下面两种方法来进入命令行。

1.6.1 通过控制台进入命令行

为了节省系统资源，Linux 服务器通常没有附带图形用户界面。当它启动完毕时，在控制台按照提示输入用户帐号及密码并登录后，所进入的即是命令行界面。以下为 Linux 服务器的登录提示：

```
login:
Password:
```

作为普通用户来说，一般使用的是具有图形用户界面的 Linux 桌面系统。在它启动后就直接进入了桌面，那么此时想要进入控制台，可以按照下列步骤执行：

1. 按 Ctrl + Alt + F1 组合键，进入编号为 1 的控制台。
2. 按 Ctrl + Alt + F2 组合键，进入编号为 2 的控制台。
3. 依次类推，可以分别进入 3 号、4 号、5 号、以及 6 号控制台。在默认情况下，Linux 一般提供 6 个控制台。
4. 如果要从控制台返回到桌面，则可以按 Ctrl + Alt + F7 组合键。

1.6.2 通过终端模拟器进入命令行

另外一种进入命令行界面的方法是使用终端模拟器。在不同的操作系统中，可以选择的终端模拟器程序也有所不同（参考 1.3 节）。本书作者在 Linux 下常用 rxvt-unicode，macOS 中则使用 iTerm2。

一般而言，终端模拟器程序会跟系统的登录 Shell（或称默认 Shell）绑定在一起。有些终端模拟器程序提供了更改 Shell 的特性，从而使用户可以方便的选择自己惯用的 Shell。如果不能从终端程序中直接更改 Shell，那么也可以通过 `chsh` 命令来改变登录 Shell。假如我们想把默认 Shell 更改成 zsh，则可以执行以下命令：

```
xiaodong@codeland:~$ chsh -s /bin/zsh
```

1.7 你好，命令行

在《C 程序设计语言》中，作者 Brian W. Kernighan 和 Dennis M. Ritchie 介绍的第一个程序是在屏幕上输出一行 “Hello world” 的消息。为了说明命令行的使用，我们也将在屏幕上输出类似的消息——“你好，命令行”。

当我们进入控制台或打开终端模拟器时，通常会看到跟图 1.8 相似的命令行界面。

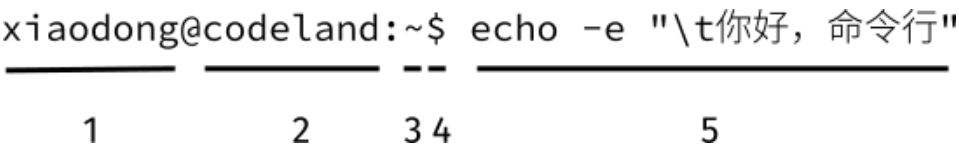


图 1.8: 命令行界面

从图 1.8 中我们可以看到命令行一般由下面几个部分组成：

1. 当前登录的用户名称，在本例中是 `xiaodong`。
2. `codeland` 是主机名称，跟 `hostname -s` 的输出一致。
3. 当前工作目录，`~` 代表用户的主目录，在 Linux 系统下也就是 `/home/<` 用户名 `>`，macOS 中则为 `/Users/<` 用户名 `>`。
4. `$` 为命令提示符。通常普通用户的命令行提示符与超级用户（root）的不同，以 bash 为例，root 用户的命令行提示符为 `#`。
5. 待执行的命令，在本例中是 `echo -e "\t` 你好，命令行`"`，除 `echo` 命令本身外，还包括该命令的选项（`-e`）以及参数（`\t` 你好，命令行）等部分。命令的选项参数一般由引号（`"`）引起，以避免诸如空格之类的特殊字符所导致的岐义。可以使用单引号（`'`）或双引号（`"`），但语意会不同。

除了这 5 个部分之外，在这个命令行中，我们还可以看到 `@`、`:`、以及 `` ` ` ``（空格）等字符。`@` 一般用来分隔用户名和主机名，其形式跟电子邮箱地址一样。`:` 在这里起到提示说明作用。空格则常常用来分隔命令的选项和参数。因为命令行提示符可以定制，所以你的命令行界面可能跟我们在这里介绍的不同。

现在，请你跟我们一起，在命令行的提示符（`$` 或 `#`）后面输入 `echo -e "\t` 你好，命令行`"`。如果在输入过程中有错误，不必慌张，按**退格键**（BackSpace）

或**删除键**（Delete）删除后重新输入即可。当所有字符全部输入完成后，按下**回车键**（Enter）。

发现了什么？命令行向我们回显了一条“你好，命令行”的消息。而且 `echo` 命令参数中的 `\t` 在输出中产生了一个制表符（Tab），从而让消息有了缩进效果。

```
xiaodong@codeland:~$ echo -e "\t 你好，命令行"
        你好，命令行
```

恭喜！你刚刚在命令行成功执行了一条命令，是否感觉并没有想象中那么恐怖呢？在后面的章节中，我们将教你如何更加高效的使用命令行，从而提升你的工作效率。

第二章　神奇补全

如果你编写过代码，那么一定听说过“代码补全”吧。在如今流行的代码编辑器和 IDE (集成开发环境) 中，这绝对是一项深受大家喜欢的功能。在此我要讲的 Shell 补全与它很相似。我相信，在学习了本章所讲的内容后，你肯定会爱上它。首先，我们会谈谈什么叫自动补全，然后看看如何触发自动补全，接着详细探讨诸如文件名或路径名、程序名或命令名、用户名、主机名、以及变量名等各种自动补全类型，最后再介绍可编程补全。

2.1　何谓补全

现在回过头来看，在学习命令行时，我最想率先学习的功能一定是自动补全。为什么这么说呢？因为自动补全这项功能让我们只需输入开头的一个或几个字符便能通过 Shell 自动补全剩下的内容。对于痛恨输入长命令或文件名的朋友而言，自动补全绝对是福音。自动补全不仅减少了输入，而且节省了时间，从而极大的提高了我们的操作效率。

让我们通过一个例子来说明何谓自动补全。首先，我通过在 bash 中直接输入完整的命令行

```
xiaodong@codeland:~$ ls -l reallylongname.txt
```

来查看 `reallylongname.txt` 这个文本文件的信息。

然后，我在输入

```
xiaodong@codeland:~$ ls -l r
```

之后按 **Tab 键**，于是 bash 帮我自动补全了该文件名剩下的部分。

```
xiaodong@codeland:~$ ls -l reallylongname.txt
```

比较两次输入，bash 帮我少输了 17 个字符。是不是感觉很爽呢?

再看一个例子：这次，我在输入

```
xiaodong@codeland:~$ ls -l f
```

后按 **Tab**，bash 自动补全了 `file`。

```
xiaodong@codeland:~$ ls -l file
```

接着，我连按两下 **Tab**，这时 bash 向我们展示了可以自动补全的文件名列表，总共包括 5 个项目。

```
xiaodong@codeland:~$ ls -l file
file1  file2  file3  file4  file5
```

我输入 `1` 来完成 bash 自动补全过程。

比较这两个例子，我们可以发现，如果我们输入的开头字符唯一，那么 bash 将直接自动补全余下的内容。反之，则提供一个可供补全的备选列表。不过，这时候需要我们连按两下 **Tab 键**。这样的话，经常操作起来感觉还是有点麻烦。

下面我们对 bash 自动补全的配置进行一番优化，使之更加好用。利用文本编辑器打开 `~/.inputrc` 文件 (若不存在，则创建一个)，加入下列内容：

```
# completion
set show-all-if-ambiguous on
set visible-stats on
set colored-completion-prefix on
```

其中，开启 `show-all-if-ambiguous` 这个选项后，我们只需按一次 **Tab** 即可看到备选补全列表；`visible-stats` 选项通过在列表项目尾部添加指示符号来说明类型，例如：`@` 代表符号链接、`/` 代表目录等；最后的 `colored-completion-prefix` 选项则给补全的前缀字符加上颜色。如图 2.1 所示。

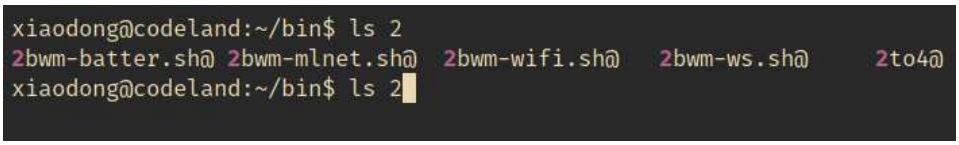


图 2.1: bash 自动补全配置结果

2.2 补全触发按键

通过这些例子，我们也可以知道，要触发自动补全，一般只要按 **Tab 键**即可。bash 和 zsh 都是这个默认设定。

2.3 文件名、路径名补全

前面的例子显示的是在 bash 中文件名自动补全的情况。下面我们看一个在 zsh 中自动补全文件名的例子。当我输入

```
xiaodong@codeland:~$ ls -l f
```

后按 **Tab**，zsh 为我自动补全了 `file`。

```
xiaodong@codeland:~$ ls -l file
```

我接着再按 **Tab 键**，这时 zsh 提供了可以备选的自动补全菜单。

```
xiaodong@codeland:~$ ls -l file
file1  file2  file3  file4  file5
```

再次按 **Tab** 则可以选择具体的菜单项目。

```
xiaodong@codeland:~$ ls -l file2
file1  **file2**  file3  file4  file5
```

然后按**回车键**完成自动补全过程。

```
xiaodong@codeland:~$ ls -l file2
```

最后再按一次**回车键**执行命令。

不知大家有没有发现与 bash 补全的区别呢？bash 提供的备选补全列表不能选择具体的项目，而 zsh 则可以。这也说明与 bash 相比，zsh 具有更棒的用户交互功能。所以我平常也更喜欢使用 zsh 一些。如果你还没有用过 zsh，那么我在此建议你一定要试一试。

说到备选补全列表，在 bash 中我喜欢使用的一组快捷键是 **Alt + ?**。当 bash 补全 `file` 后，我没有按 **Tab**，而是按 **Alt + ?**，bash 就立即呈现了备选补全列表。

```
xiaodong@codeland:~$ ls -l file
file1  file2  file3  file4  file5
```

还有一种情况，有时候我们希望 Shell 不要补全某些特别的文件类型。为了达到这种效果，我们可以使用 `FIGNORE` 变量。在下面的例子中，我想查看

Welcome.java 的内容，因此只想 Shell 补全 .java 文件，并排除 .class 文件。

```
xiaodong@codeland:~$ cat W
Welcome.class  Welcome.java
xiaodong@codeland:~$ cat Welcome.
```

在将 .class 扩展名赋给 FIGNORE 变量后，Shell 就为我剔除掉了 .class 文件类型。

```
xiaodong@codeland:~$ FIGNORE='.class'
xiaodong@codeland:~$ cat W<Tab>
xiaodong@codeland:~$ cat Welcome.java
```

如果想要排除多种文件类型，则只需用 : (冒号) 分隔即可。例如：

```
xiaodong@codeland:~$ FIGNORE='.o:.class'
```

这将让 Shell 在自动补全时排除 .o 和 .class 文件。无论是 bash，还是 zsh，当前都支持 FIGNORE。

路径名补全和文件名补全很相似，只是在补全后自动追加上 / (斜杠)，便于我们输入下一级的路径名。在下例中，我在输入

```
xiaodong@codeland:~$ cd g
```

后按 **Tab**，Shell 补完了全名，并在其后添加了一个 /。

```
xiaodong@codeland:~$ cd guessing_game/
```

之后我接着输入 s 并按 **Tab**，这次 Shell 补全了下级目录 src。

```
xiaodong@codeland:~$ cd guessing_game/src/
```

对于开头字符不唯一的情况，跟文件名补全也是一样，bash 中只要按 **Tab** 即可看到备选补全列表。

```
xiaodong@codeland:~$ cd h
hello/       hello_world/
xiaodong@codeland:~$ cd hello
```

顺便提一句，自动补全不光在命令行下使用，即便在有些图形化程序中也能使用。比如，在 GIMP 中，我也可以通过自动补全来打开文件。如图 2.2 所示。

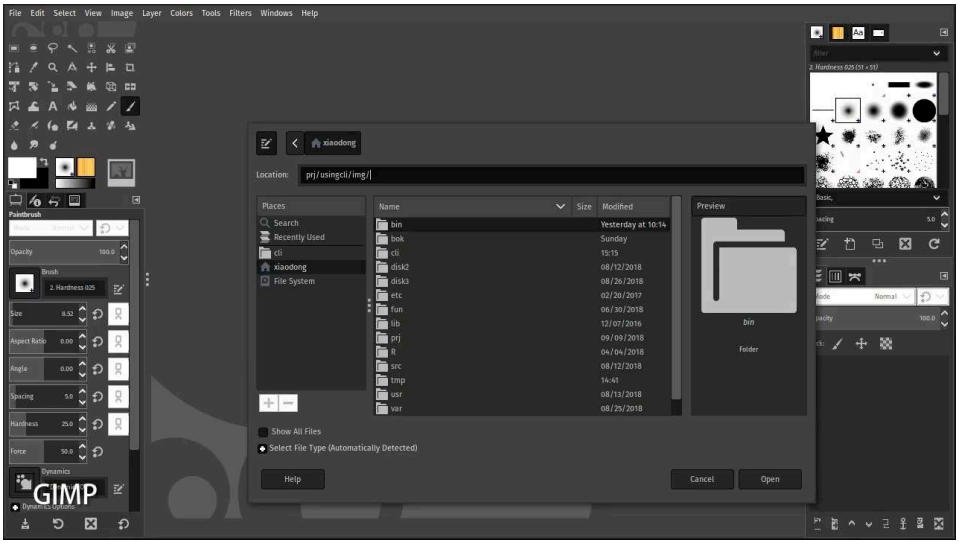


图 2.2: 在 GIMP 中自动补全文件名

2.4 程序名、命令名补全

不带选项的程序名、命令名补全几乎跟文件名补全一样，让我们来看一个例子。要是你看过《黑客帝国》这部电影，那么下面的画面你应该会很熟悉。当我输入

```
xiaodong@codeland:~$ cmat
```

后按 **Tab**，bash 立即为我自动补全了 `cmatrix` 命令。

```
xiaodong@codeland:~$ cmatrix
```

而当我在输入

```
xiaodong@codeland:~$ cma
```

后按 **Tab**，bash 为我提供了一个备选补全列表。

```
xiaodong@codeland:~$ cma
cmake         cmake-gui     cmapcube      cmark         cmark-gfm     cmatrix
```

此时，需要再输入 `t` 才能完成补全。

```
xiaodong@codeland:~$ cmat
```

如果我在仅仅输入 `c` (命令开头的第一个字符)

```
xiaodong@codeland:~$ c
```

后便按 **Tab**，这时 bash 询问我：“Display all 474 possibilities? (y or n)” (是否显示所有 474 个补全列表项目) 按 **y** 予以显示。按 **n** 则不显示。

```
xiaodong@codeland:~$ c
Display all 474 possibilities? (y or n)
```

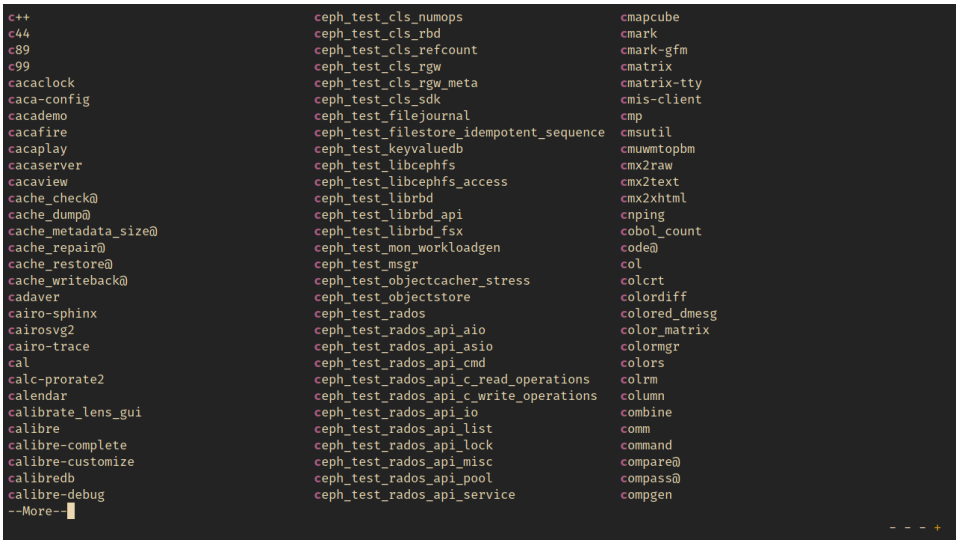


图 2.3: 命令自动补全备选列表

因为可供自动补全的列表项目太多，一屏已经显示不下了，所以 bash 使用 `more` 这个页面查看程序来呈现。现在按 **Space (空格键)** 可以翻页，如果想退出，那么按 **q** 即可。如图 2.3 所示。

除了直接补全命令名之外，Shell 也能自动补全程序的子命令，例如：`git status` 中的 `status` 以及命令的选项。不过，bash 需要安装一个单独的 `bash-completion` 包；而 zsh 因为内置了对此功能的支持，所以不需要额外的包。

bash-completion 的源代码[1]位于 GitHub 上，在此可以了解如何对其安装和配置。例如：

在 Debian 中，我们可以通过执行

```
xiaodong@codeland:~# apt install bash-completion
```

来安装它。

在 CentOS 上，除了安装 `bash-completion` 外，我推荐把 `bash-completion-extras` 也装上。

[1] https://github.com/scop/bash-completion

```
xiaodong@codeland:~# yum install bash-completion bash-completion-extras
```

而在 Arch Linux 上，则可以执行

```
xiaodong@codeland:~# pacman -S bash-completion
```

进行安装。

要配置 bash-completion，则只需要将下面这行指令加入 ~/.bashrc (个人) 或 /etc/bash.bashrc (全局) 即可。

```
[ -r /usr/share/bash-completion/bash_completion ] \
&& . /usr/share/bash-completion/bash_completion
```

在正常使用 zsh 的命令补全功能之前，我们也需要将下列内容加入到 ~/.zshrc 配置文件中：

```
# completion
autoload -U compinit
compinit -i
```

让我们先来看一个自动补全命令选项的例子。我在输入

```
xiaodong@codeland:~$ find -
```

后便立即按 **Tab**，bash 马上列出了可以自动补全的选项列表。如图 2.4 所示。

我接着输入 ina

```
xiaodong@codeland:~$ find -ina
```

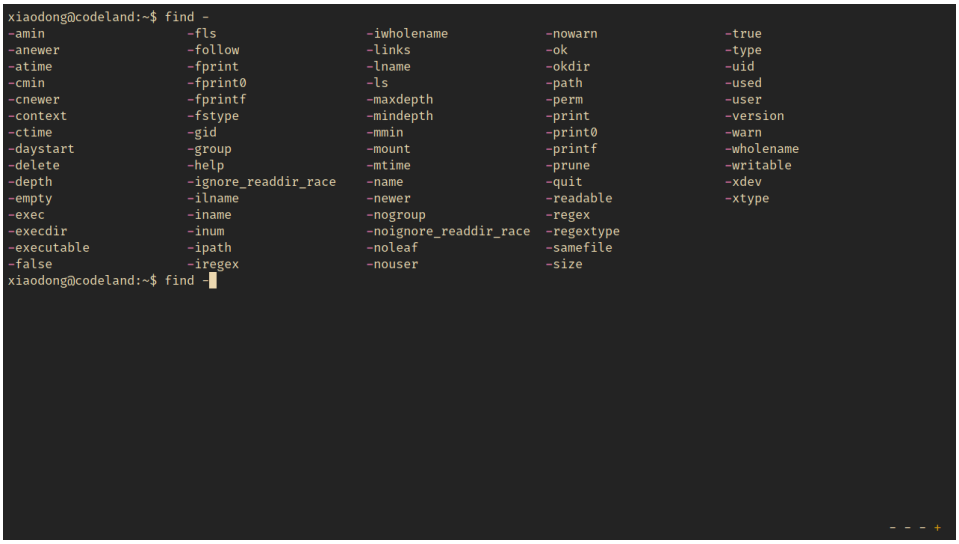


图 2.4: 命令选项自动补全备选列表

并再次按下 **Tab**，此时 bash 自动补全了 `-iname` 选项。

```
xiaodong@codeland:~$ find -iname
```

下面的例子演示了 bash 补全子命令的情形。在输入

```
xiaodong@codeland:~$ git in
```

后，按 **Tab**，bash 提供可以自动补全的子命令列表。

```
xiaodong@codeland:~$ git in
info        init        instaweb
```

跟着输入 `i`

```
xiaodong@codeland:~$ git ini
```

并再按 **Tab**，这次 bash 便自动补全了子命令 `init`。对于执行 `git status` 子命令的过程同样如此。

```
xiaodong@codeland:~$ git init
```

与 bash 比较而言，zsh 对于命令选项的补全提供更好的用户体验。在下面的例子中，你将看到，zsh 不仅列出了可供补全的选项列表，更有对该选项用途的解释。如图 2.5 所示。此外，正如前面提到的，你还可以选择这些列表项目。

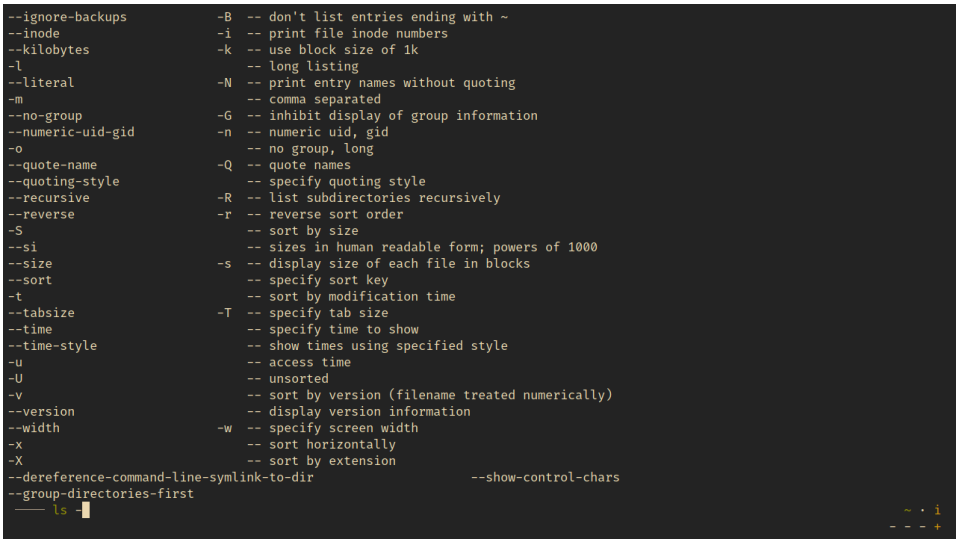


图 2.5: zsh 中的命令选项自动补全

对于子命令的补全，zsh 提供与命令选项补全相同的效果。

此外，子命令以及选项补全也可以合用。在下例中，我先补全了子命令 `git status`，然后又补全了选项 `--verbose`。

```
xiaodong@codeland:~$ git sta
stash  -- stash away changes to dirty working directory
```

```
status -- show working-tree status
xiaodong@codeland:~$ git status --v
xiaodong@codeland:~$ git status --verbose
```

2.4.1 Zsh 自动建议插件

对于使用 zsh 的朋友，我在此推荐一个好用的命令自动建议插件。这个插件叫做 zsh-autosuggestions[2]。针对命令进行自动建议这项功能源自于 `fish` shell，现在，zsh 从其借鉴过来，使得我们这些 zsh 的忠实拥趸也能使用这项好功能。

zsh-autosuggestions 的安装很简单，只需从 GitHub 将其克隆到本机，然后在 `.zshrc` 中引用 `zsh-autosuggestions.zsh` 并重新打开终端即可。

```
xiaodong@codeland:~$ git clone \
https://github.com/zsh-users/zsh-autosuggestions.git \
~/.zsh-autosuggestions
xiaodong@codeland:~$ echo source \
~/.zsh-autosuggestions/zsh-autosuggestions.zsh \
>> ~/.zshrc
xiaodong@codeland:~$ source ~/.zshrc
```

下面让我们来看看 `zsh-autosuggestions` 的用法，首先我在没有开启 `zsh-autosuggestions` 插件的情况下输入 `ls -l`、`cd hello_world` 等命令，除了能够使用命令补全之外，这儿没有命令的自动建议。当 `zsh-autosuggestions` 插件开启后，我一旦输入 `ls`，其后便会出现灰色的自动建议 `-la`。这是因为 zsh 知道我先前曾输入过 `ls -la` 这条命令，所以它给出了自动建议。如图 2.6 所示。

这时，我们有两种选择：一是按 → **(右方向箭)** 接受建议，二是继续输入新的内容，这样也就放弃建议了。对于输入 `cd h` 后，zsh 同样给出了自动建议 `ello_world`。

[2] https://github.com/zsh-users/zsh-autosuggestions

图 2.6: zsh 中的命令自动建议

2.5 用户名、主机名及变量名补全

除了常见的文件名、命令名补全外，Shell 自动补全还支持其它补全类型。这充分展现了 Shell 自动补全多才多艺的一面。下面我们就来看一看 Shell 如何自动补全用户名。

当我输入

```
xiaodong@codeland:~$ ls ~
```

之后按 **Tab**，此时 bash 为我呈现了系统中存在的用户名列表。如图 2.7 所示。

我继续输入 `x` 并再次按 **Tab**，于是 bash 补全了 `xiaodong` 这个用户名。

```
xiaodong@codeland:~$ ls ~x<Tab>
xiaodong@codeland:~$ ls ~xiaodong/
```

在 zsh 中，我们可以看到，与 bash 相比，提供的补全用户名列表表现形式略有差异。bash 中包含 `~` 前缀，并在结尾带有 `/` (斜杠)。zsh 中则仅有用户名本身。如图 2.8 所示。

如果你经常使用 `ssh` 登录远程机器的话，那么主机名自动补全将助你一臂之力。同样，我们先来看一个例子。我在输入

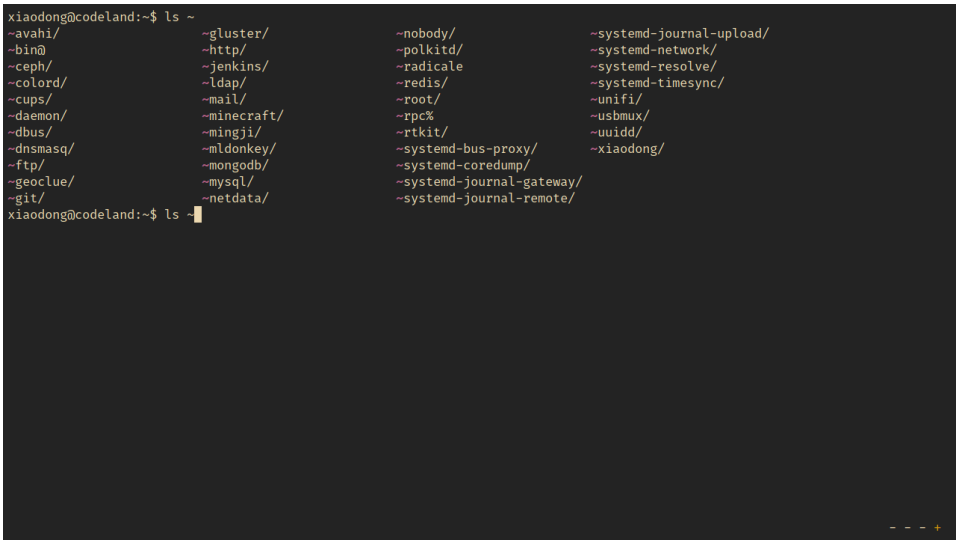


图 2.7: bash 中的用户名自动补全备选列表

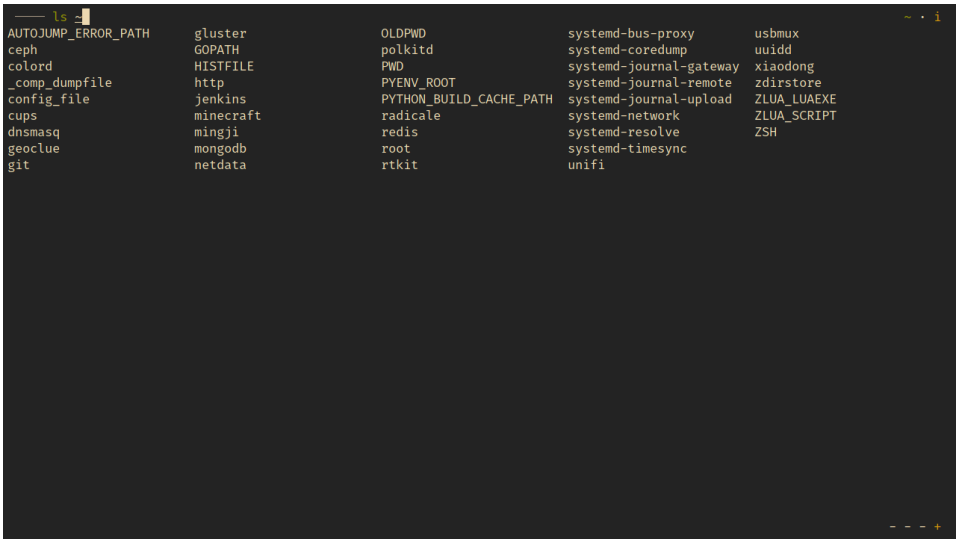


图 2.8: zsh 中的用户名自动补全备选列表

```
xiaodong@codeland:~$ ssh xiaodong@l
```

后按 **Tab**，这时 bash 展示了可以自动补全的主机名列表。

```
xiaodong@codeland:~$ ssh xiaodong@l
xiaodong@lab.github.com          xiaodong@localhost
xiaodong@linuxtoy.org            xiaodong@localhost.localdomain
xiaodong@codeland:~$ ssh xiaodong@l
```

我接着输入 `i` 并按 **Tab**，这次 bash 就自动补全了完整的主机名 `linuxtoy.org`。你也可以直接在 `@` 后按 **Tab**，这样的话就会显示全部主机名了。

```
xiaodong@codeland:~$ ssh xiaodong@li<Tab>
xiaodong@codeland:~$ ssh xiaodong@linuxtoy.org
```

不仅是主机名，而且 IP 地址也同样支持自动补全。另一种情况是，直接在输入 `ssh l` 后按 **Tab**，bash 也能对主机名进行自动补全。

```
xiaodong@codeland:~$ ssh l
lab.github.com          linuxtoy.org           localhost
```

看到这里，你或许会想，bash 从哪里找到这些可以用来自动补全的主机名呢？一个是 `/etc/hosts` 文件的内容，另一个是 `ssh` 的配置文件，比如 `~/.ssh/config`。如图 2.9 所示。所以，如果你打算让 bash 为你自动补全常用的主机名的话，那么不妨考虑将其添加到这两个文件中。此外，还包括 `~/.ssh/known_hosts` 文件。凡是通过 `ssh` 登录过的主机，便会包含其中。

zsh 对主机名的自动补全与 bash 类似，此不赘述。

最后，让我们来看看对变量名的自动补全情况。当我输入