笨办法学

笨办法学 Git

Jim Weirich 著 / 徐小东译

献给

海燕和铭基

目录

表格

插图

内容简介

本书强调通过实践来掌握 Git 的基本用法，其中包含 51 个动手实验。这些实验经过精心设计，篇幅皆十分短小，只需数分钟时间便可完成。对于想要快速学习 Git 的朋友而言，这是一份难得的指南。

更新

你可以从 https://selfhostedserver.com/learngit 获取本书的更新版本，包括 PDF、ePub 和 Mobi 格式。

- 2019.04.17：初版

作者简介

Jim Weirich，Ruby 编程语言大师，开创了流行的构建工具 Rake。

译者简介

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]、《拥抱下一代容器化工具：Podman、Buildah 和 Skopeo》[2]，译有《Perl 程序员应该知道的事》、《笨办法学 Git》等图书。Twitter：https://twitter.com/linuxtoy，Mail：xuxiaodong@pm.me[3]。

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

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

[3] mailto:xuxiaodong@pm.me

第一章　设置

1.1　目的

设置 Git 以便准备开始工作。

1.2　设置姓名和邮箱

如果你以前从来没有使用过 Git，那么你必需先做一些设置。执行下列命令能够让 Git 知道你的姓名和邮箱。如果你已经设置了 Git，那么可以跳到下一节。

```
$ git config --global user.name "Your Name"
$ git config --global user.email "your_email@whatever.com"
```

1.3　设置行尾选项

Unix/Mac 用户：

```
$ git config --global core.autocrlf input
$ git config --global core.safecrlf true
```

Windows 用户：

```
$ git config --global core.autocrlf true
$ git config --global core.safecrlf true
```

第二章　再谈设置

2.1　目的

获取教程材料，并准备执行。

2.2　获取教程包

从以下地址下载 Git 教程包：

http://selfhostedserver.com/static/file/git_tutorial.zip

2.3　解包

教程包有一个主目录 “git_tutorial” 及三个子目录：

1. html：HTML 文件。让你的浏览器打开 html/index.html。
2. work：空工作目录。你可以在这里创建仓库。
3. repos：预先打包的 Git 仓库，这样你可以在教程的任何地方进行跳转。如果你遇到问题，那么只需将想要的实验复制到工作目录。

第三章　创建项目

3.1　目的

学习如何从零开始创建 Git 仓库。

3.2　创建 “Hello, World” 程序

在空工作目录中创建一个名为 “hello” 的空目录，然后创建名为 `hello.rb` 且包含如下内容的文件。

```
$ mkdir hello
$ cd hello
```

文件：hello.rb

```
puts "Hello, World"
```

3.3　创建仓库

你现在有一个包含单文件的目录。要从该目录创建 Git 仓库，需执行 `git init` 命令。

```
$ git init
```

输出：

```
$ git init
Initialized empty Git repository in /Users/jim/working/git/
git_immersion/auto/hello/.git/
```

3.4 添加程序到仓库

现在让我们添加 “Hello, World” 程序到仓库。

```
$ git add hello.rb
$ git commit -m "First Commit"
```

你应该看到：

```
$ git add hello.rb
$ git commit -m "First Commit"
[master (root-commit) 9416416] First Commit
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 hello.rb
```

第四章　检查状态

4.1　目的

学习如何检查仓库的状态。

4.2　检查仓库的状态

使用 `git status` 命令检查仓库的当前状态。

```
$ git status
```

你应该看到：

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

`status` 命令报告没有要提交的内容。这意味着仓库包含工作目录的所有状态。没有待解决的更改要记录。

我们将继续使用 `git status` 命令来监视仓库和工作目录之间的状态。

第五章　做更改

5.1　目的

学习如何监视工作目录的状态。

5.2　更改 “Hello, World” 程序

是时候更改我们的 `hello` 程序以便使它能从命令行传递参数了。将文件更改为：

```
puts "Hello, #{ARGV.first}!"
```

5.3　检查状态

现在检查工作目录的状态。

```
$ git status
```

你应该看到：

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
```

```
#   (use "git checkout -- <file>..." to discard changes in
#   working directory)
#
#   modified:   hello.rb
#
no changes added to commit (use "git add" and/or "git commit -a")
```

值得注意的第一件事是 Git 知道 `hello.rb` 文件已被修改，但它还没有被通知到。

另外要注意的是状态信息提示你接下来需要做什么。如果你想要将这些更改添加到仓库，那么使用 `git add` 命令。否则，使用 `git checkout` 命令放弃更改。

5.4 下一步

让我们暂存更改。

第六章　暂存更改

6.1　目的

学习如何暂存更改以便稍后进行提交。

6.2　添加更改

现在告诉 Git 暂存更改，并检查状态。

```
$ git add hello.rb
$ git status
```

你应该看到：

```
$ git add hello.rb
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.rb
#
```

对 `hello.rb` 文件的更改已被暂存。这意味着 Git 现在了解这些更改，但还没有永久记录到仓库中。下面的提交操作将包含暂存的更改。

如果你决定不提交更改，那么 `status` 命令将提醒你使用 `git reset` 命令取消暂存更改。

第七章　暂存与提交

在 Git 中分开暂存是直到你需要使用源码控制处理的协调解决方法。你可以继续对工作目录做更改，然后当你想要与源码控制交互时，Git 允许你使用微小的提交来精确地记录所作的更改。

例如，假设你编辑了三个文件（`a.rb`、`b.rb` 及 `c.rb`）。现在你想提交所有更改，但你想要 `a.rb` 和 `b.rb` 中的更改作为单个提交，而 `c.rb` 的更改与前两个文件在逻辑上不相关，那么应该分开提交。

你可以执行下列命令：

```
$ git add a.rb
$ git add b.rb
$ git commit -m "Changes for a and b"

$ git add c.rb
$ git commit -m "Unrelated change to c"
```

通过分开暂存和提交，你能更容易地微调每一个提交。

第八章　提交更改

8.1　目的

学习如何提交更改到仓库。

8.2　提交更改

关于暂存谈得差不多了。让我们提交已暂存的内容到仓库。

当你先前使用 `git commit` 命令提交 `hello.rb` 文件的初始版本到仓库时，在命令行中的 `-m` 选项可以包含注释。`commit` 命令将允许你交互式地编辑提交的注释。现在让我们试试看。

如果你从命令行忽略 `-m` 选项，那么 Git 将带你到所选的编辑器中。编辑器按以下列表选择（使用优先级顺序）：

- `GIT_EDITOR` 环境变量
- `core.editor` 配置设置
- `VISUAL` 环境变量
- `EDITOR` 环境变量

我已将 `EDITOR` 变量设置为 `emacsclient`。

那么，现在提交并检查状态。

```
$ git commit
```

你应当在编辑器中看到下面的内容：

```
|
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.rb
#
```

在第一行，输入注释：“Using ARGV”。保存文件，并退出编辑器。你应该看到：

```
git commit
Waiting for Emacs...
[master 569aa96] Using ARGV
 1 files changed, 1 insertions(+), 1 deletions(-)
```

“Waiting for Emacs...” 来自 `emacsclient` 程序，该程序将文件发送到正在运行的 Emacs，并等候关闭文件。其余的输出是标准的提交信息。

8.3 检查状态

最后，让我们再检查下状态。

```
$ git status
```

你应该看到：

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

工作目录是干净的，且准备让你继续。

第九章　更改而非文件

Git 专注于文件的更改而非文件本身。当你说 `git add file` 时，你并非在告诉 Git 要添加文件到仓库。而是说 Git 应当对文件的当前状态做记录以便稍后提交。

我们将尝试在本次实验中探索其中的差异。

9.1　初次更改：允许默认名称

如果命令行参数未提供，那么更改 “Hello, World” 程序来接受一个默认值。

```
name = ARGV.first || "World"

puts "Hello, #{name}!"
```

9.2　添加更改

现在添加此次更改到 Git 的暂存区。

```
$ git add hello.rb
```

9.3　第二次更改：添加注释

现在给 “Hello, World” 程序添加一行注释。

```ruby
# Default is "World"
name = ARGV.first || "World"

puts "Hello, #{name}!"
```

9.4 检查当前状态

```
$ git status
```

你应该看到：

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.rb
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working
#   directory)
#
#   modified:   hello.rb
#
```

注意 hello.rb 在状态中被列了两次。第一次更改已被暂存，且准备提交。第二次更改还未暂存。如果你现在提交，那么注释不会保存到仓库中。

让我们试试看。

9.5 提交

提交暂存的更改，然后重新检查状态。

```
$ git commit -m "Added a default value"
$ git status
```

你应该看到：

```
$ git commit -m "Added a default value"
[master 582495a] Added a default value
 1 files changed, 3 insertions(+), 1 deletions(-)
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working
#   directory)
#
#   modified:   hello.rb
#
no changes added to commit (use "git add" and/or "git commit -a")
```

status 命令将告诉你 hello.rb 还有未记录的更改，且不在暂存区。

9.6 添加第二次更改

现在添加第二次更改到暂存区，然后执行 git status。

```
$ git add .
$ git status
```

注意：我们使用当前目录（.）作为要添加的文件。这是一种添加当前目录及其子目录下所有更改文件的惯用方式。但因为它添加所有东东，所以在做 add . 前检查状态是一个好主意，从而确信你没有添加不想要的文件。

我想你已经明白了 add . 这个技巧，但为了安全，在余下的教程中我们将继续直接添加文件。

你应该看到：

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.rb
#
```

现在第二次已经暂存，且准备提交。

9.7 提交第二次更改

```
$ git commit -m "Added a comment"
```

第十章　历史

10.1　目的

学习如何查看项目的历史。

`git log` 命令能够获得已做过的更改清单。

```
$ git log
```

你应该看到：

```
$ git log
commit 1f7ec5eaa8f37c2770dae3b984c55a1531fcc9e7
Author: Jim Weirich <jim (at) neo.com>
Date:   Sat Apr 13 15:20:42 2013 -0400

    Added a comment

commit 582495ae59ca91bca156a3372a72f88f6261698b
Author: Jim Weirich <jim (at) neo.com>
Date:   Sat Apr 13 15:20:42 2013 -0400

    Added a default value

commit 323e28d99a07d404c04f27eb6e415d4b8ab1d615
Author: Jim Weirich <jim (at) neo.com>
Date:   Sat Apr 13 15:20:42 2013 -0400
```