版本控制

> “我拒绝做计算机能够胜任的事情。”
>
> ——奥林 · 施福尔（Olin Shivers）

软件开发是一个团队工作。与他人（或整个团队）一起进行某个项目的开发时，项目成员都需要对**代码库**（codebase）进行修改并保持同步。代码库就是组成软件的那些文件夹和文件。成员可以选择将变动通过邮件进行沟通，并自我合并不同的版本，但是这样做非常耗时耗力。

另外，如果有多个成员对项目的同一处进行了修改呢？如何判断该使用谁的修改？这些都是**版本控制系统**（version control system）致力于解决的问题。版本控制系统的设计初衷，就是帮助项目成员之间更好地进行协作。

Git 和 **SVN** 是两个流行的版本控制系统。通常，版本控制系统会与一个将软件保存在云端的服务共同使用。区别在于,SVN的仓库保存在服务器, 而Git的仓库是分布式的仓库设计.这里我们会使用 Git 将软件放在 **GitHub** 上管理。

1　代码仓库

代码仓库（repository）是 Git 等版本控制系统创建的一种数据结构，用来记录编程项目中所有的变动。**数据结构**（data structure）是一种组织和保存信息的方式：字典和列表也是数据结构（本书第四部分将详细介绍数据结构）。代码仓库看上去和文件目录没有太大区别。我们将使用 Git 与记录项目变动的数据结构进行交互。

在处理由 Git 管理的项目时，通常会有多个代码仓库（每个项目成员一个）。项目成员在本地计算机上有一个**本地代码仓库**（local repository），记录自己对项目做的修改。同时还有一个托管在 Github 等类似网站的**中央代码仓库**（central repository），所有本地代码仓库要与其保持同步。项目成员可以将本地所做的修改更新到中央代码库，并将其他成员对中央代码库的修改同步到本地。如果你和其他程序员一起完成项目，项目配置大概如图所示。

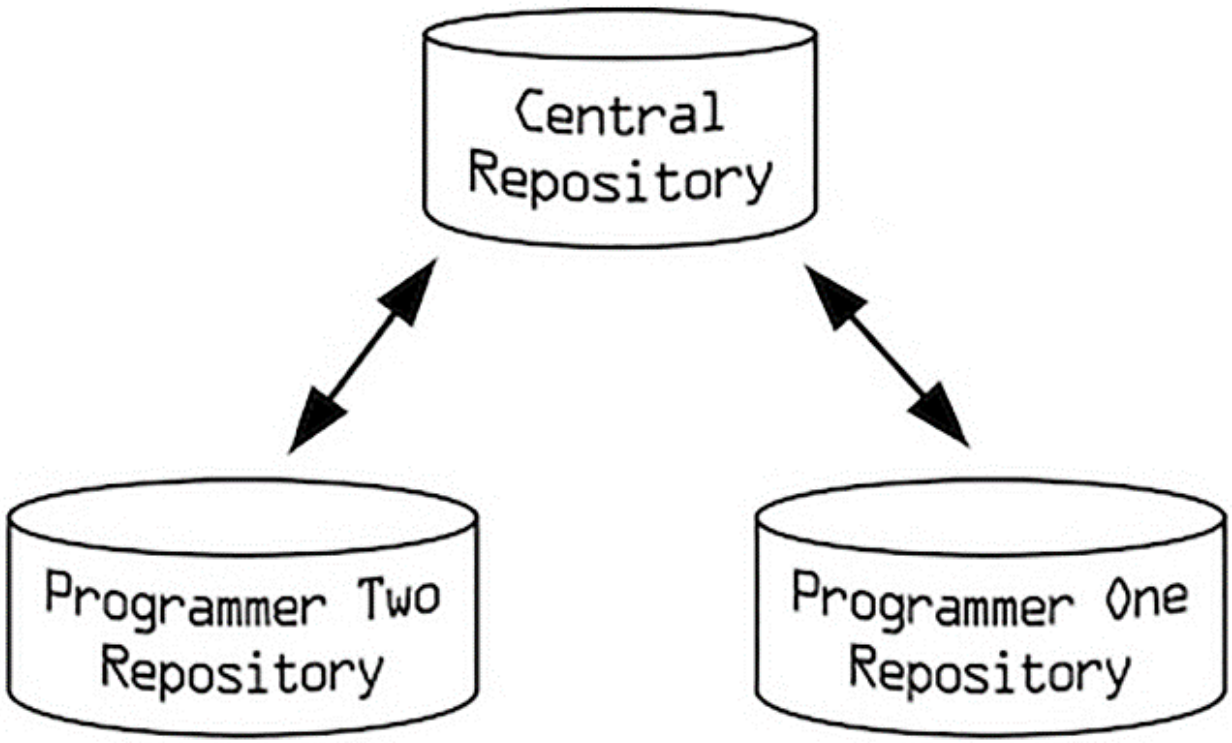


图 代码库示意

我们可以从 Github 的网站（或命令行）创建一个新的中央代码仓库。之后，即可使用 Git 创建一个可与之同步的本地代码仓库。

2 入门

在开始使用 GitHub 之前，我们需要打开 https://github.com/join 创建一个账号。之后登录账号，并点击屏幕右上角的+号按钮，即可在 Github 上创建新的代码仓库。在下拉菜单中点击 `New repository` 按钮，将仓库命名为 `hangman`，选择 `Public` 选项，并勾选 `Initialize the repository with a README`。最后，点击 `Create repository`。

接下来，点击右上角的头像并选择 Your profile 查看你的主页，如图 19-2 所示。

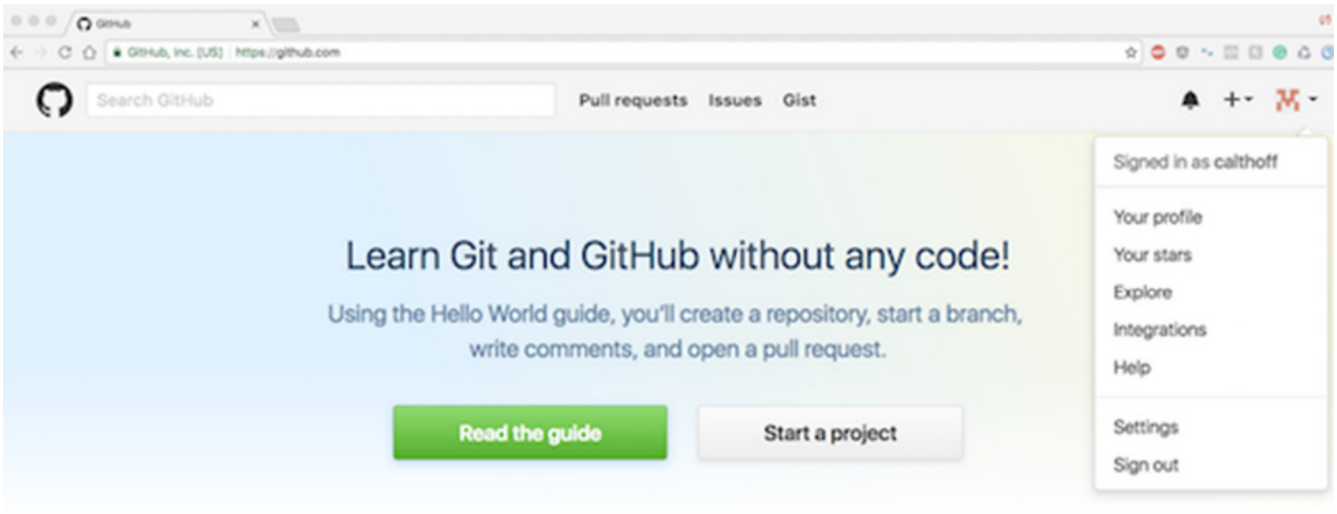


图 19-2　GitHub 主页示意

我们会看到新创建的代码仓库：`hangman`。点击该仓库，就会进入中央代码仓库的网站页面。在页面上，可以发现一个叫 `Clone Or Download` 的按钮。点击该按钮后，会看到一个链接。我们保存该链接。

在进行下一步之前，需要在本地安装 Git，安装步骤参考 https://www.git-scm.com/book/en/v2/Getting-Started-Installing-Git。

安装好之后，就可以在命令行中使用 Git。在命令行中输入 git 命令：

```
$ git

>> usage: git [--version] [--help] [-C <path>] [-cname=value] ...
```

如果在本地看到类似上面的输出，说明已经成功安装了 Git。

现在，我们可以使用之前保存的链接，通过命令 `git clone [仓库链接]` 将中央代码仓库下载到本地。下载的仓库会保存到输入命令时所在的位置。复制链接，并将其传给 `git clone` 命令：

```
$ git clone [仓库链接]

>> Cloning into 'hangman' ... remote: Counting objects: 3, done. remote:
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 Unpacking objects: 100% (3/3),
done. Checking connectivity... done.
```

使用 `ls` 命令验证本地代码仓库是否下载成功：

```
$ ls
>> hangman
```

此时应该会看到一个名为 hangman 的目录，这就是本地代码仓库。

3 推送和拉取

通过 Git 可以完成两件事情。第一件事是将本地所做的修改更新至中央代码仓库，也被称为**推送**（push）。第二件事是将中央代码仓库的新修改同步到本地，也被称为**拉取**（pull）。

命令 `git remote -v`（`-v` 是一个常用的旗标，用来打印详细信息）可打印本地代码仓库推送和拉取代码的目标 URL 链接。进入 hangman 目录，然后输入 `git remote` 命令：

```
$ cd hangman
$ git remote -v

>> origin [中央仓库链接]/hangman.git (fetch)
>> origin [中央仓库链接]/hangman.git (push)
```

输出的第一行是拉取数据的目标代码仓库 URL，第二行是推送数据的目标代码仓库 URL。通常，拉取和推送的目标仓库是相同的，因此两个 URL 也是相同的。

4 推送示例

本节将对在本地克隆的 hangman 代码仓库进行修改，然后将其推送到托管在 Github 上的中央代码仓库。

将本书第一部分挑战练习中完成的 Python 代码文件，移动到 hangman 目录。现在，本地代码仓库中有一个文件不存在于中央代码仓库，即没有与中央代码仓库保持同步。将本地的这一修改推送到中央代码仓库后，即可解决该问题。

推送修改到中央代码仓库共分 3 步。首先，**暂存**（stage）文件，告诉 Git 希望将哪个修改过的文件推送到中央代码仓库。

命令 `git status` 可以显示项目之于代码仓库的当前状态，方便我们决定暂存哪些文件。该命令会把本地代码仓库与中央代码仓库中存在差异的文件打印出来。取消文件暂存后，文件以红色字体显示。暂存的文件显示为绿色。要确保位于 hangman 目录，然后输入命令 `git status`：

```
$ git status

>> On branch master Your branch is up-to-date with 'origin/master'. Untracked files:
(use "git add <file>..." to include in what will be committed)

hangman.py
```

现在 `hangman.py` 会以红色字体显示。使用命令 `git add[文件名]` 即可暂存文件：

```
$ git add hangman.py
```

现在通过命令 `git status` 确认已经暂存了该文件：

```
$ git status

>> On branch master Your branch is up-to-date with 'origin/master'. Changes to be committed: (use "git reset HEAD <file>..." to unstage)

new file: hangman.py
```

此时 `hangman.py` 变成了绿色字体，因为已经成功暂存。

使用语法 `git reset[文件路径]` 即可取消暂存。取消暂存 `hangman.py` 的操作如下：

```
$ git reset hangman.py.
```

通过 `git status` 命令确认文件已取消暂存：

```
>> On branch master Your branch is up-to-date with 'origin/master'. Untracked files: (use "git add <file>..." to include in what will be committed)

hangman.py
```

重新暂存文件：

```
$ git add hangman.py
$ git status

>> On branch master Your branch is up-to-date with 'origin/master'. Changes to be committed: (use "git reset HEAD <file>..." to unstage)

new file: hangman.py
```

将希望更新到中央代码仓库的文件暂存之后，就可以进行下一步：提交文件，即命令 Git 记录本地代码仓库所做的修改。可使用语法 `git commit -m [信息]` 提交文件。该命令将创建一次**提交**（commit）：Git 保存的一个项目代码版本。旗标 `-m` 表示要添加一段信息，帮助记忆对项目做了什么修改以及原因（这条信息类似注释）。下一步，就是将修改推送到 Github 上的中央代码仓库，在网站上可以看到提交的信息。代码如下：

```
$ git commit -m "my first commit"

>> 1 file changed, 1 insertion(+) create mode 100644 hangman.py
```

提交文件后，即可进行最后一步。可通过命令 `git push origin master`，将本地的修改推送到中央代码库：

```
$ git push origin master

>> 1 file changed, 1 insertion(+) create mode 100644 hangman.py Corys-MacBook-Pro:hangman coryalthoff$ git push origin master Counting objects: 3, done. Delta compression using up to 4 threads. Compressing objects: 100% (2/2), done. Writing objects: 100% (3/3), 306 bytes | 0 bytes/s, done, Total 3 (delta 0), reused 0 (delta 0) To https://github.com/coryalthoff/hangman.git f5d44da..b0dab51 master -> master
```

在命令行中输入用户名和密码后，Git 就会将本地修改推送至 Github。这时再前往 Github 的网站查看，就可以看到新推送的 `hangman.py` 和提交时输入的信息，如图 19-3 所示。

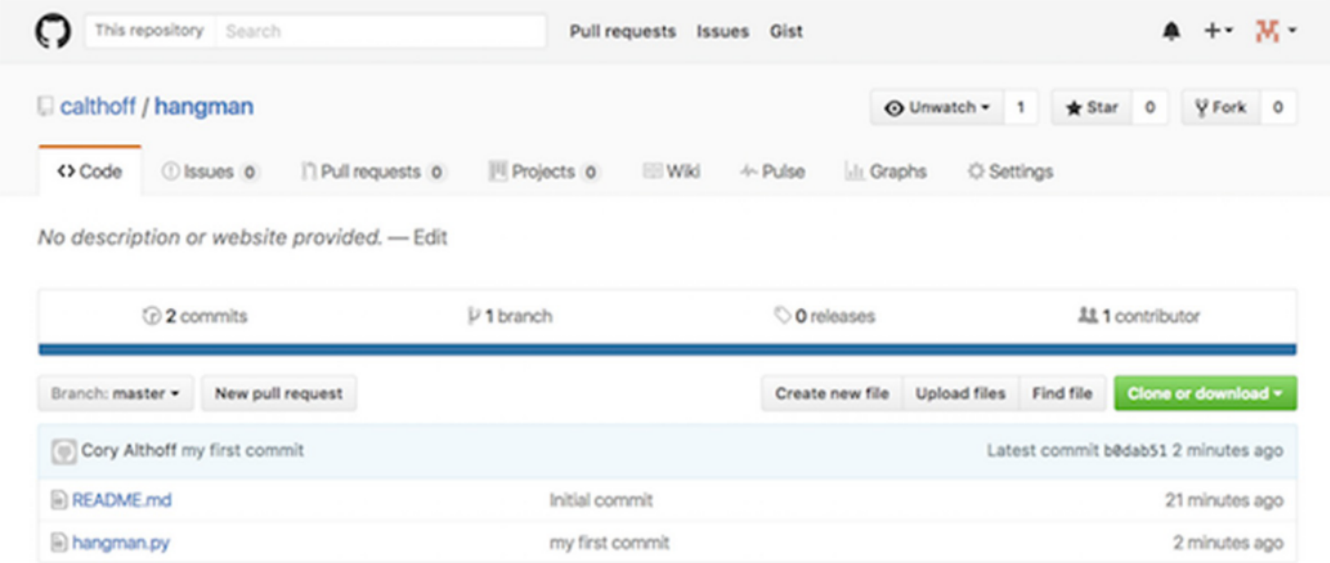


图 19-3　GitHub 网页示意

5　拉取示例

本节将拉取中央代码仓库的修改，来更新本地代码仓库。如果团队中程序员修改中央代码仓库之后，其他人都需要更新本地仓库获取修改。

前往中央代码仓库，点击按钮 Create new file，创建一个名为 `new.py` 的文件，然后点击按钮 Commit new file 提交该文件。这个文件目前没有在本地代码仓库中，因此本地代码仓库落后于中央代码仓库的版本。我们可以使用命令 `git pull origin master` 更新本地代码仓库：

```
$ git pull origin master

>> remote: Counting objects: 3, done. remote: Compressing objects: 100% (2/2), done.remote. Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 Unpacking objects: 100% (3/3), done. From https://github.com/coryalthoff/hangman b0dab51..8e032f5 master -> origin/master Updating b0dab51..8e032f5 Fast-forward new.py | 1 + 1 file changed, 1 insertion(+) create mode 100644 new.py
```

Git 会把中央代码仓库的修改应用到本地。在中央代码仓库中创建的文件 `new.py` 现在会出现在本地代码仓库中。使用 `ls` 命令确认：

```
$ bash $ ls

>> README.md hangman.py new.py
```

6 回退版本

每次提交一个文件，Git 就会保存项目代码。通过 Git，我们可以回退到任意一次代码提交，即可以做到“倒带”。例如，可以将项目回退到上周所做的某一次提交时，所有的文件和目录都与上周提交时一模一样。之后，也可以立即跳转到更近期的某一次提交。每次提交都有一个**提交编号**：Git 用来标记提交的唯一一组字符串序列。

可使用命令 `git log` 查看项目的提交历史，该命令会打印出所有做过的提交：

```
$ git log

>> commit 8e032f54d383e5b7fc640a3686067ca14fa8b43f
Author: Cory Althoff <coryedwardalthoff@gmail.com>
Date:   Thu Dec 8 16:20:03 2016 -0800

Create new.py
commit b0dab51849965144d78de21002464dc0f9297fdc
Author: Cory Althoff <coryalthoff@Corys-MacBook-Pro.local>
Date: Thu Dec 8 16:12:10 2016 -0800

my first commit
commit f5d44dab1418191f6c2bbfd4a2b2fcf74ef5a68f
Author: Cory Althoff <coryedwardalthoff@gmail.com>
Date: Thu Dec 8 15:53:25 2016 -0800 Initial commit
```

上例打印出了 3 次提交。第一次提交是在创建中央代码仓库时，第二次提交是将本地增加的 `hangman.py` 文件推送到了中央代码仓库；第三次则是创建文件 `new.py`。每次提交都有一个编号。将编号传入命令 `git checkout` 即可将项目切换到对应的提交版本。在本例中，通过命令 `git checkout f5d44dab1418191f6c2bbfd4a2b2fcf74ef5a68f`，我们可以将项目直接回退到刚开始创建时的模样。

7 diff

命令 `git diff` 可实现本地代码仓库与中央代码仓库之间文件的差别对比。在本地创建一个名为 `hello_world.py` 的新文件，然后在其中添加代码 `print("Hello, World!")`。

接着暂存该文件：

```
$ git add hello_world.py
```

确保一切正常：

```
$ git status

>> Changes to be committed: (use "git reset HEAD <file>..." to unstage) new file:
hello_world.py
```

然后提交：

```
$ git commit -m "adding new file"

>> 1 file changed, 1 insertion (+) create mode 100644 hello_world.py
```

并将修改提交至中央代码仓库：

```
$ git push origin master

>> Counting objects: 3, done. Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done. Writing objects: 100% (3/3), 383 bytes | 0
bytes/s, done. Total 3 (delta 0), reused 0 (delta 0) To https://github.com/
coryalthoff/hangman.git 8e032f5..6f679b1 master -> master
```

现在将代码 `print("Hello!")` 添加至本地代码仓库中 `hello_world.py` 文件的第二行。这样该文件就不同于中央代码仓库中的版本了。输入命令 `git diff` 查看差异：

```
$ git diff hello_world.py

>> diff --git a/hello_world.py b/hello_world.py index b376c99..83f9007 100644 ---
a/hello_world.py +++ b/hello_world.py -1 +1,2 print("Print, Hello World!")
+print("Hello!")
```

Git 会将 `print("Hello!")` 用绿色字体显示，因为这是刚添加的代码。加法操作符 `(+)` 说明这行是新添加的。如果是移除代码，删除的代码会以红色字体显示，前面则会是减法操作符 `(-)`。

8 操作示范

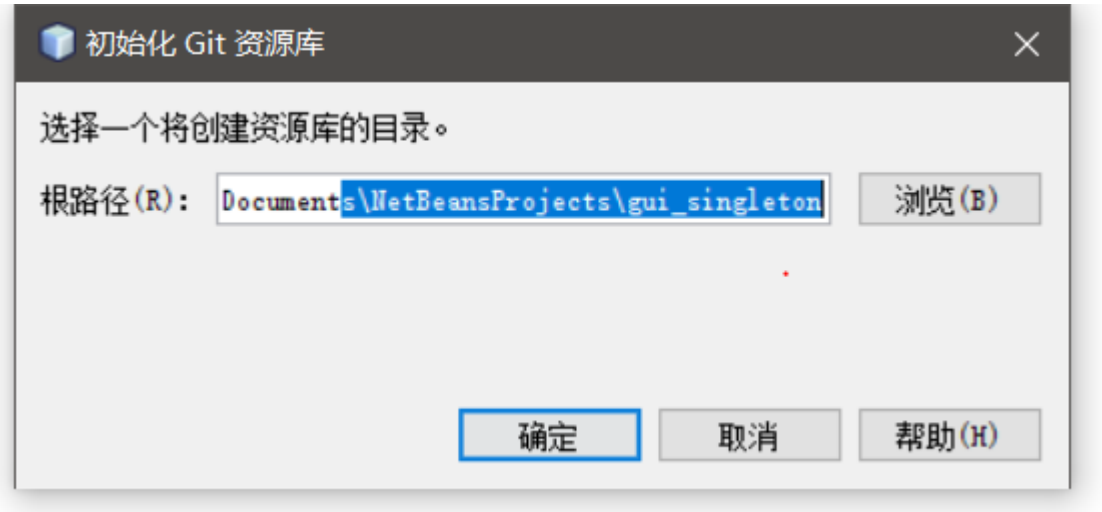
以上学习了最常用的 Git 功能。掌握这些基础后，建议大家继续学习分支和合并等更高级的功能。

下面说一下如何将NetBeans中现有项目添加GitHub的版本控制

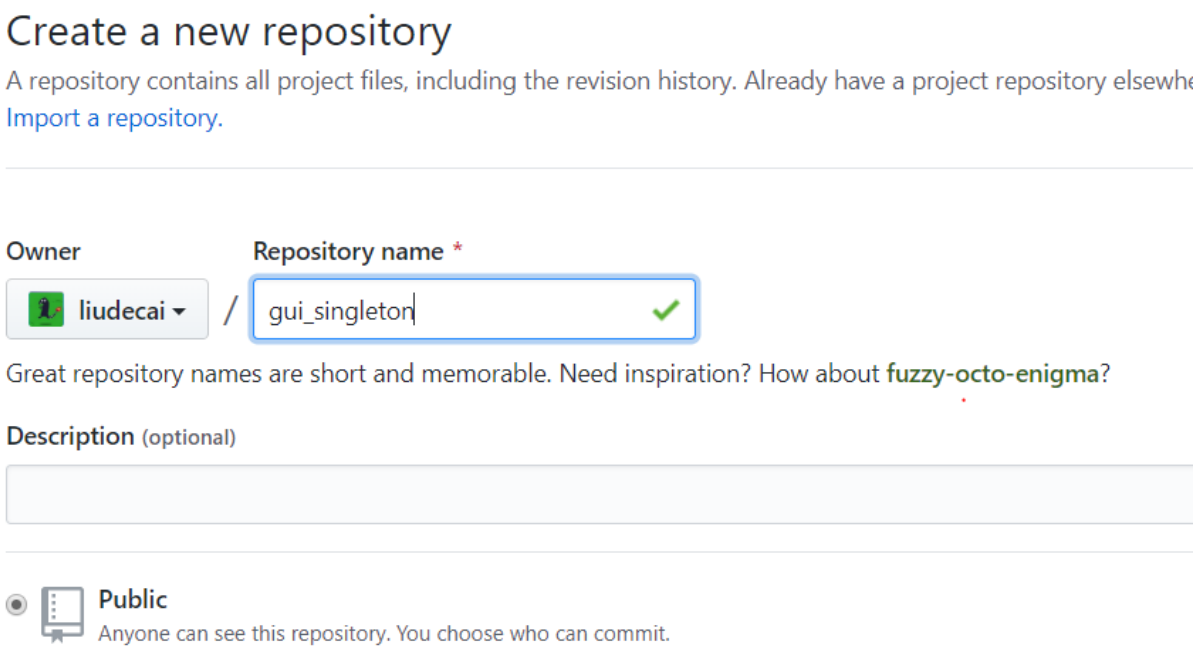
比如我有一个现有项目叫gui_singleton

1. 首先使用git初始化

   项目右键 - 版本控制 - 初始化Git资源库



2. 同时在GitHub上创建一个同名公有仓库



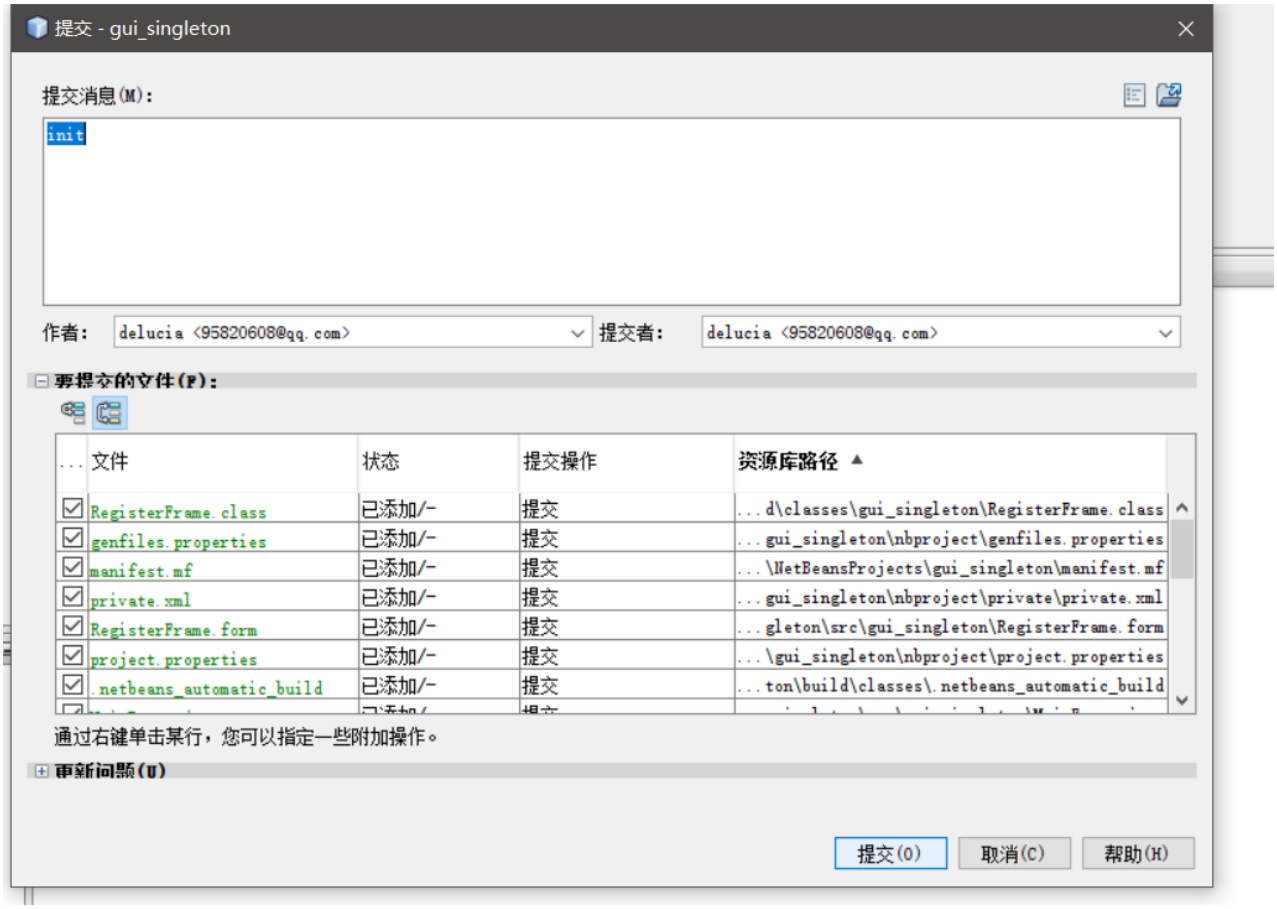
3. 配置GIT全局用户邮箱等信息

```
Lenovo@LAPTOP-DELUCIA MINGW64 ~/Documents/NetBeansP
$ git config --global user.name "liudecai"

Lenovo@LAPTOP-DELUCIA MINGW64 ~/Documents/NetBeansP
$ git config --global user.email "95820608@qq.com"
```

4. 配置本地项目的远程仓库(必须在本地仓库文件夹中)

```
Lenovo@LAPTOP-DELUCIA MINGW64 ~/Documents/NetBeansProjects/gui_singleton (master)
$ git remote add origin https://github.com/liudecai/gui_singleton.git

Lenovo@LAPTOP-DELUCIA MINGW64 ~/Documents/NetBeansProjects/gui_singleton (master)
$ git remote -v
origin  https://github.com/liudecai/gui_singleton.git (fetch)
origin  https://github.com/liudecai/gui_singleton.git (push)
```

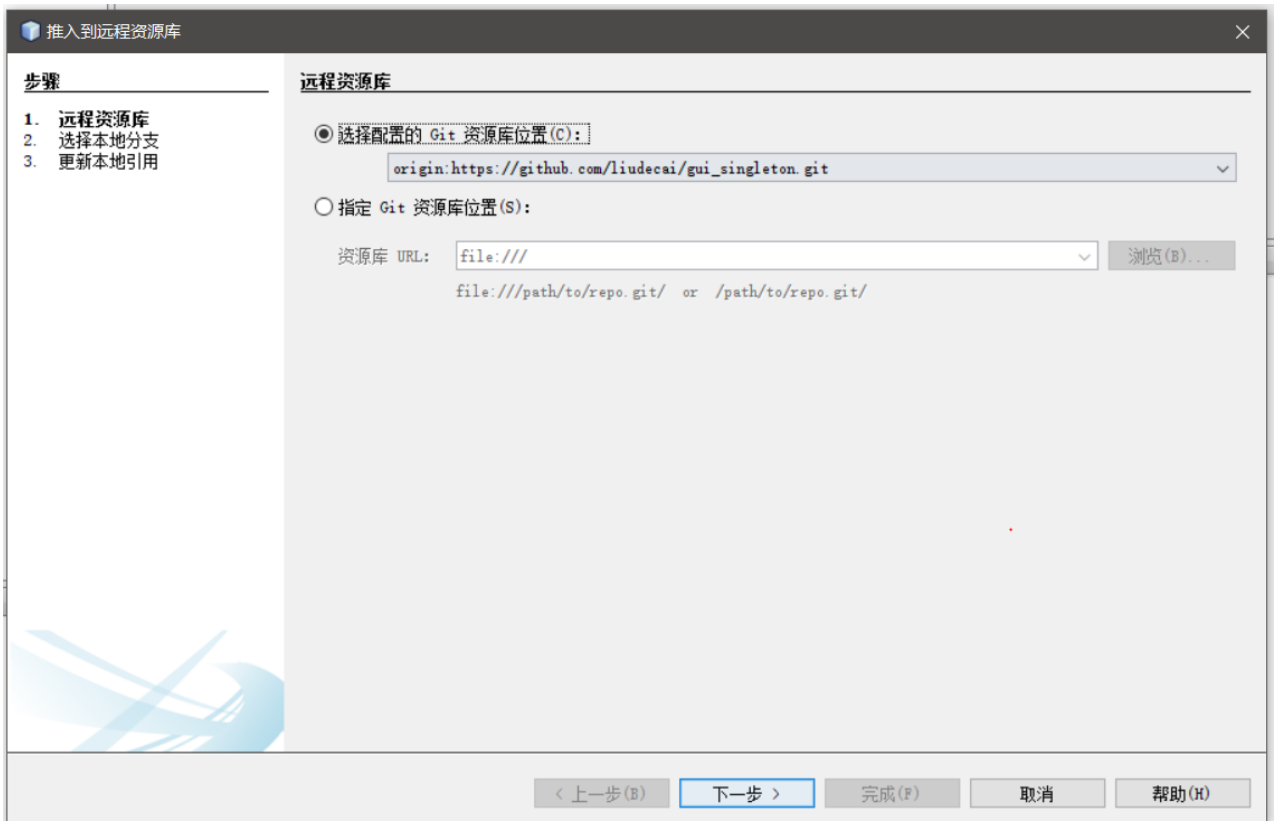
5. 将本地仓库修改提交



也可以使用命令行提交

```
git add .
git commit -m "<message>"
# 也可以使用 git commit -am "<message>"替代上面两句
```

6. 推入远程仓库



也可以使用命令行完成推送

```
Lenovo@LAPTOP-DELUCIA MINGW64 ~/Documents/NetBeansProjects/gui_singleton (master)
$ git push origin master
fatal: AggregateException encountered.
    [illegible]
fatal: AggregateException encountered.
    [illegible]
Username for 'https://github.com': liudecai
Counting objects: 27, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (24/24), done.
Writing objects: 100% (27/27), 19.34 KiB | 0 bytes/s, done.
Total 27 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/liudecai/gui_singleton.git
 * [new branch]      master -> master
```

7. 拉取远程仓库的最新代码

也可以使用命令完成推送

```
Lenovo@LAPTOP-DELUCIA MINGW64 ~/Documents/NetBeansProjects/gui_si
$ git pull origin master
From https://github.com/liudecai/gui_singleton
 * branch            master     -> FETCH_HEAD
Updating 0637b4a..9fb54e0
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

9　术语表

代码库：组成软件的目录和文件。

版本控制系统：旨在协助程序员与他人协作的程序。

Git：一款流行的版本管理系统。

SVN：一款流行的版本管理系统。

GitHub：一个将代码保存在云端的网站。

代码仓库：Git 等版本控制系统发明的一种数据结构，用来记录编程项目中的修改。

数据结构：组织和保存信息的方式。列表和字典都是数据结构。

本地代码仓库：位于本地电脑中的代码仓库。

中央代码仓库：托管在 GitHub 等网站的代码仓库，所有的本地代码仓库均需与其保持同步。

推送：将本地代码仓库的修改更新至中央代码仓库。

拉取：将中央代码仓库的修改更新至本地代码仓库。

暂存：告诉 Git 要将哪些有变动的文件推送到中央代码仓库。

提交：命令 Git 记录再代码仓库中所做的修改。

提交编号：Git 用来标识提交的字符串唯一序列。

10　挑战练习

在 GitHub 上创建一个新代码仓库。将目前项目文件集中到本地的一个目录，然后将其推送到新的代码仓库。