

# 《软件工程》实验报告（1）

## 一、实验目的

- 1、了解 GitHub，并创建账号和远程仓库。
- 2、了解版本库、工作区和暂存区的概念。
- 3、能够创建本地版本库。
- 4、在本地版本库中添加文件。
- 5、提交本地版本库到远程仓库。

## 二、实验内容

- 1、创建 GitHub 账号，本创建远程仓库。
- 2、创建本地版本库。
- 3、管理文件，并能克隆文件。

## 三、实验步骤

- 1、创建 GitHub 账号，本创建远程仓库。
- 2、了解本地版本库、工作区和暂存区的概念，创建本地版本库。
- 3、管理文件，包括添加、提交文件到远程版本库。
- 4、克隆文件。

## 四、参考实验过程

- 1、创建 GitHub 账号，创建远程版本库。

□ 注册账号：在 <https://github.com> 注册用户，需要提供用户名、邮箱和密码。

点击上方导航条的 [Signup and Pricing](#) 即可进入注册界面， 选择注册免费账户。



- 创建远程仓库：登录 *github* 上，然后在右上角找到“create a new repo”创建一个新的仓库。例如：

The screenshot shows the GitHub 'Create new repository' page. A red box highlights the 'Create new...' button in the top right corner, with a red arrow pointing to it and the label '第一步' (Step 1). Another red box highlights the 'Repository name' input field, with a red arrow pointing to it and the label '第二步填写仓库名字' (Step 2: Fill in the repository name). A third red box highlights the 'Create repository' button at the bottom, with a red arrow pointing to it and the label '第三步创建' (Step 3: Create).

在 Repository name 填入 testgit，其他保持默认设置，点击 “Create repository” 按钮，就成功地创建了一个新的 Git 仓库：

The screenshot shows the GitHub repository page for 'tughua0707 / testgit'. The page includes a 'Quick setup' section with options for 'Set up in Desktop', 'HTTP', and 'SSH'. The SSH URL is 'https://github.com/tughua0707/testgit.git'. Below this, there are sections for '...or create a new repository on the command line' and '...or push an existing repository from the command line', both with code snippets. The '...or import code from another repository' section is also visible. On the right side, there are links for 'Code', 'Issues', 'Pull Requests', 'Wiki', 'Pulse', 'Graphs', and 'Settings'.

目前，在 GitHub 上的这个 testgit 仓库还是空的，GitHub 告诉我们，可以从这个仓库克隆出新的仓库，也可以把一个已有的本地仓库与之关联，然后，把本地仓库的内容推送到 GitHub 仓库。

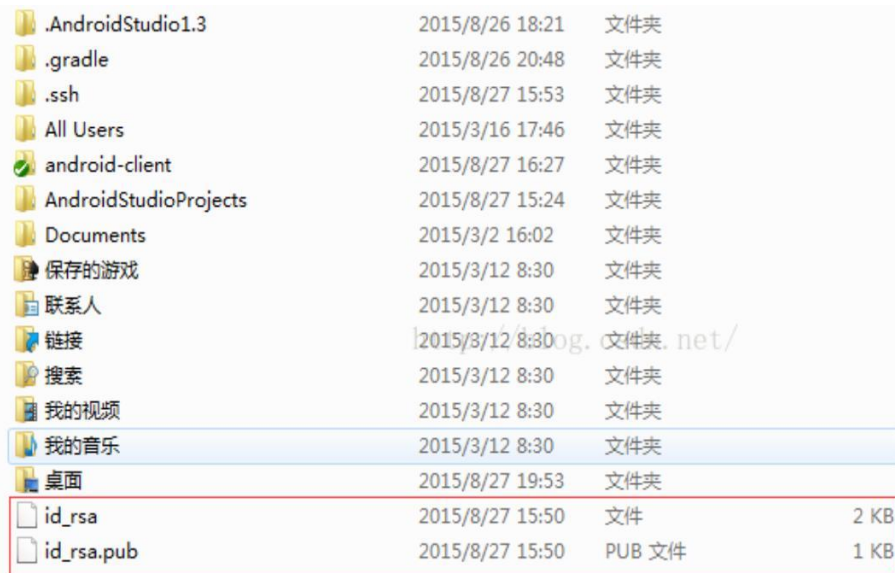
## 2、建立远程版本库的 SSH 通信。

GitHub 选择的默认通信方式是 SSH，所以要先在 Git 里面生成 SSH Key，打开 Git Bash 在其中输入如下命令：

```
ssh-keygen -t rsa -C "您的邮箱"
```

之后会让你选择是否对存放 SSH Key 的文件夹进行加密，一般都不需要的。一路回车，就 OK 了，将在 C:\Documents and Settings\Administrator\.ssh 下产生两个文件：id\_rsa（私钥）和 id\_rsa.pub（公钥）。

- 在 c 盘，当前用户文件夹下，有个 .ssh 文件夹，在里边 找到 id\_rsa.pub 文件，用记事本打开，复制其中的全部内容。
- 登陆你的 GitHub 账户，依次点击 *Account Settings > SSH Public Keys > Add another public key*，把 id\_rsa.pub 中的内容拷贝进去。
- 至此，基本的设置已经完成了。



The screenshot shows a Windows Explorer window with a list of files and folders. The .ssh folder is selected, and its contents are displayed. The files id\_rsa and id\_rsa.pub are highlighted with a red box.

名称	日期/时间	类型	大小
.AndroidStudio1.3	2015/8/26 18:21	文件夹	
.gradle	2015/8/26 20:48	文件夹	
.ssh	2015/8/27 15:53	文件夹	
All Users	2015/3/16 17:46	文件夹	
android-client	2015/8/27 16:27	文件夹	
AndroidStudioProjects	2015/8/27 15:24	文件夹	
Documents	2015/3/2 16:02	文件夹	
保存的游戏	2015/3/12 8:30	文件夹	
联系人	2015/3/12 8:30	文件夹	
链接	2015/3/12 8:30	文件夹	
搜索	2015/3/12 8:30	文件夹	
我的视频	2015/3/12 8:30	文件夹	
我的音乐	2015/3/12 8:30	文件夹	
桌面	2015/8/27 19:53	文件夹	
id_rsa	2015/8/27 15:50	文件	2 KB
id_rsa.pub	2015/8/27 15:50	PUB 文件	1 KB

## 3、测试 SSH 通信。

经过上述配置，你的 Git 应该可以通过 SSH 连接 GitHub 服务器了，让我们来测试下，输入如下命令：

会给你这样的提示：

```
$ ssh -T git@github.com
```

输入 *yes*，会显示：

```
The authenticity of host 'github.com (207.97.227.239)' can't be
established.
RSA key fingerprint is 16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48.
Are you sure you want to continue connecting (yes/no)?
```

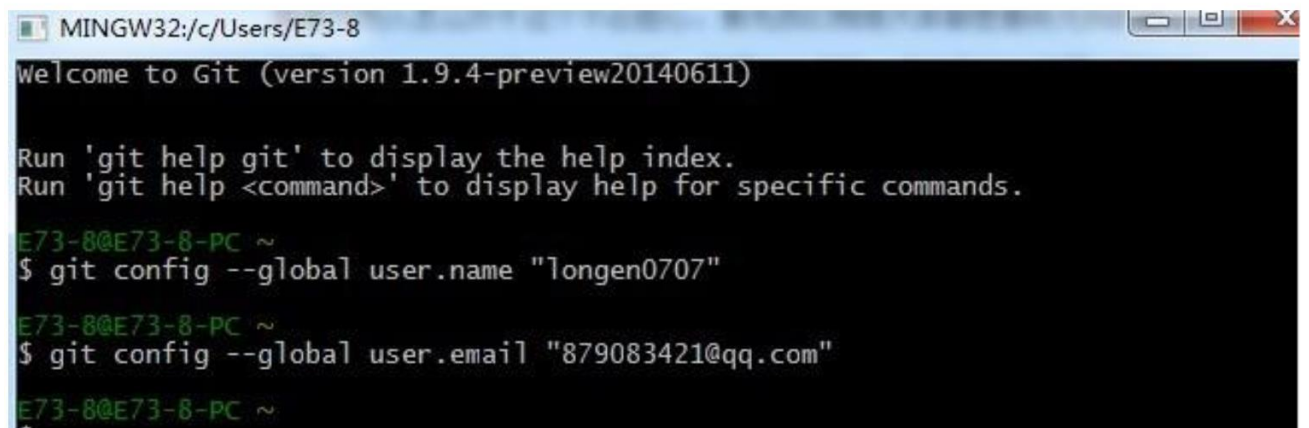
到这里，说明你的 *SSH* 运转良好。如果提示你的密钥不正确，那么你需要重新确认

```
Hi username! You've successfully authenticated, but GitHub does not provide
shell access.
```

上一步的操作是否完整无误。

#### 4、Git 用户名和邮箱设置。

Git安装完成后，还需要最后一步设置，在命令行输入如下：



```
MINGW32/c/Users/E73-8
Welcome to Git (version 1.9.4-preview20140611)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

E73-8@E73-8-PC ~
$ git config --global user.name "longen0707"

E73-8@E73-8-PC ~
$ git config --global user.email "879083421@qq.com"

E73-8@E73-8-PC ~
```

因为 Git是分布式版本控制系统，所以需要填写用户名和邮箱作为一个标识。  
注意：`git config -global` 参数，有了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置，当然你也可以对某个仓库指定的不同的用户名和邮箱。

#### 5、创建本地版本库。

创建一个本地版本库非常简单，首先，选择一个合适的地方，创建一个空目录：

```
$ mkdir learngit
$ cd learngit
$ pwd
```

```
/Users/michael/learngit
```

`pwd` 命令用于显示当前目录。在我的 Mac 上，这个仓库位于 `/Users/michael/learngit`。如果你使用 Windows 系统，为了避免遇到各种莫名其妙的问题，请确保目录名（包括父目录）不包含中文。

第二步，通过 `git init` 命令把这个目录变成 Git 可以管理的仓库：

```
$ git init
Initialized empty Git repository in /Users/michael/learngit/.git/
```

瞬间 Git 就把仓库建好了，而且告诉你是一个空的仓库（empty Git repository），细心的读者可以发现当前目录下多了一个 `.git` 的目录，这个目录是 Git 来跟踪管理版本库的，没事千万不要手动修改这个目录里面的文件，不然改乱了，就把 Git 仓库给破坏了。

如果你没有看到 `.git` 目录，那是因为这个目录默认是隐藏的，用 `ls -ah` 命令就可以看见。也不一定必须在空目录下创建 Git 仓库，选择一个已经有东西的目录也是可以的。

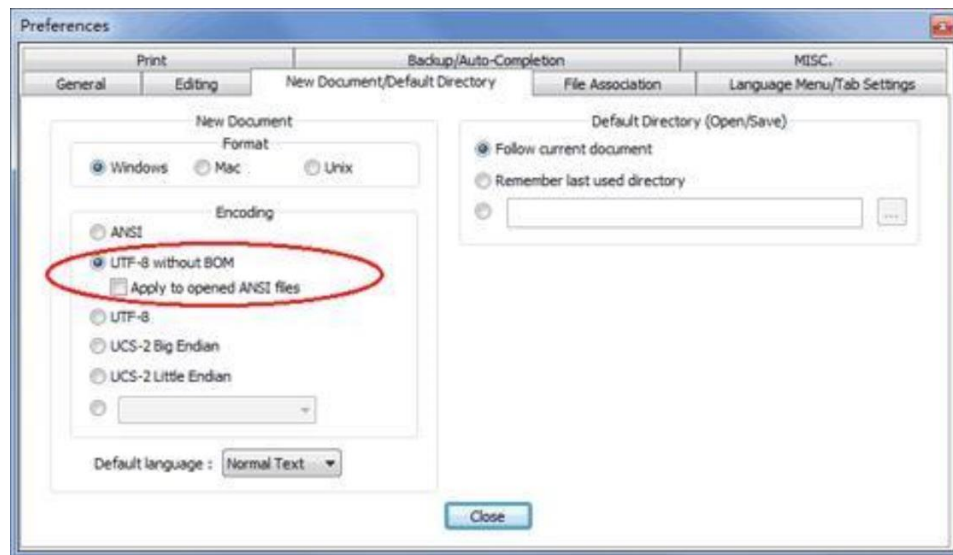
## 6、把文件添加到版本库。

首先这里再明确一下，所有的版本控制系统，其实只能跟踪文本文件的改动，比如 TXT 文件，网页，所有的程序代码等等，Git 也不例外。版本控制系统可以告诉你每次的改动，比如在第 5 行加了一个单词“Linux”，在第 8 行删了一个单词“Windows”。而图片、视频这些二进制文件，虽然也能由版本控制系统管理，但没法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是只知道图片从 100KB 改成了 120KB，但到底改了啥，版本控制系统不知道，也没法知道。

不幸的是，Microsoft 的 Word 格式是二进制格式，因此，版本控制系统是没法跟踪 Word 文件的改动的，前面我们举的例子只是为了演示，如果要真正使用版本控制系统，就要以纯文本方式编写文件。

因为文本是有编码的，比如中文有常用的 GBK 编码，日文有 Shift\_JIS 编码，如果没有历史遗留问题，强烈建议使用标准的 UTF-8 编码，所有语言使用同一种编码，既没有冲突，又被所有平台所支持。

千万不要使用 Windows 自带的记事本编辑任何文本文件。原因是 Microsoft 开发记事本的团队使用了一个非常弱智的行为来保存 UTF-8 编码的文件，他们自作聪明地在每个文件开头添加了 0xefbbbf（十六进制）的字符，你会遇到很多不可思议的问题，比如，网页第一行可能会显示一个“?”，明明正确的程序一编译就报语法错误，等等，都是由记事本的弱智行为带来的。建议你下载 Notepad++代替记事本，不但功能强大，而且免费！记得把 Notepad++ 的默认编码设置为 UTF-8 without BOM 即可：



现在我们编写一个 `readme.txt` 文件，内容如下：

```
Git is a version control system.  
Git is free software.
```

一定要放到 `learn git` 目录下（子目录也行），因为这是一个 Git 仓库，放到其他地方 Git 再厉害也找不到这个文件。

把一个文件放到 Git 仓库只需要两步。

第一步，用命令 `git add` 告诉 Git，把文件添加到仓库：

```
$ git add readme.txt
```

执行上面的命令，没有任何显示，这就对了，Unix 的哲学是“没有消息就是好消息”，说明添加成功。

第二步，用命令 `git commit` 告诉 Git，把文件提交到仓库：

```
$ git commit -m "wrote a readme file"

[master (root-commit) cb926e7] wrote a readme file

1 file changed, 2 insertions(+)

create mode 100644 readme.txt
```

简单解释一下 `git commit` 命令, `-m` 后面输入的是本次提交的说明, 可以输入任意内容, 当然最好是有意义的, 这样你就能从历史记录里方便地找到改动记录。

嫌麻烦不想输入 `-m "xxx"` 行不行? 确实有办法可以这么干, 但是强烈不建议你这么干, 因为输入说明对自己对别人阅读都很重要。

`git commit` 命令执行成功后会告诉你, 1 个文件被改动 (我们新添加的 `readme.txt` 文件), 插入了两行内容 (`readme.txt` 有两行内容)。

为什么 Git 添加文件需要 `add`, `commit` 一共两步呢? 因为 `commit` 可以一次提交很多文件, 所以你可以多次 `add` 不同的文件, 比如:

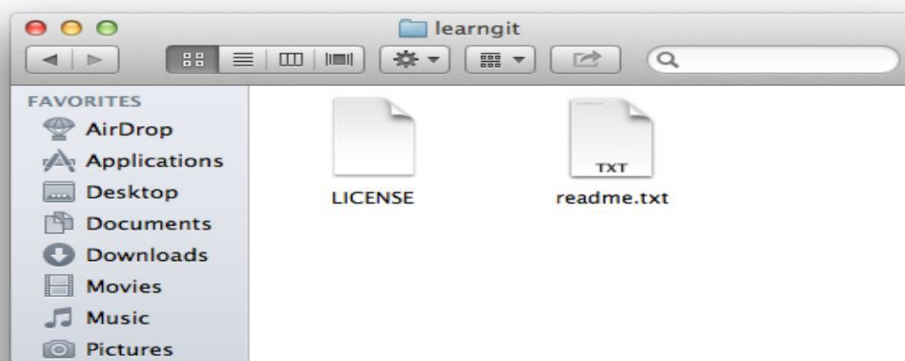
```
$ git add file1.txt

$ git add file2.txt file3.txt

$ git commit -m "add 3 files."
```

## 7、了解版本库、工作区和暂存区的概念。

- 工作区 (Working Directory) : 工作区就是你在电脑里能看到的目录, 比如我的 `learngit` 文件夹就是一个工作区。



- 版本库 (repository) : 版本库又名仓库, 可以简单理解成一个目录, 这个目录里面



的所有文件都可以被 Git 管理起来，每个文件的修改、删除，Git 都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

□ 一个案例，阐述工作区与版本库的关系。

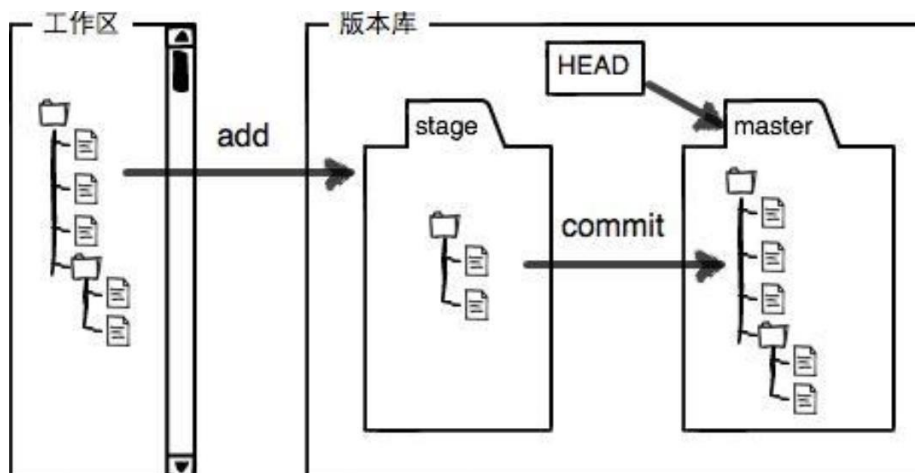
例 1 往 Git 版本库里添加文件，分析过程。

第一步是用 `git add` 把文件添加进去，实际上就是把文件修改添加到暂存区；

第二步是用 `git commit` 提交更改，实际上就是把暂存区的所有内容提交到当前分支。

解释：因为我们创建 Git 版本库时，Git 自动为我们创建了唯一一个 `master` 分支，所以，现在，`git commit` 就是往 `master` 分支上提交更改。你可以简单理解为，需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

- 1) 工作区有一个隐藏目录 `.git`，这个不算工作区，而是 Git 的版本库。
- 2) Git 的版本库里存了很多东西，其中最重要的就是称为 `stage`（或者叫 `index`）的暂存区，还有 Git 为我们自动创建的第一个分支 `master`，以及指向 `master` 的一个指针叫 `HEAD`。



□ 强化练习：现在，我们再练习一遍，先对 `readme.txt` 做个修改，比如加上一行内容：

```
Git is a distributed version control system.  
Git is free software distributed under the GPL.  
Git has a mutable index called stage.
```

然后，在工作区新增一个 `LICENSE` 文本文件（内容随便写）。

先用 `git status` 查看一下状态：

```
$ 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:   readme.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       LICENSE

no changes added to commit (use "git add" and/or "git commit -a")
```

Git 非常清楚地告诉我们，`readme.txt` 被修改了，而 `LICENSE` 还从来没有被添加过，所以它的状态是 **Untracked**。

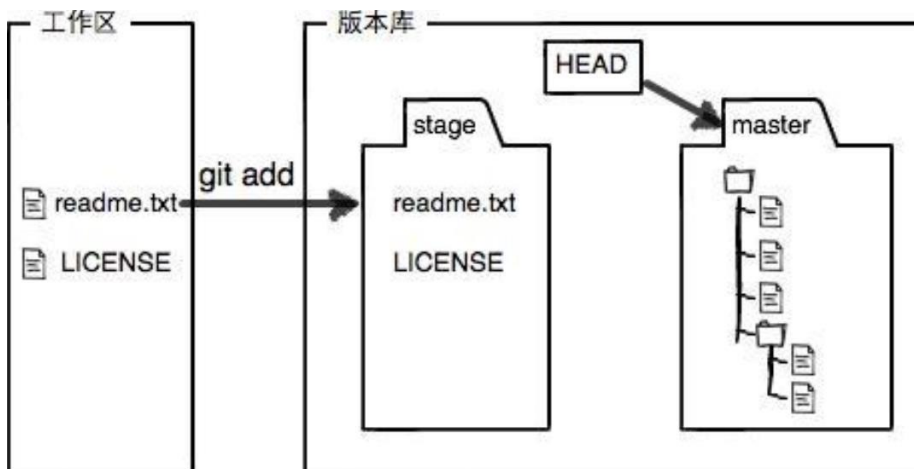
现在，使用两次命令 `git add`，把 `readme.txt` 和 `LICENSE` 都添加后，用 `git status` 再查看一下：

```
$ git status

# On branch master

# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   LICENSE
#       modified:   readme.txt
#
```

现在，暂存区的状态就变成这样了：



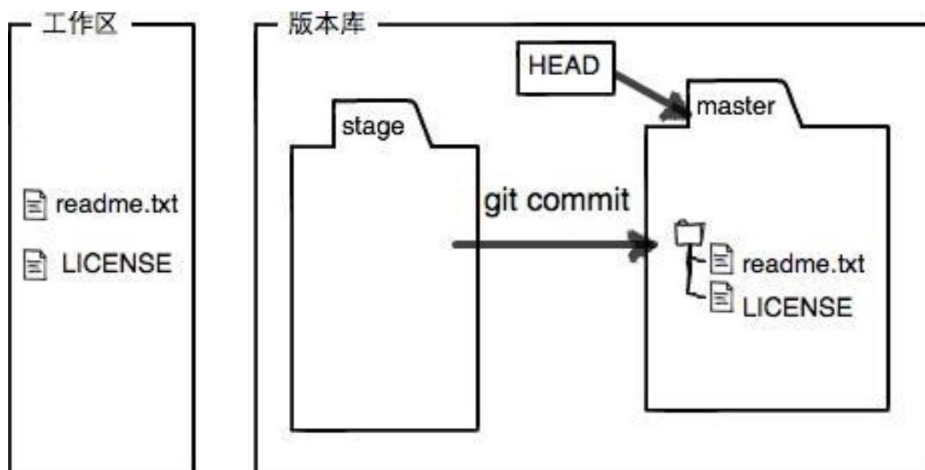
所以，`git add` 命令实际上就是把要提交的所有修改放到暂存区（Stage），然后，执行 `git commit` 就可以一次性把暂存区的所有修改提交到分支。

```
$ git commit -m "understand how stage works"
[master 27c9860] understand how stage works
2 files changed, 675 insertions(+)
create mode 100644 LICENSE
```

一旦提交后，如果你又没有对工作区做任何修改，那么工作区就是“干净”的：

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

现在版本库变成了这样，暂存区就没有任何内容了：



8、推送到远程仓库：`git push origin`

8、克隆远程仓库到本地。

假设现在你的团队其他成员已经在 git 上建好了仓库，并且也 push 过代码,这个远程 git 仓库还叫“StudyGit”，有两个文件a.txt 和 README.md,现在，您也要开始贡献代码了，那么，您首先需要把团队其他成员提交的所有东西都拉取到你的本地目录，这个时候就会用到“clone”命令了：

```
git clone git@github.com:chenxhjeo/softwareEngineer.git
```

或者

```
git clone https://github.com/chenxhjeo/softwareEngineer.git
```

(注：<https://github.com/chenxhjeo/softwareEngineer.git> 本课程的相关文件的克隆地址为：

## 五、思考与总结

总结：

- 1、建立工作区，使用 `mkdir learngit`。
- 2、初始化一个本地 Git 仓库，使用 `git init` 命令。
- 3、添加文件到 Git 仓库，分四步：

第一步，使用 notepad+创建文件。

第二步，使用命令 `git add <file>`，注意，可反复多次使用，添加多个文件；

第三步，使用命令 `git commit`，完成。

第四步，合并到远程仓库。

```
Xiang Xihui@DESKTOP-N11TPM MINGW64 ~/learngit (master)
$ git clone https://github.com/xihuixiang/projects.git
Cloning into 'projects'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

克隆远程仓库到本地

```
MINGW64/c:/Users/Xiang Xihui/learngit/projects
Xiang Xihui@DESKTOP-N11TPM MINGW64 ~/learngit (master)
$ cd projects
Xiang Xihui@DESKTOP-N11TPM MINGW64 ~/learngit/projects (master)
$ git init
Reinitialized existing Git repository in C:/Users/Xiang Xihui/learngit/projects/.git/
Xiang Xihui@DESKTOP-N11TPM MINGW64 ~/learngit/projects (master)
$ git add jsp.txt
fatal: pathspec 'jsp.txt' did not match any files
Xiang Xihui@DESKTOP-N11TPM MINGW64 ~/learngit/projects (master)
$ git add jsq.txt
Xiang Xihui@DESKTOP-N11TPM MINGW64 ~/learngit/projects (master)
$ git commit -m"wrote a jsq file"
[master 91a505a] wrote a jsq file
1 file changed, 64 insertions(+)
create mode 100644 jsq.txt
Xiang Xihui@DESKTOP-N11TPM MINGW64 ~/learngit/projects (master)
$ push -u origin master
bash: push: command not found
Xiang Xihui@DESKTOP-N11TPM MINGW64 ~/learngit/projects (master)
$ git push -u origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 838 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
Branch master set up to track remote branch master from origin.
To https://github.com/xihuixiang/projects.git
0851a04..91a505a master -> master
```

成功将 jsq.txt 文件推送到 projects 仓库中。