1小时搞懂 Git 版本控制

写在前面

很高兴在我的 Chat 分享里与你认识，前段时间，在平台上推出过几篇文章分享，在此期间收到不少读者的回复与好评，在这里再次感谢大家的支持。

> 为什么会写这篇文章？

随着秋招进行中，诸多大学生以及转行做 IT 的技术者开始应聘工作，然而当他们面试被问到 Git 时，却对其一无所知。那么 IT 工作者团队开发是怎样的呢？目前的软件开发项目通常是由一个研发小组（团队）共同分析、设计、编码、维护以及测试的。Git 则是一款分布式源代码管理工具，目前国内企业几乎都已经完成了从 SVN 到 Git 的转换。

本场 Chat 主要围绕以下内容展开讲解：

聊聊学习 Git 那些事

现在回想起来，其实我接触 Git 的时候是在我大一的时候，我的表哥带我入门的。当时因为需要做一个项目，所以他教我如何使用 Git 将写好的代码推送到 GitHub 上，然后再从远程仓库拉到本地。起初因为没有接触过 Git，觉得这玩意很难学，又是一大堆命令需

要记忆，在他教我的时候内心是抵触的，当时觉得为什么不把写好的代码发送给我呢？你是否也有过这样的疑问呢？

学习 Git 的时候，因为没有和他认真学，在他教过我一遍之后还是一脸懵逼，写命令的时候也是不时地回头查看。因为不懂得 Git 版本控制的原理，总是将代码推送不到远程服务器，同时还出现一大堆错误，只好不停地去询问他原因。

这是我当时学习 Git 版本控制的小经历，现在你可以想想你的情况。

- 你是否和我当初一样，大学才开始接触 Git？
- 是否还未听过 Git OR GitHub？
- 是否还未拥有自己的 GitHub 账号？
- 是否还不懂 Git 工作原理？
- 是否看完 Git 网络教程还是不会将代码推送到 GitHub？

如果你还是这种情况，没关系，因为这不是你一个人的问题。我相信通过本次 Chat，你一定可以懂得 Git 工作原理，也会拥有自己的 GitHub 账号，并且能将自己写好的代码推送到远程仓库，通过 GitHub 托管，再也不用担心重装系统导致代码丢失的问题。（这是建立在你动手的前提之下，当然本次 Chat 我会尽力讲的通俗易懂）

在使用 Git 之前，我们先需要学习一些 Git 知识铺垫，以备我们后面更好的学习 Git 版本控制。

Git 知识铺垫

- 程序员为什么要使用 Git 版本控制？
- 常见的版本控制？
- Git 是什么？
- Git 工作原理
- Git 安装

程序员为什么要使用 Git 版本控制?

现在的软件项目通常是由一个研发小组共同分析、设计、编码、维护以及测试的。在公司 99% 的都是团队合作开发项目，如果是团队开发项目，那么就会遇到以下问题：

- 难以恢复至以前正确版本（**版本 1.0~2.0**）
- 容易引发 bug
- 代码责任问题（**跑路**）
- 代码管理问题
- 代码冲突问题（**写同样的代码**）
- 无法进行权限控制
- 项目版本发布困难
- ……

针对以上诸多问题，源代码管理工具（**版本控制工具**）应用而生。

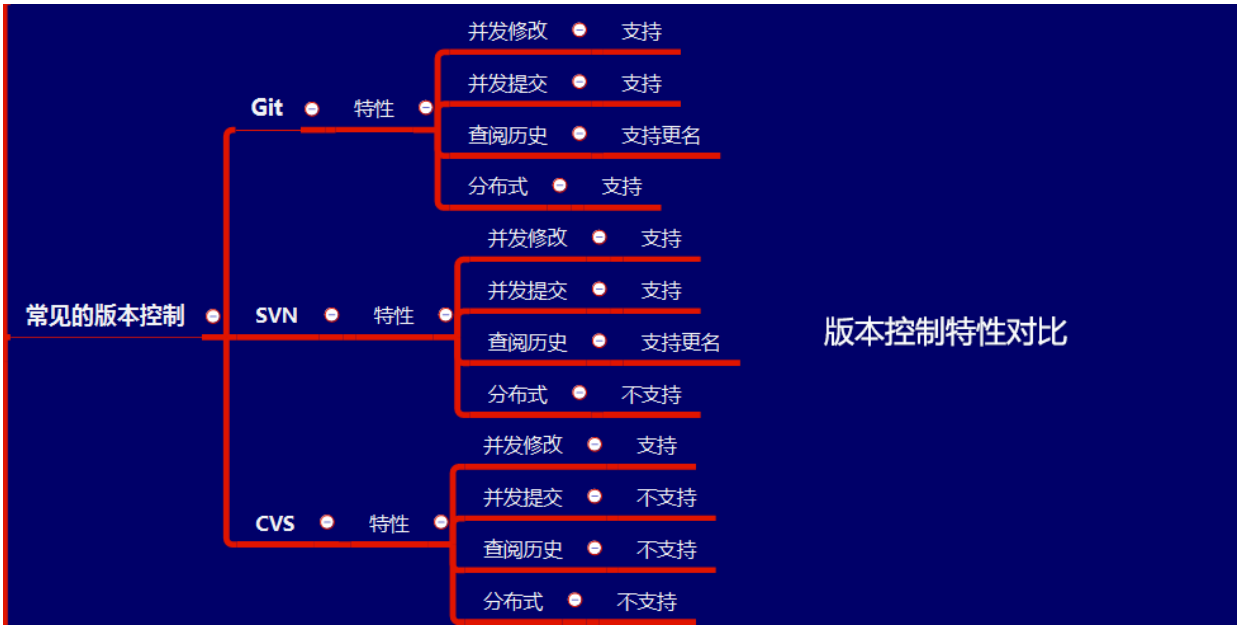
使用版本控制工具：

- 不会对现有工作造成任何损害
- 不会增加工作量
- 代码管理更方便
- 代码得以追随
- 添加新的功能拓展时，会变得更加容易
- ……

常见的版本控制

- CVS 版本控制
- SVN 版本控制
- Git 版本控制
    - **CVS**：CVS 是一个 C/S 系统，是一个常用的代码版本控制软件，1990 年诞生，10 多年前主流源代码管理工具。
    - **SVN**：SVN 又称 subversion，是一款集中式源代码管理工具。由于之前 CVS 编码的问题，大多数软件开发公司都使用 SVN 替代了CVS，前几年在国内软件企业使用最为普遍。
    - **Git**：一款分布式源代码管理工具，目前国内企业基本都使用 Git。

CVS 和 SVN 是一个集中式的版本控制器，他们需要一台专门的版本控制服务器。而 Git 是分布式的，他不要一台专门的服务器来运行这个版本控制。每个开发人员的电脑组成的网络就可以运行 Git，特别适合源代码的发布和交流，因此大部分开源项目都用 Git。目前国内企业几乎都已经完成了从 SVN 到 Git 的转换。



Git 是什么?

> **Git**：一款分布式源代码管理工具，是 Linux 之父李纳斯的第二个伟大作品。

SVN：集中式管理

- 在集中式下，开发者只能将代码提交到服务器；
- 在集中式下，只有远程服务器上有代码数据库。

Git：分布式管理

- 在分布式下，开发者可以本地提交，也可以提交到远程服务器；
- 在分布式下，每个开发者机器上都有一个代码仓库。

在世界上所有的分布式版本控制工具中，Git 是最快、最简单、最流行的。

Git 工作原理

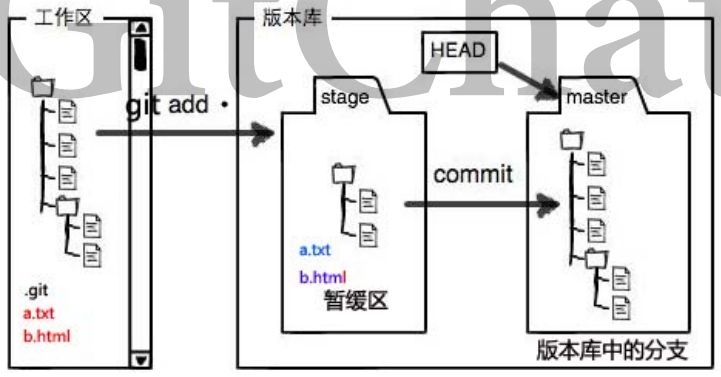
学好 Git 的前提是理解 Git 工作原理。

了解 Git 工作原理前，我们需要了解两个重要的知识，即工作区和版本库。

- 工作区：仓库文件夹里面，除了 .git 目录以外的内容（详见下图）
- 版本库：Git 目录，用于存储记录版本信息（详见下图）
  - 版本库中的暂缓区（staga）
  - 版本库中的分支（master）：Git 自动创建的第一个分支
  - 版本库中的 HEAD 指针：用于指向当前分支

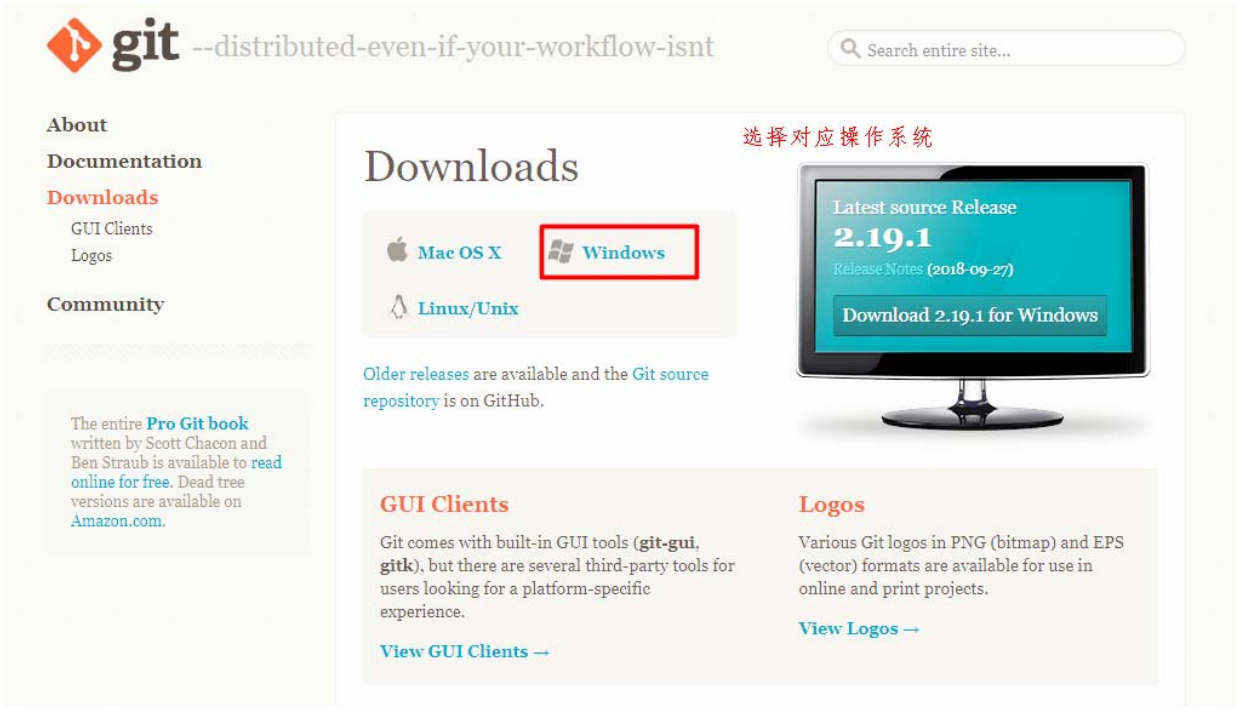
如下图所示：

在工作区中有文件 a.txt、b.html，经过执行 `git add .` 命令之后会将工作区的文件添加到暂缓区中，再经过执行 `git commit -m "说明文字"` 命令之后，会将暂缓区的文件添加到版本库的分支当中去。



Git 安装

首先进入 Git 下载地址：选择对应操作系统的版本，如下图



选择对应 Git 版本（32 位 or 64 位）适用于 Windows 安装程序的 Git，如下图

下载完成之后，安装即可（安装过程中点击下一步 **Next** 即可）。

验证 Git 是否安装成功（验证方法如下图）

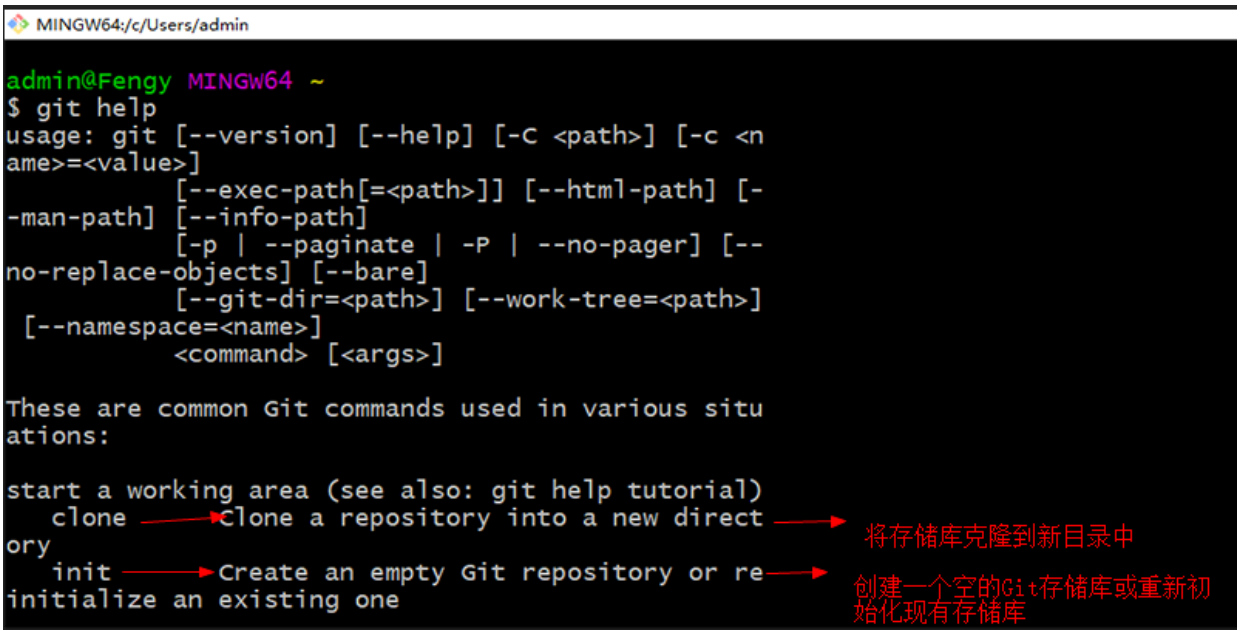
具体安装过程这里不做截图：小白请见 Git 安装百度经验

Git 命令个人开发

在学习 Git 命令个人开发之前，我们需要了解一些常用的 Git 命令。

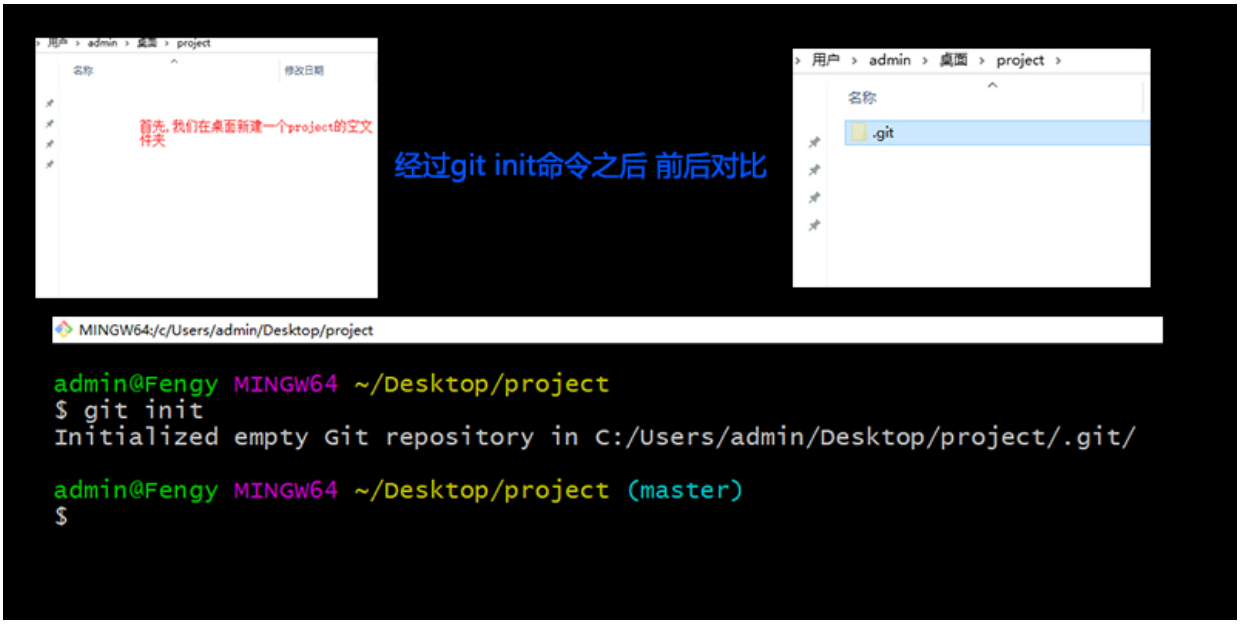
`git help`：Git 指令帮助手册

打开 git bash 终端，输入 `git help` 指令会出现如下图所示的Git命令详细解释，每个命令代表的意思（不懂英文也没关系，复制到翻译内就 ok）如 `git init` 代表初始化一个仓库。

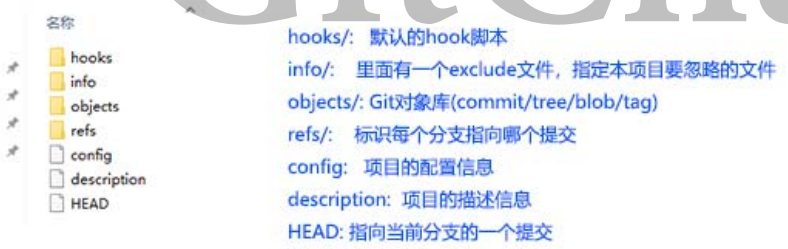


`git init`：（个人仓库）仓库初始化

首先我们在桌面新建一个 project 空文件夹之后，鼠标右键以 `git bash here` 打开 Git 终端，输入 `git init` 命令之后，观察 project 文件夹之后多了一个隐藏文件夹 Git 目录。这时我们就创建好了一个受 Git 管理的仓库，这个仓库就在本地。



隐藏的 .git 目录分别代表什么意思详见下图：



在使用 Git 之前，我们需要配置用户基本信息，即配置用户名和邮箱。（防止跑路）

当前项目下配置用户名与邮箱命令如下：

- 配置用户名：`git config user.name "用户名"`（跟踪 who 修改记录）
- 配置邮箱：`git config user.email "邮箱"`（多人开发间的沟通）

```
admin@Fengy MINGW64 ~/Desktop/project (master)
$ git config user.name "Fengy"

admin@Fengy MINGW64 ~/Desktop/project (master)
$ git config user.email "fengy@163.com"
```

git config -l：查看配置信息命令

```
$ git config -l
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
rebase.autosquash=true
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
http.sslbackend=openssl
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
credential.helper=manager
user.email=coderfeng@163.com
user.name=Fengy
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
user.name=Fengy
user.email=fengy@163.com
```

配置全局用户名与邮箱命令如下（适用于所有项目）：

- 配置全局用户名：`git config --global user.name "用户名"`（跟踪 who 修改记录）
- 配置全局邮箱：`git config --global user.email "邮箱"`（多人开发间的沟通）

git status：查看文件的状态

- 查看某个文件的状态：`git status 文件名`
- 查看当前路径所有文件的状态：`git status`

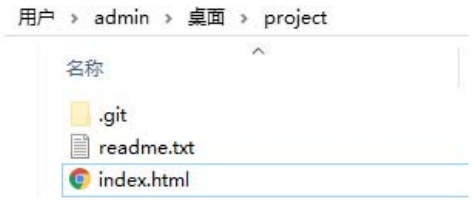
git add：将工作区的文件保存到暂缓区

- 保存某个文件到暂缓区：`git add 文件名`
- 保存当前路径的所有文件到暂缓区：`git add .`（注意，最后是一个点 `.`）

git commit：将暂缓区的文件提交到当前分支

- 提交某个文件到分支：`git commit -m "注释" 文件名`
- 保存当前路径的所有文件到分支：`git commit -m "注释"`

首先我们在工作区中，添加两个新的文件：readme.txt、index.html。如下图所示：



然后我们进入 Git 终端，输入 `git status` 命令查看文件状态如下图所示：

```
admin@Fengy MINGW64 ~/Desktop/project (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will
 be committed)

        index.html
        readme.txt

nothing added to commit but untracked files presen
t (use "git add" to track)
```

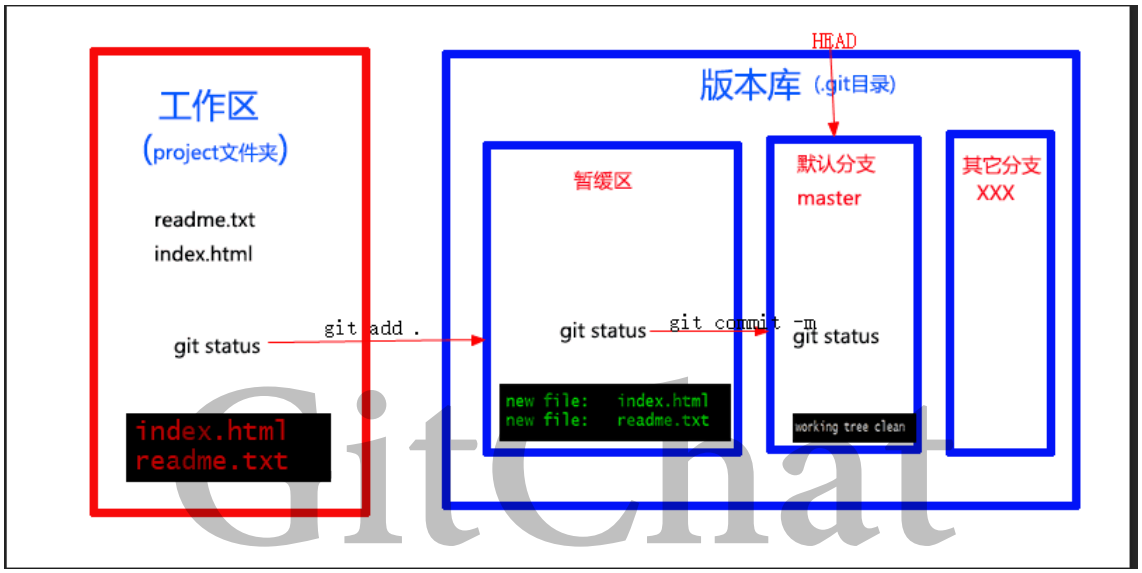
然后我们输入 `git add .` 命令 将工作区的文件保存到暂缓区，并输入 `git status` 命令再次查看文件状态如下图所示：

```
admin@Fengy MINGW64 ~/Desktop/project (master)
$ git add .

admin@Fengy MINGW64 ~/Desktop/project (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   index.html
        new file:   readme.txt


admin@Fengy MINGW64 ~/Desktop/project (master)
$
```

我们再次输入 `git commit -m""命令"添加了新文件"` 将暂缓区的文件提交到当前分支，如下图所示：

```
admin@Fengy MINGW64 ~/Desktop/project (master)
$ git commit -m "添加了新文件"
[master (root-commit) 92f8cae] 添加了新文件
 2 files changed, 1 insertion(+)
 create mode 100644 index.html
 create mode 100644 readme.txt

admin@Fengy MINGW64 ~/Desktop/project (master)
$ git status
On branch master
nothing to commit, working tree clean

admin@Fengy MINGW64 ~/Desktop/project (master)
$
```

上述 Git 命令执行分析:



git log：查看文件的修改日志

在工作区再新增 git.txt 文件，并将该文件添加到主分支（执行上述命令），然后输入 `git log` 命令，如下图所示，我们可以清楚的看到什么时候谁（**who**）干了什么事。

```
$ git log
commit 9ab054f6dbd96f620887ebd1ba7acc8fd36be75c (HEAD -> master)
Author: Fengy <fengy@163.com>
Date:   Thu Oct 25 00:54:28 2018 +0800

    新增了git.txt

commit 92f8cae41a07ad8e4cf7b7f2ec23cf5bb5d3a10a
Author: Fengy <fengy@163.com>
Date:   Wed Oct 24 22:03:57 2018 +0800

    添加了新文件
```

git reflog：查看分支引用记录（能够查看所有的版本号）

输入 `git relog` 命令，我们可以清楚的看到版本号下对应做了什么事（以简短的方式查看日志）。

```
$ git reflog
9ab054f (HEAD -> master) HEAD@{0}: commit: 新增了git.txt
92f8cae HEAD@{1}: commit (initial): 添加了新文件
```

git diff：查看文件最新改动的地方

我们为工作区中的文件 index.html 添加了一段代码如下所示，然后因为工作区文件 index.html 被修改，所以我们再次将修改后的文件提交到主分支中。

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8"/>
        <title>1024程序员节</title>
    </head>
    <body>

    </body>
</html>
```

然后我们再次修改代码，如下图所示：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8"/>
        <title>1024程序员节</title>
    </head>
    <body>
        <h1>gitchat</h1>
    </body>
</html>
```


然后我们输入 git diff 命令，如下图所示，绿色的代码被最近一次被修改的代码，还未受到 Git 版本控制。

```
$ git diff
diff --git a/index.html b/index.html
index 63a5a95..a01e9e2 100644
--- a/index.html
+++ b/index.html
@@ -5,6 +5,6 @@
                <title>1024程序员节</title>
        </head>
        <body>
-
+               <h1>gitchat</h1>        <-- 修改的代码
        </body>
 </html>
\ No newline at end of file
```

git reset：版本回退（建议加上--hard 参数，Git 支持无限次后悔）

- 回退到上一个版本：git reset --hard HEAD^
- 回退到上上一个版本：git reset --hard HEAD^^

- 回退到上 N 个版本：`git reset --hard HEAD~N`（N 是一个整数）
- 回退到任意一个版本：`git reset --hard 版本号`

注意：Git 命令团队开发与个人开发命令基本相同，这里不做详细介绍。

GitHub 简单使用

Git 命令补充：

`git clone`：下载远程仓库到本地

- 下载远程仓库到当前路径：`git clone 仓库的URL`
- 下载远程仓库到特定路径：`git clone 仓库的URL 存放仓库的路径`

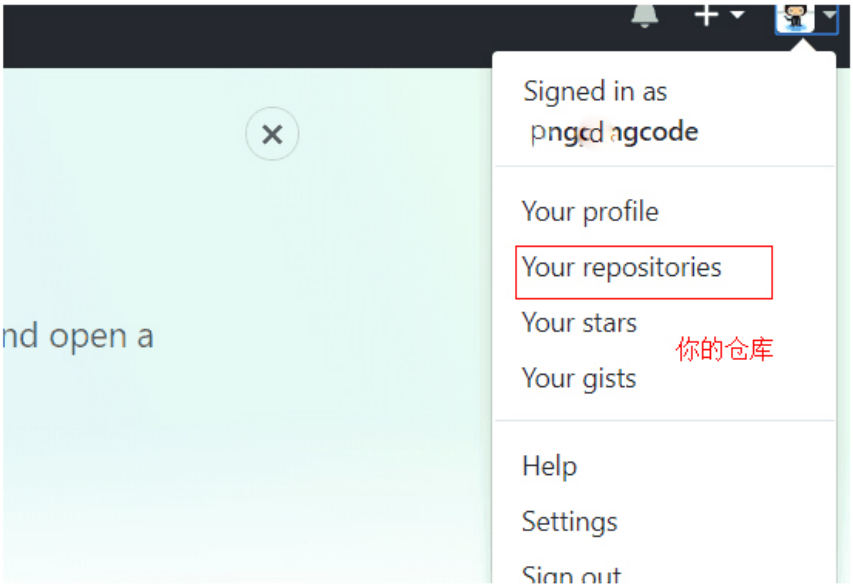
`git pull`：下载远程仓库的最新信息到本地仓库

`git push`：将本地的仓库信息推送到远程仓库

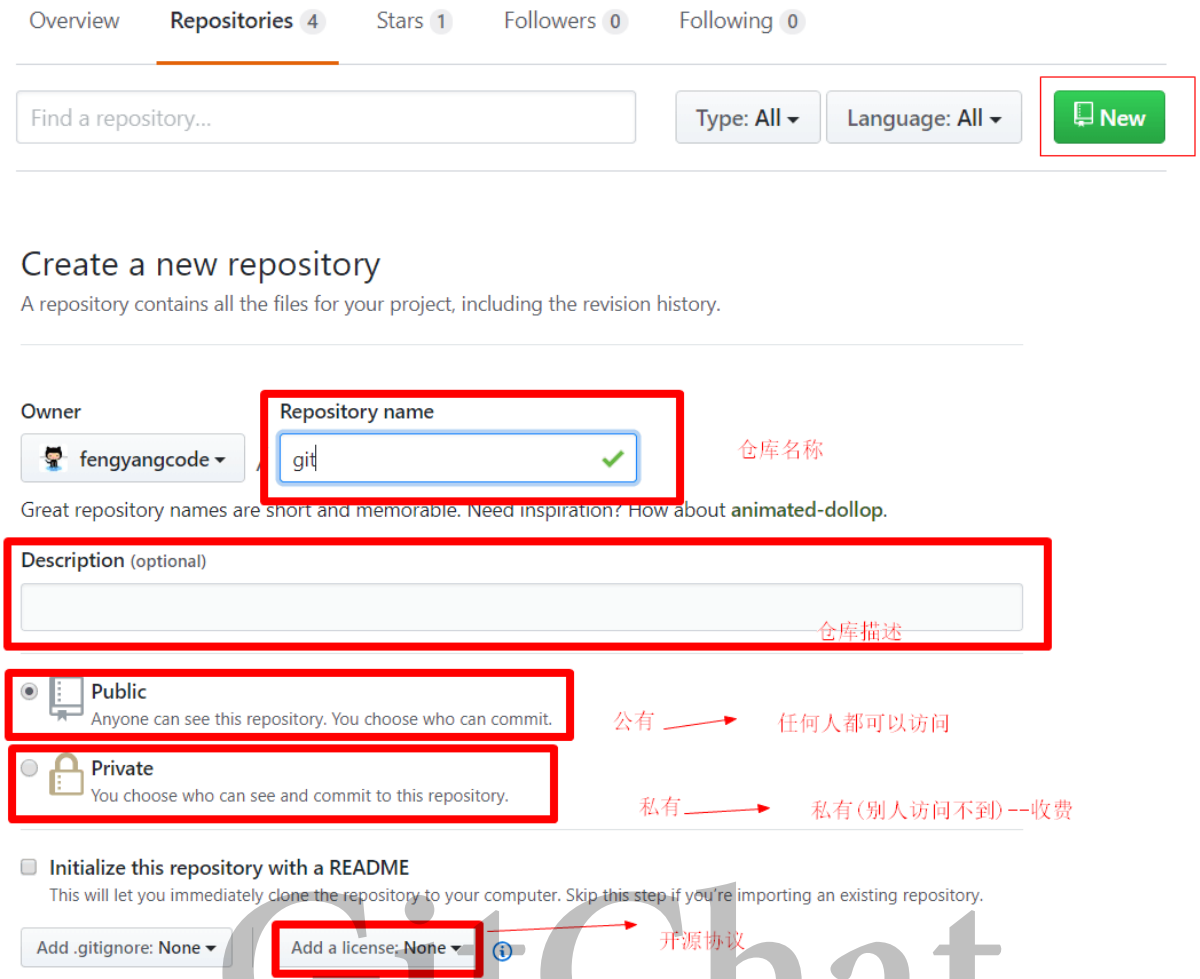
- 提交时如果远程仓库有其它人提交的最新代码，必须先 pull，再提交

> GitHub 是一个面向开源及私有软件项目的托管平台，因为只支持 Git 作为唯一的版本库格式进行托管，故名 GitHub。

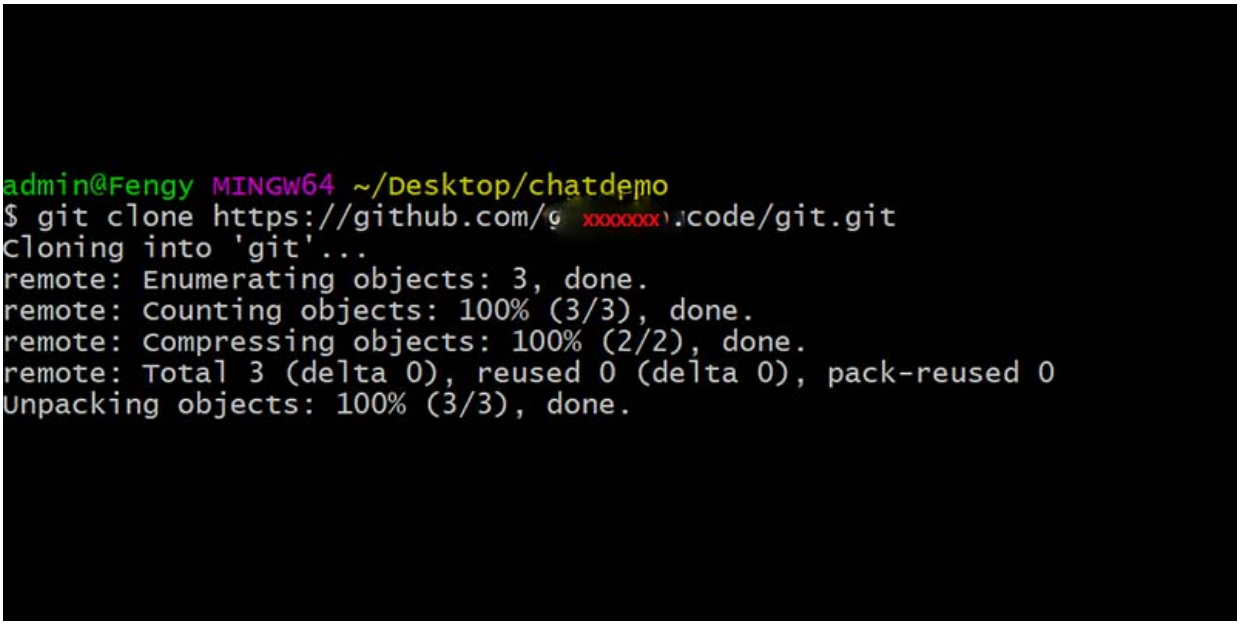
使用 GitHub 之前，我们需要去 GitHub 官网注册一个属于自己的账号，然后登录你的 GitHub 账号。选择你的仓库：



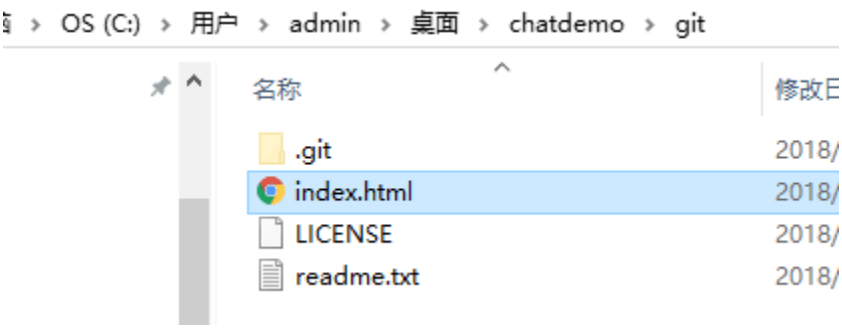
点击右侧 New 新建一个仓库，并给改仓库起一个名字，可以描述该仓库是什么项目，如下图所示：



然后我们在桌面创建一个 chatdemo 文件夹，以 git bash 终端打开，输入 `git clone 仓库的 URL 下载远程仓库到本地，即 git clone https://github.com/xxxcode/git.git`：



然后我们进入 chatdemo 下的工作区，新建 readme.txt 与 index.html 文件：



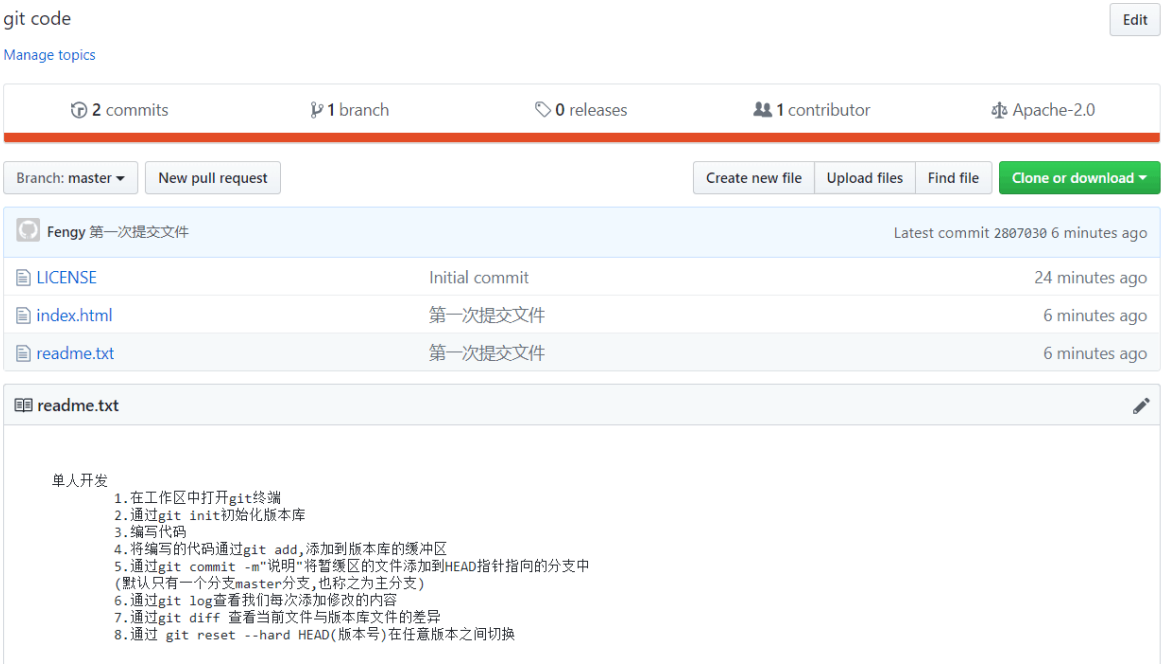
使用 Git 命令将工作区的文件提交到暂缓区，并再次提交到主分支：

```
$ 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)

        index.html
        readme.txt

nothing added to commit but untracked files present (use "git add" to track)

admin@Fengy MINGW64 ~/Desktop/chatdemo/git (master)
$ git add .

admin@Fengy MINGW64 ~/Desktop/chatdemo/git (master)
$ git commit -m"第一次提交文件"
[master 2807030] 第一次提交文件
 2 files changed, 20 insertions(+)
 create mode 100644 index.html
 create mode 100644 readme.txt
```


`git push`：将本地的仓库信息推送到远程仓库：

```
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 769 bytes | 384.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/fengyangcode/git.git
   d707888..2807030  master -> master
```

`git push` 提交之后，远程仓库新增了 2 个文件，如下图所示：



Chat 结语

通过本次 Chat 我们学习了 Git 的工作原理，以及 Git 版本控制管理代码的好处，同时也学习了 Git 的常用命令，最后我们通过 Git 版本控制的命令将本地的代码提交到 GitHub 远程仓库中。本次学习主要是通过学习 Git 工作原理，通过 Git 命令将代码提交到远程仓库中。我相信大家理解了 Git 工作原理之后，学习 Git 中的其它命令也会很容易，这里就不逐一的对 Git 各个命令做详细讲解。

本次 Chat 分享到此结束，下期 Chat 我们将探究更多 Git 内容。如果你有任何问题，欢迎读者圈留言。也欢迎各位关注个人微信公众号**IT程序员的日常**