

# 开源项目阅读与管理—概述

## 一、开源项目阅读与管理-概述

### 开源项目阅读与管理简介



《开源项目阅读与管理》学习这门课程之前，我们要搞清楚一些概念：什么是开源、开源项目、开源软件、开源协议与 GitHub？

步骤 1：相关概念简介

步骤 2：开源协议

步骤 3：Git 简介

步骤 4：安装 Git

步骤 5：设置 Git

步骤 6：安装小乌龟 TortoiseGit

## 步骤 1：相关概念简介

### 1、开源（open source）：

在互联网领域，可以简单理解为是“开放源代码”的简称。通常来说，指的是将软件项目的源代码向大众开放，允许大众获取、使用、修改和发行。在硬件等其它领域也可以套用开源的概念，本文暂不提。

它被非盈利软件组织(美国的 Open Source Initiative 协会)注册为认证标记，并对其进行了正式的定义，用于描述那些源码可以被公众使用的软件，并且此软件的使用、修改和发行也不受许可证的限制。

### 2、开源项目（open source project）：

简单理解就是，开放源代码的软件项目。可以认为，开源项目的产出物是软件程序，这包括，一个开源项目，可以不断对一款开源软件进行维护和升级，或者有可能在一个开源项目中，产出多款不同的开源软件（但很可能彼此有联系）。开源项目的所有者不属于任何组织或个人。在遵守开源协议的前提下，开源产品可通过修改代码定制成属于自己的个性化产品。

### 3、开源软件（open source software）：

直接的字面意思是公开源代码的软件，也就是说，如果软件的源代码是开源的，那么这个软件就可以称之为开源软件。不过，对于很多商业公司来说，开源软件，只能看作是某个开源项目给出的“软件示例”而已，因为软件的源代码已经开放出来了，那么这些商业公司，完全可以根据自己的需要，基于这个示例，修改或衍生出真正适合自己的软件产品。

4、开源社区（open source community），为某个开源项目的开发成员提供的一个学习和交流的空间。由于开源项目常常需要散布在全世界的开发人员共同参与推进，所以开源社区就成了他们沟通交流的必要途径。

5、开源协议（Open Source License），是指开源软件所遵循的许可协议，获得了开源软件的用户，需要在该协议的允许范围内对软件的源代码进行使用、修改和发行（包括以盈利为目的商业发行）。

5、GitHub，是一个面向软件项目的托管平台，可以用于托管各种类型的软件项目，包括开源项目和私有项目。由于大量开源项目基于 GitHub 进行托管，方便来自世界各地的开发人员共同工作以及获取开源软件，所以在开源项目领域，GitHub 的影响力很大，是开源项目的首选托管平台。

6、用一句话串联一下这几个概念：在 GitHub 上，发布了一个软件项目，是开源的，这个开源项目会产出一款使用了 MIT 开源协议的开源软件供大家免费获取，如果想加入这个开源项目共同工作，可以来这个项目的开源社区参与讨论，网址是：<http://xxx.xxx.xxx>

总结：可以说，开源的意义主要在于合作，通过合作，才能形成围绕某个开源项目的软件生态。例如最早也是最著名的开源项目之一“Linux”操作系统，其成功很大程度上是依靠开源社区为其提供的源源不断的代码支持，使其从当年一个人的“小项目”日益壮大起来。

相应的，如果不准备和不允许他人参与自己的项目中，那么这样的项目就是私有项目，而即便私有项目所产出的软件也能够免费提供出来给大家使用，但这样的软件也只能称之为免费软件，而非开源软件。

## 步骤 2：开源协议

开源软件虽然通常都是免费的，但并不等于软件的开发者们（开源社区）完全放弃了自己的权利和对软件的控制。为了保证开源软件不被一些商业机构或个人窃取，成为他们不劳而获的牟利工具，并影响开源项目的长远发展，开源社区开发出了各种开源协议（具有法律效力），用于维护自己的软件版权。

在开源协议里面，会详尽表述使用者在获得代码后拥有的权利和义务，包括可以进行何种操作，而何种操作又是被禁止的。

开源协议种类非常之多，并且同一款协议会有很多个变种版本。开源协议规定得太宽松，会导致开发者们丧失对开源软件的很多权利，而太严格又不便于使用者们的使用以及开源软件的传播。

常见的开源协议有：GPL、LGPL、BSD、Apache 2.0、MIT

### 1、GPL

Linux 就是采用了 GPL 协议。GPL 协议允许代码的获取、代码的免费使用和引用、代码的修改和衍生，但要求对修改和衍生代码的进行开源，不允许修改和衍生的代码做为私有闭源的商业软件发布和销售。

这也就是为什么我们能使用各种免费的 linux 操作系统，以及 linux 上各种各样的由个人，组织，以及商业软件公司开发的免费软件了。

GPL 协议的主要内容是，只要在一个软件中使用到了包含 GPL 协议的产品（GPL 类库），则该软件产品必须也采用 GPL 协议，既必须也是开源和免费，这就是所谓的“传染性”。

由于 GPL 严格要求使用了 GPL 类库的软件产品必须使用 GPL 协议，对于使用 GPL 协议的开源代码，商业软件或者对代码有保密要求的部门就不适合集成/采用作为类库和二次开发的基础。

### 2、LGPL

LGPL 是 GPL 的一个主要为类库使用设计的开源协议。和 GPL 不同，LGPL 允许商业软件通过类库引用(link)方式使用 LGPL 类库而不需要开源商业软件的代码。这使得采用 LGPL 协议的开源代码可以被商业软件作为类库引用并发布和销售。

但是如果修改 LGPL 协议的代码或者衍生，则所有修改的代码，涉及修改部分的额外代码和衍生的代码都必须采用 LGPL 协议。因此 LGPL 协议的开源代码很适合作为第三方类库被商业软件引用，但不适合希望以 LGPL 协议代码为基础，通过修改和衍生的方式做二次开发的商业软件采用。

### 3、BSD

BSD 开源协议是一个给予使用者很大自由的协议。开发者可以自由使用和修改源代码，也可以将修改后的源代码作为开源或者专有软件再发布。但是有以下几个要求：

如果再发布的产品中含有源代码，则在源代码中必须带有原来代码中的 **BSD** 协议。如果再发布的只是二进制类库/软件，则需要在类库/软件的文档和版权声明中包含原有代码中的 **BSD** 协议。

不可以用开源代码的作者/机构名字和原来产品的名字做市场推广。

**BSD** 代码鼓励代码共享，但需要尊重代码作者的著作权。**BSD** 由于允许使用者修改和重新发布代码，也允许使用或在 **BSD** 代码上开发商业软件发布和销售，因此是对商业集成很友好的协议。而很多的公司企业在选用开源产品的时候都首选 **BSD** 协议，因为可以完全控制这些第三方的代码，在必要的时候可以修改或者二次开发。

#### 4、Apache 2.0:

**Apache Licence 2.0** 的简称，**Apache Licence** 是著名的非盈利开源组织 **Apache** 采用的协议。该协议和 **BSD** 类似，同样鼓励代码共享和最终原作者的著作权，同样允许源代码修改和再发布。但是也需要遵循以下条件：需要给代码的用户一份 **Apache Licence**。

如果修改了代码，需要再被修改的文件中说明。

在衍生的代码中（修改和有源代码衍生的代码中）需要带有原来代码中的协议，商标，专利声明和其他原来作者规定需要包含的说明。

如果再发布的产品中包含一个 **Notice** 文件，则在 **Notice** 文件中需要带有 **Apache Licence**。你可以在 **Notice** 中增加自己的许可，但是不可以表现为对 **Apache Licence** 构成更改。

**Apache Licence** 也是对商业应用友好的许可，使用者也可以在需要的时候修改代码来满足并作为开源或商业产品发布/销售。

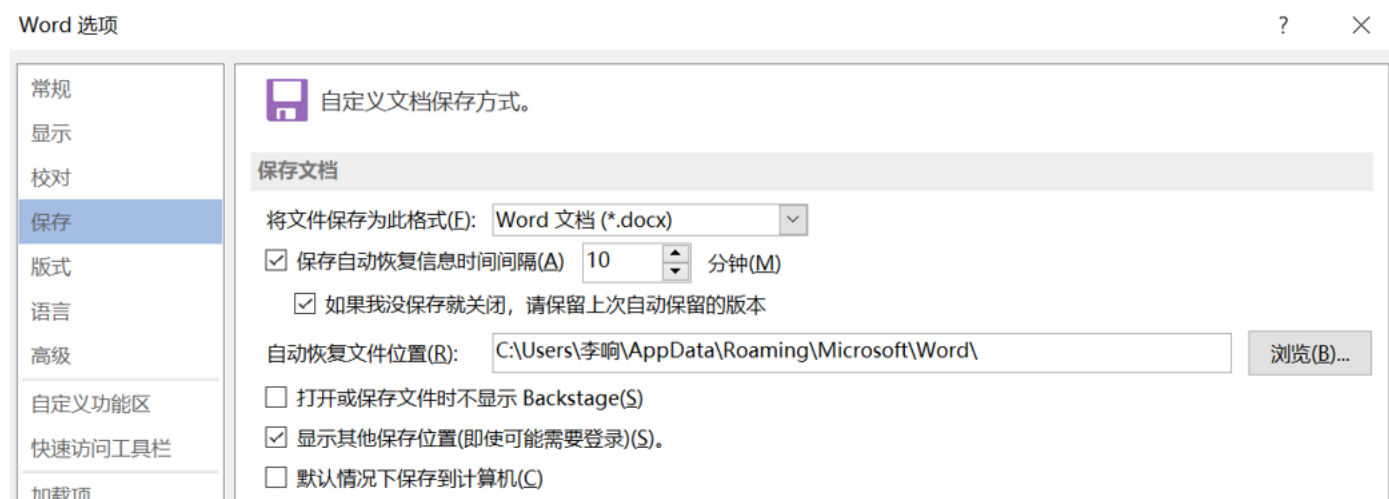
#### 5、MIT:

**MIT** 是和 **BSD** 一样宽泛的许可协议,源自麻省理工学院（**Massachusetts Institute of Technology, MIT**）。使用 **MIT** 协议的开源软件作者只保留版权,而对使用者无任何其它限制。**MIT** 与 **BSD** 类似，但是比 **BSD** 协议更加宽松，是目前最少限制的协议。这个协议唯一的条件就是在修改后的代码或者发行包中包含原作者的许可信息，且适用于商业软件。使用 **MIT** 的软件项目有：**jquery**、**Node.js**。

## 步骤 3: GIT 简介

Git 是什么？

场景一：赵日天写毕业论文，辛辛苦苦写的 word 文档没保存，电脑蓝屏死机了，东西全丢了。他来问我，有没有相关的**备份功能**或者说是自动保存？我告诉他 word 是有备份（自动保存）功能的，时间什么的要自己定义一下。



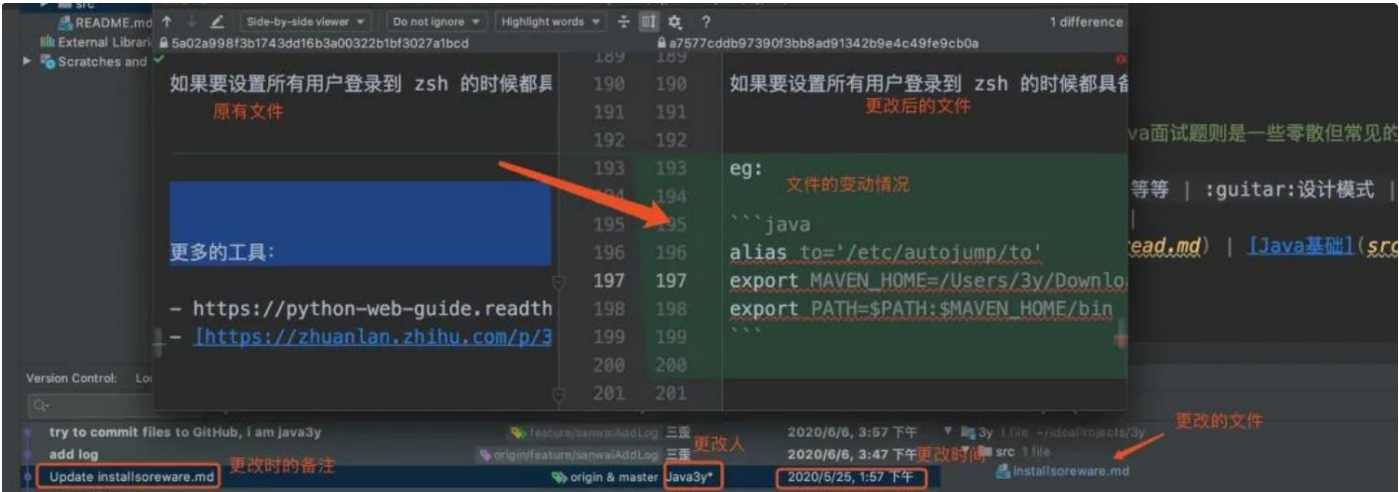
这种情况要是在程序员的手上感觉发生的概率会低一点，程序员习惯会按 ctrl+s。甚至有的时候，看着看着网页还会按 ctrl+s。看着游戏界面也会按 ctrl+s，不过像我们写代码的工具 (IntelliJ IDEA) 都不用自己手动保存了.... 在生活中，一个好的习惯就是常于备份，而 **Git 就是来帮助我们来备份代码的**。

场景二：赵日天写毕业论文，因为要修改很多次，就有很多版本。

赵日天的毕业论文		
	名称	修改日期
	毕业论文.docx	2022/5/
	毕业论文改.docx	2022/5/
	毕业论文改1.docx	2022/5/
	毕业论文改2.docx	2022/5/
	毕业论文改3.docx	2022/5/
	毕业论文改完成版.docx	2022/5/
	毕业论文改完成版1.docx	2022/5/
	毕业论文改完成版2.docx	2022/5/
	毕业论文改最终版.docx	2022/5/
yth	毕业论文改最终版1.docx	2022/5/
管理	毕业论文改最终版2.docx	2022/5/
设计	毕业论文改最终打死不改版.docx	2022/5/
板	毕业论文改最终打死不改版1.docx	2022/5/
	毕业论文改最终打死不改版2.docx	2022/5/

因为我们怕在原来的基础改错了东西，没法恢复，所以，我们可能会有多个「毕业论文」的文件。而我们写代码的时候本身就是「多人协作」的，修改是无法避免的，我们不希望有多个文件的产生，又希望能够记录每次更改的内容。

更改的内容指的就是：基于原有的基础上更改了什么，以及提交者是谁。这样子，我们就没法甩锅了。说白了就是，我们能知道的文件被改了什么，以及谁改了



也许你遇到这样的情况：你做出了一个软件并上线给到用户在用，用户在用的过程中，哪些功能体验不好，你要去改善，这时候你要在原有的代码基础上做修改，但改之前，得做下备份，别改崩了呵呵。这时候你就有了文件 A，当你将用户提给你的这些方面改善好了，这时候你就有了修改文件 B。

当又有用户觉得其他方面体验不好，你还得改，又多了文件 C，依次一直文件 D、E、F、G …直到无限。

这时候问题来了，假如当某一次改动上线后出现严重 bug，在时间不允许时，你就得拿出最近一次备份的文件先顶一项（俗称版本回滚），这个其实还好办。但假如不是最近一次，而是很久之前的，你这时候就需要去文件 ABCDEFG…一个个去找，而此时面对着 ABCD…的文件，你完全想不起来，当初备份的这个是啥？

如果有一个软件，不但能自动帮我记录每次文件的改动，还可以让同事一起协作编辑，这样就不用自己管理一堆类的文件了，也不需要把文件传来传去，如果想查看某次改动，只需要在软件里看一下就清清楚楚，这时候聪明的先贤们就想出了这么一个办法，叫**版本管理器**

版本	文件名	用户	说明	日期
1	service.doc	张三	删除了软件服务条款5	7/12 10:38
2	service.doc	张三	增加了License人数限制	7/12 18:09
3	service.doc	李四	财务部门调整了合同金额	7/13 9:51
4	service.doc	张三	延长了免费升级周期	7/14 15:17



「多人协作」是在同一个目录下对文件修改的，然后可以看到彼此改了什么。那你是在你的电脑上改，你的同事是在他的电脑上改的，你们是怎么**看到**彼此改了什么？你和你同事之间又不联网，怎么知道对方改了什么

程序员在代码写到一定程度下（比如说这个功能我们做完了、也可能做得差不多了），我们会把**当前版本**提交到**远程仓库**上，提交到远程仓库后，即便我们电脑坏了，我们可以从远程仓库再把这份数据拉取下来。”

我们从远程仓库拉取代码的时候**除了**会把有变动的代码同步到自己的电脑上，还会把**所有修改的记录也**同步到自己的电脑上。所以说，我们会知道彼此修改的内容。

版本控制软件，在这个过程中会**记录每次修改的内容，谁改了什么东西。谁改错了，谁要背锅，一个都不能跑！**而且如果出错了，可以通过这个软件回到想要的版本

目前市面上优秀的版本管理器有两个：

①集中式的 SVN

②分布式的 Git

版本管理器既然是帮我们做备份的，那么问题来了，备份的文件放在哪？SVN 既然是集中式的，那肯定就是中央集权，有一个统一的文件服务器存放这些文件，每个人单独与之做沟通，但集中式的注定了当作为核心的 SVN 服务器挂掉之后，所有人都没法干活。而 Git，它高明之处在于，人人平等，每个人都有一个完全属于自己的独立仓库，尽管它也有一个中间的交互服务器，但那仅仅只是作为一个中间媒介，当中间节点挂了，你本机有一整个的图书馆，不会对你有过大的影响。接下来我们系统的学习一下 Git。

**总结：备份功能、版本管理、多人协作。**

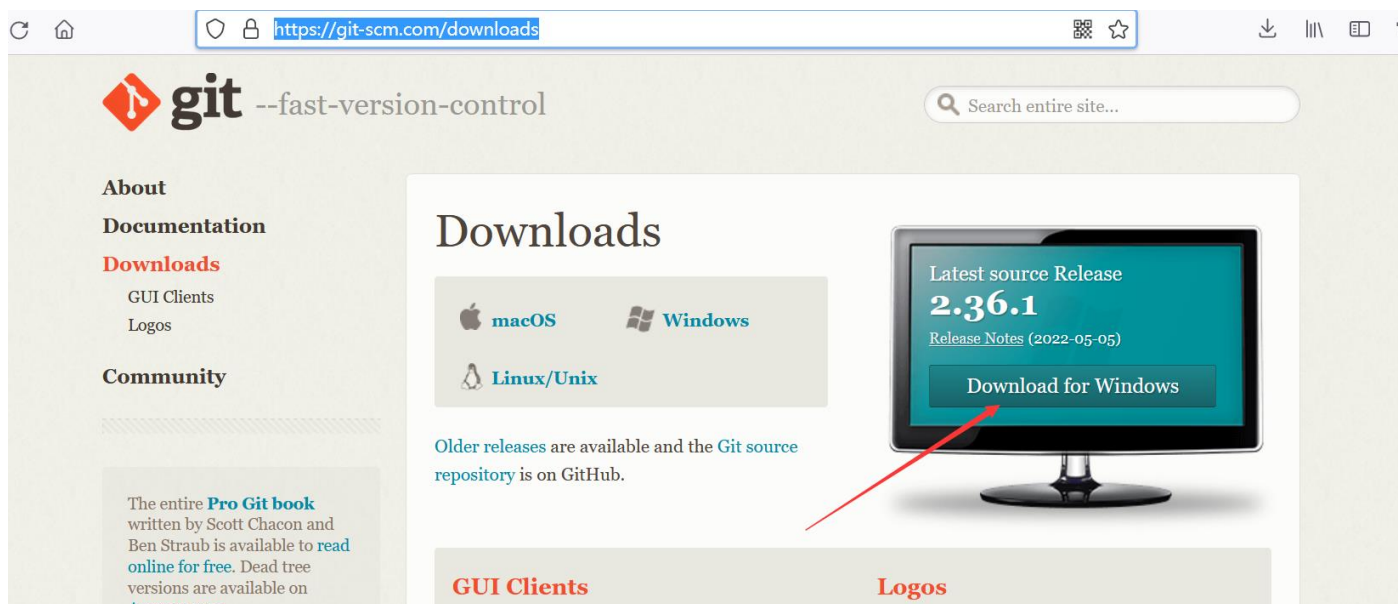
## 步骤 4：安装 GIT

最早 Git 是在 Linux 上开发的，随着应用广泛，程序员将它移植到各个系统操作平台上，根据你使用的操作系统我们来安装 Git，下面以 Windows 系统举例：

下载 Git: <https://git-scm.com/downloads>

淘宝镜像: <https://registry.npmirror.com/binary.html?path=git-for-windows/>

从 Git 官网直接下载安装程序，然后按默认选项安装即可。



选择 64 位安装程序下载：

## Download for Windows

**Click here to download** the latest (**2.36.1**) **64-bit** version of **Git for Windows**. This is the most recent **maintained build**. It was released **21 days ago**, on 2022-05-09.

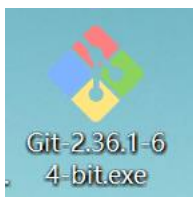
### Other Git for Windows downloads

#### Standalone Installer

**32-bit Git for Windows Setup.**

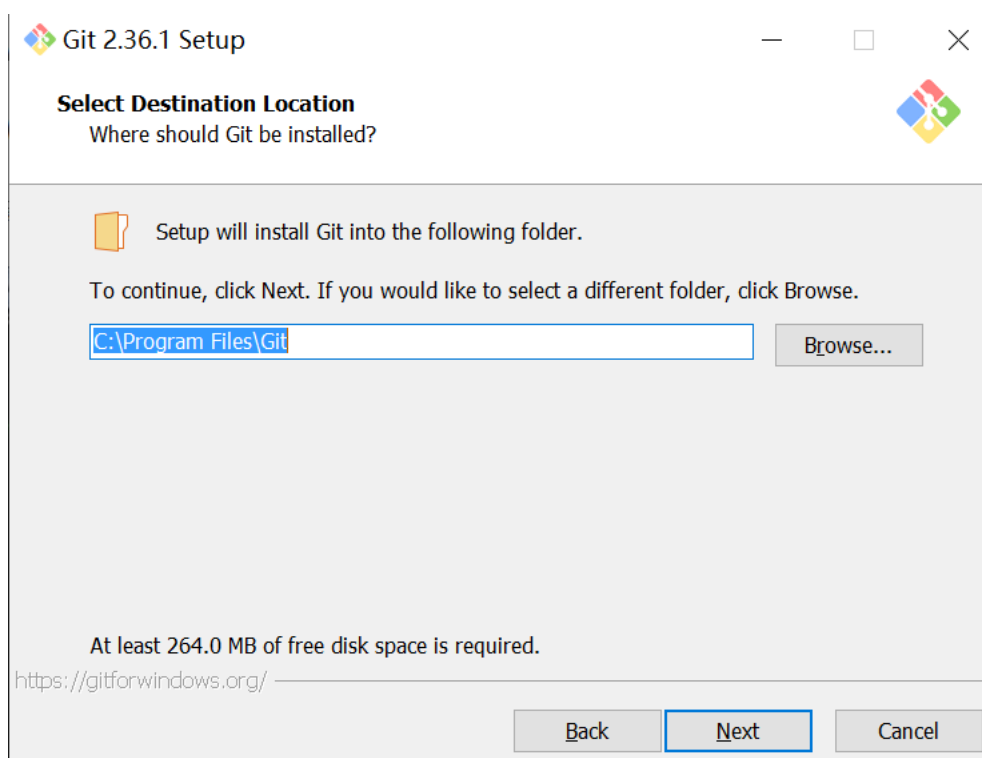
**64-bit Git for Windows Setup.**

下载完成后的安装包: Git-2.36.1-64bit.exe





双击按照默认选项安装，点击“next”



安装完成后，在开始菜单里找到“Git” -> “Git Bash”，蹦出一个类似命令行窗口的东西，就说明 Git 安装成功！



安装完在桌面点鼠标右键，弹出的菜单栏里也有“Git Bash Here”相关的选项



点击“Git Bash”进入命令行窗口

MINGW64:/c/Users/李响

```
李响1@Lixiang MINGW64 ~ (master)
$
```

## 步骤 5: 设置 GIT

Git 是分布式版本控制系统，所以，每个用户都必须提供标识：你的名字和 Email 地址。用户名和邮箱地址是本地 git 客户端的一个变量，不随 git 库而改变。每次 commit 都会用用户名和邮箱记录。


1、设置用户名和邮箱，在命令行输入：

```
李响1@LiXiang MINGW64 ~ (master)
$ git config --global user.name "lixiang"
```

```
李响1@LiXiang MINGW64 ~ (master)
$ git config --global user.email "lixiangood@qq.com"
```

2、检查是否设置成功

输入 git config user.name 然后回车，如果设置成功了就会显示你刚刚设置的用户名，同理，可以用 git config user.email 来查看你设置的邮箱

 MINGW64:/c/Users/李响

```
李响1@LiXiang MINGW64 ~ (master)
$ git config user.name
lixiang
```

```
李响1@LiXiang MINGW64 ~ (master)
$ git config user.email
lixiangood@qq.com
```

注意：git config 命令的--global 参数，用了这个参数，表示你这台机器上所有的 Git 仓库都会使用这个配置，当然也可以对某个仓库指定不同的用户名和 Email 地址。

## 步骤 6：安装小乌龟 TORTOISEGIT

装好 Git，能使用纯命令行当然好，不过新手建议装个小乌龟 TortoiseGit

访问：<https://tortoisegit.org/download/>

下载安装

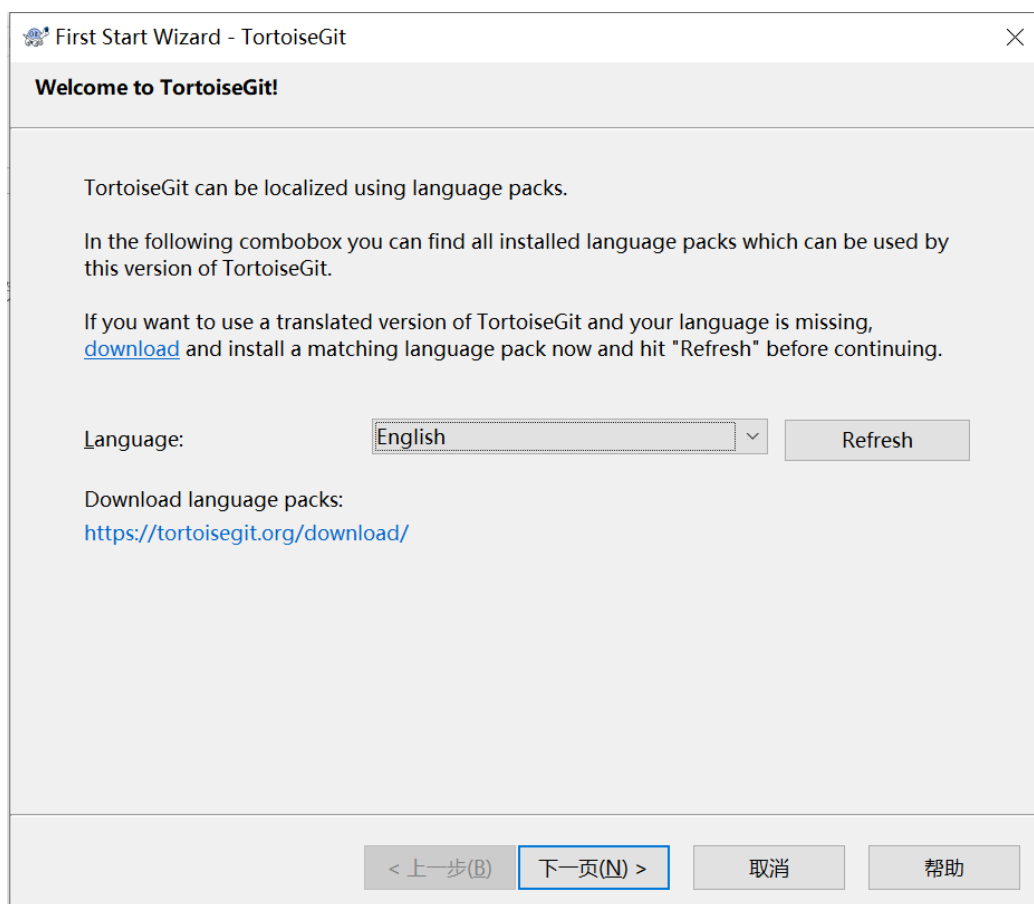


双击运行安装包：TortoiseGit-2.13.0.1-64bit.msi

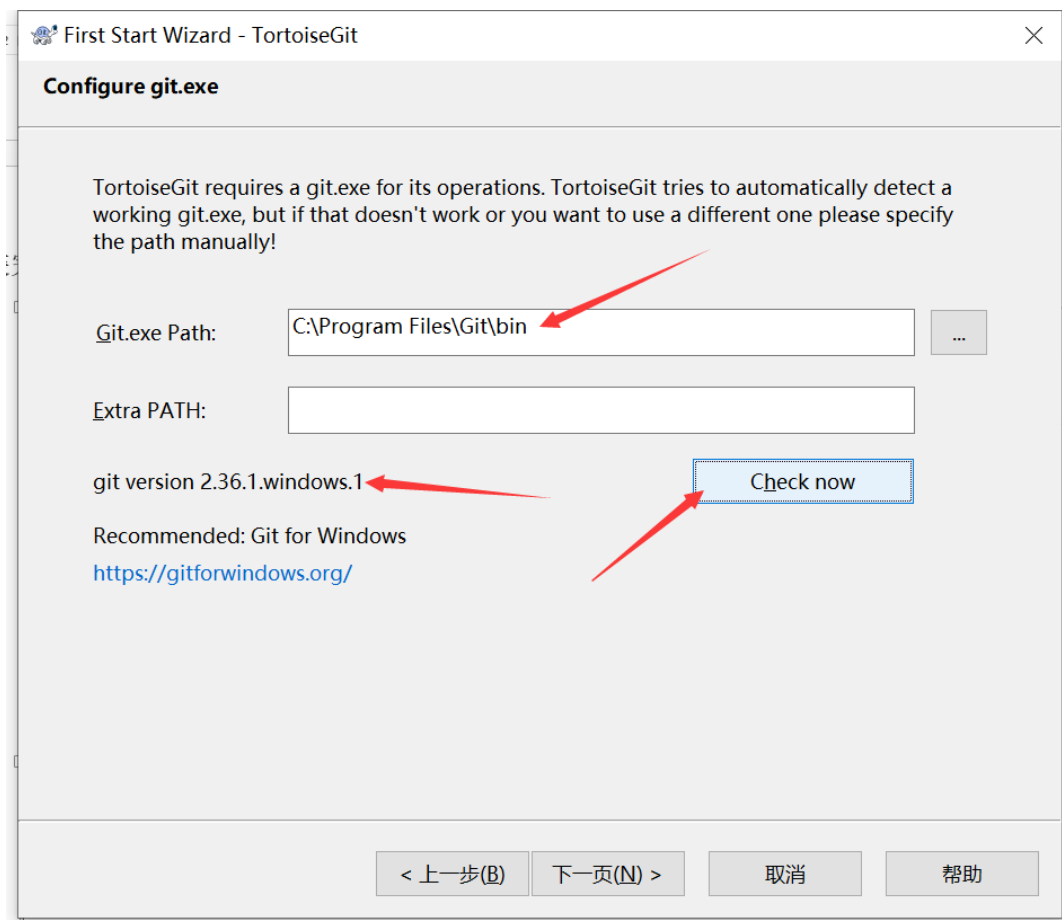
点击“Next”进行默认安装：



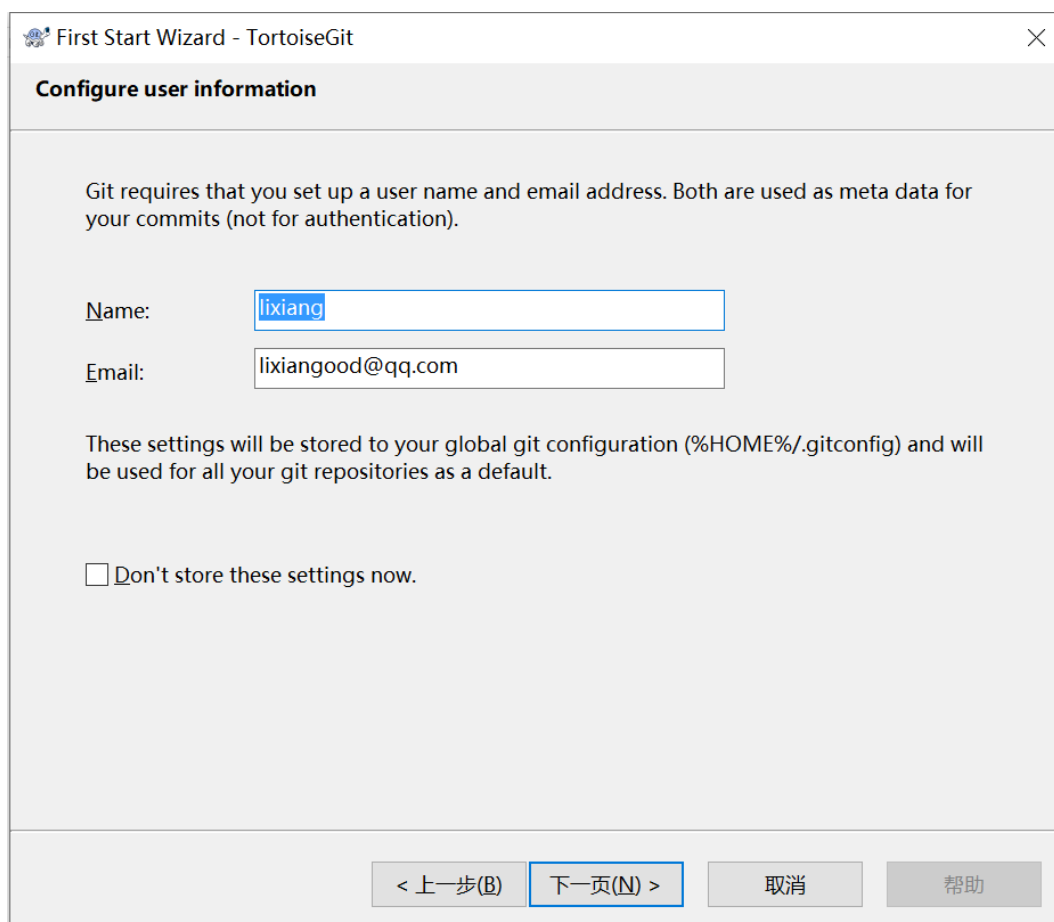
安装完成进行首次运行 TortoiseGit 设置:



设置 git.exe 可执行文件的位置, 设置完点击 Check now 进行检查:



设置用户名和邮箱，点击完成。



在桌面上点击右键，菜单上出现下面的选项，代表安装成功。



这个是装完git之后的图形化  
GUI和命令式Bash

这是小乌龟可视化Git