# Git

git是一款开源免费的版本控制软件，可以自由下载和使用。Git具有分布式版本控制的能力，可以在本地进行版本控制和提交，无需连接到服务器。Git拥有庞大的开源社区和生态系统，可以获取到大量的开源工具和插件。

# Git的本地化部署

Git有很多本地化部署方案，包括：

## Git官方提供的Git仓库

Git官方提供Git仓库的下载和安装，可以在本地搭建Git服务器，实现Git的本地部署。这种方式需要在本地安装和配置Git服务器，需要一定的技术水平。

## GitLab

GitLab是一个开源的Git仓库管理系统，可以在本地部署GitLab服务器，实现Git的本地部署。GitLab支持项目管理、代码审核、持续集成等功能，适用于团队协作开发。

## Gogs

Gogs是一个轻量级的Git仓库管理系统，可以在本地部署Gogs服务器，实现Git的本地部署。Gogs支持SSH、HTTP协议，适用于个人开发者或小团队。

## Bitbucket Server

Bitbucket Server是Atlassian公司开发的Git仓库管理系统，可以在本地部署Bitbucket服务器，实现Git的本地部署。Bitbucket Server支持Git和Mercurial版本控制系统，适用于团队协作开发。

# 准备Git环境

## 安装git

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

## 配置git

打开git bash

一般我们需要配置的有

user.name 当前用户的昵称

user.emal 当前用户的邮箱

http.proxy git使用http代理

https.proxy git使用https代理

我们本地部署只需要配置用户名和邮箱即可。

在git hash中，输入

git config –global user.name [你的昵称]

git config –global user.emal [你的邮箱]

即可。

我们还需要配置lfs

git bash 输入2

git lfs install

# Gogs的本地化部署

Gogs是一款轻量化的Git部署方案，适合小团队。

## 构建

我们需要gogs高版本的windows amd64版本，来启用lfs系统。

gogs的部署非常简单，一键开箱即用。

这里我选择自己从源码构建。

### 下载Go

Gogs (Go Git Service）使用go语言，我们需要go语言的构建环境。

<https://go.dev/doc/install>

安装完成后下载gogs源码

<https://github.com/gogs/gogs/releases/tag/v0.13.0>

解压后在目录运行go install

需要下载依赖，可能会有网络问题，可以选择把梯子开启tun mode，或者系统变量设置proxy。

### 运行web server

执行go install完成后执行gogs web, 启动服务器。

默认地址<http://localhost:3000>

会有创建引导



这里使用sqlite作为数据库，但是提示CGO\_ENABLE=0错误（官方构建不会出现这个问题），

我们需要在系统变量中增加一个这个字段，并且值设置为1.

重新构建，提示需要gcc环境。

### 下载安装mingw-win64

Gcc是linux上的c/c++构建工具。我们需要mingw-win64作为替代品。

<https://www.mingw-w64.org/downloads/#mingw-builds>

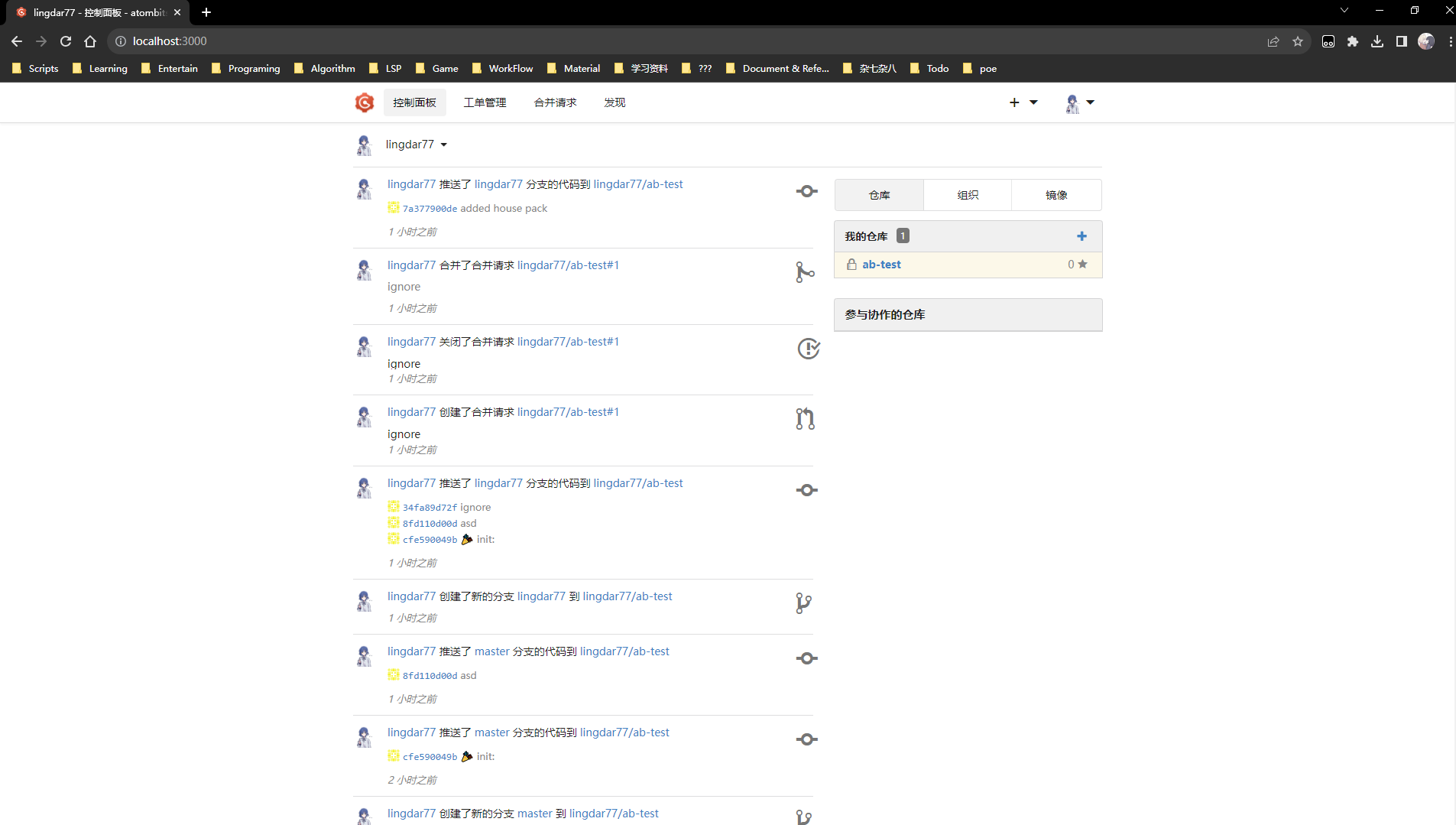
解压后，将其目录下的bin目录加入系统变量中的PATH中。

重新build，等待成功后重新启动服务器，配置好后即可使用git服务。

# Gogs的使用

登录gogs后，进入gogs的主页面。

gogs的页面布局类似github。



## 导航栏

Gogs的导航栏位于页面的顶部，包含了主要的导航链接，例如仪表板（Dashboard）、仓库（Repositories）、团队（Teams）、用户（Users）等。你可以通过导航栏快速访问各个功能模块。



## 仪表板

仪表板是Gogs的默认首页，提供了关于你的个人或组织的概览信息。在仪表板上，你可以查看最近的活动、仓库的概要信息、收到的问题（Issues）和合并请求（Pull Requests）等。



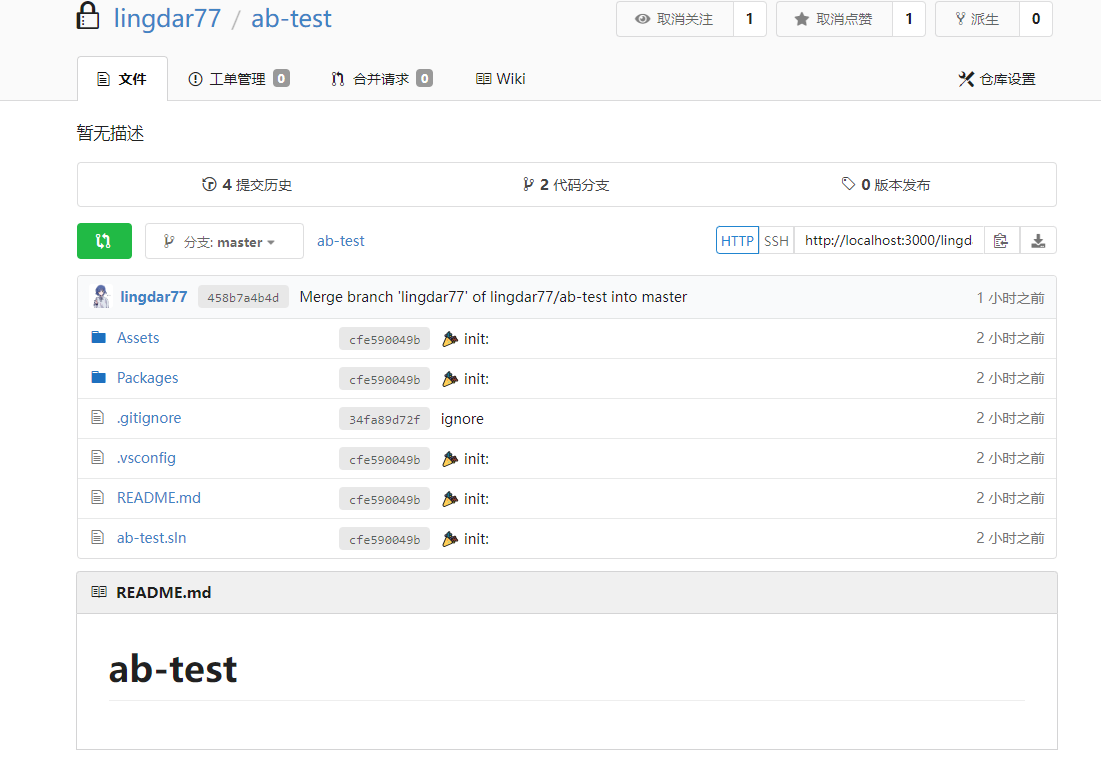
## 仓库列表

通过导航栏中的"Repositories"链接或仪表板上的"Your Repositories"模块，你可以进入仓库列表页面。仓库列表展示了你拥有或关注的仓库，并提供了搜索功能和筛选选项。



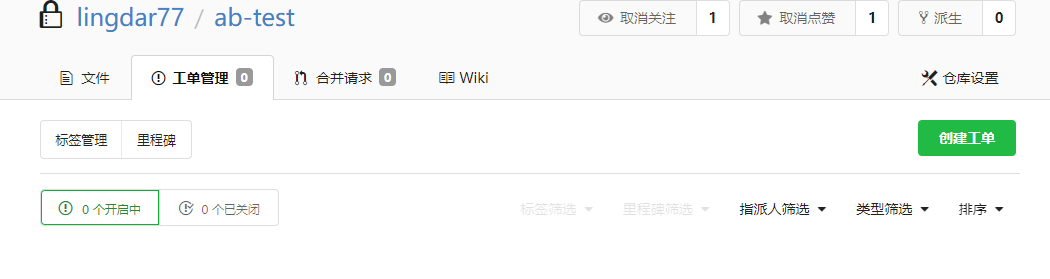
## 仓库详情页

当你点击进入一个具体的仓库时，会进入仓库的详情页。在该页面上，你可以查看仓库的代码、问题、合并请求、分支等信息。页面上也有操作按钮，例如克隆仓库、创建问题、创建合并请求等。



## 问题页面

在仓库详情页的"Issues"标签下，你可以查看和管理与该仓库相关的问题。问题页面提供了问题列表、创建新问题、编辑问题和评论等功能。



## 合并请求页面

在仓库详情页的"Pull Requests"标签下，你可以查看和管理与该仓库相关的合并请求。合并请求页面提供了合并请求列表、创建新的合并请求、审查和合并等功能。



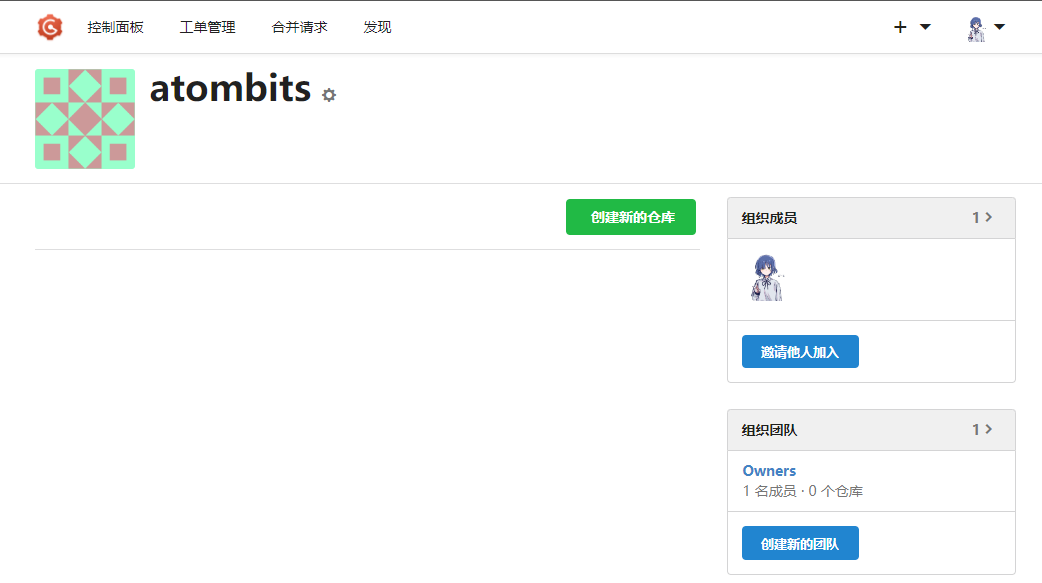
## 用户页

通过导航栏中的"Users"链接，你可以访问用户列表页面。用户列表展示了所有的用户，并提供了搜索和筛选选项。



## 团队页面

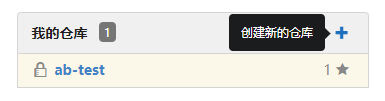
通过导航栏中的"Teams"链接，你可以访问团队列表页面。团队列表展示了所有的团队，并提供了搜索和筛选选项。在团队页面上，你可以查看团队成员、仓库和权限等信息。

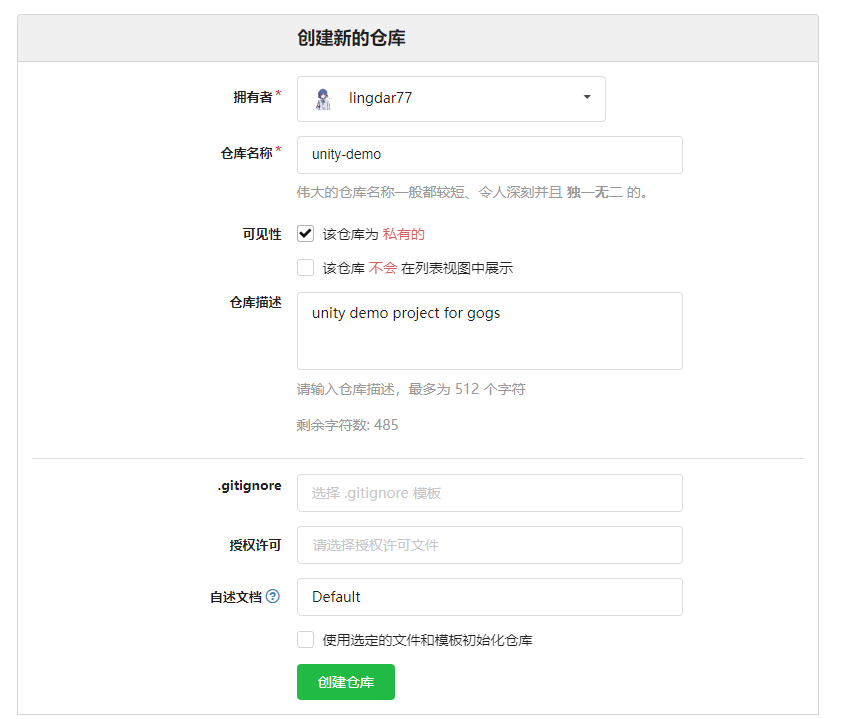


# Git工作流

## 创建一个仓库

打开gogs新建一个仓库





gogs提供了大量的语言忽略模板，但是对于unity/ue没有相关的忽略。

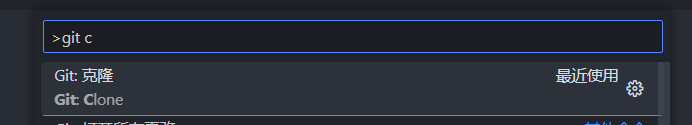


创建完毕后我们就可以将其克隆至本地。

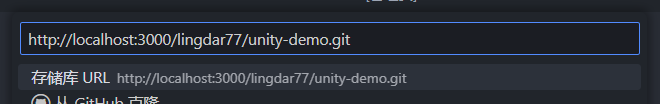
这里以vs code 为例

复制仓库链接，打开code

按F1 输入git clone



输入链接后确认

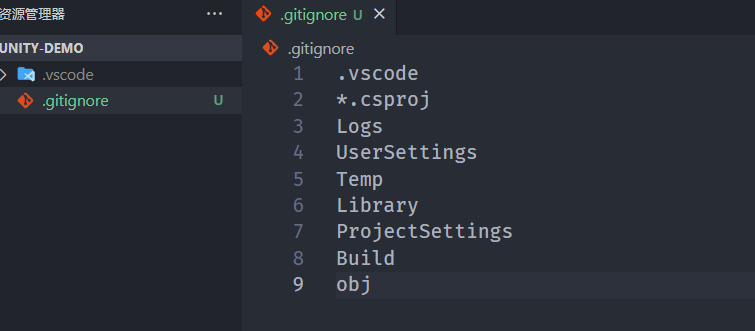


选择存储库目录后等待即可。

## 初始化仓库

对于一个新仓库，我们需要简单的配置一下。

我们首先配置好ignore文件

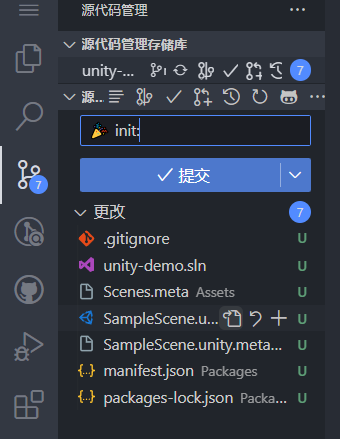


关闭code，暂且将仓库重命名一下。

然后打开unity 在这个位置新建一个项目。

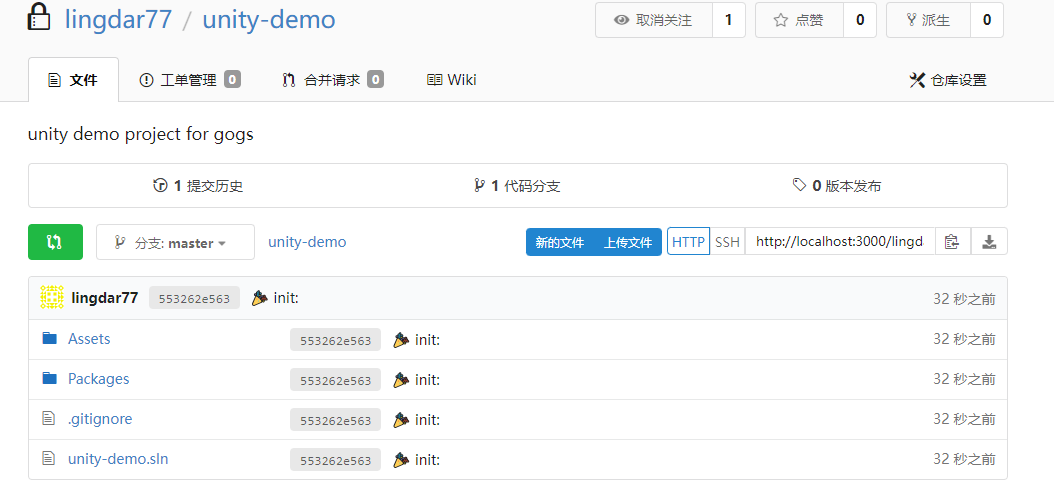
创建成功后将仓库名字改回来，并且合并两个文件夹。

重新打开code后发现已经有了更改。并且我们的忽略文件已经生效了。



写好消息后我们直接提交（你也可以写好一个README之后再提交）并且推送分支到主分支。

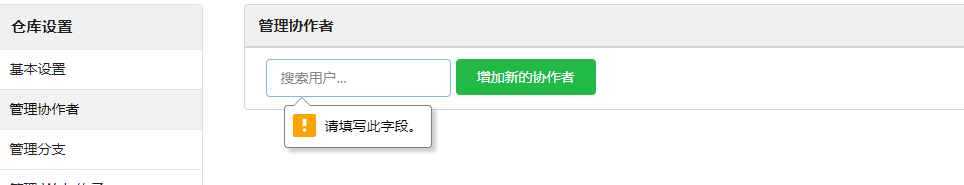
刷新一下仓库，可以看到已经同步上去了。



## 仓库设置

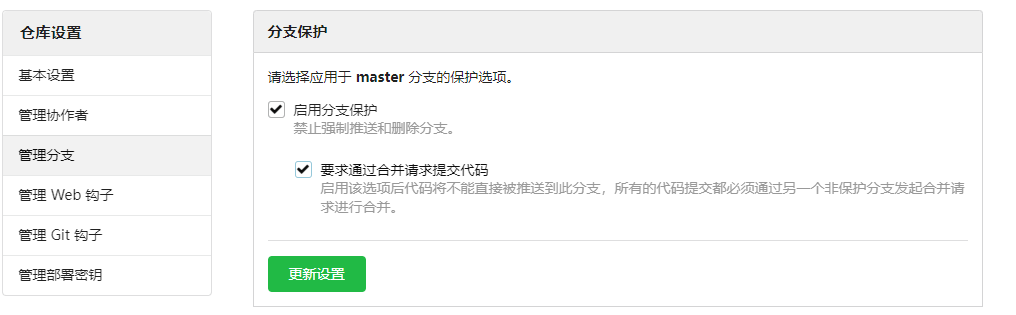
### 协作者

我们可以为仓库添加协作者，来协同工作。



### 分支管理/保护

一般的git工作流都是基于合并请求的，我们需要保护好主分支以防止恶意或者意外的修改。我们一般会启动主分支的保护。



对于复杂的项目，成员们一个各自工作在自己的分支上，互不打扰。

每个成员都会自己创建一份自己的分支，定期进行与主分支的合并、同步。管理员也可通过合并请求来进行代码审查。

对于一些特殊的开发流程。会考虑更加严谨的分支方法。

如主逻辑分支，主要负责逻辑的维护。

部分程序员在此分支上分出自己的分支，然后合并到逻辑分支。逻辑分支再定期与合并到主分支。

资产分支，主要负责资产的维护，各美术人员在此分支上工作。再合适的时间段合并到其他分支。

测试分支，一般由其他分支合并过来形成，主要进行bug的测试，代码审查。

## 适用于游戏项目的git工作流

### .gitattributes

.gitattributes 是一个文本文件，文件中的一行定义一个路径的若干个属性，主要用于定义每种文件的属性，以方便 git 帮我们统一管理。

.gitattributes 文件格式如下

要匹配的文件模式 属性1 属性2 ...

.gitattributes 文件中可以定义的属性

diff

我们知道 git 主要是用来跟踪文件版本的，跟踪文件版本自然离不开比较差异，而diff 就是用来告诉 git 声明文件需要比较版本差异的。

diff属性影响Git对特殊文件生成差异的方式。它可以告诉Git是否为路径生成文本补丁还是将路径视为二进制文件。

diff

强制视为文本文件，即使它包含一些通常从不会出现在文本文件的字节值，例如NUL。

!diff

表示为非文本文件，没有设置diff属性的路径会生成differ二进制文件（如果启用了二进制补丁，会生成二进制补丁）。

未定义

未指明diff属性的路径首先会检查其内容，如果它看起来像文本文件并且小于大文件阈值（core.bigFileThreshold），则将其视为文本文件，否则将生成differ二进制文件。

如果你不希望产生文本差异，以及行尾转换应用到任何二进制文件。可以使用系统内置的 binary，它会取消 text 和 diff 属性。

游戏项目中会产生大量的二进制文件，如何妥善的管理它们，.gitattributes可以发挥很大的作用。

### Git Large File Storage (LFS)

是一个 Git 扩展，可以将大型二进制文件存储在单独的存储库中，并使用指针替代它们。这样可以节省存储空间和带宽，并使 Git 操作更快。

我们可以使用：

git lfs track "\*.psd"

来告诉lfs跟踪所有psd文件。

这个操作会为我们生成一个.gitattributes文件。

我们可以手动配置.gitattributes来帮助我们自动化track二进制文件。

可以在 .gitattributes 文件中添加规则，告诉 Git LFS 哪些文件需要跟踪。例如：

\*.psd filter=lfs diff=lfs merge=lfs -text

这个规则告诉 Git LFS 跟踪所有 .psd 文件，并使用 Git LFS 过滤器将它们存储在 Git LFS 服务器上。

### 锁定lfs文件

不幸的是，没有解决二进制合并冲突的简便方法。 使用 Git LFS 文件锁定，你可以按扩展名或文件名锁定文件，并防止二进制文件在合并期间被覆盖。

为了利用 LFS 的文件锁定功能，你首先需要告诉 Git 哪些类型的文件是可锁定的。 在下面的示例中，在 git lfs track 命令后附加了--lockable 标志，该命令既将 PSD 文件存储在 LFS 中，又将它们标记为可锁定。

git lfs track "\*.psd" –lockable

在准备对 LFS 文件进行更改时，你将使用 lock 命令以便将文件在 Git 服务器上注册为锁定的文件。

git lfs lock images/foo.psd

Locked images/foo.psd

一旦不再需要文件锁定，你可以使用 Git LFS 的 unlock 命令将其移除。

git lfs unlock images/foo.psd

与 git push 类似，可以使用--force 标志覆盖 Git LFS 文件锁。 除非你完全确定自己在做什么，否则不要使用--force 标志。

git lfs unlock images/foo.psd --force

如果你使用vs code你可以简单的安装一个插件



你便可以快速地锁定一个文件。

更不幸的是，gogs lfs尚未支持lock

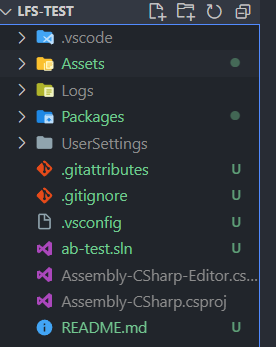
### 创建仓库

要创建一个支持lfs的仓库，我们需要在仓库配置完成后使用：

git lfs install

来为项目启用lfs服务。

这里拷贝了一个项目来作为实例：



其中已经包含了完整的ignore和attributes

使用：

git init

初始化仓库

git lfs install

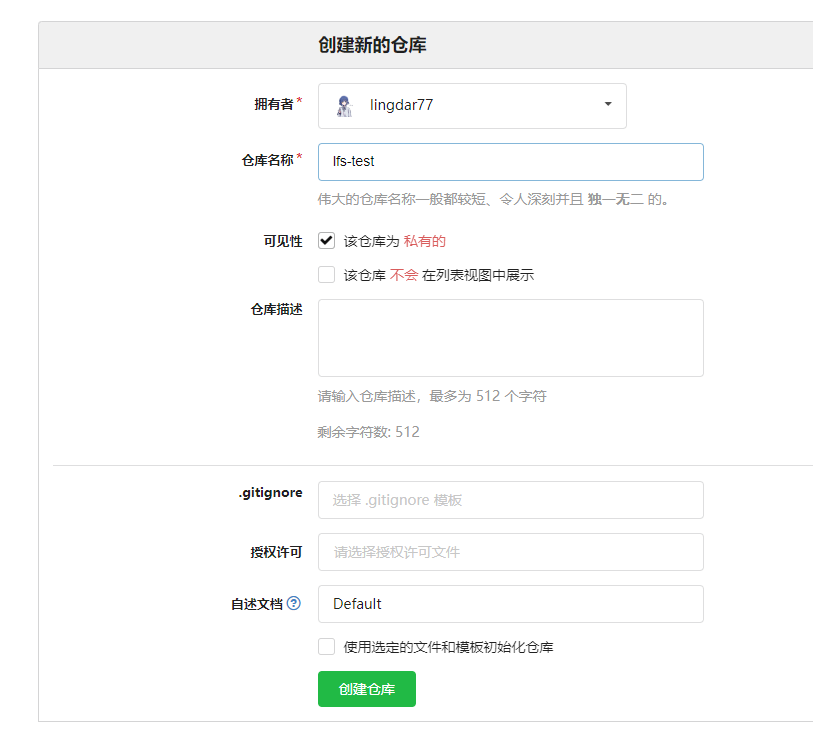
初始化lfs

git config set core.bigFileThreshold 2m （大概一个2w行左右的脚本差不多就有2m，视工程决定这个数值）

设置2m以上的文件交给lfs托管

推送到本地仓库后

我们打开gogs 创建一个仓库



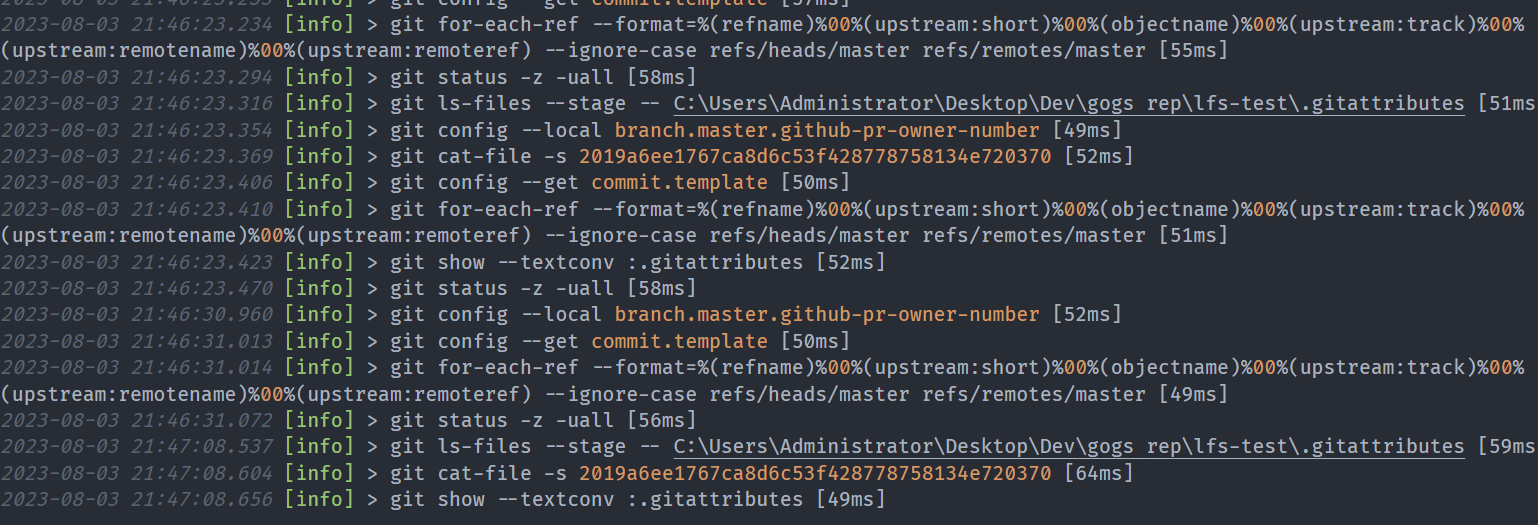
复制git链接

使用

git remote add origin <http://localhost:3000/lingdar77/lfs-test.git>

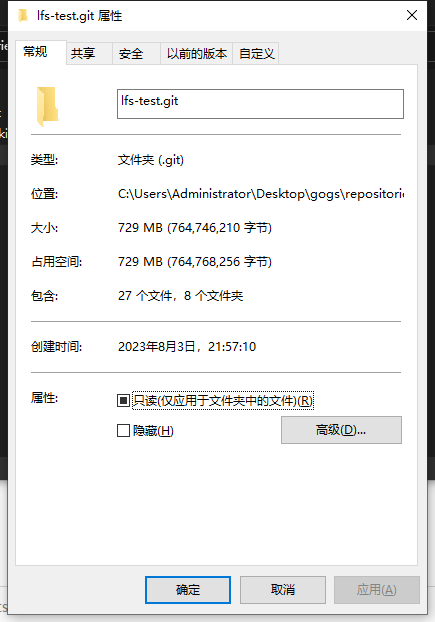
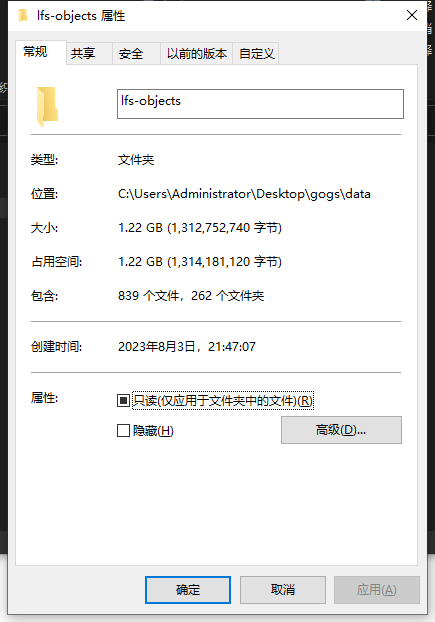
来讲我们的本地仓库链接到远端。

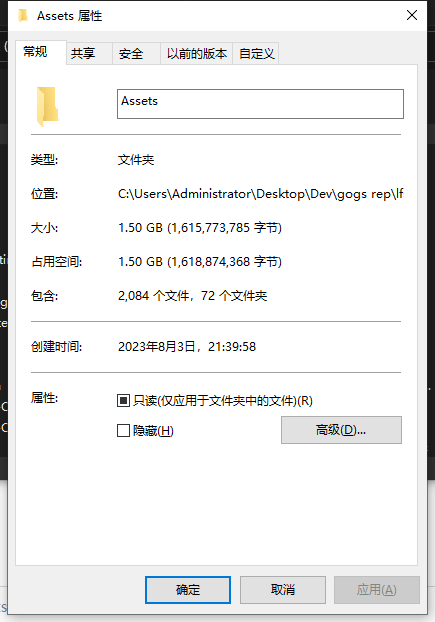
然后push到本地仓库。



可以看到lfs已经再帮我们处理了。

等待服务器上传完成，我们打开服务器文件可以看到lfs帮我存储了一些文件。





### 克隆仓库

如果你正在克隆包含大量 LFS 文件的仓库，显式使用 git lfs clone 命令可提供更好的性能。（lfs与perforce性能相当， 显示调用会跳过查询，同时拉去代码和lfs文件， 直接拉会先拉代码，再拉lfs）



普通代码的传输速度大约是20m/s(在局域网中)

就像克隆一样，你可以使用常规的 git pull 命令拉取 Git LFS 仓库。拉取完成后，所有需要的 Git LFS 文件都会作为自动检出过程的一部分而被下载。

像 git lfs clone 命令一样，git lfs pull 命令批量下载 Git LFS 文件。如果你知道自上次拉取以来已经更改了大量文件，则不妨显式使用 git lfs pull 命令来批量下载 Git LFS 内容，而禁用在检出期间自动下载 Git LFS。这可以通过在调用 git pull 命令时使用-c 选项覆盖 Git 配置来完成

### 开发分支策略

#### 主分支

开启分支保护，只允许PR。

主要由主逻辑分支与主美术分支合并得来。

自创建好项目结构后便不会频繁改变。

其他主分支都是其镜像分支。

#### 主逻辑分支

开启分支保护，只允许PR。

基于主分支开发，所有开发人员以此分支fork自己的仓库，或者建立自己的子分支（不建议，一个仓库中太多的分支存在会增加项目的复杂性，需要定时审查，删除不必要的分支）。

开发人员每完成一个功能，并且本地测试稳定后向该分支发起PR，主管审查后合并。

开发人员提交的合并申请通过之后，开发人员应该同步到主逻辑分支以待后需开发。

#### 主美术分支

### 开发流程