# 使用Gitlab

## 简介

Gitlab是一套可以self-host的基于Git的开发项目(repository)管理系统，包含了源代码管理、问题(issue)管理以及简单的用户权限(role)管理。

### 用户

用户分为两种：

1. 内部用户：可以查看内部(internal)项目以及自己加入的项目
2. 外部用户：可以查看公共(public)项目以及自己加入的项目

用户加入需要提供用户的邮箱地址，由管理员后台添加，发送邀请邮件。

#### 项目成员

用户加入项目后，也会指定不同的项目角色，关于详细的角色权限设置可以查阅[Gitlab官方文档](https://docs.gitlab.com/ee/user/permissions.html)，地址：[https://docs.gitlab.com/ee/user/permissions.html。](https://docs.gitlab.com/ee/user/permissions.html%E3%80%82)

| **操作(Action)** | **客户(Guest)** | **报告员(Reporter)** | **开发人员(Developer)** | **专家(Master)** |
| --- | --- | --- | --- | --- |
| 创建问题 | ✓ | ✓ | ✓ | ✓ |
| 查看所有问题 |  | ✓ | ✓ | ✓ |
| 创建和管理问题标签 |  | ✓ | ✓ | ✓ |
| 提交代码和合并非受保护分支代码，创建分支和合并请求 |  |  | ✓ | ✓ |
| 提交和合并受保护分支代码，创建分支和合并请求(merge request) |  |  |  | ✓ |
| 添加Wiki条目 |  |  | ✓ | ✓ |
| 创建代码片段(code snippets) |  | ✓ | ✓ | ✓ |
| 添加成员到项目 |  |  |  | ✓ |

注：项目创建者默认角色是专家。

#### 组(Group)成员

Gitlab有组的概念，每个用户都可以属于一个或多个组，在组中会有跟项目类似的成员角色设置(详情见上表)。

注：组可以对应到开发团队。

### 项目(Repository)

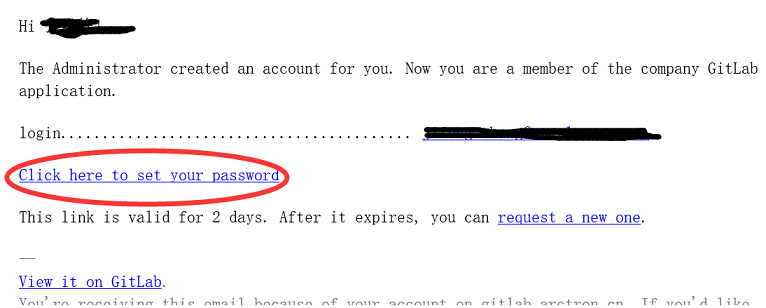
Gitlab的项目以代码为中心，计划(里程碑等)和问题追踪等使用issue来管理。

* 代码(repository)：开发人员(Developer)及以上角色
* 问题管理(issue)：所有角色
* 维基(wiki)：开发人员(Developer)及以上角色
* 代码片段(snippets)：报告人员(Reporter)及以上角色

## 项目流程

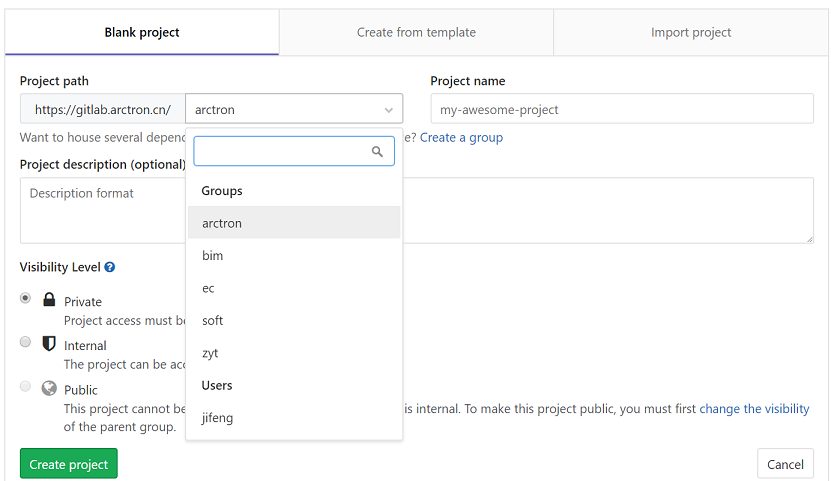
### 创建用户

1. 由管理员添加用户，用户提供常用邮箱，一般建议使用公司邮箱。
2. 管理员添加好账户以后，用户会收到一封电邮，里面包含首次登录凭证的链接。
3. 点击电邮里的链接(click here to set your password)

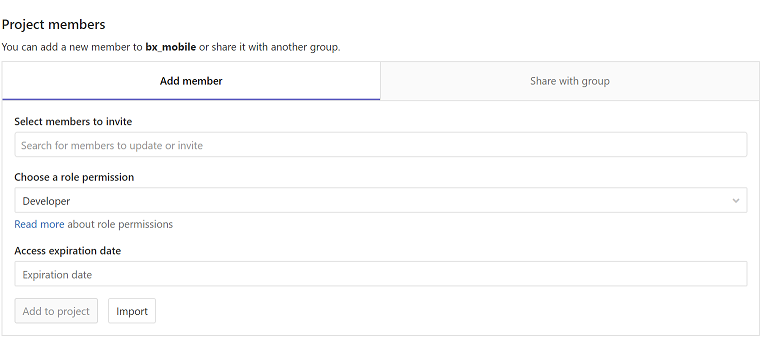


### 项目管理

1. 项目立项后，由相应的项目开发负责人在gitlab上创建项目，创建项目界面如下：



1. 创建好项目以后，开发负责人初始化代码结构，提交到默认的master分支；并根据需要创建其他分支，如dev分支(默认的master分支一般为受保护的)；若创建了开发分支，可以指定开发分支作为项目的默认分支以方便开发人员提交代码
2. 项目创建完成后，可以单击左侧的导航栏(Settings -> Members)根据前面提到的不同角色添加项目成员(开发人员为Developer，测试人员为Reporter)

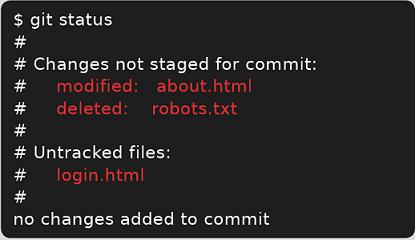


1. 项目进行过程中，开发人员提交代码，多人协作时可能会合并代码；专家(Master)角色负责合并非受保护的分支代码到受保护的主干分支；功能稳定的代码(一般是测试通过需要发布正式环境的代码)由Master负责打标签(tag)记录发布的版本
2. 项目结束后，可以由Master对项目归档(archive)，使得项目变为只读状态，不接受后续任何代码提交；文件归档可以把tag的代码统一下载下来进行归档。

## Git工作流程

### 本地工作

1. 创建项目
   * 第一次初始化项目：git init
   * 从现有项目开始：git clone <remote-url>
2. 在git中工作
   * 查看文件状态：git status



* + 添加文件：git add <filename>
  + 本地提交文件：git commit -m "message"

### 分支和合并

1. 从当前分支创建新的分支：git branch <new-branch-name>
2. 切换分支：git checkout <other-branch-name>
3. 从其他分支合并代码到当前分支：git merge <other-branch-name>

### 与远程仓库共享代码

1. 共享代码：
   * 获取远程服务器代码：git checkout --track <remote/branch>
   * 推送自己工作的分支到远程服务器以共享：git push -u <remote> <local-branch>
2. 与远程仓库代码同步：
   * 获取远程代码：git fetch <remote>
   * 合并获取的远程代码到当前(HEAD)分支：git pull
3. 推送本地代码到远程服务器以共享：git push