



Git使用教程

PI-Lab



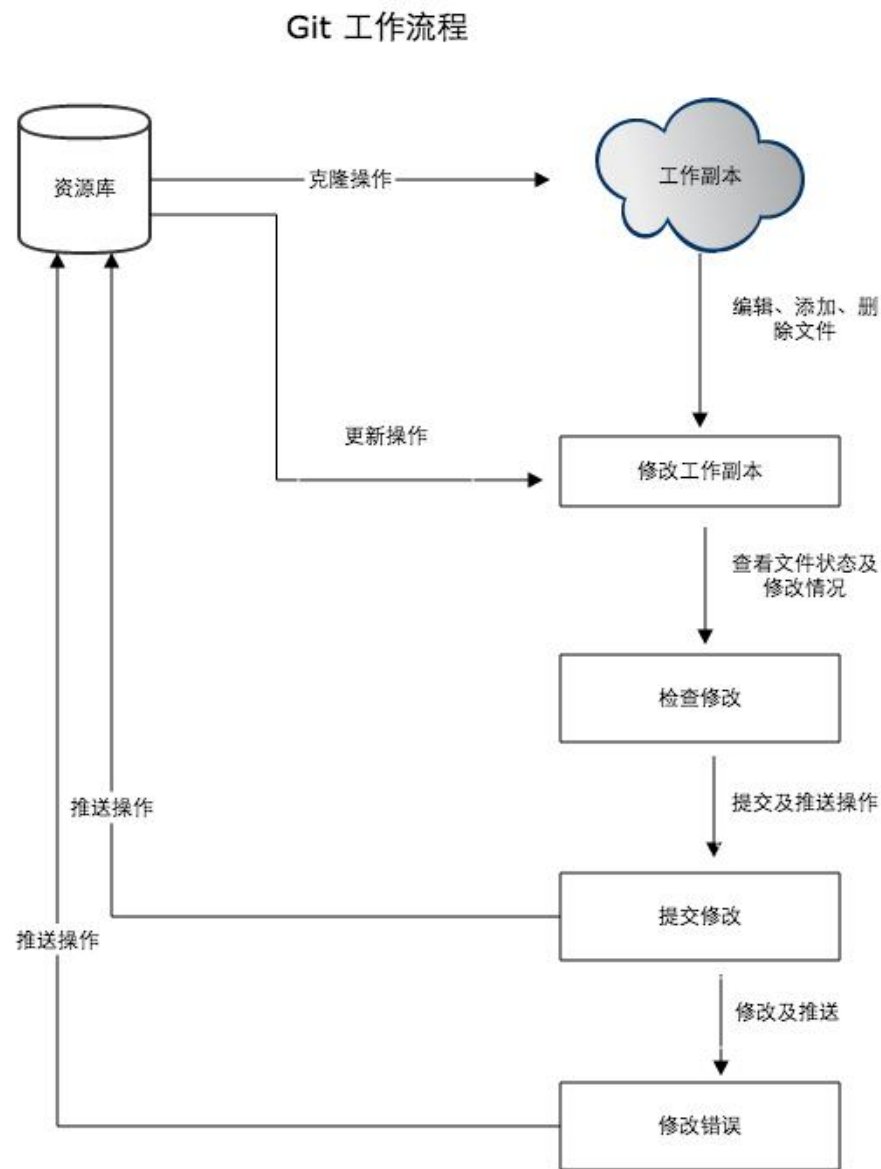
Git简介

- Git是一个分布式的版本控制工具，它出现减轻了许多开发者和开源项目对于管理分支代码的压力，对分支的良好控制：
 - **Git是分布式的，SVN不是：**这是Git和其它非分布式的版本控制系统，例如SVN，CVS等，最核心的区别。
 - **Git把内容按元数据方式存储，而SVN是按文件：**所有的资源控制系统都是把文件的元信息隐藏在一个类似.svn,.cvs等的文件夹里。
 - **Git分支和SVN的分支不同：**分支在SVN中一点也不特别，就是版本库中的另外的一个目录。
 - **Git没有一个全局的版本号，而SVN有：**目前为止这是跟SVN相比Git缺少的最大的一个特征。
 - **Git的内容完整性要优于SVN：**Git的内容存储使用的是SHA-1哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。

Git工作流程

一般工作流程如下：

- 克隆 Git 资源作为工作目录。
- 在克隆的资源上添加或修改文件。
- 如果其他人修改了，你可以更新资源。
- 在提交前查看修改。
- 提交修改。
- 在修改完成后，如果发现错误，可以撤回提交并再次修改并提交。



Git常用命令汇总

命令	功能
git init	初始化代码仓库
git clone <url>	复制仓库最常见的方式是使用 git clone，后跟仓库的 URL
git remote -v git remote add origin <url> git remote set-url origin <url>	显示Git服务器的详细信息 增加一个Git服务器 更改Git服务器的地址
git add --all git commit -m "<message>" git push origin master	提出更改（把它们添加到暂存区） 实际提交改动 将当前的代码提交到服务器“origin”的master分支
git branch git branch <name> git checkout <name> git checkout -b <name>	显示当前的分支以及所有的分支 创建分支<name>，但并不切换 切换至已有的分支<name> 创建分支<name>，并切换至该分支
git merge <branch>	将分支<branch>合并到当前分支
git pull origin <branch>	将服务器“origin”的master分支合并至自己当前的分支
git status git diff --stat git log	显示当前工作区的状态 显示目前工作区的文件内容改动情况 显示本地仓库的历史记录

Git安装

操作系统	安装
Windows	下载地址 https://git-for-windows.github.io/
Mac	下载地址 http://git-scm.com/download/mac
Linux	<code>sudo apt-get install git</code> (for Ubuntu) <code>sudo yum install git</code> (for CentOS)

设置自己的用户名和邮箱地址

当你安装Git后首先要做的事情是设置你的用户名称和e-mail地址，这是非常重要的，因为每次Git提交都会使用该信息，它被永远的嵌入到了你的提交中：

设置用户名: `git config --global user.name "Your name"`

设置邮箱: `git config --global user.email bushuhui@foxmail.com`

```
demoProject@T470p> git config --global user.name "Shuhui Bu"
demoProject@T470p> git config --global user.email bushuhui@foxmail.com
demoProject@T470p> █
```

创建新仓库

创建新文件夹，打开命令行，然后执行 `git init`

```
git_demo@T470p> git init
Initialized empty Git repository in /home/bushuhui-data/tem/programs/git_demo/.git/
git_demo@T470p> █
```

编辑`.gitignore`文件，设定那些文件或者目录是排除的，不会加入到代码仓库的

```
demoProject@T470p> more .gitignore
data
*.bak
```

复制服务器的仓库

复制远端服务器上的仓库，执行 `git clone <url>`

```
programs@T470p> git clone git@dev.scsj.net.cn:bushuhui/demoProject.git
Cloning into 'demoProject'...
remote: Counting objects: 8, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 8 (delta 0), reused 8 (delta 0)
Receiving objects: 100% (8/8), done.
Checking connectivity... done.
programs@T470p> █
```


增加、提交更改

将修改之后的代码提交到本地仓库，可以查看当前有那些更改: `git status`

增加修改的代码: `git add -A`

提交修改: `git commit -m "Modify message"`

```
demoProject@T470p> git status
On branch dev
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.md

no changes added to commit (use "git add" and/or "git commit -a")
demoProject@T470p> git add -A
demoProject@T470p> git commit -m "Modify README.md to describe changes"
[dev 7fa647b] Modify README.md to describe changes
 1 file changed, 6 insertions(+), 2 deletions(-)
demoProject@T470p> █
```

推送更改到服务器

执行如下命令以将这些改动提交到远端仓库：

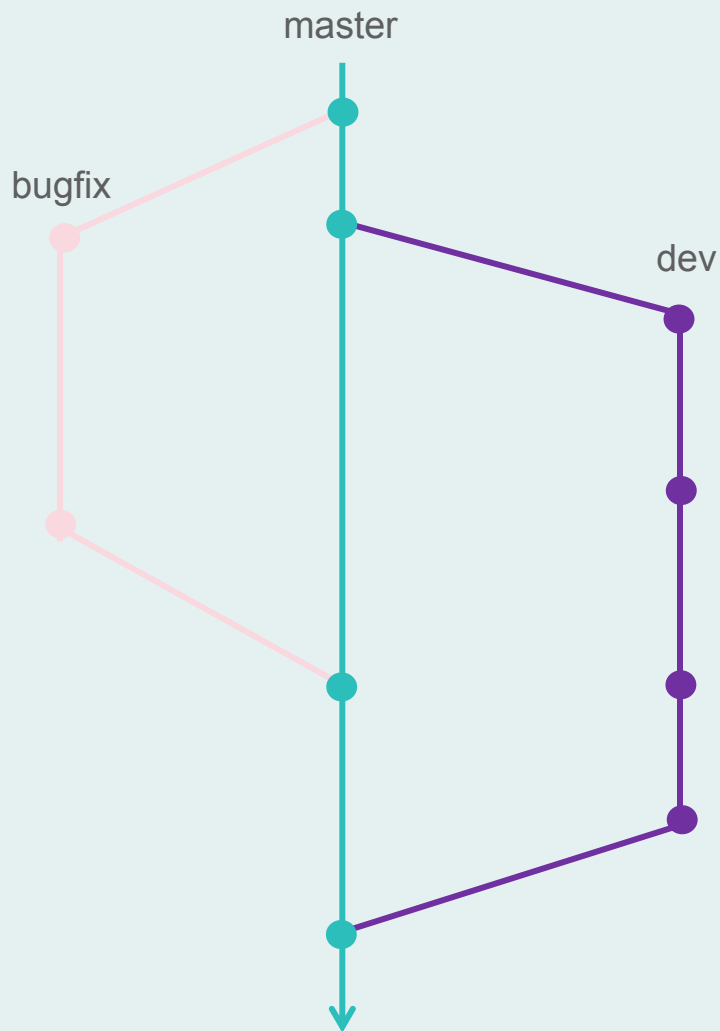
git push origin dev

可以把<dev>换成当前的分支名字

```
demoProject@T470p> git remote -v
origin  git@dev.scsj.net.cn:bushuhui/demoProject.git (fetch)
origin  git@dev.scsj.net.cn:bushuhui/demoProject.git (push)
demoProject@T470p> git push origin dev
Counting objects: 6, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 709 bytes | 0 bytes/s, done.
Total 6 (delta 0), reused 0 (delta 0)
remote:
remote: To create a merge request for dev, visit:
remote:   https://dev.scsj.net.cn/bushuhui/demoProject/merge_requests
dev
remote:
To git@dev.scsj.net.cn:bushuhui/demoProject.git
 * [new branch]      dev -> dev
demoProject@T470p> █
```

分支

分支是用来将特性开发绝缘开来的。在你创建仓库的时候，master 是“默认的”分支。在其他分支上进行开发，完成后再将它们合并到主分支上。



分支操作 - bugfix

创建 bugfix分支

`git checkout -b bugfix`

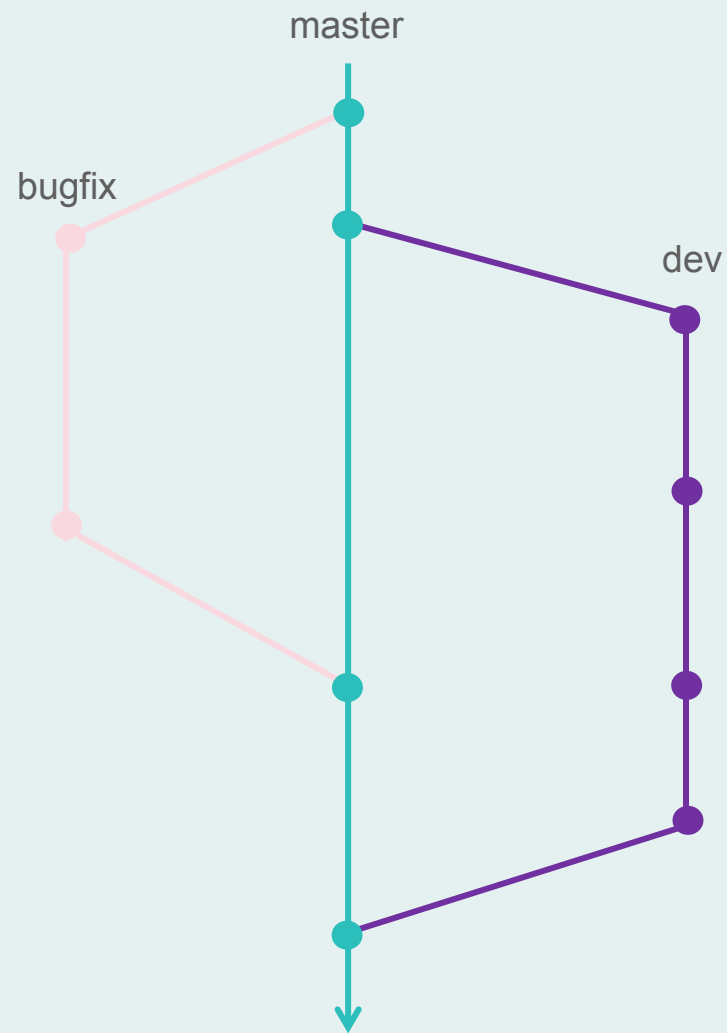
进行bug修复等

.....

合并到master分支

`git checkout master`

`git merge bugfix`



分支操作 - dev

创建 dev 分支

```
git checkout -b dev
```

```
git push origin dev
```

A用户的开发，提交代码（需要确认本地是在dev分支）

```
git pull origin dev
```

修改代码等操作...

```
git add -A
```

```
git commit -m "Modify message"
```

```
git push origin dev
```

B用户的开发，提交代码（需要确认本地是在dev分支）

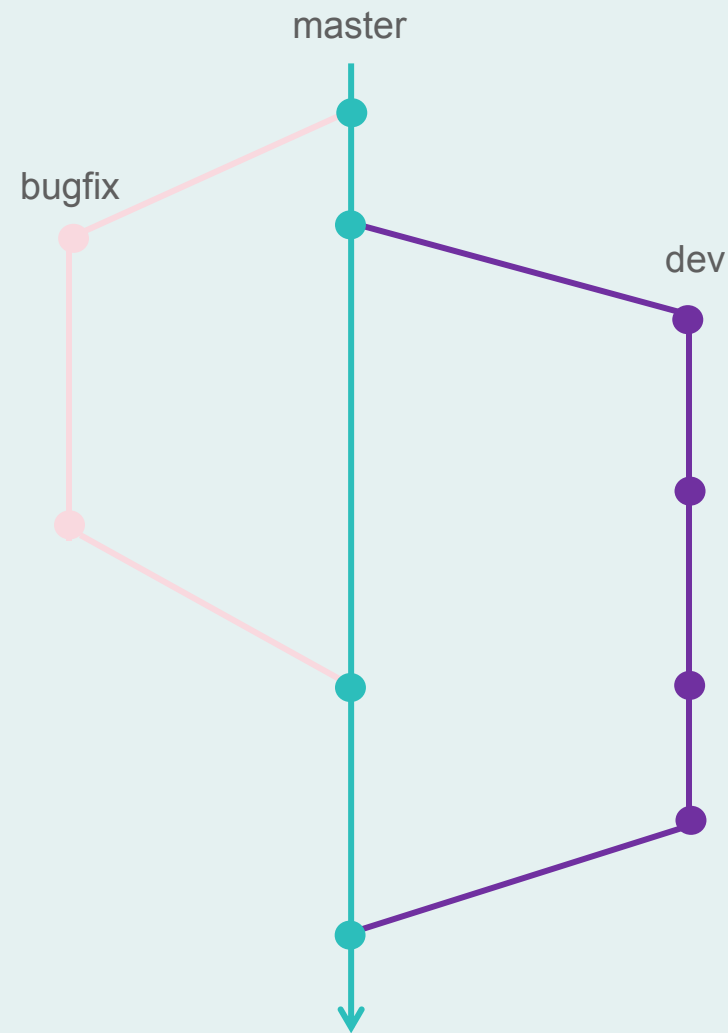
```
git pull origin dev
```

修改代码等操作...

```
git add -A
```

```
git commit -m "Modify message"
```

```
git push origin dev
```



跟新与合并 - 处理冲突

合并代码方式1

`git pull origin dev` （将服务器dev分支的代码合并到本地分支）

合并代码方式2

`git checkout master`

`git merge dev` （将本地分支dev合并到本地当前分支。例如master）

在这两种情况下，git 都会尝试去自动合并改动。遗憾的是，这可能并非每次都成功，并可能出现冲突（**conflicts**）。这时候就需要你修改这些文件来手动合并这些冲突。改完之后，你需要执行如下命令以将它们标记为合并成功：

`git add <filename> # 或者 git add -A`

在合并改动之前，你可以使用如下命令预览差异：

`git diff <source_branch> <target_branch>`

然后提交至本地并上传服务器

`git commit -m "Fix conflicts"`

`git push origin dev`

标签

为软件发布创建标签是推荐的。这个概念早已存在，在 SVN 中也有。你可以执行如下命令创建一个叫做 1.0.0 的标签：

```
git tag 1.0.0 1b2e1d63ff
```

1b2e1d63ff 是你想要标记的提交 ID 的前 10 位字符。可以使用下列命令获取提交 ID：

```
git log
```

你也可以使用少一点的提交 ID 前几位，只要它的指向具有唯一性。

Log

如果你想了解本地仓库的历史记录，最简单的命令就是使用：

`git log`

你可以添加一些参数来修改他的输出，从而得到自己想要的结果。 一个压缩后的每一条提交记录只占一行的输出：

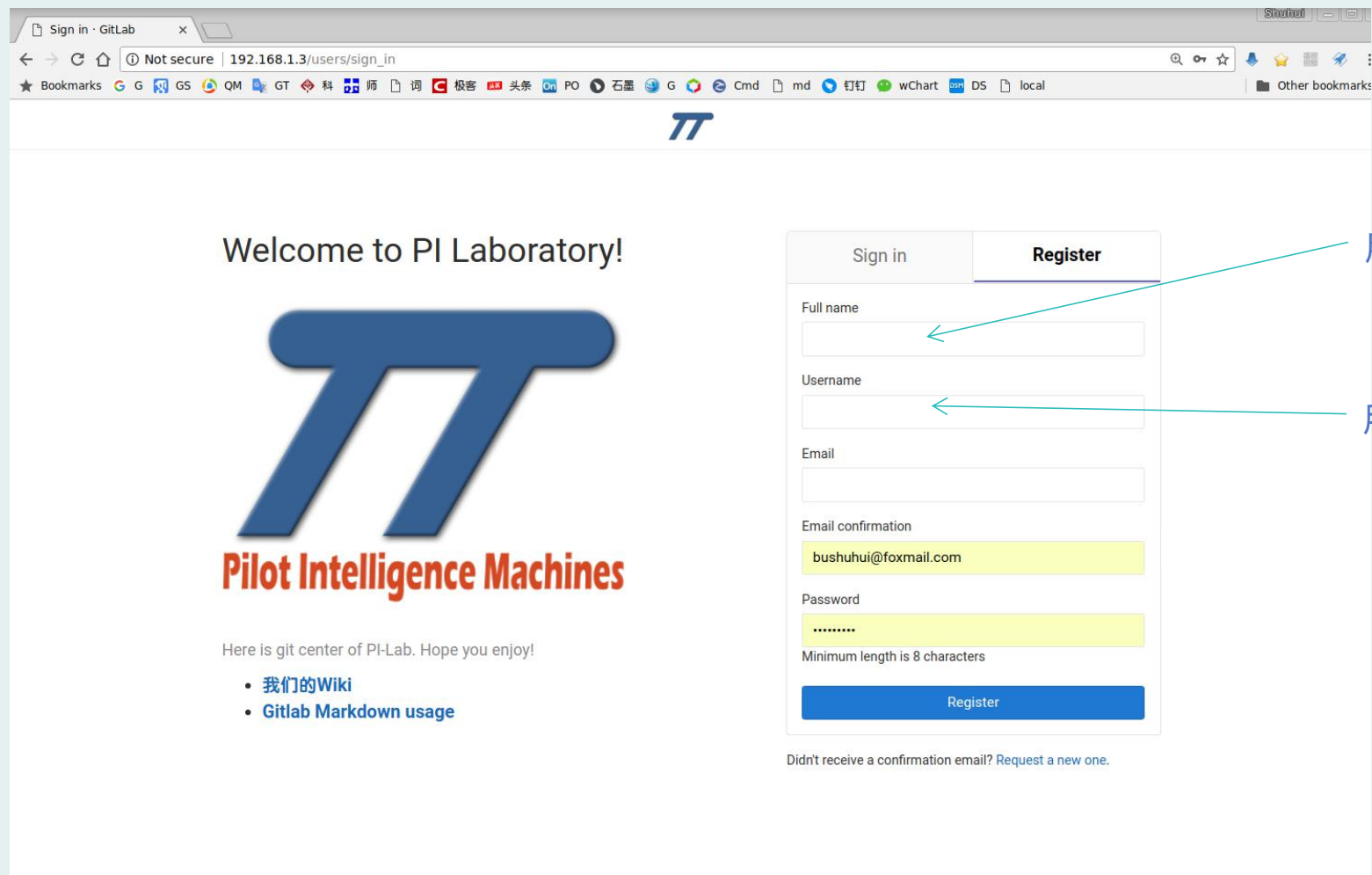
`git log --pretty=oneline`

```
demoProject@T470p> git log --pretty=oneline
7fa647b1529007940decebc1614bd0b903303dd4 Modify README.md to describe changes
effbc50f62cd370da0ad2df72bc2e419fd6f7c62 Add .gitignore
6ade351fe63564bc19c9c5b9491c7b2e524c49cc Add a line in README.md
beb57cff0f1a296fa7a43f3c966781f28d8bd585 First version
demoProject@T470p> █
```


Gitlab使用 - 注册用户

访问 <http://192.168.1.3> 点击右侧的“Register”，输入自己的姓名，用户名等信息。

Full name: 最好用中文名字，好识别每个人；Username: 最好用英文；邮箱最好写入正确的邮箱地址，方便联系



Sign in · GitLab

Not secure | 192.168.1.3/users/sign_in

Welcome to PI Laboratory!

PI

Pilot Intelligence Machines

Here is git center of PI-Lab. Hope you enjoy!

- 我们的Wiki
- Gitlab Markdown usage

Sign in Register

Full name

Username

Email

Email confirmation

bushuhui@foxmail.com

Password

Minimum length is 8 characters

Register

Didn't receive a confirmation email? [Request a new one.](#)

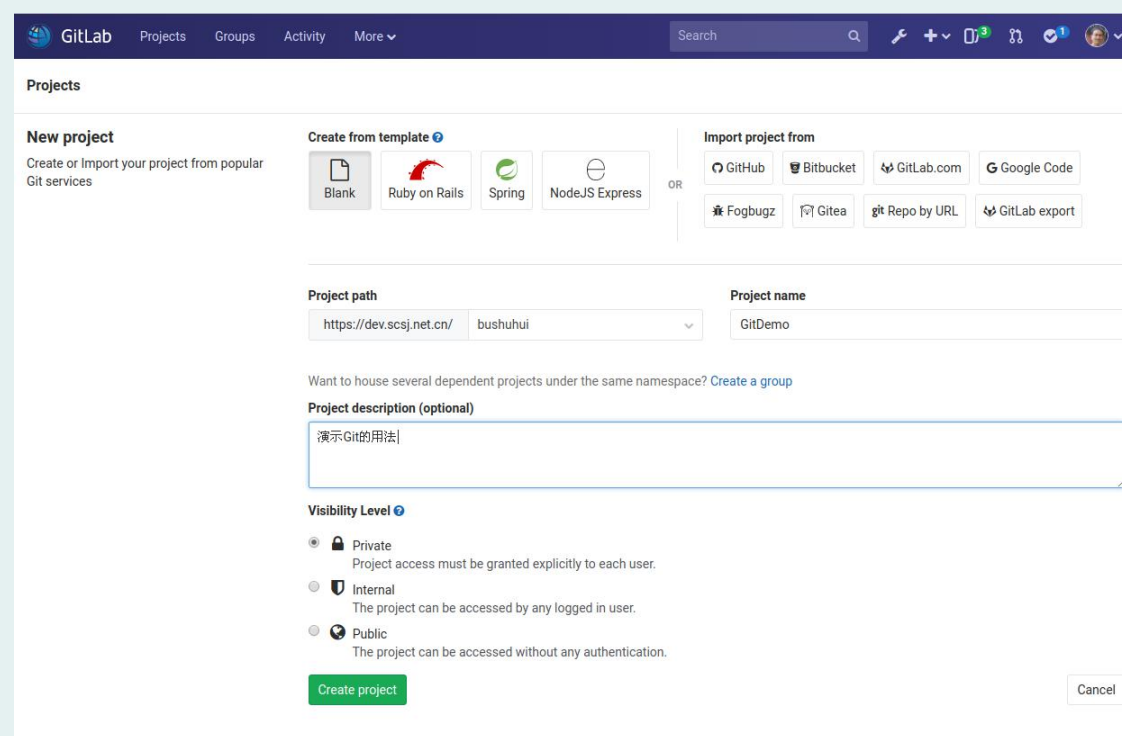
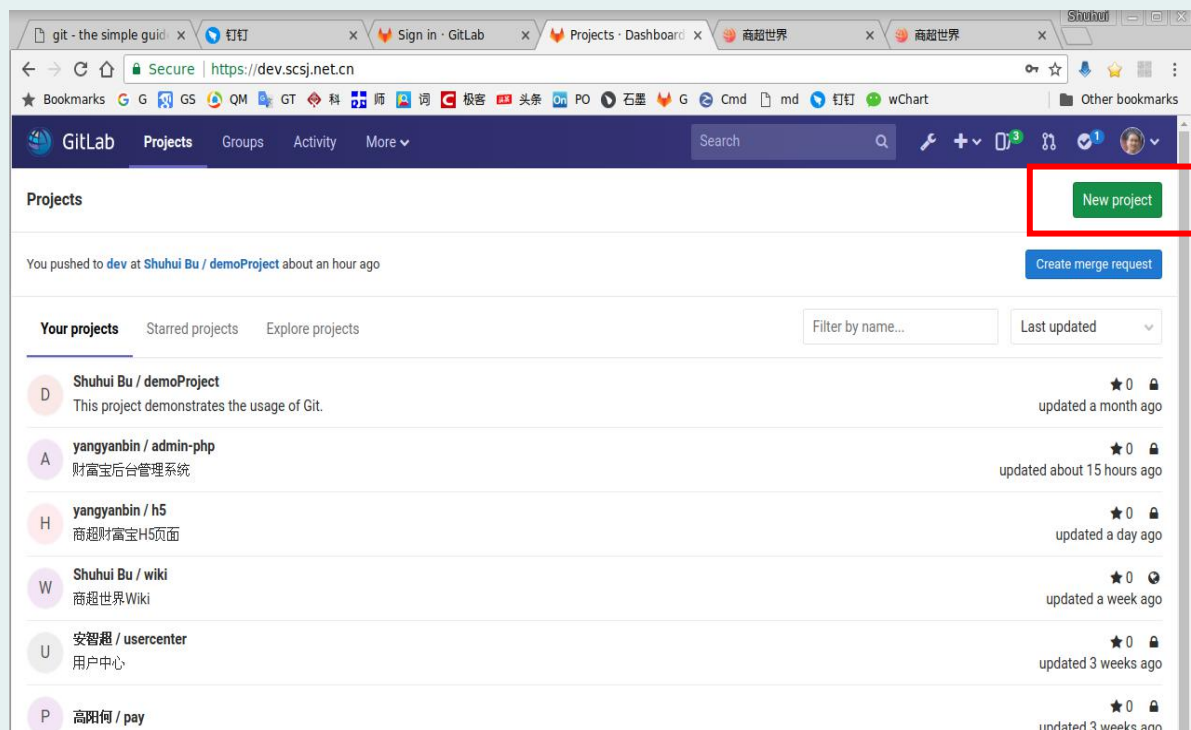
用中文更好的让其他人知道是谁

用英文，避免出现诡异的问题

Gitlab使用 - 新建项目

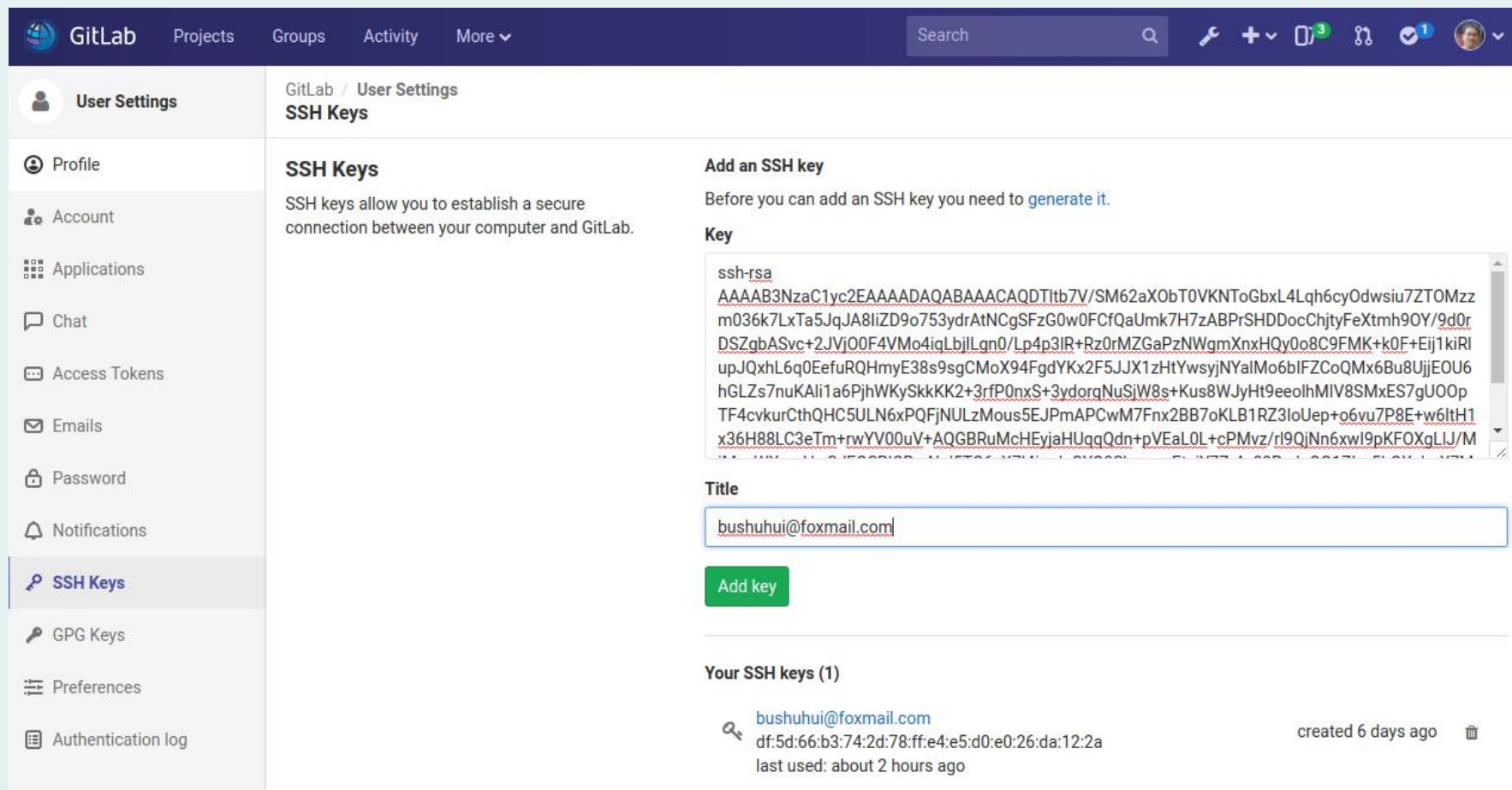
进入Gitlab系统之后，点击右侧的绿色按钮“New project”新建工程。

需要选择并填写：模板（默认Blank），Project name，Project description，Visibility Level（选择Internal或者Private）



Gitlab使用 - 设置SSH

点击右侧上部的用户图标里面的“**Settings**”，进入“**SSH Keys**”，设置自己的SSH密钥。通过SSH的方式，不用每次输入密码，更方便使用。



如何生产SSH Key，可以参考：<http://192.168.1.3/help/ssh/README>

Gitlab使用 - 设置SSH (生成key)

在自己的电脑，打开命令行终端，执行：

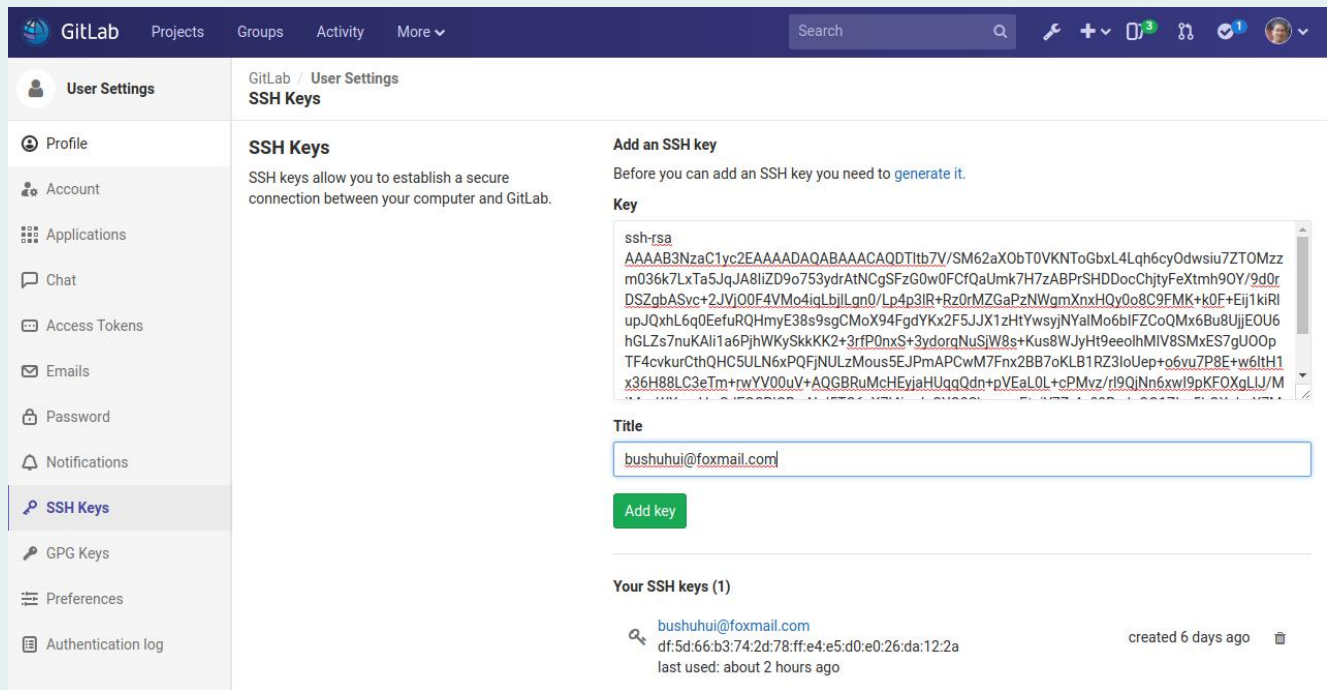
```
ssh-keygen -t rsa -C "your.email@example.com" -b 4096
```

将文件“~/.ssh/id_rsa.pub”的内容粘贴到个人设置中的“SSH Keys”当中的Key文本框

```

~@T470p> ssh-keygen -t rsa -C "bushuhui@T470"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/bushuhui/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/bushuhui/.ssh/id_rsa.
Your public key has been saved in /home/bushuhui/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:u0G7n0qc0gx6PdkUi2e+s4je14/07Xyt0b0EkIFP+nw bushuhui@T470
The key's randomart image is:
+---[RSA 2048]-----+
|
|  .
|  ..
|   .o
|    +o
|   . + . .
|  . . * S o .
| . * X .   o E..
| . o @ +.   o....
| . *.oo+.+ ..o .
|  .o **=++o=... .
|
+---[SHA256]-----+

```

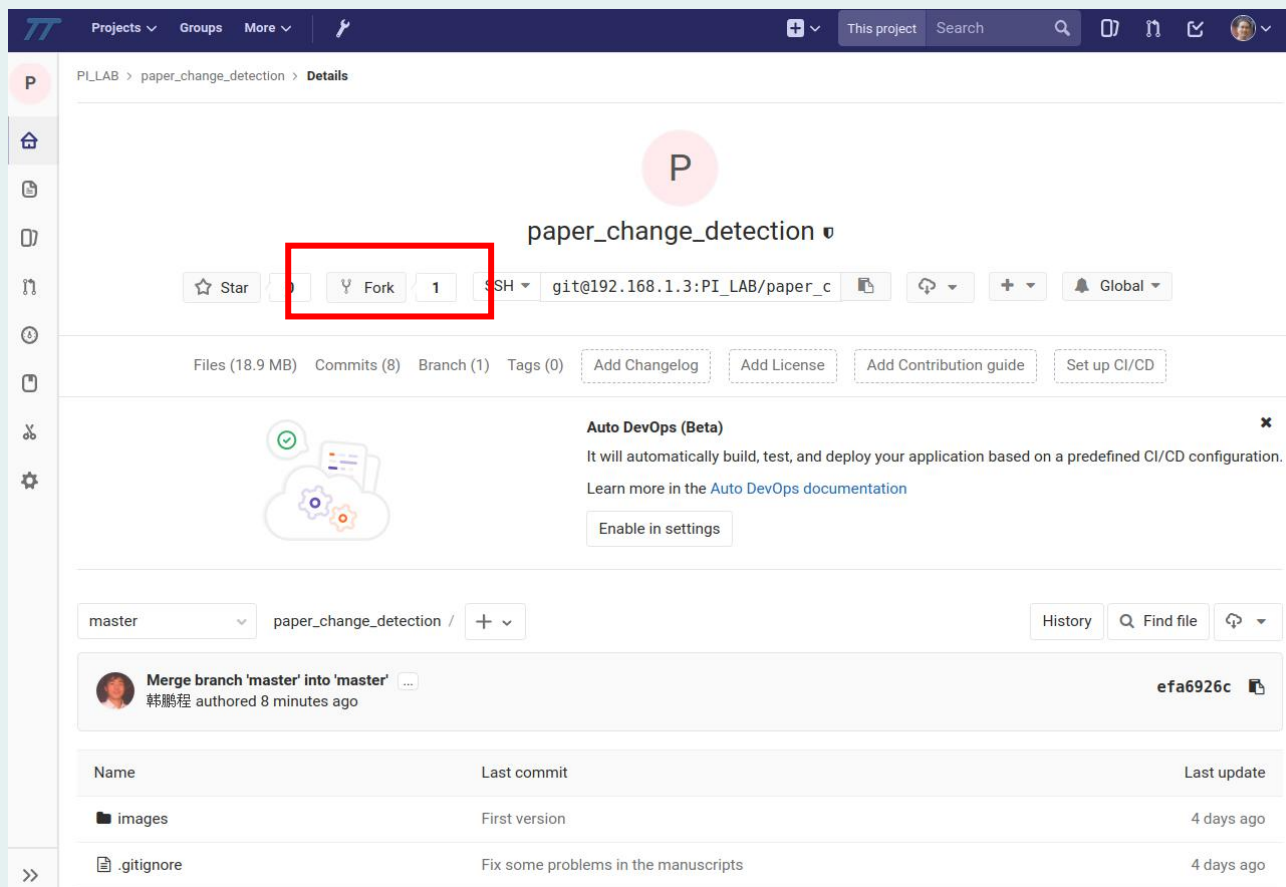


如何生产SSH Key，可以参考：<http://192.168.1.3/help/ssh/README>

项目协同开发 - （1）如何Fork项目

为了提高项目开发的质量，主项目的分支不能轻易修改，另外每个人提交的修改，在能力比较强的学长审查之后合并到主项目分支；如果代码写的有问题给出修改意见后，在自己的项目中进行修改，然后再次提交。这样的过程能够保证整个项目的代码质量，并且在整个过程中，大家能够逐步提高编程能力。

进入某个项目的主页，点击其中的“Fork”复制一份到自己的工程

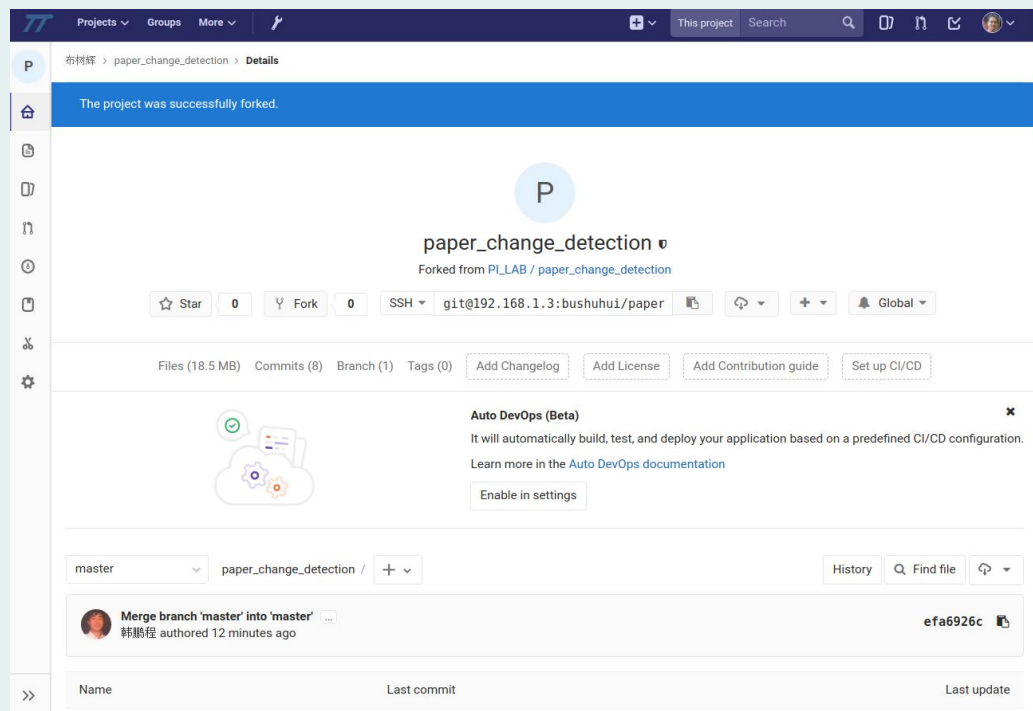


项目协同开发 - （2）如何编辑、提交修改

Fork成功之后自动进入自己的项目页面，

1. 然后clone到本地： `git clone git@192.168.1.3:bushuhui/paper_change_detection.git`
2. 并对其中的文件进行修改
3. 然后提交到自己的工程

自己的项目



提交修改到本地

```
change_detection@T470p> git add -A; git commit -m "Improve abstract"
On branch master
nothing to commit, working directory clean
```

提交修改到gitlab自己的项目

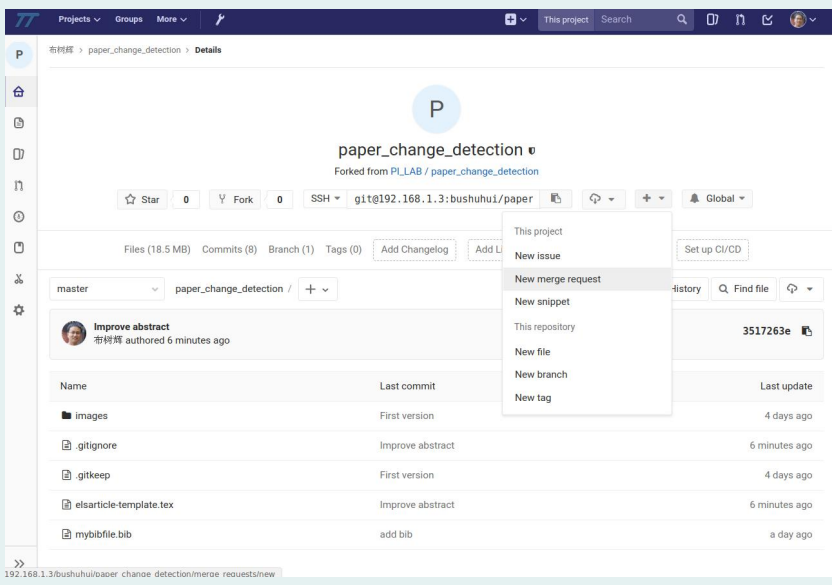
```
change_detection@T470p> git push bushuhui master
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 667 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To git@192.168.1.3:bushuhui/paper_change_detection.git
 efa6926..3517263  master -> master
```

项目协同开发 - （3）如何Merge Request

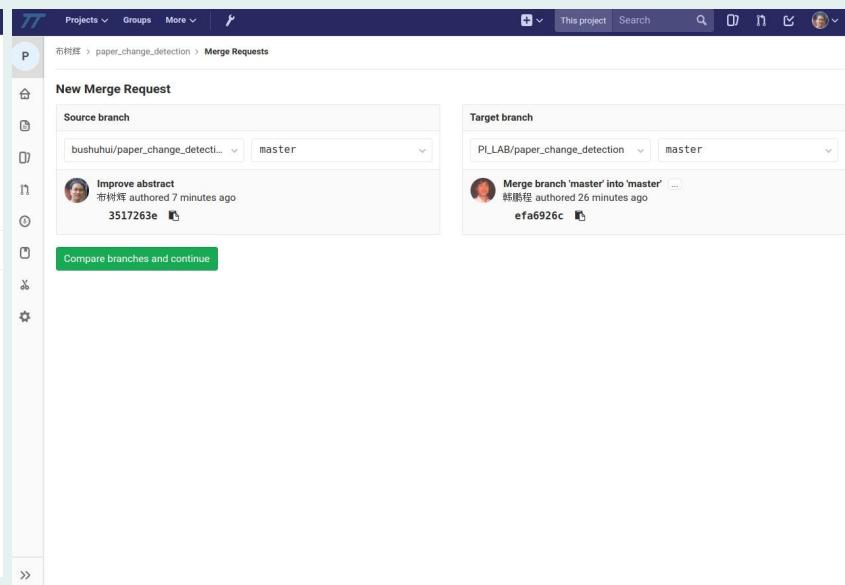
进入自己的项目页面

1. 点击其中加号的图标，选择“New Merge Request”
2. 选择自己项目的分支，目标项目和分支

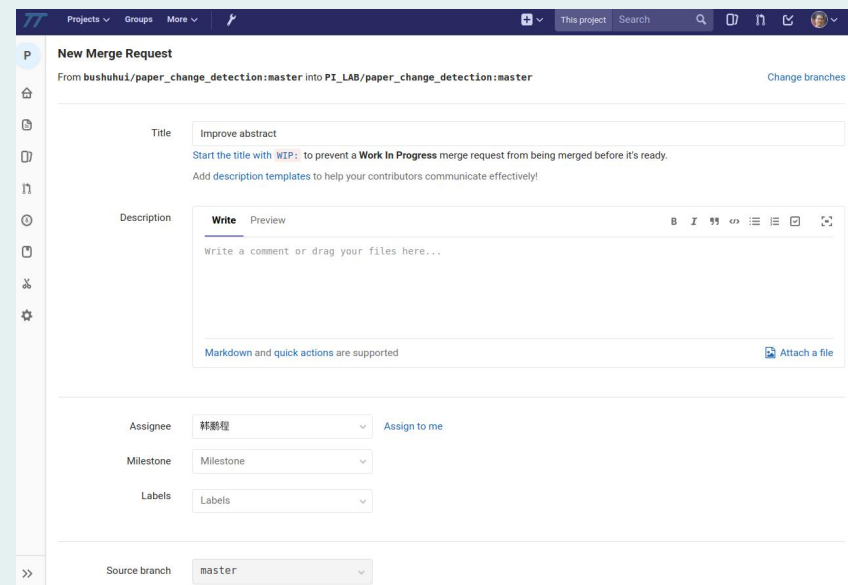
点击加号，选择“New Merge Request”



选择自己项目的分支，目标项目和分支



写出修改的内容，并选择给谁审阅

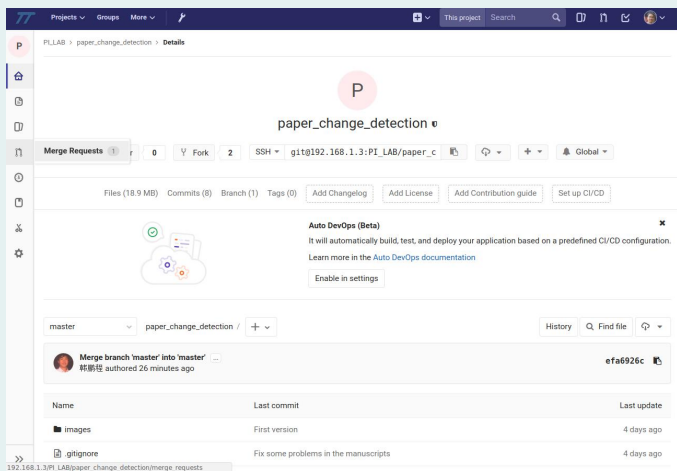


项目协同开发 - （4）审阅与合并

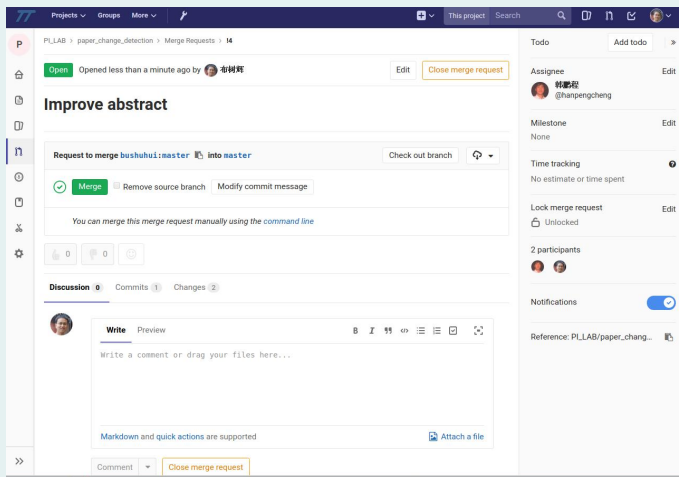
进入主项目页面

1. 点击选择左侧第4个图标，选择“Merge Request”
2. 查看具体的修改，并审查
3. 如果没有问题则点击“Merge”

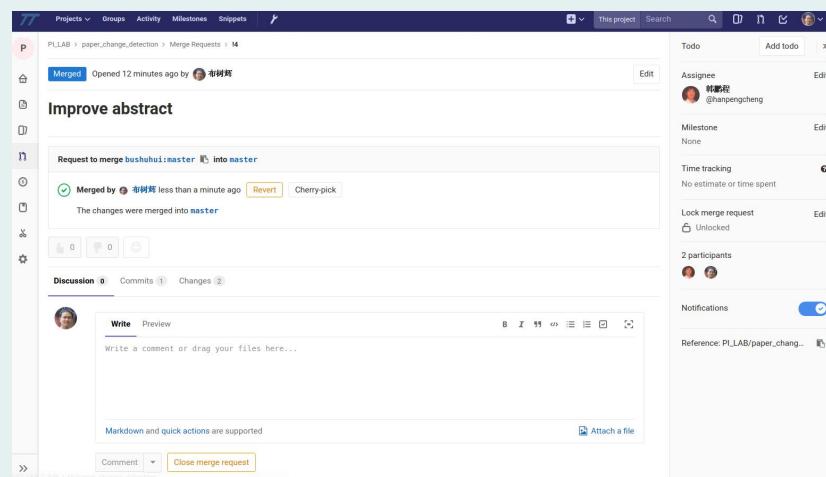
点击选择”Merge Request“



查看具体的修改，并审查



如果可行，则点击Merge



Git参考资料

一周工作所用的日常 Git 命令

<https://www.toutiao.com/a6462980515795304974/>

git - 简明指南

<http://rogerdudler.github.io/git-guide/index.zh.html>

廖雪峰的Git教程

<https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000>

提问 答疑