git风格指南

目的

本指南旨在给即将开始规范开发之旅的玩家一些规范 git 协作开发流程的建议，并作为日后开发的参考文档使用。 除了本指南，大家还可以依次阅读 廖雪峰的 git 教程 和 Pro Git v2。 廖的教程方便一小时入门，pro git 讲得足够深入，是 git 官方的推荐教程。

Git安装和配置

安装

Windows环境

首先找到git官网下载页面。

找到Windows版本的git下载地址，进行下载，直接使用msi进行安装即可。

之后打开cmd命令行窗口，输入命令 `git --version`，出现以下信息（具体版本不一定相同）时，即为安装成功。

```
git version 2.12.2.windows.1
```

值得注意的是：

- 在国内网络下直接下载Windows的地址可能速度比较慢。**建议有条件的同学使用代理，或者使用迅雷会员账号进行加速下载**。
- git-bash：windows用户由于其极其捉急的cmd，所以我们建议使用git bash来进行命令行操作(powershell也ok)，git bash就是安装git时自带的的bash命令行工具，安装时勾选对应选项即可。

Linux环境

以Ubuntu16.04为例，配置方式如下：

首先更新apt镜像源

```
sudo apt-get update
```

之后直接使用apt进行安装

```
sudo apt-get install git
```

当输入命令 `git --version`，出现以下信息（具体版本不一定相同）时，即为安装成功。

```
git version 2.7.4
```

值得注意的是：

- 此处一定要记得更新镜像源，确保安装了版本较新的git（过老的版本会在使用部分git在线平台时出现问题，**请至少确保2.0以上版本**）
- 对于其他系统，如MacOS，Centos等，操作方式也基本类似。
- 实际上如果有兴趣的话，也可以自己去git官网找源代码包进行安装。不过过程相对麻烦一些，本文中不进行详细描述，感兴趣的同学可以自行百度探索。

配置基本信息

在我们安装好git后，还需要配置一下基本信息

```
git config --global user.name "MyUserName"
git config --global user.email "my_email@buaa.edu.cn"
```

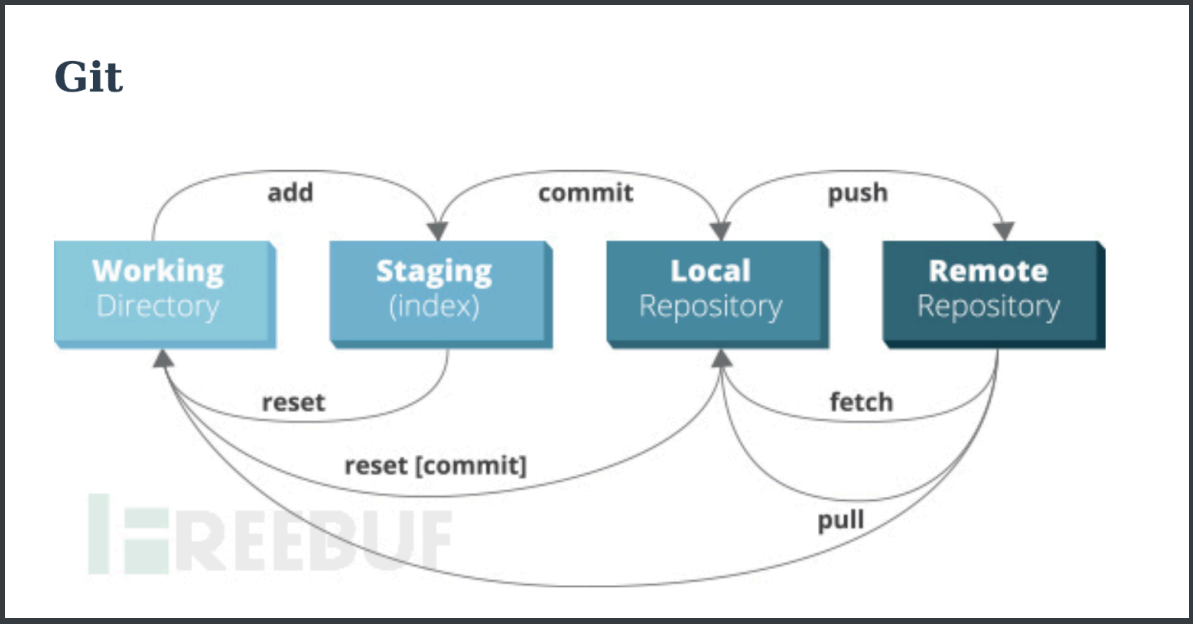
用户名替换为你的常用用户名，邮箱替换为你的常用邮箱（在本课程内，建议配置为和评测系统同样的邮箱）。

此处的配置信息也很关键，**这将作为之后你进行代码更改和提交时的人员身份凭证**。

---

git模式介绍

- 工作区，暂存区，本地仓库，远程仓库



工作流程

P.S.：以下演示操作中均在bash中进行，使用windows的玩家可以使用git-bash

P.P.S：本文中所使用的unix命令，如果不了解的玩家建议直接通过搜索引擎自学

P.P.P.S：本指南中只讲到了部分git命令的部分使用方法，主要是比较常用的

创建

1. 仓库（repository）是指项目文件仓库的意思，一般一个项目对应一个仓库，我们的每次任务可简单视作对应一个仓库。
2. 在gitlab界面中创建仓库

   详情参考gitlab基本操作中的"**创建仓库**"部分
3. 在本地某个目录创建新的仓库，并推送到远程服务器(git init, git remote)
   - **本地创建仓库(git init)**

     ```
     Swain:~/git-guide$ mkdir git-test
     Swain:~/git-guide$ cd git-test
     Swain:~/git-guide$ git init
     Initialized empty Git repository in <your own path>  # init OK
     Swain:~/git-guide$ touch README.md  # create a new readme file
     ```

   - **将创建好的本地仓库推送到远程服务器(git remote)**
4. 复制一个已经在git远程服务器上创建好的仓库(git clone)
   - 复制已创建仓库(git clone) 在gitlab或github上获取到某个仓库的HTTPS地址之后（SSH方式后文还有叙述），在bash命令行中输入下列命令即可

```
Swain:~/git-guide$ git clone https://github/Swain/git-guide.git
Swain:~/git-guide$ cd git-guide
```

5. fork机制。（多人协作）

   (TODO: Swain)

提交

现在我们来讲提交改动的三部曲---**git add\rm, git commit, git push**

1. 提交本地改动到暂存区(git status, git add)

- 假设你已经完成了一部分改动，现在需要提交本地的内容，首先使用**git status**命令来查看显示工作路径下全部已经修改的文件，然后使用**git add <file>**添加需要提交的文件

```
Swain:~/git-guide$ git status
On branch master

No commits yet

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

    README.md

nothing added to commit but untracked files present (use "git add"
to track)
Swain:~/git-guide$ git add README.md
```

- 如果希望提交当前目录下的所有文件，可以使用**git add -A**命令，但是**一定要保证**你对当前工作路径下已修改的文件有把控，这个操作会直接提交所有已修改的文件，新手玩家可能因此陷入非常麻烦的境地。

- 那么如果我们使用**git status**发现错误添加了某个不必要的文件，那我们应该怎样删除它呢，这里我们使用**git reset <file>** 或**git rm --cache <file>**命令

```
niuyazhes-MacBook-Pro:git-guide nyz$ git status
On branch master
Your branch is up to date with 'origin/master'.

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

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

        redundant.txt

    no changes added to commit (use "git add" and/or "git commit -a")
    niuyazhes-MacBook-Pro:git-guide nyz$ git add -A
    niuyazhes-MacBook-Pro:git-guide nyz$ git status
    On branch master
    Your branch is up to date with 'origin/master'.

    Changes to be committed:
      (use "git reset HEAD <file>..." to unstage)

        modified:   README.md
        new file:   redundant.txt

    niuyazhes-MacBook-Pro:git-guide nyz$ git reset redundant.txt
    niuyazhes-MacBook-Pro:git-guide nyz$ git status
    On branch master
    Your branch is up to date with 'origin/master'.

    Changes to be committed:
      (use "git reset HEAD <file>..." to unstage)

        modified:   README.md

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

        redundant.txt
```

使用**git rm**命令的话就对应:

```
    niuyazhes-MacBook-Pro:git-guide nyz$ git rm --cache redundant.txt
```

其中"－cache"是删除暂存区中内容，但是保留本地具体文件的选项，如果想连本地文件一起删除，则可以使用"-f"选项，即:

```
    niuyazhes-MacBook-Pro:git-guide nyz$ git rm -f redundant.txt
```

2. 提交本地的修改(git commit)

- 在使用**git add**之后，我们需要将本地修改提交到本地仓库，这时先用**git status**查看是否添加正确，然后使用**git commit**进行提交

```
niuyazhes-MacBook-Pro:git-guide nyz$ git status
On branch master

No commits yet

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

    new file:   README.md

niuyazhes-MacBook-Pro:git-guide nyz$ git commit -m "init"
[master (root-commit) 685cede] init
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

这样我们的改动就成功提交到了本地仓库。

- 如果粗心的某玩家在提交的时候竟然又一次犯错，比如错误commit了reduandant.txt文件，那么应该如何处理呢，这时我们需要**git reset <commit id>**命令

  错误提交过程:

```
niuyazhes-MacBook-Pro:git-guide nyz$ git commit -m "second version"
[master 8eca461] second version
 2 files changed, 1 insertion(+)
 create mode 100644 redundant.txt
niuyazhes-MacBook-Pro:git-guide nyz$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

  处理方法:

  这里首先使用**git log**命令（后文有详细叙述）查看上一次提交的commit id，然后复制这个id并执行**git reset <commit id>**命令，即可回退到上一次commit的内容，这次commit提交的内容不会丢失，而是会被标记为未添加到暂存区的修改，需要重新从**git add**开始

```
niuyazhes-MacBook-Pro:git-guide nyz$ git log
commit 8eca46103c08afc122603603bf17e08f744cf070 (HEAD -> master)
```

```
Author: PaParaZz1 <niuyazhe@buaa.edu.cn>
Date:   Tue Jan 29 15:04:20 2019 +0800

    second version

commit 685cedea89ee8dd1fccc61616914645f7e42dbc4 (origin/master)
Author: PaParaZz1 <niuyazhe@buaa.edu.cn>
Date:   Tue Jan 29 14:52:41 2019 +0800

    init
niuyazhes-MacBook-Pro:git-guide nyz$ git reset
8eca46103c08afc122603603bf17e08f744cf070
niuyazhes-MacBook-Pro:git-guide nyz$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
niuyazhes-MacBook-Pro:git-guide nyz$ git reset
685cedea89ee8dd1fccc61616914645f7e42dbc4
Unstaged changes after reset:
M   README.md
niuyazhes-MacBook-Pro:git-guide nyz$ git status
On branch master
Your branch is up to date with 'origin/master'.

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

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

    redundant.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

3. 将本地修改从本地仓库提交到远程服务器仓库(git push)

之前我们将改动提交到了本地仓库，但是本地仓库只能在本地查看，如果我们希望在其他地方或者让他人查看我们的仓库，就需要将其推送到远程服务器仓库（疯狂push），这时，就该**git push**出场了:

```
niuyazhes-MacBook-Pro:git-guide nyz$ git add README.md
niuyazhes-MacBook-Pro:git-guide nyz$ git commit -m "correct second
version"
[master 2ac875d] correct second version
 1 file changed, 1 insertion(+)
niuyazhes-MacBook-Pro:git-guide nyz$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 281 bytes | 281.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://gitlab.buaaoo.top/16131059/git-guide.git
   685cede..2ac875d  master -> master
```

这时，就成功将本地仓库推送到远程仓库。我们这里只讲了git push最简单粗暴的用法，关于其负责用法和push失败如何解决的问题，将在之后逐渐添加。

(TODO: Swain) **git push <remove> <branch>**

(TODO: Swain) **git push** 失败解决冲突

获取

git push是本地仓库到远程仓库，那么如果远程仓库有更新，我们如何将其同步到本地仓库，这里就需要**git pull**和**git fetch**命令了

1. git fetch

我们先说**git fetch**，这个命令的作用是将远程仓库的更新拉取到本地，但仅仅是拉取到本地，不进行和本地内容的合并，需要用户检查之后再手动合并。例如这里地球另一边的自己把他的改动提交到了远程仓库，我们这里本地需要同步，就使用**git fetch <remote> <branch>**（即远程服务器名和分支名）取回改动内容到本地，然后通过**git log -p FETCH_HEAD**查看改动，发现没有冲突问题之后使用**git merge**进行合并，就成功更新了本地仓库。

```
niuyazhes-MacBook-Pro:git-guide nyz$ git fetch origin master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://gitlab.buaaoo.top/16131059/git-guide
 * branch            master     -> FETCH_HEAD
   2ac875d..f60ca74  master     -> origin/master
niuyazhes-MacBook-Pro:git-guide nyz$ git log -p FETCH_HEAD
commit f60ca74d5471cfd543f21e4675f0979136f9e512 (origin/master)
Author: PaParaZz1 <niuyazhe@buaa.edu.cn>
```

```
Date:   Tue Jan 29 15:57:37 2019 +0800

    add third version

diff --git a/README.md b/README.md
index 85c8a38..d1cbe8c 100644
--- a/README.md
+++ b/README.md
@@ -1,2 +1,3 @@
 init version---1.58
 rm version--2.55
+pull version--3.57

commit 2ac875df5a791ec379f7d80b59c18d262d7f779c (HEAD -> master)
Author: PaParaZz1 <niuyazhe@buaa.edu.cn>
Date:   Tue Jan 29 15:44:24 2019 +0800

    correct second version

diff --git a/README.md b/README.md
index 3ea9f70..85c8a38 100644
--- a/README.md
+++ b/README.md
@@ -1 +1,2 @@
 init version---1.58
+rm version--2.55

commit 685cedea89ee8dd1fccc61616914645f7e42dbc4
Author: PaParaZz1 <niuyazhe@buaa.edu.cn>
Date:   Tue Jan 29 14:52:41 2019 +0800

    init

diff --git a/README.md b/README.md
new file mode 100644
index 0000000..3ea9f70
--- /dev/null
+++ b/README.md
@@ -0,0 +1 @@
+init version---1.58
niuyazhes-MacBook-Pro:git-guide nyz$ git merge
Updating 2ac875d..f60ca74
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
```

2. git pull

git pull就是一个组合操作，即**git pull = git fetch + git merge**，即取回改动后自动合并

分支和冲突处理

git checkout, git branch, git merge, 稍后更新，敬请期待

查看记录和版本控制

git reset, git checkout, git log, git status 稍后更新，敬请期待

git rebase

稍后更新，敬请期待

git diff，git show

稍后更新，敬请期待

(optional) 配置别名

稍后更新，敬请期待

完整工作流程演示

稍后更新，敬请期待

公共规范

gitignore

注意编写合适的 `.gitignore`，要在项目仓库的最底部目录，不要把任何与项目实现无关的文件引入git，例如 macOS 下的 `.DS_Store` 文件夹，IDE 配置文件夹 `.vscode` 、 `.idea`，运行日志文件等；

其具体的配置可以参见https://github.com/github/gitignore，选择和目前项目相关的部分配置即可

个人代码仓库的.gitignore可以根据以下原则来配置：

1. 忽略操作系统自动生成的文件，比如缩略图等；
2. 忽略编译生成的中间文件、可执行文件等，也就是如果一个文件是通过另一个文件自动生成的，那自动生成的文件就没必要放进版本库，比如Java编译产生的 `.class` 文件，比如python运行生

成的 `.pyc` 文件；
3. 忽略你自己的带有敏感信息的配置文件，比如存放口令的配置文件。
4. 一定记住如果没有强烈需求**不要将二进制文件放进版本库中**，比如c++编译链接生成的 `.so` 和 `.exe` 文件，这些二进制文件每次生成的具体内容都会不一样，将其放入版本库会导致自动merge必定失败，这在团队协作开发中是非常麻烦的事情。

换行符

在编辑 git repository 中的文件时，请特别留意**换行符**。默认情况下，Windows 系统使用 `CRLF` 换行，Unix 系统使用 `LF` 换行。在开发中尽量使用一种换行符，不要多种混用，可以在本地git配置中添加如下命令防止混用：

```
git config --global core.safecrlf true
```

Windows 下可以使用 `dos2unix` 批量处理整个文件夹内的换行。

大文件

git一般不允许提交大文件（几十MB及以上），如果必须有这样的需求的话，请配置使用git-lfs

---

gitlab基本操作

在本段教材中，我们暂时使用官方gitlab，进行操作说明。

登录与注册

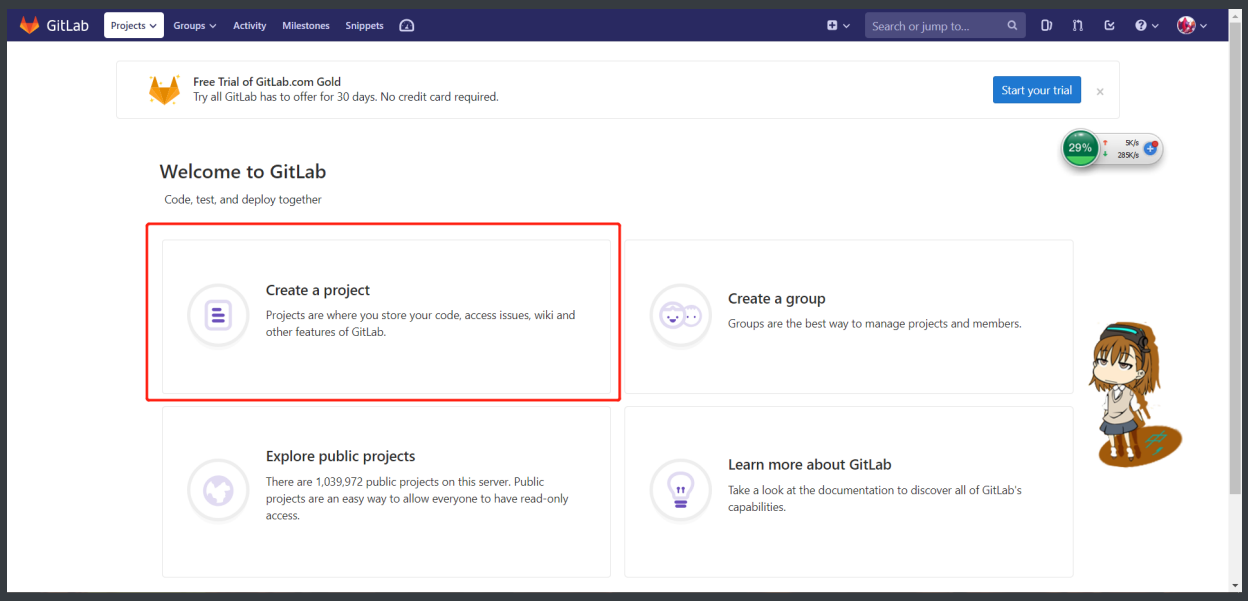
直接进入登录页面，点击 `Sign up` ，进行注册账号即可。

值得注意的是，官方gitlab部署在国外，因为一些不可描述的原因，所以**在注册的时候我们需要使用代理**，才能让Google的验证码正常显示，顺利完成注册。

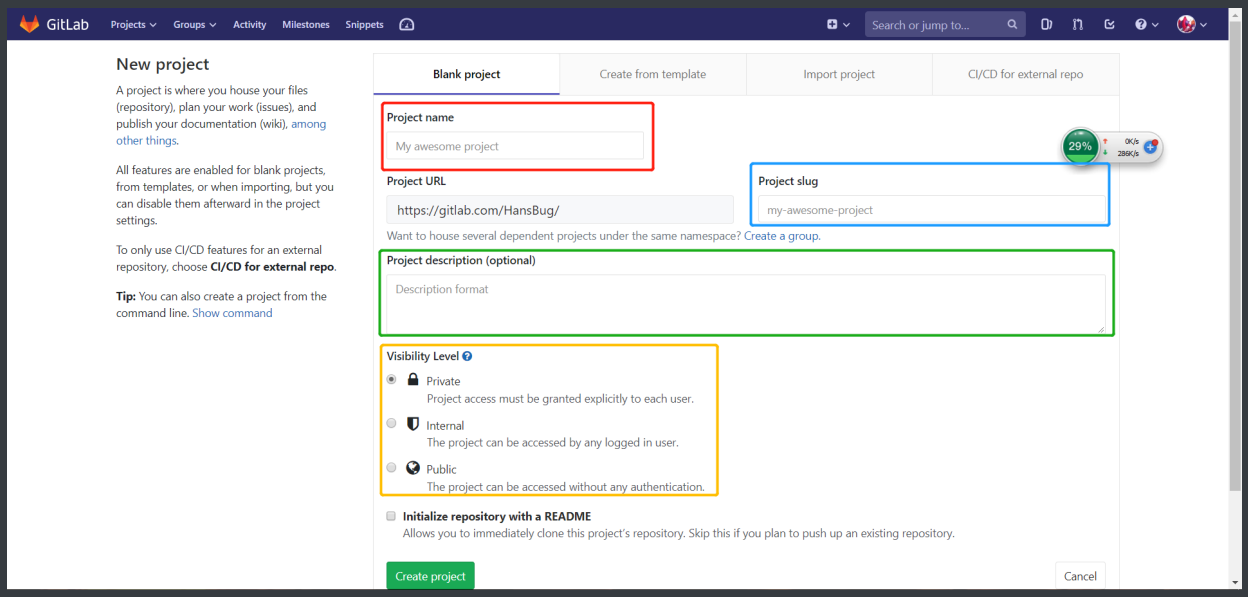
注册信息填写完毕后，去注册邮箱查收验证邮件并点击链接完成验证即可。

创建仓库

登录完毕后，进入主页面，点击 `Create project`



点击后进入创建项目页面

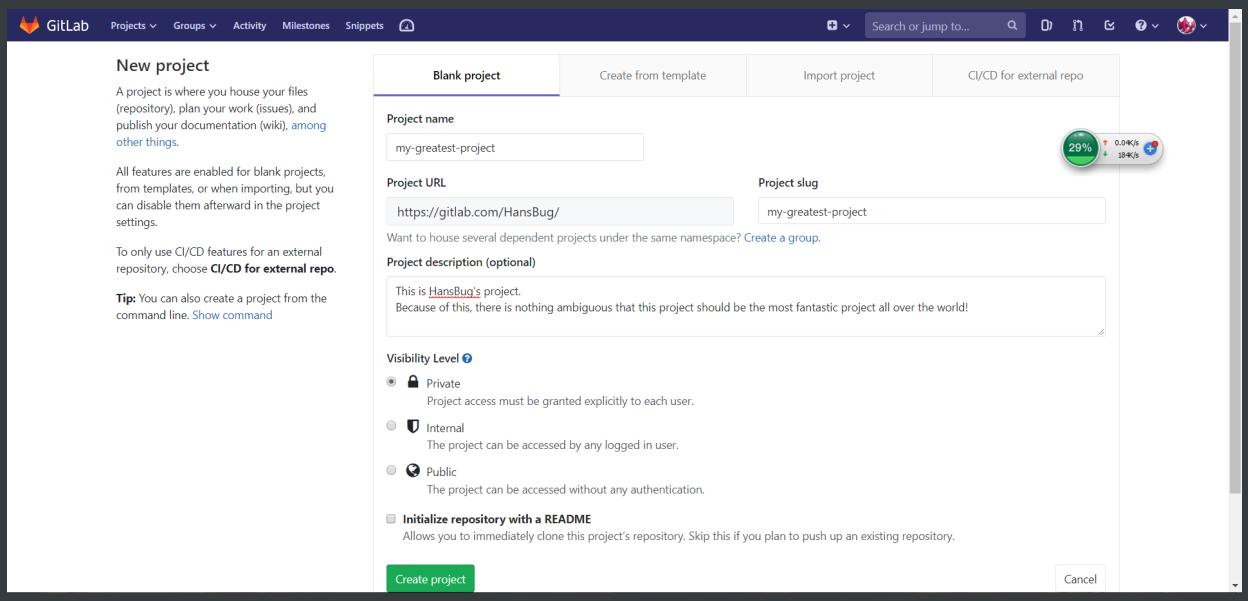


我们**主要需要填写的部分是红色框的部分，即项目名**。填写后蓝色框部分会自动与之同步，我们一般不去管它即可。**一般情况下，建议项目名采用减号分隔的的小写字符串（部分情况下使用下划线也可）**。

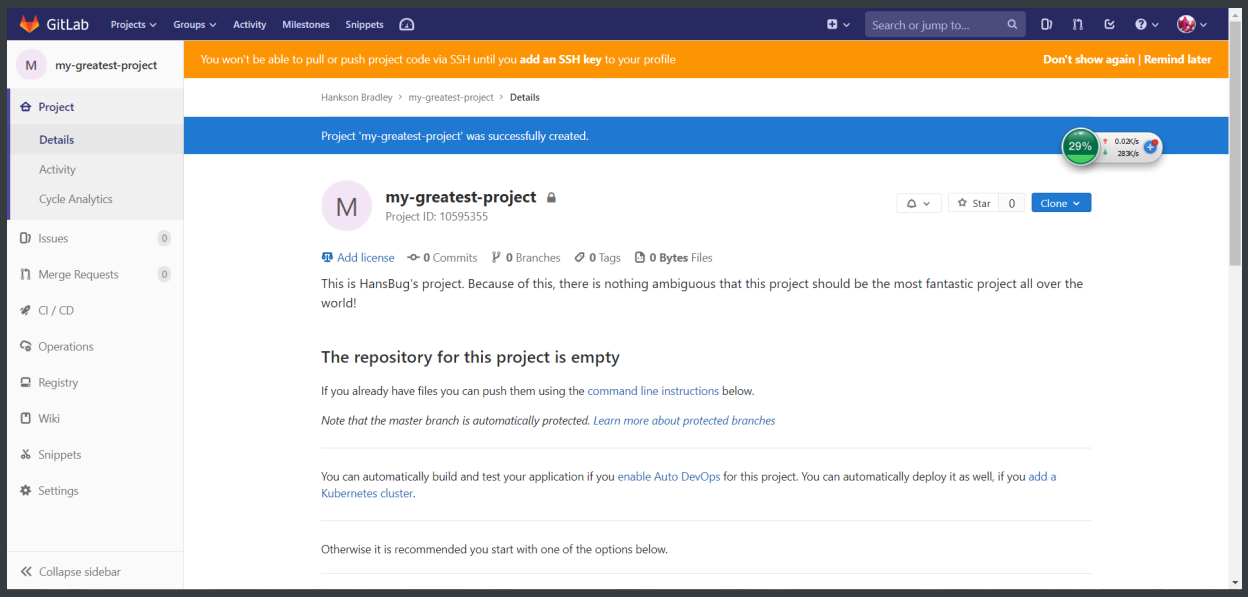
绿色框部分是项目的简述，可以在这里简单写一下你对于项目的描述。

橙色框是项目的可见性选择，**一般情况下我们选择private即可**。对于想要对所有用户公开的项目，可以选择public。

就像这样



之后我们点击 `Create project` 。



现在可以看到，一个在线项目创建完毕。

上传我的代码

首先，我们在本地的代码仓库添加远程地址

```
git remote add gitlab git@gitlab.com:HansBug/my-greatest-project.git
```

上面的命令中 `gitlab` 可以自行制定远程地址代号。后面的地址也应该换成自己的地址。

对于已经配置好ssh的玩家，可以使用上面一样的ssh地址，全程免密码操作。否则只能使用https地址进行在线操作，并且需要在push和pull的时候输入用户名、密码。

之后，我们将本地分支进行push

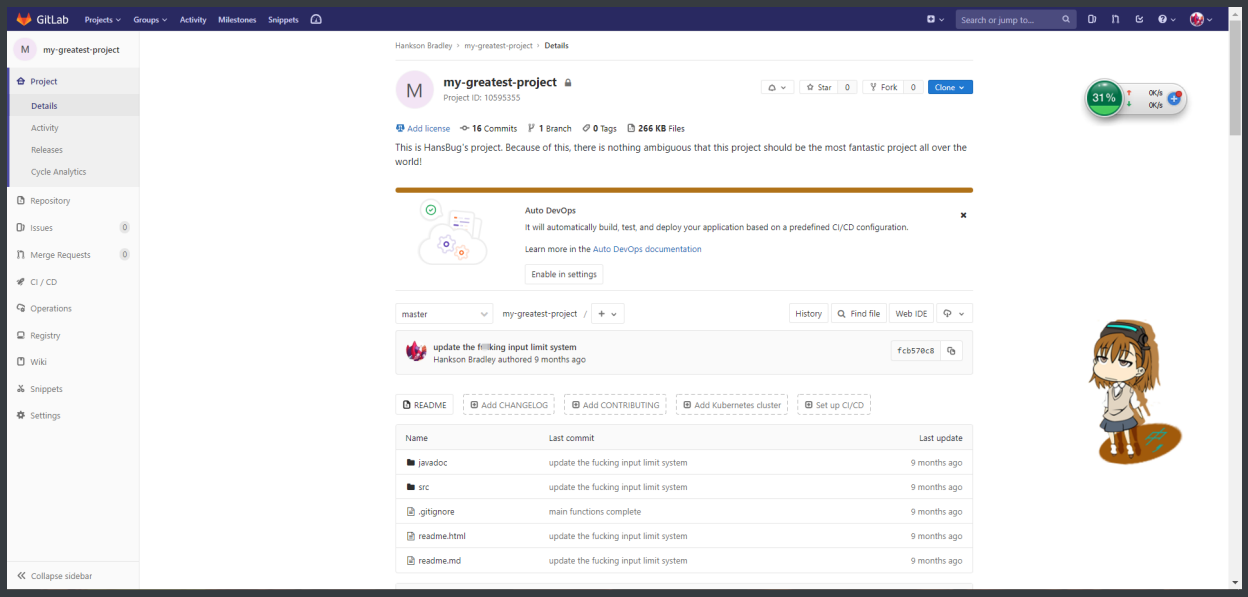
```
git push gitlab master:master
```

将本地的 `master` 分支推送至远程仓库的 `master` 分支。

运行过程大致如下：

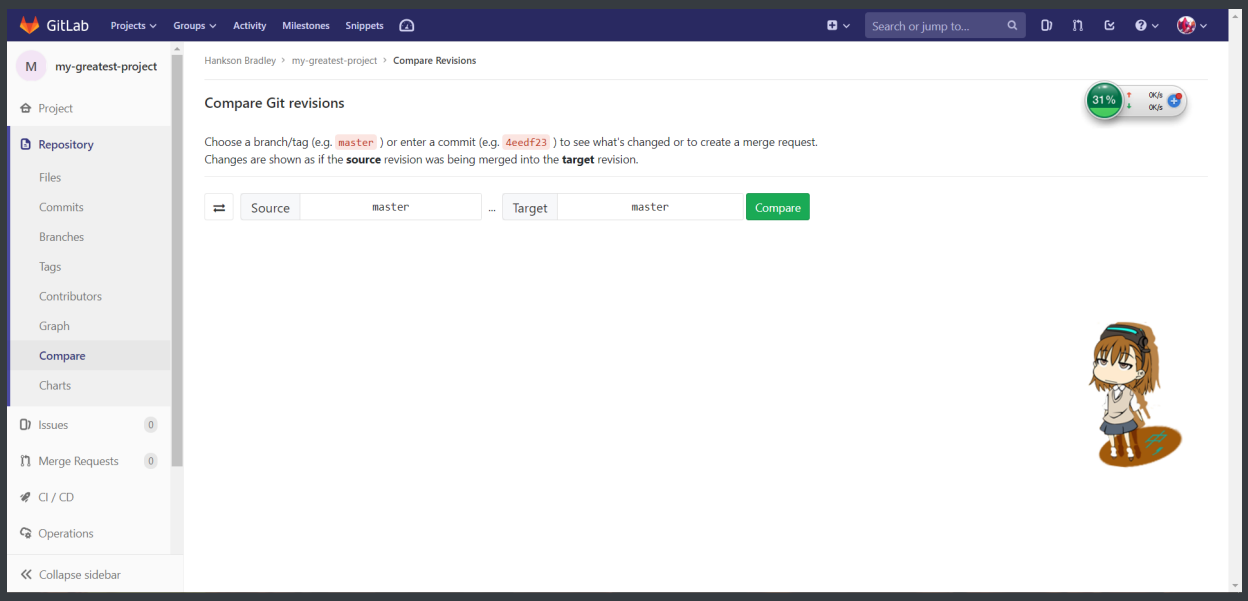
```
The authenticity of host 'gitlab.com (35.231.145.151)' can't be
established.
ECDSA key fingerprint is
SHA256:HbW3g8zUjNSksFbqTiUWPWg2Bq1x8xdGUrliXFzSnUw.
Are you sure you want to continue connecting (yes/no)? yes  # 这个地方写
yes
Warning: Permanently added 'gitlab.com,35.231.145.151' (ECDSA) to the
list of known hosts.
Counting objects: 630, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (598/598), done.
Writing objects: 100% (630/630), 187.49 KiB | 0 bytes/s, done.
Total 630 (delta 383), reused 0 (delta 0)
remote: Resolving deltas: 100% (383/383), done.
To git@gitlab.com:HansBug/my-greatest-project.git
 * [new branch]      master -> master
```

之后，我们就可以看到gitlab在线仓库上已经出现了我们的代码。

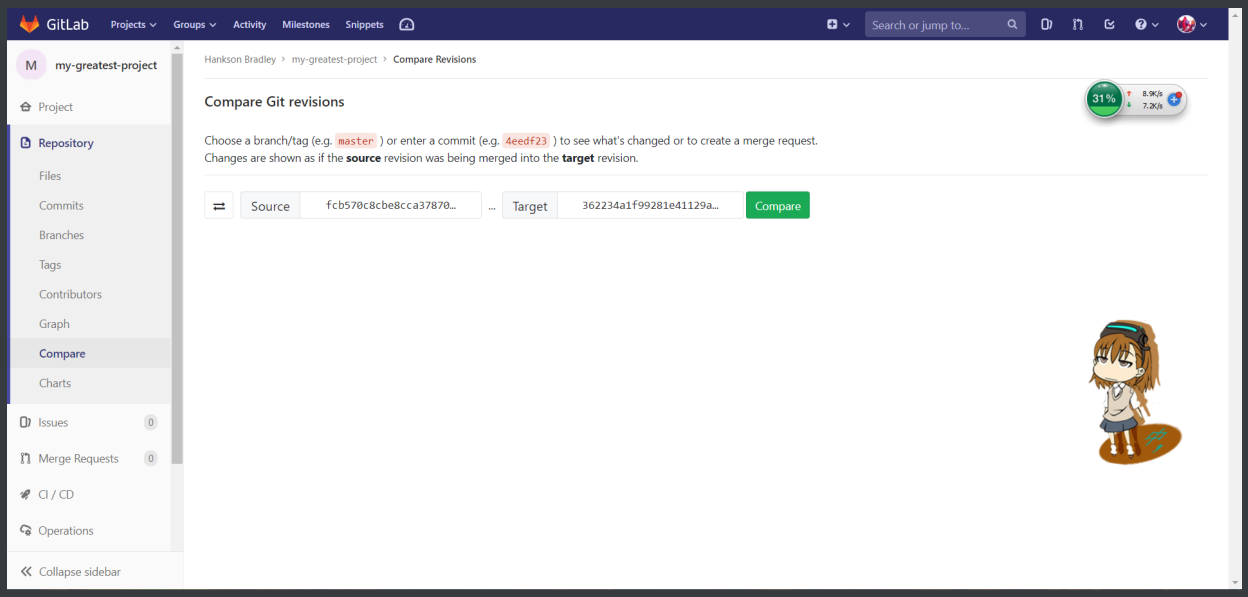


比较两次commit的改动

在项目页面上，点击 `Repository` （仓库）--> `Compare` （比较）。

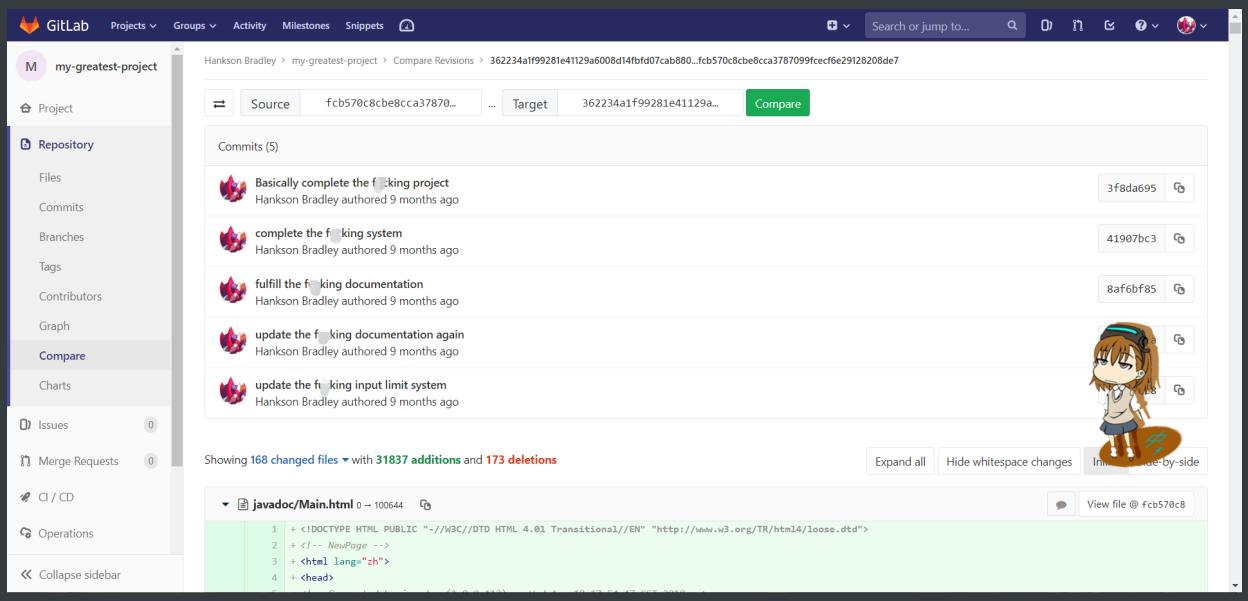


在 Source 栏填写对比的源版本号（或者分支名称），在 Target 栏填写对比的目标版本号（或者分支名称）。



点击 Compare ，即可看到对比结果。

对比结果的意思是，从 Target 所在的版本到 Source 所在的版本的代码变更状况。包括增加了哪些信息，删除了哪些信息，共计有过哪些提交等。



SSH配置

在上文中，我们说到，只要完成了ssh配置，就可以直接使用 `git@` 开头的ssh链接进行代码上传下载及管理。

在本节中，我们将讲述如何去进行ssh的配置，接下来的讲述将以Windows环境下的Git Bash为例（实际上，Linux系统上的操作也基本一样）

首先执行命令

```
ssh-keygen
```

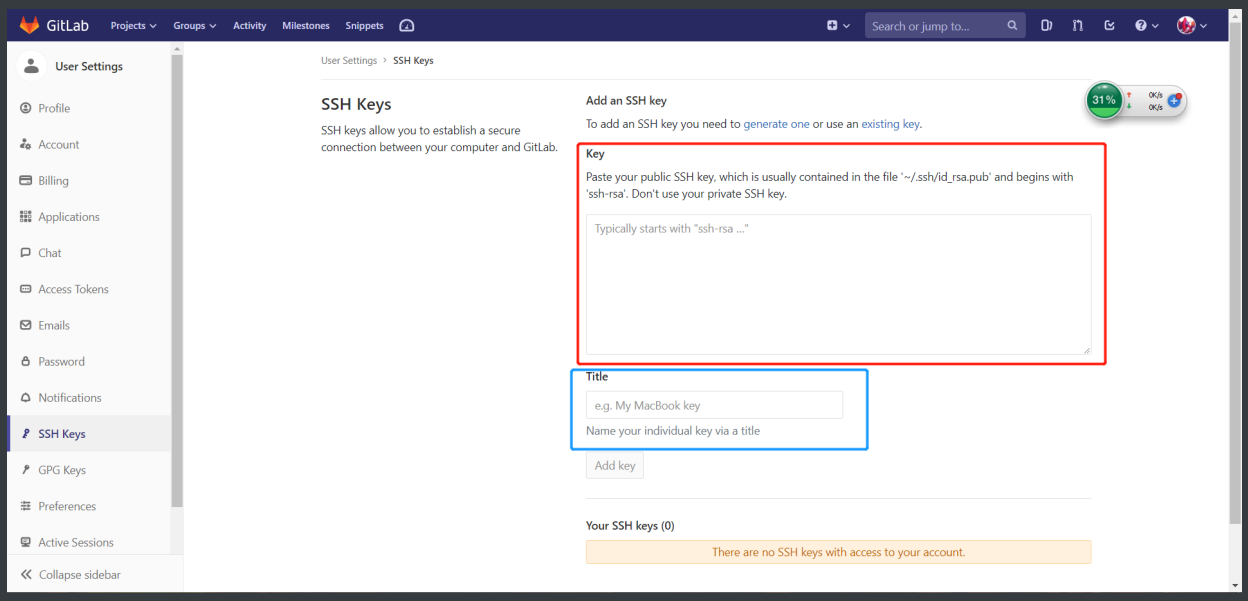
生成公钥（过程中全部选择默认即可，密码留空即可）。

接下来切到 `~/.ssh` 路径下，读取 `id_rsa.pub` 文件。

全过程如图（部分细节可能并不一样，如生成的图样、公钥内容等）

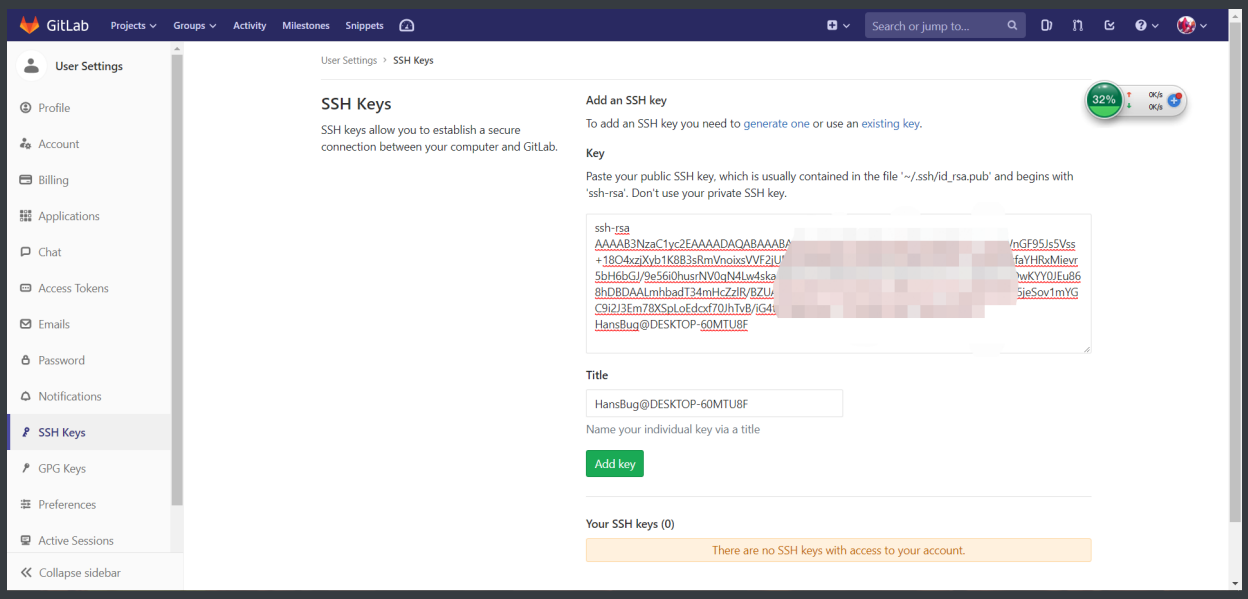
然后，将这个 `id_rsa.pub` 文件内的内容（rsa公钥），添加到平台上。

首先进入右上角用户的 `Settings` 页面，然后进入 `SSH Keys` 页面。

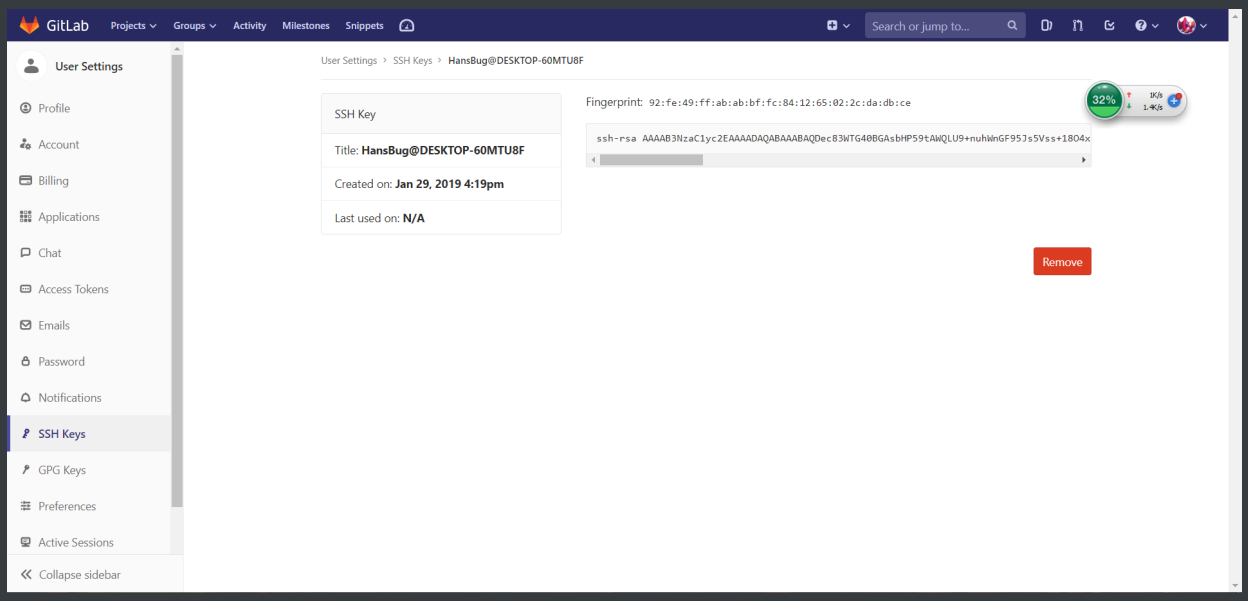


红色框的部分表示公钥信息，需要我们手动填写。

蓝色框的部分表示公钥标题，一般情况下会根据公钥信息自动生成，如果没有自动生成，则需要我们手动填写。



填写完毕后点击 `Add key` 即可。



在此之后，我们就能在使用了此默认ssh key的电脑上进行全程无密码的远程git操作了。

不过值得注意的是，一般来说对于同一个公钥，在同一个网站上，只能被同一个用户使用。

在本节，值得注意的是：

- 目前暂时使用的演示网站是gitlab的官方网站。对于其他的一些平台（如国内的coding.net，国外的github、bitbucket，以及未来课程使用的自建gitlab），基本操作也是类似的。
- 等课程系统正式完成后，我们将使用课程专用的gitlab，速度和稳定性都将超过部署于国外的gitlab官网，敬请期待。本节仅作为一个操作演示。

如果对本指南有任何问题和建议，欢迎联系作者

email: niuyazhe@buaa.edu.cn

github repo: https://github.com/OO-guide-2019/git-guide