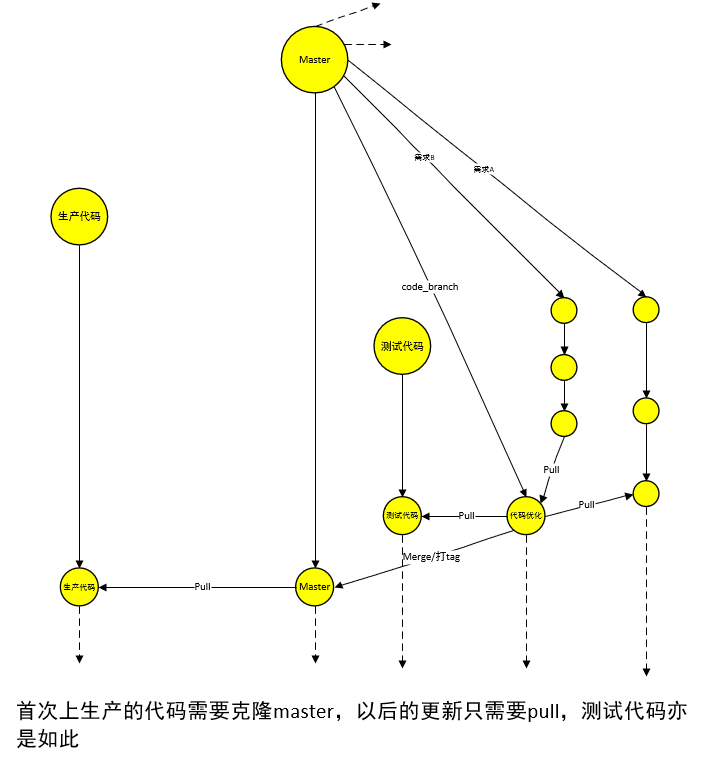
Git基础操作及GitLab使用说明

# 基本流程：



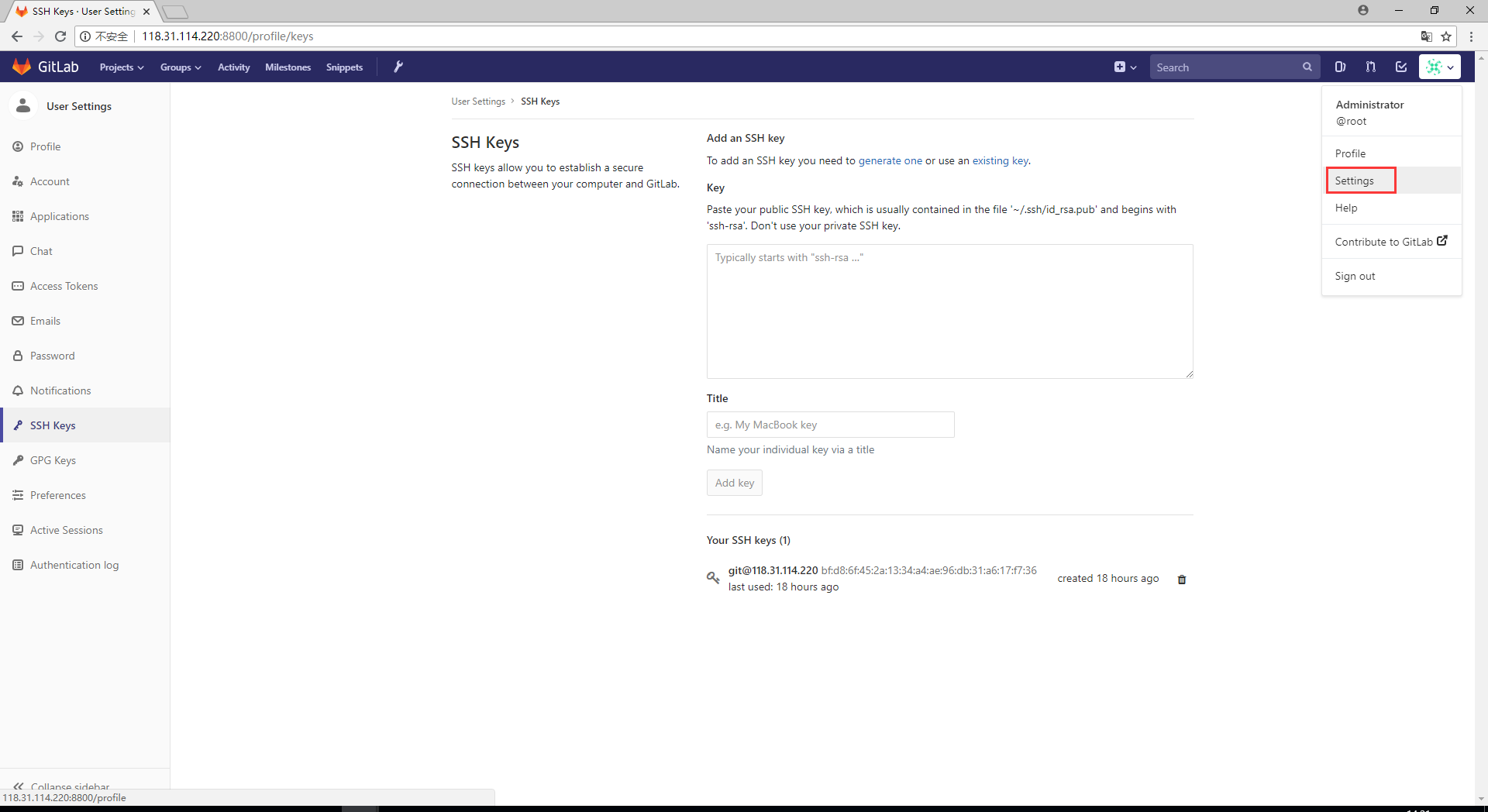
# 前提准备：

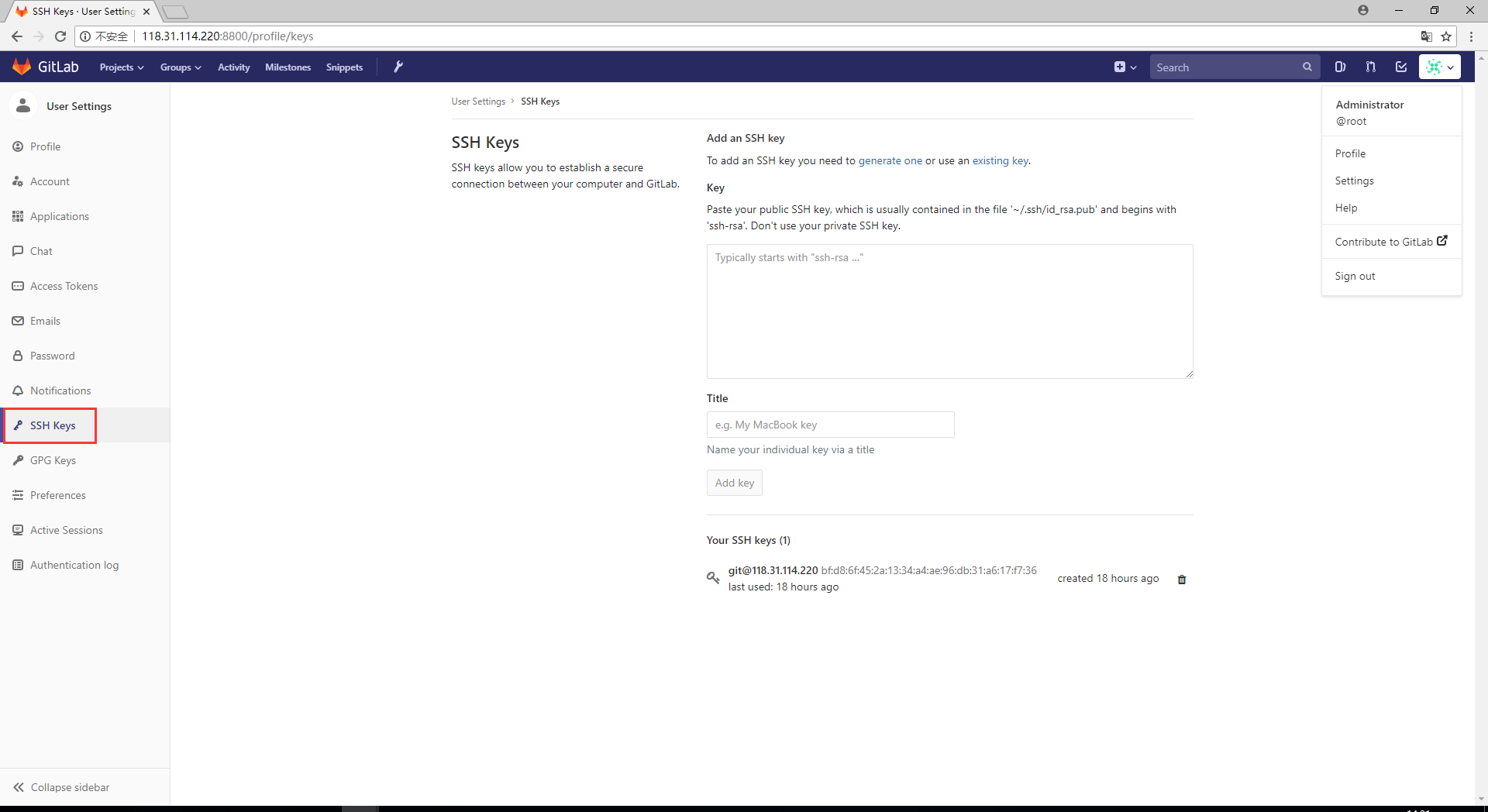
1. 每台本地电脑或服务器上需要安装git客户端
2. 相关人员的账号都为自己名字的全拼，密码默认12345678

3、 所有人在使用gitlab之前需要配置密钥登陆

(1)在本地电脑git bash里执行ssh-keygen，连续回车

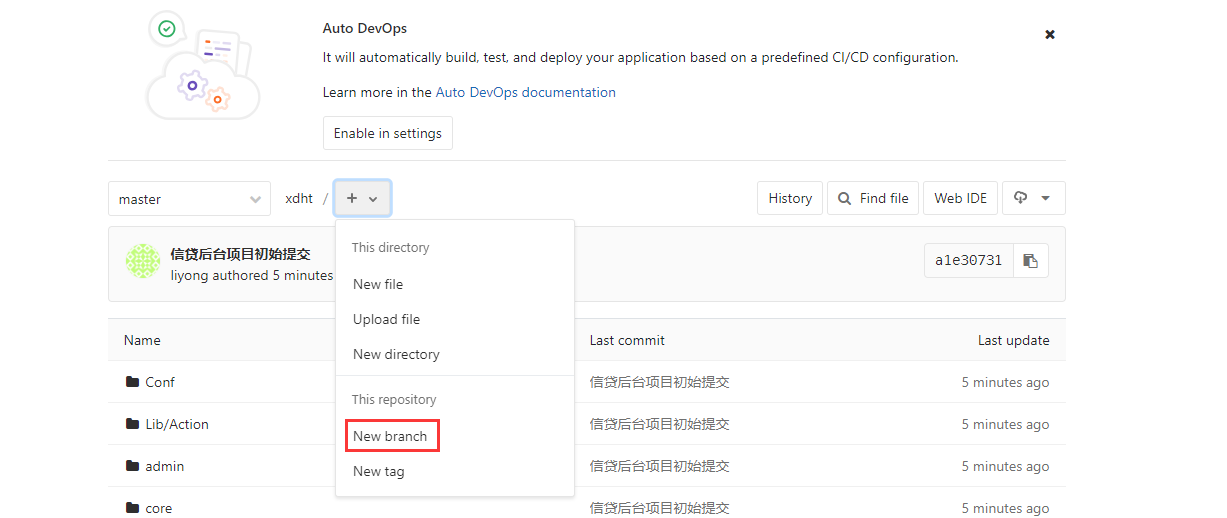
(2)打开/c/Users/mayn/.ssh/id\_rsa.pub，将内容复制到gitlab自己的账号中

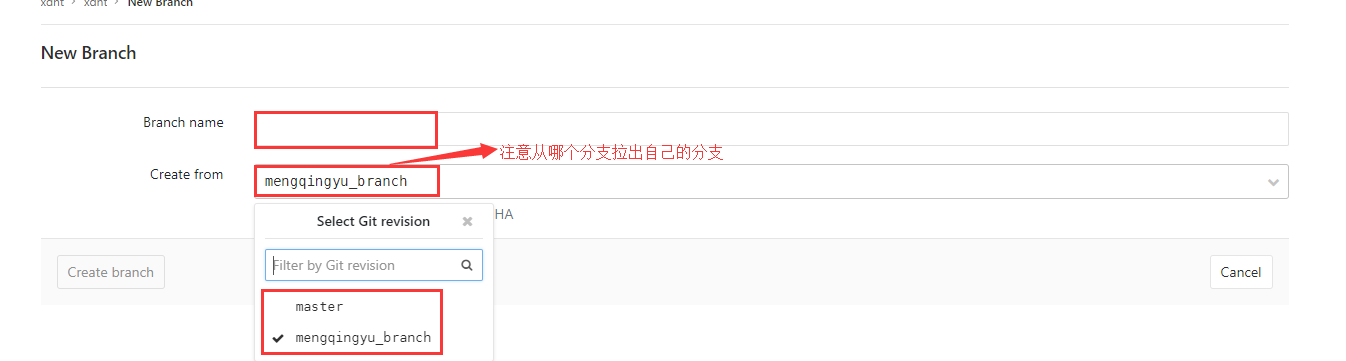




# 开发人员：

1. 开发负责人负责创建分支





1.开发负责人创建好分支之后，开发人员首次操作需要将相应的分支项目拉到本地：

git clone --branch 分支 [git@xxx.xxx.xxx.xxx:xxxx/xxxxxx.git](mailto:git@xxx.xxx.xxx.xxx:xxxx/xxxxxx.git)

git config --global user.name “name”

git config --global user.email “email”

2.查看所有分支(如果克隆master)

git branch -a

3.选择你的分支

git checkout xxx\_branch

4.每次开发前都要执行命令

git pull origin develop\_branch

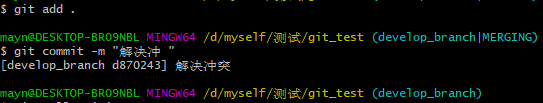
5.进行开发

6.将你的修改加入暂存区

git add .

7.将暂存区的内容加入到本地仓库

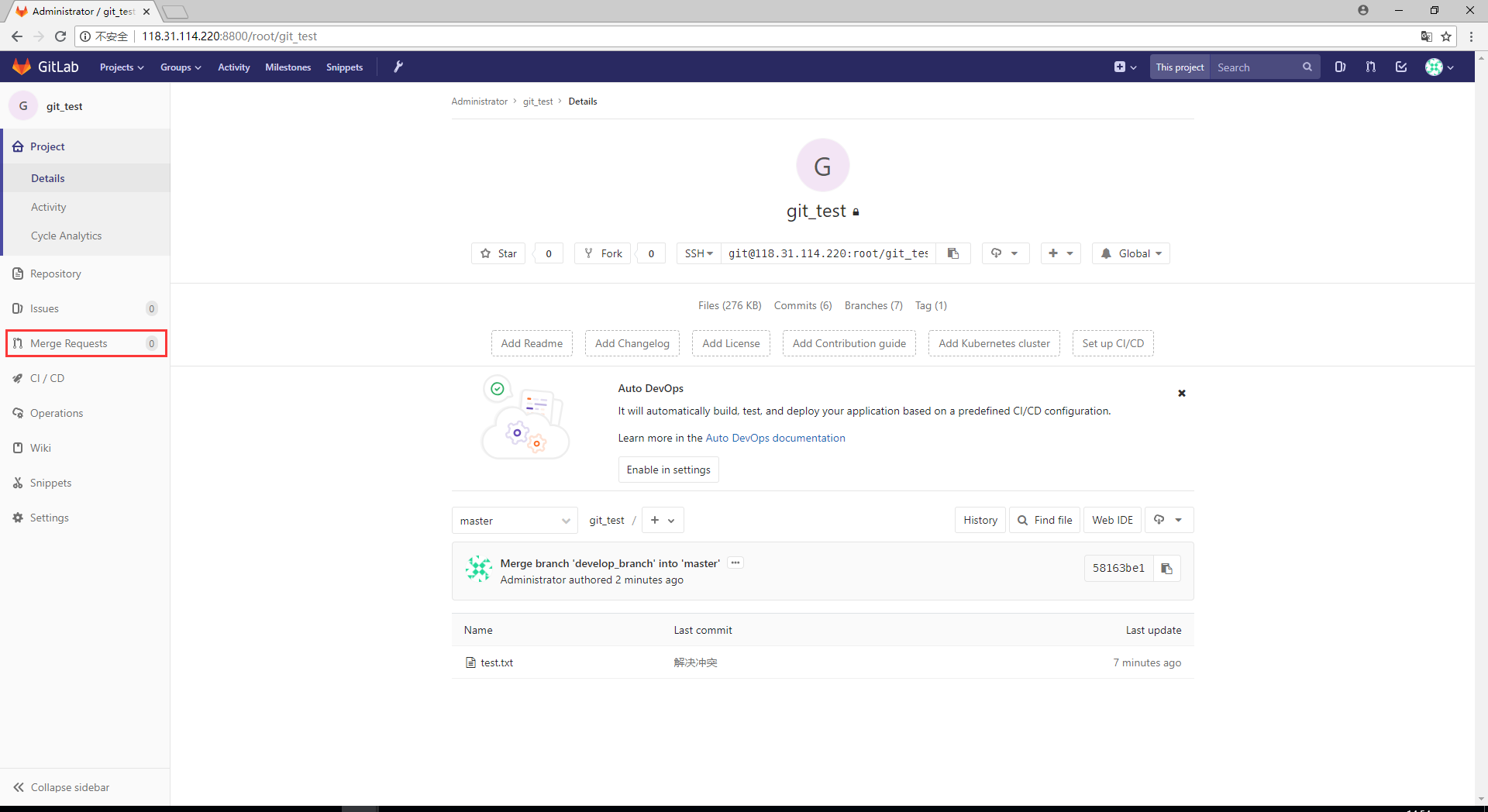
git commit -m “说明”



8.将本地仓库中的代码上传到远程分支

git push

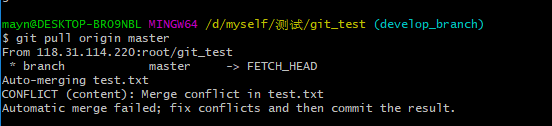
9.登陆gitlab发起合并请求



## 解决冲突

在pull远端分支最新的代码时，正常情况下可以更新为最新版本

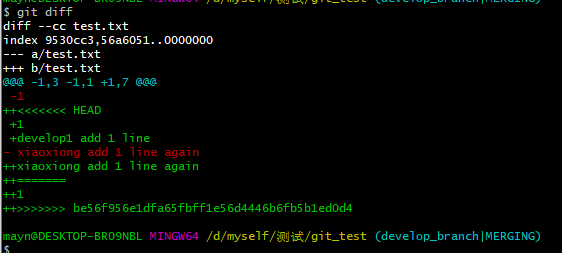
如果有冲突，通过以下几个命令来查询冲突文件



git status

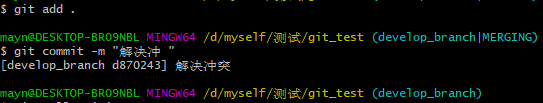


git diff

此时进入有冲突的文件中解决冲突

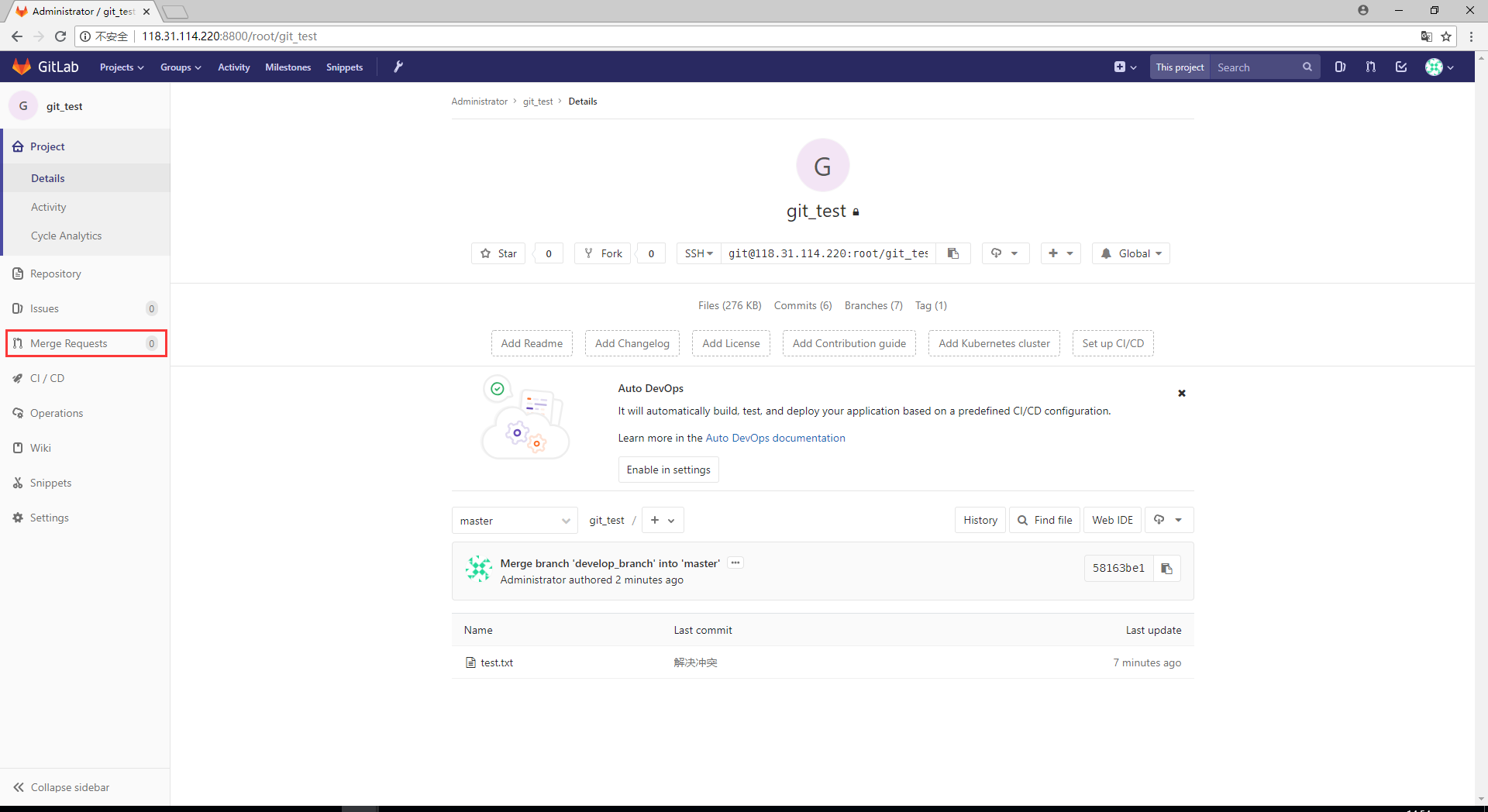
git add .

git commit -m “说明”

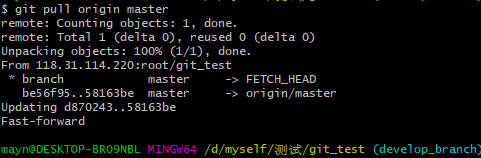


git push origin xxx\_branch

登陆gitlab发起合并请求



git pull origin xxxx\_branch



# 测试主管：

首次操作需要将xxx\_branch分支拉到测试服务器：

git clone --branch xxx\_branch git@118.31.114.220:xxxx/xxxxxx.git

git config --global user.name “name”

git config --global user.email “email”

每次有开发主管提交测试申请之后：

git pull origin xxx\_branch

# 维护者

其实可以称为master维护，有合并请求时判断是否可以合并，并负责生产服务器分支的更新，保证其与master版本一致

# 回滚操作

方案一：

如果上线的版本被测出问题，登陆到gitlab，将最近一次的合并请求撤回，生产服务器重新pull

方案二：

执行git reset --hard ORIG\_HEAD回滚刚才的pull/merge操作。

方案三：

选择git pull完后，你发现这次pull下来的修改不满意，想要回滚到pull之前的状态，从前面的介绍知道，我们可以执行git reset --hard ORIG\_HEAD，但是这个命令有个副作用就是清空你的working tree，即丢弃你的本地未add的那些改变。为了避免丢弃working tree中的内容，可以使用git reset --merge ORIG\_HEAD，注意其中的--hard 换成了 --merge，这样就可以避免在回滚时清除working tree

方案四：

如果文件只是在工作区修改了，但是还没有提交到暂存区（add），可以用git checkout --file[文件名]来撤销，如果你以及执行了add添加到了暂存区，可以使用git reset HEAD来撤销，git checkout HEAD [文件名]是以上两个命令的合成。

方案五：

git reset -hard HEAD^回退到上一个版本，HEAD^^回退到上两个版本，HEAD~100回退到上100个版本，如果回退版本你又后悔了，想要恢复最后那个版本，git reset --hard commit\_id,commit\_id是版本号，git reflog可以查看版本号

# 几种参考场景

场景一：

在实际开发中经常出现这样的情形：你正在开发一个大的feature，此时来了一个紧急的bug需要修复，但是目前在working tree中的内容还没有成型，还不足以commit，但是你又必须切换的另外的branch去fix bug。请看下面的例子

引用

$ git checkout feature ;# you were working in "feature" branch and

$ work...     ;# got interrupted

$ git commit -a -m "snapshot WIP"              (1)

$ git checkout master

$ fix...

$ git commit ;# commit with real log

$ git checkout feature

$ git reset --soft HEAD^ ;# go back to WIP state  (2)

$ git reset                           (3)

(1) 这次属于临时提交，因此随便添加一个临时注释即可。

(2) 这次reset删除了WIP commit，并且把working tree设置成提交WIP快照之前的状态。

(3) 此时，在index中依然遗留着“snapshot WIP”提交时所做的uncommit changes，git reset将会清理index成为尚未提交"snapshot WIP"时的状态便于接下来继续工作

场景二：

Reset单独的一个文件

假设你已经添加了一个文件进入index，但是而后又不打算把这个文件提交，此时可以使用git reset把这个文件从index中去除。

引用

$ git reset -- frotz.c                  (1)

$ git commit -m "Commit files in index"     (2)

$ git add frotz.c                     (3)

(1) 把文件frotz.c从index中去除，

(2) 把index中的文件提交

(3) 再次把frotz.c加入index

场景三：

保留working tree并丢弃一些之前的commit

假设你正在编辑一些文件，并且已经提交，接着继续工作，但是现在你发现当前在working tree中的内容应该属于另一个branch，与这之前的commit没有什么关系。此时，你可以开启一个新的branch，并且保留着working tree中的内容。

引用

$ git tag start

$ git checkout -b branch1

$ edit

$ git commit ...                        (1)

$ edit

$ git checkout -b branch2                   (2)

$ git reset --keep start                    (3)

(1) 这次是把在branch1中的改变提交了。

(2) 此时发现，之前的提交不属于这个branch，此时你新建了branch2，并切换到了branch2上。

(3) 此时你可以用reset --keep把在start之后的commit清除掉，但是保持working tree不变。

场景四：

将更新下来的内容和本地修改的内容有冲突，先提交你的改变或者先将本地修改暂时存储起来。

一般来说遇到这种情况，首先commit本地的修改，

然后pull版本库的代码，修改发生冲突的文件，就可以正常merge和push操作了。

这里git为什么会提示在merge之前做这些操作呢？

merge manual中有一条警告：

Warning: Running git merge with uncommitted changes is discouraged:

while possible, it leaves you in a state that is hard to back out of in the case of a conflict.

有未提交修改情况下，不要执行merge，很可能会丢失修改。

看一下如何使用stash暂存代码：

1、先将本地修改存储起来

$ git stash

这样本地的所有修改就都被暂时存储起来 。是用git stash list可以看到保存的信息：

2、pull内容

暂存了本地修改之后，就可以pull了。

$ git pull

3、还原暂存的内容

$ git stash pop stash@{0}

4、移除暂存的内容

$git stash drop 相应的如stash@{0}，或者git clear清除所有暂存

场景五：

1.You have not concluded your merge. (MERGE\_HEAD exists)

本地有修改和提交，如何强制用远程的库更新。  
**出现这种情况一般是git本地有commit，但是不能更新，也不能cleanup，导致无法push到服务器。**

使用git pull -f 强制更新，仍然提示 You have not concluded your merge. (MERGE\_HEAD exists)。

这种情况，可以**做好备份，放弃本地修改，用远程内容reset本地目录**，当然新建目录重新clone也可以。

以下是使用reset命令使用远程内容强制更新本地目录，  
git fetch 下载远程的库的内容，不做任何的合并，git reset 把HEAD指向刚刚下载的最新的版本。  
**注意，这样操作，之前所做的修改会被覆盖！！！谨慎使用！**

git fetch --all

git reset --hard origin/master(根据本地版本)

场景六：

删除 untracked files

git clean -f

# 连 untracked 的目录也一起删掉

git clean -fd

# 连 gitignore 的untrack 文件/目录也一起删掉 （慎用，一般这个是用来删掉编译出来的 .o之类的文件用的）

git clean -xfd

# 在用上述 git clean 前，强烈建议加上 -n 参数来先看看会删掉哪些文件，防止重要文件被误删

git clean -nxfd

git clean -nf

git clean -nfd

场景六：

错误：Updates were rejected because the tip of your current branch is behind

有如下几种解决方法：  
1.使用强制push的方法：  
$ git push -u origin master -f   
这样会使远程修改丢失，一般是不可取的，尤其是多人协作开发的时候。  
2.push前先将远程repository修改pull下来  
$ git pull origin master  
$ git push -u origin master  
3.若不想merge远程和本地修改，可以先创建新的分支：  
$ git branch [name]  
然后push  
$ git push -u origin [name]