**引言**

一般来说，安装好 git 后，我们都会配置一个全局的 config 信息，就像这样：

git config --global user.name "username" // 配置全局用户名

git config --global user.email "account@xxx.com" // 配置全局邮箱

但是你可能会碰到需要在一台电脑上配置多个用户信息的需求。此时就不能够用一个全局配置搞定一切了。比如因为我的个人电脑出了问题，我想要提交我的个人项目时，只能用公司配的电脑去提交。而公司的电脑配置的是私有的 gitlab 仓库，而我自己的项目存储在 github 上。这两个仓库不仅仓库地址不一样，仓库的用户名和邮箱都不一样。

**配置多用户**

本文将配置分别是 github 以及 gitlab 上的两个用户，并分别在它们所属的项目上进行 git 操作，这差不多就是配置多用户的大部分操作了。

Gitlab

用户名 gitlabuser

邮箱 [gitlabmail@ xxx.com](mailto:gitlabmail@bytedance.com)

Github

用户名 githubuser

邮箱 [githubmail@qq.com](mailto:githubmail@qq.com)

**清除全局配置**

在正式配置之前，我们先得把全局配置给清除掉（如果你配置过的话），执行命令：

git config --global –list

这会列出所有已经配置的全局配置，如果你发现其中有 user.name 和 user.email 信息，请执行以下命令将其清除掉：

git config --global --unset user.name  
git config --global --unset user.email

**生成钥对**

钥对的保存位置默认在 ~/.ssh 目录下，我们先清理下这个目录中已存在的钥对信息，即删除其中的 id\_rsa、id\_rsa.pub 之类的公钥和密钥文件。

首先我们开始生成 github 上的仓库钥对，通过 -C 参数填写 github 的邮箱：

ssh-keygen -t rsa -C “[gitlabmail@ xxx.com](mailto:gitlabmail@bytedance.com)”  
按下 ENTER 键后，会有如下提示：

Generatingpublic/privatersa key pair.Enter fileinwhich to save the key (/Users/wzc/.ssh/id\_rsa):

在这里输入公钥的名字，默认情况是叫 id\_rsa，为了和后面的 gitlab 配置区分，这里输入 Users/wzc/.ssh/id\_rsa\_gitlab。输入完毕后，一路回车，钥对就生成完毕了。

下面开始生成 gitlab 上的仓库钥对，步骤和上面一样：

ssh-keygen -t rsa -C “[githubmail@qq.com](mailto:githubmail@qq.com)”  
生成的公钥名就叫做（不要忘记前面的路径）：id\_rsa\_github。

**添加 SSH Keys**

我相信你既然都看到这篇文章了，你一定掌握了如何将公钥添加到 SSH Keys 中。请将 id\_rsa\_github.pub 和 id\_rsa\_gitlab.pub 内容分别添加到 github 和 gitlab 的 SSH Keys 中，这里就不啰嗦了。

**添加私钥**

在上一步中，我们已经将公钥添加到了github 或者 gitlab 服务器上，我们还需要将私钥添加到本地中，不然无法使用。添加命令也十分简单，如下：

ssh-add ~/.ssh/id\_rsa\_github // 将 GitHub 私钥添加到本地  
ssh-add ~/.ssh/id\_rsa\_gitlab // 将 GitLab 私钥添加到本地  
添加完毕后，可以通过执行 ssh-add -l 验证下，如果都能显示出来和下面一样，就 OK 了。

2048 SHA256:mXVNxWHZsZpKOnHlPslF2jXAWR+jc7M6P5hYbrCo jitwxs@foxmail.com (RSA)

2048 SHA256:Blhp3+Hx5mp9HDivFjDuwc/PaQ8ux45TRa6nTsfIe0PEz4 lemon@test.com (RSA)

**管理密钥**

通过以上步骤，公钥、密钥分别被添加到 git 服务器和本地了。下面我们需要在本地创建一个密钥配置文件，通过该文件，实现根据仓库的 remote 链接地址自动选择合适的私钥。

编辑 ~/.ssh 目录下的 config 文件，如果没有，请创建。

vim ~/.ssh/config

配置内容如下：

Host github

HostName github.com

User jitwxs

IdentityFile ~/.ssh/id\_rsa\_github

Host gitlab

HostName gitlab.mygitlab.com

User lemon

IdentityFile ~/.ssh/id\_rsa\_gitlab

该文件分为多个用户配置，每个用户配置包含以下几个配置项：

Host：仓库网站的别名，随意取  
HostName：仓库网站的域名（PS：IP 地址应该也可以）  
User：仓库网站上的用户名  
IdentityFile：私钥的绝对路径

注： Host 就是可以替代 HostName 来使用的别名，比如我 github 上某个仓库的 clone 地址为：

[git@github.com:wzc/express.git](mailto:git@github.com:wzc/express.git)

那么使用 Host 后就是：

[git@github:wzc/express.git](mailto:git@github:wzc/express.git)

咳咳，反正我觉得没啥用，毕竟 remote 地址都是直接复制下来的，没人会手敲吧？

可以用 ssh -T 命令检测下配置的 Host 是否是连通的：

ssh -T git@github

Hi xxx! You've successfully authenticated, but GitHub does not provide shell access.

ssh -T git@gitlab

Welcome to GitLab, @xxx!

当然不用 Host 用 HostName 也是一样的。

**注意：如果在测试过程中出现了host key verification failed.提示信息，命令行输入ssh -o StrictHostKeyChecking=no git@github.com将github真实性添加到您的known\_hosts。对于gitlab也是如此。**

**仓库配置**

恭喜你！完成以上配置后，其实你已经基本完成了所有配置。分别进入附属于 github 和 gitlab 的仓库，此时都可以进行 git 操作了。但是别急，如果你此时提交仓库修改后，你会发现提交的用户名变成了你的系统主机名。

这是因为 git 的配置分为三级别，System —> Global —>Local。System 即系统级别，Global 为配置的全局，Local 为仓库级别，优先级是 Local > Global > System。

因为我们并没有给仓库配置用户名，又在一开始清除了全局的用户名，因此此时你提交的话，就会使用 System 级别的用户名，也就是你的系统主机名了。

因此我们需要为每个仓库单独配置用户名信息，假设我们要配置 github 的某个仓库，进入该仓库后，执行：

git config --local user.name "username"

git config --local user.email "githubaccount@qq.com"

执行完毕后，通过以下命令查看本仓库的所有配置信息：

git config --local –list

至此你已经配置好了 Local 级别的配置了，此时提交该仓库的代码，提交用户名就是你设置的 Local 级别的用户名了。当然，你也可以配置一个全局的用户名信息，这样默认的就是Global级别的用户名了。