# Laravel

Git 使用手册

### 初次使用

git 和git仓库产生联系 需要先在本地安装git客户端 并且生成密钥

设置账号密码

步骤：

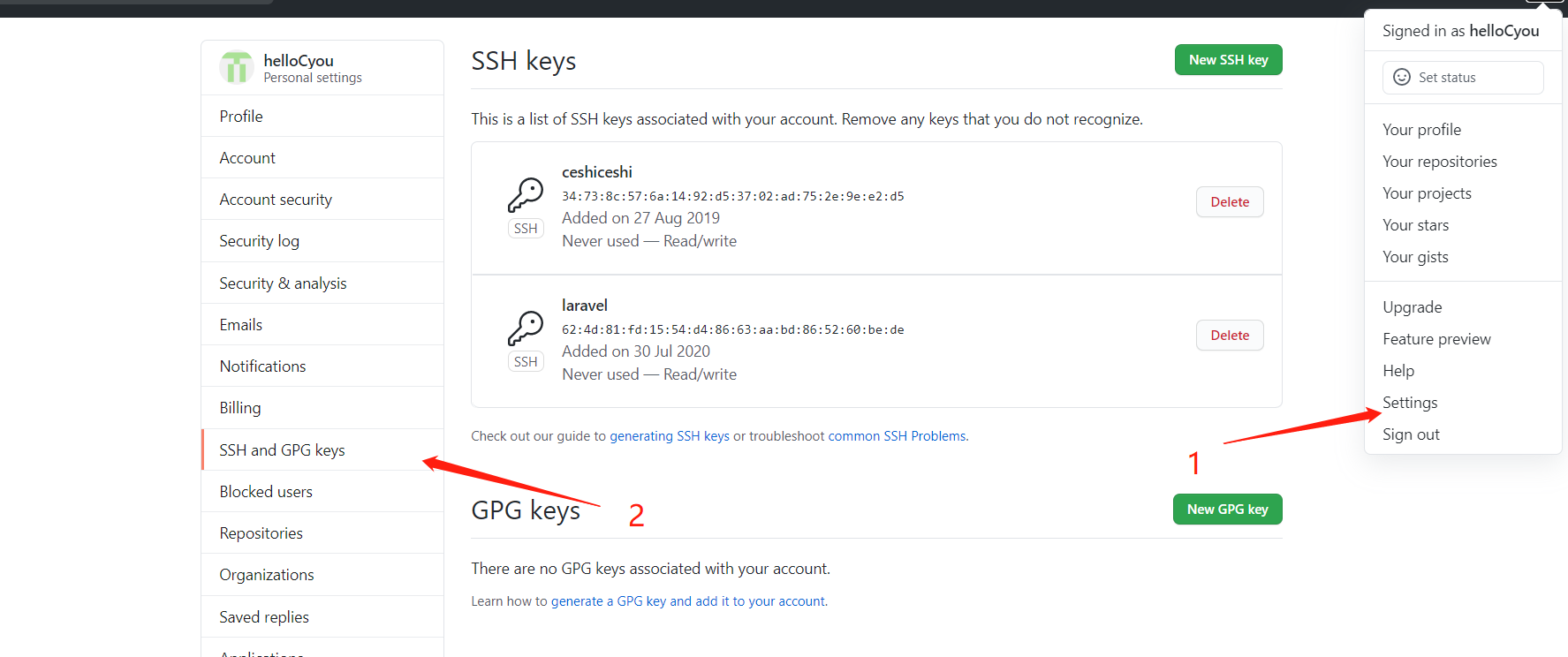
git config --global user.name 您的名字  
然后验证该配置是否正确：  
  
git config --global user.name  
使用如下命令设置邮箱地址：  
  
git config --global user.email 您的邮箱地址  
使用如下命令验证邮箱设置是否正确：  
  
git config --global user.email  
由于使用了“--global”选项，所以您只需要做一次这样的配置。Git会在您系统的任何项目中使用这个信息。如果您希望在某个特定的项目中使用不同的用户名和邮箱地址，可以在那个项目中重新运行该命令，但不加“--global”选项。

ssh-keygen -t rsa -C ""  
这个指令会要求你提供一个位置和文件名去存放键值对和密码，你可以点击Enter键去使用默认值。  
  
用以下命令获取你生成的公钥：  
  
cat ~/.ssh/id\_rsa.pub

<https://www.cnblogs.com/bliss-/p/13344036.html>

把生成的.ssh文件夹扔到文件的根目录下

获取密钥后去git上填写密钥、



##### Git 常用命令

###### 增加删除文件

git add . 提交当前文件夹全部文件

git add [dir] 添加指定目录到暂存区，包括子目录

git rm [file1] [file2] ... 删除工作区文件，并且将这次删除放入暂存区

###### 代码提交

git commit [file1] [file2] ... -m [message] 提交暂存区的指定文件到仓库区

git commit –m “备注” 暂存区文件全部提交到本地仓库

git commit –v 提交时显示所有不同的信息

###### 分支

git branch 列出所有本地分支

git branch –r 列出所有远程分支

git branch –a 列出所有本地分支和远程分支

git branch [branch-name] 新建一个分支，但依然停留在当前分支

git checkout -b [branch] 新建一个分支，并切换到该分支

git branch [branch] [commit] 新建一个分支，指向指定commit

git branch --track [branch] [remote-branch] 新建一个分支，与指定的远程分支建立追踪关系

git checkout [branch-name] 切换到指定分支，并更新工作区

git merge [branch] 合并指定分支到当前分支

git branch -d [branch-name] 删除分支

###### 查看信息

git status 显示有变更的文件

git log 显示当前分支的版本历史

git log -5 --pretty –oneline 显示过去5次提交

git fetch [remote] 下载远程仓库的所有变动

git remote –v 显示所有远程仓库

git remote add [shortname] [url] 增加一个新的远程仓库，并命名

git pull [remote] [branch] 取回远程仓库的变化，并与本地分支合并

git push [remote] [branch] 上传本地指定分支到远程仓库

git push [remote] –force 强行推送当前分支到远程仓库，即使有冲突

git push [remote] –all 推送所有分支到远程仓库

命令：

创建seed文件 database/seeds/UserSeeder.php

在databaseSeeder.php 文件中注册 $this->call(UserSeeder::class);

artisan db:seed 运行seed文件 填充数据

artisan migrate:refresh 重置表

artisan migrate:refresh –seed 重置表并填充数据

### 模型工厂的使用

模型工厂用来给数据库生成测试数据

artisan make:factory BlogFactory --model=Blog

用这个命令 生成工厂 绑定模型为Blog

$factory->define(Blog::**class**, **function** (*Faker* $faker) {  
 **return** [  
 //  
 'content'=>$faker->text(100),  
 'user\_id'=>$faker->randomElement([1,2,3])  
 ];  
});

选择 填充数据的类型

artisan make:seeder BlogSeeder

生成 对应的seeder文件

**public function** run()  
{  
 //  
 factory(\App\Blog::**class**,200)->create();  
}

命令 生成 200条

到DatabaseSeeder 文件中 注册

**public function** run()  
{  
 $this->call(BlogSeeder::**class**);

}

artisan migrate:refresh --seed 执行这条命令

连表

**class** IndexController **extends** Controller  
{  
 //  
 **public function** home()  
 {  
 $blogs = Blog::*orderby*('id','DESC')->with('user')->paginate('10');  
 **return** view("home",compact('blogs'));  
 }  
}

With（‘uses’）连的是 blog 模型中的user方法

**class** Blog **extends** Model  
{  
 //这个字段可以允许我们 批量填充  
 **protected** $fillable = ['content'];  
  
  
 **public function** User(){  
 //用来链表  
 **return** $this->belongsTo(User::**class**);  
 }  
}

User方法 通过 blog表中的User\_id 反向寻找找 user 表中的id

### 策略模型

artisan make:policy --model=Blog BlogPolicy 绑定模型 并生成 文件

生成策略模型 用与功能的身份验证功能

**public function** delete(User $user, Blog $blog)  
{  
 //  
 **return** $user->is\_admin || $user->id == $blog->user\_id;  
}

判断是否是管理员 或者 删除模型的id 是否是自己的id

**public function** destroy(Blog $blog)  
{  
 //  
 $this->authorize('delete',$blog);  
  
}

调用模型策略进行判断

**@can(**'delete',$blog**)**<form action="{{route('blog.destroy',$blog)}}" method="post">  
 **@csrf @method(**"DELETE"**)** <button class="btn btn-danger">删除</button> //前端进行判断是否显示@can  
</form>  
 **@endcan**

### 用命令创建数据库的表

make:migration --create=blogs create\_blogs\_table

artisan migrate:refresh --seed 执行