安装配置

自诞生于 2005 年以来，Git 日臻成熟完善，在高度易用的同时，仍然保留着初期设定的目标。它的速度飞快，极其适合管理大项目 。
Git 可以在 windows、Mac、Linux 全平台系统使用。登录 https://git-scm.com/downloads 下载你系统的 Git 软件并进行安装。

> windows 用户我更建议安装 git for windows ， 下载地址： https://gitforwindows.org/包信 Git Base 、Git Gui

安装后通过以下命令查看，如果显示版本号那就是安装成功了

```
git --version
```

Gui
Gui 指 Git 的图形界面管理软件，https://git-scm.com/downloads/guis 这个网址列出了多个可供基本上所有平台使用的 Gui 软件。如果要使用 Gui 而非命令行操作，我推荐 `sourcetree` 这也是我多年使用的软件，功能强大、跨平台、免费。

> 新人建议直接使用命令行管理 GIT

#初始配置

配置文件为 `~/.gitconfig` ，执行任何 Git 配置命令后文件将自动创建。
第一个要配置的是你个人的用户名称和电子邮件地址。这两条配置很重要，每次 Git 提交时都会引用这两条信息，说明是谁提交了更新，所以会随更新内容一起被永久纳入历史记录：

```
git config --global user.email "2300071698@qq.com"
git config --global user.name "2300071698@qq.com"
```

#基础入门

#常用命令

1. 初始化新仓库

git init

2. 克隆代码

git clone https://gitee.com/houdunwang/hdcms.git

3. 克隆指定分支

git clone –b dev git@gitee.com:houdunwang/hdcms.git

4. 查看状态

git status

5. 提交单个文件

git add index.php

6. 提交所有文件

git add –A

7. 使用通配符提交

git add *.js

8. 提交到仓库中

git commit –m '提示信息'

9. 提交已经跟踪过的文件，不需要执行 add

git commit –a –m '提交信息'

10. 删除版本库与项目目录中的文件

git rm index.php

11. 只删除版本库中文件但保存项目目录中文件

git rm ––cached index.php

12. 修改最后一次提交

git commit ––amend

#基础流程

1. 首先克隆你的项目

```
git clone
https://gitee.com/houdunwang/hdcms.git
git clone
```

2. 开始开发添加新文件 hd.js，这时新的文件并没有被版本库管理，可以通过以下命令查看没有被管理的文件

```
git clean -n
```

3. 将所有文件提交到暂存区

```
git add .
```

这时再通过 `clean` 命令查看会发现结果为空，即文件已经被版本库管理了

```
git clean -n
```

4. 不小心将工作区中的 hd.js 文件删除了，现在可以将暂存区中的 hd.js 恢复回来

```
git checkout hd.js
```

5. 完成工作后创建一个新的提交，并使用 –m 选项说明完成的工作

```
git commit -m '购物车开发'
```

6. 将代码提交到远程服务器，与同事或其他开发者分享代码

```
git push
```

#工作区

git clean 命令用来从工作目录中删除所有没有跟踪（tracked）过的文件

1. git clean –n 是一次 clean 的演习，告诉你哪些文件会被删除
2. git clean –f 删除当前目录下没有 tracked 过的文件，不会删除.gitignore 指定的文件
3. git clean –df 删除当前目录下没有被 tracked 过的文件和文件夹
4. git checkout . 将没有放入到暂存区的所有文件恢复
5. git checkout hd.js 放弃指定文件的修改
6. git checkout –– hd.js 将文件从暂存区恢复（如果没有提交到暂存区，将恢复到最近版本）

#暂存区

1. git add . 提交所有修改和新增的文件
2. git add –u 只提交修改文件不提交新文件
3. git ls–files –s 查看暂存区文件列表
4. git cat–file –p 6e9a94 查看暂存区文件内容
5. git reset 撤销上次提交到暂存区动作

#日志查看

1. 查看日志

`git log`

2. 查看最近 2 次提交日志并显示文件差异

`git log -p -2`

3. 显示已修改的文件清单

`git log --name-only`

4. 显示新增、修改、删除的文件清单

`git log --name-status`

5. 一行显示并只显示 SHA–1 的前几个字符

`git log --oneline`

6. 下面是自定义的精简日志信息

```
git log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit

```

向军大叔在 `~/.zshrc` 配置文件中定义了别名

```
alias gl="git log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"

```

运行结果如下

```
* 9bb43b6 - (HEAD -> master, origin/master, origin/HEAD) 开始编写坦克游戏，画布类与模型类 (25 minutes ago) <后盾人>
* 2d61dcc - canvas 基本使用 (3 weeks ago) <后盾人>
* 4167d04 - init commit (3 weeks ago) <后盾人>
```

```
4
```

#分支管理

分支用于为项目增加新功能或修复 Bug 时使用。

#分支流程

大部分情况下不会直接在 master 分支工作，我们应该保护这个分支是最终开发完成代码健康可交付运行的。

所以功能和缺陷(bug)修复都会新建分支完成，除了这个概念外与基本流程使用是一样的。

1. 新建支付功能开发分支

```
git branch pay
```

2. 切换到新分支开始开发，这里的工作内容与上面的基础流程是一样的

```
git checkout pay
```

3. 开发完成执行提交

```
git commit -m 'H5 支付功能'
```

4. 合并分支到 master

```
切换到master分支
git checkout master

合并pay分支的代码
git merge pay
```

5. 提交代码到 master 远程分支

```
git push
```

#常用命令

1. 创建分支

`git branch dev`

2. 查看分支

`git branch`

3. 切换分支

`git checkout dev`

4. 创建并切换分支

`git checkout -b feature/bbs`

5. 将分支 main 更新为 master

`git branch -m main master`

6. 合并 dev 分支到 master

```
git checkout master
git merge dev
```

7. 删除分支

`git branch -d dev`

8. 删除没有合并的分支

`git branch -D dev`

9. 删除远程分支

`git push origin :dev`

10. 查看未合并的分支(切换到 master)

`git branch --no-merged`

11. 查看已经合并的分支(切换到 master)

`git branch --merged`

#历史版本

下面演示基于历史版本创建分支
首先查看历史版本提交日志

```
git log
```

切换到提交的 commit-id 历史版本

```
git checkout commit-id
```

以历史版本 `commit-id` 创建新分支

```
git checkout commit-id -b 新分支名称
```

#reset

使用 reset 恢复到历史提交点，重置暂存区与工作目录的内容。

#可选参数

reset 有三个选项可使用

1. --hard
   重置位置的同时，直接将 **working Tree 工作目录**、 **index 暂存区**及 **repository** 都重置成目标**Reset** 节点的内容
2. --soft
   重置位置的同时，保留**working Tree 工作目录**和**index 暂存区**的内容，只让**repository**中的内容和 **reset** 目标节点保持一致
3. --mixed（默认）
   重置位置的同时，只保留**Working Tree 工作目录**的内容，但会将 **Index 暂存区** 和 **Repository** 中的内容更改和 reset 目标节点一致

#使用示例

1. git reset 将 add 到暂存区的内容回退
2. git reset --hard b7b73147ca8d6fc20e451d7b36 恢复到指定提交版本（先通过 git log 查看版本号），重置 stage 区和工作目录里的内容。
3. git reset --hard HEAD^^^ 恢复前三个版本

4. git reset --soft 保留工作区的内容，只回退 commit 的动作。保留 **working tree 工作目录**的内容，**index 暂存区**与 **working tree** 工作目录的内容一致，只是仓库**repository** 中的内容的改变。
5. git reset HEAD -- . 撤销暂存区的文件
6. git reset --hard 清空工作区和暂存区的改动
7. git reset HEAD hd.js 放弃已经 add 暂存区的文件 hd.js

#其他知识

#定义别名

通过创建命令别名可以减少命令输入量，有几种方式进行设置

配置文件定义

修改配置文件 ~/.gitconfig 并添加以下命令别名配置段

```
[alias]
        a = add .
        c = commit
        s = status
        l = log
        b = branch

```

现在可以使用 `git a` 实现 `git add .` 一样的效果了。

系统配置定义

window 用户可以修改`~/.bashrc` 或 `~/.bash_profile`文件。

mac/linux 修改 `~/.zshrc` 文件中定义常用的别名指令，需要首先安装 zsh 命令行扩展

```
alias gs="git status"
alias gc="git commit -m "
alias gl="git log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"
alias gb="git branch"
alias ga="git add -A"
alias go="git checkout"
alias gp="git push;git push github"

```

命令行直接使用 `gs` 即可以实现 `git status` 一样的效果了。

> window 系统需要使用 git for window 中的 `Git Base` 软件

#.gitignore

.gitignore 用于定义忽略提交的文件

- 所有空行或者以注释符号

开头的行都会被 Git 忽略。

- 匹配模式最后跟反斜杠（

/）说明要忽略的是目录。

- 可以使用标准的 glob 模式匹配。

```
.idea
/vendor
.env
/node_modules
/public/storage
*.txt

```

#冲突解决

不同分修改同一个文件或不同开发者修改同一个分支文件都可能造成冲突，造成无法提交代码。

1. 使用编辑器修改冲突的文件
2. 添加暂存

git add . 表示已经解决冲突

3. git commit 提交完成

#Stashing

当你正在进行项目中某一部分的工作，里面的东西处于一个比较杂乱的状态，而你想转到其他分支上进行一些工作。问题是，你不想提交进行了一半的工作，否则以后你无法回到这个工作点。

"暂存" 可以获取你工作目录的中间状态——也就是你修改过的被追踪的文件和暂存的变更——并将它保存到一个未完结变更的堆栈中，随时可以重新应用。

1. 储藏工作

git stash

2. 查看储藏列表

git stash list

3. 应用最近的储藏

git stash apply

4. 应用更早的储藏

git stash apply stash@{2}

5. 删除储藏

git stash drop stash@{0}

6. 应用并删除储藏

git stash pop

#Tag

Git 也可以对某一时间点上的版本打上标签 ， 用于发布软件版本如 v1.0

1. 添加标签

git tag v1.0

2. 列出标签

git tag

3. 推送标签

git push --tags

4. 删除标签

git tag -d v1.0.1

5. 删除远程标签

git push origin :v1.0.1

#打包发布

对 mster 分支代码生成压缩包供使用者下载使用，`--prefix` 指定目录名

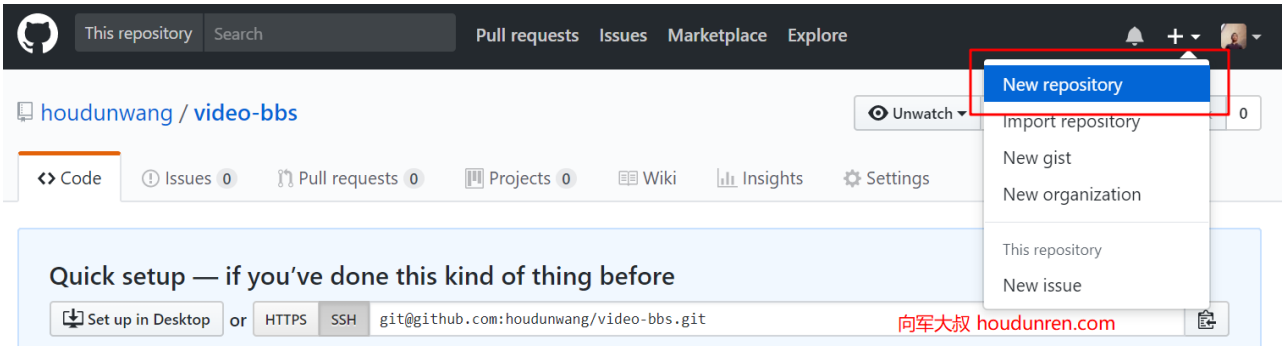
```
git archive master --prefix='hdcms/' --format=zip > hdcms.zip

```

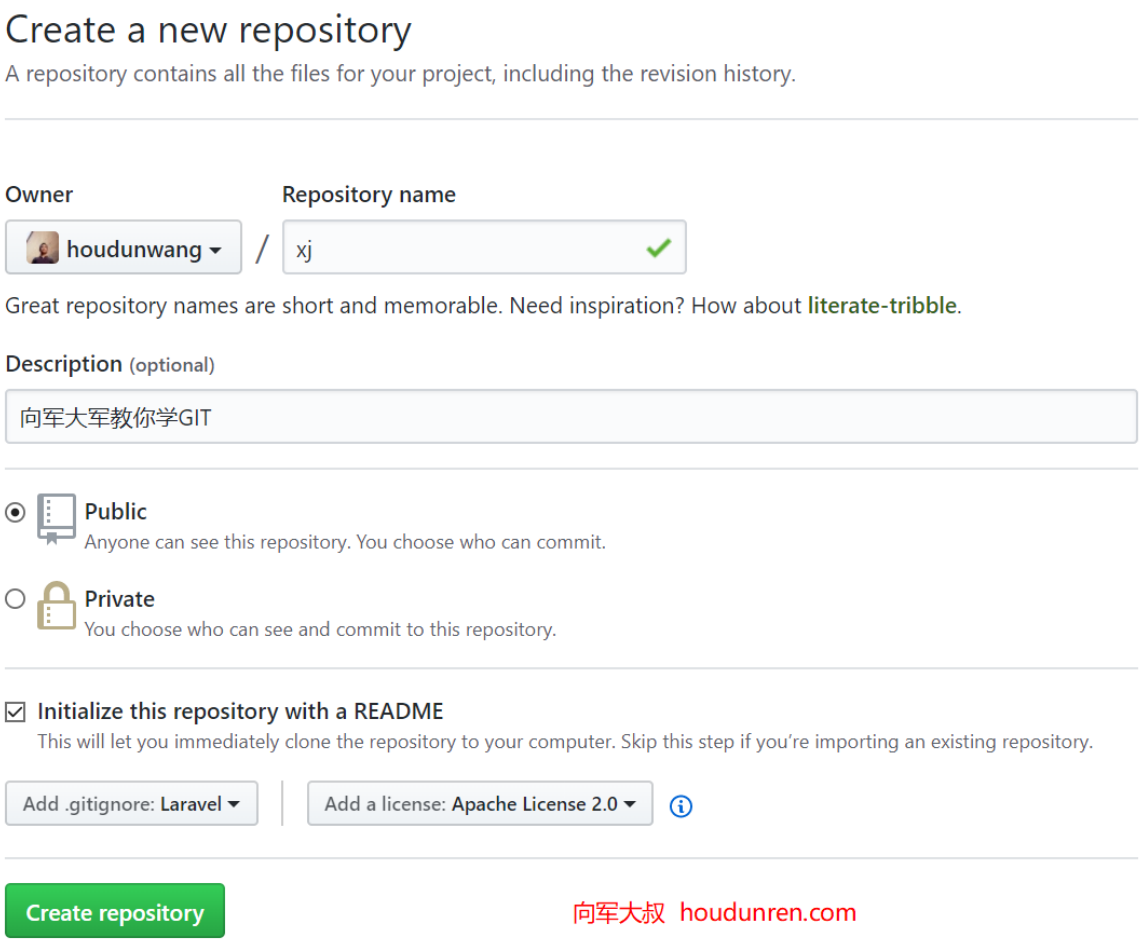
#远程仓库

下面是最热的`Github`进行讲解，使用`码云、codeing` 等国内仓库使用方式一致，就不在赘述了。

#创建仓库

为了完成以下示例，你需要在`GitHub` 创建好仓库。





#SSH

生成秘钥

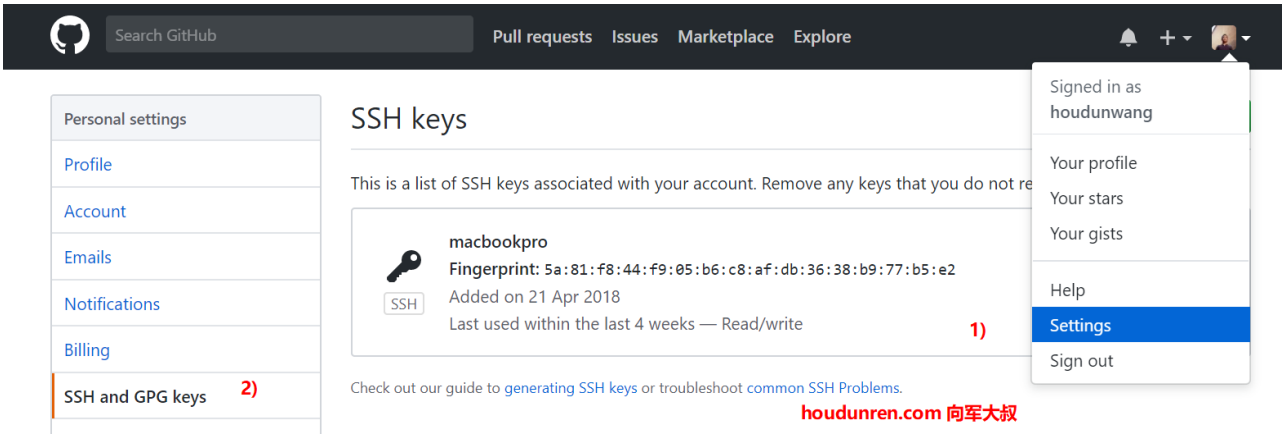
使用 ssh 连接 Github 发送指令更加安全可靠，也可以免掉每次输入密码的困扰。

在命令行中输入以下代码（windows 用户使用 Git Bash）

```
ssh-keygen -t rsa
```

一直按回车键直到结束。系统会在`~/.ssh` 目录中生成 `id_rsa`和`id_rsa.pub`，即密钥`id_rsa`和公钥`id_rsa.pub`。

向 GitHub 添加秘钥



点击 `New SSH key` 按钮，添加上面生成的 `id_rsa.pub` 公钥内容。

#关联远程

1. 创建本地库并完成初始提交

```
echo "# hd-xj" >> README.md
git init
git add README.md
git commit -m "first commit"

```

2. 添加远程仓库

```
git remote add origin git@github.com:houdunwang/hd-xj.git

```

3. 查看远程库

```
 git remote -v

```

4. 推送数据到远程仓库

```
git push -u origin master
```

5. 删除远程仓库关联

```
git remote rm origin
```

> 通过 clone 克隆的仓库，本地与远程已经自动关联，上面几步都可以省略。

#pull

拉取远程主机某个分支的更新，再与本地的指定分支合并。

1. 拉取 origin 主机的 ask 分支与本地的 master 分支合并

git pull origin ask:ask

2. 拉取 origin 主机的 ask 分支与当前分支合并

git pull origin ask

3. 如果远程分支与当前本地分支同名直接执行

git pull

#push

`git push`命令用于将本地分支的更新，推送到远程主机。它的格式与`git pull`命令相似。

1. 将当前分支推送到

`origin`主机的对应分支(如果当前分支只有一个追踪分支 ， 可省略主机名)

```
git push origin
```

2. 使用

`-u`选项指定一个默认主机 ,这样以后就可以不加任何参数直播使用`git push`。

```
$ git push -u origin master
```

3. 删除远程

`ask`分支 `git push origin --delete ask`

4. 本地 ask 分支关联远程分支并推送

`git push --set-upstream origin ask`

#多库提交

我可以将代码提交到多个远程版本库中，比如后盾人的 课程代码 (opens new window)就提交到了 Github 与 Gitee 两个库中。

```
# 增加一个远程库
git remote add github git@github.com:houdunwang/coding.git

# 提交到远程库
git push github

```

也可以在`~/.zshrc` 文件中定义别名，下面是定义的别名。这时使用 gp 将同时提供到 github 与 gitee

```
alias gp="git push & git push github"

```

#打包压缩

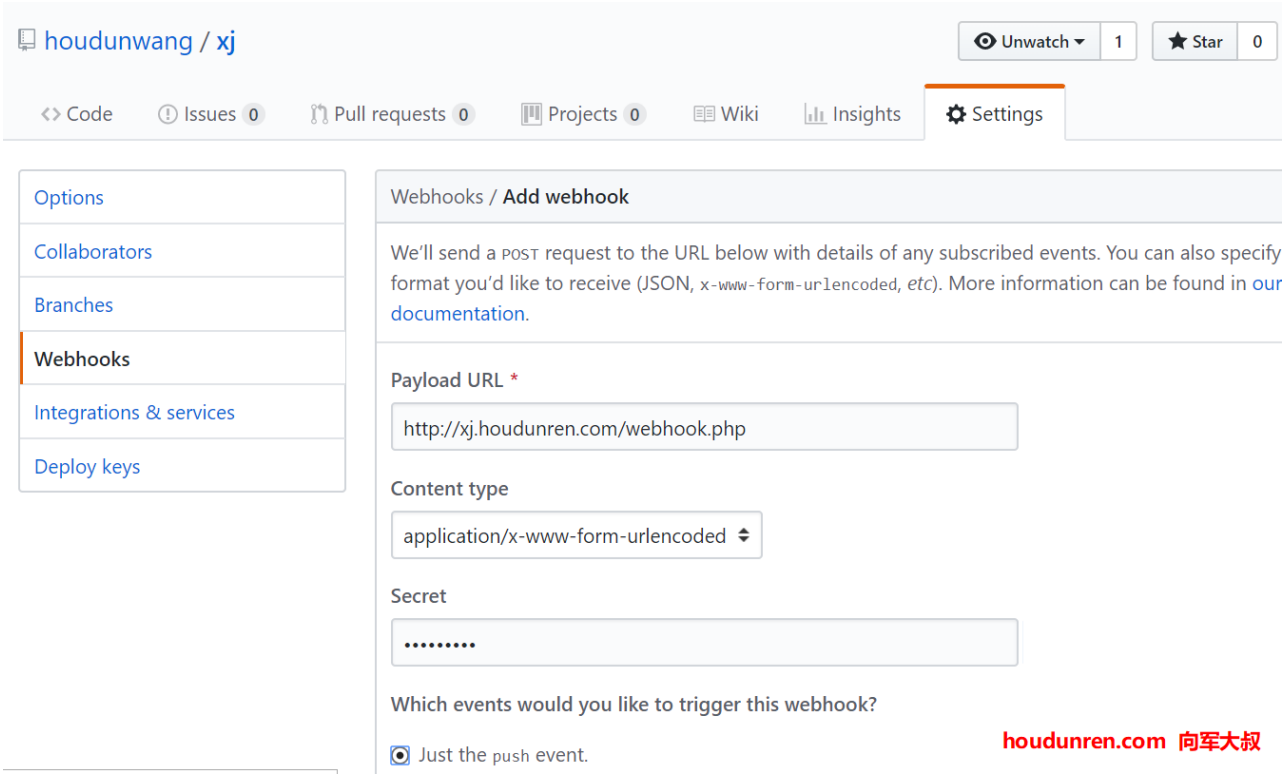
GIT 中可以使用 `git archive` 进行打包操作

将项目的 master 分支打包为 hdcms.zip

```
git archive --format zip --output hdcms.zip master

```

#自动部署

GitHub 设置 `WebHook`



#同步脚本

项目中添加处理 webhook 的 webhook.php 文件内容如下，并提交到版本库。

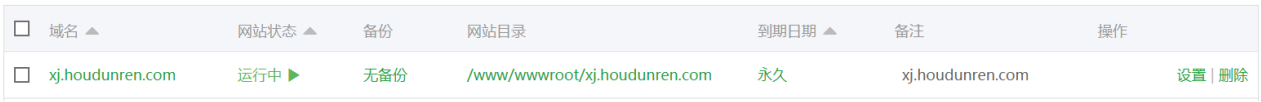
```php
<?php
// GitHub Webhook Secret.
// GitHub项目 Settings/Webhooks 中的 Secret
$secret = "houdunren";

// Path to your respostory on your server.
// e.g. "/var/www/respostory"
// 项目地址
$path = "/www/wwwroot/xj.houdunren.com";

// Headers deliveried from GitHub
$signature = $_SERVER['HTTP_X_HUB_SIGNATURE'];

if ($signature) {
  $hash = "sha1=".hash_hmac('sha1', file_get_contents("php://input"), $secret);
  if (strcmp($signature, $hash) == 0) {
    echo shell_exec("cd {$path} && /usr/bin/git reset --hard origin/master && /usr/bin/git clean -f && /usr/bin/git pull 2>&1");
    exit();
```

```
    }
}

http_response_code(404);
?>

```

#站点配置

#创建站点

下面示例我使用的是 `宝塔` 主机面板。



现在服务器上生成了站点目录 `/www/wwwroot/xj.houdunren.com` ，因为目录中存在 `.user.ini` 文件（定义站点可以访问的目录权限），造成不能 `clone` 代码，将目录随意改名。

#shell_exec

执行 `git pull` 指令需要使用 `shell_exec` 函数，删除 shell_exec 禁用函数后重启 PHP。

#clone

登录服务器并使用 https 协议 clone 项目代码

```
ssh root@xj.houdunren.com -p 22
git clone
https://github.com/houdunwang/xj.git
git clone

```

#修改权限

```
chown -R www .
chmod -R g+s .
sudo -u www git pull

```

现在向 GitHub 推送代码后，服务器将自动执行代码拉取，自动部署功能设置完成了。