

使用VuePress搭建技术文档

1. 根目录指的是存放克隆仓库文件的文件夹
2. 关闭提交时转换为LF，检出时转换为CRLF

```
$ git config --global core.autocrlf false
```

准备基础环境

1. 下载并安装git
2. 下载并安装node/yarn

vuepress初体验

1. 准备一个github仓库用来配置和存储vuepress技术性文档
 - 如果没有注册，就先注册，注册完成进行登录github账号
 - 新建一个仓库(repository)
2. 在电脑本地创建一个文件夹，拉取github上下载仓库的地址。在该文件夹目录下，鼠标右键 `Git Bash Here`，拷贝 github 上的仓库到本地 `git clone github下载仓库的链接`
 - 拉取仓库到本地，会出现如下报错：

```
1 fatal: unable to access 'https://github.com/hfllove/vuepress.git/': Recv failure: Connection was aborted
```

- 解决如下：

① 在git中执行 `git config --global --unset http.proxy` 和 `git config --global --unset https.proxy`

② 在cmd下执行 `ipconfig/flushdns` 清理DNS缓存

③ 重新执行 `git clone 链接` 即可

3. 进入从 github 仓库上拉取下来的文件夹内

```
1 cd github远程仓库的文件夹名称
```

4. 参考 [vuepress 官网](https://vuepress.vuejs.org/zh/guide/getting-started.html) <https://vuepress.vuejs.org/zh/guide/getting-started.html>

如果你想在现有项目中使用 VuePress 管理文档，从步骤 3 开始。

```
// 查看全局安装路径
npm root -g
// 查看npm的基础设置
npm config ls
// 查看安装目录路径
npm config get prefix
```

所以，根据官方的提示，进入 github 上拉取下来的文件夹内，按住 shift 加右键，以管理员的身份打开命令窗口，输入以下命令，将 vuepress 作为本地依赖进行安装

```
1 npm install -D vuepress
```

5. 在拉取下来的文件夹下，执行以下命令，初始化项目

这会在该文件夹下生成一个 package.json 的文件

```
1 npm init -y
```

6. 在拉取下来的文件夹下，创建 docs 文件夹，并且在它里面新建一个 md 文件，作为第一篇文档

```
1 mkdir docs && echo '# Hello VuePress' > docs/README.md
```

7. 在拉取下来的文件夹下的 package.json 中，添加 script 脚本

```
1 {
2   "scripts":
3     {
4       # 编写文档代码片段
5       "dev": "vuepress dev docs",
6       # 生成静态资源代码片段
7       "build": "vuepress build docs"
8     }
9 }
```

8. 在拉取下来的文件夹下，开始编写文档&&生成静态资源

```
1 # 编写文档
2 $ npm run dev
3 # 生成静态资源
4 $ npm run build
```

默认主题配置，目录结构，页面路由地址

参考vuepress官网：[默认主题配置](#) 以及 [目录结构](#)

1. 根据 package.json 文件的位置，可以确定项目的根目录位置
2. docs 一般就是 vuepress 文档的根目录，路由的路径为 `/`
3. 默认页面路由地址如下

文件的相对路径	页面路由地址
/README.md	/
/guide/README.md	/guide/
/config.md	/config.html

热更新配置

在 `package.json` 中将运行命令 由

```
1 "dev": "vuepress dev docs"
```

改为

```
1 "dev": "vuepress dev docs --temp .temp"
```

即可解决

届时运行 vuepress 会生成一个临时文件夹 `.temp`，可以在 `.gitignore`（如果没有，根目录下创建）中忽略掉该文件夹：

```
1 # vuepress temp file
2 .temp
```

页面结构

参考 vuepress 官网 [默认主题配置-首页](#)

1. 导航栏
2. 侧边栏
 - 侧边栏分组：多个路径对应一个侧边栏
 - 多个侧边栏：一个路径就对应一个侧边栏

SEO


参考vuepress官网的 [配置 -> 基本配置](#)，配置路径为 `.vuepress/config.js`

- title
- description
- author
- favicon

最后更新时间

参考 vuepress 官网 [默认主题-最后更新时间](#)

1. 最后更新时间插件，参考 [最后更新时间插件使用](#)

 要先在当前根目录下，使用 cmd 以管理员的身份[下载 moment](#) 插件


发布到github

如果有问题，或者出现无法访问最终部署到github上生成链接的错误，需要对[部署](#)这一步进行改正

1. 参考vuepress官网的 [指南 -> 部署 -> GitHub Pages](#)
 - `docs/.vuepress/config.js` 下，根据github的仓库地址，配置base路径，如果不是以 `用户名.github.io` 命名的仓库，则将 `base` 设置为 `"/<REPO>/"`

deploy.sh 文件配置

2. [配置根目录下](#) `deploy.sh` 文件

 配置 `deploy.sh` 时，如果不是以 `用户名.github.io` 命名的仓库，需要将发布地址写为：`https://<USERNAME>.github.io/<REPO>/`

当本地仓库与远程仓库连接时，本地分支默认为 **master**，而远程分支则为 自己选择的分支

`deploy.sh` 配置如下：

```
1  #!/usr/bin/env sh
2
3  # 确保脚本抛出遇到的错误
4  set -e
5
6  # 生成静态文件
7  npm run build
8
9  # 进入生成的文件夹
10 cd docs/.vuepress/dist
11
12 # 如果是发布到自定义域名
13 # echo 'www.example.com' > CNAME
14
15 git init
16 git add -A
17 git commit -m 'deploy'
18
19 # 如果发布到 https://<USERNAME>.github.io
20 # git push -f git@github.com:<USERNAME>:<USERNAME>.github.io.git master
21
22 # 如果发布到 https://<USERNAME>.github.io/<REPO>
23 git push -f git@github.com:hfllove/myblog.git master:这里填写远程分支
24
25 cd -
```

3. 配置 `package.json` 文件中，提交文件到仓库的命令符

```
1  {
2    "scripts": {
3      "deploy": "bash deploy.sh"
4    }
5  }
```

4. 将本地文件提交到github上

```
1 npm run deploy
```

5. 在接收本地提交的github仓库中，点击上面导航栏右边的settings，点击侧边栏的Pages，可以看到github自动生成的访问地址，点进去可以直接访问

自定义域名

先购买一个域名，再添加一个CNAME方式的前缀自定义域名，value值为自定义域名的连接github地址

1. 在项目根目录下的deploy.sh 中，设置自定义域名访问方式

```
1 # 如果是发布到自定义域名
2 echo 'temp2.hfllog.space' > CNAME
```

2. 然后，在 部署项目的github仓库的 settings -> Pages -> Custom domain 下，添加在域名解析处的自定义域名
3. 根目录下的 docs/.vuepress/config.js 中，注释掉基础路径，或者设置基础路径为/，即可通过自定义域名访问搭建好的vuepress文档

启用PWA

1. 参考 vuepress官网 -> 插件 -> 官方插件 -> pwa ，安装pwa 插件

```
1 npm install -D @vuepress/plugin-pwa
```

2. 在根目录下的 docs/.vuepress/config.js 中，配置 SW-Update Popup

```
1 # 使用插件的选项 Babel式
2 plugins: [
3   '@vuepress/pwa',
4   {
5     serviceWorker: true,
6     updatePopup: {
7       message: "New content is available.",
8       buttonText: "Refresh"
9     }
10  ]
11 ]
```

3. 为了让网站完全兼容PWA，需要

- 在 `.vuepress/public` 提供 Manifest 和 icons

参考链接 <https://realfavicongenerator.net/> 和 <https://manifest-gen.netlify.app/>

- 搜索 `manifest icons generator`，在参考链接的网站中，修改 App Name、Short Name、Display Mode，然后在右侧上传图片，上传完成，点击 SUBMIT。会得到一个 app-image.zip，解压之后，会得到 images 文件夹 和 manifest.json 文件。将 images 文件夹中的 icons 文件夹以及 manifest.json 文件，放到 `docs/.vuepress/public` 文件夹下。
- 在 `.vuepress/config.js` 添加正确的 head links(参见下面例子)。

这是一个在VuePress中完全地兼容 PWA 的例子：

```
1 head: [  
2     ['link', { rel: 'icon', href: '/favicon.ico' }],  
3     ['link', { rel: 'manifest', href: '/manifest.json' }],  
4     ['meta', { name: 'theme-color', content: '#3eaf7c' }],  
5     ['meta', { name: 'apple-mobile-web-app-capable', content: 'yes' }],  
6     ['meta', { name: 'apple-mobile-web-app-status-bar-style', content:  
    'black' }],  
7     ['link', { rel: 'apple-touch-icon', href: '/icons/apple-touch-icon-  
    152x152.png' }],  
8     ['link', { rel: 'mask-icon', href: '/icons/safari-pinned-tab.svg',  
    color: '#3eaf7c' }],  
9     ['meta', { name: 'msapplication-TileImage', content:  
    '/icons/msapplication-icon-144x144.png' }],  
10    ['meta', { name: 'msapplication-TileColor', content: '#000000' }]  
11    ],
```

4. 提交 .vuepress/public 新增的 Manifest 和 icons 到github上

```
1 # 检查git提交文件的状态  
2 git status  
3 # 获取所有新增的文件  
4 git add .  
5 # 提交到github仓库  
6 git commit -m "feat:pwa"
```

安装回到顶部插件

- 在根目录下运行以下命令，安装back-to-top插件

```
1 npm i @vuepress-reco/vuepress-plugin-back-to-top -D
```

2. 配置 .vuepress/config.js，使用插件

```
1 module.exports = {plugins: ['@vuepress-reco/vuepress-plugin-back-to-top']}
```


分割config

1. 在 docs/.vuepress 路径下新建一个 config 文件夹，将 docs/.vupress/config.js 中的 head、nav、sidebar、plugins 配置，以 js 文件的形式放到 config 文件夹中，并且对外暴露，实例演示如下

```
1 // 在 docs/.vuepress/config 新建并配置 headConfig.js
2 module.exports = [
3   ['link', { rel: 'icon', href: '/favicon.ico' }],
4   ....
5 ]
6
7 // docs/.vuepress/config.js 中
8 const headConfig = require('./config/headConfig')
9 module.exports = {
10   head: headConfig,
11 }
```

Git Action 自动部署

1. 在 github 的 settings 中选择左侧的 Pages，将其中的 source 修改为 Git Action，然后，在顶部 Actions 中，添加新的 workflows，进行如下配置：

 注意：

1. 需要在 settings -> Actions -> general 下的 Workflow permissions 中，给 read 和 write 权限

2. 如果需要退出 workflows，则需要点击顶部菜单栏中 Actions，然后选择左侧的Deploy Github Pages，接着选择右侧的 ... 菜单，选择其中的 Disable workflow，即可停止 workflows
3. 如果在 Action 自动部署时，报以下错误，请删除 docs/.vuepress 中的 dist 文件夹，然后重新 push 到远程分支。再次进行 git Action 部署

```
1 # build
2 No url found for submodule path 'docs/.vuepress/dist' in .gitmodules
3 # build
4 The process '/usr/bin/git' failed with exit code 128
```

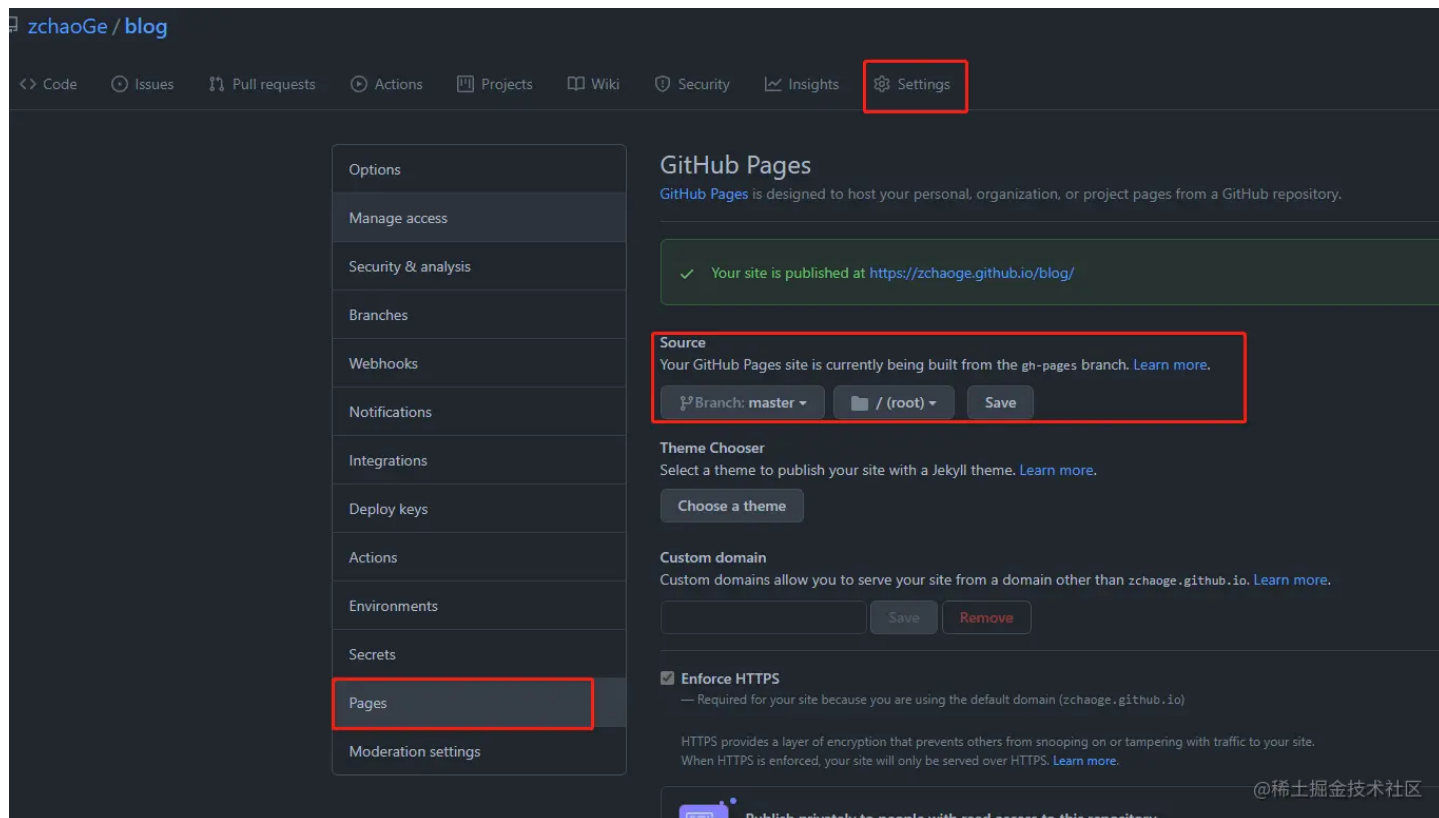
Git Action 部署详细参照

前言

首先要准备好你的VuePress项目，才能说**部署**是吧。其实手动部署也不是不可以，只是你推送了代码之后，到了GitHub Pages那边你还得再操作一遍。所以还是搞个**自动部署GitHub Pages**方便一点

开启Github Pages

先开启Github Pages，先放着。

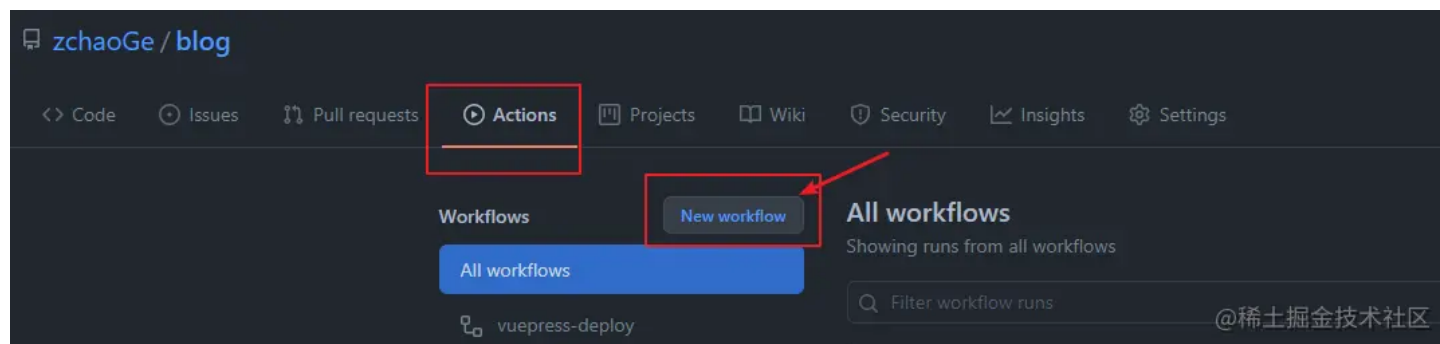


这里默认是 主分支(master)，保存之后肯定是行不通的，得把后面的步骤做完！

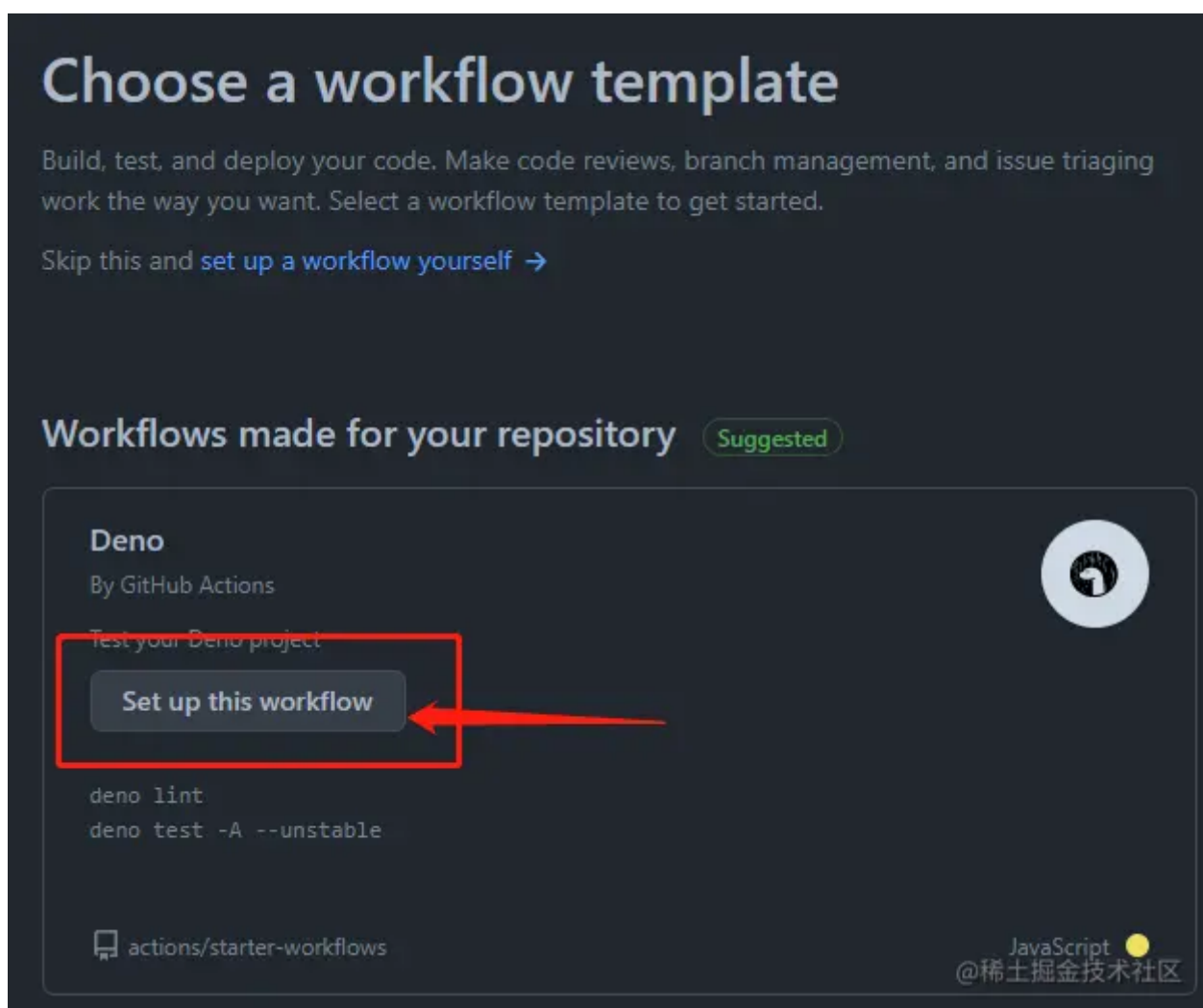
步骤

一. 创建Actions

1、在博客仓库的Actions选择 `New workflow`



2、选择 `Deno` 第一个就行



3、可以自定义名字

4、文件中写入以下代码

```
1 name: vuepress-deploy # 这里的名字就是你刚刚自定义那个文件的名字
2 on:
3   push:
4     branches:
```

```

5     - main
6 jobs:
7   build-and-deploy:
8     runs-on: ubuntu-latest
9     strategy:
10      matrix:
11        node: ['lts/fermium']
12      steps:
13        - name: Checkout
14          uses: actions/checkout@main
15          with:
16            ref: 'main'
17            persist-credentials: false
18            fetch-depth: 0
19          env:
20            TZ: Asia/Shanghai
21
22        - name: Use Node.js ${ matrix.node-version }
23          uses: actions/setup-node@main
24          with:
25            node-version: ${ matrix.node }
26
27        - name: Install dependencies
28          run: npm ci
29
30        - name: Build VuePress
31          run: npm run build
32        - name: Deploy to Pages
33          env:
34            TZ: Asia/Shanghai
35          run: |
36            cd docs/.vuepress/dist
37            git config --global init.defaultBranch main
38            echo 'myblog.hfllog.space' > CNAME
39            git init
40            git config user.name "hfllove"
41            git config user.email "${ secrets.GIT_EMAIL }"
42            git add .
43            git commit -m "Deploying to gh-pages from @ $GITHUB_SHA in $(date +"%Y-%m-%d %H:%M:%S")"
44            git push -f https://hfllove:${ secrets.ACTIONS_DEPLOY_KEY }@github.co

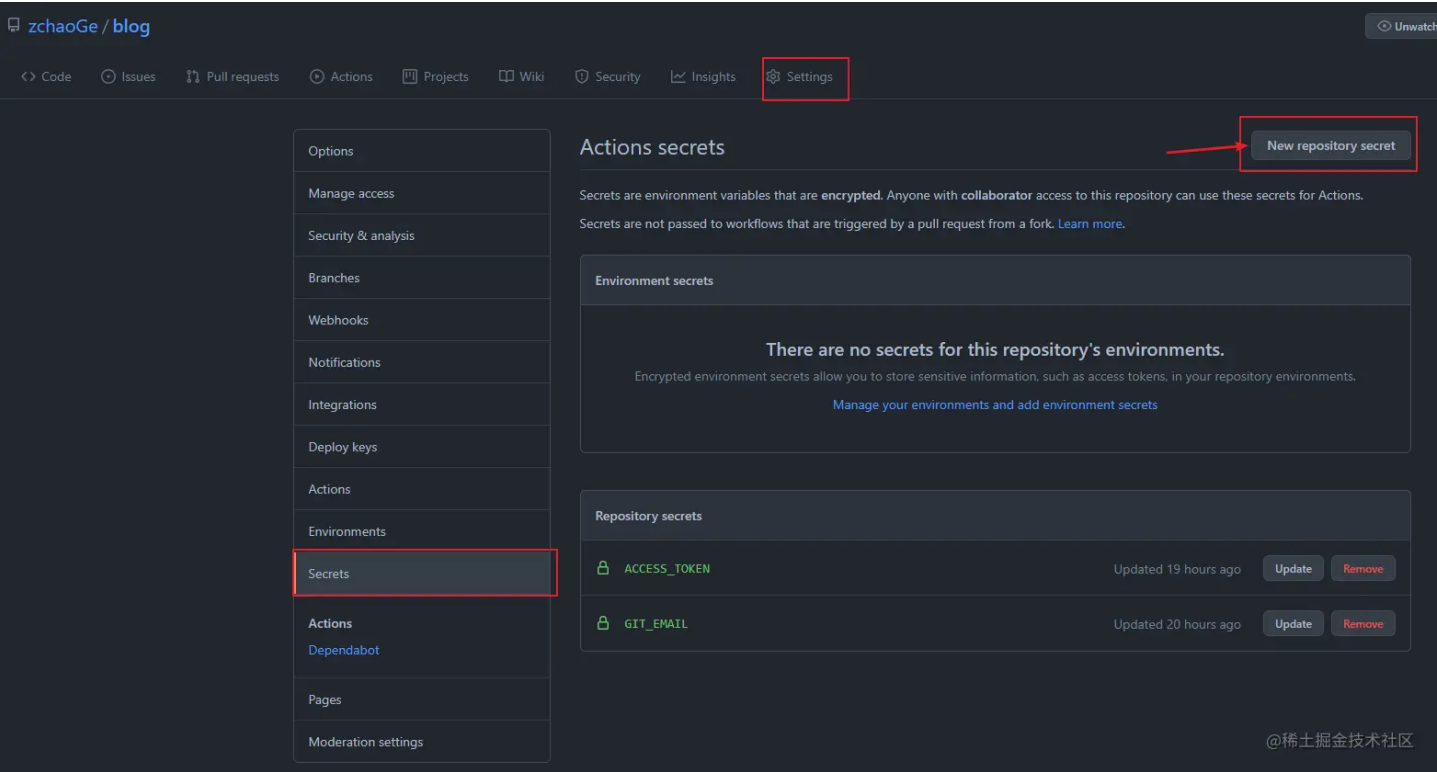
```

这里我们需要注意一下后面git操作的部分

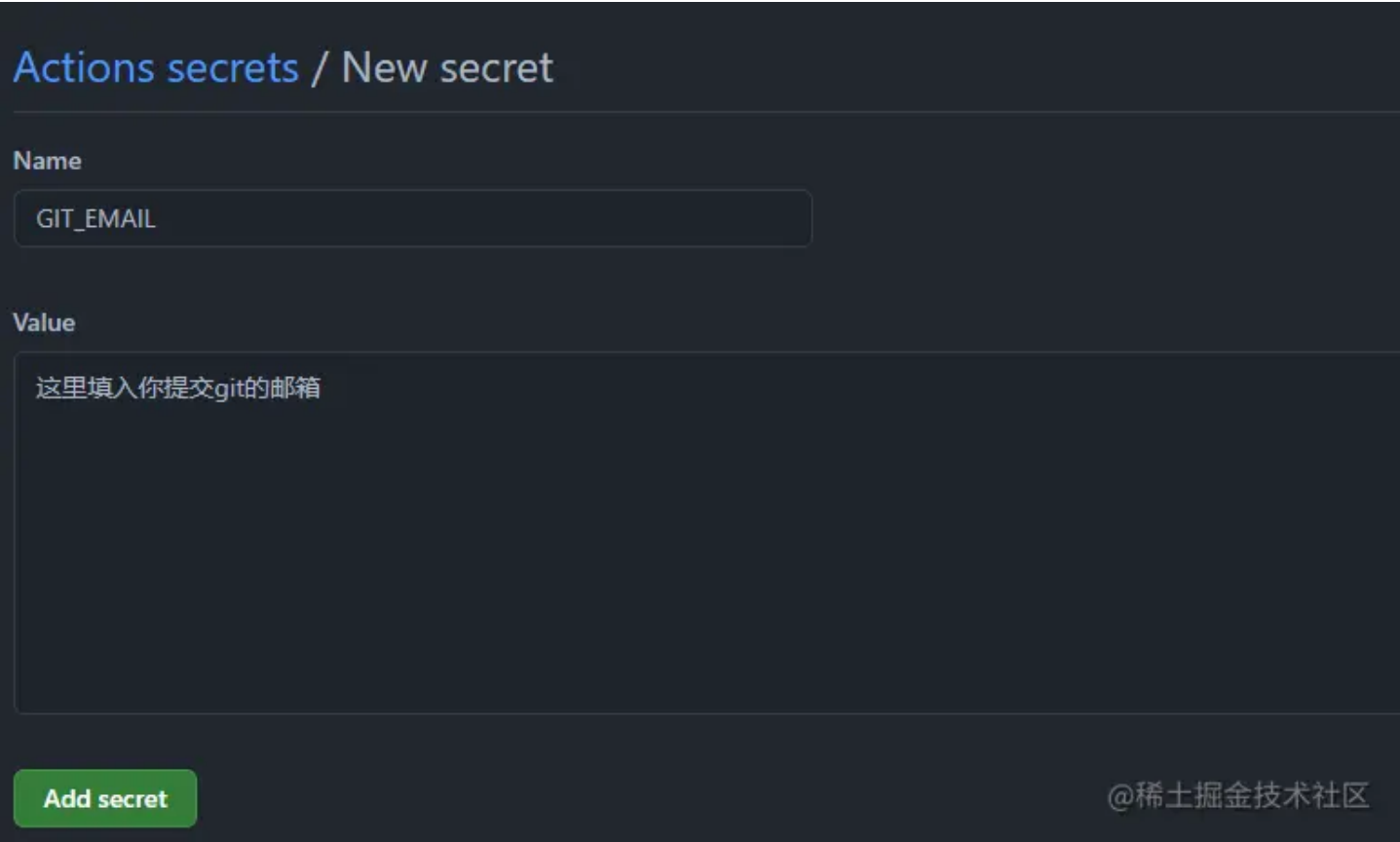
其中使用到了 `secrets.GIT_EMAIL` 和 `secrets.ACCESS_TOKEN`。接下来我们去创建这两个东西

二. 创建GIT_EMAIL和ACCESS_TOKEN

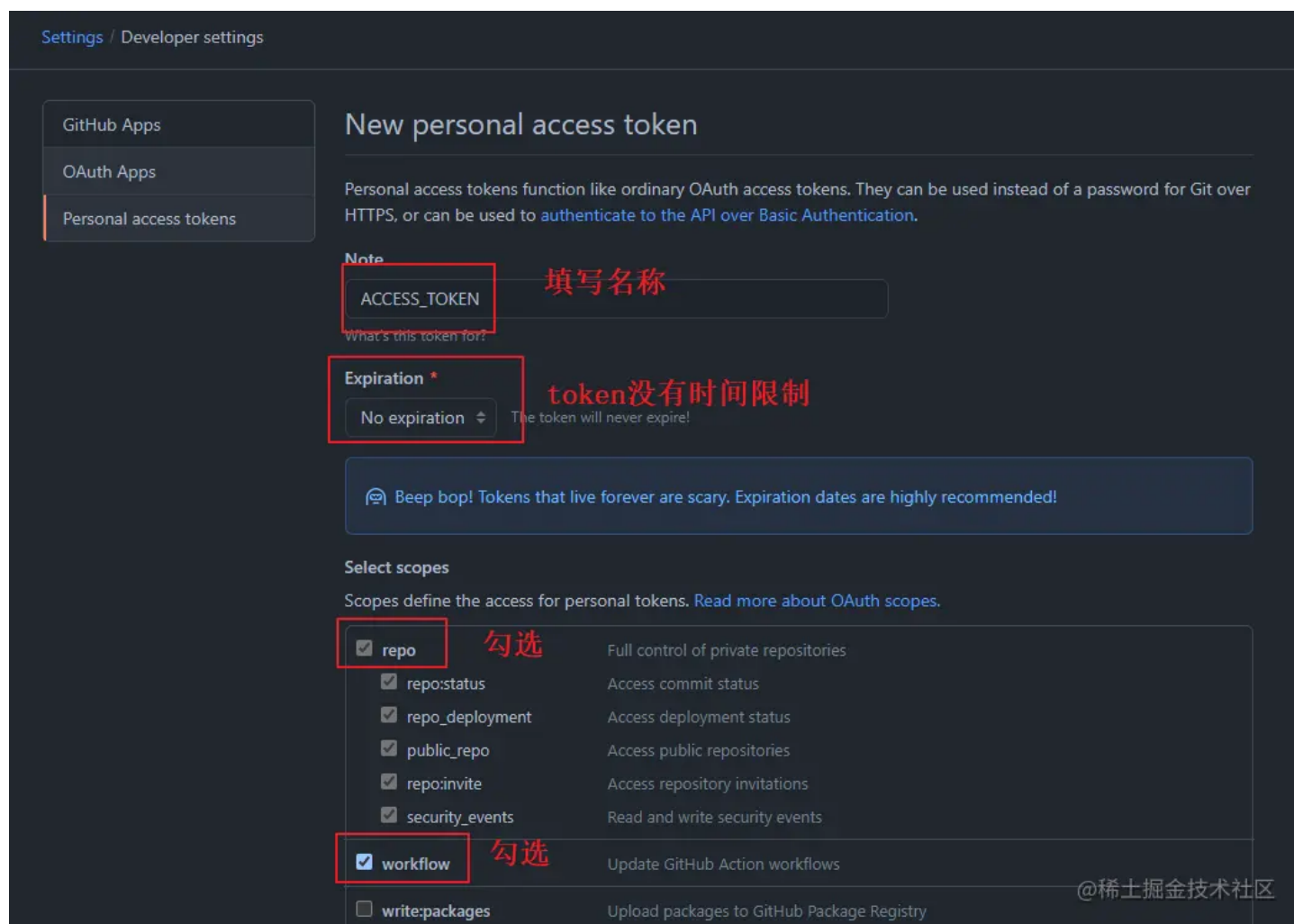
在仓库中选择Setting，然后选择Serect，然后新建



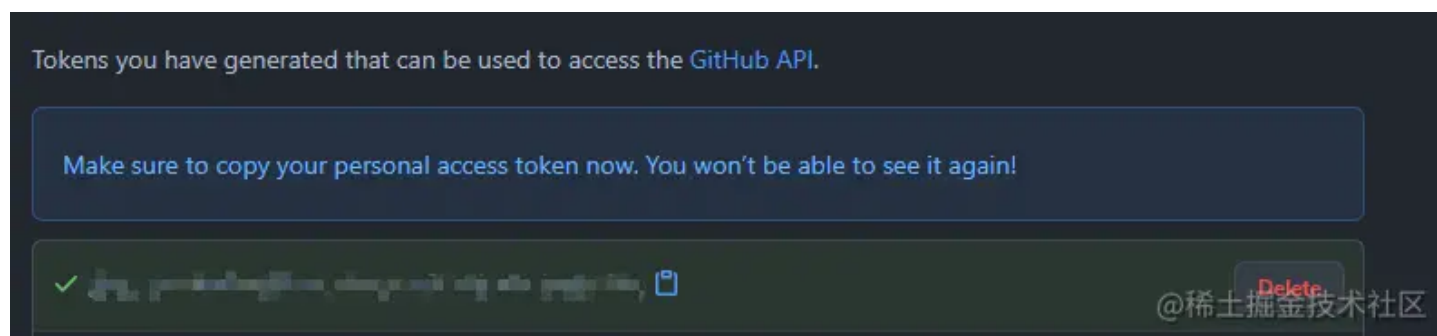
Name填写 GIT_EMAIL （当然名字可以自定义），value填入你提交git的 邮箱 ，然后添加



这里我们先去生成一个token，然后再倒回去创建Serect
创建一个有 repo 权限的 [GitHub Token](#)



这里我们 填完名称 和 勾选上repo和workflow 选项之后呢，然后直接 点击 Generate token 按钮，即可生成一个token，如下图：



注意：

那个英文的意思是：

确保立即复制您的个人访问令牌。你将无法再看到它！那万一没了**重新生成**一个就行。

到这我们token已经生成完毕，回去新建Secret。填入token的时候注意不要填多空格，不然后面部署会报错的。

Actions secrets / New secret

Name

ACCESS_TOKEN

Value

填入刚刚生成的token

Add secret

@稀土掘金技术社区

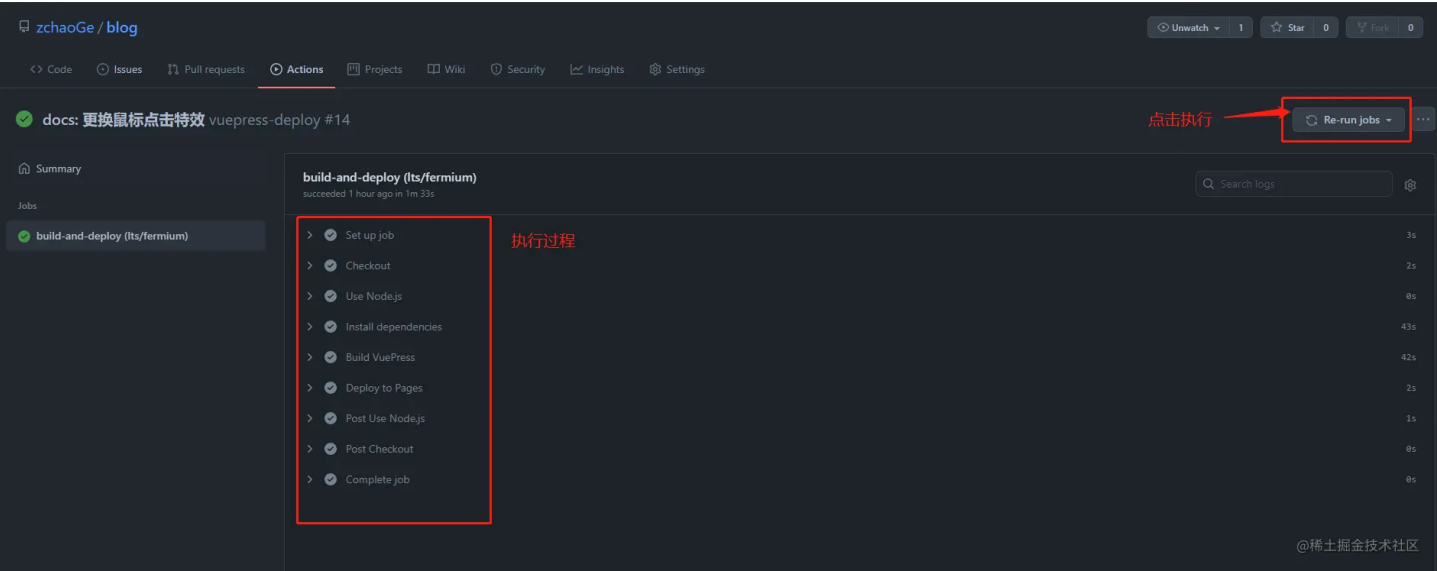
修改配置文件

这里修改一下配置文件 `.vuepress/config.js`，新增 `base` 配置

```
1 module.exports = {
2   // base: '/', // 格式: '/<仓库名>', 默认 '/'
3   base: '/blog/',
4
5
6   theme: 'vdoing', // 使用npm包主题
7   // theme: require.resolve('../..../theme-vdoing'), // 使用本地主题
8
9   title: "标题",
10  description: '描述',
11  markdown: {
12    lineNumbers: true, // 代码行号
13  },
14
15  head,
16  plugins,
17  themeConfig,
18 }
19 复制代码
```

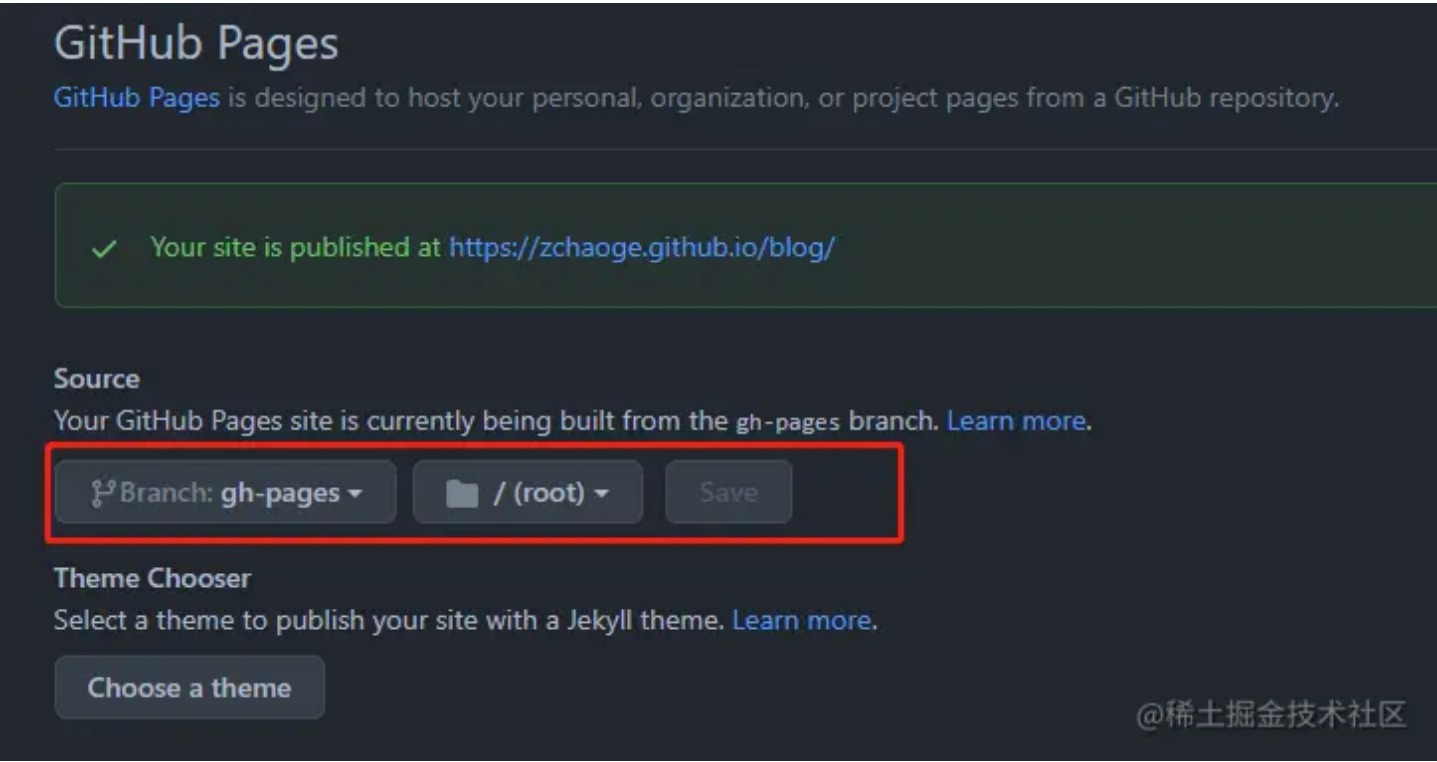
执行Actions

执行前，一定要确保以上步骤没有遗漏。还有要保证仓库是最新的代码，因为你修改过 `config.js` 配置文件了。



如果执行过程中报错了，先排错，再重新执行。成功就是绿色的勾勾。

最后一步：更改Github Pages的分支为 `gh-pages`



访问Github Pages，成功！

以后我们只需推送代码到Github仓库就可以了，Actions会自动执行，Github Pages也会更新到最新。

安装插件

为了更好的使用 vuepress ，需要[额外](#)安装一些适合的插件

图片路径与图片缩放插件

1. 参考 vuepress 官方文档 [指南 -> 静态资源](#)

图片的路径大致有两种写法

- 相对路径
- 基础路径

2. 图片缩放插件的安装与使用，参考 [vuepress 官方插件地址](#)
-

Vuepress tab 选项卡插件

Vuepress tab 选项卡插件 官方 地址：[extractCode](#)

代码一键复制插件

代码一键复制插件- [github 地址](#)

Vuepress 站点平滑滚动

[平滑滚动插件](#)

Vuepress 阅读进度条

[vuepress 阅读进度条 - github 地址](#)

vuepress 自动生成侧边栏

[自动生成侧边栏- github 地址](#)

Vuepress 图片懒加载

[vuepress 图片懒加载 - github 地址](#)

Git 部分命令

git分支

1. 查看本地与远程分支关联情况

```
1 git branch -vv
```

2. 远程有分支的情况下，

- 本地新建一个与远程同名的分支

```
1 git checkout -b 远程分支名
```

- 本地分支关联远程分支

```
1 git branch --set-upstream-to=origin/branch1 branch2 (前面branch为远程分支名，后面branch2为本地分支名)
```

3. 提交本地test分支到远程的master分支

```
1 git push <远程主机名> <本地分支名>:<远程分支名>
2
3 示例: git push origin test:master
```

4. 删除本地分支

```
1 $ git branch -D 本地分支的名字
```

5. 删除远程分支

- 1 `git push origin :master` (推送一个空分支到远程分支, 其实就相当于删除远程分支)
- 2 或
- 3 `git push origin --delete xxx`