

git的使用

程序员在日常工作当中经常会将自己的代码或项目进行修改及开发，这个时候就可能会面向以下的几个场景

1. 今天的代码是在昨天的代码上面修改的，我如果同时保存今天的代码和昨天的代码，或更久以前的代码

<input type="checkbox"/> 名称	修改日期	类型	大小
 movie-app.txt	2022/6/17 14:34	文本文档	0 KB
 movie-app1.txt	2022/6/17 14:34	文本文档	0 KB
 movie-app2.txt	2022/6/17 14:34	文本文档	0 KB
 movie-app2改.txt	2022/6/17 14:34	文本文档	0 KB
 movie-app2改_张三.txt	2022/6/17 14:34	文本文档	0 KB

上面的效果大家应该很熟悉，如果要同时保存这几天的不同的版本结果，需要使用不同的文本去存储

2. 如果多人同时进行开发的时候，如果将代码进行交换，共享

为了解决上面的问题，程序就推出一个版本控制思路，它们会反当前的文档或项目以版本迭代的形式来管理

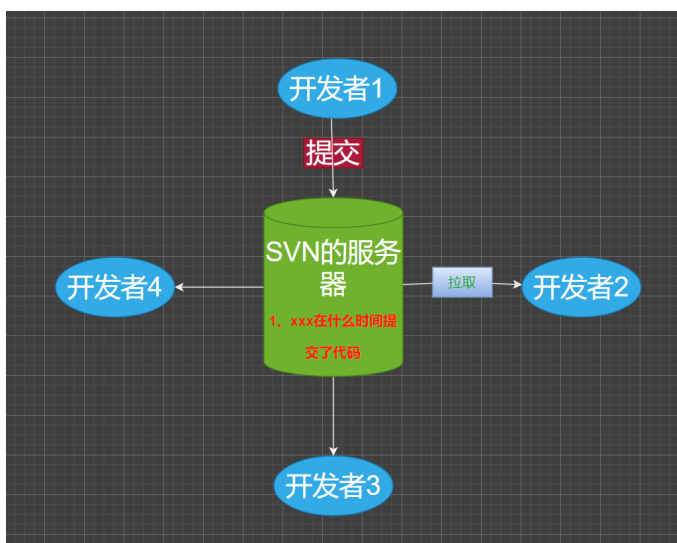
如 `movie-appV1/movie-appv1.1`

上面它只是我们产品的版本，并不是我们代码的版本，所以我们希望我们每次的代码的修改都通用记录可查，并且能够保存所有的历史版本，这时候代码的版本控制器就应运而生

目前现行的主流的版本控制器

1. `SVN`

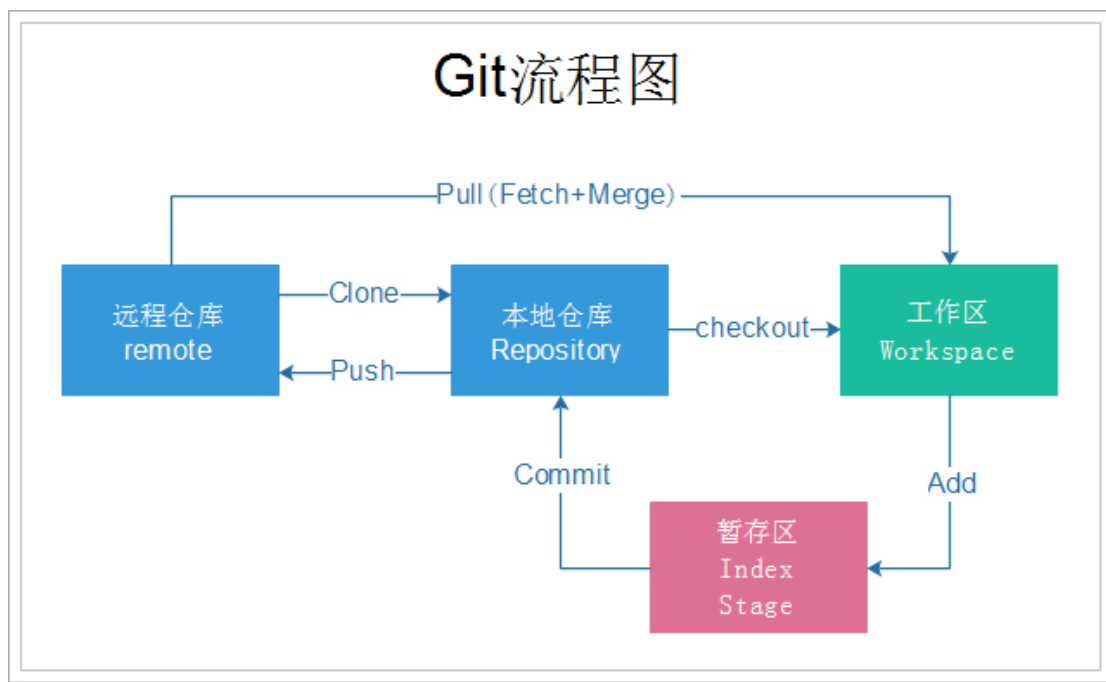
SVN是subversion的缩写，是一个**开放源代码**的版本控制系统，通过采用分支管理系统的高效管理，简而言之就是用于多个人共同开发同一个项目，实现共享资源，实现最终**集中式的管理**。



这种形式最大的特点就是操作简单，服务器搭建也方便，但有一个最大的缺点，所有的开发者都集中在这一个服务器上面，如果服务器发生了故障，那么所有的开发都都会受影响

2. `git`

Git (读音为/git/) 是一个开源的**分布式版本控制系统**，可以有效、高速地处理从很小到非常大的项目版本管理。[1] 也是 **Linus Torvalds** 为了帮助管理Linux内核开发而开发的一个开放源码的版本控制软件。

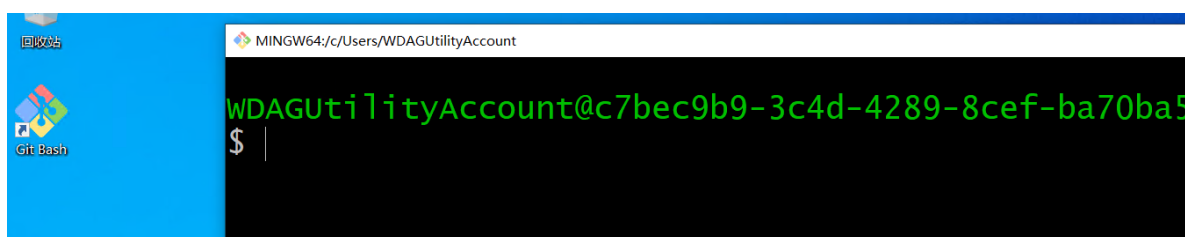


git与svn是不一样的，它没有服务器的概念，它只有远程仓库与本地仓库的概念

3. CVS

CVS是一个C/S系统，是一个常用的代码版本控制软件。主要在**开源软件**管理中使用。与它相类似的代码**版本控制软件**有 **subversion**。多个开发人员通过一个中心版本控制系统来记录**文件版本**，从而达到保证文件同步的目的。CVS版本控制系统是一种GNU软件包，主要用于在多人**开发环境**下的源码的维护。但是由于之前CVS编码的问题，大多数软件开发公司都使用 **SVN** 替代了CVS。

git安装



安装成功了以后，会在桌面上面显示当前图标

git全局配置

```
1 $ git config --global user.name "杨标"
2 $ git config --global user.email "lovesnsfi@163.com"
```

git的基本操作

要弄清楚git并且要弄清楚三个区域

1. **workspace** 工作区
2. 暂存区
3. **Repository** 本地仓库

如果想得到上面的三个区域，首先就需要进行初始化

初始化

在某一个需要进行版本控制的文件夹下，输入如下命令

```
1 $ git init
```

```
MINGW64/c:/Users/YangBiao/Desktop/aaa
YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa
$ git init
Initialized empty Git repository in C:/Users/YangBiao/Desktop/aaa/.git/
```

当出现上面的结果以后，就代表我们的仓库初始化成功了，这个时候这个目录下面就会多一个.git的文件夹【这个文件夹默认是隐藏的，你们看不到】

名称	修改日期	类型	大小
.git	2022/6/17 15:08	文件夹	
1.txt	2022/6/17 15:03	文本文档	1 KB

版本控制生成的文件夹

在上面的这个 .git 的文件夹里面就包含了3个概念【工作区/暂存区/本地仓库】

工作区

目前你能够看得见的就是工作区，工作区的东西，默认是不能够形成版本控制的，只有本地仓库的内容才会形成版本记录，现在我们就必须把工作区内容放到本地仓库 `local repository`。关键问题是不能直接放进去，要经过暂存区

暂存区

工作区里面有些东西可能 是要放到本地仓库里面去了，就先放到暂存区，放在暂存区的东西可以还可以修改，还可以撤销等进行一系列操作

将文件添加到暂存区，可以使用下面的方式

```
1 $ git status -v
```

```
YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa (master)
$ git status -v
On branch master
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  1.txt
nothing added to commit but untracked files present (use "git add" to track)
```

这个时候就会发现1.txt这个文件没有被添加到暂存区

```
YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa (master)
$ git add 1.txt
```

现在我们使用了 `git add 1.txt` 将这个文件添加到了暂存区

思考：如何添加多个文件

我们现在在工作区新建 `2.txt` 和 `3.doc` 的文件

```

YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa (master)
$ git status -v
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   1.txt
  
```

想删除暂存区的文件，可以使用这个命令

暂存区现在有一个文件

```

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    2.txt
    3.doc
  
```

还有这2个文件不在暂存区，询问是否需要添加

```

diff --git a/1.txt b/1.txt
  
```

```
1 $ git add .
```

```

MINGW64/c/Users/YangBiao/Desktop/aaa

YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa (master)
$ git add .  将当前目录下面所有的文件都添加到暂存区

YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa (master)
$ git status -v
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   1.txt
    new file:   2.txt
    new file:   3.doc
  
```

暂存区现在有3个文件了

本地仓库

当我们文件放入暂存区以后，就可以提交到本地仓库，请注意，每一次的提交都会形成一个记录

```
1 git commit -m "这里书写备注信息"
```

```

MINGW64/c/Users/YangBiao/Desktop/aaa

YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa (master)
$ git commit -m "第一次提交"
[master (root-commit) 913b64c] 第一次提交
 3 files changed, 2 insertions(+)
 create mode 100644 1.txt
 create mode 100644 2.txt
 create mode 100644 3.doc

YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa (master)
$
  
```

当出现上面的代码，就做对我们的版本已经提交到仓库，并形成了一记录，如果想查看仓库的提交记录，可以使用

```
1 $ git log
```

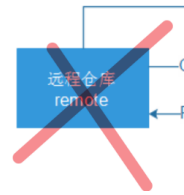
```

YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa (master)
$ git log
commit 913b64c0f0c38e1dba67d3bb6f7f94662a030224 (HEAD -> master)
Author: 二当家的 <lovesnsfi@163.com>
Date:   Fri Jun 17 15:37:18 2022 +0800    查看提交记录

    第一次提交

YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa (master)
$ git status -v
On branch master    提交以后，再次去查询状态，发生暂存区没有记录了
nothing to commit, working tree clean

```



文件修改以后

当我们在三个文件上面修改以后，再次查询状态

```

YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa (master)
$ git status -v
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working)

       modified:   1.txt
       modified:   2.txt
       modified:   3.doc

```

如果这次定稿以后，又要提交，则过程还是一样的

```

1  $ git add .
2  $ git commit -m "第二次修改"

```

版本回退

当发现自己的代码写错了，或项目当中需要回退到某一个版本，则可以使用下面的使用

```

1  $ git log

```

先查看所有的提交记录

```

YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa (master)
$ git log
commit b69c30af1b18207cbf85c3e3aa2af8783cb90ec7 (HEAD -> master)
Author: 二当家的 <lovesnsfi@163.com>
Date:   Fri Jun 17 15:42:18 2022 +0800

    第二次修改提交

commit 913b64c0f0c38e1dba67d3bb6f7f94662a030224
Author: 二当家的 <lovesnsfi@163.com>
Date:   Fri Jun 17 15:37:18 2022 +0800

    第一次提交

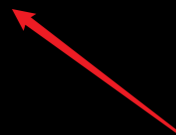
```

代表当前所处的版本
也就是工作区的版本

如果我们想回退到第一次提交，怎么办呢

第二次修改提交

```
commit 913b64c0f0c38e1dba67d3bb6f7f94662a030224
Author: 二当家的 <lovesnsfi@163.com>
Date:   Fri Jun 17 15:37:18 2022 +0800
```



第一次提交

```
YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa (master) 找到某一次提交的值
$
```

```
1 $ git checkout 913b64c0f0c38e1dba67d3bb6f7f94662a030224
```

```
YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa (master)
$ git checkout 913b64c0f0c38e1dba67d3bb6f7f94662a030224
Note: checking out '913b64c0f0c38e1dba67d3bb6f7f94662a030224'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.
```

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -b with the checkout command again. Example:

```
git checkout -b <new-branch-name>
```

```
HEAD is now at 913b64c 第一次提交
```

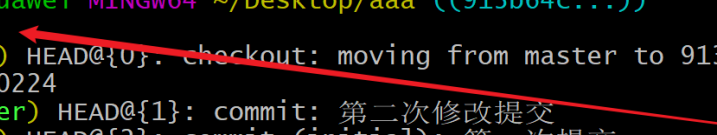
当出现这条记录就代表回退到了某一个版本

```
YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa ((913b64c...))
```

想一想：现在已经由第2个版本回退到了第1个版本，是否能够再回到第2个

```
1 $ git reflog
```

```
YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa ((913b64c...))
$ git reflog
913b64c (HEAD) HEAD@{0}: checkout: moving from master to 913b64c0f0c38e1dba6
b6f7f94662a030224
b69c30a (master) HEAD@{1}: commit: 第二次修改提交
913b64c (HEAD) HEAD@{2}: commit (initial): 第一次提交
```



可以查到所有的记录（包括回退的记录）

```
1 git checkout b69c30a
```

远程仓库

我们的远程仓库使用的是gitee

新建仓库

在其他网站已经有仓库了吗? [点击导入](#)

仓库名称 * 

aaa

归属



二当家的

路径 * 

/ aaa

仓库地址: https://gitee.com/lovesnsfi_admin/aaa

仓库介绍

13/200

这是华夏学院上课测试的仓库

- ☐ 开源 (所有人可见) 
- ☒ 私有 (仅仓库成员可见)
- ☐ 企业内部开源 (仅企业成员可见) 

- ☐ 初始化仓库 (设置语言、.gitignore、开源许可证)
- ☐ 设置模板 (添加 README、Issue、Pull Request 模板文件)
- ☐ 选择分支模型 (仓库创建后将根据所选模型创建分支)

创建

快速设置—如果你知道该怎么操作, 直接使用下面的地址

HTTPS

SSH

https://gitee.com/lovesnsfi_admin/aaa.git



我们强烈建议所有的git仓库都有一个 `README`, `LICENSE`, `.gitignore` 文件

[初始化 readme 文件](#)

Git入门? 查看 [帮助](#), [Visual Studio / TortoiseGit / Eclipse / Xcode 下如何连接本站](#), [如何导入仓库](#)

简易的命令行入门教程:

Git 全局设置:

```
git config --global user.name "二当家的"
git config --global user.email "lovesnsfi@163.com"
```

创建 git 仓库:

```
mkdir aaa
cd aaa
git init
touch README.md
git add README.md
git commit -m "first commit"
git remote add origin https://gitee.com/lovesnsfi_admin/aaa.git
git push -u origin "master"
```

已有仓库?

```
cd existing_git_repo
git remote add origin https://gitee.com/lovesnsfi_admin/aaa.git
git push -u origin "master"
```

创建成功以后, 会有上面的界面

快速说直一 如果你知道该怎么操作, 且使用下面的地址

HTTPS SSH https://gitee.com/lovesnsfi_admin/aaa.git

我们强烈建议所有的git仓库都有一个 README, LICENSE, .gitignore 文件

[初始化 readme 文件](#)

Git入门? 查看 帮助, Visual Studio / TortoiseGit / Eclipse / Xcode 下如何连接本站 如何导入仓库

简易的命令行入门教程:

Git 全局设置:

```
git config --global user.name "二当家的"
git config --global user.email "lovesnsfi@163.com"
```

创建 git 仓库:

```
mkdir aaa
cd aaa
git init
touch README.md
git add README.md
git commit -m "first commit"
git remote add origin https://gitee.com/lovesnsfi_admin/aaa.git
git push -u origin "master"
```

已有仓库?

```
cd existing_git_repo
git remote add origin https://gitee.com/lovesnsfi_admin/aaa.git
git push -u origin "master"
```

我们之前已经git init 初始化完成了, 所以现在这里有仓库, 就直接使用这的命令

如果想让本地仓库与远程仓库形成关联, 就使用下面的命令

```
1 $ git remote add origin https://gitee.com/lovesnsfi_admin/aaa.git
```

当输入完上面的命令以后, 我们的远程仓库就添加成功了, 这个时候通过下面的方式来查看

```
YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa ((b69c30a...))
$ git remote add origin https://gitee.com/lovesnsfi_admin/aaa.git

YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa ((b69c30a...))
$ git remote -v
origin https://gitee.com/lovesnsfi_admin/aaa.git (fetch)
origin https://gitee.com/lovesnsfi_admin/aaa.git (push)

YangBiao@YB-Huawei MINGW64 ~/Desktop/aaa ((b69c30a...))
$ |
```

查看远程仓库

这个时候就可以将本地代码 与远程仓库代码 实现同步

```
1 $ git push -u origin "master"
```

movie-app的发布

1. 先在 .gitignore 里面注释掉 dist

```
1 # Logs
2 logs
3 *.log
4 npm-debug.log*
5 yarn-debug.log*
6 yarn-error.log*
7 pnpm-debug.log*
8 lerna-debug.log*
9
10 node_modules
11 #dist
12 dist-ssr
```



```

13 *.local
14
15 # Editor directories and files
16 .vscode/*
17 !.vscode/extensions.json
18 .idea
19 .DS_Store
20 *.suo
21 *.ntvs*
22 *.njsproj
23 *.sln
24 *.sw?

```

2. 找到 `vite.config.js`

```

1  import { defineConfig } from 'vite'
2  import vue from '@vitejs/plugin-vue'
3
4  // https://vitejs.dev/config/
5  export default defineConfig({
6    plugins: [vue()],
7    server: {
8      port: 9998
9    },
10   base: "./"
11 })

```

3. 执行打包的命令

```
1 $ npm run build
```

打包成功以后会在当前的项目下面多出一个dist的目录

4. 在 `gitee` 上面创建远程仓库

快速设置— 如果你知道该怎么操作，直接使用下面的地址

HTTPS
SSH

我们强烈建议所有的git仓库都有一个 README, LICENSE, .gitignore 文件

[初始化 readme 文件](#)

Git入门? 查看 [帮助](#), [Visual Studio](#) / [TortoiseGit](#) / [Eclipse](#) / [Xcode](#) 下如何连接本站 [如何导入仓库](#)

简易的命令行入门教程:

Git 全局设置:

```
git config --global user.name "二当家的"
git config --global user.email "lovesnsfi@163.com"
```

创建 git 仓库:

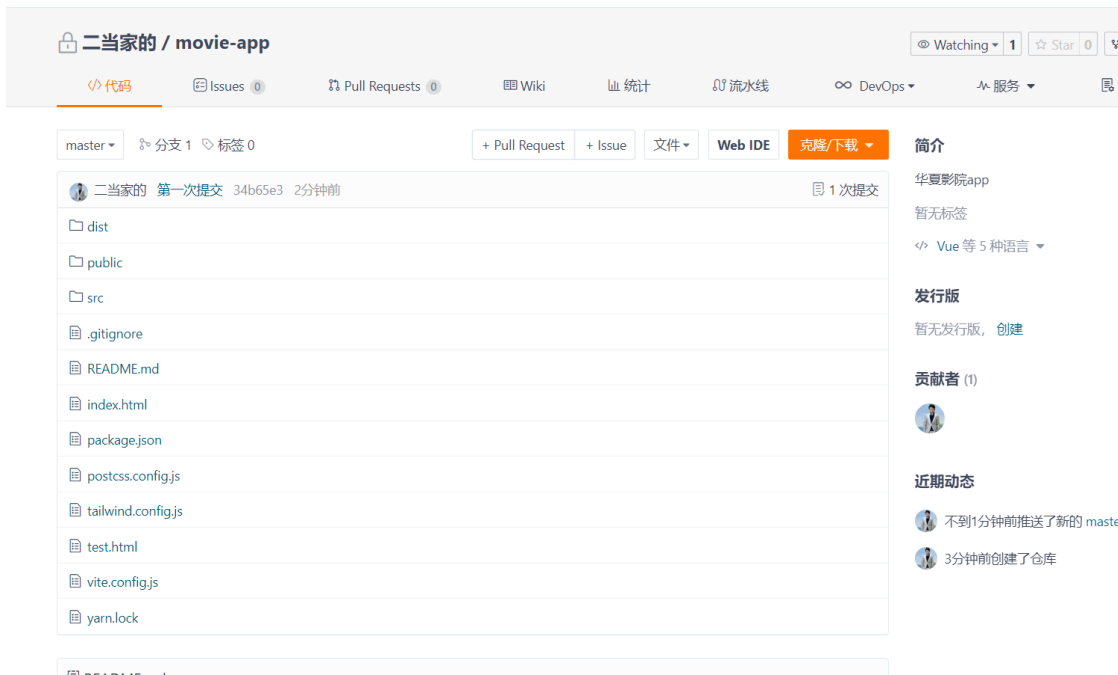
```
mkdir movie-app
cd movie-app
git init
touch README.md
git add README.md
git commit -m "first commit"
git remote add origin https://gitee.com/lovesnsfi_admin/movie-app.git
git push -u origin "master"
```

已有仓库?

```
cd existing git_repo
git remote add origin https://gitee.com/lovesnsfi_admin/movie-app.git
git push -u origin "master"
```

5. 本地初始化仓库，并提交到远程仓库

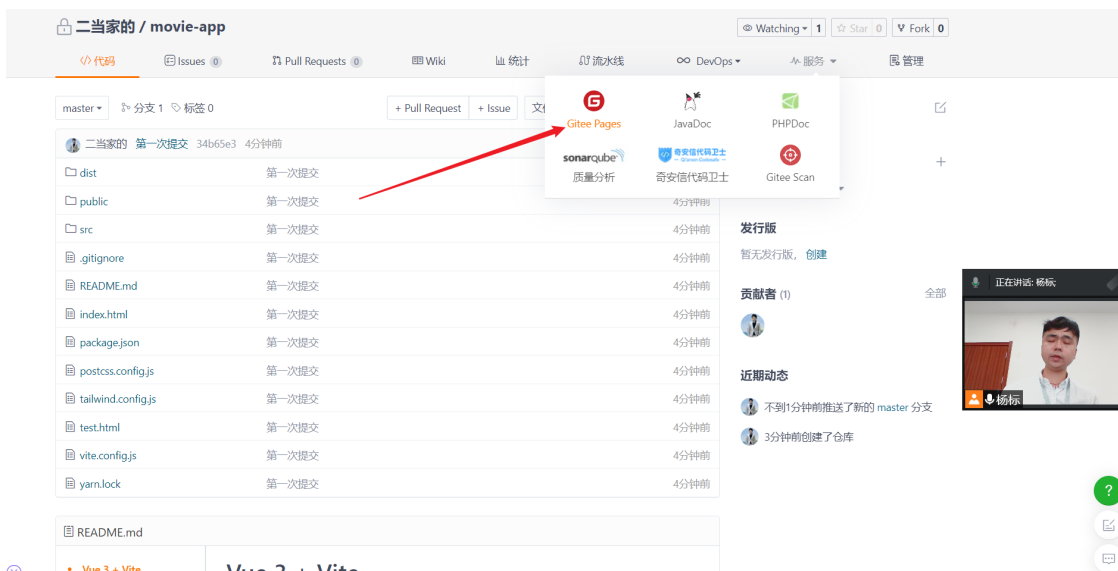
```
1 $ git init # 初始化
2 $ git add . # 添加所有文件
3 $ git commit -m "第一次提交" # 提交到本地仓库
4 $ git remote add origin https://gitee.com/lovesnsfi_admin/movie-app.git #配置
   远程仓库地址
5 $ git push -u origin master #将本地仓库推到远程仓库
```



这个时候我们的代码就到了远程仓库

6. 发布这个app

我们当时执行 `npm run build` 以后，将所有的代码打包，结果放在了 `dist` 这个目录里面去，现在只要将这个目录公开发布就可以了



开启 Pages 后所生成的静态资源将会公开
开启 Pages 后会在部署公钥中添加 pages 服务器的公钥

部署分支

master

选择您要部署的分支

部署目录

./dist

填写您要部署的分支上的目录

☐ 强制使用 HTTPS

启动

输入./dist,然后启动就可以了

平台保留 Gitee Pages 及 Gitee Pages Pro 使用规则最终解释权, 如发现上述违规行为, 平台将视违规情况严重程度予以 Pages 服务封禁以至账号永久封禁惩罚。

开启 Pages 后所生成的静态资源将会公开
开启 Pages 后会在部署公钥中添加 pages 服务器的公钥

已开启 Gitee Pages 服务, 网站地址: http://lovesnsfi_admin.gitee.io/movie-app

部署分支

master

选择您要部署的分支

部署成功以后就会有一个网址

部署目录

./dist

填写您要部署的分支上的目录

☐ 强制使用 HTTPS

更新

暂停

movie-manager的发布

1. 打开 `.gitignore`, 修改

```
1  .DS_Store
2  node_modules
3  #/dist
4
5
6  # local env files
7  #.env.local
8  #.env.*.local
9
10 # Log files
11 npm-debug.log*
12 yarn-debug.log*
13 yarn-error.log*
14 pnpm-debug.log*
15
16 # Editor directories and files
17 .idea
18 .vscode
19 *.suo
20 *.ntvs*
21 *.njsproj
22 *.sln
23 *.sw?
```

2. 修改vue.config.js文件

```
1  const { defineConfig } = require('@vue/cli-service')
2  module.exports = defineConfig({
3    transpileDependencies: true,
4    publicPath: "/"
5  })
```

3. 执行 `npm run build` 来进行打包

```
1  $ npm run build
```

打包完成以后，在当前项目下面，一样会生成一个dist的目录

4. 在 `gitee` 上面创建远程仓库

新建仓库

在其他网站已经有仓库了吗? [点击导入](#)

仓库名称 * ✓

movie-manager

归属

二当家的

路径 * ✓

movie-manager

仓库地址: https://gitee.com/lovesnsfi_admin/movie-manager

仓库介绍

4/200

华夏影院

☐ 开源 (所有人可见) ②

☒ 私有 (仅仓库成员可见)

☐ 企业内部开源 (仅企业成员可见) ②

☐ 初始化仓库 (设置语言、.gitignore、开源许可证)

☐ 设置模板 (添加 README、Issue、Pull Request 模板文件)

☐ 选择分支模型 (仓库创建后将根据所选模型创建分支)

创建

快速设置— 如果你知道该怎么操作，直接使用下面的地址

HTTPS SSH https://gitee.com/lovesnsfi_admin/movie-manager

我们强烈建议所有的git仓库都有一个 README, LICENSE, .gitignore 文件

[初始化 readme 文件](#)

Git入门? 查看 [帮助](#), [Visual Studio](#) / [TortoiseGit](#) / [Eclipse](#) / [Xcode](#) 下如何连接本站 如何导入仓库

简易的命令行入门教程:

Git 全局设置:

```
git config --global user.name "二当家的"
git config --global user.email "lovesnsfi@163.com"
```

创建 git 仓库:

```
mkdir movie-manager
cd movie-manager
git init
touch README.md
git add README.md
git commit -m "first commit"
git remote add origin https://gitee.com/lovesnsfi_admin/movie-manager.git
git push -u origin "master"
```

已有仓库?

```
cd existing_git_repo
git remote add origin https://gitee.com/lovesnsfi_admin/movie-manager.git
git push -u origin "master"
```

5. 初始化本地仓库，并提交

```

1 $ git init
2 $ git add .
3 $ git commit -m "第一次提交"
4 $ git remote add origin https://gitee.com/lovesnsfi_admin/movie-manager.git
5 $ git push -u origin master

```

6. 发布页面



Gitee Pages 服务 一个支持Jekyll、Hugo、Hexo静态网站的服务 [使用帮助](#)

Pages 服务仅供博客 / 门户 / 开源项目网站 / 开源项目静态效果演示用途，请勿用于违规内容，包括但不限于：

- 发布诱导分享/诱导关注/诱导下载/诱导跳转内容
- 发布欺诈/谣言/骚扰信息/广告信息/垃圾信息/特殊识别码、口令类信息
- 发布低俗内容/宗教性捐款及相关信息
- 发布侵害他人权利/违法经营及可疑服务类内容
- 发布其它违反国家法律法规的内容

平台保留 Gitee Pages 及 Gitee Pages Pro 使用规则最终解释权，如发现上述违规行为，平台将视违规情况严重程度予以 Pages 服务封禁以至账号永久封禁惩罚。

开启 Pages 后所生成的静态资源将会公开
开启 Pages 后会在部署公钥中添加 pages 服务器的公钥

部署分支

master

选择您要部署的分支

部署目录

./dist

填写您要部署的分支上的目录

☐ 强制使用 HTTPS

启动

输入./dist 来指定要发布的目录

开启 Pages 后所生成的静态资源将会公开
开启 Pages 后会在部署公钥中添加 pages 服务器的公钥

已开启 Gitee Pages 服务，网站地址：http://lovesnsfi_admin.gitee.io/movie-manager

部署分支

master

选择您要部署的分支

部署目录

./dist

填写您要部署的分支上的目录

☐ 强制使用 HTTPS

更新

暂停

发布成功以后，这里就有网址了

git在线学习网站

<https://learngitbranching.js.org/>