## Maven:

是一个自动化构建项目的工具，可以帮我们管理依赖（帮我们自动下载，如果有依赖传递，也会帮我们下载好），编译程序，将代码打包成jar包，部署项目。

在下载的maven的setting.xml中进行修改，能够配置本地仓库的位置

需要jar包，先从本地仓库看看有没有，没有就去中央仓库中查找，查找完后，会下载到本地仓库，下次需要就不需要去中央仓库查找了，相当于是一个二级缓存。

<dependency>中jar包的坐标（相当于我们的身份证） groupId, artifactId,version。其中version分为开发版snapshot和发布版release。依赖冲突：因为同一个groupId和artifctId下只能出现一个版本。

<scope>依赖的范围：默认是compile，在编译（构建build的时候，编程.class文件），测试（juint测试的时候有效），运行（run)都有效。

test范围（只在测试的时候有效，）

Provide范围（说明运行的容器已经提供，只在编译测试的时候有用）

Runtime范围 测试，运行时候在有用例如就JDBC驱动，

System范围：系统指定的依赖文件使用的范围。相应的jar包是由系统提供。

Compile，test，runtime，provide，system

Maven的生命周期：clean，compile， test， package， install，deploy.清理，编译，测试，打包，安装，部署。

父子模块：父模块声明：dependencyManagement 和 pluginManagement，子模块通过指定parent。写dependency就可以不写版本号，直接从父类中继承，如果写了，还是使用子类声明的版本号。

Maven：安装目录，setting.xml文件，本地仓库位置。

Maven：项目目录结构

Pom.xml

Src/main/java

Src/main/resource

Src/test/java

Src/test/resource

Target/classes

Target/test-class

Target/generated-resource

Target/generated-test-resource

git status | git config -l | git remote -v

## Git:的使用

1. 个人开发，可以穿越时光，跑到以前备份的版本。
2. 多人开发，使用push和pull进行协同办公，

Git的常用命令：

命令：设置git的user.name 和 user.email： git config --globle user.name “zhangkuan” git config --globle user.email [“3360094030@qq.com”](mailto:“3360094030@qq.com”)

git config --global user.name “zhangkuan”

git config --global user.eamil [“3360094030@qq.com”](mailto:“3360094030@qq.com”)

git init

git add,

git commit -m”提交信息”

git remote add origin “3360094030@qq.com”

git push -u origin master 将本地仓库push到远程仓库

git clone “远程仓库的地址”

通过git config -l 进行察看。

架构：工作区（working area）

版本库（暂存区stage， master分支）从工作区到暂存区使用的是add增加零星的文件。从暂存区到master分支，将暂存区当作是一个版本分支，用commit命令

通过命令git.init 创建版本库。其他的地方就叫做工作区。

git status, 察看文件的状态

git add <文件名> 点就包括所有的文件，将文件添加到暂存区。

git commit -m “版本信息” 将暂存区中的文件当作是一个版本，添加到当前分支中去。

Github和Gitee是远程服务器，我们实在这个远程服务器上构建我们的远程仓库。

没有仓库：

mkdir repo\_1

Cd repo\_1

git init

touch README.md

git add README.md

git commit -m “版本信息”

git remote add origin <https://gitee.com/zhangkuan-sit/git_test> 察看git remote -v

git push origin master

git pull origin master

已有仓库：

已经有了.git文件

每个分支相当于是一个指针。

察看分支 git branch

创建分支 git branch dev

切换分支 git checkout master git checkout dev

一个分支有多个提交版本，察看每个版本的提交信息：git log（查看提交的详细信息） | git log --oneline (查看提交的简略信息）

分支的合并：快速合并（当是属于子集关系的时候、三方合并（当是交集的时候，采用）。

git merger dev (当前是在主分支，主分支将dev进行吞并，形成新的分支)

git log --oneline -graph

git push origin 源：目的地 （本地分支：远程分支）

Git pull origin 源：目的地 （远程分支：本地分支）

git pull = git fetch + git merge FETCH\_HEAD

在每次进行git push前，进行git pull（1.防止有冲突，2.防止将别人push内容覆盖）

在文件夹中初始化一个仓库：git init

工作区到暂存区: git add 文件名

将暂存区提交到git仓库，并且形成一个版本：git commit -m “提交的信息”

添加远程关联：git remote add origin 远程地址

如果remote已经被占用：使用git remote rm origin

本地在master分支，提交到远程中：git push -u origin main

如果是其他分支，提交到远程中：git push -u origin dev

为了在push前，如果远程仓库不为空：要先将远程仓库拉取，并合并，在使用push。

在git bash中使用命令ssh-keygen -o

使用ssh和https的区别，https每次push和pull要输入github的账号和密码（我们用idea就不需要了）。而是用ssh就不需要每次都输入账号和密码了

## Junit5：

使用前，添加maven依赖：Junit-jupiter

使用：再springboot项目中，为测试类添加注解@SpringBootTest, [junit4中使用注解@RunnerWitn(SpringRunner.class),就可以使用spring中的Javabean。再配合使用自动装配注解@Autowried（通过ByType进行配对。添加注解@Qualifier（“id”）通国ByName的方式进行注解。）](mailto:junit4中使用注解@RunnerWitn(SpringRunner.class),就可以使用spring中的Javabean。再配合使用自动装配注解@Autowried（通过ByType进行配对。添加注解@Qualifier（“id”）通国ByName的方式进行注解。）)@Resource通过Byname进行配对。

@Test

void f() { // 方法默认是private ，返回类型为void。

...

}

@Disabled 测试的时候，跳过此方法进行测试。

@Before Each， @AfterEach，

@BeforeAll , @AfterAll 里面的方法要采用static 方法。

测试中使用到的类：Assertions.assertEquals, Assertions.assertNotEquals();