Git

一、Git初识

（零）SVN 回顾

- 图标：
  - 绿色对号：已经加入版本控制，同时和服务器中的某个版本完全一致；
  - 蓝色加号：已经加入版本控制，但还未提交到服务器；
  - 红色叹号：当前版本与服务器上同一版本内容不一致；
  - 黄色叹号：文件发生冲突；
- 操作
  - 下载：svn checkout 下载
  - 提交：svn commit 提交
  - 更新：svn update 更新
- 注意事项：
  - 公共文件：修改之前应该先加锁（避免别人提交造成的冲突现象），再修改，再提交
  - 个人文件：修改，再提交

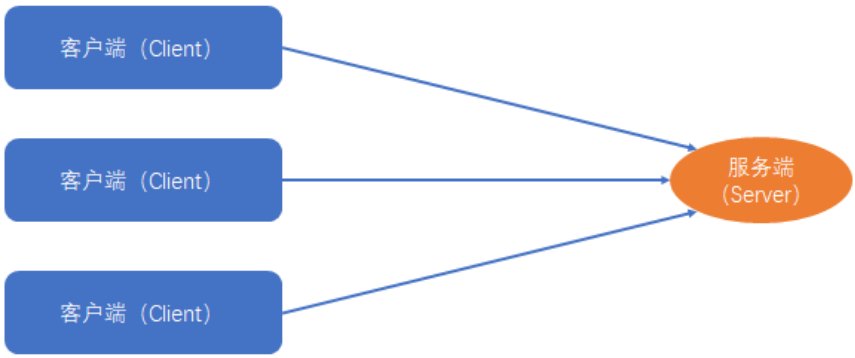
（一）版本控制

> 版本控制是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况以及回溯的系统，任何类型的文件都可以进行版本控制。
>
> **版本控制工具：**提供完备的版本管理功能，用于存储、追踪目录(文件夹)和文件的修改历史，是软件开发者的必备工具，是软件公司的基础设施。版本控制软件的最高目标，是支持软件公司的配置管理活动，追踪多个版本的开发和维护活动，及时发布软件。
>
> - Git 是一个开源的**分布式版本控制系统**，用于敏捷高效地处理任何或小或大的项目。

> - Git 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 与常用的版本控制工具 CVS、Subversion（SVN） 等不同，它采用了**分布式版本库**的方式，**不必服务器端软件支持。**

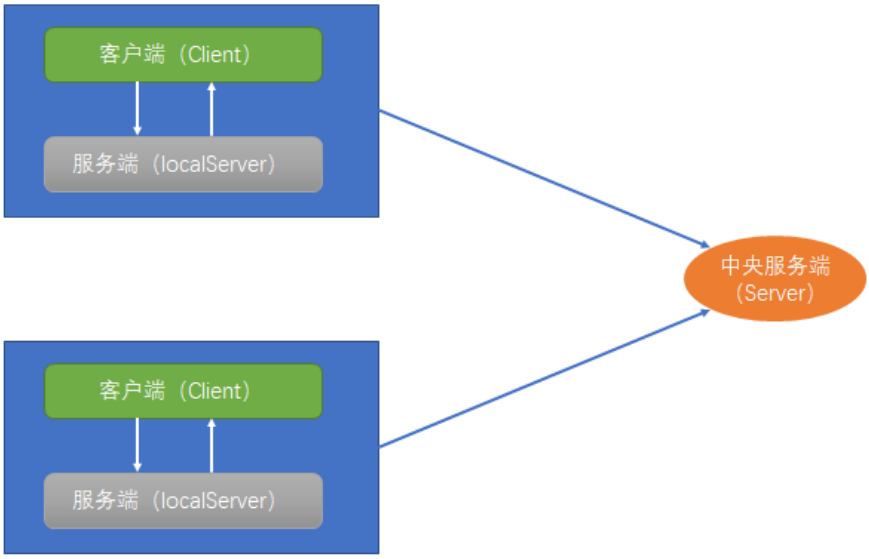
- 传统模式：

> 客户端和服务端连接紧密，在和服务段断开连接的时候无法进行代码版本管理，效率比较低。



- 分布式模式：

> 每个机器上都有**客户端和本地服务端（本地库）**，代码提交时先通过本地库进行版本控制，和**中央服务器**关联性不大。只需要在代码汇总的时候将本地的文件提交至中央服务器即可。这样不用考虑外网的问题，即使断网的情况也可以进行代码管理。



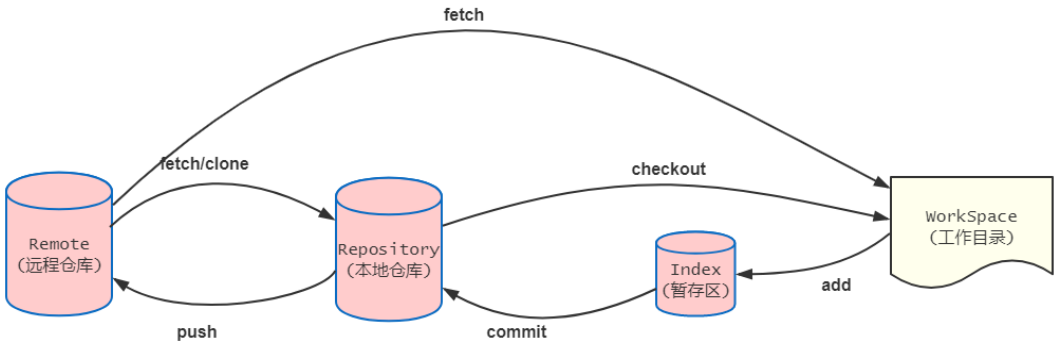
Git与SVN的区别：

> - **1、Git 是分布式的，SVN 不是**：这是 Git 和其它非分布式的版本控制系统，例如 SVN，CVS 等，最核心的区别。

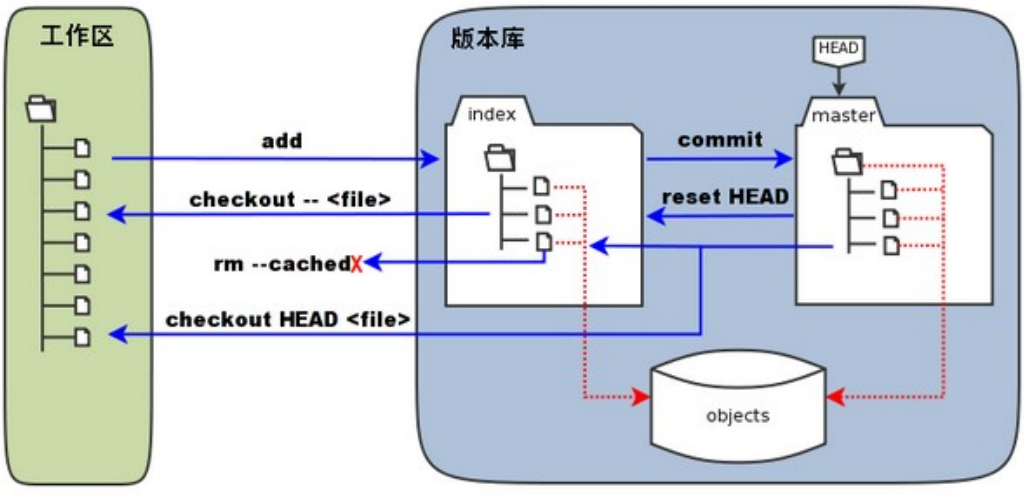
- **2、Git 把内容按元数据方式存储，而 SVN 是按文件**：所有的资源控制系统都是把文件的元信息隐藏在一个类似 .svn、.cvs 等的文件夹里。
- **3、Git 分支和 SVN 的分支不同**：分支在 SVN 中一点都不特别，其实它就是版本库中的另外一个目录。
- **4、Git 没有一个全局的版本号，而 SVN 有**：目前为止这是跟 SVN 相比 Git 缺少的最大的一个特征。
- **5、Git 的内容完整性要优于 SVN**：Git 的内容存储使用的是 SHA-1 哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。

(二) Git 基本理论

- **工作区（Working Directory）**：在某个文件夹下操作的，这个文件夹就是工作区。项目中被管理文件所在的目录。
- **暂存区（Stage、Index）**：存放在工作区根目录下隐藏的 ".git目录下" 下的index文件（.git/index）中，所以我们把暂存区有时也叫作索引（index）。
- **版本库（Repository）**：工作区有一个隐藏目录 `.git`，这个不算工作区，而是Git的版本库，也可以称为本地仓库。存放提交的所有数据以及HEAD指向最新放入仓库的版本。
- **远程仓库（Remote Directory）**：代码或者文件托管的远程服务器。



在初始化git版本库之后会生成一个**隐藏的文件夹** `.git`，可以将该文件夹理解为git的版本库 `repository`，而自己建立的项目文件夹即工作区 `working directory`，在.git 文件夹里面还有很多文件，其中有一个 `index` 文件 就是暂存区也可以叫做 `stage`，git还为我们自动生成了一个分支 `master` 以及指向该分支的指针 `head`，如下图：



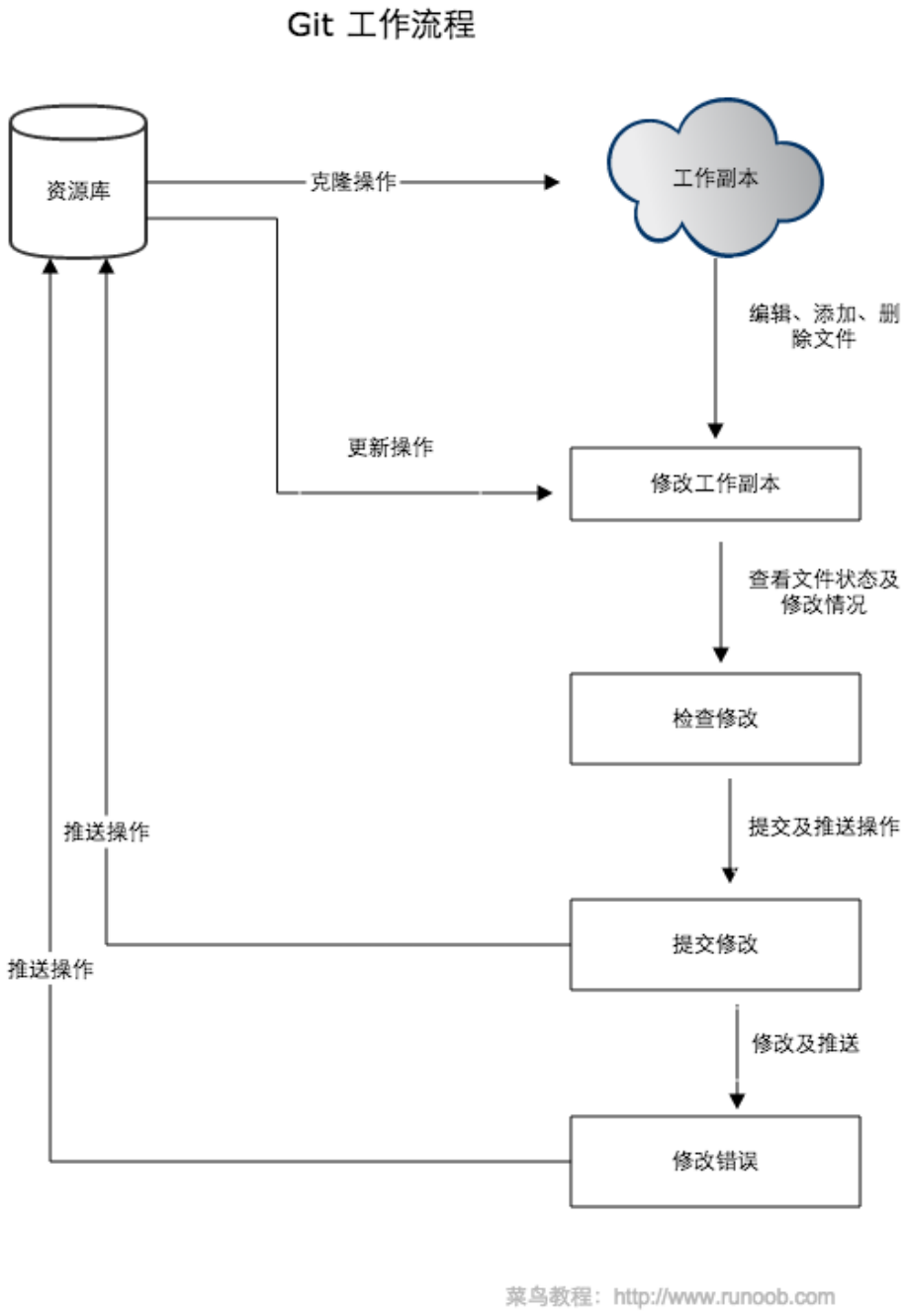
> 从图中可以看出来本地库包括分支`master`和暂存区（stage），工作区可以理解为我们打开开发环境如idea，里面的内容即工作区的内容，在工作区里面有的代码以及配置文件等我们需要提交到版本库里面，最终是到了分支master上面，**暂存区只是一个临时保存修改文件的地方。**

工作流程：

> - 图中左侧为工作区，右侧为版本库。在版本库中标记为 "index" 的区域是暂存区（stage/index），标记为 "master" 的是主分支所代表的目录树。
> - 图中我们可以看出此时 "HEAD" 实际是指向 master 分支的一个"游标"。所以图示的命令中出现 HEAD 的地方可以用 master 来替换。
> - 图中的 objects 标识的区域为 Git 的对象库，实际位于 ".git/objects" 目录下，里面包含了创建的各种对象及内容。
> - 当对工作区修改（或新增）的文件执行`git add`命令时，暂存区的目录树被更新，同时工作区修改（或新增）的文件内容被写入到对象库中的一个新的对象中，而该对象的ID被记录在暂存区的文件索引中。
> - 当执行提交操作`git commit`时，暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树。
> - 当执行`git reset HEAD`命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。
> - 当执行`git rm --cached <file>`命令时，会直接从暂存区删除文件，工作区则不做出改变。
> - 当执行`git checkout . 或者 git checkout -- <file>`命令时，会用暂存区全部或指定的文件替换工作区的文件。这个操作很危险，会清除工作区中未添加到暂存区中的改动。
> - 当执行`git checkout HEAD .`或者`git checkout HEAD <file>`命令时，会用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件。这个命令也是极具危险性的，因为不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改动。

（三）工作流程

> 一般工作流程如下：
>
> - 克隆 Git 资源作为工作区（初始化工作区）。
> - 在工作区的资源上添加或修改文件。
> - 如果其他人修改了，你可以更新资源。
> - 在提交前查看修改。
> - 提交修改。
> - 在修改完成后，如果发现错误，可以撤回提交并再次修改并提交。



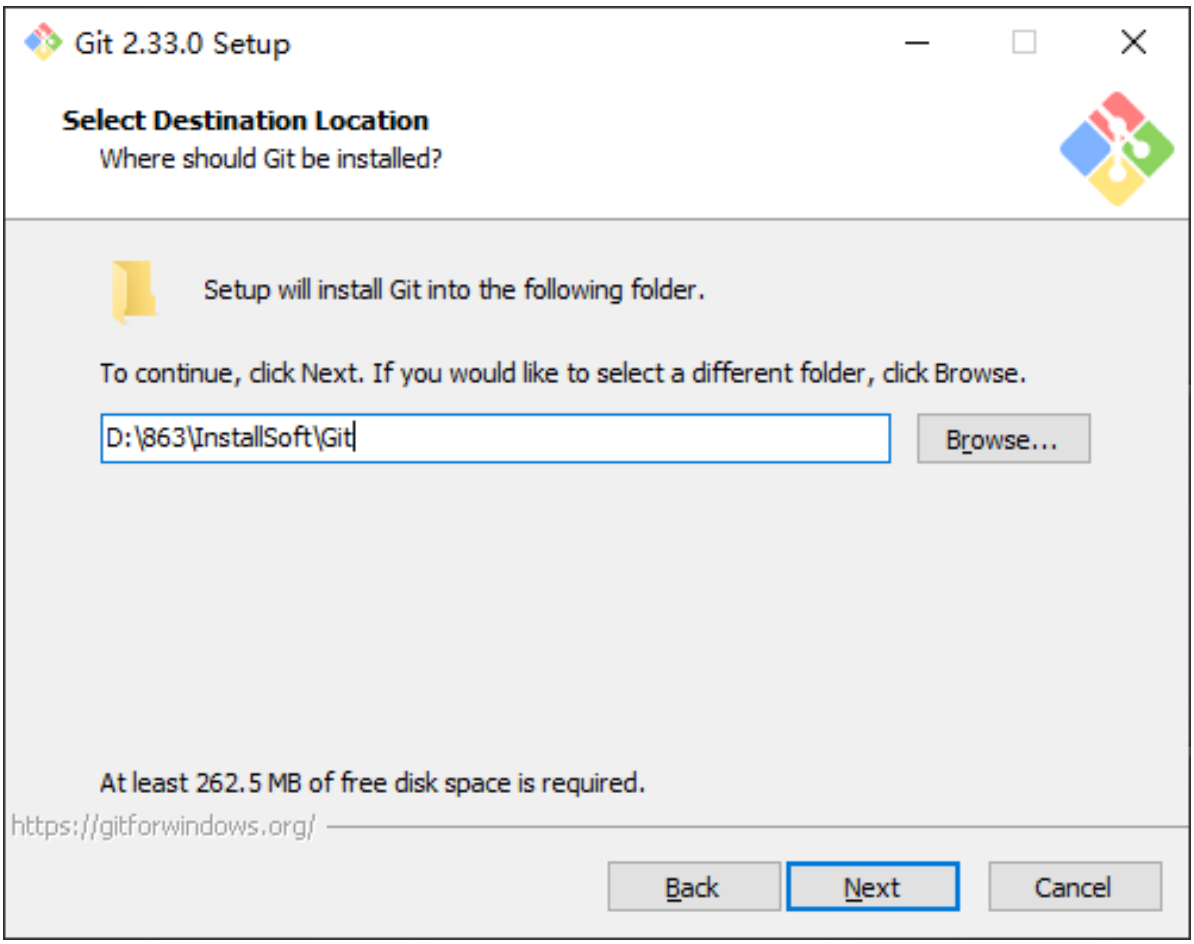
(四) Git安装

- **Git下载：**

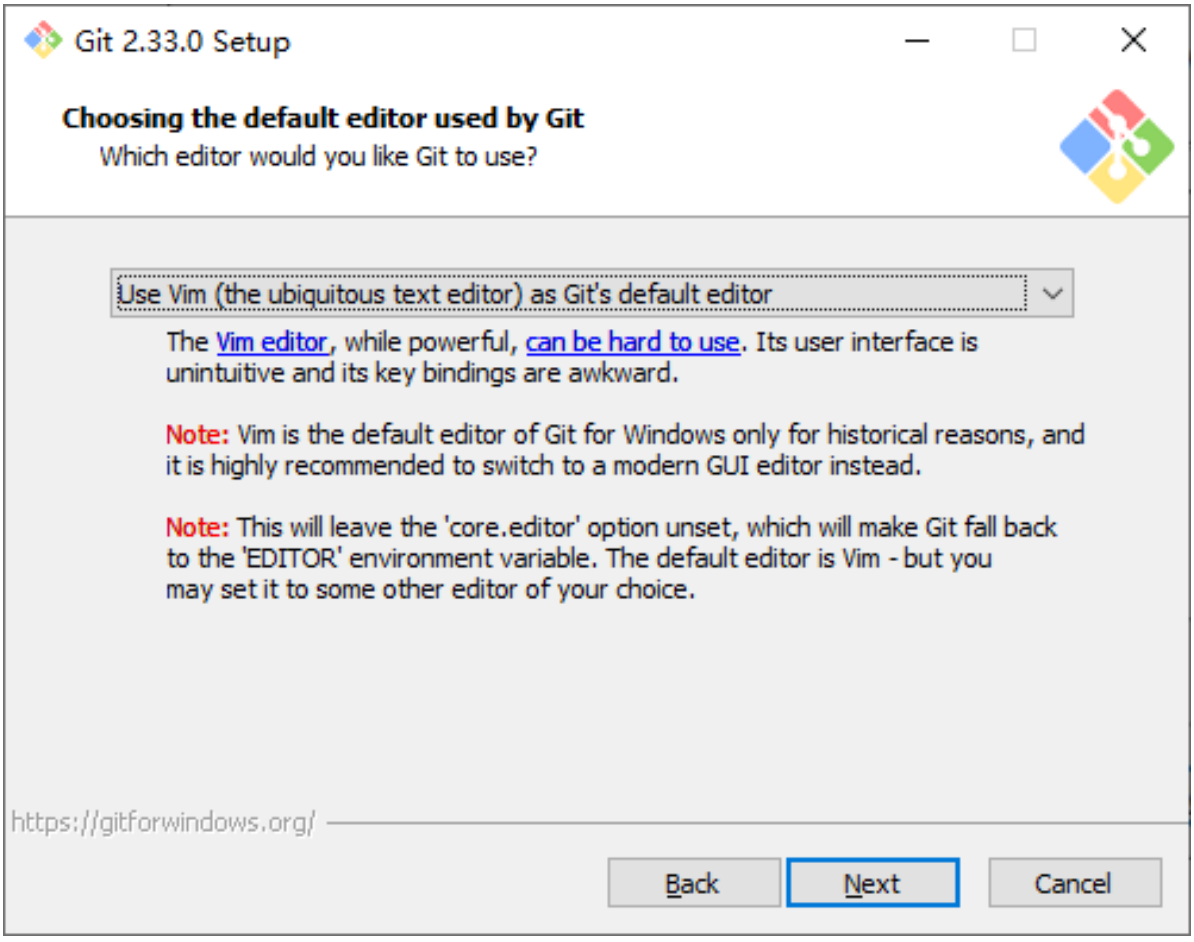
> Git官网：https://git-scm.com/ Git插件：https://tortoisegit.org/，类似SVN的图标提示。 NPM镜像下载：https://npm.taobao.org/mirrors/

- **Git安装：**

> 选择安装地址，不要安装到C盘，安装路径不要含中文、日文等特殊字符。

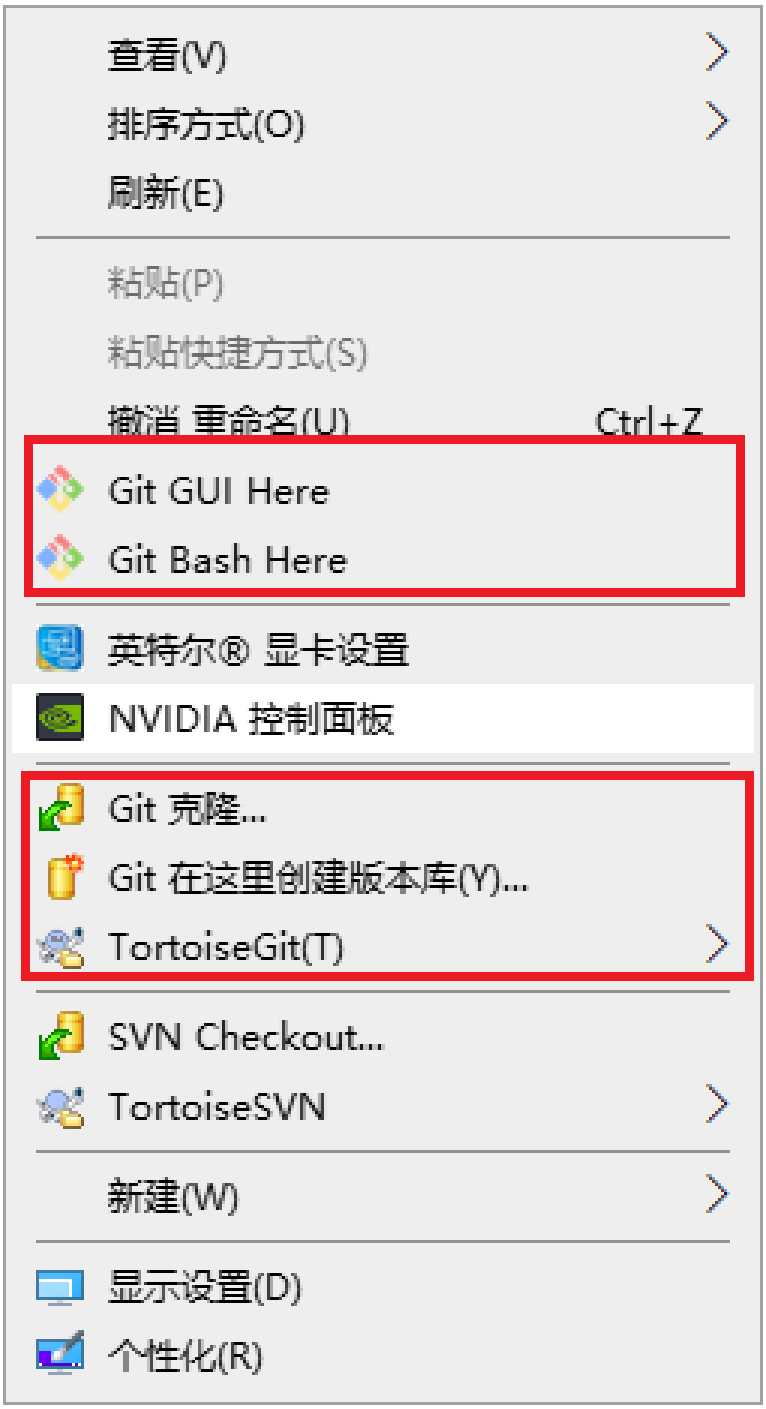


> 选择编辑器：默认使用vim编辑器，也可以根据需求自选



- **启动Git：**

Git和Git插件的安装相对简单，默认安装即可。安装成功之后右键菜单会出现图中红框选项，第一个红框为Git选项，第二个红框为Git插件选项；



开始菜单对应程序

> **Git Bash :** Unix和Linux风格的命令行（使用较多） **Git CMD :** Windows风格命令行 **Git GUI :** 可视化界面Git

（五）Linux 中常用命令

> 工作中建议使用 Linux 命令操作目录。

```
# 根目录；目录的分割标识
/
# 切换跳转目录，change directory，cd 后面通过空格分割目录。例如：cd dir、cd ..
cd
# 查看当前所在目录，print working directory
pwd
# 查看当前目录下所有的文件
ll、ls
# 清屏
clear
# 创建文件，例如：touch a.txt
touch
# 删除一个文件，remove，例如：rm a.txt
rm
# 创建一个目录（新建文件夹），make directory
mkdir
# 删除一个文件夹，例如：rm -r src
rm -r
# 移动文件，例如：mv a.txt src，a.txt是要移动的文件，src是要移动到的目录。
mv
# 查看已经使用的历史命令
history
# 退出
exit
```

（六）Git配置

> 在没有核心仓库的前提下，GIt中的账号信息，需要自己配置，配置的这个名字，会在未来的各个操作中体现。

```
# 查看全局的配置 #
# 查看所有的配置(系统+全局)
git config -l
# 查看系统默认配置
git config --system --list
# 查看全局配置
git config --global --list
```

```
# 设置全局的账号和邮箱，是本地用户的标识
git config --global user.name "用户名"
# 邮箱如果不连接远程库可以随意，如果需要远程库，必须是远程库注册的邮箱
git config --global user.email "用户邮箱"
```

Git相关的配置文件：

1. `Git\etc\gitconfig`：git安装目录下，默认的系统配置，--system
2. `Users\user\.gitconfig`：系统用户目录下当前用户的配置，--global

二、Git基本操作

（一）搭建本地仓库

```
# 初始化仓库，在目录下会生成一个.git的文件：
git init
```

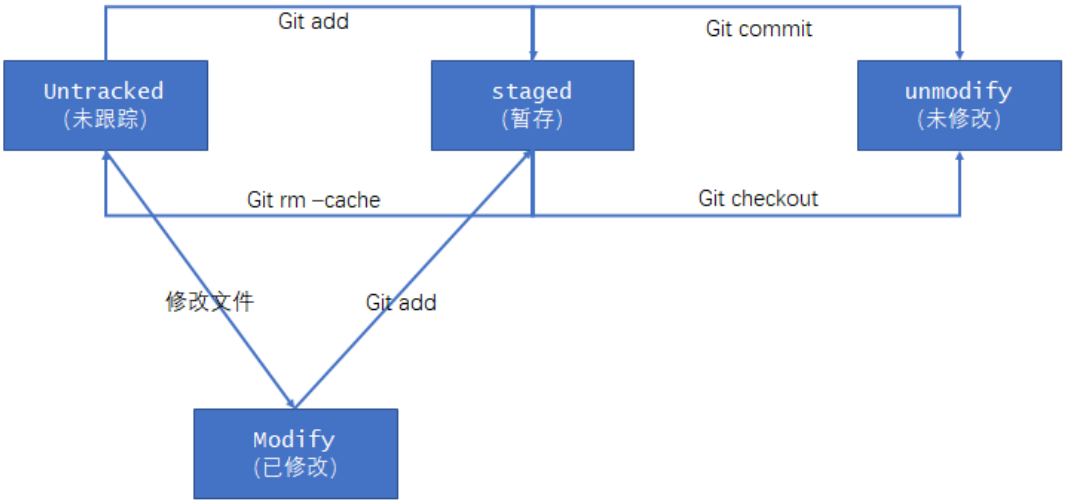
```
# 从远程仓库克隆
git clone 地址
```

（二）本地库操作

1、文件的状态

> 版本控制就是对文件的版本控制，要对文件进行修改提交等操作，必须要先知道文件的当前状态，不然可能会提交了不想提交的文件或者提交的文件没有提交上。
>
> - **untracked**：未跟踪，此文件在当前文件夹中，并未添加到 git 本地库中（暂存区），可以通过`git add`命令把当前文件的状态变成**staged**
> - **unmodify**：文件已经添加到本地库，未修改，即本地库中的文件版本和文件夹中的文件版本完全一致，当前状态的文件如果被修改，将变成**modify**。或者使用`git rm`移出版本库，变成**untracked**状态。
> - **modify**：文件被修改。当前状态的文件可以使用`git add`添加到暂存区变成**staged**状态。使用`git checkout`丢弃修改内容，回到**unmodify**状态。
> - **staged**：暂存状态，使用`git commit`将修改提交到本地库中，此时本地库中的文件和本地文件完全一致，文件变为**unmodify**状态。使用`git reset HEAD 文件名`取消暂存状态，变成**modify**状态。

```
# 查看所有文件状态
git status

# 查看指定文件状态
git status fileName
```



- **图标含义**
  - 没有图标：文件未跟踪，untracked 状态
  - 蓝色加号图表：文件已经添加到缓存区，变成了 staged 状态
  - 绿色对号图标：文件已经提交本地库，且本地没有发生变化
  - 红色叹号图标：文件已经提交本地库，且本地文件发生了变化

2、本地库操作

注意：修改的文件或者新建的文件，一定先添加到暂存区再提交到本地库。

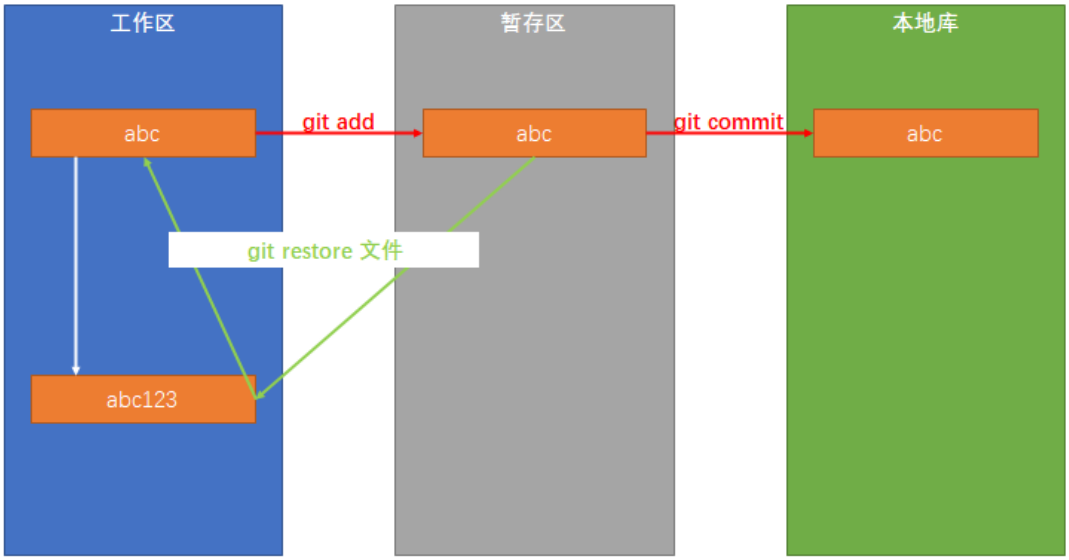
通用命令：

```
# 查看文件的状态（红色表示未被管理或者已经修改；绿色表示已经被管理但未提交）
git status
```

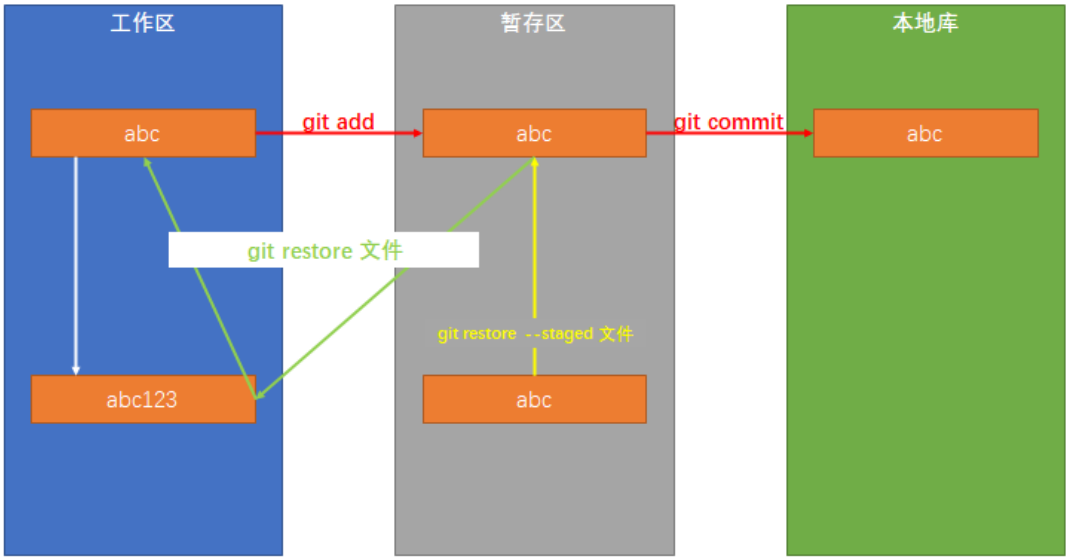
工作区与暂存区：

```
# 添加文件到暂存区
git add [文件名称]

# 查看已经提交至暂存区的文件
git ls-files

# 撤销在暂存区提交的文件
git restore --staged 文件

# 用暂存区的文件替换工作区文件
git restore [文件名]
```

> 工作区文件修改，并未提交至暂存区。希望撤销修改内容，执行`git restore 文件名`命令。



> 工作区文件修改，并提交至暂存区。希望撤销修改内容，执行`git restore --staged 文件名`命令撤销提交状态，再执行`git restore 文件名`命令恢复。




```
# 删除暂存区的文件，工作区文件内容不变（不希望该文件被版本控制）。
git rm --cached 文件

# 删除文件（暂存区和本地磁盘文件均被删除）
git rm 文件名称

# 用暂存区的文件替换工作区文件
git checkout [文件名]
```

暂存区与本地库：

```
# 把文件提交到版本库，一定要加注释
git commit -m "备注"

# 将最新指针指向的内容覆盖掉暂存区的内容
git reset HEAD
```

3、忽略提交

> 有些情况下并不希望将所有的文件都提交，例如：IDE自动生成的配置文件、target目录、out目录等。可以在git主目录下创建一个`.gitignore`文件进行配置，规则如下：
>
> 1、注释：# 2、通配符：星号（*）表示任意多个字符，问号（?）表示一个字符，方括号（[abc]）表示可选字符范围，大括号（{str1,str2,str3,.....}）表示可选字符串。 3、叹号（!）：非，表示反向例外的含义 4、路径分割符（/）：分割符在路径前表示要忽略的文件在当前目录下，子目录中的文件不被忽略；分隔符在路径后表示要忽略的文件是当前目录的子目录，非文件

```
# 注释
*.txt # 忽略所有的txt结尾的文件
!a.txt # a.txt文件不被忽略
/dir # 忽略根目录下dir直接目录下的文件
dir/ # 忽略dir目录下的所有文件
dir/*.txt # 忽略dir直接目录下的所有txt文件
```

Java开发通用版：

```
#java
*.class

#package file
*.war
*.ear
*.zip
*.tar.gz
*.rar
#maven ignore
target/
build/

#eclipse ignore
.settings/
.project
.classpatch

#Intellij idea
.idea/
/idea/
*.ipr
*.iml
*.iws

# temp file
*.log
*.cache
*.diff
*.patch
*.tmp
```

```
32
33 # system ignore
34 .DS_Store
35 Thumbs.db
```

（三）远程库操作

- 远程服务器

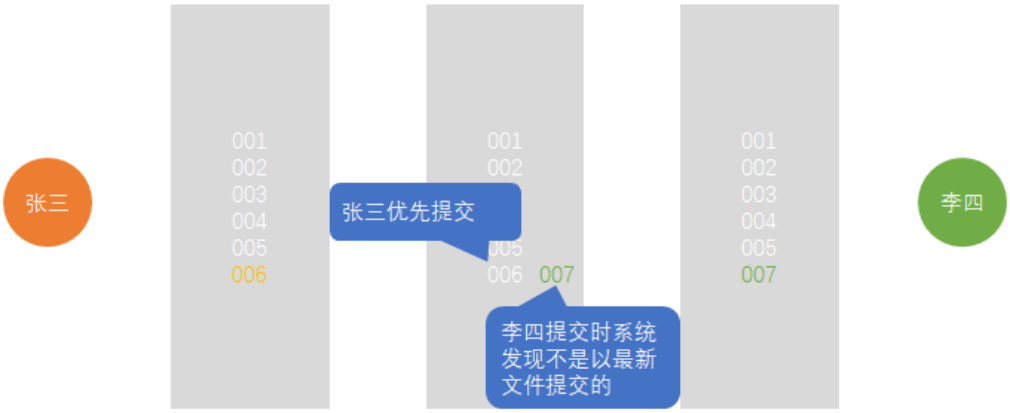
```
1 gitHub：https://github.com/
2 gitee（码云）：https://gitee.com/
3 gitblit（搭建个人Git服务器）：http://gitblit.github.io/gitblit/
```

- 操作远程库

```
1  # 查看远程仓库列表
2  git remote -v
3  git remote
4
5  # 配置远程仓库，从本地初始化仓库的时候添加/本地切换远程地址时添加
6  git remote add 别名 地址
7
8  # 提交到远程仓库
9  # 提交的时候一定要保证本地和服务器上的某一个版本一致才能提交
10 # 信息要互通
11 git push 别名 分支名称
12
13 # 强制提交：强制提交可能把服务器上的文件覆盖掉
14 git push -u origin master -f
15
16 # 更新远程仓库到本地
17 git pull 别名 分支名称
18
19 # 解决更新冲突问题：强制合并
20 git pull 别名 分支名称 --allow-unrelated-histories
21
22 # 克隆现有的远程仓库，在本地构建一个仓库
23 git clone 地址
24
25 # 删除现有仓库配置
26 git remote rm 别名
```

- 冲突解决

> 冲突是指 2 个以上的用户，操作远程库中同一个文件时，一个用户优先提交。在另一个用户不知情（没有更新到服务器最新版）的情况下，进行了提交，造成文件不能合并的现象。使用`git pull origin master`更新，此时会出现黄色感叹号，表示冲突。



解决冲突之前建议一定要手动备份自己的文件。

手动对比冲突内容：

1. 冲突的文件中包含自己的内容和服务器上的新内容，手动删除冲突的内容，完成手动解决冲突。
2. 使用`git add`和`git commit -m "注释"`命令提交至本地库。
3. 使用`git push origin master`命令提交至远程库。

远程覆盖本地文件：覆盖指定文件

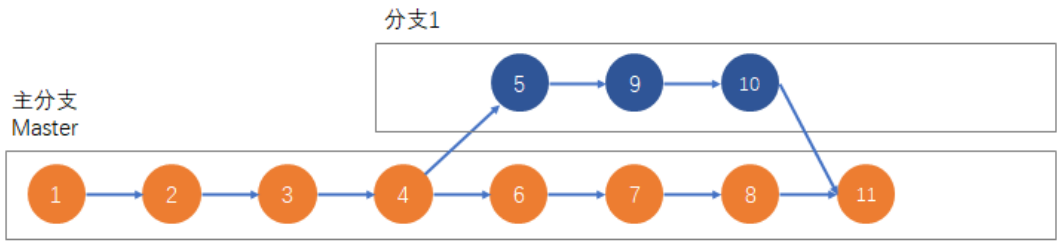
1. 使用`git fetch`命令将服务器上的遍历到本地库。
2. 使用`git checkout origin/master 文件路径/文件名`命令将本地文件覆盖。

远程覆盖本地：覆盖所有文件

1. 使用`git fetch --all`命令将服务器上的遍历到本地库。
2. 使用`git reset --hard origin/master`命令将本地文件覆盖。
3. 最后执行`git pull`更新。

三、分支

> 游戏中的主线任务和副本任务。正常情况下按照主线任务一步步积累经验升级。当到一定等级的，主线任务不足以支持升级的时候，系统会推荐副本任务，来帮助积累经验进行升级。同时主线任务也可以正常完成。
>
> 企业开发中，都会按照需求逐步完成，提交到主分支上。如果需求进行了变更，这个需求可能不被采纳，但还还要尝试进行完成需求。此时不建议在主分支上进行需求的变更，可以在某个版本上创建分支，在分支中进行开发。如果开发完成不了直接删除分支，不会对主分支产生影响。如果开发完成了，可以将分支合并到主分支上，相对比较灵活。



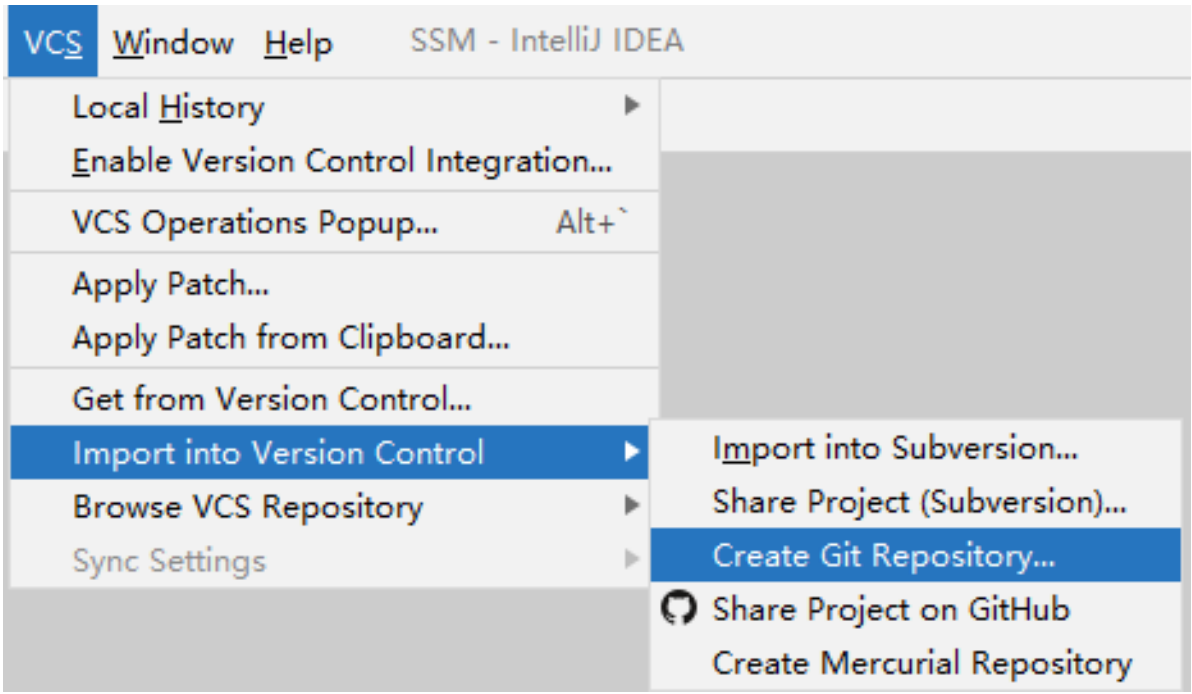
```
# 查看当前本地所有的分支
git branch

# 查看所有远程分支
git branch -r

# 创建新的分支，但不切换分支
git branch 分支名称

# 切换分支
git checkout 分支名称

# 合并指定分支到当前分支
git merge [分支名称]

# 删除本地服务器分支
git branch -d 别名/分支名称

# 删除远程服务器分支
git branch -r -d 别名/分支名称
```

四、Idea集成Git

> 通过Idea集成Git操作项目，远程地址创建仓库的时候，仓库名字和项目名称保持一致。保证远程服务器是一个空的仓库。

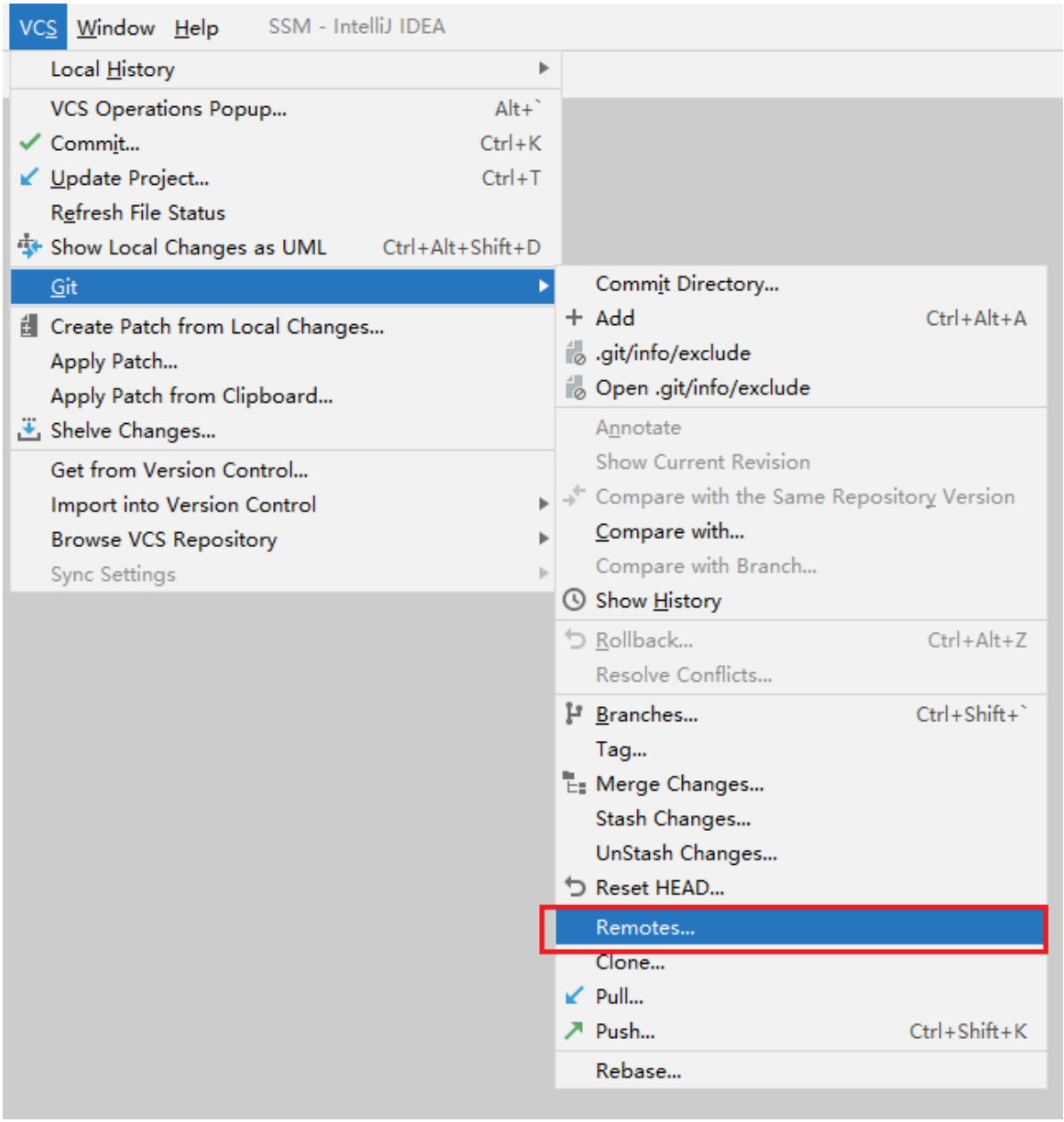
（一）本地项目初始化提交

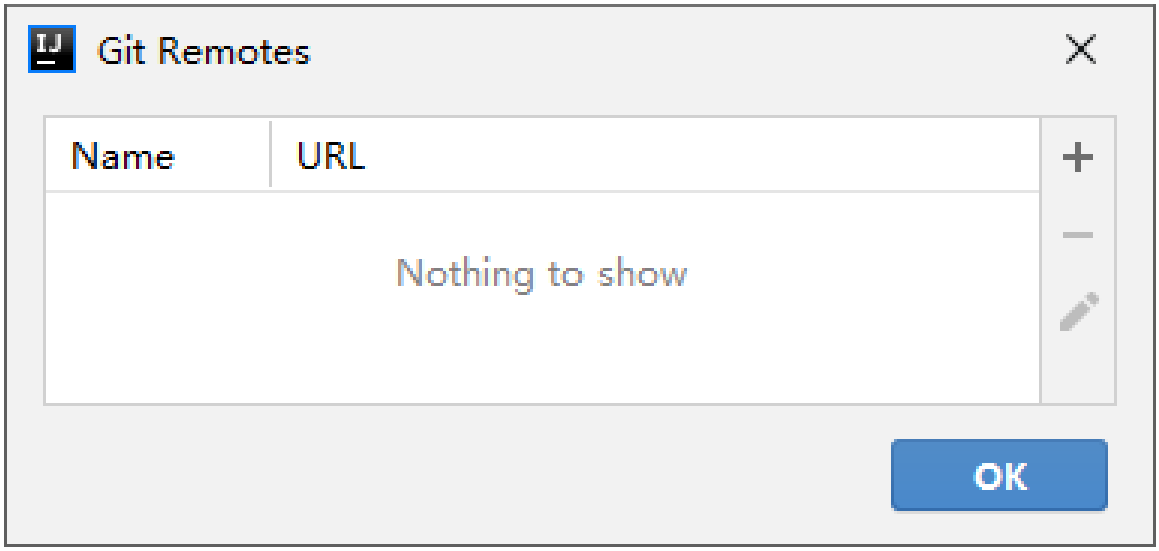
1、初始化本地库

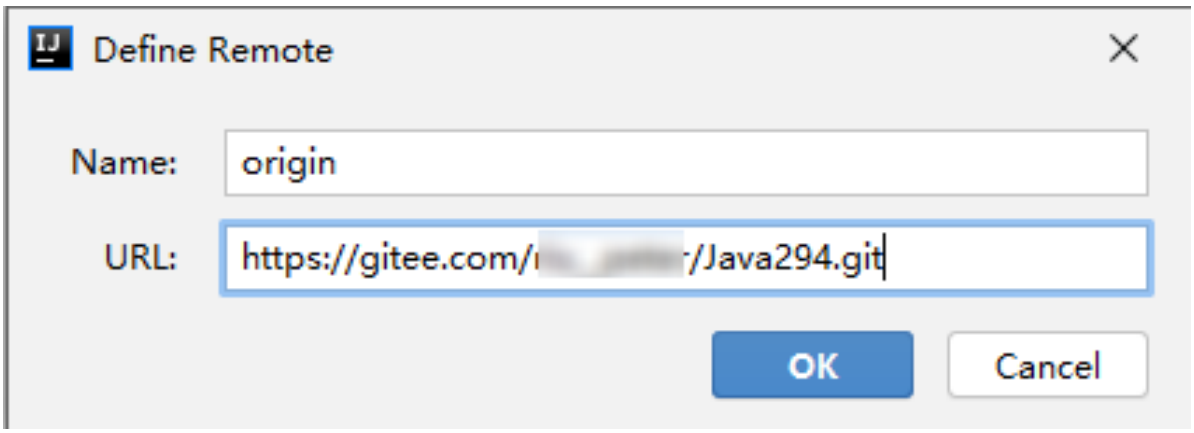


2、将项目提交到本地库（先添加再提交），注意可以使用`.gitignore`文件忽略不需要提交内容

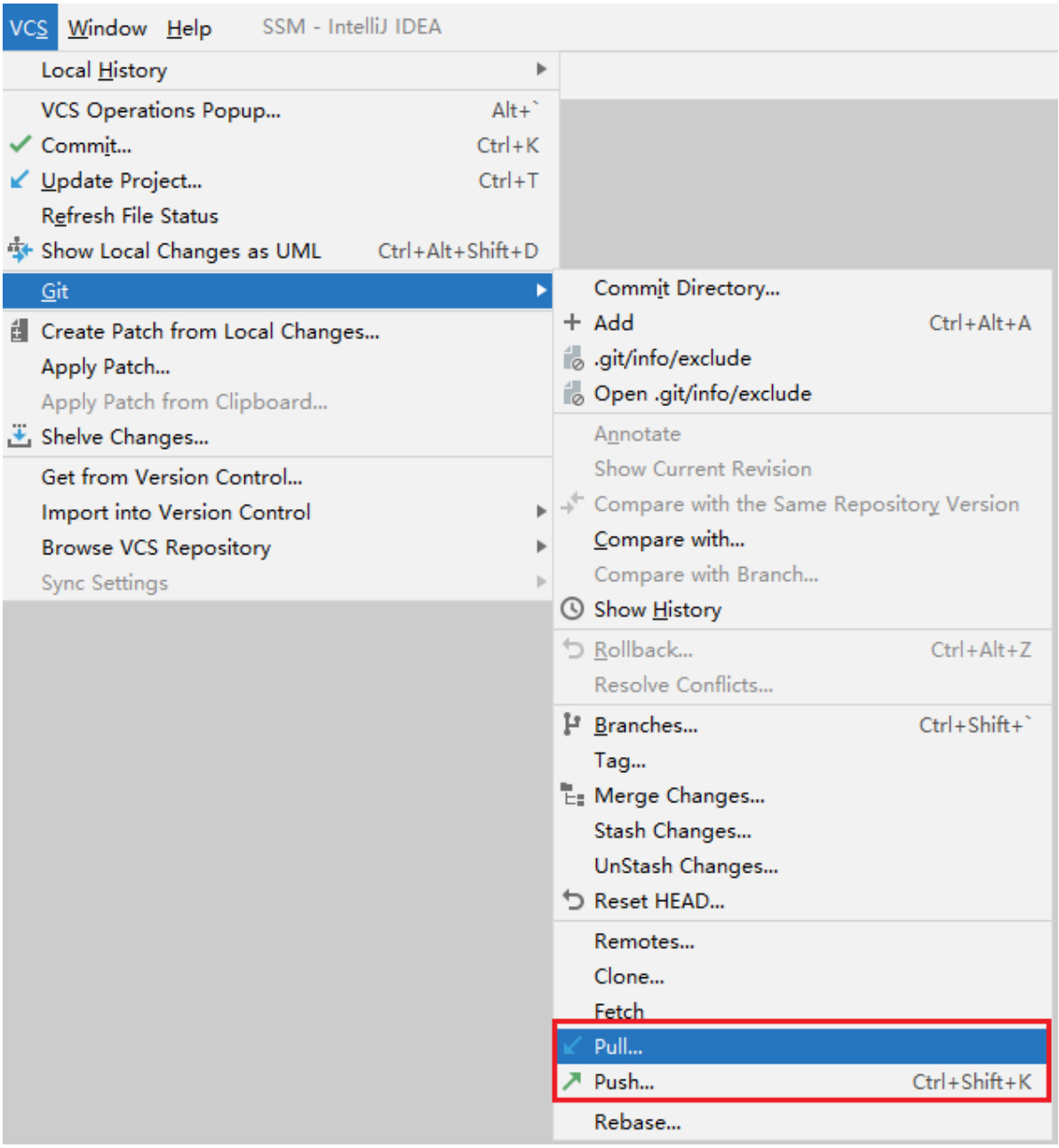
3、配置远程服务器地址


VCS Window Help SSM - IntelliJ IDEA
Local History
VCS Operations Popup... Alt+`
Commit... Ctrl+K
Update Project... Ctrl+T
Refresh File Status
Show Local Changes as UML Ctrl+Alt+Shift+D
Git
Create Patch from Local Changes...
Apply Patch...
Apply Patch from Clipboard...
Shelve Changes...
Get from Version Control...
Import into Version Control
Browse VCS Repository
Sync Settings
Commit Directory...
Add Ctrl+Alt+A
.git/info/exclude
Open .git/info/exclude
Annotate
Show Current Revision
Compare with the Same Repository Version
Compare with...
Compare with Branch...
Show History
Rollback... Ctrl+Alt+Z
Resolve Conflicts...
Branches... Ctrl+Shift+`
Tag...
Merge Changes...
Stash Changes...
UnStash Changes...
Reset HEAD...
Remotes...
Clone...
Pull...
Push... Ctrl+Shift+K
Rebase...


Git Remotes
Name
URL
Nothing to show
OK



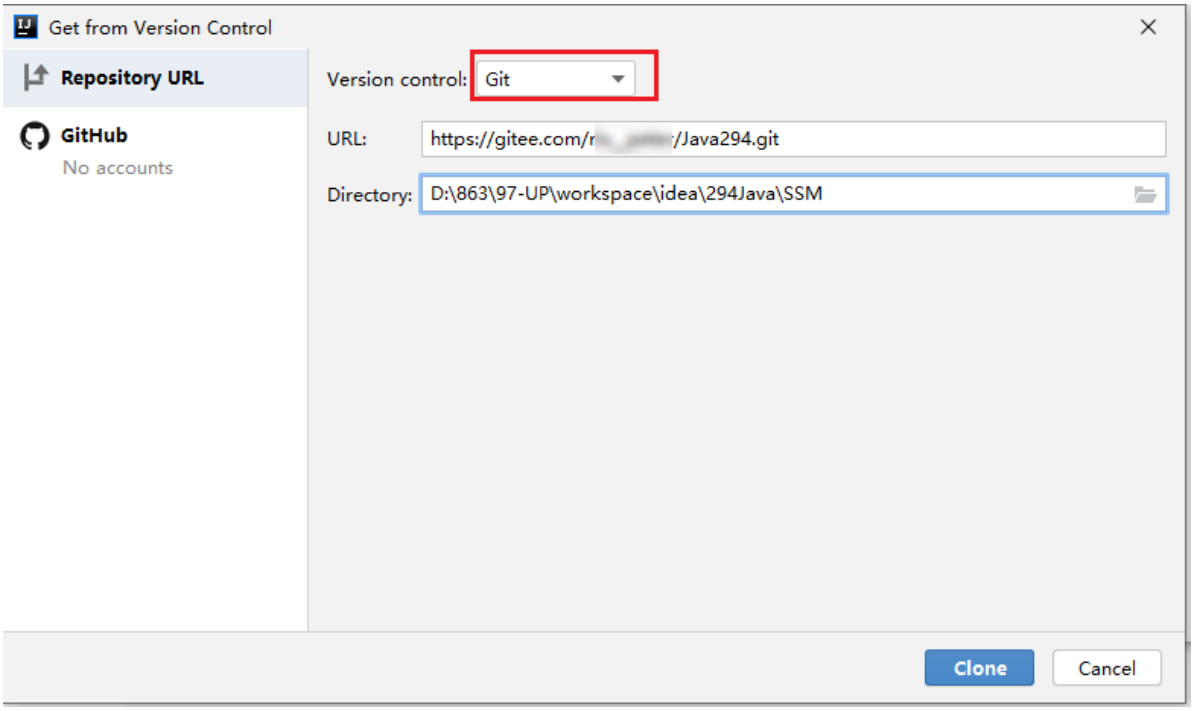
4、提交到远程服务器或者从远程服务器拉取



（二）服务器拉取项目

1、从主页面或者CVS菜单，点击【Get from Version Control】

2、选择版本控制工具【Git】，填写URL地址，选择合适目录Directory存放项目



3、等待克隆完成