

# 视频同步笔记：狂神聊Git

秦疆 狂神说 2020-03-21 12:54

狂神聊Git课程，通俗易懂，欢迎各位狂粉转发关注学习，拒绝白嫖，从我做起。

未经作者授权，禁止转载



狂神聊Git

学习git之前，我们需要先明白一个概念

**版本控制！**

版本控制

## 什么是版本控制

版本控制（Revision control）是一种在开发的过程中用于管理我们对文件、目录或工程等内容的修改历史，方便查看更改历史记录，备份以便恢复以前的版本的软件工程技术。

- 实现跨区域多人协同开发
- 追踪和记载一个或者多个文件的历史记录
- 组织和保护你的源代码和文档
- 统计工作量
- 并行开发、提高开发效率
- 跟踪记录整个软件的开发过程
- 减轻开发人员的负担，节省时间，同时降低人为错误

简单说就是用于管理多人协同开发项目的技术。

没有进行版本控制或者版本控制本身缺乏正确的流程管理，在软件开发过程中将会引入很多问题，如软件代码的一致性、软件内容的冗余、软件过程的事物性、软件开发过程中的并行性、软件源代码的安全性，以及软件的整合等问题。

无论是工作还是学习，或者是自己做笔记，都经历过这样一个阶段！我们就迫切需要一个版本控制工具！



多人开发就必须要使用版本控制！

### 常见的版本控制工具

我们学习的东西，一定是当下最流行的！

主流的版本控制器有如下这些：

- **Git**
- **SVN** (Subversion)
- **CVS** (Concurrent Versions System)

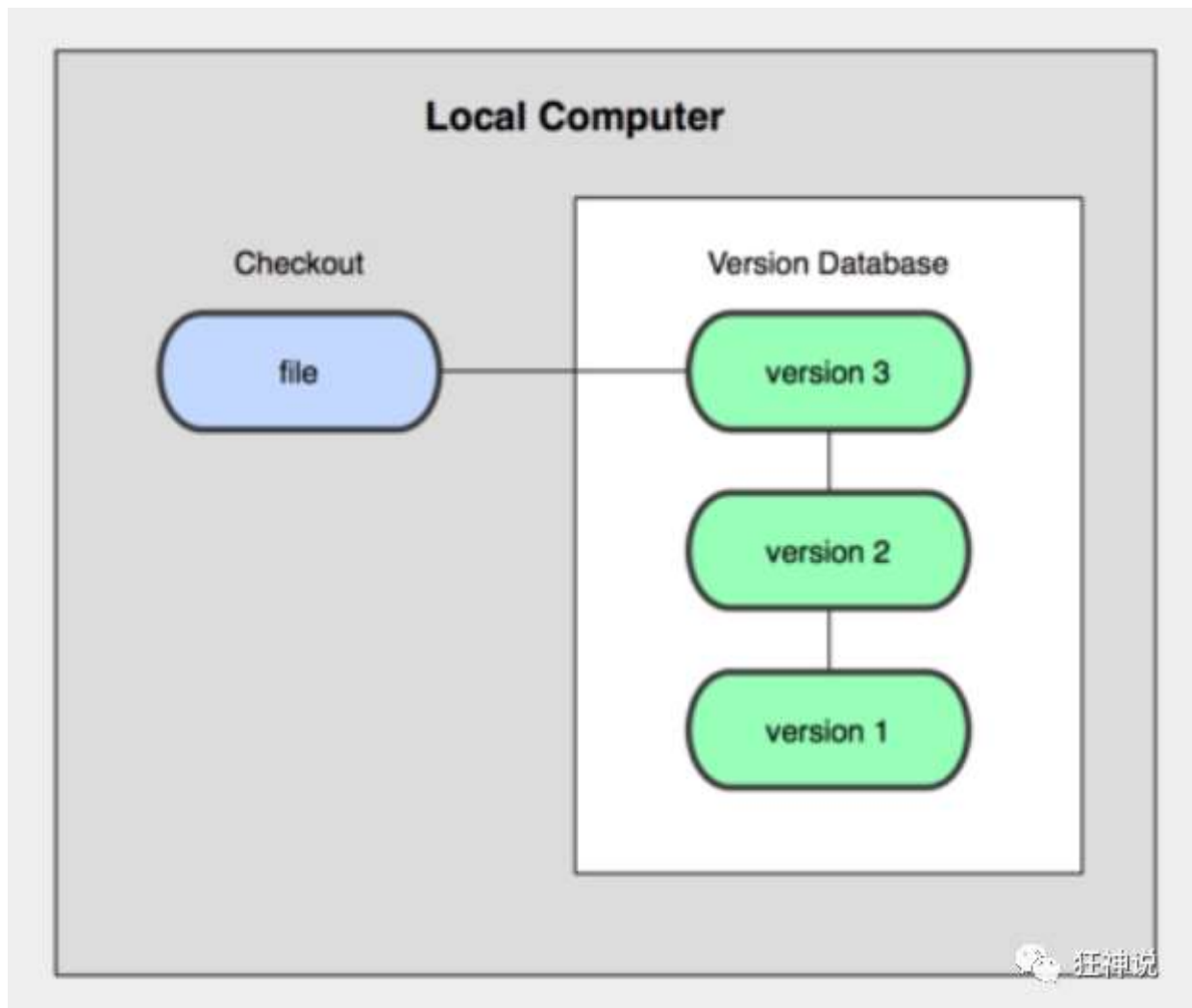
- **VSS** (Micorosoft Visual SourceSafe)
- **TFS** (Team Foundation Server)
- Visual Studio Online

版本控制产品非常的多（Perforce、Rational ClearCase、RCS（GNU Revision Control System）、Serena Dimention、SVK、BitKeeper、Monotone、Bazaar、Mercurial、SourceGear Vault），现在影响力最大且使用最广泛的是Git与SVN

## 版本控制分类

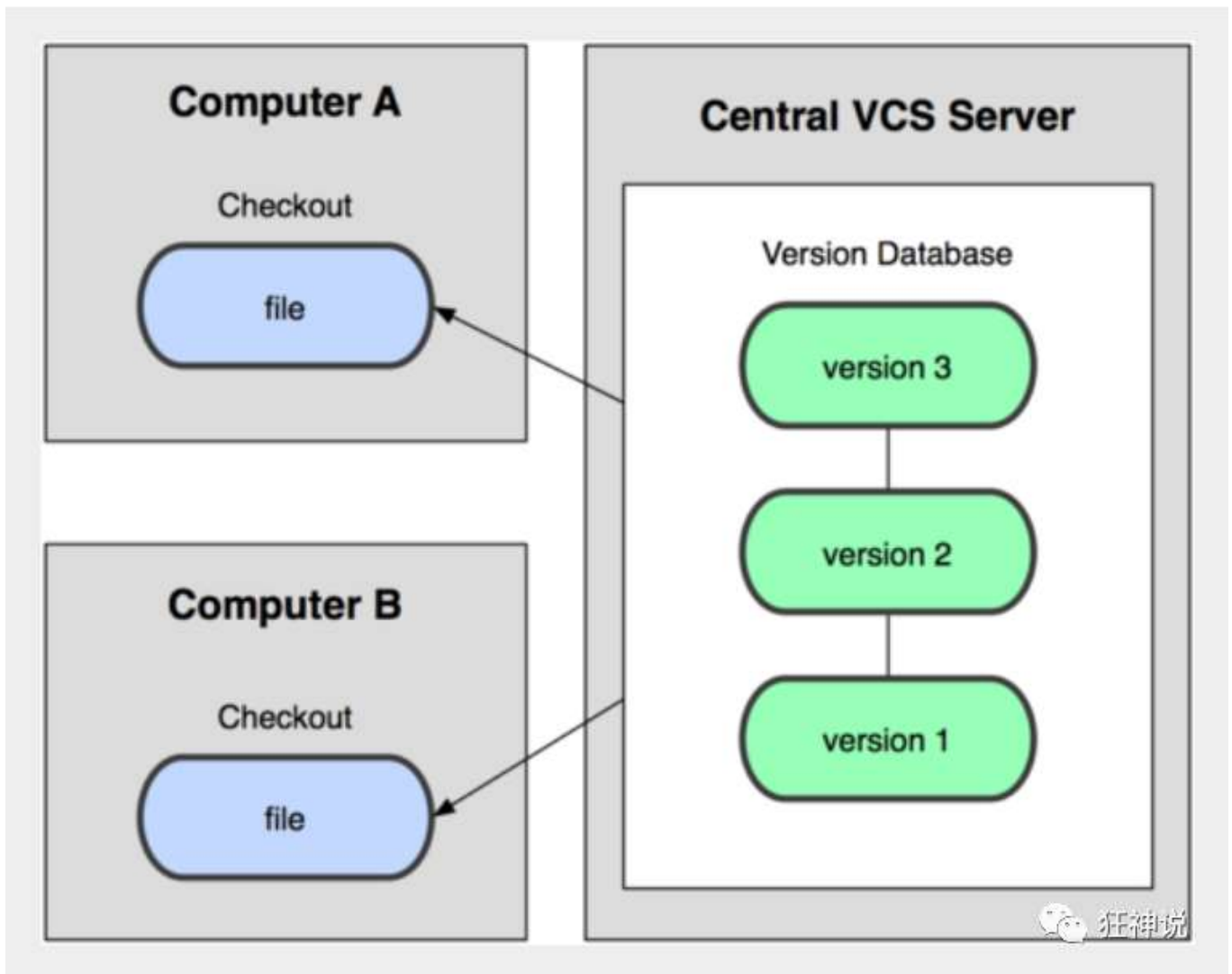
### 1、本地版本控制

记录文件每次的更新，可以对每个版本做一个快照，或是记录补丁文件，适合个人用，如RCS。



### 2、集中版本控制 SVN

所有的版本数据都保存在服务器上，协同开发者从服务器上同步更新或上传自己的修改



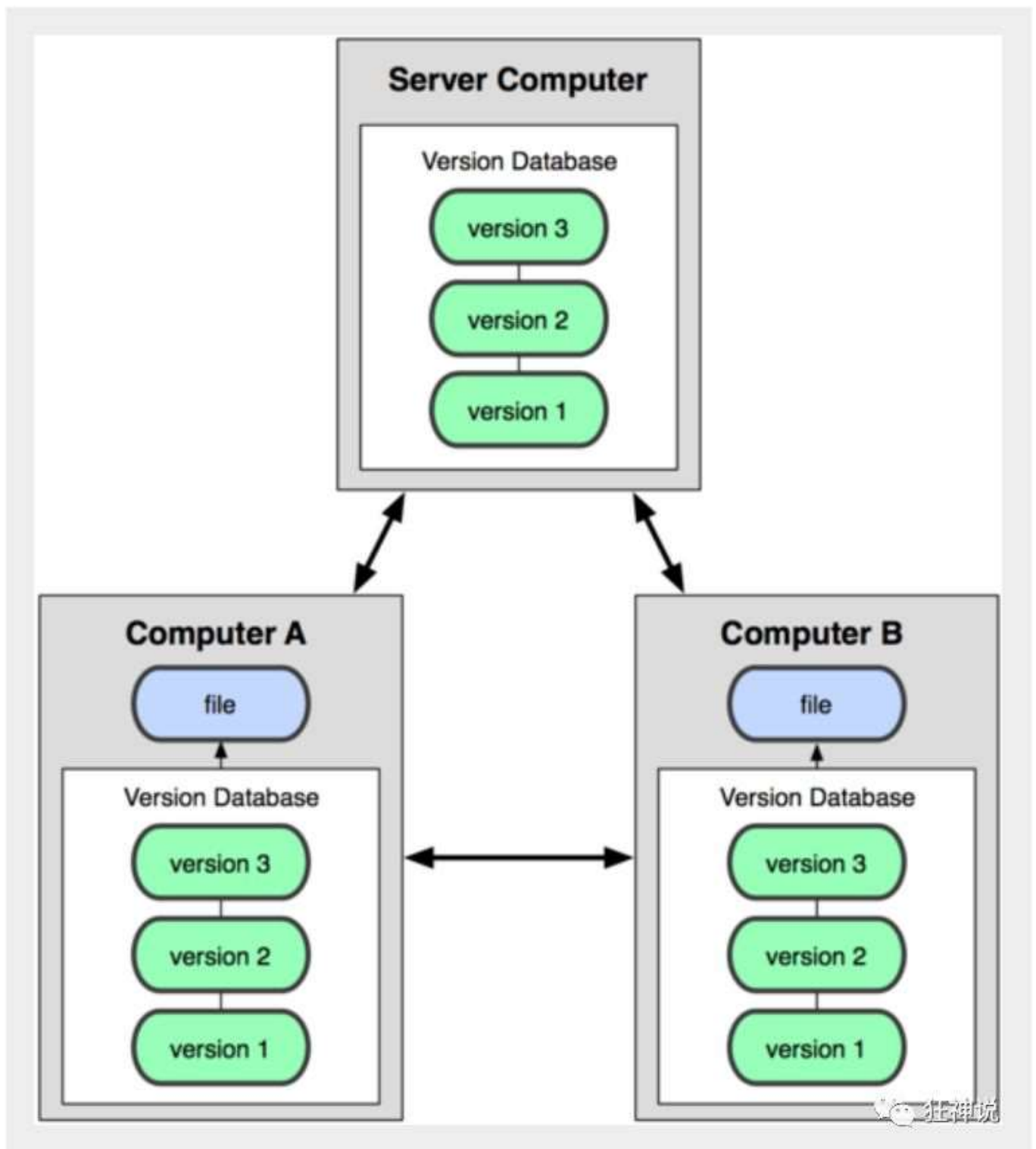
所有的版本数据都存在服务器上，用户的本地只有自己以前所同步的版本，如果不连网的话，用户就看不到历史版本，也无法切换版本验证问题，或在不同分支工作。而且，所有数据都保存在单一的服务器上，有很大的风险这个服务器会损坏，这样就会丢失所有的数据，当然可以定期备份。代表产品：SVN、CVS、VSS

### 3、分布式版本控制      Git

每个人都拥有全部的代码！安全隐患！

所有版本信息仓库全部同步到本地的每个用户，这样就可以在本地查看所有版本历史，可以离线在本地提交，只需在连网时push到相应的服务器或其他用户那里。由于每个用户那里保存的都是所有的版本数据，只要有一个用户的设备没有问题就可以恢复所有的数据，但这增加了本地存储空间的占用。

不会因为服务器损坏或者网络问题，造成不能工作的情况！



## Git与SVN的主要区别

SVN是集中式版本控制系统，版本库是集中放在中央服务器的，而工作的时候，用的都是自己的电脑，所以首先要从中央服务器得到最新的版本，然后工作，完成工作后，需要把自己做完的活推送到中央服务器。集中式版本控制系统是必须联网才能工作，对网络带宽要求较高。



Git是分布式版本控制系统，没有中央服务器，每个人的电脑就是一个完整的版本库，工作的时候不需要联网了，因为版本都在自己电脑上。协同的方法是这样的：比如说自己在电脑上改了文件A，其他人也在电脑上改了文件A，这时，你们两之间只需把各自的修改推送给对方，就可以互相看到对方的修改了。Git可以直接看到更新了哪些代码和文件！

**Git是目前世界上最先进的分布式版本控制系统。**

## 聊聊Git的历史

同生活中的许多伟大事物一样，Git 诞生于一个极富纷争大举创新的年代。

Linux 内核开源项目有着为数众广的参与者。绝大多数的 Linux 内核维护工作都花在了提交补丁和保存归档的繁琐事务上(1991 - 2002年间)。到 2002 年，整个项目组开始启用一个专有的分布式版本控制系统 BitKeeper 来管理和维护代码。

Linux社区中存在很多的大佬！破解研究 BitKeeper ！

到了 2005 年，开发 BitKeeper 的商业公司同 Linux 内核开源社区的合作关系结束，他们收回了 Linux 内核社区免费使用 BitKeeper 的权力。这就迫使 Linux 开源社区(特别是 Linux 的缔造者 Linus Torvalds)基于使用 BitKeeper 时的经验教训，开发出自己的版本系统。（2周左右！）也就是后来的 Git！

**Git是目前世界上最先进的分布式版本控制系统。**

Git是免费、开源的，最初Git是为辅助 Linux 内核开发的，来替代 BitKeeper！



Linux和Git之父李纳斯·托沃兹（Linus Benedic Torvalds）1969、芬兰



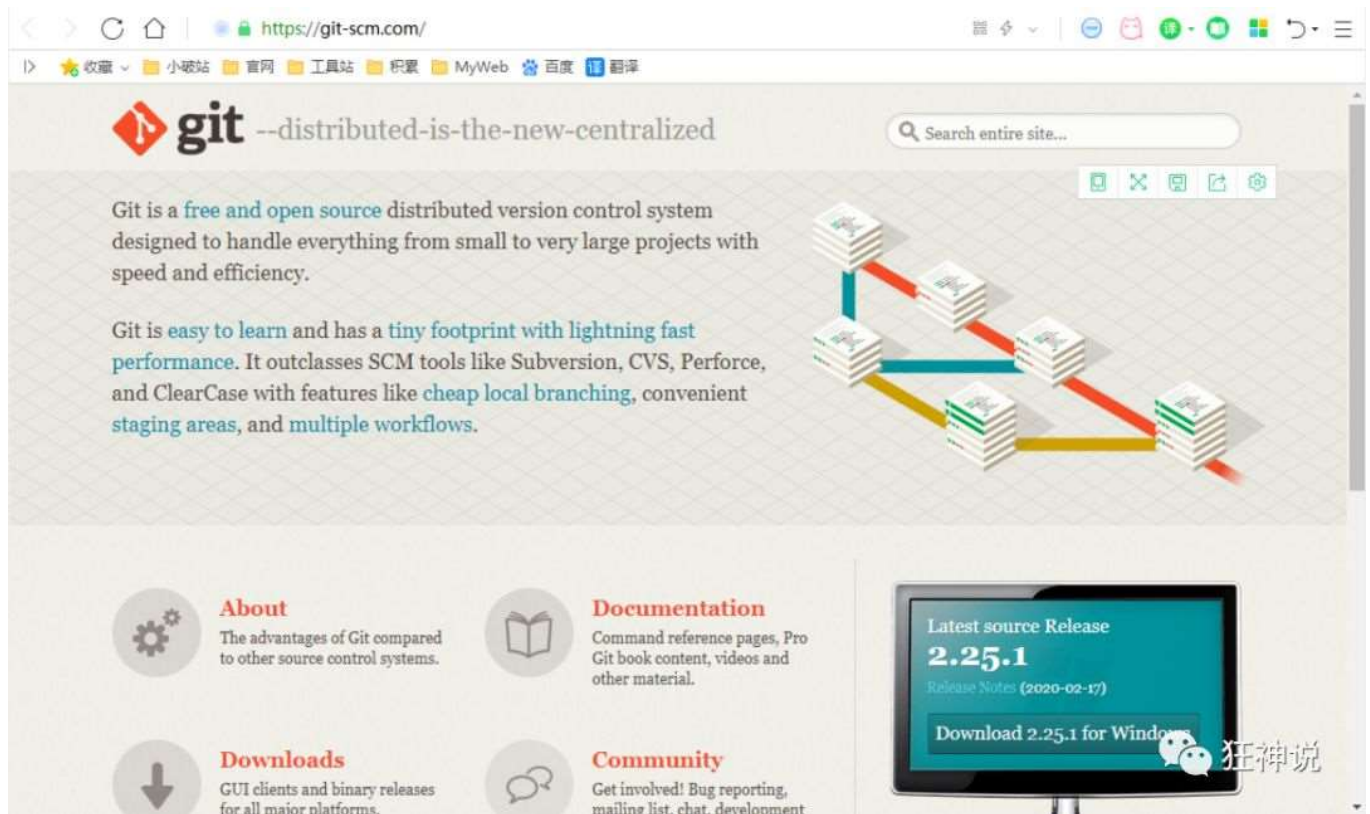
## Git环境配置

### 软件下载

打开 [git官网] <https://git-scm.com/>，下载git对应操作系统的版本。

所有东西下载慢的话就可以去找镜像！

官网下载太慢，我们可以使用淘宝镜像下载：<http://npm.taobao.org/mirrors/git-for-windows/>

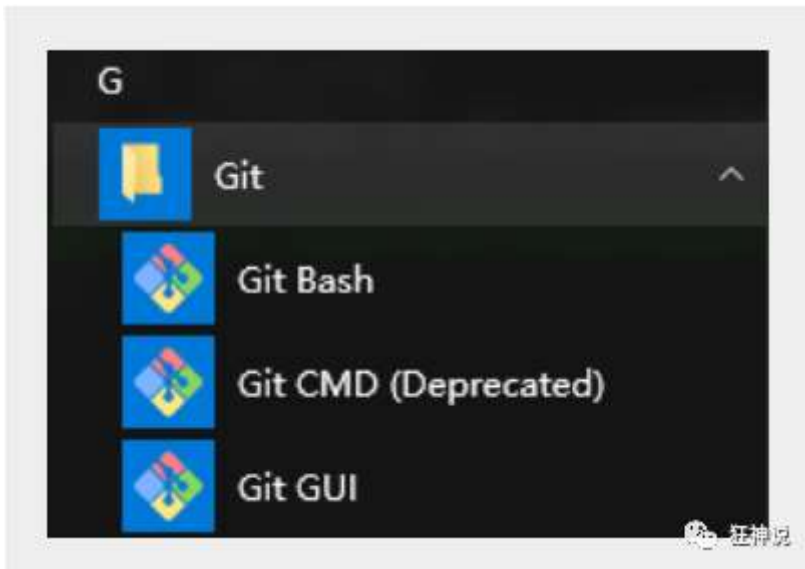


下载对应的版本即可安装！

安装：无脑下一步即可！安装完毕就可以使用了！

### 启动Git

安装成功后在开始菜单中会有Git项，菜单下有3个程序：任意文件夹下右键也可以看到对应的程序！



**Git Bash：** Unix与Linux风格的命令行，使用最多，推荐最多

**Git CMD：** Windows风格的命令行

**Git GUI：** 图形界面的Git，不建议初学者使用，尽量先熟悉常用命令

## 常用的Linux命令

平时一定要多使用这些基础的命令！

- 1)、cd：改变目录。
- 2)、cd .. 回退到上一个目录，直接cd进入默认目录
- 3)、pwd：显示当前所在的目录路径。
- 4)、ls(ll)：都是列出当前目录中的所有文件，只不过ll(两个l)列出的内容更为详细。
- 5)、touch：新建一个文件 如 touch index.js 就会在当前目录下新建一个index.js文件。
- 6)、rm：删除一个文件, rm index.js 就会把index.js文件删除。
- 7)、mkdir：新建一个目录,就是新建一个文件夹。
- 8)、rm -r：删除一个文件夹, rm -r src 删除src目录

rm -rf / 切勿在Linux中尝试！删除电脑中全部文件！

9)、mv 移动文件, mv index.html src index.html 是我们要移动的文件, src 是目标文件夹,当然, 这样写,必须保证文件和目标文件夹在同一目录下。

10)、reset 重新初始化终端/清屏。

11)、clear 清屏。



12)、history 查看命令历史。

13)、help 帮助。

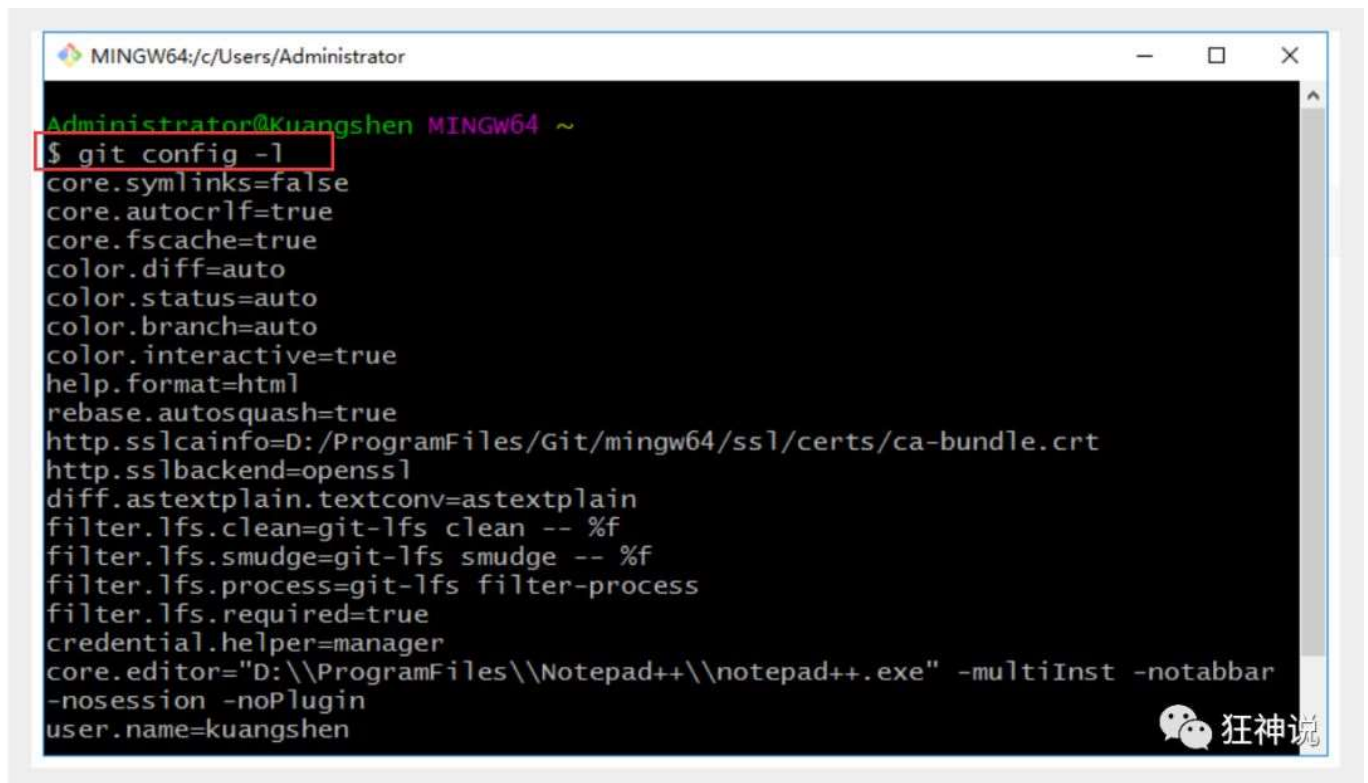
14)、exit 退出。

15)、#表示注释

## Git配置

所有的配置文件，其实都保存在本地！

查看配置 `git config -l`



```
Administrator@Kuangshen MINGW64 ~
$ git config -l
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
rebase.autosquash=true
http.sslcainfo=D:/ProgramFiles/Git/mingw64/ssl/certs/ca-bundle.crt
http.sslbackend=openssl
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
credential.helper=manager
core.editor="D:\\ProgramFiles\\Notepad++\\notepad++.exe" -multiInst -notabbar
-nosession -noPlugin
user.name=kuangshen
```

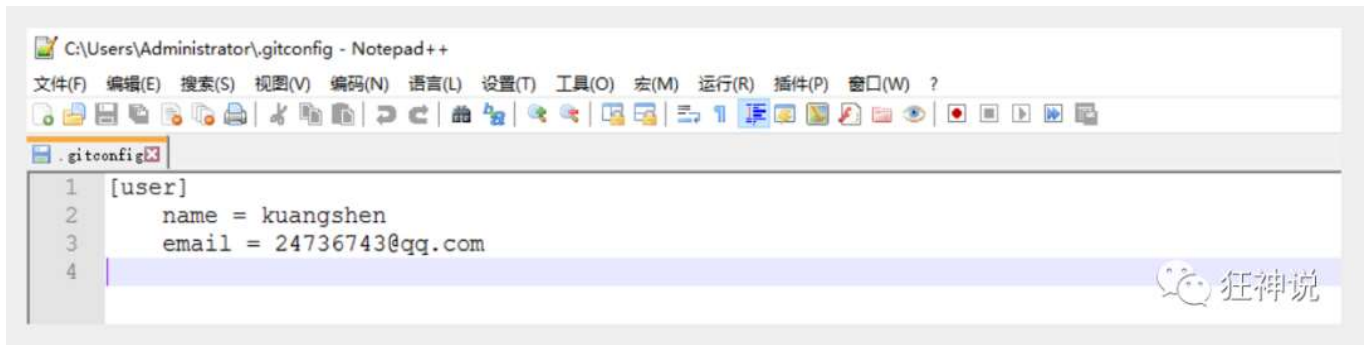
查看不同级别的配置文件：

```
1 #查看系统config
2 git config --system --list
3
4 #查看当前用户（global）配置
5 git config --global --list
```

**Git相关的配置文件：**

1)、Git\etc\gitconfig ： Git 安装目录下的 gitconfig --system 系统级

2)、C:\Users\Administrator\.gitconfig 只适用于当前登录用户的配置 --global 全局



这里可以直接编辑配置文件，通过命令设置后会响应到这里。

## 设置用户名与邮箱（用户标识，必要）

当你安装Git后首先要做的事情是设置你的用户名称和e-mail地址。这是非常重要的，因为每次Git提交都会使用该信息。它被永远的嵌入到了你的提交中：

```
1 git config --global user.name "kuangshen" #名称
2 git config --global user.email 24736743@qq.com #邮箱
```

只需要做一次这个设置，如果你传递了--global 选项，因为Git将总是会使用该信息来处理你在系统中所做的一切操作。如果你希望在一个特定的项目中使用不同的名称或e-mail地址，你可以在该项目中运行该命令而不要--global选项。总之--global为全局配置，不加为某个项目的特定配置。

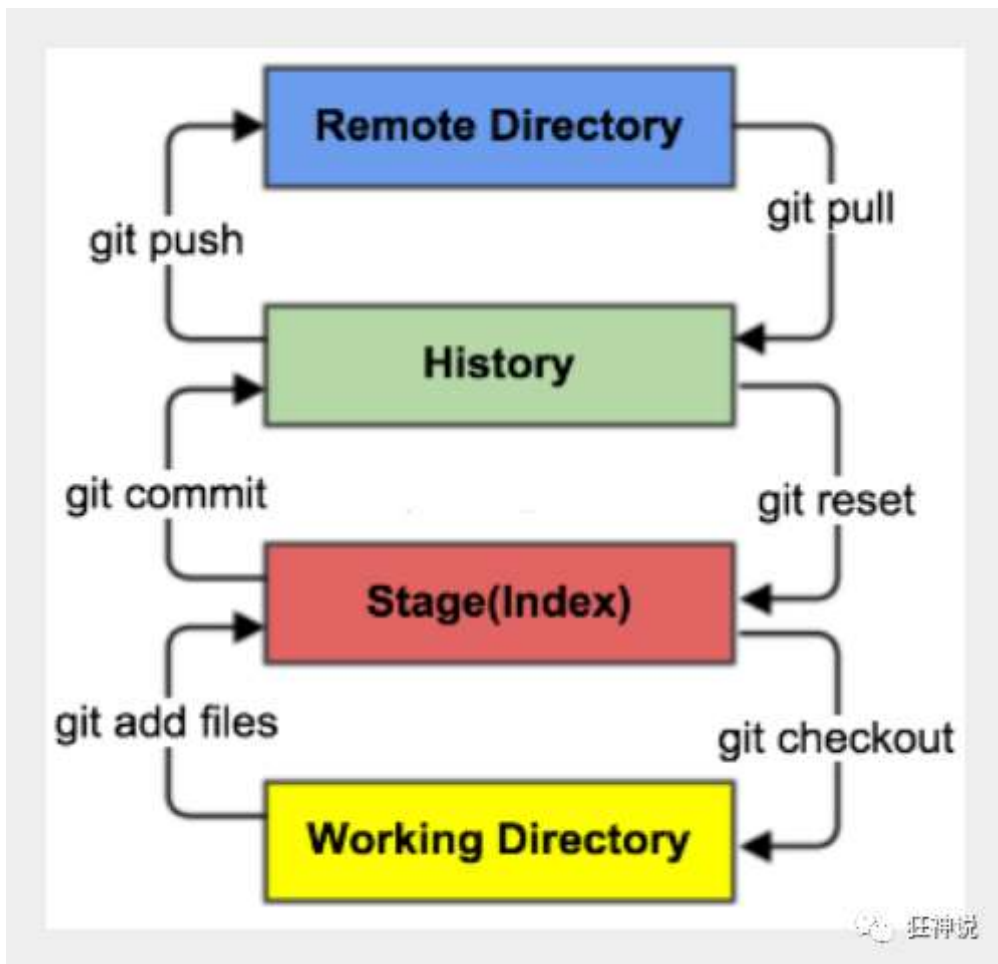
```
Administrator@kuangshen MINGW64 ~/Desktop/狂神说Git/gitcode
$ git config --global --list
user.name=kuangshen
user.email=24736743@qq.com
```

狂神说

## Git基本理论（重要）

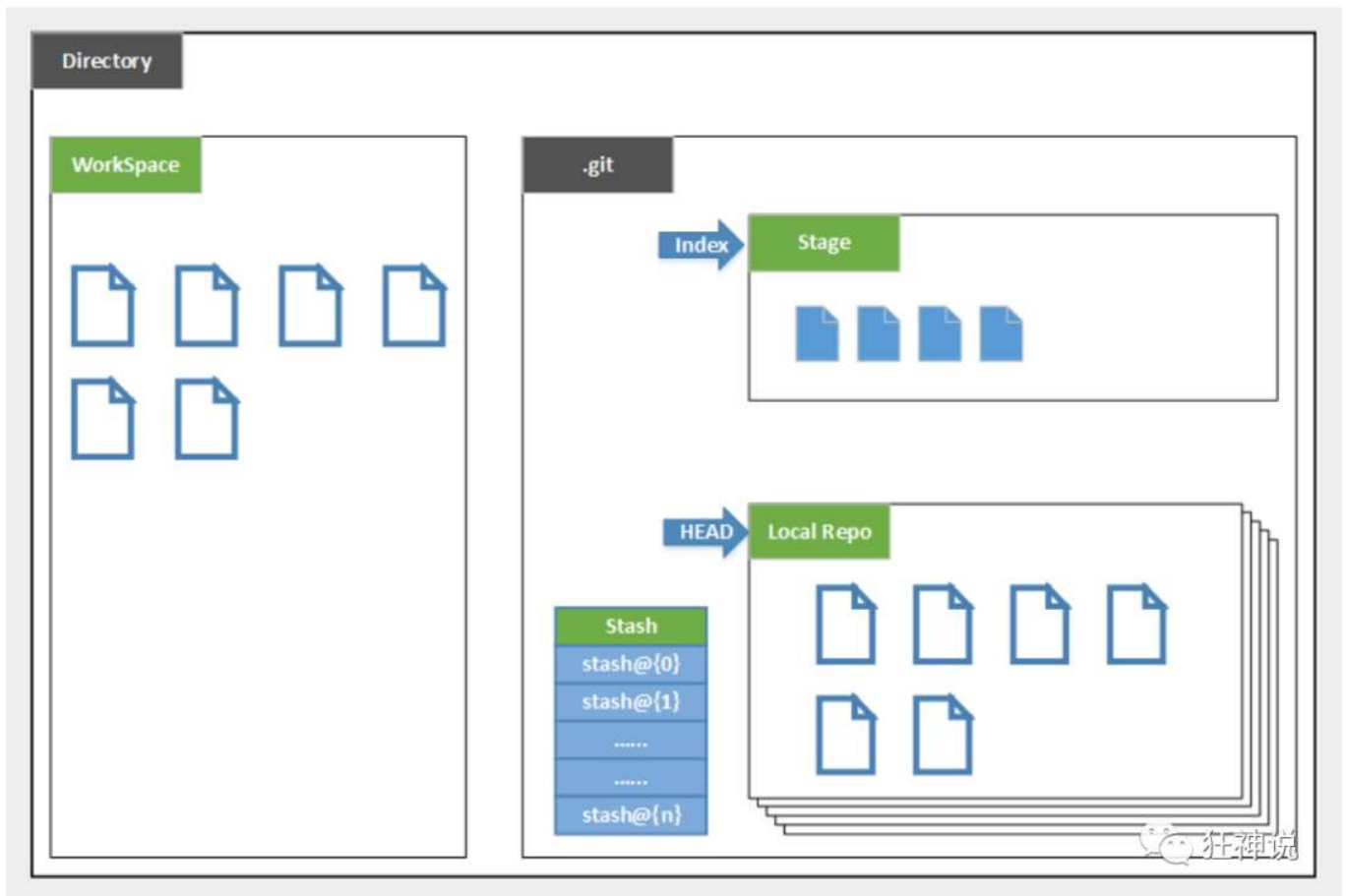
### 三个区域

Git本地有三个工作区域：工作目录（Working Directory）、暂存区(Stage/Index)、资源库(Repository或Git Directory)。如果在加上远程的git仓库(Remote Directory)就可以分为四个工作区域。文件在这四个区域之间的转换关系如下：



- **Workspace**: 工作区，就是你平时存放项目代码的地方
- **Index / Stage**: 暂存区，用于临时存放你的改动，事实上它只是一个文件，保存即将提交到文件列表信息
- **Repository**: 仓库区（或本地仓库），就是安全存放数据的位置，这里面有你提交到所有版本的数据。其中HEAD指向最新放入仓库的版本
- **Remote**: 远程仓库，托管代码的服务器，可以简单的认为是你项目组中的一台电脑用于远程数据交换

本地的三个区域确切的说是应该是git仓库中HEAD指向的版本：



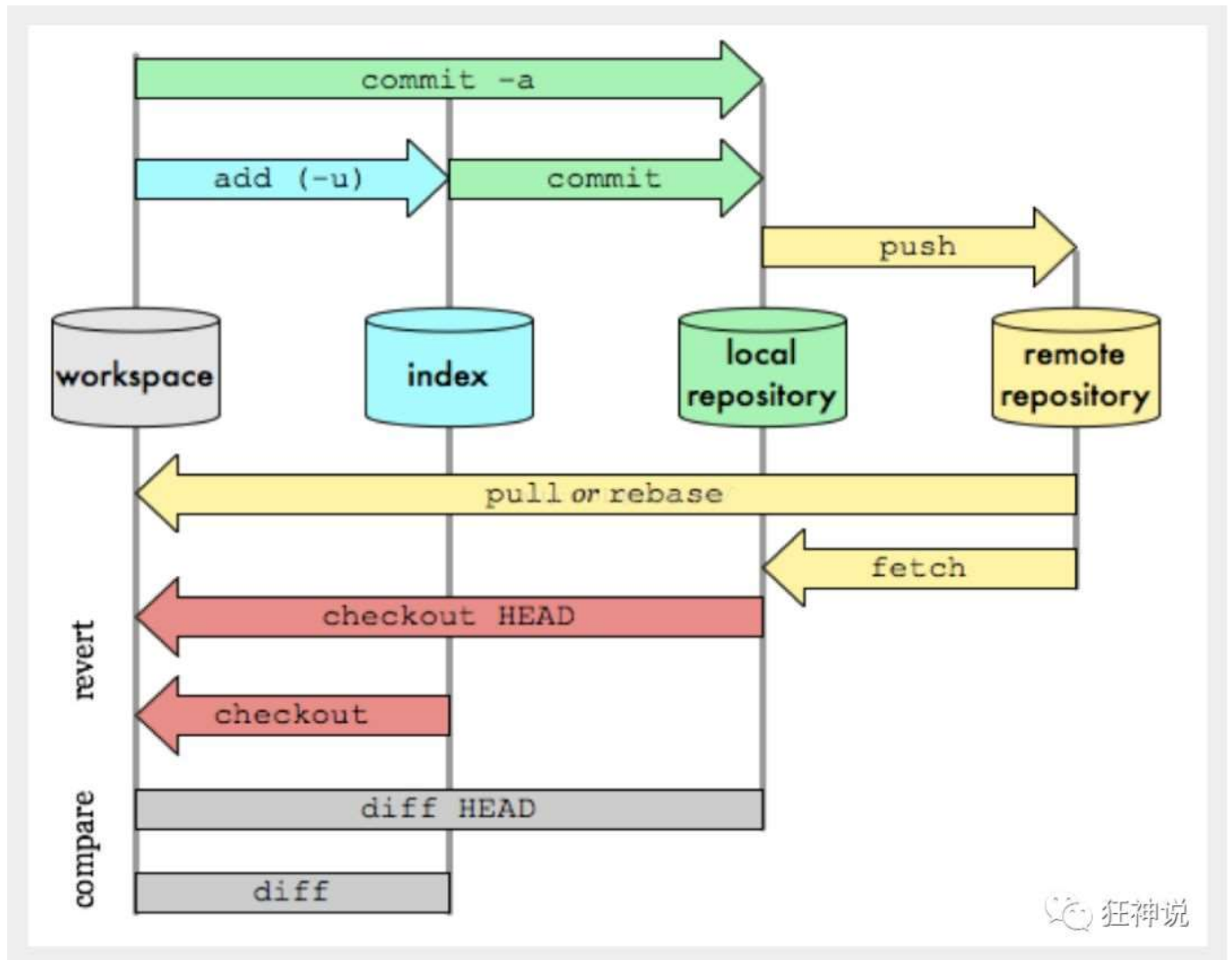
- Directory：使用Git管理的一个目录，也就是一个仓库，包含我们的工作空间和Git的管理空间。
- WorkSpace：需要通过Git进行版本控制的目录和文件，这些目录和文件组成了工作空间。
- .git：存放Git管理信息的目录，初始化仓库的时候自动创建。
- Index/Stage：暂存区，或者叫待提交更新区，在提交进入repo之前，我们可以把所有的更新放在暂存区。
- Local Repo：本地仓库，一个存放在本地的版本库；HEAD会只是当前的开发分支（branch）。
- Stash：隐藏，是一个工作状态保存栈，用于保存/恢复WorkSpace中的临时状态。

## 工作流程

git的工作流程一般是这样的：

- 1、在工作目录中添加、修改文件；
- 2、将需要进行版本管理的文件放入暂存区域；
- 3、将暂存区域的文件提交到git仓库。

因此, git管理的文件有三种状态: 已修改 (modified) ,已暂存 (staged) ,已提交 (committed)

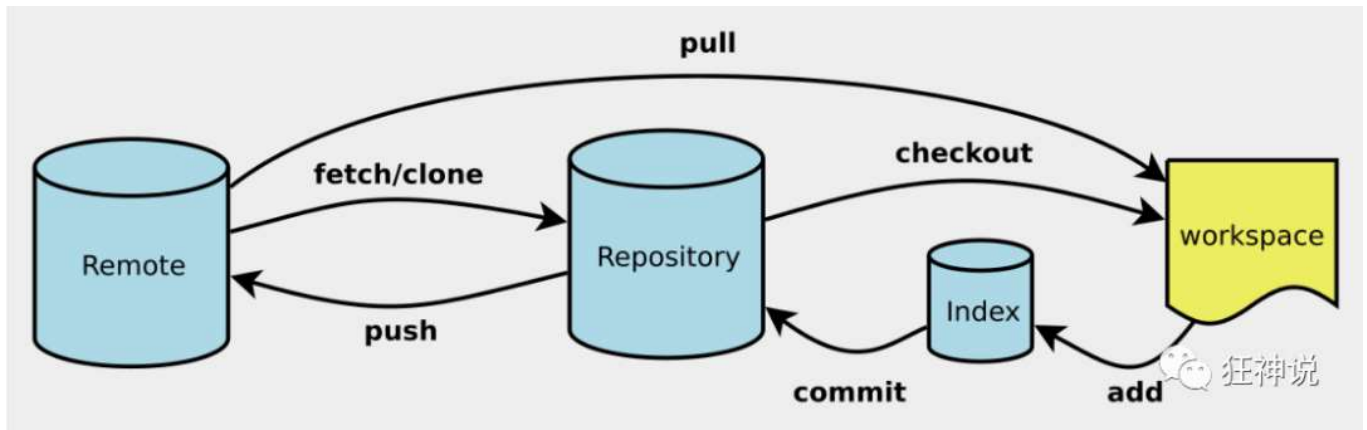


## Git项目搭建

### 创建工作目录与常用指令

工作目录 (WorkSpace)一般就是你希望Git帮助你管理的文件夹, 可以是你项目的目录, 也可以是一个空目录, 建议不要有中文。

日常使用只要记住下图6个命令:



## 本地仓库搭建

创建本地仓库的方法有两种：一种是创建全新的仓库，另一种是克隆远程仓库。

1、创建全新的仓库，需要用GIT管理的项目的根目录执行：

```
1 # 在当前目录新建一个Git代码库
2 $ git init
```

2、执行后可以看到，仅仅在项目目录多出了一个.git目录，关于版本等的所有信息都在这个目录里面。

## 克隆远程仓库

1、另一种方式是克隆远程目录，由于是将远程服务器上的仓库完全镜像一份至本地！

```
1 # 克隆一个项目和它的整个代码历史(版本信息)
2 $ git clone [url] # https://gitee.com/kuangstudy/openClass.git
```

2、去 gitee 或者 github 上克隆一个测试！

## Git文件操作

### 文件的四种状态

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，首先要知道文件当前在什么状态，不然可能会提交了现在还不想提交的文件，或者要提交的文件没提交上。

- **Untracked**: 未跟踪，此文件在文件夹中，但并没有加入到git库，不参与版本控制。通过git add 状态变为Staged.
- **Unmodify**: 文件已经入库，未修改，即版本库中的文件快照内容与文件夹中完全一致。这种类型的文件有两种去处，如果它被修改，而变为Modified. 如果使用git rm移出版本库，则成为Untracked文件



- Modified: 文件已修改, 仅仅是修改, 并没有进行其他的操作. 这个文件也有两个去处, 通过 `git add` 可进入暂存 `staged` 状态, 使用 `git checkout` 则丢弃修改过, 返回到 `unmodify` 状态, 这个 `git checkout` 即从库中取出文件, 覆盖当前修改!
- Staged: 暂存状态. 执行 `git commit` 则将修改同步到库中, 这时库中的文件和本地文件又变为一致, 文件为 `Unmodify` 状态. 执行 `git reset HEAD filename` 取消暂存, 文件状态为 `Modified`

## 查看文件状态

上面说文件有4种状态, 通过如下命令可以查看到文件的状态:

```
1 #查看指定文件状态
2 git status [filename]
3
4 #查看所有文件状态
5 git status
6
7 # git add .                添加所有文件到暂存区
8 # git commit -m "消息内容" 提交暂存区中的内容到本地仓库 -m 提交信息
```

## 忽略文件

有些时候我们不想把某些文件纳入版本控制中, 比如数据库文件, 临时文件, 设计文件等  
在主目录下建立 ".gitignore" 文件, 此文件有如下规则:

1. 忽略文件中的空行或以井号 (#) 开始的行将会被忽略。
2. 可以使用Linux通配符。例如: 星号 (\*) 代表任意多个字符, 问号 (?) 代表一个字符, 方括号 ([abc]) 代表可选字符范围, 大括号 ({string1,string2,...}) 代表可选的字符串等。
3. 如果名称的最前面有一个感叹号 (!), 表示例外规则, 将不被忽略。
4. 如果名称的最前面是一个路径分隔符 (/), 表示要忽略的文件在此目录下, 而子目录中的文件不忽略。
5. 如果名称的最后面是一个路径分隔符 (/), 表示要忽略的是此目录下该名称的子目录, 而非文件 (默认文件或目录都忽略)。

```
1 #为注释
2 *.txt          #忽略所有 .txt结尾的文件, 这样的话上传就不会被选中!
3 !lib.txt       #但lib.txt除外
4 /temp          #仅忽略项目根目录下的TODO文件, 不包括其它目录temp
```

```
5 build/          #忽略build/目录下的所有文件
6 doc/*.txt       #会忽略 doc/notes.txt 但不包括 doc/server/arch.txt
```

## 使用码云

github 是有墙的，比较慢，在国内的话，我们一般使用 gitee，公司中有时会搭建自己的gitlab服务器

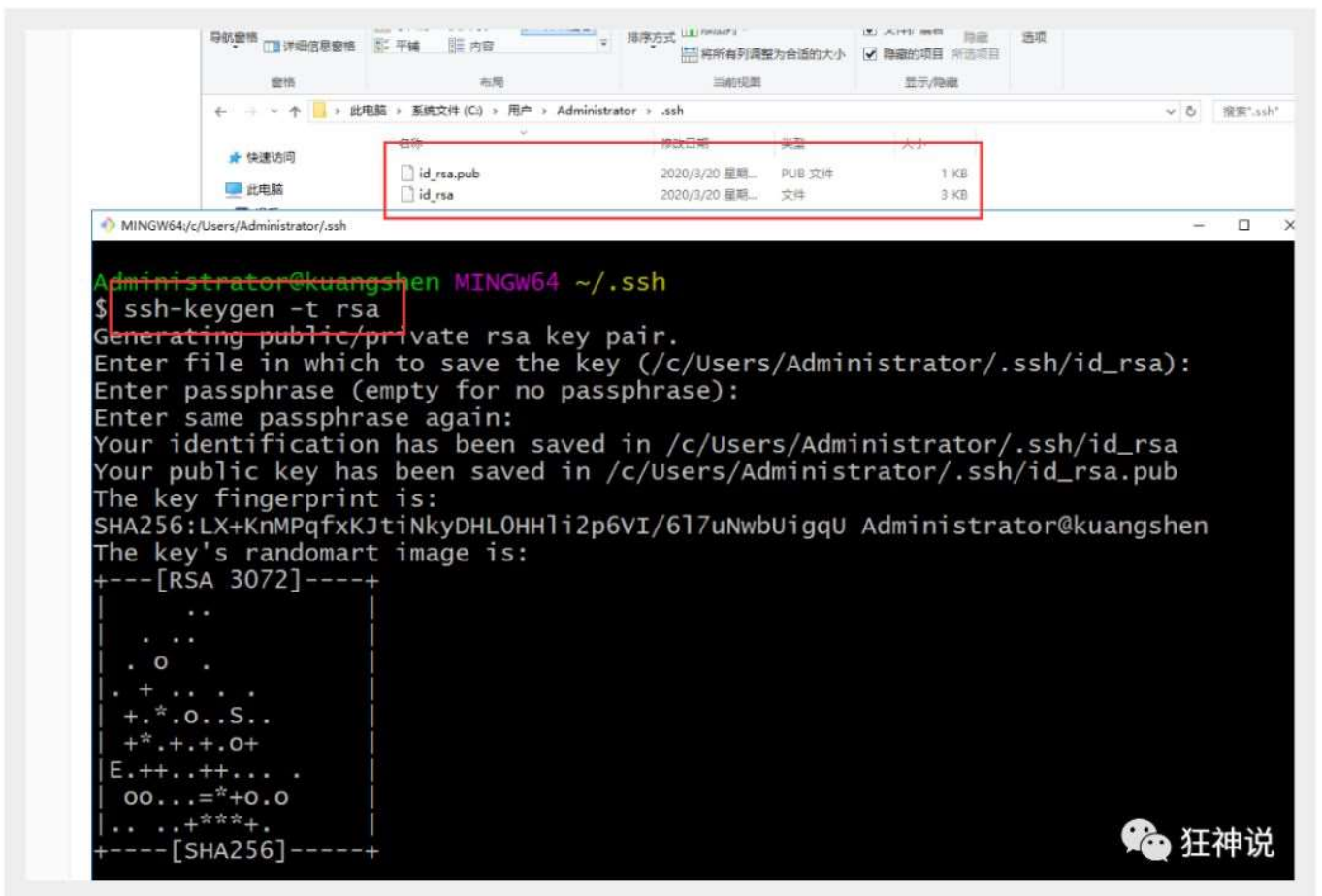
这个其实可以作为大家未来找工作的重要信息！

### 1、注册登录码云，完善个人信息



### 2、设置本机绑定SSH公钥，实现免密码登录！（免密码登录，这一步挺重要的，码云是远程仓库，我们是平时工作在本地仓库！）

```
1 # 进入 C:\Users\Administrator\.ssh 目录
2 # 生成公钥
3 ssh-keygen
```



3、将公钥信息public key 添加到码云账户中即可！



4、使用码云创建一个自己的仓库！



许可证：开源是否可以随意转载，开源但是不能商业使用，不能转载，... 限制！

仓库名称

gitstudy

归属

狂神说

/

gitstudy

仓库地址：https://gitee.com/kuangstudy/gitstudy

仓库介绍 非必填

狂神聊Git，一定记得三连投币，然后关注公众号：狂神说

是否开源

☐ 私有

☒ 公开

任何人都可以访问该仓库的代码和其他任何形式的资源

选择语言

添加 .gitignore

添加开源许可证

Java

Java

GPL-3.0

☒ 使用Readme文件初始化这个仓库

☐ 使用Issue模板文件初始化这个仓库

☐ 使用Pull Request模板文件初始化这个仓库

选择分支模型 (仓库初始化后将根据所选分支模型创建分支)

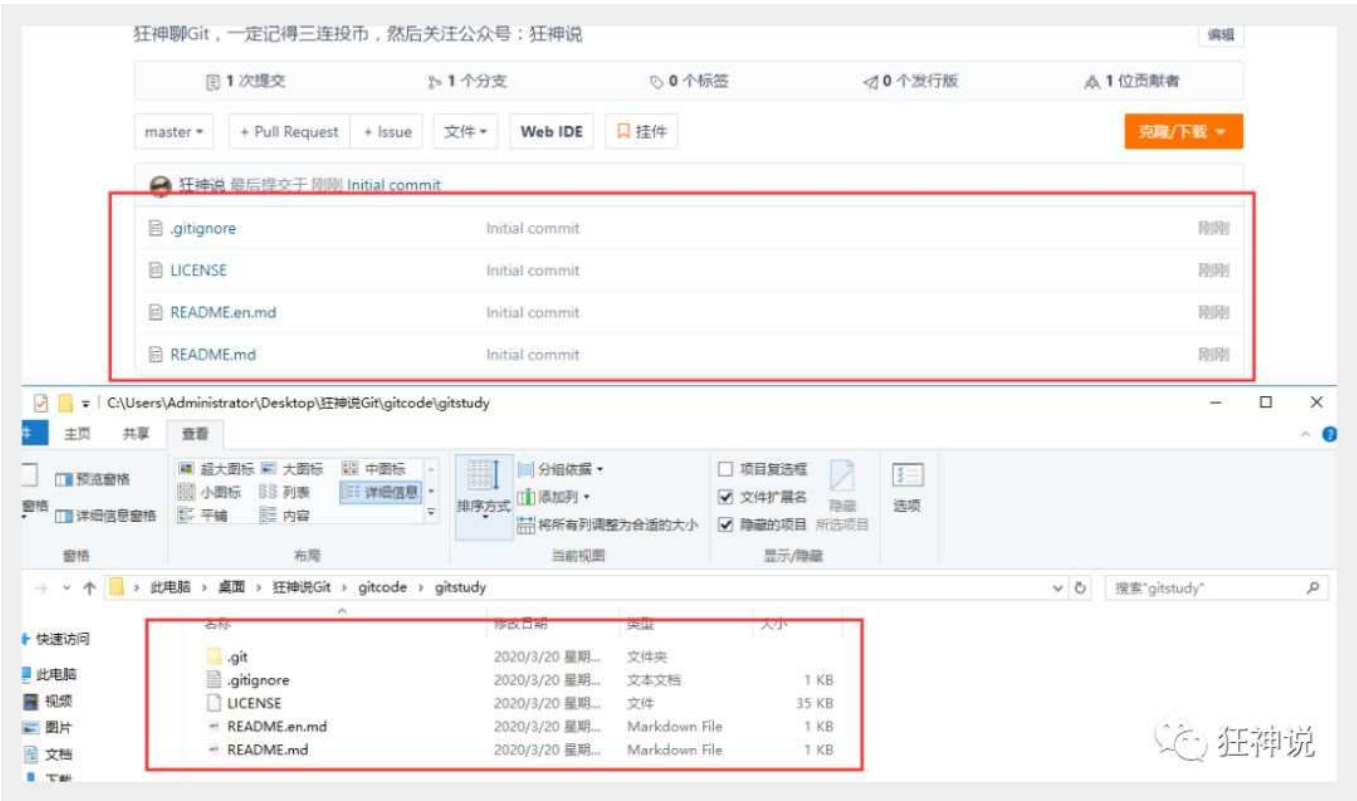
单分支模型 (只创建 master 分支)

导入已有仓库

创建

狂神说

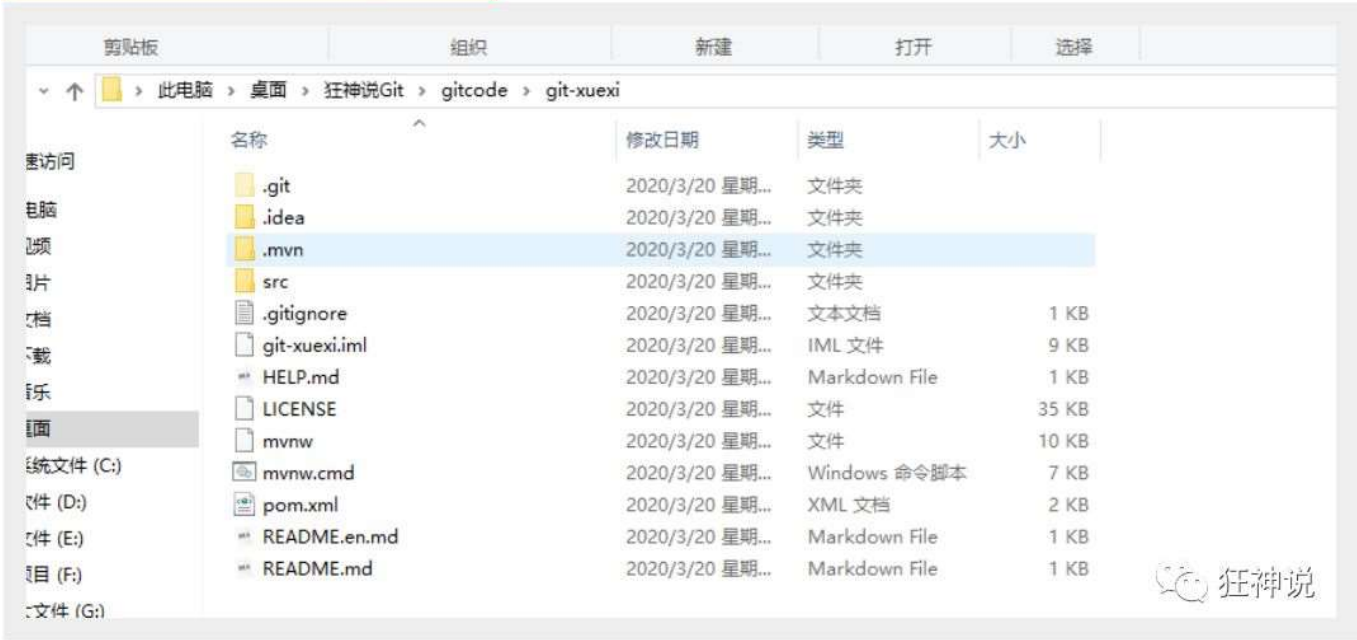
克隆到本地！



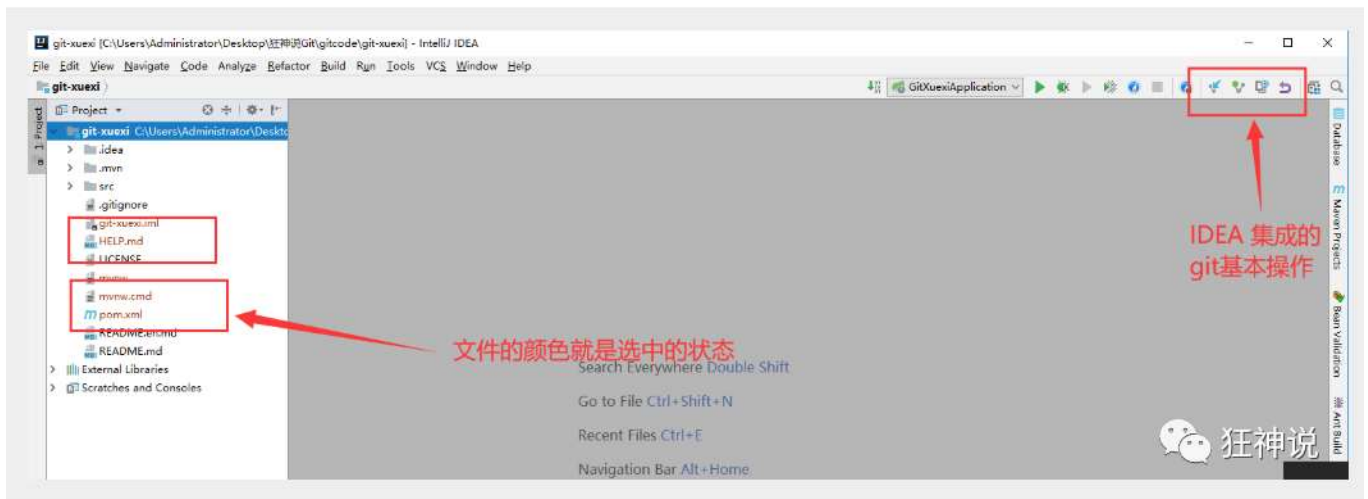
## IDEA中集成Git

### 1、新建项目，绑定git。

将我们远程的git文件目录拷贝到项目中即可！



注意观察idea中的变化



## 2、修改文件，使用IDEA操作git。

- 添加到暂存区
- commit 提交
- push到远程仓库

## 3、提交测试



这些都是单个人的操作！

学习的方式最重要！学会学习！我上课的更多时候都是在教大家去学习一种理念和思想（学习方式）

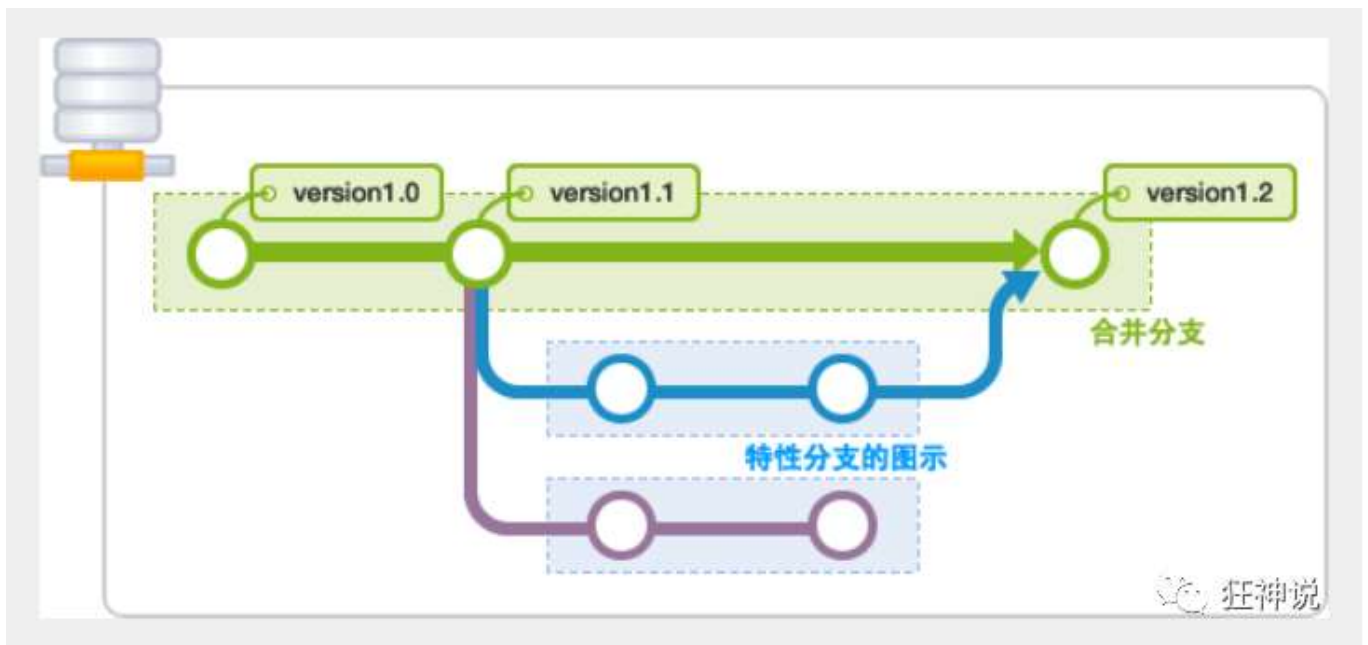
有道无术、术尚可求。有术无道、止于术！

真正的教学，授人以渔！



# 说明：GIT分支

分支在GIT中相对较难，分支就是科幻电影里面的平行宇宙，如果两个平行宇宙互不干扰，那对现在的你也没啥影响。不过，在某个时间点，两个平行宇宙合并了，我们就需要处理一些问题了！

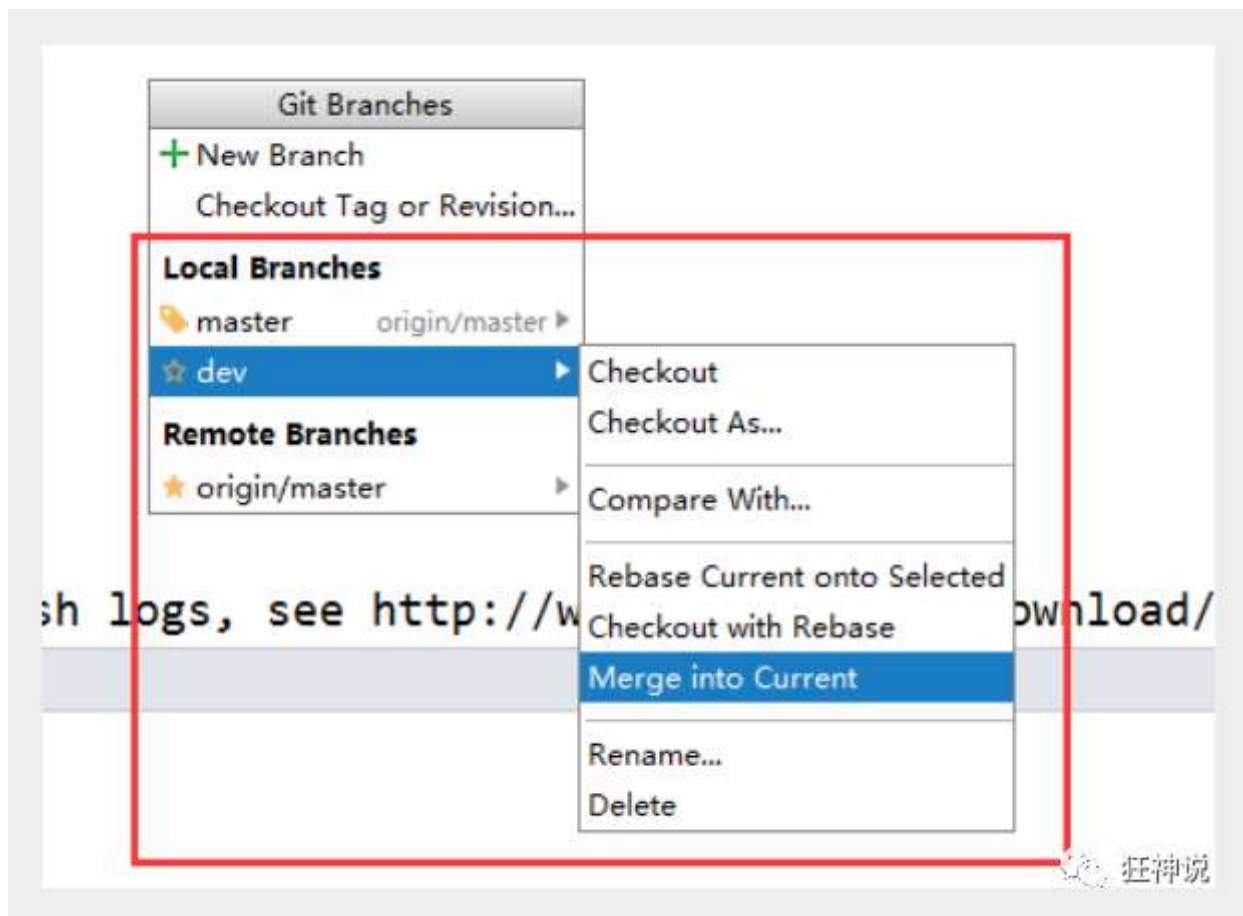


git分支中常用指令：

- 1 # 列出所有本地分支
- 2 `git branch`
- 3
- 4 # 列出所有远程分支

```
5 git branch -r
6
7 # 新建一个分支，但依然停留在当前分支
8 git branch [branch-name]
9
10 # 新建一个分支，并切换到该分支
11 git checkout -b [branch]
12
13 # 合并指定分支到当前分支
14 $ git merge [branch]
15
16 # 删除分支
17 $ git branch -d [branch-name]
18
19 # 删除远程分支
20 $ git push origin --delete [branch-name]
21 $ git branch -dr [remote/branch]
```

## IDEA中操作



如果同一个文件在合并分支时都被修改了则会引起冲突：解决的办法是我们可以修改冲突文件后重新提交！选择要保留他的代码还是你的代码！

master主分支应该非常稳定，用来发布新版本，一般情况下不允许在上面工作，工作一般情况下在新建的dev分支上工作，工作完后，比如上要发布，或者说dev分支代码稳定后可以合并到主分支master上来。

作业练习：找一个小伙伴，一起搭建一个远程仓库，来练习Git！

- 1、不要把Git想的很难，工作中多练习使用就自然而然的会了！
- 2、Git的学习也十分多，看完我的Git教程之后，可以多去思考，总结到自己博客！

视频教程同步更新，请这次一定！



“赠人玫瑰，手有余香”

狂神说的赞赏码





喜欢此内容的人还喜欢

重庆好热吖🥵你们还好吗

狂神说