聊一聊TortoiseGit

学习TortoiseGit之前，我们需要先明白一个概念

**版本控制！**

什么是版本控制

版本控制（Revision control）也就是版本迭代。

* 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。

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！

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：

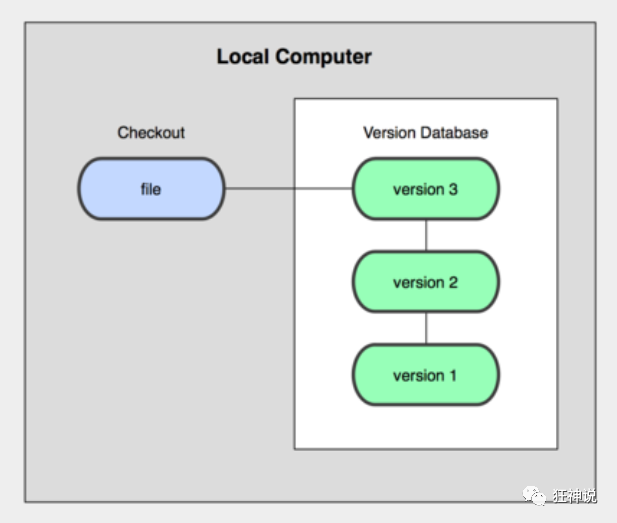
* **Git**
* **SVN**（Subversion）
* **CVS**（Concurrent Versions System）
* **VSS**（Micorosoft Visual SourceSafe）
* **TFS**（Team Foundation Server）
* Visual Studio Online

版本控制产品非常的多，现在影响力最大且使用最广泛的是Git与SVN

版本控制分类

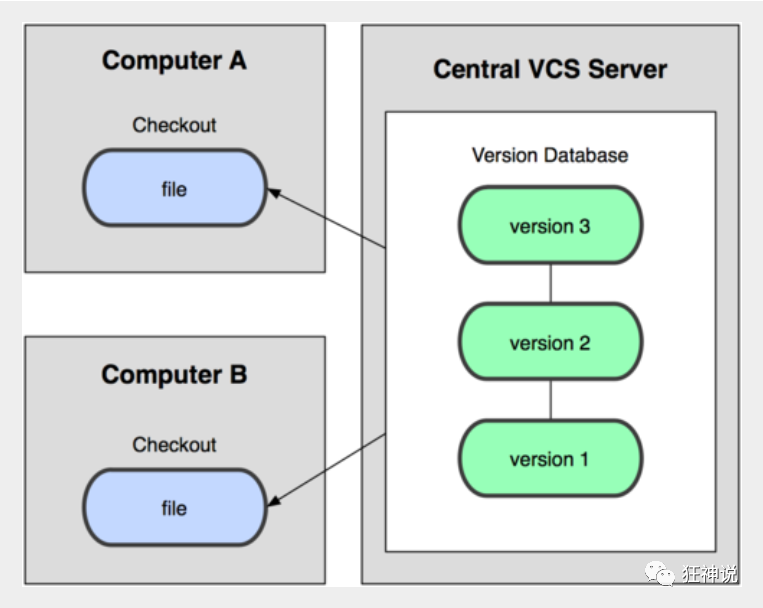
**1、本地版本控制**

记录文件每次的更新，可以对每个版本记录补丁文件（比如通过日期进行分割），适合个人用，如RCS。

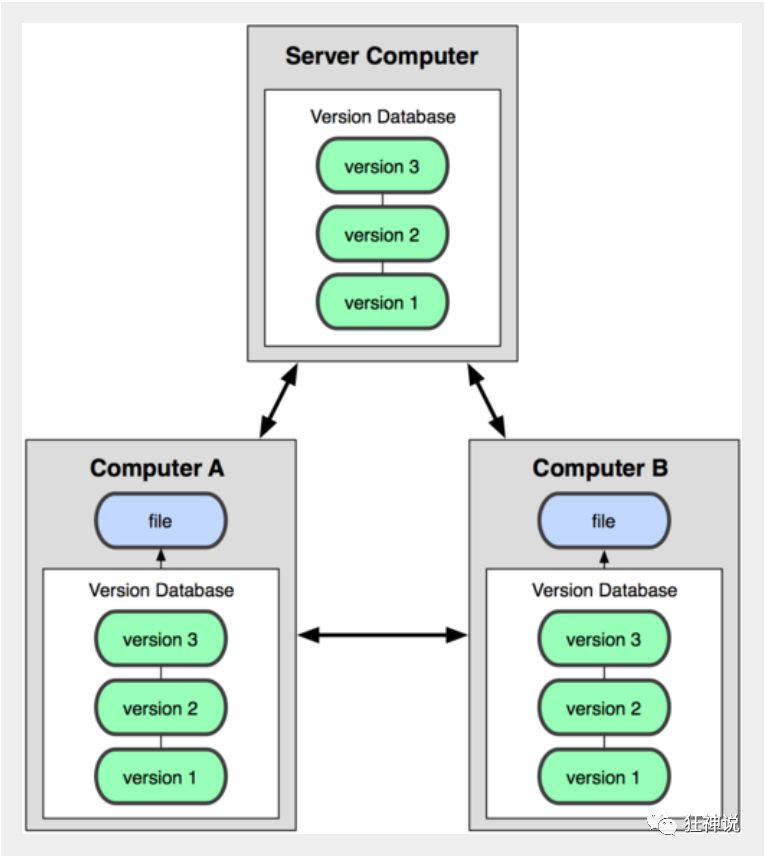


**2、集中版本控制  SVN**

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（如果断网就无法同步历史数据）

**3、分布式版本控制 Git**

每个人都拥有全部的代码！安全隐患！



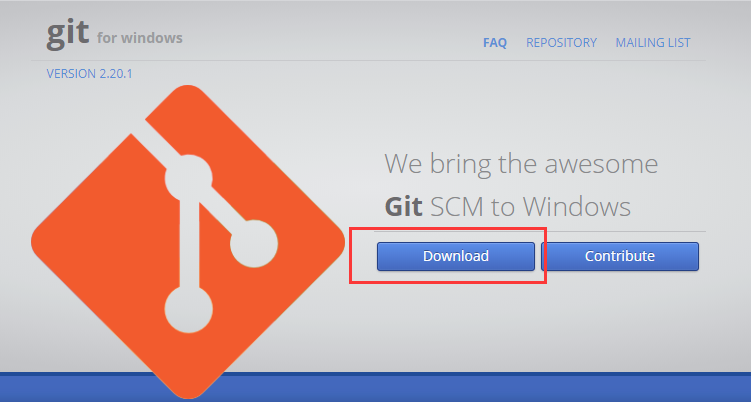
所有版本信息仓库全部同步到本地的每个用户，这样就可以在本地查看所有版本历史（可以离线提交）

Git与TortoiseGit的关系

**其实TorToiseGit就是Git的一个Gui包装**， Git作为一个复杂的版本控制系统，命令之多，即使经常使用，一些命令也记不住，一般只记住几个常用的命令，不是所有使用Git命令行都是高效的。要借助一个图形化的软件，TortoiseGit来操作git。用GUI（图形界面）再谈CLI（命令行），我相信这会更容易让人接受。

下载Git

第1步：下载Git，下载地址：<https://git-for-windows.github.io/>，一般官网下载慢，所以采用镜向下载http://npm.taobao.org/mirrors/git-for-windows/



**第2步：**下载安装Tortoisegit

<https://tortoisegit.org/download/>

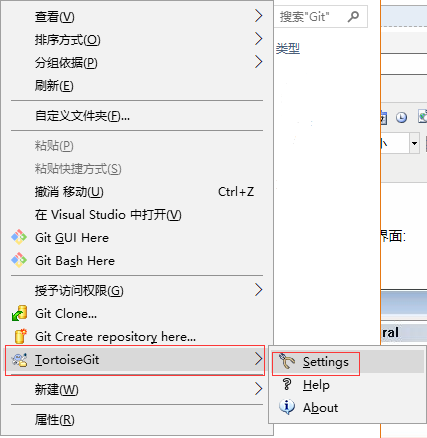


**Tortoisegit 配置**

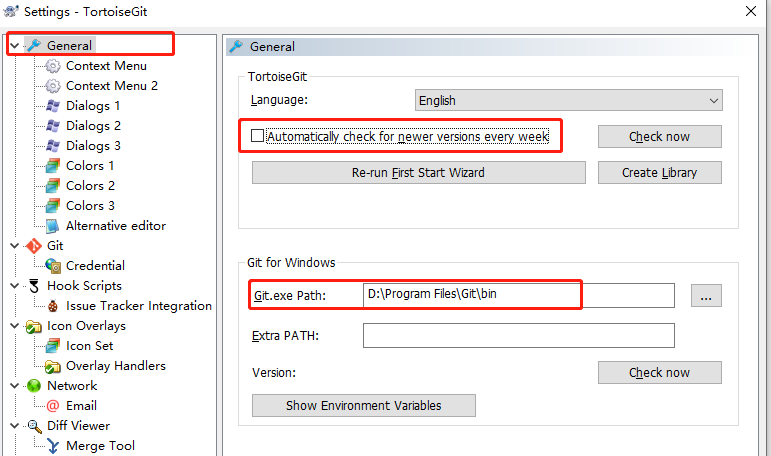
**第1步：**首先，请选定一个存放Git项目的目录，这样方便管理. 如: D:\Git，然后在资源管理器中打开：



**第2步：**在空白处点击鼠标右键，选择 --> TortoiseGit --> Settings，然后就可以看到配置界面：

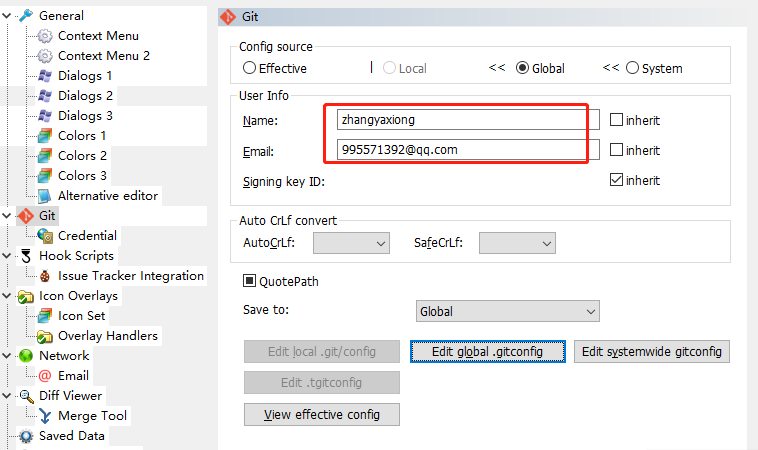


**第3步：**选中General，在右边的 Language中选择简体中文(当然,你也可以继续使用英文)。取消勾选升级检查的复选框，可能还需要指定 git.exe 文件的路径，如 "D:\Program Files\Git\bin"。完成后，点击应用，确定关闭对话框:



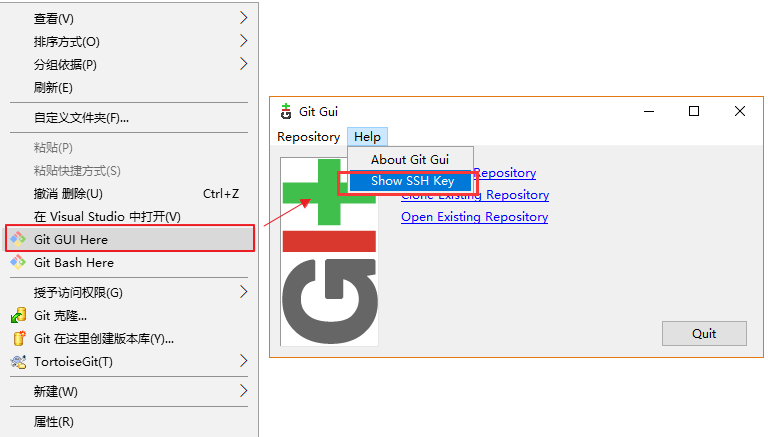
**第4步：**配置用户，用户作为你操作git的个人标识，进入设置，点选左边的Git标签，可以发现,右边可以配置用户的名字与Email信息(可以通过命令去添加相应的信息，比如记住密码等credential.helper=store

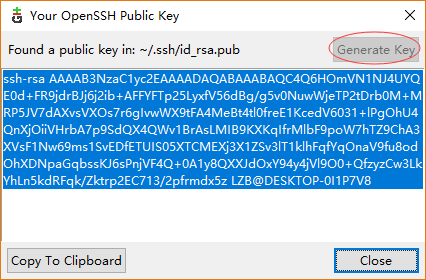
). 如下图所示：



**Tortoisegit 使用示例**

**第1步：**通过**SSH URL方式传输**，使用之前需要先将本地的 SSH 公钥配置到 Git 服务器上。

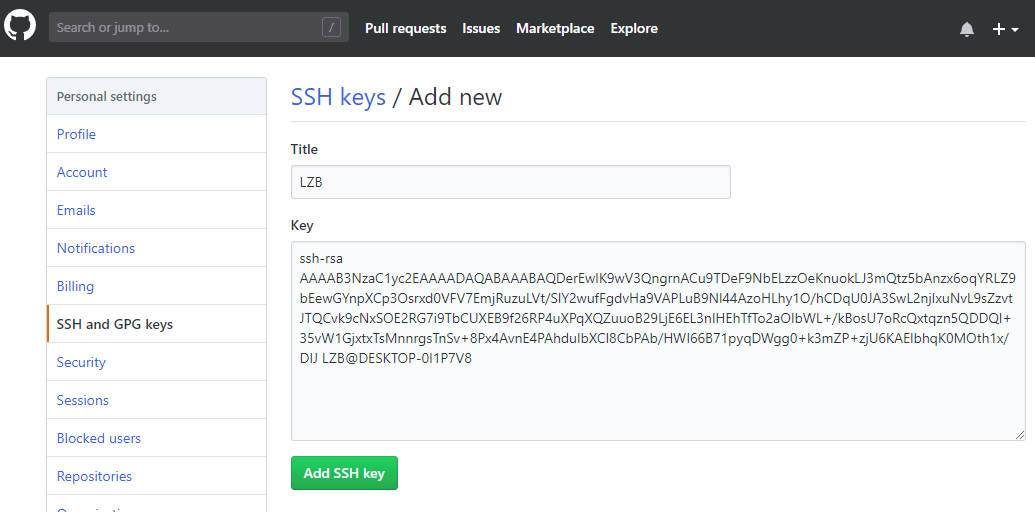
* 需要创建一个SSH Key，在任意文件夹下点击右键，选择 Git GUI Here。在弹出的程序中选择主菜单的【Help】→【Show SSH Key】：
* 如果没有Key，则点击“Generate Key”生成一个SSH Key：



把生成的全部内容拷出来。

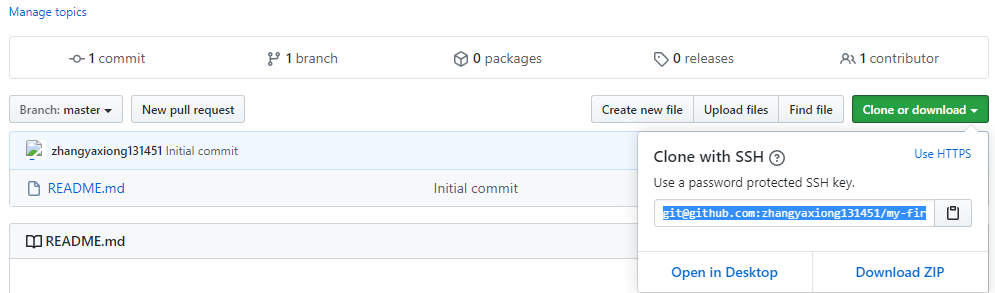
**第2步：**把 SSH Key 填到Git 服务器的配置中(zhangyaxiong131451/14789632@zyx)。

**GitHub服务器：**登陆GitHub，【头像】→【Settings】→【SSH and GPG Keys】→【New SSH Key】，填入上一步的复制的SSHKey，然后【Add SSH Key】：

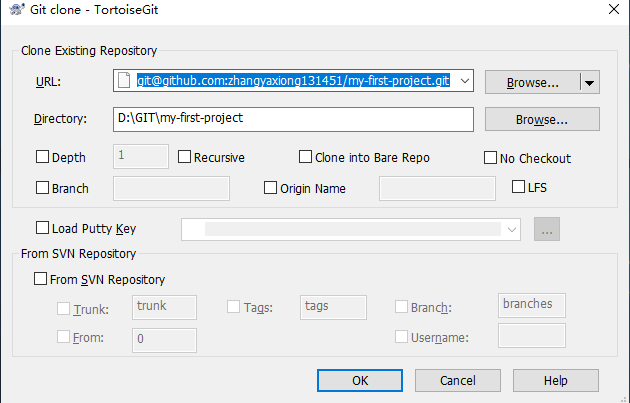


**第3步**：要克隆 GitHub 或 Gitblit 上的项目，先要获取对应远程存储库的路径（注意：此处使用 SSH URL 方式）：

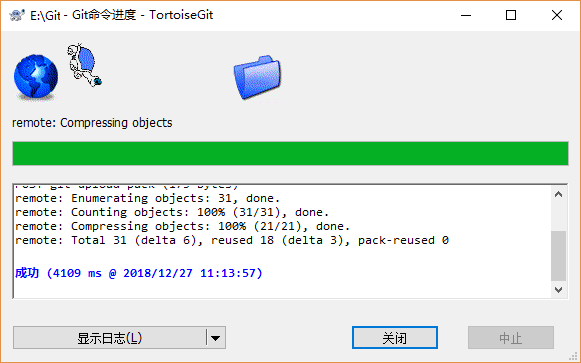
　　GitHub仓库：从 GitHub项目页面上得到对应存储库的 giturl 路径：



**第4步：**在本地文件夹的空白位置处，右击鼠标，在菜单中选择【Git克隆】：



　　把git 路径填写到URL，并选择本地文件目录，点击【确定】，弹出下载进度窗口。（服务器端至少要有一个文件，否则 pull的时候会报错 fatal: Couldn't find remote ref HEAD）



克隆成功，则在文件夹上就可以看到角标标识：



* Git文件上角标符号说明：

文件上的图标，可以反映出当前文件或者文件夹的状态：

　　1、正常的：绿色的对号 https://img2018.cnblogs.com/blog/1545277/201812/1545277-20181226130633376-1375659427.png

　　2、被修改过的：红色感叹号  https://img2018.cnblogs.com/blog/1545277/201812/1545277-20181226130659002-1901548592.png

　　3、新添加的：蓝色的加号https://img2018.cnblogs.com/blog/1545277/201812/1545277-20181226130708046-702059723.png

　　4、未受控的（无版本控制的）：蓝色的问号https://img2018.cnblogs.com/blog/1545277/201812/1545277-20181226130715848-1344413806.png

　　5、忽略不受控的：灰色的减号https://img2018.cnblogs.com/blog/1545277/201812/1545277-20181226130727649-1356151074.png

　　6、删除的：红色的x号 https://img2018.cnblogs.com/blog/1545277/201812/1545277-20181226130737919-811538319.png

　　7、有冲突的：黄色的感叹号 

**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：

git config --global user.name "kuangshen" #名称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。

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：

# 在当前目录新建一个Git代码库$ 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！

# 克隆一个项目和它的整个代码历史(版本信息)$ 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：

*#查看指定文件状态*git status [filename]  
*#查看所有文件状态*git status  
*# git add . 添加所有文件到暂存区# 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）。

*#为注释*\*.txt *#忽略所有 .txt结尾的文件,这样的话上传就不会被选中！*!lib.txt *#但lib.txt除外*/temp *#仅忽略项目根目录下的TODO文件,不包括其它目录temp*build/ *#忽略build/目录下的所有文件*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 ，公司中有时候会搭建自己的gitlab服务器

这个其实可以作为大家未来找工作的一个重要信息！

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！)

*# 进入 C:\Users\Administrator\.ssh 目录# 生成公钥*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！

克隆到本地！

IDEA中集成Git

1、新建项目，绑定git。

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：

# 列出所有本地分支git branch  
# 列出所有远程分支git branch -r  
# 新建一个分支，但依然停留在当前分支git branch [branch-name]  
# 新建一个分支，并切换到该分支git checkout -b [branch]  
# 合并指定分支到当前分支$ git merge [branch]  
# 删除分支$ git branch -d [branch-name]  
# 删除远程分支$ git push origin --delete [branch-name]$ 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！



微信扫一扫  
关注该公众号