

Revision Control

常见的版本控制工具

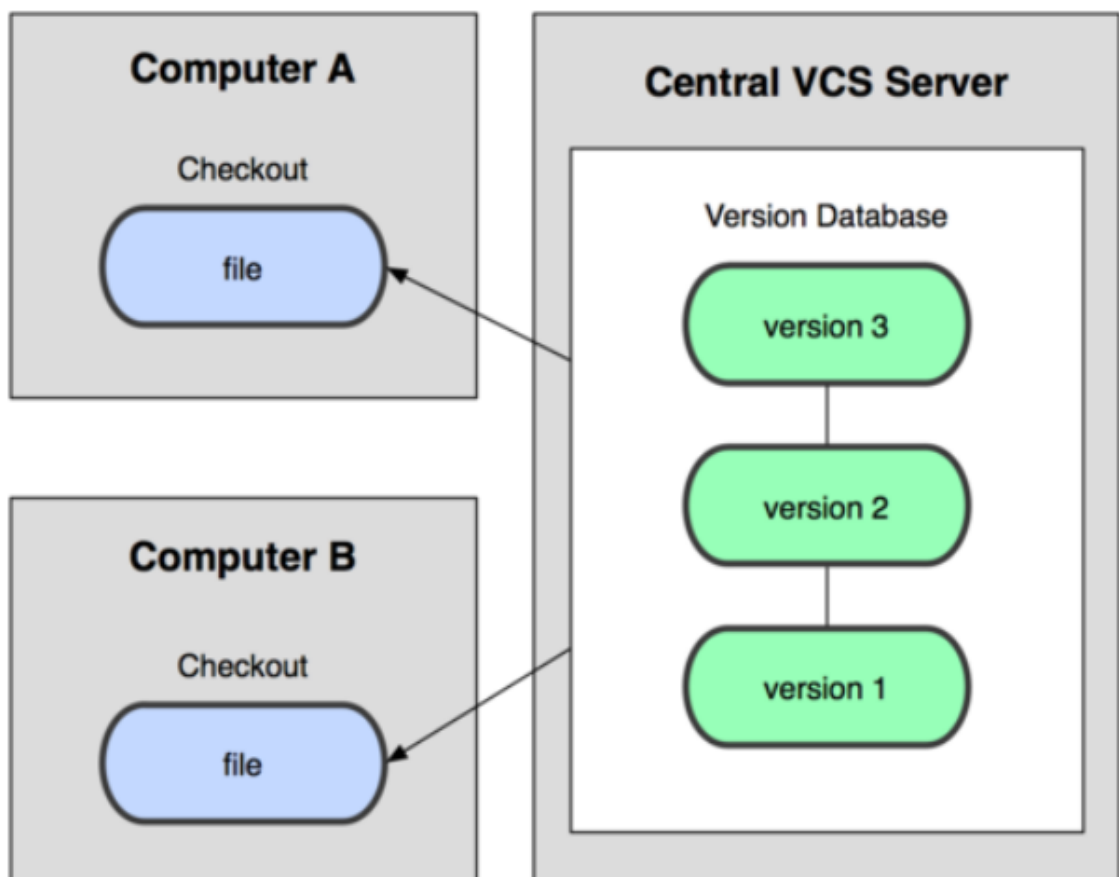
1. CVS 1990年诞生，远古时代的主流源代码管理工具
2. SVN 集中式版本控制之王者

SVN:又称subversion，是CVS的接班人，是一款 **集中式** 源代码管理工具。曾经是绝大多数开源软件的代码管理工具，前几年在国内软件企业使用最为普遍

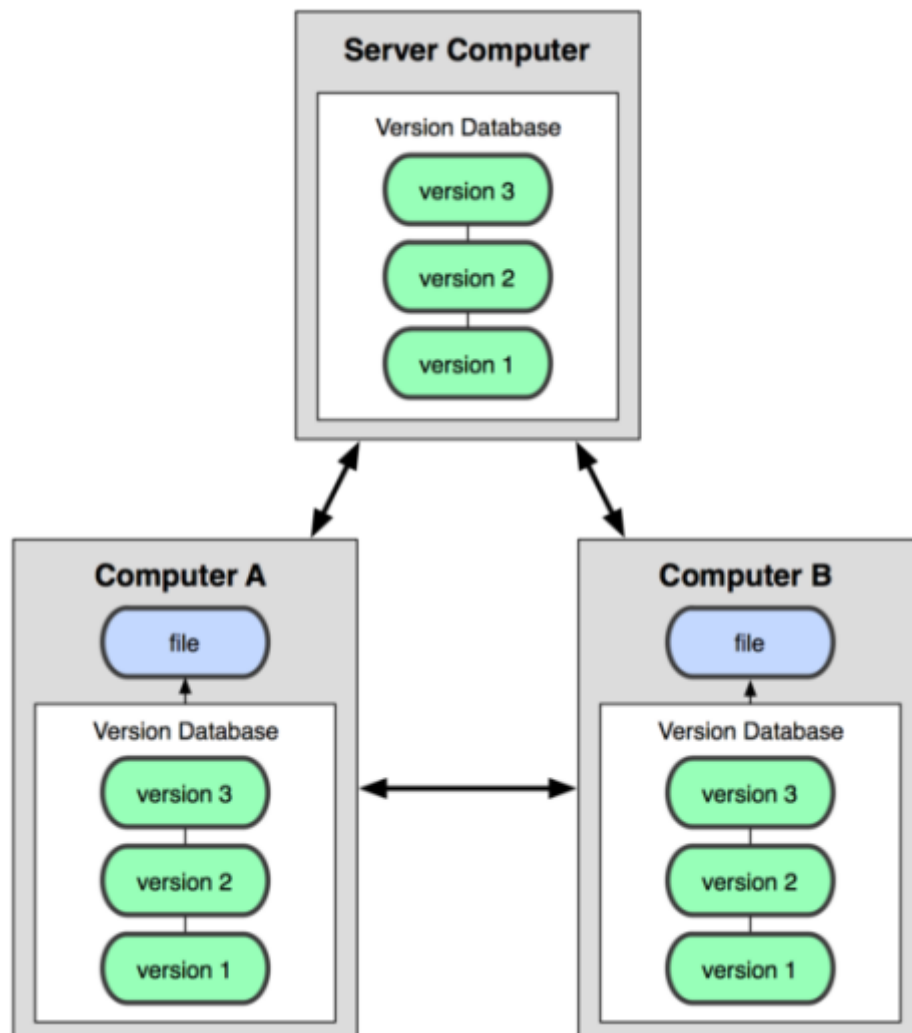
3. GIT 分布式版本控制之伟大作品

GIT:一款 **分布式** 源代码管理工具，目前国内企业大多都已经完成了从SVN到GIT的转换

- 集中式源代码管理



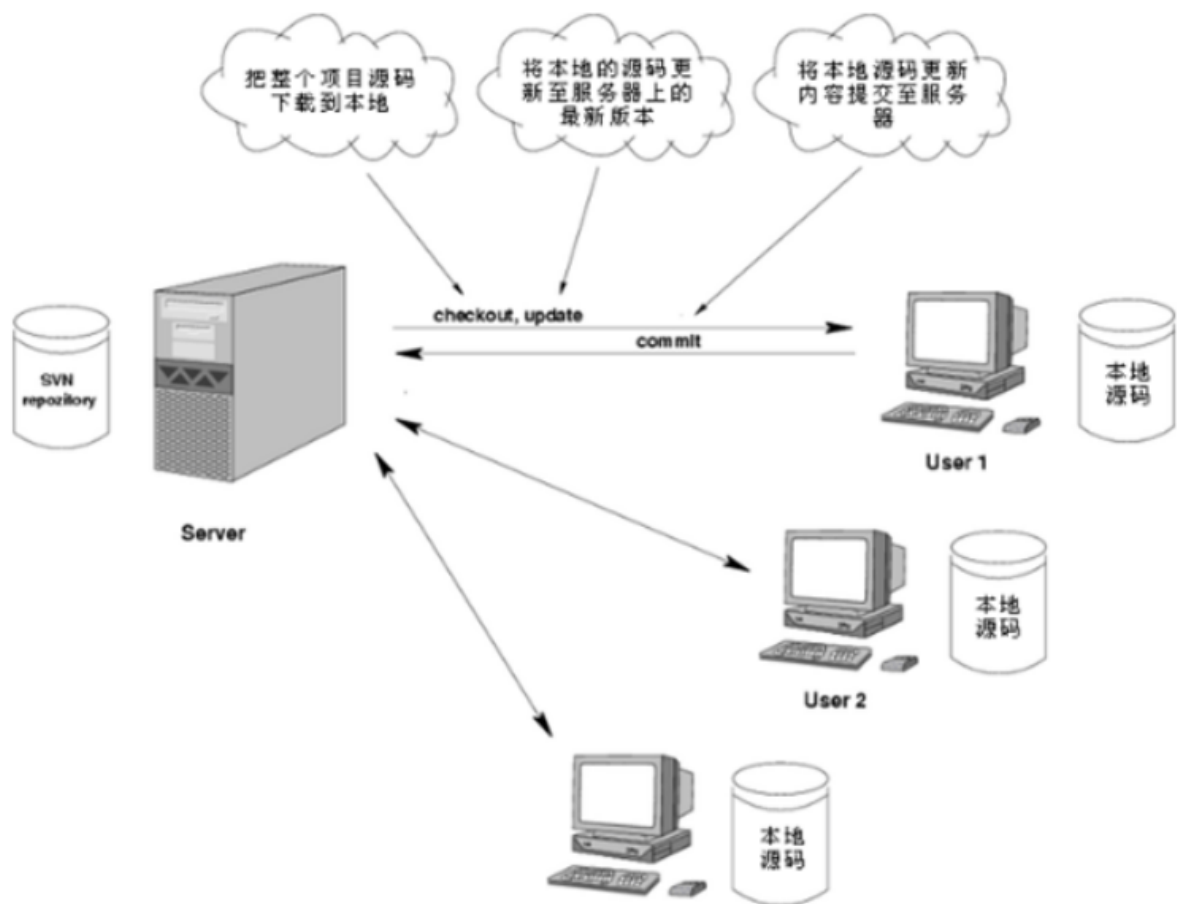
- 分布式源代码管理



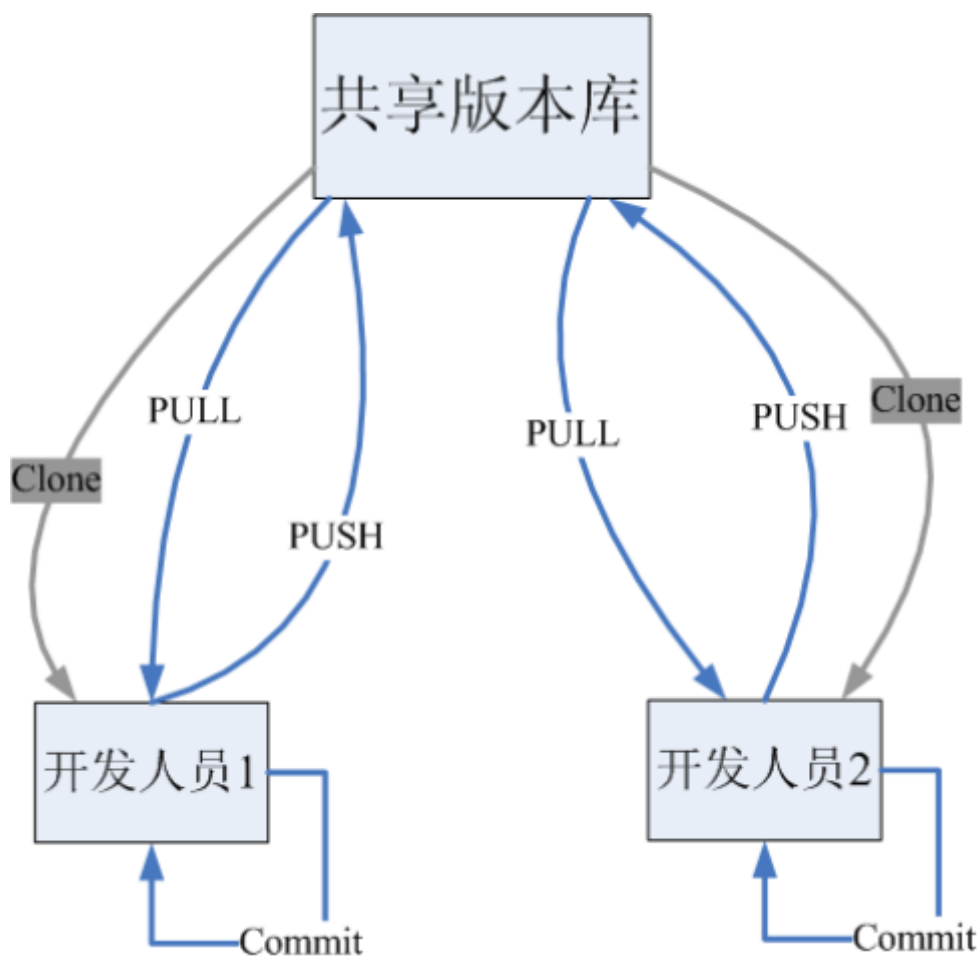
分布式和集中式的最大区别

- 在集中式下, 开发者只能将代码提交到服务器, 在分布式下, 开发者可以本地提交
- 在集中式下, 只有远程服务器上有代码数据库, 在分布式下, 每个开发者机器上都有一个代码数据库

SVN(集中式)



Git(分布式)



SVN和Git的简单对比

- 在多数情况下，Git的速度远比SVN快

- SVN是集中式的，Git是分布式的
- SVN使用分支比较笨拙，Git可以拥有无限多个分支
- SVN必须联网才能使用，Git支持本地版本控制工作
- 旧版本的SVN会在每个文件夹下放一个.SVN，Git只会在根目录下放一个.git文件夹

Git简介

Linux之父李纳斯的第二个伟大作品

Git的工作原理

- **工作区 (Working Directory)**

仓库文件夹里面, 除了 `.git` 目录 以外的内容

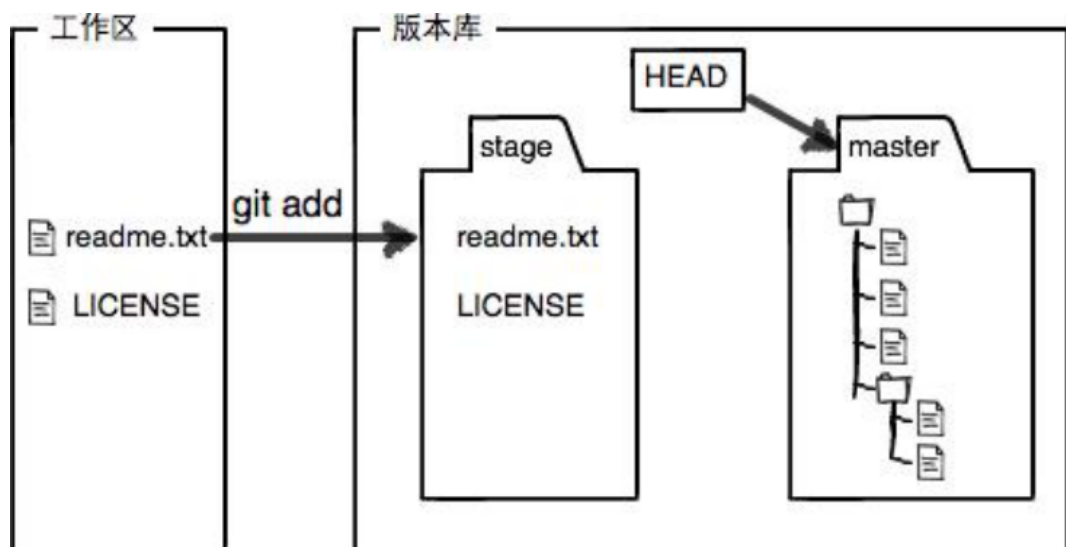
- **版本库(Repository)**

`.git` 目录, 用于存储记录版本信息

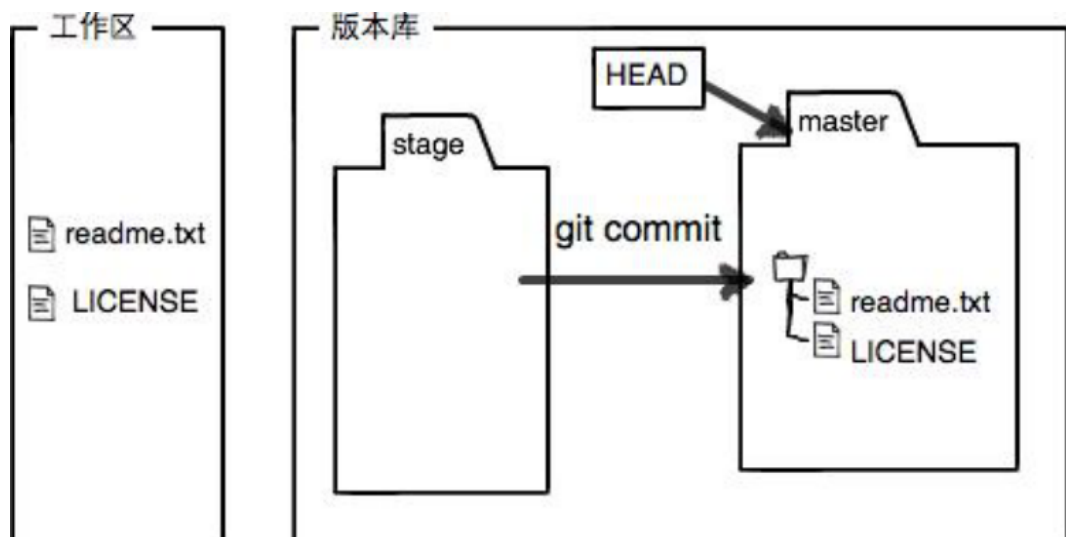
- 版本库中的**暂缓区(stage)**:
- 版本库中的**分支(master)**: git自动创建的第一个分支
- 版本库中的**HEAD指针**:用于指向当前分支

- `git add` 和 `git commit` 命令的作用

- `git add`: 把文件修改添加到暂缓区



- `git commit`: 把暂缓区的所有内容提交到当前HEAD指针指向的分支



单人开发

git工具安装完成后操作

1. 初始化工作区

- `git init` 之后工作区文件夹就会多一个 `.git` 的隐藏目录
- 等于在本地创建了一个版本库

2. 配置使用人的名称和联系邮箱

- `git config user.name "zp"`
- `git config user.email "gritpeng@gmail.com"`

该操作是使用git的第一步，也是让其他人知道是谁操作git,以及如何联系他

- `git config -l` 查看是否配置成功

3. 接下来，就可以使用git管理工作区中的文件

使用git管理文件

1. 查看文件有没有被git管理

- `git status` 如果有输出文件，并且文件是红色，说明没有被git管理，当前文件处于工作区中
- 如果需要git管理需要将工作区中的文件添加到暂存区

2. 将工作区中的文件添加到版本库的暂存区

- `git add 文件名`
- 此时，`git status`输出的文件呈绿色，说明已经被添加到暂存区了
- 接下来，需要将文件添加到HEAD指向的分支当中

3. 将暂存去的所有文件添加到header指向的分支当中

- `git commit -m "写上相应的注释"`
- 再次 `git status` ,如果没有输出文件，说明当前文件被git管理

以上就是使用git最简单的流程

使用git有什么好处

当我们修改了某个文件之后

1. 执行 `git status` 命令，就会输出那个文件被修改了

```
Peng@DESKTOP-JU446S0 MINGW64 ~/Desktop/product (master)
$ git status
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 directory)

        modified:   index.js

no changes added to commit (use "git add" and/or "git commit -a")
```

2. 执行 `git diff index.js` 也可以查看该文件那些地方被修改了

```
Peng@DESKTOP-JU446SO MINGW64 ~/Desktop/product (master)
$ git diff index.js
diff --git a/index.js b/index.js
index 8ed2346..d2bad3f 100644
--- a/index.js
+++ b/index.js
@@ -1,4 +1,3 @@
  window.onload = function{
-
-
+   console.log("输出的内容");
  }
\ No newline at end of file
```

此时被修改的文件处于工作区中，需要再次添加到版本库去管理

3. `git add index.js`

```
Peng@DESKTOP-JU446SO MINGW64 ~/Desktop/product (master)
$ git add index.js

Peng@DESKTOP-JU446SO MINGW64 ~/Desktop/product (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   index.js
```

4. `git commit -m "添加输出的内容"`

```
Peng@DESKTOP-JU446SO MINGW64 ~/Desktop/product (master)
$ git commit -m "添加输出的内容"
[master 6c87a9b] 添加输出的内容
 1 file changed, 1 insertion(+), 2 deletions(-)

Peng@DESKTOP-JU446SO MINGW64 ~/Desktop/product (master)
$ git status
On branch master
nothing to commit, working tree clean
```

回到之前的版本

1. 查看index文件修改的历史

- `git log index.js`

```
Peng@DESKTOP-JU446SO MINGW64 ~/Desktop/product (master)
$ git log index.js
commit 6c87a9b1d900261377e08015519e9896d593eebe (HEAD -> master)
Author: zp <gritpeng@gmail.com>
Date: Fri Oct 4 22:56:03 2019 +0800
```

添加输出的内容

```
commit 6ee88b285acadbf84cedefaa47f0e6c6968e974
Author: zp <gritpeng@gmail.com>
Date: Fri Oct 4 22:22:53 2019 +0800
```

初始化 添加index.js文件到master分支

2. 查看整个项目被修改的历史

- `git log`

```
Peng@DESKTOP-JU446SO MINGW64 ~/Desktop/product (master)
$ git log
commit 6c87a9b1d900261377e08015519e9896d593eebe (HEAD -> master)
Author: zp <gritpeng@gmail.com>
Date: Fri Oct 4 22:56:03 2019 +0800
```

添加输出的内容

```
commit cfa2fb74b1f7de80235c5fb24752278a01fe10a3
Author: zp <gritpeng@gmail.com>
Date: Fri Oct 4 22:42:47 2019 +0800
```

添加home.js文件到版本库的暂存区

```
commit 6ee88b285acadbf84cedefaa47f0e6c6968e974
Author: zp <gritpeng@gmail.com>
Date: Fri Oct 4 22:22:53 2019 +0800
```

初始化 添加index.js文件到master分支

3. 查看修改版本历史的简化内容

- `git reflog`

```
Peng@DESKTOP-JU446SO MINGW64 ~/Desktop/product (master)
$ git reflog
6c87a9b (HEAD -> master) HEAD@{0}: commit: 添加输出的内容
cfa2fb7 HEAD@{1}: commit: 添加home.js文件到版本库的暂存区
6ee88b2 HEAD@{2}: commit (initial): 初始化 添加index.js文件到master分支
```

4. 回到之前的某个版本

- `git reset --hard HEAD^` 代表回到HEAD分支的上一个版本

```
Peng@DESKTOP-JU446SO MINGW64 ~/Desktop/product (master)
$ git reset --hard HEAD^
HEAD is now at cfa2fb7 添加home.js文件到版本库的暂存区
```

```

window.onload = function{
  }

```

5. 再回到有输出语句的版本

- `git reflog`

```
Peng@DESKTOP-JU446SO MINGW64 ~/Desktop/product (master)
$ git reflog
cfa2fb7 (HEAD -> master) HEAD@{0}: reset: moving to HEAD^
6c87a9b HEAD@{1}: commit: 添加输出的内容
cfa2fb7 (HEAD -> master) HEAD@{2}: commit: 添加home.js文件到版本库的暂存区
6ee88b2 HEAD@{3}: commit (initial): 初始化 添加index.js文件到master分支
```

- `git reset --hard 6c87a9b`

```

window.onload = function{
    console.log("输出的内容");
}

```

6. 如果有那些文件不需要管理，那么可以在工作区文件夹中创建一个 `.gitignore` 文件

- 执行命令 `touch .gitignore` 就会在工作区中新建一个 `.gitignore` 文件







```
Peng@DESKTOP-JU446SO MINGW64 ~/Desktop/product (master)
$ touch .gitignore

Peng@DESKTOP-JU446SO MINGW64 ~/Desktop/product (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
```

```

.gitignore
home.css
index.css

```

	.git	2019/10/4 23:34	文件夹
	.gitignore	2019/10/4 23:34	文本文档
	home.js	2019/10/4 22:40	JetBrains WebStorm
	index.css	2019/10/4 23:34	JetBrains WebStorm
	index.js	2019/10/4 23:19	JetBrains WebStorm
	home.css	2019/10/4 23:34	JetBrains WebStorm

- 然后在 `.gitignore` 文件中，添加那些文件需要被忽略

```

.gitignore
1 *.css

```

```
Peng@DESKTOP-JU446SO MINGW64 ~/Desktop/product (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

.gitignore

nothing added to commit but untracked files present (use "git add" to track)

Peng@DESKTOP-JU446SO MINGW64 ~/Desktop/product (master)
$
```

- 此时，以 `css` 结尾的两个文件就不被git管理了
- 至于怎么编写 `.gitignore` 文件，直接查资料

单人流程

准备工作（只做一次）

1. 创建一个工作区
2. 在工作区中打开 git 终端 `Git Bash Here`
3. 通过 `git init` 指令，初始化版本库
4. 通过 `git config user.name "姓名"` 和 `git config user.email "邮箱地址"` 来设置用户名和邮箱（不设置要挨骂）
5. 通过 `git config -l` 查看设置情况

开发阶段（反复执行）

1. 编写代码
2. 通过 `git add 文件名称` 或者 `git add .` 添加到版本库的暂存区当中
3. 通过 `git commit -m "操作说明"` 将暂存区的文件添加到HEAD指针指向的分支中（默认只有一个分支，master分支，也叫做主分支）
4. 注意点
 - 不是写一句代码就 `add commit` 一次，应该是完成一个功能后再 `add commit`
 - `commit` 时 `-m` 注释一定要认真编写，与当前提交内容保持一致，否则完蛋

单人使用git管理项目的好处

1. 可以通过 `git status` 查看哪些文件没有被管理，以及修改了哪些文件
 - 红色（没有被管理或者被修改了）、绿色（在缓存区）
2. 可以通过 `git diff 文件名称` 来查看具体修改了哪些代码
3. 可以通过 `git log` 或者 `git reflog` 来查看项目的演变历史
4. 可以通过 `git reset --hard 版本号` 在任意版本之间切换
5. 无需备份多个文件，每次 `commit` 提交Git会自动备份

多人开发

在远程服务器上创建一个共享版本库

此操作不需要开发人员来操作

1. 项目负责人打开远程服务器，然后创建一个工作区
2. 在远程的服务器上打开工作区，在工作区中打开git终端工具（指令工具）
3. 在git终端工具中输入 `git init --bare`

```
Peng@zp MINGW64 ~/Desktop/project
$ git init --bare
Initialized empty Git repository in C:/Users/Peng/Desktop/project/
```

4. 经过以上步骤，在代表远程服务器的共享版本库已经创建好了

project		▼ ↺	搜索"project"
名称	修改日期	类型	
hooks	2019/10/5 10:51	文件夹	
info	2019/10/5 10:51	文件夹	
objects	2019/10/5 10:51	文件夹	
refs	2019/10/5 10:51	文件夹	
config	2019/10/5 10:51	文件	
description	2019/10/5 10:51	文件	
HEAD	2019/10/5 10:51	文件	

开发人员下载远程版本库

1. 开发人员在自己的电脑上打开Git终端工具
2. 从远程的服务器上下载当前项目的共享版本库
 - `git clone` 远程服务器共享版本库地址

```
Peng@zp MINGW64 ~/Desktop/zs
$ git clone /c/Users/Peng/Desktop/project
Cloning into 'project'...
warning: You appear to have cloned an empty repository.
done.
```

- 和单人开发使用Git的区别：
 - 单人开发是自己创建版本库，而多人开发是从远程服务器下载版本库

进入开发阶段

和单人开发一样

1. 设置用户名和邮箱(切记)
2. 编写代码
3. `git add .` 添加到暂缓区
4. `git commit -m "备注"` 添加到HEAD指针指向的分支
5. 注意点

`commit` 是将编写好的代码提交到本地的版本库，所以其他开发人员是拿不到我们提交的代码的

如果想让其他开发人员也能拿到我们提交的代码，还必须将我们编写好的代码提交到远程服务器

多人开发特有：

6. 将代码提交到远程的服务器 `git push`
7. 其他开发人员只需要通过 `git pull` 就可以拿到更新的代码了

多人开发使用Git的注意事项

1. 不能将不能运行的代码提交到本地和远程服务器（切记）
2. 如果服务器上有其他开发人员的更新内容（这里假设更新的不是同一个文件），那么我们不能直接通过 `push` 将我们的代码提交到远程服务器

如果服务器上有其他开发人员更新的内容，我们必须先将其他开发人员更新的内容更新到本地之后才能通过 `push` 提交我们的内容

3. 如果我们更新的内容和其他同事更新的内容有冲突（修改了同一个文件的同一行代码），这个时候需要我们自己手动修改冲突，修改完冲突之后才能将代码提交到远程服务器

开发技巧：

只要开发完一个功能就要立即提交代码，因为在企业开发中谁后提交谁就负责解决冲突，谁的工作量就大

Git分支的使用

如何查看有多少个分支

1. 通过 `git branch` 指令就可以查看当前版本库中有多少分支

注意点：

1. 如果当前版本库是空的，那么就无法查看
2. 如果通过 `git branch` 指令查看当前版本库中有多少个分支，输出的内容中哪一个分支前面有*号，就代表当前HEAD指针指向哪一个分支，我们提交的代码就会提交到HEAD指针指向的分支中

如何创建一个分支

1. 通过 `git branch 分支名称` 来创建一个新的分支

注意点：

在那个分支中创建了新的分支，那么创建出来的分支就会继承当前分支的所有状态

例如：

在master分支中做了两个操作，然后在master分支下创建了Dev分支，那么创建出来的分支就会继承master分支中的这两个操作

注意：一旦分支被创建出来之后，分支就是独立的，分支之间不会相互影响，如果需要同步两个分支就必须合并

如何将分支提交到远程服务器

1. 通过 `git branch -r` 来查看远程服务器上有多少个分支
2. 首先需要在本地切换到新建的分支当中，然后通过 `git push` 指令提交新建的分支到远程的服务器
 - `git push --set-upstream origin Dev`

如何合并分支

可以通过 `git merge 分支名称` 来合并分支

例如：

在master分支中执行 `git merge Dev` 就代表需要将Dev分支中的所有代码合并到master分支中

在Dev分支中执行 `git merge master` 就代表需要将master分支中的所有代码合并到Dev分支中

如何删除分支

1. 可以通过 `git branch -d 分支名称` 来删除本地的分支
2. 可以通过 `git push origin --delete 分支名称` 来删除远程服务器的分支，一般用的比较少

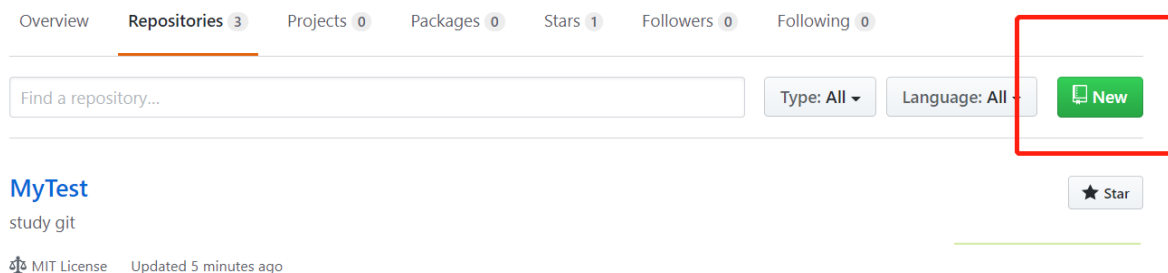
Git工具的升级

`git update-git-for-windows`

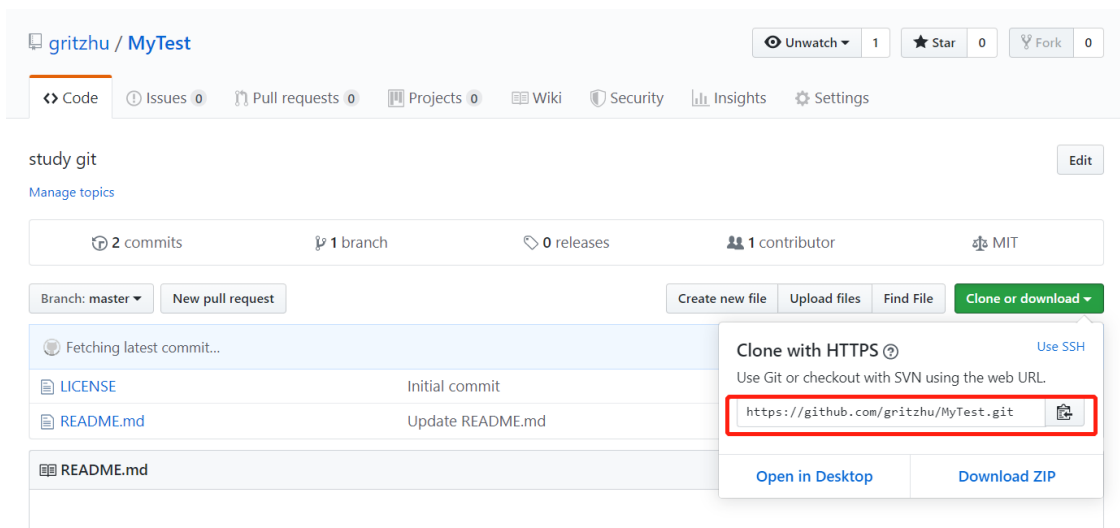
GitHub的使用

怎么在GitHub上创建工作区？怎么在GitHub上创建共享版本库？

1. 在GitHub上创建仓库



2. 创建好工作区之后，copy仓库的地址



3. 在本地创建一个工作区文件夹，然后把GitHub上的工作区clone下来

```
Peng@zp MINGW64 ~/Desktop/swiper
$ git clone https://github.com/gritzhu/MyTest.git
Cloning into 'MyTest'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (7/7), done.
```

名称	修改日期	类型
MyTest	2019/10/5 22:11	文件夹
swiper	2019/10/5 21:54	文件夹

- 然后在本地工作区中，打开git终端，告诉他我是谁，我的联系方式

```
Peng@zp MINGW64 ~/Desktop/swiper/MyTest (master)
$ git config user.name "zhupeng"

Peng@zp MINGW64 ~/Desktop/swiper/MyTest (master)
$ git config user.email "gritpeng@sina.com"
```

- 此时，我们使用http的方式，git push代码的话，会弹出让我们登录GitHub账号，每次提交都要登录的话，就很麻烦，所以一般企业开发中会使用SSH的方式来操作

怎么配置SSH

- 首先在随便一个文件夹下打开git终端

```
Peng@zp MINGW64 ~/Desktop/新建文件夹
$ ssh-keygen -t rsa -C "gritpeng@sina.com"
```

- 就会在电脑，生成公钥和私钥

此电脑 > System (C:) > 用户 > Peng > .ssh


名称	修改日期	类型	大小
id_rsa	2019/10/5 22:35	文件	3 KB
id_rsa.pub	2019/10/5 22:35	PUB 文件	1 KB

私钥 (指向 id_rsa)
公钥 (指向 id_rsa.pub)

- 将公钥复制并设置到GitHub上

SSH keys New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.



MySelf PC Dell
 8b:d8:c1:5b:03:e2:7c:d5:fb:ad:dc:45:36:91:94:43
 Added on 5 Oct 2019
 Never used — Read/write

Delete

- 然后再git终端执行

```
ssh -T git@github.com
```

```
Peng@zp MINGW64 ~/Desktop/新建文件夹
$ ssh -T git@github.com
The authenticity of host 'github.com (13.229.188.59)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXUpJWG17E1IGOCspRomTxdCARLviKw6E5SY8.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com,13.229.188.59' (RSA) to the list of known hosts.
Hi gritzhu! You've successfully authenticated, but GitHub does not provide shell access.
```

输出hi github的用户名，意味着配置成功！

- 然后再返回GitHub上，使用SSH方式clone 代码
- clone 完成之后，进入工作区，打开git终端

```
Peng@zp MINGW64 ~/Desktop/新建文件夹/MyTest (master)
$ git config user.name "zhupeng"
```

```
Peng@zp MINGW64 ~/Desktop/新建文件夹/MyTest (master)
$ git config user.email "gritpeng@sina.com"
```

```
Peng@zp MINGW64 ~/Desktop/新建文件夹/MyTest (master)
$ git config --global user.name "zhupeng"
```

```
Peng@zp MINGW64 ~/Desktop/新建文件夹/MyTest (master)
$ git config --global user.email "gritpeng@sina.com"
```

7. 然后就可以将修改的代码，`git push`到GitHub上了

Gitflow使用

待更