Git版本控制

❖是否还未听过Git OR GitHub?

❖是否还未拥有自己的GitHub账号?

❖是否还不懂Git工作原理?

❖是否看完Git网络教程还是不会将代码推送到GitHub?

Git知识铺垫

- 程序员为什么要使用Git版本控制?
- 常见的版本控制?
- Git是什么?
- Git安装
- Git工作原理
- 常用git服务器
- Git常用命令
- GitHub简单使用，如何拥有自己的仓库
- 忽略文件
- SSH-keygen用法
- 工作方式介绍

1. 程序员为什么要使用Git版本控制?

现在的软件项目通常是由一个研发小组共同分析、设计、编码、维护以及测试的。在公司99%的都是团队合作开发项目，如果是团队开发项目，那么就会遇到以下问题:

- 难以恢复至以前正确版本 (版本**1.0-2.0**)
- 容易引发bug
- 代码责任问题
- 代码管理问题
- 无法进行权限控制
- 项目版本发布困难.....

使用版本控制工具：

- 不会对现有工作造成任何损害
- 不会增加工作量
- 代码管理更方便
- 代码得以追随
- 添加新的功能拓展时，会变得更加容易.....

2. 常见的版本控制

- CVS版本控制
- SVN版本控制
- GIT 版本控制

CVS：CVS是一个C/S系统，是一个常用的代码版本控制软件，1990年诞生，10多年前主流源代码管理工具。

SVN：SVN:又称subversion，是一款集中式源代码管理工具。由于之前CVS编码的问题，大多数软件开发公司都使用SVN替代了CVS)，前几年在国内软件企业使用最为普遍。

GIT：一款分布式源代码管理工具，目前国内企业基本都使用Git。

CVS和SVN是一个集中式的版本控制器，他们需要一台专门的版本控制服务器，开发者只能将代码提交到服务器。只有远程服务器上有代码数据库。
GIT是分布式的管理，他不要一台专门的服务器来运行这个版本控制。分布式相比于集中式的最大区别在于开发者可以提交到本地，也可以提交到远程，每个开发者通过克隆（git clone），在本地机器上拷贝一个完整的Git仓库。

而SVN作为集中式的版本管理系统，依然有他的优缺点

优点：

1.管理方便，逻辑明确，操作简单，上手快。

2.易于管理，集中式服务器更能保证安全性。

3.代码一致性非常高。

4.有良好的目录级权限控制系统。

·劣势

1.对服务器性能要求高，数据库容量经常暴增，体量大。

2.必须联网。如果不能连接到服务器上，基本上不可以工作，如果服务器不能连接上，就不能提交，还原，对比等等。

3.不适合开源开发。

4.分支的管控方式不灵活

Git的优缺点如下：

1．适合分布式开发，每一个个体都可以作为服务器。每一次Clone就是从服务器上pull到了所有的内容，包括版本信息。

2．公共服务器压力和数据量都不会太大。

3．速度快、灵活，分支之间可以任意切换。

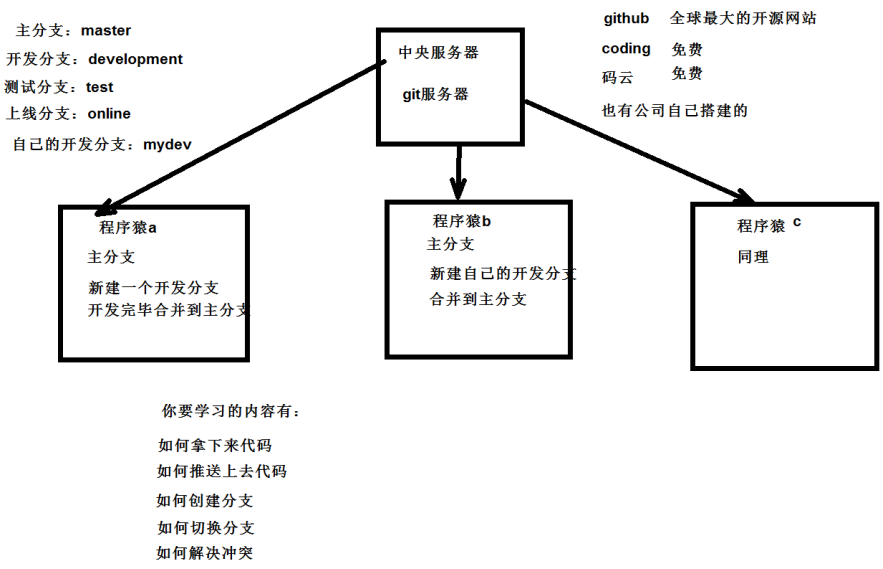
4．任意两个开发者之间可以很容易的解决冲突，并且单机上就可以进行分支合并。

5．离线工作，不影响本地代码编写，等有网络连接以后可以再上传代码，并且在本地可以根据不同的需要，本地新建自己的分支。

3. git是什么?

Git:一款分布式源代码管理工具，是Linux之父李纳斯（Linus Torvalds）的第二个伟大作品。

在世界上所有的分布式版本控制工具中，git是最快、最简单、最流行的。

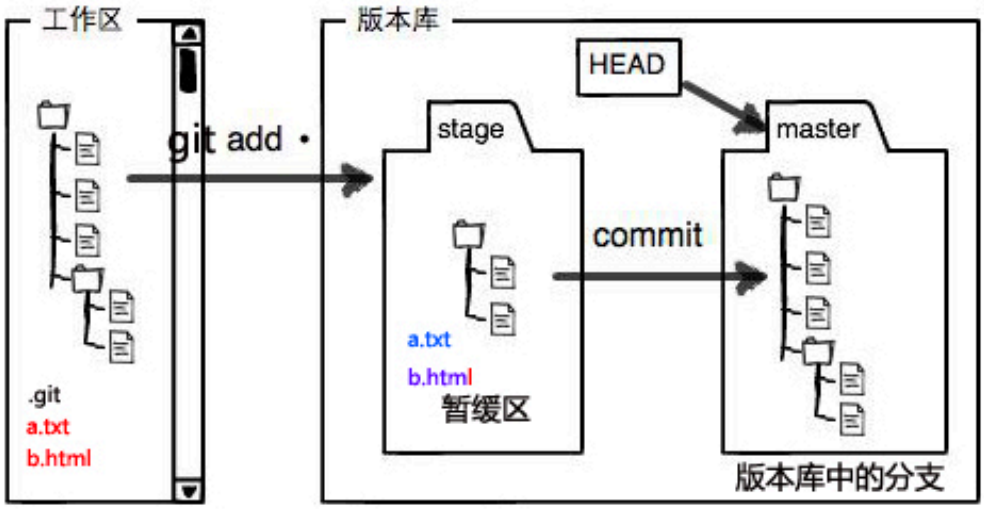


4. Git安装

首先进入Git下载地址:选择对应操作系统的版本，如下图
https://git-scm.com/downloads

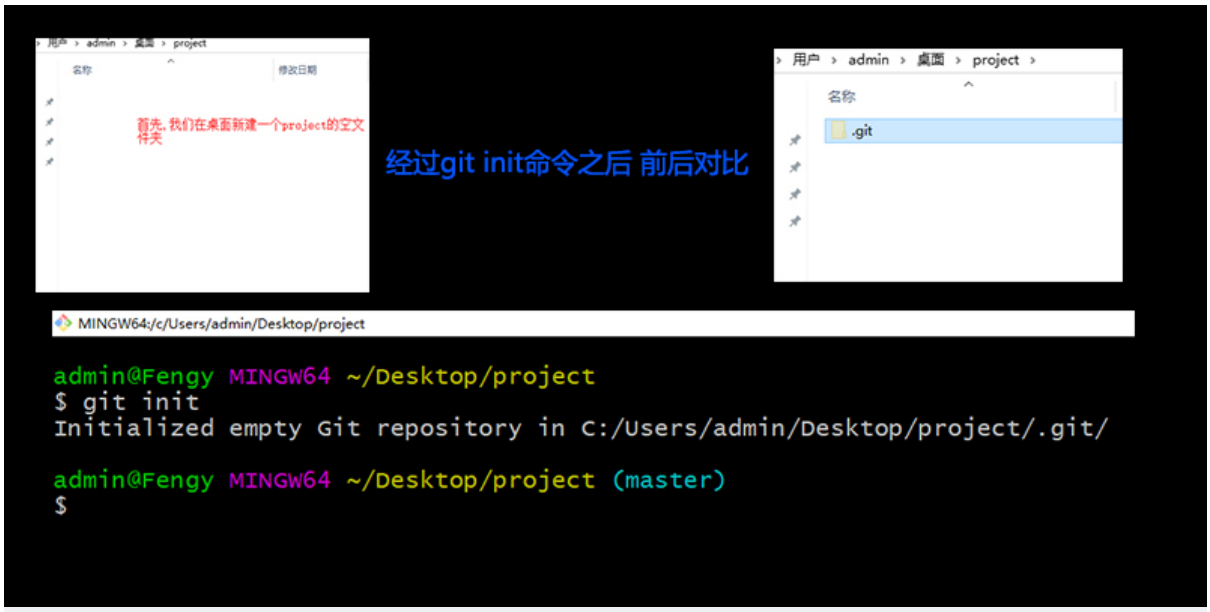
选择对应Git版本(32位or64位):适用于Windows安装程序的Git,
下载完成之后，安装即可(安装过程中点击下一步(**Next**)即可)

验证Git是否安装成功:(验证方法如上图)



5. Git工作原理

学好Git的前提是理解Git工作原理
了解Git工作原理前，我们需要了解两个重要的知识，即工作区和版本库。
工作区：就是你在电脑里能看到的目录 （仓库文件夹里面, 除了.git目录以外的内容.)



版本库：git目录, 用于存储记录版本信息.（详见下图）

版本库中的暂缓区(staga):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

隐藏的.git目录分别代表什么意思如下:

hooks: 默认的hook脚本
info: 里面有个exclude文件，指定本项目要忽略的文件
objects: Git对象库（commit/tree/blob/tag）
refs： 标识每个分支指向哪个提交
config:项目的配置信息
description: 项目的描述信息
HEAD :指向当前分支的一个提交

经过执行**git add .** 命令之后会将工作区的文件添加到暂缓区中，再经过执行**git commit -m** "说明文字" 命令之后，会将暂缓区的文件添加到版本库的分支当中去.

6.常用Git服务器

1.github==全球最大的开源网站 https://github.com/

2.码云==免费的，国内的 https://gitee.com/

3.coding 码市==国内的 https://coding.net/login

Git命令个人开发

在学习Git命令个人开发之前，我们需要了解一些常用的git命令.

git help

打开git bash终端，输入git help 指令会出现如下图所示的git命令详细解释，每个命令代表的意思

git init : (个人仓库)仓库初始化

手写我们在桌面新建一个project空文件夹之后,鼠标右键以git bash here 打开git终端,输入git init 命令之后,观察project文件夹之后多了一个隐藏文件夹.git目录.这时我们就创建好了一个受git管理的仓库,这个仓库就在本地.

配置用户基本信息

即配置用户名和邮箱. 当前项目下配置用户名与邮箱命令如下:

配置用户名：**git config —global user.name** “用户名**"**（跟踪who修改记录）

配置邮箱：**git config —global user.email** “邮箱”（多人开发间的沟通）

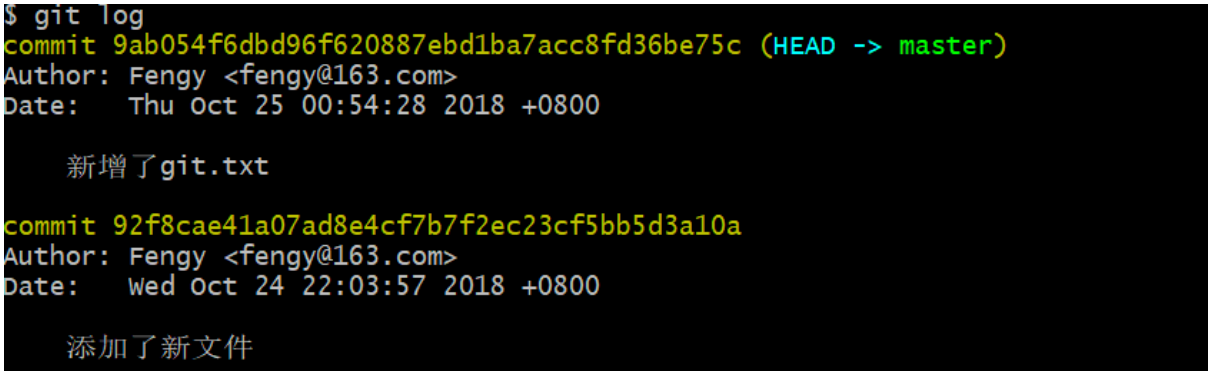
```
admin@Fengy MINGW64 ~/Desktop/project (master)
$ git config user.name "Fengy"

admin@Fengy MINGW64 ~/Desktop/project (master)
$ git config user.email "fengy@163.com"
```

查看配置信息命令 **git config -l**

git push --set-upstream origin 分支名

git log： 查看文件的修改日志

在工作区再新增git.txt文件,并将该文件添加到主分支(执行上述命令)，然后输入git log



命令,如下图所示,我们可以清楚的看到什么时候谁(**who**)干了什么事。

git diff : 查看文件最新改动的地方
git reset：版本回退（建议加上—hard参数，git支持无限次后悔）

回退到上一个版本的方式：
回退到上上一个版本：git reset —hard HEAD^^
回退到上N个版本：git reset —hard HEAD~N（N是一个整数）
回退到任意一个版本：git reset —hard 版本号

5. GitHub简单使用

1.注册并登录Github网站

使用GitHub之前，我们需要去GitHub官网注册一个属于自己的账号,然后登录你的,常规注册，注意一点需要邮箱认证，否则不能创建项目，

2.Github上创建仓库

（1）新建一个仓库（Repositories），填写项目名和项目说明，创建成功会生成一个网址

3.获取项目—从服务器（云端）获取项目到本地

本地找一个文件夹存放文件，然后右键打开git执行下面代码，下载远程仓库到本地 ,可将自己或者别人项目拿到本地

git clone https 地址====将仓库克隆到本地

注意：【创建ssh密钥】也可以在之前创建

创建的目的是为了保证安全性，只有具有密钥的管理员才可上传，其他人只可以下载

下拉项目更新本地 -- git pull ==== 从服务器更新代码

本地推送到服务器

先在本地实施版本控制，即创建版本库，又叫仓库（repository）

(1). 创建仓库（版本库),选择一个合适的位置创建一个空目录

mkdir +文件名 #文件名和路用英文，文件编辑可用notepad++

cd 文件名

(2). 进入目标文件夹（想推送的文件所在文件夹）

git init 初始化Git仓库，将这个目录可以变为仓库

瞬间Git就把仓库建好了，而且告诉你是一个空的仓库（empty Git repository）， 如果你没有看到.git目录，那是因为这个目录默认是隐藏的，用ls -ah命令就可以看见。

也不一定必须在空目录下创建Git仓库，选择一个已经有东西的目录也是可以的。不过，不建议你使用自己正在开发的公司项目来学习Git，否则造成的一切后果概不负责。

git add +文件名 添加指定文件 （或者git add . 添加所有文件）

git status 查看当前状态

（3）.忽略文件

创建一个名为.gitignore的特殊文件，（这个文件是以局点打头，没有扩展名，并且让在其中添加下面一行内容）

(4). 提交文件

git commit -m "我写的内容原因" (注意：提交原因必须写，否则不能推送)

(5).git pull：下载远程仓库的最新信息到本地仓库

(6）推送文件到服务器, 将本地的仓库信息推送到远程仓库提交时如果远程仓库有其它人提交的最新代码, 必须先pull, 再提交

git push 或者git push origin(服务器项目默认名字) master

6. SSH-keygen用法

很多朋友在用github管理项目的时候，都是直接使用https url克隆到本地，当然也有有些人使用 SSH url 克隆到本地。然而，为什么绝大多数人会使用https url克隆呢?

这是因为，使用https url克隆对初学者来说会比较方便，复制https url 然后到 git Bash 里面直接用clone命令克隆到本地就好了。而使用 SSH url 克隆却需要在克隆之前先配置和添加好 SSH key 。

因此，如果你想要使用 SSH url 克隆的话，你必须是这个项目的拥有者。否则你是无法添加 SSH key 的。

https 和 **SSH** 的区别：

1、前者可以随意克隆github上的项目，而不管是谁的；而后者则是你必须是你要克隆的项目的拥有者或管理员，且需要先添加 SSH key ，否则无法克隆。

2、https url 在push的时候是需要验证用户名和密码的；而 SSH 在push的时候，是不需要输入用户名的，如果配置SSH key的时候设置了密码，则需要输入密码的，否则直接是不需要输入密码的。

在 **github** 上添加 **SSH key** 的步骤：

1、首先需要检查你电脑是否已经有 **SSH key**

运行 git Bash 客户端，输入如下代码：

```
$ cd ~/.ssh
$ ls
```

这两个命令就是检查是否已经存在 id_rsa.pub 或 id_dsa.pub 文件，如果文件已经存在，那么你可以跳过步骤2，直接进入步骤3。

2、创建一个 **SSH key**

```
$ ssh-keygen -t rsa -C "your_email@example.com"
```

代码参数含义：

-t 指定密钥类型，默认是 rsa ，可以省略。
-C 设置注释文字，比如邮箱。
-f 指定密钥文件存储文件名。

以上代码省略了 -f 参数，因此，运行上面那条命令后会让你输入一个文件名，用于保存刚才生成的 SSH key 代码，如：

```
Generating public/private rsa key pair.
# Enter file in which to save the key (/c/Users/you/.ssh/id_rsa): [Press enter]
```

当然，你也可以不输入文件名，使用默认文件名（推荐），那么就会生成 id_rsa 和 id_rsa.pub 两个秘钥文件。

接着又会提示你输入两次密码（该密码是你push文件的时候要输入的密码，而不是github管理者的密码），

当然，你也可以不输入密码，直接按回车。那么push的时候就不需要输入密码，直接提交到github上了，如：

```
Enter passphrase (empty for no passphrase):
# Enter same passphrase again:
```

接下来，就会显示如下代码提示，如：

```
Your identification has been saved in /c/Users/you/.ssh/id_rsa.
# Your public key has been saved in /c/Users/you/.ssh/id_rsa.pub.
# The key fingerprint is:
# 01:0f:f4:3b:ca:85:d6:17:a1:7d:f0:68:9d:f0:a2:db your_email@example.com
```

当你看到上面这段代码的收，那就说明，你的 SSH key 已经创建成功，你只需要添加到github的SSH key上就可以了。

3、添加你的 SSH key 到 github上面去

a、首先你需要拷贝 id_rsa.pub 文件的内容，你可以用编辑器打开文件复制，也可以用git命令复制该文件的内容，如：

```
$ pbcopy < ~/.ssh/id_rsa.pub  //Mac系统复制命令
$ clip < ~/.ssh/id_rsa.pub   //Window系统复制命令
```

b、登录你的github账号，从又上角的设置（ Account Settings ）进入，然后点击菜单栏的 SSH key 进入页面添加 SSH key。

c、点击 Add SSH key 按钮添加一个 SSH key 。把你复制的 SSH key 代码粘贴到 key 所对应的输入框中，记得 SSH key 代码的前后不要留有空格或者回车。当然，上面的 Title 所对应的输入框你也可以输入一个该 SSH key 显示在 github 上的一个别名。默认的会使用你的邮件名称。

4、测试一下该SSH key

在git Bash 中输入以下代码

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

当你输入以上代码时，会有一段警告代码，如：

```
The authenticity of host 'github.com (207.97.227.239)' can't be established.
# RSA key fingerprint is 16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48.
# Are you sure you want to continue connecting (yes/no)?
```

这是正常的，你输入 yes 回车既可。如果你创建 SSH key 的时候设置了密码，接下来就会提示你输入密码，如：

```
Enter passphrase for key '/c/Users/Administrator/.ssh/id_rsa':
```

当然如果你密码输错了，会再要求你输入，知道对了为止。

注意：输入密码时如果输错一个字就会不正确，使用删除键是无法更正的。

密码正确后你会看到下面这段话，如：

```
Hi username! You've successfully authenticated, but GitHub does not
# provide shell access.
```

如果用户名是正确的,你已经成功设置SSH密钥。如果你看到 “access denied”，者表示拒绝访问，那么你就需要使用 https 去访问，而不是 SSH 。

Github， Git， GitLab 有什么区别

Github 和 Git 是两回事。

Git是版本控制系统，Github是在线的基于Git的代码托管服务。

GitHub和GitLab都是基于web的版本控制界面，服务于互联网，Github可以直接注册使用，Gitlab需要部署到服务器。

GitLab创建的项目的默认属性是Private（私人的），当然，你也可以选择Public（公开的）或Internal（内部的）。

GitHub 和 GitLab 都是基于 web 的 Git 仓库，使用起来二者差不多，它们都提供了分享开源项目的平台，

为开发团队提供了存储、分享、发布和合作开发项目的中心化云存储的场所。

GitHub 作为开源代码库，拥有超过 900 万的开发者用户，目前仍然是最火的开源项目托管平台，GitHub 同时

提供公共仓库和私有仓库，但如果使用私有仓库，是需要付费的。

GitLab 解决了这个问题，你可以在上面创建私人的免费仓库。

https://gitlab.com/users/sign_in

GitLab 让开发团队对他们的代码仓库拥有更多的控制，相比较 GitHub , 它有不少特色：

(1) 允许免费设置仓库权限；

(2) 允许用户选择分享一个 project 的部分代码；

(3) 允许用户设置 project 的获取权限，进一步提升安全性；

(4) 可以设置获取到团队整体的改进进度；

(5) 通过 innersourcing 让不在权限范围内的人访问不到该资源；

所以，从代码的私有性上来看，GitLab 是一个更好的选择。但是对于开源项目而言，GitHub 依然是代码托管的首选。