

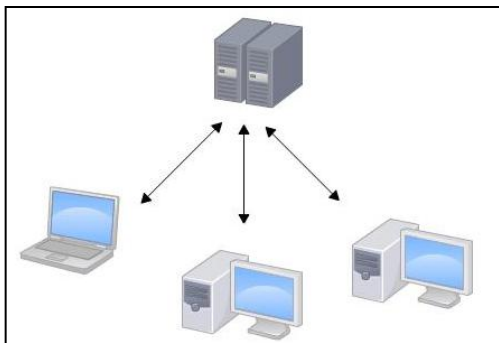
## 版本控制工具 GIT

### 1、Git 简介：

Git 是最流行的版本管理工具，其实 Git 和 linux 的创始人都是 linus，Git 也是为了 linux 代码的托管而开发的。最初 Git 刚开源那会进行了修改，并且 linus 通过合并请求后都是靠 linus 手工进行代码合并，后来随着社区的活跃度和其发展的速度迅猛，linus 力不从心了，当时出现就借助了 Bitkeeper 进行代码托管，当时这个也是收费的（考虑到 linux 的社区的强大，免费提供给了 linux 社区），社区中的众多大牛也总是会搞些事情，就开始尝试破解 Bitkeeper，后来也就因为这个原因，Bitkeeper 公司终止了 linux 的托管，linus 其实并不看好 svn 或者是 cvs 等，后来就利用了两周时间开发了 Git，Git 也就由此诞生。后来随着 github 的发展也是一火再火。

#### ➤ 集中式版本控制系统：

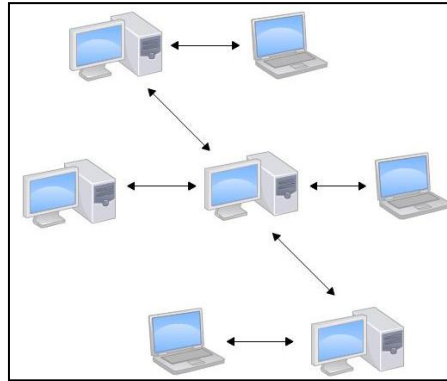
版本库是集中存放在中央服务器的，而干活的时候，用的都是自己的电脑，所以要先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器。中央服务器就好比是一个图书馆，你要改一本书，必须先从图书馆借出来，然后回到家自己改，改完了，再放回图书馆。



#### ➤ 分布式版本控制系统：

首先，分布式版本控制系统根本没有“中央服务器”，每个人的电脑上都是一个完整的版本库，这样，你工作的时候，就不需要联网了，因为版本库就在你自己的电脑上。既然每个人电脑上都有一个完整的版本库，那多个人如何协作呢？比方说你在自己电脑上改了文件 A，你的同事也在他的电脑上改了文件 A，这时，你们俩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。

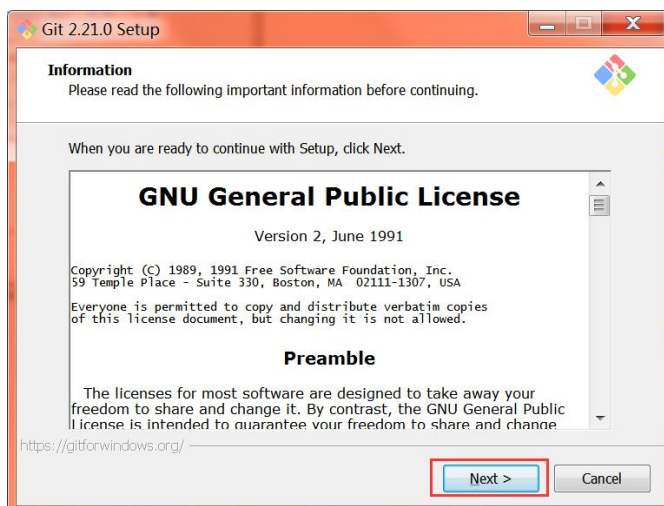
➤ 在实际使用分布式版本控制系统的时候，其实很少在两人之间的电脑上推送版本库的修改，因为可能你们俩不在一个局域网内，两台电脑互相访问不了，也可能今天你的同事病了，他的电脑压根没有开机。因此，分布式版本控制系统通常也有一台充当“中央服务器”的电脑，但这个服务器的作用仅仅是用来方便“交换”大家的修改，没有它大家也一样干活，只是交换修改不方便而已。



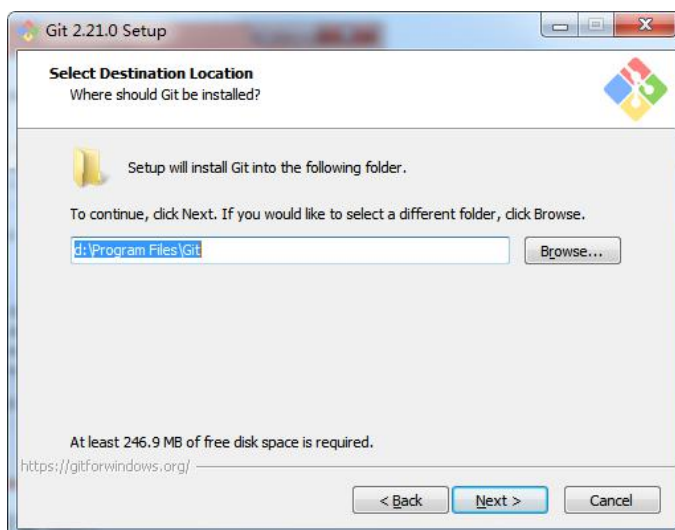
## 2、Git 安装

下载地址: <https://git-scm.com/downloads>

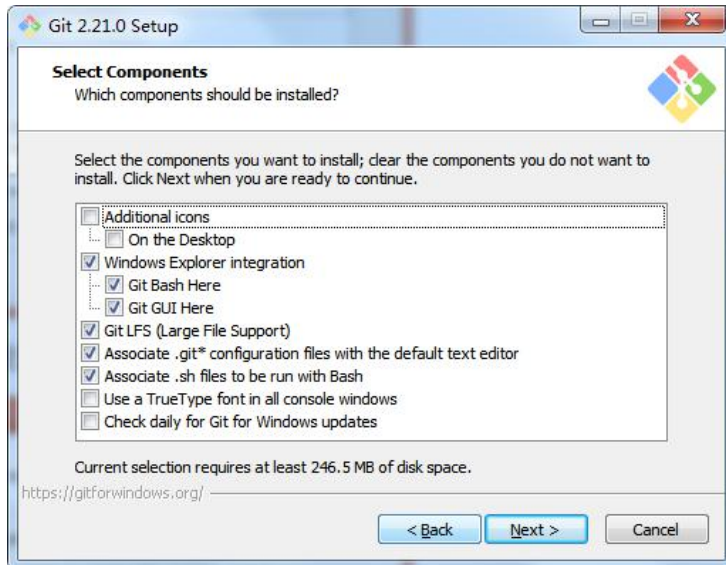
### 1、点击安装包安装 Git



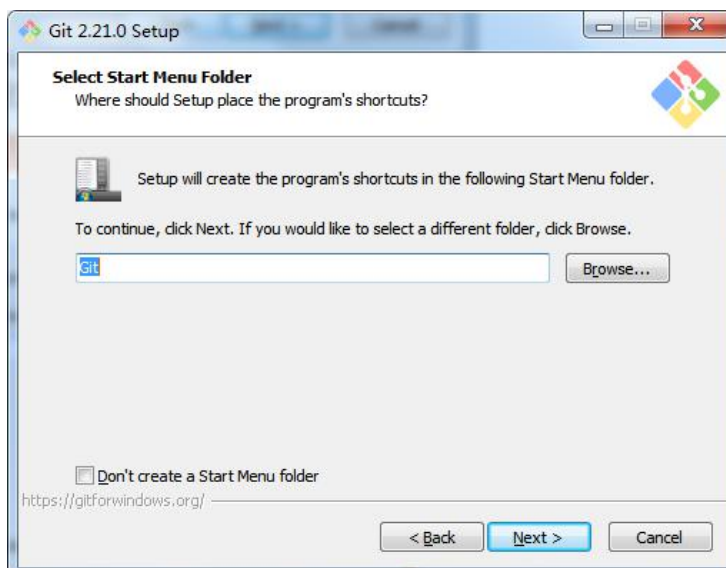
### 2、指定安装路径



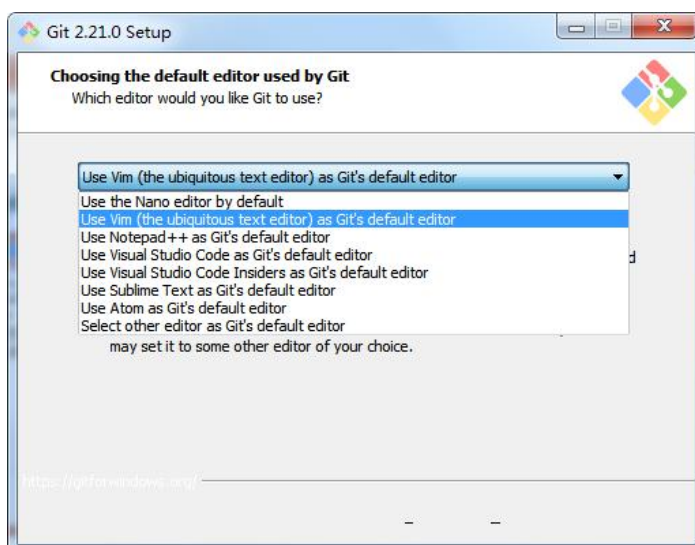
### 3、采用默认设置: 命令行模式+图形界面



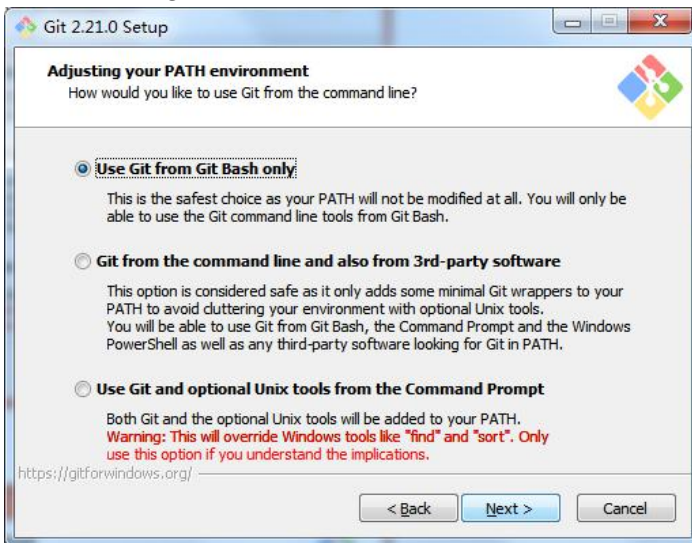
#### 4、开始菜单目录名设置 next



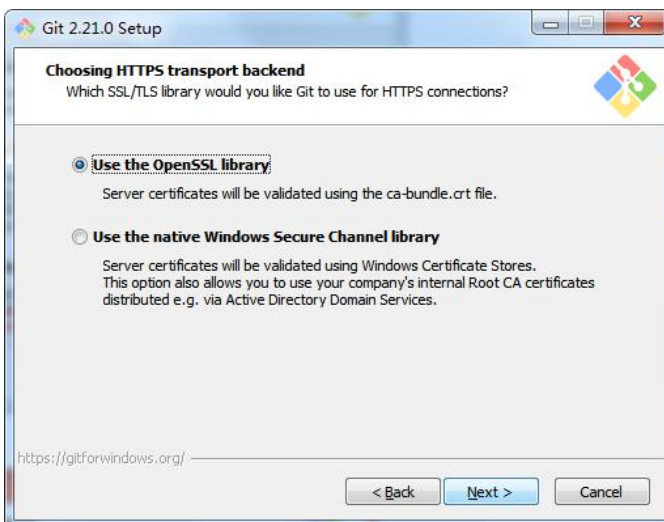
#### 5、设置默认编辑器



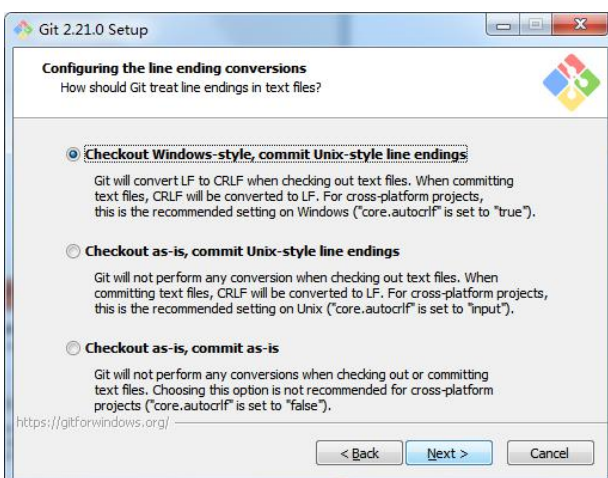
## 6、选择使用 git 的命令行工具



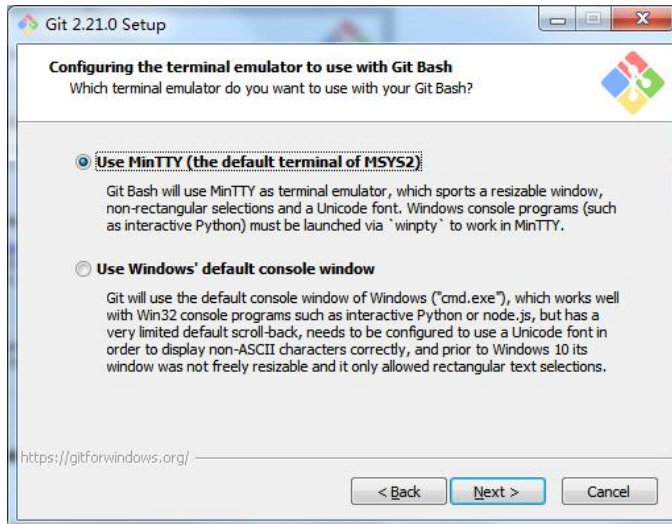
## 7、next



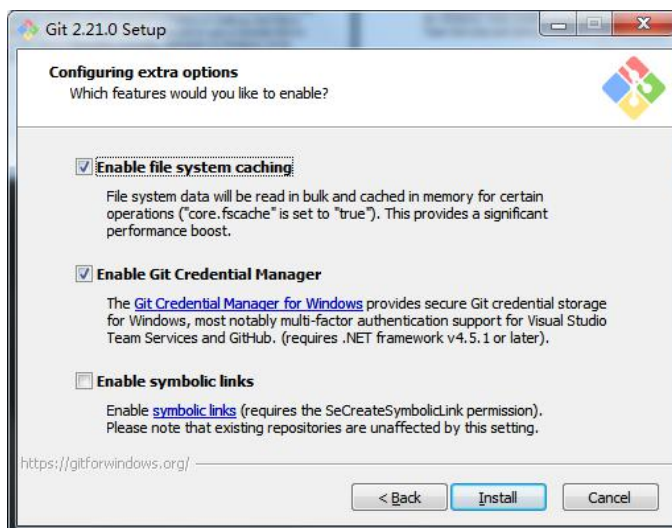
## 8、next



## 9、Next



10、install



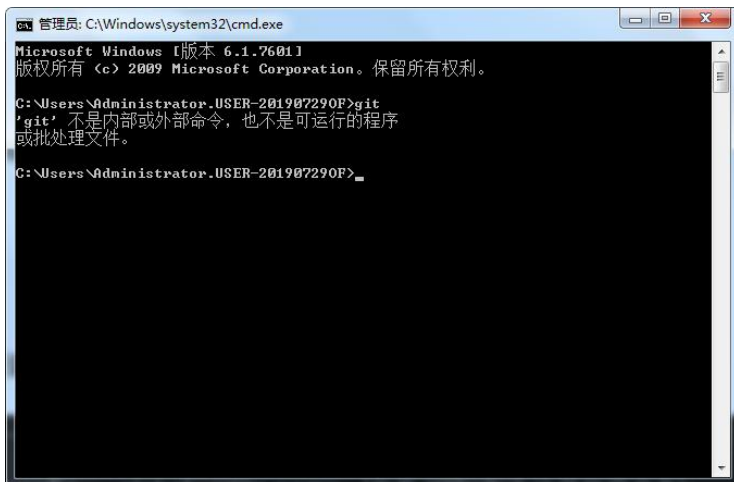
11、finish



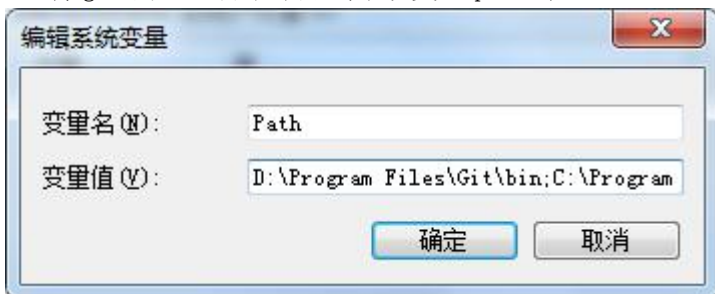


### 3、Git 配置

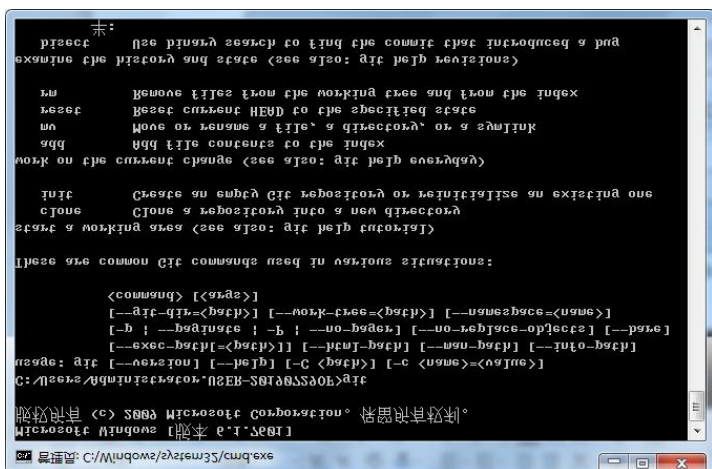
1) 安装完毕之后打开 cmd 输入 git，观察是否安装成功



2) 将 git 的 bin 目录添加到系统变量 path 中



3) 重新打开 cmd 输入 git，显示以下内容表示安装成功

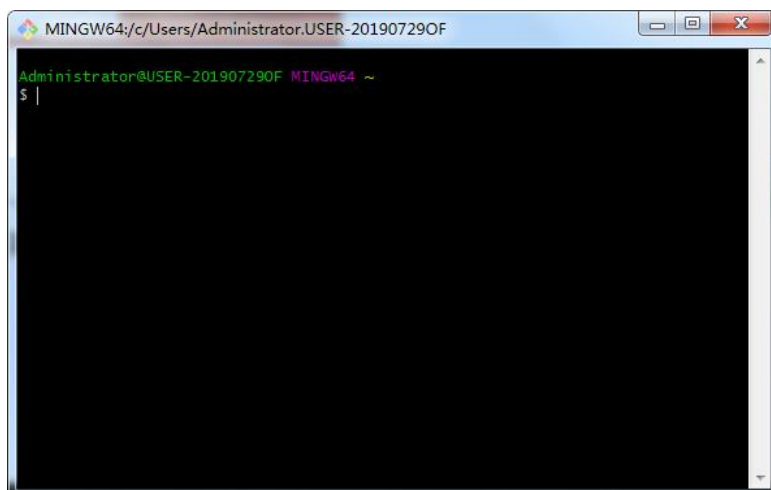
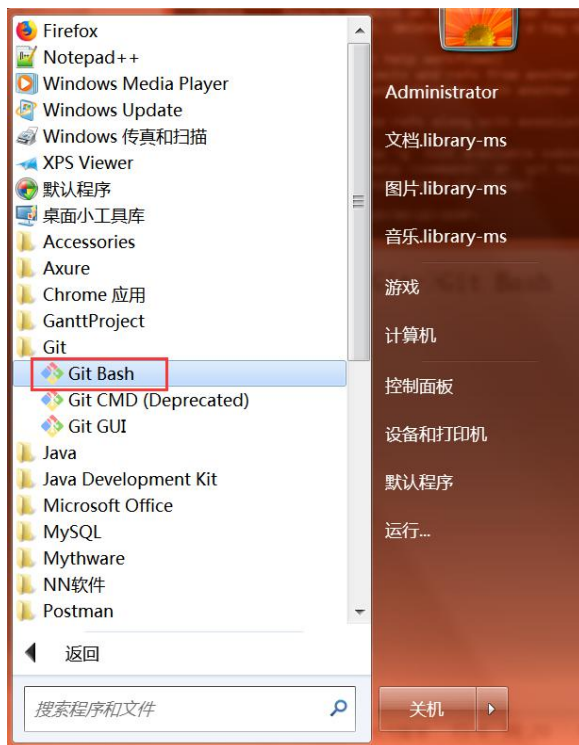


### 4、Git 初始化及仓库创建和操作

## 1、Git 安装之后需要进行一些基本信息设置

- a、设置用户名: `git config -- global user.name '你再 github 上注册的用户名';`
- b、设置用户邮箱: `git config -- global user.email '注册时候的邮箱';`
- c、查看是否配置成功: `git config -- list`

### ➤ 打开 Git bash

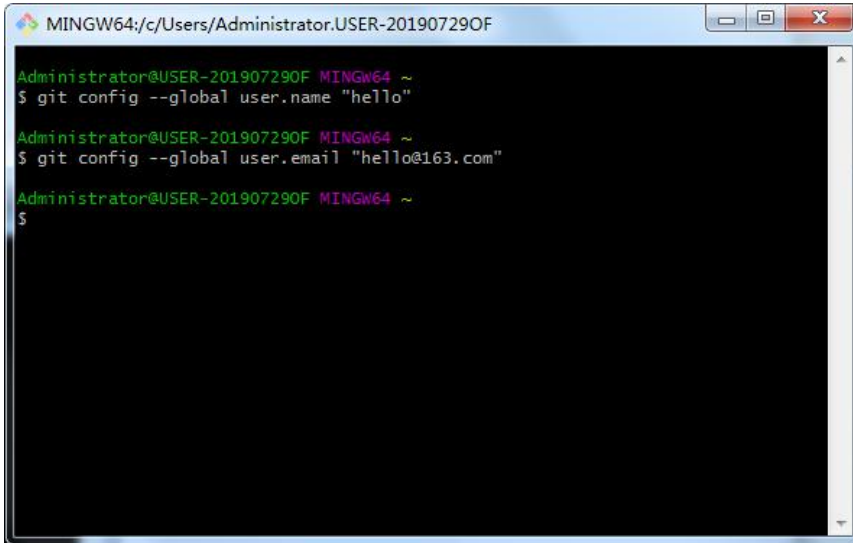


### ➤ 设置用户名和邮箱（注册）：

区分不同开发人员的身份，这里设置的签名和登录远程仓库（代码托管中心）的账号、密码没有任何关系。

`git config --global user.name "你的名字"`

`git config --global user.email "你的 Email"`

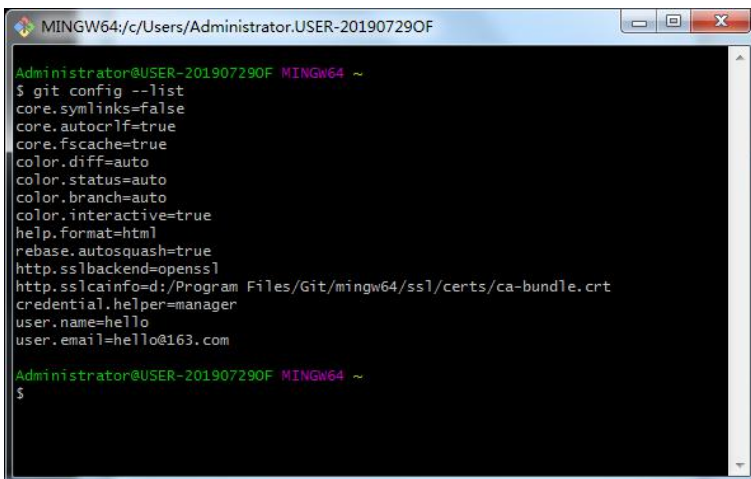


```

MINGW64:/c/Users/Administrator.USER-20190729OF
Administrator@USER-20190729OF MINGW64 ~
$ git config --global user.name "hello"
Administrator@USER-20190729OF MINGW64 ~
$ git config --global user.email "hello@163.com"
Administrator@USER-20190729OF MINGW64 ~
$

```

➤ 查看配置信息: `git config --list`

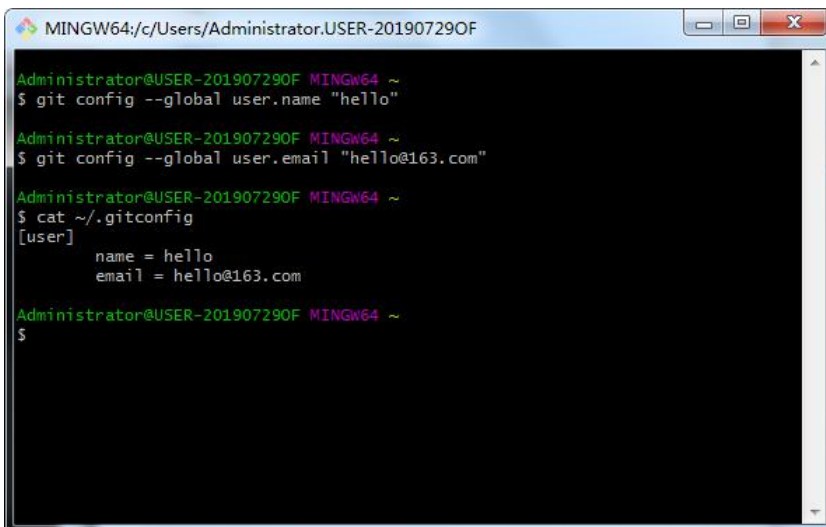


```

MINGW64:/c/Users/Administrator.USER-20190729OF
Administrator@USER-20190729OF MINGW64 ~
$ git config --list
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.sslbackend=openssl
http.sslcainfo=d:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
credential.helper=manager
user.name=hello
user.email=hello@163.com
Administrator@USER-20190729OF MINGW64 ~
$

```

➤ 保存位置: `~/.gitconfig`



```

MINGW64:/c/Users/Administrator.USER-20190729OF
Administrator@USER-20190729OF MINGW64 ~
$ git config --global user.name "hello"
Administrator@USER-20190729OF MINGW64 ~
$ git config --global user.email "hello@163.com"
Administrator@USER-20190729OF MINGW64 ~
$ cat ~/.gitconfig
[user]
  name = hello
  email = hello@163.com
Administrator@USER-20190729OF MINGW64 ~
$

```

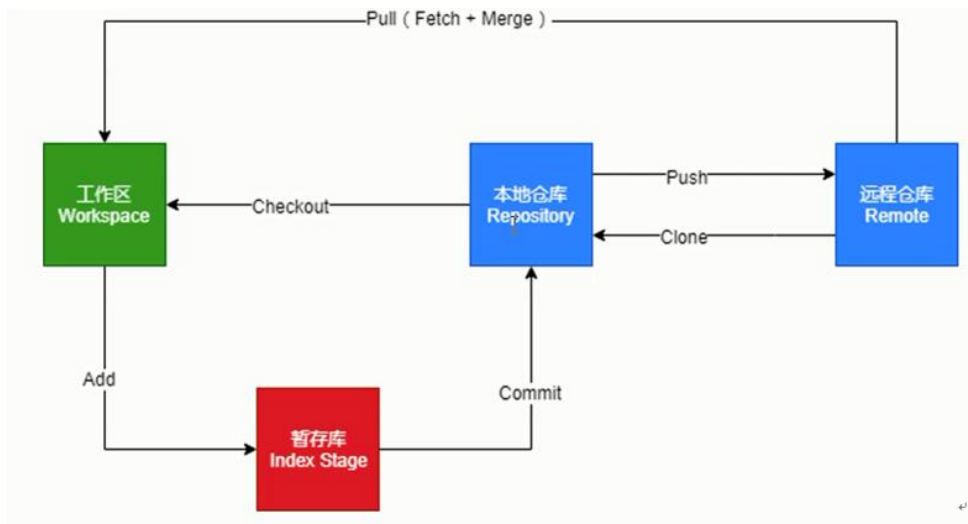
## 5、GIT 的基本工作流程



## 1) Git 的工作区域



## 2) 向仓库中添加文件的流程

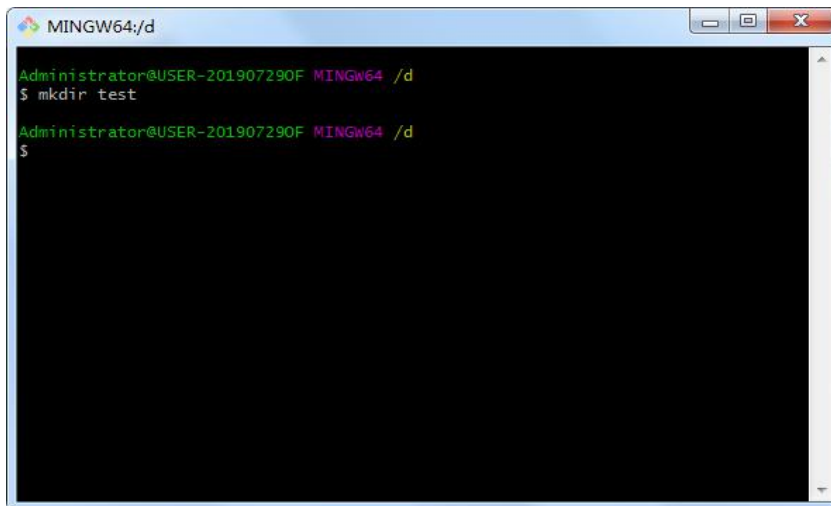


## 1、初始化一个新的 git 仓库

### a、创建文件夹

方法一：可以鼠标右击-->点击新建文件夹 test

方法二：使用 git 新建：\$ mkdir test

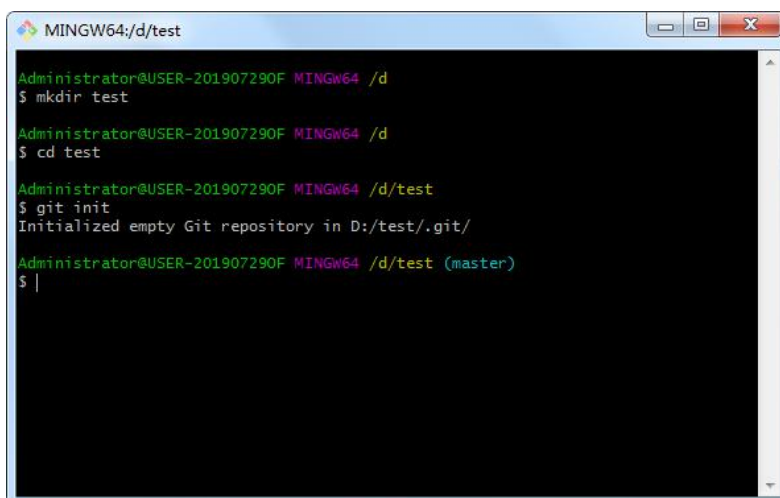


```
MINGW64:/d
Administrator@USER-201907290F MINGW64 /d
$ mkdir test
Administrator@USER-201907290F MINGW64 /d
$
```

### b、在文件内初始化 git（创建 git 仓库）

方法一：直接输入 \$ cd test

方法一：点击 test 文件下进去之后->鼠标右击选择 Git Bash Here->输入\$ git init



```
MINGW64:/d/test
Administrator@USER-201907290F MINGW64 /d
$ mkdir test
Administrator@USER-201907290F MINGW64 /d
$ cd test
Administrator@USER-201907290F MINGW64 /d/test
$ git init
Initialized empty Git repository in D:/test/.git/
Administrator@USER-201907290F MINGW64 /d/test (master)
$ |
```

这时仓库目录下多了一个.git的目录（如果没有，可能你的 windows 设置不显示隐藏的文件、文件夹，在文件夹选项中修改一下设置就好了。），这个目录是 Git 来跟踪管理版本库的。

## 2、向仓库中添加文件

方法一：用打开编辑器新建 **index.html** 文件

方法二：使用 **git** 命令。**\$ touch '文件名'**，然后把文件通过**\$ git add '文件名'**添加到暂存区，最后提交操作

```

MINGW64:/d/test
$ cd test

Administrator@USER-201907290F MINGW64 /d/test
$ git init
Initialized empty Git repository in D:/test/.git/

Administrator@USER-201907290F MINGW64 /d/test (master)
$ touch 'index.html'

Administrator@USER-201907290F MINGW64 /d/test (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    index.html

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

Administrator@USER-201907290F MINGW64 /d/test (master)
$ |
    
```

### ➤ 将文件添加到暂存区

```

MINGW64:/d/test
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    index.html

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

Administrator@USER-201907290F MINGW64 /d/test (master)
$ git add 'index.html'

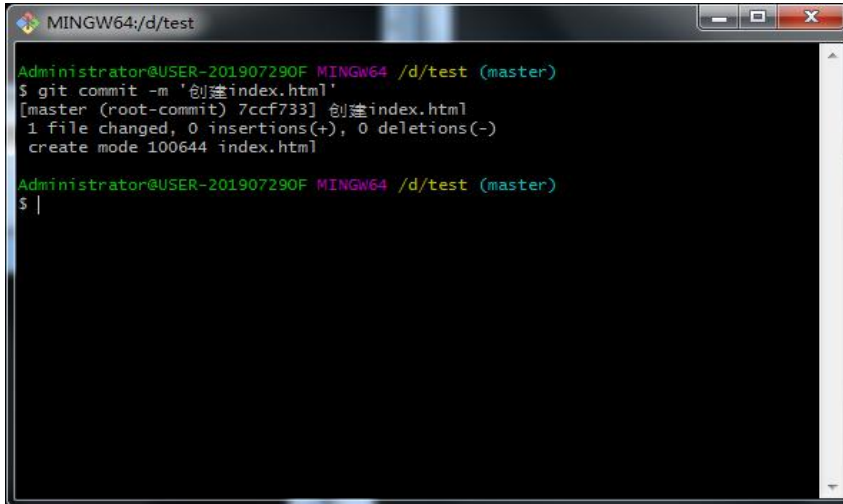
Administrator@USER-201907290F MINGW64 /d/test (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   index.html

Administrator@USER-201907290F MINGW64 /d/test (master)
$ |
    
```

### ➤ 将文件从暂存区提交到仓库



```

MINGW64:/d/test
Administrator@USER-201907290F MINGW64 /d/test (master)
$ git commit -m '创建index.html'
[master (root-commit) 7ccf733] 创建index.html
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 index.html

Administrator@USER-201907290F MINGW64 /d/test (master)
$ |

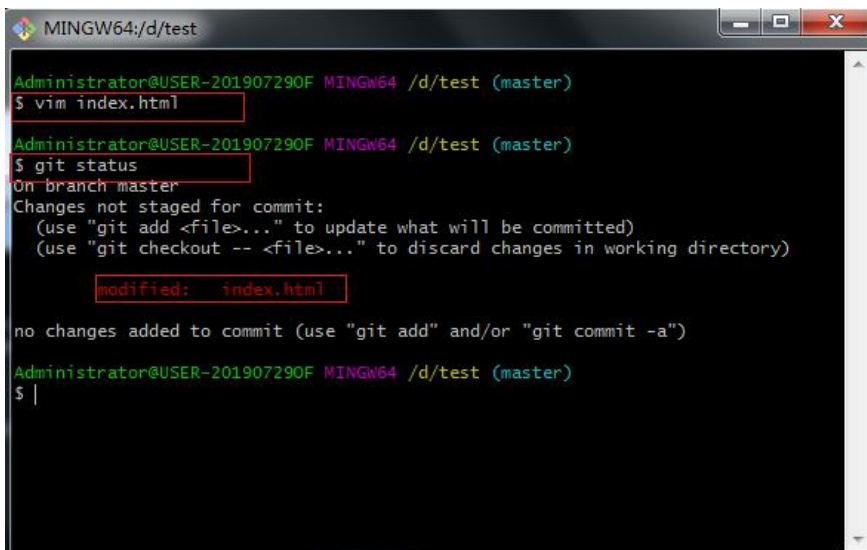
```

### 3、修改仓库文件

方法一：用编辑器打开 **index.html** 进行修改

方法二：使用 **git** 命令。**\$ vim '文件名'**，然后在中间写内容，最后提交操作

修改后，查看状态



```

MINGW64:/d/test
Administrator@USER-201907290F MINGW64 /d/test (master)
$ vim index.html

Administrator@USER-201907290F MINGW64 /d/test (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)

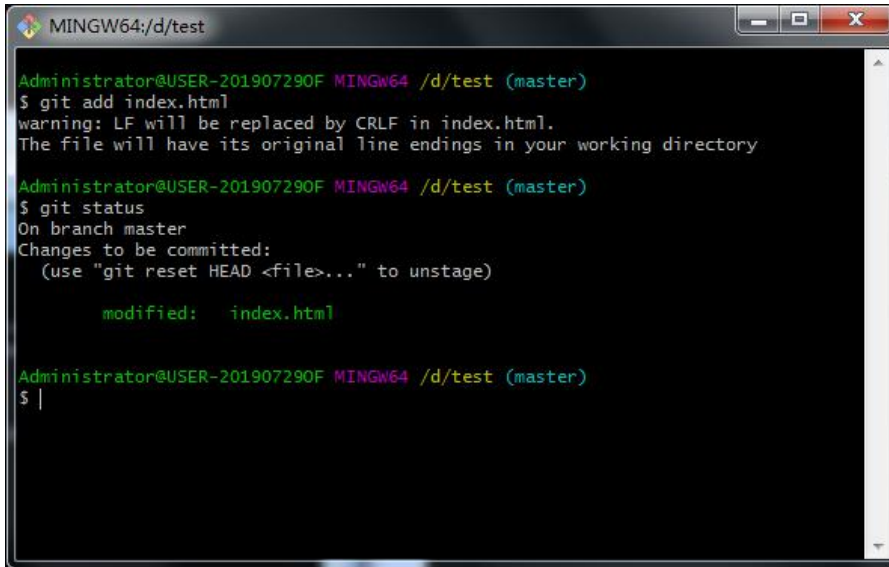
    modified:   index.html

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

Administrator@USER-201907290F MINGW64 /d/test (master)
$ |

```

添加到暂存区，查看状态



```
MINGW64:/d/test

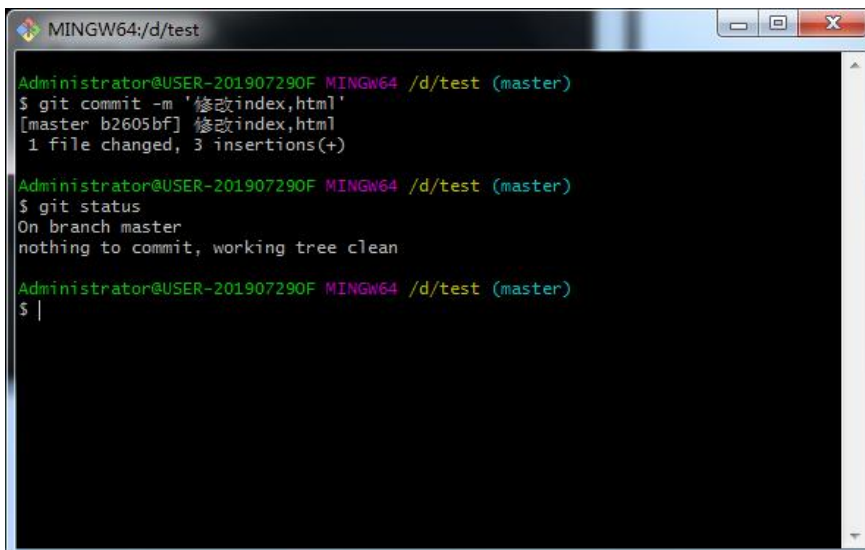
Administrator@USER-201907290F MINGW64 /d/test (master)
$ git add index.html
warning: LF will be replaced by CRLF in index.html.
The file will have its original line endings in your working directory

Administrator@USER-201907290F MINGW64 /d/test (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   index.html

Administrator@USER-201907290F MINGW64 /d/test (master)
$ |
```

把修改的内容提交到 git 仓库



```
MINGW64:/d/test

Administrator@USER-201907290F MINGW64 /d/test (master)
$ git commit -m '修改index.html'
[master b2605bf] 修改index.html
1 file changed, 3 insertions(+)

Administrator@USER-201907290F MINGW64 /d/test (master)
$ git status
On branch master
nothing to commit, working tree clean

Administrator@USER-201907290F MINGW64 /d/test (master)
$ |
```

删除文件，查看状态



```

MINGW64:/d/test
$ git commit -m '修改index.html'
[master b2605bf] 修改index.html
1 file changed, 3 insertions(+)

Administrator@USER-201907290F MINGW64 /d/test (master)
$ git status
On branch master
nothing to commit, working tree clean

Administrator@USER-201907290F MINGW64 /d/test (master)
$ git rm index.html
rm 'index.html'

Administrator@USER-201907290F MINGW64 /d/test (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    index.html

Administrator@USER-201907290F MINGW64 /d/test (master)
$ |

```

提交

```

MINGW64:/d/test
Administrator@USER-201907290F MINGW64 /d/test (master)
$ git commit -m '删除index.html'
[master 2f84d09] 删除index.html
1 file changed, 3 deletions(-)
delete mode 100644 index.html

Administrator@USER-201907290F MINGW64 /d/test (master)
$ |

```

## 6、Git 的常用命令

- git init 将当前目录初始化为 git 本地仓库
- git add 添加指定文件到暂存区
- git rm 将指定目录或文件移出暂存区
- git commit file -m “提交日志” 提交至本地仓库
- git status 命令查看目前仓库状态
- git log 查看当前分支下所有提交日志
  - 可以查看到各个版本的 hash 值索引
  - head 指针指向当前的版本
- git log --oneline 单行显示日志
- git reflog 显示版本历史、分支切换历史
- git reset --hard HEAD^ 回退到上一个版本

- `git reset --hard` 版本号

`git push -set-upstream origin master` 上传新分支至远程

`git clone` 基于远程仓库克隆至本地

## 7、分支操作

几乎每一种版本控制系统都以某种形式支持分支。分支就是从主线上分离出来进行另外的操作，而又不影响主线，主线又可以继续干它的事，最后分支做完事后合并到主线上而分支的任务完成可以删掉。分支功能解决了正在开发版本与上线版本稳定性冲突的问题。

如果某个分支开发失败不会对其它分支造成影响。

`git branch` 【查看当前分支】

`git branch` 分支名 【基于当前分支新建分支】

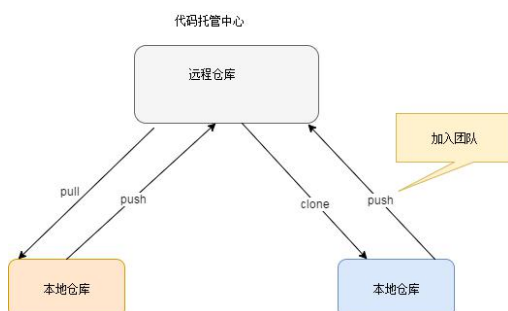
`git checkout` 【切换分支】

`git merge` 【合并分支】

`git branch -D` 分支名 【删除分支】

## 8、Git 远程仓库

- 局域网
  - GitLab 服务器
- 外网仓库
  - Github <https://github.com/>
  - 码云 <https://gitee.com>
  - Coding <https://coding.net>
- 本地仓库与远程仓库的交互方式



- 命令
  - 关联远程仓库: `git remote add origin` 【远程仓库地址】
  - 查看关联的远程仓库: `git remote -v`
  - 删除远程仓库
    - ◆ `git remove` 地址别名
    - ◆ 地址别名指的是上面的 `origin`，也可以是其他别名。
  - 推送代码到远程仓库

- ◆ git push **origin** master
- ◆ git push -u origin master
- ◆ 如果加上了 -u 参数，以后再次提交时可以省略地址别名和分支名称，直接执行下面命令进行提交 git push
  - 拉取远程仓库的代码
    - ◆ git pull origin master
  - 克隆项目
    - ◆ git clone 项目地址 项目别名
    - ◆ 上面命令项目别名是可选的，相当于给项目根文件夹重新命名。

## 9、IDEA 安装 Git 插件，团队协同开发

### 安装 Gitee 插件

1、【File】-【Settings】-【Plugins】，查看 gitee 插件并安装

安装 Gitee 插件

安装完成后，重启 IDEA

重启 IDEA

### 2、添加 Gitee 账户

【File】-【Settings】-【Version Control】-【Gitee】，点击【Add account】添加账户

添加 Gitee 账户

输入账号和密码，点击【Log In】

输入账号和密码

登录成功后会显示账户信息

### 3、IDEA 配置 Git 环境：

【file】->【Settings】->【Version Control】->【Git】->【配置本地 git.exe 安装路径】->【Apply】->【ok】

### 4、IDEA 项目分享至 Gitee

选中要同步的项目，点击菜单【VCS】-【Import into Version Control】-【Share Project on Gitee】

分享至 Gitee

点击【Share】按钮

分享成功

登录 Gitee 可看到项目

## 5、Gitee 项目克隆至 IDEA

**【VCS】 - 【Checkout from Version Control】 - 【Git】**

克隆 Gitee 中项目

输入 Gitee 项目的 URL，点击 **【Clone】**，开始克隆

## 6、Gitee 团队协作

拉取代码

**【VCS】 - 【Update Project】**

*注：提交代码前，应先拉取。若拉取时出现代码冲突，应先解决冲突后再提交*

**【VCS】 - 【Commit】**

## 7、代码冲突

当在提交或拉取代码时，出现冲突会弹出对话框，选中冲突的文件，点击 **【Merge】**

解决代码冲突：修改中间的代码，可根据情况选择 **【Accept Left】** 或 **【Accept Right】**；亦或者手动修改冲突代码后点击 **【Apply】**

当所有冲突文件解决后，重新提交即可