

# Git教程

---

Git是什么？

git 读[git] gèi tè

Git是目前世界上最先进的分布式版本控制系统（没有之一）。

Git有什么特点？简单来说就是：高端大气上档次！

## Git教程

[什么是VCS版本控制系统](#)

[VCS软件](#)

[Git软件安装](#)

[下载](#)

[初始配置](#)

[Git使用](#)

[本地使用](#)

[体验](#)

[Git实现的原理](#)

[回退操作](#)

[仓库状态查看](#)

[仓库提交日志查看](#)

[回退到指定仓库](#)

[从暂存区恢复](#)

[删除暂存区的文件](#)

[删除文件](#)

[练习](#)

[分支操作](#)

[创建与合并分支](#)

[实战](#)

[分支冲突解决](#)

[远程仓库使用](#)

[码云使用](#)

[点击这里, 新建仓库](#)

[创建一个仓库](#)

[找到你的仓库](#)

[找到你仓库的地址](#)

[帮助文档](#)

[使用方式](#)

[SSH方式](#)

[.gitignore](#)

[GitHub使用](#)

[注册你的账号](#)

[创建新的仓库](#)

[查看你的所有仓库](#)

[找到下载地址](#)

[SSH方式的秘钥设置](#)

[可视化客户端的使用](#)

## 什么是VCS版本控制系统

---

在实际开发时,往往遇到以下场景

- 换了台电脑,需要把代码拷贝过去
- 新增代码运行出错,找不出BUG,希望恢复到之前正确的状态
- 周一上班时,忘记上一周的具体工作进度
- 多人协作开发时,自动合并多人的代码到一起,并且识别冲突情况
- 分享自己的代码给其他人

**VCS系统:** 全称Version Control System, 版本控制系统, 用于项目中的存储, 共享, 合并, 历史回退功能等.

## VCS软件

常用的VSC软件有:

- **VCS:** 2000年之前
- **SVN:** 2010年之前
- **GIT:** 2010年之后

## Git软件安装

### 下载

可以从 <https://git-scm.com/downloads> 获得Git在Windows/macOS/Linux三个操作系统相关的安装包.

下载完毕后, 双击打开安装文件. 按照默认选项一直安装到最后一步即可



The screenshot shows the 'Downloads' section of the Git website. It features three large buttons for different operating systems: Mac OS X, Windows, and Linux/Unix. The 'Windows' button is highlighted with a red border. Below the buttons, there is a note: 'Older releases are available and the Git source repository is on GitHub.' To the right of the download buttons is a monitor icon displaying the text 'Latest source Release 2.23.0' and a 'Download 2.23.0 for Windows' button.

### GUI Clients

Git comes with built-in GUI tools (`git-gui`, `gitk`), but there are several third-party tools for users looking for a platform-specific experience.

[View GUI Clients →](#)

### Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

[View Logos →](#)

### Git via Git

If you already have Git installed, you can get the latest development version via Git itself:

```
git clone https://github.com/git/git
```

You can also always browse the current contents of the git repository using the [web interface](#).

安装完毕后, 可以通过**CMD**命令行工具, 查看是否安装成功.

```
# 查看安装的git软件版本号  
> git --version
```

```
C:\Users\小新>git --version  
git version 2.23.0.windows.1
```

## 初始配置

安装完毕后, 还需要最后一步设置, 在命令行输入

config 读 [kən'fig] kēn fèi gē 配置

global 读 ['gləʊbl] gé lōu bū ōu 全局

```
# 设置你的用户名  
> git config --global user.name "你的姓名"  
  
# 设置你的邮箱地址  
> git config --global user.email "你的邮箱地址"
```

### 名字最好用英文, 中文会有乱码的风险

```
C:\Users\小新>git config --global user.name "xiaoxin"  
C:\Users\小新>git config --global user.email '736907613@qq.com'
```

因为git是多人协作开发的软件, 每个用户都可以修改代码. 所以每个用户必须设置自己的用户名和邮箱.

这样提交的修改就会带有独特的标识, 便于查看改动人的信息.

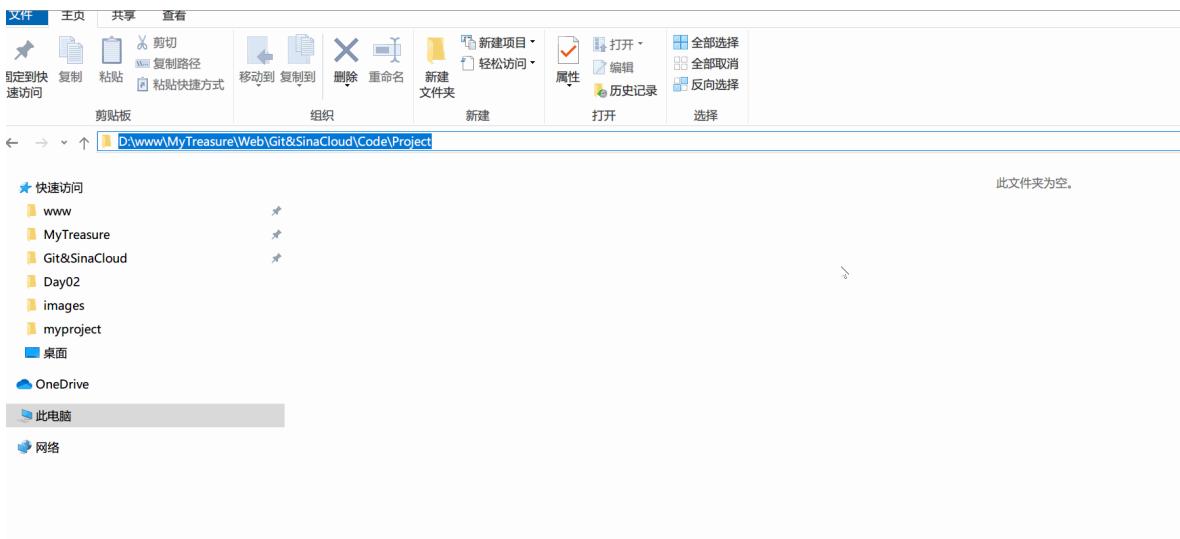
注意 `git config` 命令的`--global`参数, 用了这个参数, 表示你这台机器上所有的Git仓库都会使用这个配置, 当然也可以对某个仓库指定不同的用户名和Email地址

## Git使用

### 本地使用

在硬盘上新建一个文件夹或者找到你之前项目所在的文件夹根目录, 通过**命令行工具**打开文件夹.

在硬盘上找到目标文件夹, 在地址栏中输入 `cmd` 然后 **回车**, 即可快速在目录下打开命令行工具.



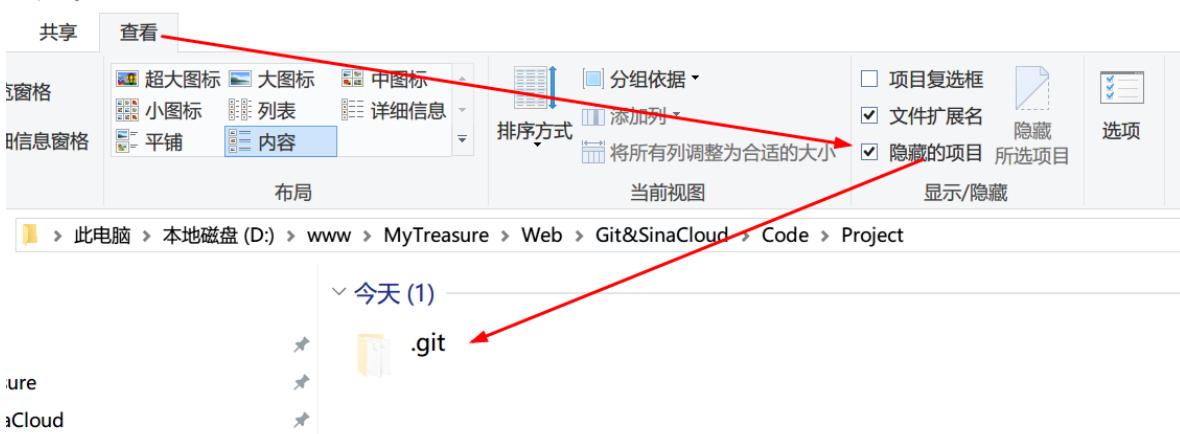
初始化命令可以把当前目录变成Git管理的仓库:

init 读 [ɪ'nɪt] yī nì tē 初始化

```
> git init
```

```
D:\www\MyTreasure\Web\Git&SinaCloud\Code\Project>git init
Initialized empty Git repository in D:/www/MyTreasure/Web/Git&SinaCloud/Code/Project/.git/
```

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)，细心的读者可以发现当前目录下多了一个 `.git` 的目录，这个目录是Git来跟踪管理版本库的，没事千万不要手动修改这个目录里面的文件，不然改乱了，就把Git仓库给破坏了。



## 体验

**千万不要用windows自带的记事本来编辑文件**，原因是Microsoft开发记事本的团队使用了一个非常弱智的行为来保存UTF-8编码的文件，他们自作聪明地在每个文件开头添加了0xefbbbb (十六进制) 的字节，你会遇到很多不可思议的问题，比如，网页第一行可能会显示一个“?”，明明正确的程序一编译就报语法错误，等等，都是由记事本的弱智行为带来的。这里推荐使用 `Sublime`, `VSCode`, `Notepad++` 等专业的编程软件来编辑文件。

在项目目录下创建一个 `readme.txt` 文件，文件内容如下:

```
Hello Everyone!
I'm Teacher JiYingXin.
Hope You A Happy Day!
```

接下来我们需要通过命令行来告诉Git工具, 此处新增了一个 `readme.txt` 文件.

add 读 [æd] aī dē 添加

```
> git add readme.txt
```

```
D:\www\MyTreasure\Web\Git&SinaCloud\Code\Project>git add readme.txt
```

```
D:\www\MyTreasure\Web\Git&SinaCloud\Code\Project>
```

执行命令以后, 没有任何提示, 这就对了. Unix的哲学是"没有消息就是好消息", 说明添加成功!

然后用命令告诉Git, 把文件提交到仓库

commit 读 [kə'mɪt] kē mì tè 把...交给, 提交

```
> git commit -m "本次提交操作的说明"
```

**注意:** 信息必须写在双引号中, 单引号有可能导致错误

```
D:\www\MyTreasure\Web\Git&SinaCloud\Code\Project>git add readme.txt  
D:\www\MyTreasure\Web\Git&SinaCloud\Code\Project>git commit -m "新建readme文件, 写入一些基本信息"  
[master (root-commit) 6f3d501] 新建readme文件, 写入一些基本信息  
 1 file changed, 3 insertions(+)  
 create mode 100644 readme.txt  
D:\www\MyTreasure\Web\Git&SinaCloud\Code\Project>
```

这里 `1 file changed`: 代表有一个文件被改动了.

`3 insertions(+)`: 代表新增了3行代码

**练习:** 随意新建3个文件, 依次执行两个命令, 保存到本地仓库中.

---

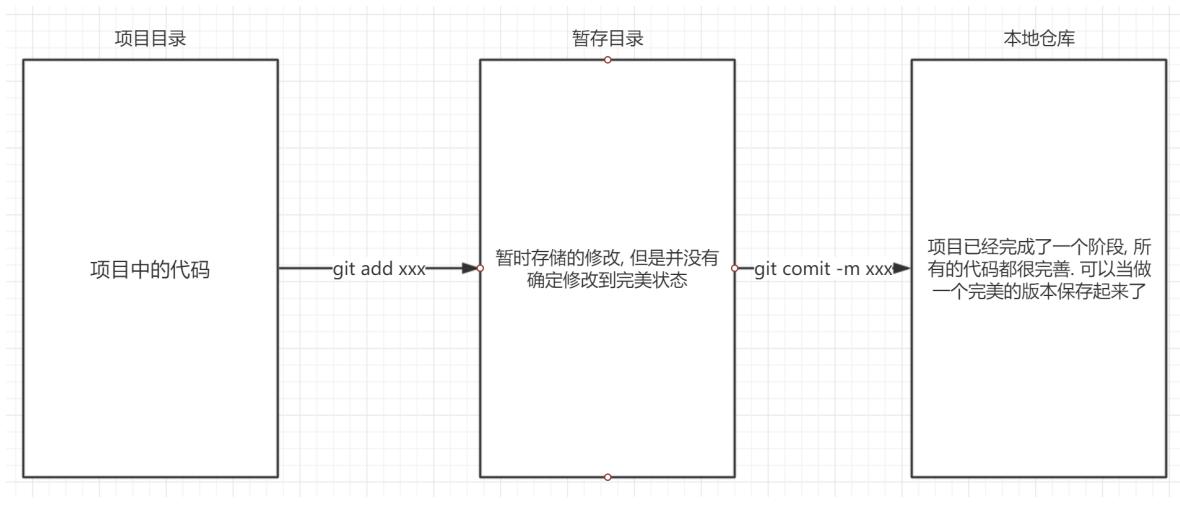
通过 `git add` 命令只能一个个文件进行暂存, 当文件数量增多是就非常不便了.

此时可以使用通配符来一起暂存

```
> git add .
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git add .
```

## Git实现的原理



## 回退操作

### 仓库状态查看

查看当前git仓库的状态

status ['steɪtəs] 状态

modified ['mɒdɪfaɪd] 修改的

```
> git status
```

```
D:\www\MyTreasure\Web\Git&SinaCloud\Code\Project>git add readme.txt
```

```
D:\www\MyTreasure\Web\Git&SinaCloud\Code\Project>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   readme.txt
```

```
D:\www\MyTreasure\Web\Git&SinaCloud\Code\Project>■
```

此处绿色提示, 代表有一个暂存文件 `readme.txt` 处于被修改状态.

---

此时我们再次修改 `readme.txt` 文件内容, 随意增加一些内容, 再次运行

```
> git status
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   readme.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   readme.txt
```

```
D:\www\MyTreasure\Web\Git\Code\Project>
```

此时会发现有红色文字 `modified: readme.txt` 出现, 这代表有修改但是**没有暂存起来**.

查看相对于已暂存文件, 具体修改的内容

```
> git diff readme.txt
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git diff readme.txt
diff --git a/readme.txt b/readme.txt
index 2be7f2c..08c144b 100644
--- a/readme.txt
+++ b/readme.txt
@@ -2,4 +2,5 @@ Hello Everyone!
 I'm Teacher JiYingXin.
 Hope You A Happy Day!

-New Messages-
\ No newline at end of file
+New Messages-
+hahaha
```

```
D:\www\MyTreasure\Web\Git\Code\Project>
```

这里绿色文字代表新增内容, 红色文字代表删除的内容.

```
# 保存新的修改到暂存文件中
> git add readme.txt
```

```
# 查看当前git状态
> git status
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git add readme.txt
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   readme.txt
```

```
D:\www\MyTreasure\Web\Git\Code\Project>■
```

没有红色文字, 代表 `readme.txt` 目前所有修改都被暂存起来了.

提交所有暂存的修改到仓库中, 形成新的版本保存起来

```
> git commit -m "版本的相关描述信息"
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git commit -m "reamde.txt文件再次修改"
[master 8fa150e] reamde.txt文件再次修改
 1 file changed, 4 insertions(+), 1 deletion(-)
```

```
D:\www\MyTreasure\Web\Git\Code\Project>
```

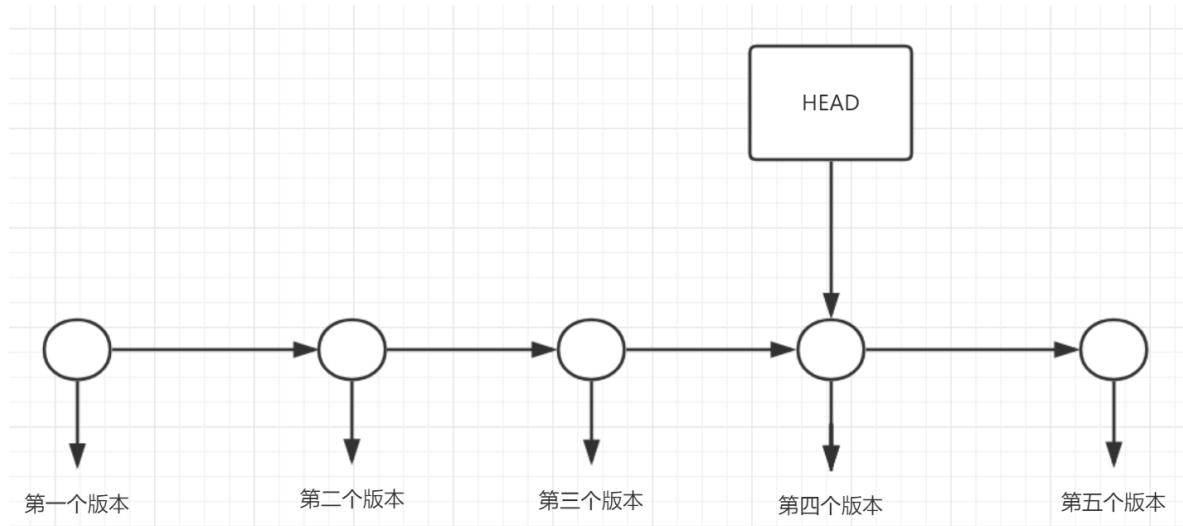
提交为实际版本以后, 再次查看git状态

```
> git status
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git status
On branch master
nothing to commit, working tree clean
```

`nothing to commit, working tree clean` 代表没有任何新的暂存修改需要提交.

## 仓库提交日志查看



```
> git log
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git log
commit 8fa150e261622cbcbbd3c39915f3885228cac26e (HEAD -> master)
Author: <E5><B0><8F><E6><96><B0><E8><80><81><E5><B8><88> <736907613@qq.com>
Date:   Sun Oct 20 14:36:08 2019 +0800

  README.txt<E6><96><87><E4><BB><B6><E5><86><8D><E6><AC><A1><E4><BF><AE><E6><94><B9>

commit 6f3d501d358de582b5549f305a0c5cbaa3c3dda2
Author: <E5><B0><8F><E6><96><B0><E8><80><81><E5><B8><88> <736907613@qq.com>
Date:   Sun Oct 20 13:57:41 2019 +0800

  <E6><96><B0><E5><BB><BA>README<E6><96><87><E4><BB><B6>, <E5><86><99><E5><85><A5><E4><B8><80><E4><BA><E5>
  <9F><BA><E6><9C><AC><E4><BF><A1><E6><81><AF>
```

D:\www\MyTreasure\Web\Git\Code\Project>

## 中文乱码

解决办法有两种

1. 使用 `vscode` 的命令行工具来执行, 不会乱码. 拉起命令行快捷键是 `ctrl+``
2. 在命令行工具中输入指定代码, 临时解决乱码问题

```
git config --global core.quotepath false
git config --global gui.encoding utf-8
git config --global i18n.commit.encoding utf-8
git config --global i18n.logoutputencoding utf-8
set LESSCHARSET=utf-8
```

再次输入, 不会乱码

```
> git log
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git config --global core.quotepath false
D:\www\MyTreasure\Web\Git\Code\Project>git config --global gui.encoding utf-8
D:\www\MyTreasure\Web\Git\Code\Project>git config --global i18n.commit.encoding utf-8
D:\www\MyTreasure\Web\Git\Code\Project>git config --global i18n.logoutputencoding utf-8
D:\www\MyTreasure\Web\Git\Code\Project>set LESSCHARSET=utf-8

D:\www\MyTreasure\Web\Git\Code\Project>git log
commit 8fa150e261622cbcbbd3c39915f3885228cac26e (HEAD -> master)
Author: 小新老师 <736907613@qq.com>
Date:   Sun Oct 20 14:36:08 2019 +0800

    reamde.txt文件再次修改

commit 6f3d501d358de582b5549f305a0c5cbaa3c3dda2
Author: 小新老师 <736907613@qq.com>
Date:   Sun Oct 20 13:57:41 2019 +0800

    新建readme文件，写入一些基本信息

D:\www\MyTreasure\Web\Git\Code\Project>
```

HEAD -> master 代表当前仓库最新的一条记录

通过键盘**下方向键**查看更多的记录. 按**Q**可以直接退出查看操作

## 简略版日志

```
> git log --pretty=oneline
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git log --pretty=oneline
019819f62765cca614e9e0da7edc07ff0a854acd (HEAD -> master) adsfasdf
98f3e586e5985a3ddd69488606c2b0e61cdd58a1 adsfasdf
8cdb3d6686d465587543b9aa1109ba86a031b148 adsfa
8fa150e261622cbcbbd3c39915f3885228cac26e reamde.txt文件再次修改
6f3d501d358de582b5549f305a0c5cbaa3c3dda2 新建readme文件，写入一些基本信息
```

## 回退到指定仓库

`HEAD` 代表当前版本, 上一个版本用 `HEAD^` 代表. 上上个版本就是 `HEAD^^`, 上100个版本就是 `HEAD^^^^.....` 100个`^`, 太麻烦了. 可以使用 `HEAD~100` 来代表

同时也可以使用唯一标识来代表版本, 就是上图中黄色的16进制数字.

^ 符号用 `shift + 6` 组合键输入

```
> git reset --hard HEAD^
```

此命令行在 `power shell`, `vscode` 中都可以完美执行.

**注意, cmd命令行工具有问题, 需要输入**

```
> git reset --hard "HEAD^"
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git reset --hard "HEAD^"
HEAD is now at 8cdb3d6 adsfa
```

也可以使用唯一标识方式, 来回退指定版本

```
> git reset --hard 8cdb3d6686d465587543b9aa1109ba86a031b148
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git reset --hard 8cdb3d6686d465587543b9aa1109ba86a031b148  
HEAD is now at 8cdb3d6 adsfa
```

版本号唯一标识也可以不写全, 稍微写几位, git会自己找到对应的记录

```
> git reset --hard 8cdb3d
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git reset --hard 8cdb3d  
HEAD is now at 8cdb3d6 adsfa
```

此时再次使用日志命令, 会发现记录只剩下恢复到的记录及之前的记录了.

```
> git log --pretty=oneline
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git log --pretty=oneline  
8cdb3d6686d465587543b9aa1109ba86a031b148 (HEAD -> master) adsfa  
8fa150e261622cbcbbd3c39915f3885228cac26e reamde.txt文件再次修改  
6f3d501d358de582b5549f305a0c5cbaa3c3dda2 新建readme文件, 写入一些基本信息
```

此时要想恢复到不见的记录, 则需要新的命令.

显示所有提交和恢复记录, 找到唯一标识就可以恢复指定记录了

```
> git reflog
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git reflog  
8cdb3d6 (HEAD -> master) HEAD@{0}: reset: moving to 8cdb3d  
8cdb3d6 (HEAD -> master) HEAD@{1}: reset: moving to 8cdb3d6686d465587543b9aa1109ba86a031b148  
8cdb3d6 (HEAD -> master) HEAD@{2}: reset: moving to HEAD^  
98f3e58 HEAD@{3}: reset: moving to HEAD^  
019819f HEAD@{4}: reset: moving to HEAD  
019819f HEAD@{5}: commit: adsfadsf  
98f3e58 HEAD@{6}: commit: adsfadsf  
8cdb3d6 (HEAD -> master) HEAD@{7}: commit: adsfa  
8Fa150e HEAD@{8}: commit: reamde.txt文件再次修改  
6f3d501 HEAD@{9}: commit (initial): 新建readme文件, 写入一些基本信息
```

```
D:\www\MyTreasure\Web\Git\Code\Project>
```

## 从暂存区恢复

之前提过, 项目的文件会先通过 `git add` 命令存储在暂存区, 当确认修改到完美状态以后, 再通过 `git commit` 命令提交到仓库中.

我们除了可以从仓库恢复文件记录, 也可以从暂存区恢复记录

```
> git checkout -- 文件名称
```

按照如下方式进行操作:

1. 在 `readme.txt` 中添加一行新的内容
2. 使用 `git add` 命令进行暂存

```
> git add readme.txt
```

3. 再次修改 `readme.txt` 中的内容

4. 执行 `git checkout` 命令, 从暂存文件恢复

```
> git checkout -- readme.txt
```

5. 查看 `readme.txt` 内容, 已经恢复到之前暂存的内容

**注意:** -- 后面有空格, 不要写成 `--reame.txt`, 会报错

```
D:\www\MyTreasure\Web\Git\Code\Project>git add readme.txt
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git checkout -- readme.txt
```

## 删除暂存区的文件

一旦暂存的文件有问题, 需要删除掉暂存记录, 使用命令

```
> git rm --cached 文件名称
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git add .
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   readme.txt
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git rm --cached readme.txt
rm 'readme.txt'
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:   readme.txt

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

## 删除文件

从仓库中删除某个文件

```
>git rm 文件名
```

## 练习

- 修改 `readme.txt` 文件后查看状态, 并添加到暂存区中

- 在工作目录中创建文件夹, 放3张图片, 最终提交到本地仓库
- 新建 `index.html` 文件, 随意填写一些内容, 暂存并提交到仓库中
- 创建 `list.html` 文件, 添加内容; 添加到暂存区; 修改 `list.html` 内容, 从暂存区恢复 `list.html` 内容; 删除暂存区中的 `list.html`. 回退到仓库的上一个版本.

## 分支操作

分支就是科幻电影里面的平行宇宙, 在分支中可以为所欲为, 不会影响主宇宙中任何事情.

分支在实际中有什么用呢? 假设你准备开发一个新功能, 但是需要两周才能完成, 第一周你写了50%的代码, 如果立刻提交, 由于代码还没写完, 不完整的代码库会导致别人不能干活了。如果等代码全部写完再一次提交, 又存在丢失每天进度的巨大风险。

现在有了分支, 就不用怕了。你创建了一个属于你自己的分支, 别人看不到, 还继续在原来的分支上正常工作, 而你在自己的分支上干活, 想提交就提交, 直到开发完毕后, 再一次性合并到原来的分支上, 这样, 既安全, 又不影响别人工作。

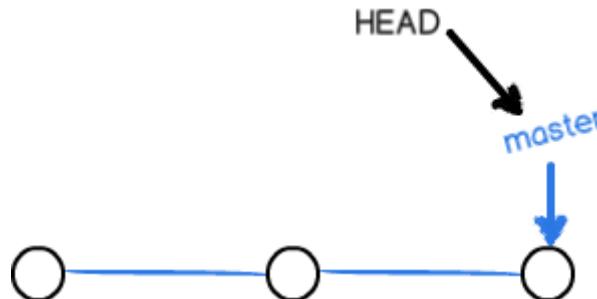
其他版本控制系统如SVN等都有分支管理, 但是用过之后你会发现, 这些版本控制系统创建和切换分支比蜗牛还慢, 简直让人无法忍受, 结果分支功能成了摆设, 大家都不去用。

但Git的分支是与众不同的, 无论创建、切换和删除分支, Git在1秒钟之内就能完成! 无论你的版本库是1个文件还是1万个文件。

## 创建与合并分支

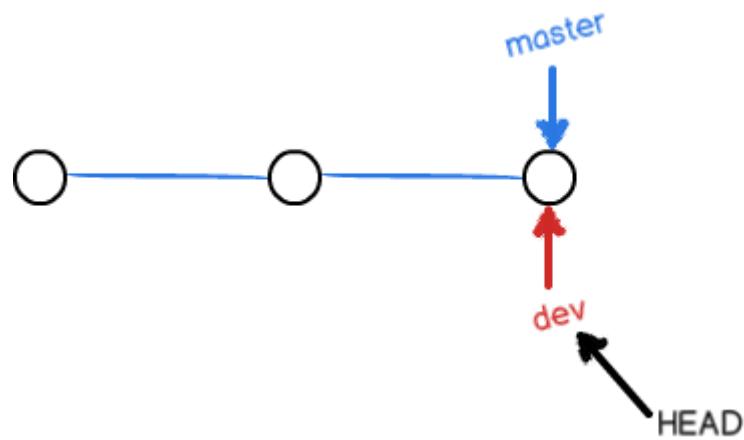
在版本回退里, 你已经知道, 每次提交, Git都把它们串成一条时间线, 这条时间线就是一个分支。截止到目前, 只有一条时间线, 在Git里, 这个分支叫主分支, 即 `master` 分支。`HEAD` 严格来说不是指向提交, 而是指向 `master`, `master` 才是指向提交的, 所以, `HEAD` 指向的就是当前分支。

一开始的时候, `master` 分支是一条线, Git用 `master` 指向最新的提交, 再用 `HEAD` 指向 `master`, 就能确定当前分支, 以及当前分支的提交点:



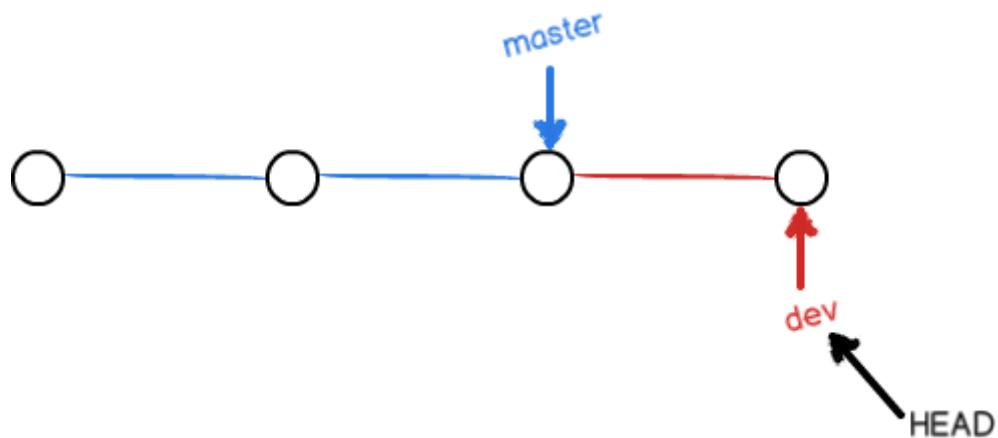
每次提交, `master` 分支都会向前移动一步, 这样, 随着你不断提交, `master` 分支的线也越来越长。

当我们创建新的分支, 例如 `dev` 时, Git新建了一个指针叫 `dev`, 指向 `master` 相同的提交, 再把 `HEAD` 指向 `dev`, 就表示当前分支在 `dev` 上:

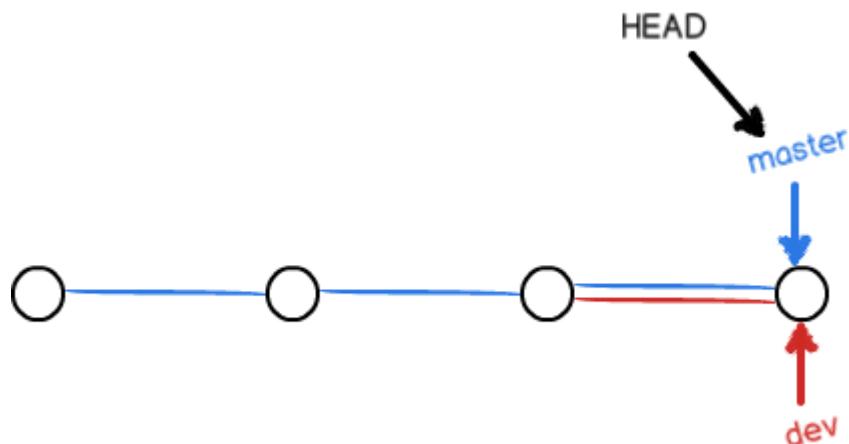


你看，Git创建一个分支很快，因为除了增加一个`dev`指针，改改`HEAD`的指向，工作区的文件都没有任何变化！

不过，从现在开始，对工作区的修改和提交就是针对`dev`分支了，比如新提交一次后，`dev`指针往前移动一步，而`master`指针不变：

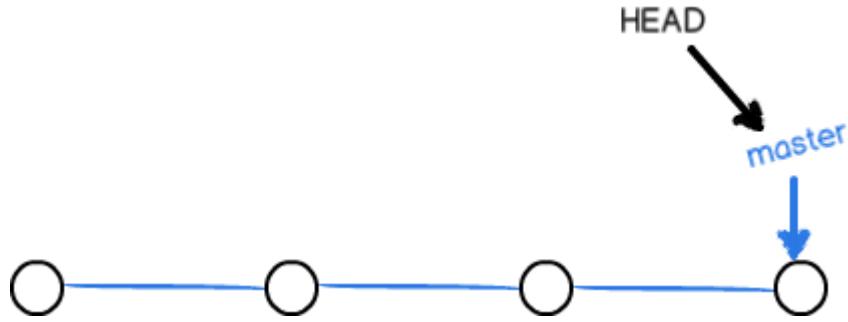


假如我们在`dev`上的工作完成了，就可以把`dev`合并到`master`上。Git怎么合并呢？最简单的方法，就是直接把`master`指向`dev`的当前提交，就完成了合并：



所以Git合并分支也很快！就改改指针，工作区内容也不变！

合并完分支后，甚至可以删除`dev`分支。删除`dev`分支就是把`dev`指针给删掉，删掉后，我们就剩下了  
一条`master`分支：



## 实战

首先, 我们来总览分支有关的命令:

- 查看所有分支

```
> git branch
```

- 创建分支

```
> git branch 分支名
```

- 切换分支

```
> git checkout 分支名
```

```
# 新版本中提供另一种写法  
> git switch 分支名
```

- 创建分支+切换分支 合写方式

```
> git checkout -b 分支名
```

- 合并分支

```
> git merge 分支名
```

- 删除分支

```
# 普通删除  
> git branch -d 分支名  
  
# 强行删除未合并的分支  
> git branch -D 分支名
```

接下来, 我们来实现一个分支操作:

- 查看当前所有分支
- 创建一个新的分支 dev
- 查看当前所有分支
- 切换到 dev 分支上
- 查看当前所有分支
- 在项目中创建新的文件 news.html
- 添加所有文件到暂存, 然后提交到仓库中

- 切换当前分支到 master 主分支, 观察项目中是否存在 news.html 文件
- 再次切换到 dev 分支, 观察是否存在 news.html 文件
- 切换回到 master 分支, 合并 dev 分支到当前所在分支, 观察 news.html 是否出现
- 删除 dev 分支
- 查看当前所有分支

```
# 查看当前所有分支
>git branch

# 创建一个新的分支`dev`
>git branch dev

# 查看当前所有分支
>git branch

# 切换到`dev`分支上
>git checkout dev

# 查看当前所有分支
>git branch

# 在项目中创建新的文件`news.html`
# 添加所有文件到暂存, 然后提交到仓库中
>git add .
>git commit -m "news.html add"

# 切换当前分支到`master`主分支, 观察项目中是否存在`news.html`文件
>git checkout master

# 再次切换到`dev`分支, 观察是否存在`news.html`文件
>git checkout dev

# 切换回到`master`分支, 合并`dev`分支到当前所在分支, 观察`news.html`是否出现
>git checkout master
>git merge dev

# 删除`dev`分支
>git branch -d dev

# 查看当前所有分支
>git branch
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git branch
* master

D:\www\MyTreasure\Web\Git\Code\Project>git branch dev

D:\www\MyTreasure\Web\Git\Code\Project>git branch
  dev
* master

D:\www\MyTreasure\Web\Git\Code\Project>git checkout dev
Switched to branch 'dev'

D:\www\MyTreasure\Web\Git\Code\Project>git branch
* dev
  master

D:\www\MyTreasure\Web\Git\Code\Project>git add .

D:\www\MyTreasure\Web\Git\Code\Project>git commit -m "news.html add"
[dev 350301b] news.html add
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 news.html

D:\www\MyTreasure\Web\Git\Code\Project>git checkout master
Switched to branch 'master'

D:\www\MyTreasure\Web\Git\Code\Project>git checkout dev
Switched to branch 'dev'

D:\www\MyTreasure\Web\Git\Code\Project>git checkout master
Switched to branch 'master'

D:\www\MyTreasure\Web\Git\Code\Project>git merge dev
Updating 716fd77..350301b
Fast-forward
 news.html | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 news.html

D:\www\MyTreasure\Web\Git\Code\Project>git branch -d dev
Deleted branch dev (was 350301b).

D:\www\MyTreasure\Web\Git\Code\Project>git branch
* master

D:\www\MyTreasure\Web\Git\Code\Project>
```

## 分支冲突解决

有这样一种情况, 主分支 master 上修改了文件的第1行代码, 而分支中也修改了第1行代码.

此时当分支内容合并到master时, 就会出现冲突情况

这里有一个冲突的例子:

- 在主分支创建一个 product.txt 文件, 随意填写一些内容, 保存到暂存, 然后提交到仓库
- 创建新的分支 todo1, 切换到 todo1 分支.
- 修改 product.txt 文件, 在末尾行添加一些内容, 然后保存到暂存, 提交到仓库
- 切换到 master 主分支, 同样修改 product.txt 文件, 在末尾行添加一些内容, 暂存并提交到仓库

- 合并 todo1 分支内容到 master, 此时会看到冲突出现, 需要人为来解决冲突.
- 解决完冲突以后, 即可暂存并提交到仓库
- 删除 todo1 分支

```
# 在主分支创建一个`product.txt`文件, 随意填写一些内容, 保存到暂存, 然后提交到仓库
> git switch master
> git add .
> git commit -m "new product"

# 创建新的分支`todo1`, 切换到`todo1`分支.
> git checkout -b todo1

# 修改`product.txt`文件, 在末尾行添加一些内容, 然后保存到暂存, 提交到仓库
> git add .
> git commit -m "new product xxx"

# 切换到`master`主分支, 同样修改`product.txt`文件, 在末尾行添加一些内容, 暂存并提交到仓库
> git switch master

# 修改 product.txt 文件内容
> git add .
> git commit -m "new product 22"

# 合并`todo1`分支内容到`master`, 此时会看到冲突出现, 需要人为来解决冲突.
> git merge todo1

# 解决完冲突以后, 即可暂存并提交到仓库
> git add .
> git commit -m "冲突解决"

# 删除`todo1`分支
> git branch -d todo1
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git switch master
Already on 'master'

D:\www\MyTreasure\Web\Git\Code\Project>git add .

D:\www\MyTreasure\Web\Git\Code\Project>git commit -m "new product"
[master b8f7288] new product
 1 file changed, 1 insertion(+)
 create mode 100644 product.txt

D:\www\MyTreasure\Web\Git\Code\Project>git checkout -b todo1
Switched to a new branch 'todo1'

D:\www\MyTreasure\Web\Git\Code\Project>git add .

D:\www\MyTreasure\Web\Git\Code\Project>git commit -m "new product xxx"
[todo1 70f8b07] new product xxx
 1 file changed, 2 insertions(+)

D:\www\MyTreasure\Web\Git\Code\Project>git switch master
Switched to branch 'master'

D:\www\MyTreasure\Web\Git\Code\Project>git add .

D:\www\MyTreasure\Web\Git\Code\Project>git commit -m "冲突解决"
[master a453afe] 冲突解决

D:\www\MyTreasure\Web\Git\Code\Project>git branch -d todo1
Deleted branch todo1 (was 70f8b07).

D:\www\MyTreasure\Web\Git\Code\Project>
```

冲突产生时的样式, vscode 比较智能, 会提供一些按钮快速解决冲突

```
product.txt x
1 一些产品信息
2 采用当前更改 | 采用传入的更改 | 保留双方更改 | 比较变更
3 <<<<< HEAD (当前更改)
4 00000000d
5 =====
6 2222222
7 3333333
8 >>>>> todo1 (传入的更改)
```

通过命令可以查看分支合并图

```
>git log --graph
```

```
D:\www\MyTreasure\Web\Git\Code\Project>git log --graph
* commit a453afe69abae81be6fff675c929c39260403f81 (HEAD -> master)
| Merge: 034efe1 70f8b07
| Author: xiaoxin <736907613@qq.com>
| Date:   Sun Oct 20 17:29:06 2019 +0800
|
|       冲突解决
|
* commit 70f8b071aaeed6cc3b84cae54849a14adb0e99d1
| Author: xiaoxin <736907613@qq.com>
| Date:   Sun Oct 20 17:26:42 2019 +0800
|
|       new product xxx
|
* commit 034efe14262a5fa301ee24e6eed3facdd1572134
| Author: xiaoxin <736907613@qq.com>
| Date:   Sun Oct 20 17:27:03 2019 +0800
|
|       new product 22
|
* commit b8f728806817b559d0681fb6b19ae3e7fa8cee88
| Author: xiaoxin <736907613@qq.com>
| Date:   Sun Oct 20 17:26:05 2019 +0800
|
|       new product
|
* commit d81a1df06b212d90dd5637e44b2c2134ec1dcf65
| Author: xiaoxin <736907613@qq.com>
| Date:   Sun Oct 20 17:25:29 2019 +0800
|
|       xxx
|
* commit 1eee8bfc0977ec2c5a99b048bd443144366da416
| Merge: 8602fe3 54373b1
| Author: xiaoxin <736907613@qq.com>
| Date:   Sun Oct 20 17:24:02 2019 +0800
|
|       冲突解决
|
* commit 54373b1932529fd0cc0edb879ae7582206501b23
| Author: xiaoxin <736907613@qq.com>
| Date:   Sun Oct 20 17:22:57 2019 +0800
|
|       new product xxx
|
* commit 8602fe333eae624417f1d2688130a3df02bfdee0
| Author: xiaoxin <736907613@qq.com>
| Date:   Sun Oct 20 17:23:17 2019 +0800
|
|       new product 22
```

## 远程仓库使用

本地仓库只能在自己电脑上使用, 如果希望能够跨电脑使用版本控制工具, 则需要把仓库存放在互联网上.

### 有名的互联网Git仓库

- **码云:** 国内, 访问速度快, 个人免费, 空间大. 纯中文界面.
- **GitHub:** 世界有名, 国外的, 访问速度慢, 个人免费, 空间大, 全英文界面.

### 码云使用

官方网站<https://gitee.com/>

注册一个账号密码, 牢牢记住. 提交代码时需要使用

---

[点击这里, 新建仓库](#)

The screenshot shows the Gitee homepage. At the top right, there is a red box highlighting the '+ 新建仓库' (Create Repository) button. Below the header, there's a sidebar with user statistics (仓库 8, Pull Requests 0, 任务 0, 代码片段 0), a '码云金库' section, and a '企业' section. The main area shows a timeline of recent activities. On the far right, there are several repository cards with '关注' (Follow) buttons.

## 创建一个仓库

The screenshot shows the 'Create Repository' page. The '仓库名称' (Repository Name) field contains 'Demo'. A red arrow points from the '只能写 英文 字母' (Can only write English letters) validation message to the input field.

## 新建仓库

仓库名称 ✓ 只能写 英文 字母

Demo

归属



路径

自动出现在此处

/ Demo

仓库地址: <https://gitee.com/736907613/Demo>

这就是你的仓库远程地址了

仓库介绍 非必填

用简短的语言来描述一下吧

随意填写, 可以不填

是否开源

私有  公开

按照你的需求决定是否公开

私有仓库的非仓库成员无法访问该仓库的代码和其他任何形式的资源

私有仓库最多支持 5 人协作 (如拥有多个私有仓库, 所有协作人数总计不得超过 5 人)

企业仓库, 更适合使用码云企业版, [了解更多 >>](#)

选择语言

添加 .gitignore

请选择语言

请选择 .gitignore 模板

使用 README 文件初始化这个仓库

这里都可以不管

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

点击创建

## 找到你的仓库

The screenshot shows the Gitee homepage. On the left, there's a sidebar with sections for '仓库' (Repositories), '企业' (Enterprise), '组织' (Organizations), and '仓库' (Repositories). The main area displays a list of repositories under '我的码云' (My Gitee). A red box highlights the '仓库' section and the list of repositories. The repositories listed are: 小石头/Demo, 小石头/MyTreasure, 小石头/myfresh, 小石头/webdemo, 小石头/phpcore, and 小石头/MyTreasure. To the right, there's a '所有动态' (All Activity) feed showing various pushes and creations from other users like Wizzer, Deament, seagull, and John. There's also a '推荐仓库' (Recommended Repositories) section.

## 找到你仓库的地址

仓库地址分 https 和 ssh 两种方式

- https 方式: 每次克隆, 推送, 拉取都需要输入账号密码
- ssh 方式: 克隆, 推送, 拉取不需要输入账号密码, 但是需要提前设置秘钥

The screenshot shows a repository page for '小石头 / Demo'. At the top, there are tabs for '代码' (Code), 'Issues' (0), 'Pull Requests' (0), 'Attachments' (0), 'Wiki' (0), 'Statistics', 'DevOps', 'Services', and 'Management'. Below the tabs, there's a summary bar with '1 次提交', '1 个分支', '0 个标签', '0 个发行版', and '1 位贡献者'. A red arrow points to the '克隆/下载' (Clone/Download) button. Another red arrow points to the '复制' (Copy) link next to the HTTPS URL 'https://gitee.com/736907613/Demo.git'. A green banner at the bottom right says '完备安全策略' (Comprehensive Security Strategy).

## 帮助文档

<https://gitee.com/help>

The screenshot shows a tutorial page for cloning a Git repository. The main content is titled "如何通过 git clone 克隆仓库/项目". It includes sections for cloning via HTTPS and SSH. A red arrow points from the "通过HTTPS协议克隆" section to a command-line terminal window showing the execution of "git clone https://gitee.com/zxzllyj/sample-project.git". The terminal output shows the cloning process, including object counting, compression, and receiving objects. The top right corner of the browser window has a dropdown menu with options like "个人主页", "设置", "代码片段", "帮助", and "退出". The "通过HTTPS" option is highlighted with a red box.

## 使用方式

首先在你的硬盘上找到你想要存放代码的文件夹, 通过命令行工具打开, 克隆远程仓库到本地.

**注意, 必须是空文件夹. 我们可以之后把代码拷贝进来即可**

这里使用 https 地址, 方便快捷. 使用 ssh 地址需要额外的秘钥配置, 繁琐.

这里的地址是之前步骤中复制得到的

```
git clone https://gitee.com/736907613/Demo.git
```

此处需要录入码云的账号和密码, 就是第一步注册时申请的

**注意密码录入时, 是不显示的, 这是为了安全考虑防止其他人知道你密码的位数**

输入正确的密码, 回车确认即可

```
Microsoft Windows [版本 10.0.18362.418]
(c) 2019 Microsoft Corporation. 保留所有权利。
填写你自己的项目远程地址, 要https开头
D:\www\MyTreasure\Web\Git\Code\Project1>git clone https://gitee.com/736907613/Demo.git
Cloning into 'Demo'...
Username for 'https://gitee.com': 18800108022 输入码云的账号
Password for 'https://18800108022@gitee.com': 此处输入密码, 但是并不会显示出来密码位数, 更加安全
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.

D:\www\MyTreasure\Web\Git\Code\Project1>
```

文件夹中会多出一个Demo文件夹, 这就是从网上下载的文件夹

✓ 今天 (1) —



Demo

文件夹中默认会带有一些内容

✓ 今天 (3) —



.git

修改日期: 201



README.en.md

修改日期: 201

类型: Typora

大小: 984 字节



README.md

修改日期: 201

类型: Typora

大小: 1.32 KB

复制我们的项目代码到这个文件夹中, 以后就在这个文件夹中编写项目即可.

远程仓库使用相关的命令行

- 克隆远程仓库到本地

```
git clone 仓库地址
```

- 推送本地提交的版本到远程仓库

```
# 默认推送到 origin 仓库的master分支  
git push
```

```
# 可以主动声明  
git push origin master
```

- 从远程仓库拉取代码到本地

```
git pull
```

- 添加远程仓库地址

```
git remote add origin 你的新远程仓库地址
```

- 查看当前所有远程仓库别名

```
git remote
```

- 查看某远程仓库地址

```
git remote get-url 仓库别名
```

```
D:\www\MyTreasure>git remote get-url origin  
git@gitee.com:736907613/MyTreasure.git
```

- 删除当前远程仓库地址

```
git remote rm 仓库别名
```

```
D:\www\MyTreasure>git remote rm origin
```

- 新增远程仓库地址并起别名, 这里 origin 是习惯上的别名

```
git remote add origin 你的新远程仓库地址
```

- 更改仓库地址

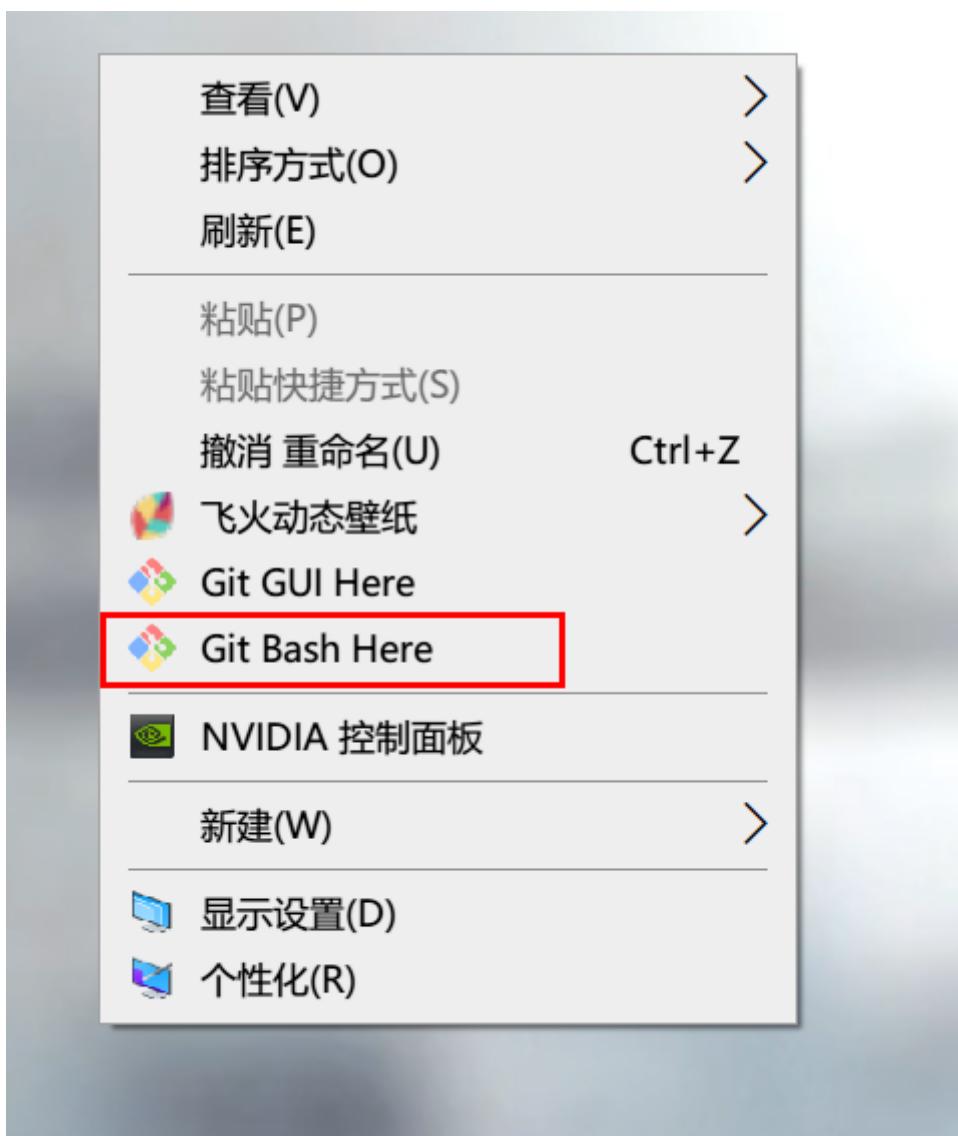
```
git remote set-url origin 新的仓库地址
```

```
D:\www\MyTreasure>git remote set-url origin git@gitee.com:736907613/MyTreasure.git
```

## SSH方式

此方式的好处是不需要每次操作都输入账号密码, **但是必须设定秘钥**

在电脑任意位置右键, 打开 `Git Bash Here`

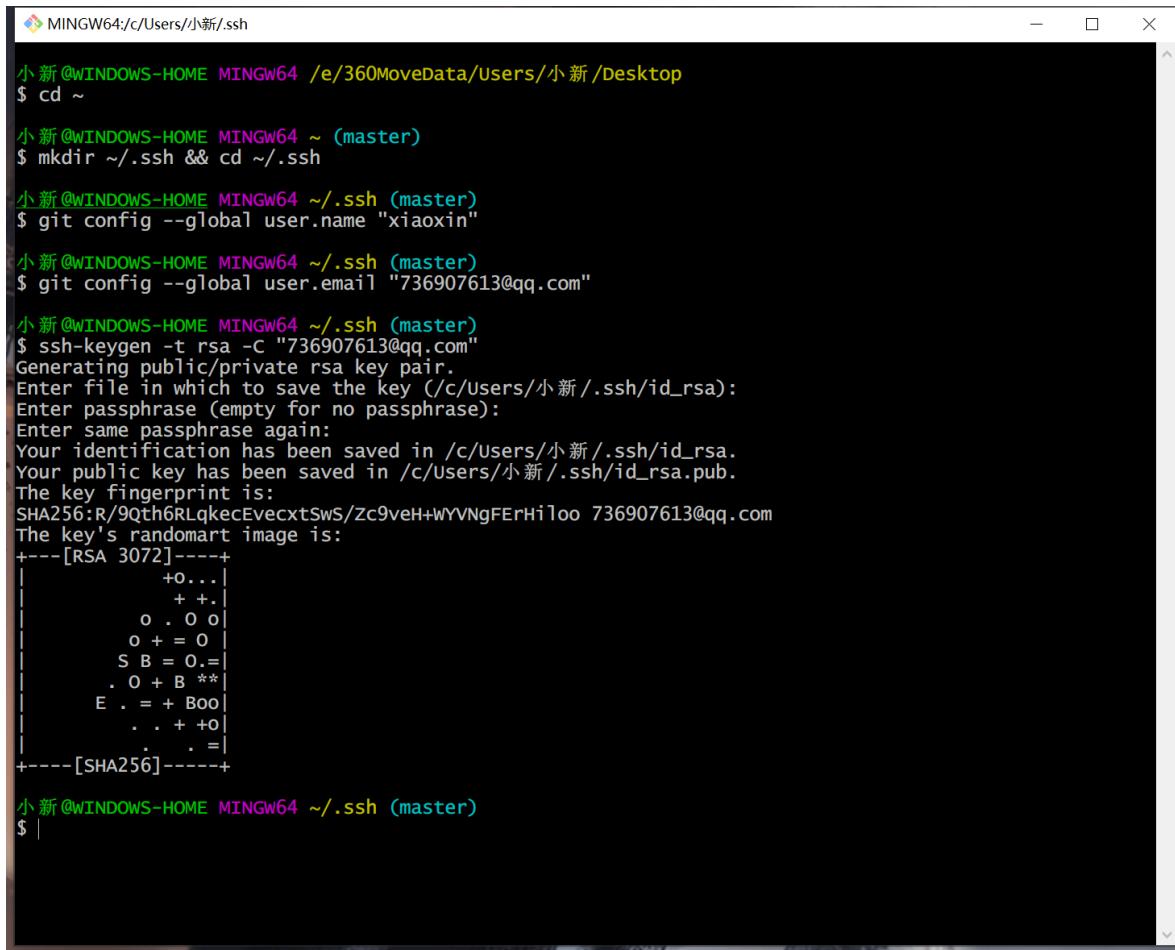


稍作等待, 在弹出的命令提示框中依次输入以下命令

```
$ cd ~  
  
$ mkdir ~/.ssh && cd ~/.ssh  
  
# 下方两行代码之前应该执行过, 则本次不需要执行  
$ git config --global user.name "你的名字"  
$ git config --global user.email "你的邮箱"  
  
$ ssh-keygen -t rsa -C "你的email"
```

实例:

最后一个命令行结束后, 会有几个输入请求, 都按回车即可.



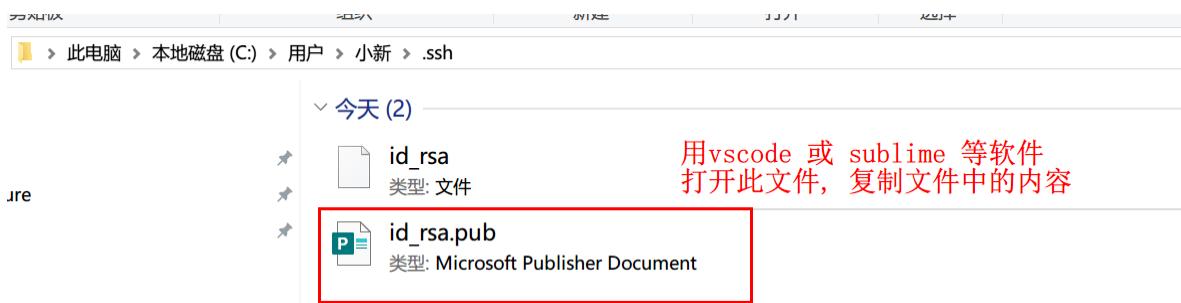
The screenshot shows a terminal window titled 'MINGW64:/c/Users/小新/.ssh'. The command history is as follows:

```
小新@WINDOWS-HOME MINGW64 /e/360MoveData/Users/小新/Desktop
$ cd ~
小新@WINDOWS-HOME MINGW64 ~ (master)
$ mkdir ~/.ssh && cd ~/.ssh
小新@WINDOWS-HOME MINGW64 ~/.ssh (master)
$ git config --global user.name "xiaoxin"
小新@WINDOWS-HOME MINGW64 ~/.ssh (master)
$ git config --global user.email "736907613@qq.com"
小新@WINDOWS-HOME MINGW64 ~/.ssh (master)
$ ssh-keygen -t rsa -C "736907613@qq.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/小新/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/小新/.ssh/id_rsa.
Your public key has been saved in /c/Users/小新/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:R/9Qth6RLqkecEvecxtSwS/Zc9veH+WYVNgFErHiloo 736907613@qq.com
The key's randomart image is:
+---[RSA 3072]----+
| +o... |
| + . . |
| o . O o |
| o + = O |
| S B = O.= |
| . O + B. **|
| E . = + Boo |
| . . + +o |
| . . = |
+---[SHA256]----+
小新@WINDOWS-HOME MINGW64 ~/.ssh (master)
$ |
```

接下来, 到你的C盘下, 用户文件夹, 找到你的用户名, 找到 .ssh 文件夹.

## 管理员: 命令提示符

```
Microsoft Windows [版本 10.0.18363]
(c) 2019 Microsoft Corporation。保
↓
C:\Users\小新>_
```



复制 id\_rsa.pub 中的内容。

填写到 <https://gitee.com/profile/sshkeys> 此位置中. 需要登录操作

The screenshot shows the 'SSH公钥' (SSH Public Key) section of a Gitee profile. On the left, there's a sidebar with user info ('小石头' joined 6 years ago), messages, notifications, and various settings like basic profile and security. The main area has a title 'SSH公钥' with a note: '使用SSH公钥可以让你在你的电脑和码云通讯的时候使用安全连接 (Git的Remote要使用SSH地址)'. It lists one existing key: '您当前的SSH公钥数: 1' with the value 'suibian' (SHA256:tcAKWnGBq1aPVQtLw9rJcqoVAN0ZwzqrnriyKHeatno'). A '删除' (Delete) button is next to it. Below this is a '添加公钥' (Add Key) form. The '标题' (Title) field contains '此处随便写, 写 123456 都可以' (Write anything here, it can be 123456). The '公钥' (Key) field contains placeholder text: '把你的公钥粘贴到这里, 查看 [怎样生成公钥](#)' (Paste your key here, see [How to generate a key](#)). A note below says '支持以'ssh-rsa', 'ssh-dss', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384' or 'ecdsa-sha2-nistp521' 开头' (Supported formats: ssh-rsa, ssh-dss, ssh-ed25519, ecdsa-sha2-nistp256, ecdsa-sha2-nistp384 or ecdsa-sha2-nistp521). A red box highlights the instruction '复制 id\_rsa.pub 的内容到这里' (Copy the content of id\_rsa.pub here). At the bottom right is a '确定' (Confirm) button.

确定完成后, 复制 SSH 地址

The screenshot shows a Gitee repository page for '小石头 / Demo'. It displays basic statistics: 4 commits, 1 branch, 0 tags, 0 releases, and 1 contributor. Below the stats, there's a list of files: README.en.md, README.md, index.txt, news.html, product.txt, and readme.txt. On the right, there are clone options for 'HTTPS' and 'SSH', with 'SSH' being highlighted with a red box. A large watermark for '限制 Git 强推 远离强制推送风险' is overlaid on the page.

然后在命令行中,更改你的远程仓库地址为 `SSH` 方式

```
git remote set-url origin 你的仓库地址
```

```
D:\www\MyTreasure>git remote set-url origin git@gitee.com:736907613/MyTreasure.git
```

之后再执行各种远程操作命令,就不会要求输入账号和密码了

### .gitignore

有时候,并不想把所有文件都上传到远程服务器,那么可以在项目根目录下创建一个 `.gitignore` 文件,在文件中按照固定格式,指定不想上传到远程服务器的文件

一行写一个规则

The screenshot shows a file browser listing several files and a `.gitignore` file. The `.gitignore` file is highlighted with a red box. The files listed are: .git, iOS, PHP, Web, .gitignore, readme.md, and xxx.txt. The `.gitignore` file is described as a 'GITIGNORE 文件'.

文件的内容格式大概为:

\*是通配符

```
*.log  
*.temp  
/vendor
```

## GitHub使用

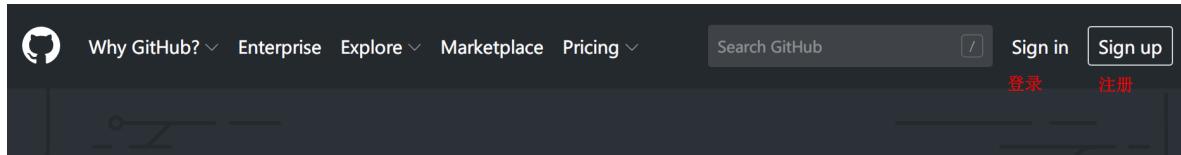
官方网站<https://github.com/>

这是目前全球最大的代码分享网站.

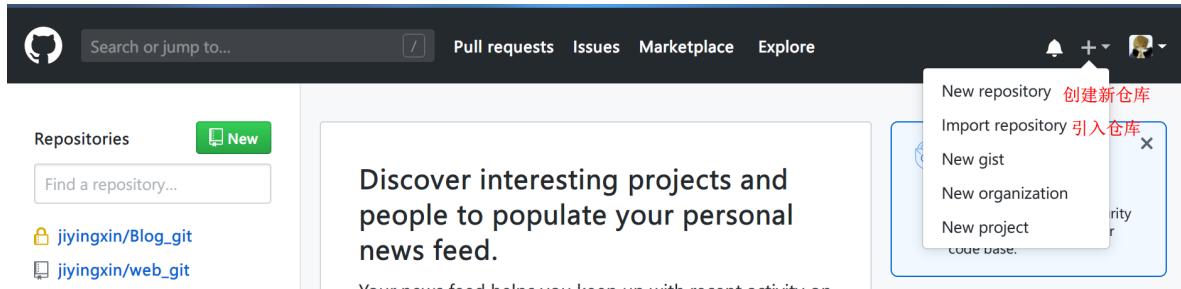
GitHub的操作方式与码云几乎一致, 不过GitHub提供一个可视化工具, 让Git使用起来更加方便.

可视化工具下载地址: <https://desktop.github.com/>

## 注册你的账号



登录之后



## 创建新的仓库

### Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?  
[Import a repository](#).

Owner      Repository name \*

jiyingxin /

Great repository names are short and memorable. Need inspiration? How about `refactored-disco`?

Description (optional)

Public 公开的  
Anyone can see this repository. You choose who can commit.

Private 私有的, 但是私有的要收费  
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

Initialize this repository with a README  
This will let you immediately clone the repository to your computer.

Add .gitignore:  Add a license:

查看你的所有仓库

Signed in as jiyingxin

Repositories 15

Find a repository... Type: All ▾

web\_git

Updated 4 days ago

Blog\_git Private

一个测试项目

PHP Updated 24 days ago

DaDaMusic

达内音乐

达内教育

Your repositories

Feature preview Help Settings Sign out

## 找到下载地址

jiyingxin / web\_git

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Security Insights Settings

与码云一样，分两种下载方式 HTTPS 和 SSH

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/jiyingxin/web\_git.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line [这里有些使用简介](#)

```
echo "# web_git" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/jiyingxin/web_git.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/jiyingxin/web_git.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

## SSH方式的秘钥设置

The screenshot shows the GitHub user profile page for 'jiyingxin'. The left sidebar lists various settings: Personal settings, Profile, Account, Security, Emails, Notifications, Billing, **SSH and GPG keys**, Blocked users, Repositories, Organizations, Saved replies, Applications, and Developer settings. A red box highlights 'SSH and GPG keys', and a red arrow points from it to the 'Key' input field on the main page. Another red arrow points from the 'Settings' link in the top right to the 'Settings' link in the bottom right of the sidebar. The main content area displays the 'SSH keys / Add new' page, which includes a title input field, a key input field containing placeholder text, and a green 'Add SSH key' button. A red box highlights the 'Add SSH key' button.

Signed in as jiyingxin

Profile Set status

Account Your profile

Security Your repositories

Emails Your projects

Notifications Your stars

Billing Your gists

**SSH and GPG keys**

Blocked users

Repositories

Organizations

Saved replies

Applications

Developer settings

Add SSH key

同于之前码云的方式，使用SSH需要设定秘钥

Title

Key

Begins with 'ssh-rsa', 'ssh-dss', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384'

Add SSH key

Settings

Sign out

## 可视化客户端的使用

