什么是Github？

要先知道什么git。

# GIT

## （分布式版本控制系统）

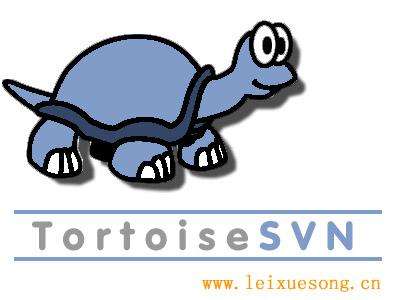
[编辑](https://baike.baidu.com/item/GIT/javascript:;) [讨论](https://baike.baidu.com/planet/talk?lemmaId=12647237" \t "https://baike.baidu.com/item/GIT/_blank)

Git(读音为/gɪt/。)是一个开源的分布式版本控制系统，可以有效、高速地处理从很小到非常大的项目版本管理。 [1]  Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。

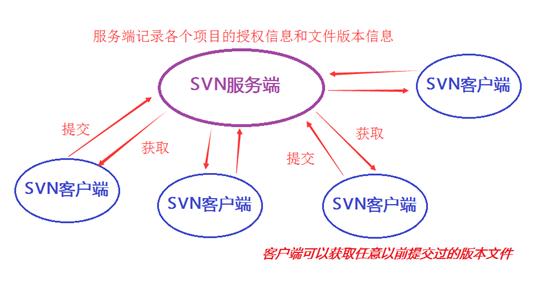
Linux操作系统之父Linus的又一伟大工具。

在学习Git之前，还要知道什么是软件管理。以及之前的软件版本管理工具SVN。

SVN是Subversion的简称，是一个开放源代码的版本控制系统，相较于RCS、CVS，它采用了分支管理系统，它的设计目标就是取代CVS。互联网上很多版本控制服务已从CVS迁移到Subversion。说得简单一点SVN就是用于多个人共同开发同一个项目，共用资源的目的。

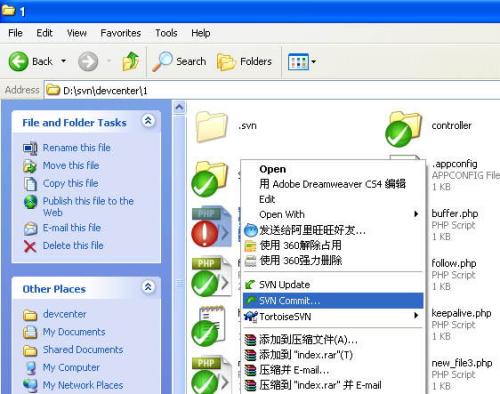
IMG_256IMG_256

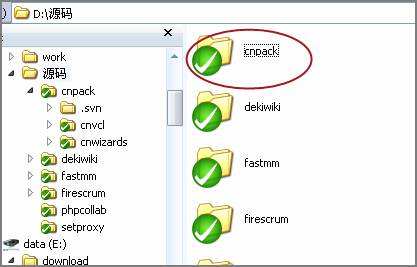
SVN的架构



SVN的操作界面







回到Git。

GIT特点和优势：

<https://www.cnblogs.com/Sungeek/p/9152223.html>

# **[SVN与Git比较的优缺点差异](https://www.cnblogs.com/Sungeek/p/9152223.html)**

<https://www.cnblogs.com/mtl-key/p/6902627.html>

# [Svn与Git的区别](https://www.cnblogs.com/mtl-key/p/6902627.html)

**1．SVN优缺点**

优点：

1、 管理方便，逻辑明确，符合一般人思维习惯。

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、 适合开发人数不多的项目开发。

缺点：

1、 服务器压力太大，数据库容量暴增。

2、 如果不能连接到服务器上，基本上不可以工作，看上面第二步，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（开发人数非常非常多，但是Google app engine就是用svn的）。但是一般集中式管理的有非常明确的权限管理机制（例如分支访问限制），可以实现分层管理，从而很好的解决开发人数众多的问题。

**2．Git优缺点**  
优点：

1、适合分布式开发，强调个体。

2、公共服务器压力和数据量都不会太大。

3、速度快、灵活。

4、任意两个开发者之间可以很容易的解决冲突。

5、离线工作。

缺点：

1、学习周期相对而言比较长。

2、不符合常规思维。

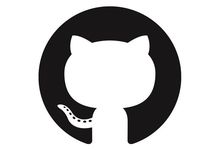
3、代码保密性差，一旦开发者把整个库克隆下来就可以完全公开所有代码和版本信息。

<https://www.cnblogs.com/wupeiqi/p/7295372.html>

# **[老铁，这年头不会点Git真不行！！！](https://www.cnblogs.com/wupeiqi/p/7295372.html)**

有趣的帖子

Git和GitHub



gitHub是一个面向[开源](https://baike.baidu.com/item/%E5%BC%80%E6%BA%90/20720669" \t "https://baike.baidu.com/item/github/_blank)及私有[软件](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6/12053" \t "https://baike.baidu.com/item/github/_blank)项目的托管平台，因为只支持git 作为唯一的版本库格式进行托管，故名gitHub。

gitHub于2008年4月10日正式上线，除了git代码仓库托管及基本的 Web管理界面以外，还提供了订阅、讨论组、文本渲染、在线文件编辑器、协作图谱（报表）、代码片段分享（Gist）等功能。目前，其注册用户已经超过350万，托管版本数量也是非常之多，其中不乏知名开源项目 [Ruby](https://baike.baidu.com/item/Ruby/11419" \t "https://baike.baidu.com/item/github/_blank) on Rails、[jQuery](https://baike.baidu.com/item/jQuery/5385065" \t "https://baike.baidu.com/item/github/_blank)、[python](https://baike.baidu.com/item/python/407313" \t "https://baike.baidu.com/item/github/_blank) 等。

2018年6月4日，微软宣布，通过75亿美元的股票交易收购代码托管平台GitHub。



<https://github.com/>

GitHub官网

学习Git和GitHub

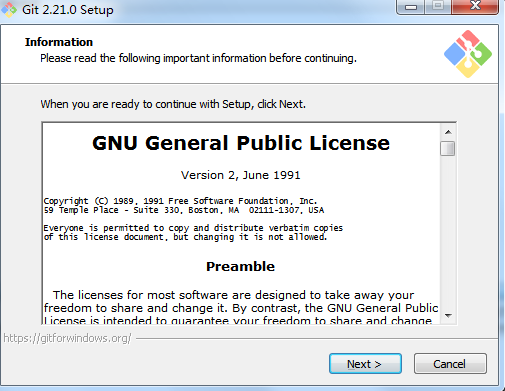
<https://git-scm.com/book/zh/v2>

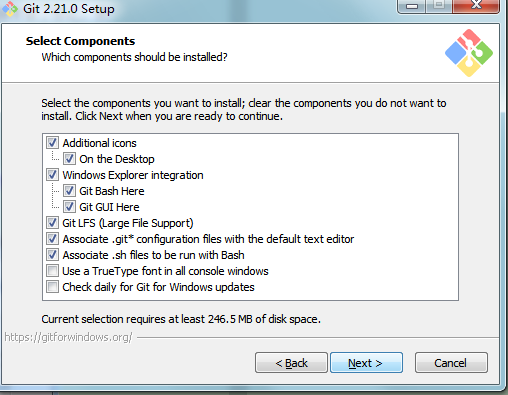
Git使用手册

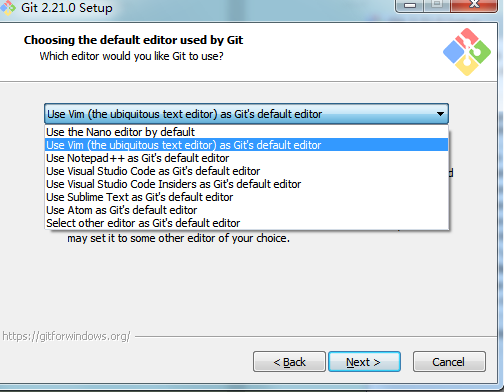
Git 有三种状态，你的文件可能处于其中之一：已提交（committed）、已修改（modified）和已暂存（staged）。 已提交表示数据已经安全的保存在本地数据库中。 已修改表示修改了文件，但还没保存到数据库中。 已暂存表示对一个已修改文件的当前版本做了标记，使之包含在下次提交的快照中。

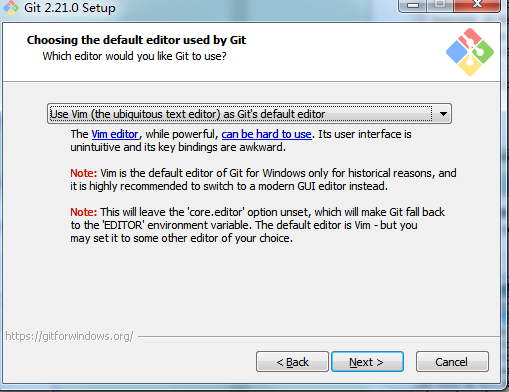
Git的安装

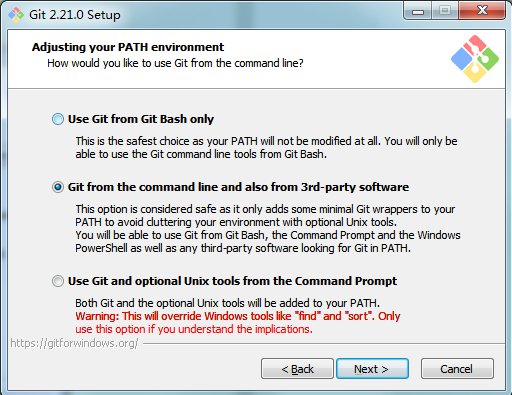
<https://git-scm.com/book/zh/v2/%E8%B5%B7%E6%AD%A5-%E5%AE%89%E8%A3%85-Git>

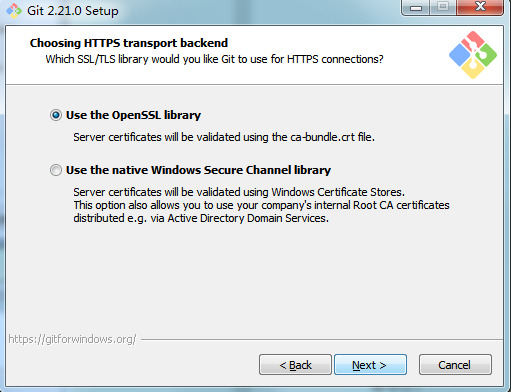


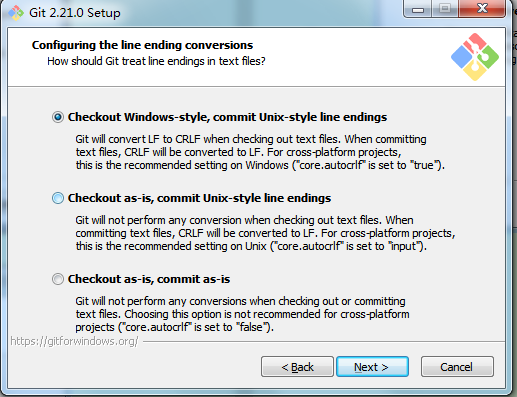


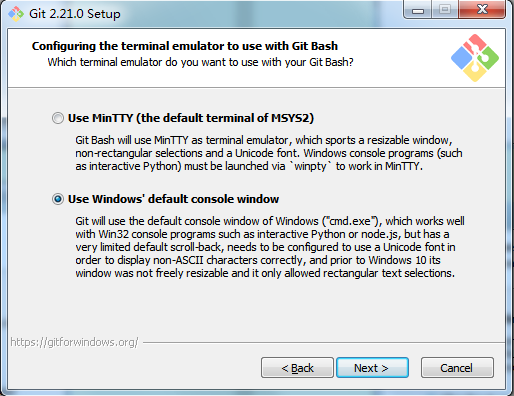


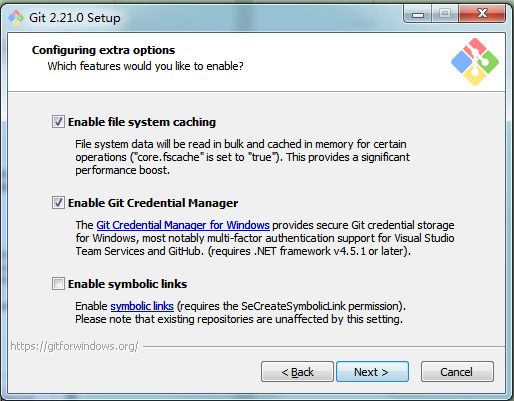




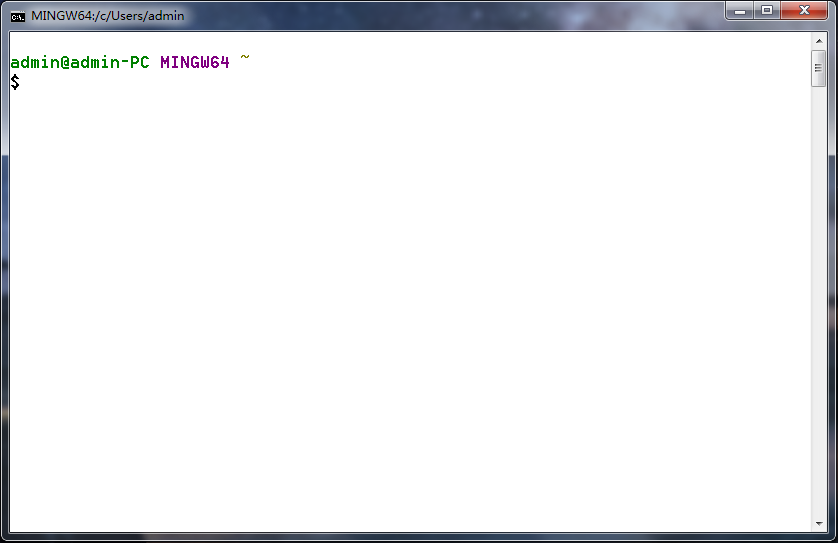


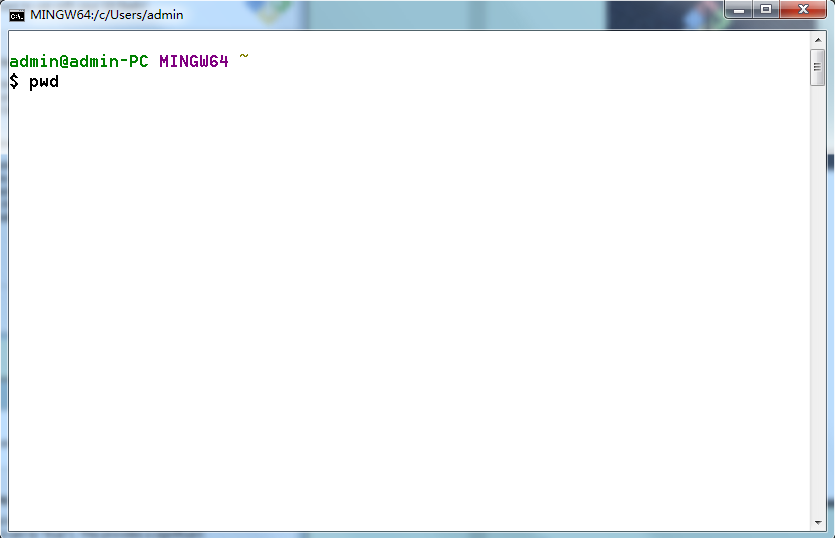


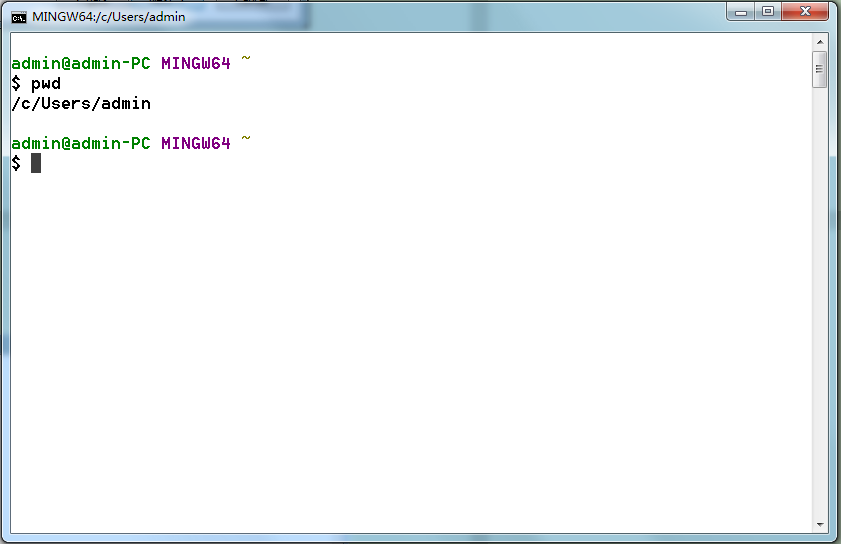












出现以上内容说明已经正确安装了git

初次运行 Git 前的配置

既然已经在系统上安装了 Git，你会想要做几件事来定制你的 Git 环境。 每台计算机上只需要配置一次，程序升级时会保留配置信息。 你可以在任何时候再次通过运行命令来修改它们。

Git 自带一个 git config 的工具来帮助设置控制 Git 外观和行为的配置变量。 这些变量存储在三个不同的位置：

/etc/gitconfig 文件: 包含系统上每一个用户及他们仓库的通用配置。 如果使用带有 --system 选项的 git config 时，它会从此文件读写配置变量。

~/.gitconfig 或 ~/.config/git/config 文件：只针对当前用户。 可以传递 --global 选项让 Git 读写此文件。

当前使用仓库的 Git 目录中的 config 文件（就是 .git/config）：针对该仓库。

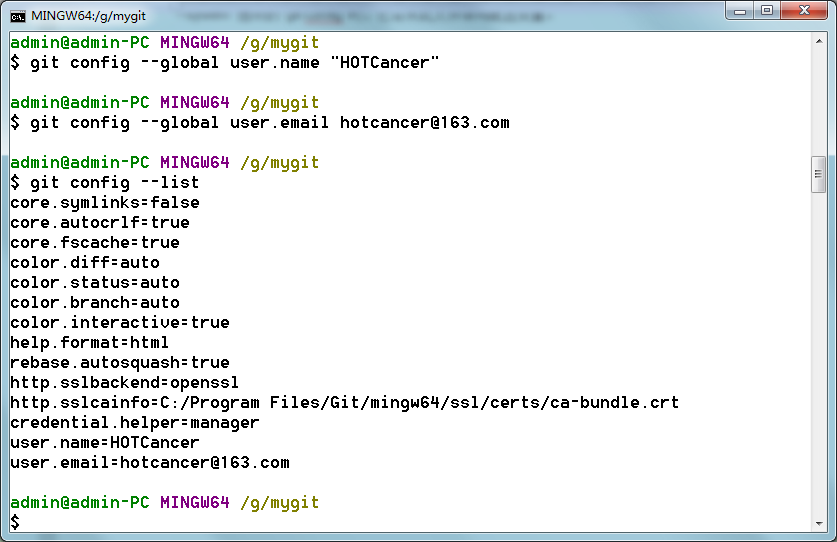
在 Windows 系统中，Git 会查找 $HOME 目录下（一般情况下是 C:\Users\$USER）的 .gitconfig 文件。 Git 同样也会寻找 /etc/gitconfig 文件，但只限于 MSys 的根目录下，即安装 Git 时所选的目标位置。

Git的使用

命令行：

检查配置信息

如果想要检查你的配置，可以使用 git config --list 命令来列出所有 Git 当时能找到的配置。



你自己的邮箱

填写你自己的用户名

# 2.1 Git 基础 - 获取 Git 仓库

关键字：

仓库（repository）

跟踪（track）

暂存（stage）

提交（commit)

推送（push）

拉取（pull）

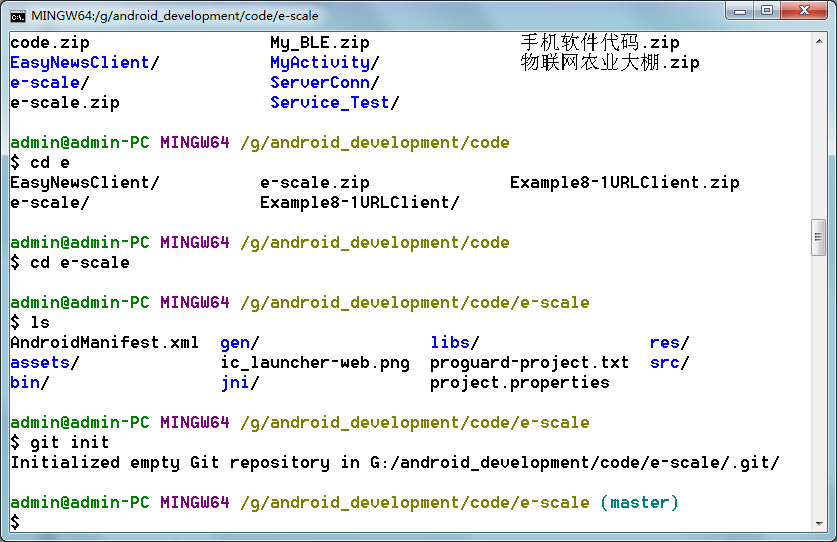
## **获取 Git 仓库**

有两种取得 Git 项目仓库的方法。 第一种是在现有项目或目录下导入所有文件到 Git 中； 第二种是从一个服务器克隆一个现有的 Git 仓库。

### **在现有目录中初始化仓库**

如果你打算使用 Git 来对现有的项目进行管理，你只需要进入该项目目录并输入：

$ git init



如果你是在一个已经存在文件的文件夹（而不是空文件夹）中初始化 Git 仓库来进行版本控制的话，你应该开始跟踪这些文件并提交。 你可通过 git add 命令来实现对指定文件的跟踪，然后执行 git commit 提交：

$ git add \*.c

$ git add LICENSE

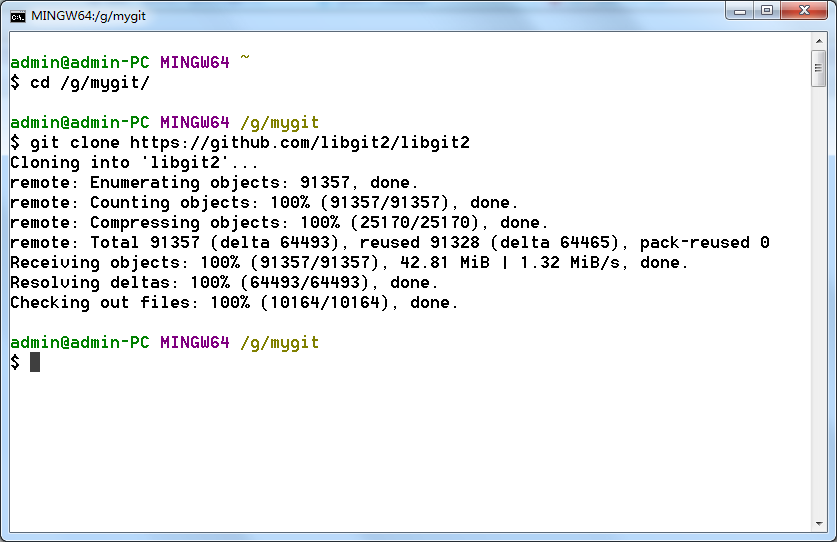
$ git commit -m 'initial project version'

### **克隆现有的仓库**

如果你想获得一份已经存在了的 Git 仓库的拷贝，比如说，你想为某个开源项目贡献自己的一份力，这时就要用到 git clone 命令。 如果你对其它的 VCS 系统（比如说Subversion）很熟悉，请留心一下你所使用的命令是"clone"而不是"checkout"。 这是 Git 区别于其它版本控制系统的一个重要特性，Git 克隆的是该 Git 仓库服务器上的几乎所有数据，而不是仅仅复制完成你的工作所需要文件。 当你执行 git clone 命令的时候，默认配置下远程 Git 仓库中的每一个文件的每一个版本都将被拉取下来。

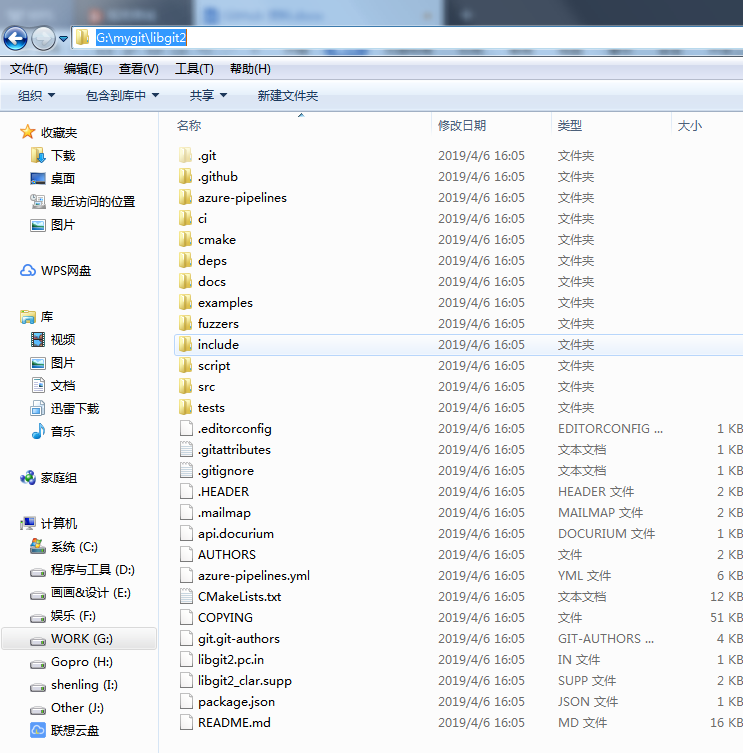
克隆仓库的命令格式是 git clone [url] 。 比如，要克隆 Git 的可链接库 libgit2，可以用下面的命令：

$ git clone https://github.com/libgit2/libgit2



在电脑端查看。

libgit2的源码被克隆了下来！



这会在当前目录下创建一个名为 “libgit2” 的目录，并在这个目录下初始化一个 .git 文件夹，从远程仓库拉取下所有数据放入 .git 文件夹，然后从中读取最新版本的文件的拷贝。 如果你进入到这个新建的 libgit2 文件夹，你会发现所有的项目文件已经在里面了，准备就绪等待后续的开发和使用。

如果你想在克隆远程仓库的时候，自定义本地仓库的名字，你可以使用如下命令：

$ git clone https://github.com/libgit2/libgit2 mylibgit

这将执行与上一个命令相同的操作，不过在本地创建的仓库名字变为 mylibgit。

Git 支持多种数据传输协议。 上面的例子使用的是 https:// 协议，

# 2.2 Git 基础 - 记录每次更新到仓库

## **记录每次更新到仓库**

【未跟踪状态】 【 已跟踪状态 】

未修改的 已修改的 缓存区

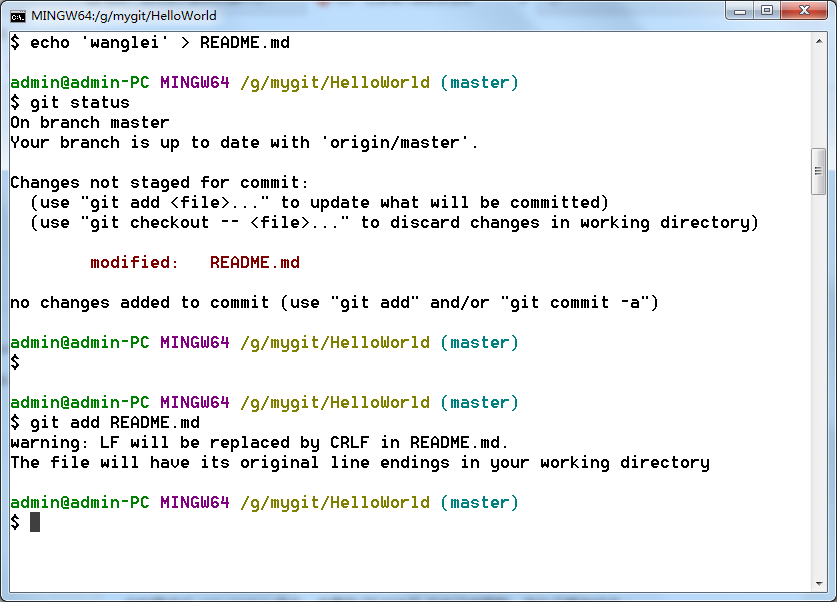


### **检查当前文件状态**

要查看哪些文件处于什么状态，可以用 git status 命令。

$ echo 'wanglei > README.nd

$ git status



可以看到有变化了，提示发现更新。

然后使用git add

### **跟踪新文件**

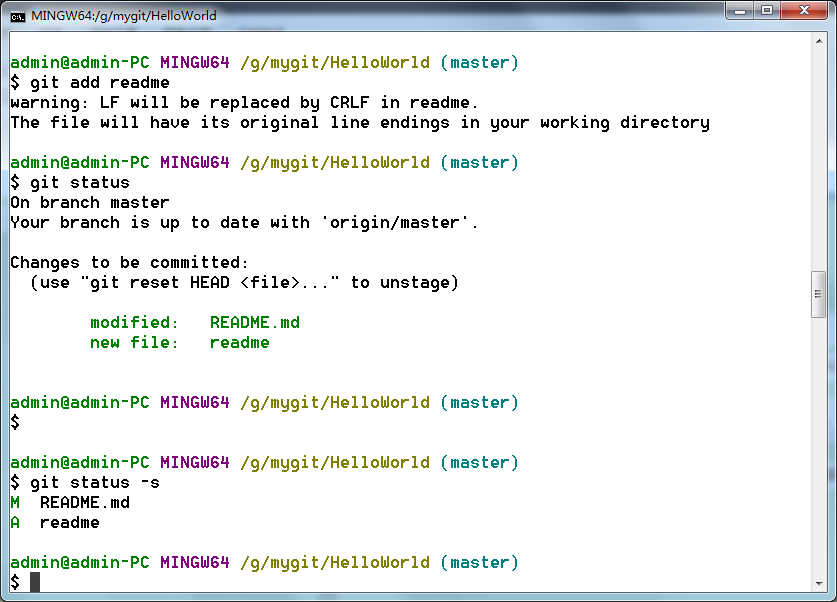
使用命令 git add 开始跟踪一个文件。 所以，要跟踪 README 文件，运行：

$ git add README.md

此时再运行 git status 命令，会看到 README 文件已被跟踪，并处于暂存状态：

### **状态简览**

git status 命令的输出十分详细，但其用语有些繁琐。 如果你使用 git status -s 命令或 git status --short 命令，你将得到一种更为紧凑的格式输出。 运行 git status -s ，状态报告输出如下：



让修改结果输出更容易发现。

### **忽略文件**

一般我们总会有些文件无需纳入 Git 的管理，也不希望它们总出现在未跟踪文件列表。 通常都是些自动生成的文件，比如日志文件，或者编译过程中创建的临时文件等。 在这种情况下，我们可以创建一个名为 .gitignore 的文件，列出要忽略的文件模式。 来看一个实际的例子：

$ cat .gitignore

\*.[oa]

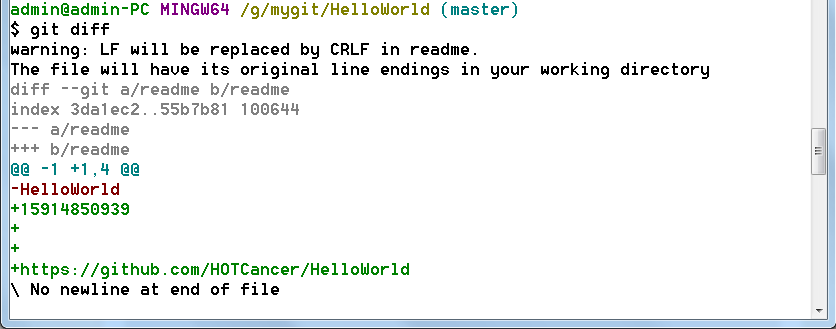
\*~

第一行告诉 Git 忽略所有以 .o 或 .a 结尾的文件。一般这类对象文件和存档文件都是编译过程中出现的。 第二行告诉 Git 忽略所有以波浪符（~）结尾的文件，许多文本编辑软件（比如 Emacs）都用这样的文件名保存副本。 此外，你可能还需要忽略 log，tmp 或者 pid 目录，以及自动生成的文档等等。 要养成一开始就设置好 .gitignore 文件的习惯，以免将来误提交这类无用的文件。

### **查看已暂存和未暂存的修改**

 git diff 很好用！！

查看修改了哪些内容



### **提交更新**

现在的暂存区域已经准备妥当可以提交了。 在此之前，请一定要确认还有什么修改过的或新建的文件还没有 git add 过，否则提交的时候不会记录这些还没暂存起来的变化。 这些修改过的文件只保留在本地磁盘。 所以，每次准备提交前，先用 git status 看下，是不是都已暂存起来了， 然后再运行提交命令 git commit：

$ git commit

这种方式会启动文本编辑器以便输入本次提交的说明。 (默认会启用 shell 的环境变量 $EDITOR 所指定的软件，一般都是 vim 或 emacs。

编辑器会显示类似下面的文本信息（本例选用 Vim 的屏显方式展示）：

# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

# On branch master

# Changes to be committed:

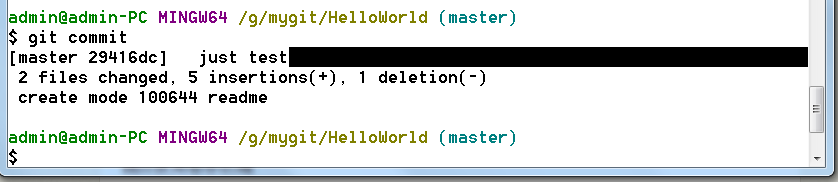
# new file: README

# modified: CONTRIBUTING.md

#

~

~".git/COMMIT\_EDITMSG" 9L, 283C



另外，你也可以在 commit 命令后添加 -m 选项，将提交信息与命令放在同一行，如下所示：

$ git commit -m "Story 182: Fix benchmarks for speed"

[master 463dc4f] Story 182: Fix benchmarks for speed

2 files changed, 2 insertions(+)

create mode 100644 README

提交后它会告诉你，当前是在哪个分支（master）提交的，本次提交的完整 SHA-1 校验和是什么（463dc4f），以及在本次提交中，有多少文件修订过，多少行添加和删改过。

### **移除文件**

要从 Git 中移除某个文件，就必须要从已跟踪文件清单中移除（确切地说，是从暂存区域移除），然后提交。 可以用 git rm 命令完成此项工作，并连带从工作目录中删除指定的文件，这样以后就不会出现在未跟踪文件清单中了。

### **移动文件**

不像其它的 VCS 系统，Git 并不显式跟踪文件移动操作。 如果在 Git 中重命名了某个文件，仓库中存储的元数据并不会体现出这是一次改名操作。 不过 Git 非常聪明，它会推断出究竟发生了什么，至于具体是如何做到的，我们稍后再谈。

既然如此，当你看到 Git 的 mv 命令时一定会困惑不已。 要在 Git 中对文件改名，可以这么做：

$ git mv file\_from file\_to

它会恰如预期般正常工作。 实际上，即便此时查看状态信息，也会明白无误地看到关于重命名操作的说明：

$ git mv README.md README

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

renamed: README.md -> README

其实，运行 git mv 就相当于运行了下面三条命令：

$ mv README.md README

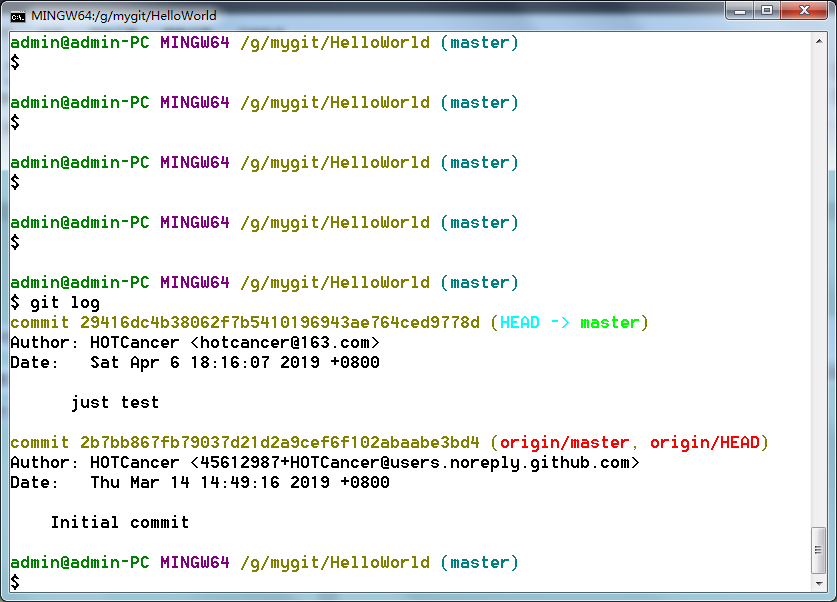
$ git rm README.md

$ git add README

如此分开操作，Git 也会意识到这是一次改名，所以不管何种方式结果都一样。 两者唯一的区别是，mv 是一条命令而另一种方式需要三条命令，直接用 git mv 轻便得多。 不过有时候用其他工具批处理改名的话，要记得在提交前删除老的文件名，再添加新的文件名。

## **查看提交历史**

在提交了若干更新，又或者克隆了某个项目之后，你也许想回顾下提交历史。 完成这个任务最简单而又有效的工具是 git log 命令。



# 2.5 Git 基础 - 远程仓库的使用

## **远程仓库的使用**

为了能在任意 Git 项目上协作，你需要知道如何管理自己的远程仓库。 远程仓库是指托管在因特网或其他网络中的你的项目的版本库。 你可以有好几个远程仓库，通常有些仓库对你只读，有些则可以读写。 与他人协作涉及管理远程仓库以及根据需要推送或拉取数据。 管理远程仓库包括了解如何添加远程仓库、移除无效的远程仓库、管理不同的远程分支并定义它们是否被跟踪等等。

### **查看远程仓库**

如果想查看你已经配置的远程仓库服务器，可以运行 git remote 命令。 它会列出你指定的每一个远程服务器的简写。 如果你已经克隆了自己的仓库，那么至少应该能看到 origin - 这是 Git 给你克隆的仓库服务器的默认名字：

运行 git remote 命令

$ git remote

origin

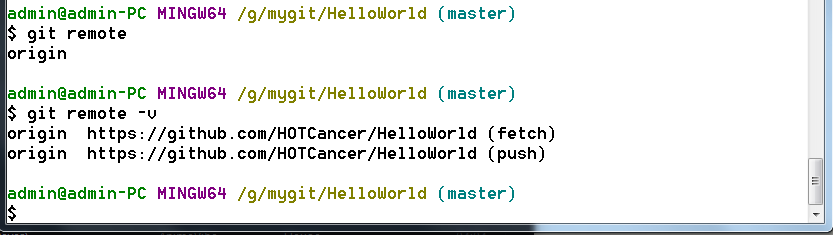
显示origin是正常的。

也可以指定选项 -v，会显示需要读写远程仓库使用的 Git 保存的简写与其对应的 URL。

$ git remote -v

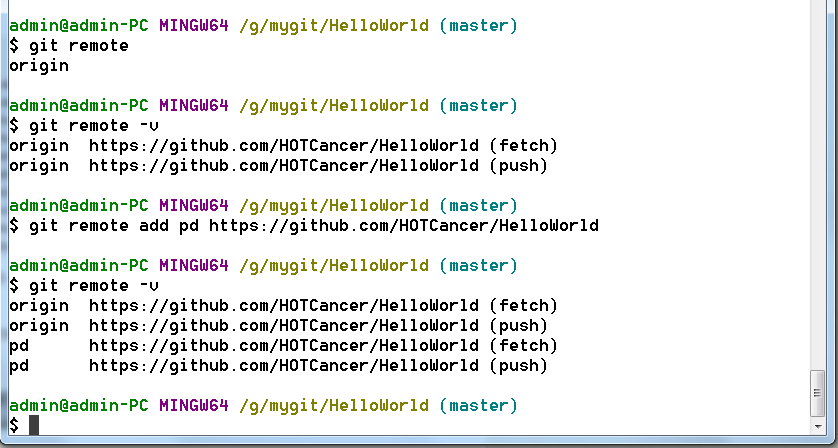
origin https://github.com/schacon/ticgit (fetch)

origin https://github.com/schacon/ticgit (push)



### **添加远程仓库**

我在之前的章节中已经提到并展示了如何添加远程仓库的示例，不过这里将告诉你如何明确地做到这一点。 运行 git remote add <shortname> <url> 添加一个新的远程 Git 仓库，同时指定一个你可以轻松引用的简写：

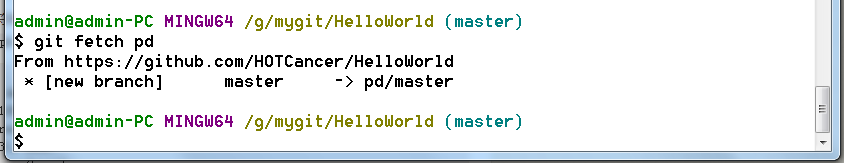


现在你可以在命令行中使用字符串 pb 来代替整个 URL。 例如，如果你想拉取 HOTCancer的仓库中有但你没有的信息，可以运行 git fetch pb：

### **从远程仓库中抓取与拉取**

就如刚才所见，从远程仓库中获得数据，可以执行：

$ git fetch [remote-name]



这个命令会访问远程仓库，从中拉取所有你还没有的数据。 执行完成后，你将会拥有那个远程仓库中所有分支的引用，可以随时合并或查看。

如果你使用 clone 命令克隆了一个仓库，命令会自动将其添加为远程仓库并默认以 “origin” 为简写。 所以，git fetch origin 会抓取克隆（或上一次抓取）后新推送的所有工作。 必须注意 git fetch 命令会将数据拉取到你的本地仓库 - 它并不会自动合并或修改你当前的工作。 当准备好时你必须手动将其合并入你的工作。

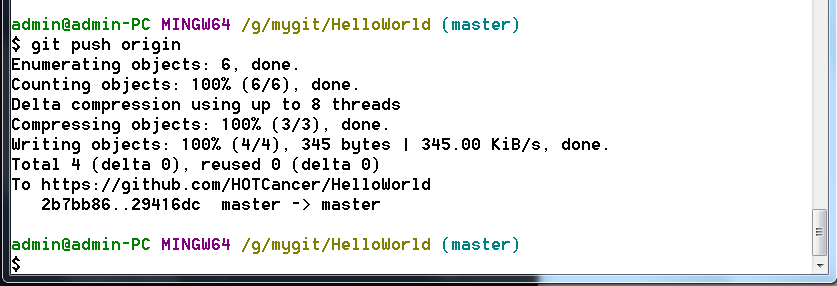
如果你有一个分支设置为跟踪一个远程分支（阅读下一节与 Git 分支 了解更多信息），可以使用 git pull 命令来自动的抓取然后合并远程分支到当前分支。 这对你来说可能是一个更简单或更舒服的工作流程；默认情况下，git clone 命令会自动设置本地 master 分支跟踪克隆的远程仓库的 master 分支（或不管是什么名字的默认分支）。 运行 git pull 通常会从最初克隆的服务器上抓取数据并自动尝试合并到当前所在的分支。

### **推送到远程仓库**

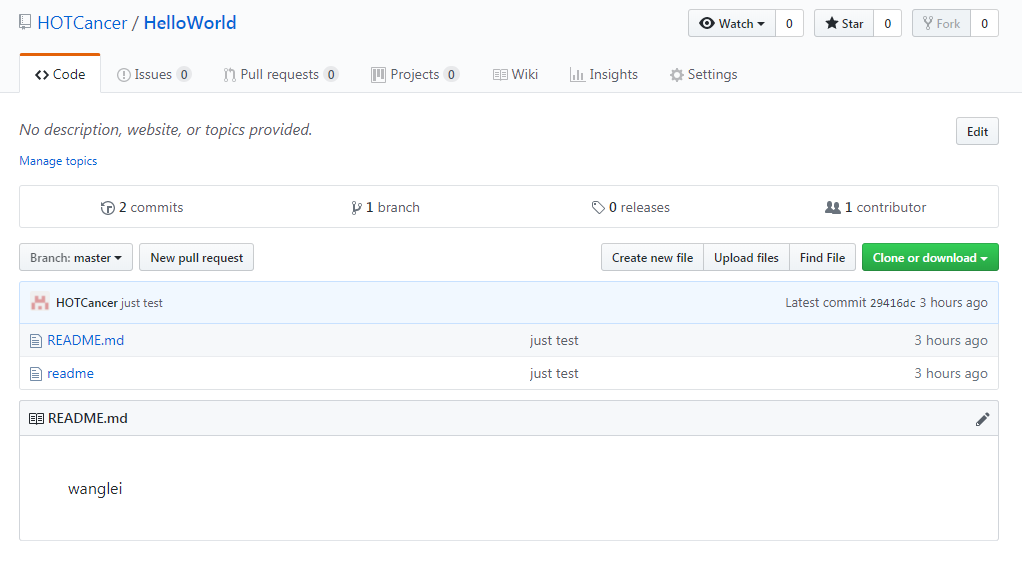
当你想分享你的项目时，必须将其推送到上游。 这个命令很简单：git push [remote-name] [branch-name]。 当你想要将 master 分支推送到 origin 服务器时（再次说明，克隆时通常会自动帮你设置好那两个名字），那么运行这个命令就可以将你所做的备份到服务器：

$ git push origin master

只有当你有所克隆服务器的写入权限，并且之前没有人推送过时，这条命令才能生效。 当你和其他人在同一时间克隆，他们先推送到上游然后你再推送到上游，你的推送就会毫无疑问地被拒绝。 你必须先将他们的工作拉取下来并将其合并进你的工作后才能推送。 阅读 [Git 分支](https://git-scm.com/book/zh/v2/ch00/ch03-git-branching) 了解如何推送到远程仓库服务器的详细信息。

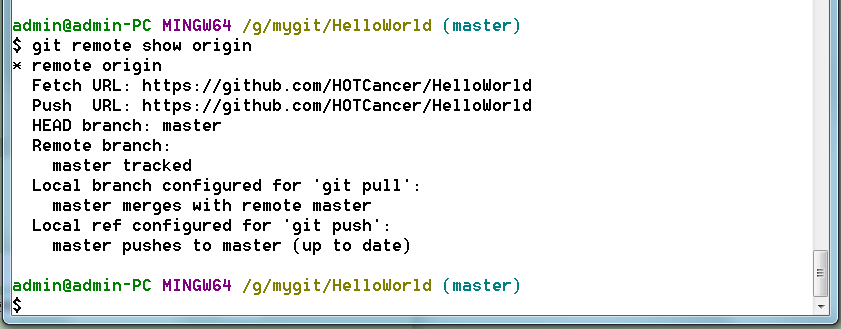


网页端也更新好了



### **查看远程仓库**

如果想要查看某一个远程仓库的更多信息，可以使用 git remote show [remote-name] 命令。



它同样会列出远程仓库的 URL 与跟踪分支的信息。 这些信息非常有用，它告诉你正处于 master 分支，并且如果运行 git pull，就会抓取所有的远程引用，然后将远程 master 分支合并到本地 master 分支。 它也会列出拉取到的所有远程引用。

### **列出标签**

在 Git 中列出已有的标签是非常简单直观的。 只需要输入 git tag：

### **创建标签**

Git 使用两种主要类型的标签：轻量标签（lightweight）与附注标签（annotated）。

一个轻量标签很像一个不会改变的分支 - 它只是一个特定提交的引用。

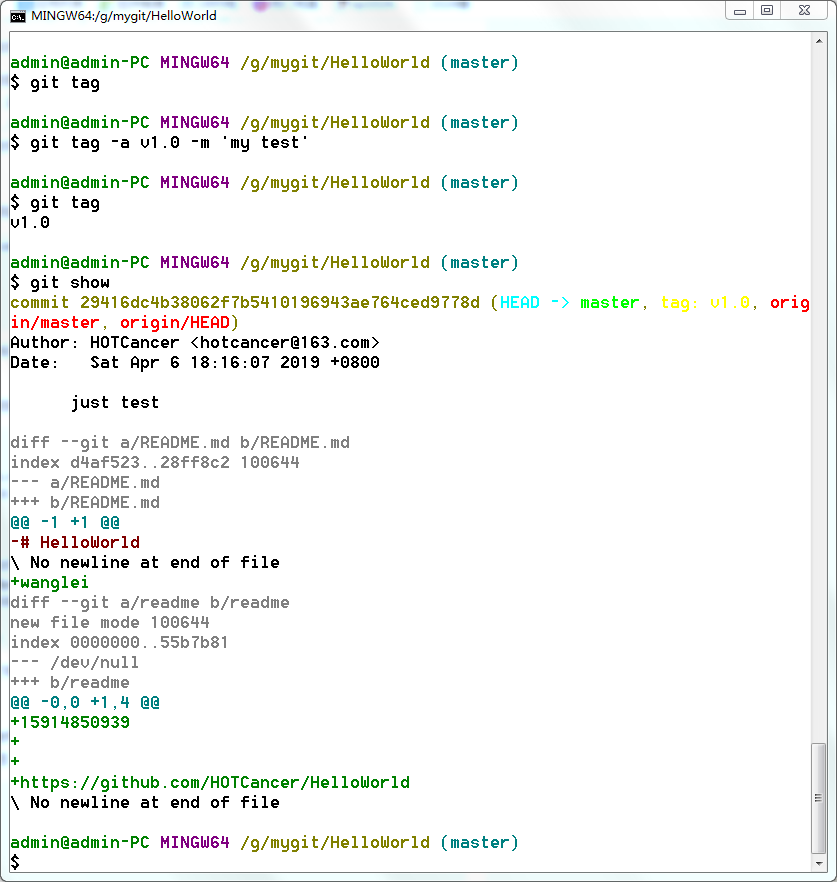
然而，附注标签是存储在 Git 数据库中的一个完整对象。 它们是可以被校验的；其中包含打标签者的名字、电子邮件地址、日期时间；还有一个标签信息；并且可以使用 GNU Privacy Guard （GPG）签名与验证。 通常建议创建附注标签，这样你可以拥有以上所有信息；但是如果你只是想用一个临时的标签，或者因为某些原因不想要保存那些信息，轻量标签也是可用的。

### **附注标签**

在 Git 中创建一个附注标签是很简单的。 最简单的方式是当你在运行 tag 命令时指定 -a 选项：

$ git tag -a v1.0 -m 'my test'

通过使用 git show 命令可以看到标签信息与对应的提交信息：



小结：

Git配合GitHub的使用。

命令

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 命令 | 意义 | 用法举例 |
| 1 | git config | 配置本机（本地）git工具 | git config --list  git config --global uesr.name=” ”  git config --global uesr.email= |
| 2 | git init | 初始化 |  |
| 3 | git clone | 克隆 | git clone https://github.com/libgit2/libgit2 |
| 4 | git status | 哪些文件处于什么状态 | 可以查看目录下文件状态，如未跟踪的，已跟踪的（未修改的，已修改的，缓存区） |
| 5 | git add | 开始跟踪一个新文件 | 向本地git里添加跟踪文件。 |
| 6 | git diff | 查看已暂存和未暂存的修改 | 查看文件修改详情，添加了哪些，删除了哪些 |
| 7 | git commit | 提交更新 | 提交前要先检查git add和git stat有没有什么异常，才能提交到本地git |
| 8 | git log | 查看提交历史 |  |
| 9 | git remote | 查看远程仓库 | git remote -v |
| 10 | git push | 推送到远程仓库 | git push origin |
| 11 | git tag | 标签 | git tag -a v1.0 -m 'my test' |
| 12 | git show | 看到标签信息与对应的提交信息 |  |
|  |  |  |  |
| 拓展命令 |  |  |  |
|  | git reset | 回滚到指定版本 | git reset --hard 6c439d2fd0d943f36f3ee84e158ff |
|  | git reflog | 查看追回的记录 | git reflog  git reset --hard 0972f4b |
|  |  |  |  |
|  | git branch | 创建分支 |  |
|  | git checkout | 切换分支 | git checkout 分支名称 |
|  | git branch -m | 切换到指定分支 | git branch -m 分支名称  创建并切换到指定分支 |
|  | git branch | 查看所有分支 | git branch |
|  | git branch -d | 删除分支 | git branch -d 分支名称 |
|  | git merge | 将指定分支合并到当前分支 | git merge 分支名称 |

# 3.1 Git 分支 - 分支简介

使用分支意味着你可以把你的工作从开发主线上分离开来，以免影响开发主线。 在很多版本控制系统中，这是一个略微低效的过程——常常需要完全创建一个源代码目录的副本。对于大项目来说，这样的过程会耗费很多时间。

有人把 Git 的分支模型称为它的`‘必杀技特性’'，也正因为这一特性，使得 Git 从众多版本控制系统中脱颖而出。 为何 Git 的分支模型如此出众呢？ Git 处理分支的方式可谓是难以置信的轻量，创建新分支这一操作几乎能在瞬间完成，并且在不同分支之间的切换操作也是一样便捷。

