[浅谈使用git进行版本控制](https://www.cnblogs.com/wj-1314/p/7992543.html)

     小编在学习可视化的时候，接触到git，所以这里写一下关于GitHub的有关知识，写这个的目的还是巩固自己的学习，一方面可以提高自己，另一方面回头看一下，有什么更深层次的东西还可以再记录。

     首先说一下版本控制软件，它可以让我们能够拍摄处于可行状态的项目的快照，修改项目（如实现新功能）后，如果项目不能正常运行，可以恢复到前一个可行状态。

    通过使用版本控制，我们可以无忧无虑的改进项目，不用担心项目因为我们犯错而遭到破坏，对于大型项目来说，这显得尤为重要，但是对于小项目来说，哪怕只包含一个文件的程序，也大有裨益。

    GibHub的名字源于Git，Git是一个分布式版本控制系统，让程序员团队能够协作开发项目，Git帮助大家管理为项目所做的工作，避免一个人所做的修改影响其他人所做的修改。你在项目中实现一个新功能的时候，Git将跟踪你对每个文件所做的修改。确定代码可行后，你将提交所做的修改，而Git将记录项目最新的状态，如果你犯了错，想撤销所做的修改，可轻松的返回以前的任何可行状态。GitHub上的项目都存储在仓库中，后者包含与项目相关联的一切：代码，项目参与者的信息，问题和bug报告等

   下面讲一下安装Git（小编是windows系统，就只演示windows系统的安装）

**第一步：下载（https://git-for-windows.github.io/），下载好如下图**

https://images2017.cnblogs.com/blog/1226410/201712/1226410-20171206164351800-1556555364.png

**第二步：安装过程，此处省略，要想安装可以点击下面链接**

**【Git的安装步骤博文<http://www.cnblogs.com/wj-1314/p/7993819.html>】**

**第三步：配置**

Git跟踪谁修改了项目，哪怕参与项目开发的只有一个人。为此，Git需要知道你的用户名和电子邮件。你必须提供用户名，但可以使用虚构的电子邮件地址：

git config --global user.name "Username"

git config --global user.email "Username@example.com"

         因为Git是分布式版本控制系统，所以，每个机器都必须自报家门：你的名字和Email地址。你也许会担心，如果有人故意冒充别人怎么办？这个不必担心，首先我们相信大家都是善良无知的群众，其次，真的有冒充的也是有办法可查的。

        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。

**第四步：创建项目**

我们来创建一个要进行版本控制的项目（又称版本库）。在你的系统创建一个文件夹，并将其命名为learngit。

我创建learngit的程序如下：

mkdir learngit #创建文件learngit  
cd learngit #进入learngit文件里面  
pwd #显示learngit的存在目录

我创建了一个hello.world.py程序，如下：

print("hello world")

**第五步：忽略文件**

忽略扩展名为.pyc的文件，它是根据.py文件自动生成啊，我们无需让Git去跟踪。这些文件存储在目录\_\_pycache\_\_中，未来让git忽略它，我们创建一个名为.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）

\_pycache\_/

https://images2017.cnblogs.com/blog/1226410/201712/1226410-20171208223454952-383251847.png

**第六步：初始化仓库**

我们创建了一个文件，并通过git init命令把这个目录变成Git可以管理的仓库：

$ git init

Initialized empty Git repository in C:/Users/learngit/.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。要是删除这个东西，则丢弃项目的所有记录。

       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。

**第七步：检查状态**

在执行其他操作之前，先来看一下状态：

[复制代码](javascript:void(0);)

$ git status  
On branch master

No commits yet

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

        hello\_world.py

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

[复制代码](javascript:void(0);)

　　在Git中，分支是项目的一个版本，从这里的输出我们可以知道，我们位于分支的master

　　正常的项目，查看状态应该如下：

|  |  |
| --- | --- |
| 1  2  3  4  5 | $ git status  On branch master  Your branch is up to date with 'origin/master'.    nothing to commit, working tree clean |

　　我们每次查看项目的状态时候，输出的都是我们位于分支master上，接下里的输出表明，我们将进行初始项目的日叫，**提交**是项目在特定时间的快照。

  Git指出了项目中未被跟踪的文件，因为我们还没有告诉他要跟踪那些文件，接下里我们被告知没有任何东西添加到当前提交里面，但我们可能需要将为跟踪的文件加入仓库

**第八步：将文件加入到仓库**

[复制代码](javascript:void(0);)

$ git add .

。。。@▒▒▒▒▒ĵ▒▒▒ MINGW64 ~/learngit (master)

$ git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: hello\_world.py

[复制代码](javascript:void(0);)

命令  git add .   将项目中未被跟踪的文件都加入到仓库中，它不提交这些文件，而只是让git开始关注他们。现在我们检查这个项目的状态，发现Git找到了需要提交的文件的一些修改，标签 new file  表示这些文件是新加入的。

**第九步：执行提交**

$ git commit -m "Started project"

[master (root-commit) 5d6ceca] Started project

1 file changed, 1 insertion(+)

create mode 100644 hello\_world.py

     我们在执行   git commit -m "Started project"    的时候以拍摄项目的快照。标志-m 让Git接下里的消息（“Started project"）记录到项目中的历史记录中，输出表明我们在分支master 上，而且有一个文件被修改了

     简单解释一下git commit命令，-m后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录。

**第十步：查看提交历史**

[复制代码](javascript:void(0);)

$ git log

commit 5d6cecad80427924b94b14c6fd2bb82a4fa86840 (HEAD -> master)

Author: username <xxxxxxxxxx.example.com>

Date: Sat Dec 9 20:16:17 2017 +0800

Started project

[复制代码](javascript:void(0);)

我们每次提交的时候，Git都会生成一个包含40字符的独一无二的引用ID，它记录提交是谁执行的，提交的时间以及提交的指定消息，并非在任何情况下你都需要所有的这些信息，因此Git提供一个选项，让我们能够打印提交历史条目的更简单的版本。

$ git log --pretty=oneline

5d6cecad80427924b94b14c6fd2bb82a4fa86840 (HEAD -> master) Started project

标志 --pretty=oneline   指定显示一项最重要的信息，提交的引用ID以及为提交记录的消息。

**第十一步：第二次提交**

为了显示版本控制的强大，我们需要对项目进行修改，并提交所做的修改。为此，我们在文件hello.world.py中再添加一行代码。

print("hello world")

print("hello git")

如果我们查看项目的状态，将发现Git注意到这个文件的变化

[复制代码](javascript:void(0);)

$ 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: hello\_world.py

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

[复制代码](javascript:void(0);)

  这个之处了我们当前所在的分支为master,其中做出修改的文件是hello\_world.py，而且指出所做的修改还没有提交。

    接下来我们提交所做操作，并在查看操作。

     这一步，我们执行了提交，并且在执行命令git commit 的时候指定了标志-am.标志-a 让Git 将仓库中所有修改了的文件都加入当前提交中，（如果我们两次提交之间加入了新文件，我们执行get add . 操作，将新文件加入到仓库中）标志-m让Git咱提交历史中记录一条消息。

$ git commit -am "Extrended greeting."

[master b4ee15d] Extrended greeting.

1 file changed, 2 insertions(+), 1 deletion(-)

$ git status

On branch master

nothing to commit, working tree clean

$ git log --pretty=oneline

b4ee15ddf5274f488db8c74c327065c6f331ec5e (HEAD -> master) Extrended greeting.

5d6cecad80427924b94b14c6fd2bb82a4fa86840 Started project

我们在查看项目的状态的时候，发现工作目录也是干净的，最后我们发现提交历史中包含两个提交。

**第十二步：撤销修改**

下面来看看如何放弃所做的修改，恢复到一个可行状态，为此，我们首先在hello\_world.py中添加一行代码

hello\_world.py

print("hello world")

print("hello git")

print("the world is bad")

保存并运行这个文件

我们查看状态，发现Git注意到所做的修改

[复制代码](javascript:void(0);)

$ 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: hello\_world.py

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

[复制代码](javascript:void(0);)

Git注意到我们修改了hello\_world.py，我么可以提交所做的修改，但是我们不提交所做的修改，而要恢复到最后一个提交，为此我们不对hello\_world.py执行任何操作————不删除刚添加的代码行，也不使用文本编辑器的撤销功能，而是在终端会话中执行如下命令：

$ git checkout .

命令  git checkout .  能够让我们恢复到以前的任何提交。命令git checkout . 放弃最后一次提交所做的所有操作，将项目恢复到最后一次提交的状态。

$ git status

On branch master

nothing to commit, working tree clean

    就这个项目而言，我们恢复到前一个状态微不足道，但是如果我们开发的是大型项目，其中数十个文件都被修改了，那么恢复到前一个状态，将撤销来自最后一次提交的对这个文件所做的所有修改，这个功能很有用，比如：实现新功能，我们可以根据需要做任意数量的修改，如果这些修改都不行，可以撤销他们，而不会对项目有任何伤害，你无需记住做了那些修改，因而不必手工撤销所做的修改，Git会替我们完成所有的工作。

**第十三步：检查以前的提交**

 我们可以检查提交历史中的任何一次提交，而不仅仅是最后一次，为此我们可以在命令git check末尾指定该提交的引用ID的前6个字符（而不是局点）。通过检查出以前的提交，我们可以对其进行审核么然后返回到最后一次提交，或者放弃最近所做的工作，并选择以前的提交。

$ git log --pretty=oneline

b4ee15ddf5274f488db8c74c327065c6f331ec5e (HEAD -> master) Extrended greeting.

5d6cecad80427924b94b14c6fd2bb82a4fa86840 Started project

[复制代码](javascript:void(0);)

$ git checkout 5d6cec

Note: checking out '5d6cec'.

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -b with the checkout command again. Example:

git checkout -b <new-branch-name>

HEAD is now at 5d6ceca... Started project

[复制代码](javascript:void(0);)

     检查出以前的提交，我们将离开分支master，并进入Git所说的Git所说的分离头指针（detached HEAD）状态，HEAD表示项目的当前状态，之所以说我们处于分离状态是因为我们离开了一个命名分支（这里是master）

      要回到分支master，可以检查出它：

|  |  |
| --- | --- |
| 1  2  3 | $ git checkout master  Previous HEAD position was 5d6ceca... Started project  Switched to branch 'master' |

　　这就会让你回到分支master 。除非i使用Git的高级功能，否则在提交以前的项目后，最好不要对项目做任何修改，然而，如果参与项目开发的人只有我们自己，而我自己又想放弃所有提交，并恢复到以前的状态，也可以将项目重置到以前的状态，为此，可在处于分支master上的任何情况下，执行如下命令。

$ git status

On branch master

nothing to commit, working tree clean

$ git log --pretty=oneline

b4ee15ddf5274f488db8c74c327065c6f331ec5e (HEAD -> master) Extrended greeting.

5d6cecad80427924b94b14c6fd2bb82a4fa86840 Started project

$ git reset --hard 5d6cec

HEAD is now at 5d6ceca Started project

$ git status

On branch master

nothing to commit, working tree clean

$ git log --pretty=oneline

5d6cecad80427924b94b14c6fd2bb82a4fa86840 (HEAD -> master) Started project

首先，我们查看了状态，确认我们在分支master上，查看历史提交时，我们看见了两个提交。

然后，我们执行了命令    git reset --hard    ，并在其指定了要永久性的恢复到的提交的引用ID的前6个字符。

接下来，我们在次查看状态，大仙，我们在分支master上，并且没有需要任何修改，

最后，我们再次查看提交的历史状态时候，我们发现我们处于重新开始的提交中。

**第十四步：删除仓库**

  有时候，仓库的历史纪录被我们搞乱了，而我们又不知道如何恢复，这时候我们首先应该考虑百度一下，看看自己的问题出在那里，如果无法恢复，而且参与项目的人只有自己，可以继续使用这些文件，但需要将这些项目的历史纪录删除——删除.git 这不会影响任何文件的当前状态，而只会删除文件的所有提交，因此我们将无法检查出项目的其他任何状态。

   为此，可以打开一个文件浏览器，并将目录.git 删除，也可以通过命令完成这个任务。但是这样做过我们需要创建一个新的仓库，以重新对这些修改进行跟踪。

  下面演示了如何在终端会话中完成这个过程。

$ git status

On branch master

nothing to commit, working tree clean

$ rm -rf .git

rm -rf .git 表示删除目录.git,删除后我们将继续查看状态。

$ git status

fatal: Not a git repository (or any of the parent directories): .git

    这个意思是被告知我们，这不是一个仓库，（git用来跟踪仓库的信息都存储爱文件夹.git中，因此删除该文件夹也将会删除整个仓库）

接下来，我们用命令 git init 来创建一个新的仓库，并查看状态

$ git init

Initialized empty Git repository in C:/Users//learngit/.git/

[复制代码](javascript:void(0);)

$ git status

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

hello\_world.py

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

[复制代码](javascript:void(0);)

  从状态中我们发现，又回到了初始状态，等待第一次提交，我们下面将所有的文件都添加到仓库中，并执行第一次提交，最后检查状态，如下：

$ git add .

$ git commit -m "Starting over"

[master (root-commit) 81350ab] Starting over

1 file changed, 1 insertion(+)

create mode 100644 hello\_world.py

$ git status

On branch master

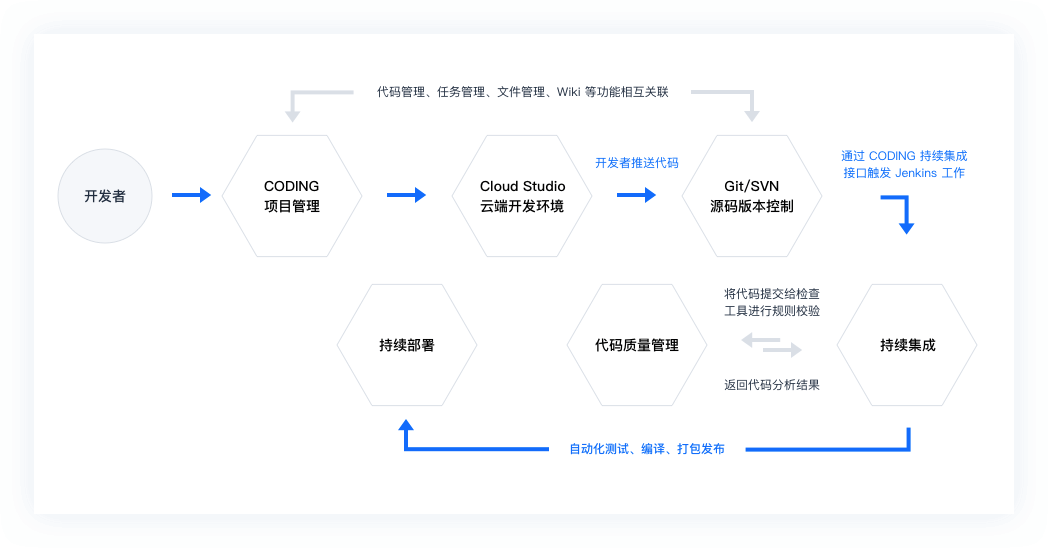
nothing to commit, working tree clean

从检查状态我们发现，我们在分支master上，并且没有任何未提交的修改。

**这就是版本控制的基本操作，希望能多多练习，这样我们才能学会版本控制。**

**要想好好了解GitHub的使用，请参考小编使用GitHub的点点滴滴的博客：[https://www.cnblogs.com/wj-1314/p/9901763.html](https://www.cnblogs.com/wj-1314/p/9901763.html" \t "_blank)**

* **推荐一个版本控制工具**
* **CODING**：研发管理系统：<https://coding.net/> ，以 Git 代码托管与发布为核心，涵盖代码提交、代码审查、Bug 追踪等开发场景，通过任务、文件、Wiki 等工具，获取需求管理、任务追踪、知识库管理等一系列写作功能。基于 Git 的版本控制保障了公司代码资产安全，实现自动构建，减少人工干预，提升业务系统质量。
* **系统基本架构：**

[](https://coding.net/?utm_source=zzrc&utm_medium=article&utm_campaign=April2019)

* **企业级项目工作一体化：**

[](https://coding.net/?utm_source=zzrc&utm_medium=article&utm_campaign=April2019)