# Git及Github总结

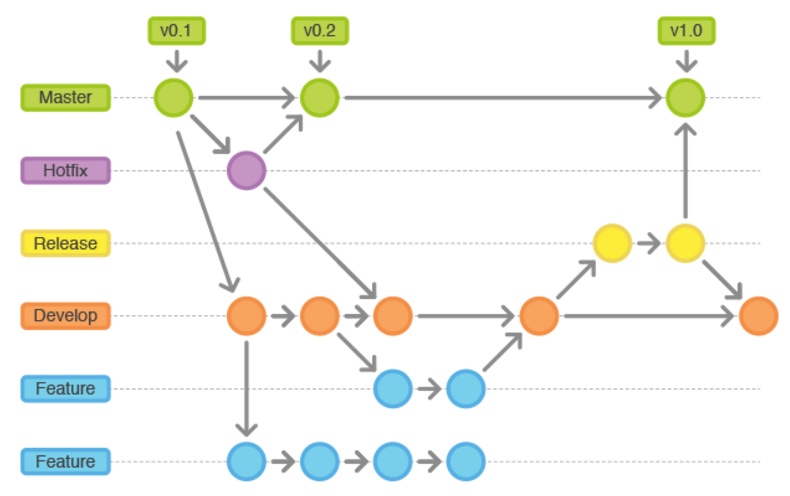
GitHub是通过Git进行版本控制的软件源代码托管服务，是世界上最大的代码存放网站和开源社区。

## Git版本控制

Git如何进行版本控制了，下面举个例子：

1. Master为主干，只有版本经理可以进行管理，这里对应Version。
2. 基于Master拉一个Develop分支，这个分支是给开发进行新需求开发和bug修改使用，这里可以每日出个版本，就对应Daily和Bxx（忘了那个词了），Daily版本给开发自验bug修复，和查看新需求功能是否正常，B版本是给测试转测版本。
3. 开发到一定程度，需求合入的差不多了，bug也改的差不多，可以交付了，基于Develop拉个Release版本，然后交付。同时进入下一个Release。Release也只有版本经理可以管理。
4. 一定周期，版本经理会将Release版本合入Master，然后进入下一个Verion。
5. Hotfix即补丁，看哪个分支需要。理论上Develop分支不需要补丁。Release分支周期长，也是要发货的版本，对于以前的版本要出补丁。
6. 如果有个别厂商有特殊需求，就基于某个B版本拉个分支进行开发。

注：描述部分和图不太一样，主要是自己随便想的，没按图解释，其次最最主要的是自己懒，盗图不想修改。其实如和进行版本控制全看各个公司规定，只要提前规划好GitFlow，和命名规范就好。比如这里的version，可以同一个Master这么发展过来，但是有些公司不同version之间可能基于的语言、框架都变了这个时候就需要另起个Repository（代码仓）了。当然，



## Git基本用法

如果是第一次使用 Git，均需要配置用户信息，包括用户名与 Email（如下所示），以便以后每次 Git 提交时都可以自动引用这两条信息，说明是谁更新与提交了代码。

$ git config --global user.name "Pi Guang Ming"

$ git config --global user.email [piguangming@gmail.com](mailto:piguangming@gmail.com)

详细用法可以参考：

<https://www.ibm.com/developerworks/cn/opensource/os-cn-tourofgit/index.html>

<https://git-scm.com/doc>

<https://www.runoob.com/git/git-tutorial.html>

<https://zhuanlan.zhihu.com/p/23478654>

## 版本控制实现简介

创建一个项目，假设叫做LearningForerver。这一步可以在自己电脑上先

mkdir LearningForever

然后进入该文件夹

git init

然后

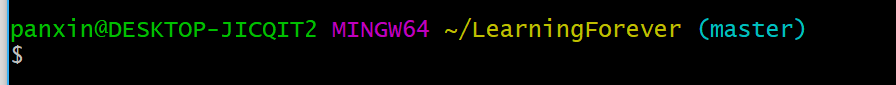
git push --set-upstream origin master

当然可以直接在GitHub上点击右上角的+号，然后点击New Repository来创建一个Repository，然后进入该Repository，点击clone or download复制链接，然后在pc端

git clone <https://github.com/XinePan/LearningForever.git>

也可以

就会在本地生成一个本地仓库



这个时候可先编辑需求文档、设计文档等，放到doc目录下

然后先

git add doc/\*

添加文件到缓存区

然后

git commit -m ”Why”

然后

git commit

提交到本地仓库，然后

git push

上传到远端仓库。（或者指定远端仓库git push origin master）

创建分支可以用

git branch panxin1

然后git checkout panxin1切换到panxin1分支。

这个只是在本地创建了分支，如果要上传到远端，第一次需要使用

git push --set-upstream origin master

如果要将panxin1合并到master，可以先切会master，然后

git merge panxin1

与 CVS、SVN 等其它版本控制系统一样，Git 也支持打 Git 标签 。在程序开发到一个阶段后，我们需要打个标签，发布一个版本，如 0.1.2，v0.1.2 等。

Git 使用的标签有两种类型：轻量级的（lightweight）和含附注的（annotated）。轻量级标签实际上就是个指向特定提交对象的引用；而含附注标签实际上是存储在仓库中的一个独立 Git 对象。相比之下，含附注标签包含信息更多，包括自身校验信息，标签名字，Email，标签日期，以及标签说明等。含附注标签本身也允许使用 GNU Privacy Guard (GPG) 来签署或验证，因此我们推荐使用含附注的标签，以便保留相关信息。

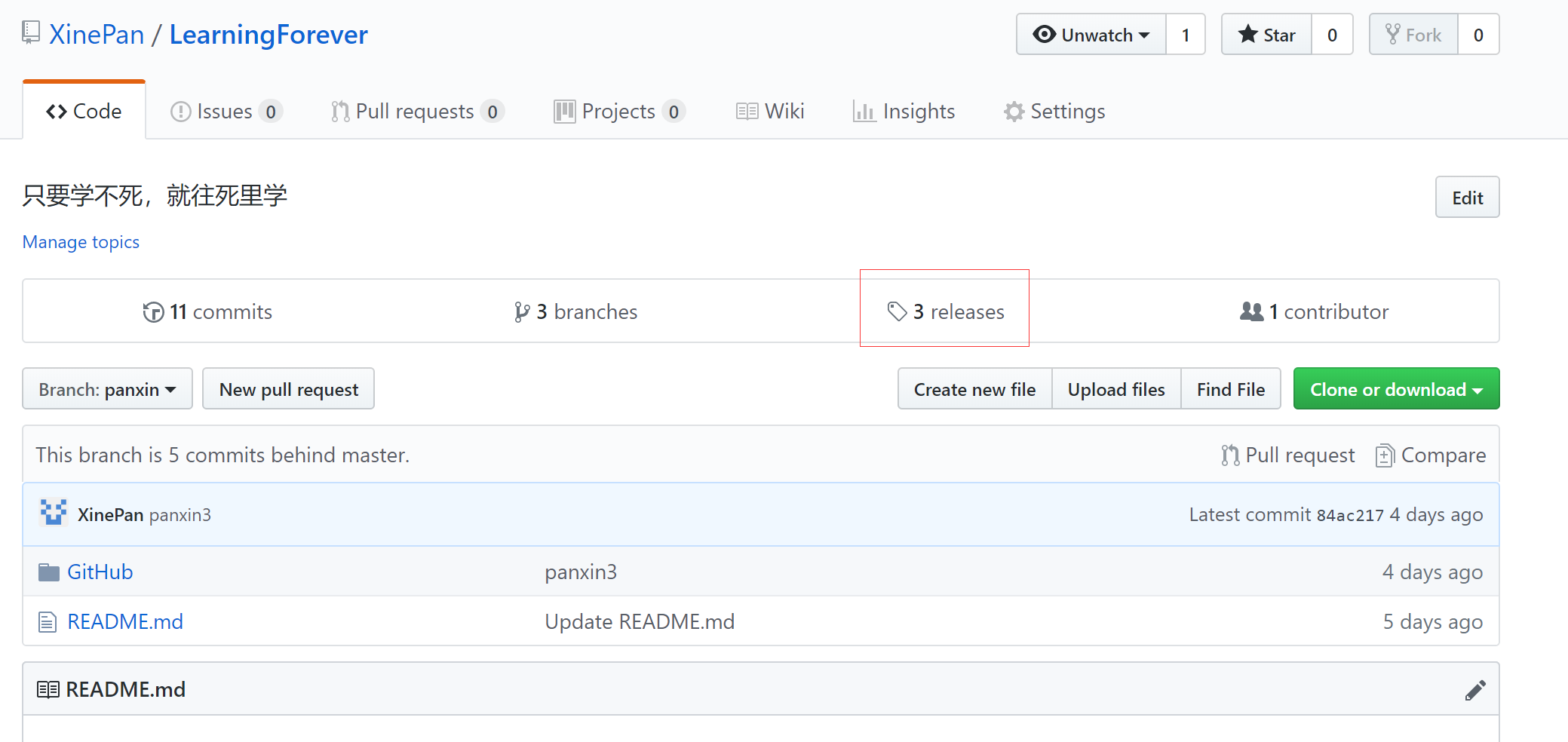
比如在某个分支下，（不一定得Master，每个分支均可以打标签，v0.1安装你规划的写就好）

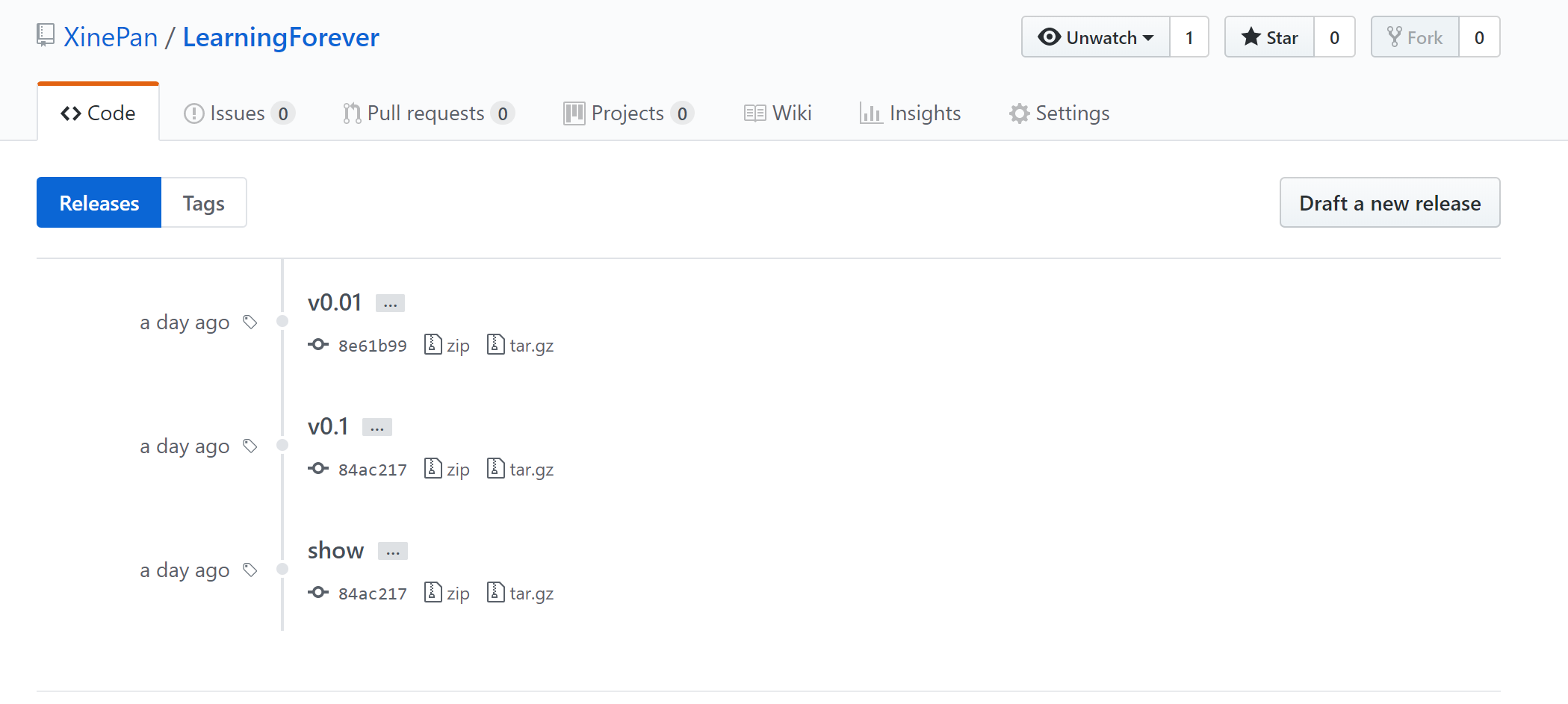
git tag -a v0.1 -m "Release version 0.1"

然后推送到远端

git push origin v0.1.2

然后在Github上可以看到打了标签的时候的压缩包。





也可以对过去的提交打标签，先查提交历史

git log --pretty=oneline

比如：

$ git log --pretty=oneline

15027957951b64cf874c3557a0f3547bd83b3ff6 Merge branch 'experiment'

a6b4c97498bd301d84096da251c98a07c7723e65 beginning write support

0d52aaab4479697da7686c15f77a3d64d9165190 one more thing

6d52a271eda8725415634dd79daabbc4d9b6008e Merge branch 'experiment'

0b7434d86859cc7b8c3d5e1dddfed66ff742fcbc added a commit function

4682c3261057305bdd616e23b64b0857d832627b added a todo file

166ae0c4d3f420721acbb115cc33848dfcc2121a started write support

9fceb02d0ae598e95dc970b74767f19372d61af8 updated rakefile

964f16d36dfccde844893cac5b347e7b3d44abbc commit the todo

8a5cbc430f1a9c3d00faaeffd07798508422908a updated readme

现在，假设在 v1.2 时你忘记给项目打标签，也就是在 “updated rakefile” 提交。 你可以在之后补上标签。 要在那个提交上打标签，你需要在命令的末尾指定提交的校验和（或部分校验和）:

$ git tag -a v1.2 9fceb02

## 分布控制

Git还有个有点就是分布控制，就是比如你可以在多个pc端对某个项目进行控制，也可以多人协作。现大型项目都不可能是一个人能完成的，因此多人协作越来越重要。

上述我们讲到了git clone将项目克隆到本地，也讲了怎么上传（git add/git commit/git push），那如果远端有更新，我们可以通过

Git fetch

然后比较 git merge

或者直接git pull

git pull相当于Git fetch和git merge。不过，少用，多用Git fetch和git merge。Git fetch和git merge，在还没有merge的时候，可以自己把控选择是否合并。

说到多人协作，不可能多人用一个账号。多人协作我就不写了，可以自己搜索，也可以参考

<https://www.jianshu.com/p/8c69d1021d98>

## 说明

本文只是象征性介绍，没有具体将命令，而且很多可以Github操作的，像如何删除branch，删除tag，可以自己搜索。太底层的原理，也没有介绍，主要现在就当个工具在用，后期在深入了解吧。