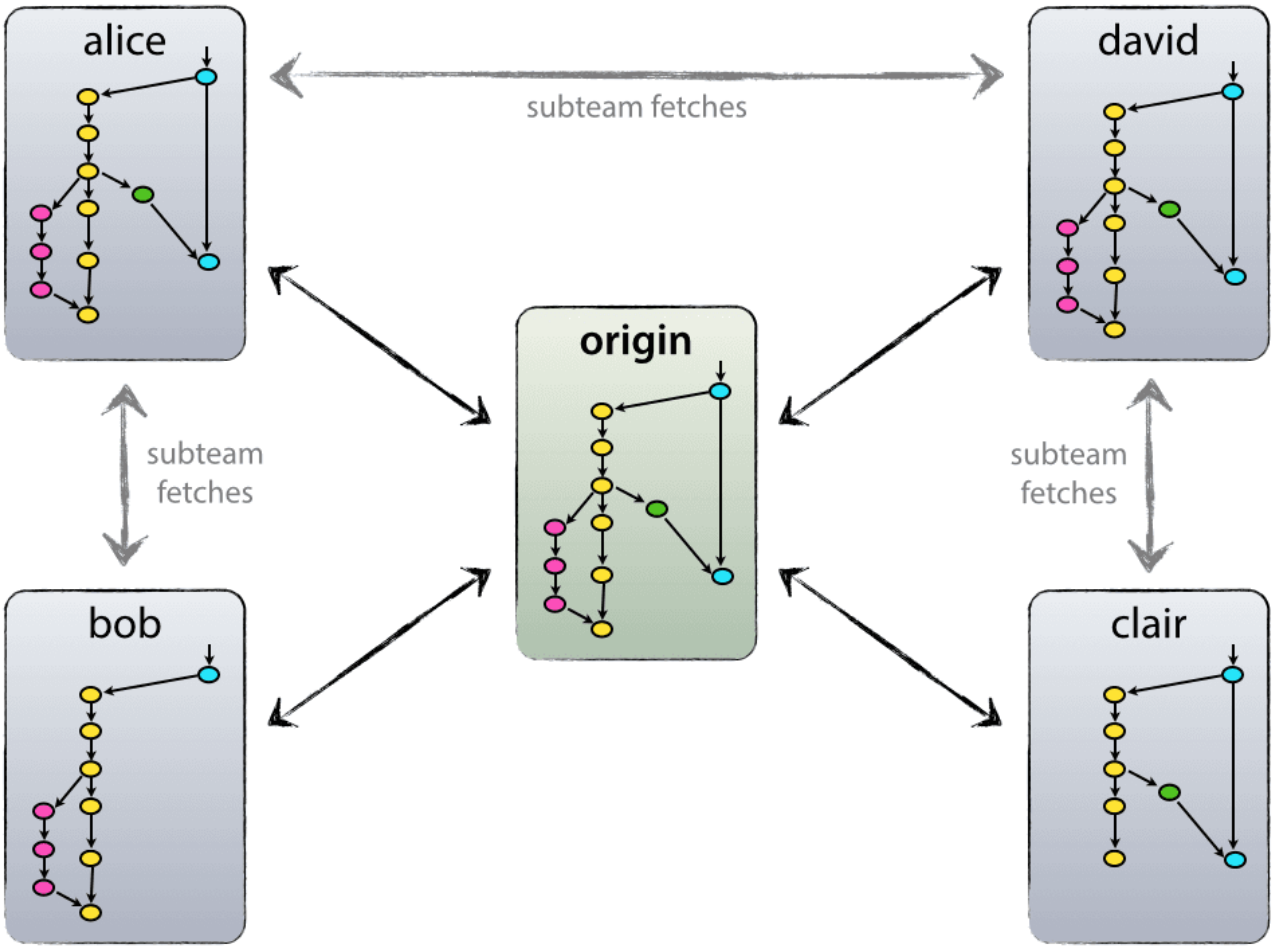
Git 在团队中的最佳实践--如何正确使用Git Flow

我们已经从SVN 切换到Git很多年了，现在几乎所有的项目都在使用Github管理, 本篇文章讲一下为什么使用Git, 以及如何在团队中正确使用。

首先先看一张图，对git版本控制有个大致的了解：



Git的优点

Git的优点很多，但是这里只列出我认为非常突出的几点。

1. 由于是分布式，所有本地库包含了远程库的所有内容。
2. 优秀的分支模型，打分支以及合并分支，机器方便。
3. 快速，在这个时间就是金钱的时代，Git由于代码都在本地，打分支和合并分支机器快速，使用个SVN的能深刻体会到这种优势。

感兴趣的，可以去看一下Git本身的设计，内在的架构体现了很多的优势，不愧是出资天才程序员Linus (Linux之父)之手。

版本管理的挑战

虽然有这么优秀的版本管理工具，但是我们面对版本管理的时候，依然有非常大得挑战，我们都知道大家工作在同一个仓库上，那么彼此的代码协作必然带来很多问题和挑战，如下：

1. 如何开始一个Feature的开发，而不影响别的Feature?
2. 由于很容易创建新分支，分支多了如何管理，时间久了，如何知道每个分支是干什么的?
3. 哪些分支已经合并回了主干?
4. 如何进行Release的管理？开始一个Release的时候如何冻结Feature, 如何在Prepare Release的时候，开发人员可以继续开发新的功能?
5. 线上代码出Bug了，如何快速修复？而且修复的代码要包含到开发人员的分支以及下一个Release?

大部分开发人员现在使用Git就只是用三个甚至两个分支，一个是Master, 一个是Develop, 还有一个是基于Develop打得各种分支。这个在小项目规模的时候还勉强可以支撑，因为很多人做项目就只有一个Release, 但是人员一多，而且项目周期一长就会出现各种问题。

Git的工作方式

分为集中式工作流、功能分支工作流、Gitflow工作流和Forking。

集中式工作流

一个远程仓库，一个主分支master，团队每个成员都有一个本地仓库，在本地仓库中进行代码的编辑、暂存和提交工作：

```
git add <some file> 或 git add .>
//`some file`代表要暂存的文件，`.`代表工作目录下的所有文件
gie commit -m "一些描述"
//提交文，描述指的是本次提交修改了什么功能或者修改了什么bug，方便以后的查看
git push -u origin master
//-u选项设置本地分支去跟踪远程对应的分支。设置好跟踪的分支后，就可以使用git push命令省去指定推送分支的参数
//发布本地仓库到远程的中央仓库中，origin是远程仓库名，master是参数告诉Git的分支，master代表主分支，当然分支可以不是主分支
```

注意:在一种情况下push命令会出错，即如果小明第一次发布代码到远程仓库，此时小红在本地开发自己的功能，那么在小红push自己的本地库到远程的时候会报错，原因是小红的本地库和远程库有分歧，**需要先pull远程库到本地，与本地库合并之后再push到远程库。**

功能分支工作流

在集中式工作流的基础上，为各个新功能分配一个专门的分支来开发，即在master主分支外在创建一个分支，程序员开发的新功能全部push到此分支上，等到功能成熟的时候再把此分支合并到主分支master上。

```
git checkout -b newbranch master
# checkout代表创建切换带新分支newbranch
#-b代表如果新分支不存在则会创建一个新分支
# 最后的master代表新分支是基于主分支创建的
```

新分支创建之后，对其的编辑、暂存和提交工作与之前一样，对其push的命令变为`git push origin newbranch`，等到新功能完善之后，通过以下命令：

```
git checkout master
git pull
git pull origin newbranch
git push
```

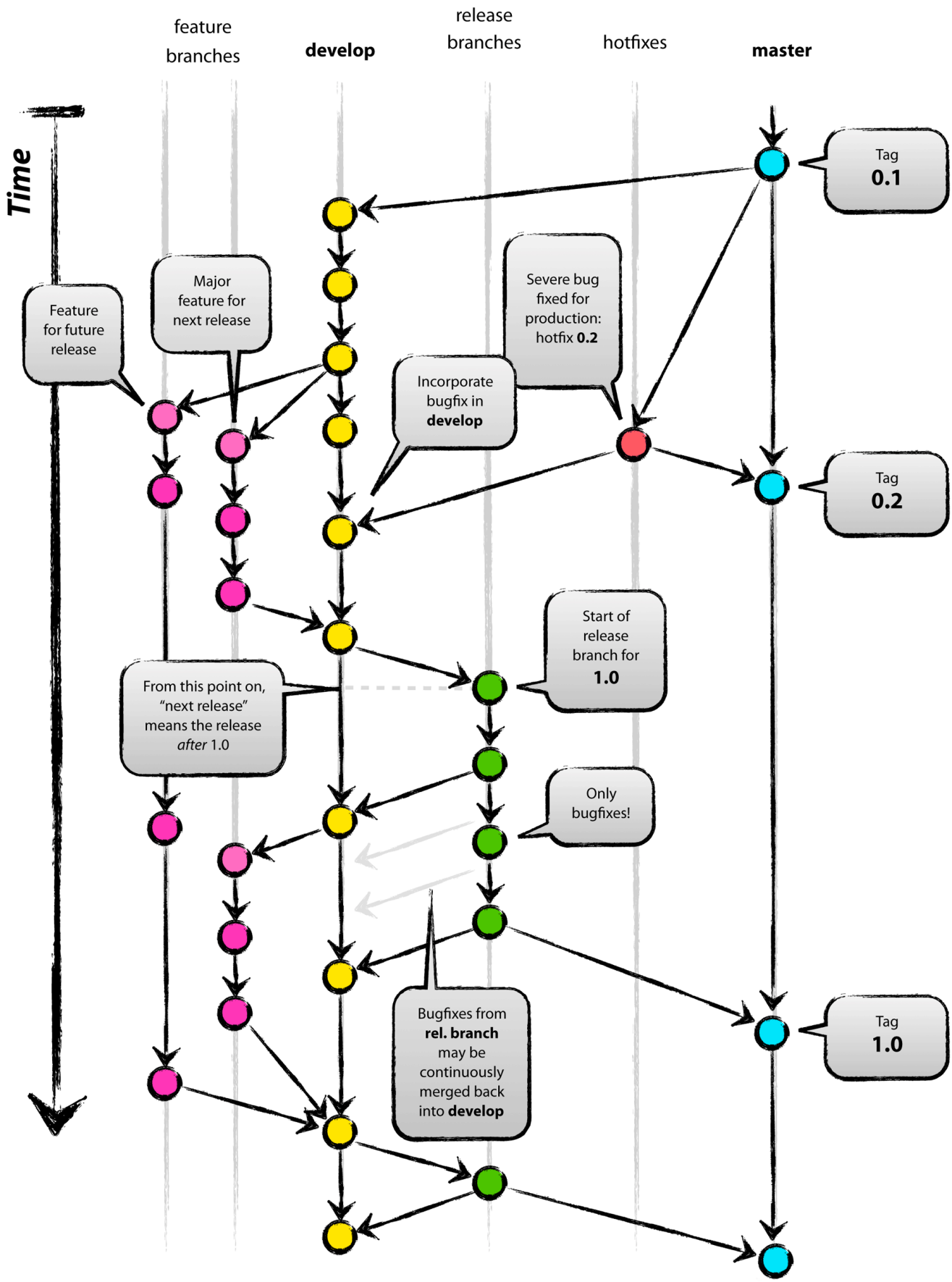
首先`git checkout master`切换到主分支，然后执行`git pull`把本地仓库的主分支上传到远程库，**再执行`git pull origin newbranch`保证合并`newbranch`分支和已经和远程一致的本地master分支，你可以使用简单`git merge newbranch`命令，但前面的命令可以保证总是最新的新功能分支。** 最后把更新的master分支重新push到远程库。

Git Flow工作流

Gitflow工作流没有用超出功能分支工作流的概念和命令，而是为不同的分支分配一个很明确的角色，并定义分支之间如何和什么时候进行交互。

除了有master主分支（用于存储正式发布的历史）外，还有一个作为功能集成分支的develop分支。当初始化完成后，某个程序员想要开发一个性能，并不是直接从master分支上拉出新分支，而是使用develop分支作为父分支，当新功能完成后，再合并会父分支，**新功能的提交并不与master分支直接交互。**

下面是Git Flow的流程图



上面的图你理解不了？ 没关系，这不是你的错，我觉得这张图本身有点问题，这张图应该左转90度，大家应该就很用以理解了。

Git Flow常用的分支

- Production 分支

也就是我们经常使用的Master分支，这个分支最近发布到生产环境的代码，最近发布的Release， 这个分支只能从其他分支合并，不能在这个分支直接修改

- Develop 分支

这个分支是我们是我们的主开发分支，包含所有要发布到下一个Release的代码，这个主要合并与其他分支，比如Feature分支

- Feature 分支

这个分支主要是用来开发一个新的功能，一旦开发完成，我们合并回Develop分支进入下一个Release

- Release分支

当你需要一个发布一个新Release的时候，我们基于Develop分支创建一个Release分支，完成Release后，我们合并到Master和Develop分支

- Hotfix分支

当我们在Production发现新的Bug时候，我们需要创建一个Hotfix, 完成Hotfix后，我们合并回Master和Develop分支，所以Hotfix的改动会进入下一个Release

Git Flow如何工作

初始分支

master主分支用于存储正式发布的历史，所有在Master分支上的Commit应该打上版本标签Tag

Feature 分支

分支名 feature/*

Feature分支做完后，必须合并回Develop分支, 合并完分支后一般会删点这个Feature分支，但是我们也可以保留

Release分支

分支名 release/*

Release分支基于Develop分支创建，打完Release分之后，我们可以在这个Release分支上测试，修改Bug等。同时，其它开发人员可以基于开发新的Feature (记住：一旦打了Release分支之后不要从Develop分支上合并新的改动到Release分支)

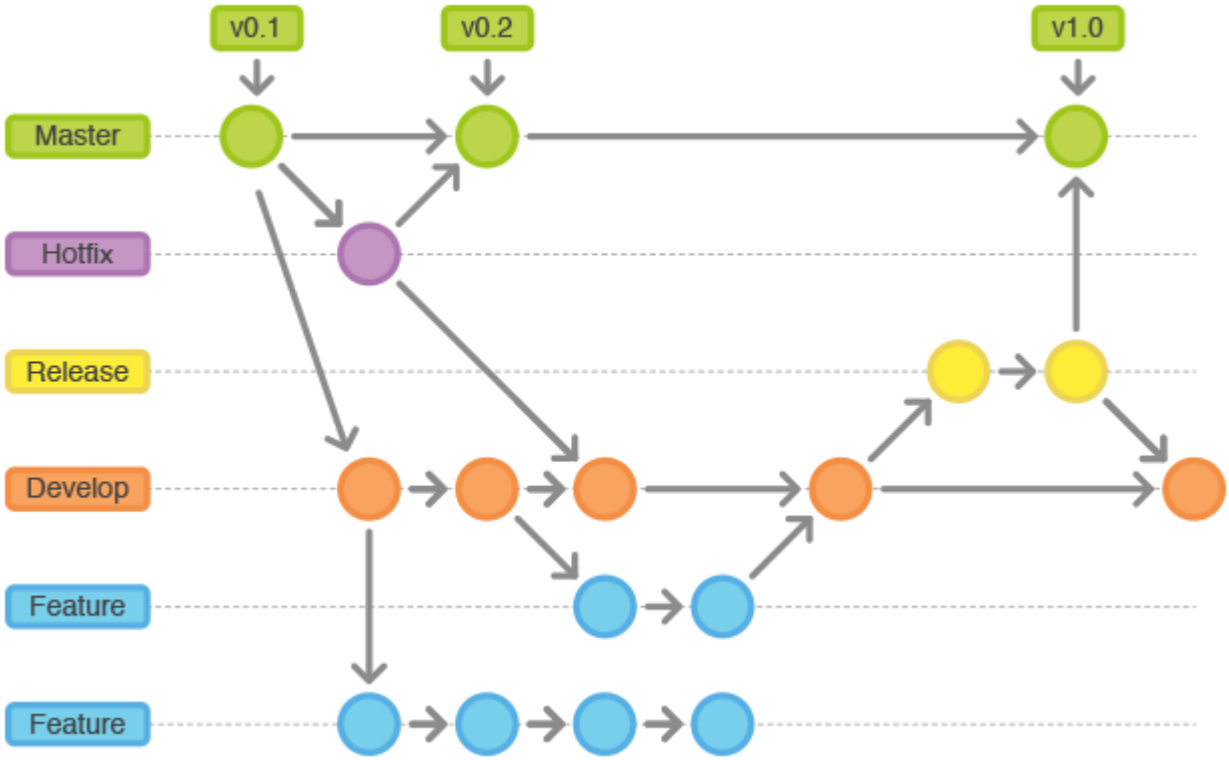
发布Release分支时，合并Release到Master和Develop， 同时在Master分支上打个Tag记住Release版本号，然后可以删除Release分支了。

维护分支 Hotfix

分支名 hotfix/*

维护分支或说是热修复（hotfix）分支用于生成快速给产品发布版本（production releases）打补丁，这是唯一可以直接从master分支fork出来的分支。 修复完成，修改应该马上合并回master分支和develop分支（当前的发布分支），master分支应该用新的版本号打好Tag。

为Bug修复使用专门分支，让团队可以处理掉问题而不用打断其它工作或是等待下一个发布循环。 你可以把维护分支想成是一个直接在master分支上处理的临时发布。



Git Flow代码示例

a. 创建develop分支

```
git branch develop
git push -u origin develop
```

b. 开始新Feature开发

```
git checkout -b some-feature develop
# Optionally, push branch to origin:
git push -u origin some-feature

# 做一些改动
git status
git add some-file
git commit
```

c. 完成Feature

```
git pull origin develop
git checkout develop
git merge --no-ff some-feature
git push origin develop

git branch -d some-feature

# If you pushed branch to origin:
git push origin --delete some-feature
```

d. 开始Relase

```
git checkout -b release-0.1.0 develop

# Optional: Bump version number, commit
# Prepare release, commit
```

e. 完成Release

```
git checkout master
git merge --no-ff release-0.1.0
git push

git checkout develop
git merge --no-ff release-0.1.0
git push

git branch -d release-0.1.0

# If you pushed branch to origin:
git push origin --delete release-0.1.0
```

git tag -a v0.1.0 master git push --tags

f. 开始Hotfix

```
git checkout -b hotfix-0.1.1 master
```

g. 完成Hotfix

```
git checkout master
git merge --no-ff hotfix-0.1.1
git push
```

git checkout develop git merge --no-ff hotfix-0.1.1 git push
git branch -d hotfix-0.1.1
git tag -a v0.1.1 master git push --tags

Git flow工具

实际上，当你理解了上面的流程后，你完全不用使用工具，但是实际上我们大部分人很多命令就是记不住呀，流程就是记不住呀，肿么办呢?

总有聪明的人创造好的工具给大家用, 那就是Git flow script.

安装

- OS X

brew install git-flow

- Linux

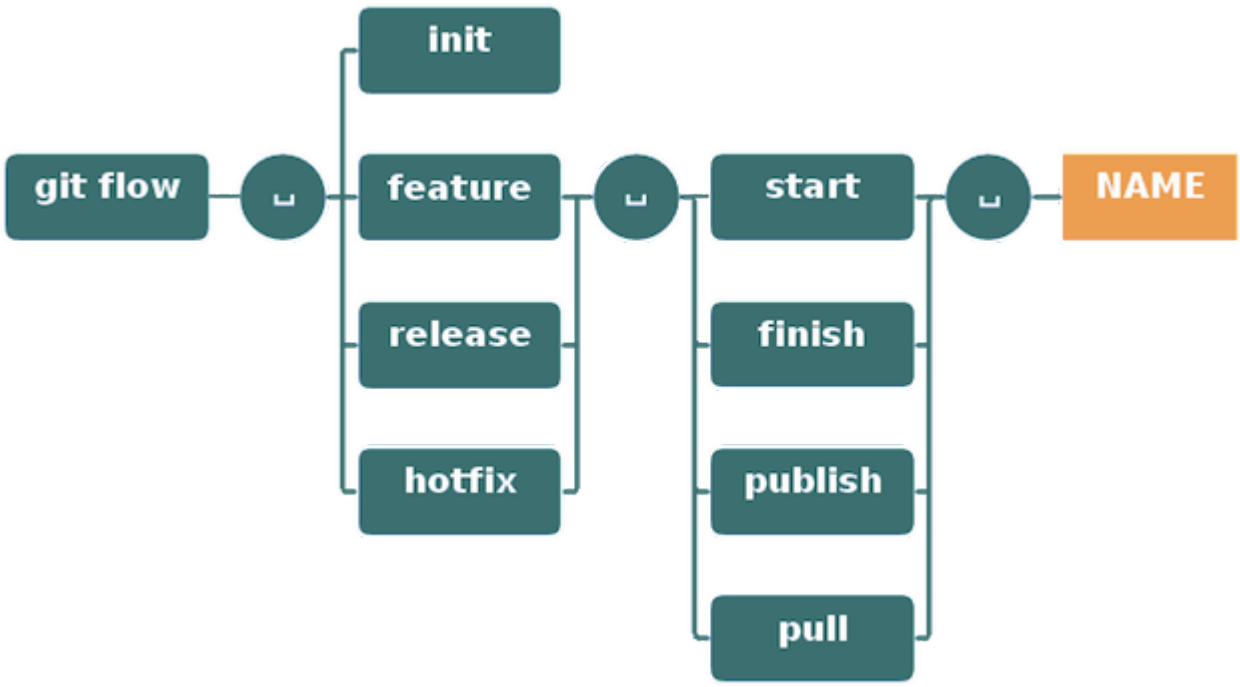
apt-get install git-flow

- Windows

wget -q -O - --no-check-certificate https://github.com/nvie/gitflow/raw/develop/contrib/gitflow-installer.sh | bash

使用

- **初始化:** git flow init
- **开始新Feature:** git flow feature start MYFEATURE
- **Publish一个Feature(也就是push到远程):** git flow feature publish MYFEATURE
- **获取Publish的Feature:** git flow feature pull origin MYFEATURE
- **完成一个Feature:** git flow feature finish MYFEATURE
- **开始一个Release:** git flow release start RELEASE [BASE]
- **Publish一个Release:** git flow release publish RELEASE
- **发布Release:** git flow release finish RELEASE
  别忘了git push --tags
- **开始一个Hotfix:** git flow hotfix start VERSION [BASENAME]
- **发布一个Hotfix:** git flow hotfix finish VERSION



Git Flow GUI

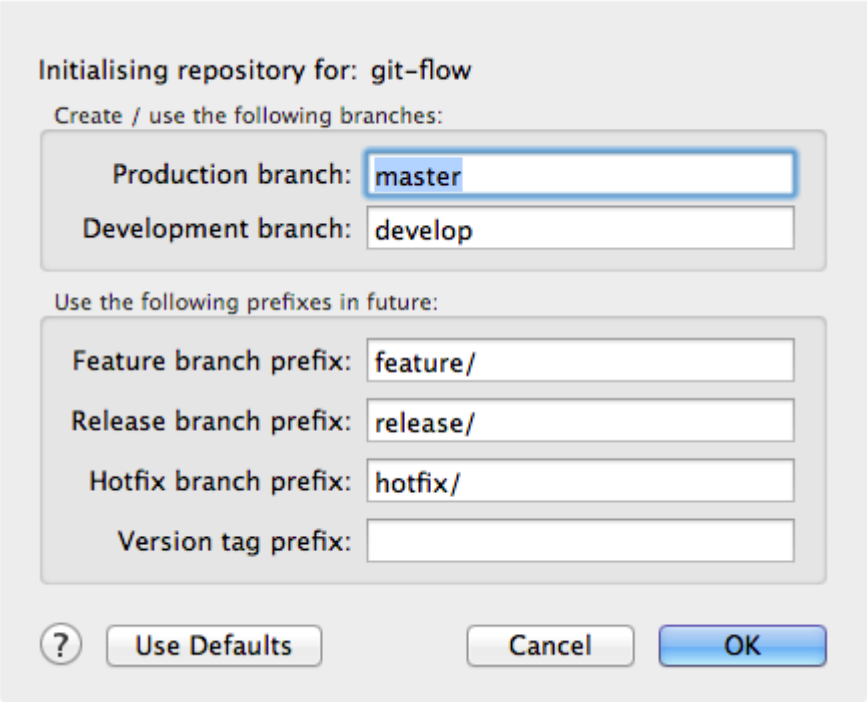
上面讲了这么多，我知道还有人记不住，那么又有人做出了GUI 工具，你只需要点击下一步就行，工具帮你干这些事！！！

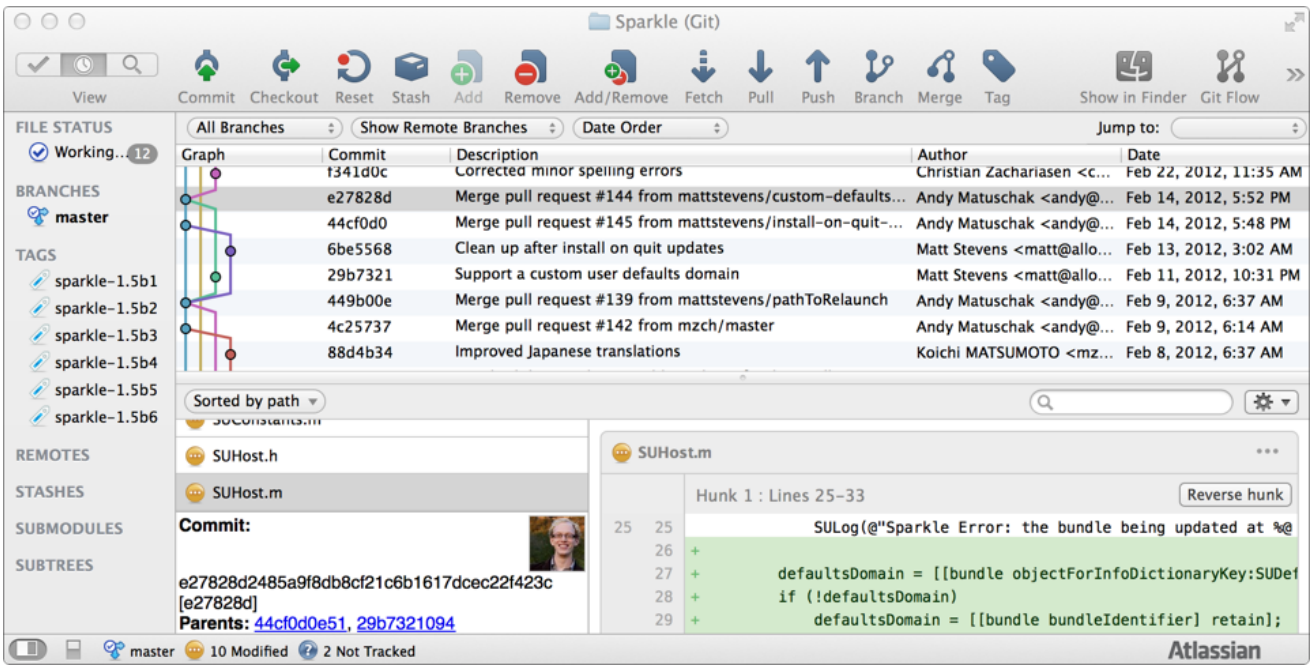
SourceTree

当你用Git-flow初始化后，基本上你只需要点击git flow菜单选择start feature, release或者hotfix, 做完后再次选择git flow菜单，点击Done Action. 我勒个去，我实在想不到还有比这更简单的了。

目前SourceTree支持Mac, Windows, Linux.

这么好的工具请问多少钱呢？ **免费!!!!**

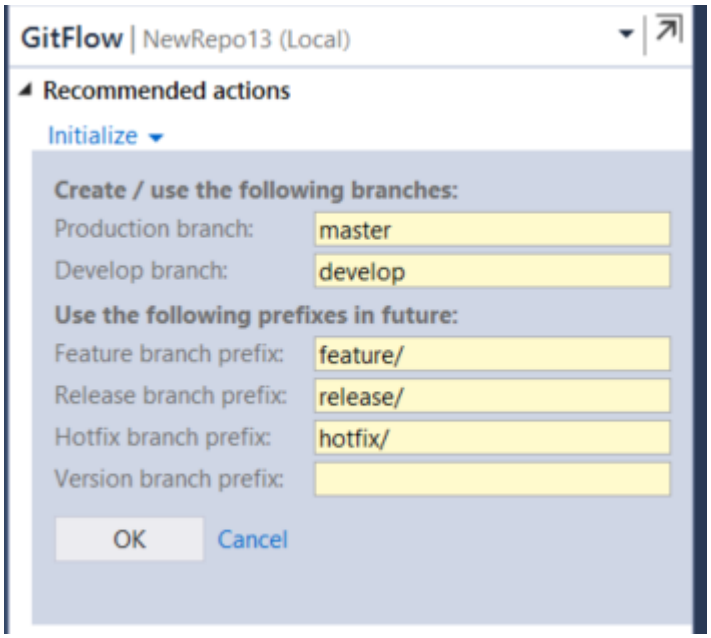


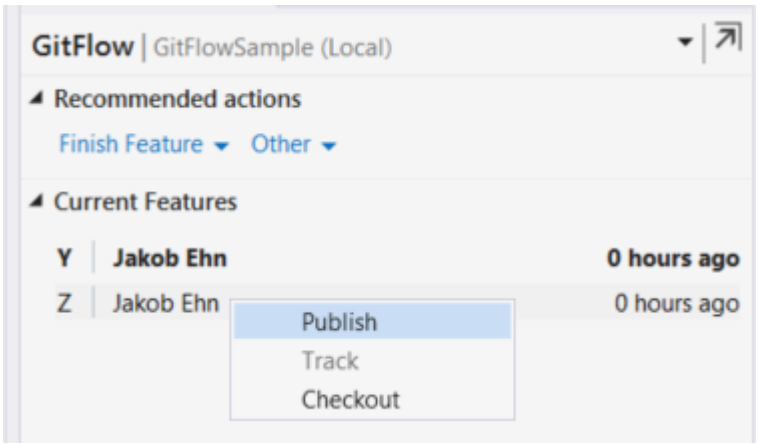


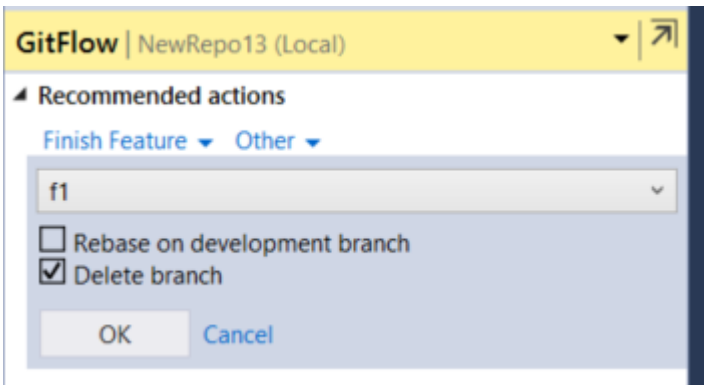
Git flow for visual studio

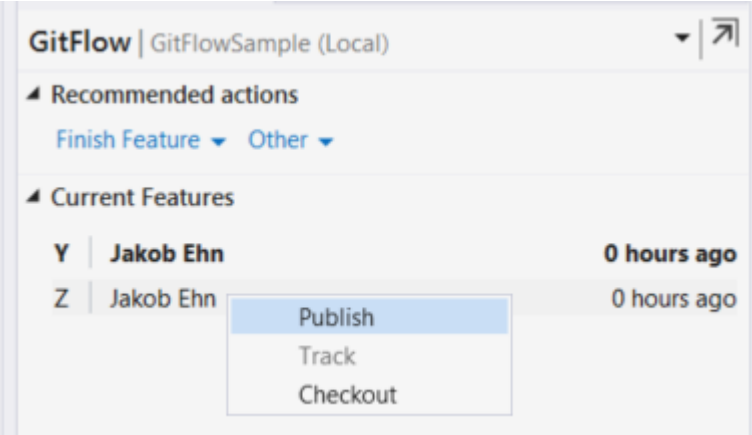
广大VS的福音

[GitFlow for Visual Studio]









Forking工作流

分布式工作流，充分利用了Git在分支和克隆上的优势，既可以管理大团队的开发者（developer）和接受不信任贡献者（contributor）的提交。**这种工作流使得每个开发者都有一个服务端仓库（此仓库只有自己可以push，但是所有人都可以pull修改）**，每个程序员都push代码到自己的服务端仓库，但不能push到正式仓库，只有项目维护者才能push到正式仓库，这样项目维护者可以接受任何开发者的提交，但无需给他正式代码库的写权限。

这种工作流适合网上开源社区的开源项目，大家统一对项目做贡献，但是有一个人或一个团队作为开发者来管理项目，所有的贡献者的代码由开发者审核，其功能完善之后再由开发者push到正式仓库中。