Git flow 实践手册

参与人员

姓名	联系方式
Elson	elson2010(at)126.com

发布记录

版本	日期	修改人	备注
1.0	2011-04-02	Elson	新建
1.1	2011-04-04	Elson	添加作者简介

作者简介

Elson，网名小肥鱼，花城原著居民，网络专业科班出身，英文水准估计有美国幼儿园小朋友三成功力，读书时候喜欢参加比赛，没拿过国家级奖项；毕业多年一直在 IT 行业打滚，曾做过 Oracle ERP 功能顾问，对企业管理有一定认识；也尝试过带小团队从事外包开发；擅长 php，是 Zend 认证 php 工程师，另外对 Java，python 和 ruby 略有认识。不是技术狂人，不刻意追求系统运行性能。由于性格上喜欢偷懒，所以总希望能用最少的代码实现最多的功能；喜欢事情有条理，所以想尽办法避免一切可能导致情况失控的因素出现；喜欢电子商务，觉得在网上做买卖很 cool，很有成就感。现供职于国内某高速成长的电子商务公司，主要从事网站系统编码工作。

一， 概述

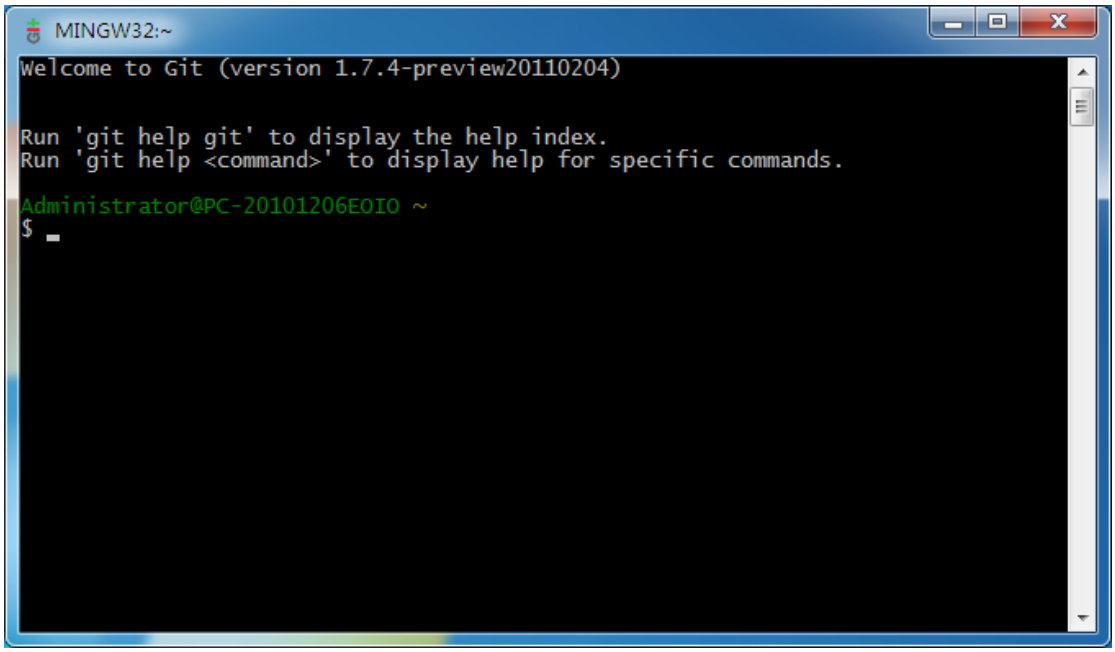
Git 是一个分布式版本控制软件，由 Linux 的发明人 Linus 开发并维护，其开发的目的是为了方便成百上千的 Linux 内核开发人员协同开发。

Git flow 是一套根据 Vincent Driessen 总结的 Git 最佳实践方法而编写的 Git 指令快捷命令。它主旨是方便开发人员更容易地使用 Git 进行版本控制。

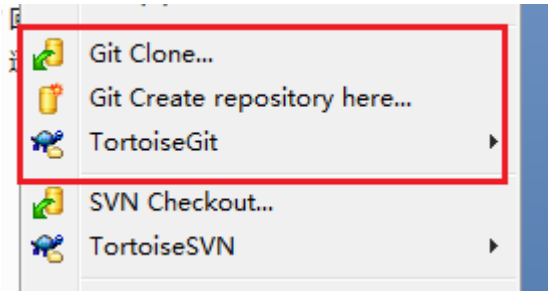
二， 安装

这里讲解的是如何在 Windows 上安装 Git Flow。所要用到的软件都在附带的附件包里面，其中包涵以下四个文件。

首先安装 Git-1.7.4-preview20110204.exe，然后分别把 bin.zip 文件和 git-core.zip 文件解压到“X:\Program Files\Git\bin”目录和“X:\Program Files\Git\libexec\git-core”下。这样子就已经把 git flow 安装到 windows 上了。



假如以前有使用 TortoiseSVN 的开发人员刚开始不习惯 Git 的使用，可以安装 Tortoisegit-1.6.5.0-32bit.exe。这是一个类似 TortoiseSVN 的工具。可以帮助 SVN 用户过渡到 Git 上。安装完成后能在右键菜单上看到熟悉的命令。



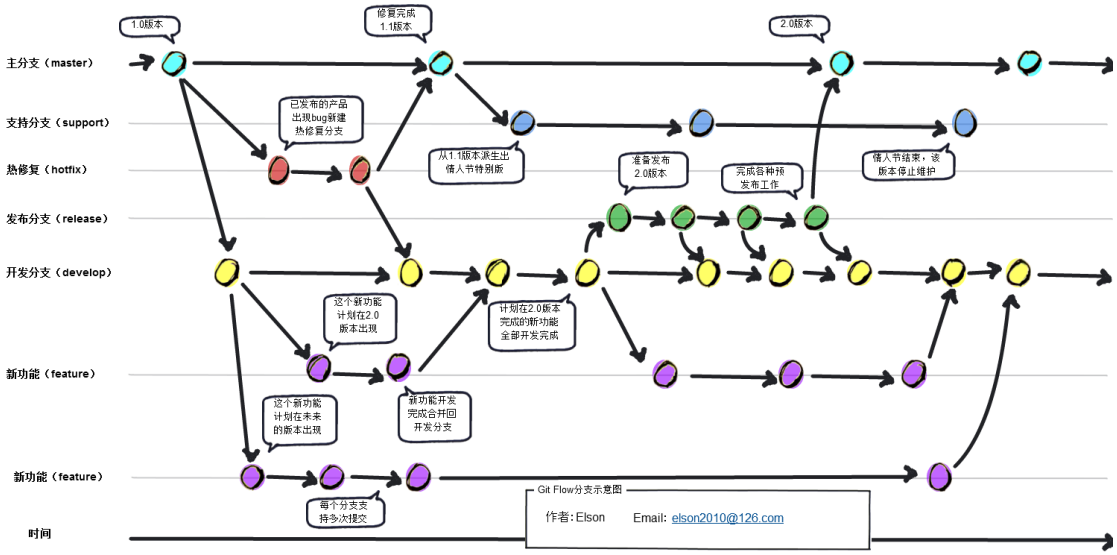
Git 是分布式版本控制软件，并没有严格意义上的客户端和服务器端之分，所以其实安装了 Git 的同时，可以理解成既安装了客户端，也安装了服务器端。读者可以对比 TortoiseSVN 和 TortoiseGit 的菜单就能发现，TortoiseGit 比 TortoiseSVN 多了一个 Git Create Repository here...这个菜单，这个菜单的作用就是在当前文件夹创建版本库，因为一般来说 TortoiseSVN 的版本库是在服务器端创建好，客户端只能是从服务器端 checkout 到本地的。而 TortoiseGit 则是在一个文件夹创建版本库，然后通过 clone 把当前版本库“克隆”到别的文件夹，这就是分布式版本控制软件的一大特点。

三， 分支

对于 svn 等集中式版本控制软件，一般是客户端从服务器 update 最新代码，然后修改或编写新功能后再 commit 到服务器。当开发人员正在试图增强一个模块，工作做到一半，由于会改变原模块的行为导致代码服务器上许多测试的失败，所以并没有提交代码。这时候上级说，现在有一个很紧急的 Bug 需要处理， 必须在短时间内完成。那结果就是开发人员八仙过来，各显神通了，有的人会用 diff 命令，把修改过的代码输出成一个 patch，然后回滚到原来的代码，修补好那个紧急 bug 并提交后再把 patch 应用回去，前前后后非常繁琐。有的人不会使用 diff 命令，就直接把修改过的文件复制一份，再回滚。可以说无论什么方式，都没有 Git 那么方便和科学。

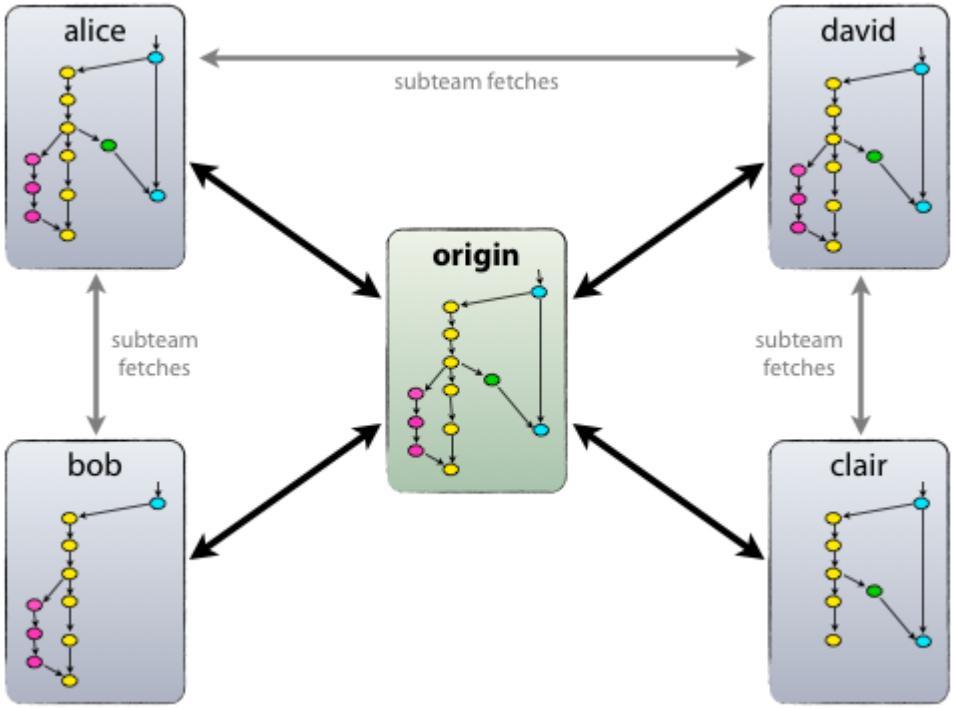
对于 Git 版本控制，最显著的另一个特点就是鼓励每个改动都先创建一个分支，完成改动后再把分支合并回主分支里面。这样子做的话就可以方便用户同时维护多个修改而不互相干扰，同时每个分支又能实现独立的版本控制。

下面这幅图是 Git flow 的分支示意图，要看更大的图请查看附件。



四， 分布式集中管理

Git 是一个分布式版本管理软件，但是并不是说不能集中式管理。从下图可以看到，origin 版本库就是整个开发的核心版本库，其它版本库通过 pull 和 push 与 origin 版本库同步。其它本地版本库不仅能和 origin 版本库同步，也可以互相同步代码，做到多人共同开发同一个新功能，最后再把新功能 push 到 origin 版本库里面。



五， 长期分支

Git flow 这种模式主要维护着两个长期分支，一个是开发分支（develop），另一个是主分支（master），主分支永远放的是可以稳定发布的产品。

六， 开发新功能（new feature）

这是许多项目的主要工作之一。每个新功能的开发，都要先创建一个 feature 分支，完成后再合并到 develop 分支上，假如这个新功能开发失败了，可以直接丢弃而不影响 develop 分支上的代码。

在这个分支上，开发人员主要是开发新功能并进行单元测试（unit test），保证模块和功能的实现。

当开发完成后，在合并到 develop 分支时，开发人员还要进行集成测试，也就是保证合并之后的系统不会出现兼容性问题。

七， 预备发布（release）

当一个开发周期内的所有新功能（feature）已经开发完毕后就能从开发（develop）分支创建一个发布（release）分支。在发布分支上我们不再开发新功能，而是主要进行用户接受度测试（UAT），并作简单的调整。

在测试人员对发布分支进行用户接受度测试的同时，开发人员可以继续对（develop）分支进行新功能的开发，互不干扰。

当所有 UAT 都完成后，就能把 release 分支合并到 master 分支上，系统运维的人员就能从 master 分支上拿到最新版本的产品并进行部署了。

八， 热修复（hotfix）

再多的测试也不能保证发布的产品完全没有任何 bug，假如在 master 分支运行期间出现了 bug，就要在 master 分支上创建一个 hotfix 分支，修复这个 bug，然后合并到 master 分支上，热修复（hotfix）就像是微软不定期发布的补丁一样。其作用就是修复 master 分支上出现的 bug。

九， 支持（support）

这是一个不是所有开发都会用到的分支，有这么一个情形会用得到，你有一个产品已经开发到 2.0 版本了，但是一个非常重要的客户还在使用 1.0 版本的产品，而同时他又不想升级到 2.0 版本，那么你就得为这个客户额外地增加一个从 1.0 版本的 master 分支分出来的 support 分支，一直为这个客户维护他的产品，直到后续维护的合同到期。

还有另一种情形，读者假如玩过《愤怒的小鸡》就会知道，它除了原版，还会不定期出情人节版，万圣节版，这些各种衍生版本只会在特定时期存在。这些版本就可以使用 support 分支来进行开发。

十， Git 小结

分支	类型	用途	分支建立	分支合并	位置
Master	长期分支	稳定版本	项目初始		本地，远程
develop	长期分支	开发版本	项目初始		本地，远程
feature	短期分支	从 develop 分支建立,可以指定起始点	开发新功能	新功能开发完成	本地，远程（可以通过 feature publish 发布到远程上,供团队协作）
release	短期分支	从 develop 分支建立,可以指定起始点	预备发布	合并回 develop 和 master 分支	本地，远程(可以通过 release publish 发布到远程上,供团队协作)
hotfix	短期分支	紧急 Bug 修复	从 master 建立分支	合并回 master 和 develop	本地
support	短期分支	特殊版本	从 master 建立分支	一般不合并	本地

十一， 一个例子

这是以开发一个博客为例子的视频，详情请看附件。

十二， 关于版本号的问题

每个产品都会有一个版本号，下面是对版本号的一般定义

名称	例子	说明
主版本号	2	主版本号一般是一个比较长的时期才改变一次，例如半年或一个季度。
发布号	12	发布号是每个开发周期结束时候改变一次的，假如开发周期是一周，那就是一周改变一次，假如发布周期是一个月，则是一个月改变一次。
补丁号	3	对于完成了一定数量的热修复后，就会改变一次。
特殊版本	情人节版	一个支持性版本。

十三， 总结

Git 是一个比 svn 年轻的版本管理软件，其强大的功能更是大有长江后浪推前浪之势，很多 IDE 都集成了 Git，Git flow 的出现更是加速了这种趋势。但是在某写程度上 Git 也表现了其不成熟的一面，那就是对 windows 支持不足，这可能是开发 git 的初衷是管理 Linux 内核源码，考虑到大部分 Linux 内核开发人员都习惯命令行方式的开发模式，所以 git 大部分功能现在还是主要通过敲命令实现，这对于习惯 TortoiseSVN 的用户来说，这或许是一个不小的障碍。

十四， 子命令

feature

用法:

git flow feature [list] [-v]

git flow feature start [-F] <name> [<base>]

git flow feature finish [-rFk] <name|nameprefix>

git flow feature publish <name>

git flow feature track <name>

git flow feature diff [<name|nameprefix>]

git flow feature rebase [-i] [<name|nameprefix>]

git flow feature checkout [<name|nameprefix>]

git flow feature pull <remote> [<name>]

feature start

用法: git flow feature start [-F] <name> [<base>]

功能: 以指定的 commit 名称(由 base 参数指定)创建一个 feature 分支

参数: -F

'fetch from origin before performing local operation' (建立分支前先从 origin 下载数据) 默认为 false

该参数在 0.4 版本有 bug,不可用,可以使用 git fetch -q orgin develop 代替

name

feature 的名称,对应的分支名称为 feature/name

base

建立 feature 的 start point,默认为 develop 分支

等价的 Git 命令:

1. git fetch -q origin develop //当 -F 设置的时候执行,只是更新.git 中的 remote 内容,不做 merge 操作

2. 检查本地的 develop 分支和远程的 develop 分支是否一致,即 refs/develop 和 refs/remote/origin/develop(建议先执行 git pull origin develop 或者 git fetch origin develop)

3. git checkout -b feature/name develop //建立分支

例子:

git flow feature start story_1

feature publish

用法: git flow feature publish <name>

功能: 将一个本地的 feature 分支 push 到远程的仓库中,该命令可用于与团队其他成员合作开发或者备份自己的代码

参数: name

本地 feature 的名称

等价的 Git 命令:

1. 检查本地的工作目录及分支

2. git fetch -q origin

3. git push origin feature///name://refs/heads/feature///name//

4. git fetch -q origin

5. git checkout feature/name 例子:

git feature publish story_1

feature track

用法: git flow feature track <name>

功能: 将由 feature publish 发布的 feature 分支从远程仓库下载到本地,并建立同名分支

参数: name

远程 feature 的名称,对应 feature publish 的名称

等价的 Git 命令:

1. 检查本地的工作目录是否"干净";检查分支是否已经存在,如果已经存在,则报错退出

2. git fetch -q origin

3. git checkout -b name origin/feature/name 例子:

git feature track name

feature pull

用法: git flow feature pull <remote> [<name>]

功能: 将由 feature publish 发布的 feature 分支从远程仓库下载到本地,并建立同名分支;如果本地已经有同名分支,则对其执行 pull 操作

参数: name

远程 feature 的名称,对应 feature publish 的名称

等价的 Git 命令:

a. 如果本地已有 name 分支

1. git pull -q origin feature/name

b. 如果本地没有 name 分支

1. git fetch -q origin feature/name
2. git branch --no-track feature/name
3. git checkout -q feature/name 例子:

git flow feature pull origin story_1

feature finish

用法: git flow feature finish [-rFk] <name|nameprefix>

功能: 完成由 name 指定的 feature 分支的开发,将其合并到本地的 develop 分支,合并成功后删除该分支

参数: -r

在合并到 develop 分子时,使用 rebase 机制,而不是 merge

-F

在执行 finish 操作前,先执行 fetch,从远程仓库下载更新

-k

执行完 finsh 后,保留 feature 分支,即不删除分支

name

feature 的名称,对应 feature start 的名称

等价的 Git 命令:

1. git fetch -q origin feature/name #当参数中设置了-F 时

2. git flow feature rebase name develop #当参数中设置了-r 时

3. git checkout develop

4. git merge feature/name #根据 develop 分支和 feature/name 分支之间的提交的个数决定是否设置--no--ff

5. #如果设置了-F 参数,则删除远程的分支 git push origin :ref/heads/feature/name

6. #如果没有设置-k 参数,则删除本地的分支

例子:

git flow feature finish story_1

feature rebase

用法: git flow feature rebase [-i] [<name|nameprefix>]

功能: 以 develop 分支作为 upstream,对指定的 feature 分支执行 rebase 操作

参数: -i

等价与 rebase -i

name

feature 的名称

等价的 Git 命令:

1. git checkout -q feature/name
2. git rebase develop

例子:

git flow feature rebase -i story_1

release

用法:

git flow release [list] [-v]

git flow release start [-F] <version>

git flow release finish [-Fsumpk] <version>

git flow release publish <name>

git flow release track <name>

release start

用法: git flow release start [-F] <version> [<base>]

功能: 从 develop 分支指定的起始点(可选,默认为 HEAD)建立版本发布的分支

参数: version

版本号

base

建立分支的起始点,可选参数,默认为 develop HEAD

等价的 Git 命令:

1. git checkout -b release/version develop

例子:

git flow release start v0.1

release publish

用法: git flow release publish <name>

功能: 将 release 分支发布的远程仓库,供团队协作

参数: name

release 名称,与 start 中的 version 相对应

等价的 Git 命令:

1. git fetch -q origin

2. git push origin release/name:refs/heads/release/name 例子:

git flow release publish v0.1

release track

用法: git flow release track <name>

功能: 将由 publish 发布的 feature 分支从远程仓库下载到本地,并建立同名分支,供团队协作

参数: name

远程 feature 的名称,对应 feature publish 的名称

等价的 Git 命令:

1. git fetch -q origin

2. git checkout -b release/name origin/release/name 例子:

git flow release track v0.1

release finish

用法: git flow release finish [-Fsumpk] <version>

功能: 完成由 version 指定的 release 分支的开发,将其合并到 develop 和 master 分支,并为该分支创建一个 tag

参数: -F

执行操作前先执行 fetch

-s

对新建的 tag 签名

-u

签名使用的 GPG-key

-m

使用指定的注释作为 tag 的注释

-p

当操作结束后,push 到远程仓库中

-k

保留分支

-n

不创建 tag

version

版本号

等价的 Git 命令:

1. 如果设置了 -F 参数,下载更新

1.1 git fetch -q origin master

1.2 git fetch -q origin develop

2. 合并回 master 分支

2.1 git checkout master

2.2 git merge --no-ff release/version

3. 如果没有设置 -n 参数,创建 tag

3.1 git tag

4. 合并回 develop 分支

4.1 git checkout develop

4.2 git merge --no-ff release/version

5. 如果没有设置 -k 参数

5.1 git branch -d release/version

6. 如果设置 -p 参数

6.1 git push origin develop

6.2 git push origin master

6.3 git push --tags origin

6.4 git push orign :release/version #删除远程仓库中的 release 分支 例子:

git flow release finish -p v0.1

十五， 参考文档

http://nvie.com/posts/a-successful-git-branching-model/

http://yakiloo.com/getting-started-git-flow/

http://agilejava.blogbus.com/logs/103552611.html