Git 基础培训

2014/12/19	文档做成	Kevin Tang
2014/12/22	Ch11: Update Git push	Kevin Tang
2014/12/23	Ch11: Add Retrigger & Ch18: Add Gerrit Error Ref	Kevin Tang
2014/12/25	Ch13: Add Git rebase	Kevin Tang

1. Git 简介

Git 是分布式版本控制软件。

关于什么是 Git，CVS、SVN 和 Git 的关系、区别百度、谷歌都有。

Git 的书也有两三英寸那么厚，Linus Torvalds 的 Linux Kernel 和 Git 的故事也不是一时半会说的完的，略。

所以 Git 肯定也不是简单的东西，以下只做普及，更多的还是需要大家自学。

2. Git help & Git help <command>

学会使用帮助命令

git help 列出了常用的命令帮助

最后一行写了# git help <command> 可以获得命令的详细信息。

3. Git init & Git clone

git init 初始化一个空的 Git 仓库，后续我们可以在此进行 Git 练习。

```
tangwenkai@tangwenkai:~$ cd /opt/temp/
tangwenkai@tangwenkai:temp$ mkdir git-local
tangwenkai@tangwenkai:temp$ cd git-local/
tangwenkai@tangwenkai:git-local$ ls -al
总用量 8
drwxrwxr-x  2 tangwenkai tangwenkai 4096 12月  19 17:36 .
drwxrwxr-x 60 tangwenkai tangwenkai 4096 12月  19 17:36 ..
tangwenkai@tangwenkai:git-local$ git init
初始化空的 Git 版本库于 /opt/temp/git-local/.git/
tangwenkai@tangwenkai:git-local$ tree -a
.
└── .git
    ├── branches
    ├── config
    ├── description
    ├── HEAD
    ├── hooks
    │   ├── applypatch-msg.sample
    │   ├── commit-msg.sample
    │   ├── post-update.sample
    │   ├── pre-applypatch.sample
    │   ├── pre-commit.sample
    │   ├── prepare-commit-msg.sample
    │   ├── pre-rebase.sample
    │   └── update.sample
    ├── info
    │   └── exclude
    ├── objects
    │   ├── info
    │   └── pack
    └── refs
        ├── heads
        └── tags

10 directories, 12 files
```

* 不要在主文件夹（Home 目录）下直接做此操作

* 以上操作# mkdir GIT && cd GIT && git init 相当于 # git init --bare GIT

Git clone 拉取本地代码

刚刚只是创建了一个空的仓库（相当于 Git 的数据库，并非在此操作 Git）

git clone PATH_TO_YOUR_GIT_INIT
cd YOUR_CLONE_DIR
do what you want git do

Git clone 拉取远程仓库

在项目中我们使用 repo 管理，其实 repo 是 Git 的管理者。

Repo 是 python 写的，通过读取服务器上的 xml 配置文件（配置着若干的仓库），然后调用 Git 进行拉取/更新代码等操作。

比如拉取 WizNote 的源码：

```
# cd YOUR_WORK_PATH
# git clone https://github.com/WizTeam/WizQTClient.git
```

```
tangwenkai@tangwenkai:WizTemp$ git clone https://github.com/WizTeam/WizQTClient.git
正克隆到 'WizQTClient'...
remote: Counting objects: 16467, done.
remote: Compressing objects: 100% (96/96), done.
remote: Total 16467 (delta 39), reused 0 (delta 0)
接收对象中：100% (16467/16467), 37.84 MiB | 7.86 MiB/s, done.
处理 delta 中：100% (10599/10599), done.
tangwenkai@tangwenkai:WizTemp$
```

4. Git pull & Git fetch

功能都是拉取代码

区别是 git pull = git fetch + git merge

在日常中，我们经常用 git pull，而几乎不会用到 git fetch，了解下就行。

5. Git add & Git mv & Git rm

基础命令：

git add：添加文件到 Git 仓库

git mv：重命名文件

git rm：从 Git 仓库删除受控的文件

6. Git status & Git log

基础命令：

git status：查看当前仓库的状态

git log：查看历史 Log

7. Git diff

重要命令：查看差异

在环境搭建中，刘全已经明确指出如何使用 Beyond Compare 进行代码比较。

在提交代码前，比较代码是一个重要环节。

在此，我再贴下我的 Git 配置文件：

```
tangwenkai@tangwenkai:~$ cat ~/.gitconfig
[user]
        name = tangwenkai
        email = tangwenkai@pset.suntec.net
[core]
        autocrlf = input
        safecrlf = true
[color]
        ui = auto
[diff]
        external = ~/bin/git-diff.sh
[merge]
        tool = bc3
[mergetool "bc3"]
        cmd = /usr/bin/bcompare $LOCAL $REMOTE $BASE $MERGED
        trustExitCode = true
[push]
        default = simple
[alias]
        st = status
        co = checkout
        ci = commit
```

比较脚本：

```
tangwenkai@tangwenkai:~$ cat ~/bin/git-diff.sh
#!/bin/bash
# diff is called by git with 7 parameters:
# path old-file old-hex old-mode new-file new-hex new-mode

/usr/bin/bcompare $2 $5

exit 0
```

* 也可以参考百度、Google 按照自己的习惯配置

8. Git commit

提交代码到本地仓库

一般用法： # git commit -a -m “YOUR COMIIT LOG”

例如：

```
tangwenkai@tangwenkai:git-local$ git add file3 file4
tangwenkai@tangwenkai:git-local$ git status
# 位于分支 master
#
# 初始提交
#
# 要提交的变更：
#   （使用 "git rm --cached <file>..." 撤出暂存区）
#
#       新文件：     file3
#       新文件：     file4
#
# Untracked files:
#   （使用 "git add <file>..." 以包含要提交的内容）
#
#       file1
#       file2
#       file5
tangwenkai@tangwenkai:git-local$ git commit -a -m "add file3 file4"
[master （根提交） 1a0bfac] add file3 file4
 2 files changed, 2 insertions(+)
 create mode 100644 file3
 create mode 100644 file4
```

如果不小心写错注释了？修改已提交的注释：

git commit --amend

9. Git branch

重要命令：分支操作

创建分支：

```
# git branch new_dev_branch
```

删除分支（*删除分支，项目慎用，本地练习随意）：

```
# git branch -d new_dev_branch
# git branch -D new_dev_branch
```

查看分支

```
# git branch -a （查看所有分支）
```

```
# git branch -r （查看远程分支）
```

10. Git checkout

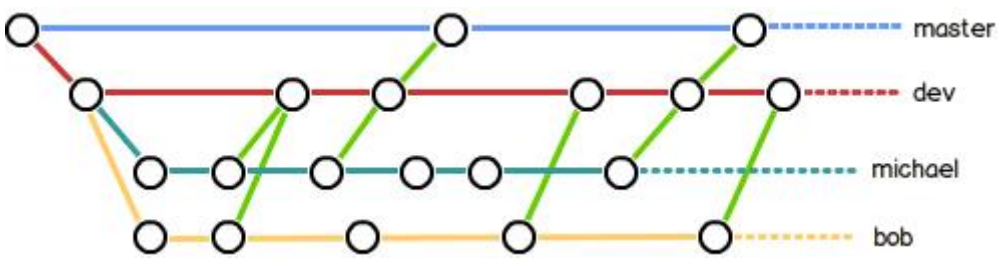
Checkout 文件：

如果不小心修改了文件，在没有 commit 的时候可以通过 checkout 回退。

Checkout 分支：

切换工作分支。

团队合作的分支看起来就像这样：



11. Git push

基础命令：

推送代码到远端仓库（commit 是更新本地代码，push 到服务器，团队的其他人才能通过 git pull 拉取）

简单示例：

```
# git push
```

其实以上命令代表： `# git push origin master`

在 17CY 项目中由于有 gerrit 用作代码 review，所以我们使用的是：

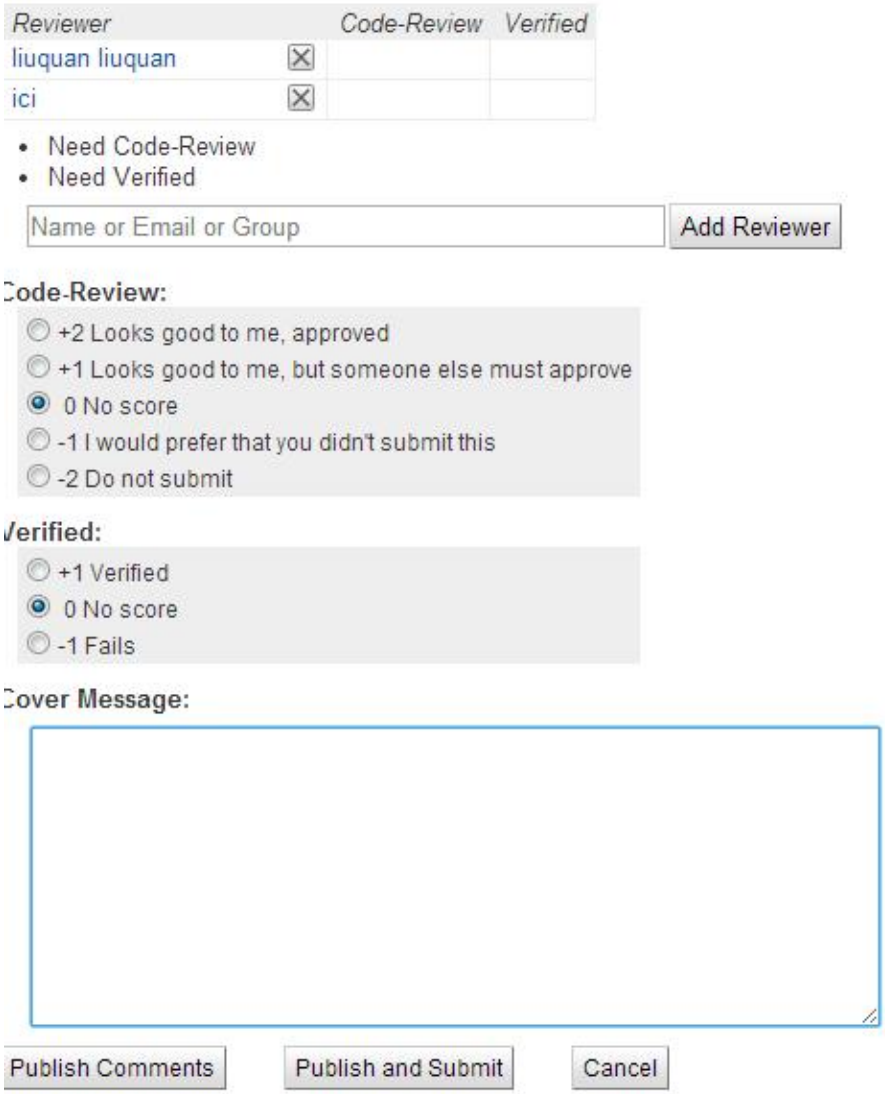
```
# git push origin master:refs/for/master
```

```
liuquan@liuquanPC:~/Proj/17Cy/packages/QmlApps/QmlDemos/UICC$ git status
位于分支 master
要提交的变更：
  （使用 "git reset HEAD <file>..." 撤出暂存区）

        修改:         src/UI/View/Demo.cpp

未跟踪的文件:
  （使用 "git add <file>..." 以包含要提交的内容）

        ../build-UICC-QT5_3_1_emulator-Release/

liuquan@liuquanPC:~/Proj/17Cy/packages/QmlApps/QmlDemos/UICC$ git commit -m "test"
[master ba65399] test
 1 file changed, 1 insertion(+)
liuquan@liuquanPC:~/Proj/17Cy/packages/QmlApps/QmlDemos/UICC$ git push origin master:refs/for/master
Counting objects: 26, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 644 bytes | 0 bytes/s, done.
Total 7 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4)
remote:
remote: Processing changes: new: 1, refs: 1, done
remote:
remote: New Changes:
remote:   http://igerrit/30457
remote:
To ssh://igerrit.storm:29418/Src/17Model/17Cy/packages/QmlApps/QmlDemos
 * [new branch]      master -> refs/for/master
liuquan@liuquanPC:~/Proj/17Cy/packages/QmlApps/QmlDemos/UICC$ gitk
```

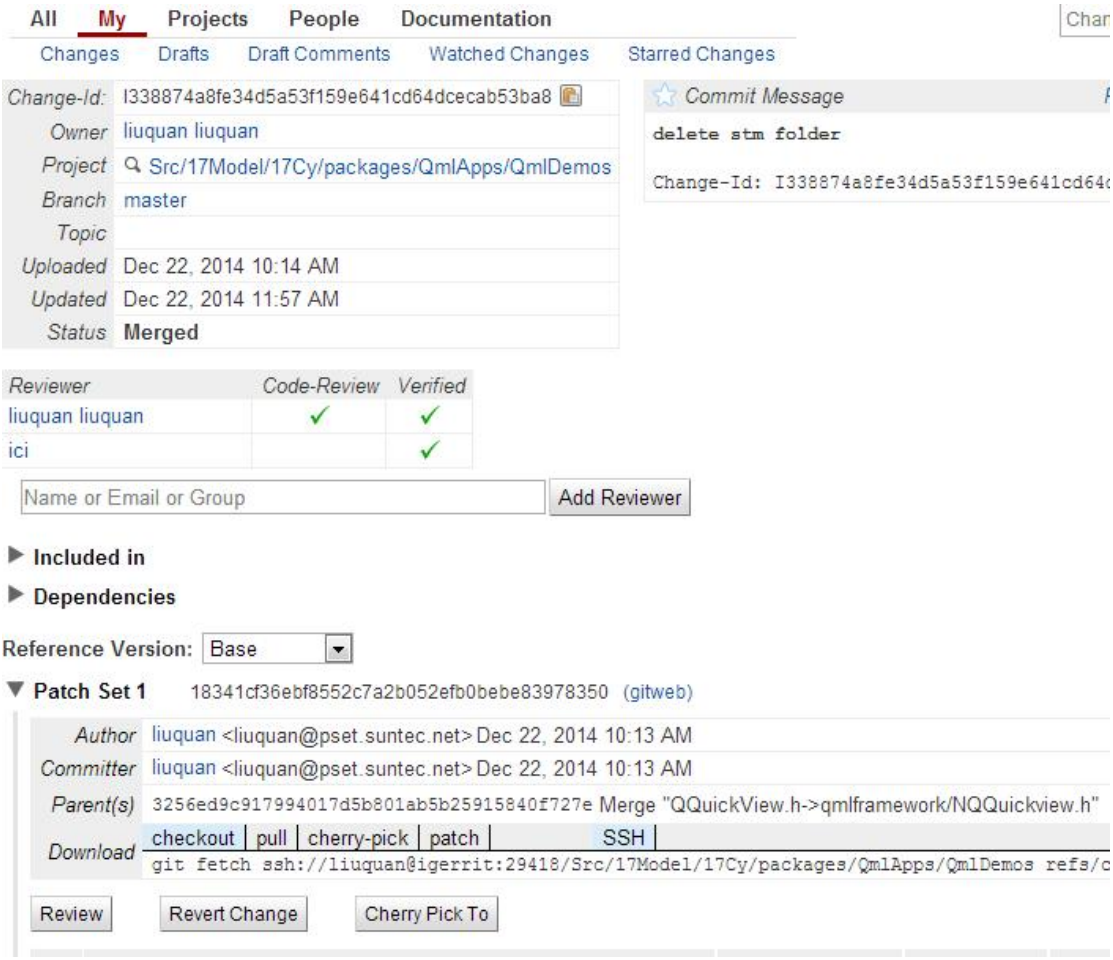
为了保证代码的质量和正常的编译，17CY 引入了静态检查、编译检查、Code Review 等机制保证每次提交的可靠性。所以至此代码并未真正提交，还需要通过 ICI 的编译检查和相关人员进行代码 Review。

通过访问 http://igerrit/30457 （每次提交成功都会生成一个页面）

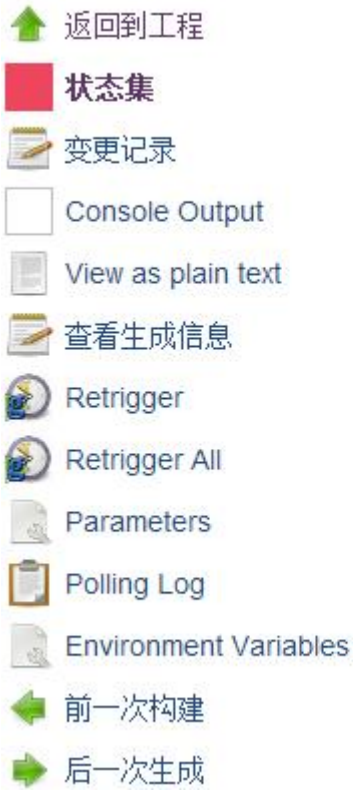
下图是 Gerrit 界面



正常情况如下：


All My Projects People Documentation
Changes Drafts Draft Comments Watched Changes Starred Changes
Change-Id: I338874a8fe34d5a53f159e641cd64dcecab53ba8
Owner liuquan liuquan
Project Src/17Model/17Cy/packages/QmlApps/QmlDemos
Branch master
Topic
Uploaded Dec 22, 2014 10:14 AM
Updated Dec 22, 2014 11:57 AM
Status Merged
Commit Message
delete stm folder
Change-Id: I338874a8fe34d5a53f159e641cd64
Reviewer Code-Review Verified
liuquan liuquan
ici
Name or Email or Group
Add Reviewer
Included in
Dependencies
Reference Version: Base
Patch Set 1 18341cf36ebf8552c7a2b052efb0bebe83978350 (gitweb)
Author liuquan <liuquan@pset.suntec.net> Dec 22, 2014 10:13 AM
Committer liuquan <liuquan@pset.suntec.net> Dec 22, 2014 10:13 AM
Parent(s) 3256ed9c917994017d5b801ab5b25915840f727e Merge "QQuickView.h->qmlframework/NQQuickview.h"
Download checkout | pull | cherry-pick | patch | SSH |
git fetch ssh://liuquan@igerrit:29418/Src/17Model/17Cy/packages/QmlApps/QmlDemos refs/c
Review
Revert Change
Cherry Pick To

如果 ICI 编译出错：



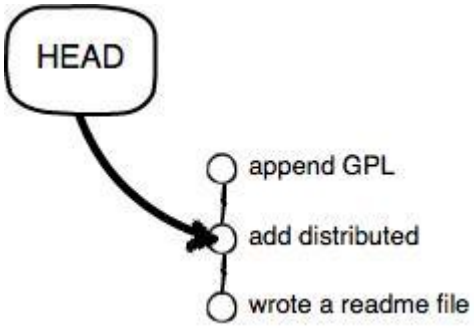
请点击 Console Output 查看编译 Log，如果不是自己的问题，请点击 Retrigger 重新出触发下编译，也可以邮件 CI 组抄送 Leader 和相关人员。

12. Git reset & Git reflog

后悔药，如果改错代码了之类：

- 未 push 到服务器：

git reset --hard HEAD^ （回退一个版本，提交代码被打回也需要回退一个版本）

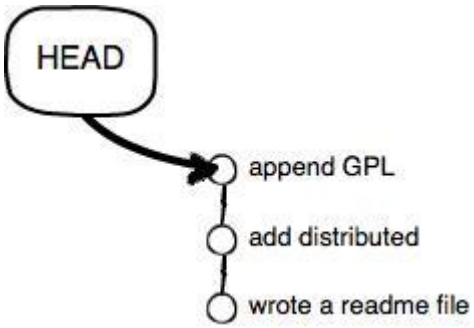


现在，你回退到了某个版本，关掉了电脑，第二天早上就后悔了，想恢复到新版

本怎么办？找不到新版本的 commit id 怎么办？

```
$ git reflog
ea34578 HEAD@{0}: reset: moving to HEAD^
3628164 HEAD@{1}: commit: append GPL
ea34578 HEAD@{2}: commit: add distributed
cb926e7 HEAD@{3}: commit (initial): wrote a readme file
```

git reset --hard 3628164 （回退到某个版本）



Git 的版本回退速度非常快，因为 Git 在内部有个指向当前版本的 HEAD 指针，当你回退版本的时候，Git 仅仅是移动 HEAD 指针。

- 已 push 到服务器：

不推荐以上方法，还是老老实实通过 git diff 比较差异，然后在现有的代码基础上进行修改。当然如果你十分自信和有把握，一切都不是问题。

13. Git merge & Git rebase

在协作开发过程中是需要尽量避免 Merge 的，但也是不可避免的，人生不如意之事十之八九，合并分支往往也不是一帆风顺的。

准备新的 feature1 分支，继续我们的新分支开发：

```
$ git checkout -b feature1
Switched to a new branch 'feature1'
```

修改 readme.txt 最后一行，改为：

```
Creating a new branch is quick AND simple.
```

在 feature1 分支上提交：

```
$ git add readme.txt
$ git commit -m "AND simple"
[feature1 75a857c] AND simple
 1 file changed, 1 insertion(+), 1 deletion(-)
```

切换到 master 分支：

```
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
```

Git 还会自动提示我们当前 master 分支比远程的 master 分支要超前 1 个提交。

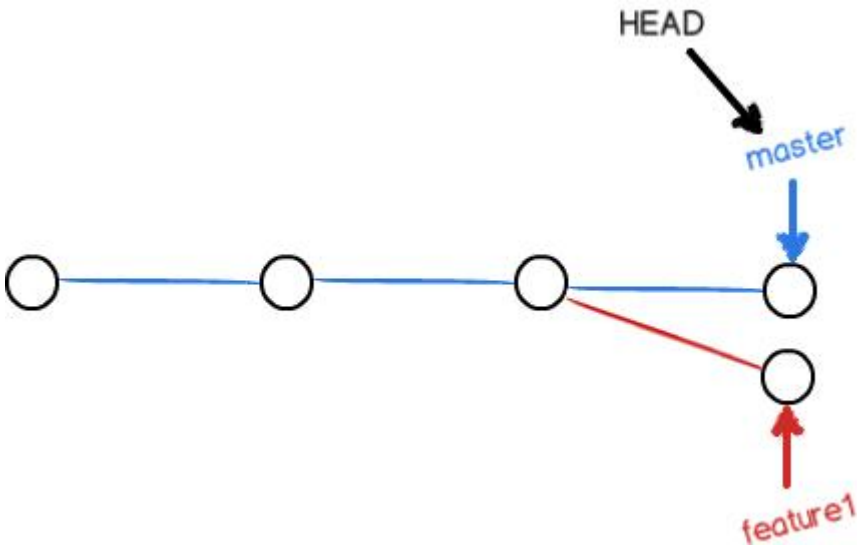
在 master 分支上把 readme.txt 文件的最后一行改为：

```
Creating a new branch is quick & simple.
```

提交：

```
$ git add readme.txt
$ git commit -m "& simple"
[master 400b400] & simple
 1 file changed, 1 insertion(+), 1 deletion(-)
```

现在，master 分支和 feature1 分支各自都分别有新的提交，变成了这样：



这种情况下，Git 无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并就可能会有冲突，我们试试看：

```
$ git merge feature1
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the result.
```

果然冲突了！Git 告诉我们，readme.txt 文件存在冲突，必须手动解决冲突后再提交。Git status 也可以告诉我们冲突的文件：

```
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 2 commits.
#
# Unmerged paths:
#   (use "git add/rm <file>..." as appropriate to mark resolution)
#
#       both modified:      readme.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

我们可以直接查看 readme.txt 的内容：

```
Git is a distributed version control system.
Git is free software distributed under the GPL.
Git has a mutable index called stage.
Git tracks changes of files.
<<<<<<< HEAD
Creating a new branch is quick & simple.
=======
Creating a new branch is quick AND simple.
>>>>>>> feature1
```

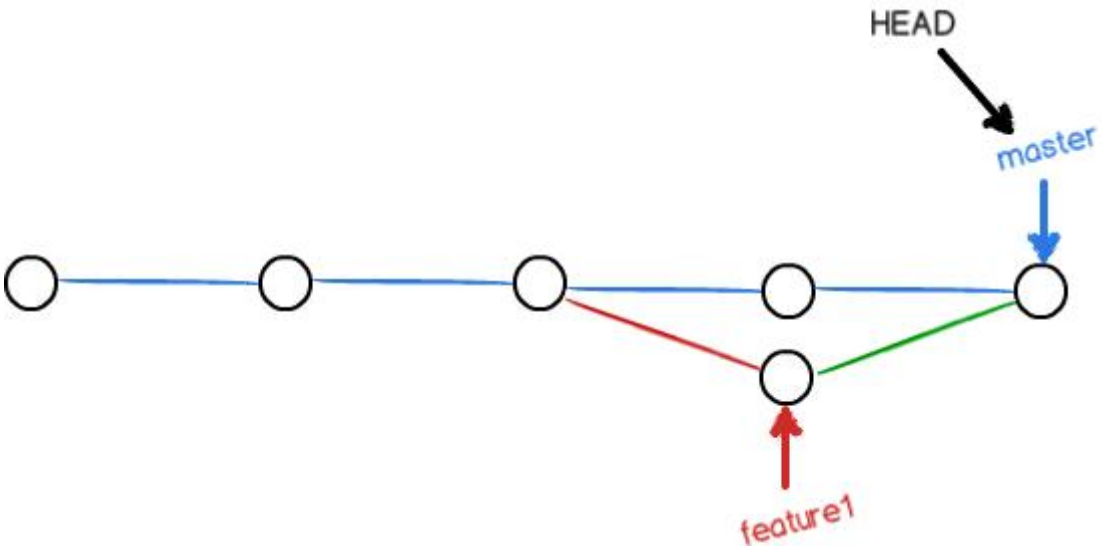
Git 用<<<<<<<，=======，>>>>>>>标记出不同分支的内容，我们修改如下后保存：

```
Creating a new branch is quick and simple.
```

再提交：

```
$ git add readme.txt
$ git commit -m "conflict fixed"
[master 59bc1cb] conflict fixed
```

现在，master 分支和 feature1 分支变成了下图所示：



用带参数的 git log 也可以看到分支的合并情况：

```
$ git log --graph --pretty=oneline --abbrev-commit
*   59bc1cb conflict fixed
|\
| * 75a857c AND simple
* | 400b400 & simple
|/
* fec145a branch test
...
```

现在，删除 feature1 分支：

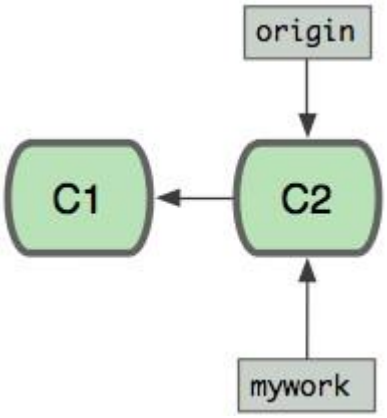
```
$ git branch -d feature1
Deleted branch feature1 (was 75a857c).
```

至此，合并工作完成。

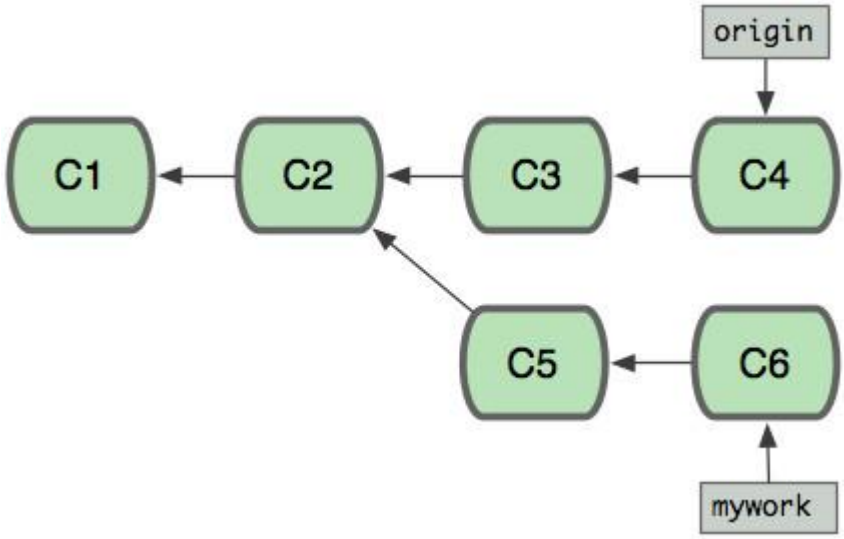
- Git rebase

于把一个分支的修改合并到当前分支。

假设远程分支"origin"已经有了 2 个提交，如图：

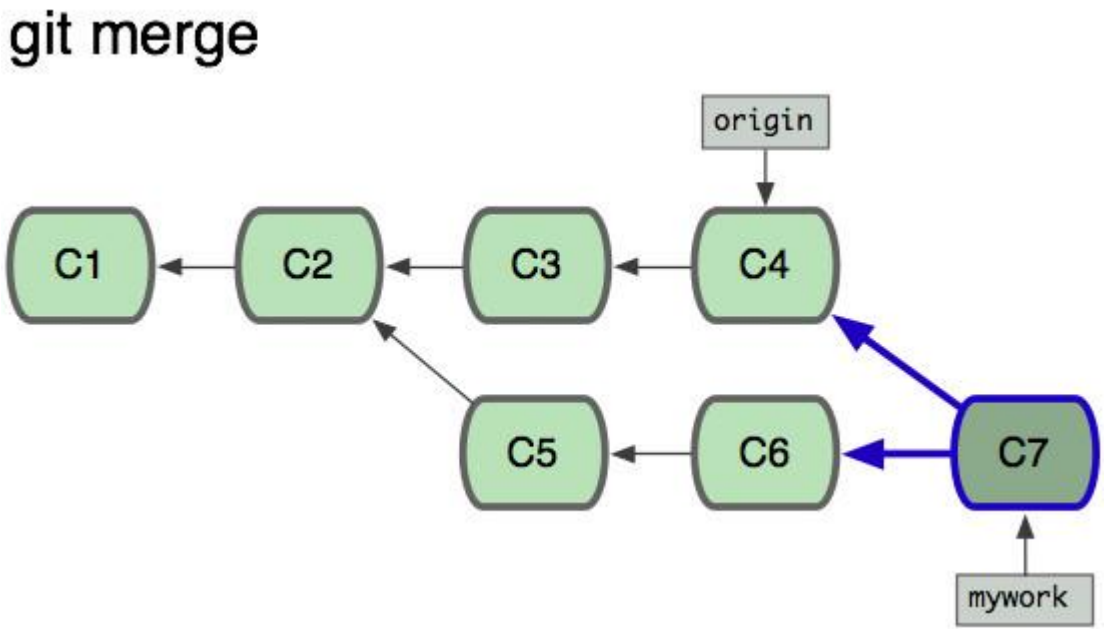


现在我们生成两个提交，与此同时，"origin"分支上也生成了提交，如图：



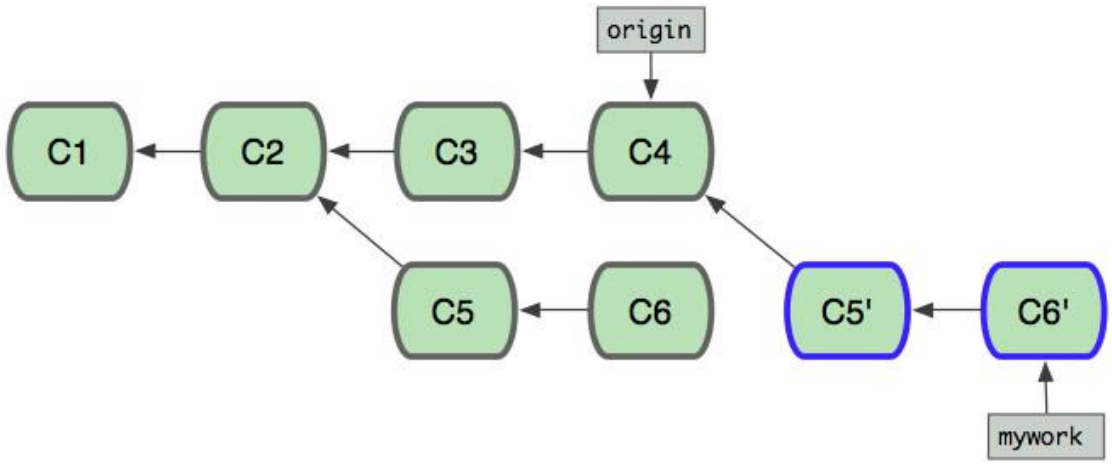
在这里,你可以用"pull"命令把"origin"分支上的修改拉下来并且和你的修改合并;

结果看起来就像一个新的"合并的提交"。

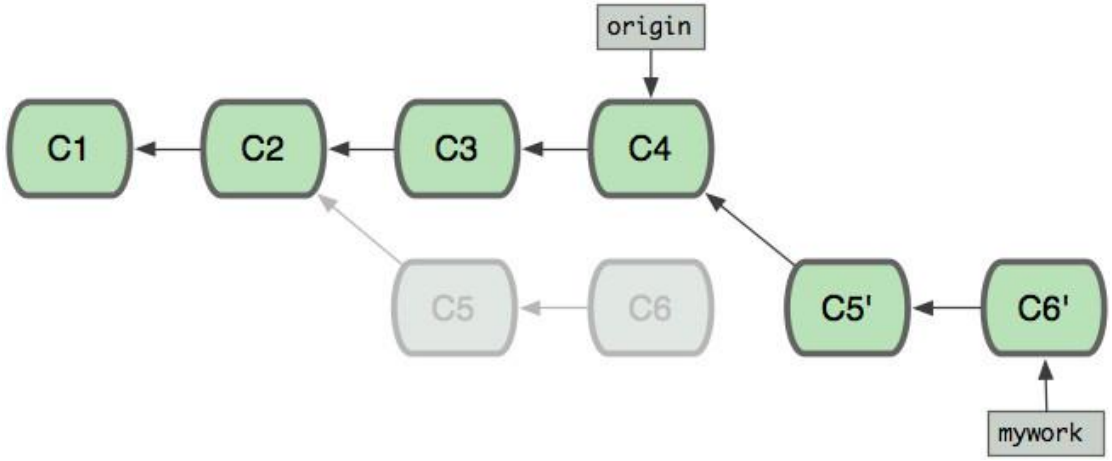


如果你想让"mywork"分支历史看起来像没有合并一样，可以用 git rebase：

```
# git checkout mywork
# git rebase origin
```



当"mywork"分支更新之后，如果运行垃圾收集命令(git gc)，如图：



在 rebase 的过程中，也许会出现冲突，Git 会停止 rebase 并且让你去解决冲突，

在解决完冲突后，用"git-add"命令去更新这些内容的索引，然后执行：

```
# git rebase --continue
```

如果需要终止 rebase 的行动，并且让"mywork" 分支回到 rebase 开始前的状态：

```
# git rebase --abort
```

14. Git tag

发布一个版本时，我们通常先在版本库中打一个标签，这样，就唯一确定了打标签时刻的版本。将来无论什么时候，取某个标签的版本，就是把那个打标签的时刻的历史版本取出来。所以，标签也是版本库的一个快照。

创建标签：

```
# git tag v1.0
```

删除标签：

```
# git tag -d v0.1
```

推送标签：

```
# git push origin --tags
```

（或者单独推送某个标签`# git push origin v1.0`）

15. Gitg & Gitk

图形化的 Git 操作，GUI 是很方便的，但也容易让我们忘记命令，自己取舍吧。Windows 下有小乌龟，Ubuntu 下有 Gitg 和 Gitk。

16. Git remote & Git stash & Git cherry-pick

- Git remote 命令就和分布式有关系了，Git 其实不需要服务器因为 Git init 以后本身就是服务器。但是要协作开发，那就需要添加了。

如果服务器挂了，根据 Git 本身特性仍然可以提交到本地，如果需要协作开发？分布式嘛，改下远端信息就好。

```
# git remote add tang ubuntu@172.26.182.116:/opt/nfs/tranning
```

当然还有删除、重命名等其他操作，自行查看帮助文档吧。

添加了服务器还不够，要把分支关联起来，就是映射关系，这样后续才能和远端协同开发。

```
# git branch --set-upstream-to=REMOTE_BRANCH
```

添加远端服务器后续的代码提交就要注意提交到哪里了，默认提交至 origin

```
# git push tang LOCAL_BRANCH
```
（现在回头看第 11 章节，可以理解这几个参数了吧）

- Git stash

在当前分支工作，此时有新任务插入的时候可以暂存当前的工作，这样可以减少很多不必要的 commit

```
# git stash save “save work progress”
# git stash list
# git stash pop
```

- Git cherry-pick

把已经提交的 commit 提交到另一个分支，减轻 bug fix 的负担

```
# git checkout old_dev_branch
# git cherry-pick 38361a68
```

* 如果顺利就会正常提交，如果出现冲突就需要手动解决一下

17. 其他

如果你不介意外网流量的话可以自己去申请个 github、bitbucket 进行练习

以下我提供一个内网的（IP 可能会变，Ping 下 tangwenkai 或者 tangwenkai.local）

几个人的话可以体验下协作开发(可以克隆以下仓库，也可以通过 git remote add）

git clone ubuntu@172.26.182.116:/opt/nfs/training
Passwd: ubuntu

练习流程参考：本地创建仓库 -> 本地克隆仓库 -> 新建分支 -> 基本操作 -> 添加远程仓库 -> 合并分支 -> 提交代码 -> 等等...

项目开发参考：更新代码 -> 从开发分支创建各自开发分支 -> 编码 -> 切换至主开发分支 -> 更新代码 -> 合并分支/解决冲突 -> 提交代码

18. 参考

之前也有很多 Git 的培训文档和视频，本文第 13 章节主要参考廖雪峰的博客，他的博客写的非常详细和系统，大家可以参考下。

Git Wiki：http://en.wikipedia.org/wiki/Git_%28software%29
Git 录像：samba://172.26.10.69/Projects/17CY/traning/02 - 培训资料(From Solar)/培训录像/
Git 文档：samba://172.26.10.69/Projects/17CY/traning/08 - git 相关资料/
Git 教程 - 廖雪峰的官方网站
http://www.liaoxuefeng.com/wiki/001373951630592960 6dd18361248578c67b8067c8c017b000
Gerrit 错误处理：http://iwiki.storm/develop-manual/gerrit-review-problem-solution