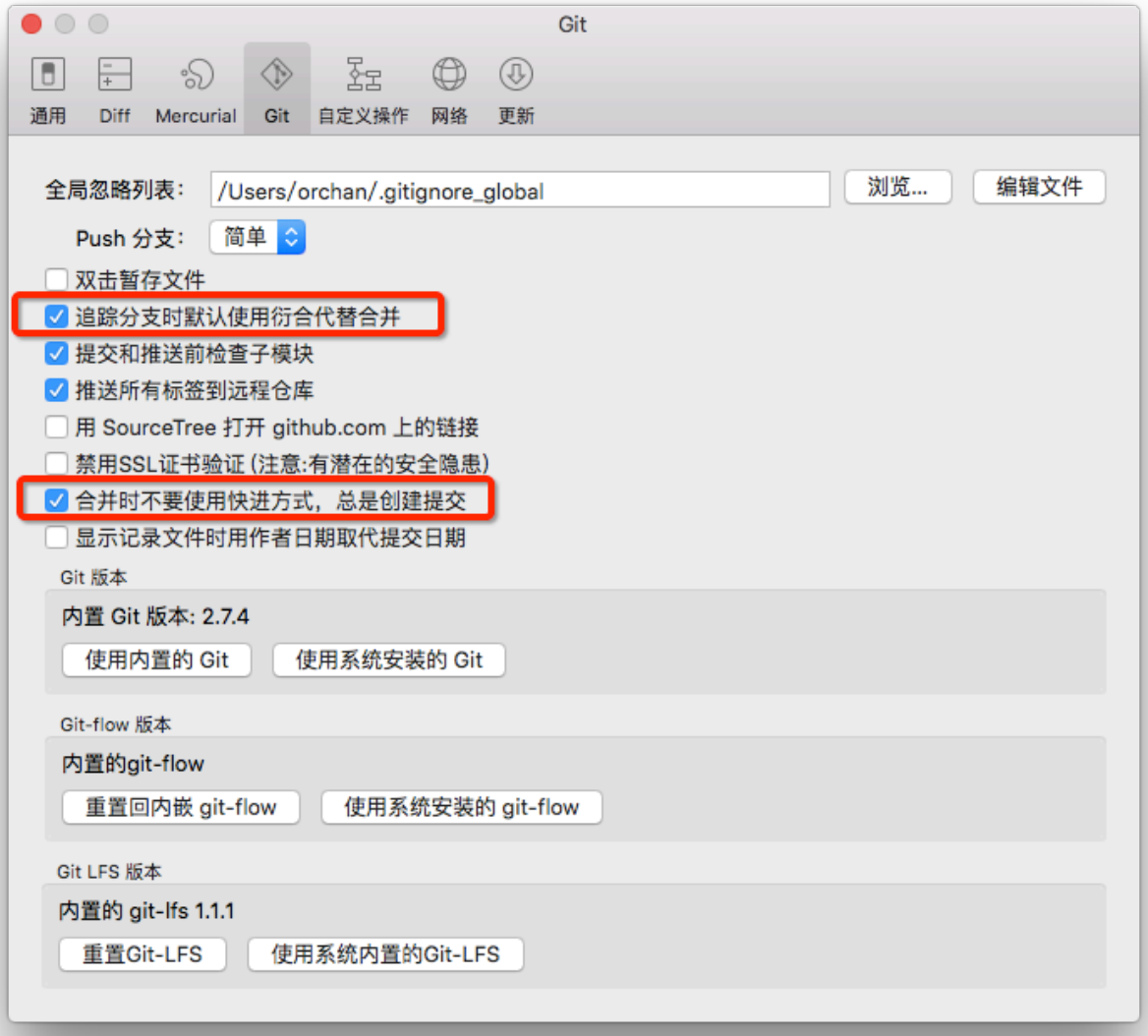
Git内部规范流程初探

> 团队开发中，遵循一个合理、清晰的Git使用流程，是非常重要的。否则，每个人都提交一堆杂乱无章的commit，项目很快就会变得难以协调和维护。

关于工具

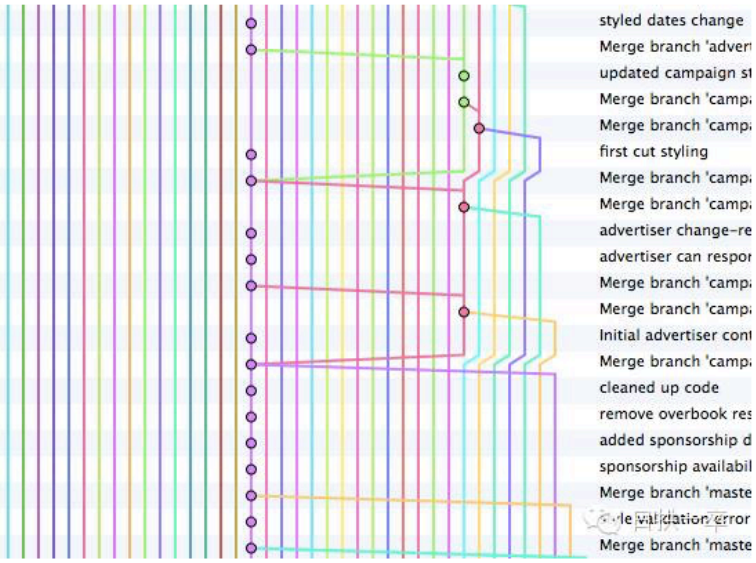
- *1.* 工具: Git可视化工具SourceTree、合并代码工具BeyondCompare
- *2.* 配置SourceTree：使用`rebase`替代`merge`，不使用`fast-forward`；Diff外部工具选择：`BeyondCompare`



打开终端，执行下列命令配置对比工具。

```
1 ln -s /Applications/Beyond\ Compare.app/Contents/MacOS/bcomp /usr/local/bin/
```

关于 `rebase` 和 `merge`



团队的开发模式是本地的`branch`和远端的`branch`会同步地非常频繁，这两个`branch`几乎是完全同步。这时候就会发现这些`merge`动作其实没有必要，会造成线图错综复杂。

Rebase大致操作：

- *1.* 把本地的`repo`从上次 pull 之后的的变更暂存起来
- *2.* 返回到上次 pull 时的情况
- *3.* 套用远端的变更
- *4.* 最后再套用刚才暂存下来的变更。

合并之前示意图：

```
    D---E master
   /
A---B---C---F origin/master
```

使用 `merge` 合并示意图：

```
    D--------E
   /          \
A---B---C---F----G   master, origin/master
```

使用 `rebase` 的方式，就不會有 `G` 合并点：

```
A---B---C---F---D'---E'   master, origin/master
```

> 注意到，其中 `D'`, `E'` 的 `commit SHA` 序号跟本來 `D`, `E` 是不同的，因為算是砍掉重新 `commit` 了。

关于使用时遇到`conflict`的情况：
`merge` 如果发生 `conflict`，你只需要解决衝突一次；`rebase` 如果发生 `conflict`，你需要解决冲突多次。

分支策略

在实际开发中，我们应该按照几个基本原则进行分支管理：

`master分支`：应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活。

`dev分支`（远端）：干活都在`dev`分支上，也就是说，`dev`分支是不稳定的，到某个时候，比如`1.0`版本发布时，再把`dev`分支合并到`master`上，在`master`分支发布1.0版本。

`fix分支`（远端）：使用`master分支`发布版本后出现的bug，在此分支上修复最后合并到`master分支`。

`feature分支`（远端）：用于开发新功能，如：Purifit的专项运动。

`local分支`（本地）：每个人都有自己的分支，时不时地往dev分支上合并就可以了。

所以，团队合作的分支看起来就像这样：