Git简单教程

Git基础

- 直接记录快照，而非差异

Git 和其它版本控制系统(包括 Subversion 和近似工具)的主要差别在于 Git 对待数据的方法。 概念上来区 分，其它大部分系统以文件变更列表的方式存储信息。 这类系统(CVS、Subversion、Perforce、Bazaar 等 等)将它们保存的信息看作是一组基本文件和每个文件随时间逐步累积的差异。

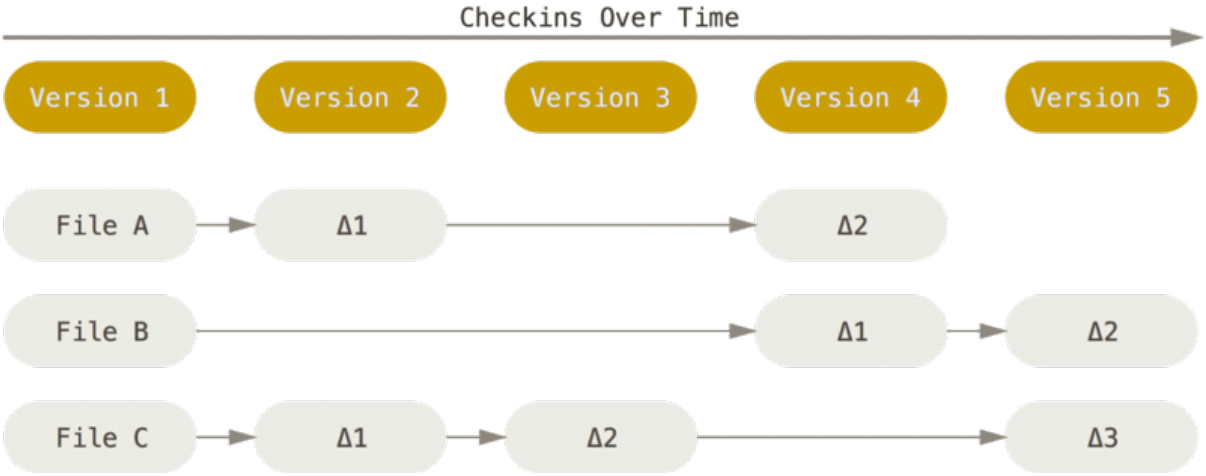


Figure 4. 存储每个文件与初始版本的差异.

Git 不按照以上方式对待或保存数据。 反之，Git 更像是把数据看作是对小型文件系统的一组快照。 每次你提交 更新，或在 Git 中保存项目状态时，它主要对当时的全部文件制作一个快照并保存这个快照的索引。 为了高效， 如果文件没有修改，Git 不再重新存储该文件，而是只保留一个链接指向之前存储的文件。 Git 对待数据更像是 一个 快照流。

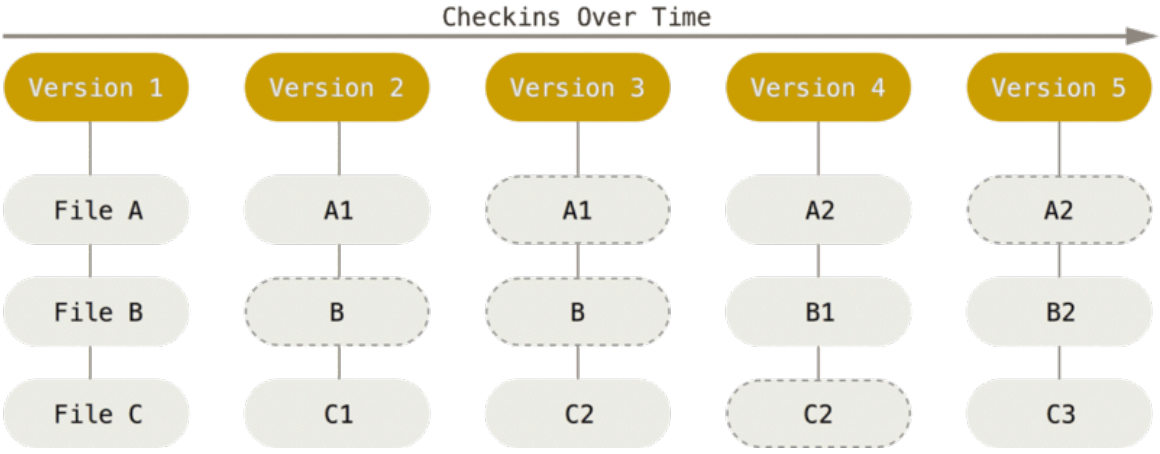


Figure 5. 存储项目随时间改变的快照.

这是 Git 与几乎所有其它版本控制系统的重要区别。 因此 Git 重新考虑了以前每一代版本控制系统延续下来的诸 多方面。 Git 更像是一个小型的文件系统，提供了许多以此为基础构建的超强工具，而不只是一个简单的 VCS。 稍后我们在Git 分支讨论 Git 分支管理时，将探究这种方式对待数据所能获得的益处。

- 三种状态

Git 有三种状态，你的文件可能处 于其中之一:已提交(committed)、已修改(modified)和已暂存(staged)。 已提交表示数据已经安全的 保存在本地数据库中。 已修改表示修改了文件，但还没保存到数据库中。 已暂存表示对一个已修改文件的当前 版本做了标记，使之包含在下次提交的快照中。

由此引入 Git 项目的三个工作区域的概念:Git 仓库、工作目录以及暂存区域。

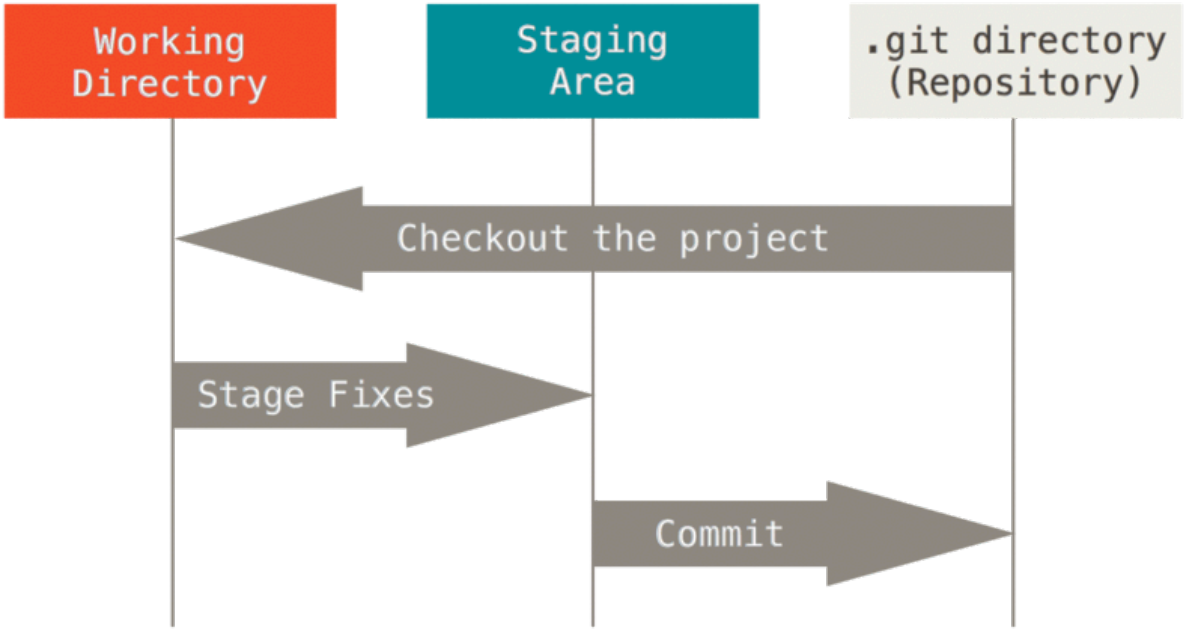


Figure 6. 工作目录、暂存区域以及 Git 仓库.

仓库目录是 Git 用来保存项目的元数据和对象数据库的地方。 这是 Git 中最重要的部分，从其它计算机克隆仓库时，拷贝的就是这里的数据。

工作目录是对项目的某个版本独立提取出来的内容。 这些从 Git 仓库的压缩数据库中提取出来的文件，放在磁盘上供你使用或修改。

暂存区域是一个文件，保存了下次将提交的文件列表信息，一般在 Git 仓库目录中。有时候也被称作‘索引’，不过一般说法还是叫暂存区域。 基本的 Git 工作流程如下:

1. 在工作目录中修改文件。
2. 暂存文件，将文件的快照放入暂存区域。
3. 提交更新，找到暂存区域的文件，将快照永久性存储到 Git 仓库目录。

- 记录每次更新到仓库

你工作目录下的每一个文件都不外乎这两种状态:已跟踪或未跟踪。 已跟踪的文件是指那些被纳入了 版本控制的文件，在上一次快照中有它们的记录，在工作一段时间后，它们的状态可能处于未修改，已修改或已 放入暂存区。 工作目录中除已跟踪文件以外的所有其它文件都属于未跟踪文件，它们既不存在于上次快照的记 录中，也没有放入暂存区。 初次克隆某个仓库的时候，工作目录中的所有文件都属于已跟踪文件，并处于未修 改状态。

编辑过某些文件之后，由于自上次提交后你对它们做了修改，Git 将它们标记为已修改文件。 我们逐步将这些修 改过的文件放入暂存区，然后提交所有暂存了的修改，如此反复。 所以使用 Git 时文件的生命周期如下:

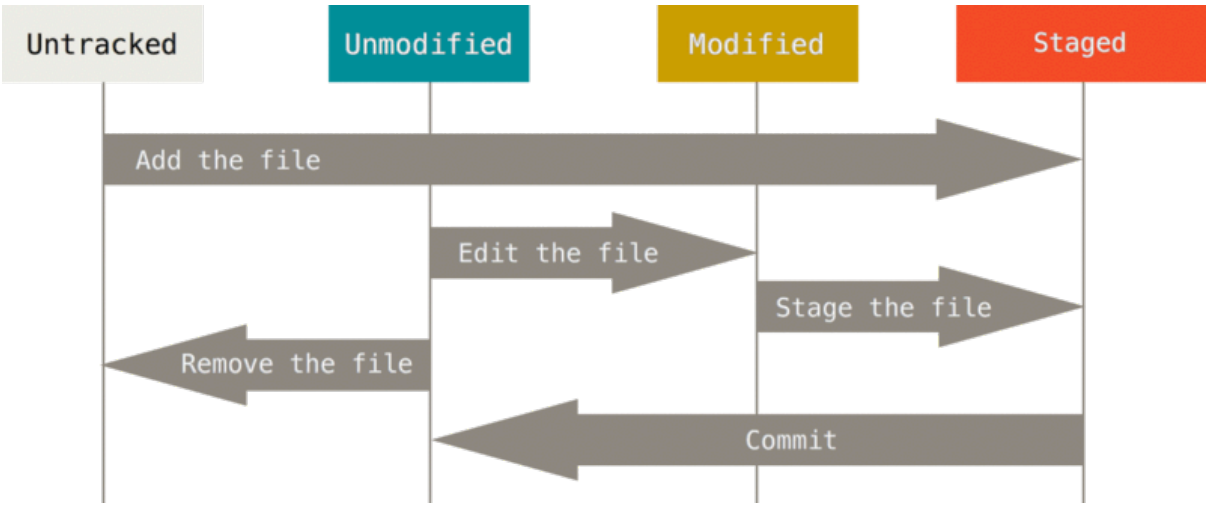


Figure 8. 文件的状态变化周期

- 检查当前文件状态

```
 $ git status
On branch master
nothing to commit, working directory clean
```

- 跟踪新文件

  git add -A 提交所有变化

  git add -u 提交被修改(modified)和被删除(deleted)文件，不包括新文件(new)

  git add . 提交新文件(new)和被修改(modified)文件，不包括被删除(deleted)文件

- 忽略文件

  $ cat .gitignore

- 查看已暂存和未暂存的修改

```
$ git diff
```

- 提交更新

```
$ git commit
$ git commit -a
```

- 查看提交历史

```
$ git log -p -2
$ git show commitID
```

- 取消暂存的文件

```
$ git reset HEAD <file>
```

- 撤销对文件的修改

```
$ git checkout — [file]
```

- 远程仓库的使用

- 查看远程仓库

```
$ git clone -b master https://github.com/schacon/ticgit
```

- 从远程仓库中抓取与拉取

```
$ git fetch [remote-name]
```

相当于是从远程获取最新版本到本地，不会自动merge

```
$ git pull
```

相当于是从远程获取最新版本并merge到本地

- 推送到远程仓库

```
$ git push origin master
```

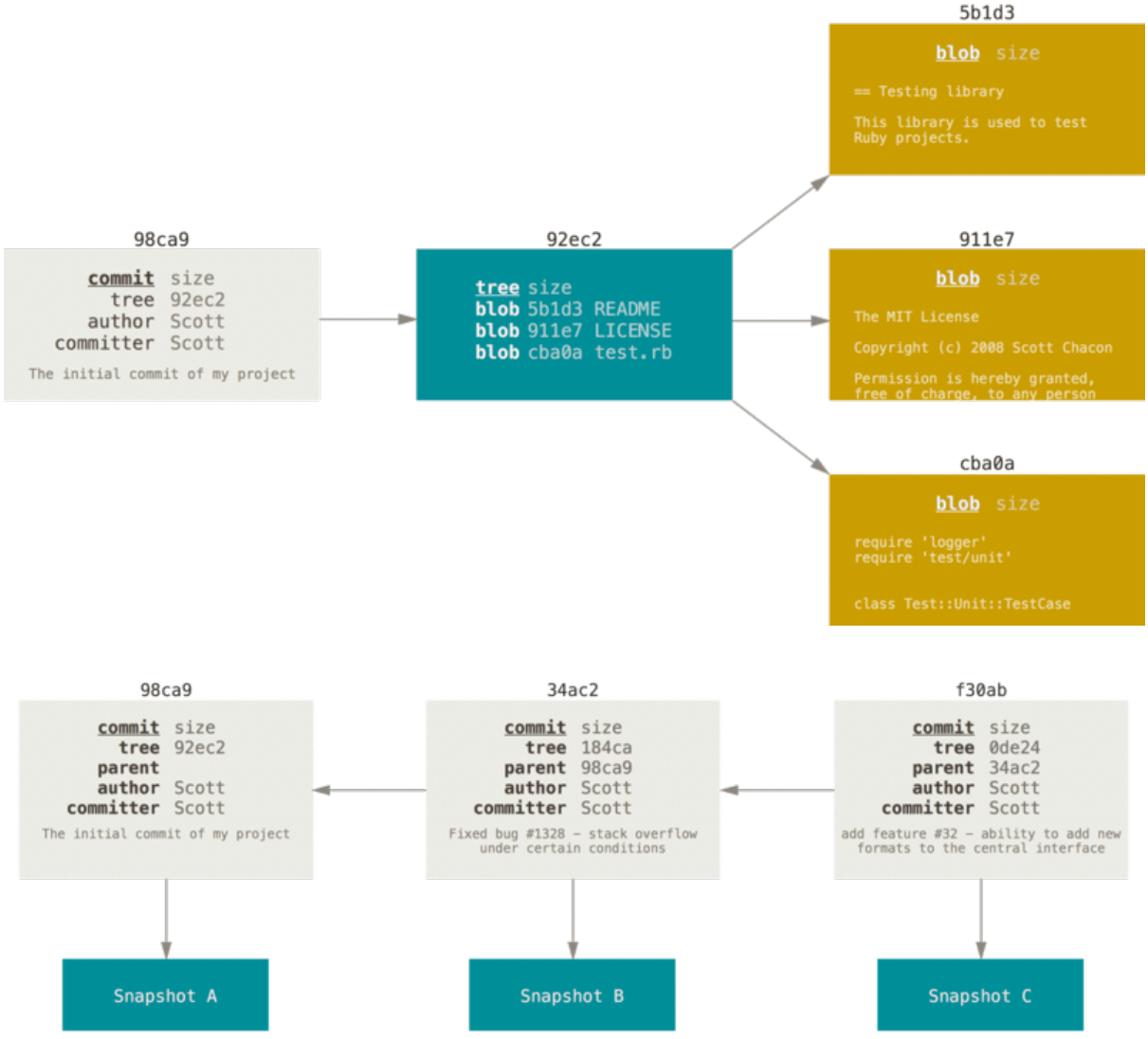
Git分支

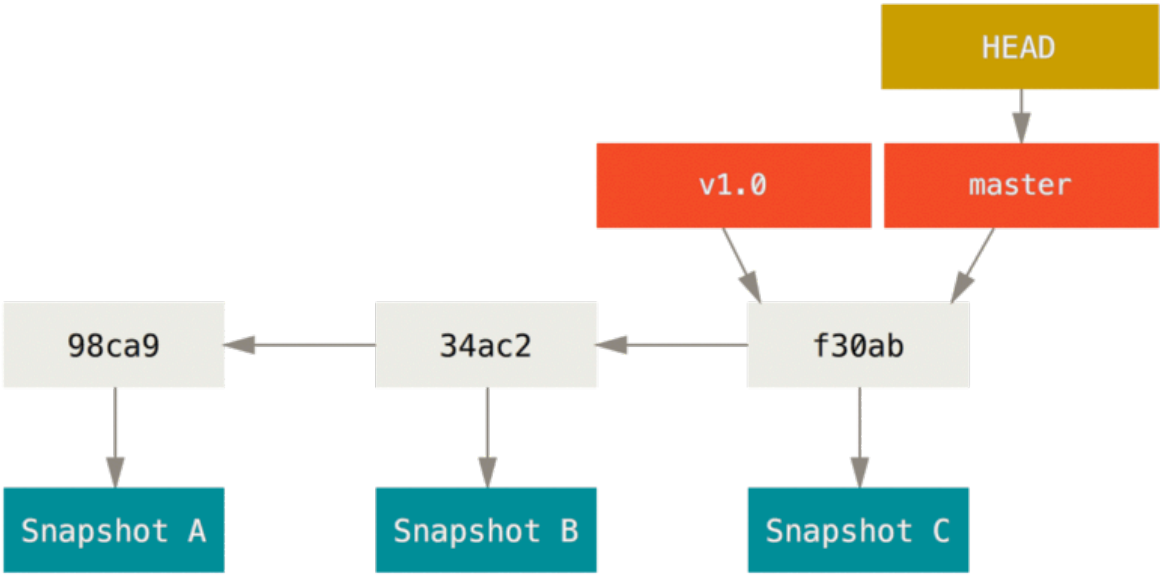
- 分支简介

当使用git commit进行提交操作时，Git会先计算每一个子目录(本例中只有项目根目录)的校验和，然后在 Git 仓库中这些校验和保存为树对象。 随后，Git 便会创建一个提交对象，

它除了包含上面提到的那些信息外， 还包含指向这个树对象(项目根目录)的指针。如此一来，Git 就可以在需要的时候重现此次保存的快照。

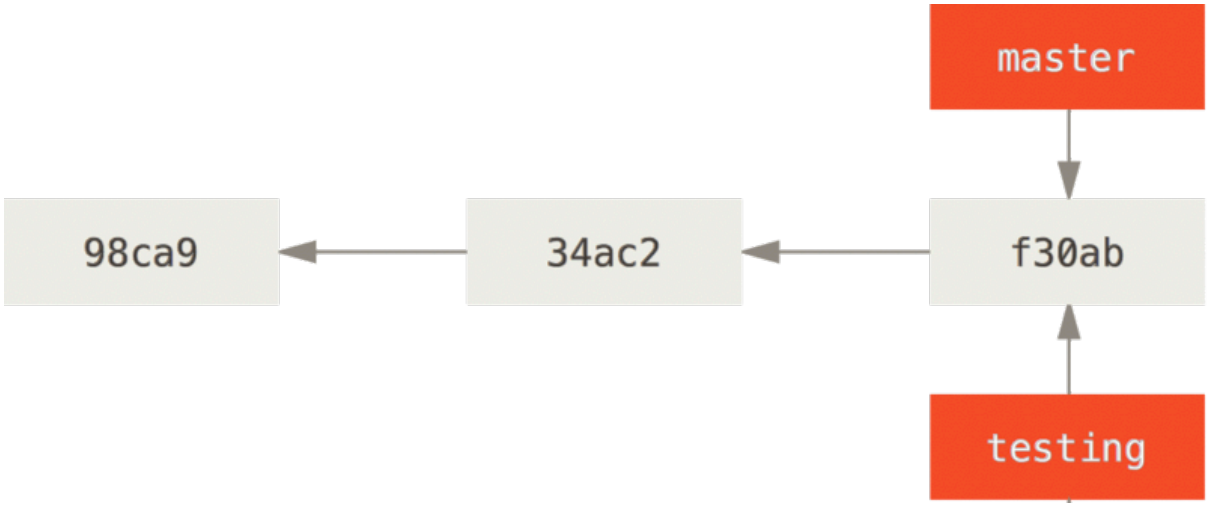
现在，Git 仓库中有五个对象:三个 blob 对象(保存着文件快照)、一个树对象(记录着目录结构和 blob 对象 索引)以及一个提交对象(包含着指向前述树对象的指针和所有提交信息)。





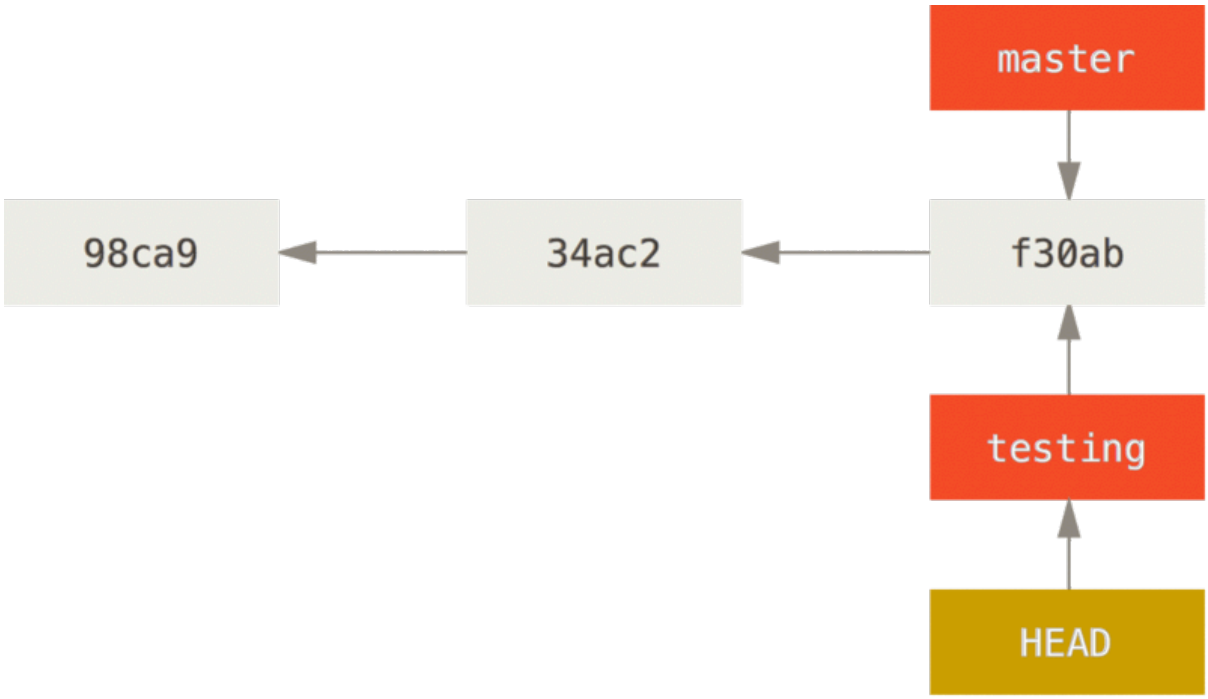
- 分支创建

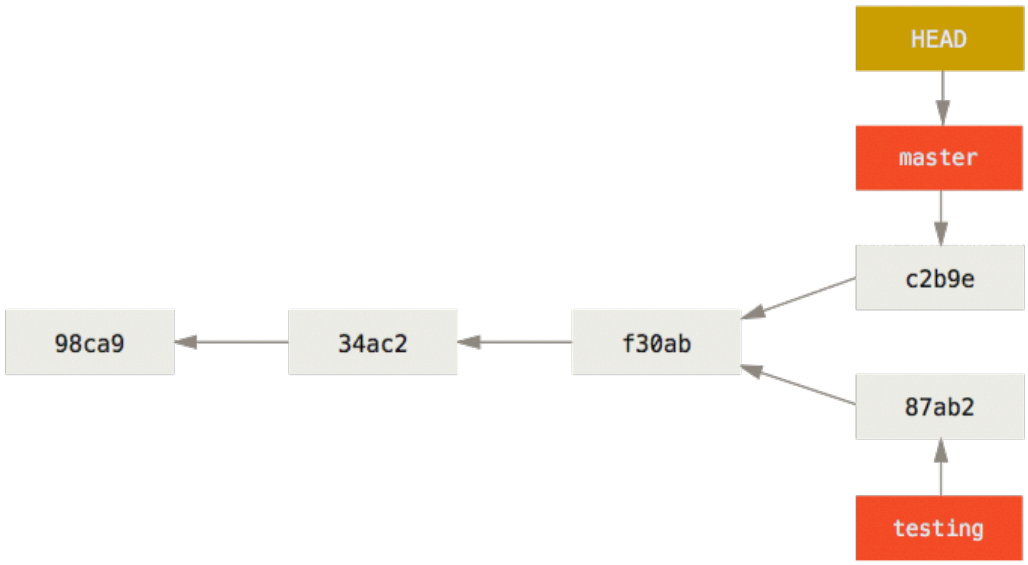
  $git branch testing



- 分支切换

  $ git checkout testing





- 分支合并

  $ git checkout master
  Switched to branch 'master'
  $ git merge iss53
  Merge made by the 'recursive' strategy.

```
 index.html |    1 +
1 file changed, 1 insertion(+)
```

- 遇到冲突时的分支合并

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

任何因包含合并冲突而有待解决的文件，都会以未合并状态标识出来。 Git 会在有冲突的文件中加入标准的冲突 解决标记，这样你可以打开这些包含冲突的文件然后手动解决冲突。 出现冲突的文件会包含一些特殊区段，看 起来像下面这个样子:

```
<<<<<<< HEAD:index.html

 <div id="footer">contact : email.support@github.com</div>
 =======
 <div id="footer">
  please contact us at support@github.com
 </div>
 >>>>>>> iss53:index.html
```

在你解决了所有文件里的冲突之后，对每个文件使用 git add 命令来将其标记为冲突已解决。 一旦暂存这 些原本有冲突的文件，Git 就会将它们标记为冲突已解决。

```
$ git mergetool
```