https://git-scm.com/book/zh/v1/Git-%E5%88%86%E6%94%AF-%E8%BF%9C%E7%A8%8B%E5%88%86%E6%94%AF

.5 Git 分支 - 远程分支

**远程分支**

远程分支（remote branch）是对远程仓库中的分支的索引。它们是一些无法移动的本地分支；只有在 Git 进行网络交互时才会更新。远程分支就像是书签，提醒着你上次连接远程仓库时上面各分支的位置。

我们用 (远程仓库名)/(分支名) 这样的形式表示远程分支。比如我们想看看上次同 origin 仓库通讯时 master 分支的样子，就应该查看 origin/master 分支。如果你和同伴一起修复某个问题，但他们先推送了一个 iss53 分支到远程仓库，虽然你可能也有一个本地的 iss53 分支，但指向服务器上最新更新的却应该是 origin/iss53 分支。

可能有点乱，我们不妨举例说明。假设你们团队有个地址为 git.ourcompany.com 的 Git 服务器。如果你从这里克隆，Git 会自动为你将此远程仓库命名为 origin，并下载其中所有的数据，建立一个指向它的 master 分支的指针，在本地命名为 origin/master，但你无法在本地更改其数据。接着，Git 建立一个属于你自己的本地 master 分支，始于 origin 上 master 分支相同的位置，你可以就此开始工作（见图 3-22）：

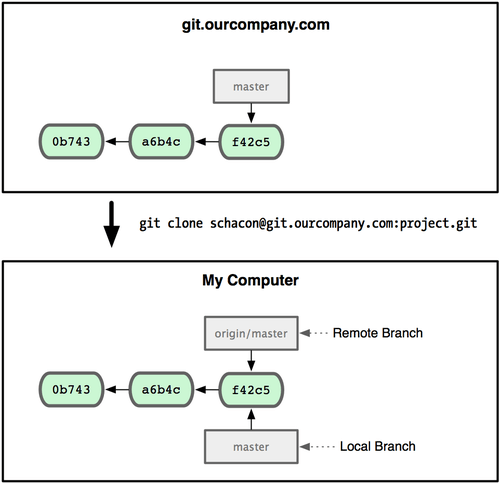


图 3-22. 一次 Git 克隆会建立你自己的本地分支 master 和远程分支 origin/master，并且将它们都指向 origin 上的 master 分支。

如果你在本地 master 分支做了些改动，与此同时，其他人向 git.ourcompany.com 推送了他们的更新，那么服务器上的 master 分支就会向前推进，而与此同时，你在本地的提交历史正朝向不同方向发展。不过只要你不和服务器通讯，你的 origin/master 指针仍然保持原位不会移动（见图 3-23）。

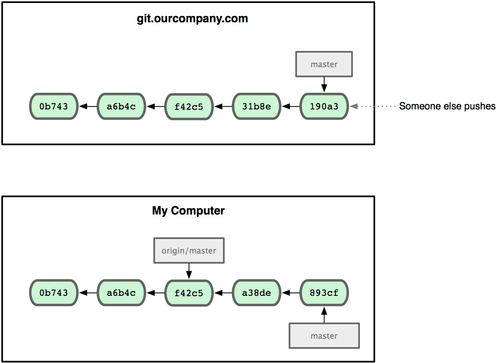


图 3-23. 在本地工作的同时有人向远程仓库推送内容会让提交历史开始分流。

可以运行 git fetch origin 来同步远程服务器上的数据到本地。该命令首先找到 origin 是哪个服务器（本例为 git.ourcompany.com），从上面获取你尚未拥有的数据，更新你本地的数据库，然后把 origin/master 的指针移到它最新的位置上（见图 3-24）。

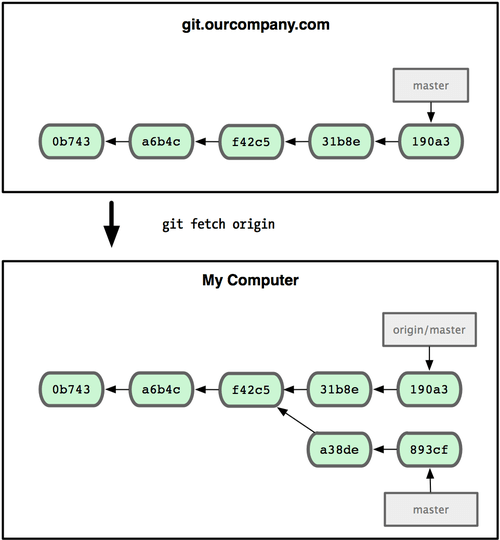


图 3-24. git fetch 命令会更新 remote 索引。

为了演示拥有多个远程分支（在不同的远程服务器上）的项目是如何工作的，我们假设你还有另一个仅供你的敏捷开发小组使用的内部服务器 git.team1.ourcompany.com。可以用第二章中提到的 git remote add 命令把它加为当前项目的远程分支之一。我们把它命名为 teamone，以便代替完整的 Git URL 以方便使用（见图 3-25）。

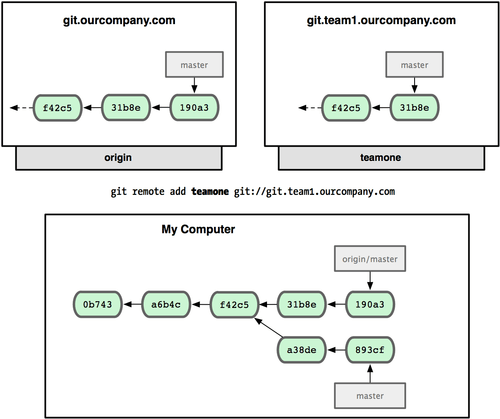


图 3-25. 把另一个服务器加为远程仓库

现在你可以用 git fetch teamone 来获取小组服务器上你还没有的数据了。由于当前该服务器上的内容是你 origin 服务器上的子集，Git 不会下载任何数据，而只是简单地创建一个名为 teamone/master 的远程分支，指向 teamone 服务器上 master 分支所在的提交对象 31b8e（见图 3-26）。

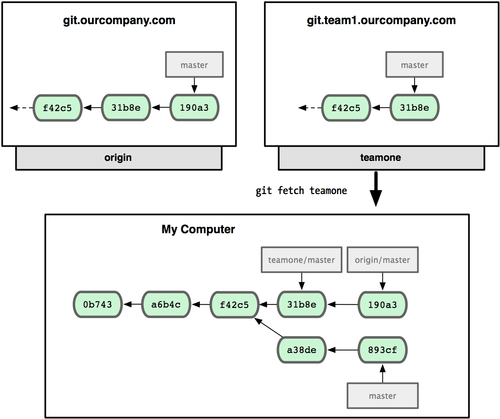


图 3-26. 你在本地有了一个指向 teamone 服务器上 master 分支的索引。

[**推送本地分支**](https://git-scm.com/book/zh/v1/Git-%E5%88%86%E6%94%AF-%E8%BF%9C%E7%A8%8B%E5%88%86%E6%94%AF#%E6%8E%A8%E9%80%81%E6%9C%AC%E5%9C%B0%E5%88%86%E6%94%AF)

要想和其他人分享某个本地分支，你需要把它推送到一个你拥有写权限的远程仓库。你创建的本地分支不会因为你的写入操作而被自动同步到你引入的远程服务器上，你需要明确地执行推送分支的操作。换句话说，对于无意分享的分支，你尽管保留为私人分支好了，而只推送那些协同工作要用到的特性分支。

如果你有个叫 serverfix 的分支需要和他人一起开发，可以运行 git push (远程仓库名) (分支名)：

$ git push origin serverfix

Counting objects: 20, done.

Compressing objects: 100% (14/14), done.

Writing objects: 100% (15/15), 1.74 KiB, done.

Total 15 (delta 5), reused 0 (delta 0)

To git@github.com:schacon/simplegit.git

\* [new branch] serverfix -> serverfix

这里其实走了一点捷径。Git 自动把 serverfix 分支名扩展为 refs/heads/serverfix:refs/heads/serverfix，意为“取出我在本地的 serverfix 分支，推送到远程仓库的 serverfix 分支中去”。我们将在第九章进一步介绍 refs/heads/ 部分的细节，不过一般使用的时候都可以省略它。也可以运行 git push origin serverfix:serverfix 来实现相同的效果，它的意思是“上传我本地的 serverfix 分支到远程仓库中去，仍旧称它为 serverfix 分支”。通过此语法，你可以把本地分支推送到某个命名不同的远程分支：若想把远程分支叫作 awesomebranch，可以用 git push origin serverfix:awesomebranch 来推送数据。

接下来，当你的协作者再次从服务器上获取数据时，他们将得到一个新的远程分支 origin/serverfix，并指向服务器上 serverfix 所指向的版本：

$ git fetch origin

remote: Counting objects: 20, done.

remote: Compressing objects: 100% (14/14), done.

remote: Total 15 (delta 5), reused 0 (delta 0)

Unpacking objects: 100% (15/15), done.

From git@github.com:schacon/simplegit

\* [new branch] serverfix -> origin/serverfix

值得注意的是，在 fetch 操作下载好新的远程分支之后，你仍然无法在本地编辑该远程仓库中的分支。换句话说，在本例中，你不会有一个新的 serverfix 分支，有的只是一个你无法移动的 origin/serverfix 指针。

如果要把该远程分支的内容合并到当前分支，可以运行 git merge origin/serverfix。如果想要一份自己的 serverfix 来开发，可以在远程分支的基础上分化出一个新的分支来：

$ git checkout -b serverfix origin/serverfix

Branch serverfix set up to track remote branch serverfix from origin.

Switched to a new branch 'serverfix'

这会切换到新建的 serverfix 本地分支，其内容同远程分支 origin/serverfix 一致，这样你就可以在里面继续开发了。

[**跟踪远程分支**](https://git-scm.com/book/zh/v1/Git-%E5%88%86%E6%94%AF-%E8%BF%9C%E7%A8%8B%E5%88%86%E6%94%AF#%E8%B7%9F%E8%B8%AA%E8%BF%9C%E7%A8%8B%E5%88%86%E6%94%AF)

从远程分支 checkout 出来的本地分支，称为 *跟踪分支* (tracking branch)。跟踪分支是一种和某个远程分支有直接联系的本地分支。在跟踪分支里输入 git push，Git 会自行推断应该向哪个服务器的哪个分支推送数据。同样，在这些分支里运行 git pull 会获取所有远程索引，并把它们的数据都合并到本地分支中来。

在克隆仓库时，Git 通常会自动创建一个名为 master 的分支来跟踪 origin/master。这正是 git push 和 git pull 一开始就能正常工作的原因。当然，你可以随心所欲地设定为其它跟踪分支，比如 origin 上除了 master 之外的其它分支。刚才我们已经看到了这样的一个例子：git checkout -b [分支名] [远程名]/[分支名]。如果你有 1.6.2 以上版本的 Git，还可以用 --track 选项简化：

$ git checkout --track origin/serverfix

Branch serverfix set up to track remote branch serverfix from origin.

Switched to a new branch 'serverfix'

要为本地分支设定不同于远程分支的名字，只需在第一个版本的命令里换个名字：

$ git checkout -b sf origin/serverfix

Branch sf set up to track remote branch serverfix from origin.

Switched to a new branch 'sf'

现在你的本地分支 sf 会自动将推送和抓取数据的位置定位到 origin/serverfix 了。

[**删除远程分支**](https://git-scm.com/book/zh/v1/Git-%E5%88%86%E6%94%AF-%E8%BF%9C%E7%A8%8B%E5%88%86%E6%94%AF#%E5%88%A0%E9%99%A4%E8%BF%9C%E7%A8%8B%E5%88%86%E6%94%AF)

如果不再需要某个远程分支了，比如搞定了某个特性并把它合并进了远程的 master 分支（或任何其他存放稳定代码的分支），可以用这个非常无厘头的语法来删除它：git push [远程名] :[分支名]。如果想在服务器上删除 serverfix 分支，运行下面的命令：

$ git push origin :serverfix

To git@github.com:schacon/simplegit.git

- [deleted] serverfix

咚！服务器上的分支没了。你最好特别留心这一页，因为你一定会用到那个命令，而且你很可能会忘掉它的语法。有种方便记忆这条命令的方法：记住我们不久前见过的 git push [远程名] [本地分支]:[远程分支] 语法，如果省略 [本地分支]，那就等于是在说“在这里提取空白然后把它变成[远程分支]”。

[prev](https://git-scm.com/book/zh/v1/Git-%E5%88%86%E6%94%AF-%E5%88%A9%E7%94%A8%E5%88%86%E6%94%AF%E8%BF%9B%E8%A1%8C%E5%BC%80%E5%8F%91%E7%9A%84%E5%B7%A5%E4%BD%9C%E6%B5%81%E7%A8%8B) | [next](https://git-scm.com/book/zh/v1/Git-%E5%88%86%E6%94%AF-%E5%88%86%E6%94%AF%E7%9A%84%E5%8F%98%E5%9F%BA)