# git record

1. aaa

2. change user.name use git config --local

3. git config --local core.editor "" 改变默认的编辑器

4. 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，因为 Git 刚刚检出了它们， 而你尚未编辑过它们。

5. git status -s 查看简短状态命令

6. git diff 比较的是工作目录中当前文件和暂存区快照之间的差异。也就是修改后还没有暂存起来的变化的内容。

7. 若要查看已暂存的将要添加到下次提交里的内容，可以用 git diff --staged 命令。 这条命令将比对已暂存文件与最后一次提交的文件差异。

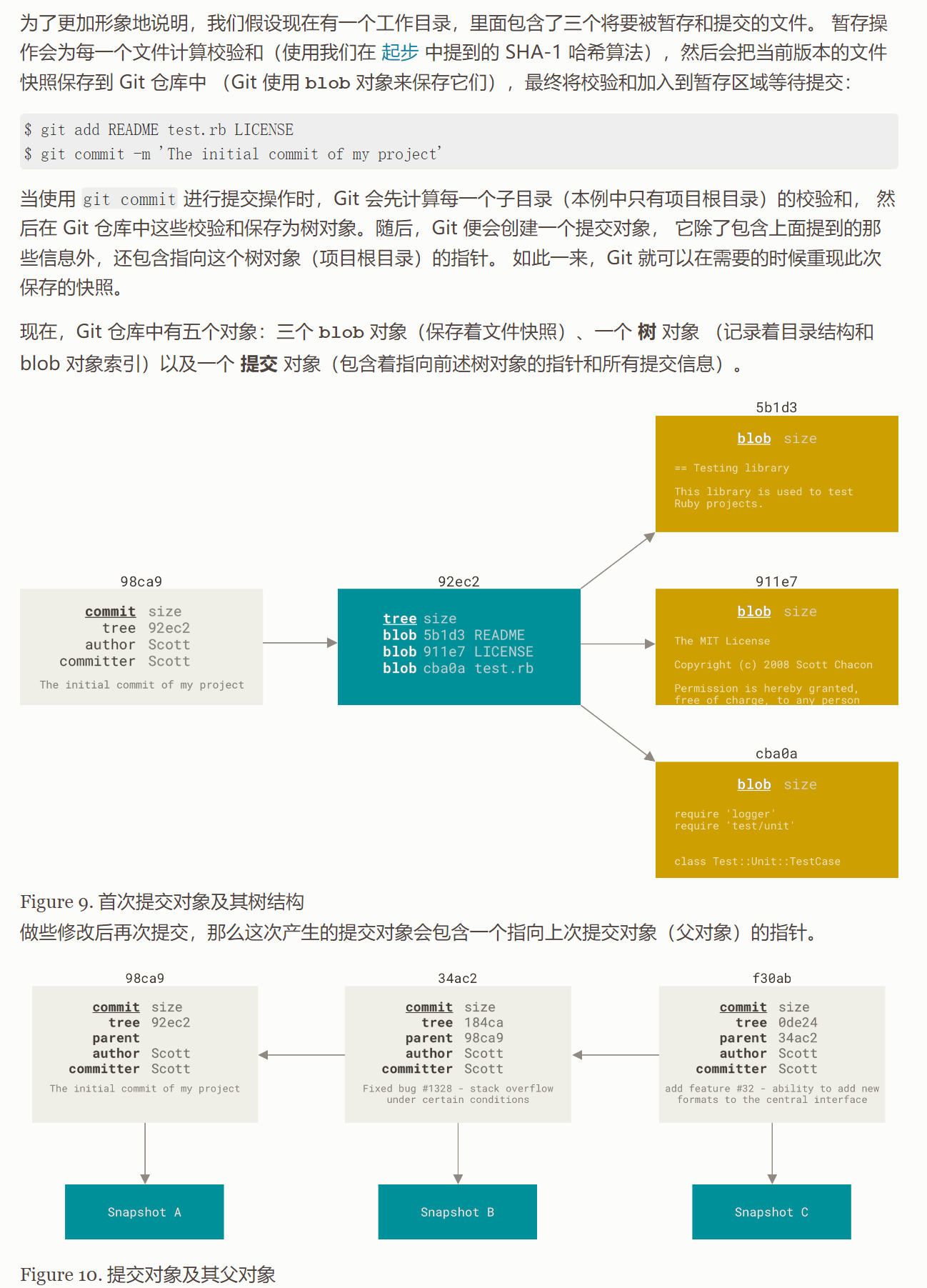
8. git rm 从已跟踪文件清单中删除，即从暂存区移除。这样以后就不会出现在未跟踪文件清单中了。。

9. git diff --staged

10.git fetch <remote> 命令从远程仓库获得数据，但是并不会合并到当前分支。例如远程分支fetch下来后有新的文件，那么在执行fetch后本地磁盘目录中并没有这个文件，。你需要把这个远程分支合并到本地，或者切换到这个远程分支查看。

11.Git 的分支，其实本质上仅仅是指向提交对象的可变指针。 Git 的默认分支名字是 master。 在多次提交操作之后，你其实已经有一个指向最后那个提交对象的 master 分支。 master 分支会在每次提交时自动向前移动。

那么，Git 又是怎么知道当前在哪一个分支上呢？ 也很简单，它有一个名为 HEAD 的特殊指针。 请注意它和许多其它版本控制系统（如 Subversion 或 CVS）里的 HEAD 概念完全不同。 在 Git 中，它是一个指针，指向当前所在的本地分支（译注：将 HEAD 想象为当前分支的别名）。

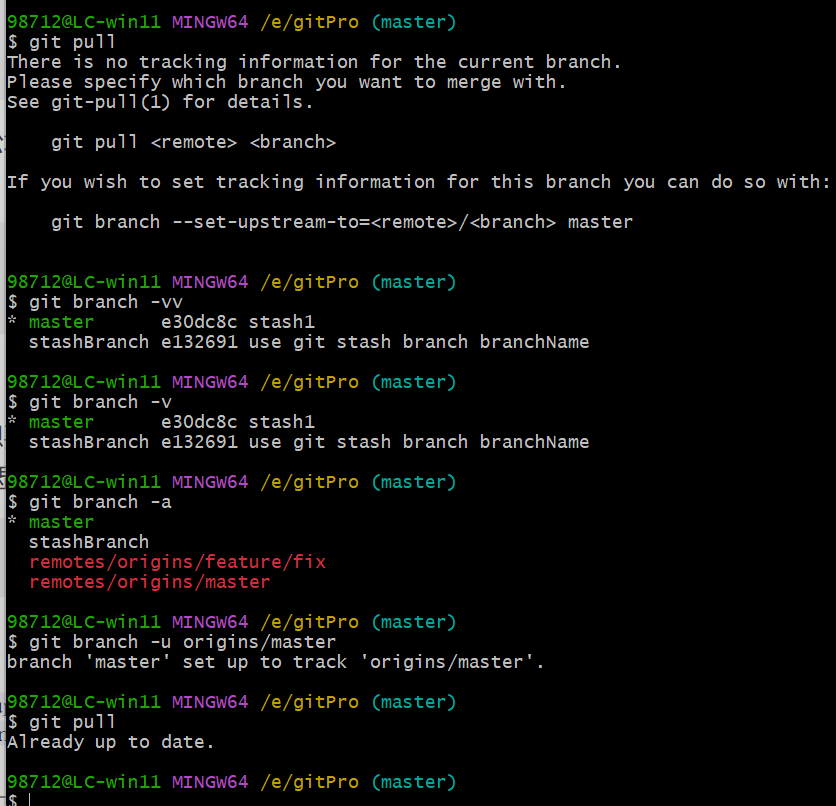




1. git stash：（stash）会处理工作目录的脏的状态——即跟踪文件的修改与暂存的改动——然后将未完成的修改保存到一个栈上， 而你可以在任何时候重新应用这些改动（甚至在不同的分支上）。

### **从贮藏创建一个分支**

如果贮藏了一些工作，将它留在那儿了一会儿，然后继续在贮藏的分支上工作，在重新应用工作时可能会有问题。 如果应用尝试修改刚刚修改的文件，你会得到一个合并冲突并不得不解决它。 如果想要一个轻松的方式来再次测试贮藏的改动，可以运行 git stash branch <new branchname> 以你指定的分支名创建一个新分支，检出贮藏工作时所在的提交，重新在那应用工作，然后在应用成功后丢弃贮藏：

1. 

# 3.5 Git 分支 - 远程分支

## **远程分支**

远程引用是对远程仓库的引用（指针），包括分支、标签等等。 你可以通过 git ls-remote <remote> 来显式地获得远程引用的完整列表， 或者通过 git remote show <remote> 获得远程分支的更多信息。 然而，一个更常见的做法是利用远程跟踪分支。

远程跟踪分支是远程分支状态的引用。它们是你无法移动的本地引用。一旦你进行了网络通信， Git 就会为你移动它们以精确反映远程仓库的状态。请将它们看做书签， 这样可以提醒你该分支在远程仓库中的位置就是你最后一次连接到它们的位置。

它们以 <remote>/<branch> 的形式命名。 例如，如果你想要看你最后一次与远程仓库 origin 通信时 master 分支的状态，你可以查看 origin/master 分支。 你与同事合作解决一个问题并且他们推送了一个 iss53 分支，你可能有自己的本地 iss53 分支， 然而在服务器上的分支会以 origin/iss53 来表示。

这可能有一点儿难以理解，让我们来看一个例子。 假设你的网络里有一个在 git.ourcompany.com 的 Git 服务器。 如果你从这里克隆，Git 的 clone 命令会为你自动将其命名为 origin，拉取它的所有数据， 创建一个指向它的 master 分支的指针，并且在本地将其命名为 origin/master。 Git 也会给你一个与 origin 的 master 分支在指向同一个地方的本地 master 分支，这样你就有工作的基础。

|  |  |
| --- | --- |
| **Note** | **“origin” 并无特殊含义**  远程仓库名字 “origin” 与分支名字 “master” 一样，在 Git 中并没有任何特别的含义一样。 同时 “master” 是当你运行 git init 时默认的起始分支名字，原因仅仅是它的广泛使用， “origin” 是当你运行 git clone 时默认的远程仓库名字。 如果你运行 git clone -o booyah，那么你默认的远程分支名字将会是 booyah/master。 |

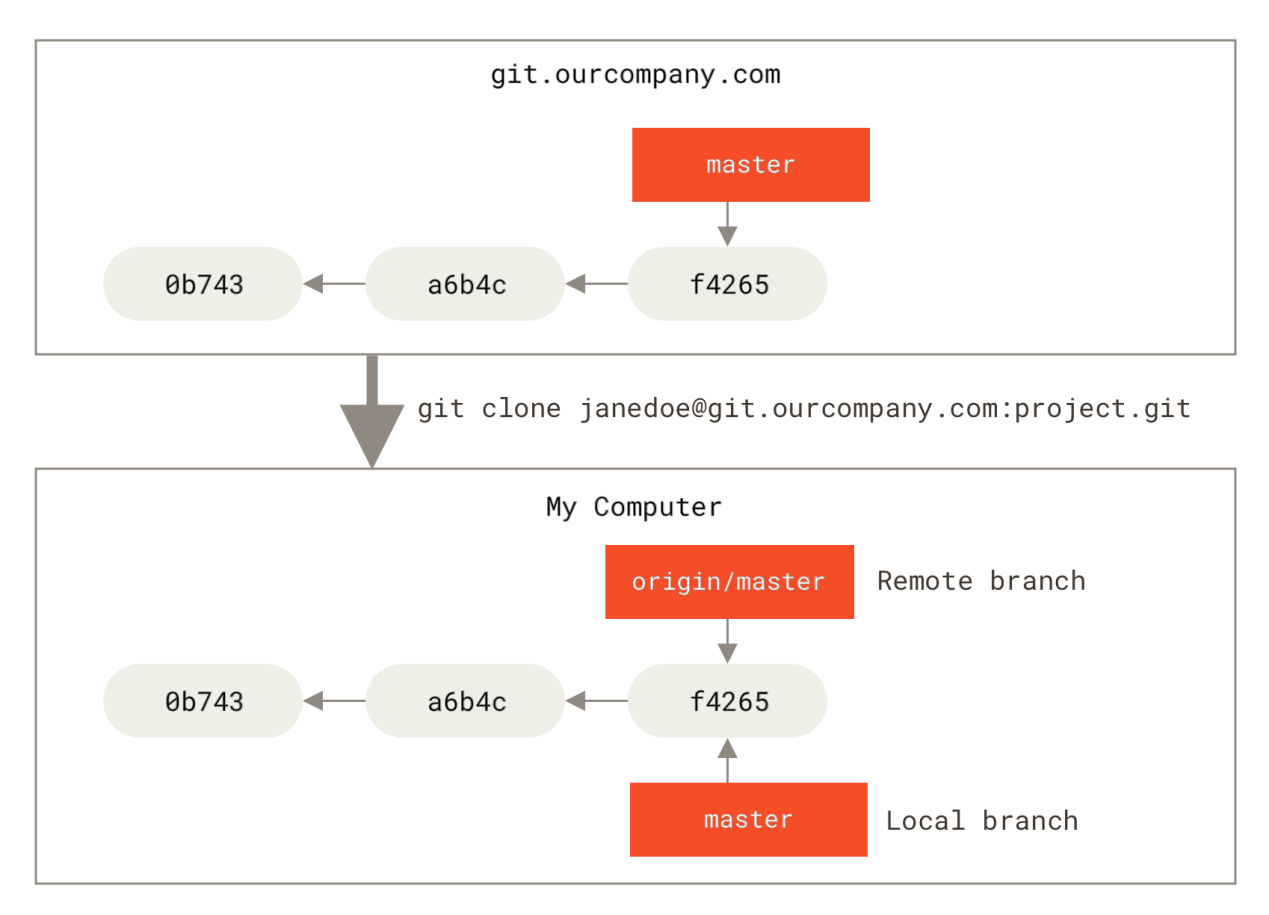


Figure 30. 克隆之后的服务器与本地仓库

如果你在本地的 master 分支做了一些工作，在同一段时间内有其他人推送提交到 git.ourcompany.com 并且更新了它的 master 分支，这就是说你们的提交历史已走向不同的方向。 即便这样，只要你保持不与 origin 服务器连接（并拉取数据），你的 origin/master 指针就不会移动。

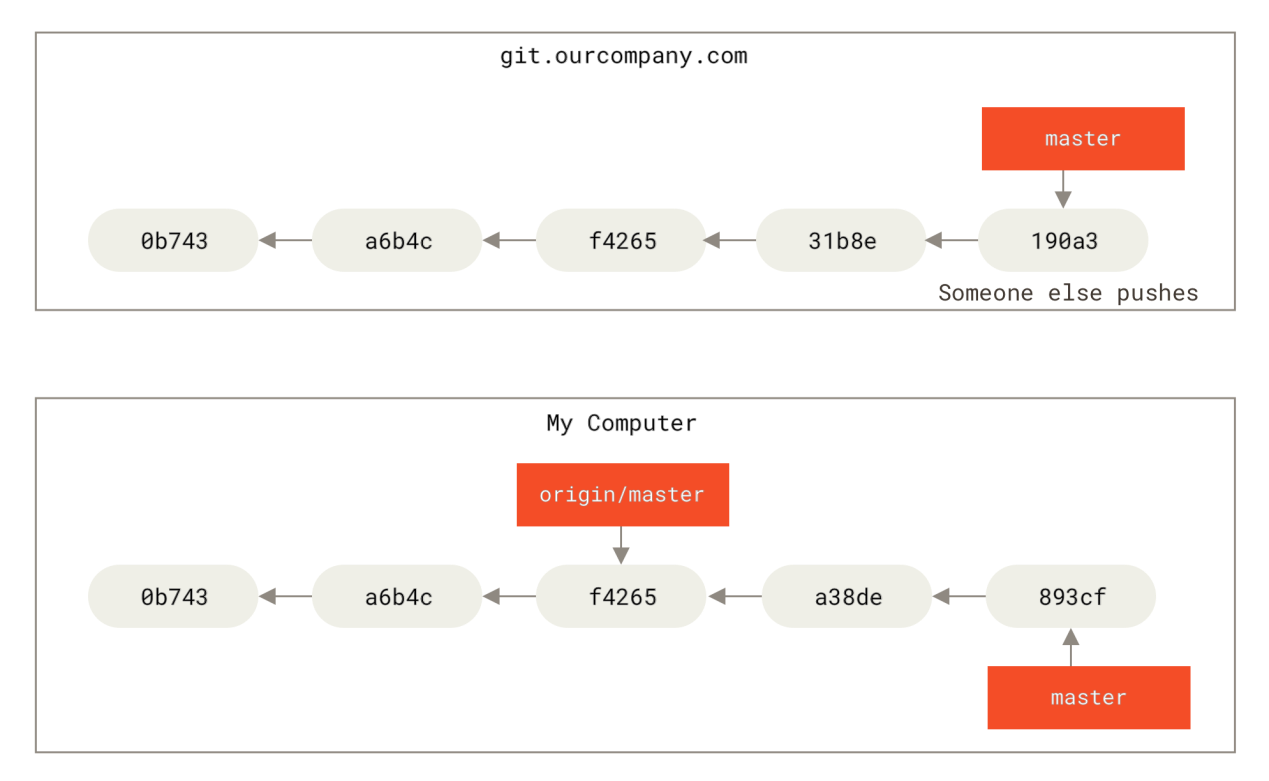


Figure 31. 本地与远程的工作可以分叉

如果要与给定的远程仓库同步数据，运行 git fetch <remote> 命令（在本例中为 git fetch origin）。 这个命令查找 “origin” 是哪一个服务器（在本例中，它是 git.ourcompany.com）， 从中抓取本地没有的数据，并且更新本地数据库，移动 origin/master 指针到更新之后的位置。

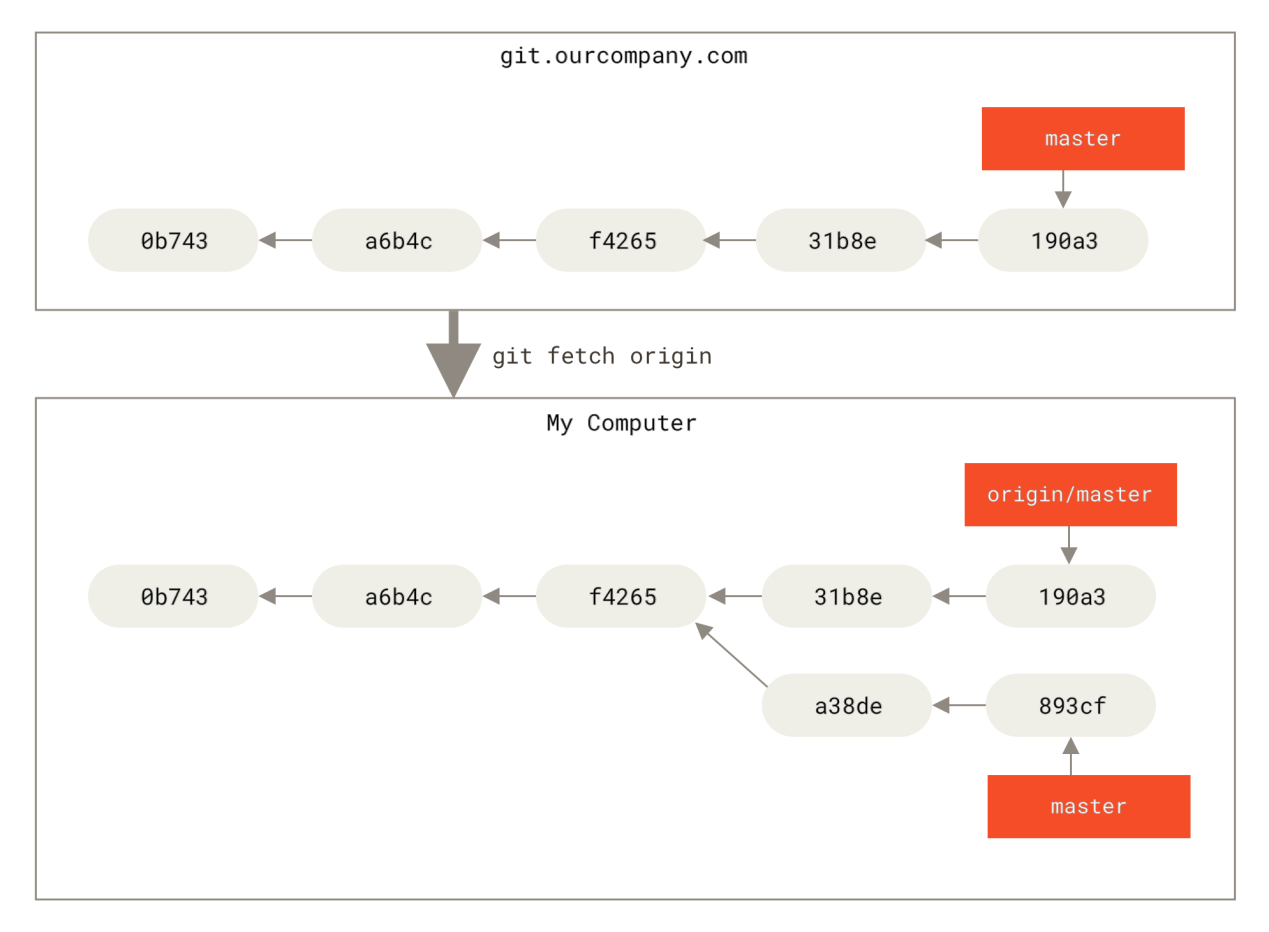


Figure 32. git fetch 更新你的远程跟踪分支

为了演示有多个远程仓库与远程分支的情况，我们假定你有另一个内部 Git 服务器，仅服务于你的某个敏捷开发团队。 这个服务器位于 git.team1.ourcompany.com。 你可以运行 git remote add 命令添加一个新的远程仓库引用到当前的项目，这个命令我们会在 [Git 基础](https://git-scm.com/book/zh/v2/ch00/ch02-git-basics-chapter) 中详细说明。 将这个远程仓库命名为 teamone，将其作为完整 URL 的缩写。

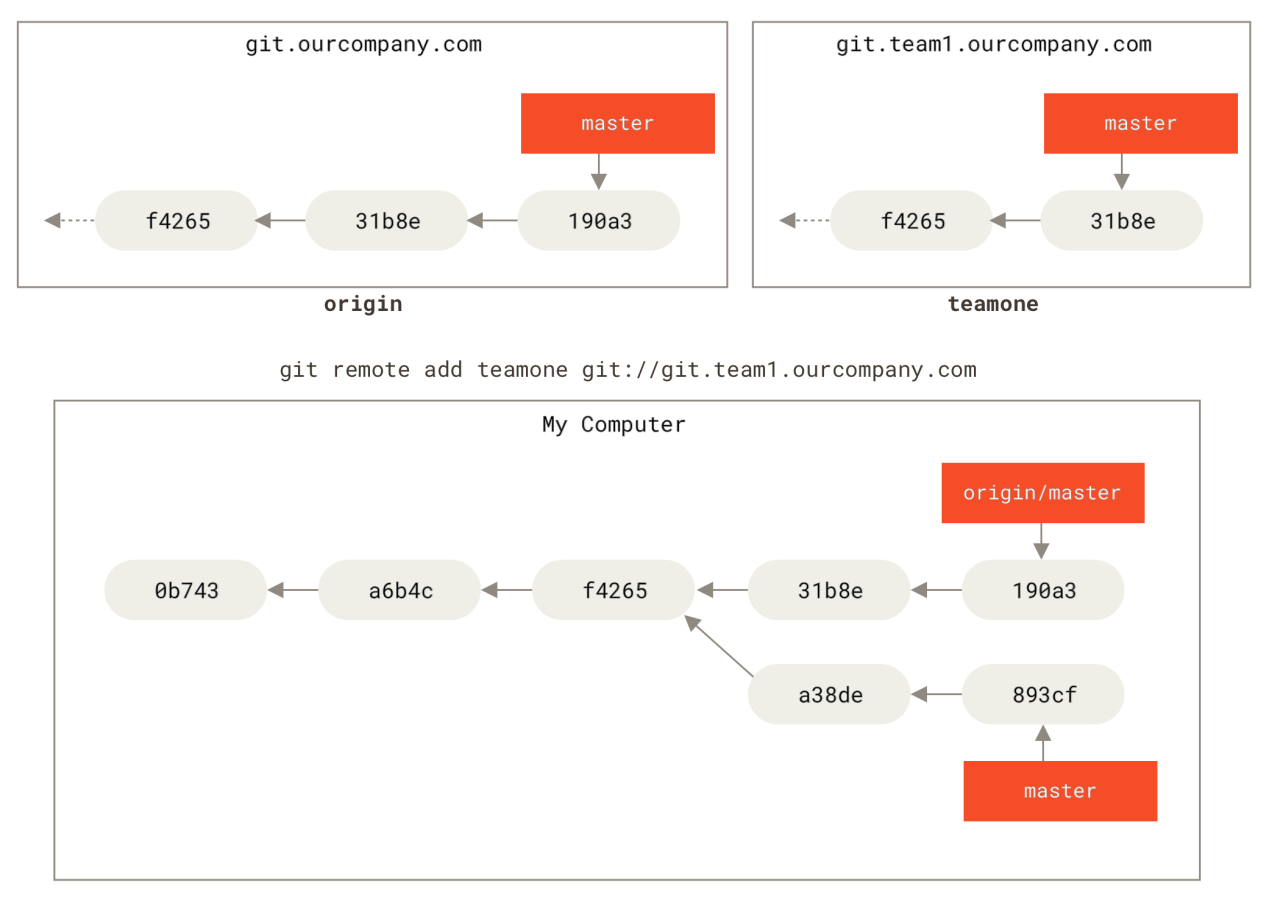


Figure 33. 添加另一个远程仓库

现在，可以运行 git fetch teamone 来抓取远程仓库 teamone 有而本地没有的数据。 因为那台服务器上现有的数据是 origin 服务器上的一个子集， 所以 Git 并不会抓取数据而是会设置远程跟踪分支 teamone/master 指向 teamone 的 master 分支。

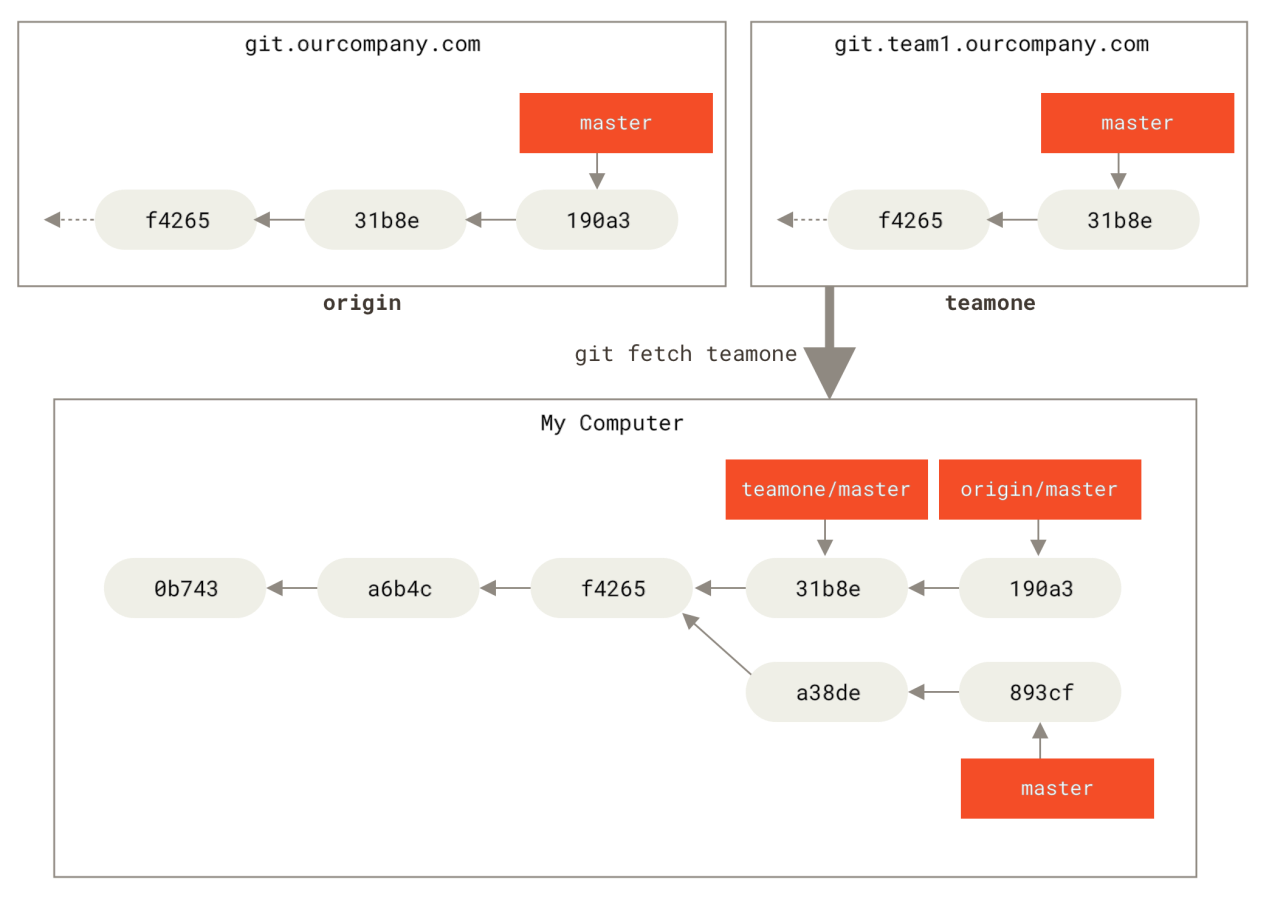


Figure 34. 远程跟踪分支 teamone/master

### **推送**

当你想要公开分享一个分支时，需要将其推送到有写入权限的远程仓库上。 本地的分支并不会自动与远程仓库同步——你必须显式地推送想要分享的分支。 这样，你就可以把不愿意分享的内容放到私人分支上，而将需要和别人协作的内容推送到公开分支。

如果希望和别人一起在名为 serverfix 的分支上工作，你可以像推送第一个分支那样推送它。 运行 git push <remote> <branch>:

$ git push origin serverfix

Counting objects: 24, done.

Delta compression using up to 8 threads.

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

Writing objects: 100% (24/24), 1.91 KiB | 0 bytes/s, done.

Total 24 (delta 2), reused 0 (delta 0)

To https://github.com/schacon/simplegit

\* [new branch] serverfix -> serverfix

这里有些工作被简化了。 Git 自动将 serverfix 分支名字展开为 refs/heads/serverfix:refs/heads/serverfix， 那意味着，“推送本地的 serverfix 分支来更新远程仓库上的 serverfix 分支。” 我们将会详细学习 [Git 内部原理](https://git-scm.com/book/zh/v2/ch00/ch10-git-internals) 的 refs/heads/ 部分， 但是现在可以先把它放在儿。你也可以运行 git push origin serverfix:serverfix， 它会做同样的事——也就是说“推送本地的 serverfix 分支，将其作为远程仓库的 serverfix 分支” 可以通过这种格式来推送本地分支到一个命名不相同的远程分支。 如果并不想让远程仓库上的分支叫做 serverfix，可以运行 git push origin serverfix:awesomebranch 来将本地的 serverfix 分支推送到远程仓库上的 awesomebranch 分支。

|  |  |
| --- | --- |
| **Note** | **如何避免每次输入密码**  如果你正在使用 HTTPS URL 来推送，Git 服务器会询问用户名与密码。 默认情况下它会在终端中提示服务器是否允许你进行推送。  如果不想在每一次推送时都输入用户名与密码，你可以设置一个 “credential cache”。 最简单的方式就是将其保存在内存中几分钟，可以简单地运行 git config --global credential.helper cache 来设置它。  想要了解更多关于不同验证缓存的可用选项，查看 [凭证存储](https://git-scm.com/book/zh/v2/ch00/_credential_caching)。 |

下一次其他协作者从服务器上抓取数据时，他们会在本地生成一个远程分支 origin/serverfix，指向服务器的 serverfix 分支的引用：

$ git fetch origin

remote: Counting objects: 7, done.

remote: Compressing objects: 100% (2/2), done.

remote: Total 3 (delta 0), reused 3 (delta 0)

Unpacking objects: 100% (3/3), done.

From https://github.com/schacon/simplegit

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

要特别注意的一点是当抓取到新的远程跟踪分支时，本地不会自动生成一份可编辑的副本（拷贝）。 换一句话说，这种情况下，不会有一个新的 serverfix 分支——只有一个不可以修改的 origin/serverfix 指针。

可以运行 git merge origin/serverfix 将这些工作合并到当前所在的分支。 如果想要在自己的 serverfix 分支上工作，可以将其建立在远程跟踪分支之上：

$ git checkout -b serverfix origin/serverfix

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

Switched to a new branch 'serverfix'

这会给你一个用于工作的本地分支，并且起点位于 origin/serverfix。

### **跟踪分支**

从一个远程跟踪分支检出一个本地分支会自动创建所谓的“跟踪分支”（它跟踪的分支叫做“上游分支”）。 跟踪分支是与远程分支有直接关系的本地分支。 如果在一个跟踪分支上输入 git pull，Git 能自动地识别去哪个服务器上抓取、合并到哪个分支。

当克隆一个仓库时，它通常会自动地创建一个跟踪 origin/master 的 master 分支。 然而，如果你愿意的话可以设置其他的跟踪分支，或是一个在其他远程仓库上的跟踪分支，又或者不跟踪 master 分支。 最简单的实例就是像之前看到的那样，运行 git checkout -b <branch> <remote>/<branch>。 这是一个十分常用的操作所以 Git 提供了 --track 快捷方式：

$ git checkout --track origin/serverfix

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

Switched to a new branch 'serverfix'

由于这个操作太常用了，该捷径本身还有一个捷径。 如果你尝试检出的分支 (a) 不存在且 (b) 刚好只有一个名字与之匹配的远程分支，那么 Git 就会为你创建一个跟踪分支：

$ git checkout serverfix

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

Switched to a new branch 'serverfix'

如果想要将本地分支与远程分支设置为不同的名字，你可以轻松地使用上一个命令增加一个不同名字的本地分支：

$ 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，本地分支 sf 会自动从 origin/serverfix 拉取。

设置已有的本地分支跟踪一个刚刚拉取下来的远程分支，或者想要修改正在跟踪的上游分支， 你可以在任意时间使用 -u 或 --set-upstream-to 选项运行 git branch 来显式地设置。

$ git branch -u origin/serverfix

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

|  |  |
| --- | --- |
| **Note** | **上游快捷方式**  当设置好跟踪分支后，可以通过简写 @{upstream} 或 @{u} 来引用它的上游分支。 所以在 master 分支时并且它正在跟踪 origin/master 时，如果愿意的话可以使用 git merge @{u} 来取代 git merge origin/master。 |

如果想要查看设置的所有跟踪分支，可以使用 git branch 的 -vv 选项。 这会将所有的本地分支列出来并且包含更多的信息，如每一个分支正在跟踪哪个远程分支与本地分支是否是领先、落后或是都有。

$ git branch -vv

iss53 7e424c3 [origin/iss53: ahead 2] forgot the brackets

master 1ae2a45 [origin/master] deploying index fix

\* serverfix f8674d9 [teamone/server-fix-good: ahead 3, behind 1] this should do it

testing 5ea463a trying something new

这里可以看到 iss53 分支正在跟踪 origin/iss53 并且 “ahead” 是 2，意味着本地有两个提交还没有推送到服务器上。 也能看到 master 分支正在跟踪 origin/master 分支并且是最新的。 接下来可以看到 serverfix 分支正在跟踪 teamone 服务器上的 server-fix-good 分支并且领先 3 落后 1， 意味着服务器上有一次提交还没有合并入同时本地有三次提交还没有推送。 最后看到 testing 分支并没有跟踪任何远程分支。

需要重点注意的一点是这些数字的值来自于你从每个服务器上最后一次抓取的数据。 这个命令并没有连接服务器，它只会告诉你关于本地缓存的服务器数据。 如果想要统计最新的领先与落后数字，需要在运行此命令前抓取所有的远程仓库。 可以像这样做：

$ git fetch --all; git branch -vv

### **拉取**

当 git fetch 命令从服务器上抓取本地没有的数据时，它并不会修改工作目录中的内容。 它只会获取数据然后让你自己合并。 然而，有一个命令叫作 git pull 在大多数情况下它的含义是一个 git fetch 紧接着一个 git merge 命令。 如果有一个像之前章节中演示的设置好的跟踪分支，不管它是显式地设置还是通过 clone 或 checkout 命令为你创建的，git pull 都会查找当前分支所跟踪的服务器与分支， 从服务器上抓取数据然后尝试合并入那个远程分支。

由于 git pull 的魔法经常令人困惑所以通常单独显式地使用 fetch 与 merge 命令会更好一些。

### **删除远程分支**

假设你已经通过远程分支做完所有的工作了——也就是说你和你的协作者已经完成了一个特性， 并且将其合并到了远程仓库的 master 分支（或任何其他稳定代码分支）。 可以运行带有 --delete 选项的 git push 命令来删除一个远程分支。 如果想要从服务器上删除 serverfix 分支，运行下面的命令：

$ git push origin --delete serverfix

To https://github.com/schacon/simplegit

- [deleted] serverfix

基本上这个命令做的只是从服务器上移除这个指针。 Git 服务器通常会保留数据一段时间直到垃圾回收运行，所以如果不小心删除掉了，通常是很容易恢复的。

15.git 生成pub key添加到git服务器