

在应用程序开发过程中，很重要的一部分工作就是如何进行源码的版本控制。当代码出现问题时，我们就需要将代码恢复到原先正常的版本。如果是多个人共同开发一个项目，那么代码的控制就会非常复杂。幸运的是，开发者不需要自己控制这些，因为有专门的软件来负责，叫做版本控制系统。

版本控制系统，或者说修改控制系统，实际上是一种检测源文件的改变并将其保存留作以后参考使用的机制（软件）。此外，它还能记录其他有用信息，比如是哪个开发者修改了代码，何时修改的，修改了哪一部分，以及其他历史信息。版本控制系统可以比较不同版本代码的不同，必要时能恢复整个项目到以前的版本，追踪有害代码从而减少产品的错误。

通过版本控制系统，开发者可以在一个项目的不同分支上工作，当项目的各个部分开发完备时，将它们放到一起形成最终的版本，这个过程被称为合并。事实上，这种做法再团队和软件公司中相当常见：每个人负责项目的一部分，最终所有部分被整合到一起形成最终产品。

对于个人开发者来说，版本控制系统并不是必需的，但是我们仍然强烈推荐开发者使用它，因为它可以使代码方便的在有错误的版本和可以工作的版本之间转换。事实上，很多开发者从来不使用类似的工具，他们会在项目添加新的功能时手动保存原先的项目。这其实是一个很不好的习惯，因为版本控制软件可以更好更高效地完成这项任务。

Git是一个常见的版本控制系统，它最开始是由Linux之父Linus Torvalds开发的，Git使用虚拟目录，又称为repositories，来管理一切事物。Git可以通过命令行调用，也有专门为它设计的桌面应用软件。如果Git对你来说很陌生，我建议你在网上查看一些它的相关信息。关于Git更深层次的内容都不在本文的讨论范围之内。

从Xcode5开始引入了使用git的一些新特性。它将git的各项功能整合到一个菜单中，并提供子菜单来进行软件合并的控制。在接下来的阅读中你会发现，使用git来进行版本控制相当的简单快捷。

我们接下来的任务就是学习如何在Xcode中使用git，以及Xcode是如何整合Git的各项功能。如果你觉得对这些很陌生，我建议你先上网搜索一下相关的内容。在接下来的教程中，我会假定你已经了解了版本控制系统和git是什么，并将注意力集中在Xcode如何管理它上。

GIT Demo概述 (GIT Demo Overview)

与其他教程中的demo app不同，这次我们不会去实现一个应用来演示某一项iOS SDK特性，最终我们也不会产生一个示例产品。实际上，我们会新建一个demo工程，写几行代码，然后利用这个工程来演示Xcode提供的版本管理功能。换句话说，我们会集中注意里于IDE上，而不是iOS本身。

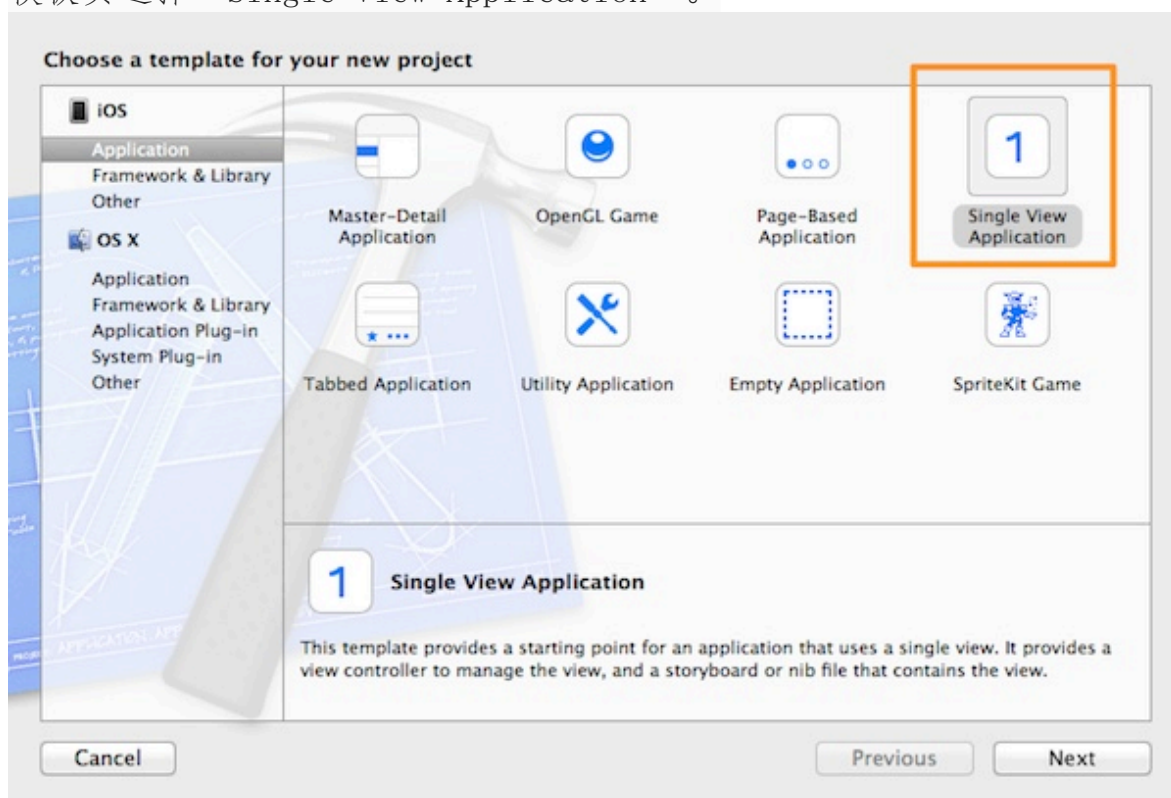
我建议你跟着我一起一步一步实现这个实例项目，在相应的地方手动添加代码，不用担心，代码量不是很多。跟着教程的步骤，我们将执行多种重复的版本控制相关的操作，并且我们必须实时看到结果。如果我只是提供了一个具备所有操作的的应用，那么你就无法体会到这些改变。

好了，废话不多说了，让我们仔细看看使用Xcode进行版本控制的要点吧。

创建一个Git源 (Creating a Git repository)

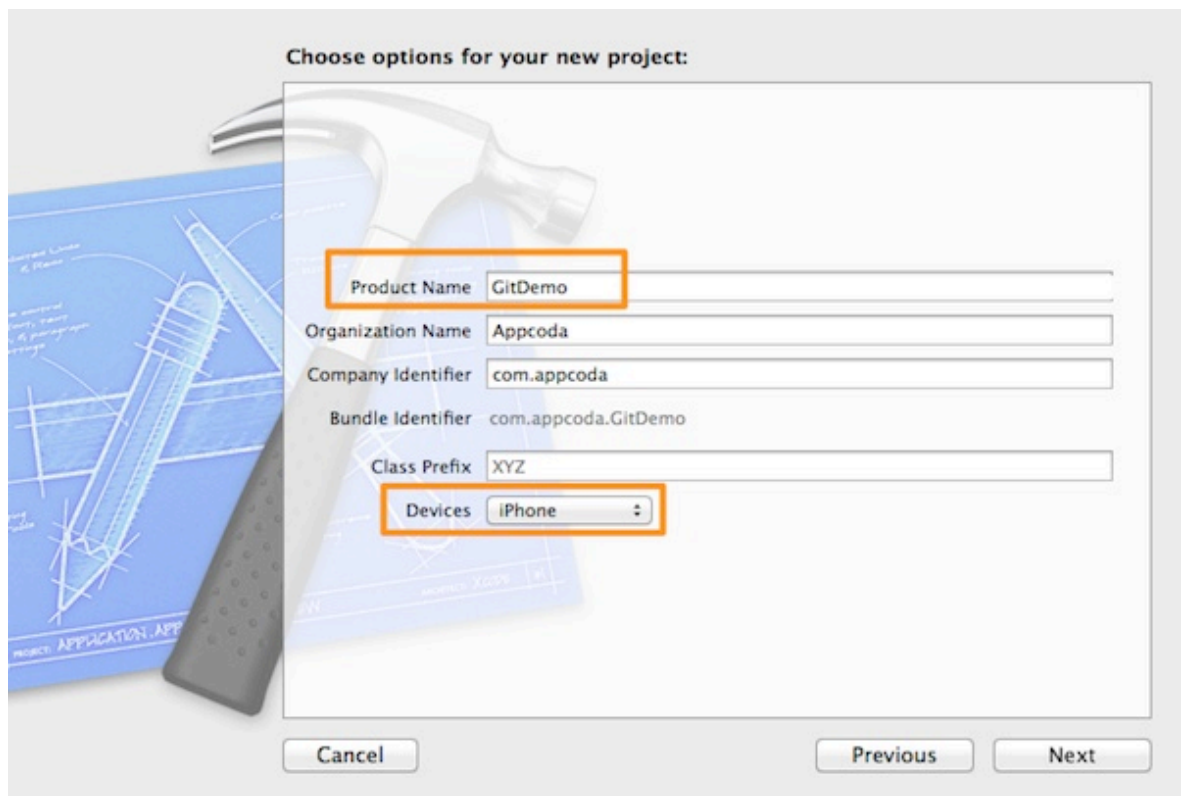
每次在Xcode中创建新工程的时候，都会提示开发者是否将项目作为一个本地的git源。在创建工程的最后一步Xcode会有一个复选框，如果选择了它，git源就会被添加到工程目录中。通常这个选项会被忽视，或是被认为是Xcode的另外一个没用的功能，尤其是从未用过git的开发者，或是编程新手。

打开Xcode，创建一个新的工程。选择iOS区的“Application”，在应用模板页选择“Single View Application”。



选择下一步，在项目名中输入GitDemo，确保下面的Devices菜单选择iPhone，无需iPad或者universal app。

Choose options for your new project:



Product Name

Organization Name

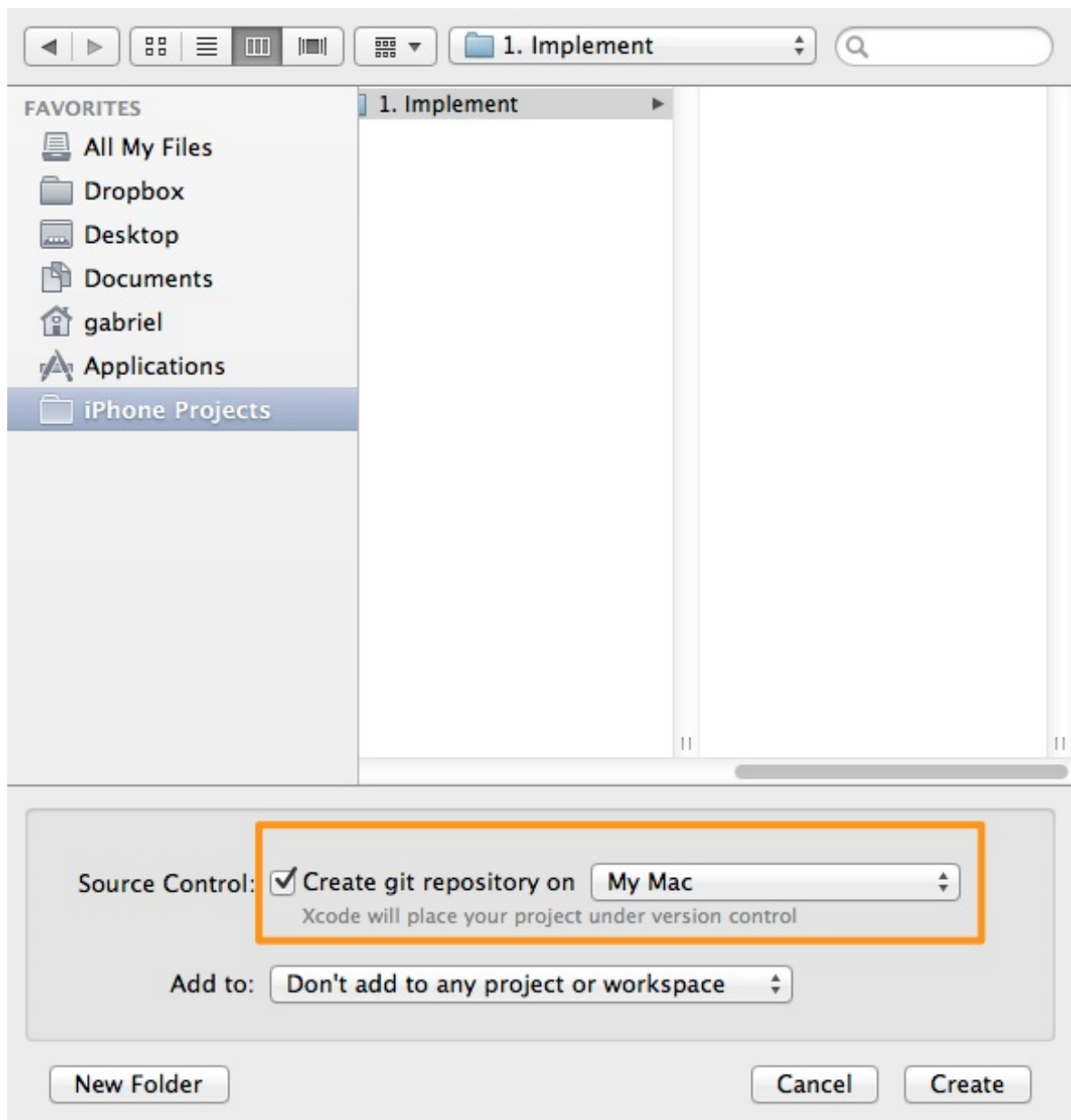
Company Identifier

Bundle Identifier

Class Prefix

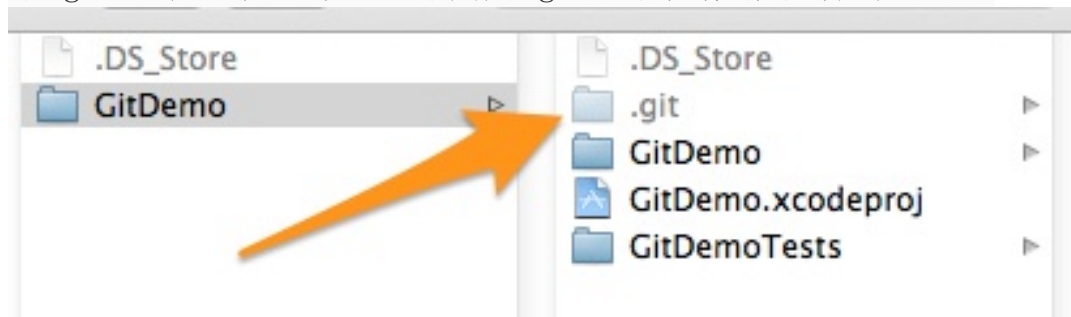
Devices

点击下一步，也就是最后一个步骤，在这里先选择一个要保持工程的目录，然后在窗口底部选上Create git repository on (My Mac):



默认情况下，这个选项是被选上的，如果你不想使用git，你可以取消它，但是我不建议这么做。本教程中，你需要将它勾选上，然后点击创建按钮。

创建完项目之后，打开Finder，找到项目存储的目录，在目录中，有一个.git的子目录，是Xcode为存储git源相关数据自动创建的。



如果你看不到.git目录，你需要让隐藏的文件可见。具体做法就是打开一

个Terminal窗口，输入以下命令：

对于OS X Mavericks 10.9:

```
1 defaults write com.apple.finder AppleShowAllFiles TRUE
```

对于以前的OS X版本，

```
1 defaults write com.apple.Finder AppleShowAllFiles TRUE
```

为了重启Finder应用，输入

```
1 killall Finder
```

这就是本项目在本地git源保存的位置。实际上，如果你选上了相应的选项，这个目录就会被创建。相应地，在你创建新应用时，.git子目录也会一同被创建。

显然使用Xcode创建一个git源毫不费力，然而，如果你在项目创建时未创建git源，之后又想加上这个功能怎么办呢？好吧，其实你可以在任何时候为你的项目创建源，但是不是使用Xcode。尽管这种情况很少发生，我还是会告诉你该怎么做。

如果你愿意的话，你可以直接跳到本教程的下一部分。我建议你接着读下去，因为接下来这些信息还是很有用的。

在进行演示前，你需要首先通过Xcode下载Command Line Tools，因为我们要在Terminal下操作，并且需要一些额外的工具。如果你还没有下载，那就去Xcode>Preferences…菜单，选择Download选项卡，展开Components区，点击Command Line Tools右边下载按钮。下载完成后，一个对勾符号会取代下载按钮。



现在，为这个例子再创建一个工程，完事后可以删了它。在创建时取消那个创建git源的选项。这次我们不想让Xcode为我们准备一个源。把这个工程命名为NoGitExample，保存到桌面，然后你可以跟我接下来输入的命令一样。

一切准备妥当后，打开Terminal窗口（如果你之前打开了一个，那就先关掉它再重启，从而使我们安装的命令行工具生效）。下面切换到新项目的

目录:

```
1 cd /Users/YOUR-USERNAME/Desktop/NoGitExample
```

别忘了在上边命令中设置Mac的用户名, 接下来, 输入:

```
1 git init
```

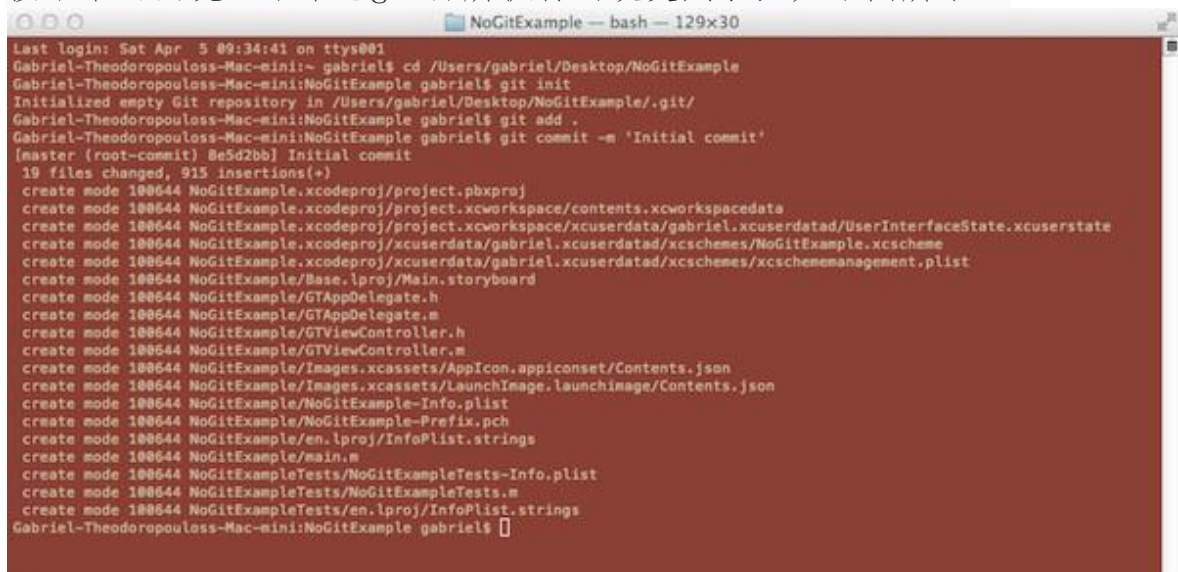
这会初始化一个空的源, 如果你在Finder里面查看或是输入ls命令, 你会看到.git子目录已经被创建, 很好, 接下来输入:

```
1 git add .
```

这样, 当前目录所有的内容就被添加到源里面去了, 最后, 输入以下命令:

```
1 git commit -m 'Initial commit'
```

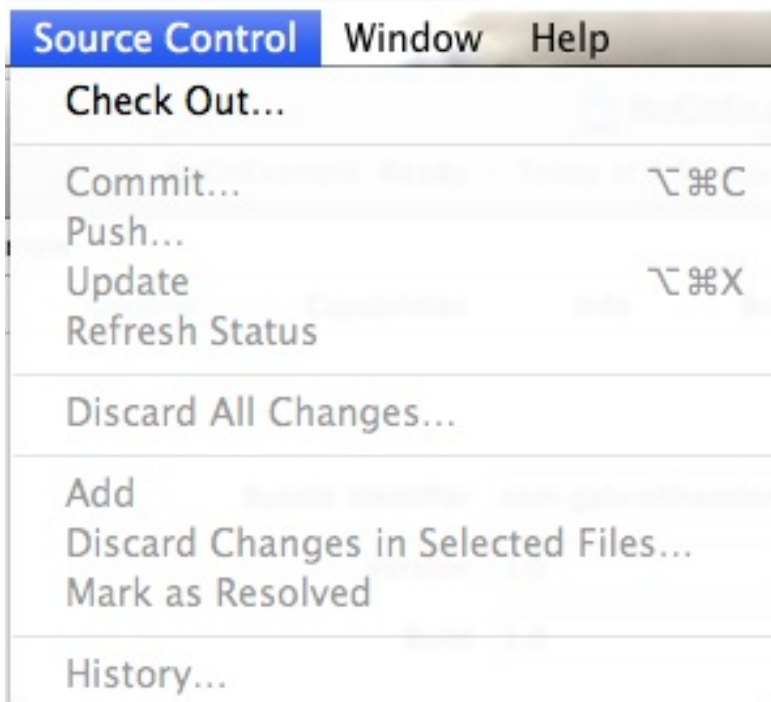
接下来会出现一个本地git源所执行的改变列表, 如下图所示:



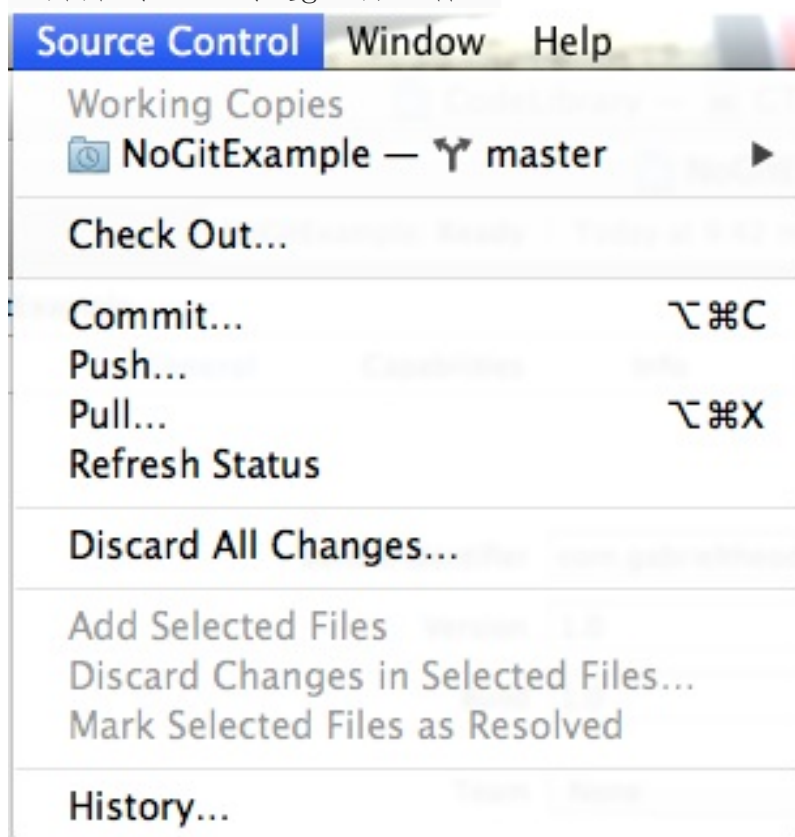
```

Last login: Sat Apr 5 09:34:41 on ttys001
Gabriel-Theodoropoulos-Mac-mini:~ gabriel$ cd /Users/gabriel/Desktop/NoGitExample
Gabriel-Theodoropoulos-Mac-mini:NoGitExample gabriel$ git init
Initialized empty Git repository in /Users/gabriel/Desktop/NoGitExample/.git/
Gabriel-Theodoropoulos-Mac-mini:NoGitExample gabriel$ git add .
Gabriel-Theodoropoulos-Mac-mini:NoGitExample gabriel$ git commit -m 'Initial commit'
[master (root-commit) 8e5d2bb] Initial commit
19 files changed, 915 insertions(+)
create mode 100644 NoGitExample.xcodeproj/project.pbxproj
create mode 100644 NoGitExample.xcodeproj/project.xcworkspace/contents.xcworkspacedata
create mode 100644 NoGitExample.xcodeproj/project.xcworkspace/xcuserdata/gabriel.xcuserdatad/UserInterfaceState.xcuserstate
create mode 100644 NoGitExample.xcodeproj/xcuserdata/gabriel.xcuserdatad/xcschemes/NoGitExample.xcscheme
create mode 100644 NoGitExample.xcodeproj/xcuserdata/gabriel.xcuserdatad/xcschemes/xcschememanagement.plist
create mode 100644 NoGitExample/Base.lproj/Main.storyboard
create mode 100644 NoGitExample/GTAppDelegate.h
create mode 100644 NoGitExample/GTAppDelegate.m
create mode 100644 NoGitExample/GTViewController.h
create mode 100644 NoGitExample/GTViewController.m
create mode 100644 NoGitExample/Images.xcassets/AppIcon.appiconset/Contents.json
create mode 100644 NoGitExample/Images.xcassets/LaunchImage.launchimage/Contents.json
create mode 100644 NoGitExample/NoGitExample-Info.plist
create mode 100644 NoGitExample/NoGitExample-Prefix.pch
create mode 100644 NoGitExample/en.lproj/InfoPlist.strings
create mode 100644 NoGitExample/main.m
create mode 100644 NoGitExampleTests/NoGitExampleTests-Info.plist
create mode 100644 NoGitExampleTests/NoGitExampleTests.m
create mode 100644 NoGitExampleTests/en.lproj/InfoPlist.strings
Gabriel-Theodoropoulos-Mac-mini:NoGitExample gabriel$
```

现在git源就建好了, 但是如果你回到Xcode, 打开Source Control菜单, 你会发现一切仍然是被禁用。



这是因为当我们使用命令行工具创建git源时，Xcode并未被通知，下面点击Xcode>Quit Xcode，然后重新启动它，在NoGitExample项目中，如果你再次打开Source Control菜单，你会发现所有的选项已经被使能了，就像一开始勾选上创建git源一样。



现在这个项目的使命已经结束，你可以在桌面上删除它。

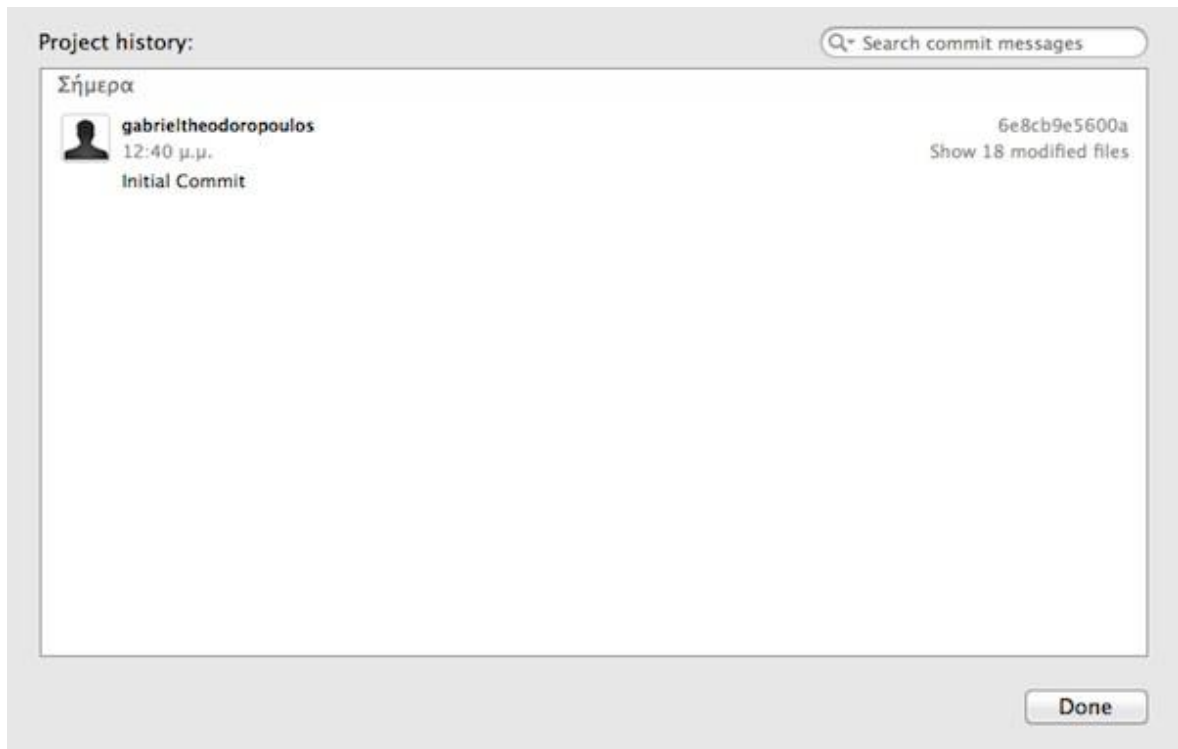
现在你知道如何为你所有的项目添加git源了，即使你在创建时没有添加，你也可以在以后任何时候为它手动添加源。

提交更改 (Committing Changes)

提交更改指的是储存一个包含所有更改的新版本。一般来说，当我们做了一些有意义的工作，并且项目处于某一个稳定状态时，就可以提交一次更改。然而具体什么时候提交更改并没有硬性的规定。我的建议是：从上次提交更改之后，如果你怕花费大量时间和精力做的新工作被误删很难恢复，你就需要提交更改了。

默认情况下，Xcode在项目创建之初会提交一次更改，这是为了保存项目初始状态。这项工作会在后台完成，不会打扰你或者要求你进行确认。如果你在项目创建时没有添加git源，但是之后你手动添加了，你可以通过我们先前使用过的命令来进行提交：`git commit -m 'Initial commit'`

实际上，你如果去Source Control>History...菜单，你就会看到初次提交更改的记录，以后每次提交更改，都会在这里有所记录。



接下来让我们小幅修改一下我们的工程，在ViewController.m文件中，添加以下属性声明：

```
1  @interface ViewController ()
2
3  @property (nonatomic) int sum;
4
5  @end
```

接下来，像下面这样修改viewDidLoad方法：

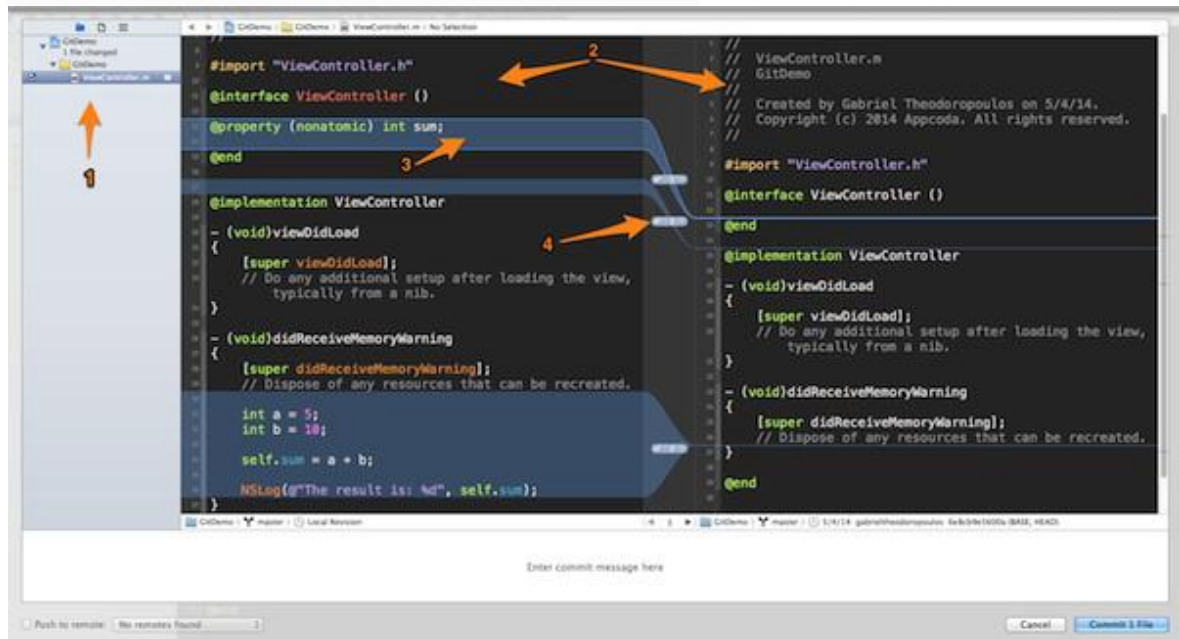
```
1  - (void) didReceiveMemoryWarning
2  {
3      [super didReceiveMemoryWarning];
4      // Dispose of any resources that can be recreated.
5
6      int a = 5;
7      int b = 10;
8
9      self.sum = a + b;
10
11     NSLog("The result is: %d", self.sum);
12 }
```

看一下Project navigator面板，你会发现现在ViewController.m文件旁边，添加了一个M字母，像下面这样：



这意味着那个文件已经被修改，相比上一次提交更改，文件有所改变。一般来说，你每次改变文件，都会出现这个M字母，提醒你有未提交的更改。

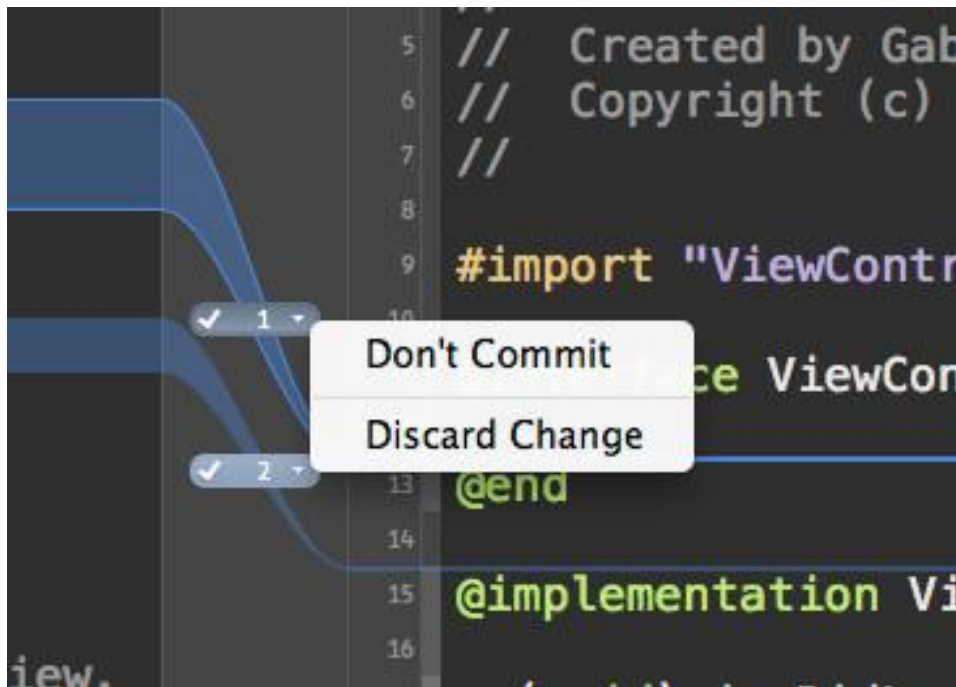
下面看看如何提交更改，其实非常简单，只需要打开Source Control>Commit菜单，下面窗口就会出现：



让我们一步步看看它告诉了我们了什么。在左边（标1的区域），列出了所有被更改的文件，在这个例子中，只有ViewController.m这个文件被改变，因此列表中只有它被显示。如果你仔细观察，你会发现文件左边有一个选择框，默认情况下是被选中的，如果你取消它，这个文件的更改就不会被提交。

在窗口的中间区域，有两个预览窗口，左边那个是文件当前版本，右边是文件上一次提交更改的版本。因为我们目前只是创建时提交过一次更改，因此右边显示的是文件的初始状态。

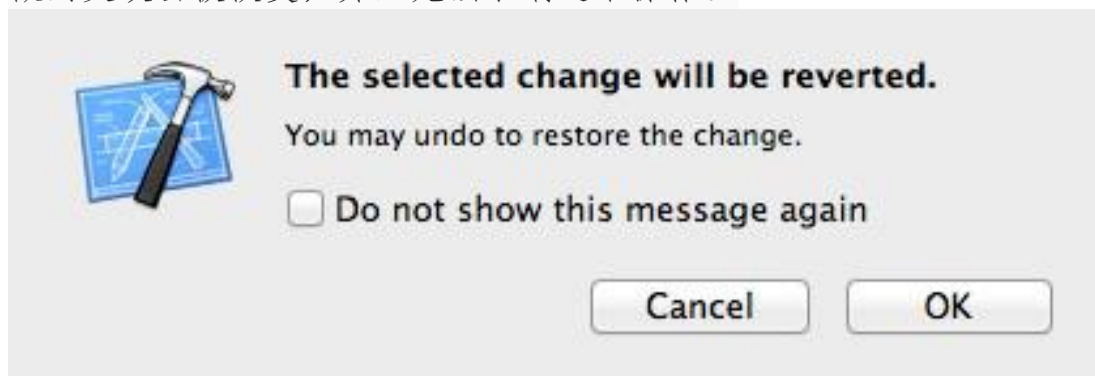
左边窗口蓝色区域标出的就是更改的内容，这样的表示让我们可以清楚地看出所有的修改。如果你仔细看，会发现在两个窗口之间还有一个带数字的小标签，这个数字一一表示了各项更改。在数字旁边，默认情况下有一个小对勾，表示本更改会被提交，如果你点击右边的小箭头，会弹出一个选项菜单，你可以选择不提交这个更改或是忽略它。



如果你选择了Don't Commit这个选项，小对勾就会被一个停止标志取代，这项更改就不会被保存到源中。



如果你选择了Discard Change这个选项，会弹出一个确认窗口，提示你所做的更改会被恢复，并且无法取消这个操作。

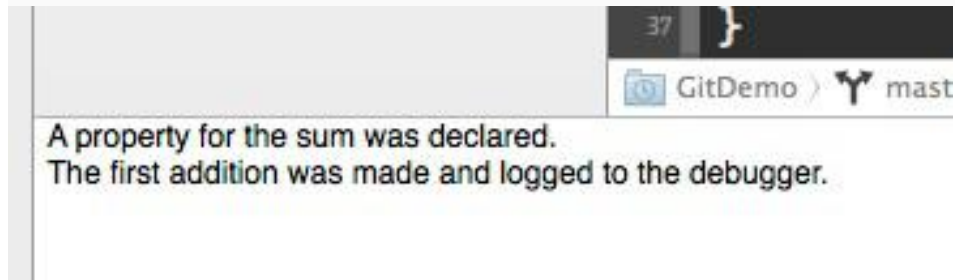


如果你点击了OK按钮，所选区域的改变就会消失，就像他们从未出现过一样。

如果你仔细观察上面这个提交窗口，你会看到你所做的所有修改都会被Xcode看做改变，即使是一个空行。实际上空行相当于回车，在屏幕上是不可见的，因此作为改变也是理所当然的。

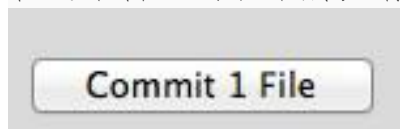
在本例子中，你不用忽略任何修改，而是允许提交所有更改，因此所有的改变标签旁边必须都是小对勾。

在两个窗口下面是一个空白的区域，中间显示了提交更改的信息。这个地方可以添加一些关于此次更改的简短描述，点击它，加入如下内容：

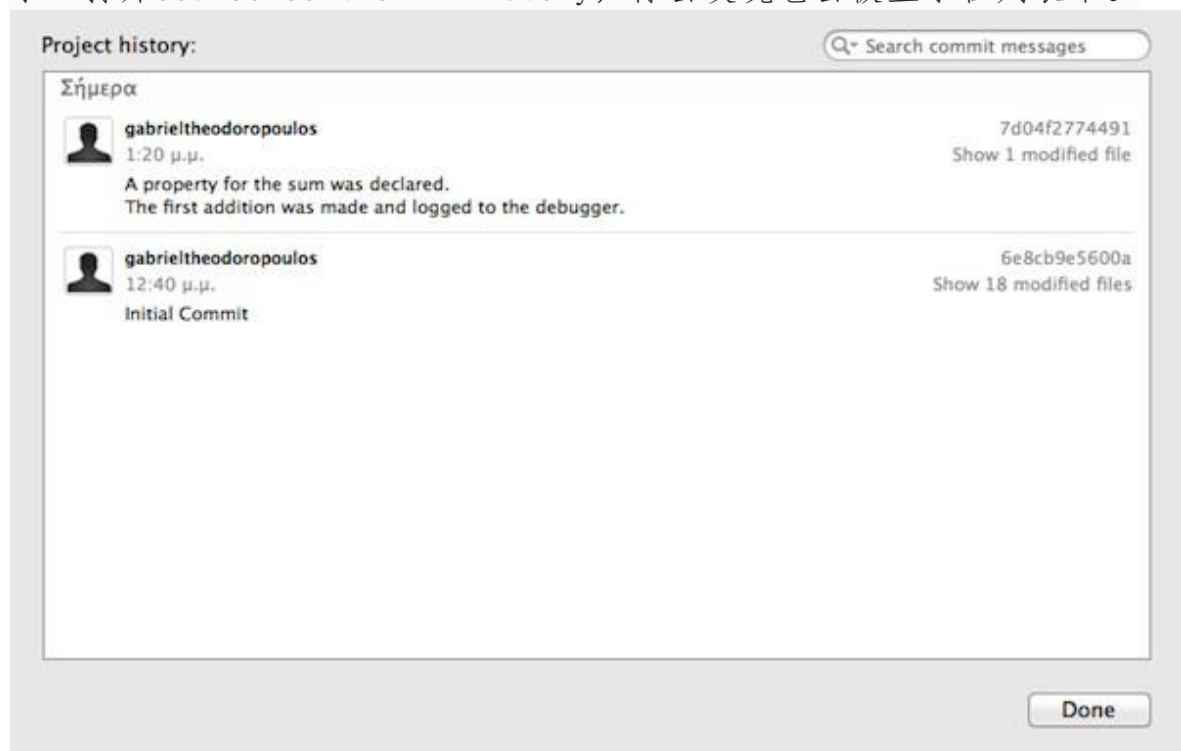


书写有意义的提交信息非常有用，尤其是当你频繁提交的时候。因此，把它当做一个必要的步骤。

现在这个窗口的基本信息看的差不多了，是时候做我们第一次的提交了。在这个窗口的右下角，有一个按钮上面写着：Commit 1 file。



这个按钮会显示需要提交的文件总数。点击它之后你的第一次提交就完成了！打开Source control > History，你会发现它会被显示在列表中。



从上图中可以看出，我们编写的信息以及更改的文件数量会被显示出来。Xcode执行初始提交，所有文件都会被提交一下，而这次只有我们修改的那个文件被提交。

另外，关闭历史窗口，看一下Project Navigator，你会发现ViewController.m旁边的M符号已经消失了。

现在，让我们准备下一次提交。这次，我们给工程添加一些新的文件。添加文件最好的方式就是创建个新类，因此，按下Command+N组合键，添加一个Objective-C类。让这个类继承NSObject类，取名叫TestClass，然后添加到工程中。

完成之后，注意一下Project Navigator，你会发现两个新的类文件旁边有个A的字母标识，这意味着这些文件已经被添加到项目中，当然，他们还没有被提交。

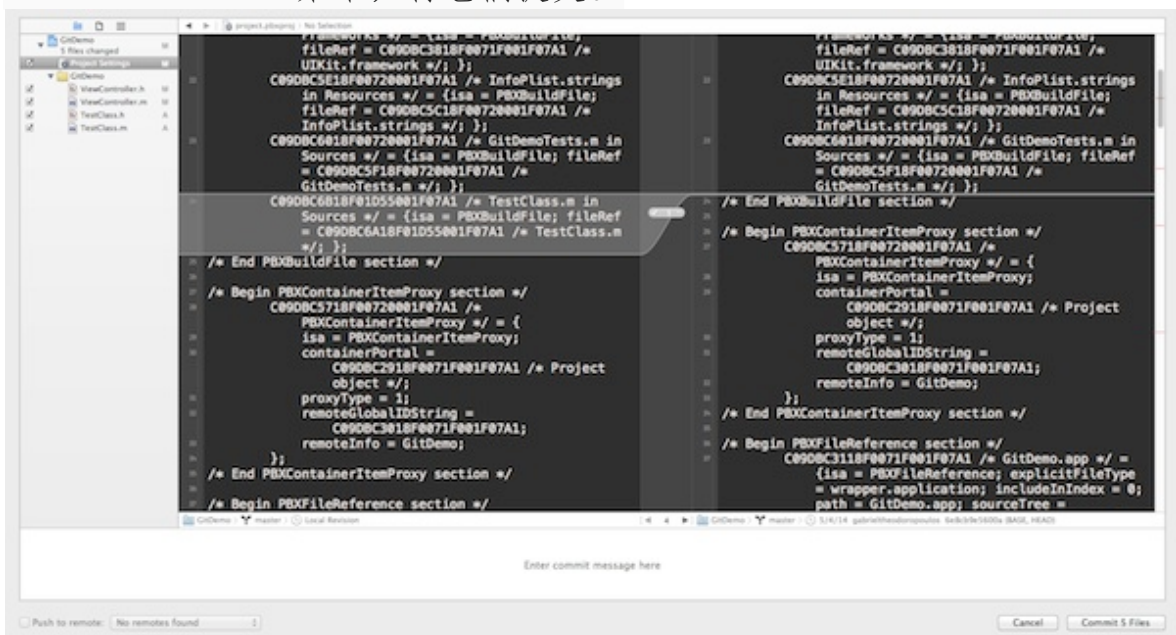
打开ViewController.h文件，导入我们的新类：

```
1 #import "TestClass.h"
```

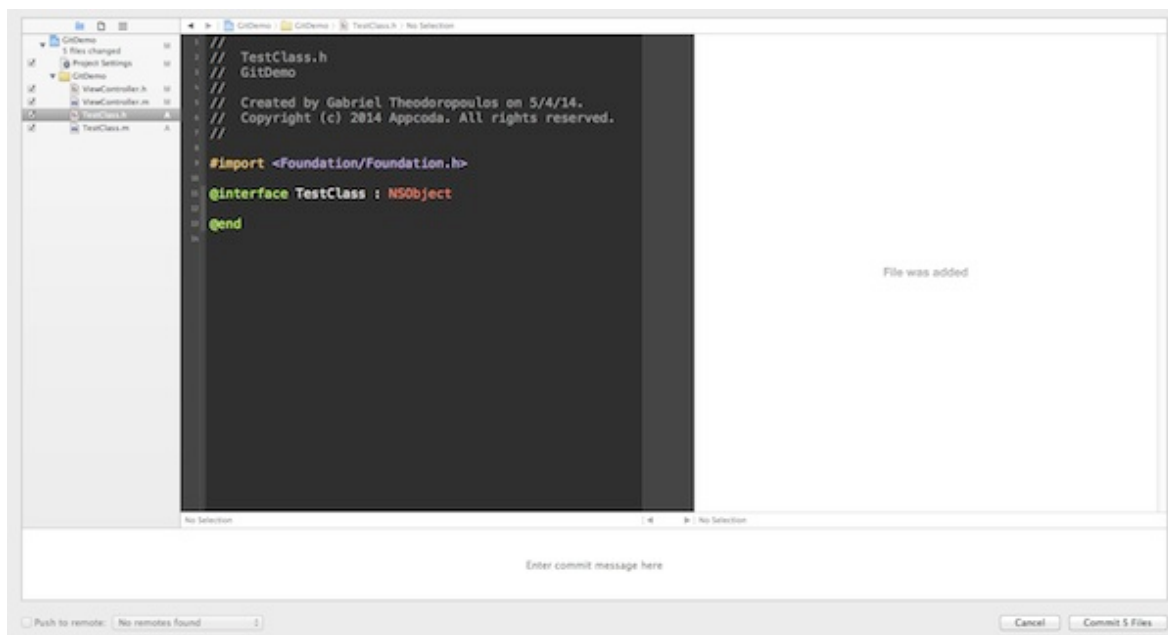
下一步，打开ViewController.m文件，像下面一样声明一个私有属性：

```
1 @interface ViewController ()
2
3 @property (nonatomic) int sum;
4
5 @property (nonatomic, strong) TestClass *testClass;
6
7 @end
```

看一下项目导航栏，这次有四个文件有待提交。让我们打开Source Control > Commit菜单，将它们提交。



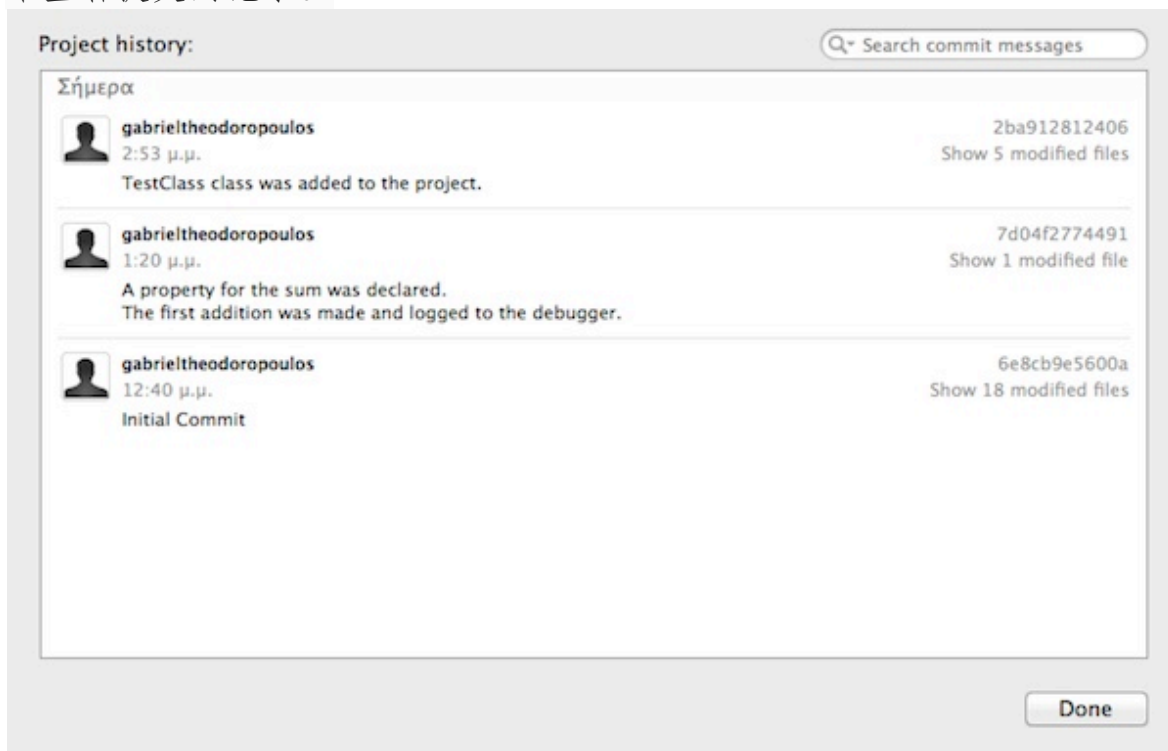
需要提交的一共有5个文件。除了之前修改的四个之外，还有一个项目配置文件。Xcode会在新类被添加到项目之后自动修改这个文件。如果你打开TestClass.h或TestClass.m文件，左边的窗口没有任何显示，如下图所示。



这是因为在这个文件在之前没有被提交的记录，因此没有一个可以比较的版本，在右边只显示了File was added。

在消息区写上这样一个描述：TestClass was added to project.. 之后点击Commit 5 files按钮即可。

这样第二次手动提交就成功了。你可以到Source Control > History 菜单查看提交的记录。



版本之间的比较 (Comparing Versions)

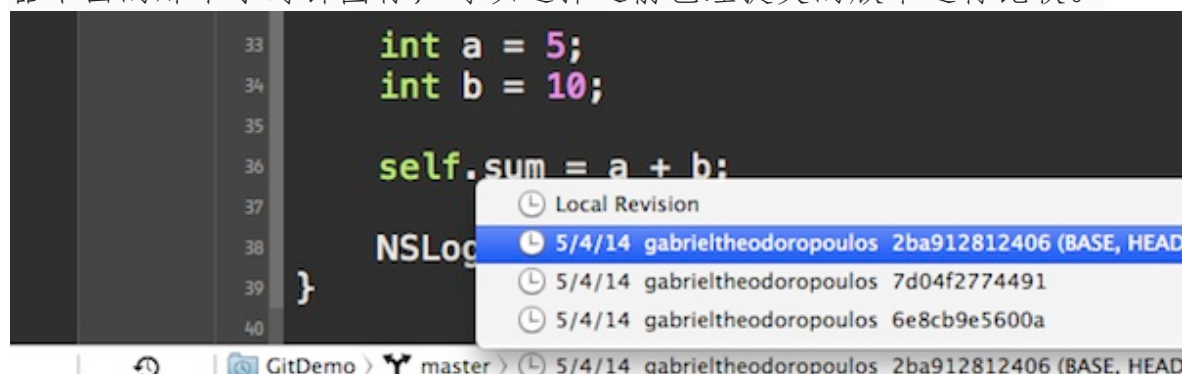
当你提交了同一工程的不同版本之后，在他们之间比较，追踪修改信息就会非常方便。当新添加的代码不能运行时，这时与之间版本进行比较就非

常重要了，你可以看出新版本相比上个稳定版有了哪些更改。

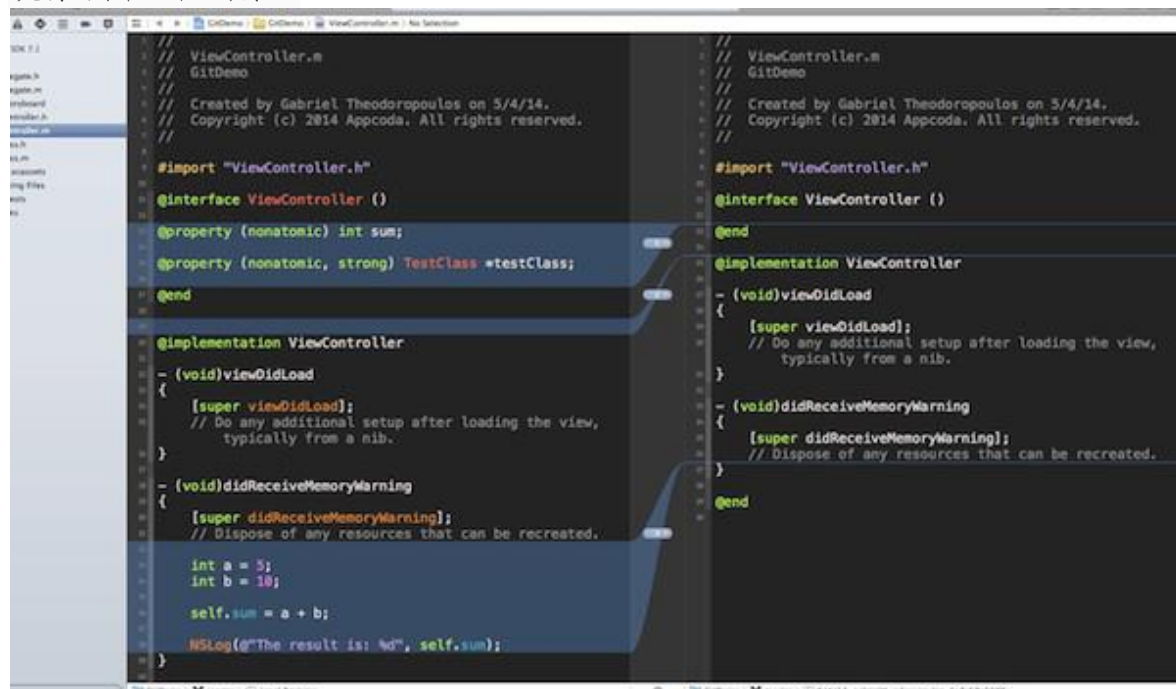
要比较同一个文件的两个版本，你可以使用View>Version Editor>Show version editor，或是点击工具栏上的Version Editor按钮：



点击之后，编辑器会分为两栏。最初，两栏会显示相同的内容，点击编辑器下面的那个小时钟图标，可以选择之前已经提交的版本进行比较。



点击之后，两个版本的区别会在编辑器中显示出来。通常，左边显示的是当前版本的文件，右边显示的是之前的版本。蓝色高亮的区域显示了被更改的代码，因此比较代码的变化非常容易。继续选择任何此前的版本，并观察两栏的区别。



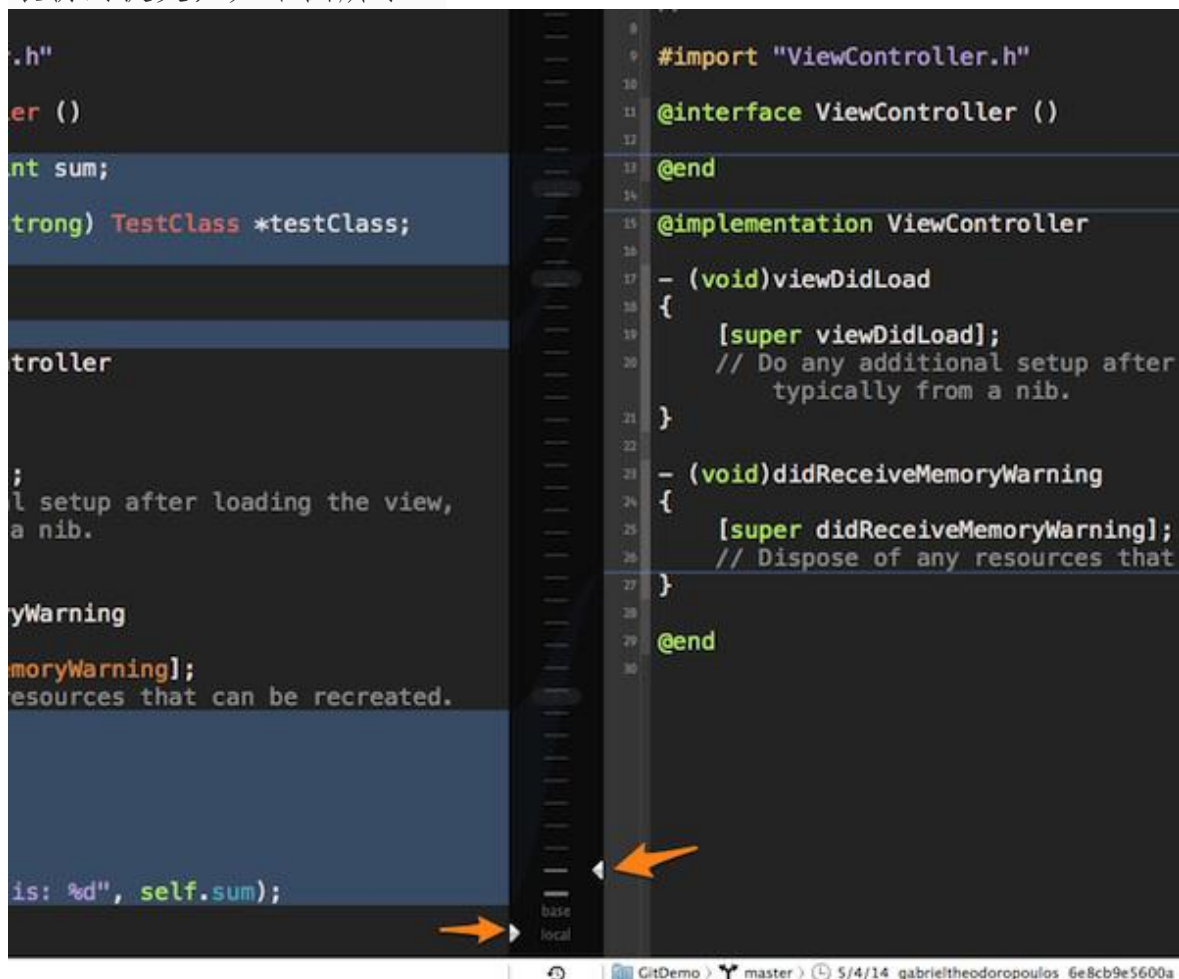
你可能会注意到，在两个编辑器中间，还有在提交窗口看到的小标签。点击向下的按钮可以跳出让你忽略更改的选项。如果你点击了忽略更改，Xcode会提示你是否同意。如果你同意忽略，这些被忽略的代码将会永远消失，无法再找回来。所以要注意不要无意中忽略任何代码。

除了上面说到的方法，还有一种你回到之前版本的方法。如果你仔细观察两个编辑器下面的工具栏，在中间有个带箭头的时钟图标：



点击它之后，两个面板之间的纵列内容就发生了改变，变成了一系列表示之前更改的时间戳。注意并不是所有的都代表实际提交。代表先前版本的圆角矩形的数量取决于提交的次数。在这个例子中，只有两个这样的图形，代表了两次提交。

在这一列的下面，有两个箭头。左边的那个属于左边的面板，右边的箭头属于右边的面板。将箭头移动到任意之前的版本，你会看到在相应面板中的改变。如果你想比较当前版本和之前任意版本的区别，让一个箭头指向 local 行，然后移动第二个箭头。时间戳从底部到顶部代表了从新到旧的代码。在 base 行，你会看到上一次提交的内容。继续向上移动，你会看到最初的提交，如下图所示：

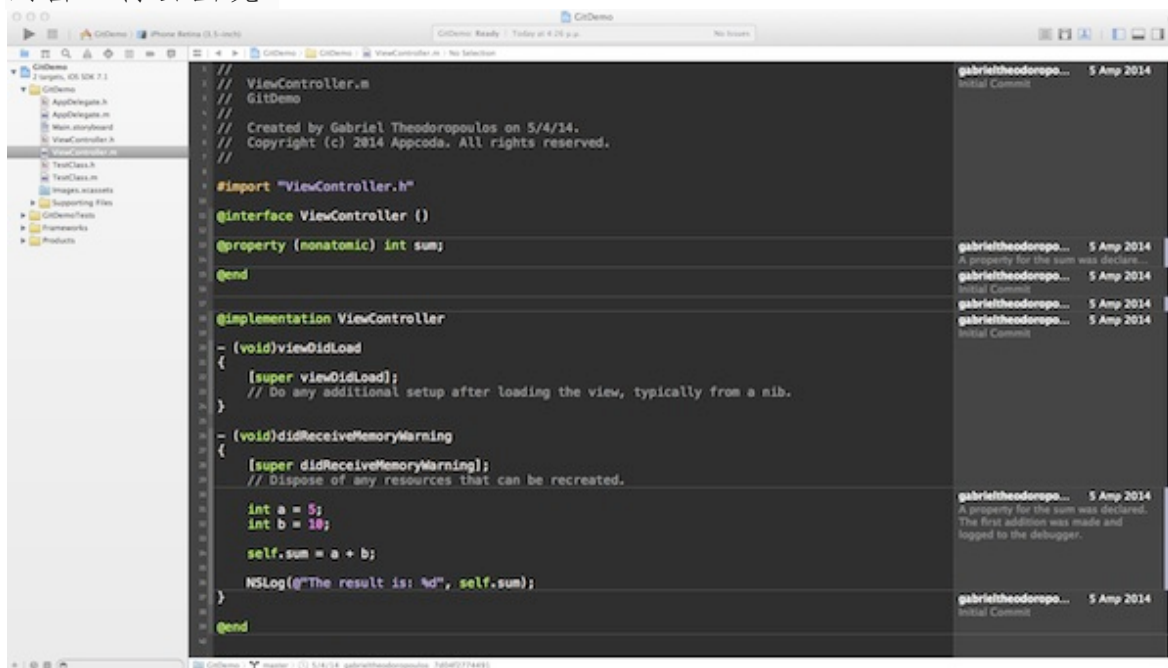


现在你知道如何比较版本之间的区别了。再继续深入之前，把前面学习的练习一下玩玩吧。

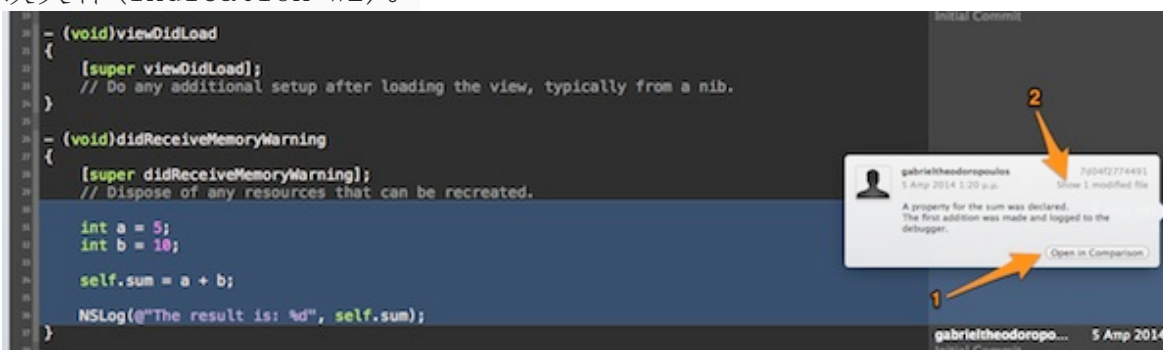
究竟是谁的错？（Who's Got the Blame）

除了比较文件的版本外，Xcode还可以让你追踪文件的提交者，以及是谁改变了哪一部分代码。在一个多人的团队中，这非常有用。要使用这个功能，点击View > Version Editor > Show Blame View菜单。或是讲鼠标

放在工具栏的Version editor 按钮上，选择Blame选项。一个与上面类似的窗口将会出现：



正如你看到的，当前文件依据不同的提交被水平线分成几段，每个代码段的作者，以及提交信息和其他信息显示在窗口右边的一个特殊面板中。如果你还没有做过，那自己动手打开这个blame视图，注意一下Xcode展现代码段作者的方式。在这个视图中，可以方便地找到某一代码在何时被谁提交以及其他你想要的信息。将鼠标放在blame面板上，将会显示修改的一些其他信息。当指针停在提交段上时，一个带图片的小按钮就会出现在它的右边。点击选中该段代码，就会弹出一个附带提交信息窗口。在这个窗口中，你还可以跳转到比较窗口(indication #1)，以及特定提交的修改文件(indication #2)。



除了比较视图和blame视图，其实还有一个日志视图(Log view)。你可以通过View > Version Editor > Show Log View来打开它。或者如果你在这里就不在详细说它了。你可以自己去看看，毕竟这个用起来也没那么复杂。

分支(Branches)

试想一下，你现在的工程有一个即将发布的版本，或是已经发布的版本，你突然想添加一些新的特性，如何防止这些新添加的代码让整个项目陷入

瘫痪呢？答案很简单：你需要使用分支。

如何简单的理解分支呢？你可以把你的项目想象成一棵树，稳定版本就是树的主干。任何添加新功能的版本都必须是树干的一部分。分支，就像是树的枝干，它从树干生长出来，向不同的方向生长。在git中，你可以通过创建分支来为你的代码设置一个新的路径来实现新特性，而不用担心在开发中破坏主干。

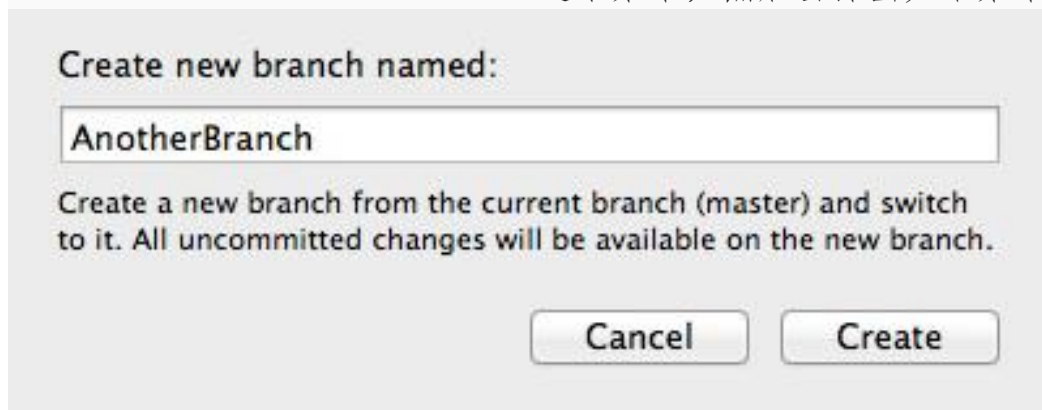
实际上，在git中默认都会有一个分支，叫做master。Xcode自动执行的第一次提交中就发生在这个分支中。通常，单独的开发者只在master这个分支开发，这其实不是一个好习惯。无论你是单打独斗还是组团合作，我认为在对项目作出重大改变或添加重大功能时，使用分支是十分重要的，它会为你避免很多麻烦。当然，在团队项目中，为你自己负责部分的代码搞一个分支几乎是必须的。

关于分支，你必须记住以下两点：

1. 提交到App Store或客户的最终产品必须是项目中的master分支项目。
2. 任何在第二分支中实现的代码或者功能最终都必须合并到master分支，这样正式发布的应用程序才是完整的。（以后再讲这一点）

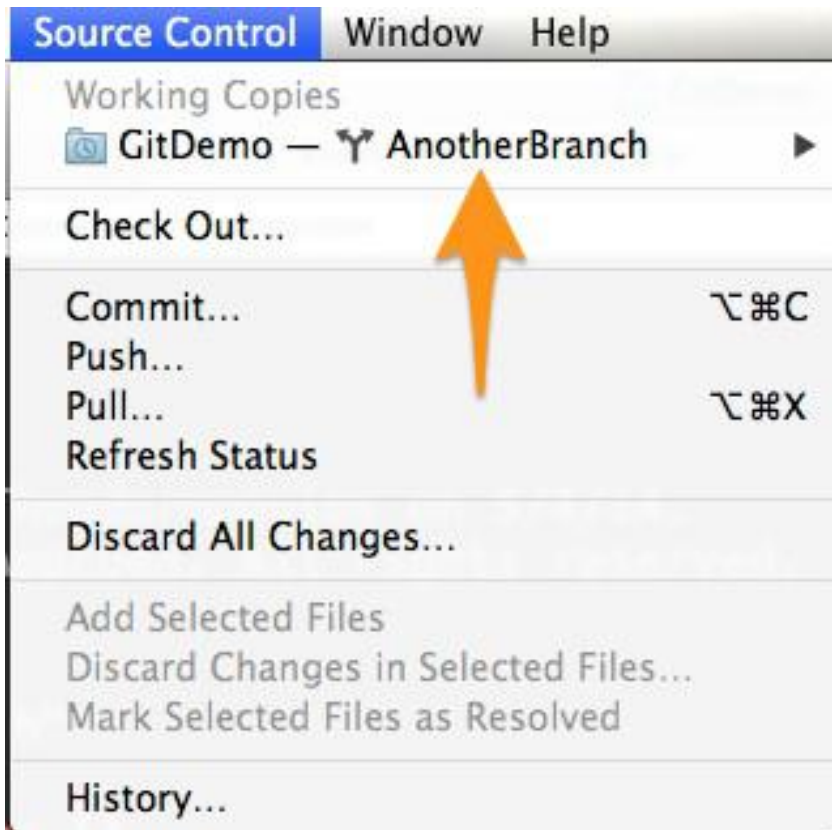
当你开始一个新分支时，你实际上是以当前工作状态作为起点，即使你有任何未提交的更改。从这个时候起，所有的改变都会只体现在分支中。

现在让我们回到Xcode，要创建一个分支，点击Source Control > GitDemo-master > New Branch...这个菜单，然后会弹出如下菜单：



为这个分支起一个名字，我就把它起名为AnotherBranch好了。现在你怎么给它起名其实都无所谓。点击OK按钮，等一下新的分支就会被创建，而当前的代码也会复制到新分支中去。

打开Source Control菜单，你就可以轻松地找出活动分支是哪一个：它就在项目名字的旁边。



现在，让我们做一次新的分支的提交。在这之前，让我们添加一些新的代码。打开类文件，在私有属性区添加以下方法声明：

```
1  @interface ViewController ()
2
3  ...
4
5  -(void) sayHello;
6
7  @end
```

然后实现它：

```
1  -(void) sayHello {
2      NSLog("Hello");
3  }
```

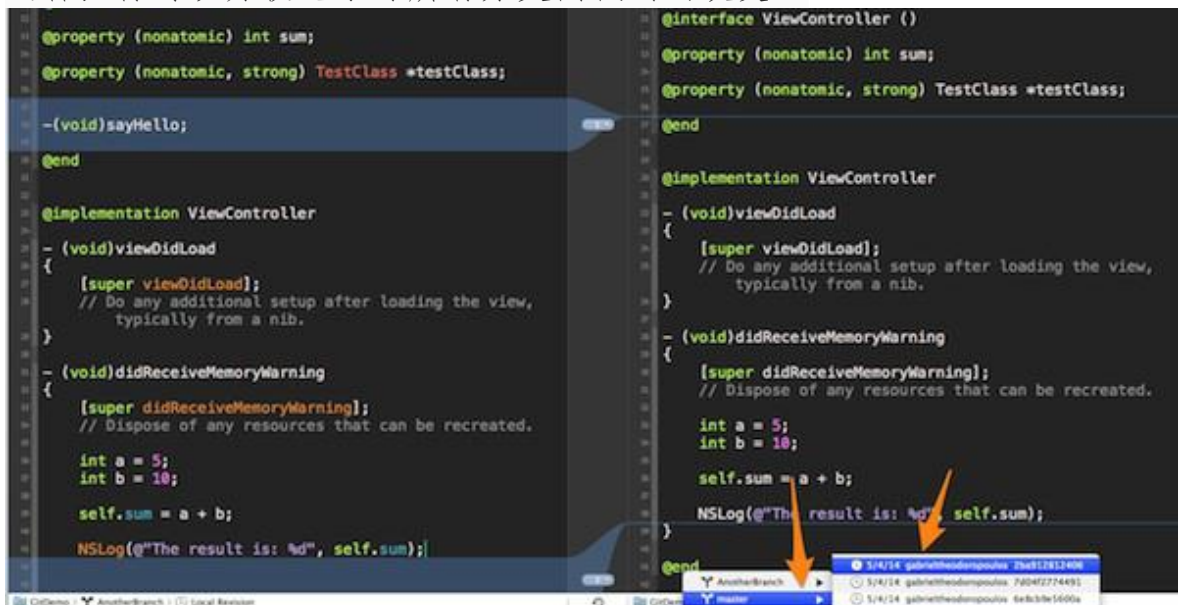
最后，在viewDidLoad中调用它：

```
1  -(void) didReceiveMemoryWarning
2  {
3      ...
4
5      [self sayHello];
6  }
```

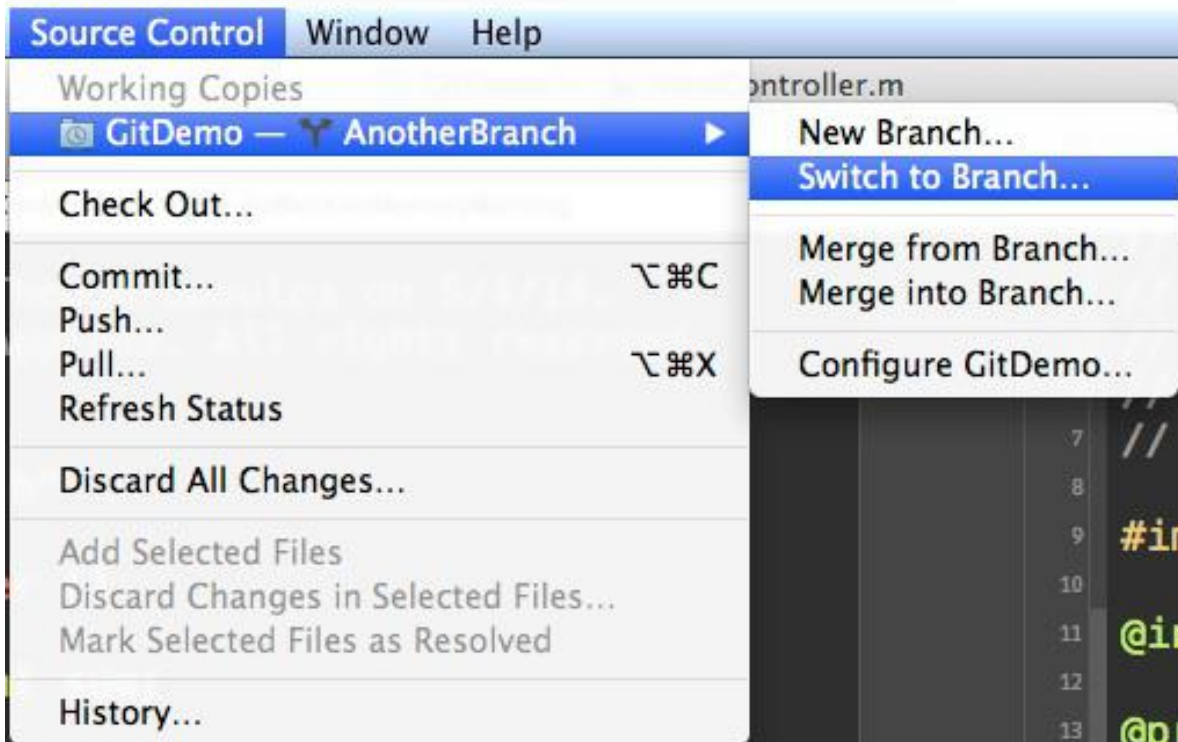
现在，点击Source Control > Commit菜单，版本比较窗口将会出现，你会看到只有一个被修改过的文件--ViewController.m文件，新添加的部分会被高亮显示。

输入下一个提交信息: First commit to a new branch, 然后点击commit
1 file按钮。现在AnotherBranch分支的改变就会被提交了。

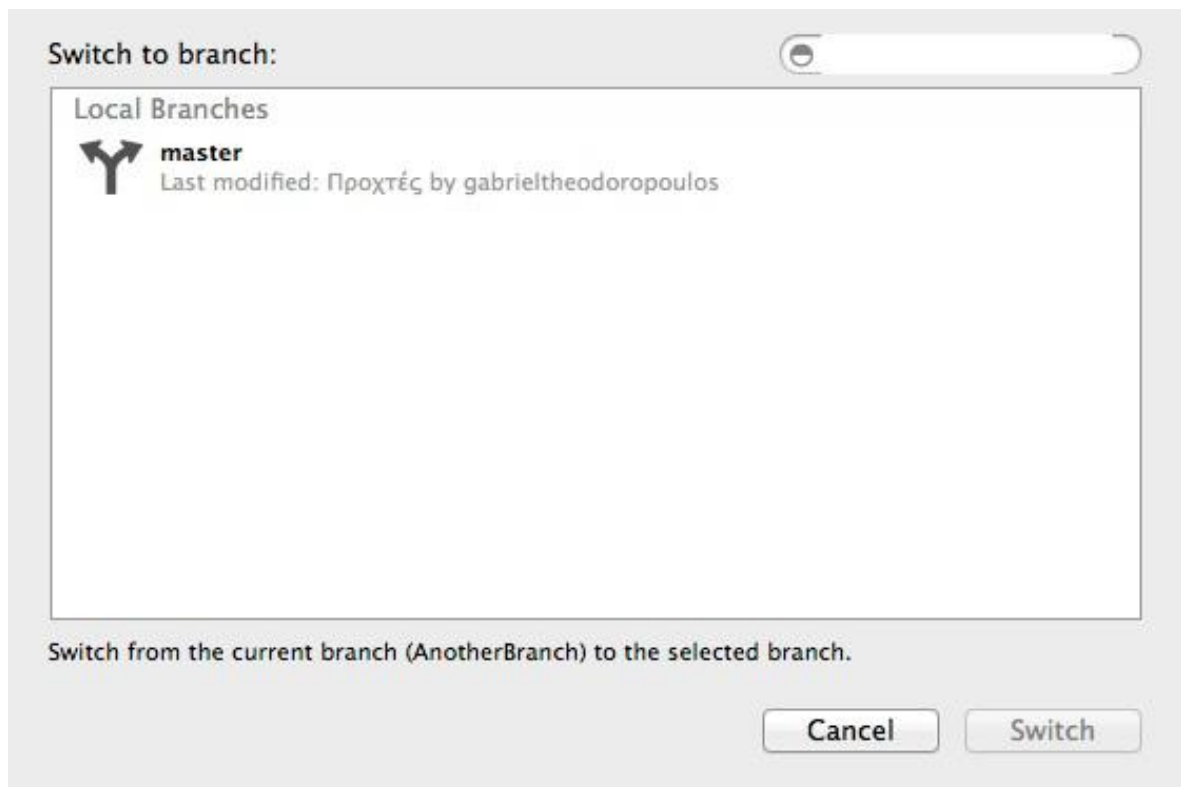
打开Version Editor(menu View > Version Editor > Show Version Editor), 找到右边编辑面板下面的工具栏, 你会看到被选中的分支是AnotherBranch, 点击它, 你会看到这个分支和master分支同时出现, 从master分支中选择任意版本, Xcode都会高亮显示两者之间的区别。通过这样, 你可以方便地跟踪所有分支间代码的改变。



最后, 切换到另一个分支, 或是master分支, 你可以点击Source Control > GitDemo - AnotherBranch > Switch to Branch...菜单。



从这个窗口你可以选择想要跳转的分支, 在这里让我们跳回master分支:



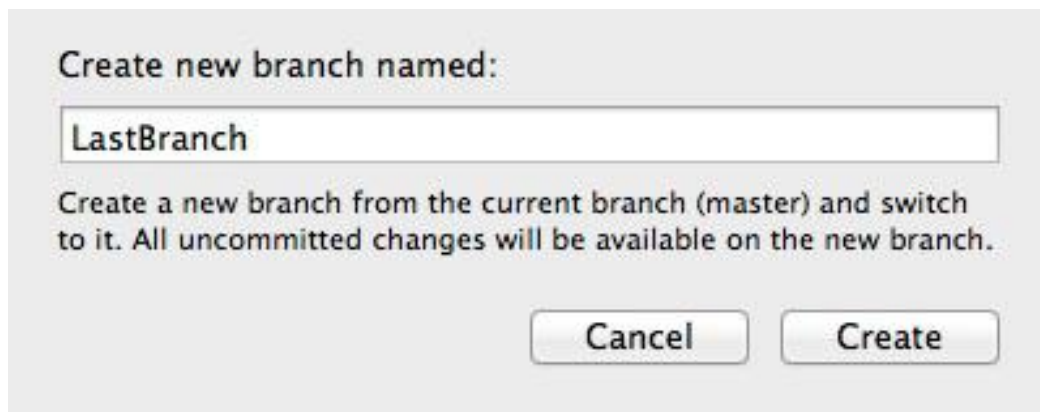
选择它并点击Switch按钮，master分支就会成为当前活动分支。你会发现，在AnotherBranch中做出的改变并没有出现在master分支。很好，我们在管理工程推进的同时，却没有修改稳定版本。

合并分支 (Merging Branches)

在分支中进行开发是一种好习惯，然而，如果代码改变要体现在发行版中，那么分支就必须被合并到master分支中。这一节我们将会告诉你怎样合并它们。在Xcode里，将两个分支合并成一个非常简单。

让我们做一个小实验来看看合并是怎样工作的。首先，确保master分支是现在的活动分支。如果不是，赶紧改过来：Source Control > GitDemo - AnotherBranch > Switch To Branch... menu，并从展示窗口选择master分支。

下一步，创建一个新的分支：Source Control > GitDemo - master > New Branch... menu，命名为LastBranch



先让Xcode飞一会，然后，到ViewController.m文件中，再创建一个私有方法，首先声明它：

```
1  @interface ViewController ()
2
3  ...
4
5
6  -(void) sayByeBye;
7
8  @end
```

然后实现它：

```
1  -(void) sayByeBye {
2      NSLog("Bye - Bye");
3  }
```

最后，在ViewDidLoad方法中调用它：

```
1  -(void) viewDidLoad
2  {
3      ...
4
5      [self sayByeBye];
6  }
```

在合并之前，先提交这些更改。使用Source Control > Commit菜单来执行提交。

终于还是来到这一步，关于把两个不同的分支合并成一个，你有两种选择 “

1. 从分支合并：与你选择的分支相关的任何改变都会被合并到现在活动分支中。
2. 合并到分支：当前活动分支的任何改变都会被合并到你选择的分支中。

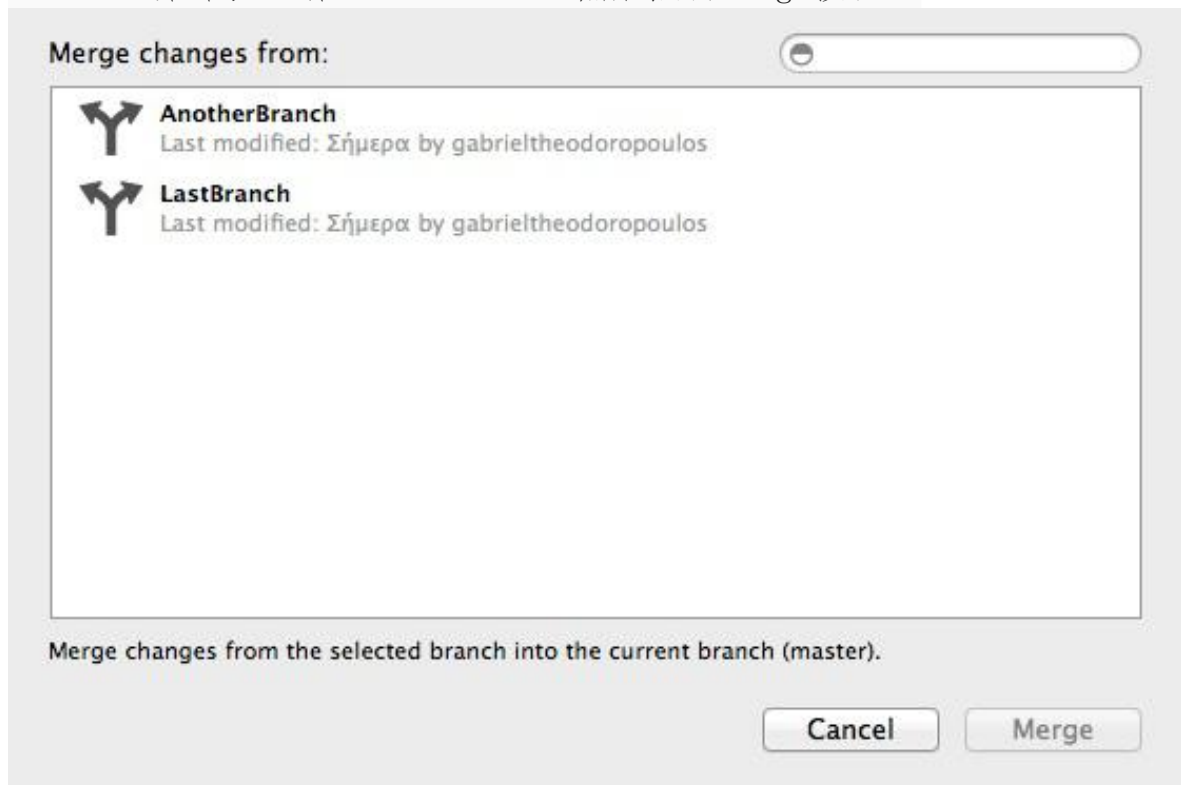
这两种方式你都可以在Source Control > GitDemo 菜单中找到。注意当你的活动分支是master分支时，第二个选项是不可选的。

假设一个开发者在Anotherbranch分支实现一个sayHello方法，另外一个

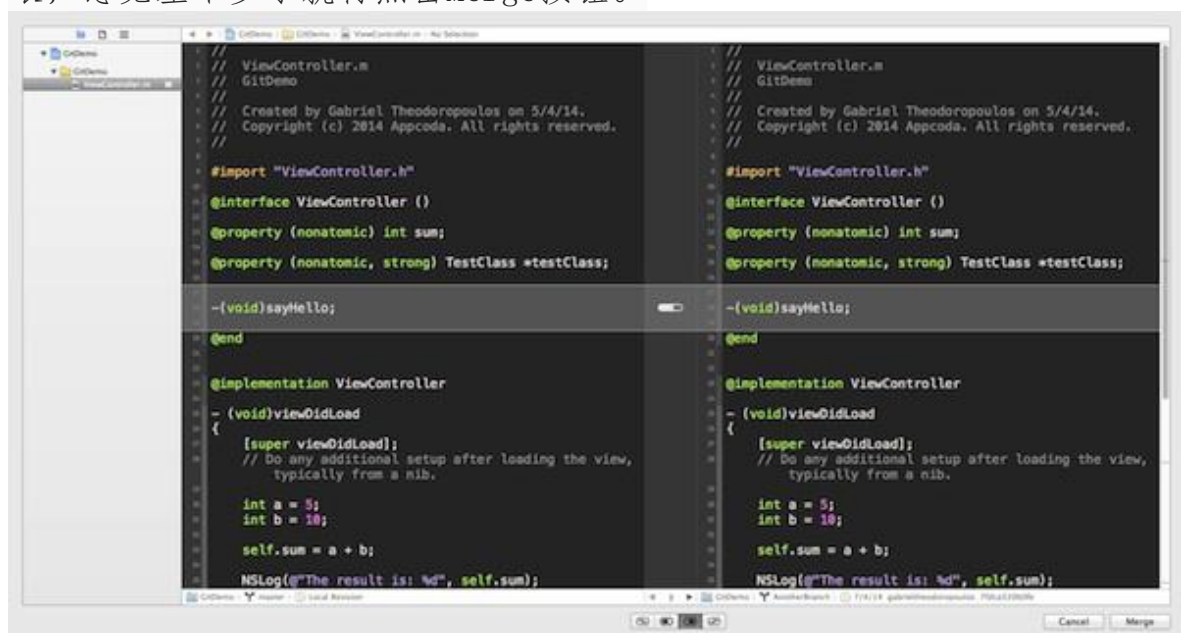
开发者在LastBranch中创建实现了sayByeBye方法，现在你需要将两个人的工作合并到下一个稳定版本中，想一想你需要怎么做？很简单，按以下方法将改变从两个分支中合并进来：

首先，确保当前活跃分支是master分支。

然后，打开Source Control > GitDemo - master > Merge From Branch...菜单，选择AnotherBranch然后点击Merge按钮。

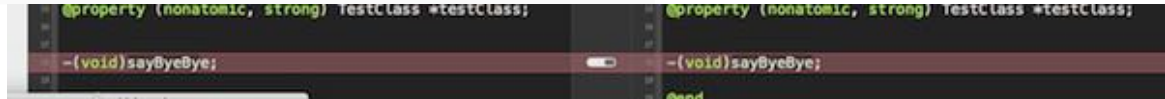


接下来会出现一个比较窗口，在里面你会看到合并之后代码的更改，看一眼，感觉差不多了就再点击Merge按钮。

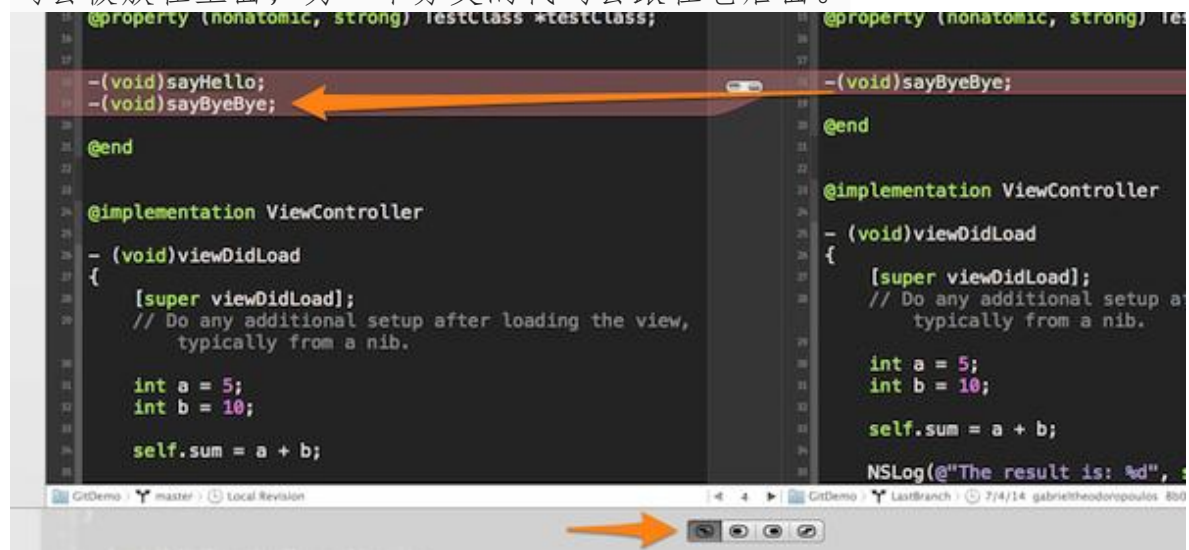


Xcode会询问你是否保存项目的快照，点击Enable按钮。让Xcode飞一会，然后就好啦。AnotherBranch里面添加的内容已经合并到master分支中。

使用同样的方法来合并LastBranch。你会发现如果你不提交更改，Xcode不会让你再次合并。于是，我们只好先提交一下。在比较窗口你会发现一个红色的区域显示合并之后的更改，而不是之前的蓝色。这意味着分支中的代码将会替换原先活动分支中的代码。



你可以轻松地避免这种现象的发生。在编辑面板的下面有几个小按钮，你可以试试他们都有什么作用，我选了第一个，它的意思是master分支的代码会被放在上面，另一个分支的代码会跟在它后面。



处理接下来所有需要更改的代码，不要有遗漏。完事后就点击Merge按钮。

恭喜你！你已经成功的学会从多个分支合并了代码，类似的情形你也应该会了。

忽略更改 (Discarding Changes)

放弃不想要的代码更改功能非常有用，只需轻轻一点，自从上一次提交之后的更改都会被放弃。当你在开发过程中发现出了大乱子，你想从上一个稳定状态重新开始时，这个功能就派上用场啦。注意放弃更改这个功能没有回头路，点完之后你就没有办法再撤销这个操作，所以，要小心使用啊！

之前，当我们在讨论版本比较时，我们学会了如何忽略某一部分更改的方法，下面，我们要学一下如何一下忽略自从上一次提交之后的所有更改。

为了测试这个功能，首先写一些代码打开ViewController.h，添加一个公共方法声明：

```
1 @interface ViewController : UIViewController
```



```

2
3 -(void)aVeryCoolMethod;
4
5 @end

```

现在，让我们在ViewController.m中添加一个这个方法的实现，简单点就行：

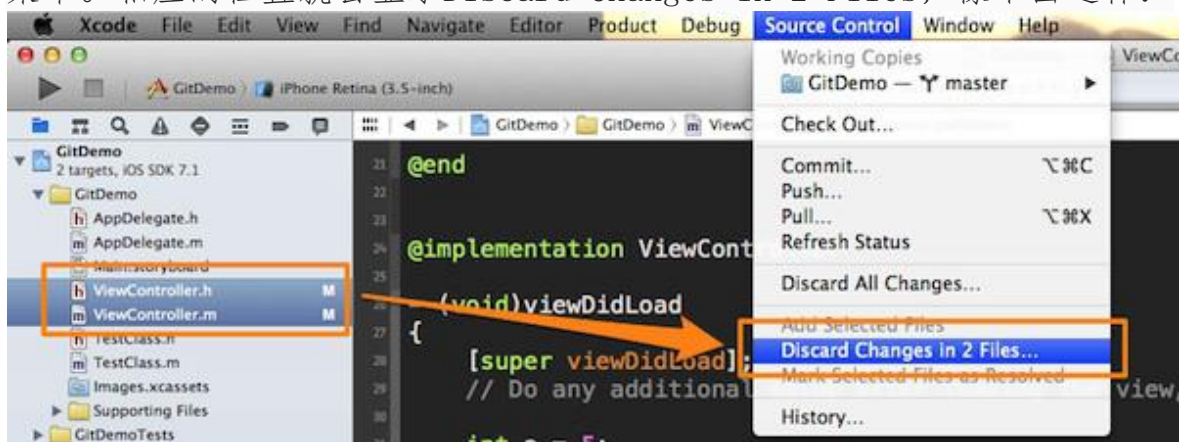
```

1 -(void)aVeryCoolMethod{
2     NSLog("I'm feeling that you'll discard me... Really?");
3 }

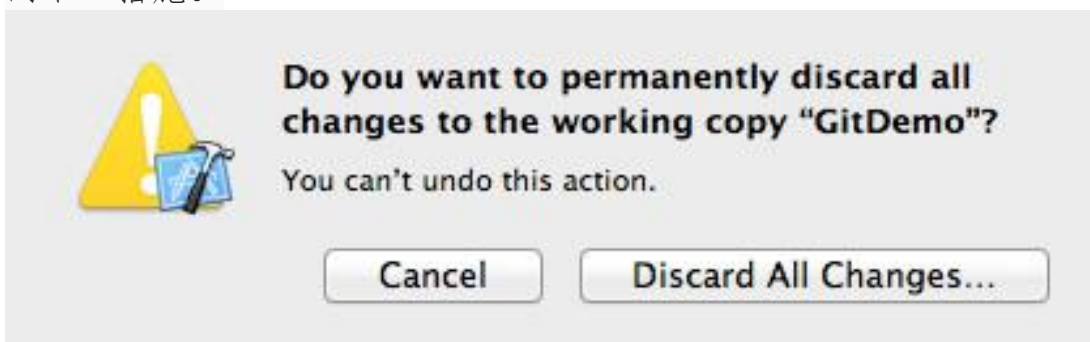
```

如果你注意到Project Navigator，我们刚刚更改的文件旁边有了一个M标识，很好，我们想看看如果忽略这些更改，这些文件是否会回到更改之前的状态。

这里有一个重要的细节：你可以选择忽略所有文件的更改，也可以选择忽略单个文件的更改，这完全取决于你。如果你想忽略一个文件的更改，首先选定这个文件。在这个例子里，如果你只选择ViewController.m文件然后打开Source Control菜单，你会在ViewController.m中发现Discard Changes这个选项。类似的，如果你只选择ViewController.h也是一个道理。然而，如果你想忽视这两个文件的更改（这里假定有两个以上的更改），就在Project Navigator中选中它们，然后再打开Source Control菜单。相应的位置就会显示Discard Changes in 2 Files，像下面这样：



然而，这次我们不会使用这个按钮，我们要用Discard All Changes。点击它之后，一个确定提示框就会出现，这是这部分Xcode防止你误删代码的唯一措施。



点击Discard All Changes，那你刚才写的那个公共方法就永远属于过去了。看到了吧，只需几步就可以让你从当前工作状态恢复到之前的提交，所以我再一次提醒你要在使用Source Control 中小心点，别误点了这个按钮。

总结

通过这篇教程，我尽力详述了在Xcode中进行版本控制的方法。其实在幕后，真正起作用的是git----地球上应用最多的版本控制系统。你可能注意到我在教程中并没有过多的提到GitHub或者任何Xcode的一些功能----其实我是故意这样的。我想把注意力集中在使用Xcode进行git管理的内容上。只有当你懂得了如何进行版本控制之后，才能真正的使用GitHub。我想再重申一下，如果你是一个团队在工作，使用版本控制工具是必须的！如果你是单打独斗，使用版本控制工具也是很有必要的，它可以为你花大量时间和精力所做的工作提供保障，并且在你添加新功能时可简单地进行扩展。这个工具就像有些人说的那样，一旦用了，就再也回不去了！最后，我希望这个教程会对你有用