# 需要搞清楚的

V7里面的SimpleCursorAdapter

V7里面的AlertDialog

mListView.setChoiceMode(AbsListView.CHOICE\_MODE\_MULTIPLE\_MODAL);，还有另一个常量：AbsListView.CHOICE\_MODE\_MULTIPLE，有舍区别呢？

<view

class="android.view.View"

android:id="@+id/row\_tab"

android:background="@color/green"

android:layout\_width="10dp"

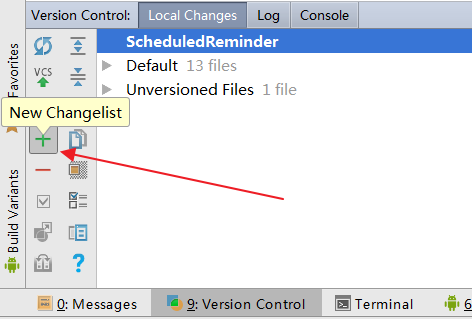
android:layout\_height="match\_parent"/>

为什么要这样写一样View控件？有什么好处

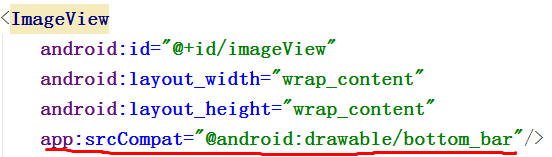
android:background="?android:attr/activatedBackgroundIndicator" 这种问号打头的东西要怎么样看它的真实内容，它跟@android:drawalbe这些有什么区别

你的清单文件配置最低版本是8，但是你使用的一些API在11版本才有，运行到8的话肯定会挂的，但是你通过代码判断了是11版本才会运行那些代码，所以运行到8也不会挂，但是AndroidStudio还是会突出显示为错误，这是由Lint生成的，它是Android SDK中内置的静态分析工具，而且已经完全集成到了AndroidStudio中，要想不显示这个错误，可以在方法上加这个注解：@TargetApi(Build.VERSION\_CODES.HONEYCOM)。

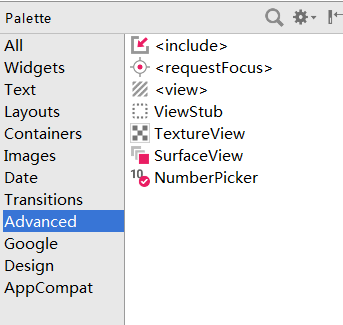
打开一个项目和导入一个项目有什么区别？



这个功能有什么用？



这个属性是什么时候出来的？有什么不一样？



这里面没用过的控件要搞清楚干嘛的

# 技巧或特殊的地方

## Android中的颜色表示方式

有4种：

#ARGB，颜色组合数量：16 x 16 x 16 x 16 = 65536

#RGB， 颜色组合数量： 1 x 16 x 16 x 16 = 4096，这种模式透明度组合只有一种，就是完全不透明

#AARRGGBB，颜色组合数量：256 x 256 x 256 x 256 = 4228250625

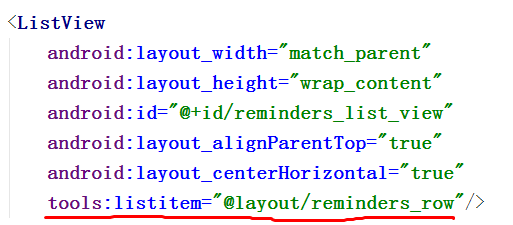
#RRGGBB， 颜色组合数量：1 x 256 x 256 x 256 = 16581375，这种模式透明度组合只有一种，就是完全不透明

这些都是16进制表示的，每一个字母用一个16进制来表示。

#ARGB和#RGB使用的是4比特通道，因为它的每种颜色最大值只有16，用4个比特位就能表示16种组合了

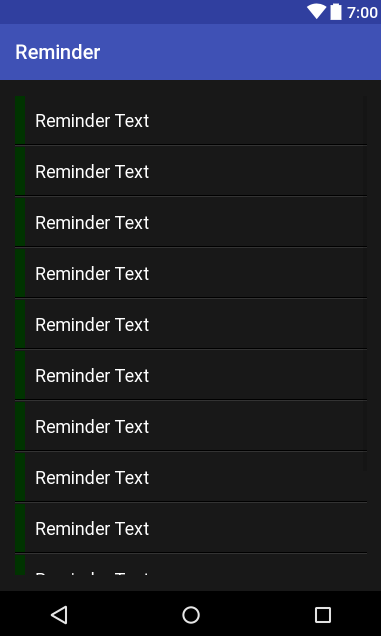
#AARRGGBB和#RRGGBB使用的是8比特通道，因为它的每种颜色最大值只有255，用8个比特位就能表示255种组合了

## 让ListView在布局中预览的时候就能看到item的布局：



这只是预览，在运行的时候这个ListView并不会使用这个item布局，如要使用得在适配器中进行配置。

## 我当前用的是AndroidStudio2.3版本，奇怪的现象，这里把id属性删除，在预览的时候发现看不到item效果，有id与无id的预览效果如下：

## 在ListActivity中使用自己的布局

在ListActivity中可以不设置布局，也就是不需要setContentView这个代码，直接调用setListAdapter即可显示列表，如果需要自己定义一个布局界面，则需要调用setContentView，还是调用setListAdapter来展示列表，这时布局中的ListView的id必须是@android:id/list，这样系统才能够找到这个ListView并绑定Adapter。

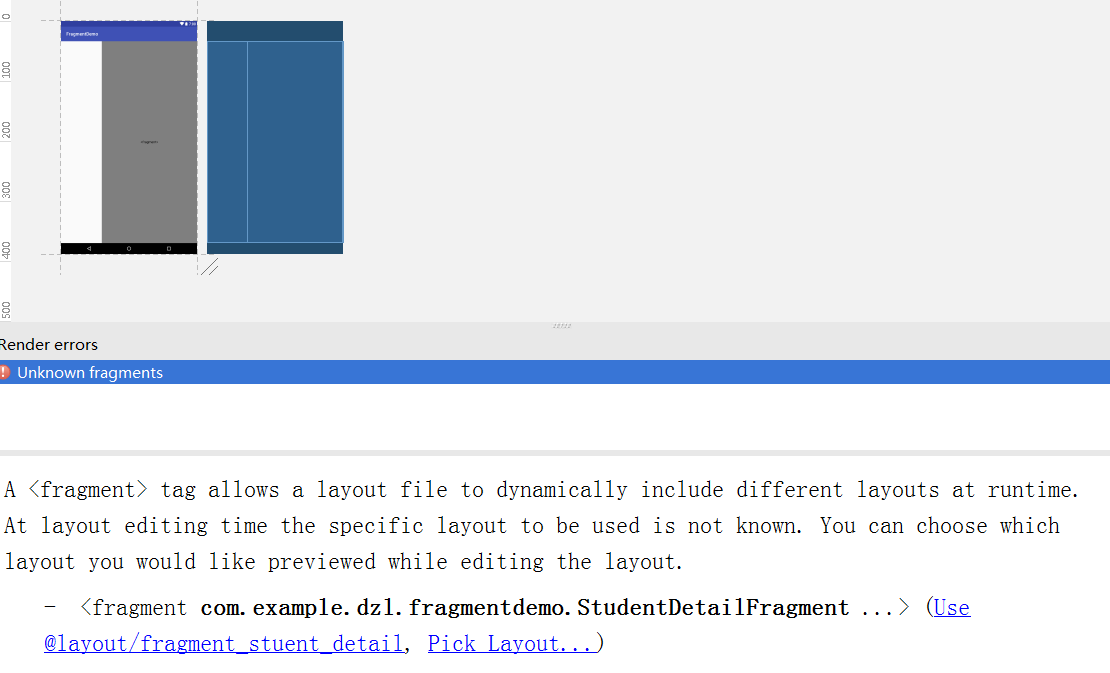
## 在Fragment中保存Activity位置

在Fragment的onAttach(Activity activity)方法中保存，看方法含意，这时的activity肯定不可能为空的。

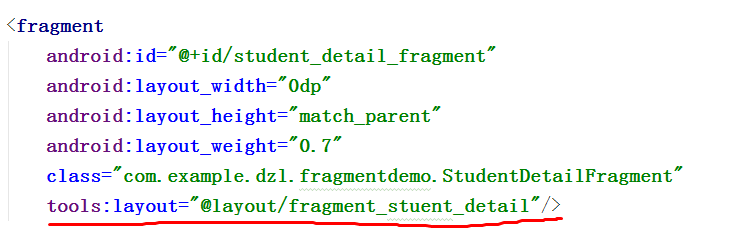
## 资源文件夹修饰符

资源文件夹可以有哪些修饰符，记不住，太多了，其实不需要记，在res目录右击🡪New🡪Layout Resource File🡪在出现的对话框左下角即可选择修饰符属性，然后AndroidStudio会自动创建所需修饰符的目录。

## 在xml中预览Fragment



当布局中嵌入了Fragemnt，在预览时会报上面的错误，提示信息大概意思为“一个<fragment>标签允许布局文件在运行时包含不同的布局，在编辑布局的时候要使用哪一个布局没法知道，你可以选择在编辑布局时希望预览的布局。”点击超链接来选择布局即可显示预览。这其实是加了下面的一个属性而已：



## 自定义颜色



点击布局左边的颜色预览可以打开一个选择颜色的对话框

## 抽取style/layout

定位在布局控件上，执行Refactor🡪Extract🡪Style/layout，样式可以抽取多次，每次都会以上一次的样式作为父样式。

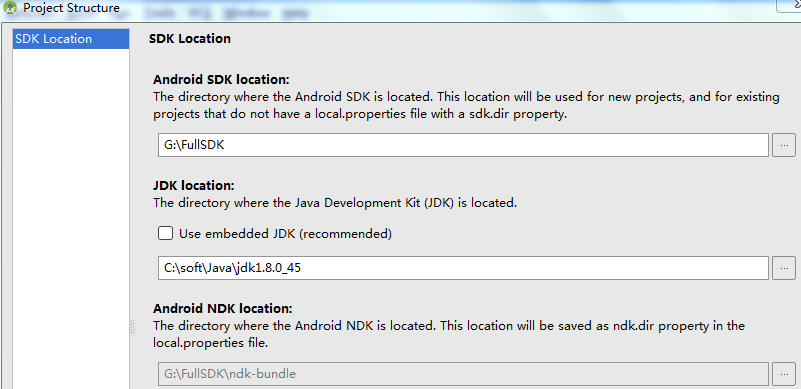
## 设置Spinner的下拉列表布局

显示Spinner时需要给Spinner传一个适配器，比如传一个ArrayAdapter，该Adapter有一个方法setDropDownViewResource(android.R.layout.simple\_spinner\_dropdown\_item)方法接收一个布局，可参考系统的这个布局修改下拉列表的显示。

## SharePrefrence的读写要比SQLite慢一些。

## ANR异常是UI线程被阻塞的时间达到5000毫秒

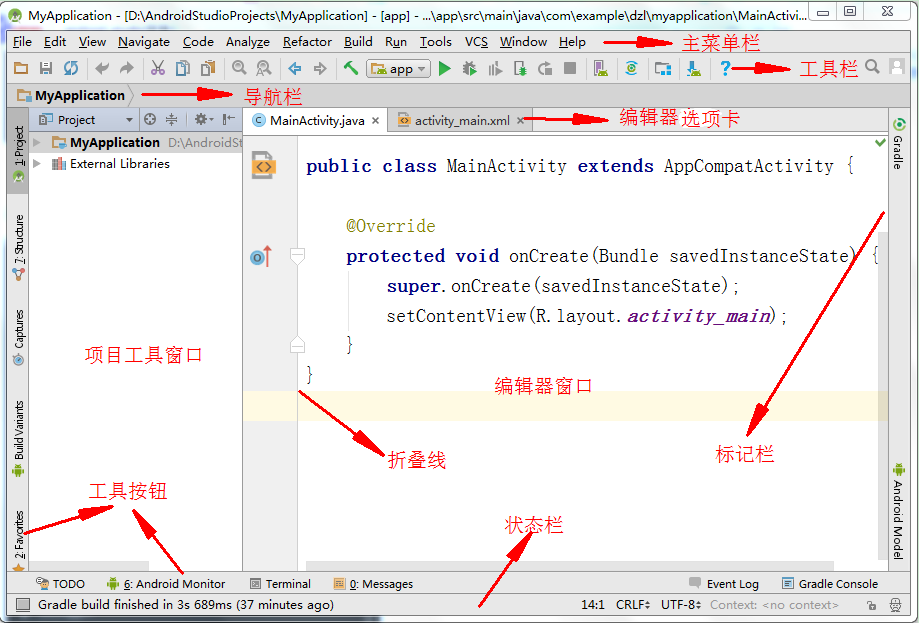
# 第一章：AndroidStudio入门

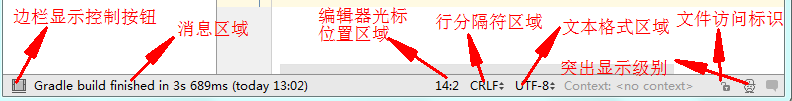
1、安装JDK1.8，设置JAVA\_HOME，现在最新版本的AndroidStudio不会使用这个JDK，需要在AndroidStudio中配置：File🡪Other Settings🡪Default Project Structure,如下图：

2、安装AndroidStudio

# 第二章：在AndroidStudio中遨游

## 认识AndroidStudio窗口



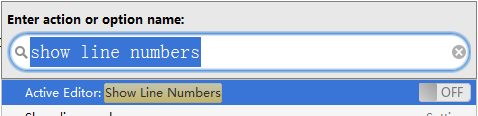
* 编辑器：总是位于中心面板，其它所有的窗口均被称为工具窗口，它们位于编辑器窗口的周围（左侧、下方和右侧）
* 编辑器选项卡：在编辑器选项卡之间导航：Alt + 左右箭头，关于编辑器选项卡的命令在：Window🡪Editor Tabs，在编辑器选项卡上右击可出现相关的上下文菜单。
* 折叠线：折叠代码，也可用于设置断点。
* 标记栏：标记代码，标记自动出现，如有报错的会有红色标记，这些标记不会跟随滚动条滚动，它显示的位置是文件的相对比例的位置，点击可以跳转到标记的代码的位置。
* 工具按钮：通过View🡪Tool Windows可查看所有的工具按钮，有些工具按钮上标有数字，意味着可以按下Alt和对应的数字打开该工具窗口。如Alt + 1打开Project工具窗口
* Window菜单中有保存布局和恢复默认布局的命令
* Project工具窗口：最常用的工具窗口，显示项目文件
* Stucture工具窗口：显示文件元素的层次结构，如果是Java类，则会显示字段、方法和内部类等元素，如果是XML则显示XML的树结构，点击Structure中的任意元素则会跳转到编辑器中的对应位置，这在较大的文件中导航非常有用。
* Favorites工具窗口：用于收藏文件、书签、断点，在编辑器选项卡有右击🡪Add toFavorites/Add All to Favorites即可收藏文件，书签功能则可以让你收藏代码的某一行，定位到代码的某一行，然后按下F11则可以把该放入书签，在该行的左侧会有一个勾。在折叠线上单击则可以增加断点，在折叠线左侧单击也可以，并且会显示红点，该行代码也会有红色背景显示。
* TODO工具窗口：代办事项，TODO本质上就是注释，用于提醒自己或者其它同事还有尚未完成的工作。
* 主菜单栏：通过它几乎可以执行任何操作，它无法隐藏，不要怕命令多，即使是最教练的Android开发者在日常工作中也仅仅会用到这些操作的一小部分而已，并且大多数命令均有快捷键或上下文菜单。
* 工具栏：包含频繁使用的操作。View🡪Toolbar可隐藏
* 导航栏：用于在项目的资源之间进行导航。
* 状态栏：显示相关上下文敏感的反馈。  
  
  + 边栏显示控制按钮：点击可以显示或隐藏窗口左右两边的工具按钮，把鼠标放到上面还可以切换显示工具窗口
  + 提供反馈并显示关于当前正在运行进程的所有信息。当把鼠标移到菜单项或工具栏中的按钮等UI元素时，此区域还会显示提示，在这个区域单击会打开Event日志。
  + 编辑器光标位置区域：显示当前行列位置，点击可输入行进行定位（列可以输入也可不输入）
  + 行分隔符区域：显示了文本文件中使用的回车换行格式。在Windows上，默认值是CRLF，表示回车换行符，LF是Unix和Mac机器上使用的标准格式，也是Git中所采用的。如果在Windows电脑上进行开发，Git通常会在向仓库提交代码时将CRLF转换为LF。
  + 文本格式区域：描述了用于源文件的文本编码。默认值是UTF-8，这旨ASCII的超集且涵盖了大多数西方字符，包括你能够在标准Java或XML文件中找到的所有字符。
  + 文件访问标识符区域：允许你在文件的读/写和只读之间进行切换。
  + 突出显示级别区域：允许你设置想要在代码中看到的突出显示级别。默认设置是Inspections（检查），此设置意味着你要准备好授受严格的检查，因为Inspections Manager会严格识别代码中的语法错误和问题（称为警告）。你会看到Inspections Manager在标记栏中以黄色标记的形式生成的一些警告。如果将级别设置为Syntax，则Inspections Manager会忽略警告。虽然没有Inwspections模式严格，但是仍然会突出显示那些将会阻止代码编译的语法问题。另一个模式为None，这意味着Inspections Manager已经不再关心你的代码了，在些模式下，即使是最严重的语法错误也会被忽略。

## 常用操作

* 双击单词可选择整个单词，在一行中三击，则选中整行。
* 结构化选择：将光标置于任意单词，按下Ctrl + W，选择整个单词，继续按，则选中的区域会扩展至包含任意数量的相邻文本，按Ctrl + Shift + W，则选中区域会缩小
* Undo和Redo：Ctrl + Z和Ctrl + Shift + Z
* 找到最近的文件：AndroidStudio最优秀的特性还包括它会记录你在近期操作过的所有文件：View🡪Recent Files，或者按Ctrl + E，它会列出最近修改过的文件，最多50个，可以通过File🡪Settings🡪Limits🡪Editor🡪Recent Files Limit修改这个限制。
* 遍历最的的导航操作：AndroidStudio还会记住你最近的导航操作，包含光标移动、选项卡切换和文件打开。要遍历你的导航操作历史，按Ctrl + Alt + 左右箭头。
* 复制：
  1. 复制路径：Ctrl + Shift + C，可以复制Project工具窗口以及任意编辑器选项卡中选中的文件或目录的全路径
  2. 复制引用：Ctrl + Alt + Shift + C，可以复制方法、变量或类的逻辑引用。

## 获取帮助

Help菜单中有几个有用的菜单项。

File Action（Ctrl + Shift + A）是在AS中获取帮助 的最常用命令，此命令会打开一个对话框，允许你搜索AS中所有特性，比如输入Show Line Numbers，如下：  
  
点击OFF开关即可打开，在编辑器窗口中即可显示行号，在Settings🡪Editor🡪Appearance中也能看到Show Line Numbers复选框。

Help🡪Android Studio Help是所有AS技术规范文档的来源，这是关于AS最详细的文档。

Help🡪 Keymap Reference是AS所有的快捷键

## 使用键盘导航

键盘或许是AS中导航的最有效方式了。查看Navigate菜单。

Select In： Alt + F1

Class：Ctrl + N，导航到指定的Java类，包含内部类

File：Ctrl + Shift + N，导航到项目中的任意文件，如果正在查找项目中的XML文件，可以采用这种操作方法

Line：Ctrl + G，导航到源文件中的某一行

Related File：Ctrl + Alt + Home，打开相关联的文件，如打开Activity关联的XML布局文件。

Last Edit Location：Ctrl + Shift + Backspace，导航到最后一次编辑的地方。可以连续向后导航。

Type Hierarchy：Ctrl + H，列出Java类的继承树结构图。

Declaration：Ctrl + B，跳转到字段或方法的声明出。

## 查找和替换文本

Find：Ctrl + F 在单独的文件中查找，在标记栏会标记出查找到的位置，点击可直接跳转

Find in Path：Ctrl + Shift + F 在指定的路径下进行搜索，可以使用正则表达式，并使用文件掩码来限定结果范围

Replace：Ctrl + R 它是Find的超集，用于替换文本，更安全的做法应该使用Refector🡪Rename

Replace in Path：Ctrl + Shift + R 是Find inPath的超集，用于替换指定路径下的文本，更安全的做法应该使用Refector🡪Rename

# 第三章 在AndroidStudio中编程

关于代码操作的许多快捷功能都在Code菜单，如果记不住快捷键，则可以使用Code菜单中的命令。

## 3.1代码折叠

Settings🡪Editor🡪General🡪Code Folding可修改代码折叠选项

Expand（展开）：Ctrl + 数字加号，展开光标所处位置的已折叠代码块

Collapse（折叠）：Ctrl + 数字减号，折叠光标处已展开的代码块

Expand All：Ctrl + Shift + 数字加号，展开窗口中的全部代码

Collapse All：Ctrl + Shift + 数字减号，折叠窗口中的全部代码

Toggle Fold：Ctrl + 句号，折叠/展开光标所处位置的代码块

## 3.2 执行代码补全

AS的4种代码补全类型：

1. Default代码补全，无快捷键，会在你开始输入的时候自动出现一个建议列表，按上下箭头选择，回车确认
2. Basic代码补全，Ctrl + 空格，类似于Default代码补全，但还会在建议列表中当前选中条目的旁边显示一个Javadoc窗口，单击这个窗口中的向上箭头可显示详细文档
3. SmartType代码补全，Ctrl + Shift + 空格
4. Cyclic Expand Word代码补全，Alt + /，输入单词前面的1 ~ n个字母后，按此快捷键，提供文档中使用过的单词，向上循环。
5. Cyclic Expand Word代码补全，Alt + Shift + /，输入单词前面的1 ~ n个字母后，按此快捷键，提供文档中使用过的单词，向下循环。

## 3.4使用代码生成

Alt + Insert，生成构造函数、getter、setter、equals()、hashCode()、toString()等等。

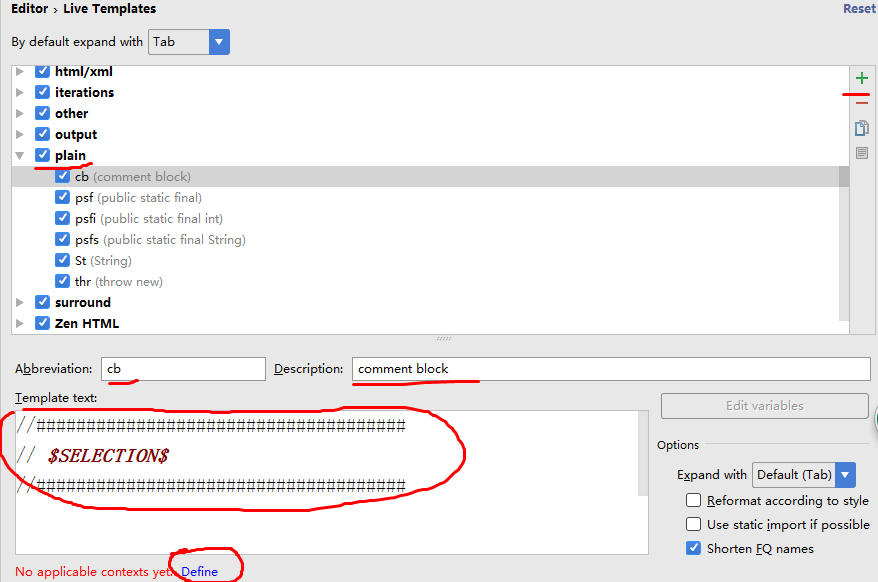
配置忽略成员变量名称的前缀m和s：File🡪Settings🡪Editor🡪Code Style🡪Java🡪Code Generation，如下：  


AS掌握类成员的信息，因而允许在类中定义代理方法并将行为代理到类成员的方法，比如在为类中有一个成员变量为private List<String> names，按下Alt + Insert，选择Delegate Methods，选择names，再选择需要代理的方法即可，如果需要代理多个方法可按住Ctrl键选择多个方法。

## 3.5 插入动态模板

在许多IDE中，生成的代码只是从模板中粘贴过来，并未考虑作用域，AS的模板是作用域敏感的，而且还可以集成变量数据。

创建AS模板：File🡪Settings🡪Editor🡪Live Templates🡪选择一个模板分组，如选择plain分组，然后点击右上角的加号按钮，再选择Live Templates，然后填写Abbreviation、Description和Template text，然后点击窗口底部的Define按钮，这个按钮看起来像一个蓝色的超链接，点击它选择Java并选择所有范围（语句、表达式、声明等，这代表在Java源文件的任何作用域都有效），点击Apply按钮，如下：



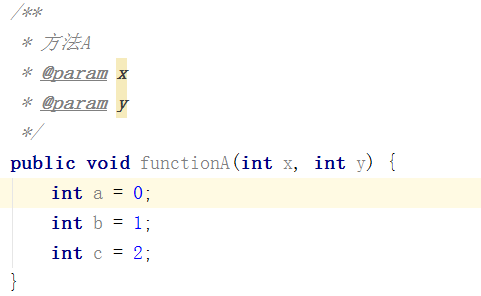
上图模板中的$SELECTION$是一个变量，表示选择的内容，在字段或方法的上方输入注释文字，然后选中这些文字，按Ctrl + Alt + J，选择cb即可自动生成注释。

* Insert Live Templates：Ctrl + J，激活作用域敏感的Live Templates列表，其实直接输入模板名称就会提示列表，如果不小心列表隐藏了，则再按Ctrl + J来显示。
* Surround with Live Templates：Ctrl + Shift + J，激活作用域敏感的Surround with Live Templates列表。在模板中的模板，如果作用域符合，并且模板中有变量$SELECTION$，则可以使用这个快键，而且有$SELECTION$变量的模板使用Ctrl + J是调不出来的。

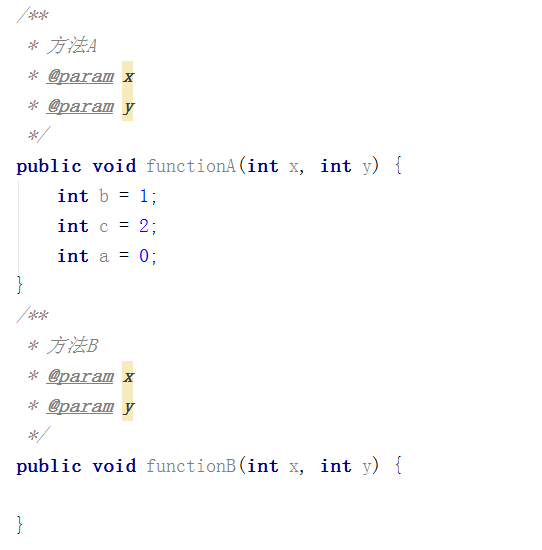
## 3.6 移动代码

Ctrl + Shift + 上下箭头移动，这种移动会限制在作用域范围内，

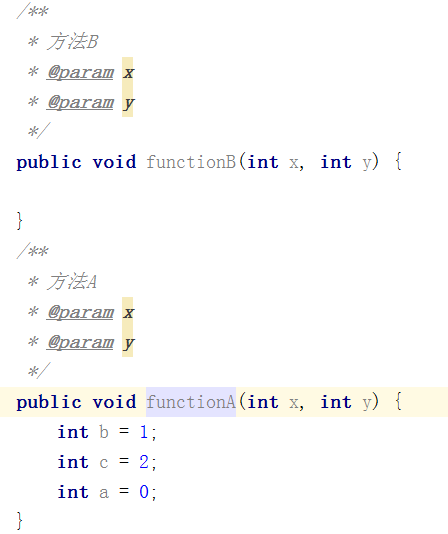
Alt + Shift + 上下箭头移动，这种移动没有作用域的限制，如下：



把光标定位在int a = 0这一行，然后按下Ctrl + Shift + 箭头下，会发现这一行代码可以往下移动两行到int c = 2的后面，再按就不会移动了，因为如果再往下移动的话就超出这行代码的作用域了。如果希望可以一直往下移动，则使用Shift + Alt + 下箭头。

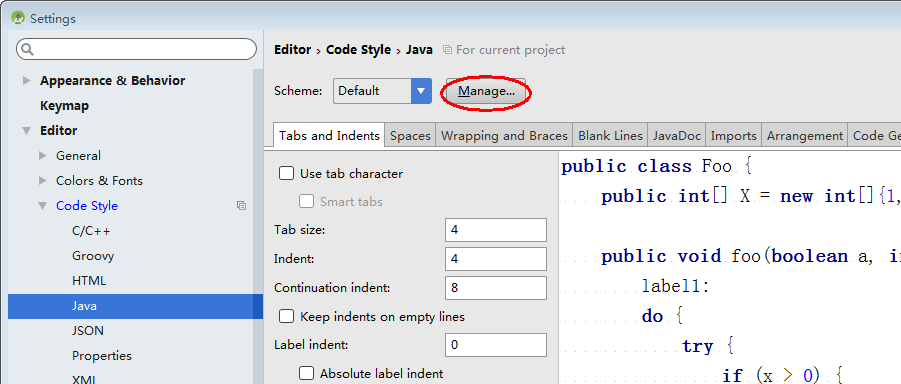


把光标定位在functionA的这一行，然后按下Ctrl + Shift + 箭头下，会发现连同注释一起移到了functionB下，如下：

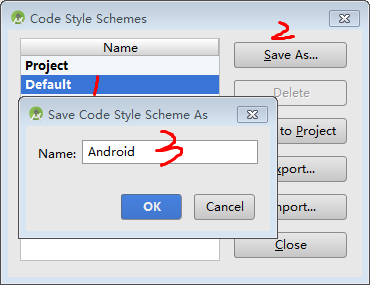


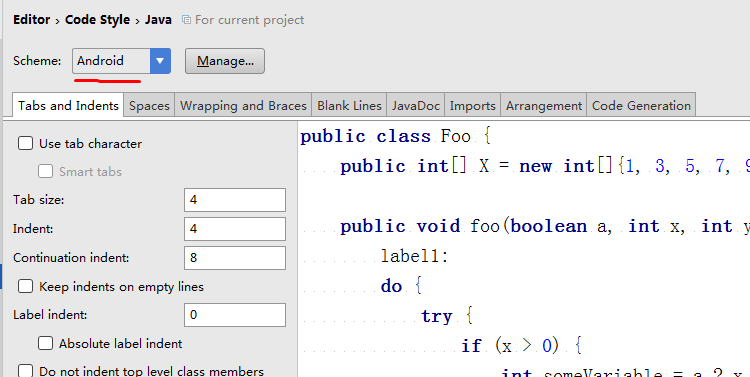
## 3.7 设计代码风格

代码风格约定处于不断演进之中。有关在方法后面应该留多少个空格，或者开始花括号应该与方法签名出现在同一行还是在下面一行，这些内容本来就没有定论。不同的组织倾向于定义其自身的代码风格，但每个程序员的代码风格也各不相同；况且你也可以有一种自己习惯的代码风格。AndroidStudio可以让你设计自己的代码风格，在File🡪Settings-🡪Editor🡪Code Style，Java和XML是我们在Android开发中需要关注的语言。操作如下：



点击“Manage。。。”，弹出如下对话框：





在这个对话框中，我们另存了一个配置，所以可以随意修改了。 当我们按Ctrl + Alt + L 来格式化代码时，上面的代码风格设置会被应用。

### 3.7.1 自动缩进行

快捷键：Ctrl + Alt + I，它会把选中行的代码进行缩进，缩进的规则是在File🡪Settings🡪Editor🡪Code Style🡪Java🡪Tabs and Indents中设置的。

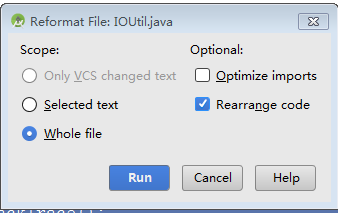
### 3.7.2 重新排版代码顺序

Arrangement管理着代码中元素的顺序。例如，大多数人喜欢将类成员声明置于类的顶部，接着是构造函数，然后是getter和setter，等等。可以在Arrangement选项卡中编辑Arrangement设置：File🡪Settings🡪Editor🡪Code Style🡪Java🡪Arrangement。

当需要重新排版代码顺序时，使用命令：Code🡪Rearrange Code即可。

### 3.7.3 重新格式化代码

这是最强大的Code Style操作，因为它会应用Code Style设置中定义的所有代码风格。快捷键为Ctrl + Alt + L，经本人试验，该快捷键并没有应用Code Style中的所有设置，如果想要应用更多，可以按Ctrl + Shift + Alt + L，这时会弹出一个对话框，可选择应用更多的Code Style，如下：



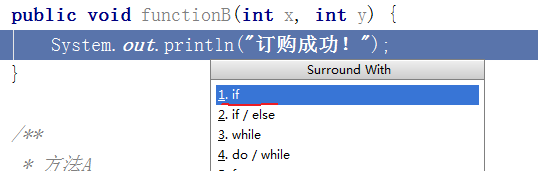
* Selected text：对选择的文本生效
* Whole file：整个文件生效
* Optionze imports：删除没有使用到的导入语句，这个优化可直接按Ctrl + Alt + O
* Rearrange code: 重新排版代码格式，这个优化可点击菜单Code🡪Rearrange code

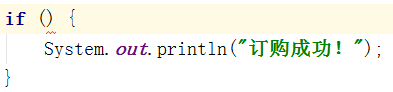
### 3.7.4 Surrond with

Surround With（Ctrl + Alt + T），它是Surround with Live Template（Ctrl + Alt + J）功能的超集。不过，Surround with还包含使用Java块（例如if/else、for、try/catch等，这些不属于模板）来包裹选中的一条或多条语句的选项。使用try/catch块包裹可能会抛出异常的语句是Surround with的最常用的应用场景之一，这或者是Ctrl + Alt + T 使用T的原因。

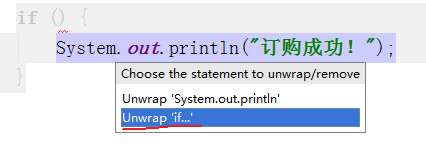
Ctrl + Shift + Delete则是删除使用的代码块，示例如下：

选中需要应用代码块的代码，如果是单行，则可以在单行上单击，然后就可以按下Ctrl + Alt + T了，如下：





在if块内任意位置单击，然后按下Ctrl + Shift + Delete，如下：



结果if块被移除，如下：

 。

我们说Surround With是Surround with Live Template的超集，意思是Surround With包含了Surround with Live Template的功能，如，我们有一个names为变量名的集合，输入这个变量名，然后按下Ctrl + Alt + T，上面是代码块，下面则是 Live Template模板，如下：



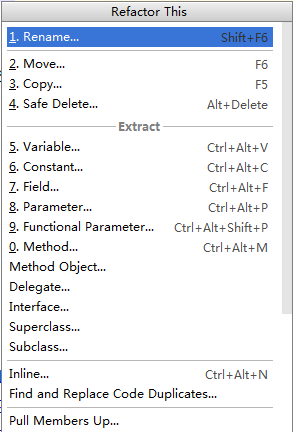
# 第四章 代码重构

重构代码的最大风险是可能会引入意外的错误，AndroidStudio通过分析某些具有危险性的重构操作来降低这些风险，并直接激活Find工具窗口，其中标识出了所有的错误和冲突，使你可以在提交之前预览所做的修改。尽量不要使用蛮力来重构，如全局查找替换，这种情况下AndroidStudio无法保证不会引入错误。如果使用AndroidStudio来重构，则AndroidStudio会帮你分析有哪些危险并处理掉，如在Project工具窗口中将某个Java源文件从一个包拖到另一个包中会触发一次Refactor🡪Move操作，它会分析移动操作所产生的影响，让你预览修改，并接着优雅地将整个项目中所有针对该类的import语句修改为新包名的完整路径。

大多数的重构操作都局限于一个方法或类中，因此不太可能向项目中引入错误。有风险的重构操作是指那些涉及两个或更多个资源的操作。如果重构操作引入了编译错误，Inspections Manager将会在Editor中使用红色标签标识出受影响的资源。在这种情况下，可以尝试修改它们，或者通过按Ctrl + Z来简单地撤消整个重构操作。如果重构操作成功且没有编译错误，但却涉及大量资源，那么仍然需要进行测试，以验证没有引入任何运行时错误。第11章涵盖了有关测试的内容。

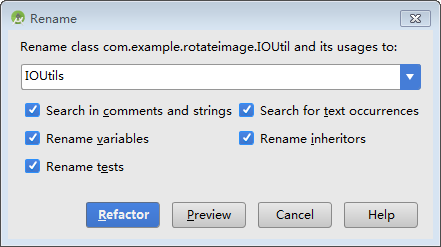
提示： 应该将所有重要的重构修改单独做一次Git提交，以便之后可以轻松地回退。第7章涵盖了有关git的内容。

在开始探讨个别重构操作之前，我们需要指出AndroidStudio有一项极其便利的重构操作，称为Refactor->Refactor This，此操作会显示一个上下文菜单，其中囊括了最有用的重构操作，它对应的快捷键为：Ctrl + Alt + Shift+ T，重构的命令很多，如果记不住的话记这一个就够了，界面如下：



## 4.1 重命名

在Project工具窗口中选中某个.java文件，按下Shift + F6，出现的对话框允许重命名类，以及该名称在注释、测试用例和继承类中所有出现的地方



在代码中的类上定位一下，然后按Shift + F6，这样的方式也可以重构类名，但是不会出现上面的对话框。

Search in comments and strings：在注释中和代码的String字符串中，如果出现与类名一样的字符，则也可以重命名

Rename variables：重命名量名

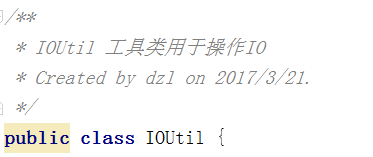
Rename tests：重命名测试用例

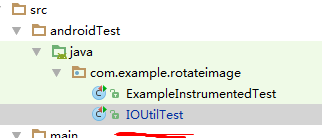
Search for text occurrences：暂不知道干嘛用的

Rename inheritors：重命名继承者的类名

演示如下：

在项目中有IOUtil.java文件，如下：



有该类的测试用例类，如下：  


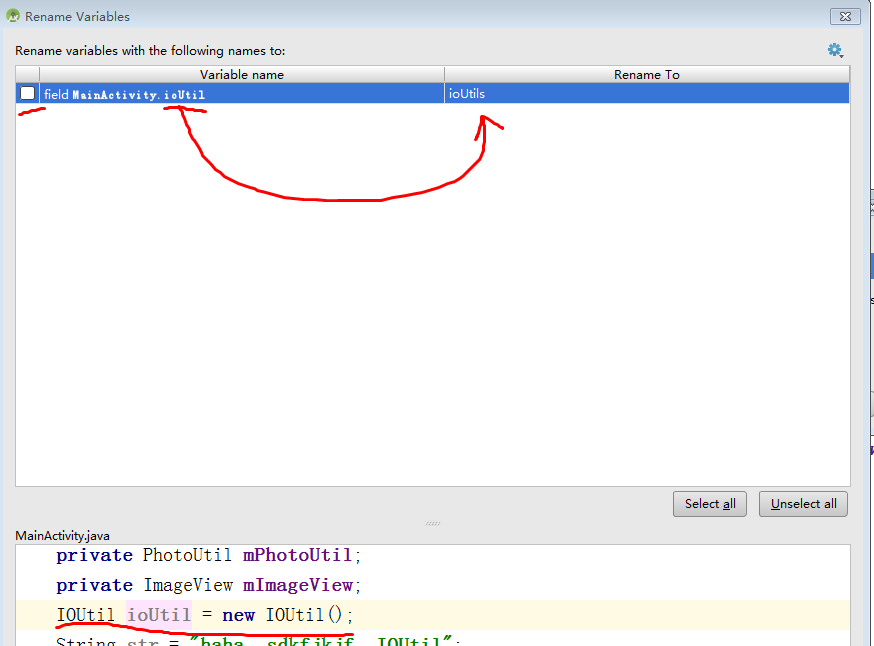
有该类的继承类：  


在MainActivity中有使用到MyUtil，并且在String中有IOUtil的字符：

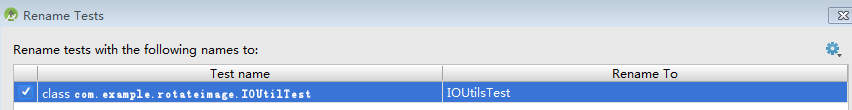


在Project工具窗口中选中某个IOUtil.java文件，按下Shift + F6，勾选如下：

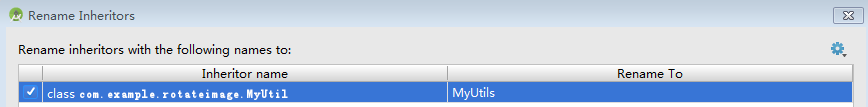


接下来将会出现一系列的对话框来显示可重构的地方，如下：  


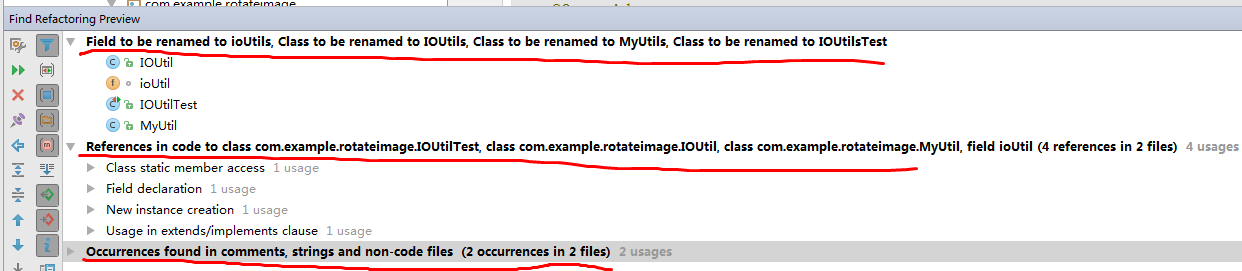
标题告诉我们这个操作是要Rename Variables，下面有一个复选框，打上勾代码要重命名，不打勾则不会，上图中显示字段MainActivity.ioUtil会被重命名为ioUtils，下方窗口有显示IOUtil 变量的源代码。点击OK，将出现下一个对话框，如下：



这是重命名测试用例的类名，把IOUtilTest重命名为IOUtilsTest，点击OK，出现下一个对话框，如下：



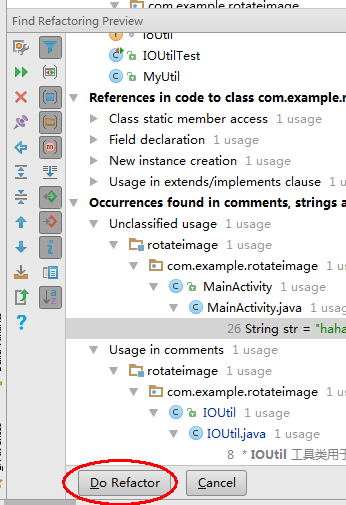
这是重命名继承类的类名，把MyUtil重命名为MyUtils，点击OK，这时不会再出现对话框了，出现了一个重构预览界面，如下：



上图中画了3条红色，中间红色的内容不需要看，它是把引用到IOUtilTest、IOUtil、MyUtil、变量ioUtil这4个我们需要重构的东西所在的地方给我们列出来，这个看不看无所谓。主要看第一行红色，它告诉我们字段会被重命名为ioUtils，类会被重命名为IOUtils，类会被重命名为MyUtils，类会被重命名为IOUtilsTest，其实就是告诉我们哪些东西会被重构。 第3天红色线地方也是我们需要看的，这个地方是告诉我们在注释和代码的字符串中找到了IOUtil单词，这些单词也会被重命名，可展开查看具体内容，如下：

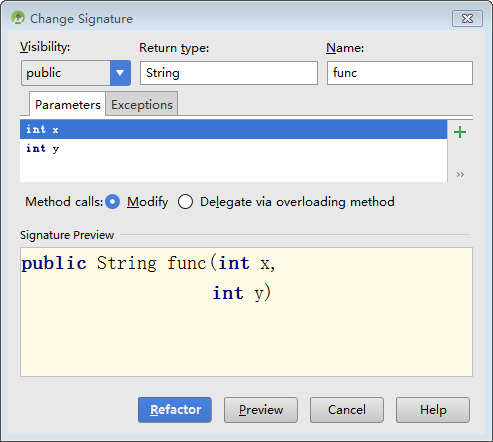


这告诉我们这两个地方的字符串也会被修改，双击红色行可跳转到这些字符串所在源文件位置。最后点击“Do Refactor“按钮执行重构，如下：



## 4.2 修改签名

快捷键：Ctrl + F6 ，可以修改方法的可见性、名称、返回类型、参数和抛出异常。把光标定位在方法名上，然后按下Ctrl + F6(Refactor🡪Change Signature)，出现如下对话框：

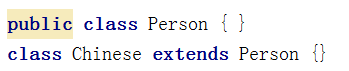


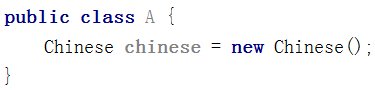
在Method calls选项中，如果选择“Delegate via overloading method”，它是给你生成一个新方法，对原来的方法没有任何影响。

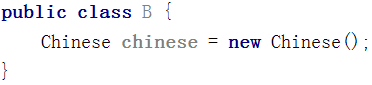
## 4.3 类型迁移

Refactor🡪Type Migration允许把一种Java类型改为另一种。假设创建了Chinese类，如果想要将Chinese的所有实例修改为Person，那么Type Migaration很容易实现这一点。

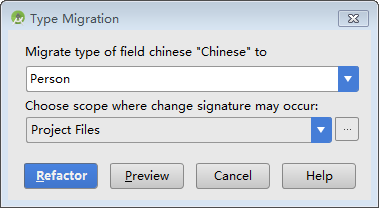
有如下代码：







现要将A和B中的Chinese实例换成Person，在A类中的Chinese上定位光标，然后按下Ctrl + Shift + F6，出现如下对话框：



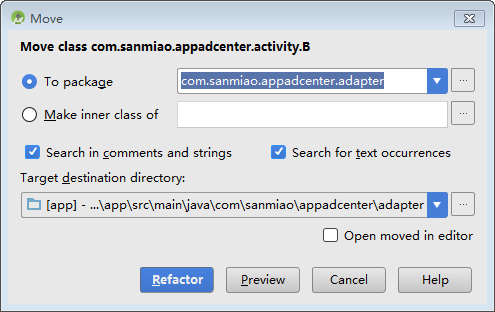
目前发现这个功能并不管用，只能把当前的一个Chinese变成Person。

## 4.4 移动

有三种方法可以移动源文件：

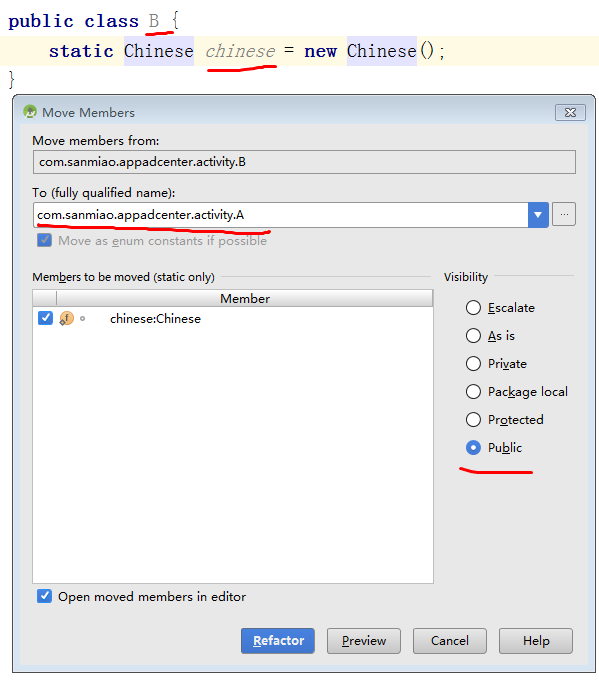
1. 在Project窗口中把源文件从一个包拖曳到另一个包下
2. 选择该源文件并从主菜单中选择Refactor🡪Move
3. 在Project窗口中选择文件，并按F6键

这些操作都会弹出如下窗口：



如果选择Make inner class of，则可以把选择的类变成另一个类的内部类

除了移动类，还可以移动静态的成员变量，把一个类的静态成员变量变成另一个类的静态成员变量，在静态成员变量的变量名上定位，然后按F6，如下：



如上图，我们把B类中的静态成员变量chinese移到了A类中，并且把可见性设置为Public，点击Refactor之后则B类中的chinese成员变量则没有了，变到A类中去。 可见性中的“Escalate”和“As is”暂时不知道起什么效果。

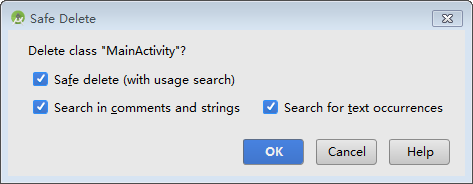
## 4.5 复制

有3种方式

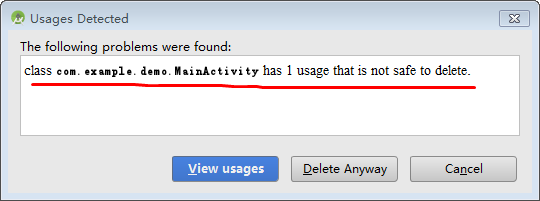
1. Ctrl + C与 Ctrl + V
2. 选择文件🡪Refactor🡪Copy
3. 选择文件🡪F5

## 4.6 安全删除

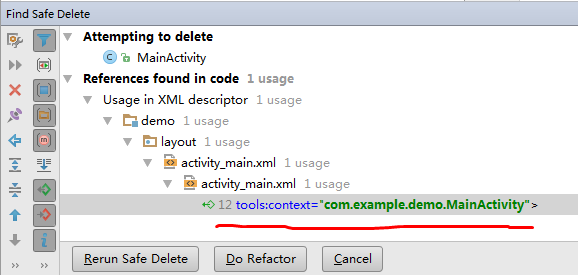
在Project窗口选择MainActivity，按下Delete键，弹出如下对话框：



点击OK后出现如下对话框：



上图红线意思是MainActivity有一个使用不能安全删除，点击Delete Anyway表示不管如何都要删除，点击ViewUsages可查看有什么使用，如下：

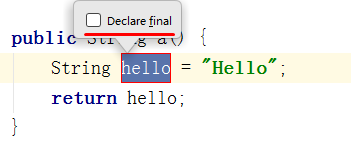


这说明在activity\_main.xml中有对MainActivity的依赖，点击Do Refactor执行删除。

## 4.7 抽取

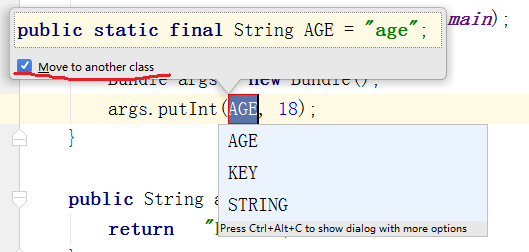
### 4.7.1 抽取局部变量

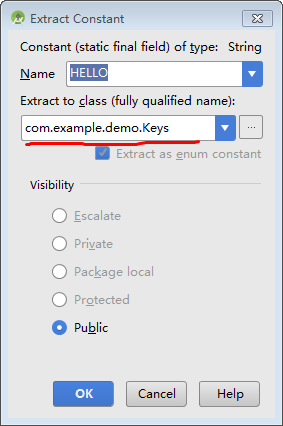
Ctrl + Alt + V (Variable)，在出现的对话框中可以选择是否要抽取成final类型的局部变量，如下：



### 4.7.2 抽取常量

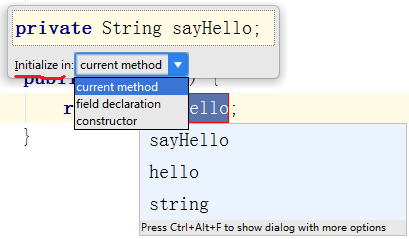
Ctrl + Alt + C (Constant)，在出现的对话框中可以选择这个常量是否要声明到其它类中去，如下：





### 4.7.3 抽取字段

Ctrl + Alt + F (Field) ，在出现的对话框中可以选择初始化字段的位置，如下：



Current method：在当前方法中进行初始化

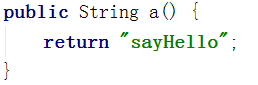
Field Declaration：在声明字段的地方进行初始化

Constructor：在构造方法中进行初始化

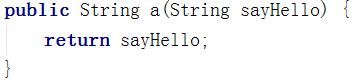
### 4.7.4 抽取参数

Ctrl + Alt + P (Params)，把方法中的局部变量或字段抽取成参数。

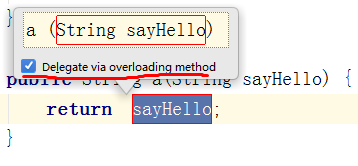
有如下方法：

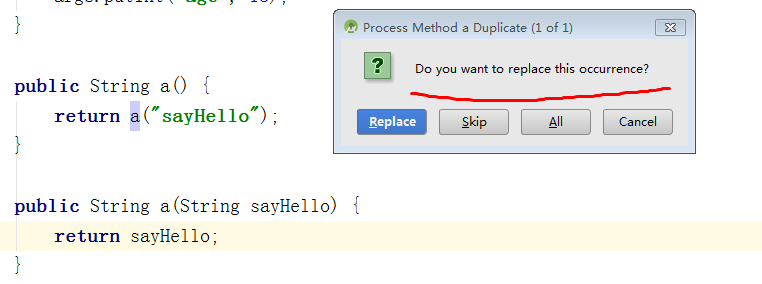


对“sayHello”进行参数抽取，效果如下：



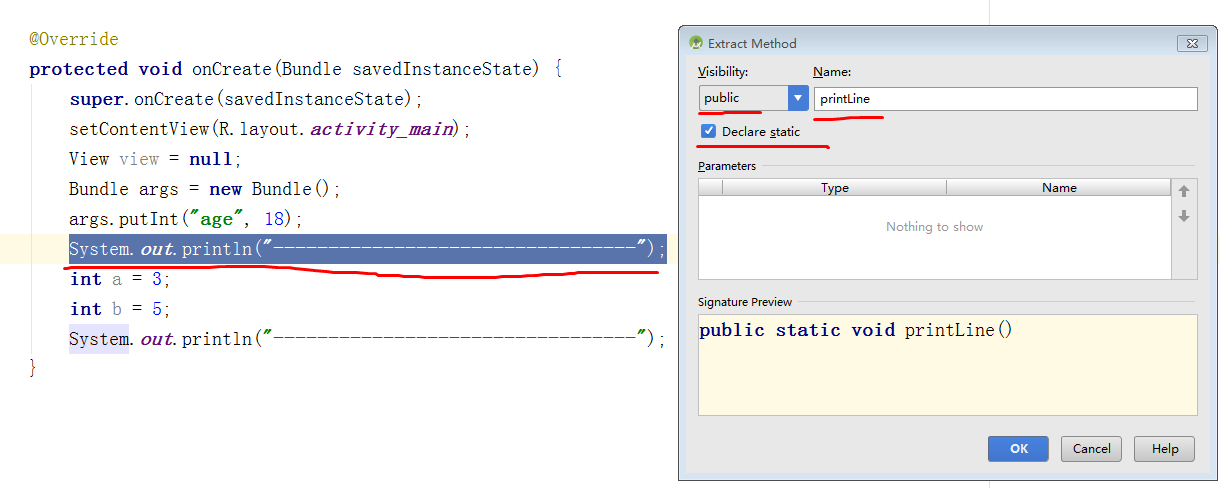
在抽取的时候，如果勾选“Delegate via overloading method”，则生成一个重载方法，原方法不变，如下：





上述对话框中，点击“Replace”、”Skip”、”All”发现效果一样，不知道有什么区别。

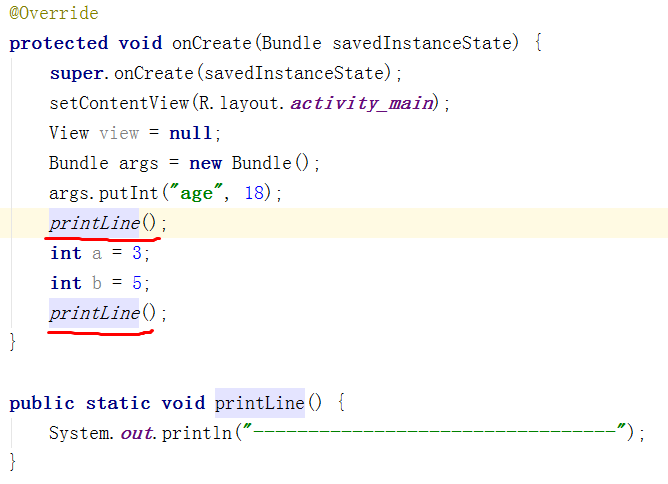
### 4.7.5 抽取方法



“Visibility”选择方法的可见性，”Declare static”把方法设置成静态，点击上图的OK，出现如下提示：



大概意思为Studio中有1个代码片断和我们现在抽取的代码片断是一样的，可以使用我们抽取的方法替换这个代码片断，你想要检验并且替换它吗？我们点击Yes，效果如下：

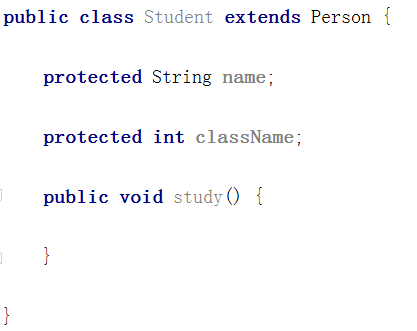


## 4.8 高级重构

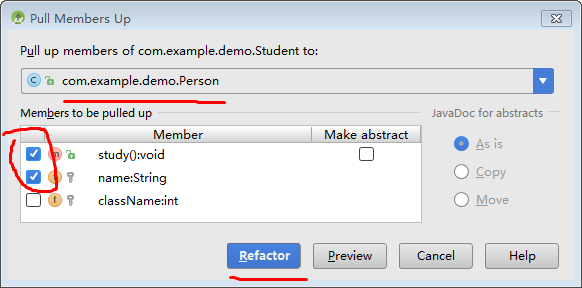
### 4.8.1 下推成员/上拉成员，成员包括字段和方法

有如下两个类，Student继承了Person：

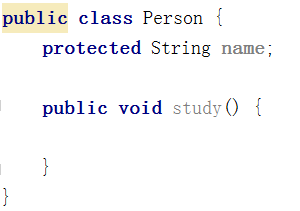


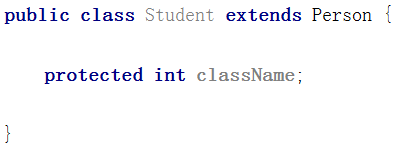


我们发现，人都有名字，人都会学习，所以Student类中的name成员变量和study 成员方法应该属性Person类，把光标定位在Student类的任意位置，执行：Refactor🡪Pull Members Up，出现如下对话框：



选择要提升到的父类（Person），并选勾先需要提升的成员（studey与name），然后点击Refactor，效果如下：

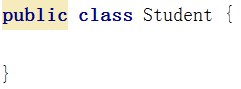




如果觉得父类中的某些成员只属于某个具体的子类，则执行：Refactor🡪Push Members Down

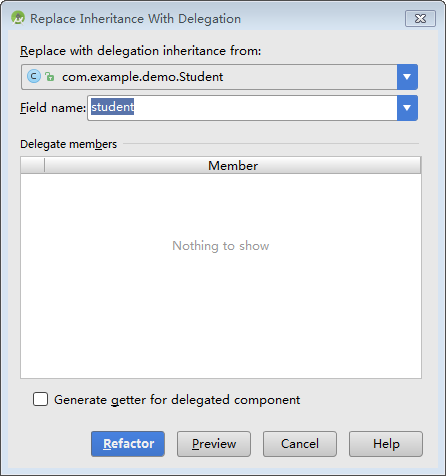
### 4.8.2 使用代理代替继承

有如下两个类：

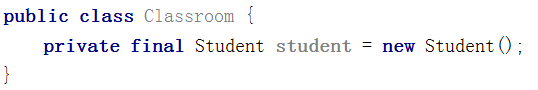




我们发现Classroom不是Student，而是包含Student，要修改这个关系，先把光标定位在Classroom类的任何位置，然后执行：Refactor🡪Replace Inheritance with Delegation，效果如下：

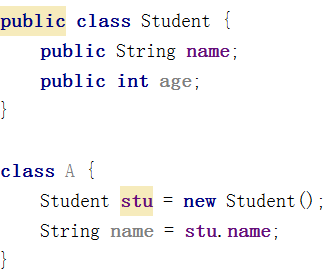


这里可以给Student字段起一个名字，最终效果如下：

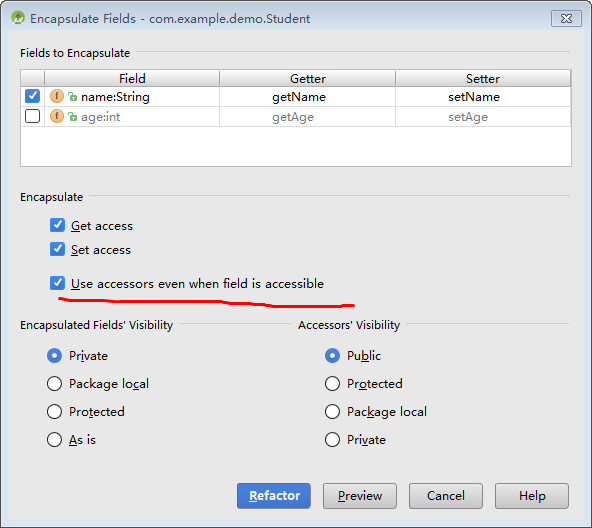


### 4.8.3 封装字段

封装是一种面向对象策略，它通过将字段的访问级别设为私有来隐藏它们，接着通过公共可性的getter/setter方法为这些字段提供公开接口。Refactor🡪Encapsulate Fields类似于Code🡪Generate🡪Getter and Setter，只不过是Encapsulate Fields提供了更多的选项，把光标定位在类的任意位置，然后执行：Refactor🡪Encapsulate Fields，Encapsulate单词意思为封装。效果如下：



点击Refactor按钮后，效果如下：

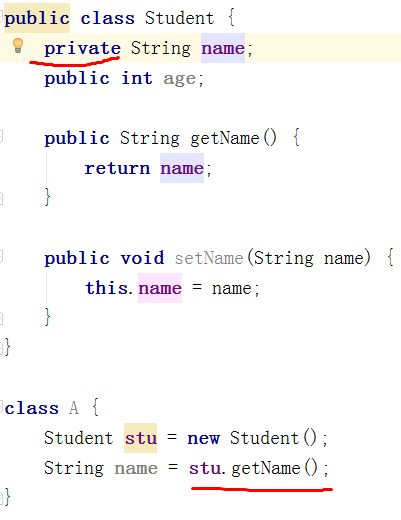


选择需要封装的字段， Encapsulate有3个选项：

* Get access 产生getter方法
* Set access 产生setter方法
* Use accessors even when field is accessible 不知道干嘛用的

Encapsulate Fields’ Visbility 可以修改字段的可见性，As is的意思为照现在的样式，即当前字段的可见性保持不变

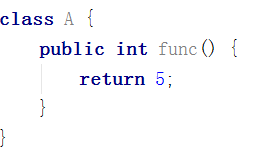
Accessors’ Visibility 可以修改setter/getter方法的可见性



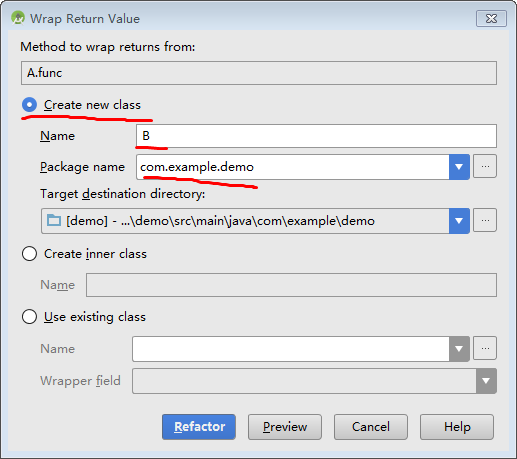
从上面结果中红色画张中可看出，Encapsulate Fields功能可以修改原字段的可见性，并且会把调用了name属性的地方改为调用对应的getter方法，而Code🡪Generate🡪Getter and Setter是不能修改字段可见性的，也不会把调用了name属性的地方改变getter方法调用。

### 4.8.4 封装方法返回值

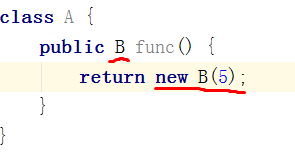
Refactor🡪Wrap Method Return Value

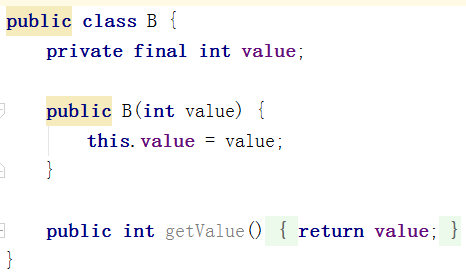


在func方法中定位一下，然后执行重构命名，出现如下对话框：



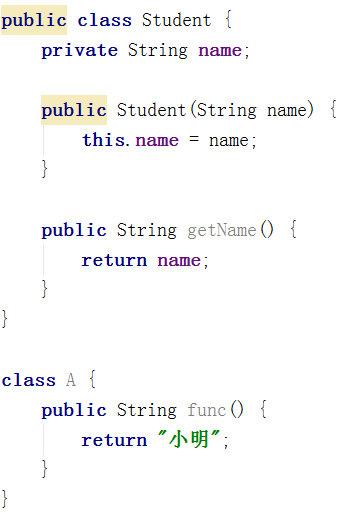
包装方法返回值有3种方式，Create new class方式为创建一个新类来包装返回值，点击“Refactor”按钮，效果如下：



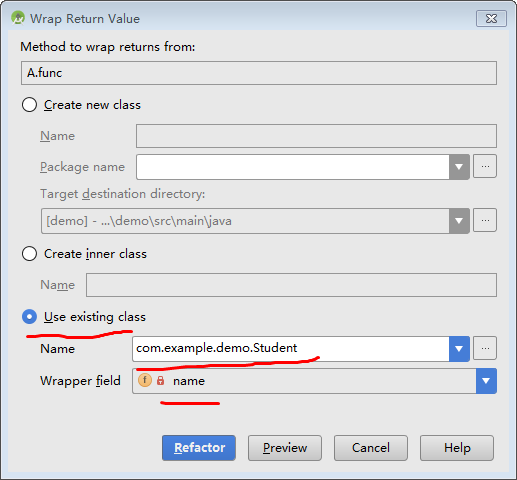


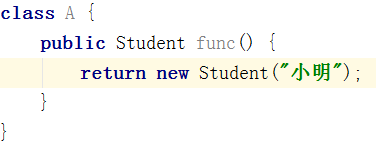
Create inner class为创建内部类来封装方法返回值

Use existing class 为使用已经存在的类。 从生成的自动生成的B类的截图来看，如果要使用已经存在的类，则结果要与上面的B类相似，即要有对应类型的字段，且要有接收这个字段的构造方法，并且要有获取这个字段的getter方法，如下：



在func中的返回值可用已经存在的Student类进行包装，效果如下：

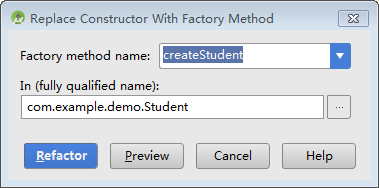


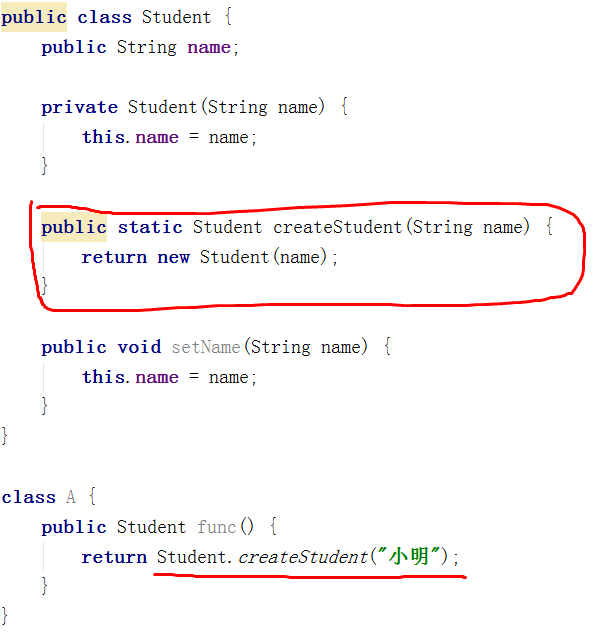


### 4.8.5 使用工厂方法代替构造函数

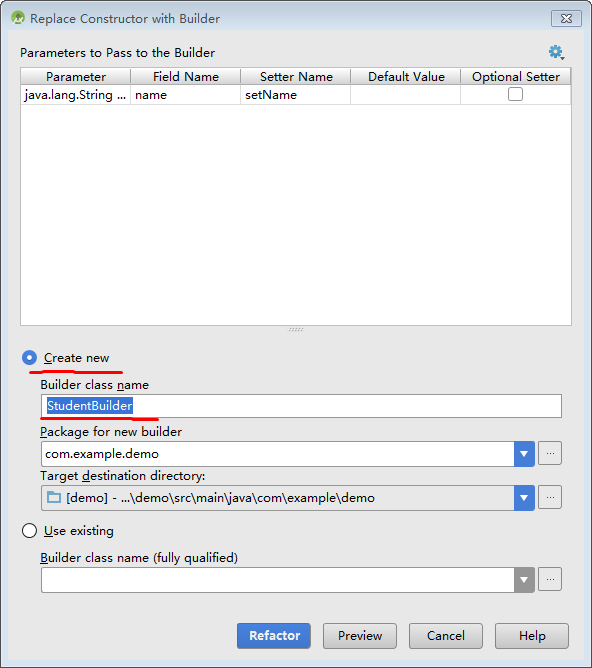
有如下类：

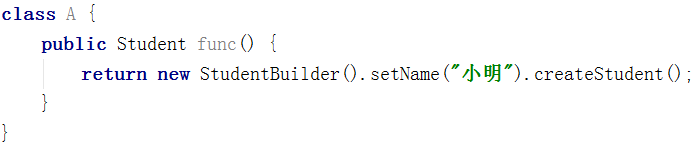
把光标定位在构造方法名上，然后执行Refactor🡪Replace Constructor with Factory Method，如下：

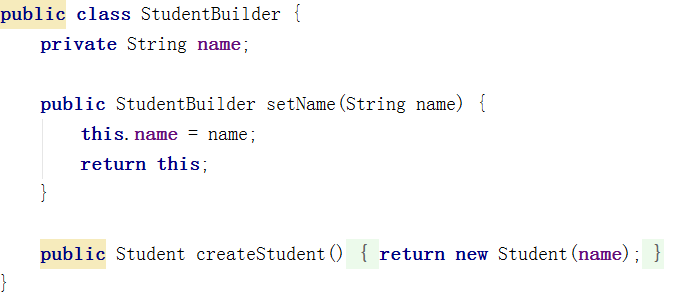




有一个类似的命令：Refactor🡪Replace Constructor with Builder，效果如下：



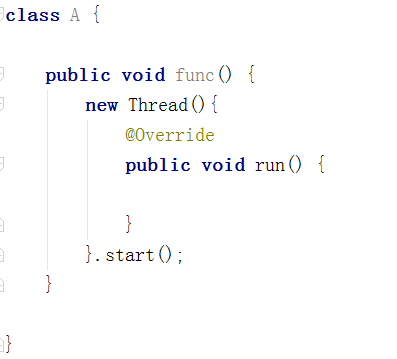




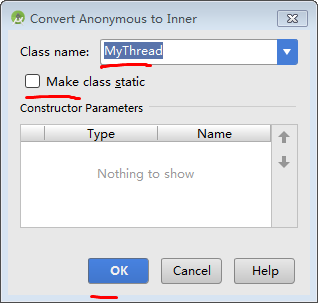
上面对话框还有一个“Use existing”的选项，可以选择使用已经存在的Builder类来替换，类的内部结果要与上面的StudentBuilder类的内部类似如Stuent的构造方法接收的参数名是name，则要求你的Builder类中有name属性与setter方法，和createStudent方法，如果没有，则AndroidStudio会自动创建类似的结构。

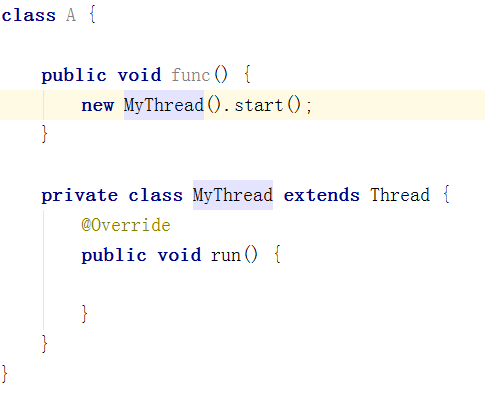
### 4.8.6 将匿名类转换为内部类

有如下代码：



把光标定位在匿名类的任意位置，然后执行：Refactor🡪Convert Anonymous to Inner，效果如下：



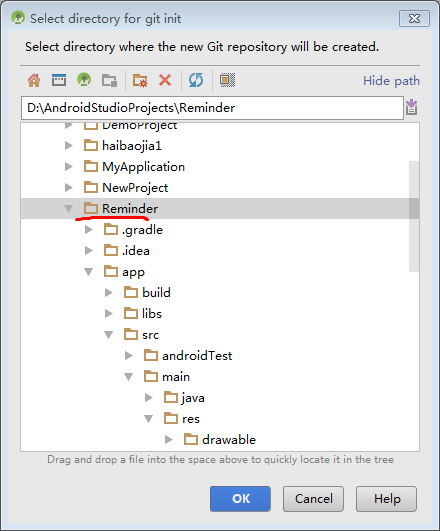


# 第5章 备忘录实验：第1部分

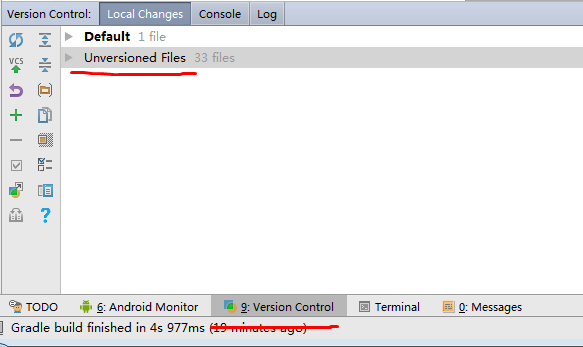
## 5.1 创建新项目

## 5.2 初始化Git仓库

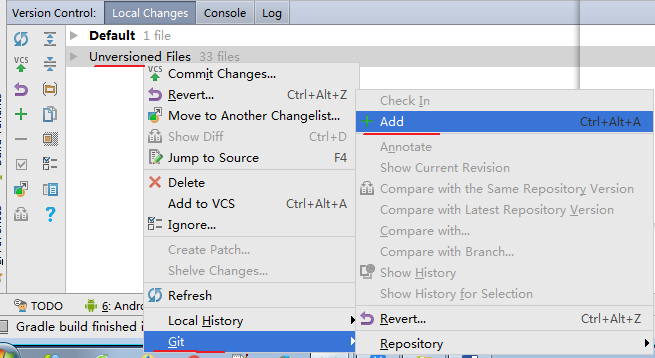
创建新项目之后的第一步应该是使用版本控制来管理源代码。执行：VCS🡪Import into Version Control🡪Create Git Repository，这时会弹出一个选择目录以便git进行初始化的对话框，我们选择项目的根目录，如下图：



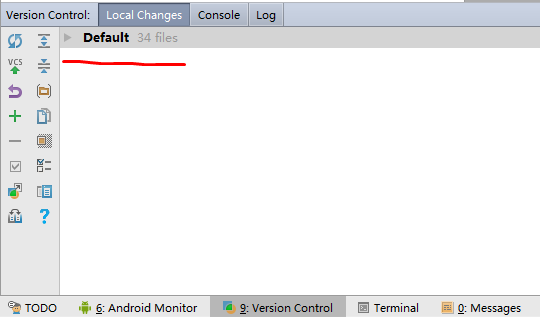
执行后发现Project窗口中的大多数文件均变为了棕色，这意味着它们已经被Git跟踪，但尚未添加到Git仓库中。 打开窗口底部的Version Control工具窗口，如下：



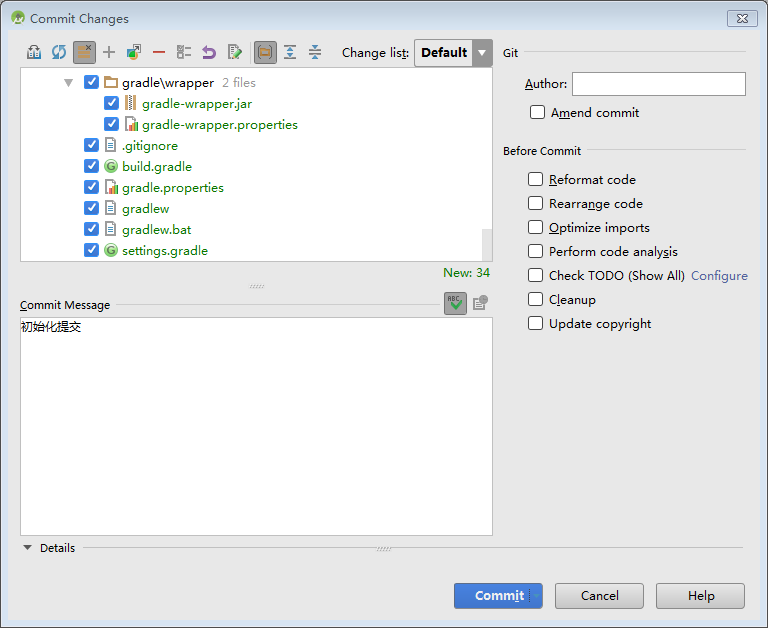
这里显示有33个文件没有加入到Git的版本控制，右击Unversioned Files，如下：



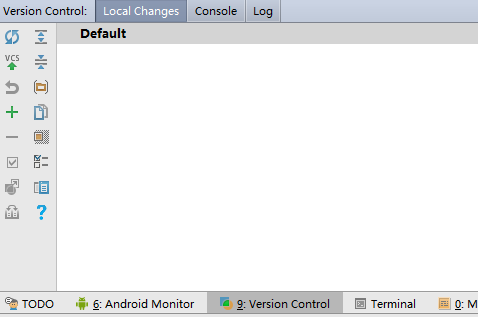
这个操作可以直接按快捷键Ctrl + Alt + A，执行后发现文件都变成了绿色，而且归类到了Default分类中，如下：



加入到版本控制之后就可以提交了，按Ctrl + K，出现提交对话框，如下：

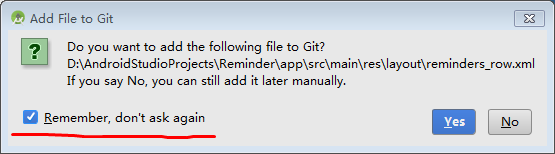


右边的Author留空，AndroidStudio将会使用在Git安装时设置的初始化默认值。取消Before commit 区域中所有复选框，在Commit Message中输入提交信息，然后点击Commit。提交后再看Version Control工具窗口，发现Default中的文件没有了，如下：



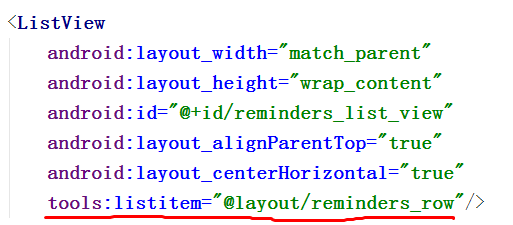
## 5.3 构建用户界面

在添加新文件时会出现如下对话框：



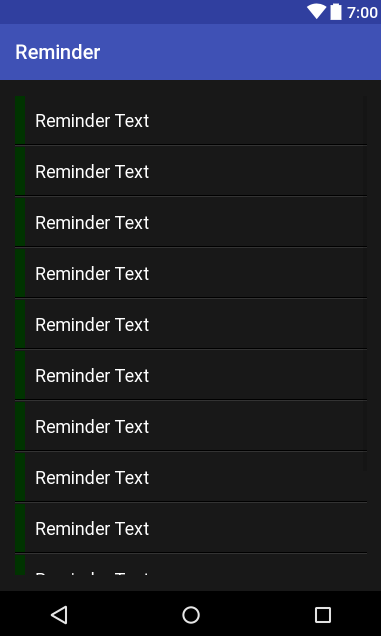
我们一定要勾选上“Remember, don’t ask again，这样有新文件时会自动加入到Git版本控制，不需要每次都询问。

让ListView在布局中预览的时候就能看到item的布局：



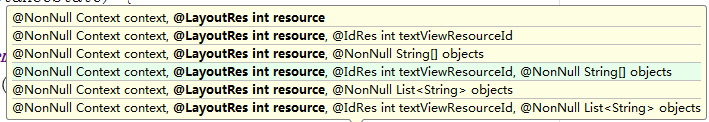
这只是预览，在运行的时候这个ListView并不会使用这个item布局，如要使用得在适配器中进行配置。

我当前用的是AndroidStudio2.3版本，奇怪的现象，这里把id属性删除，在预览的时候发现看不到item效果，有id与无id的预览效果如下：

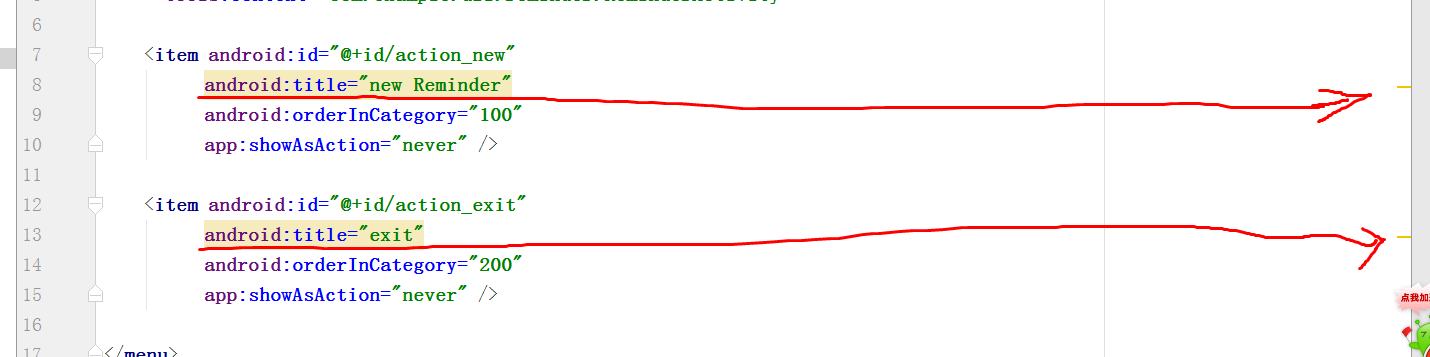
 

ListView中的Adapter是控制器，ListView是视图，SQLite数据库是模型，这就是MVC模式。

ArrayAdapter有多个构造方法，且有泛型。泛型如果是String，则显示这个String，如果不是，则显示这个泛型对象的toString。构造方法如下：



第一和第二个参数都是一样的，第三个参数，如果是IdRes，则是要指定把内容显示到哪个TextView上，如果这个参数，则这个item的布局根元素必须是一个TextView。



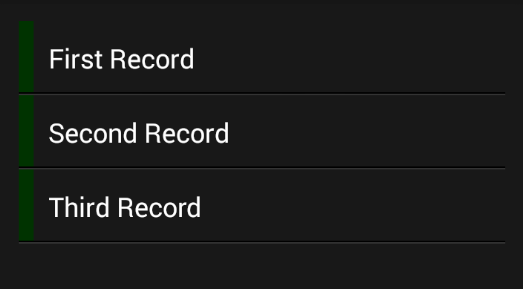
这里使用了硬编码，在右边的橙色标记就是警告，按F2在各警告之间进行跳转，再按Alt + Enter，在出现的IntelliSense建议中选择抽取字符串。

在Activity中有getLocalClassName()方法可以获取到 当前Activity的类名，在打log的时候特别方便

创建带斜面效果的线：

<**view  
 class="android.view.View"  
 android:layout\_width="match\_parent"  
 android:layout\_height="1dp"  
 android:background="#000"**/>  
<**view  
 class="android.view.View"  
 android:layout\_width="match\_parent"  
 android:layout\_height="1dp"  
 android:background="#333"**/>

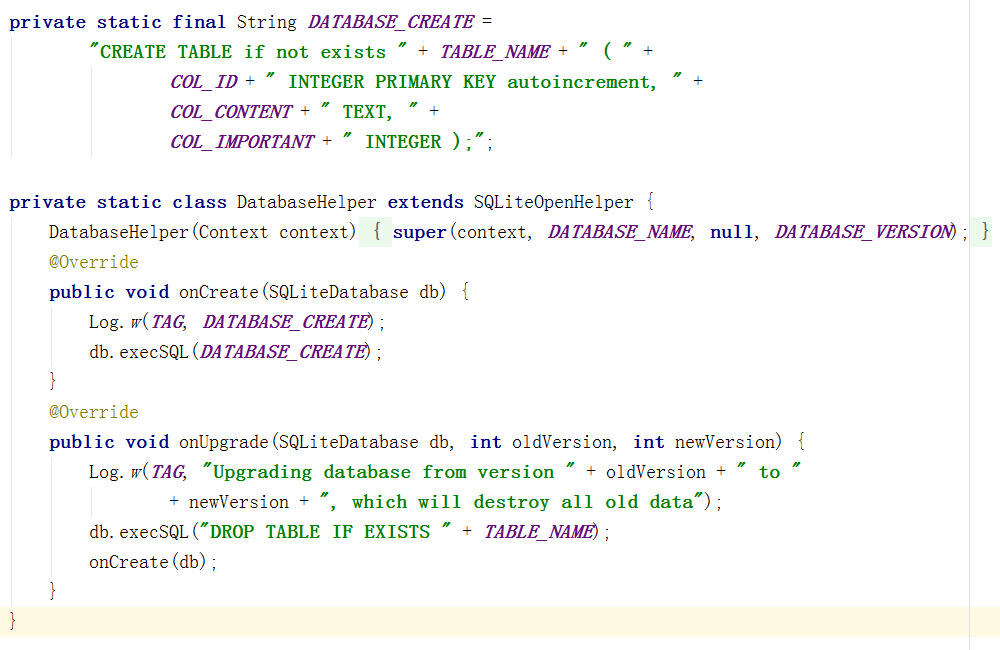
垂直排列这两条线，看起来就是一条，而且有斜面立体感，效果如下：



## 5.4 持久化备忘录

本节函盖了数据模型、数据库代理类和CursorAdapter，模型将保存从数据库读取以及写入数据库的数据，代理将会是一个适配器类，它把来自己App的简单调用转换为对SQLite数据库的API调用。最后，CursorAdapter将继承以抽象方式处理数据访问的标准Android类。

POJO（Plain Old Java Object，普通Java对象），其实就是一个标准的JavaBean

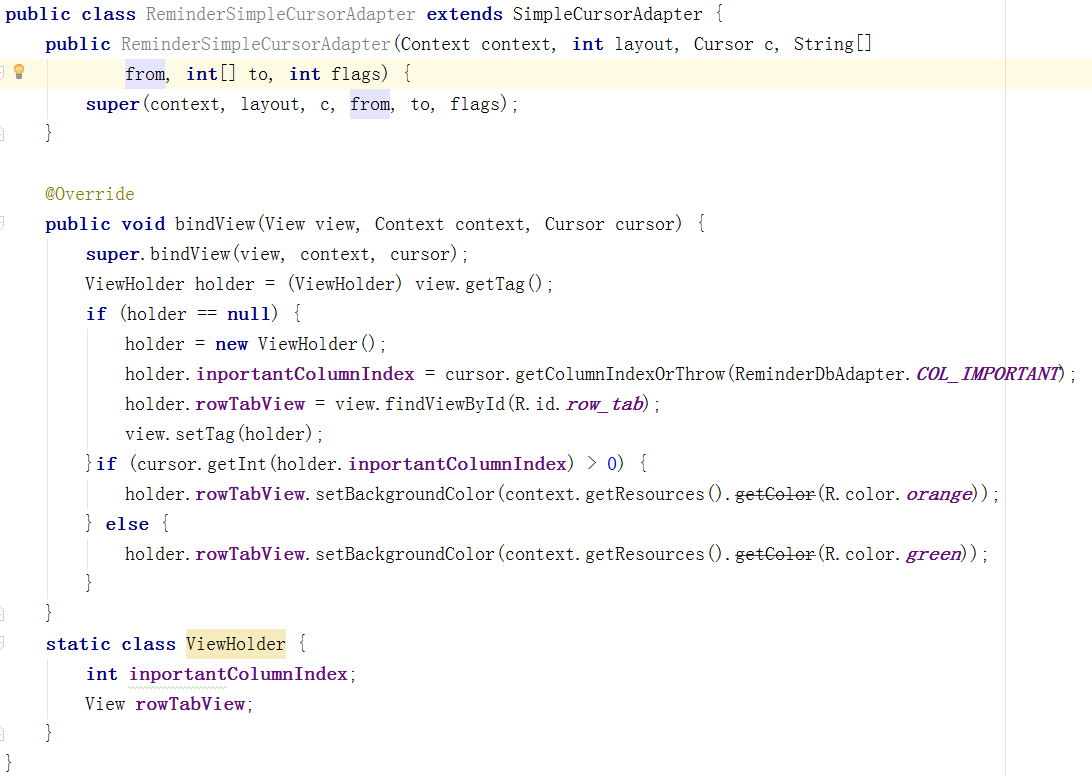


这里创建数据库表时是先判断了不存在再创建。而在更新时，先删除表，再重新创建表。

SQLiteOpenHelper的构造方法运行时即创建了数据库，数据库创建后会执行onCreate方法，这个方法只会执行一次。当需要更新数据库时，把DATABASE\_VERSION的值增加1，这样onUpgrade方法就会被执行。

CRUD不是代表增删改查，而是代表创建、读取、更新和删除。

SimpleCursorAdapter的模型类似于ArrayAdapter，只是SimpleCursorAdapter的数据来源于Cursor，它只是简单的显示文本，如果想要更改更多，可以继承这个类来实现，如下：文本的显示父类已经完成，这里是增加了修改item的一个tab的背景颜色，而且这里的ViewHolder不但保存了View，也保存了ViewHolder对应的位置的数据在Cursor中的索引，以减少获取索引的次数。



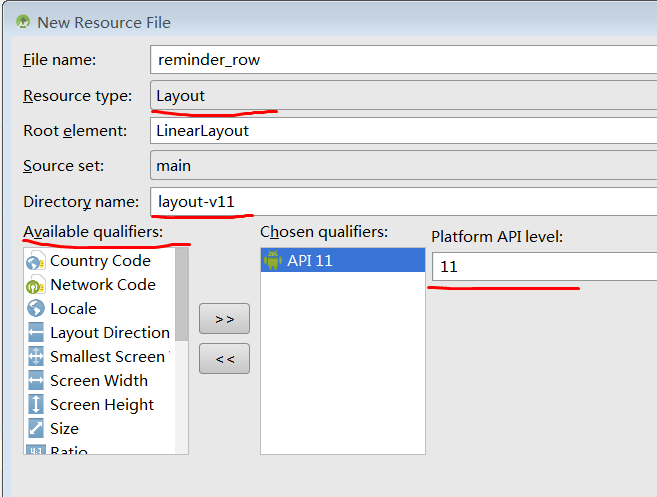
# 第6章 备忘录实验：第2部分

## 6.1 添加/删除备忘

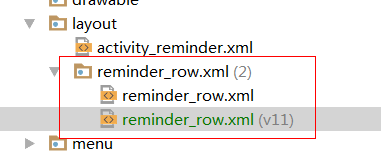
## 6.2 响应用户交互

## 6.3 提供多选上下文菜单

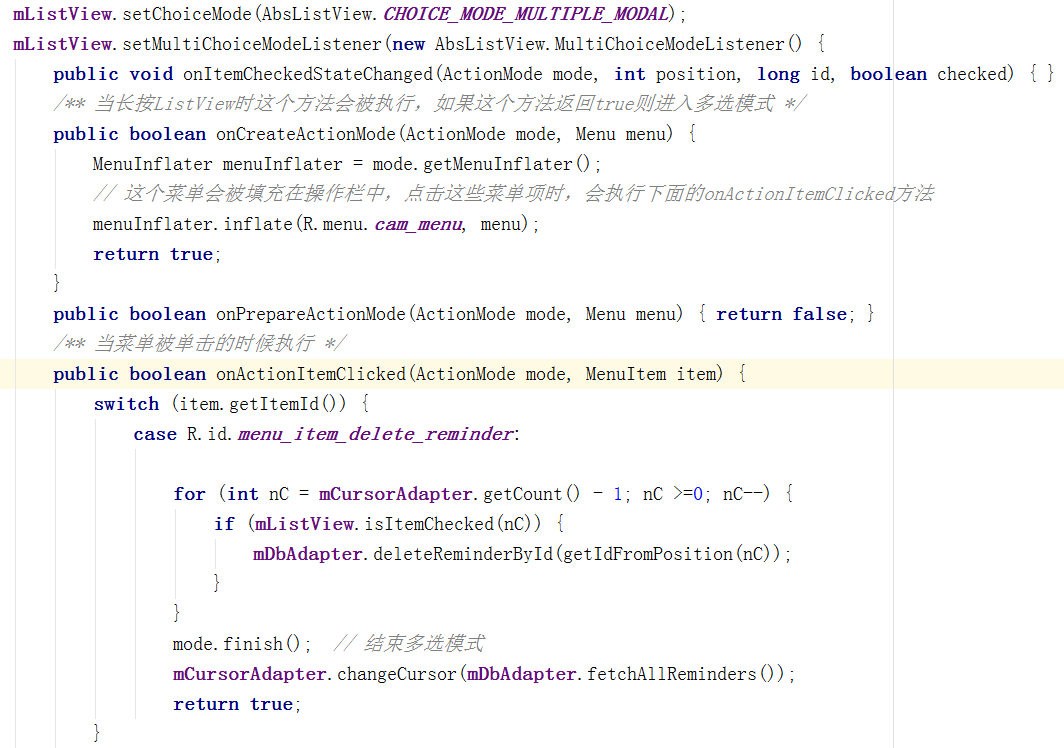
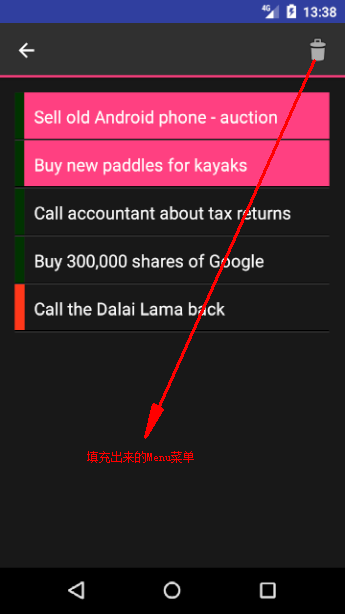
对于资源目录后面可以加限定符（限定这个目录的功能），比如屏幕适配，版本限定等的限定符，如果不记该怎么写，可以在创建资源的时候选择Resource type，则Directory name会自动生成，在Available qualifiers中选择需要的限定符，如下面是选择了Version，然后输入了版本11，代码我们现在创建的reminder\_row这个资源用于API11，当然对于这个资源我们在layout（没有修改）目录下也有一个，如果版本小于API11则系统会自动使用默认布局。



Project窗口的Android视图会把相同名字的资源组织到一起，方便高效的进行管理，在layout目录和layout-v11目录中都有相同名字的布局，如下：



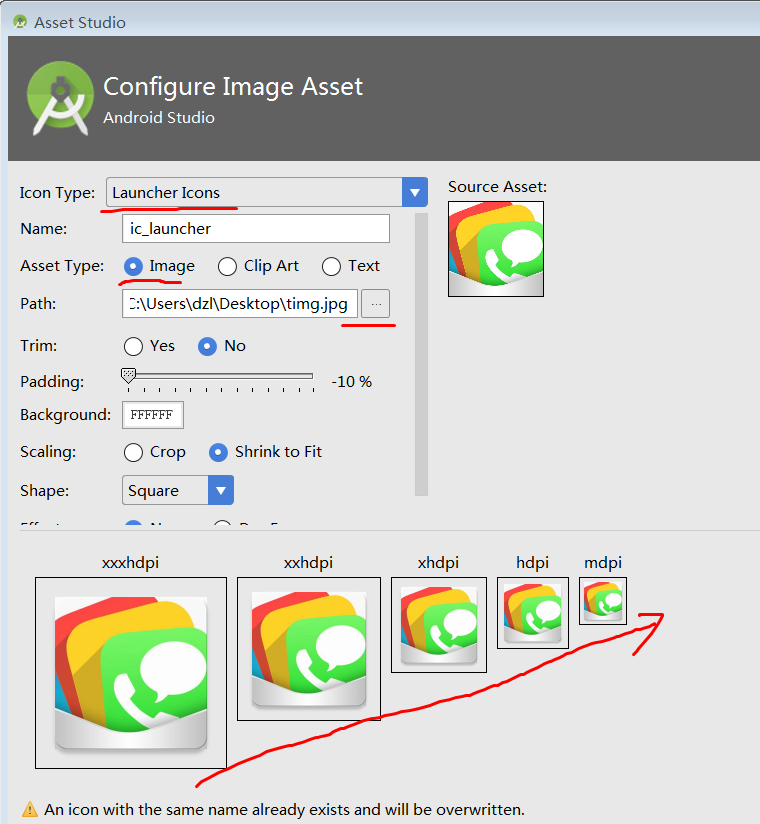
一个背景选择器

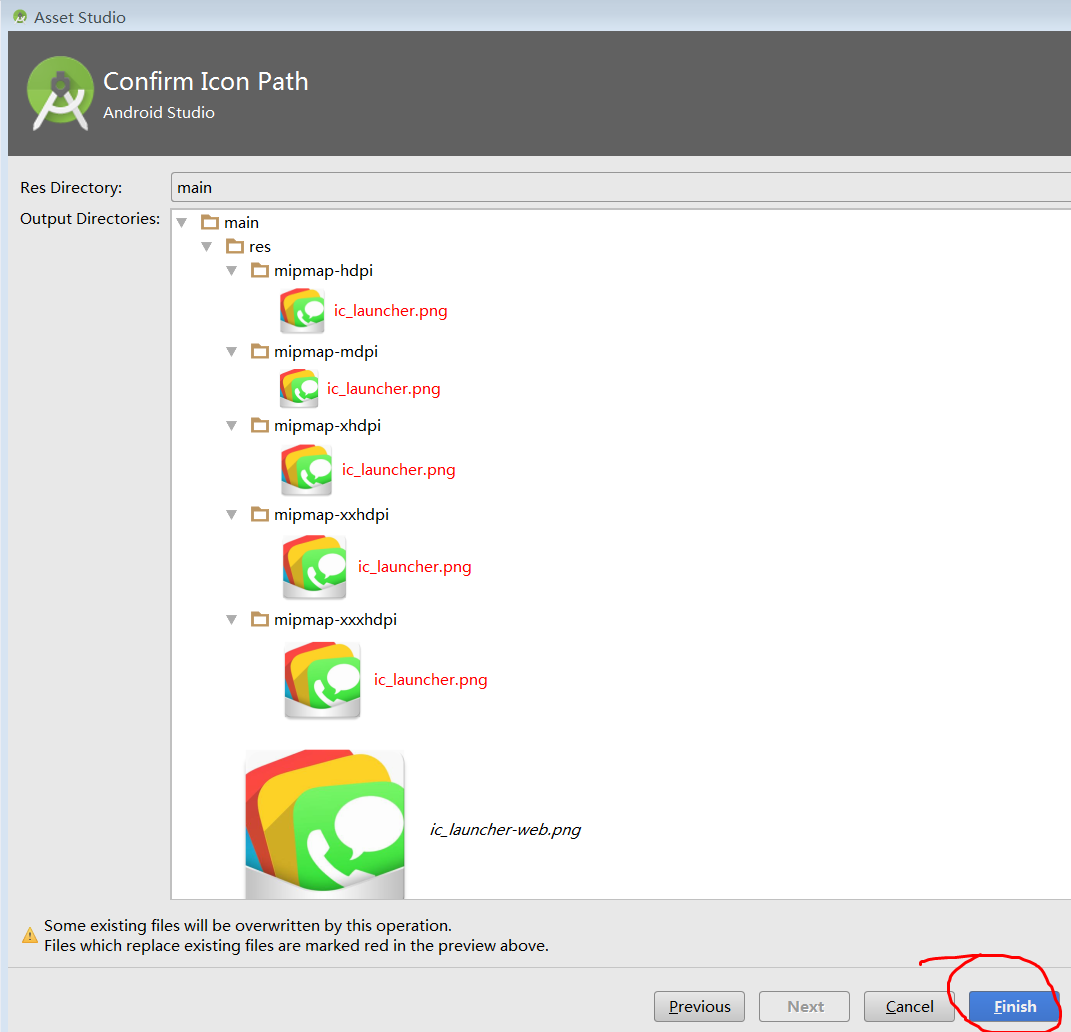
 

## 6.4 实现添加、编辑和删除

当需要快速替换应用图标时，可以这样操作：

在mipmap文件夹上右击：New🡪Image Asserts





设置ActionBar上显示一个图标：

ActionBar actionBar = getSupportActionBar();

actionBar.setHomeButtonEnabled(true);

actionBar.setDisplayShowHomeEnabled(true);

actionBar.setIcon(R.mipmap.ic\_launcher);

# 第7章 Git入门

## 7.1 安装Git

安装后设置自己的名字和电子邮箱，以便你提交时有相应的作者：

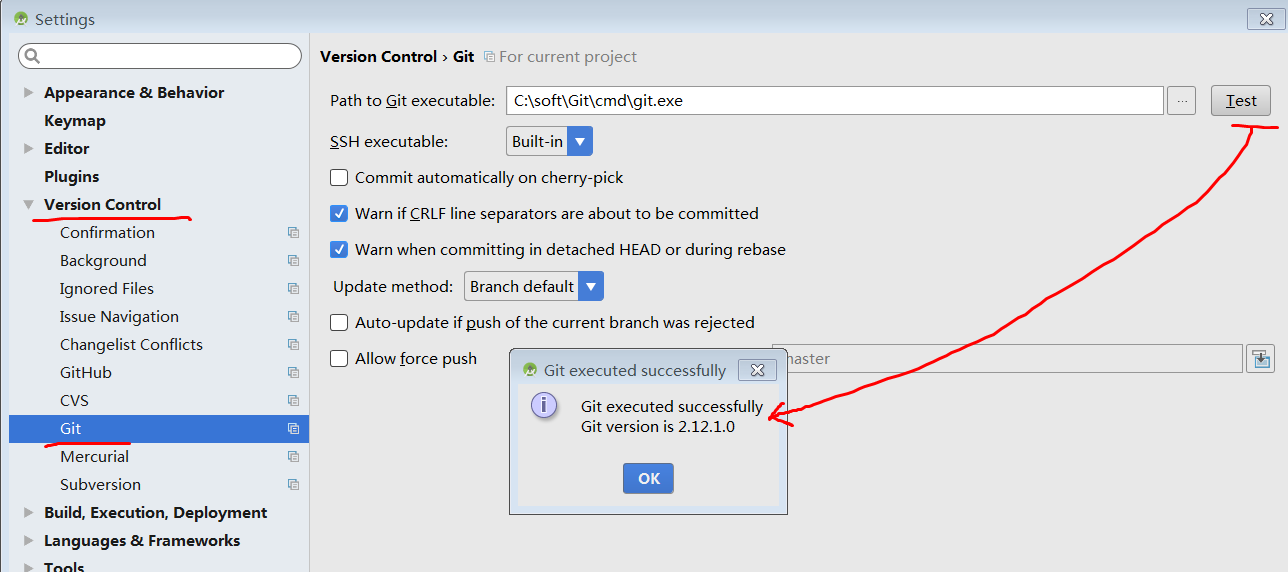
打开Bit Bash，然后输入下面的命令：

git config --global user.name "Event"

git config --global user.email dai\_zhenliang@163.com

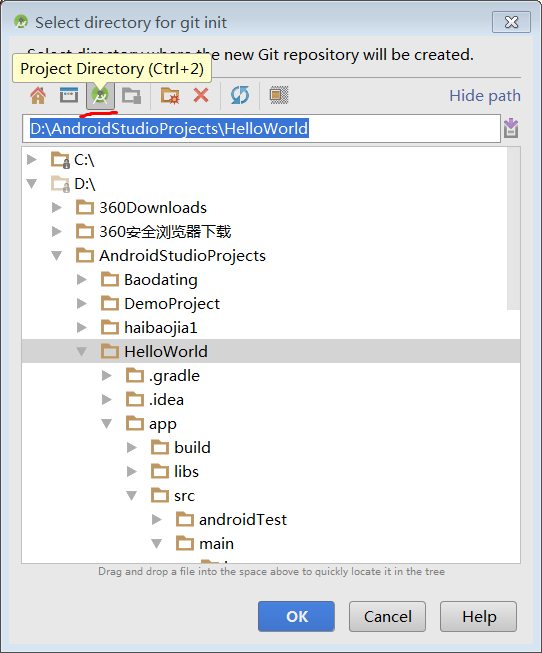
这个命令使用<https://git-for-windows.github.io/>这个网点中装的Git好像不管用，这本书是让安装<http://git-scm.com/> 这个网站中的Git

查看AndroidStudio是否已经集成了Git，我是没有配置过，看来是自动集成的，如下：



点击“Test”按钮以便测试集成的Git是否可用。

创建项目后，第一时间把项目放到Git中进行管理：在AndroidStudio中执行VCS🡪Import into Version Control🡪Create Git Repository，这时出现对话框，一定要选择项目的根目录，可点击对话框中的AndroidStudio小图标，此图标将导航至项目的根目录，如下：



点击OK后，项目中的大多数文件都变成了棕色，这意味着Git在本地识别了这些文件，但是尚未跟踪，而且没有计划添加。

## 7.2 忽略文件

项目根目录和每个Module中都有.gitingore文件

/.idea/workspace.xml文件用于控制AndroidStudio本地配置的设置，这个文件中的内容在每台电脑上都是不一样的

local.properties 文件中的内容对于每一台电脑来说也是不同的。

/build是自动生成的一些东西

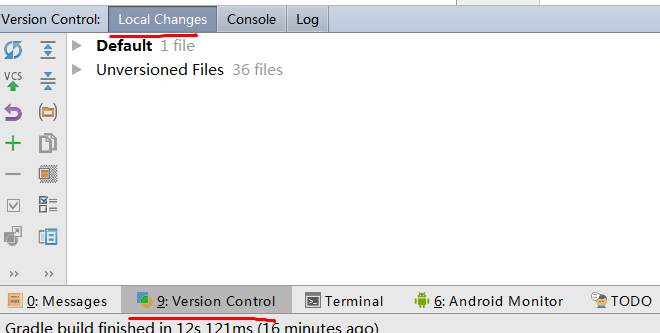
关于版本控制文件的颜色：

* 棕色：表示已经被Git本地识别，但是尚未跟踪，而且没有计划添加。
* 蓝色：表示被Git跟踪而且已经被修改
* 绿色：表示被Git跟踪的全新文件
* 黑色：表示没有改动或者没有被跟踪的文件。

AndroidStudio会持续跟踪已经加入到项目中的文件并提示你尽可能地保持这些文件与Git同步。

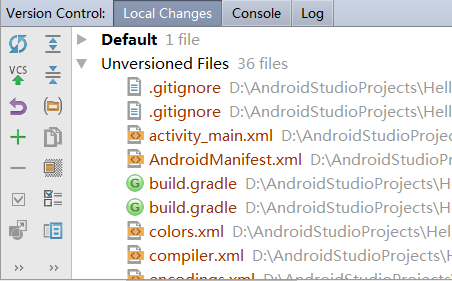
## 7.3 添加文件

打开Version Contrl工具窗口中的Local Changes窗口，如下：

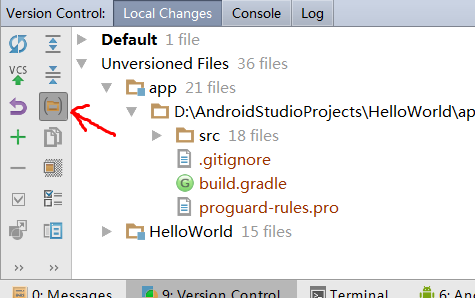


Default表示活动修改列表，当你修改和创建文件时，它们将出现在这个区域中，因为其中保存了已经准备好提交到VCS的文件。

Unversioned Files区域包含了没有被VCS跟踪的所有内容。



如上图，文件是以列表形式展示的，点击如下按钮，将以文件夹的形式展示：



右击Unversioned Files并选择“Add to VCS”或者直接把Unversioned Files拖到Default中，这些文件将被添加到Git索引。添加到Git索引之后即可提交，按下Ctrl +Ｋ。



展开右边的列表

折叠右边的列表

以文件夹的形式对文件进行分组显示

复制

显示忽略文件

 配置忽略文件

显示不同

刷新

提交改变

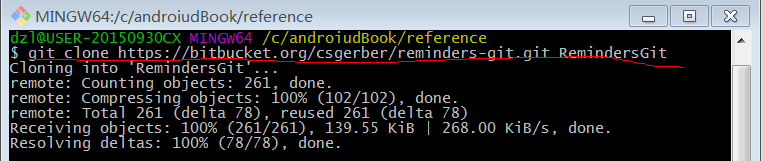
Revert 恢复

## 7.4 克隆参考App：Reminder

导航至：C:\androiudBook\reference\(这个目录可以是其它的，随便创建一个，用于保存克隆的项目)，然后右击选择Git-bash，如下：



在出现的窗口中输入Git命令：git clone https://bitbucket.org/csgerber/reminders-git.git RemindersGit，如下：



这里在我们的电脑中会得到如下内容：

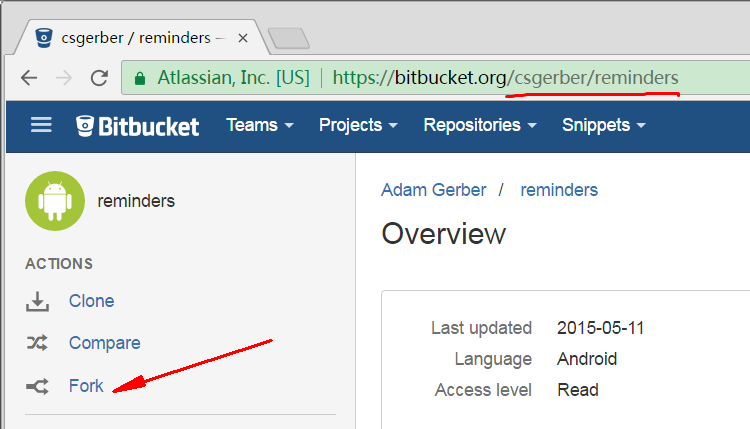


### 7.4.1 分叉和克隆

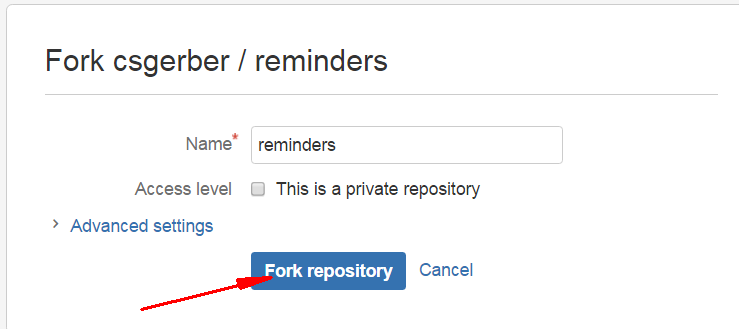
分叉远程仓库是指在某个网络服务中从一个远程账号/分区克隆到另一个远程账号/分区。Fork并不是git命令，它是网络服务（例如Bitbucket或Github）的一种操作。据我们所知，两个较知名的网络服务Bitbucket和GitHub并不允许在它们的服务器之间进行分叉操作。分叉项目是一个将项目从其原来的远程仓库复制到自己的远程Git仓库的过程，目的是要修改它或者做一些衍生工作。

历史上，分叉有一定的贬义，因为它通常是项目成员之间最终目标不一致或者存在分歧的结果。这些差异通常会导致来自多个小组的看似相同软件的多种版本，而用户社区却没有清晰的官方版本可以依赖。然而现在，多亏了Git，分叉得到了极大的鼓励。分叉已经成为协作的一个自然组成部分。许多开源项目将分叉作为提高整体代码层次的一种方法，成员们鼓励彼此做分叉并对代码做出改进。这些改进可以通过pull请求的方式迁入到原始项目中，或者将单个人的bug修改或特性加入到主线中。由于使用Git进行合并和分支非常灵活，因此可以将任何东西(从单独的一次提交到整个分支)迁入自己的仓库。

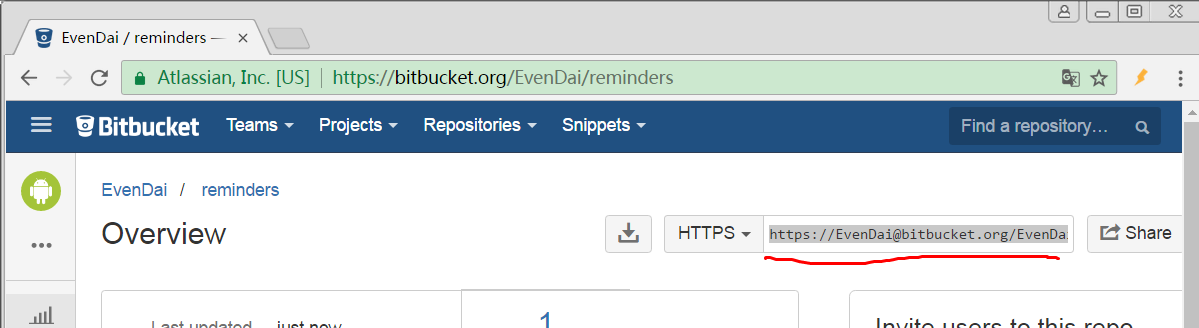
本章并不会涵盖拉取请求和开放源码合作的完整内容，但会涵盖使用此项强大协作功能的方法。登录你的Bitbucket账号并查看Bitbucket上的案例分析。如果你没有Bitbucket账号，那么使用浏览器访问<https://bitbucket.org> 并注册。注册需要花费大约30秒。一旦登录Bitbucket，就可以通过使用Bitbucket网页界面右上角的搜索框找到Reminders仓库。在捜索框中，输入cswrber/reminders。此外，不要对所得结果中的reminders-git仓库感到困惑，你之前已经将其克隆出来作为参考了。要分叉此项目，单击左侧边缘的Fork按钮，如下图所示：



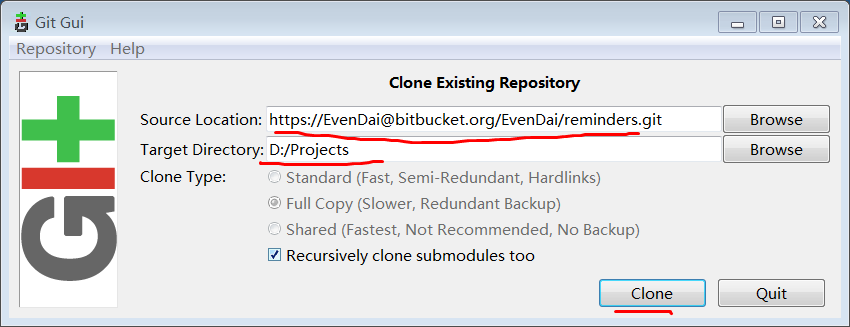
接受默认值并单击Fork repository按钮，如下图所示：



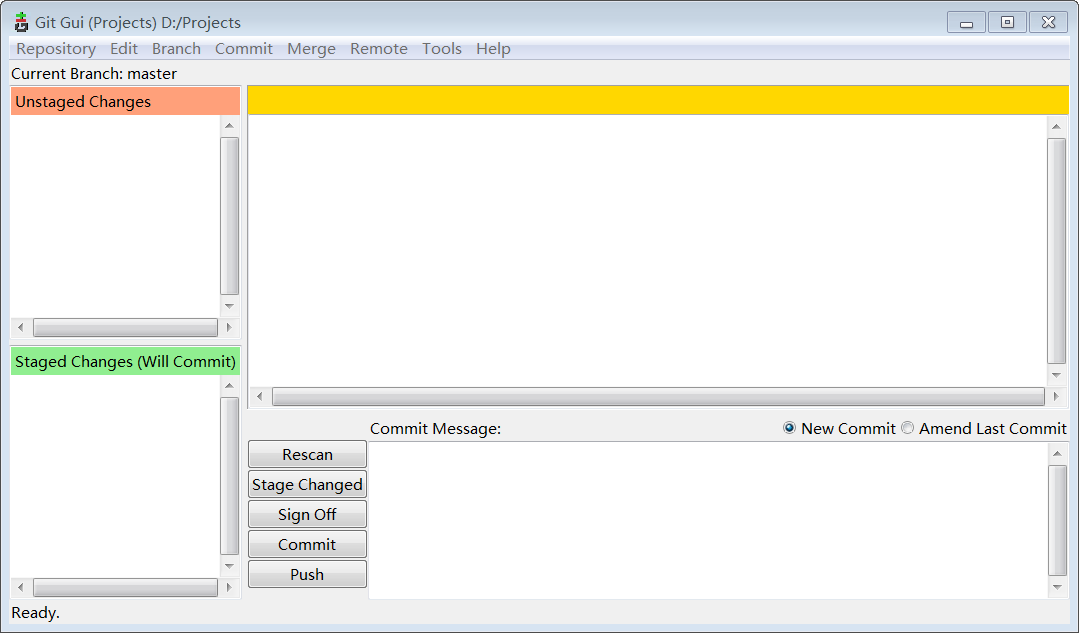
然后复制这个地址：



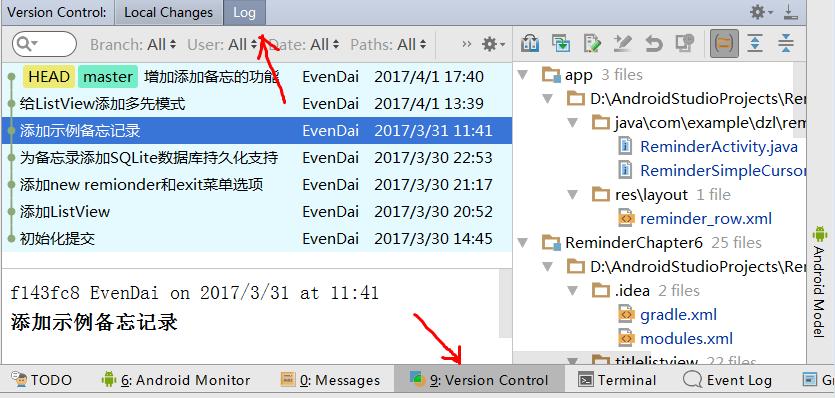
打开 Git Gui，并输入克隆地址，如下：



注：指定的Projects目录不可以存在，它会自动创建。如果已经存在则不能进行克隆。克隆结束后将会出现如下界面，关闭即可：

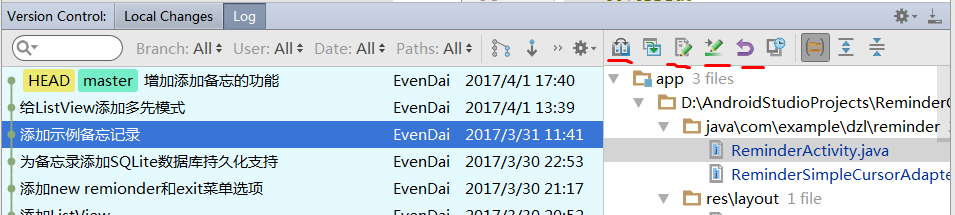


### 7.4.2 使用Git日志

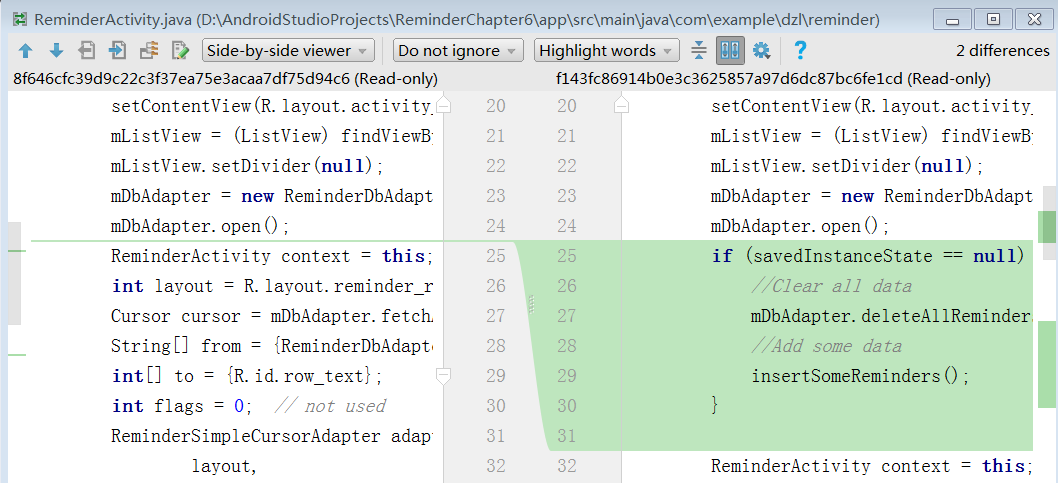


这个视图展示了与仓库中单个分支相关联的时间线。单击时间线中的单个条目会在右侧的修改列表中显示文件这些是提交时修改的文件。也可以使用日志下方的窗口查看提交作者、日期、时间和哈希代码ID，这些哈希代码是唯一ID，当使用Git的某些高级特性时，可以用于标识单次提交。

可以使用提交列表上方的工具栏按钮来编辑源码、打开文件的仓库版本或者回退选中的修改，如下：



单击任意特定提交中的文件并按Ctrl+D (或者仅是双击它们)会得到可视化文本dim它是一种并排的比较工具，突出显示了文件中的修改，如下：



可以使用修改列表上方的工具栏按钮来编辑源代码、打开文件的仓库版本或者回退选中的修改。也可以使用日志下

### 7.4.3 分支

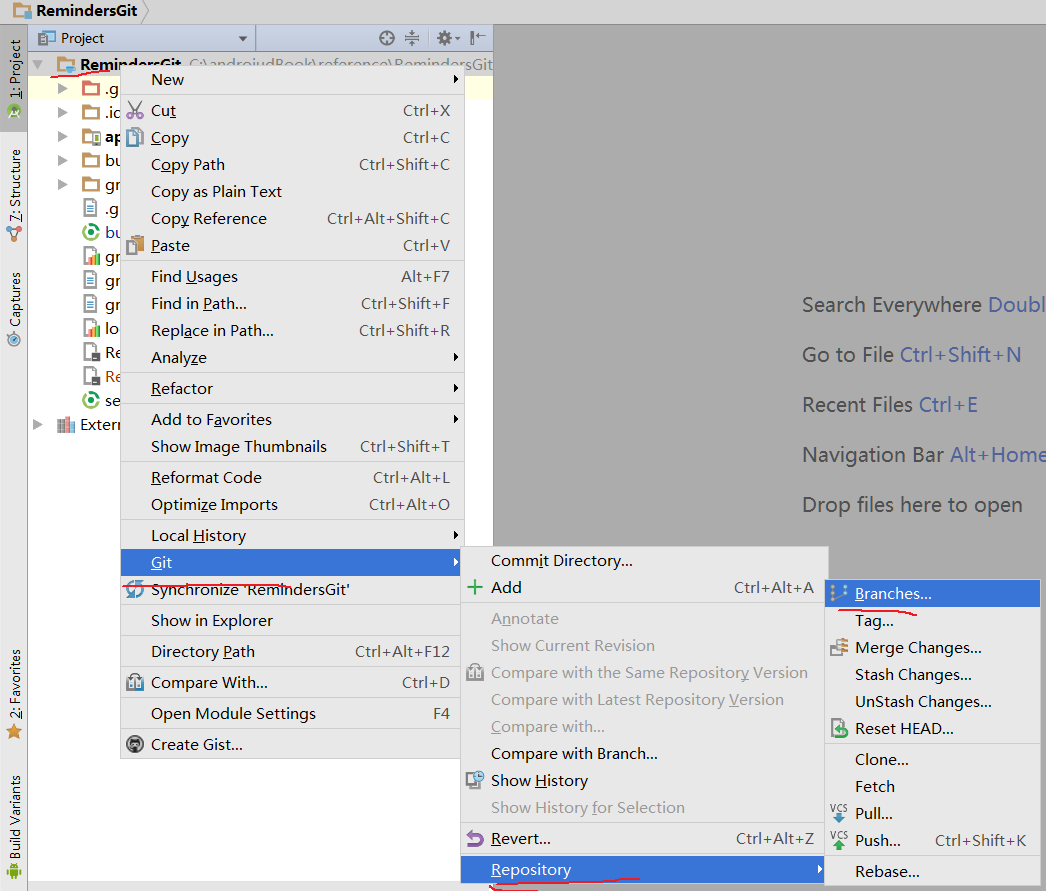
到现在为止，你的所有提交均是在一个称为master的单独分支中进行的，这是默认的分支名。然而，你并不需要一直在master分支上工作。Git允许你创建任意数量的分支，而分支在Git中可以实现多种用途。以下是一种可能的场景。假定你正在和一支开发团队一起工作，并且在开发周期中接到了某些特殊任务。这些任务涉及增加新特性和一些bug修改。完成这些工作的一种合理方法是为每项任务建立一个分支。开发者都会认同这点——当任务完成并通过测试时，开发者需要把任务分支合并到名为dev的分支中，然后删除任务分支。开发周期结束时，由QA团队测试dev分支，他们可能会拒绝修改并将项目退回开发团队，或者结束周期并将dev合并到master。这个过程被称为“Git工作流”，它是使用Git进行团队软件开发的推荐方式。关于“Git工作流”的更多内容请参见如下网址：

<https://guides.github.com/introduction/flow/index.html>

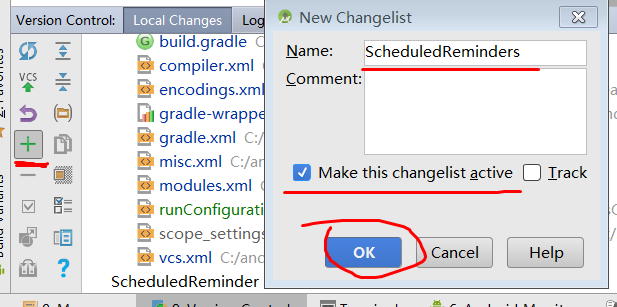
Git工作流对于大型团队来说工作良好，但是如果你单独开发或者仅与一两个开发者合作的话，那么你可能需要其他工作流。不管采用何种工作流，Git中的分支功能都很灵活而且允许你将工作流与Git相互适配。在本节中，我们假定你正在参与一个团队项目并且得到了在Reminders App中添加特性的任务，该任务是允许用户为Reminders指定一天内的某个特定时间。

## 7.5 在分支上开发

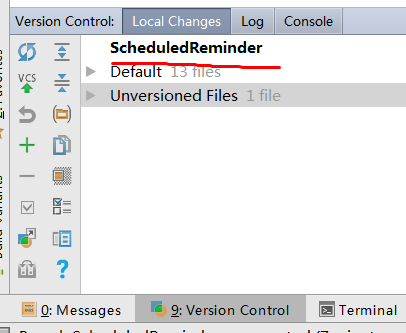
通过File🡪Import Project打开之前克隆的Reminders-Git项目，右击项目视图中的Reminders-Git根文件夹并选择Git🡪Repository🡪Branches，打开Branches提示窗口。这个提示窗口允许你浏览所有的可用分支。单击提示窗口中的New Branche，将分支命名为ScheduledReminders，如下图：



接来出现输入分支名的对话框，取名为：ScheduledReminder，确定，新的分支将被创建并迁出以供你使用。打开Version Control工具窗口中的Changes窗口，并点击绿色加号以创建新的修改列表，和新的分支一样，将其命名为ScheduledReminder，因为下一轮修改将会引入设定备忘时间的特性。确保选择了Make this changelist active复选框，如下：



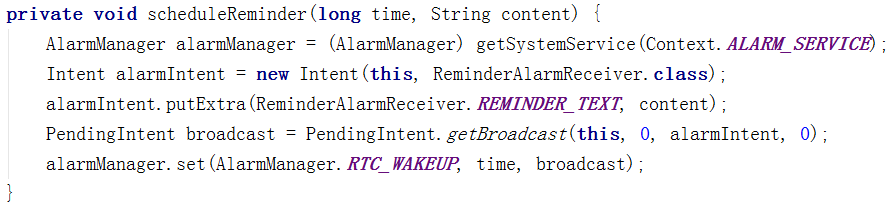
设置以后变化如下：



选择时间对话框的创建：



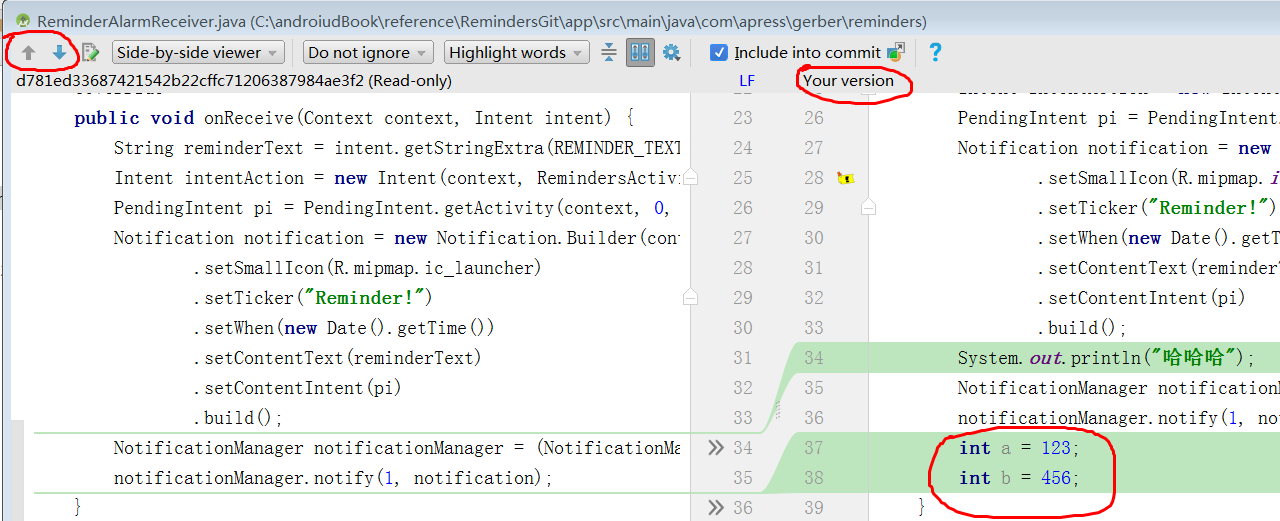
使用AlarmManager在指定的时候到的时候发送一个广播：



上面的广播指定的广播接收者为ReminderAlarmReceiver，如下：



这个接收者收到广播后又发出一个通知，用户点击通知后又打开了应用的界面。妙妙妙呀！

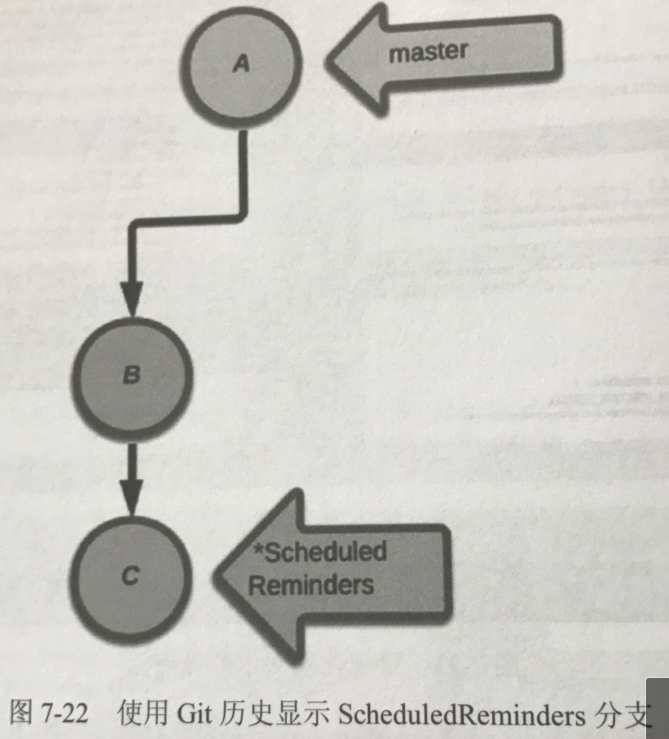


在提交的时候，点击如上图的按钮可以打开一个比较的窗口，左边是Git版本的代码，右边是你本地的代码，可以直接在右边窗口修改代码，而且也有代码补全等特性，在窗口的左上角有上下箭头，单击可以上下查找两个版本之间的差异，而且我们看到上面的变量a、b名字是灰色的，这代码这些变量没有被使用，在此视图下，你不仅可以回忆起已经完成的修改，而且还会发现那些被遗漏的修改，这再一次为你提供了在提交之前修改问题的机会。

### 7.5.1 Git提交和分支

如果有着使用类似Perforce或Subversion等传统VCS工具的背景，那么你会觉得Git风格的分支提交虽然很类似，但又与之前所熟悉的工具有一定不同。你将会希望了解Git在提交和分支管理方式上的细微差别。这些差别会让新手感到困惑，但它们是让Git具备强大功能和灵活特性的核心。

Git中的提交被视作项目历史中的头等实体，可以通过一个特殊的提交哈希代码来标已识它。虽然你不需要了解Git实现单次提交和版本控制的细节，但很重要的一点是将提交视作存在于历史时间轴中代表仓库整体状态的对象或实体。提交是发生在Git历史中某个时间点上具有原子性的工作单元，带有一段用于描述工作的提交消息。每次提交均有之前的一次或多次提交作为其父级提交。你可以将分支视作指向历史中某一次提交的标签。当创建分支时，历史中的该点处会创建一个标签而当你向该分支提交时，该标签会跟踪提交历史。以下框图(从图7-22开始)演示了Git中记录的Reminders项目历史。

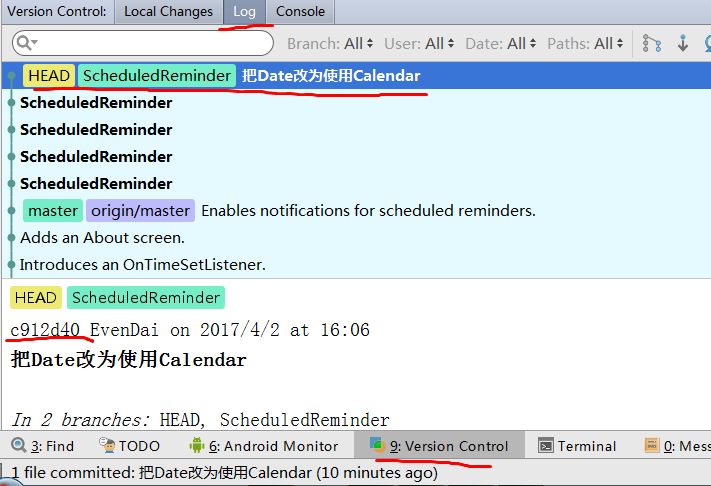
注意：Android Studio从下到上展示提交日志，而我们的框图从上到下进行展示。

主分支由灰色箭头表示，它指向己克隆项目的最后一次提交A(与Git日志视图做比较，你将会发现A之前还有其他提交，但为了简洁，这里将它们略去)。ScheduledReminders分支是绿色箭头，指向系列提交B和C(实现了新特性)中最新的一次。为了简单，我们使用单个字母作为标签，但Git使用提交哈希代码，它有着更长的十六进制名称，例如c04ee425eb5068e95cle5758mb36c6bb96f5938

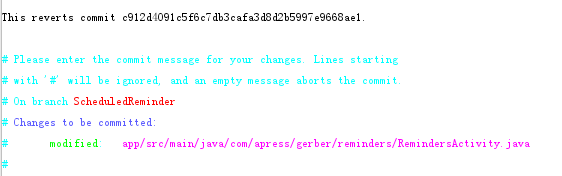
，你可以使用哈希代码的前几个字符来弓用某次特定提交，只要它们是唯一的或者与其他哈希代码的前几个字母不同即可。

### 7.5.2 回退在哪里？

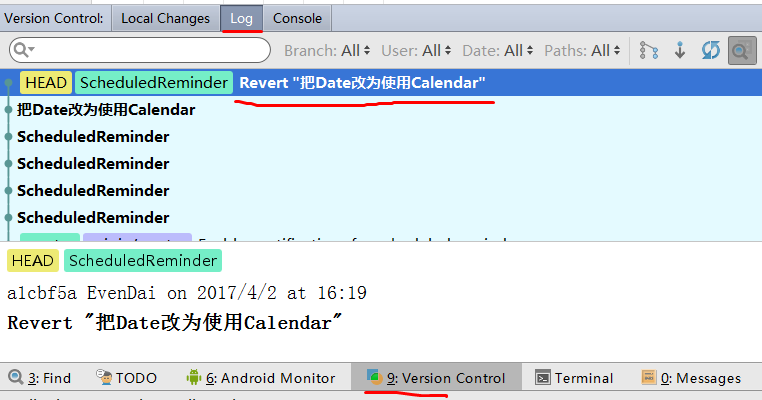
提交代码，提交信息为：“把Date改为使用Calendar”，如果想要撤销这个提交，也就是还想使用Date类，不使用Calendar，找到提交的哈希代码，如下：



如上图的c912d40就是提交的哈希值，打开AndroidStudio的命令行终端，输入命名：git revert c912d40，这会打开如下会话界面：

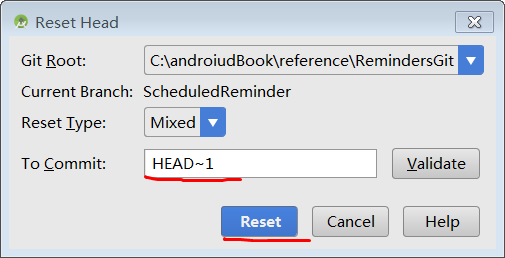


输入“:q“，然后回车退出编辑会话，此操作会保存默认提交消息并执行提交。这时看看提交记录，这个撤销操作也当作了Git提交的一条记录，如下：

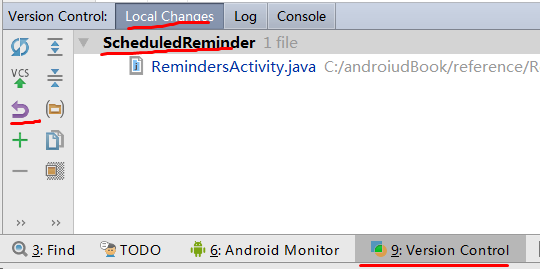


查看代码，发现Calendar又变回了之前的Date。这个Revert操作产生的提交信息为：“Revert + 最后一次提交的消息”

撤销提交的另一种方法是使用reset命令，它类似于revert，但是有着微妙的差别，revert会把撤销当成一次提交，并且代码即时恢复。而reset不会有提交，也不会恢复代码。再次把Date改成Calendar，然后提交，然后执行VCS🡪Git🡪Reset Head，在弹出的对话框中输入HEAD~1,点击“Reset”按钮，如下：



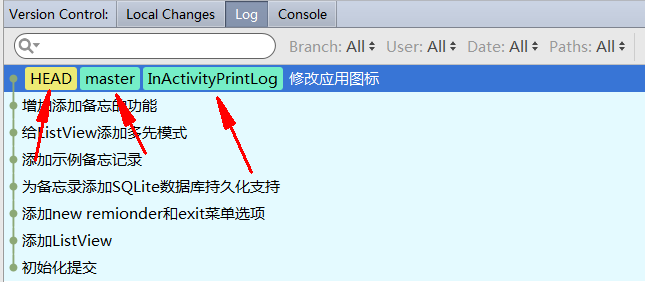
执行完该操作之后，我们再看提交记录，之前的提交记录消失了，但是代码并没有变，如果你希望代码也恢复回来，则可以点击下面的Revert按钮：



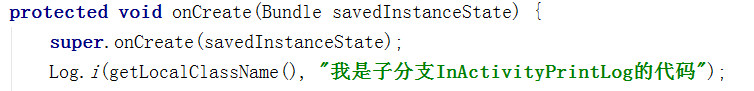
### 7.5.3 合并

合并是一种组合来自两个独立分支工作的方法。

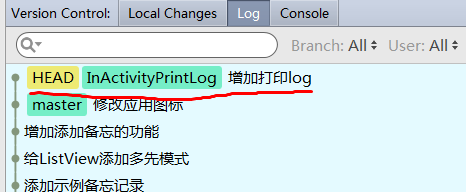
VCS🡪Git🡪Branches🡪New Branch，输入名称为InActivityPrintLog为新的分支名称，这时查看提交记录：



HEAD代表这是最新的提交记录，后面是分支名称，“master InActivityPrintLog”，这说明InActivityPrintLog和master分支都指向了同一条提交记录，这条记录是这两条分支的最新提交记录。接下来我们在ReminderActivity的onCreate方法中输入一句Log代码，如下：



提交代码，查看提交记录如下：

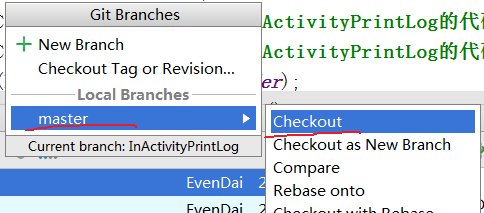


再增加一行打印log，再提交，提交记录如下：

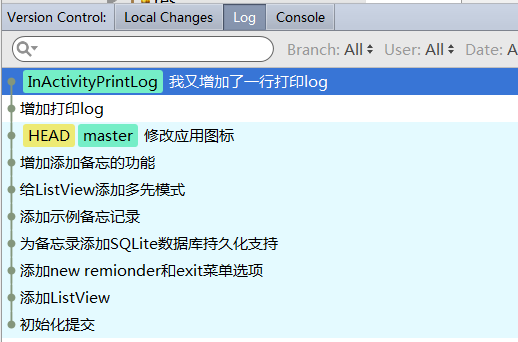


从上面的截图可知，HEAD代表当前分支的最新提交记录，第一个红框的提交记录都属性InActivityPrintLog分支，第二个红色的提交记录都属于master分支，在master分支的提交记录“修改应用图标”分出了一个子分支“InActivityPrintLog”。

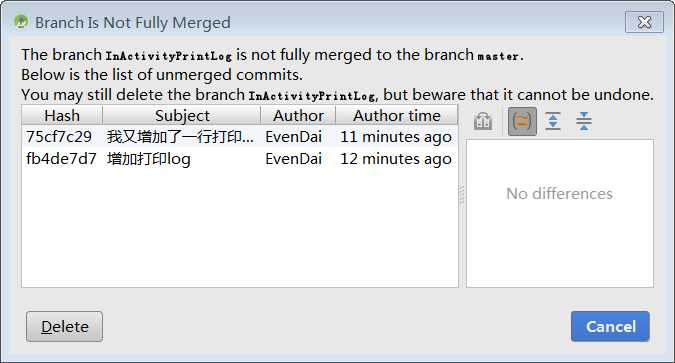
这时我们想切换回主分支继续工作，执行VCS🡪Git🡪Branches🡪master🡪Checkout，如下：



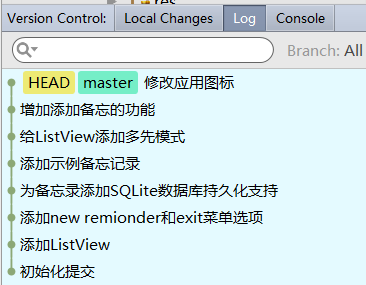
这时注意看代码，已经恢复到主分支的代码了，ReminderActivity中的打印语句已经没有了，因为当前是工作在master分支了，但是查看一下提交记录：



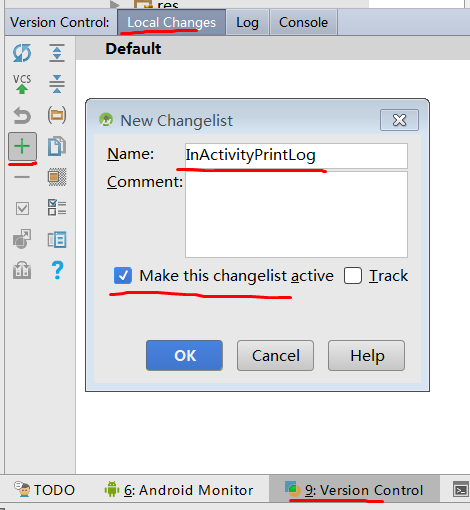
这是我们不希望看到的结果，这里有两个提交是在InActivityPrintLog中提交，怎么在master分支也显示了呢？现在把InActivityPrintLog分支删除，执行VCS->Git🡪Branches🡪InActivityPrintLog🡪Delete，这时弹出对话框如下：



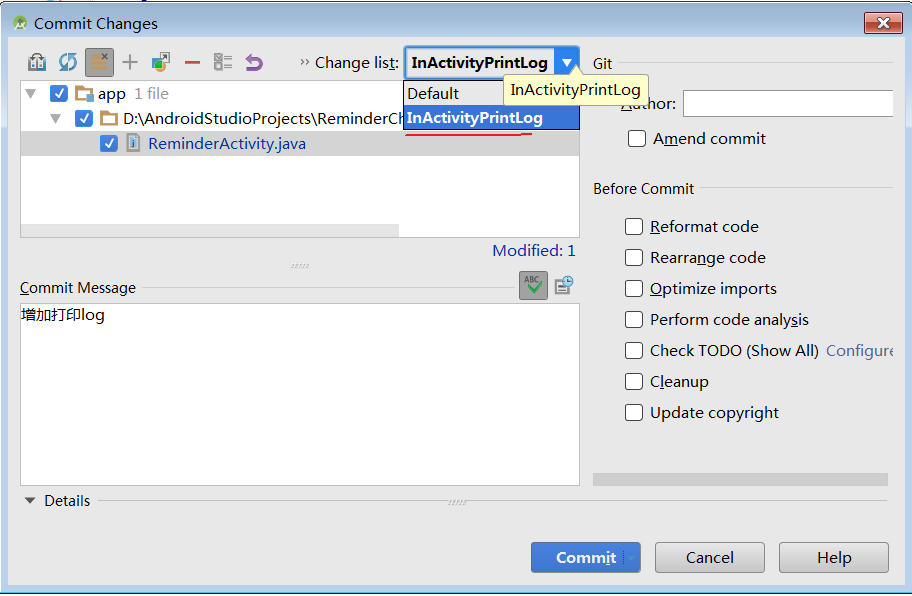
意思是说InActivityPrintLog分支没有完全整合到master分支中，而且显示了该分支有哪些提交，如果决定要删除就点击“Delete”按钮。这时再查看提交记录，InActivityPrintLog分支的记录没有了：



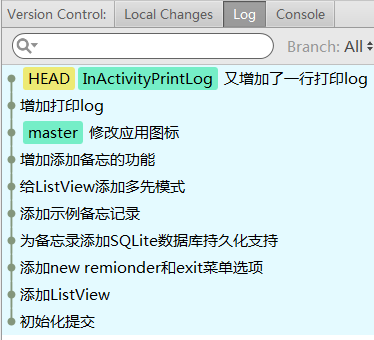
接下来我们重新创建InActivityPrintLog分支，然后执行下面的操作：



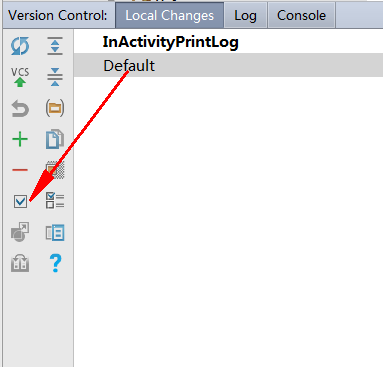
然后再像之前一样添加打印log的代码然后提交，如下：



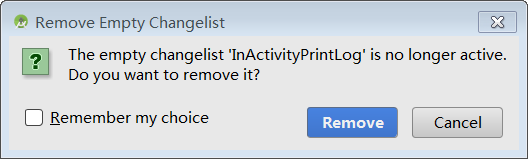
在这里我们可以看到提交的记录是保存在Change List中的InActivityPringLog这个列表中，默认选择的就是这个列表，因为我们在创建这个列表的时候选择了“Make this change list active”。再增加一行打印log，再提交，这时的提交记录跟以前还是一样的，如下：



这时切换到master分支，选择Default，然后点击如下按钮，让这个Default（它一个Change List的名称）成为活动的：

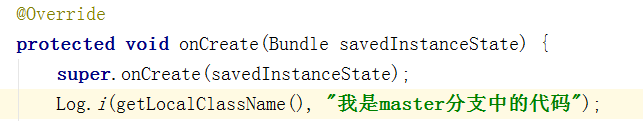


这时会出现如下对话框：

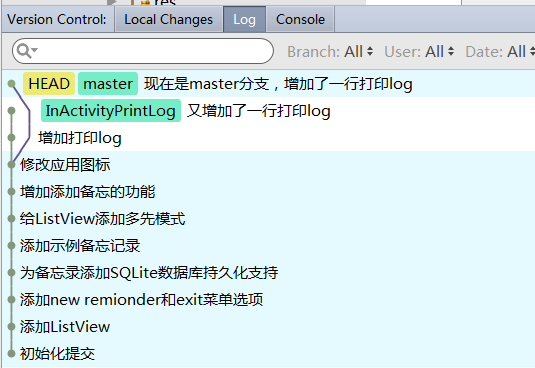


大概意思是说“空的changelist InActivityPrintLog不再是活动的，你想要移除它吗？” 这个改变列表，其实就是当我们有可以提交的文件时，这些文件会显示到这个列表中。

我们ReminderActivity中增加如下打印语句：“

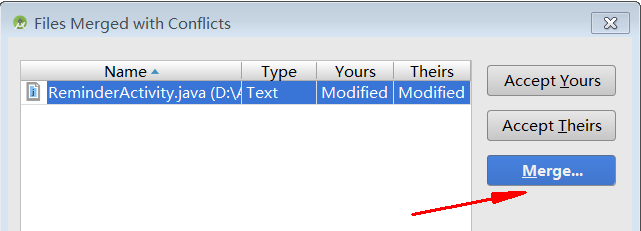


提交。这时查看提交记录：



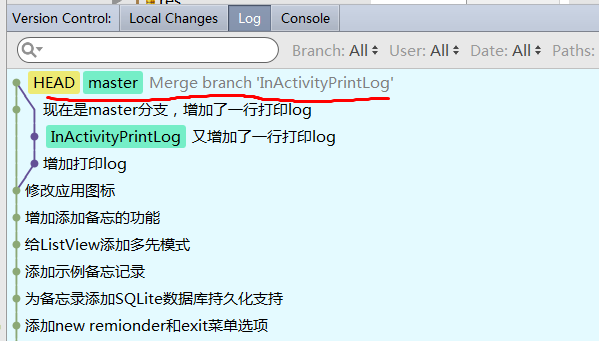
我们发现master分支的修改记录是从”修改应用图标”然后到“现在是master分支，增加了一行打印log”这条记录的。

现在我们希望master和InActivityPrintLog分支进行合并，执行VCS🡪Git🡪Branches🡪InActivityPrintLog🡪Merge，这样，InActivityPrintLog分支所有的修改都会融入当前分支（master分支）。这里会出现冲突，弹出如下对话框：



我们点击Merge解决冲突即可。

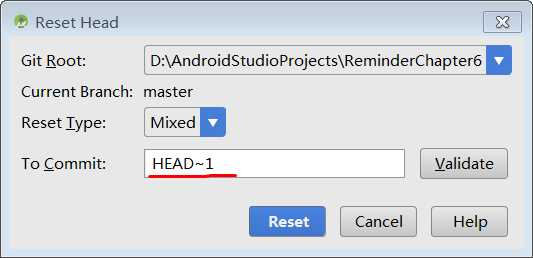
这时查看提交记录如下：



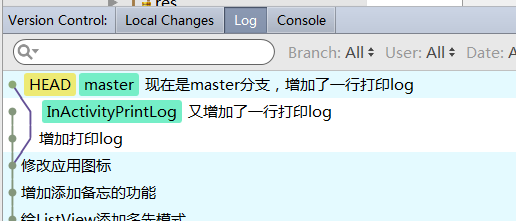
### 7.5.4 Git 重置修改历史

这节完全看不懂，不知道是什么意思

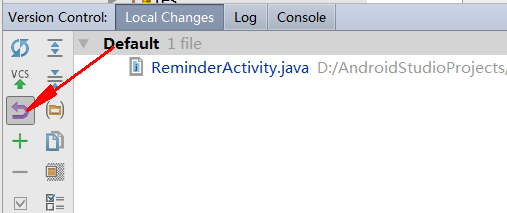
如果我们在master分支中增加的log代码想作为一个分支怎么办？现在需要强制主分支向后退，执行VCS🡪Git🡪Reset Head



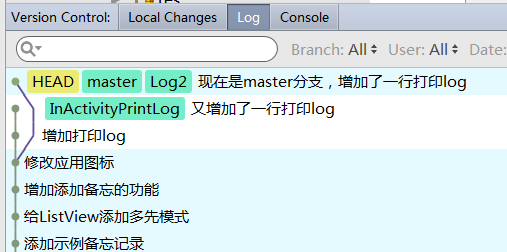
查看提交记录如下：



合并的记录没有了，但是代码并没有恢复，操作如下：



选择Default按钮，并点击Revert按钮恢复代码。这是master分支中还是有打印log的代码，只是没有了InActivityPrintLog分支中合并的代码了。我们希望打印log的语句变到分支中去，执行VCS🡪Git—Branches🡪New Branch🡪输入名称为Log2，这时查看提交记录，master和log2分支都指向了同一条提交记录，如下：



### 7.5.5 Git变基

这节完全看不懂，不知道是什么意思

### 7.5.6 分离头部

这节完全看不懂，不知道是什么意思

### 7.5.7 相对引用

这节完全看不懂，不知道是什么意思

### 7.5.8 在变基时解决冲突

这节完全看不懂，不知道是什么意思

### 7.5.9 Git远端

这节完全看不懂，不知道是什么意思

# 第8章 设计布局

## 8.3 布局设计指导原则

需要注意屏幕分辨率和像素密度，屏幕分辨率是屏幕在水平方向和垂直方向可以容纳的总像素数，以二维数字的形式给出。通常使用标准VGA单位来描述分辨率，VGA表示视频图形阵列，台式机和笔记本电脑的标准值为640 x 480，目前，你会发现一些移动设备上的变化形式，例如半VGA（HVGA）：320x480，四分之一VGA（QVGA）：320x240，宽屏VGA（WVGA）：800x480，扩展图形阵列（XGA），宽屏XGA（WXGA）等等，还会有其他一些可能的分辨率。

像素密度表示单位长度可以容纳的总像素数。这个尺寸与屏幕大小无关，尽管它会受屏幕大小的影响。例如，设想将分辨率为1024×768像素的20英寸显示器与分辨率为1024×768像素的5英寸显示器做比较。两者有着相同的像素数，但是后者将这些像素压缩到更小的区域内，而这增加了它们的密度。像素密度的单位是点每英寸(dpi)，表示每英教寸区域内可以容纳的点(或者说像素)的数量。在Android屏幕上，密度通常使用一种称为密度无关像素(dp)的单位来度量。它是一种基线度量，1dp等于160dpi屏幕上的1像素。使用dp作为测量单位使得布局能够在不同密度的设备上正确地缩放。

* ldpi 低密度屏幕 120dpi (即每英寸有120个像素)
* mdpi 中密度屏幕 160dpi
* hdpi 高密度屏幕 320dpi
* xhdpi 超高分辨率屏幕 480dpi
* xxhdpi 640dpi

在创建图片时，要想适配这些密度，则需要创建多个不同尺寸的图片，在创建图片时先创建最大分辨率的图片，然后再缩小出各种小的图片，按照dpi大小来缩小即可，如下：

为xxhdpi切了一张200 x 200的图片，则需要按比例再切出如下大小的图片：

Xhdpi 需要150 x 150的图片，因为xhdpi是480dpi，而xxhdpi是640dpi，xhdpi是xxhdpi的0.75比例，要想不变形，xhdpi的图片宽高也必须是xxhdpi图片宽高的0.75，也就是150（200 x 0.75）

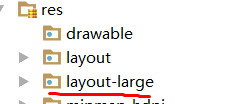
|  |  |  |
| --- | --- | --- |
| 文件夹 | 比例 | 图片大小 |
| drawable-xxhdpi（640dpi） | 640 / 640 = 1 | 200 x200（200 x 1） |
| drawable-xhdpi （480dpi） | 480 / 640 = 0.75 | 150 x150（200 x 0.75） |
| drawable-hdpi （320dpi） | 320 / 640 = 0.5 | 100 x 100（200 x 0.5） |
| drawable-mhdpi（160dpi） | 160 / 640 = 0.25 | 50 x 50（200 x 0.25） |

## 8.4 Fragment

Fragment是可重用的XML自片段，它类似于include：

* 相同点：
  + 都是把一个布局嵌入到Activity布局中的某个位置
  + 都可以在不同的布局中复用，甚至在同一个布局中复用也是可以的
* 不同点：
  + 在布局中嵌入include，这个include的布局控件的控制逻辑写在Activity中
  + 在布局中嵌入fragment，这个fragment的布局控件的控制逻辑写在Fragment中

Fragment的经典使用场景：ListView的Item点击详情，在手机上，ListView是一个单独的界面，点击item跳到另一个界面展示详情，如果使用Fragment，则这两个单独的界面可以复用，在平板上一个界面同时显示，左边显示ListView，右边显示详情。具体实现例子可查看笔记同目录中的“FragmentDemo.rar”。



Layout-large：大屏幕设备（例如7英寸或更大尺寸的平板电脑）将会选用这个文件夹下的布局

这个修改符的创建方式：在res目录右击🡪New🡪Layout Resource File🡪在出现的对话框左下角添加Size属性为large，则创建的布局自动会在layout-large目录中，如果不存在就会自动创建该目录。