新员工入职常用系列—2.开发工具安装

编写	张思宇	日期	2017-06-14
审核		日期	
批准		日期	

一配网络科技（北京）有限公司中心

修改记录

版本	日期	修改纪要	修改人
V0.1	2016-02-27	从原来新员工入职常用软件安装中拆分出来	张思宇
V0.2	2016-03-14	更新采用Git版本2.0.0，Git for Windows 2.7.2版本。	张思宇
V0.3	2016-03-19	更新采用Web Storm 2016.1版本，并增加3.8 Atom描述，以后再完善。	张思宇
V0.4	2017-06-14	添加lint的详细说明	张思宇
V0.5	2019-03-19	修订Git用法	张思宇

目　录

1. 概述

1.1 简介

新员工入门指南系列文档主要面向新入职的员工，使大家能够快速部署好自己的办公软件环境。

欢迎新员工加入一配网络科技公司研发中心，新员工入职常用系列目前分为四个分册：

- 第1分册是描述常用办公软件安装，适用于公司所有人员，包括行政、财务、开发等团队的人员；
- 第2分册是描述常用开发工具安装，适用于公司开发团队的人员；
- 第3分册是常用开发资源，主要描述公司研发中心开发参考资源：参考网站、视频学习资源、竞争对手信息等。
- 第4分册是入职须知，主要描述入职后须知的一些注意事项，避免出现各种问题。

这些文档的目的是使新员工能快速地获取信息，适应新环境，融入研发中心大家庭，在工作上尽快上手。本文档是该系列的第2分册。

1.2 适用范围

本文档适用于一配网络公司研发中心的新入职员工或由其他部门调配到研发中心的员工，调配员工除无需办理入职手续外，其他事项同新入职员工。长期实习生和外包合同员工除入职手续及个别方面不同外，其他同样适用说明本文档的适用范围。

1.3 缩写和术语

无。

1.4 参考资料

1pei-IS 2014-0009.1-新员工入职常用系列-1.办公软件安装.docx

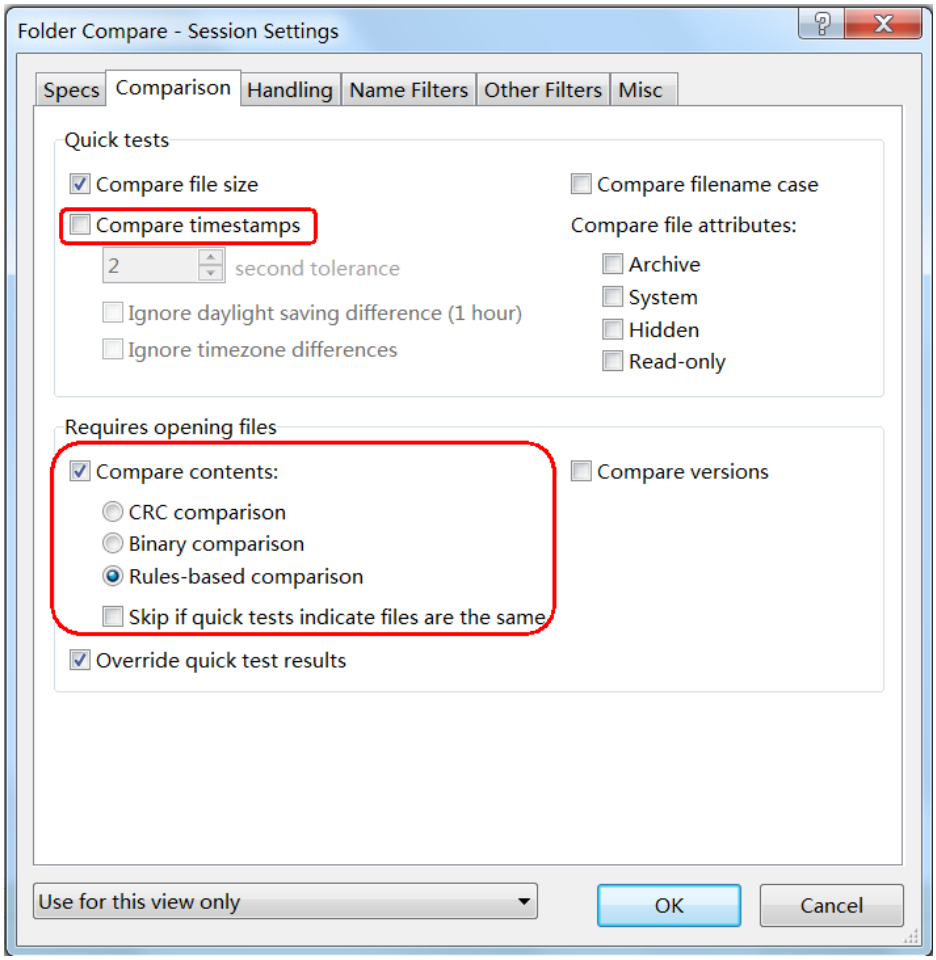
1pei-IS 2014-0009.3-新员工入职常用系列-3.开发资源.docx

2. 文件比较工具Beyond Comapre

Beyond Comapre工具是最好用的文件diff比较工具，可方便地比较出一个目录下每个代码文件的详细差异。

推荐安装BeyondCompareV3.1.6，可参见ftp://tools:tools@192.168.1.100//T03-开发工具/Beyond Compare/BeyondCompareV3.1.6.rar。

注意在比较的时候，将时戳比较（Compare timestamps）去掉，采用基于规则的比较方式（Rules-based comparision）来比较文件的差异，见下图：

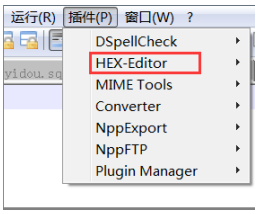


3. 文本编辑器软件

3.1 Notepad++

建议安装文本编辑器软件Notepad++, 可参见http://notepad-plus-plus.org。

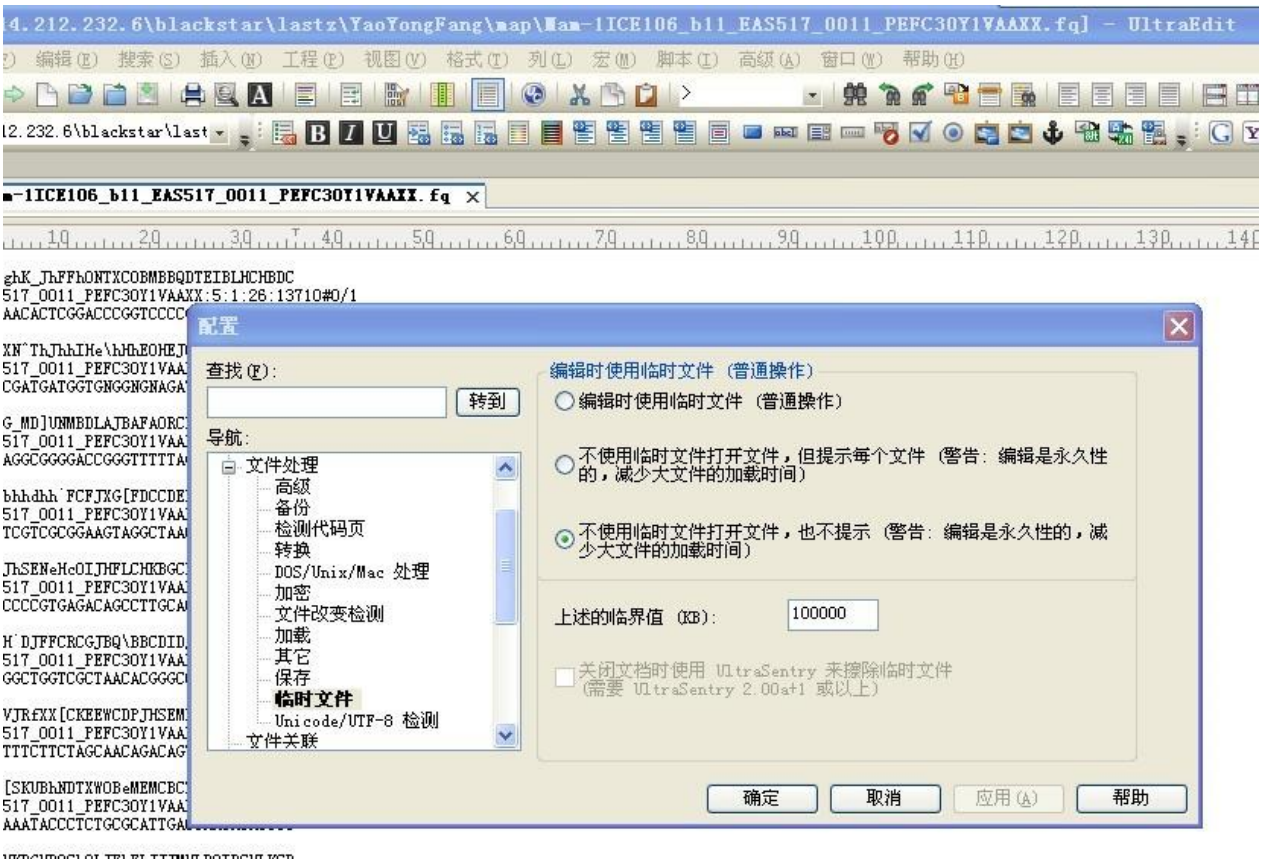
推荐：安装过程中或者安装完成后，可通过Plugin Manager来安装DSpellCheck、HEX-Editor等常用插件。



3.2 ultraEdit v24.20 (released 2017-08-31)

Notepad++只有32位，因此只能打开几百兆的文件，面对一些超大文本文件例如2G、4G以上文件，打开就力不从心了，尤其是需要修改一些几百兆的sql文件。

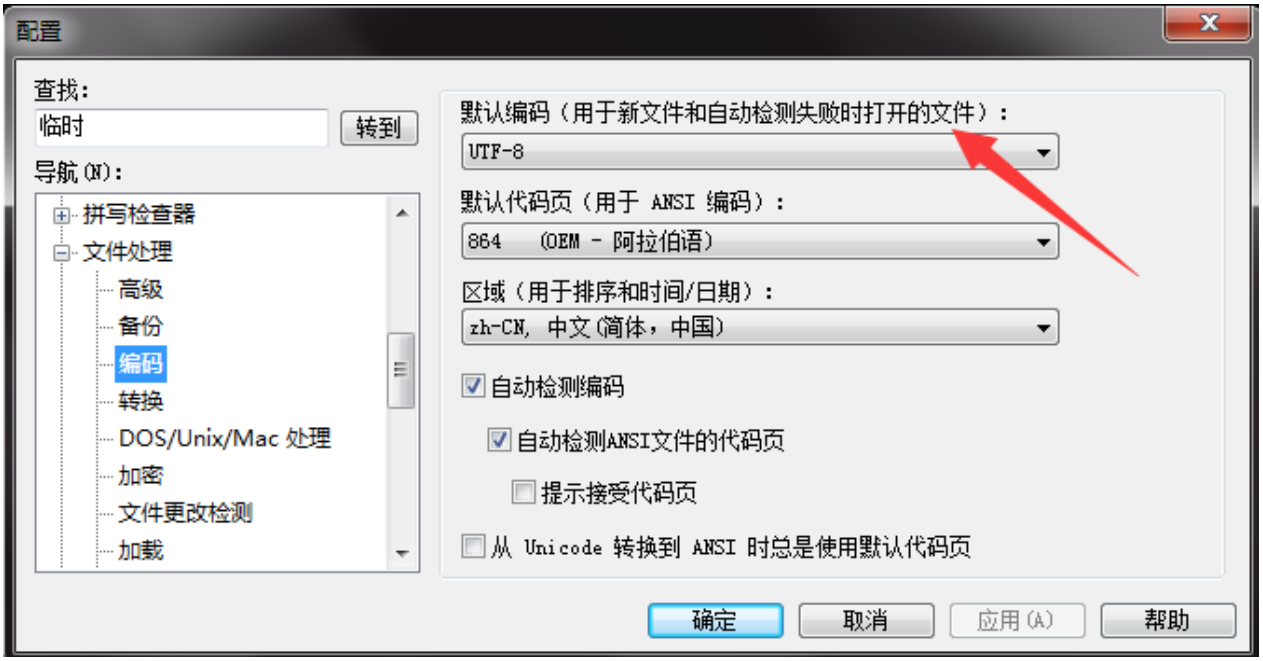
此时可以使用ultraEdit工具，只需要进行简单的配置：高级——配置——临时文件——



其他常用配置：

Ctrl+H在16进制与文本之间切换

在高级——配置——文件处理——编码中设置默认编码为UTF-8。



3.3 Netbeans IDE8（不推荐安装）

3.3.1 NetBeans IDE简介

NetBeans 由Sun公司（2009年被甲骨文收购）在2000年创立，它是开放源运动以及开发人员和客户社区的家园，旨在构建世界级的Java IDE。NetBeans当前可以在Solaris、Windows、Linux和Macintosh OS X平台上进行开发，并在SPL(Sun公用许可)范围内使用。

NetBeans包括开源的开发环境和应用平台，NetBeans IDE可以使开发人员利用Java平台能够快速创建Web、企业、桌面以及移动的应用程序。

NetBeans IDE支持开发语言：Java、C/C++、PHP、JavaScript、HTML/CSS、Ruby、Groovy、Grails等开发语言，

支持版本管理工具：支持与Git、SVN、ClearCase、CSV等版本工具的集成，

支持语言：支持中文、英文等多种语言。

支持JavaScript框架：支持jQuery、AngulaJS等。

支持单元测试：phpUnit等。

NetBeans项目由一个活跃的开发社区提供支持， NetBean开发环境提供了丰富的产品文档和培训资源以及大量的第三方插件。

对于我们后端使用PHP环境，前端使用HTML, Javascript/CSS来说，NetBeans IDE是一个非常理想的开发环境。

3.3.2 下载与安装

可以通过两种方式来下载。

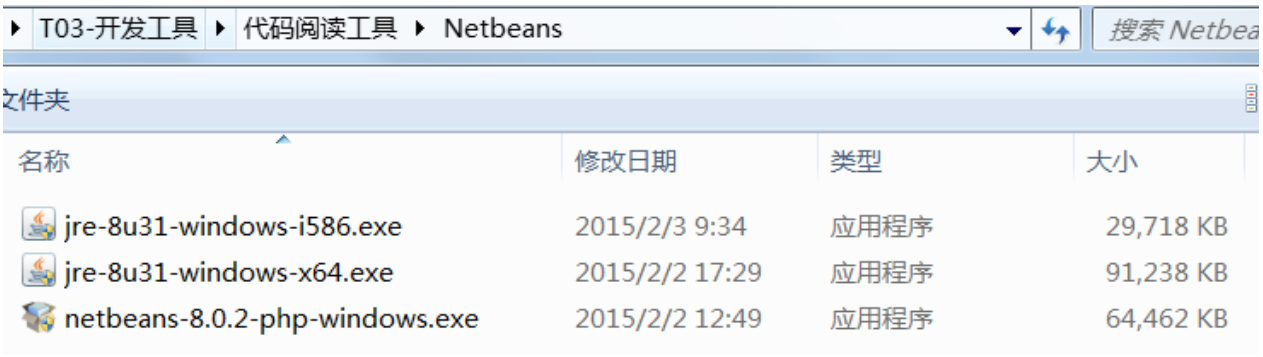
步骤1：下载JRE8并安装

http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html下载JRE8版本win64位。

步骤2：下载Netbeans IDE8并安装

https://netbeans.org/downloads/ 下载Netbeans IDE8.

也可以访问ftp://tools@tools:192.168.1.100/ 在“代码阅读工具\Netbeans”下查看到下面文件：



其中jre-8u31-windows-i586.exe是当操作系统为win7 32位时安装。

jre-8u31-windows-x64.exe是当操作系统为win7 64位时安装。

首先安装jre-8u31-windows-XXX.exe文件，然后安装netbeans-8.0.2-php-windows.exe。

3.3.3 安装插件

以下均参考自：https://netbeans.org/kb/trails/php.html

步骤1：安装NetBeans Connector Chrome Extension

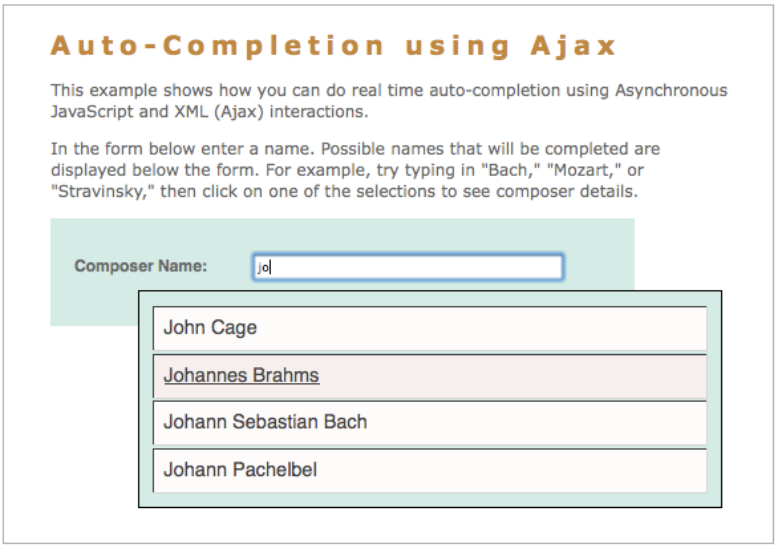
参考https://netbeans.org/kb/docs/webclient/html5-gettingstarted.html 来安装NetBeans Connector Chrome Extension（注意由于国内防火墙原因，需要在Chrome浏览器中有红杏等翻墙账号才可以访问Chrome Store），这样就可以在NetBeans IDE中运行HTML或PHP程序时选择“Chrome”，就可以打开Chrome浏览器来查看运行效果。

也可以在NetBeans IDE中运行HTML或PHP程序时选择“嵌入式WebKit浏览器”来查看运行效果，见下图。

步骤2：安装PHP调试工具xdebug, 可参见http://www.xdebug.org/download.php， 留待后续研究。

下载与安装各种插件：http://plugins.netbeans.org/PluginPortal/

https://netbeans.org/kb/docs/php/ajax-quickstart.html 中提供了通过Ajax在一个搜索框中输入任意字符可自动完成的示例：

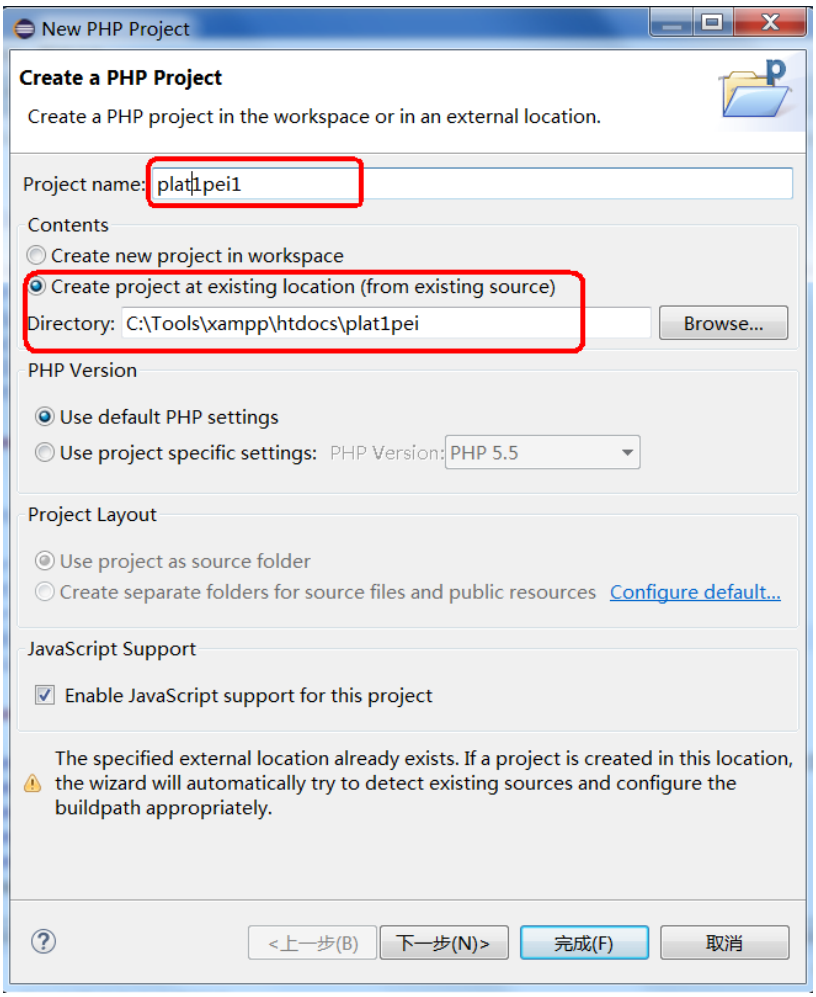


3.4 Eclipse for PHP

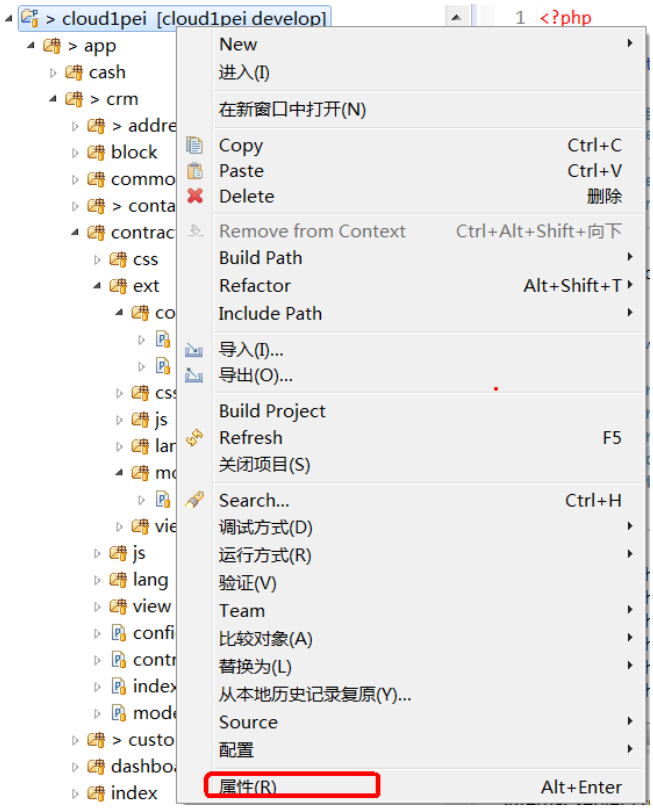
由于Netbeans需要占用大量的内存，因此推荐可使用EPP Studio来开发PHP代码。

安装完成后，点击“文件”→“新建”→“PHP Project”来新建一个php工程，

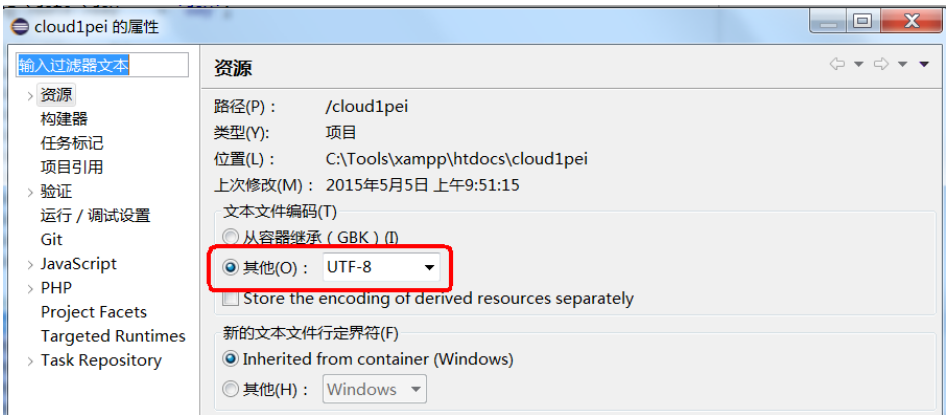
然后在Project Name中输入工程名称，从已有位置选择代码来新建该工程，见下图：



建好工程后，鼠标右键点击选择“属性”，见下图：



然后选择编码格式为UTF-8，见下图，否则代码中的中文注释等将出现乱码：



快捷键：

CTRL+h， 整个工程搜索

3.5 Sublime text2

存在Sublime text2与text3两个版本，推荐使用 Sublime text 3，可参见下一章节3.6。

官网可参见http://www.sublimetext.com/。

Sublime Text3技巧使用介绍 可参考 http://jingyan.baidu.com/article/14bd256e2da001bb6d26129f.html

注意：本章节的内容只是为了记录，可跳过本章节内容。

3.5.1 插件Package Control

使用sublime几乎都会首先安装这个插件Package Control，这个插件是管理插件的功能，先安装它，再安装其他插件就方便了。

安装方法：

点击sublime的菜单栏 view->show console ；现在打开了控制台， 这个控制台有上下两栏， 上面一栏会实时显示sublime执行了什么插件，输出执行结果， 如果你安装的某个插件不能正常运行，应该先在这里看看有没有报错。下面栏是一个输入框，可以运行python代码。我们输入下面的代码点击回车运行， 就能安装好package control了。

```
import urllib2,os;pf='Package Control.sublime-package';ipp=sublime.installed_packages_path();os.makedirs(ipp) if not os.path.exists(ipp) else None;open(os.path.join(ipp,pf),'wb').write(urllib2.urlopen('http://sublime.wbond.net/'+pf.replace(' ','%20')).read())
```

然后我们按住 ctrl+shift+p。此时会输出一个输入框， 输入install。 选择package contrl： install package 回车 ，需要稍定一会儿，右下角状态栏会显示正在连接的提示文字。 使用sublime时注意看右下角状态栏，很多插件的提示信息都显示在这里，这个状态栏很小，初次使用的人都有可能没有注意到它。

稍等一会儿后，它会出现一个插件列表， 你也可以在输入框中输入文字进行搜索插件。 搜索到自己想安装的插件，再选择它，回车。 就自动给你安装好了。

如果要卸载插件， ctrl+shift+p 输入 remove， 选择package control:remove package 然后再选择已安装的插件， 回车即可卸载。

如果package control 安装插件时失败了， 我们可以采用手动安装的方式， 在google上去搜索插件，下载插件的源代码。在sublime的菜单栏点击 preferences->Browse package.. 此时会打开插件目录。然后把你下载的插件源代码复制进去就可以了。

ctrl+shift+p 打开的输入框面板是什么？ 英文叫做 “Anything panel” ，任何操作都可以在这个面板里面完成。我暂且翻译为“万能面板”。 打开万能面板有几种方式。

ctrl+shift+p 打开时，我们需要在面板中输入一个命令，然后执行命令。所有菜单栏能操作事都可以在这里输入命令进行操作。

ctrl+p 打开时，能快速查找文件。

ctrl+r 打开时， 能查找当前文件中的函数。

ctrl+g 打开时，能跳转到指定行。

3.5.2 跨文件跳转函数插件CTags

这个插件能跨文件跳转，跳转到指定函数声明的地方。 使用Package Control 搜索ctags 进行安装（安装ctags插件就可以了， 还有一个 CTags for PHP 插件没什么用）。注意安装好插件后要需要安装ctags命令。window 下载http://prdownloads.sourceforge.net/ctags/ctags58.zip ，解压后找到ctags.exe文件，将ctags.exe文件放在一个环境变量能访问到的地方，例如可放在C:\Program Files\Sublime Text 2目录下，并将该目录放在PATH环境变量中。然后打开cmd， 输入ctags，如果有这个命令，证明成功了。

然后在sublime项目文件夹右键， 会出现Ctag:Rebuild Tags 的菜单。点击它，然后在project的文件夹下面会生成.tags的文件。

在代码中， 光标放在某个函数上， 点击ctrl+shift+鼠标左键 就可以跳转到函数声明的地方。

3.5.3 函数调用关系插件CScope

这个插件能查找函数被哪些函数调用关系。 使用Package Control 搜索cScope 进行安装。注意安装好插件后要需要安装cscope命令。window 下载http://code.google.com/p/cscope-win32，指向http://cscope-win32.googlecode.com/files/cscope-15.8a-win64rev1-static.zip ，解压后找到cscope.exe文件， 将cscope.exe文件放在一个环境变量能访问到的地方，例如可放在C:\Program Files\Sublime Text 2目录下，并将该目录放在PATH环境变量中。然后打开cmd， 输入cscope，如果有这个命令，证明成功了。

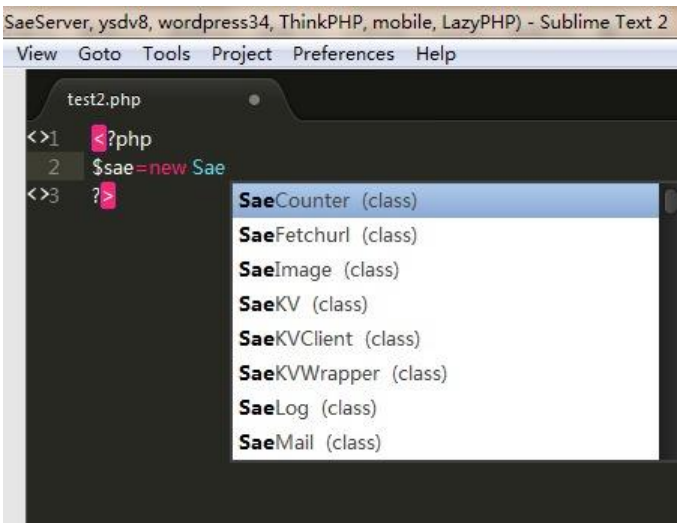
cscope默认只遍历以.c .h .y .l作为文件扩展名的文件，当项目中包含C++或者Java源文件，你就必须要创建cscope.files文件了。

待续。。。

3.5.4 自定义函数智能提示SublimeCodeIntel

Online - http://sublimecodeintel.github.io/SublimeCodeIntel/

sublime默认的代码提示只能提示系统函数，用户自己创建的函数、类不能提示。 如果想要提示自己建立的函数。 可以安装SublimeCodeIntel插件。



sublimecodeintel插件安装后需要配置，sublimeLinter 可以对很多语言进行提示操作。我配置了php ， js 和css 。必须文件名为.php，.js，.css。

配置： Perference -> Package Setting->SublimeLinter->setting – user，复制以下代码：

```
{
    "sublimelinter": "save-only", //在保存时提示
   "sublimelinter_popup_errors_on_save": true, //弹出提示，可直接定位到错误位置
    "sublimelinter_gutter_marks": true,
    "sublimelinter_delay": 1,
   "sublimelinter_executable_map":
   {
      "php": "C:\\Tools\\xampp\\php\\php.exe"
      //"javascript":"D:/nodejs/node.exe",
      //"css":"D:/nodejs/node.exe"
   },
   "jshint_options":
   {
      "strict": true,
      "noarg": true,
      "noempty": true,
      "eqeqeq": true,
      "undef": true,
      "curly": true,
      "forin": true,
      "devel": true,
      "jquery": true,
      "browser": true,
      "wsh": true,
      "evil": true
   },
   "csslint_options":
   {
      "adjoining-classes": "warning",
      "box-model": true,
      "box-sizing": "warning",
      "compatible-vendor-prefixes": "warning",
      "display-property-grouping": true,
      "duplicate-background-images": "warning",
      "duplicate-properties": true,
```

```
        "empty-rules": true,
        "errors": true,
        "fallback-colors": "warning",
        "floats": "warning",
        "font-faces": "warning",
        "font-sizes": "warning",
        "gradients": "warning",
        "ids": "warning",
        "import": "warning",
        "important": "warning",
        "known-properties": true,
        "outline-none": "warning",
        "overqualified-elements": "warning",
        "qualified-headings": "warning",
        "regex-selectors": "warning",
        "rules-count": "warning",
        "shorthand": "warning",
        "star-property-hack": "warning",
        "text-indent": "warning",
        "underscore-property-hack": "warning",
        "unique-headings": "warning",
        "universal-selector": "warning",
        "vendor-prefix": true,
        "zero-units": "warning"
    }
}
```

安装sublimecodeintel后， 按“alt+鼠标左键”也能和ctags一样跳转到函数声明的地方。但是如果有两个文件声明了同样名称的函数， sublimecodeintel只会跳转到第一个找到的函数， 而ctags会让你选择要跳转到哪个文件。所以我们一般还是用ctags的跳转功能。

3.5.5 成对匹配插件BracketHighlighter

安装完Sublime后，像这些符号是成对的：花括号{}， 中括号[],括号：() ，引号“ ” 等。 这些符号当我们鼠标放在开始符号的位置的时候， 希望能明显看到结尾符号在哪儿sublime默认是下划线，很不明显，想要明显一点，推荐安装插件BracketHighlighter。

安装完BracketHighlighter后，需要修改以下文件的"bracket_styles"内容，将一些color与style都打开并修改style为highlight，注意打开后上面的"icon"语句后面需要添加逗号：

修改文件：C:\Users\<username>\AppData\Roaming\Sublime Text 2\Packages\BracketHighlighter\bh_core.sublime-settings

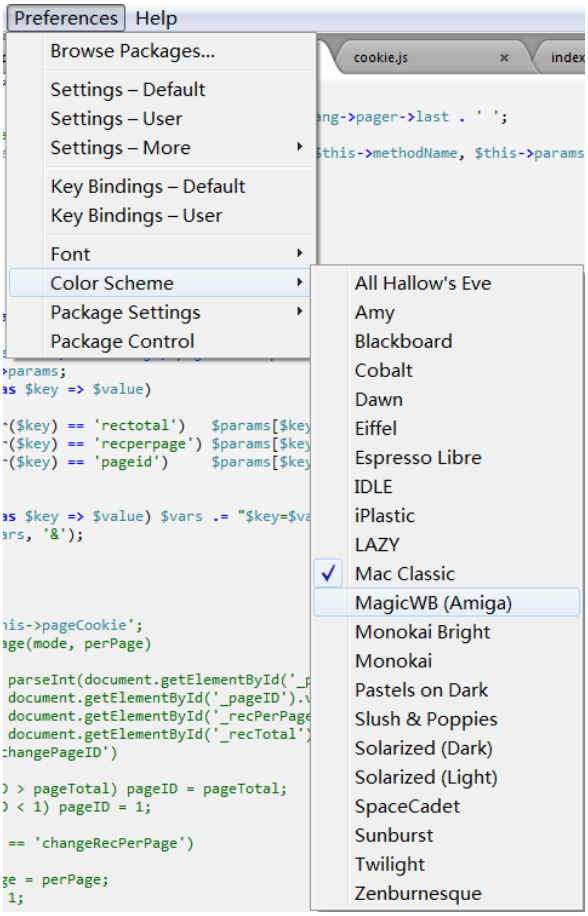
修改以下内容:

```
// Define region highlight styles
"bracket_styles": {
    // "default" and "unmatched" styles are special
    // styles. If they are not defined here,
    // they will be generated internally with
    // internal defaults.

    // "default" style defines attributes that
    // will be used for any style that does not
    // explicitly define that attribute.  So if
    // a style does not define a color, it will
    // use the color from the "default" style.
    "default": {
        "icon": "dot",
        // BH1's original default color for reference
        // "color": "entity.name.class",
        "color": "brackethighlighter.default",
        "style": "highlight"
    },

    // This particular style is used to highlight
    // unmatched bracekt pairs.  It is a special
    // style.
    "unmatched": {
        "icon": "question",
        "color": "brackethighlighter.unmatched",
        "style": "highlight"
    },
    // User defined region styles
    "curly": {
        "icon": "curly_bracket",
        "color": "brackethighlighter.curly",
        "style": "highlight"
```

```
},
"round": {
    "icon": "round_bracket",
    "color": "brackethighlighter.round",
    "style": "highlight"
},
"square": {
    "icon": "square_bracket",
    "color": "brackethighlighter.square",
    "style": "highlight"
},
"angle": {
    "icon": "angle_bracket",
    "color": "brackethighlighter.angle",
    "style": "highlight"
},
"tag": {
    "icon": "tag",
    // "endpoints": true,
    "color": "brackethighlighter.tag",
    "style": "highlight"
},
"c_define": {
    "icon": "hash",
    "color": "brackethighlighter.c_define",
    "style": "highlight"
},
"single_quote": {
    "icon": "single_quote",
    "color": "brackethighlighter.quote",
    "style": "highlight"
},
"double_quote": {
    "icon": "double_quote",
    "color": "brackethighlighter.quote",
    "style": "highlight"
},
"regex": {
```

```
        "icon": "regex"
        // "color": "brackethighlighter.quote",
        // "style": "underline"
    }
},
```

在color scheme查看你所用的配色类型，我用的是Mac Classic，见下图：



在程序安装路径中，C:\Users\<username>\AppData\Roaming\Sublime Text 2\Packages\Color Scheme - Default\文件中找到你所使用的配色文件（我的是Mac Classic.tmTheme），打开并添加11行开始的代码：

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
3  <plist version="1.0">
4  <dict>
5      <key>author</key>
6      <string>Chris Thomas</string>
7      <key>name</key>
8      <string>Mac Classic</string>
9      <key>settings</key>
10     <array>
11         <!-- Bracket 开始 -->
12         ......
13         <!-- Bracket 结束 -->
```

11行开始添加的代码为：

```
<!-- Bracket 开始 -->
                <dict>
                        <key>name</key>
                        <string>Bracket Default</string>
                        <key>scope</key>
                        <string>brackethighlighter.default</string>
                        <key>settings</key>
                        <dict>
                                <key>foreground</key>
                                <string>#FFFFFF</string>
                                <key>background</key>
                                <string>#A6E22E</string>
                        </dict>
                </dict>

                <dict>
                        <key>name</key>
                        <string>Bracket Unmatched</string>
                        <key>scope</key>
                        <string>brackethighlighter.unmatched</string>
                        <key>settings</key>
                        <dict>
                                <key>foreground</key>
                                <string>#FFFFFF</string>
                                <key>background</key>
                                <string>#FF0000</string>
                        </dict>
                </dict>

                <dict>
                        <key>name</key>
                        <string>Bracket Curly</string>
                        <key>scope</key>
                        <string>brackethighlighter.curly</string>
                        <key>settings</key>
                        <dict>
                                <key>foreground</key>
                                <string>#FF00FF</string>
```

```
            </dict>
    </dict>

    <dict>
            <key>name</key>
            <string>Bracket Round</string>
            <key>scope</key>
            <string>brackethighlighter.round</string>
            <key>settings</key>
            <dict>
                    <key>foreground</key>
                    <string>#E7FF04</string>
            </dict>
    </dict>

    <dict>
            <key>name</key>
            <string>Bracket Square</string>
            <key>scope</key>
            <string>brackethighlighter.square</string>
            <key>settings</key>
            <dict>
                    <key>foreground</key>
                    <string>#FE4800</string>
            </dict>
    </dict>

    <dict>
            <key>name</key>
            <string>Bracket Angle</string>
            <key>scope</key>
            <string>brackethighlighter.angle</string>
            <key>settings</key>
            <dict>
                    <key>foreground</key>
                    <string>#02F78E</string>
            </dict>
    </dict>
```

```
<dict>
        <key>name</key>
        <string>Bracket Tag</string>
        <key>scope</key>
        <string>brackethighlighter.tag</string>
        <key>settings</key>
        <dict>
                <key>foreground</key>
                <string>#FFFFFF</string>
                <key>background</key>
                <string>#0080FF</string>
        </dict>
</dict>

<dict>
        <key>name</key>
        <string>Bracket Quote</string>
        <key>scope</key>
        <string>brackethighlighter.quote</string>
        <key>settings</key>
        <dict>
                <key>foreground</key>
                <string>#56FF00</string>
        </dict>
</dict>
<!-- Bracket 结束 -->
```

3.6 Sublime text3

Sublime text 3官网可参见http://www.sublimetext.com/3。 推荐安装Build 3083, Release Date: 26 March 2015及更高版本，低版本存在占用大量CPU与内存的严重BUG。

Sublime Text3技巧使用介绍 可参考 http://jingyan.baidu.com/article/14bd256e2da001bb6d26129f.html

3.6.1 设置空格缺省显示

Prefences -> Settings -> Default

将原来的"draw_white_space": "selection",

修改为"draw_white_space": "all",

3.6.2 设置TAB替换为空格

// Set to true to insert spaces when tab is pressed

"translate_tabs_to_spaces": true,

3.6.3 插件Package Control

点击sublime的菜单栏 view->show console ；现在打开了控制台， 这个控制台有上下两栏， 上面一栏会实时显示sublime执行了什么插件，输出执行结果， 如果你安装的某个插件不能正常运行，应该先在这里看看有没有报错。下面栏是一个输入框，可以运行python代码。我们输入下面的代码点击回车运行， 就能安装好package control了。

在Sublime Text3中安装package control有所变化。

```
import urllib.request,os; pf = 'Package Control.sublime-package'; ipp = sublime.installed_packages_path();
urllib.request.install_opener( urllib.request.build_opener( urllib.request.ProxyHandler()) );
open(os.path.join(ipp, pf), 'wb').write(urllib.request.urlopen( 'http://sublime.wbond.net/' + pf.replace('
','%20')).read())
```

然后我们按住 ctrl+shift+p。此时会输出一个输入框， 输入install。 选择package contrl： install package 回车 ，需要稍定一会儿，右下角状态栏会显示正在连接的提示文字。 使用sublime时注意看右下角状态栏，很多插件的提示信息都显示在这里，这个状态栏很小，初次使用的人都有可能没有注意到它。

稍等一会儿后，它会出现一个插件列表， 你也可以在输入框中输入文字进行搜索插件。 搜索到自己想安装的插件，再选择它，回车。 就自动给你安装好了。

跨文件跳转函数插件CTags不再需要安装。

自定义函数智能提示插件SublimeCodeIntel也不需要安装。

3.6.4 插件SublimeLinter

来源：http://www.cnblogs.com/lhb25/archive/2013/05/02/sublimelinter-for-js-css-coding.html

SublimeLinter 是前端编码利器——Sublime Text 的一款插件，用于高亮提示用户编写的代码中存在的不规范和错误的写法，支持 JavaScript、CSS、HTML、Java、PHP、Python、Ruby 等十多种开发语言。这篇文章介绍如何在 Windows 中配置 SublimeLinter 进行 JS & CSS & PHP 校验。

准备工作：

安装 Sublime Text 包管理工具：http://wbond.net/sublime_packages/package_control

安装 Node.js，建议安装 Windows Installer 版本：http://nodejs.org

使用 Sublime Text 包管理工具安装 SublimeLinter：https://github.com/SublimeLinter/SublimeLinter

SublimeLinter 3不包含linters，所以必须分开安装linter plugins. 使用Ctrl+Shift+P，可以在Package Control中输入“SublimeLinter-<linter>”来查找，例如 “SublimeLinter-jshint”。分别安装3个linter：

SublimeLinter-csslint、

SublimeLinter-jshint、

SublimeLinter-php：https://github.com/SublimeLinter/SublimeLinter-php

```
Package Control Messages
========================

SublimeLinter-php
-----------------

  SublimeLinter-php
  -------------------------------
  This linter plugin for SublimeLinter provides an interface to php -l.
  
  ** IMPORTANT! **
  
  Before this plugin will activate, you *must*
  follow the installation instructions here:
  
  https://github.com/SublimeLinter/SublimeLinter-php
```

安装完成后，会看到安装消息告诉接下来需要做的工作，来继续后面的安装。

```
E:\nodejs>npm install -g jshint
E:\nodejs\jshint -> E:\nodejs\node_modules\jshint\bin\jshint
E:\nodejs
`-- jshint@2.9.4
  +-- cli@1.0.1
  | `-- glob@7.1.2
  |   +-- fs.realpath@1.0.0
  |   +-- inflight@1.0.6
  |   | `-- wrappy@1.0.2
  |   +-- inherits@2.0.3
  |   +-- once@1.4.0
  |   `-- path-is-absolute@1.0.1
  +-- console-browserify@1.1.0
  | `-- date-now@0.1.4
  +-- exit@0.1.2
  +-- htmlparser2@3.8.3
  | +-- domelementtype@1.3.0
  | +-- domhandler@2.3.0
  | +-- domutils@1.5.1
  | | `-- dom-serializer@0.1.0
  | |   +-- domelementtype@1.1.3
  | |   `-- entities@1.1.1
  | +-- entities@1.0.0
  | `-- readable-stream@1.1.14
  |   +-- core-util-is@1.0.2
  |   +-- isarray@0.0.1
  |   `-- string_decoder@0.10.31
  +-- lodash@3.7.0
  +-- minimatch@3.0.4
  | `-- brace-expansion@1.1.8
  |   +-- balanced-match@1.0.0
  |   `-- concat-map@0.0.1
  +-- shelljs@0.3.0
  `-- strip-json-comments@1.0.4
```

```
E:\nodejs>npm install -g csslint
E:\nodejs\csslint -> E:\nodejs\node_modules\csslint\dist\cli.js
E:\nodejs
`-- csslint@1.0.5
  +-- clone@2.1.1
  `-- parserlib@1.1.1
```

参数配置

打开 SublimeLinter 的配置文件，Preferences->Package Settings->SublimeLinter->Settings - User，进行如下配置：

3.6.4.1 运行模式

"sublimelinter": "save-only",

SublimeLinter 的运行模式，总共有四种，含义分别如下：

true - 在用户输入时在后台进行即时校验；

false - 只有在初始化的时候才进行校验；

"load-save" - 当文件加载和保存的时候进行校验；

"save-only" - 当文件被保存的时候进行校验；

推荐设置为“save-only”，这样只在编写完代码，保存的时候才校验，Sublime Text 运行会更加流畅。

其中的lint_mode，表示运行模式，可选的值有background, load/save, save only, 和 manual，这里我设置为了save only，只有才保存时才进行检查。

其中的mark_style，表示出错的显示样式，可选的值有"fill", "outline", "solid underline", "squiggly underline", "stippled underline", 和 "none"，默认值为outline，出错的情况显示如下。

```
"lint_mode": "save-only",
"linters": {},
"mark_style": "outline",
"no_column_highlights_line": false,
"passive_warnings": false,
"paths": {
    "linux": [],
    "osx": [],
    "windows": []
},
"python_paths": {
    "linux": [],
    "osx": [],
    "windows": []
},
"rc_search_limit": 3,
"shell_timeout": 10,
"show_errors_on_save": false,
"show_marks_in_minimap": true,
"sublimelinter": "save-only",
"sublimelinter_executable_map": {
    "css": "E:/nodejs/node.exe",
    "javascript": "E:/nodejs/node.exe",
    "php": "E:\\xampp\\php\\php.exe"
},
```

3.6.4.2 校验引擎

"sublimelinter_executable_map":

{

"javascript":"D:/nodejs/node.exe",

"css":"D:/nodejs/node.exe",

"php": "E:\\xampp\\php\\php.exe" // windows下要使用\\

}

这里是配置 JavaScript 和 CSS 校验需要用到的 JS 引擎（这里用的是 Node.js）的安装路径，配置PHP校验引擎（使用绝对路径）。

3.6.4.3 JSHint 选项

SublimeLinter 使用 JSHint 作为默认的 JavaScript 校验器，也可以配置为 jslint 和 gjslint（closure linter）。下面我使用的 jshint 校验选项，大家可以根据自己的编码风格自行配置，选项的含义可以参考这里：http://www.jshint.com/docs/#options

"jshint_options":

{

```
    "strict": true,
    "noarg": true,
    "noempty": true,
    "eqeqeq": true,
    "undef": true,
    "curly": true,
    "forin": true,
    "devel": true,
    "jquery": true,
    "browser": true,
    "wsh": true,
    "evil": true
}
```

3.6.4.4 CSSLint 选项

SublimeLinter 使用 CSSLint 作为 CSS 的校验器，下面是默认配置的校验选项，可以根据个人编码风格修改：

```
"csslint_options":
{
    "adjoining-classes": "warning",
    "box-model": true,
    "box-sizing": "warning",
    "compatible-vendor-prefixes": "warning",
    "display-property-grouping": true,
    "duplicate-background-images": "warning",
    "duplicate-properties": true,
    "empty-rules": true,
    "errors": true,
    "fallback-colors": "warning",
    "floats": "warning",
    "font-faces": "warning",
    "font-sizes": "warning",
    "gradients": "warning",
    "ids": "warning",
    "import": "warning",
    "important": "warning",
    "known-properties": true,
    "outline-none": "warning",
```

```
    "overqualified-elements": "warning",
    "qualified-headings": "warning",
    "regex-selectors": "warning",
    "rules-count": "warning",
    "shorthand": "warning",
    "star-property-hack": "warning",
    "text-indent": "warning",
    "underscore-property-hack": "warning",
    "unique-headings": "warning",
    "universal-selector": "warning",
    "vendor-prefix": true,
    "zero-units": "warning"
}
```

3.7 Android Studio

Android Studio是Google于2013 I/O大会针对Android开发推出的新的开发工具，目前很多开源项目都已经在采用，而且Android Studio明显优于Eclipse，目前最新的是1.3.1版本（2015-08-07通过canary通道发布），可通过ftp://tools:tools@192.168.1.100访问 Tools\T03-开发工具\Web\Android\ android-studio-bundle-141.2135290-windows.exe来安装1.3.1版本（安装与基本使用可参见下面系列blog中的系列1-系列5），然后安装Genymotion模拟设备（参见下面系列blog中的系列6）。

Android Studio相关的系列blog：

Windows环境下Android Studio系列1—下载与安装, http://my.oschina.net/1pei/blog/467210

Windows环境下Android Studio系列2—初次运行, http://my.oschina.net/1pei/blog/467736

Windows环境下Android Studio系列3—简单设置, http://my.oschina.net/1pei/blog/469674

Windows环境下Android Studio系列4—界面介绍, http://my.oschina.net/1pei/blog/469845

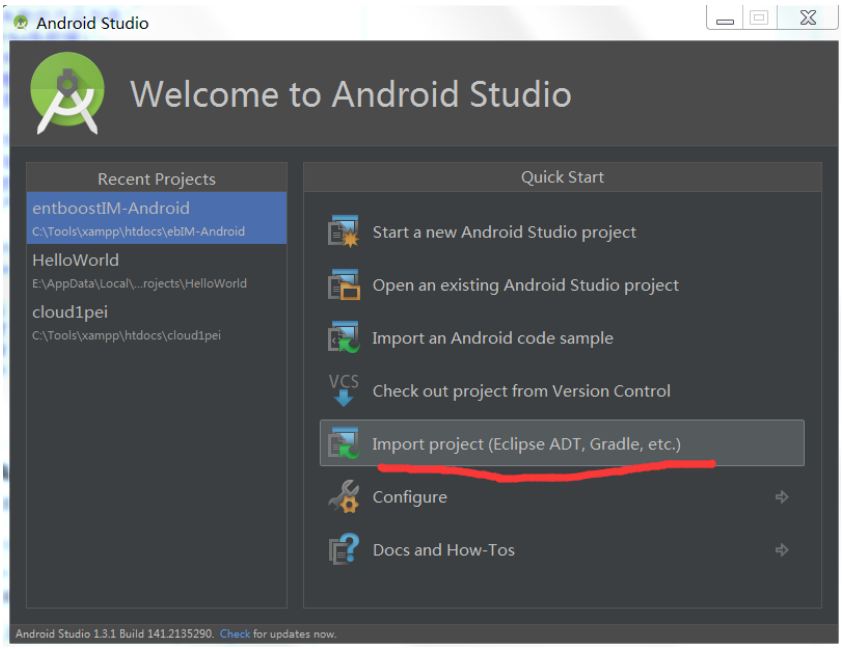
Windows环境下Android Studio系列5—日志调试, http://my.oschina.net/1pei/blog/478672

Windows环境下Android Studio系列6—Genymotion模拟设备, http://my.oschina.net/1pei/blog/480076

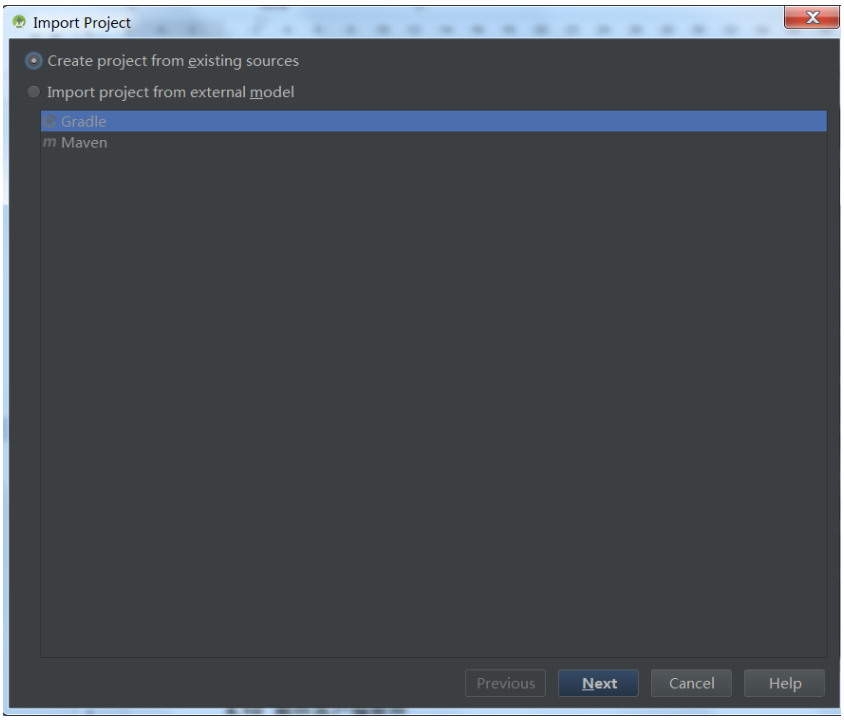
Windows环境下Android Studio系列7—ADB，http://my.oschina.net/1pei/blog/482484

3.7.1 从源码转到Android Studio中的项目

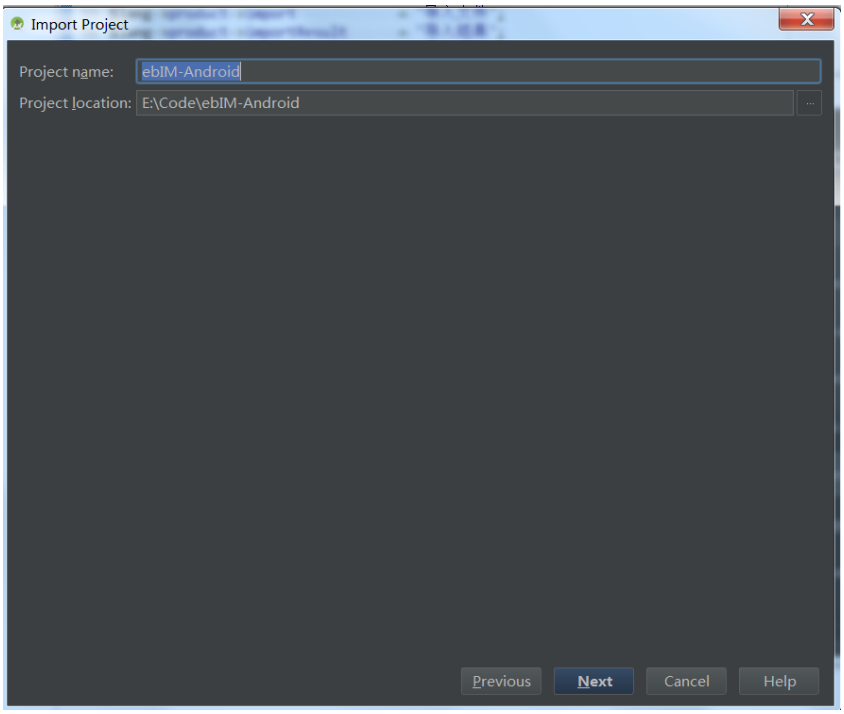
在通过Git下载到ebIM-Android代码后，按照以下方式操作：



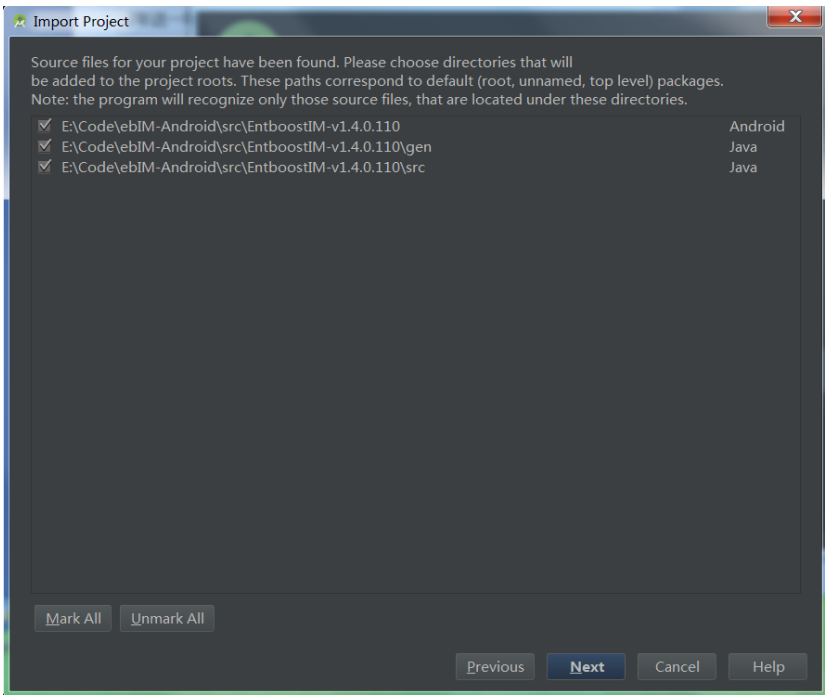
选择“Import project(Eclipse ADT, Gradle, etc)”后出现下面窗口：



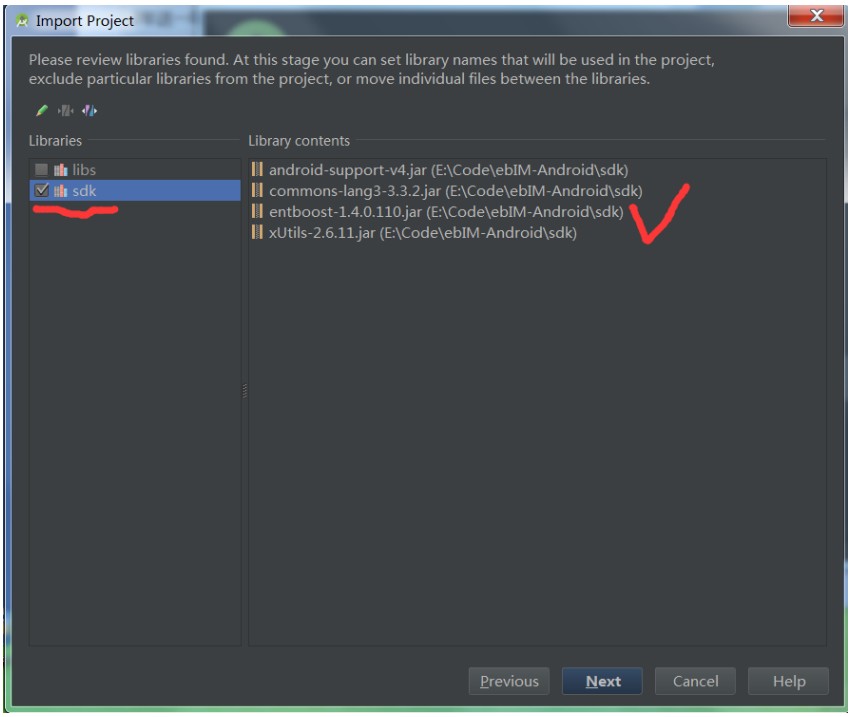
选择“Create project from existing sources”，点击Next按钮：



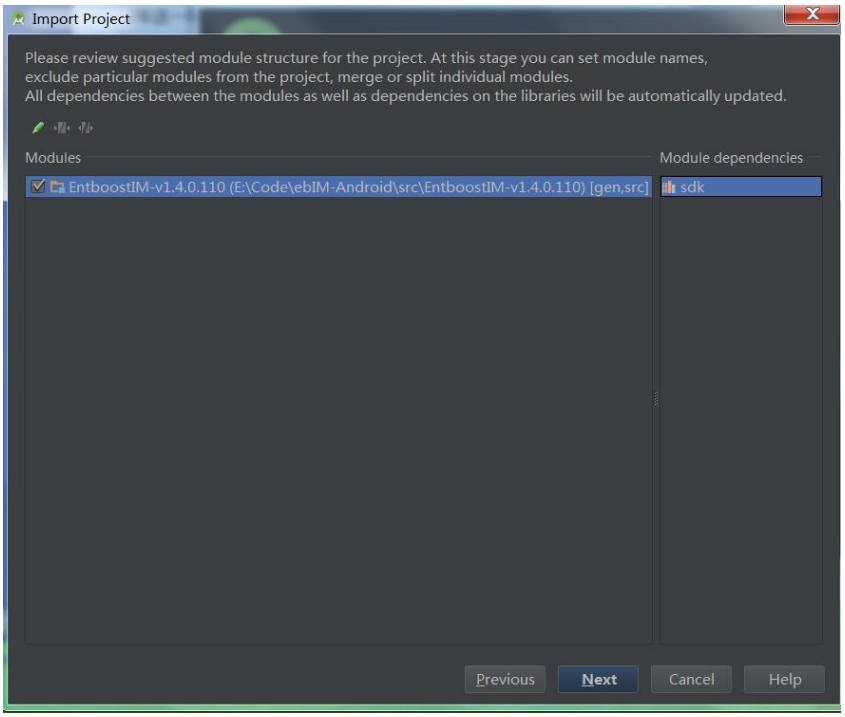
设置好项目源代码路径后，点击”Next”按钮：



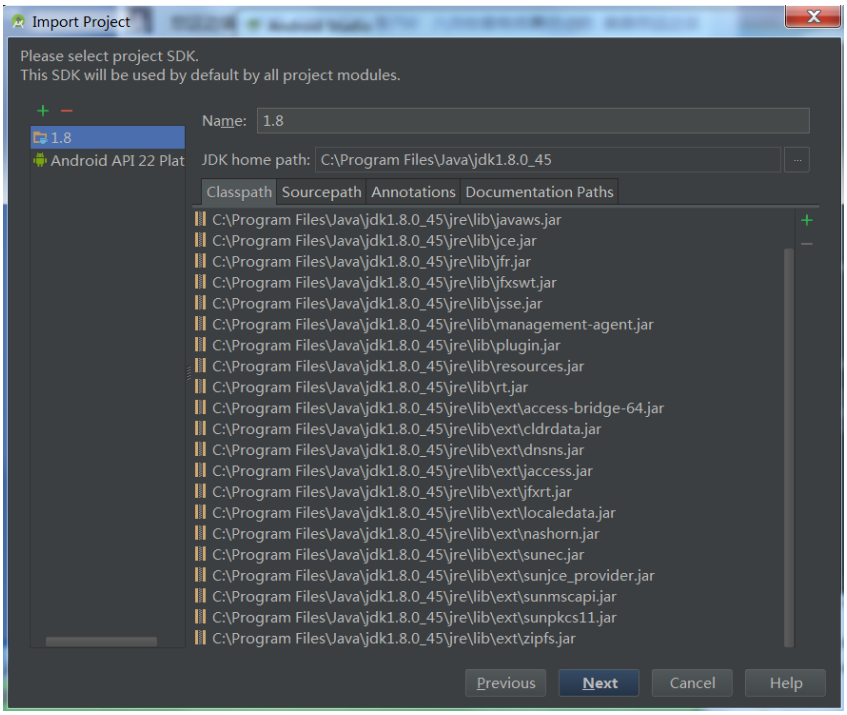
选择源代码路径为E:\Code\ebIM-Android\src\Entboost-v1.4.0.110目录，点击”Next”按钮：

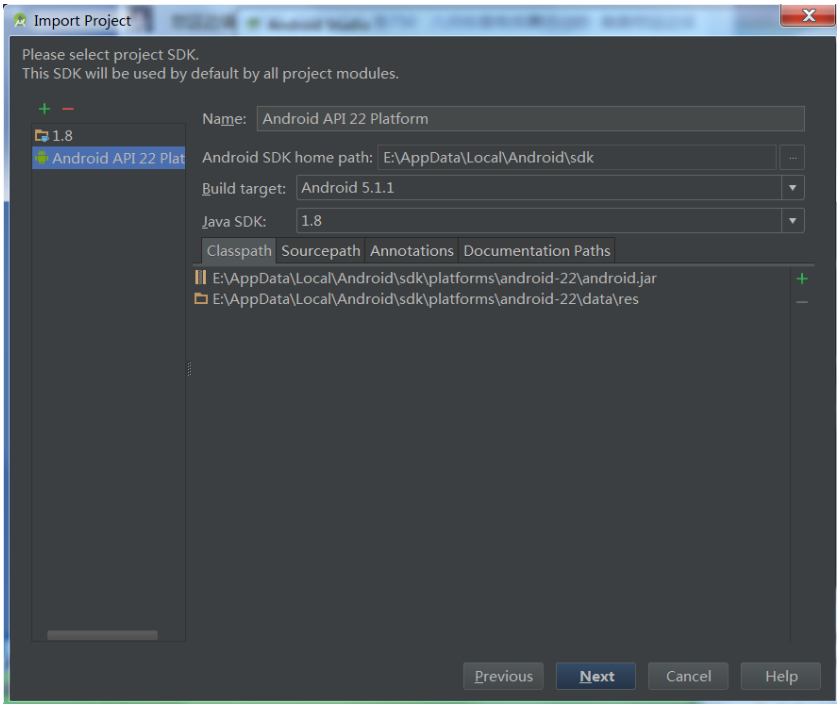


然后选择sdk，不要选择libs目录，点击”Next”按钮：

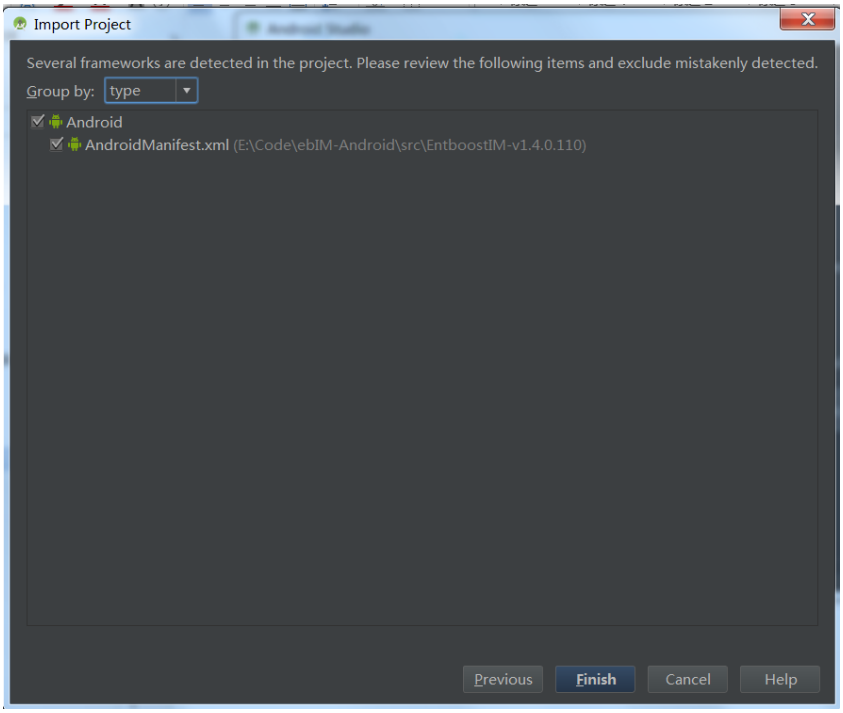


点击”Next”按钮：





当前采用的JDK是1.8，Android API是22，点击”Next”按钮：



点击”Finish”按钮，完成设置。

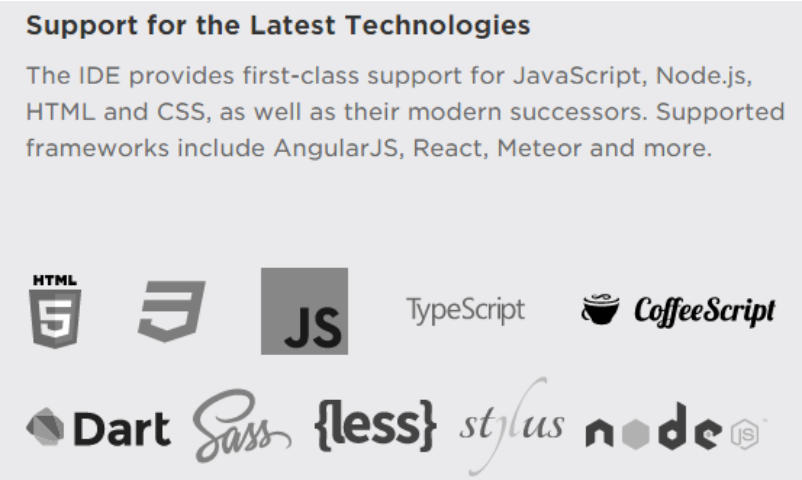
但这种导入缺省是Maven方式编译方式，还没有转为缺省的Gradle编译。如何转为Gradle方式来编译呢？

3.7.2 从源码转到Android Studio中的项目

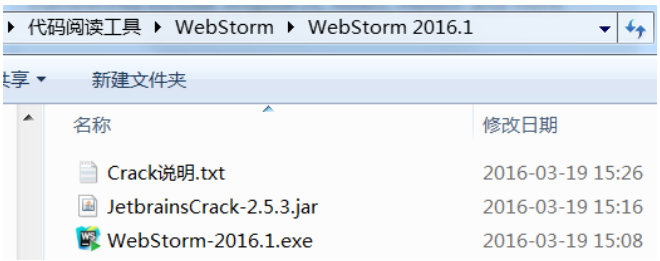
3.8 WebStorm

WebStorm 是jetbrains公司旗下一款JavaScript 开发工具。被广大中国JS开发者誉为“Web前端开发神器”、“最强大的HTML5编辑器”、“最智能的JavaScript IDE”等。与IntelliJ IDEA同源，继承了IntelliJ IDEA强大的JS部分的功能。

WebStorm版本2016.1（Build: 145.258，Released: March 17, 2016）对Node.js、ECMAScript 2015、TypeScript 1.8、CoffeScript、Angular2、React、Metor、Dart、less、TSLint等支持良好。



可访问ftp://tools:tools@192.168.1.100，从下面访问WebStorm 2016.1版本：



3.9 Atom

ATOM是由 GitHub 打造的一个现代的代码编辑器； 它开源免费跨平台，并且整合 GIT 并提供类似 SublimeText 的包管理功能，支持插件扩展，可配置性非常高；

功能上非常丰富，支持各种编程语言的代码高亮、 与大多数其他编辑器相比，Atom的语言支持已经算是覆盖非常全面了。

开发团队将 Atom 称为一个“为 21 世纪创造的可配置的编辑器”，它拥有非常精致细腻的界面，并且可配置项丰富，加上它提供了与 SublimeText 上类似的 Package Control (包管理)功能，人们可以非常方便地安装和管理各种插件，并将 Atom 打造成真正适合自己的开发工具。

作为一个现代的代码编辑器，Atom 有着各种流行编辑器都有的特性，功能上非常丰富，支持各种编程语言的代码高亮(HTML / CSS / Javascript / PHP / Python / C / C++ / Objective C / Java / JSON / Perl / CoffeeScript / Go / Sass / YAML / Markdown 等等)、 与大多数其他编辑器相比，Atom的语言支持已经算是覆盖非常全面了。另外，它的代码补全功能（也叫Snippets） 也非常好用，你只需输入几个字符即可展开成各种常用代码，可以极大提高编程效率。

另外，Atom同样支持 SublimeText 的几个重要的功能，譬如 Goto Anything、Goto Symbol、Goto Line、命令面板等等，就连快捷键也是一模一样的！相信 ST 的用户将会比较轻松地过渡到 Atom 去。实际上，从功能上来看，目前的 Atom 编辑器基本就是 Sublime 的一个复刻版，只是技术实现方式不同，插件的技术也有所差异。

安装Atom

1、 打开Atom官网https://atom.io/，点击Download Windows Installer下载安装包。

2、 下载之后双击“AtomSetup.exe”等待，他会默默的帮你安装，安装完之后在桌面会有快捷方式。

安装完成后，可以安装JSLint、CSSLint、

PHP-Class-Tree、Linter-PHP（配置php.exe所在的路径：C:/Tools/xampp/php/）等插件。

但目前还没有发现这些插件好用。

4. Web Server软件

4.1 LAMP、WAMP与XAMPP

XAMPP官方网站：https://www.apachefriends.org/index.html。进入官网后建议安装7.1.6最新版本。

LAMP是基于Linux，Apache，MySQL/MariaDB和PHP的开放资源网络开发平台，开发者在Windows操作系统下使用这些Linux环境里的工具称为使用WAMP。

Windows下的Apache + Mysql/MariaDB + Perl/PHP/Python，一组常用来搭建动态网站或者服务器的开源软件，本身都是各自独立的程序，但是因为常被放在一起使用，拥有了越来越高的兼容度，共同组成了一个强大的Web应用程序平台。

总的来说，无论从安全性和性能上来讲，LAMP都优于WAMP，不过由于Windows具有易用的特点，WAMP也未尝不是初学者的一个不错的选择。

XAMPP（Apache+MySQL+PHP+PERL）是一个功能强大的建 XAMPP 软件站集成软件包。这个软件包原来的名字是 LAMPP，但是为了避免误解，最新的几个版本就改名为 XAMPP 了。

它可以在Windows、Linux、Solaris、Mac OS X 等多种操作系统下安装使用，支持多语言：英文、简体中文、繁体中文、韩文、俄文、日文等。

XAMPP只能用于开发环境，不能应用于生产环境。

4.2 XAMPP安装与启动

由于一些php API在最新的PHP 7.1.*/7.2.*版本中已经删除，所以尽量确保本地XAMPP总是最新版本。我们所有开发环境要求安装PHP 7.1.27对应的XAMPP版本。

2017.12.20发布了XAMPP with PHP 7.2.0版本。

PHP to 7.2.0

Apache to 2.4.29

MariaDB to 10.1.29

OpenSSL to 1.0.2n

phpMyAdmin 4.7.6

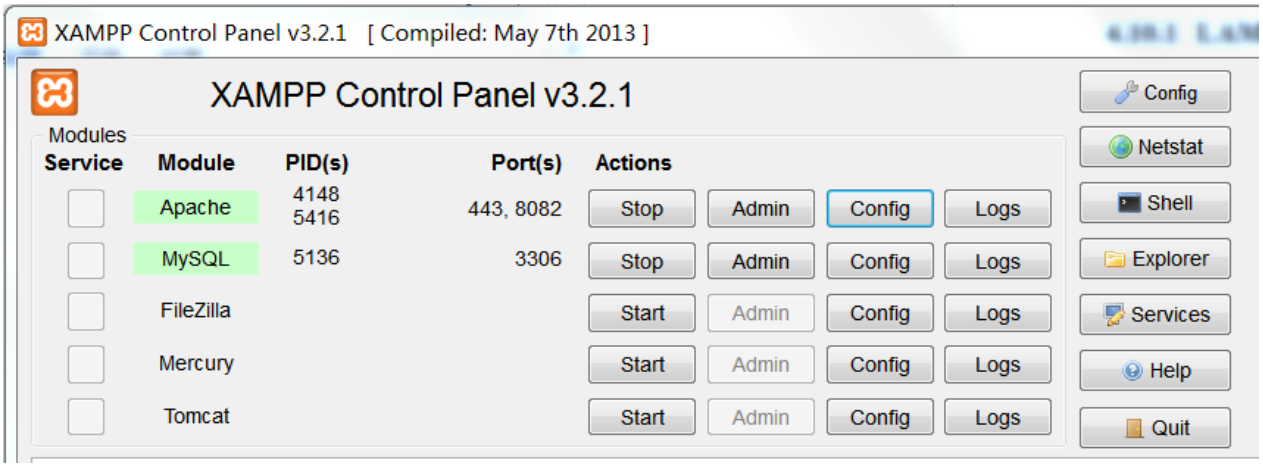
2017.10.20发布了XAMPP with PHP 7.1.10版本。

……

2015.07.23发布了XAMPP with PHP 5.6.11版本。

通常只需要启动Apache与MySQL两个服务。

注意：如果在windows10的开发环境下会出现Apache服务启动不了的情况，检查Netstat会看到80端口被其他进程监听，通常是Windows 10上缺省安装的IIS服务导致，可以去关闭IIS服务即可解决问题。



安装并启动XAMPP的Apache与MySQL模块后，可参照《1pei-IT 2015-0003-一配云平台简介和开发部署》文档在自己的机器上部署一配云平台的测试环境。

XAMPP安装后的路径：

路径	内容
\xampp\anonymous	匿名 FTP 的样例文件夹
\xampp\apache	Apache 服务器
\xampp\cgi-bin	可执行的 CGI 脚本
\xampp\FileZillaFTP	FileZilla FTP 服务器
\xampp\htdocs	http 文档的主文件夹
\xampp\install	用于 XAMPP 的安装（请勿删除！）
\xampp\licenses	同上
\xampp\MercuryMail	Mercury 邮件 SMTP POP3 IMAP 服务器
\xampp\mysql	MySQL 服务器
\xampp\perl	Perl
\xampp\php	PHP（4 和 5）
\xampp\phpmyadmin	phpMyAdmin
\xampp\security	额外的安全配置
\xampp\tmp	临时文件夹
\xampp\webalizer	Webalizer 网络状态
\xampp\webdav	WebDAV 样例

4.3 XAMPP版本升级

如果之前使用的是XAMPP 7.1.9, 安装在E:|\xampp目录下。

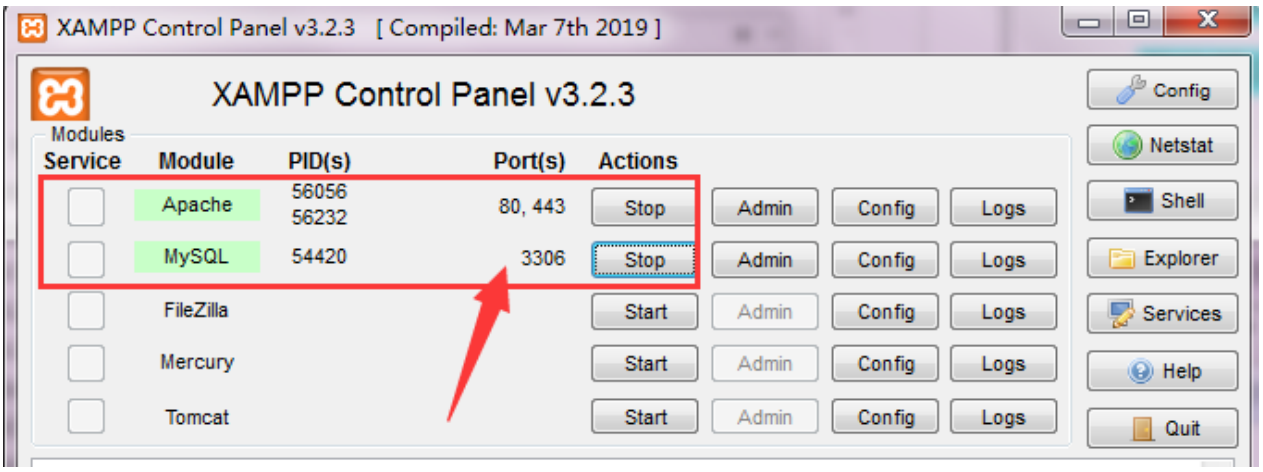
现在想要升级到XAMPP 7.1.27，安装在E:\xampp7127目录下，怎么来做呢？

步骤1：解压下载的XAMPP 件夹的名称修改为“xampp7127”然后进入下一步。一路安装。安装完成后在浏览器中输入检查php info：

PHP Version 7.1.27

System	Windows NT SKY-20180728BUM 6.1 build 7601 (Windows 7 Ultimate Edition Service Pack 1) AMD64
Build Date	Mar 6 2019 20:46:26
Compiler	MSVC14 (Visual C++ 2015)
Architecture	x64
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\xampp\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20160303
PHP Extension	20160303
Zend Extension	320160303
Zend Extension Build	API320160303,TS,VC14
PHP Extension Build	API20160303,TS,VC14
Debug Build	no
Thread Safety	enabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	php, file, glob, data, http, ftp, zip, compress.zlib, compress.bzip2, https, ftps, phar
Registered Stream Socket Transports	tcp, udp, ssl, sslv3, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	convert.iconv.*, mcrypt.*, mdecrypt.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, zlib.*, bzip2.*

启动XAMPP control，只需要Apache和MySQL启动就可以了：



步骤2：需要将之前的xampp

5. 代码控制工具Git

代码版本工具经历了从CVS、SVN、Git的发展过程，我们内部通过Git进行代码版本控制。Windows上Git客户端工具分为以下两种：

- 命令行式的：gitbash（参见http://www.git-scm.com/download/win）
- 图形化的：TortoiseGit, sourcetree等。

开发人员可以根据自己的喜好在自己的windows客户端来安装Git客户端工具，这里我们推荐采用TortoiseGit。

在服务器192.168.1.160上已经安装了Git Server。

5.1 关于TortoiseGit

在2008年，Frank Li发现Git缺少一个好用的GUI client界面. The idea for a Git client as a Windows shell integration was inspired by the similar client for SVN named TortoiseSVN.

Frank研究了TortoiseSVN代码，将它用作TortoiseGit开发的基础代码。然后开始研发，在code.google.com上注册了该项目，并开放了源代码。

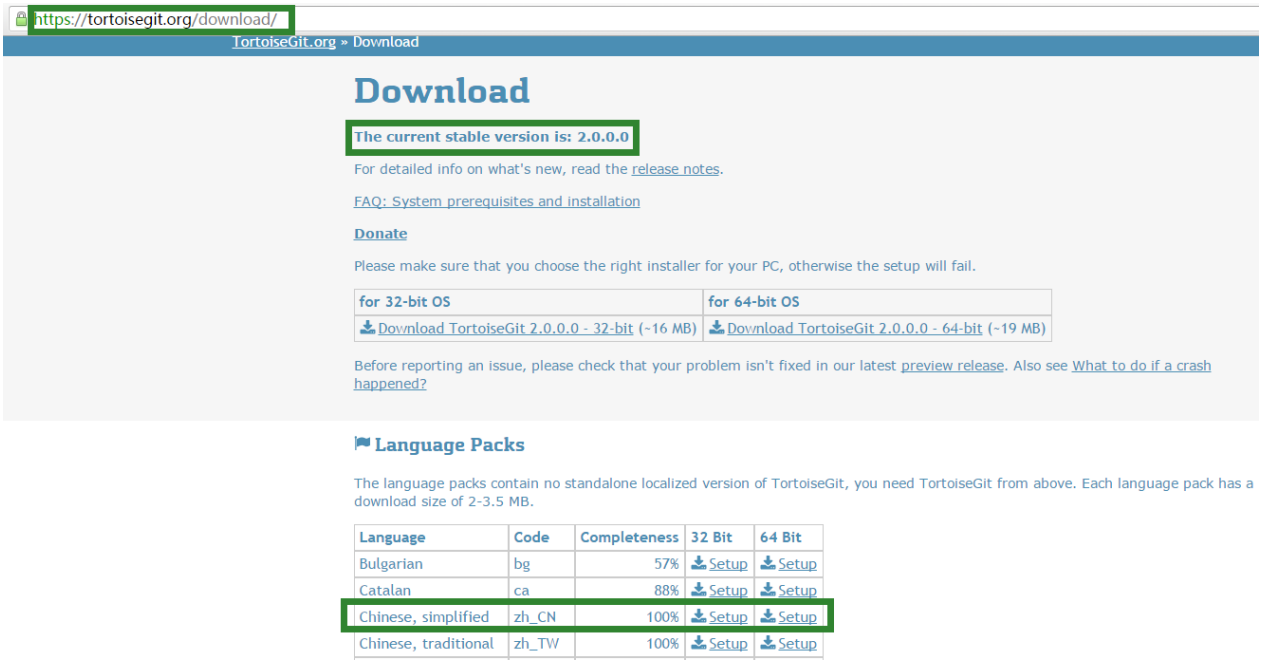
5.2 windows下安装Git客户端

参考资料：https://blog.csdn.net/xiaobin_hlj80/article/details/10953701

每个开发人员在自己的windows客户端安装图形化的Git客户端tortoisegit的步骤如下:

步骤1：从https://msysgit.github.io/ 或从http://www.git-scm.com/download/win下载git for windows 2.21.0（2019-02-27发布）版本。

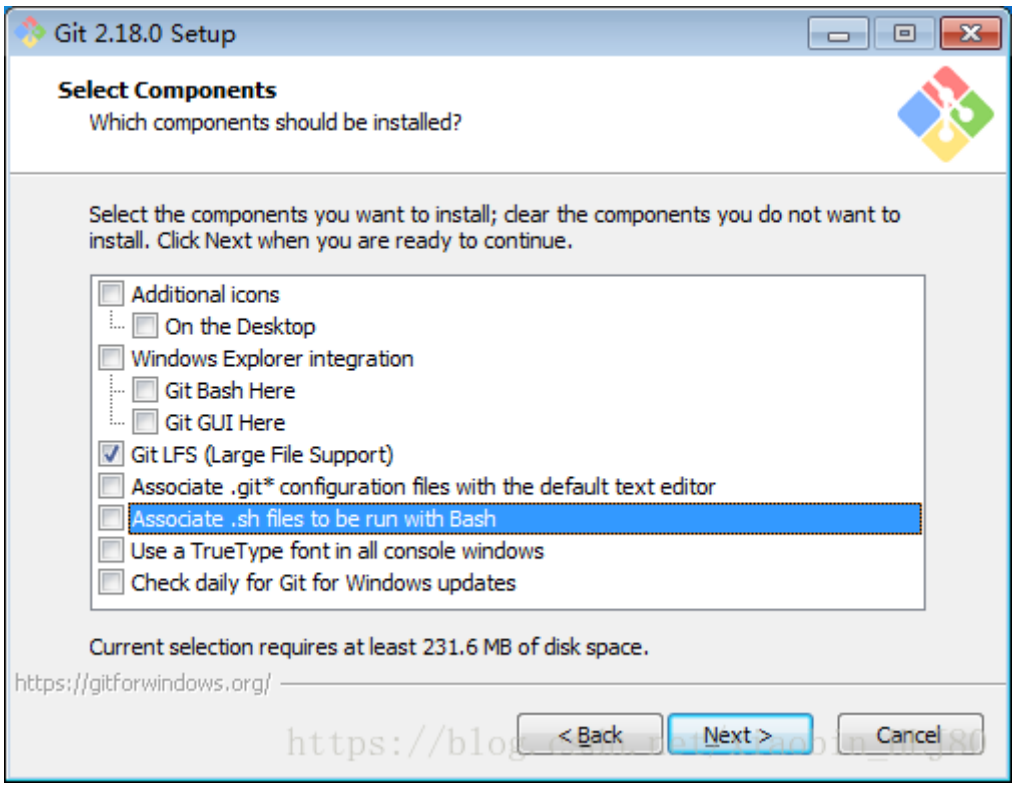
步骤2：从 https://tortoisegit.org/download/ 下载 TortoiseGit 2.8.0版本（2019-02-28发布），注意Git for Windows 2.19.0 or newer is now required。

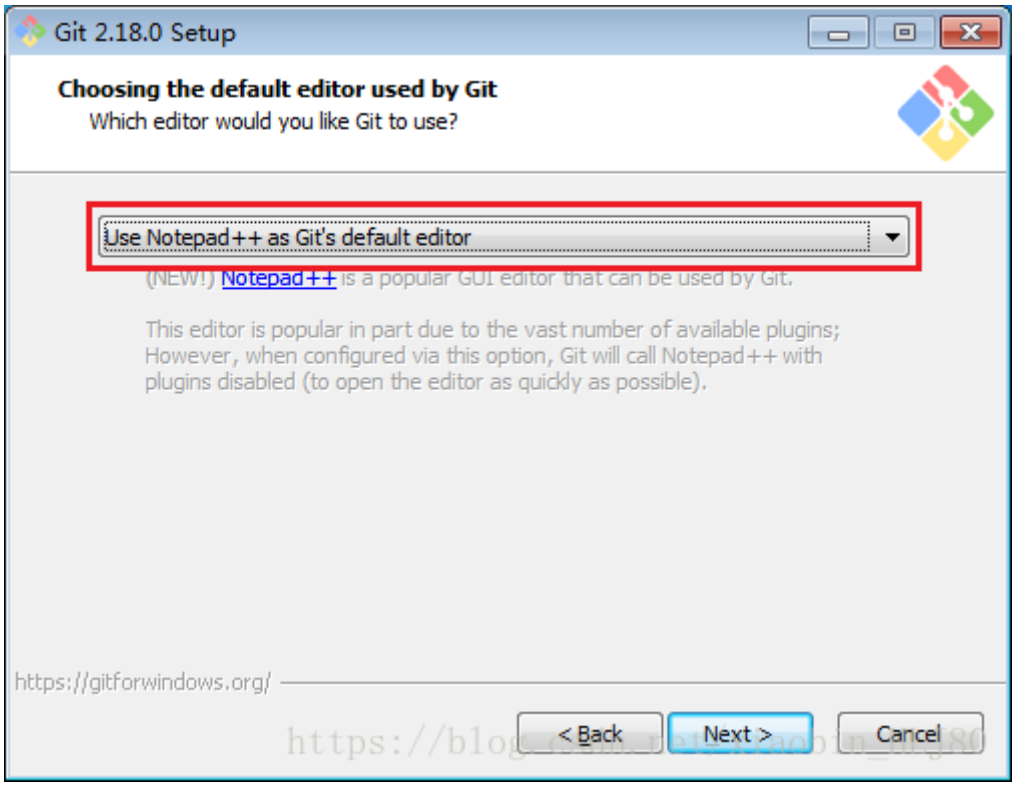


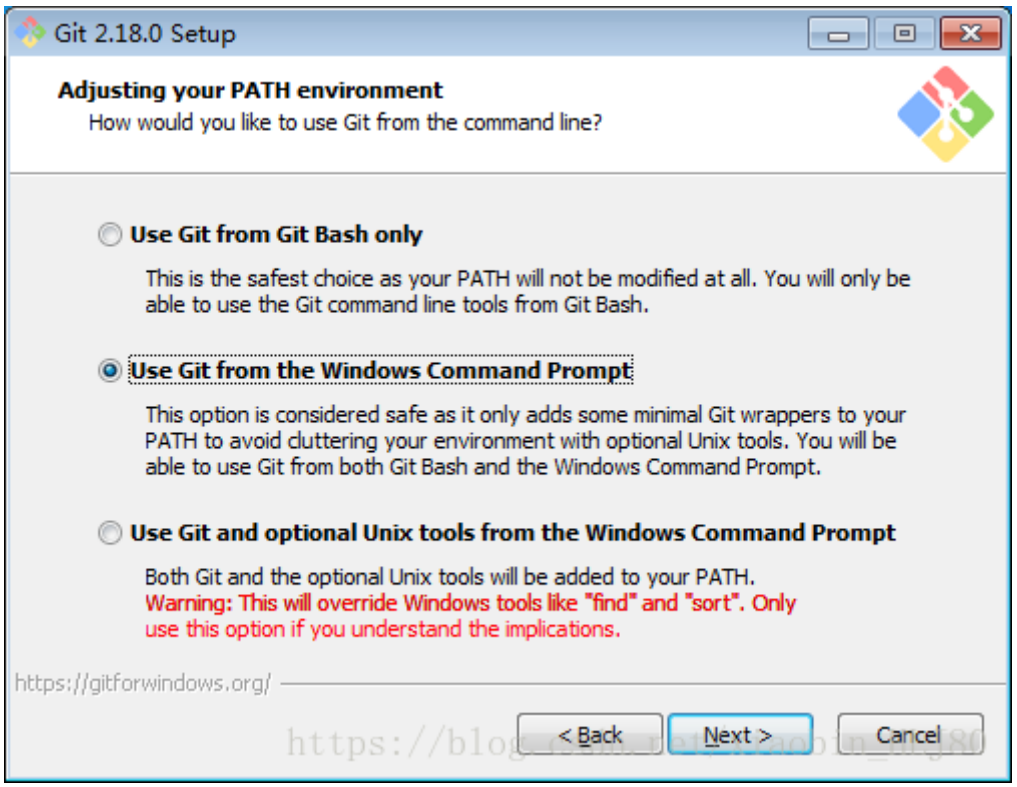
步骤3：安装顺序：先安装TortoiseGit 2.8.0版本, 然后安装git for windows 2.21.0版本。

安装git for windows时注意事项：

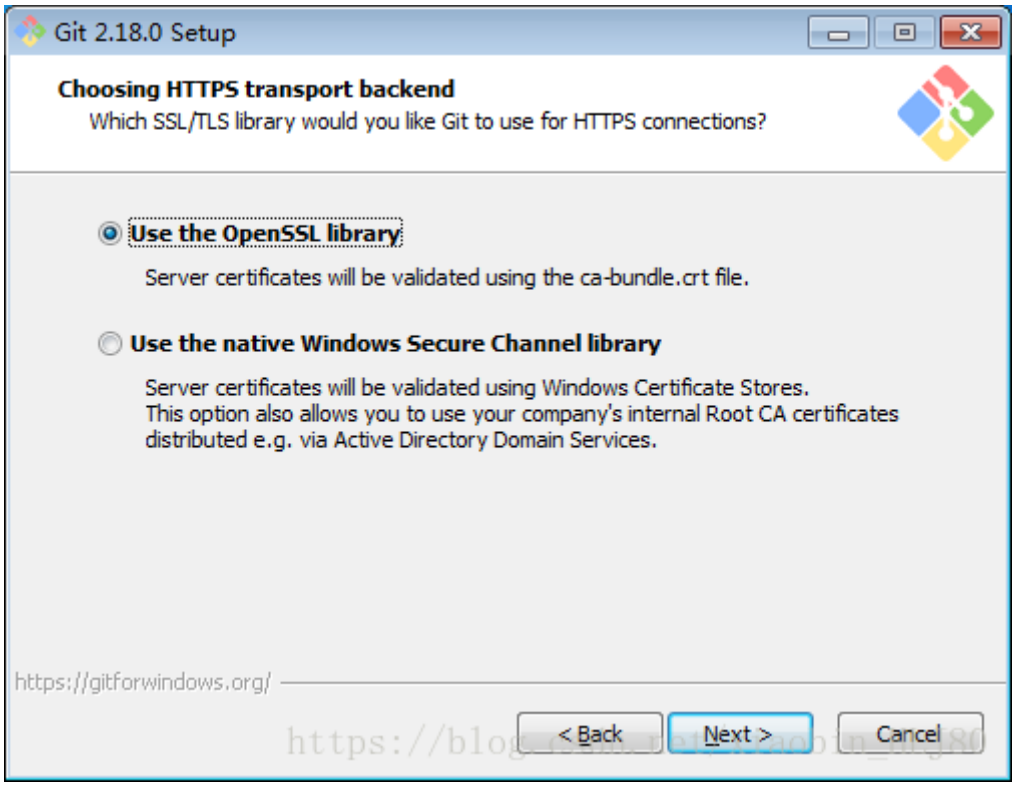
(1) 当安装Git for Windows时，不需要安装"Windows explorer integration".



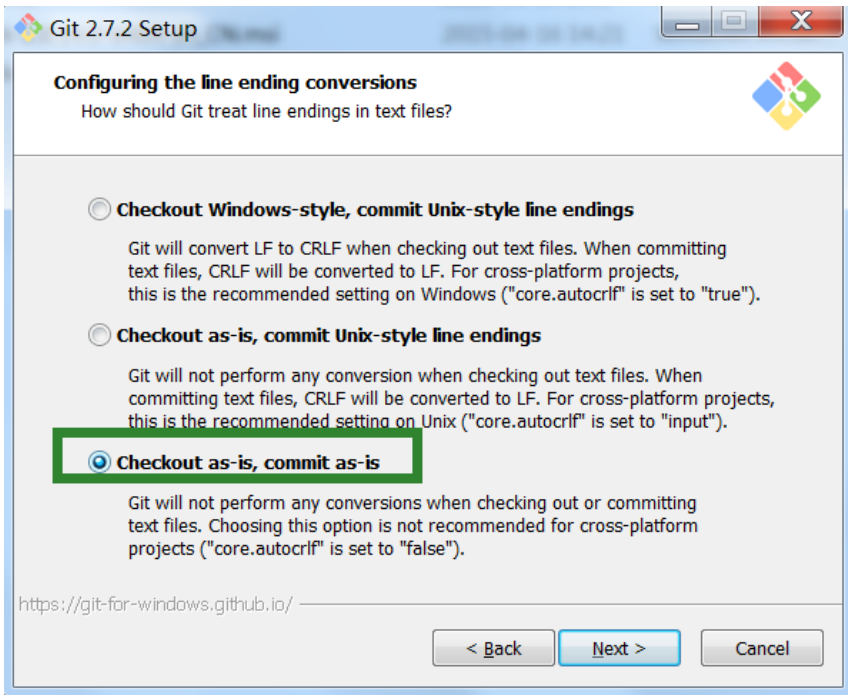


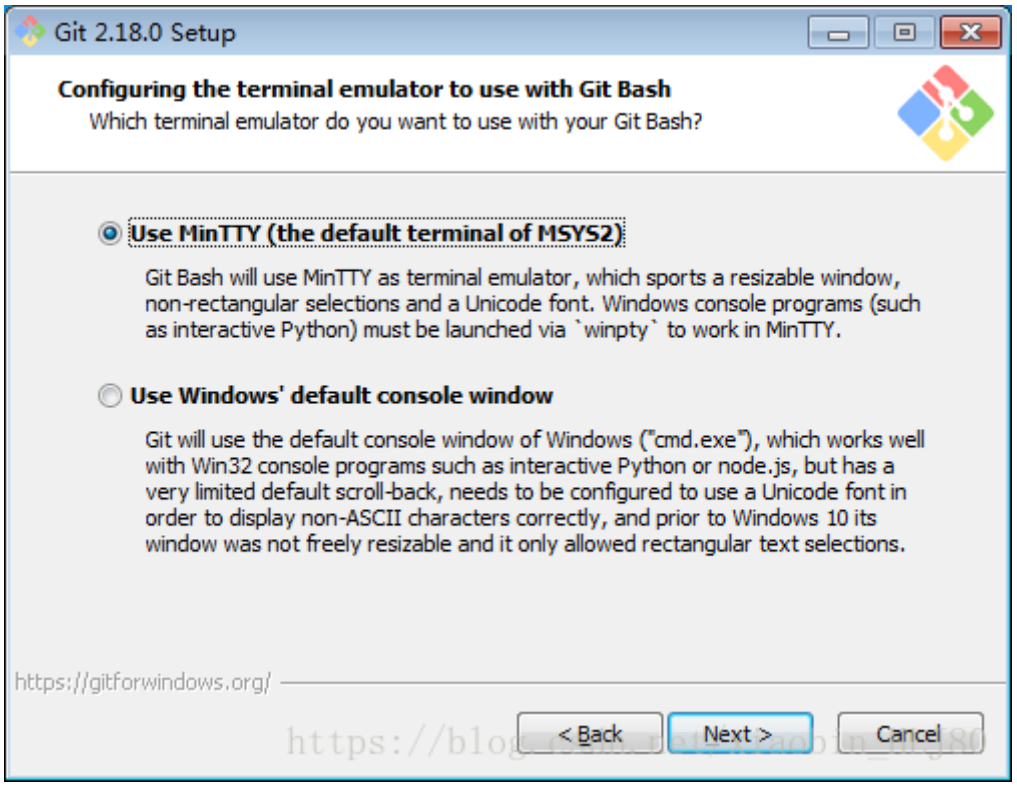


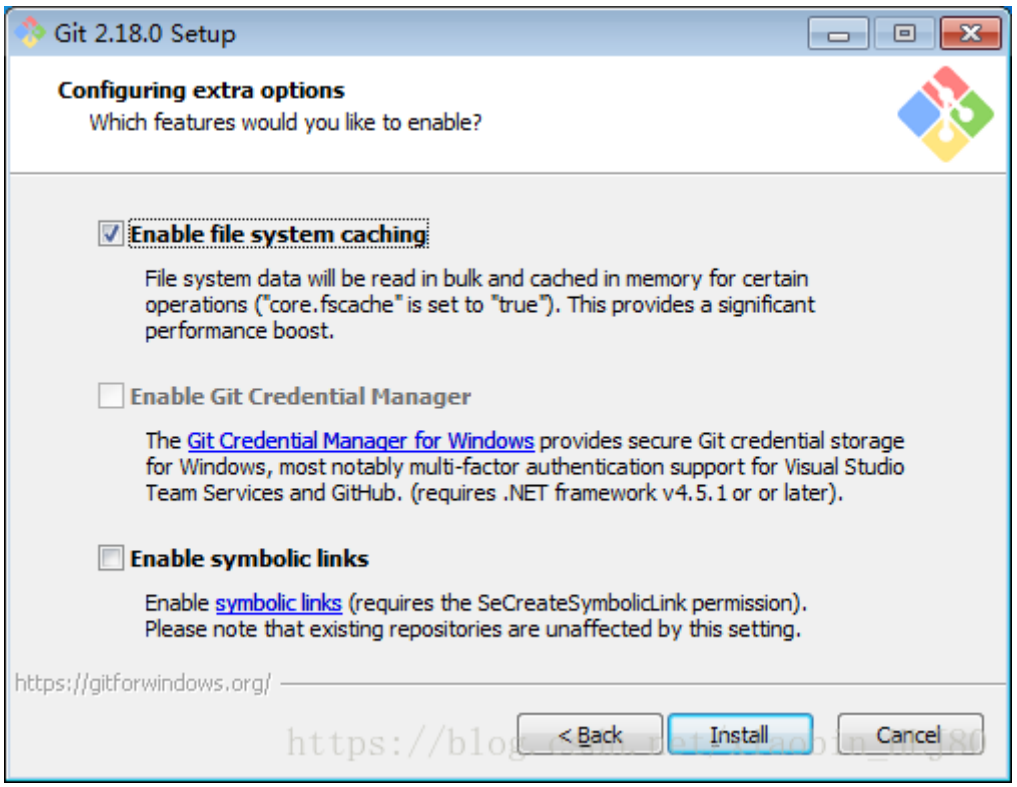
(2) 选择openSSH，这里Plink没有用过以后再研究。



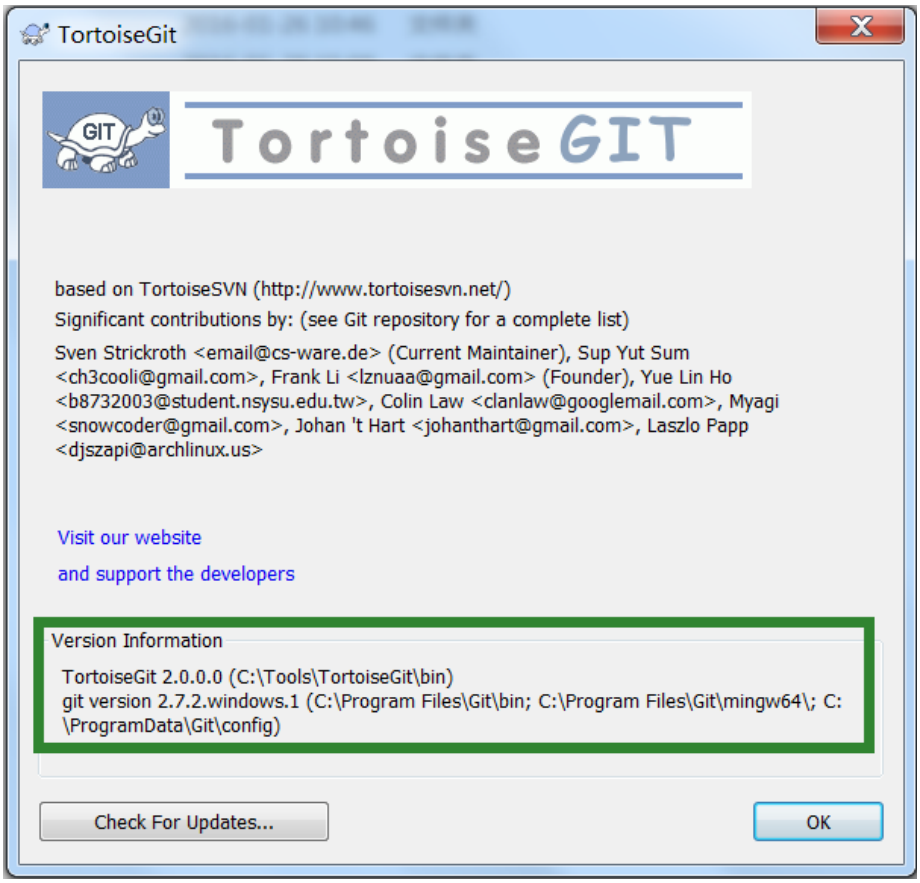
(3) 应当选择"Checkout as-is, commit as-is"以便阻止编辑器做换行自动转换。







安装完成后鼠标右键，进入TortoiseGit > About可查看相关的版本信息：



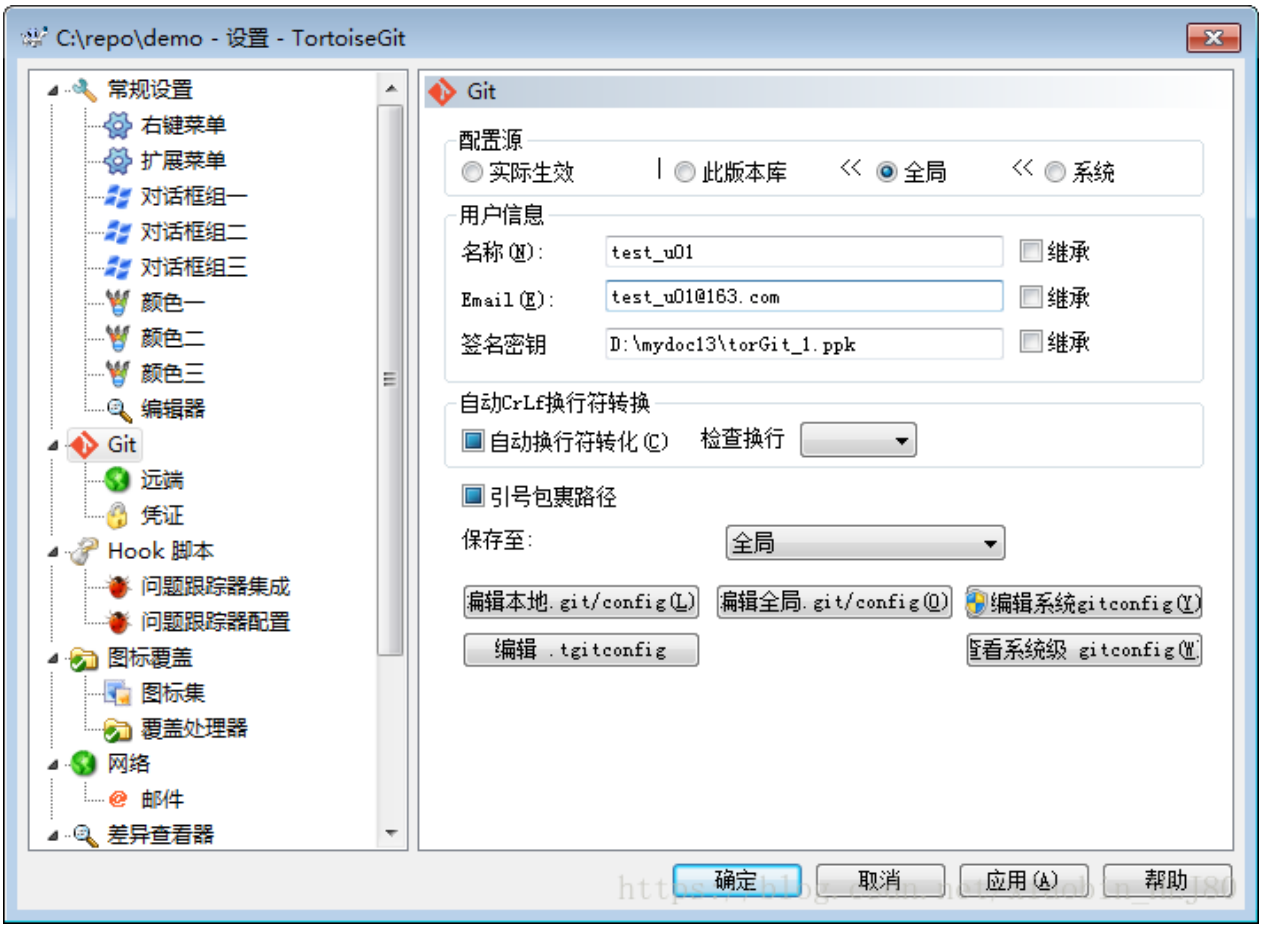
TortoiseGit官方的使用文档可参考：http://tortoisegit.org/docs/tortoisegit。

5.3 windows下配置Git客户端

安装完成Git客户端TortoiseGit后需要做以下一些配置：

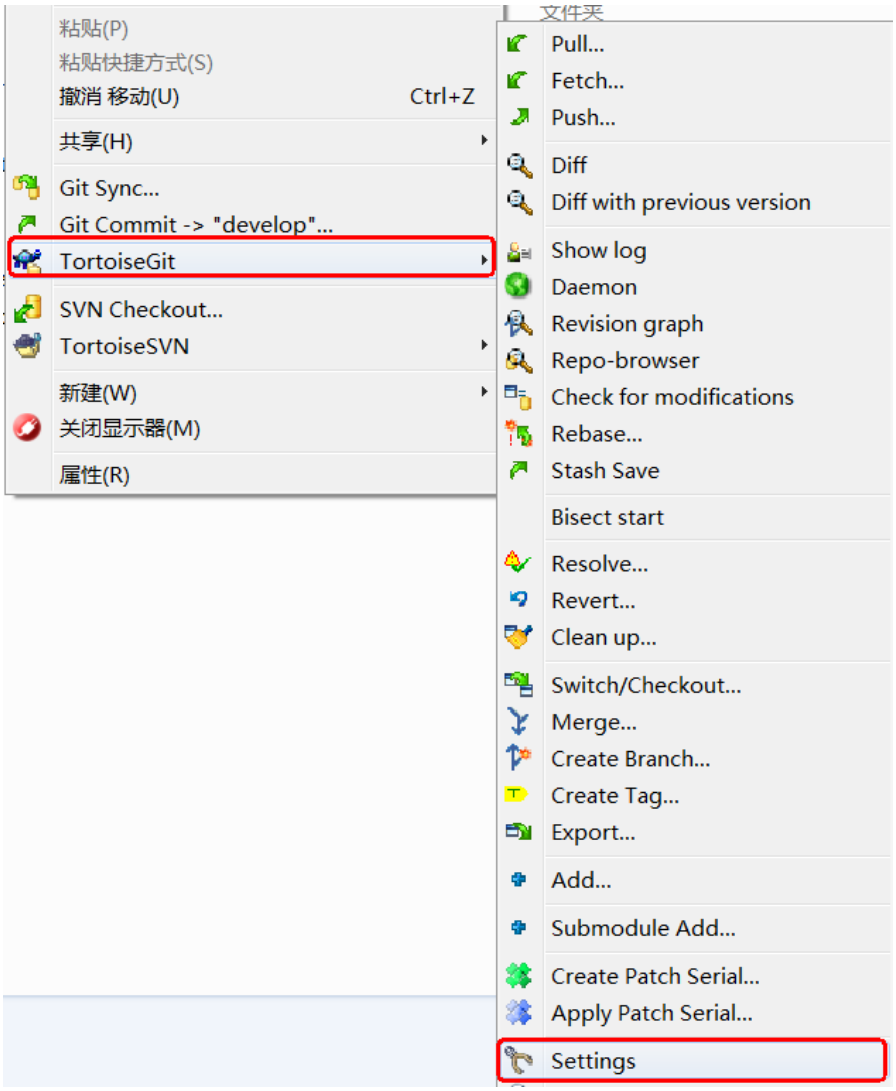
5.3.1 配置签名文件

左侧点击Git，配置源选择全局，输入用户名、Email、和签名密钥文件.ppk（这个文件在下面5.4.1章节中来生成）。

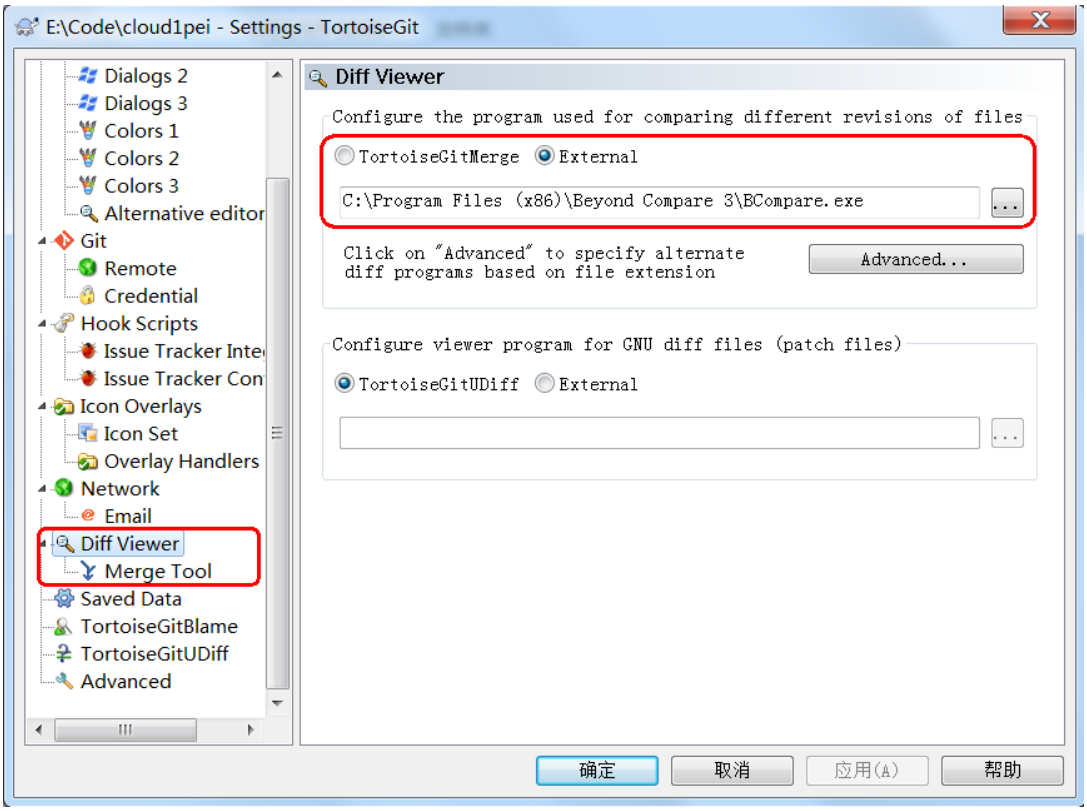


5.3.2 配置文件diff采用Beyond Comapre工具来比较

点击鼠标右键→Tortoise→Setting来修改配置，参见下图：

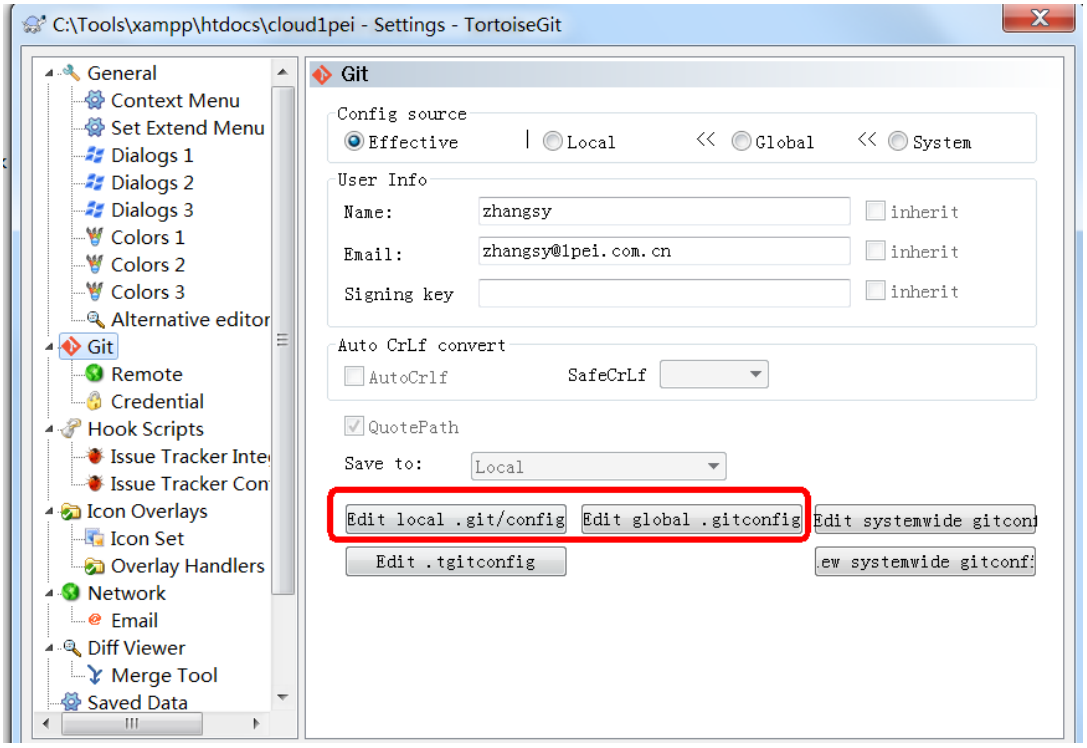


在Setting窗口中点击”Diff Viewer”，选择“External”来设置Byeond Comapre工具对应的exe可执行文件作为外置差异比较工具。



5.3.3 配置文件名大小写敏感

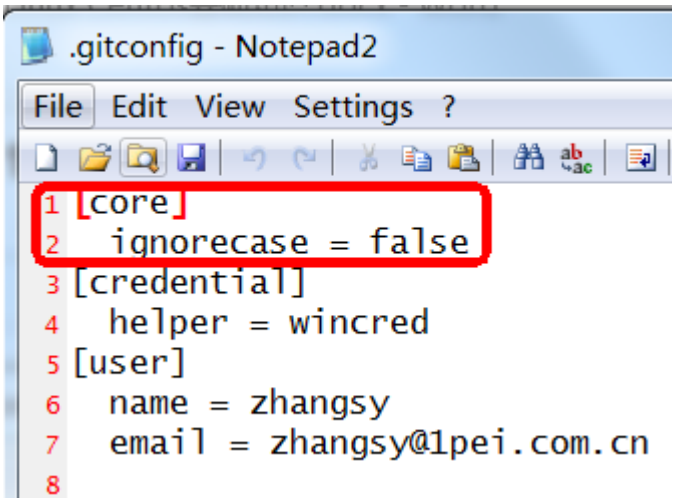
由于代码后续需要在Linux上运行，而Linux下对文件名大小写敏感。因此在Windows Git下需要设置对文件名大小写敏感，点击TortoiseGit→Setting后出现下面窗口：



Edit Local .git/config文件，修改[core]下ignorecase=false后保存退出：

```
[core]
  repositoryformatversion = 0
  filemode = false
  bare = false
  logallrefupdates = true
  symlinks = false
  ignorecase = false
  hideDotFiles = dotGitOnly
[remote "origin"]
  url = git@192.168.1.160:/home/git/repositories/cloud1pei.git
  fetch = +refs/heads/*:refs/remotes/origin/*
  puttykeyfile = D:\\Tools\\T03-开发工具\\Switch终端\\putty0.64\\zhangsy.ppk
[branch "master"]
  remote = origin
  merge = refs/heads/master
[branch "develop"]
  remote = origin
  merge = refs/heads/develop
```

Edit Global .gitconfig文件，增加[core]下ignorecase=false后保存退出：



5.4 公钥生成

每个开发人员在自己的windows机器上安装完tortoisegit后，需要生成公钥。生成公钥有两种方法，一种是利用puttygen.exe窗口界面生成，一种是利用gitbash命令行生成，推荐采用第2种方式来生成。

5.4.1 利用puttygen.exe生成公钥与密钥对

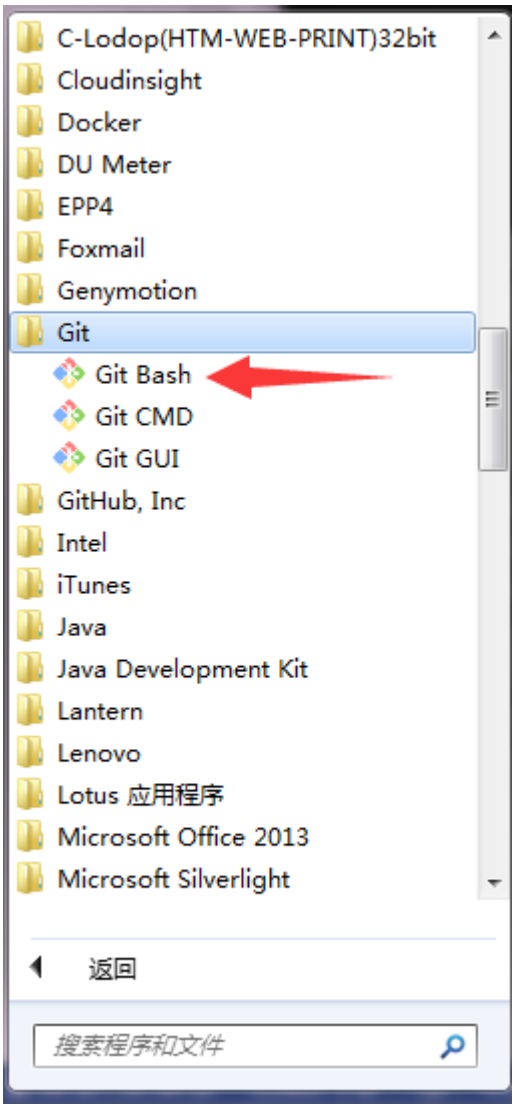
可参见http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html来下载putty包。 putty包中包含PAGEANT.exe, plink.exe, pscp.exe, psftp.exe, putty.exe, puttygen.exe等文件。**参照 http://blog.csdn.net/shiqidide/article/details/8110958利用puttygen.exe生成一个公钥与密钥对文件，生成文件时注意3点：**

- Types of key选择：SSH-2 RSA, Number of bits选择2048.
- password: XXXXXX（说明：这里请自行设置并牢记）
- 鼠标需要在空白处移动，否则生成密钥的进度就停滞不前了。

生成的私有文件（用户名为zhangsan则保存文件名为zhangsan.ppk）与公有文件（用户名为zhangsan则保存文件名为zhangsan.pub）存放在自己机器的某个目录下，该目录需要记牢。

5.4.2 利用gitbash生成公钥与密钥对

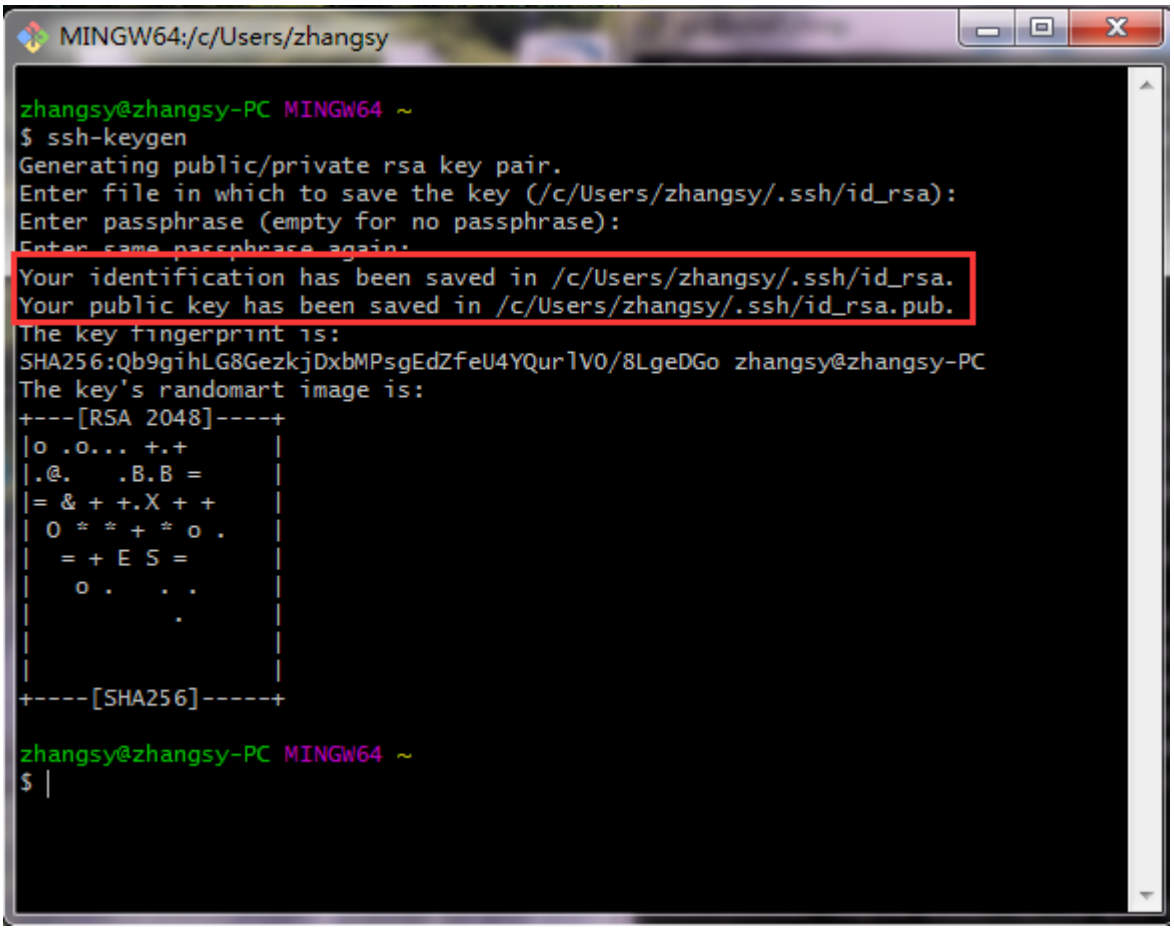
前面安装了MSysGit 包，即git for windows，在程序中就有了Git Bash。



生成SSH私钥与公钥步骤如下：

步骤1：进入Bit Bash命令行，执行ssh-keygen -t rsa，存储位置为默认，password为访问密钥所需的密码，请设置为常用密码避免遗忘。

步骤2：将生成私钥文件id_rsa与公钥文件id_rsa.pub。如果用户名是zhangsan，则修改文件名为zhangsan与zhangsan.pub。【备注：这里请根据自己的用户名来实际修改】



这里的文件名id_rsa.pub可以随意修改文件名称，例如id_zhangsy_rsa.pub等。

当产生公钥文件zhangsan.pub后，通过邮件发送给Git管理员张思宇。

5.5 初始化项目仓库

当需要新增一个项目时，可参照本章节内容来操作。

5.5.1 一些基础知识

Git使用4种协议：本地传输，SSH 协议，Git 协议和HTTP 协议。由于SSH是唯一一个同时便于读和写操作的网络协议，因此采用SSH协议。而管理SSH连接需要统一通过Gitosis来管理。

这里以CentOS充当服务器为例给大家讲解一下Gitosis的安装和配置。

同时，你还需要另外一台PC，当然Linux、Mac主机也是可以的。之所以要两台机器，是因为一台做服务器，另外一台充当客户端。

需要在服务端建立一个账户"git", 它被用于所有开发人员的Git访问。gitosis管理git虽然原理是通过ssh连接git服务器的，但是它管理的用户并不需要在服务器上存在实际的linux用户，客户端通过git用户和个人私钥访问git服务器。这样的好处有2点：

好处1：只有拥有个人私钥的用户才能访问git。

好处2：git用户的密码也不需要透露给客户端用户。

客户端生成个人的公钥通过gitosis-admin上传到gitosis中后，gitosis会做自动一个公钥转发。可以查看/home/git/.ssh/authorized_keys文件(同时将公钥存一份到gitosis-export/keydir目录)，该文件中保存了每个用户的公钥信息。

5.5.2 使用Gitosis管理Git服务器和项目

【步骤1-服务端-安装Gitosis】

Gitosis是用Python编写的，所以，在初始化一些环境的时候，要特别注意。通过下面的命令安装所有需要的软件。

Step1：yum install -y python python-setuptools git-core

Step2：git clone git://github.com/res0nat0r/gitosis.git

Step3：cd gitosis

Step4：python setup.py install

【步骤2-客户端-生成Gitosis管理账号git的公钥】需要在客户端Linux上生成管理账号git的ssh公钥与密钥来初始化服务端的gitosis。

1 ssh-keygen -t rsa

2 将客户端的公钥文件通过scp上传到服务端：scp ~/.ssh/id_rsa.pub 服务端root@xxx:/tmp/

则在服务器的/tmp/目录下生成git.pub文件。

【步骤3-服务端-初始化Gitosis】下面就利用刚才git账户的公钥初始化gitosis

1 #新建用户git与组git

useradd -m git

#passwd git // 修改git用户的密码为git

#chown git:git /tmp/git.pub

2 su - git

3 gitosis-init </tmp/git.pub

[git@server1 home]$ gitosis-init < /tmp/git.pub

Initialized empty Git repository in /home/git/repositories/gitosis-admin.git/

Reinitialized existing Git repository in /home/git/repositories/gitosis-admin.git/

这里创建了gitosis-admin.git仓库。

初始化会在/home/git下面生成gitosis和repositories两个目录，后者即为存放公共库的目录。

确保/home/git/repositories/gitosis-admin.git/hooks/post-update具有执行权限（755）。

```
[root@server1 git]# pwd
/home/git
[root@server1 git]# ls -l
total 8
drwxr-xr-x. 2 git git 4096 Aug 18 10:15 gitosis
drwxr-xr-x. 8 git git 4096 Aug 18 10:30 repositories
[root@server1 git]#
```

这样该公钥的拥有者git就能修改包含着Gitosis设置的那个Git仓库了。

【步骤4-客户端PC-Gitosis管理员git-克隆gitosis-admin.git项目】

1. 在git用户的Linux机器上克隆gitosis-admin.git项目，它是用来管理所有git项目及用户的。

url: git clone git@192.168.1.160:/home/git/repositories/gitosis-admin.git

url: git clone git@192.168.1.160:gitosis-admin.git

然后本地可以clone成功gitosis-admin.git库。

在gitosis-admin项目下有一个gitosis.conf文件和一个gitosis-export/keydir目录。

gitosis.conf用来配置git项目和用户，keydir存放用户的公钥，这里的公钥对命名有严格要求，要是用户名.pub，必须以.pub后缀结尾。

gitosis.conf的格式如下：

[gitosis]

[group gitosis-admin]

writable = gitosis-admin

members = git #对应keydir下有一个git管理员git.pub 公钥文件

【步骤5-客户端PC-Git管理员git-增加开发人员与项目权限设置】

接下来git在自己客户端Linux机器上添加其他开发人员公钥与访问权限：

每个用户的公钥文件名格式为: 用户名.pub，因此，lirh的公钥文件名为lirh.pub, jiatian的公钥文件名为jiatian.pub。

公钥内容或者私钥内容采用不同方法生成时可能有不同，例如lirh采用gitbash生成的公钥文件中可能包含john@john-pc.

但zhangsy采用puttygen.exe生成的公钥文件中没有任何与用户相关的信息，其私钥中包含一个Private-MAC: 7c306619c4b811a7ba2a9915a41c3f95741a4139

说明：每台机器上生成的公钥内容或者私钥内容与用户名无关, 只要在每台机器上生成一个私钥与公钥文件，然后改公钥文件命名为"<用户名>.pub"，

当管理员git收到大家传过来的pub文件后，git管理员把用户lirh的公钥复制一份修改成lirh.pub，放入e:\code\gitosis-admin\keydir\路径下, 然后commit, push。

$ git commit -am "add 添加用户lirh公钥"

$ git push

并修改gitosis.conf文件,

[gitosis]

[group gitosis-admin]

writable = gitosis-admin

members = git #对应keydir下有一个 git.pub 公钥文件

// 增加部分：将开发人员分组, 可针对不同仓库分配读写权限

[group develop]

writable = testProject // 这个对应服务器上的testProject.git仓库的写权限

或readable = cloud1pei_deploy

members = lirh zhangsy jiatian zhouyy renzj #多个用户用空格分开

然后commit, push.

【步骤6-客户端PC-开发人员-上传新项目的代码仓库】

gitosis管理员git或其他开发人员现在就可以将本地新建立的仓库push到服务器上。

此时在本地win7/Linux机器上建立一个空仓库testProject并push到服务器上，例如先在e:\code目录下新建一个空目录testProject, 然后点击git create repository here..., 则e:\code\testProject目录下新建立.git子目录。

然后在e:\code\testProject目录下新建立readme.txt文件，add, commit.

也可以参考https://www.cnblogs.com/lidabo/p/7457998.html的图形流程来操作。

但在push到server git@192.168.1.160:/home/git/repositories/testProject.git一直报告以下错误：

fatal: '/home/git/repositories/testProject.git' does not appear to be a git repository

fatal: Could not read from remote repository.

Please make sure you have the correct access rights and the repository exists.

git did not exit cleanly (exit code 128) (5787 ms @ 2015/3/5 8:58:02)

参考https://wiki.archlinux.org/index.php/Gitosis，需要手动先在server上以git用户身份建立这样一个裸仓库：

```
# su git
$ git init --bare /home/git/repositories/testProject.git
```

然后重试上面过程就可以了。

注意：如果在server上是以root身份建立的这个裸仓库，则可能会报告上述fatal: Could not read from remote repository.错误。

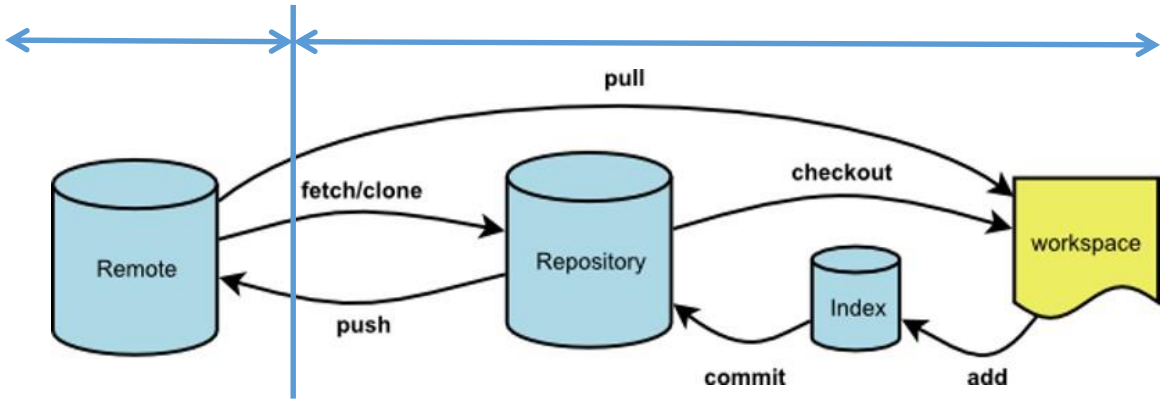
此时可执行下面操作更改own与group都是git:

```
root@stu-system:/home/git/repositories# chown -R git testProject.git
root@stu-system:/home/git/repositories# chgrp -R git testProject.git
```

【步骤7-客户端PC-开发人员-下载testProject项目的代码仓库testProject.git】

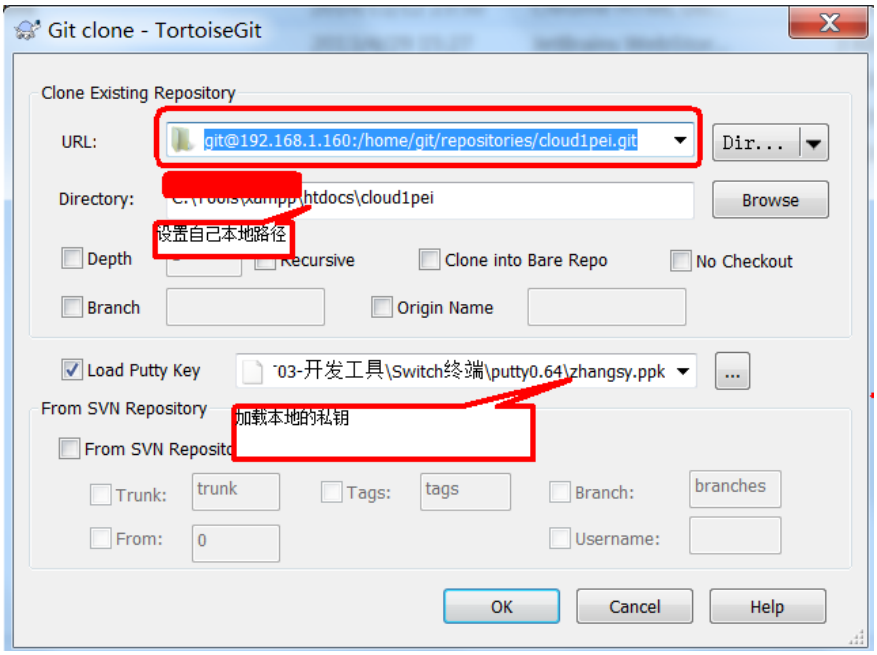
开发人员lirh使用git@192.168.1.160/home/git/repositories/testProject.git后就可以clone成功仓库。

5.6 Git客户端常用操作

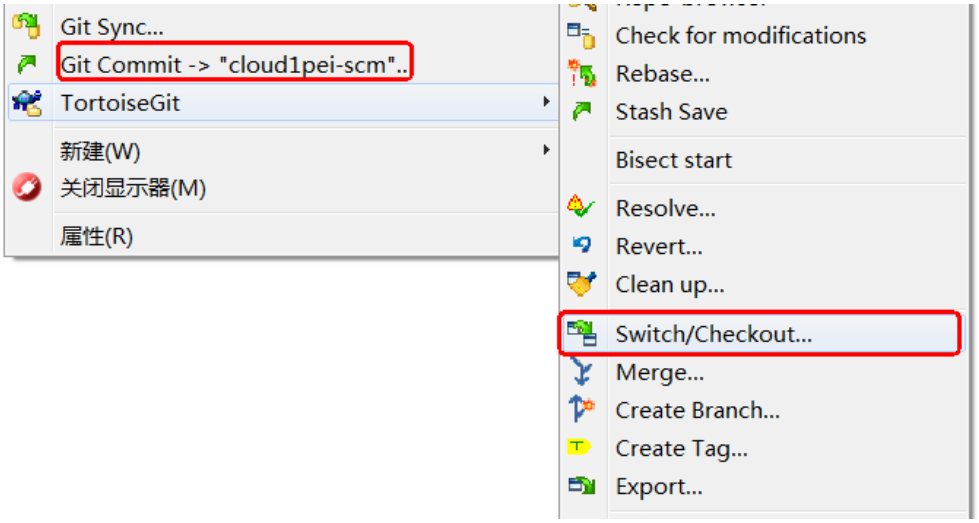


远端Git仓库　　本地Git仓库　　本地工作空间

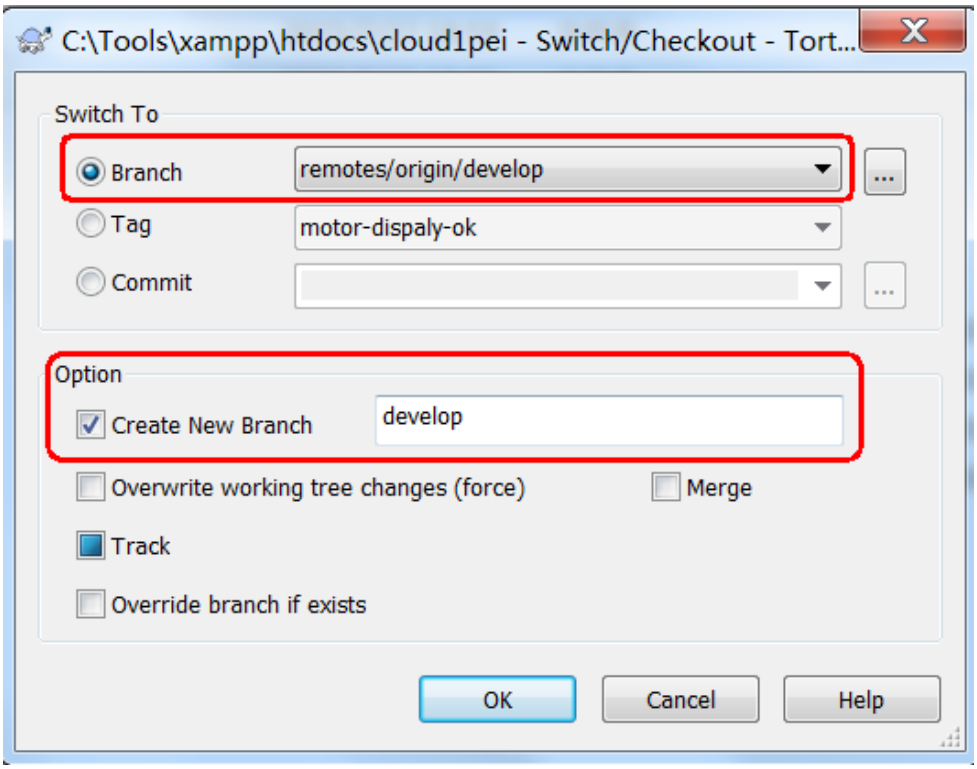
开发人员在收到Git管理员配置成功的反馈邮件后，切换到工作目录（例如c:\xampp\htdocs目录）下，在git Shell下执行git clone git@192.168.1.160:cloud1pei.git或者在tortoiseGit中 Git Clone中设置访问git@192.168.1.160:/home/git/repositories/cloud1pei.git（见下图）：



即可从服务器获取到cloud1pei代码的master分支并自动在本地建立cloud1pei目录。然后cd cloud1pei进入该子目录下后通过git checkout develop命令，即可将本地代码从master分支切换到develop分支，或者在代码的根目录下点击鼠标右键，出现下图，选择“Switch/Checkout”菜单项。



然后，选择切换到remotes/origin/develop分支，并选中“Create New Branch” 就可以在本地创建develop分支，然后就可以在develop分支上做开发与测试了。



注意：

1. **新员工前5次代码commit以及push操作需要由富有经验的员工来把关和指导，之后才能由新员工自己来操作。**
2. **代码需要在本地测试通过才可以做push操作。**

常用的git操作：

git clone：从远程服务器上取仓库到本地

git checkout：切换到某一个分支

git add：添加一个文件到本地仓库

git commit：增加或修改文件后，通过commit操作提交活动到本地仓库

git push：将本地修改推送到远程服务器上。

git pull：从远端服务器上获取最新代码

Git 之所以能够提供方便的本地分支等特性，是与它的文件存储机制有关的。Git存储版本控制信息时使用它自己定义的一套文件系统存储机制，在代码根目录下有一个.git文件夹，会有如下这样的目录结构：

```
COMMIT_EDITMSG  HEAD            branches/       description     index
logs/           refs/
FETCH_HEAD      ORIG_HEAD       config          hooks/          info/
objects/
```

有几个比较重要的文件和目录需要解释一下：

HEAD文件存放根节点的信息，其实目录结构就表示一个树型结构，Git采用这种树形结构来存储版本信息，那么HEAD就表示根；

refs目录存储了你在当前版本控制目录下的各种不同引用（引用指的是你本地和远程所用到的各个树分支的信息），它有heads、remotes、stash、tags四个子目录，分别存储对不同的根、远程版本库、Git栈和标签的四种引用，你可以通过命令'git show-ref'更清晰地查看引用信息；

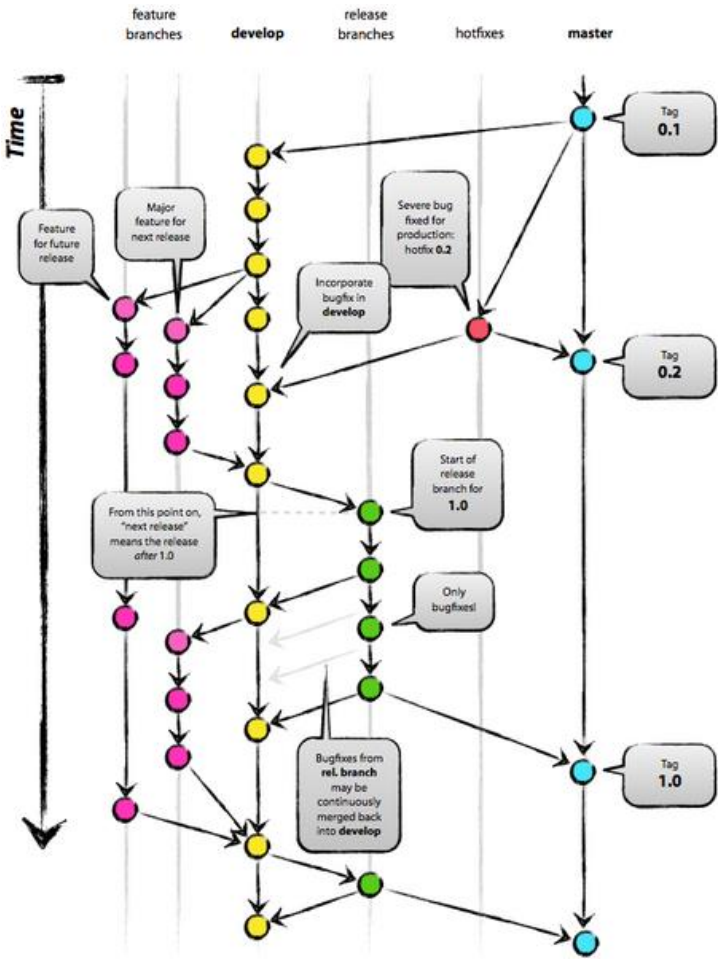
logs目录根据不同的引用存储了日志信息。

因此，Git只需要代码根目录下的这一个.git目录就可以记录完整的版本控制信息，而不是像SVN那样根目录和子目录下都有.svn目录。

5.6.1 常用操作—分支创建与维护（经典理论）

下面来自http://nvie.com/posts/a-successful-git-branching-model/的中文翻译链接 http://segmentfault.com/a/1190000000434973。

在实际工作中，通常除了master与develop, release分支以外，经常需要2-3人在短时间内开发一个小的新特性，这时可以考虑在当前开发分支上拉出一个feature1_XXX、feature2_XXX分支，而且需要考虑到今后的分支的维护与销毁。



5.6.1.1 主要分支

中央仓库中有两个长期的分支：master 和 develop。

`master` 用作生产分支，里面的代码是准备部署到生产环境的。

`develop` 是可交付的开发代码，也可以看成是用于集成分支，每晚构建从 `develop` 获取代码。

当 `develop` 分支中的代码足够稳定的时候，就将改动合并到 `master` 分支，同时打上一个标签，标签的名称为发布的版本号。

5.6.1.2 辅助分支

通过辅助分支来帮助并行开发，和主要分支不同，这些分支的生命周期是有限的：

- 特性 feature 分支
- 发布 release 分支
- 紧急修复 hotfix 分支

5.6.1.3 特性分支

特性分支可能从 `develop` 分支分出，最终**必须**合并回 `develop`。

特性分支（也叫主题分支）用于开发新特性。每个新特性开一个新分支，最终会合并回 `develop`（当特性开发完毕的时候），或者放弃（如果最终决定不开发这个特性）。

特性分支只存在于开发者的仓库中。

5.6.1.3.1 创建一个特性分支

从 `develop` 分支分出：

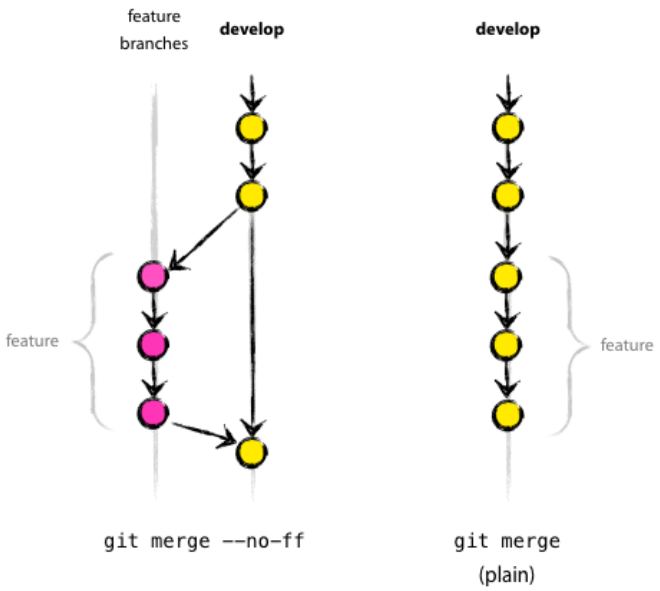
```
$ git checkout -b myfeature develop
```

5.6.1.3.2 合并回`develop`

完成的特性需要合并回 `develop`：

```
$ git checkout develop

$ git merge --no-ff myfeature

$ git branch -d myfeature

$ git push origin develop
```

使用`--no-ff`时，将特性分支上所有的修改 Commits 汇总起来，新生成一个提交 Commit 统一提交到 develop 分支上，这样就能够方便地看出在 develop 分支上 Merge Commit 对应哪一个特性。比较：



5.6.1.4 发布分支

发布分支可能从`develop`分出，最终**必须**合并回`develop`和`master`。发布分支以`release-*`的方式命名。

发布分支为新的发布版本作准备，包括一些小 bug 的修正和发布的元信息（版本号、发布日期等）的添加。这样`develop`分支就可以接受针对以后的发布的新特性。

在代码基本可以发布的时候从`develop`分支分出发布分支。这时要确保此次发布包括的特性都已经合并到`develop`分支了（同时，为下一版发布准备的特性不能合并到`develop`分支，必须等待发布分支分出后才能合并）。

5.6.1.4.1 创建发布分支

```
$ git checkout -b release-1.2 develop

$ ./bump-version.sh 1.2

$ git commit -a -m "Bumped version number to 1.2"
```

`bump-version.sh`是一个脚本，修改相应文件的信息，以体现版本号已经改变了。

5.6.1.4.2 完成发布分支

当发布分支中的代码可以发布的时候，将代码合并到`master`分支，并打上相应的标签。同时还需要合并到`develop`分支，因为发布分支里可能包含一些修正 bug 的代码，合并回去可以确保以后的版本也包含这些修正。

```
$ git checkout master

$ git merge --no-ff release-1.2

$ git tag -a 1.2



$ git checkout develop

$ git merge --no-ff release-1.2
```

注意，合并回 `develop` 分支很可能导致合并冲突，我们需要手工修复一下，然后提交。之后可以删除发布分支：

```
$ git branch -d release-1.2
```

5.6.1.5 紧急修复分支

可能从 `master` 分出，**必须**合并回 `develop` 和 `master`。分支名以 `hotfix-*`开头。

紧急修复分支和发布分支很像，只不过它们是意料之外的。如果生产系统里有一个紧急的 bug，必须马上修复的话，我们就从 `master` 里分出一个紧急修复分支。

这样，某个人修复紧急 bug 的同时，团队其他成员可以继续在 `develop` 分支上开发。

5.6.1.5.1 创建紧急修复分支

```
$ git checkout -b hotfix-1.2.1 master

$ ./bump-version.sh 1.2.1

$ git commit -a -m "Bumped version number to 1.2.1"
```

修复 bug 并提交

```
$ git commit -m "Fixed severe production problem"
```

5.6.1.5.2 完成紧急修复分支

修复 bug 之后，需要合并回 `master`，同时也需要合并回 `develop`。

```
$ git checkout master

$ git merge --no-ff hotfix-1.2.1

$ git tag -a 1.2.1
```

```
$ git checkout develop

$ git merge --no-ff hotfix-1.2.1
```

以上情况假定不存在发布分支。假设存在发布分支的话，代码不应该合并回 `develop`，而应该合并回发布分支，确保正在准备的发布分支也能收到这个补丁（由于发布分支最终会合并到 `develop`，因此这时不用再另外合并到 `develop`）。

最后，删除这个紧急修复分支：

```
$ git branch -d hotfix-1.2.1
```

5.6.2 常用操作—分支创建与维护（TortoiseGit实战）

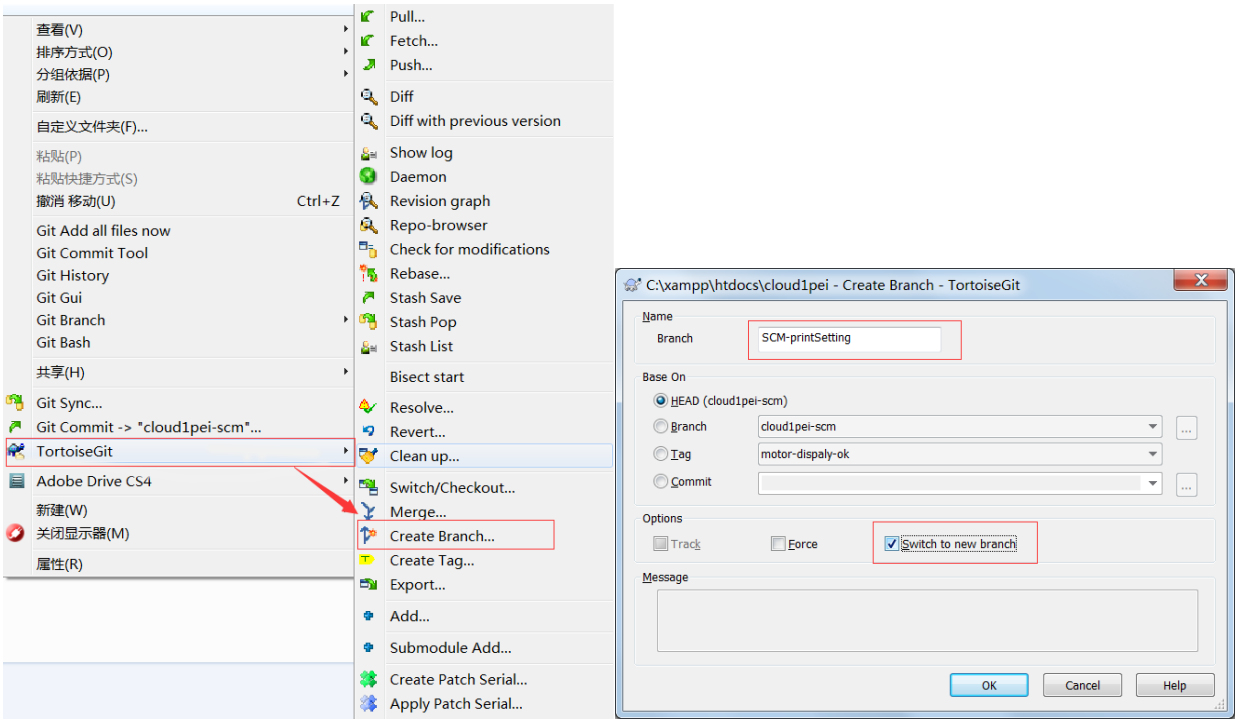
假定团队成员一开始都在develop分支上工作，后来有个新特性需要由2-3人在几天时间内完成，或出现一个紧急严重BUG需要在几天内完成修复，则首先由团队中成员甲在本地创建新特性分支或修复BUG分支（参见5.6.2.1小节），然后将本地分支推送到远端服务器中（参见5.6.2.2小节），则团队中成员乙、丙可以切换到这个新分支开始工作（参见5.6.2.3小节）。

5.6.2.1 创建本地特性分支或修复BUG分支

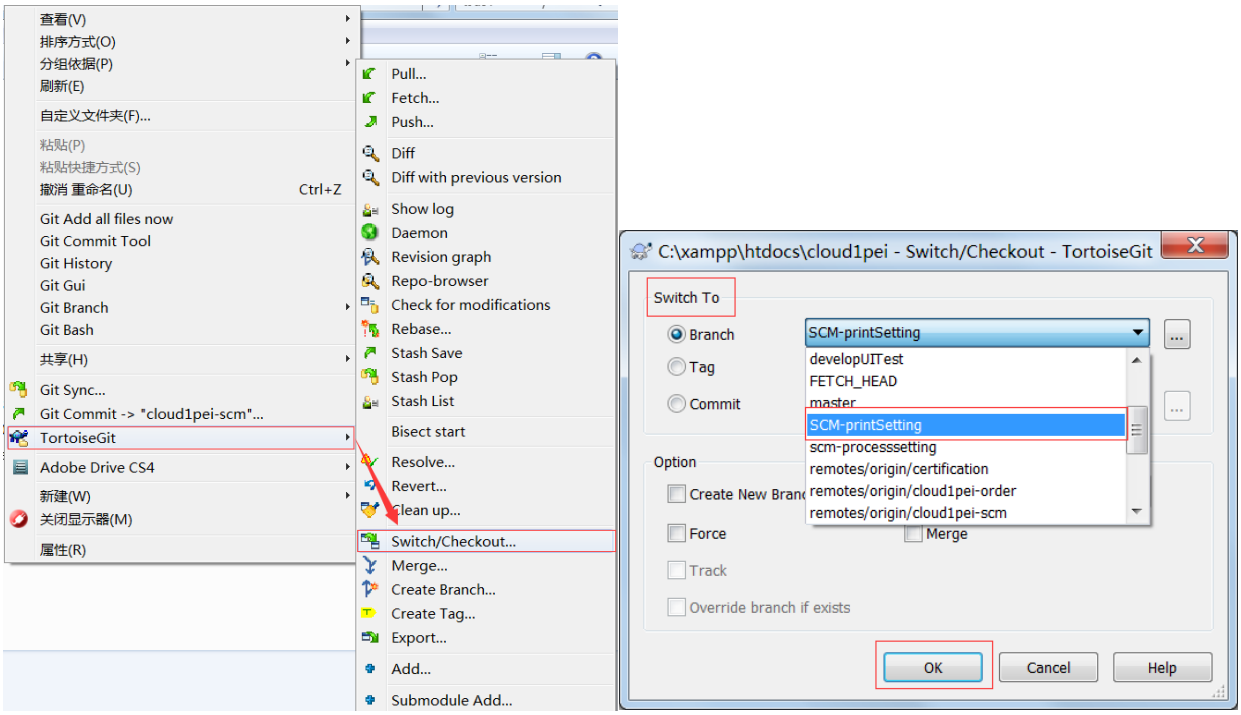
本地创建特性分支或修复BUG分支分为两个步骤。

步骤1：创建本地特性分支或修复BUG分支

点击右键选择TortoiseGit，选择Create Branch…，在Branch框中填写新分支的名称“SCM-printSetting”，若选中“Switch to new branch”则直接转到新分支上，省去**步骤2**，点击OK按钮：

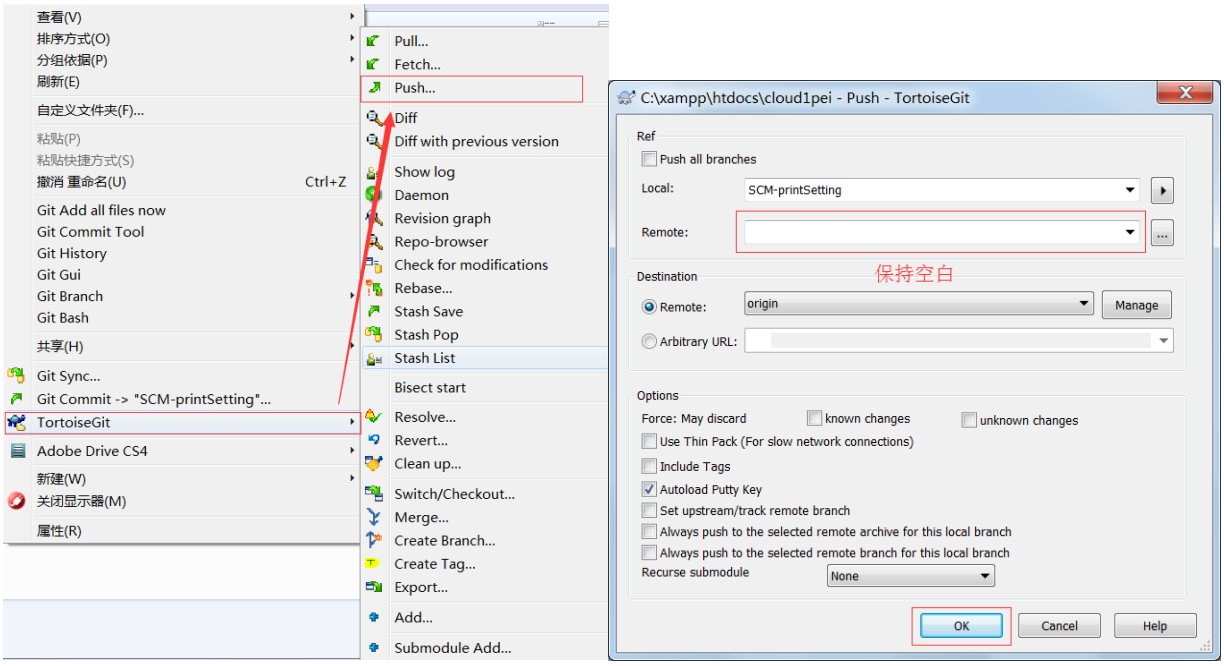


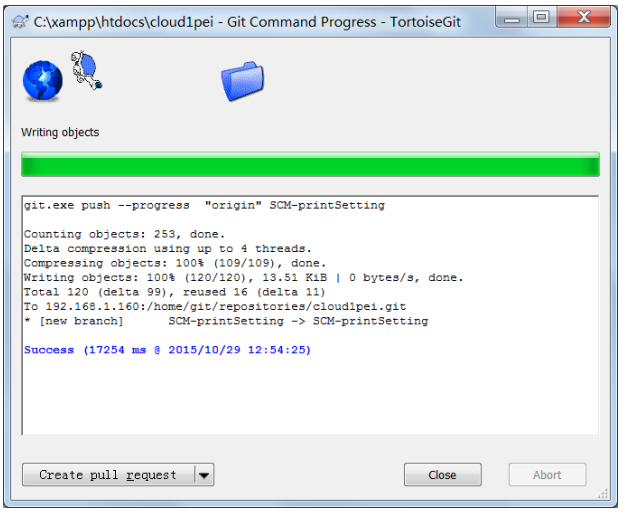
步骤2：通过“Switch/Checkout”切换到新创建的分支“SCM-printSetting”，点击OK：



5.6.2.2 将本地分支推送到远端服务器中

在新分支下执行Push操作，在对话框中保持远程分支Remote为空白，点击OK，则将在远程创建了新的分支（在Push的时候远程服务器发现远程没有该分支，此时会自动创建一个和本地分支名称一样的分支，并将本地分支的内容上传到该分支）。





5.6.2.3 其他成员切换到该新分支后并行开发

团队中其他成员在本地develop分支上做Pull操作，这样这些成员本地就可以看到该新分支，然后切换到新分支“SCM-printSetting”（过程同5.6.2.1小节的步骤2），开始开发工作。

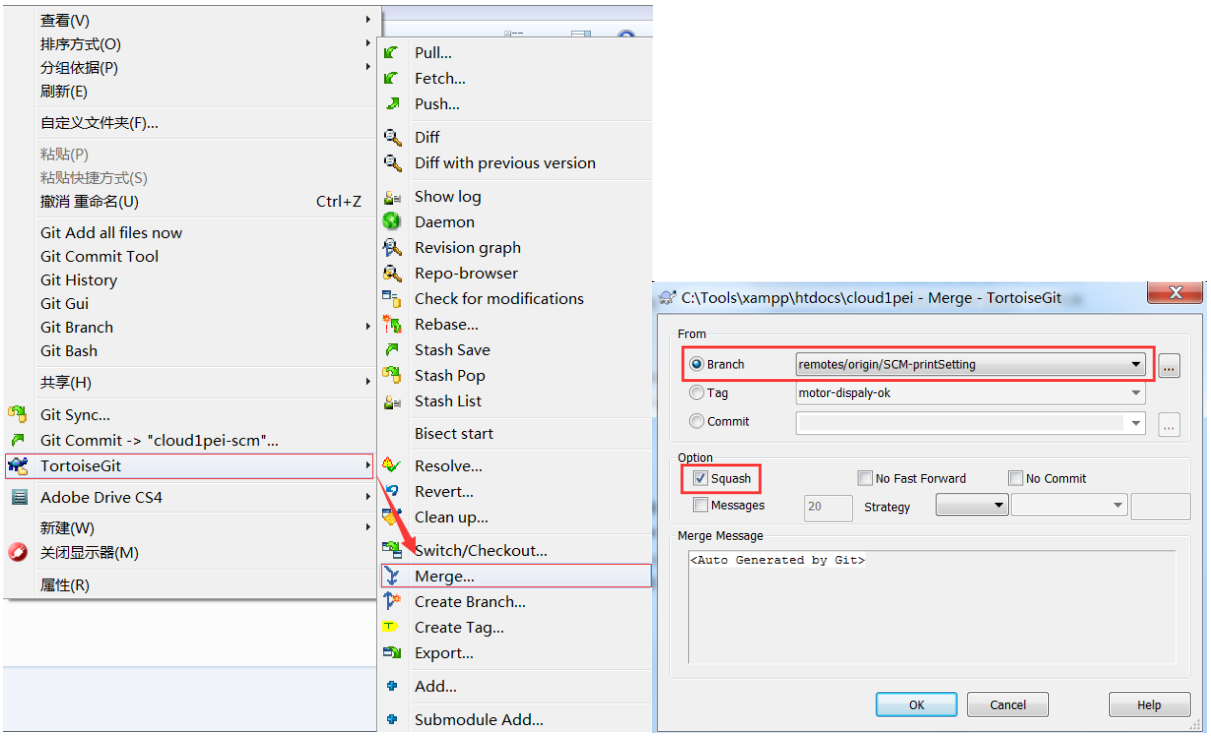
5.6.2.4 开发完成后分支合并

分支合并之前我们需要明确哪个分支将要合并到哪个分支，

- 首先通过“Switch/CheckOut”切换到本地主干分支（如develop分支），
- 然后通过“Merge…”进行合并操作，在对话框中选择需要合并的本地分支，将选项Squash打上勾，并填写Merge Message内容后按OK键执行Merge操作。

备注：git squash就是整理压缩的意思，可用来一个临时分支流上琐碎的多个commit结合起来形成一个完整的commit。

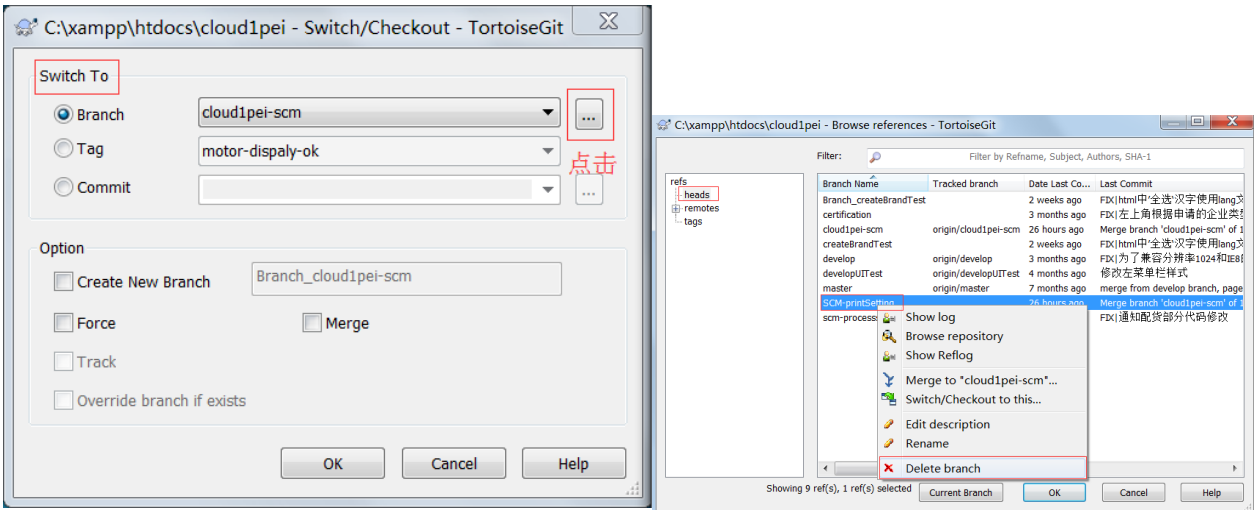
- Merge过程中可能会出现文件冲突，此时需要修订冲突文件，然后commit操作上载修改到本地Git。
- 分支合并成功后，我们就可以通过Push操作将合并上传到中心服务器。



5.6.2.5 删除分支

当我们已将新分支合并到主分支后，如果该分支不再作为维护分支，或者是个临时小组开发分支而需要放弃该分支的时候，可以对该分支进行删除操作。

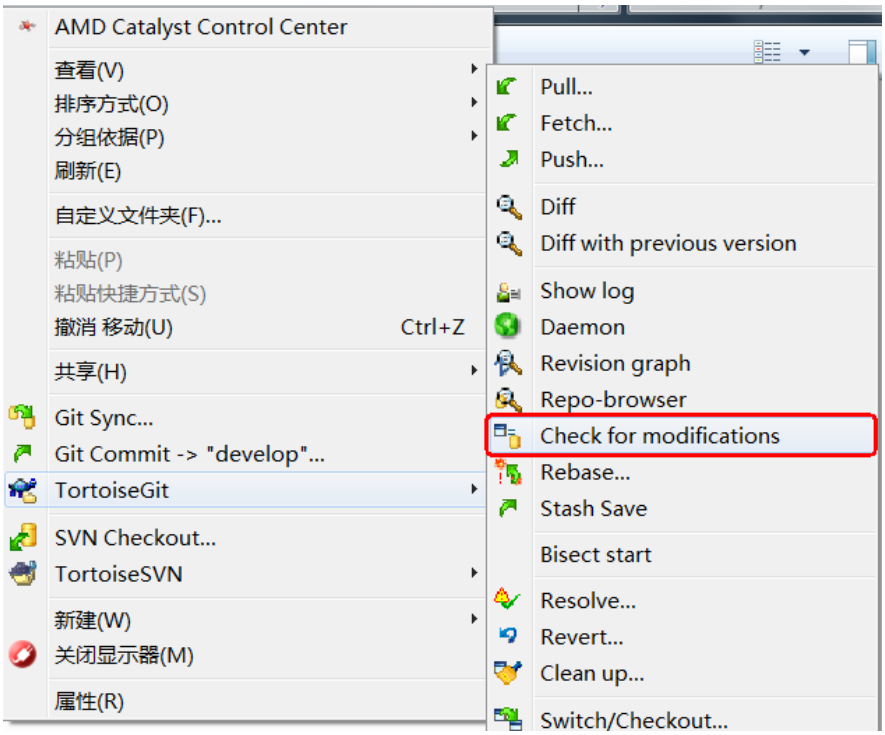
首先通过“CheckOut/Switch”打开对话框，点击Switch to区域中Branch条目后面的更多按钮，打开分支列表对话框，右键点击要删除的分支，选择Delete branch进行删除。



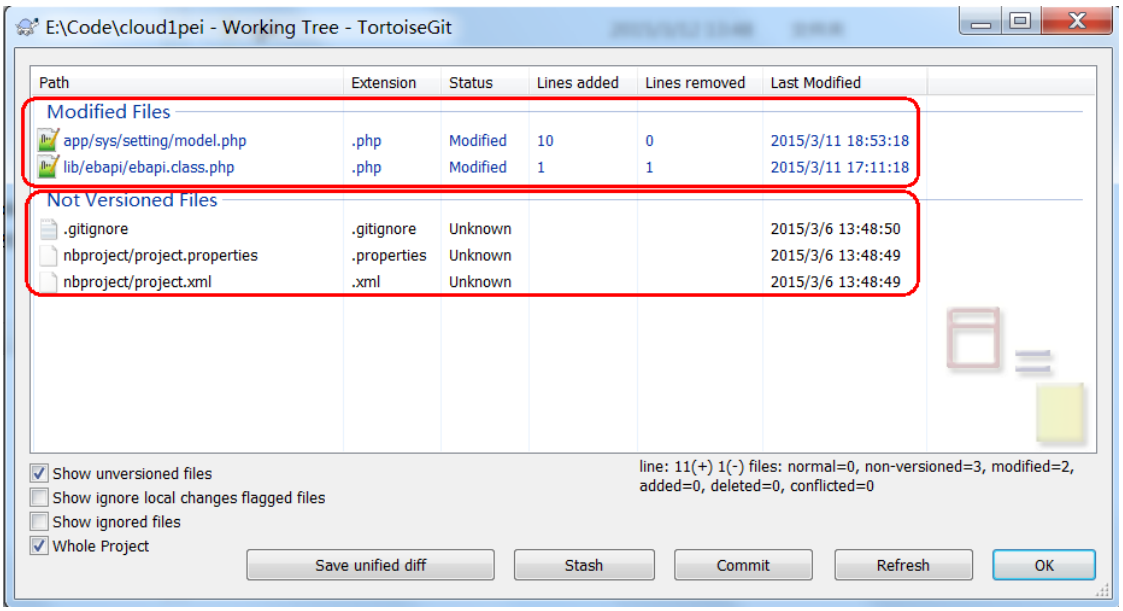
注意，在删除远程分支的时候，本地分支并不会删除，这也说明了本地分支与远程分支并无从属关系。

5.6.3 常用操作—检查本地文件修改

当本地为某个任务修改多个文件或新增/删除一些文件时，可在Git本地代码仓库目录下，点击鼠标右键→Tortoise→Check for Modification来查看修改文件。



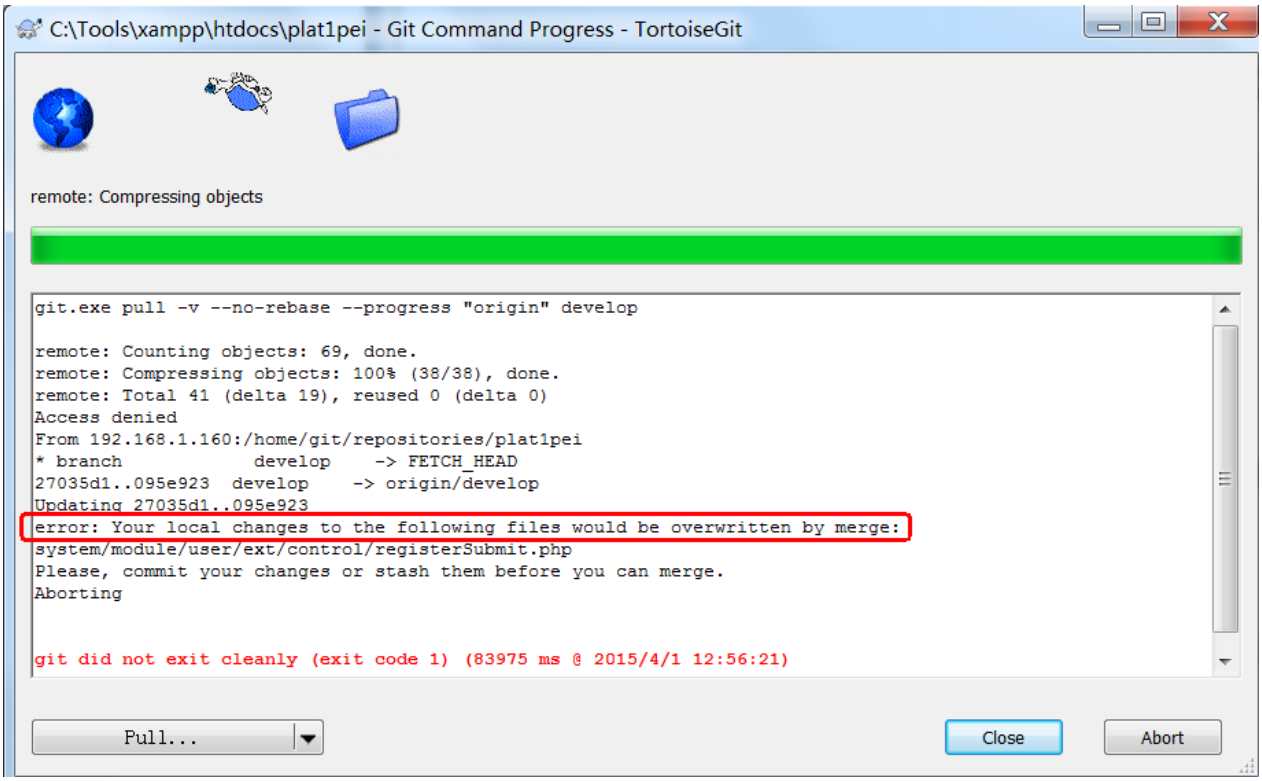
就可以查看本地有哪些文件被修改（见下面Modified Files），以及还没有放入代码仓库的文件（通常是新增文件、临时文件或.bak备份文件等）（见下面Not Versioned Files）。



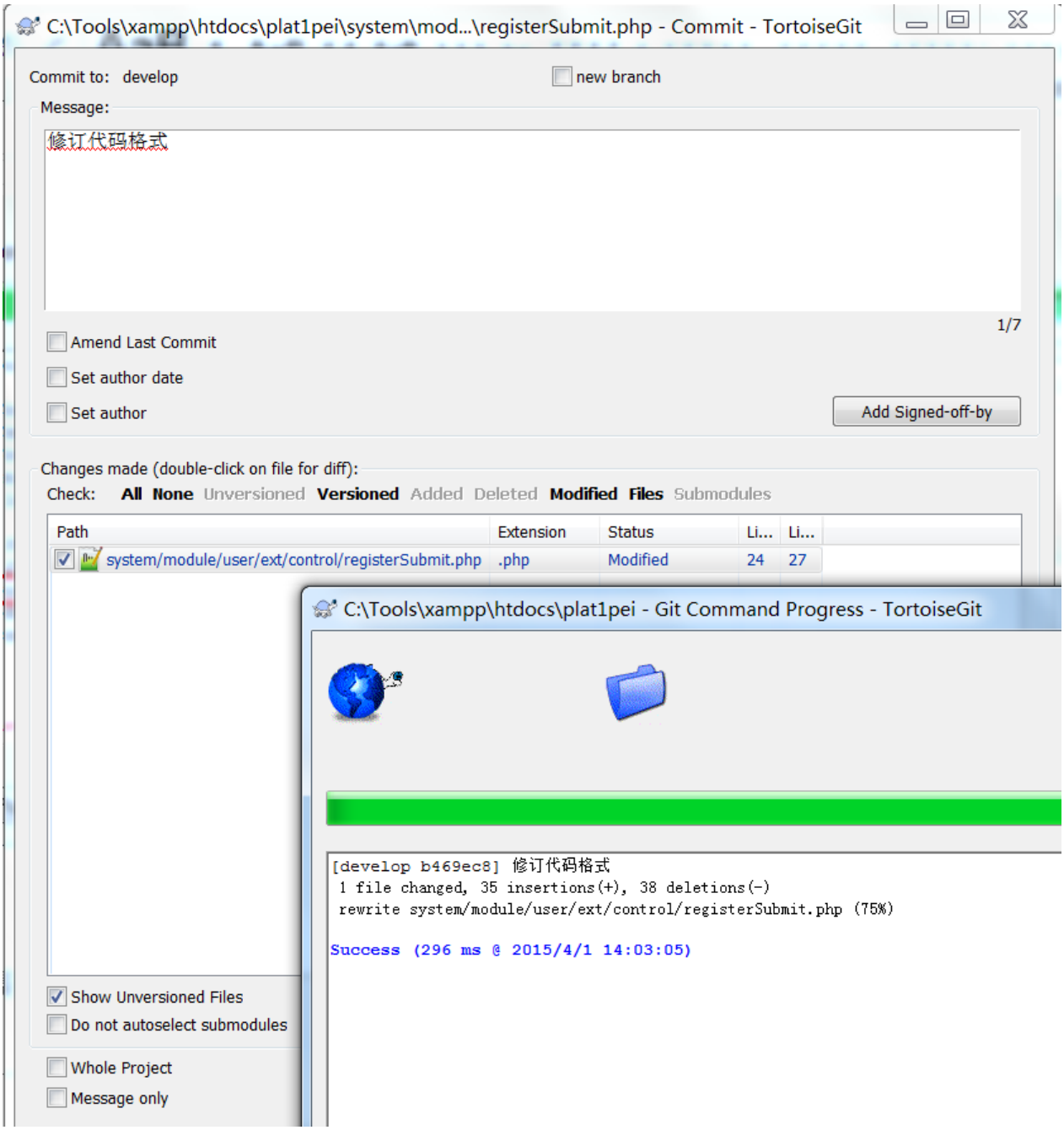
然后可双击每一个文件来查看它与本地仓库的差异部分。

5.6.4 常用操作—pull到本地工作空间时覆盖错误解决

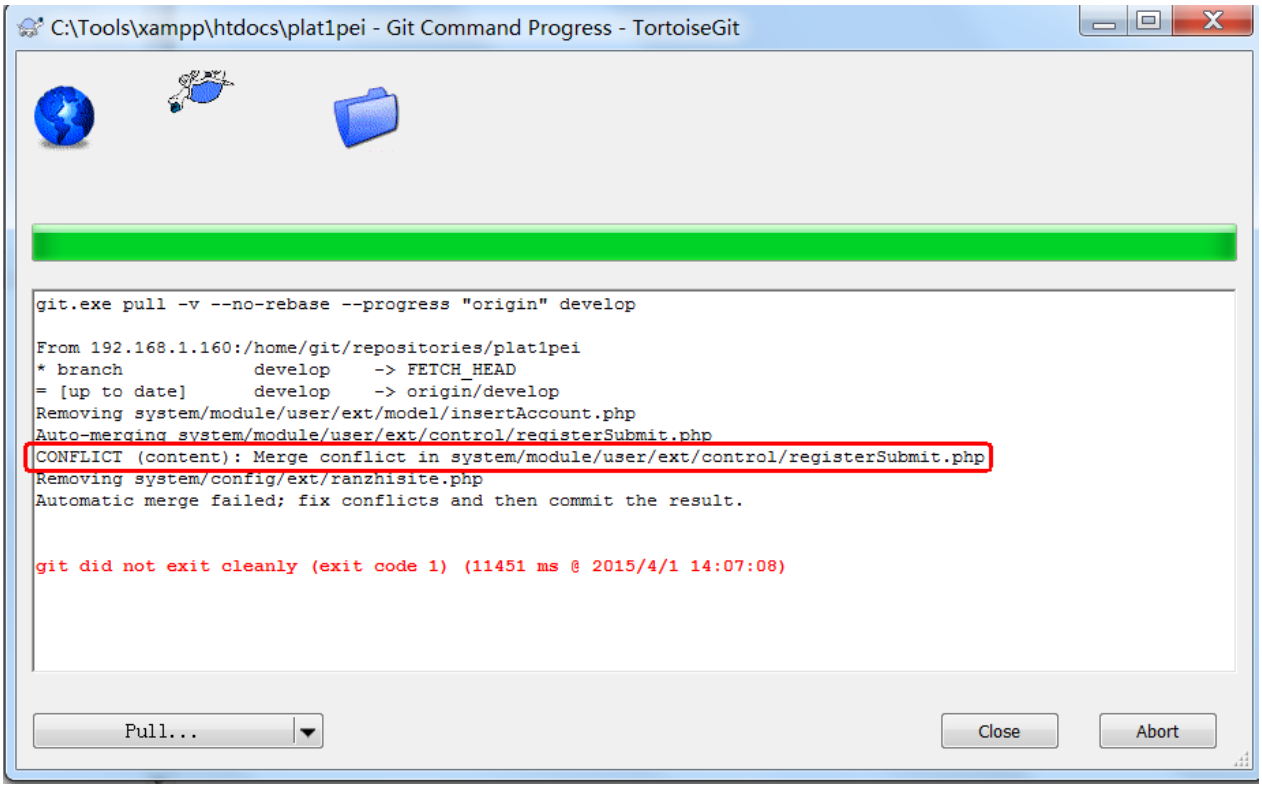
当从远端Server上pull代码到本地工作空间时，会报告本地修改可能会被merge覆盖的错误提示信息，类似于下面的截图：



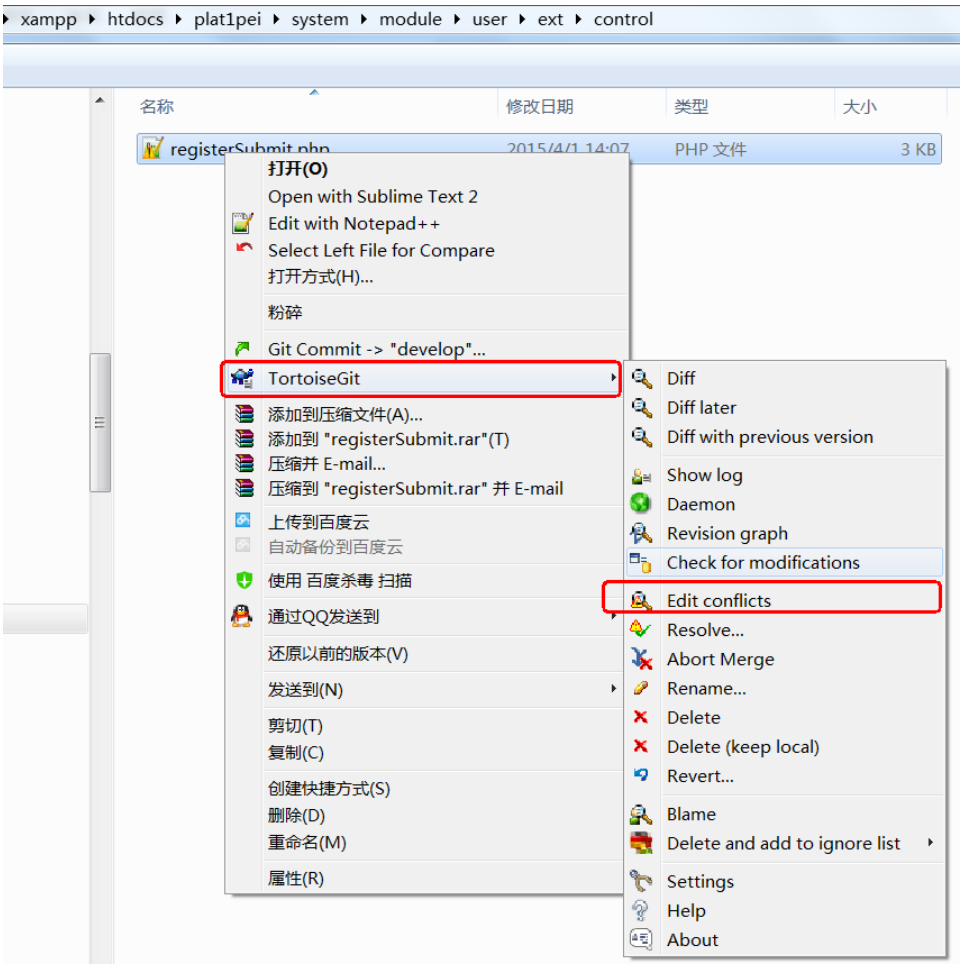
此时可以先将该文件commit到本地的git仓库，见下面截图：



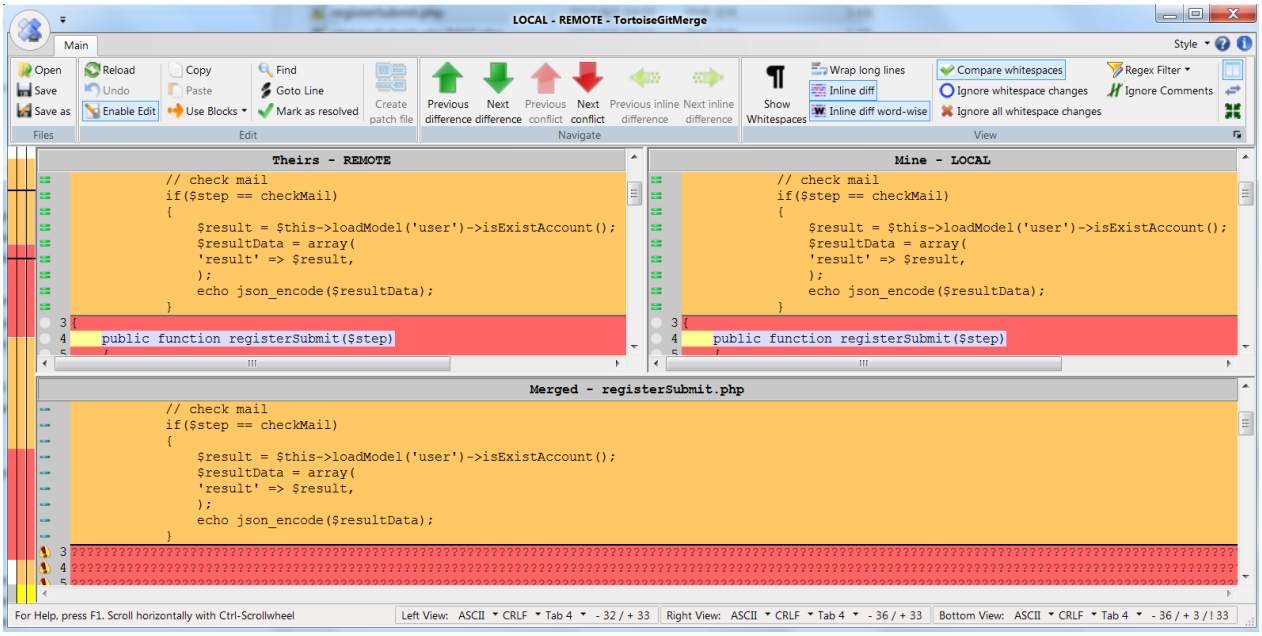
注意此时不能做Push操作，而是需要先再重新做pull操作，此时系统开始做merge操作，可能出现合并冲突问题，见下面截图。



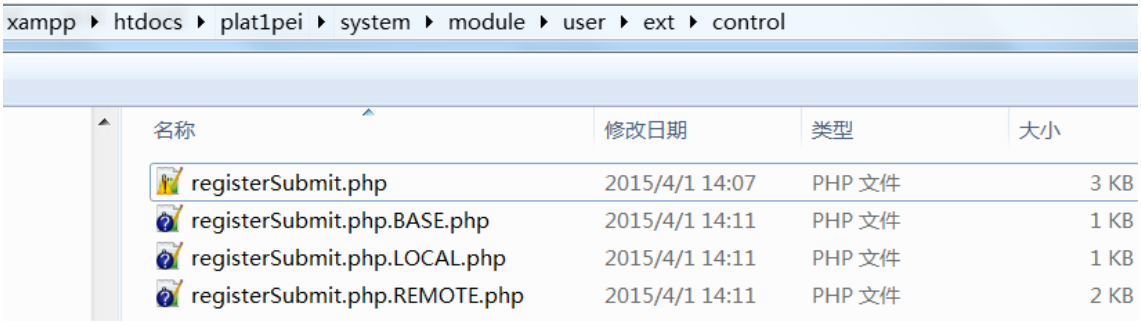
接下来解决冲突，此时先找到冲突文件，然后鼠标右键点击TortoiseGit→Edit conflicts，见下面截图：



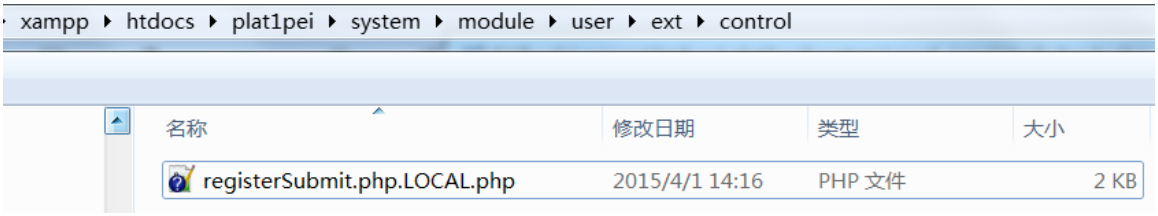
此时会自动弹出TortoiseGitMerge窗口，见下图。



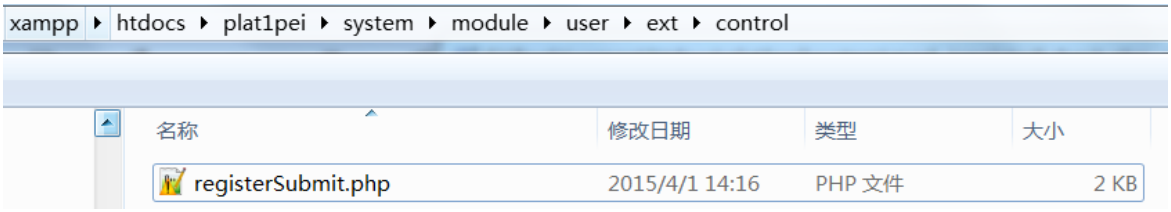
但不要关闭该窗口，直接到该文件所在的目录下解决冲突。可以看到每一个冲突文件会生成以下4个文件：



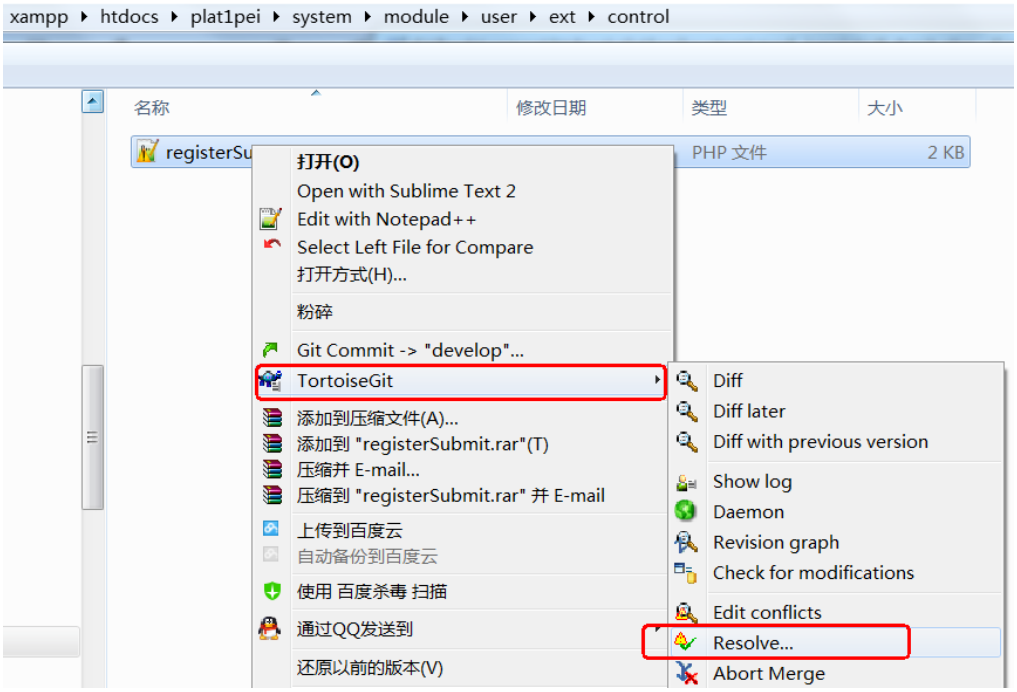
然后手动比较*.local.php与*.remote.php文件来解决冲突。**注意：修改冲突时一定不能随意私自删除他人之前的变动，需要找相关人员一起面对面讨论冲突原因及解决方案，协商解决冲突。**如果最终在将*.remote.php中他人修改的一部分内容合并到*.local.php文件后，删除其余3个文件，只保留下*.local.php文件（注意：如果对比差异后保留*.remote.php文件，则删除其余3个文件，只保留下*.remote.php文件，然后做下面类似的操作）：



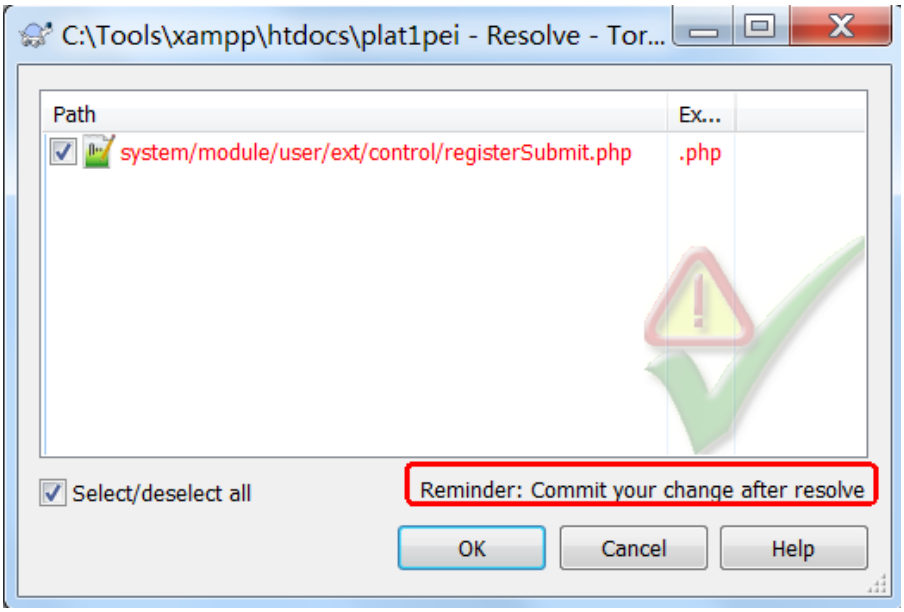
然后删除文件名中的“.local.php”，只保留下原始文件名：



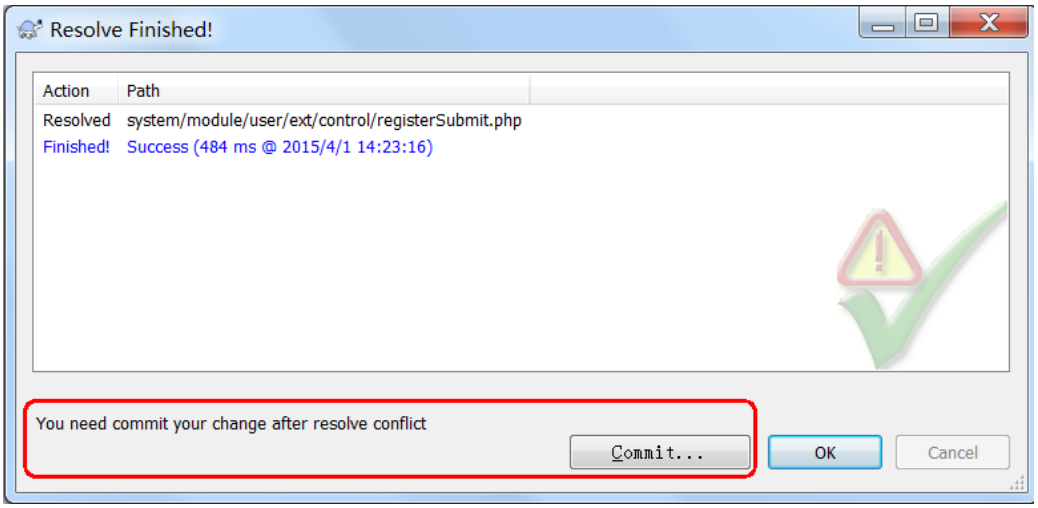
然后鼠标右键点击TortoiseGit→Resolve，告知已经解决冲突。



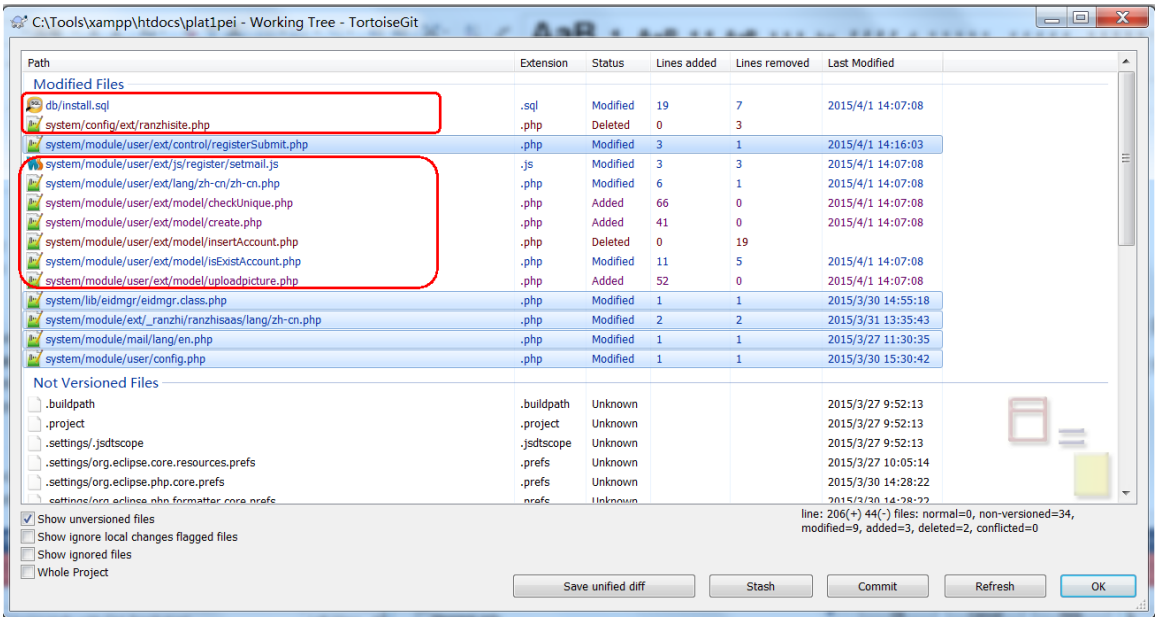
此时弹出冲突解决对话框，并提醒冲突解决后需要做commit操作。



点击OK后弹出冲突解决完成对话框，并提醒冲突解决后需要做commit操作。

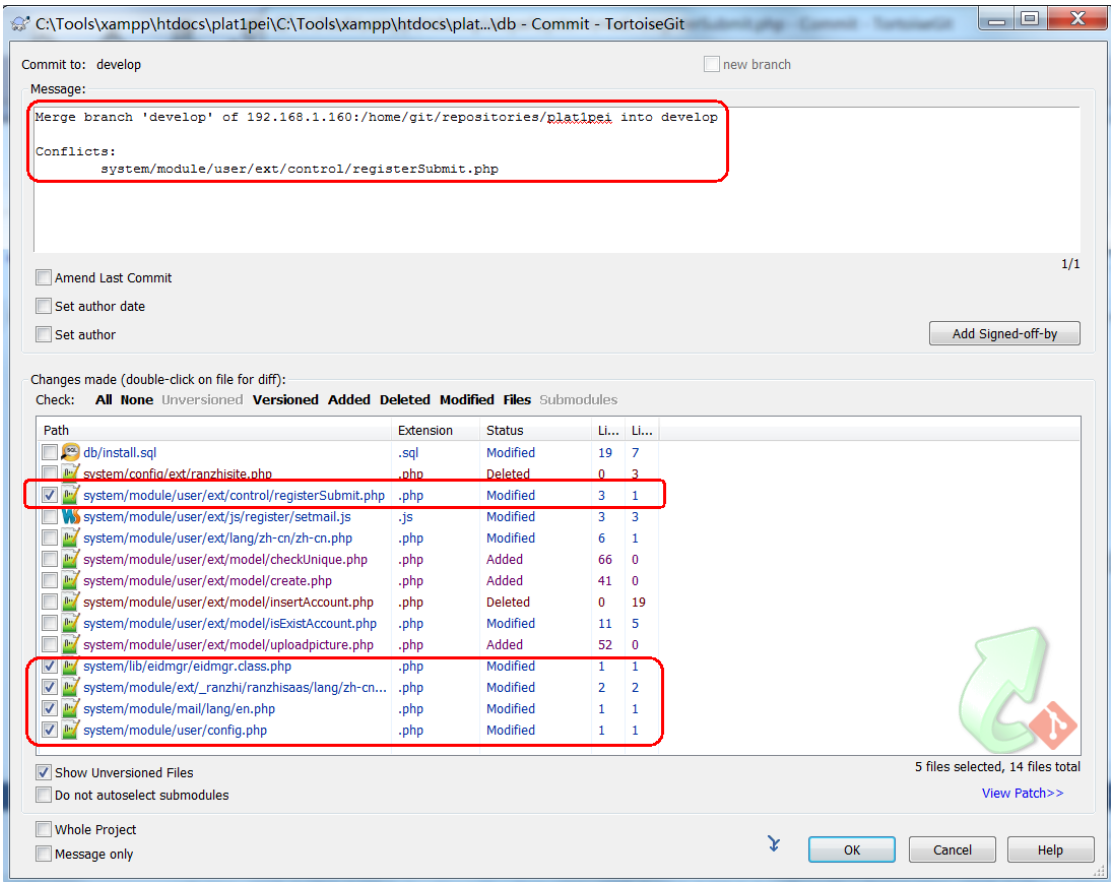


此时需要注意：在整个代码仓库根目录下点击TortoiseGit→Check for modification后，看到有许多文件被修改。这是因为前面的pull操作会将远端服务器的代码merge到本地工作空间，而这其中可能有一些其他人新修改/删除/ 增加的文件，例如下面红色框中的文件都是他人修改的文件，这些文件已经下载到本地的工作空间但还没有commit到本地的Git仓库。



现在需要将本地的工作空间中由自己修改的文件以及他人修改的文件都commit到本地的Git仓库，选中这些文件然后鼠标右键点击Commit，此时Commit message自动被填充为“Merge branch ‘develop’ of 192.168.1.160:xxx/plat1pei into develop”，这表示是将服务器Merge到本地工作空间的代码需要上载到本地的Git仓库，而且此时系统自动选择了所有文件都需要上载。

假如此时如果对他人的代码有疑义，也可以先不选择上载，先仅仅commit被自己修改有把握的文件。注意commit之前需要对每一个文件利用Beyond Compare仔细检查下每一行，确保基本的代码对齐没有问题。



commit到本地的Git仓库后，**注意此时千万不能点击push操作将本次修改推送到远端服务器上，否则Merge到本地工作空间的一些新增文件由于存在疑义没有Commit到本地Git库，此时Push是将本地Git库的内容推送到服务器上，将造成服务器上这些代码又被deleted。**

只有当Merge到本地工作空间的他人新修改/删除/增加的文件已经解决疑义后，并Commit到本地Git库后，才可以做Push操作。

5.6.5 常用操作—push到远端Server时冲突解决

当本地修改一个文件后，先commit到本地分支上，然后push到远端server上时会报告冲突。此时需要先做一次pull，然后针对每一个冲突文件会生成以下4个文件：

然后手动比较*.local.php与*.remote.php文件来解决冲突。

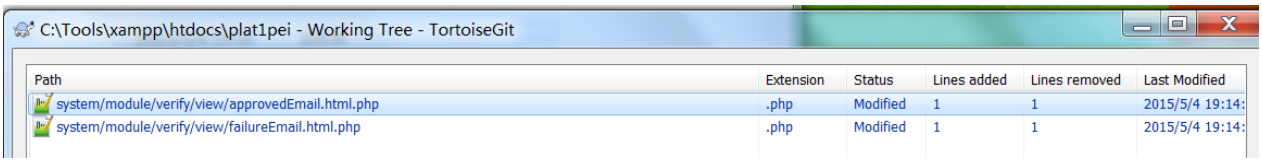
当需要解决的冲突文件比较多，建议采用beyond compare来手动解决冲突，进行不同版本代码的手动合并。

5.6.6 常用操作—解决文件名大小写问题

在一配云平台上线过程中，发现存在文件名称大小写问题，需要将扩展下针对control, model, view, language, config等所有的文件名称都使用小写，这是因为Linux对文件名大小写敏感而windows不敏感。这个在今后扩展中需要作为一项checklist。这个问题在plat1pei与cloud1pei都存在。

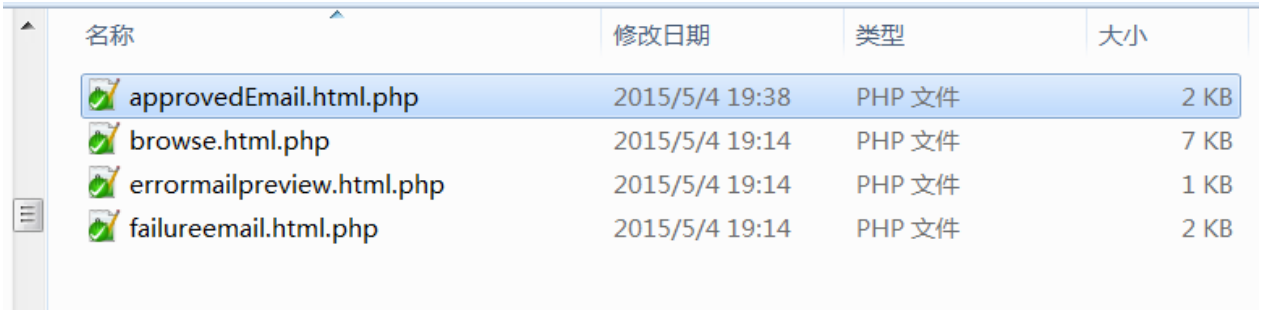
```
10:57:25 ERROR: the module user has no registersubmit method in framework/router.class.php on line 1139, last called by index.php on line 43 through function loadModule.
 in framework/router.class.php on line 1578 when visiting user-registerSubmit-checkMail
```

由于Windows与Mac操作系统下文件名大小写不敏感，造成git下如果改了名字，譬如大写统一修改为大写，推送到linux服务器的时候会没有效果，从server git上clone下新代码后，明明目录下都是小写文件名，但check for modification时可以看到许多文件名还是存在大小写混合情况且状态为modified，见下图。

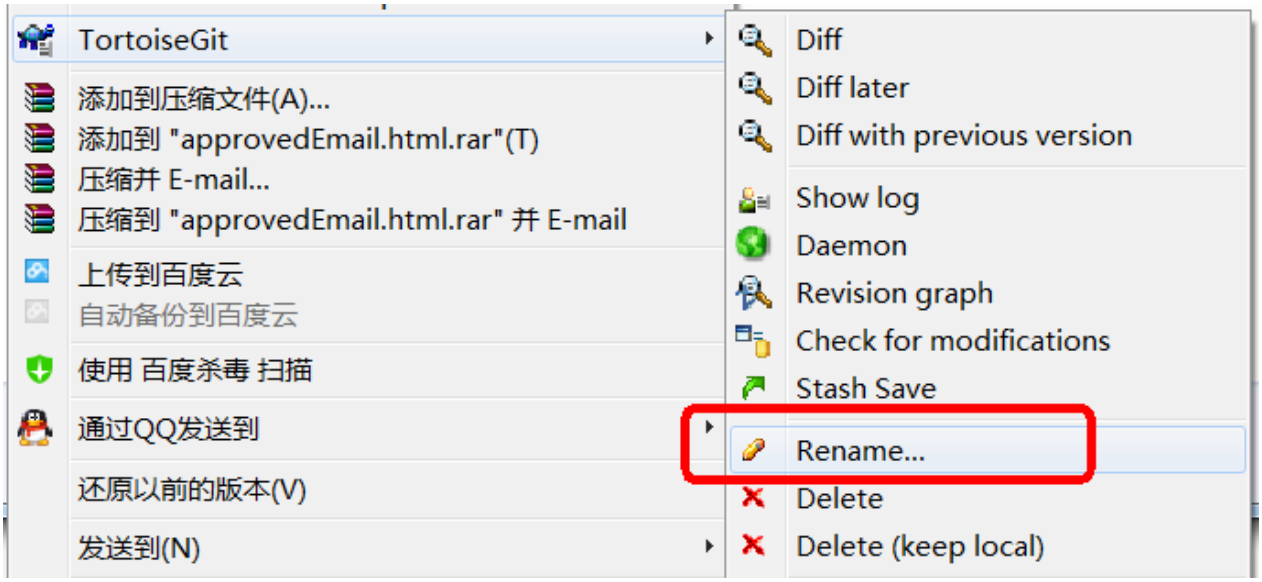


后来找到一种解决方案：

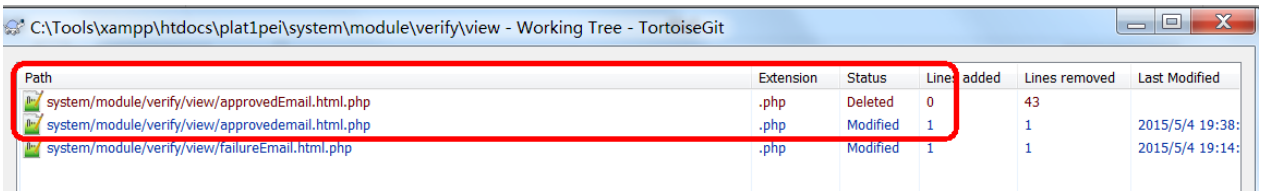
(1) 先revert文件approvedEmail.html.php文件，则在原目录下approvedemail.html.php文件消失，生成一个approvedEmail.html.php文件。



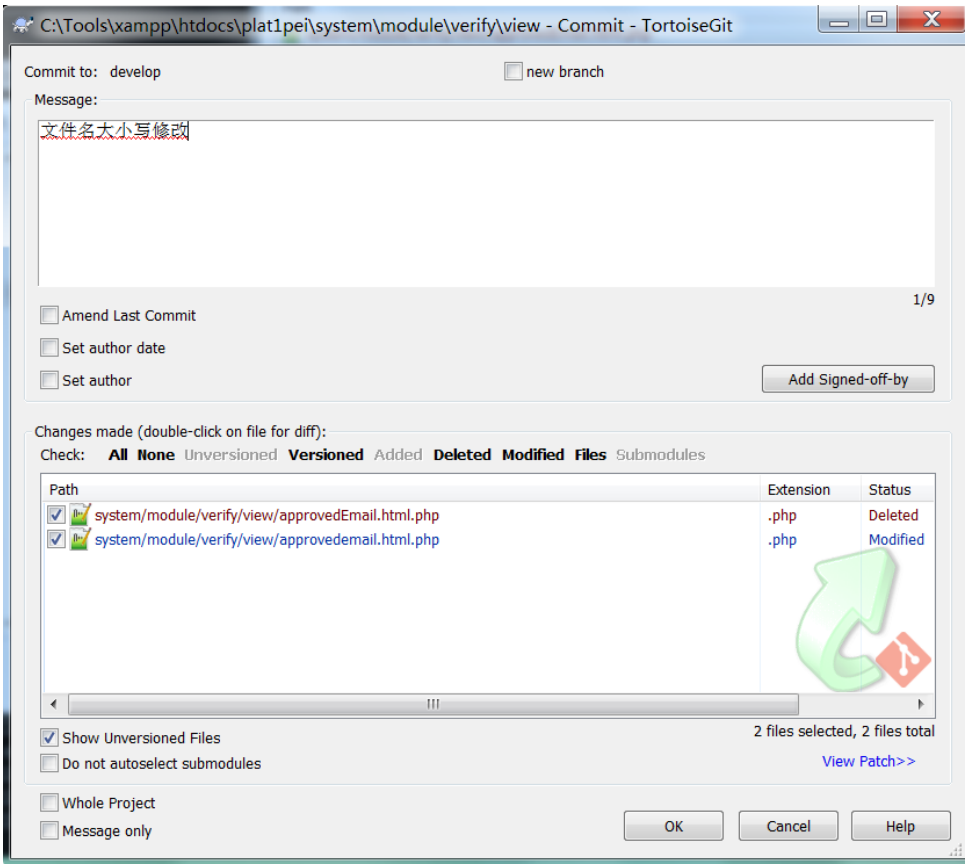
(2) 通过TortoiseGit→rename来重命名approvedEmail.html.php文件为approvedemail.html.php文件。



(3) 通过TortoiseGit→check for Modifications来检查文件修改，可以看到两个文件。



(4) 接下来合并approvedemail.html.php文件的修改，然后commit, push后解决问题。



同时需要配置文件名大小写敏感，可参见5.3.3小节。

5.6.7 常用操作—pull到本地工作空间时未跟踪文件覆盖错误解决

在一配云平台上线过程中，发现存在还出现一个问题，在Pull时提示一些未跟踪的文件需要解决覆盖问题：

```
git.exe pull -v --progress "origin"

remote: Counting objects: 291, done.
remote: Compressing objects: 100% (54/54), done.
Receiving objects: 100% (167/167), 83.77 KiB | 0 bytes/s, done.
Resolving deltas: 100% (130/130), completed with 63 local objects.
remote: Total 167 (delta 130), reused 138 (delta 108)
From 192.168.1.160:/home/git/repositories/cloud1pei
b4339c7..54ea187  develop    -> origin/develop
= [up to date]      master     -> origin/master
Updating b4339c7..54ea187
error: Your local changes to the following files would be overwritten by merge:
app/sys/user/ext/lang/zh-cn/newmemberemail.php
Please, commit your changes or stash them before you can merge.
error: The following untracked working tree files would be overwritten by merge:
app/crm/address/ext/model/getareaInfo.php
app/crm/contract/ext/model/getList.php
app/sys/search/ext/view/buildBasicForm.html.php
app/sys/user/ext/lang/zh-cn/newMemberEmail.php
Please move or remove them before you can merge.
```

解决方案：在命令行下面执行rm这些未跟踪的文件，然后执行git pull即可成功。

```
C:\Tools\xampp\htdocs\cloud1pei>rm app/crm/address/ext/model/getareaInfo.php

C:\Tools\xampp\htdocs\cloud1pei>rm app/crm/contract/ext/model/getList.php

C:\Tools\xampp\htdocs\cloud1pei>rm app/sys/search/ext/view/buildBasicForm.html.php

C:\Tools\xampp\htdocs\cloud1pei>rm app/sys/user/ext/lang/zh-cn/newMemberEmail.php

C:\Tools\xampp\htdocs\cloud1pei>
```

5.6.8 常用操作—Git Stash

git stash save: 备份当前的工作区的内容，从最近的一次提交中读取相关内容，让工作区保证和上次提交的内容一致。同时，将当前的工作区内容保存到Git栈中。

git stash list: 显示Git栈内的所有备份，可以利用这个列表来决定从那个地方恢复。

git stash pop: 从Git栈中读取最近一次保存的内容，恢复工作区的相关内容。由于可能存在多个Stash的内容，所以用栈来管理，pop会从最近的一个stash中读取内容并恢复。

git stash apply：类似于git stash pop，但不会从stash list中删除状态。

git stash clear: 清空Git栈。此时使用gitg等图形化工具会发现，原来stash的哪些节点都消失了。

为什么需要使用Git Stash呢？使用git的时候，我们往往使用branch解决任务切换问题，例如，我们往往会建一个自己的分支去修改和调试代码, 如果别人或者自己发现原有的分支上有个不得不修改的bug，我们往往会把完成一半的代码 commit提交到本地仓库，然后切换分支去修改bug，改好之后再切换回来。这样的话往往log上会有大量不必要的记录。其实如果我们不想提交完成一半或者不完善的代码，但是却不得不去修改一个紧急Bug，那么使用'git stash'就可以将你当前未提交到本地（和服务器）

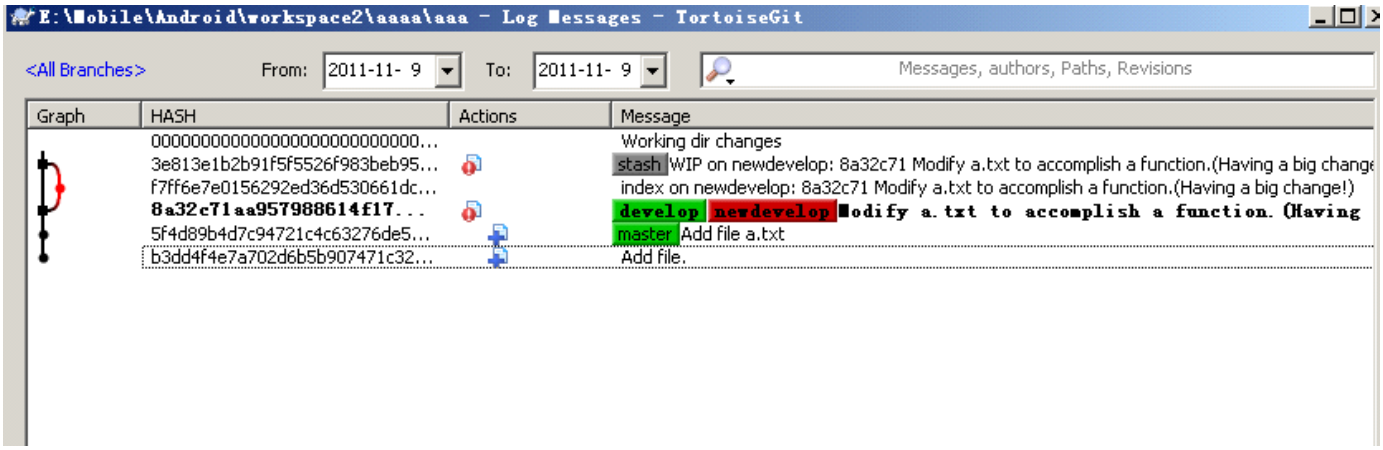
的代码推入到Git的栈（stash区，或者说是一个暂存区）中，这时候你的工作区间和上一次提交的内容是完全一样的，所以你可以放心的切换分支去修 Bug，等到修完Bug，提交到服务器上后，再使用'git stash apply'将以前一半的工作应用回来。也许有的人会说，那我可不可以多次将未提交的代码压入到栈中？答案是可以的。当你多次使用'git stash'命令后，你的栈里将充满了未提交的代码，这时候你会对将哪个版本应用回来有些困惑，'git stash list'命令可以将当前的Git栈信息打印出来，你只需要将找到对应的版本号，例如使用'git stash apply stash@{1}'就可以将你指定版本号为stash@{1}的工作取出来，当你将所有的栈都应用回来的时候，可以使用'git stash clear'来将栈清空。

在这里顺便提下git format-patch -n , n是具体某个数字， 例如 'git format-patch -1' 这时便会根据log生成一个对应的补丁，如果 'git format-patch -2' 那么便会生成2个补丁，当然前提是你的log上有至少有两个记录。

看过上面的信息，就可以知道使用场合了：当前工作区内容（假定是develop分支）已被修改，但是并未完成。这时Boss来了，说前面的release1.0分支上面有一个Bug，需要立即修复。可是我又不想提交目前的修改到本地develop分支上，因为修改没有完成。但是，不提交的话，又没有办法checkout到前面的release1.0分支。此时用Git Stash就相当于备份工作区了。然后再Checkout过去修改BUG，就能够达到保存当前工作区，并及时恢复的作用。

Git bash图示：

从图中可以看到，develop和newdevelop是在同一个分支上，因为分支newdevelop是在develop分支的基础上开发的。想加入一个新的特性，所以就开了newdevelop分支，然后就在上面加东西，加特性，该代码。这个时候工作的内容已经变化了，但是develop和newdevelop都是指向同一个提交的，因为newdevelop上面还没有提交。这个时候，Boss来了，说develop上面有个Bug，赶快改一下，手头的工作先放放，稳定版本不能有缺陷。没办法，当前正在newdevelop上搞的high呢，就Git Stash一下。在Git Stash之后，提交图如下所示：



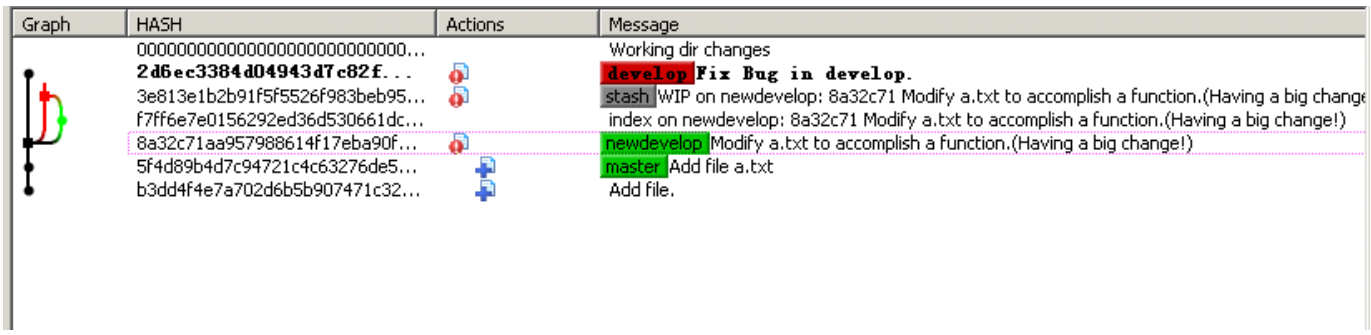
stash后会看到上面有两个节点，红色以及上面一个。就是stash之后的结果，注意是在newdevelop上面进行的stash。

正如前面所说，stash会暂存当前的工作区内容，然后将工作区内容保持和上次提交相同，此时内容都是上面8a32那个提交的内容。从终端中查看相应的信息内容，如下：

```
Administrator@LIUCHANG /e/Mobile/Android/workspace2/aaaa/aaa (newdevelop)
$ git status
# On branch newdevelop
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   a.txt
#
no changes added to commit (use "git add" and/or "git commit -a")

Administrator@LIUCHANG /e/Mobile/Android/workspace2/aaaa/aaa (newdevelop)
$ git stash
Saved working directory and index state WIP on newdevelop: 8a32c71 Modify a.txt
to accomplish a function.(Having a big change!)
HEAD is now at 8a32c71 Modify a.txt to accomplish a function.(Having a big chang
e!)

Administrator@LIUCHANG /e/Mobile/Android/workspace2/aaaa/aaa (newdevelop)
$ git status
# On branch newdevelop
nothing to commit (working directory clean)

Administrator@LIUCHANG /e/Mobile/Android/workspace2/aaaa/aaa (newdevelop)
$
```

印证了前面的说法，newdevelop是有修改，modified，然后stash之后，工作区是最近一次提交，此时newdevelop和develop都是相同的，所以再git status查看发现，都一样，nothing to commit.

然后Stash完成之后，就要Fix Bug了。为此，回到develop分支上进行修复，然后提交，完成后的提交图如下所示：



从上图可以看到，newdevelop还是在下面，因为指向的是老的那个8a32的commit。新的develop由于修复了Bug，所以产生一个新提交。

```
Administrator@LIUCHANG /e/Mobile/Android/workspace2/aaaa/aaa (develop)
$ git checkout newdevelop
Switched to branch 'newdevelop'

Administrator@LIUCHANG /e/Mobile/Android/workspace2/aaaa/aaa (newdevelop)
$ git status
# On branch newdevelop
nothing to commit (working directory clean)

Administrator@LIUCHANG /e/Mobile/Android/workspace2/aaaa/aaa (newdevelop)
$
```

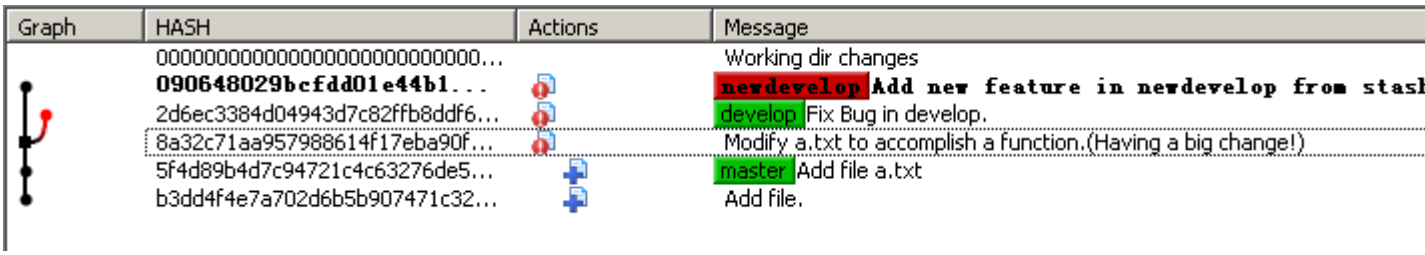
然后在develop上面修复了Bug之后，在回到newdevelop上面进行一个新的特性的继续编码，此时checkout回去的时候，没有什么内容可以提交，因为都存在Stash中了，没有任何修改。如上图。

那么，恢复工作区内容吧。于是git stash pop（注意这里由于只Stash了一次所以使用pop，具体你存放了多少，要恢复哪一个要自己清楚，否则会出错！）

```
Administrator@LIUCHANG /e/Mobile/Android/workspace2/aaaa/aaa (newdevelop)
$ git status
# On branch newdevelop
nothing to commit (working directory clean)

Administrator@LIUCHANG /e/Mobile/Android/workspace2/aaaa/aaa (newdevelop)
$ git stash pop
# On branch newdevelop
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   a.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (3e813e1b2691f5f5526f983beb955cfa3166fd21)

Administrator@LIUCHANG /e/Mobile/Android/workspace2/aaaa/aaa (newdevelop)
$ git status
# On branch newdevelop
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   a.txt
#
no changes added to commit (use "git add" and/or "git commit -a")

Administrator@LIUCHANG /e/Mobile/Android/workspace2/aaaa/aaa (newdevelop)
$
```

恢复之后，从上图中可以看到，此时再git status就会发现文件有修改，说明恢复过来了。然后就继续编码，提交一个稳定的新特性版本，如下图，产生的新提交为0906.

然后再查看提交图，会发现，stash pop之后，对应的存放的stash被清空掉了，提交图中，newdevelop上面对应一个新的提交。并且在develop上面。分支的develop那个红色，即为前面修复Bug的那个提交。



总结起来：

操作很简单，但是头脑要清楚。要在哪个分支上修复Bug，要暂存哪个地方的内容，之后修复完了在那个地方提交，然后要到哪个分支上面恢复工作区，都是需要注意的，否则，很容易造成提交图混乱。只有弄清楚了工作流程，才不容易出错，才能保证很高的工作效率。

5.6.9 常用操作—合并特定Commits到另一个分支

Git合并特定commits 到另一个分支

5.6.9.1 合并某个分支上的单个commit

首先，用git log或GitX工具查看一下你想选择哪些commits进行合并，例如：

dd2e86 - 946992 -9143a9 - a6fd86 - 5a6057 [master]

\

76cada - 62ecb3 - b886a0 [feature]

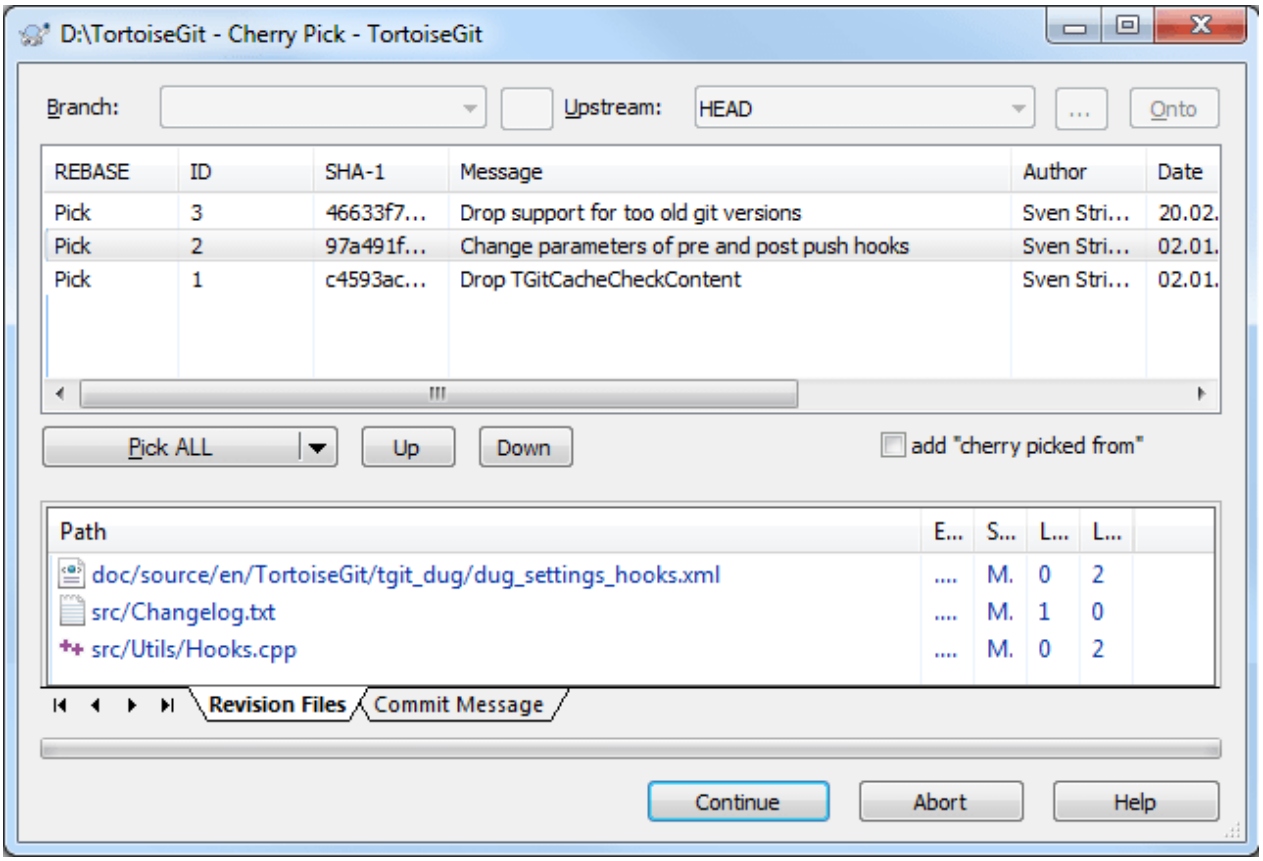
比如，feature 分支上的commit 62ecb3 非常重要，它含有一个bug的修改，或其他人想访问的内容。无论什么原因，你现在只需要将62ecb3 合并到master，而不合并feature上的其他commits，所以我们用git cherry-pick命令来做：

git checkout master

git cherry-pick 62ecb3

这样就好啦。现在62ecb3 就被合并到master分支，并在master中添加了commit（作为一个新的commit）。cherry-pick 和merge比较类似，如果git不能合并代码改动（比如遇到合并冲突），git需要你自己来解决冲突并手动添加commit。

当选择某一个commit合并时：Select Cherry Pick this commit...，当选择多个commits时：Cherry Pick select commits...。但TortoiseGit怎么样来使用呢？



查看仓库历史记录

git log有三个你应该知道的选项。

--oneline- 压缩模式，在每个提交的旁边显示经过精简的提交哈希码和提交信息，以一行显示。

--graph- 图形模式，使用该选项会在输出的左边绘制一张基于文本格式的历史信息表示图。如果你查看的是单个分支的历史记录的话，该选项无效。

--all- 显示所有分支的历史记录

把这些选项组合起来之后，输出看起来会像这样：

```
$ git log --oneline --graph --all
* cfda895 FIX|仓储配货优化之合并变色-mahy
* a5aa562 FIX|合并调仓07.12修改
* 84c3395 FIX|待配货页面打印初步做成模态框样式-mahy
* febaa5a FIX|待配货页面打印初步做成模态框样式-mahy
|   ed01f11 Merge branch 'cloud1pei-scm' of 192.168.1.160:/home/git/repositories/cloud1pei into cloud1pei-scm
| |\
| | * 6645ee9 FIX|待配货页面打印初步做成模态框样式。
| |/
|/|
| * b24836a FIX|合并allocation分支上的一些优化修改
|/
* 0b5842a FIX|新增调仓功能
*   b1599f2 Merge branch 'cloud1pei-scm' of 192.168.1.160:/home/git/repositories/cloud1pei into cloud1pei-scm
|\
| * 1ae386d BUG[723]|修改排序时的三角标的上下朝向
| * 49f2300 FIX|仓储中待配货以及配货历史界面表格响应式修改
| * f8a8f35 FIX|
| * 215a58e FIX|主要资产及产品删除时模态框提醒 tab制表符改为空格显示
* | c693ba0 FIX|添加注释信息
|/
* 8a5000d FIX|修改库存查询下快捷键最底部异常bug
* 155f48d FIX|配货历史界面频幕宽度过小时隐藏结款状态和审核时间列。
* c02fd45 FIX|库存查询和盘点详情界面添加产品标签打印功能。
* 732f45f BUG[927]|采购需求一待采购，创建采购订单选择一条数据时 ，无法完成。点击无效。
* 70e2e40 FIX|待配货与配货历史默认选择今日
* e1a022e FIX|解决motor brand表不在ranzhi_eid数据库中问题
* 47194ab FIX|在date类中添加一个新的公共函数computeBeginAndEnd($period)
```

5.6.9.2 合并某个分支上的一系列commits

在一些特性情况下，合并单个commit并不够，你需要合并一系列相连的commits。这种情况下就不要选择cherry-pick了，rebase 更适合（但rebase很复杂，通常不建议使用，除非对其原理非常清楚）。还以上例为例，假设你需要合并feature分支的commit 76cada ~62ecb3 到master分支。

首先需要基于feature创建一个新的分支，并指明新分支的最后一个commit：

git checkout –b newbranch 62ecb3

然后，rebase这个新分支的commit到master（--ontomaster）。76cada^ 指明你想从哪个特定的commit开始。

git rebase --ontomaster 76cada^

得到的结果就是feature分支的commit 76cada ~62ecb3 都被合并到了master分支。

5.7 权限管理器Gitosis（Git管理人员用）

Git使用4种协议：本地传输，SSH 协议，Git 协议和HTTP 协议。由于SSH是唯一一个同时便于读和写操作的网络协议，因此开发项目中通常采用SSH协议。而管理多个开发人员的SSH连接需要统一通过Gitosis来管理。Gitosis 简单的说就是一套用来管理authorized_keys文件和实现简单连接限制的脚本。最有意思的是，该软件用来添加用户和设定权限的界面不是网页，而是一个特殊的Git 仓库。你只需要设定好某个项目；然后推送，Gitosis 就会随之改变服务器设定。

6. 代码统计工具

6.1 cloc代码统计工具

可从https://github.com/AlDanial/cloc/releases/tag/v1.66下载cloc-1.66.exe工具到windows机器上，来统计代码量。

7. Dreamweaver软件

安装DreamWeaver等软件。

8. 网络抓包工具

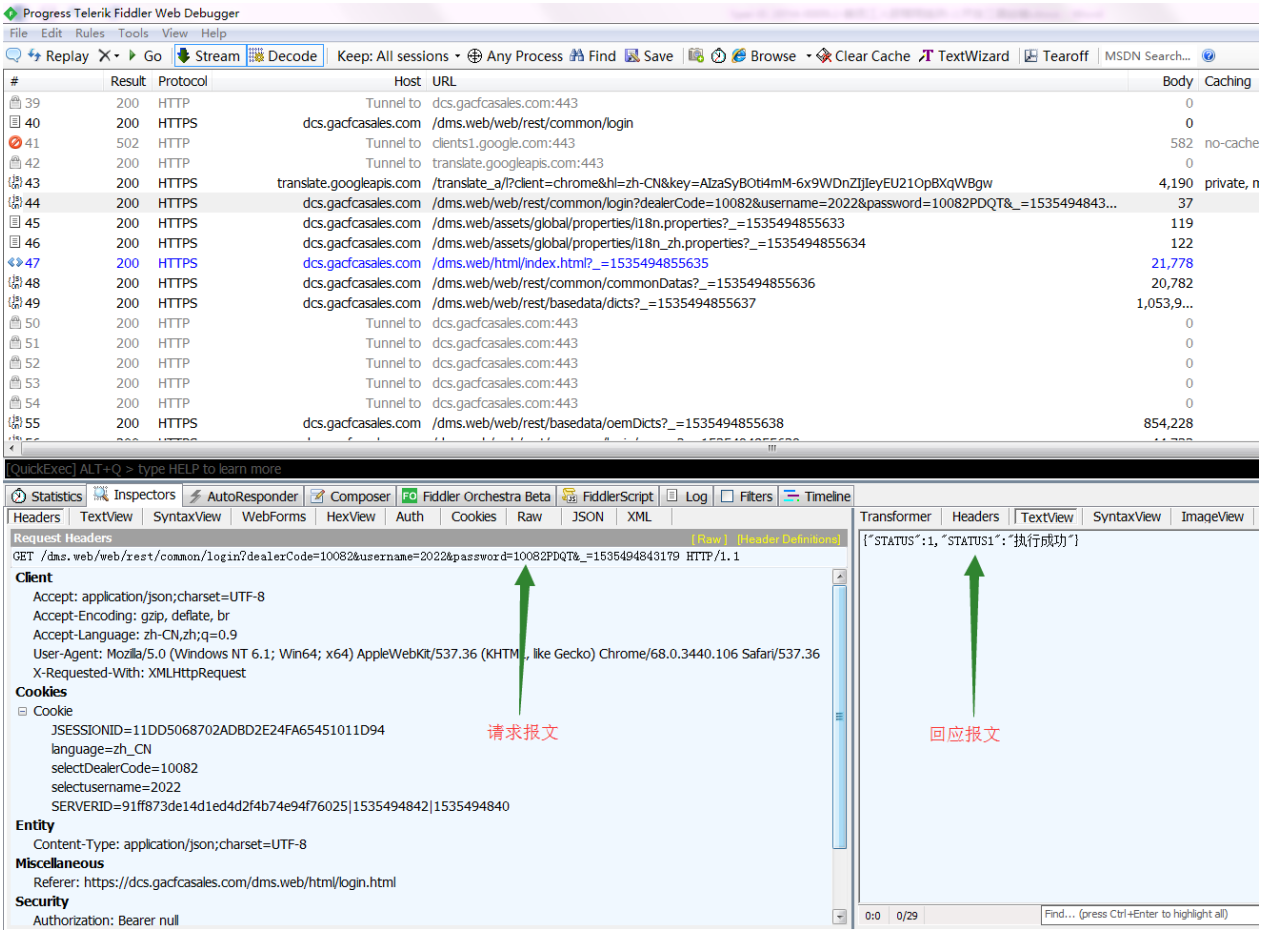
8.1 Fiddler

Fiddler是一款由C#语言开发的免费http调试代理软件，有.net 2 和 .net 4（Win8+以上适用） 两种版本。Fiddler4能够记录所有的你电脑和互联网之间的http通讯，Fiddler4可以也可以让你检查所有的https通讯，设置断点，以及Fiddle 所有的“进出”的数据。

可参见http://www.telerik.com/download/fiddler

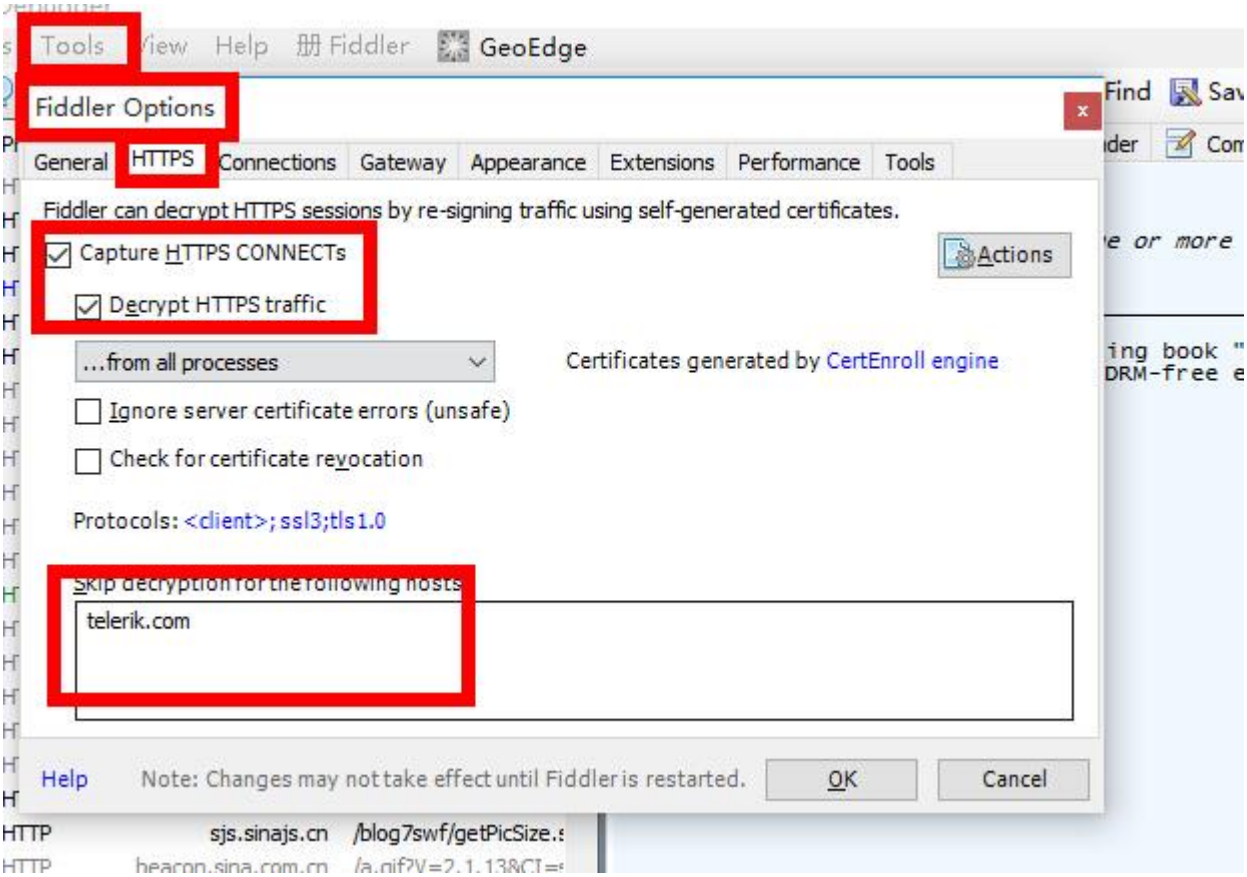
8.1.1 常用功能使用

Inspectors面板



8.1.2 监控https报文

Fiddler 想要监听https，还需要相应的设置才行，依次选中Tools->Fiddler Options->HTTPS→Decrypt HTTPS traffic：



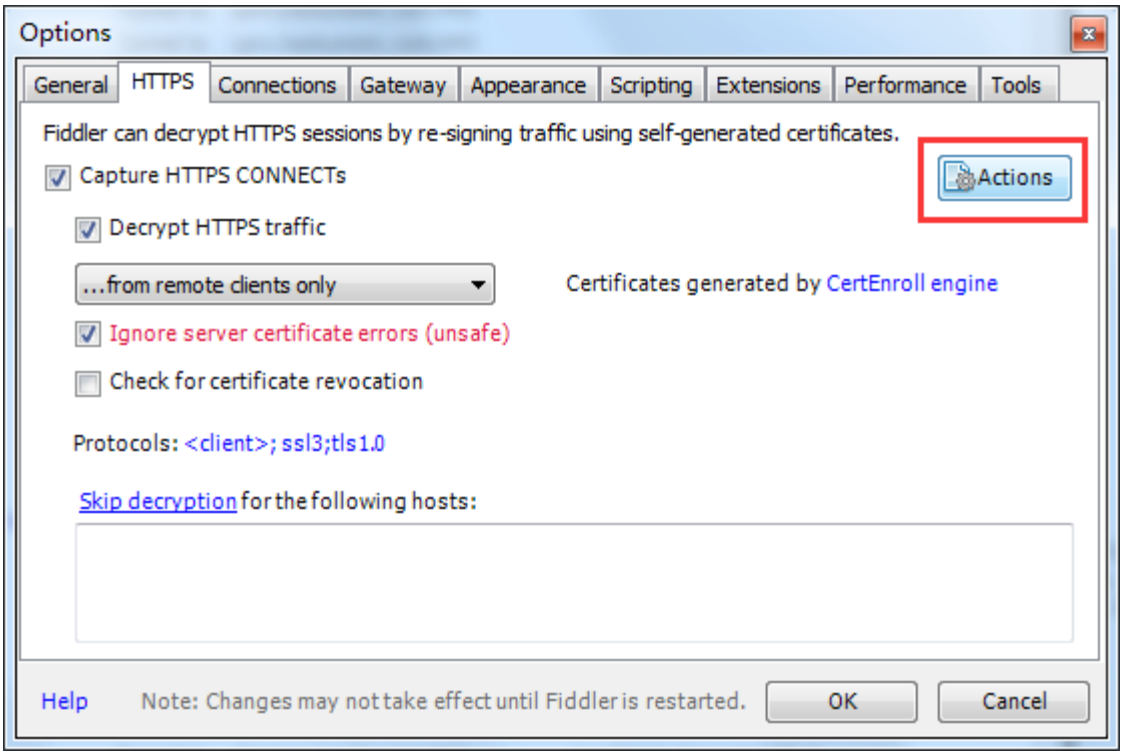
此时可能出现Fiddler启动后，但是Chrome浏览器无法https访问其他网站问题。

解决方案参考来源：

[1] Fiddler启动后Chrome浏览器无法浏览网页的解决方案, http://www.mamicode.com/info-detail-2225065.html

[2] fiddler 解决chrome不能抓取https及证书问题？ https://jingyan.baidu.com/article/f54ae2fc680be81e92b849ed.html

步骤1：导出Fiddler的信任根证书。



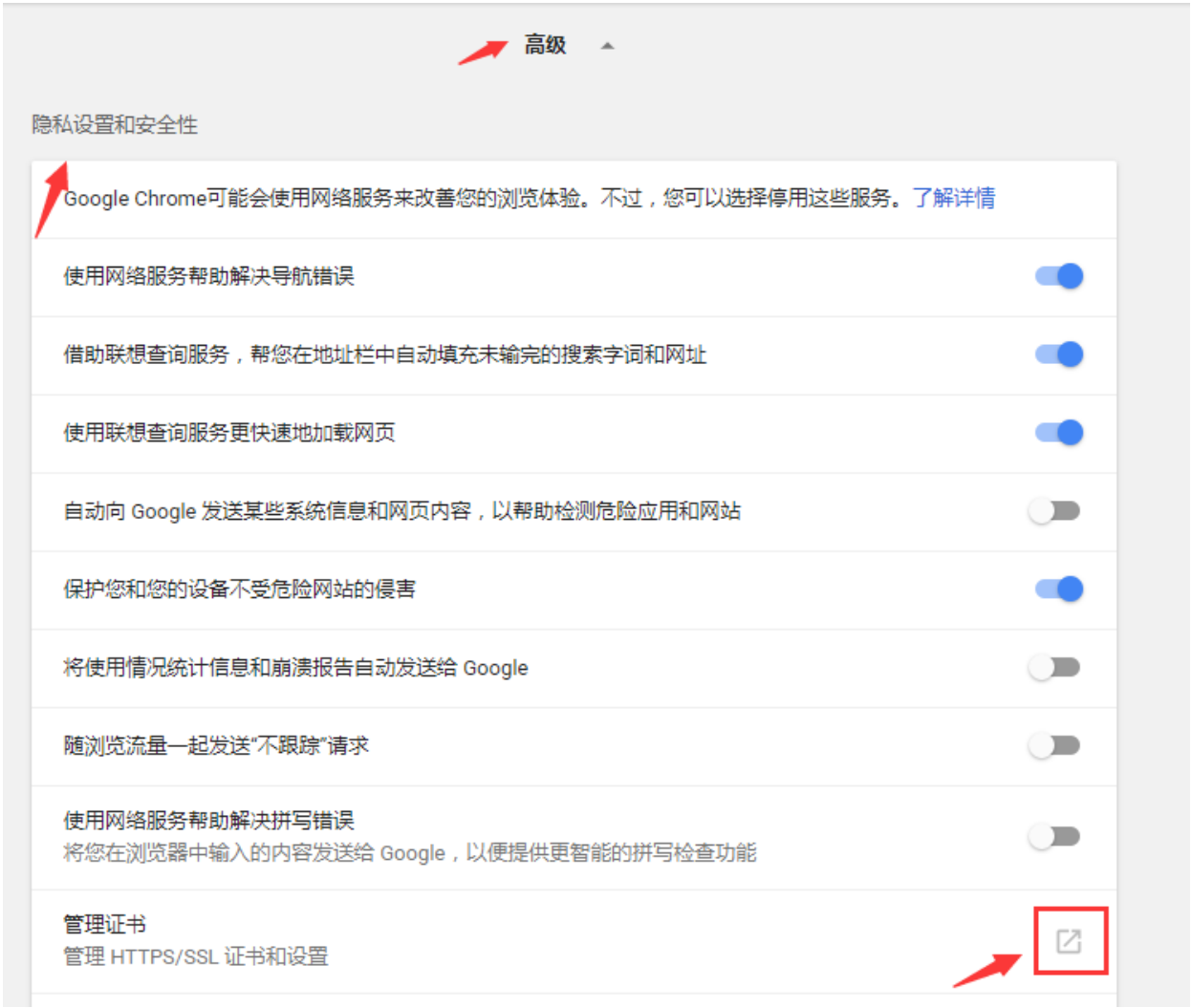
1.打开Tools>Options>HTTPS选项卡；

2.点击"Actions"下拉选择“Trust Root Certificate”-弹框选择“yes”-弹框选择“yes”；

3.点击"Actions"下拉选择“Export Root Certificate to Desktop”，将fiddler证书导出到桌面，会在桌面上生成一个文件FiddlerRoot.cer；

步骤2：将Fiddler信任根证书导入到Chrome浏览器。

打开谷歌浏览器，设置>高级>隐私设置和安全性>管理证书：



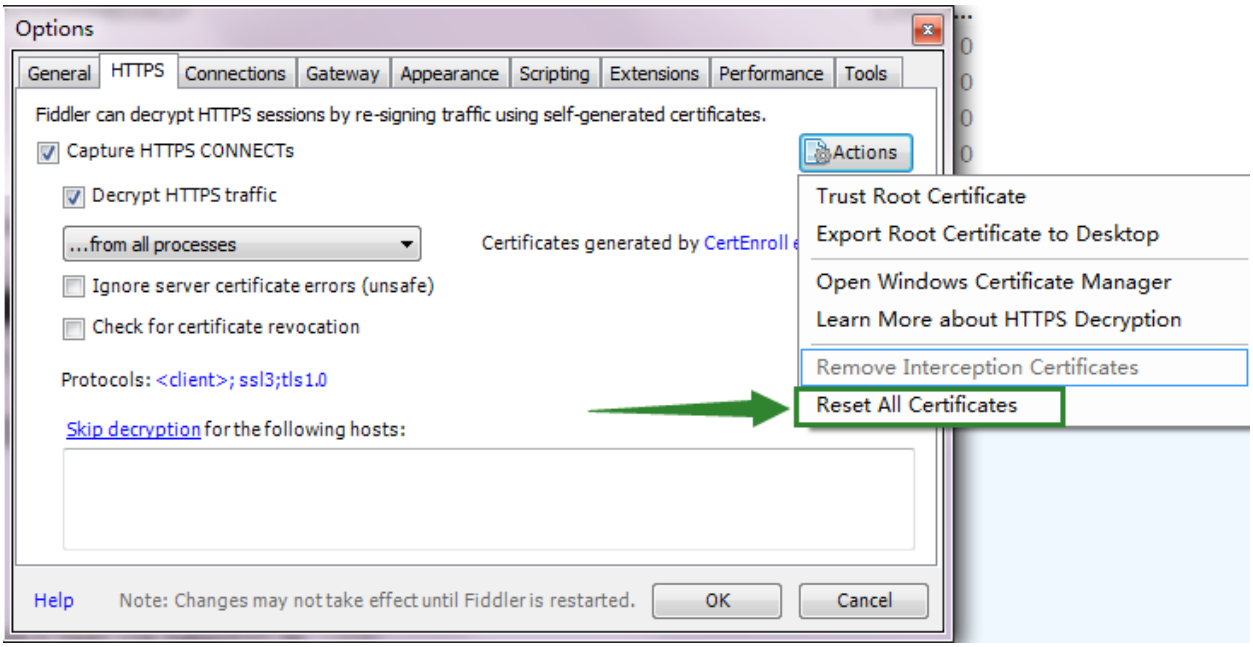
导入>下一步>浏览证书路径>下一步>下一步；

步骤3：重启浏览器和Fiddler4。

重启浏览器和Fiddler4，按理应该可以重新打开https页面了，但是不能。

步骤4：重置Fiddler4证书。

再次运行Fiddler4，依次点击Tools>Options>HTTPS选项卡，再点击actions下的Reset All Certificates，重置证书。



步骤5：再重启浏览器和Fiddler4。

重启浏览器和Fiddler，发现已经可以访问https页面了。

8.2 Wireshark

安装wireshark软件，用于网络抓包软件分析，可参见https://www.wireshark.org/。

9. 虚拟机与容器安装

9.1 虚拟机VirtualBox安装

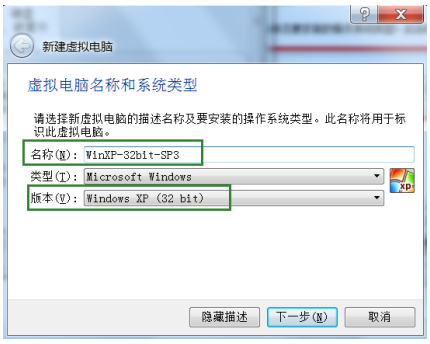
在Windows机器上有时候需要安装Linux Centos6.*或Windows XP等操作系统，此时首先需要安装虚拟机VirtualBox 5.0（2015-07-09发布）或者VirtualBox 4.3.30（2015-07-10发布），下载地址：https://www.virtualbox.org/wiki/Downloads。

9.1.1 Windows XP安装

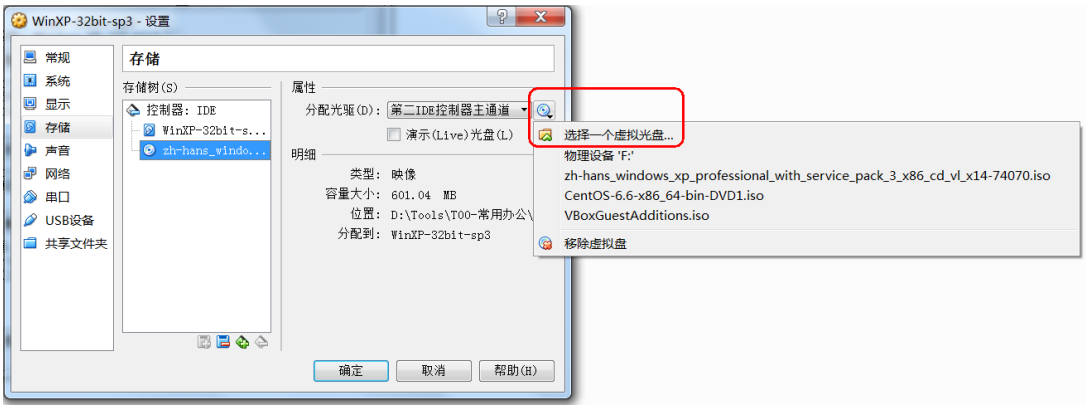
由于北京五方天雅汽配城等典型汽配城中，各个公司基本都仍然在使用WinXP SP3+IE8系统，因此一配云客户端需要能够一方面在应用HTML5+CSS3等新技术的同时，还需要能够兼容这些老的系统，在一配云客户端开发完成后，必须在WinXP SP3+IE8中来测试，确保各个按钮，输入框的功能都可以正常使用。

需要在VirtualBox中安装WinXP。

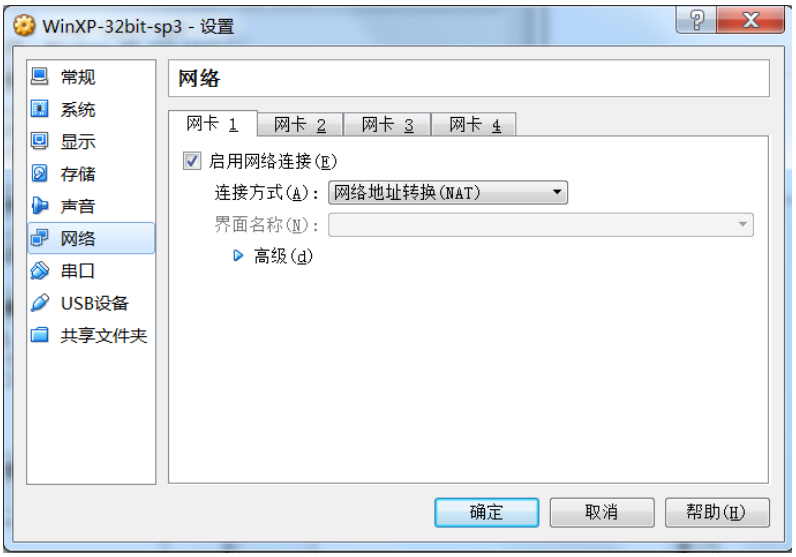
步骤1：在virtualbox里点击“新建”，指定虚拟机名称，一路next，默认情况下虚拟机的内存是512M、虚拟磁盘8G，这个配置足够运行模拟器，而且基本不会影响宿主机。

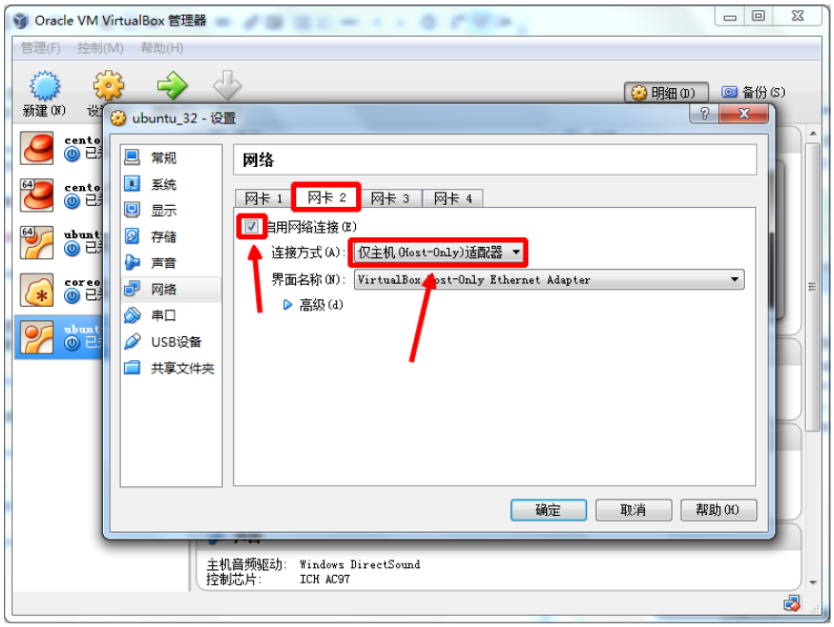


步骤2：选择刚刚创建的虚拟机，单击设置，为虚拟机选择一个ios镜像文件，准备安装系统。



步骤3：3、配置虚拟机网卡，建议为虚拟机创建两个网卡，一个NAT网卡连接外网（virtualbox默认已经创建），一个直连到主机。如果需要连接实验网还可以另外创建一个桥接到实验网网卡。





步骤4：启动虚拟机，安装WindowsXP-SP3操作系统。如果安装Linux则建议安装server版（desktop版会有图形界面，占用资源很多，很卡），进入菜单，选择“install centos”。建议一路next，语言使用英文（中文支持不好，可能会有乱码）。注意打开sshserver。

9.1.2 Win7环境中安装WinXP双操作系统

有时候需要测试WinXP环境中打印，此时在VirtualBox中通过桥接方式来获取网络打印机，但还需要安装相应的打印驱动，有的打印机在安装驱动时需要保证打印机串口连在PC机上，此时没有找到好的办法来解决，后续也有的出现VirtualBox中的WinXP无法打印的问题。此时可以考虑安装双操作系统。

可参见：颠覆微软潜规则，Win7下强行安装XP，http://itbbs.pconline.com.cn/soft/12878065.html

9.1.3 Linux Centos6.6安装

Windows环境下与Linux环境下最大的区别在于Linux中是区分文件名大小写的，而Windows不区分大小写，曾经在初次部署一配云平台时遇到大量类似的问题，因此开发环境内部的WWW服务器应当尽可能部署Linux下面的XAMPP或Apache服务器来测试。

需要在VirtualBox中安装Linux Centos6.6。

9.2 容器安装

自从1960年开始就有了虚拟化的概念，当时IBM在Cambridge科研中心大力投入研究该方向。相关开发一直在推进，但是真正的突破性进展是1999年VMware推出的虚拟化平台，最终将虚拟化成功推向了市场。正是在Vmware助力下，才开创出了hypervisor级别虚拟化的巨大市场。

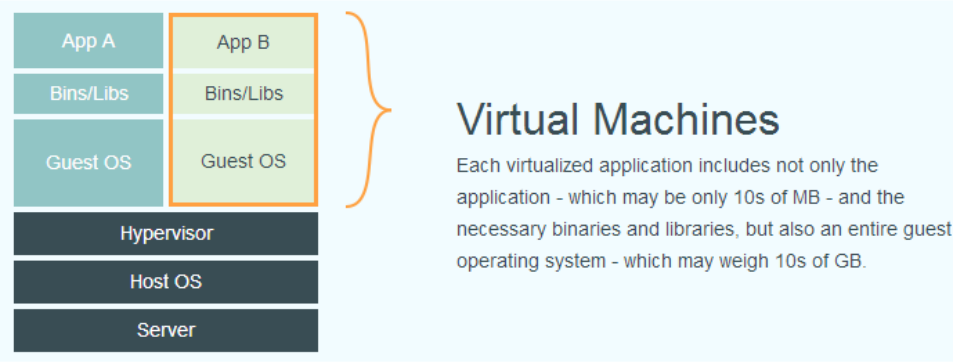
但是OS级别虚拟化（包括Vmware、VirtualBox等），这也正是Docker的基础，并没有在刚出现时就迅速火起来。这些年，OS级别虚拟化领域涌现了很多产品，但是都没有像Docker这样具有划时代的意义。

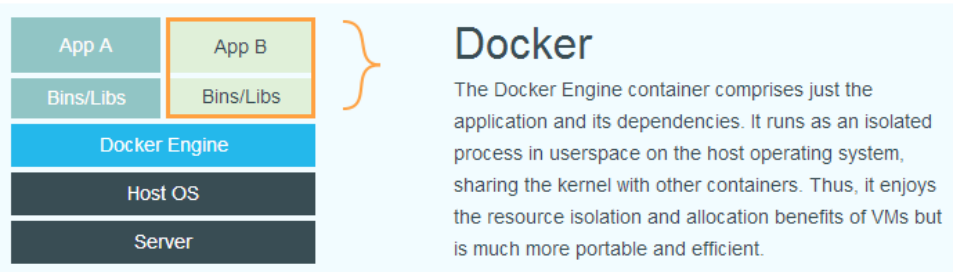
9.2.1 什么是Docker

Docker 是一个开源项目，诞生于 2013 年初，最初是 dotCloud 公司内部的一个业余项目。它基于 Google 公司推出的 Go 语言实现。项目后来加入了 Linux 基金会，遵从了 Apache 2.0 协议，项目代码在 GitHub 上进行维护。

Docker 自开源后受到广泛的关注和讨论，以至于 dotCloud 公司后来都改名为 Docker Inc。Redhat 已经在其 RHEL6.5 中集中支持 Docker；Google 也在其 PaaS 产品中广泛应用。

Docker 项目的目标是实现轻量级的操作系统虚拟化解决方案。 Docker 的基础是 Linux 容器（LXC）等技术。在 LXC 的基础上 Docker 进行了进一步的封装，让用户不需要去关心容器的管理，使得操作更为简便。用户操作 Docker 的容器就像操作一个快速轻量级的虚拟机一样简单。





从上图可以看出，容器的成本要远小于虚拟机：容器占用更少的内存和硬盘空间。虚拟机需要安装整个OS，但Docker容器则不需要。Docker只是宿主机用户空间的一个独立进程，利用内核特征如命名空间和组管理来提供这种隔离。启动一个容器只是启动了一个普通的UNIX进程；创建一个容器只是复制了一个copy-on-wirte文件系统的镜像。容器拥有虚拟机资源隔离与分配的优点，但更加轻量化、效率更高。

9.2.2 Docker的优势

首先，Docker 容器的启动可以在秒级实现，这相比传统的虚拟机方式要快得多。其次，Docker 对系统资源的利用率很高，一台主机上可以同时运行数千/数万个 Docker 容器。

容器除了运行其中应用外，基本不消耗额外的系统资源，使得应用的性能很高，同时系统的开销尽量小。传统虚拟机方式运行10个不同的应用就要起 10 个虚拟机，而Docker 只需要启动10个隔离的应用即可。

具体说来，Docker 在如下几个方面具有较大的优势。

更快速的交付和部署

对开发和运维（devop）人员来说，最希望的就是一次创建或配置，可以在任意地方正常运行。

开发者可以使用一个标准的镜像来构建一套开发容器，开发完成之后，运维人员可以直接使用这个容器来部署代码。 Docker 可以快速创建容器，快速迭代应用程序，并让整个过程全程可见，使团队中的其他成员更容易理解应用程序是如何创建和工作的。 Docker 容器很轻很快！容器的启动时间是秒级的，大量地节约开发、测试、部署的时间。

更高效的虚拟化

Docker 容器的运行不需要额外的 hypervisor 支持，**它是内核级的虚拟化，**因此可以实现更高的性能和效率。

更轻松的迁移和扩展

Docker 容器几乎可以在任意的平台上运行，包括物理机、虚拟机、公有云、私有云、个人电脑、服务器等。 这种兼容性可以让用户把一个应用程序从一个平台直接迁移到另外一个。

更简单的管理

使用 Docker，只需要小小的修改，就可以替代以往大量的更新工作。所有的修改都以增量的方式被分发和更新，从而实现自动化并且高效的管理。

对比传统虚拟机总结

特性	容器	虚拟机
启动	秒级	分钟级
硬盘使用	一般为 MB	一般为 GB
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个

9.2.3 Docker的基本概念

Docker 包括三个基本概念

- 镜像（Image）
- 容器（Container）
- 仓库（Repository）

理解了这三个概念，就理解了 Docker 的整个生命周期。

Docker 镜像就是一个只读的模板。

例如：一个镜像可以包含一个完整的 ubuntu 操作系统环境，里面仅安装了 Apache 或用户需要的其它应用程序。

镜像可以用来创建 Docker 容器。

Docker 提供了一个很简单的机制来创建镜像或者更新现有的镜像，用户甚至可以直接从其他人那里下载一个已经做好的镜像来直接使用。

Docker 利用容器来运行应用。

容器是从镜像创建的运行实例。它可以被启动、开始、停止、删除。每个容器都是相互隔离的、保证安全的平台。

可以把容器看做是一个简易版的 Linux 环境（包括root用户权限、进程空间、用户空间和网络空间等）和运行在其中的应用程序。

*注：镜像是只读的，容器在启动的时候创建一层可写层作为最上层。

仓库是集中存放镜像文件的场所。有时候会把仓库和仓库注册服务器（Registry）混为一谈，并不严格区分。实际上，仓库注册服务器上往往存放着多个仓库，每个仓库中又包含了多个镜像，每个镜像有不同的标签（tag）。

仓库分为公开仓库（Public）和私有仓库（Private）两种形式。

最大的公开仓库是 Docker Hub，存放了数量庞大的镜像供用户下载。 国内的公开仓库包括 Docker Pool 等，可以提供大陆用户更稳定快速的访问。

当然，用户也可以在本地网络内创建一个私有仓库。

当用户创建了自己的镜像之后就可以使用 push 命令将它上传到公有或者私有仓库，这样下次在另外一台机器上使用这个镜像时候，只需要从仓库上 pull 下来就可以了。

*注：Docker 仓库的概念跟 Git 类似，注册服务器可以理解为 GitHub 这样的托管服务。

9.2.4 DockerToolbox

最新版本：docker toolbox v1.9.1g（For Windows-64bit）, 2016-01-04

Included Components：

docker Client 1.9.1

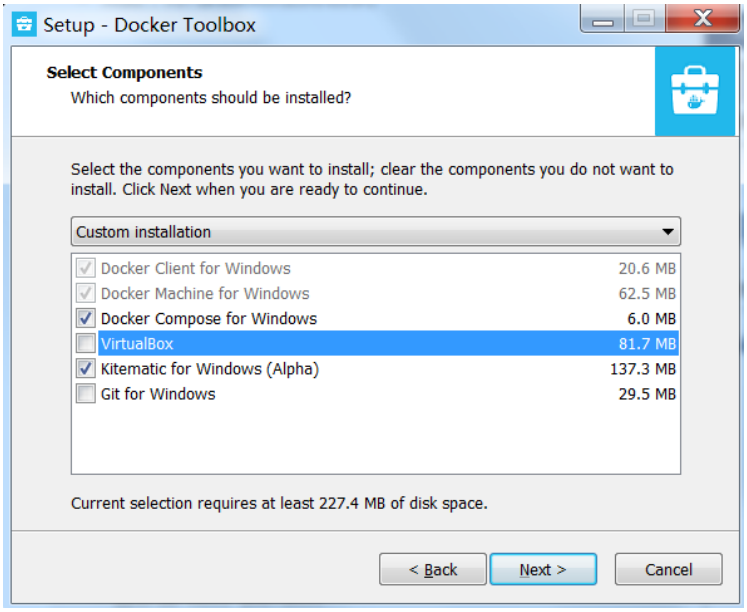
docker-machine 0.5.5

docker-compose 1.5.2

Kitematic 0.9.5

Boot2Docker ISO 1.9.1

VirtualBox 5.0.12



Docker Toolbox取代了Boot2Docker吗?

答：尽管Boot2Docker安装程序已经相当的受欢迎，但Docker Toolbox是设计用来安装正在不断发展的Docker开发者工具集合，比如Kitematic、Machine、Swarm还有Compose。

之前Boot2Docker还安装了一个叫Boot2Docker的命令行工具，以用来管理Docker虚拟机，在Toolbox中它已经被Machine取代了。然而，在这个引擎下，Machine依然采用了Boot2Docker Linux发行版来运行容器。所不同的是，现在由Machine代替Boot2Docker命令行工具来管理这些容器。

如果你现在正在使用官方Boot2Docker（boot2docker-VM），Docker Toolbox会提示你自动迁移到使用Docker Machine的虚拟机上。

国内获取网站：https://get.daocloud.io/toolbox/

9.2.5 CentOS 系列安装 Docker

Docker 支持 CentOS6 及以后的版本。

CentOS6

对于 CentOS6，可以使用 EPEL 库安装 Docker，命令如下

$ sudo yum install http://mirrors.yun-idc.com/epel/6/i386/epel-release-6-8.noarch.rpm

$ sudo yum install docker-io

CentOS7

CentOS7 系统 CentOS-Extras 库中已带 Docker，可以直接安装：

$ sudo yum install docker

安装之后启动 Docker 服务，并让它随系统启动自动加载。

$ sudo service docker start

$ sudo chkconfig docker on

在容器领域，主要供应商重要发布包括Docker、CoreOS、Kubernetes和Mesos。

10. 代码静态分析工具for PHP

10.1 windows下安装composer方法

Java有Maven，Python有pip，Ruby有gem，Node.js有npm，这些语言的程序员在开心地使用包管理工具加速开发效率时，PHPer们还在复制粘贴的黑暗中。PHP在Composer之前，包管理的历史不堪回首。

在相当长的一段时间内，如果应用依赖于第三方库，PHPer需要拷贝这些库的源代码， 或者通过PEAR、PECL安装。不过PEAR坑不少：依赖处理容易出问题、配置非常复杂、难用的命令行接口。如果第三方库又依赖于更多的第三方库，那么很快就会进入依赖的黑洞。直到Composer出现，PHPer们看到了属于PHP的包管理的曙光。

Composer 是由 Jordi Boggiano 和 Nils Aderman 创造的一个命令行工具，它的使命就是帮你为项目自动安装所依赖的开发包。Composer 中的很多理念都借鉴了 npm 和 Bundler，如果你对这两个工具有所了解的话，就会在 composer 中发现他们的身影。Composer 包含了一个依赖解析器，用来处理开发包之间复杂的依赖关系；另外，它还包含了下载器、安装器等有趣的东西。

可参考链接：http://www.phpcomposer.com/

通过浏览器访问https://getcomposer.org/installer，下载installer文件到c:/Tools/xampp/php目录下，然后修改文件名为“composer_installer.phar”；接下来在dos窗口中执行php composer_installer.phar，

```
C:\Tools\xampp\php>php composer_installer.phar
All settings correct for using Composer
Downloading...

Composer successfully installed to: C:\Tools\xampp\php\composer.phar
Use it: php composer.phar
```

然后运行“php composer.phar -V”，如果显示版本号，表示安装成功。以后可以在其他目录，通过“php composer.phar 其他命令行参数”执行composer的其他功能了。

```
C:\Tools\xampp\php>php composer.phar -V
Composer version 1.0-dev (bc45d9185513575434021527d7756420e9f4b2cf) 2015-05-11 14:49:39

C:\Tools\xampp\php>
```

直接运行“php composer.phar”，就可以看到composer所有命令行功能了。

```
C:\Tools\xampp\php>php composer.phar
   ______
  / ____/___  ____ ___  ____  ____  ________  _____
 / /   / __ \/ __ `__ \/ __ \/ __ \/ ___/ _ \/ ___/
/ /___/ /_/ / / / / / / /_/ / /_/ (__  )  __/ /
\____/\____/_/ /_/ /_/ .___/\____/____/\___/_/
                    /_/
Composer version 1.0-dev (bc45d9185513575434021527d7756420e9f4b2cf) 2015-05-11 14:49:39

Usage:
 command [options] [arguments]

Options:
 --help (-h)           Display this help message
 --quiet (-q)          Do not output any message
 --verbose (-v|vv|vvv) Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
 --version (-V)        Display this application version
 --ansi                Force ANSI output
 --no-ansi             Disable ANSI output
 --no-interaction (-n) Do not ask any interactive question
 --profile             Display timing and memory usage information
 --working-dir (-d)    If specified, use the given directory as working directory.

Available commands:
 about            Short information about Composer
 archive          Create an archive of this composer package
 browse           Opens the package's repository URL or homepage in your browser.
 clear-cache      Clears composer's internal package cache.
 clearcache       Clears composer's internal package cache.
 config           Set config options
 create-project   Create new project from a package into given directory.
 depends          Shows which packages depend on the given package
 diagnose         Diagnoses the system to identify common errors.
 dump-autoload    Dumps the autoloader
 dumpautoload     Dumps the autoloader
 global           Allows running commands in the global composer dir ($COMPOSER_HOME).
 help             Displays help for a command
 home             Opens the package's repository URL or homepage in your browser.
 info             Show information about packages
 init             Creates a basic composer.json file in current directory.
 install          Installs the project dependencies from the composer.lock file if present, or falls back on the composer.json.
 licenses         Show information about licenses of dependencies
 list             Lists commands
 remove           Removes a package from the require or require-dev
 require          Adds required packages to your composer.json and installs them
 run-script       Run the scripts defined in composer.json.
 search           Search for packages
 self-update      Updates composer.phar to the latest version.
 selfupdate       Updates composer.phar to the latest version.
 show             Show information about packages
 status           Show a list of locally modified packages
 update           Updates your dependencies to the latest version according to composer.json, and updates the composer.lock file.
```

10.2 windows下安装代码静态分析工具phpmd

在安装好composer后，准备安装代码静态分析工具phpmd。

首先修改C:\tools\xampp\php\php.ini文件，搜索“pgsql”并打开pgsql功能，搜索”SSL”打开SSL功能：

extension=php_pgsql.dll

extension=php_openssl.dll

然后重启xampp中的apache服务。

下载http://static.pdepend.org/php/2.0.6/pdepend.phar与http://static.phpmd.org/php/latest/phpmd.phar文件，将这两个文件保存到c:/Tools/xampp/php目录下。

然后进入C:\Tools\xampp\htdocs\plat1pei项目目录下，新建一个文件composer.json，其内容为：

```
{
    "require": {
        "pdepend/pdepend" : "2.0.6",
        "phpmd/phpmd" : "2.2.2"
    }
}
```

执行命令行：

C:\Tools\xampp\htdocs\plat1pei>c:\Tools\xampp\php\php.exe c:\Tools\xampp\php\composer.phar install –verbose

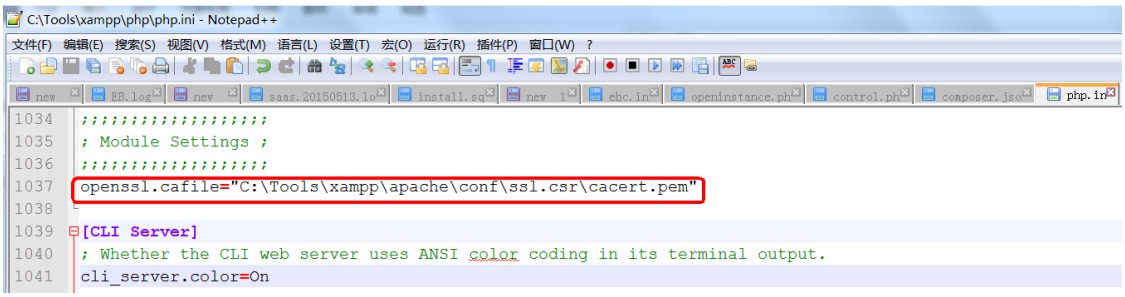
但是出现下面错误信息：

```
C:\Tools\xampp\htdocs\plat1pei>c:\Tools\xampp\php\php.exe c:\Tools\xampp\php\composer.phar install --verbose
Loading composer repositories with package information
The "https://packagist.org/packages.json" file could not be downloaded: SSL operation failed with code 1. OpenSSL Error messages:
error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate verify failed
Failed to enable crypto
failed to open stream: operation failed
https://packagist.org could not be fully loaded, package information was loaded from the local cache and may be out of date
Installing dependencies (including require-dev)
```

检查ssl cert信息：

```
C:\Tools\xampp\htdocs\plat1pei>c:\Tools\xampp\php\php.exe -r var_dump(openssl_get_cert_locations());
array(8) {
  ["default_cert_file"]=>
  string(41) "f:\repo\winlibs_openssl_vc11_x86/cert.pem"
  ["default_cert_file_env"]=>
  string(13) "SSL_CERT_FILE"
  ["default_cert_dir"]=>
  string(38) "f:\repo\winlibs_openssl_vc11_x86/certs"
  ["default_cert_dir_env"]=>
  string(12) "SSL_CERT_DIR"
  ["default_private_dir"]=>
  string(40) "f:\repo\winlibs_openssl_vc11_x86/private"
  ["default_default_cert_area"]=>
  string(32) "f:\repo\winlibs_openssl_vc11_x86"
  ["ini_cafile"]=>
  string(0) ""
  ["ini_capath"]=>
  string(0) ""
}

C:\Tools\xampp\htdocs\plat1pei>
```

从http://curl.haxx.se/ca/cacert.pem下载文件到C:\Tools\xampp\apache\conf\ssl.csr\目录下，然后设置php.ini，并重启apache：

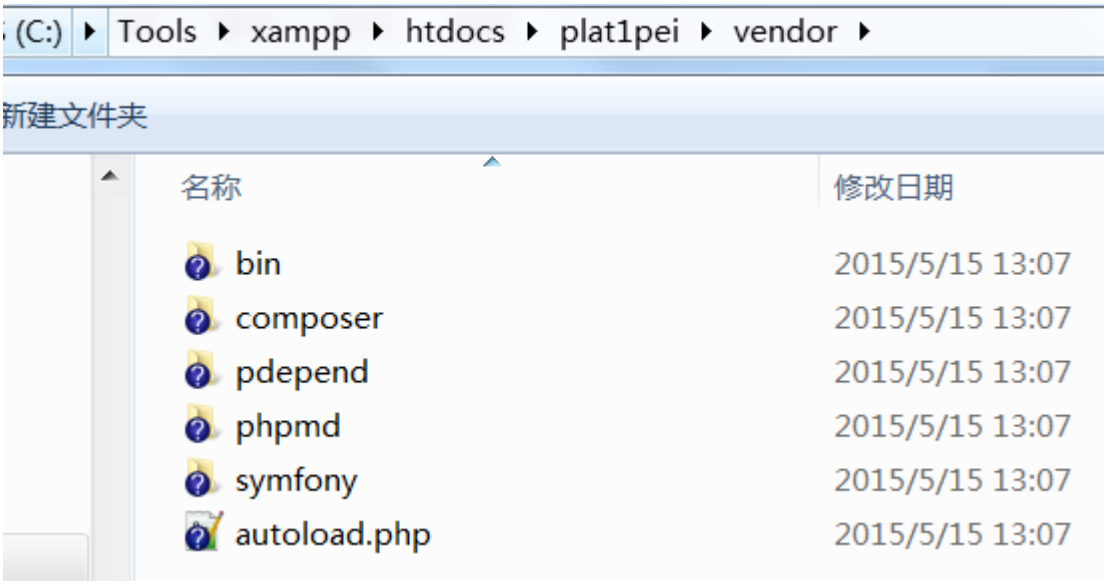


接下来执行

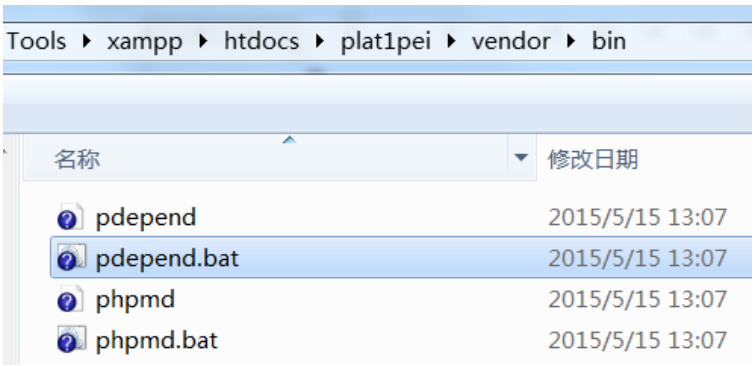
c:\Tools\xampp\php\php.exe c:\Tools\xampp\php\composer.phar install –verbose，就开始安装相应的phpmd：

```
C:\Tools\xampp\htdocs\plat1pei>c:\Tools\xampp\php\php.exe c:\Tools\xampp\php\composer.phar install --verbose
Loading composer repositories with package information
Installing dependencies (including require-dev)
  - Installing symfony/filesystem (v2.6.7)
    Downloading: 100%
    Extracting archive

  - Installing symfony/config (v2.6.7)
    Downloading: 100%
    Extracting archive

  - Installing symfony/dependency-injection (v2.6.7)
    Downloading: 100%
    Extracting archive

  - Installing pdepend/pdepend (2.0.6)
    Downloading: 100%
    Extracting archive

  - Installing phpmd/phpmd (2.2.2)
    Downloading: 100%
    Extracting archive

symfony/dependency-injection suggests installing symfony/yaml ()
symfony/dependency-injection suggests installing symfony/proxy-manager-bridge (Generate service proxies to lazy load them)
Writing lock file
Generating autoload files

C:\Tools\xampp\htdocs\plat1pei>
```

查看c:\Tools\xampp\htdocs\plat1pei目录下，新生成了vendor文件夹，其中包含pdepend, phpmd文件夹：



查看vendor/bin目录，其中有phpmd.bat文件，在windows中就利用该bat文件来执行代码检查：



此时，就可以执行命令行c:\Tools\xampp\htdocs\plat1pei\vendor\bin\phpmd c:\Tools\xampp\htdocs\plat1pei xml/txt/html codesize，即可对plat1pei项目进行测试。

下面介绍phpmd的使用

1. 语法

phpmd [文件名|目录] [报告格式] [规则集文件]

2. 参数

--minimumpriority

设定检查的规则优先级别，低于该值的优先级检查将不再使用。

--reportfile

以文件方式报告，而非默认在命令行中报告。

如： --reportfile phpmd.xml，将会在window用户目录下生成phpmd.xml文件。

--suffixes

设定只检查哪些后缀的文件程序。

--exclude

以目录形式，将该目录下的文件排除检查，可以以逗号形式排除多个目录。

--strict

仍然报告哪些通过设置“抑制报告错误”的问题。

3. 抑制报告错误

可以通过在被检查的文件添加注释的方法，抑制报告某些类型的错误，方法详见：http://phpmd.org/documentation/suppress-warnings.html。

4. 自定义使用规则集

（1）使用自带的多个规则集（每个规则的描述与举例可参见http://phpmd.org/rules/index.html）

PHPMD默认使用一些规则集检查代码，但也可以自行调用其自带的不同规则集进行检查，命令如下：

> phpmd /path/to/source text codesize,unusedcode,naming

unusedcode,naming即为PHPMD cli中不同的规则集，以逗号分开。

目前有的规则集包括：codesize,unusedcode,naming,cleancode,controversial,design共6类。

（2）使用自定义规则集

> phpmd /path/to/source text codesize,/my/rules.xml

/my/rules.xml为自定义规则集路径。

自定义规则集方法见http://phpmd.org/documentation/creating-a-ruleset.html。

10.3 windows下安装代码静态分析工具php CodeSniffer

一、安装

1.安装pear

通常，php安装包内有pear的安装包，但默认不安装。安装方法：php pear.bat（前面加文件目录）。

2.安装php codesniffer

（1）设置pear为环境变量。

（2）执行命令：pear install PHP_codesniffer即可。

二、使用

1.使用

安装codesniffer后，即可使用phpcs命令。

检查某个文件，直接执行命令：phpcs 文件名/目录。

2.常用参数

--help 输出帮助

-i 输出已安装的检测规则集

--report=<report> 输出文件格式，有xml，json等格式可选

--report-file=<reportfile> 输出文件存放地址

-s 只看错误的总结信息

2.过滤部分规则集

默认情况下，PHP_CodeSniffer将按安装的所有标准进行检查，可能达数万条错误，有如下方法可以简化：

（1）制定规则集检查

> phpcs --standard=PEAR --sniffs=Generic.PHP.LowerCaseConstant,PEAR.WhiteSpace.ScopeIndent /path/to/code

上面的命令表示：只使用PEAR标准集中的Generic.PHP.LowerCaseConstant和PEAR.WhiteSpace.ScopeIndent两个参考集合来检查

不同规则集用“，”分开。

（2）按错误和警告的严重程度进行过滤

默认情况下,PHP_CodeSniffer按5个等级分类所有错误和警告，可设置如下：

> phpcs --severity=3 /path/to/code

来过滤严重等级低于3级的错误。

11. 附录

列出本文档的附件资料（可裁剪）