

gclxry

更吹落，星如雨

管理Chromium源代码的利器——depot_tools

由于Chromium项目的代码量巨大，又依赖了很多第三方代码库，所以如何有效的管理这些代码是个难题。

Chromium官方提供了一个depot_tools来管理Chromium源代码的工具，官方开发工作流也是基于depot_tools。最初接触depot_tools觉得它很难用，最近研究了depot_tools的代码，发现掌握了它能够让我们事半功倍。

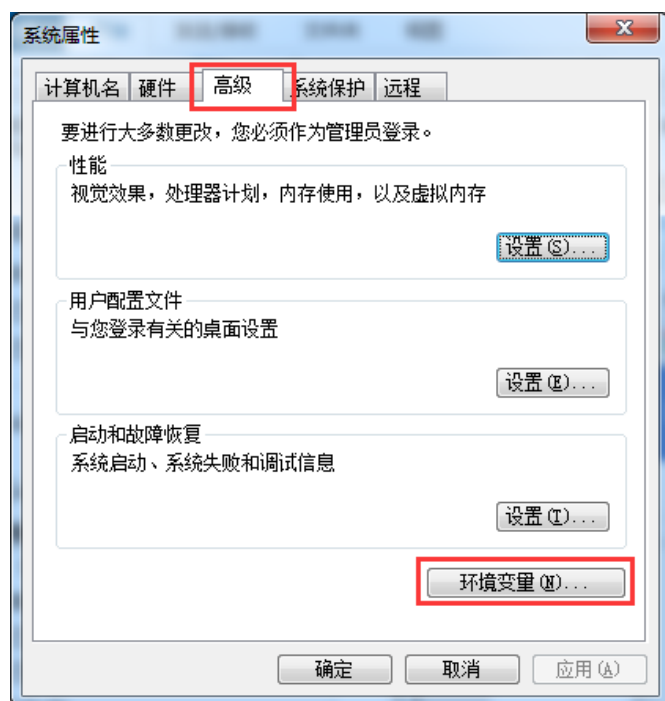
配置depot_tools

因为我是在Windows上开发Chromium，所以只介绍Windows上配置depot_tools的流程。

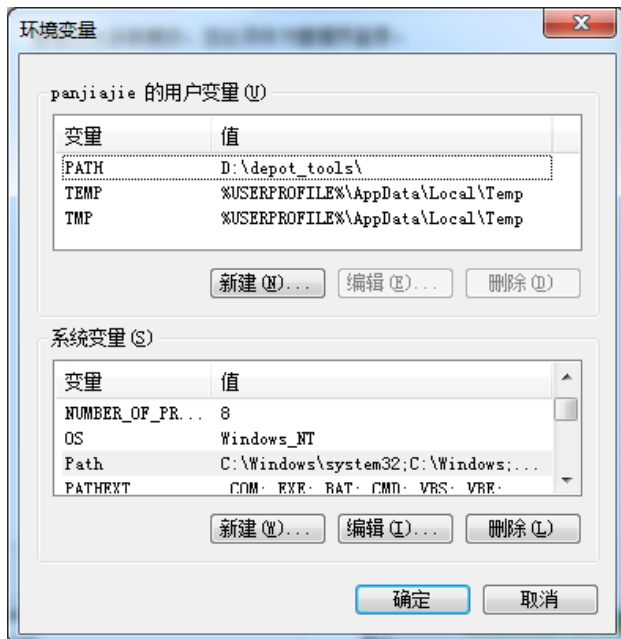
根据https://commondatastorage.googleapis.com/chrome-infra-docs/flat/depot_tools/docs/html/depot_tools_tutorial.html#_setting_up的介绍，从这里

https://src.chromium.org/svn/trunk/tools/depot_tools.zip 下载depot_tools的压缩包。然后把这个压缩包解压到本地的一个路径，如：F:depot_tools。

接下来就是把depot_tools所在的目录F:depot_tools加入到系统的PATH环境变量中。在 控制面板->系统->高级系统设置，弹出如下界面：

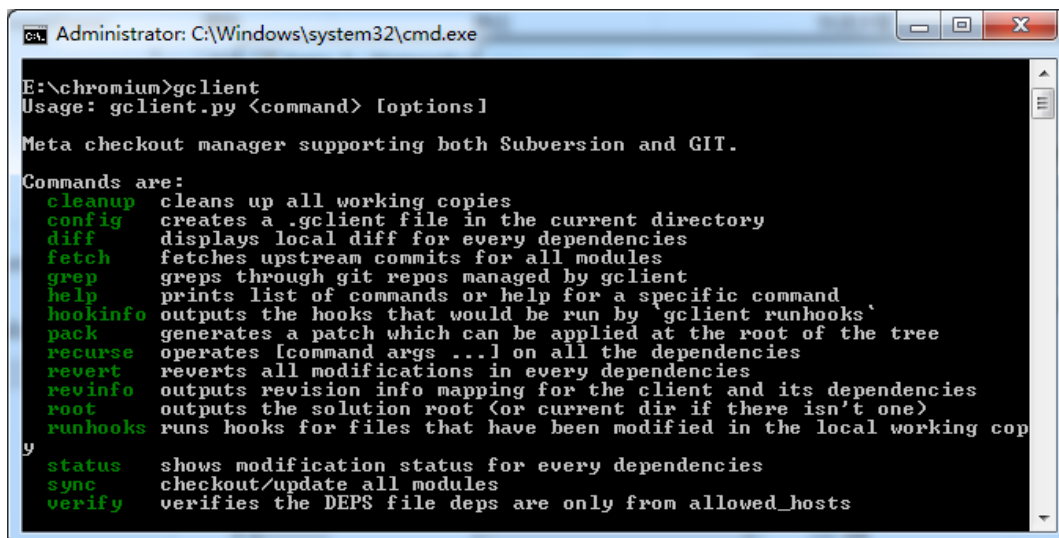


然后点击“环境变量”按钮，弹出如下对话框：



在这里面添加PATH变量，值为depot_tools的路径。

然后在任意路径里打开CMD，然后运行gclient命令，如果可以正常运行，则代表depot_tools配置成功了，如下图：



获取Chromium代码

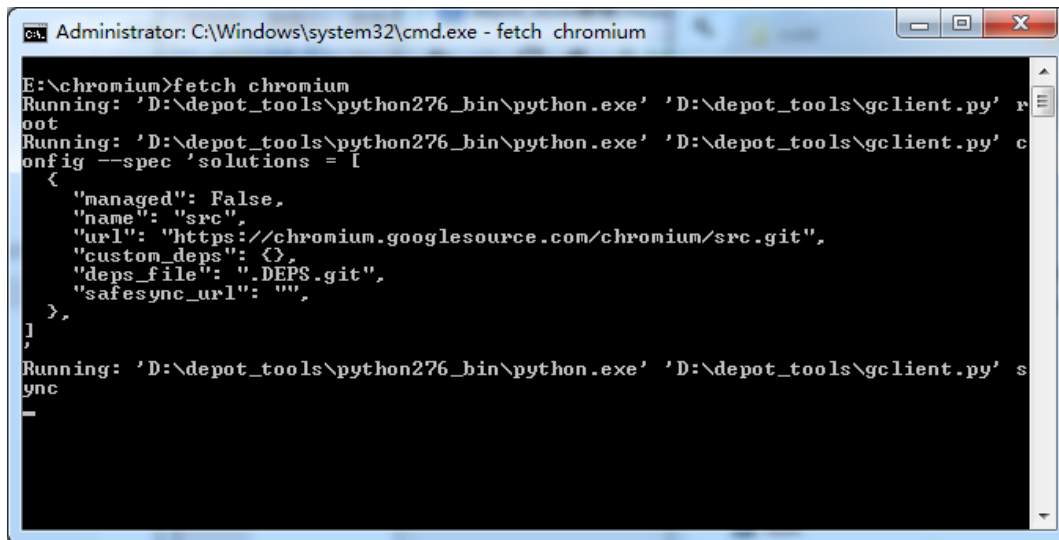
根据Chromium的文档<http://dev.chromium.org/developers/how-tos/get-the-code> 介绍，使用depot_tools获取代码非常简单。

运行`fetch --nohooks chromium`命令来获取Chromium以及它依赖的第三方库代码。然后再运行`gclient runhooks`命令来配置一些开发环境和生成可以编译的工程。

以后就可以使用`git rebase-update`命令来同步最新的Chromium代码，然后再使用`gclient sync`同步第三库代码。

fetch命令

在“获取Chromium代码”章节里，`fetch`命令是用来获取Chromium代码，运行`fetch`命令：



```
Administrator: C:\Windows\system32\cmd.exe - fetch chromium

E:\chromium>fetch chromium
Running: 'D:\depot_tools\python276_bin\python.exe' 'D:\depot_tools\gclient.py' r
oot
Running: 'D:\depot_tools\python276_bin\python.exe' 'D:\depot_tools\gclient.py' c
onfig --spec 'solutions = [
  {
    "managed": False,
    "name": "src",
    "url": "https://chromium.googlesource.com/chromium/src.git",
    "custom_deps": {},
    "deps_file": ".DEPS.git",
    "safesync_url": "",
  },
]
'
Running: 'D:\depot_tools\python276_bin\python.exe' 'D:\depot_tools\gclient.py' s
ync
```

我们可以看到在当前目录生成了一个.gclient的文件，内容如下：

```
1 solutions = [
2   {
3     "managed": False,
4     "name": "src",
5     "url": "https://chromium.googlesource.com/chromium/src.git",
6     "custom_deps": {},
7     "deps_file": ".DEPS.git",
8     "safesync_url": "",
9   },
10 ]
```

fetch命令其实调用的是depot_toolsfetch.py脚本，脚本的用法如下：

```
1 fetch <config> [--nohooks] [--no-history]
```

config的值是一个配置名，其实也是调用depot_toolsfetch_configs目录下的py脚本文件。不同的配置名获取不同的项目代码。我们是获取Chromium代码，所以用的是fetch chromium，即运行的是depot_toolsfetch_configschromium.py脚本里面的逻辑。chromium.py就是生成.gclient文件里的内容。

另外fetch命令还有2个可选的参数：

- **-nohooks**。这个参数表示获取代码完成之后不执行runhooks动作。也就仅仅获取代码。
- **-no-history**。这个参数表示对代码仓库执行git shallow clones，就不会获得原仓库的全部历史提交，这样可以减少拷贝代码仓库的大小。按照Chromium文档的介绍，不加上这个参数大概会获取22GB大小的数据，而加上这个参数只会获取6.5GB大小的数据。之后可以对这个仓库再执行-unshallow操作 就会获得完整的历史记录。

可以看到fetch命令最后调用的是gclient sync来获取代码的。接下来介绍gclient脚本。

gclient命令

gclient命令其实对应着gclient.py脚本，它是用来管理多个模块源代码仓库的工具。它封装一些常用的git命令，对所有的模块生效。可以看到gclient的功能众多，如下总结一下：

- **config**。创建一个.gclient配置文件。
- **diff**。类似git的diff命令，用来比较所有模块提交代码的差异。
- **fetch**。获取所有模块上游的提交。
- **help**。显示命令的帮助。

- **revert**。revert一个提交。
- **runhooks**。根据DEPS文件的描述执行hook任务。
- **stauts**。类似git status命令，用来显示所有模块代码的状态。
- **sync**。用来同步所有模块的代码。

sync和runhooks命令比较重要，以下分别介绍一下。

sync

sync就是用来同步所有模块的代码。当我们把Chromium仓库的代码切换到某个版本，则必须也要运行gclient sync来同步对应依赖的其他仓库代码，否则就有可能代码编译失败。

运行gclient sync的时候，会读取DEPS文件中描述的依赖的其他库的url和版本，然后把这些仓库切换到正确的版本。

下面介绍一些sync的参数：

- **-n, --nohooks**。检出代码之后不运行hooks里面的动作。
- **-r, --revision**。强制切换到某个代码的某个版本，后面可以使tag名如gclient sync -r 48.0.2564.74。或者提交id，如gclient sync -r f194a61f5ecd56f744273318100300045586d3dc。
- **--with_branch_heads**。Git在Clone代码仓库时也获取到每个分支的head。
- **--with_tags**。Git在Clone代码仓库时也获取tags。
- **-R, --reset**。重置本地的修改。
- **-M, --merge**。合并上游的修改。
- **-A, --auto_rebase**。等同于git pull --rebase。
- **--upstream**。让本地仓库状态匹配上游分支。
- **--no-history**。为了减小仓库大小，不检出提交历史。
- **--shallow**。对换成目录进行浅拷贝。

runhooks

runhooks是在同步完代码之后执行的。根据DEPS的文件的描述，执行一些获取代码之后的工作，其中包括生成平台可编译的工程。至于执行了哪些任务，可以参考DEPS文件里面的hooks内容。

runhooks的最后调用src/build/gyp_chromium.py脚本来生成可编译的代码工程。所以如果改变了gyp文件，可以仅运行src/build/gyp_chromium.py脚本来重新生成工程。

 gclxry / 一月 11, 2016 / 代不该代的码 / chromium、depot_tools、gclient、runhooks、代码

《管理Chromium源代码的利器——depot_tools》有1个想法
