1安装git,按照默认的设置安装。安装后有Git Bash、Git CMD 、Git GUI三个选项，选择习惯的命令窗口。

2、到Github注册账号和密码。

[ssh-keygen -t rsa -C "](http://www.cnblogs.com/schaepher/p/5561193.html#ssh_key)你的邮箱" 建设得到SSH key

ssh -T git@git.bonjour.tools测试连接

[**Github简单教程**](http://www.cnblogs.com/schaepher/p/5561193.html)

原文链接：[Git和Github简单教程](http://www.cnblogs.com/schaepher/p/5561193.html)

网络上关于Git和GitHub的教程不少，但是这些教程有的命令太少不够用，有的命令太多，使得初期学习的时候需要额外花不少时间在一些当前用不到的命令上。  
这篇文章主要的目标是用较少的时间学习Git和GitHub的基本使用。在足够一般使用的前提下，尽量减少命令。  
如果需要其他命令，到时候再去其他地方了解就行了。

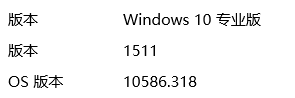
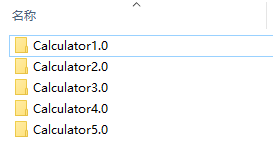
目录：

* [零、](http://www.cnblogs.com/schaepher/p/5561193.html#what)Git是什么
* [一、](http://www.cnblogs.com/schaepher/p/5561193.html#function)Git的主要功能：版本控制
* [二、概览](http://www.cnblogs.com/schaepher/p/5561193.html#overview)
* [三、](http://www.cnblogs.com/schaepher/p/5561193.html#msysgit)Git for Windows软件安装
* [四、本地](http://www.cnblogs.com/schaepher/p/5561193.html#local)Git的使用
* [五、](http://www.cnblogs.com/schaepher/p/5561193.html#github)Github与Git的关联
* [六、团队合作开发](http://www.cnblogs.com/schaepher/p/5561193.html#team)
* [七、](http://www.cnblogs.com/schaepher/p/5561193.html#others)Github的其他介绍
* [八、一些可能碰到的问题](http://www.cnblogs.com/schaepher/p/5561193.html#problem)

**零、Git是什么**

我才不告诉你嘞

**一、Git的主要功能：版本控制**

1. 版本：  
   想想你平时用的软件，在软件升级之后，你用的就是新版本的软件。你应该见过这样的版本号：v2.0 或者 1511（表示发布时为15年11月），如下图：  
     
   那么如果你修改并保存了一个文件，从版本管理的角度来说，你得到的是这个文件的新版本。  
   可是很多情况下，这种修改是不可逆的。你修改完之后，无法回到你修改前的样子。为了避免这种情况，有的人会把新版本的内容保存到一个新的文件里面。  
   由于 Git 更多地用于代码管理，举个程序员的例子。比如以下是计算机专业学生的作业：  
     
   这样存储多个文件夹，可能会造成混乱。你可能想保存以前写的代码，因为它们可能在以后会用到。但是更多的时候是，你不知道各个文件夹都做了什么修改。  
   这时候你需要一款软件帮你管理版本，它就是Git。
2. 控制：  
   你可以用Git来对这些不同的版本进行控制。还可以很方便地查看两个不同版本之间的不同之处。  
   ——使用Git，你只保存最新的一份文件就可以了。  
   ——那我以前的文件怎么办？  
   ——可以用Git的 reset 帮你把文件回退到你想要的版本。  
   ——如果回去了，那我的最新版本呢？  
   ——还可以用 reflog 和 reset 的组合来还原。

**二、概览**

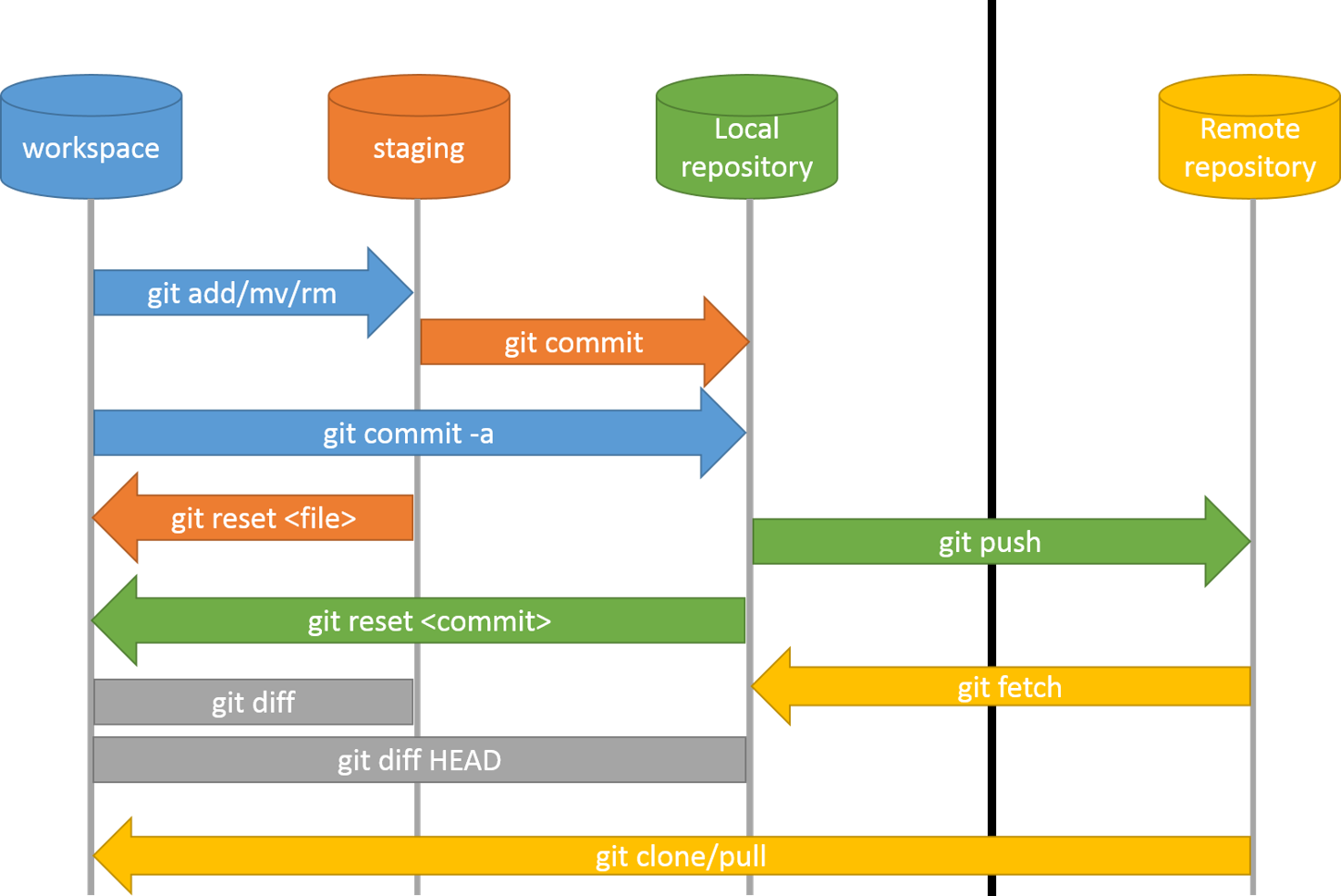
所有命令前都要加 git，如表中的init是指 git init。  
点击命令可直接跳转至本文第一次使用的地方。  
以下命令都在命令行里执行。

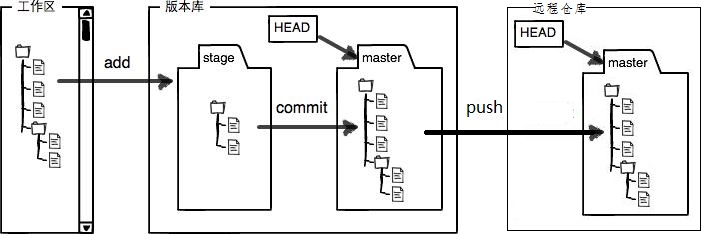
**1.个人本地使用**

| **行为** | **命令** | **备注** |
| --- | --- | --- |
| 初始化 | [init](http://www.cnblogs.com/schaepher/p/5561193.html#init) | 在本地的当前目录里初始化git仓库 |
|  | [clone](http://www.cnblogs.com/schaepher/p/5561193.html#clone) 地址 | 从网络上某个地址拷贝仓库(repository)到本地 |
| 查看当前状态 | [status](http://www.cnblogs.com/schaepher/p/5561193.html#status) | 查看当前仓库的状态。碰到问题不知道怎么办的时候，可以通过看它给出的提示来解决问题 |
| 查看不同 | [diff](http://www.cnblogs.com/schaepher/p/5561193.html#diff) | 查看当前状态和最新的commit之间不同的地方 |
|  | diff 版本号1 版本号2 | 查看两个指定的版本之间不同的地方。这里的版本号指的是commit的hash值 |
| 添加文件 | [add -A](http://www.cnblogs.com/schaepher/p/5561193.html#add) | 这算是相当通用的了。在commit之前要先add |
| 撤回stage的东西 | [checkout -- .](http://www.cnblogs.com/schaepher/p/5561193.html#checkout) | 这里用小数点表示撤回所有修改，在--的前后都有空格 |
| 提交 | [commit -m "](http://www.cnblogs.com/schaepher/p/5561193.html#commit)提交信息" | 提交信息最好能体现更改了什么 |
| 删除未tracked | [clean -xf](http://www.cnblogs.com/schaepher/p/5561193.html#clean) | 删除当前目录下所有没有track过的文件。不管它是否是.gitignore文件里面指定的文件夹和文件 |
| 查看提交记录 | [log](http://www.cnblogs.com/schaepher/p/5561193.html#log) | 查看当前版本及之前的commit记录 |
|  | [reflog](http://www.cnblogs.com/schaepher/p/5561193.html#reflog) | HEAD的变更记录 |
| 版本回退 | [reset --hard](http://www.cnblogs.com/schaepher/p/5561193.html#reset) 版本号 | 回退到指定版本号的版本，该版本之后的修改都被删除。同时也是通过这个命令回到最新版本。需要reflog配合 |

**2.个人使用远程仓库**

| **行为** | **命令** | **备注** |
| --- | --- | --- |
| 设置用户名 | [config --global user.name "](http://www.cnblogs.com/schaepher/p/5561193.html#config_user)你的用户名" |  |
| 设置邮箱 | [config --global user.email "](http://www.cnblogs.com/schaepher/p/5561193.html#config_email)你的邮箱" |  |
| 生成ssh key | [ssh-keygen -t rsa -C "](http://www.cnblogs.com/schaepher/p/5561193.html#ssh_key)你的邮箱" | 这条命令前面不用加git |
| 添加远程仓库 | [remote add origin](http://www.cnblogs.com/schaepher/p/5561193.html#remote_add) 你复制的地址 | 设置origin |
| 上传并指定默认 | [push -u origin master](http://www.cnblogs.com/schaepher/p/5561193.html#push_origin) | 指定origin为默认主机，以后push默认上传到origin上 |
| 提交到远程仓库 | [push](http://www.cnblogs.com/schaepher/p/5561193.html#push) | 将当前分支增加的commit提交到远程仓库 |
| 从远程仓库同步 | pull | 在本地版本低于远程仓库版本的时候，获取远程仓库的commit |

可以用一张图直观地看出以上主要的命令对仓库的影响。  
  
图片引用自：[Git introduction for CVS/SVN/TFS users](http://blog.podrezo.com/git-introduction-for-cvssvntfs-users/)

  
图片引用自：[工作区和暂存区 - 廖雪峰的官方网站](http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013745374151782eb658c5a5ca454eaa451661275886c6000) （做了点修改）

对照查看两张图：

* workspace 即工作区，逻辑上是本地计算机，还没添加到repository的状态；
* staging 即版本库中的stage，是暂存区。修改已经添加进repository，但还没有作为commit提交，类似于缓存；
* Local repository 即版本库中master那个地方。到这一步才算是成功生成一个新版本；
* Remote repository 则是远程仓库。用来将本地仓库上传到网络，可以用于备份、共享、合作。本文将使用Github作为远程仓库的例子。

**三、Git for Windows软件安装**

  
Git for Windows（又msysgit）

* 安装包可以到官方网站[[1]](https://git-for-windows.github.io/)下载，或者在github[[2]](https://github.com/git-for-windows/git/releases)下载。如果下载不下来，可以把链接复制下来用迅雷下载。如果用迅雷下载不放心，在下载完后去在github下载的那个地方查看SHA-256值，并和下载的文件对比，如果值一样就可以放心使用。
* 安装的时候一路点击Next就行了。
* 刚安装完打开后，窗口比较小。如果不太习惯，可以把它改大一些。

1. 首先移到窗口右下角边缘，出现箭头后把窗口拉大。
2. 点击窗口顶部左边的图标 -> Options... -> Window -> Current size -> OK  
   这样以后打开窗口都会是调整后的大小。

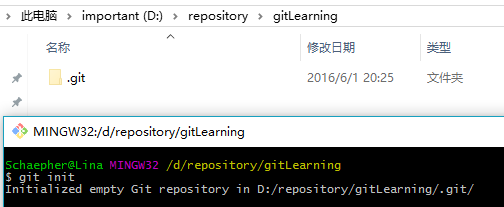
Git for Windows从2.8.0版本[[3]](https://github.com/git-for-windows/git/releases/tag/v2.8.0.windows.1)开始，默认添加环境变量，所以环境变量部分就不用再手动配置了。（这句可以无视）

**四、本地Git的使用**

这里先不引入Github，而是在本地计算机上的操作。

打开命令行（cmd）或者在想要创建repository的地方右键鼠标并点击 Git Bash Here 打开窗口。

**1.新的仓库-》初始化**

运行 git init 来初始化仓库，如下图：  


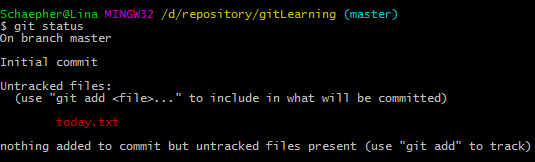
它会创建一个隐藏的文件夹 .git 这里不去管它是用来干嘛的。关闭windows的显示隐藏的项目吧。

**2.文件的添加和提交**

我在这个文件夹里面创建了一个 today.txt 的文件。并且不要脸地祝自己儿童节快乐：

http://images2015.cnblogs.com/blog/809218/201606/809218-20160604213956149-723619418.png

这时我使用 git status 来查看有什么变化：

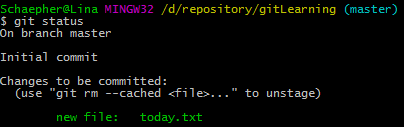


它告诉我有一个还未追踪的文件，并提示我可以使用 git add <file>... 把它加进去。

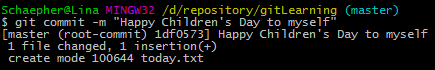
但是我并不打算把所有命令都介绍一遍，所以我选择使用上面概览时所提到的 git add -A 命令。

http://images2015.cnblogs.com/blog/809218/201606/809218-20160604214016508-253679176.png

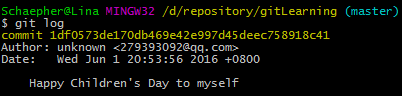
嗯，什么提示都没有。没关系，我们再次使用 git status ：



状态变了！说明add成功。再看看它的提示 Changes to be committed ，也就是说现在可以执行commit了。下面一行则告诉你如何将文件从stage里移出，这里不管。  
执行 git commit -m "提交信息" 将文件提交到repository里。提交信息用英文的双引号括起来。



这时运行 git log 就可以看到提交的记录了：

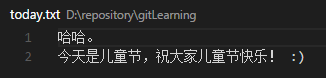


这样第一步就完成了。

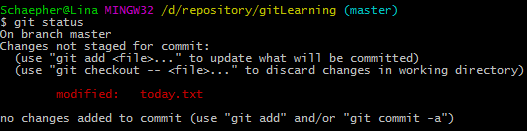
也许你会奇怪：为什么要有一个add，直接commit不就行了？这是因为stage有很多用处，具体可以去查找相关资料。这里就算不了解问题也不大。

**3.文件的修改**

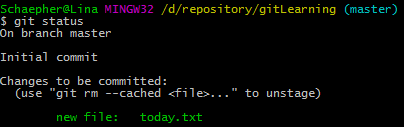
接着我修改文件内容。改成祝大家儿童节快乐好了 (～￣▽￣)～



我们用 git status 看看有什么变化：

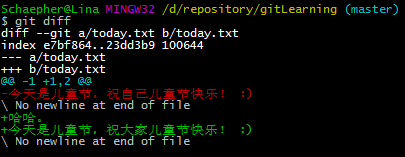


这和之前的提示不一样了。之前是这个：



比较一下就会看到，之前的是添加新文件，当时文件还没被追踪（untracked），而这次是更改已经追踪（tracked）的文件。

现在我们通过git看看文件做了哪些变化，执行 git diff ：



它默认跟最新的一个commit进行比较。  
红色（前面有减号-）表示删除，绿色（前面有加号+）表示添加。  
因此，在git看来，我们是删除了原来那一行，并添加了新的两行。这在文件内容特别多的时候效果比较明显。  
这个命令在以下情况可以使用：

* 你忘记改了什么，又想知道
* 别人发给你新版本，你想知道更改了什么

注：如果你用 windows 创建 txt 文件，并用自带文本编辑器来编辑文本，得到的编码是 GBK 。而 Git 读取文件时，使用 UTF-8 无 ROM 编码。因此会出现中文无法正常显示的情况。

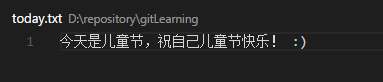
假如我现在想撤销这些更改，执行 git checkout -- . 就行了：

http://images2015.cnblogs.com/blog/809218/201606/809218-20160604214206821-1921800298.png

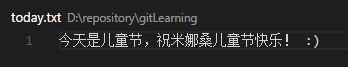
恩，仍然没有任何提示。执行 git status 看看：

http://images2015.cnblogs.com/blog/809218/201606/809218-20160604214215774-1596938603.png

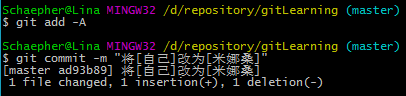
上一个status的提示已经不见咯。再来看看文件：



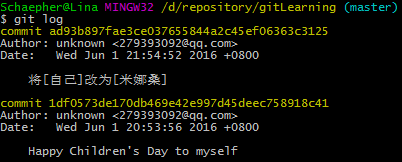
果然复原了！那么再次进行修改：



接着：  
git add -A  
git commit -m "将[自己]改为[米娜桑]"



用 git log 看看提交（commit）记录：

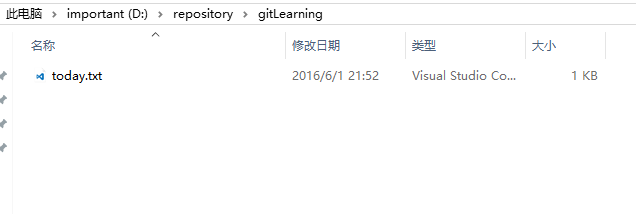


嗯。现在有两个提交了。

**4.版本回退！**

如果我写的是一篇很长的文章，并且在之前的版本基础上修改了一部分内容，生成一个新的commit，现在我发现我在修改的时候删掉了一部分内容，而这部分内容是我现在需要用到的，怎么办？版本回退！

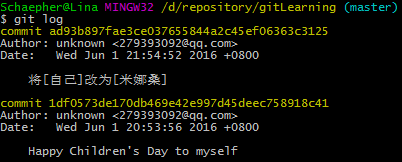
还是以刚才的文件为例，现在我试着将文件回退到第一个commit时的状态。但在这之前，我们看看这个文件夹里面的东西：



.git 文件夹因为本来就是隐藏的，我在关了 显示隐藏的项目 的选项后，它就不显示了。  
现在看到的是只有一个文件，而且是最新的一个版本。

文件的修改日期为 2016\6\1 21:52

从刚才的 git log ：

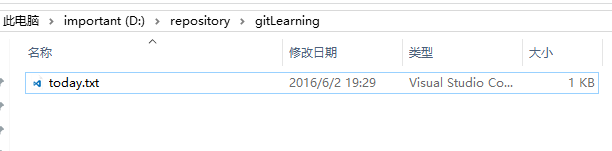


我们看到两行黄色部分是以 commit 开头的，后面接着一串字符。这一串字符是16进制的数，是一串哈希值。我们叫它版本号就行了。

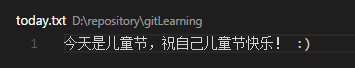
开始回退，执行 git reset --hard 1df0573 （取版本号前7位就可以了）：

http://images2015.cnblogs.com/blog/809218/201606/809218-20160604215412196-1252165674.png

这里提示HEAD已经更改指向至1df0573了。此时文件：



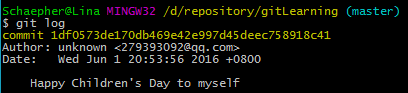
其内容：



已经回到我的第一个版本的状态。

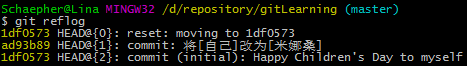
这里文件的修改日期被更改为我现在的时间 2016\6\2 19:29  
这是由于文件的修改日期是由windows修改的，因为它检测到这个文件被修改了。而我们刚才从最新版本回退到现在这个版本，就像是我们手动修改了文件内容一样，事实上是由git来完成的。  
其实可以不管上面这一段

现在再执行 git log ：



新版本的commit记录不见了！这就是 reset --hard 的力量，很好很强硬！

现在已经看到了之前版本的内容，那么如何回到最新版呢？  
先执行 git reflog ：

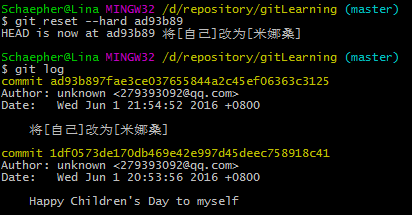


可以看到HEAD的变化情况。  
第一行表示当前HEAD所在的版本号是 1df0573 ，而之所以在这个版本号，是由于我们执行了reset命令。  
看第二行，它告诉我们，这个HEAD所在的版本号是 ad93b89 ，这个版本号是在执行commit之后形成的。

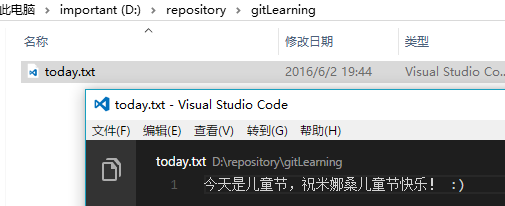
此时我再用一次reset，将HEAD指向 ad93b89 ， 同时查看log ：

git reset --hard ad93b89

git log



回到第一次reset前的状态了！



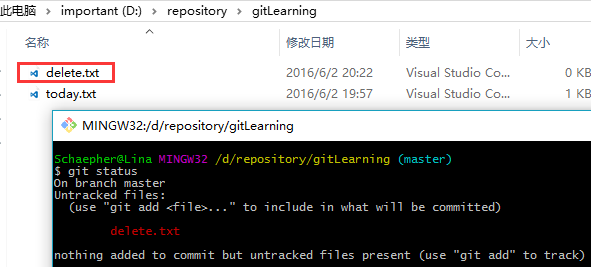
**5.清除未追踪的文件**

通常在reset或者pull（后面会讲）之前要做两件事：

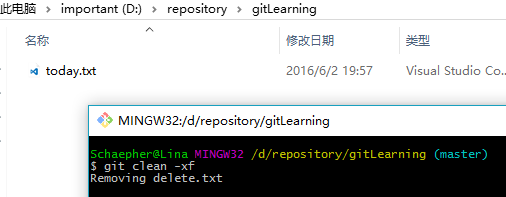
* 将新添加且为追踪的文件删除掉（比如编译程序后所产生的文件）
* 已追踪的文件已有修改，但又不需要这些修改，则将它们还原

还原已做修改的tracked文件，上面已经讲过。  
现在看看如何用命令删除新加的文件。

首先我手动创建个文件，用来演示：



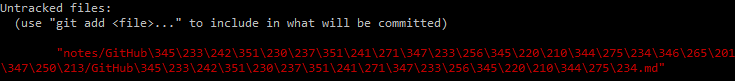
用checkout是没办法删除掉它的，使用 git clean -xf ：



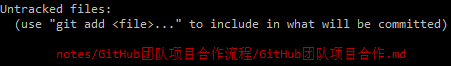
这个命令的杀伤力比较大，它删除当前目录下所有没有track过的文件。不管它是否是.gitignore文件里面指定的文件夹和文件。当然，也有杀伤力比较小的，但这里就不介绍了。

**6.关于git status中文乱码问题**

如果你的文件名是中文的，在使用git status时会乱码。如下图所示：



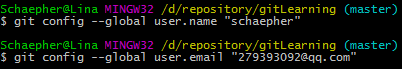
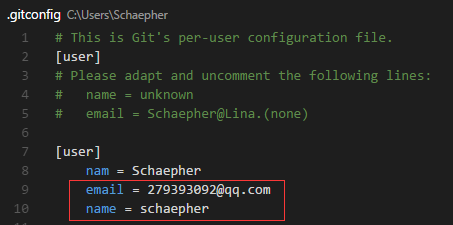
如果要使它显示为中文，在命令行里执行：git config --global core.quotepath false。  
再使用 git status：

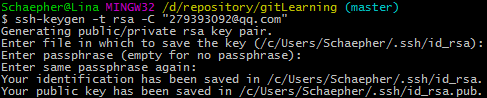
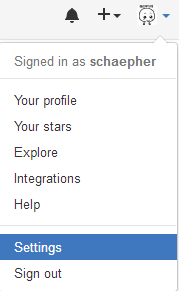
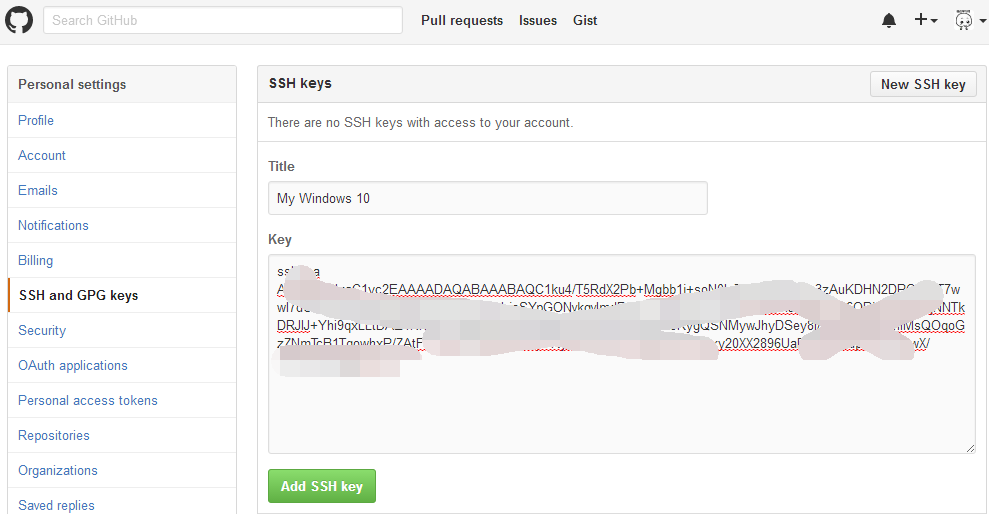
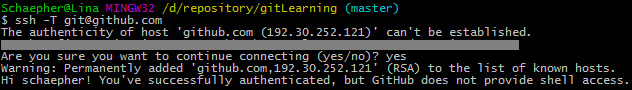


**五、Github与Git的关联**

上面的操作都是在本地计算机上产生影响的，一般也够用了。  
如果你是程序员，想和其他人分享你的代码，或者合作开发，可以用Github。

**1）本地Git和Github的连接**

1. 到Github[[4]](https://github.com/join?source=header)注册账号。
2. 本地配置用户名和邮箱（如果已经设置好，跳过该步）：
   * git config --global user.name "你的用户名"  
     git config --global user.email "你的邮箱"  
     如图所示：  
     
   * 或者你直接在config文件里改，位置在 C:\Users\你的用户名\.gitconfig 。如下图所示，添加相应信息：  
     

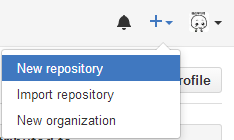
* 生成ssh key  
  运行 ssh-keygen -t rsa -C "你的邮箱" ，它会有三次等待你输入，直接回车即可。  
    
  将生成的ssh key复制到剪贴板，执行 clip < ~/.ssh/id\_rsa.pub （或者到上图提示的路径里去打开文件并复制）：  
  http://images2015.cnblogs.com/blog/809218/201606/809218-20160604214651414-743953505.png
* 打开Github，进入Settings：  
    
  点击左边的 SSH and GPG keys ，将ssh key粘贴到右边的Key里面。Title随便命名即可。  
    
  点击下面的 Add SSH key 就添加成功了。  
  测试一下吧，执行 ssh -T git@github.com ：  
    
  嗯，这样就成功了！

注：

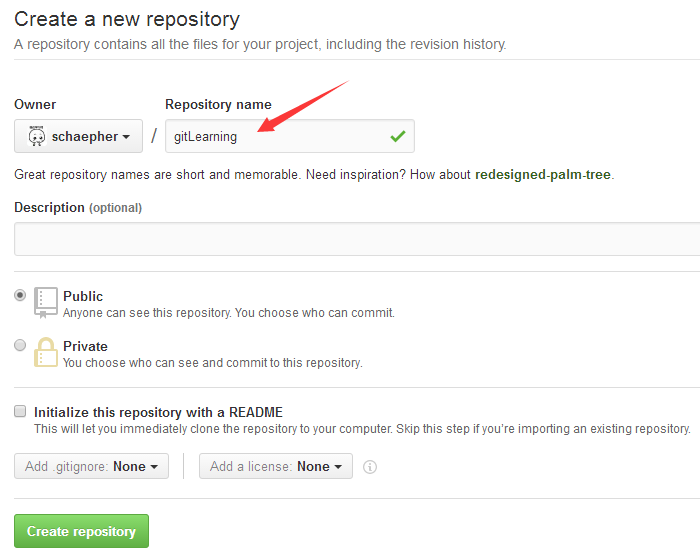
* 对于 oschina 的 “码云” ，执行 ssh -T git@git.oschina.net
* 对于 coding 的 “码市” ，执行 ssh -T git@git.coding.net

**2）创建远程仓库并与本地关联**

1. 创建远程仓库  
   首先是在右上角点击进入创建界面：



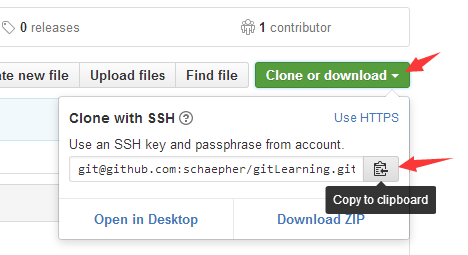
接着输入远程仓库名：



点击 Create repository 就创建好了。其他选项可以暂时不管。

1. 将远程仓库和本地仓库关联起来

先到Github上复制远程仓库的SSH地址：



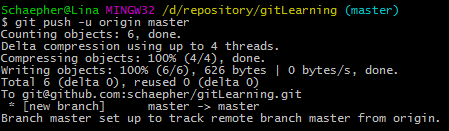
有两种方式可以关联，一种是SSH，一种是HTTPS。由于**HTTPS比较慢**，所以推荐使用SSH。  
注意SSH的地址格式是这样开头的： git@github.com

运行 git remote add origin 你复制的地址 ：

http://images2015.cnblogs.com/blog/809218/201606/809218-20160604214829571-1197635501.png

如果你在创建 repository 的时候，加入了 README.md 或者 LICENSE ，那么 github 会拒绝你的 push 。你需要先执行 git pull origin master。

执行 git push -u origin master 将本地仓库上传至Github的仓库并进行关联：

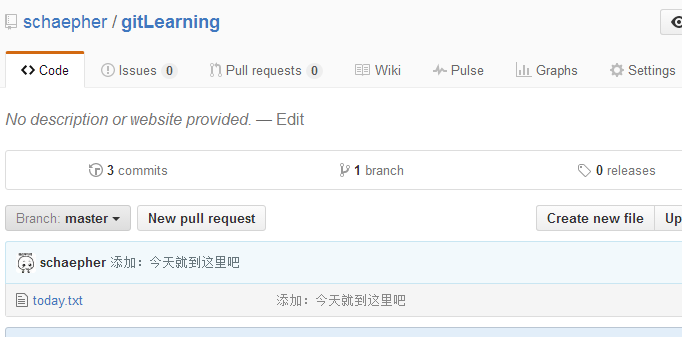


关联已经完成！

以后想在commit后同步到Github上，只要直接执行 git push 就行啦：



可以在Github上看到修改：



**六、团队合作开发**

关于团队合作开发，我在之前已经专门用一篇文章来说明了。  
原文链接： [GitHub团队项目合作流程](http://www.cnblogs.com/schaepher/p/4933873.html)

上文的目录：

* [零、前期准备](http://www.cnblogs.com/schaepher/p/4933873.html#first)
* [一、创建开发分支](http://www.cnblogs.com/schaepher/p/4933873.html#second)
* [二、](http://www.cnblogs.com/schaepher/p/4933873.html#third)Fork项目到个人的仓库
* [三、](http://www.cnblogs.com/schaepher/p/4933873.html#four)Clone项目到本地
* [四、和团队项目保持同步](http://www.cnblogs.com/schaepher/p/4933873.html#five)
* [五、](http://www.cnblogs.com/schaepher/p/4933873.html#six)push修改到自己的项目上
* [六、请求合并到团队项目上](http://www.cnblogs.com/schaepher/p/4933873.html#seven)
* [七、团队项目负责人审核及同意合并请求](http://www.cnblogs.com/schaepher/p/4933873.html#eight)

其中 零、一、七 是由团队项目负责人来完成的。

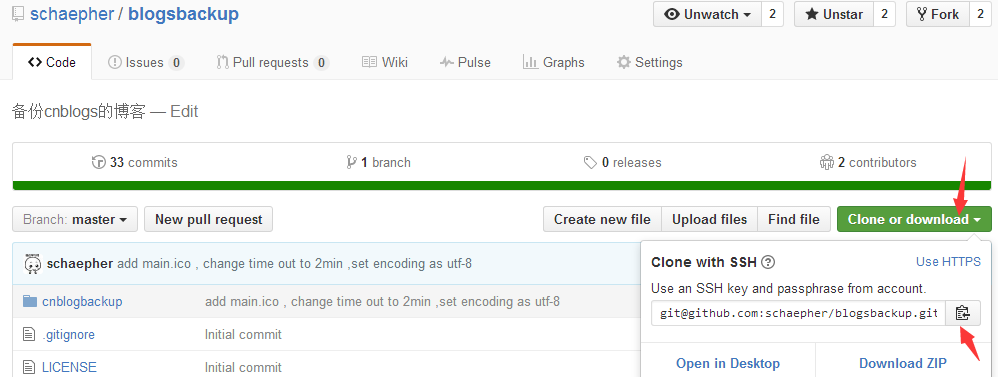
**七、Github的其他介绍**

**1）获取其他人的远程仓库**

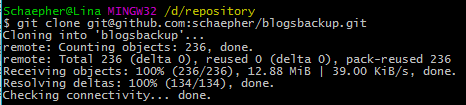
看到别人的代码，想要获取到本地计算机慢慢研究或者修改，可以用Git将其下载下来。  
以我和一位同学合作的仓库为例，链接：[schaepher/blogsbackup](https://github.com/schaepher/blogsbackup)

这是我们用来备份博客园博客的一个小程序，主要用于助教备份学生的博客。

1. 复制别人远程仓库的SSH。方法和上面关联git一样，进入仓库页面后，就能找到SSH地址。



1. 执行 git clone 复制的SSH地址



整个项目42M，花了点时间才clone下来。

有一点要注意：这样直接clone别人的仓库后，不能push上自己的修改。

**2）另一种关联远程空仓库的方法**

在知道了clone之后，你就可以更简单的创建并关联一个空仓库了。

1. 在Github上创建仓库，上面有提到，这里不多讲
2. 复制刚才创建的仓库的SSH
3. clone到本地

这样可以不用再做关联了。省去了上面的 init 和 git remote add origin 以及 git push -u origin master 。

**3）使用GitHub的好处**

* 全球最大的同♂性交友网站（逃
* （严肃脸）很多牛逼程序员和牛逼公司的开源项目都放在这上面，有丰富的资源可以学习
* 别人（如HR）可以通过你的Github大致了解你的水平。《[怎样花两年时间去面试一个人](http://mindhacks.cn/2011/11/04/how-to-interview-a-person-for-two-years/)》 这篇文章的后面部分讲到了Github

**八、一些可能碰到的问题**

这篇文章基本只介绍主线操作，而在操作过程中，可能有误操作或者其他问题。我把这些问题集中放在另一篇博客里面（这里列出目录）。这样这篇文章不会显得太长。

* [一、 解决](http://www.cnblogs.com/schaepher/p/4970291.html#conflict)merge时出现的冲突
* [二、 回退一个](http://www.cnblogs.com/schaepher/p/4970291.html#back)merge
* [三、 获取某一](http://www.cnblogs.com/schaepher/p/4970291.html#pick)commit的修改
* [四、 将低版本](http://www.cnblogs.com/schaepher/p/4970291.html#push_force)push到Github（删掉高版本Commit）

Git文件的三种状态：已提交（committed）,已修改（modified），已暂存（staged）;由此引入了Git项目的三个工作区域的概念：Git仓库（.git directory(Repository)），工作目录（Working Directory），暂存区域（staging Area）。

git config命令用于获取并设置存储库或全局选项。这些变量可以控制Git的外观和操作的各个方面。

**语法简介**

git config [<file-option>] [type] [--show-origin] [-z|--null] name [value [value\_regex]]

git config [<file-option>] [type] --add name value

git config [<file-option>] [type] --replace-all name value [value\_regex]

git config [<file-option>] [type] [--show-origin] [-z|--null] --get name [value\_regex]

git config [<file-option>] [type] [--show-origin] [-z|--null] --get-all name [value\_regex]

git config [<file-option>] [type] [--show-origin] [-z|--null] [--name-only] --get-regexp name\_regex [value\_regex]

git config [<file-option>] [type] [-z|--null] --get-urlmatch name URL

git config [<file-option>] --unset name [value\_regex]

git config [<file-option>] --unset-all name [value\_regex]

git config [<file-option>] --rename-section old\_name new\_name

git config [<file-option>] --remove-section name

git config [<file-option>] [--show-origin] [-z|--null] [--name-only] -l | --list

git config [<file-option>] --get-color name [default]

git config [<file-option>] --get-colorbool name [stdout-is-tty]

git config [<file-option>] -e | --edit

Shell

**描述**

可以使用此命令查询/设置/替换/取消设置选项。该名称实际上是由点(.)分隔键，该值将被转义。

可以使用--add选项将多行添加到选项。如果要更新或取消设置多行可能出现的选项，则需要给出POSIX正则表达式value\_regex。 只有与正则表达式匹配的现有值已更新或未设置。如果要处理与正则表达式不匹配的行，只需在前面添加一个感叹号。

类型说明符可以是--int或--bool，以使git config确保变量是给定类型，并将该值转换为规范形式(int是简单十进制数，“true”或 “false” 的布尔字符串表示)，或者--path，它进行一些路径扩展。如果没有类型说明符传递，则不会对该值执行检查或转换。

读取时，默认情况下从系统，全局和存储库本地配置文件读取这些值，而选项--system，--global，--local和--file <filename>可用于告知命令从只有那个位置设置和读取。

写入时，默认情况下将新值写入存储库本地配置文件，并且可以使用选项--system，--global，--file <filename>来告诉命令写入该位置(可以指定为 --local，但这是默认值)。

此错误后，此命令将失败，并显示非零状态。 一些退出代码是：

* 部分或键无效(ret = 1)，
* 没有提供部分或名称(ret = 2)，
* 配置文件无效(ret = 3)，
* 配置文件无法写入(ret = 4)，
* 尝试取消设置不存在的选项(ret = 5)，
* 尝试取消设置/设置多个行匹配的选项(ret = 5)或
* 尝试使用无效的正则表达式(ret = 6)。

成功后，命令返回退出代码是：0 。

**一. 配置文件的存储位置**

这些变量可以被存储在三个不同的位置：

* /etc/gitconfig 文件：包含了适用于系统所有用户和所有库的值。如果你传递参数选项’—system’ 给 git config，它将明确的读和写这个文件。
* ~/.gitconfig 文件 ：具体到你的用户。你可以通过传递—global 选项使Git 读或写这个特定的文件。
* 位于git目录的config文件 (也就是 .git/config) ：无论当前在用的库是什么，特定指向该单一的库。每个级别重写前一个级别的值。因此，在.git/config中的值覆盖了在/etc/gitconfig中的同一个值。

**二.配置用户名和密码**

当安装Git后首先要做的事情是设置用户名称和e-mail地址。这是非常重要的，因为每次Git提交都会使用该信息。它被永远的嵌入到了你的提交中：

$ git config --global user.name "maxsu"

$ git config --global user.email "yiibai.com@gmail.com"

Shell

重申一遍，只需要做一次这个设置。如果传递了 --global 选项，因为Git将总是会使用该信息来处理在系统中所做的一切操作。如果希望在一个特定的项目中使用不同的名称或e-mail地址，可以在该项目中运行该命令而不要--global选项。

$ git config user.name "maxsu"

$ git config user.email "yiibai.com@gmail.com"

**Command line instructions**

**Git global setup**

git config --global user.name "whzhang"

git config --global user.email "whzhang@idthk.com"

**Create a new repository**

git clone git@git.bonjour.tools:zhangwh/android7.git

cd android7

touch README.md

git add README.md

git commit -m "add README"

git push -u origin master

**Existing folder**

cd existing\_folder

git init

git remote add origin git@git.bonjour.tools:zhangwh/android7.git

git add .

git commit

git push -u origin master

**Existing Git repository**

cd existing\_repo

git remote add origin git@git.bonjour.tools:zhangwh/android7.git

git push -u origin --all

git push -u origin --tags

git init

git add .

git remote add origin git@git.bonjour.tools:bonjour-android/lichee.git

git commit -m “initial commit”

git pull origin master // 同步远程版本

git branch idtDevelop //创建idtDevelop分支

git checkout idtDevelop // 使用idtDevelop分支

git push origin idtDevelop // 上传idtDevelop分支

git add –f ./keys // 强制上传keys目录

git commit –m “add secure keys dir” // 上传到本地仓库

git push --set-upstream origin idtDevelop // 上传到远程仓库

Git .gitignore 文件的使用; 忽略规则

在.git同级目录下创建.gitignore文件

touch .gitignore

gedit .gitignore

.gitignore文件：

#此行为注释，不会起作用

filename.txt

\*.html

!except.html

all

all/

/all

1. 注释

#此行为注释，不会起作用

以“#”开始的行，被视为注释

2. 完整文件名

filename.txt

忽略项目下所有名为filename.txt的文件

3. 文件后缀

\*.html

忽略html后缀的所有文件，其中星号（\*）为通配符，可代表多个字符，另外问号（？）代表一个字符，方括号（[abc]）代表可选字符范围

4. 例外文件

!except.html

前面加上感叹号，取消对except.html文件的忽略

5. 不带后缀的名称

all

忽略所有名称为all的文件和目录

6. 带后斜杠的名称

all/

忽略所有all目录

7. 带前斜杠的名称

/all

只忽略当前目录下的文件和目录，不涉及到子目录

以斜杠“/”开头表示目录；  
以星号“\*”通配多个字符；  
以问号“?”通配单个字符  
以方括号“[]”包含单个字符的匹配列表；  
以叹号“!”表示不忽略(跟踪)匹配到的文件或目录；

git和 repo

**window7下配置下载android源码环境,安装Repo**

安装git，参考<https://github.com/esrlabs/git-repo>

1.安装git（版本控制工具）

下载git：https://git-scm.com/download/win

双击exe文件，按提示一步一步地安装，然后将安装后的文件的cmd文件夹配置到环境变量中; 例如我安装后的cmd文件夹的路径D:\Java\Git\cmd ; 将MinGW添加到环境变量中

D:\Java\Git\bin;

2.安装pyton

下载pyton：https://www.python.org/downloads/（版本选择2.7+，否则会报异常ypeError: startswith first arg must be bytes or a tuple of bytes, not str）

将该路径配置到环境变量path中; C:\Python27;

3.下载Repo

在git bash中执行命令：

mkdir ~/bin

curl https://raw.githubusercontent.com/esrlabs/git-repo/stable/repo > ~/bin/repo

curl https://raw.githubusercontent.com/esrlabs/git-repo/stable/repo.cmd > ~/bin/repo.cmd

下载安装完后在C:\Users\whzhang\bin中有repo和repo.cmd

下载android sourceCode

使用本地镜像（）https://source.android.com/source/downloading

当您使用多个客户端时（尤其是在带宽不足的情况下），最好为所有服务器内容创建一个本地镜像，并从该镜像同步客户端（不需要访问网络）。一个完整镜像的下载文件比两个客户端的下载文件要小一些，而且包含更多信息。

以下说明假定在 /usr/local/aosp/mirror 中创建镜像。第一步是创建并同步镜像本身。请注意 --mirror 标志，该标志只能在创建新客户端时指定：

$ mkdir -p /usr/local/aosp/mirror

$ cd /usr/local/aosp/mirror

$ repo init -u https://android.googlesource.com/mirror/manifest --mirror

$ repo sync

同步镜像后，您就可以从镜像创建新客户端了。请注意，务必要指定一个绝对路径：

$ mkdir -p /usr/local/aosp/master

$ cd /usr/local/aosp/master

$ repo init -u /usr/local/aosp/mirror/platform/manifest.git

$ repo sync

最后，要将客户端与服务器同步，您需要将镜像与服务器同步，然后再将客户端与镜像同步：

$ cd /usr/local/aosp/mirror

$ repo sync

$ cd /usr/local/aosp/master

$ repo sync

您可以将镜像存储在 LAN 服务器上，然后通过 NFS、SSH 或 Git 访问它。您还可以将其存储在移动存储盘上，并在用户之间或计算机之间传用该存储盘。

## How to compile Android 7.1 for Pine A64

1. Checkout <http://source.android.com/source/downloading.html>
2. Create a new directory:
3. mkdir android
4. cd android
5. Initialize manifests:
6. repo init -u https://android.googlesource.com/platform/manifest -b android-7.1.2\_r11 --depth=1
7. git clone git@git.bonjour.tools:abailon/local\_manifests.git -b nougat-7.1 .repo/local\_manifests
8. Configure git: Add this to ~/.gitconfig:
9. [url "git@git.bonjour.tools:"]
10. insteadOf = https://git.bonjour.tools/
11. Checkout sources:
12. repo sync -c
13. Compile sources:
14. source build/envsetup.sh
15. # tulip\_ococci-eng: use to build Android with debug tools and permissive security
16. # tulip\_ococci-user: use to build a normal Android (currently doesn't work)
17. lunch tulip\_ococci-eng
18. make
19. Build a full OTA:
20. make dist

This will create the OTA out/dist/tulip\_ococci-ota-yyyymmdd.zip

1. Wipe the userdata partition (optionnal): Wipe data (also know as factory reset) will remove all userdata. This useful in the case of OTA install doesn't work as expected, or if the system doesn't work correctly after the install.
2. adb shell "echo --wipe\_data > /cache/recovery/command"
3. adb reboot recovery
4. Wipe the cache partition (optionnal): This useful in the case of OTA install doesn't work.
5. adb shell "echo --wipe\_cache > /cache/recovery/command"
6. adb reboot recovery
7. Install the OTA manually:
8. adb push out/dist/tulip\_ococci-ota-yyyymmdd.zip /cache/update.zip
9. adb shell "echo --update\_package=/cache/update.zip > /cache/recovery/command"
10. adb reboot recovery
11. Disable some Android UI customization:
12. adb shell setprop persist.holi.android\_ui 1
13. adb reboot
14. Enable automatically adb tcpip transport:
15. adb shell setprop persist.adb.tcp.port 5555

## Flash Bonjour using PhoenixSuit

1. Build Android using the previous instrctions
2. Generate the image file to flash using PhoenixSuit:
3. pack -d

This will create the file sun50iw1p1\_android\_ococci\_card0.img in ../lichee/tools/pack/.

1. Follow the instructions here to install and use PhoenixTools: [How\_to\_use\_PhoenixSuit\_to\_Flash\_New\_Image\_to\_MarsBoard\_A10/A20](http://www.hotmcu.com/wiki/How_to_use_PhoenixSuit_to_Flash_New_Image_to_MarsBoard_A10/A20)
2. Tips / troubleshoot:
   * PhoenixSuit is able to reboot the board in flash mode but you could do it manually by executing adb reboot efex
   * If PhonixSuit doesn't detect the device though it is in efex mode, try to unplug it and then replug it. Phoenix should detect it and start to flash the device.

## Tips

* Open settings without Launcher: adb shell am start -a android.settings.SETTINGS
* Open GoogleAPpp without Launcher: adb shell monkey -p com.google.android.googlequicksearchbox -c android.intent.category.LAUNCHER 1`

错误解决

使用没有密码的ssh

[git clone error: RPC failed; curl 56 OpenSSL SSL\_read: SSL\_ERROR\_SYSCALL, errno 10054](https://stackoverflow.com/questions/46232906/git-clone-error-rpc-failed-curl-56-openssl-ssl-read-ssl-error-syscall-errno)

**使用：**I've now tried increasing the buffer again

git config --global http.postBuffer 1048576000

* error.GitError: cannot initialize work tree
* 解决方法：
* 1 先用repo --trace sync -cdf将repo的所有动作详细输出，
* 会发现在这里出错导致：
* : export GIT\_DIR=......./vendor/mediatek/proprietary/hardware/gsm0710muxd  
  : git rev-parse --verify refs/remotes/github/cm-13.0-caf-8996^0 1>| 2>|  
    
    
  : cd ......../vendor/mediatek/proprietary/hardware/gsm0710muxd  
  : git read-tree --reset -u -v HEAD 1>| 2>|  
  fatal: This operation must be run in a work tree  
  Traceback (most recent call last):  
    File "/home/panos/android/cm13/.repo/repo/main.py", line 531, in <module>  
      \_Main(sys.argv[1:])  
    File "/home/panos/android/cm13/.repo/repo/main.py", line 507, in \_Main  
      result = repo.\_Run(argv) or 0  
    File "/home/panos/android/cm13/.repo/repo/main.py", line 180, in \_Run  
      result = cmd.Execute(copts, cargs)  
    File "/home/panos/android/cm13/.repo/repo/subcmds/sync.py", line 769, in Execute  
      project.Sync\_LocalHalf(syncbuf, force\_sync=opt.force\_sync)  
    File "/home/panos/android/cm13/.repo/repo/project.py", line 1247, in Sync\_LocalHalf  
      self.\_InitWorkTree(force\_sync=force\_sync)  
    File "/home/panos/android/cm13/.repo/repo/project.py", line 2393, in \_InitWorkTree  
      raise GitError("cannot initialize work tree")  
  error.GitError: cannot initialize work tree
* 2 因此，将vendor/mediatek/proprietary/hardware/gsm0710muxd这个文件夹删除，在manifest中查找这个路径对应的project是
* <project name="alps/vendor/mediatek/proprietary/hardware/gsm0710muxd" path="vendor/mediatek/proprietary/hardware/gsm0710muxd" />
* 于是，在.repo/projects下删除.repo/projects/vendor/mediatek/proprietary/hardware/gsm0710muxd.git目录，以及
* .repo/project-objects/vendor/mediatek/proprietary/hardware/gsm0710muxd.git目录
* 3 最后重新repo sync即可

在external\libunwind中创建aux目录（使用git bash）