基本操作：

git init （在文件夹内创建一个git本地仓库，也叫缓存区（就是一个.git的隐藏文件夹），此时，原文件夹叫做工作区）

git add 文件名 （将修改的文件从工作区添加到本地仓库）

git commit -m '一些版本信息' （在本地仓库创建一个版本）

git status (查询工作区的状态)

git log (查询本地仓库的版本列表)

回退版本：

git reset --hard HEAD^ （回退到上一个版本）

git reset --hard HEAD^^ (回退到上两个版本)

git reset --hard 版本id （跳到指定ID的版本，注意，id是使用git log查出各个版本的commit字段的一长串字符。只写到前7位就行）

放弃操作：

git checkout -- a1.txt (放弃工作区中a1.txt的修改，使其恢复到仓库中最后一个版本的状态)

git reset HEAD a1.txt (把暂存区的修改撤销掉，这里指的是使用git add命令添加的文件。撤销后，退回到git add之前的状态)

分支管理：

git checkout -b 分支名 (创建一个新分支，并切换到该分支)

git checkout 分支名 （切换到一个已有分支）

git branch （查看分支）

git merge dev2 (当前指针需要指在主分支，然后将dev2分支合并到主分支)

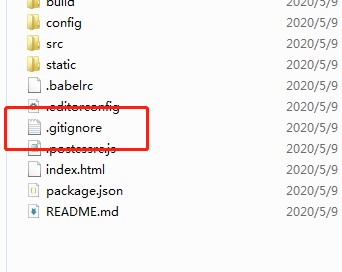
git branch -d dev2 （合并完，删除dev2分支）

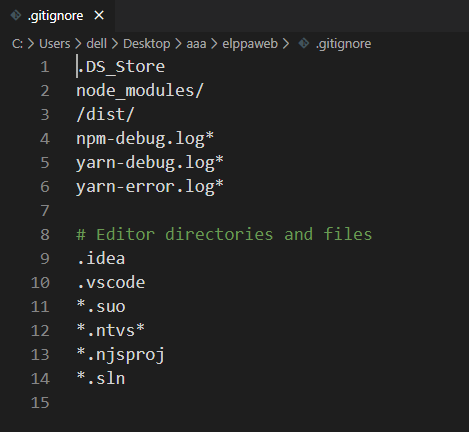
●直接克隆一个分支，这样clone完，修改然后提交会直接提交到这个分支里

git clone -b 分支名 路径

忽略文件

建立一个.gitignore文件





.DS\_Store

node\_modules/

/dist/

npm-debug.log\*

yarn-debug.log\*

yarn-error.log\*

# Editor directories and files

.idea

.vscode

\*.suo

\*.ntvs\*

\*.njsproj

\*.sln

<https://www.cnblogs.com/linsx/p/9335757.html>

# [git设置忽略文件.gitignore](https://www.cnblogs.com/linsx/p/9335757.html)

在仓库目录下新建一个名为.gitignore的文件（因为是点开头，没有文件名，没办法直接在windows目录下直接创建，必须通过右键Git Bash，按照linux的方式来新建.gitignore文件）。如下图所示。

.gitignore文件对其所在的目录及所在目录的全部子目录均有效。通过将.gitignore文件添加到仓库，其他开发者更新该文件到本地仓库，以共享同一套忽略规则。

以下涉及的ignore文件均为如下格式：

[复制代码](javascript:void(0);)

# 以'#'开始的行，被视为注释.

# 忽略掉所有文件名是 foo.txt的文件.

foo.txt

# 忽略所有生成的 html文件,

\*.html

# foo.html是手工维护的，所以例外.

!foo.html

# 忽略所有.o和 .a文件.

\*.[oa]

配置语法：  
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；

[复制代码](javascript:void(0);)

常用的规则：  
1）/mtk/               过滤整个文件夹  
2）\*.zip                过滤所有.zip文件  
3）/mtk/do.c         过滤某个具体文件

被过滤掉的文件就不会出现在git仓库中（gitlab或github）了，当然本地库中还有，只是push的时候不会上传。  
需要注意的是，gitignore还可以指定要将哪些文件添加到版本管理中：  
1）!\*.zip  
2）!/mtk/one.txt

唯一的区别就是规则开头多了一个感叹号，Git会将满足这类规则的文件添加到版本管理中。  
为什么要有两种规则呢？想象一个场景：假如我们只需要管理/mtk/目录中的one.txt文件，这个目录中的其他文件都不需要管理，那么我们就需要使用：  
1）/mtk/  
2）!/mtk/one.txt  
假设我们只有过滤规则，而没有添加规则，那么我们就需要把/mtk/目录下除了one.txt以外的所有文件都写出来！

最后需要强调的一点是，如果你不慎在创建.gitignore文件之前就push了项目，那么即使你在.gitignore文件中写入新的过滤规则，这些规则也不会起作用，Git仍然会对所有文件进行版本管理。  
简单来说，出现这种问题的原因就是Git已经开始管理这些文件了，所以你无法再通过过滤规则过滤它们。因此一定要养成在项目开始就创建.gitignore文件的习惯，否则一旦push，处理起来会非常麻烦。

更改用户

先输入

git config --global credential.helper wincred

再输入

git credential-manager remove

然后git操作时就提示你输入新用户了

<https://blog.csdn.net/fuyongsheng135/article/details/94618432>

<http://blog.csdn.net/kevindgk/article/details/51606925>

■简要概述：

●设置账号名和邮箱

$ git config --global user.name "Your Name"

$ git config --global user.email "email@example.com"

●克隆一个项目到本地

git clone 项目的url

●修改后先查看以下状态

$ git status

●添加文件

$ git add . 添加所有的文件、文件夹

$ git add <file> 添加指定名称的文件,<>内部写文件全称包括扩展名

git add . ：他会监控工作区的状态树，使用它会把工作时的**所有变化提交**到暂存区，包括文件内容修改(modified)以及新文件(new)，但不包括被删除的文件。

git add -u ：他仅监控**已经被add的文件**（即tracked file），他会将被修改的文件提交到暂存区。add -u 不会提交新文件（untracked file）。（git add --update的缩写）

git add -A ：是上面两个功能的合集（git add --all的缩写）

●提交文件

$ git commit –m “commit info” 提交本次事务，即将add的文件提交到git仓库，引号内部表示本次提交的提示信息

●push

git push

●最终检查确认

$ git status

如果上述都完成没有问题，会提示：nothing to commit, working tree clean等类似的信息

Git总结

简介

说明：下列文本性内容部分来自廖雪峰的网站，一部分来自公司实战，一部分来自官方网站   
版权声明：本文为原创文章，未经允许不得转载   
博客地址：[http://blog.csdn.net/kevindgk](http://blog.csdn.net/kevindgk" \t "_blank)

学习网址

1. http://www.liaoxuefeng.com/ Git的完整学习教程

2. https://git-for-windows.github.io/ windows上安装msysgit，内部包含模拟环境和Git

3.如果英文不好，可以使用中文版，然后直接使用图形化界面Git Gui，而不使用Git bash

其他学习网址：

https://blog.cnbluebox.com/blog/2014/04/15/gitlabde-shi-yong/

http://www.oschina.net/translate/10-tips-git-next-level

基本命令

初始化设置

配置本机的用户名和Email地址

$ git config --global user.name "Your Name"

$ git config --global user.email "email@example.com"

创建版本库(仓库)

版本库又叫仓库(repository)，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除都能被跟踪。

在合适的位置直接鼠标右键创建一个空目录作为仓库，然后从Git-Bash命令行进入到该目录，或者也可以使用命令行创建空目录，再进入到该空目录中。

以下给出创建并初始化git仓库的代码：

进入到仓库的位置，我将仓库放在了C:\Android\git-repositories目录下，注意，使用cd命令进入到目录中时，在Git-Bash中应该使用斜线”/”，

而不是反斜线”\”

$ cd C:/Android/git-repositories

$ mkdir new\_repository\_1 创建新的目录

$ cd new\_repository\_1 进入到创建的目录

使用init命令将当前目录初始化为Git仓库

$ git init

Initialized empty Git repository in C:/Android/git-repositories/new\_repository\_1/.git/

(显示信息意思为：初始化了一个空的Git仓库，new\_repository\_1目录下多了一个.git目录，时用来管理版本库的)

将数据提交到git仓库（本地仓库）

第一步：添加文件

$ git add . 添加所有的文件、文件夹

$ git add <file> 添加指定名称的文件,<>内部写文件全称

注：如果文件没有做出任何修改，则默认不会添加任何文件

第二步：提交文件

$ git commit –m “commit info” 提交本次事务，即将add的文件提交到git仓库，引号内部表示本次提交的提示信息

查询提交状态

$ git status 显示提交的状态：已经添加，等待提交事务的文件(绿色字体表示)；已经改变但是没有添加(not staged)的文件(红色字体表示)；

查询该文件和git仓库中的文件的区别，即做了什么修改

$ git diff <文件全称> 如果已经add了，就打印不出有什么修改了，这一步骤应该在add之前，即添加之前可以用来看看做了什么修改。

打印历史记录

$ git log

Commit xxx commit id 版本号

Author:xxx<xxx@xxx.com> 提交人和邮箱

Date：xxx 提交的时间

XXXXXXXXXXXXXX 提交的信息(所以说，提交信息很重要！！！)

$ cat <文件全名称> 显示整个文件的内容

版本回退

$ git reset --hard head^

在Git中，HEAD表示当前版本，就是最新提交的版本，即使用git log打印出来的位于第一位的版本，上一个版本就是HEAD^，上上个版本就是HEAD^^，

当前向上100个可以写成HEAD~100。当然，还有一种方式就是直接使用commit id来代替HEAD^，比如版本号是cadab353589f3eef075817b890dafe8b722d802b，

那么就可以直接使用命令：

$ git reset --hard cadab353589f 使用前几位表示即可，git会自动查找

注：版本回退以后，使用git log打印的历史记录都是回退版本之前的数据，之后的都没有了，不过放心，git总有后悔药可以吃哒~

1.如果命令行窗口没有关闭，直接去前面找commit id即可；

2.如果命令行窗口关闭了，或者第二天后悔了，可以进入到该目录下，使用git reflog命令来查看以前的每一次命令，可以获得每次提交的commit id，

就可以版本回退了。

$ git reflog 可以查看命令历史，包含提交的commit id

版本回退原理

   
简单讲，就是说只要进行了代码提交，git内部都会按照时间节点进行记录，每条记录都有commit id作为唯一标识(就像是链表每个节点都有唯一的地址一样)，HEAD总是指向当前版本(就像指针一样)。所谓的版本回退，仅仅是讲Head从当前版本指向了指定的版本，然后将工作区的文件也修改了。

工作区和暂存区

Git和其他版本控制系统的一个不同之处就是有暂存区的概念。

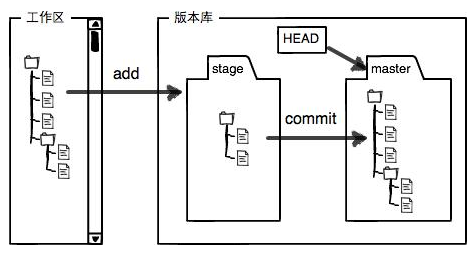
- 工作区

就是电脑里能看到的目录，比如上面创建的C:\Android\git-repositories\new\_repository\_1文件夹就是一个工作区。

- 版本库

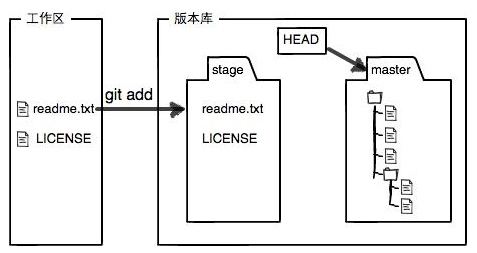
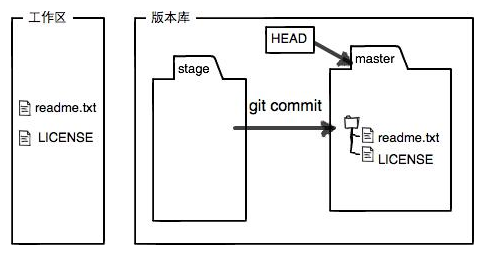
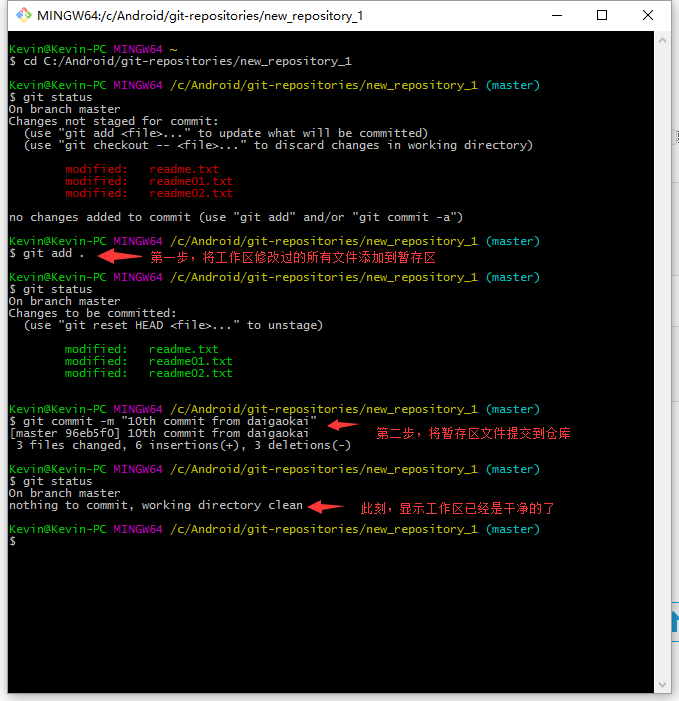
工作区中有一个隐藏目录.git，就是Git的版本库，版本库里存放了很多的东西，其中最重要的就是state(或者叫index)的暂存区，

还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。



前面讲到，将文件存入到Git版本库里，分两步执行：

第一步：用git add命令将工作区的修改文件添加到暂存区； （多次操作）

   
第二步：用git commit命令将暂存区的所有修改内容提交到当前分支。（事务提交，包含第一步多次操作，注意，不在暂存区的修改不会被commit）   
   
一旦事务提交之后，如果对工作区没有做什么修改，那么工作区就是干净的。   
因为创建Git版本库的时候，Git自动创建了一个master分支，所以现在git commit 就是往master分支上提交事务。   
Eg：   


项目开发实战-\*\*app

1. 需要安装的软件：msysgit

2. 需要申请的账号：

2.1 公司GitLab账号：向公司GitLab管理人员申请 – \*\*\*

2.2 项目GitLab权限：向本项目的创建/管理人员申请 – 比如\*\*app管理者 \*\*\*

3. 进入到GitBash命令行操作：

在合适的位置点击右键，选择GitBash Here

本机地址为：C:\Android\git-repositories，自己创建的git仓库地址

关闭证书验证：原因是因为本公司服务器证书已经过期，所以直接关闭证书验证即可

$ git config --global http.sslVerify false

使用克隆命令将远程仓库的代码复制一份到本地，注意此处应该用https访问

$ git clone https://\*\*\*.\*\*\*.\*\*\*.\*\*\*/android/salestool.git

(输入用户名和密码之后，将开始下载远程仓库，这里仅仅下的是主分支-master)

进入到项目，即从命令行进入已经下载下来的git仓库，saletool/表示本项目的目录名

$ cd salestool/

查看仓库的分支情况

$ git branch –a

显示如下：

\* master

remotes/origin/HEAD -> origin/master HEAD-远程仓库的当前分支是主分支

remotes/origin/dev dev分支(所有操作都会合并到该分支)

remotes/origin/master master分支-主分支

创建本地仓库的dev分支

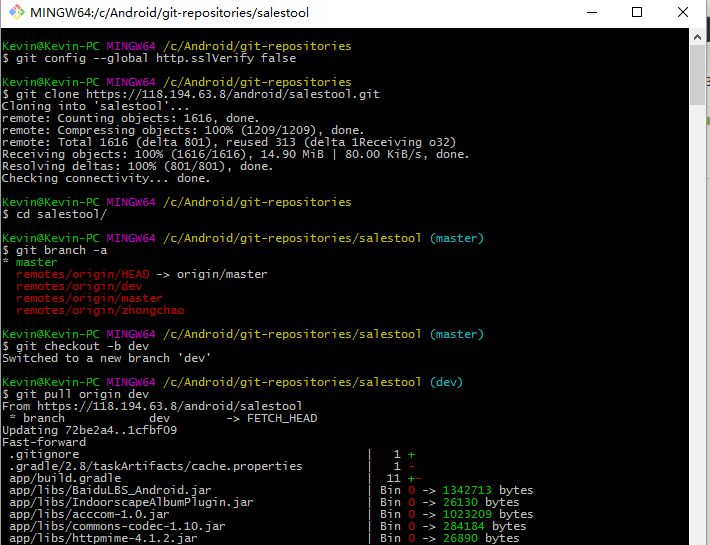
$ git checkout -b dev

将远程仓库的dev分支代码复制到本地dev分支

$ git pull origin dev

(由于公司服务器比较慢，所以你懂得... ...)

以上操作如下图所示：



查看本地git仓库状态

$ git status

On branch dev -只有一个本地dev分支(但是内容已经是远程仓库dev的内容了)

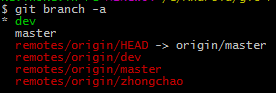
nothing to commit, working directory clean -此时没有任何修改，工作区很干净

查看分支状态

$ git branch –a

上面显示的是本地分支，绿色字体和”\*”表示的是当前所在的分支，

下面红色部分显示的是远程仓库的分支。



创建自己的本地分支，并切换到该分支，自己在此分支上写代码

$ git checkout -b dai

此时开始在Android studio中对该项目进行编程~~~

将所有修改文件提交到本地暂存区(staged)，等待提交

$ git add . 注意：确保此时在自己的分支上进行操作，eg：dai(我自己的名字)

$ git commit –m “” 将本地暂存区的代码提交到自己的分支上

切换到本地dev分支，并将远程仓库的dev分支的最新代码拉下来

$ git checkout dev

$ git pull origin dev

(此时，本地仓库的dev分支已经确保是最新的了)

切换到自己的分支，将dev分支合并到自己的分支上

$ git checkout dai

$ git merge dev 将本地dev分支合并到自己的分支上

注意：此时已经将dev分支合并到本地的自己的分支上了，有时候可能需要解决代码冲突问题，解决完毕后进行下面的操作。

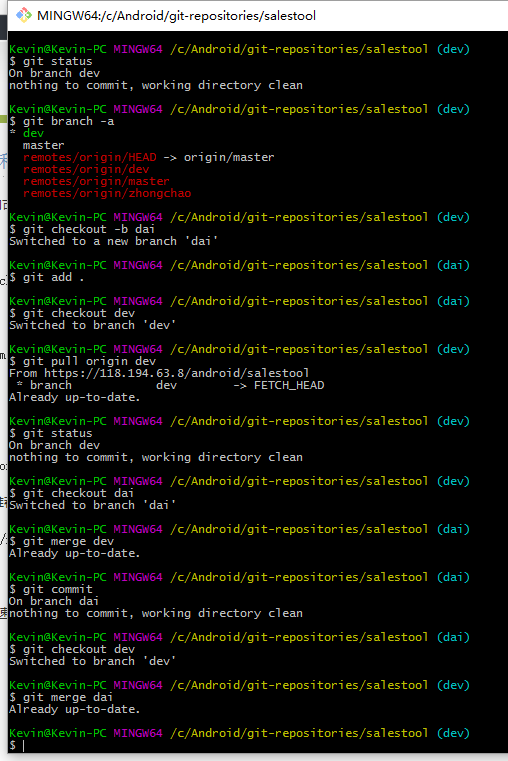
如果有冲突，则需要再次进行add,commit操作。

解决冲突完毕后，切换到本地dev分支，将合并完毕的自己的分支合并到本地dev

$ git checkout dev

$ git merge dai

以上操作的代码如下：



接下来的操作，就是将本地dev分支推到远程仓库的dev分支上了... ...

推送到远程服务器

$ git push origin dev

git进阶

其他命令

1. git remote –v 显示远程分支的名称和url

忽略文件

有一些文件并不能上传到git上。

1.Android Studio自动生成配置文件：不能上传到git上，否则的话，如果你的同事下载下来，但是它的studio(gradle)版本和你的不一样，

或者其他配置的各种路径不一样，就需要重建项目，严重的话，根本无法重建项目，一片爆红！网上的方法也解决不了。

2.保存了数据库密码或者什么不能上传的文件；

。。。

所以需要在项目的根目录下创建一个名称为.gitignore文件，然后把要忽略的文件名填进去，Git就会自动忽略这些文件。

不需要从头写.gitignore文件，GitHub已经为我们准备了各种配置文件，只需要组合一下就可以使用了。

忽略文件的原则是：

1.忽略操作系统自动生成的文件，比如缩略图等；

2.忽略编译生成的中间文件、可执行文件等，也就是如果一个文件是通过另一个文件自动生成的，那自动生成的文件就没必要放进版本库，

比如Java编译产生的.class文件；

3.忽略你自己的带有敏感信息的配置文件，比如存放口令的配置文件。

想知道忽略那些文件吗，很简单，找个大神的github，看看他的项目中怎么写的，就ok了！！！

例如，下面是我从张鸿祥哪里copy的，仅做参考：

/captures

# Built application files

\*.apk

\*.ap\_

# Generated files

bin/

gen/

# Gradle files

.gradle/

/build

/\*/build/

# Local configuration file (sdk path, etc)

local.properties

# Proguard folder generated by Eclipse

proguard/

# Log Files

\*.log

# Eclipse project files

.classpath

.project

.settings/

# Intellij project files

\*.iml

\*.ipr

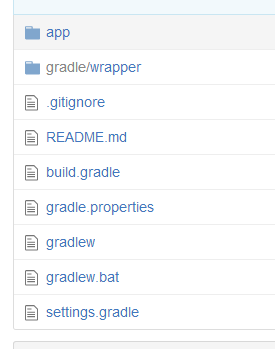
\*.iws

.idea/

# System files

.DS\_Store

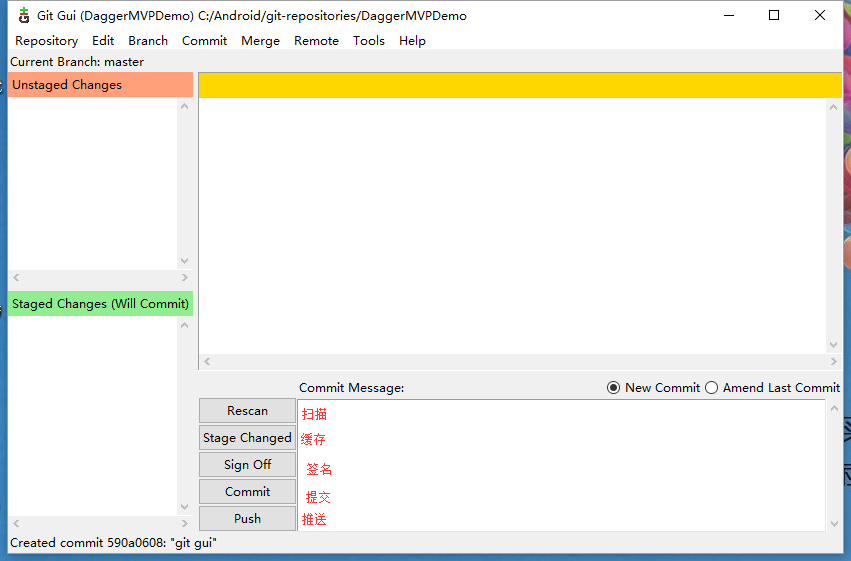
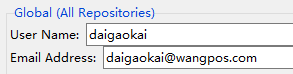
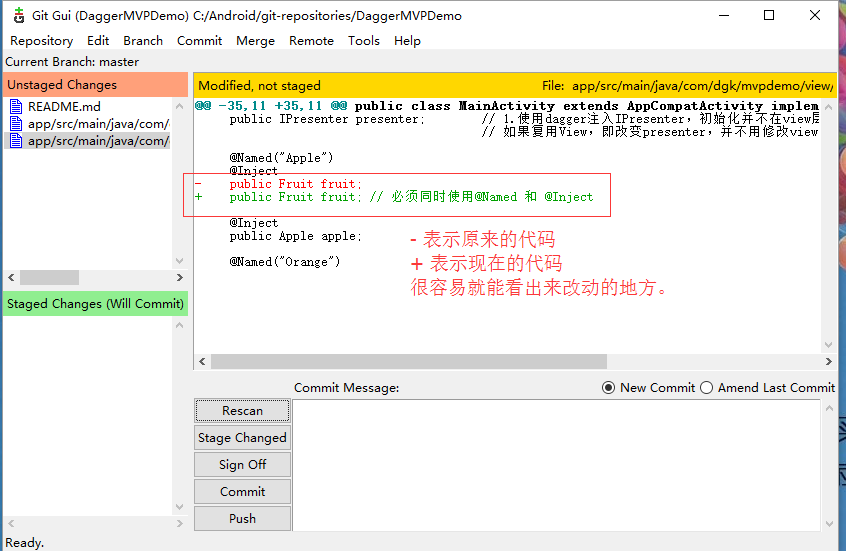
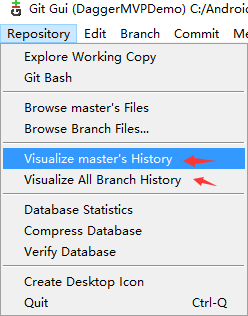
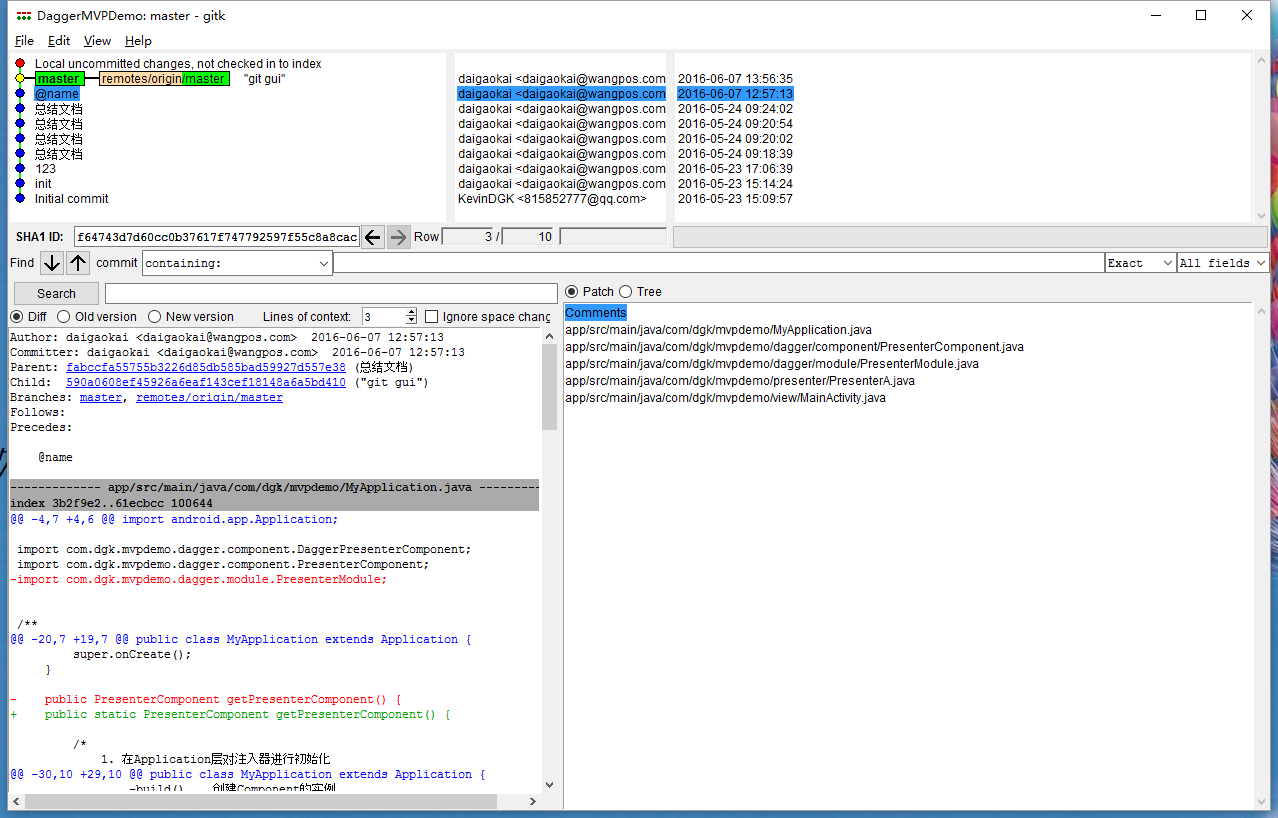
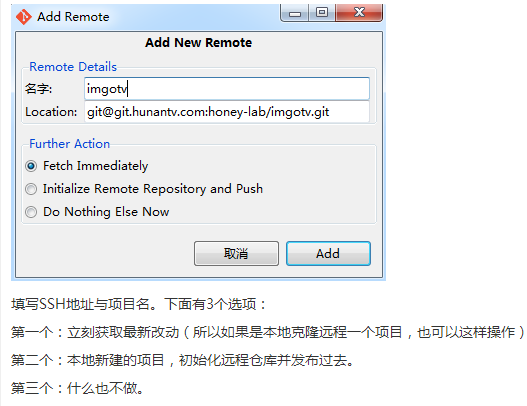
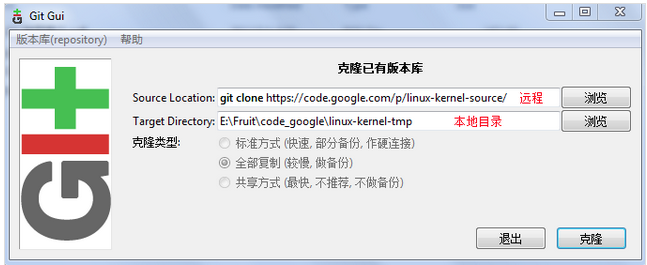
下面是比较清晰的目录结构：



GIT GUI简单使用#

注意：个人建议使用命令行方式进行版本管理，但是可以使用图形化界面看本次代码的改动，比较方便。

1. 在所在项目，右键选择git gui

   
2. 界面如下，如果会使用命令行，那么一看就明白了   
   
3. 配置   
UTF-8:Edit-Options:   
http://i.imgur.com/VBTBQrn.png   
如果之前设置好了，直接在项目中右键进入，那么这些都不用设置：   
   
4. Add commit push很快完成，不用输i入命令   
   
5. 查看代码对比   
   
如果想要查看所有的改动历史，可以：   
   
就可以看到所有的代码改动历史，而不用去网上看。注意，这里能看到所有人的改动哦！！！非常强大！   
   
6. 设置和远程仓库关联（如果从项目根目录进入，则自动关联，不用设置）   
如果需要设置，选择remote-◊Add，参考如下：   
   
7. 新建项目，从远程仓库克隆   
右键选择git gui:   
   
选择克隆已有版本库：   


linux命令

1. $ pwd 用于显示当前目录

2.$ ls –ah 用于显示当前目录下的所有子目录和文件(包含隐藏的)

3.$ clear 清屏

4.$ exit 退出linux模式

注意：通过cmd进入linux模式，命令是 adb s4.hell

注意事项

1.不要使用记事本打开编辑任何文本文件，可以使用Notepad++，默认编码格式设置为UTF-8 无BOM;

2.开发Android studio项目，GitBash命令行和as中的Terminal，使用效果是一样的。

基本操作：

git init （在文件夹内创建一个git本地仓库，也叫缓存区（就是一个.git的隐藏文件夹），此时，原文件夹叫做工作区）

git add 文件名 （将修改的文件从工作区添加到本地仓库）

git commit -m '一些版本信息' （在本地仓库创建一个版本）

git status (查询工作区的状态)

git log (查询本地仓库的版本列表)

回退版本：

git reset --hard HEAD^ （回退到上一个版本）

git reset --hard HEAD^^ (回退到上两个版本)

git reset --hard 版本id （跳到指定ID的版本）

放弃操作：

git checkout -- a1.txt (放弃工作区中a1.txt的修改，使其恢复到仓库中最后一个版本的状态)

git reset HEAD a1.txt (把暂存区的修改撤销掉)

分支管理：

git checkout -b 分支名 (创建一个新分支，并切换到该分支)

git checkout 分支名 （切换到一个已有分支）

git branch （查看分支）

git merge dev2 (当前指针需要指在主分支，然后将dev2分支合并到主分支)

git branch -d dev2 （合并完，删除dev2分支）