# 安装下载

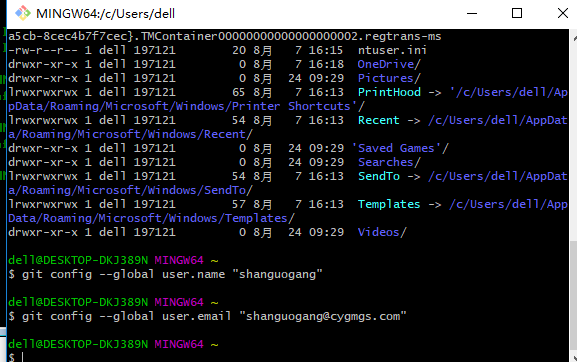
## 下载安装

下载地址：<http://msysgit.github.io>

安装就不再多描述了，安装指引操作就行。

## 基本配置

安装完成后桌面上会生成一个Git bash图标，双击打开会弹出一个类似命令行窗口的东西，然后配置名字和邮箱：



1.$ git config --global user.name "shanguogang"

2.$ git config --global user.email “mingqian\_chen@163.com”

Git是分布式控制系统，所以，每个客户端都必须设置：你的名字和邮箱。这样才容易区分是哪个用户提交的Git版本。大家可以看到git config 后面接的是--global参数，这表示你的机器上所有的Git仓库都会使用这个配置，但你也可以对不同的仓库指定不同的用户名和邮箱。这个我们会在后面的文章中讲解，这里不作说明。有兴趣的博友可以先提前google一下。

# 本地仓库

## 概述

版本库或者是仓库，英文名Repository，其实啊说白了就是一个目录而且，这个目录中的所以文件都被git管理而且，不管你做什么操作都会被记录，包括：增加、删除、修改文件等，都会被记录下来，以便后来跟踪与修改相关记录，甚至被还原。

## 创建本地仓库

### 创建目录

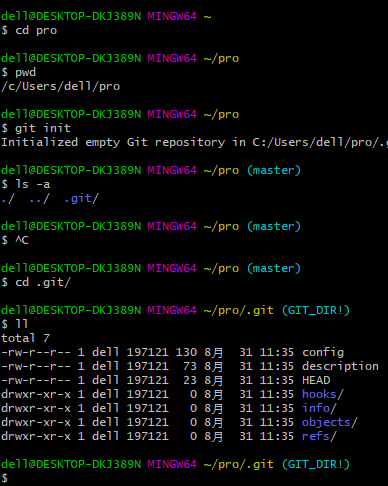
$ mkdir 文件名



在/c/Users/dell/下创建一个空目录pro即可project简写。还有一点需要说明的在Windows系统中目录名称不要使用中文，不然会出现神马问题，我不负责。

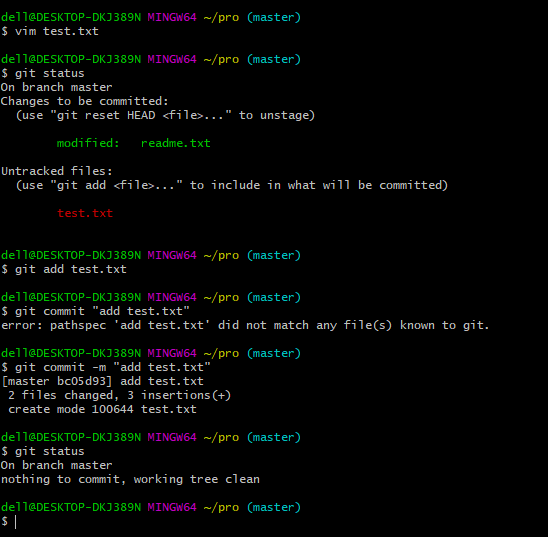
### 目录变成仓库

执行cd pro 命令进入pro文件夹，再执行git init命令初始化一个本地仓库，路径为C:/Users/dell/pro/.git/。大家可以看到pro目录中生成了一个.git目录，里面有很多文件，这里的文件是控制和管理版本库，不要随便修改里面文件。



### 在本地库中添加文件

版本库中只能跟踪和管理文件，比如txt、js文件、php文件、java文件等等，所有的程序代码都可以的。但视频、图片这些二进制文件，虽然能有git管理，但只能记录大小，无法跟踪具体修改了什么。



首先，我们在pro目录新建test.txt文件。然后，使用gitadd命令文件放到版本库中，再使用git commit命令提交。

1. git status 命令

查看版本库状态，如上图。大家可以看到git记录我们新增加一个文件test.txt，并且提示这个文件还没有被提交。下面我们用git add命令提交一下。

1. git add 命令

git add命令是告诉git, 我们要把什么文件提交到仓库中去，大家可以看我们执行git add test.txt命令后，没有任何提示。那就说明我们提交完成了。下面我们通过git commit命令，将test.txt文件提交到版本库中。

1. git commit 命令

$ git commit -m "add test.txt"

[master bc05d93] add test.txt

2 files changed, 3 insertions(+)

create mode 100644 test.txt

大家可以看到我们用git commit命令提交readme.txt文件，给出的提示是 1 file changed, 3 insertions(+)，一个文件改变，插入了三行内容。与我们上面增加三行内容一致。嘿嘿！下面我们简单的说明一下git commit命令，其中的参数-m后面输入的是本次提交的版本说明，可以输入任意内容，但需要说明的是，最好写有意义的说明，便于以后查看。

### 总结

1）初始化一个git仓库，使用git init命令

2）添加文件到本地仓库，分两步

使用git add命令，可以多次使用添加多个文件。

使用git commit命令,完成添加。

为什么git添加文件需要两步呢？一步add一步commit。因为产品，commit一次可以提交多个文件，下面案例：

$ git add file1.php

$ git add file2.php

$ git add file3.php

$ git commit –m “add file1.php file2.php file3.php”

# 远程仓库

## 查看远程仓库

$ git remote -v列出当前项目远程库地址

$ git branch –a 查看项目远程分支列表

## 添加远程仓库

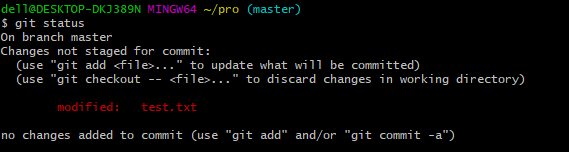
## 远程仓库拉取项目在本地

$ git clone 项目地址

# 常用命令

## git status 查看状态命令

查看创库状态，



## git add添加文件到创库命令

添加文件到创库

$ git add 文件名

## git commit 提交命令

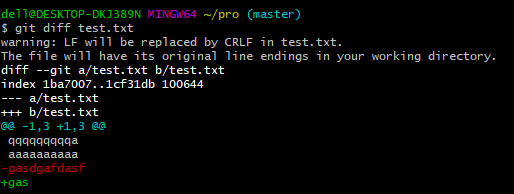
提交命令，把文件提交到仓库

$ git commit –m “备注”

## git diff 查看文件修改内容命令

查看修改内容：

$ git diff 文件名



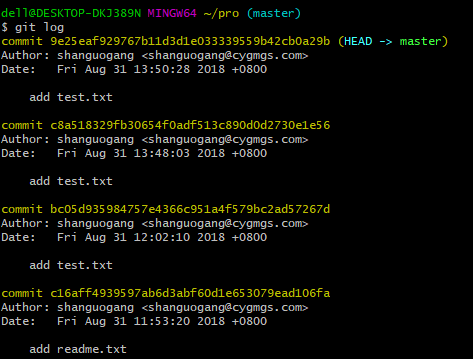
红颜色标出来的为原来内容，绿色为修改后的内容。

## git log 查看版本命令

$ git log

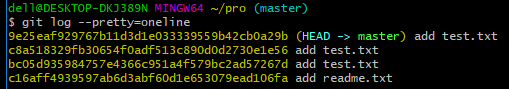
$ git log --pretty=oneline

我们使用git log查看最近提交版本



如果内容太多使用下面命令：

$ git log --pretty=oneline



## git reset 回退版本命令

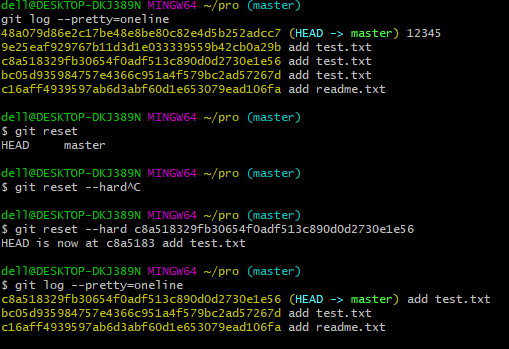
$ git reset –hard HEAD

Git的当前是在哪个版本，在Git中我们用HEAD表示当前版本，也就是我们最新提交的”749f84ccb87……ff2q6ad“，上一个版本就应该这么表示”HEAD^“，再上一个版本就这样表示”HEAD^^“

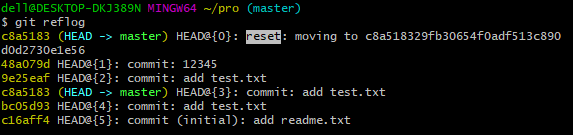
$ git reset –hard 提交id

例子：$ git reset --hard c8a518329fb30654f0adf513c890d0d2730e1e56

git checkout



## git reflog 记录执行命令

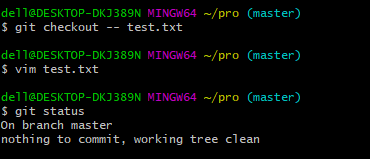


## git checkout --丢弃工作区的修改

命令git checkout -- test.txt意思就是，把test.txt文件在工作区的修改全部撤销，这里有两种情况：

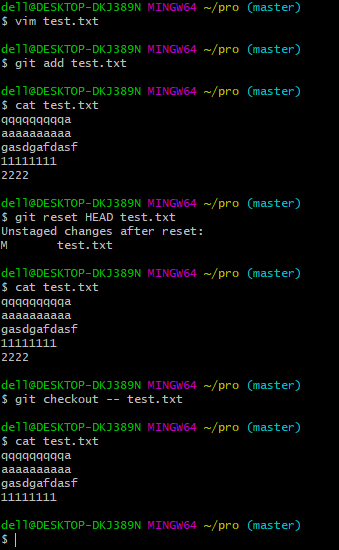
一种是test.txt自修改后还没有被放到暂存区，现在撤销修改就回到和版本库一模一样的状态；

另一种是test.txt已经添加到暂存区后，又作了修改，现在撤销修改就回到添加到暂存区后的状态。



使用git add 提交到了缓存区，回退原来状态：

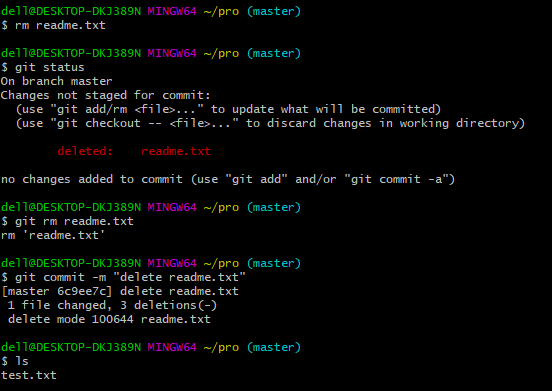
第一步用命令git reset HEAD file，就回到了第1步，第二步按第1步进行操作，就是执行一下git checkout -- file。



## git rm 版本库中删除文件

有两种情况，

1）从版本库中删除该文件，就 git rm 命令删除掉并用 git commit提交



2）删除文件后，回退使用 git checkout – test2.txt

## git remote 查看远程仓库

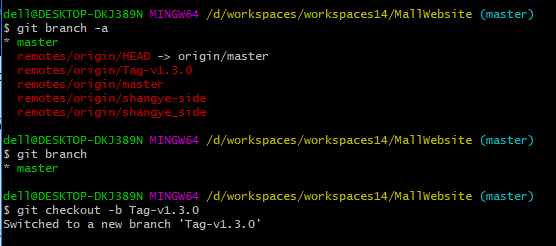
## git checkout切换分支

**1. 查看远程分支** $ git branch –a

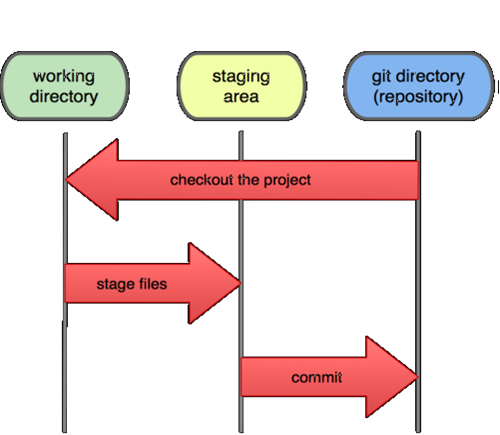
**2. 查看本地分支** $ git branch

**3. 切换分支** $ git checkout -b Tag-v1.3.0

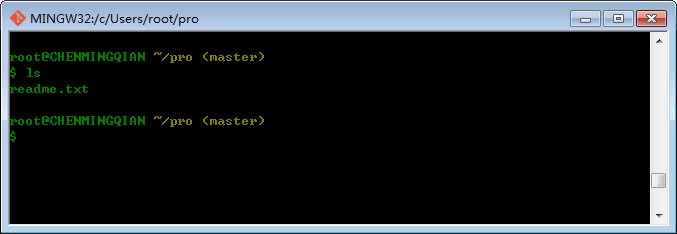
**当远程分支与本地分支名相同会自动关联**

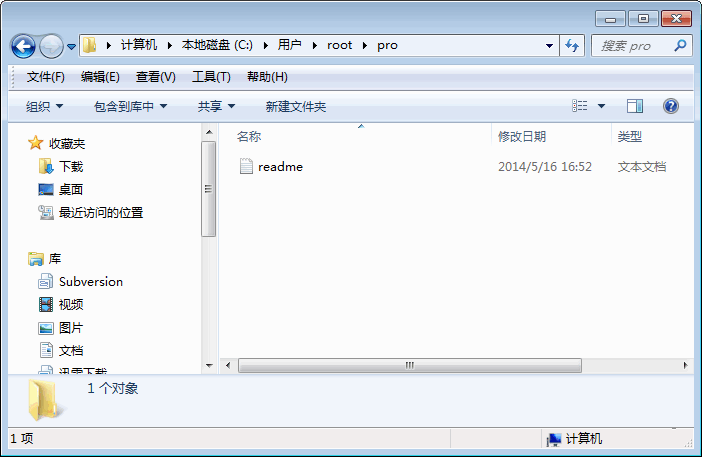


# 工作区、提交区/暂存区（stage/index）、版本库

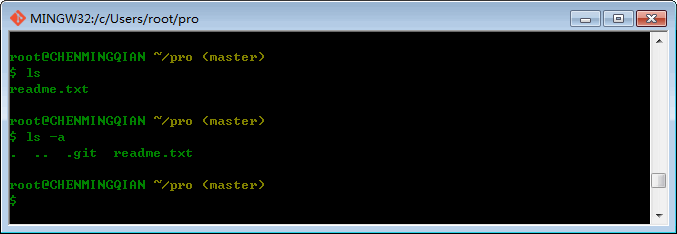


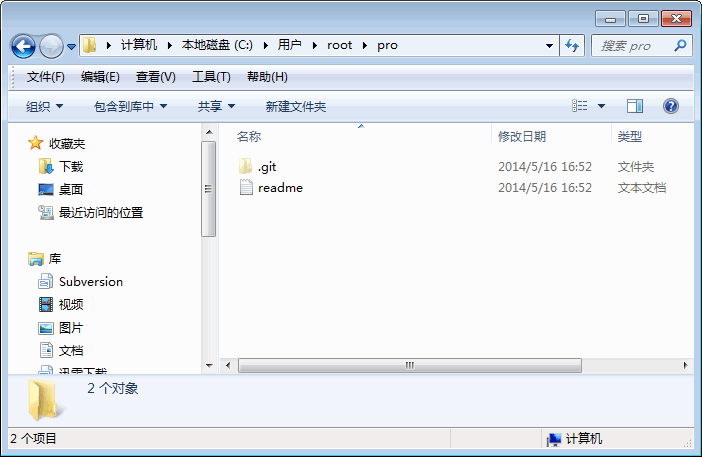
其实呢，工作区、提交区/暂存区（stage/index）、版本库的概念问题，从上图中就能看的很清楚，本来不想细讲的，但想想还是说一下。Git与其他版本版本控制器其中之一的不同之处就在于有提交区/暂存区（stage/index）的概念。下面我们先来看一下工作区：



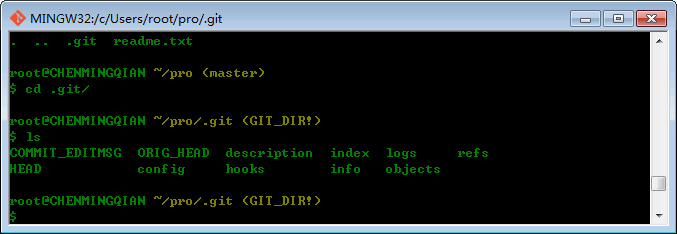


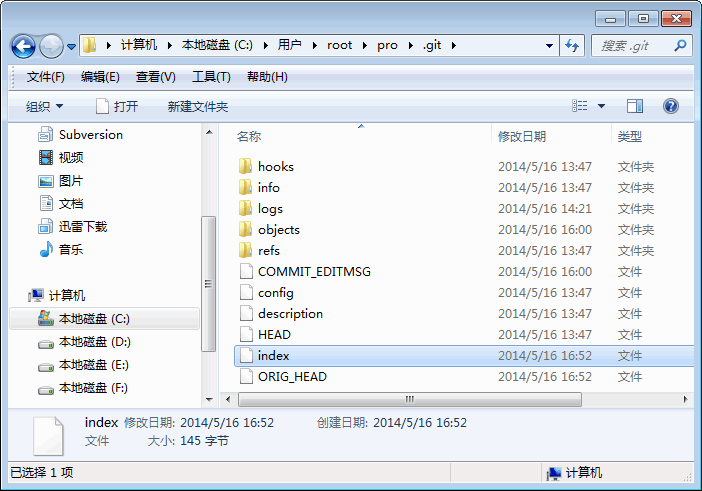
其实呢，工作区就是我们开发目录了，在电脑中是可能看到的，比如我们这里的pro目录，就是一个工作区。大家再来看一下，下面的两张图：





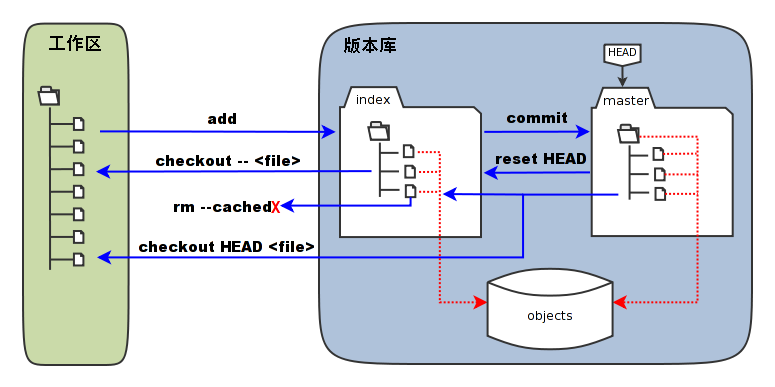
大家可以看到，工作区中有个隐藏的目录“.git”，这个不是工作区哦，这个就是Git的版本库。大家再看下面两张图：





大家可以看到，在“.git”目录中有很多文件，其中一个重要的文件index，就是我们说的提交区/暂存区（stage/index）。暂存区（stage, index）是 Git 最重要的概念之一，理解了这个概念很多 Git 命令就不再那么神秘了。对于 Git 暂存区（stage） 。

Git 暂存区（stage或称为 index）的设计是 Git 最成功的设计之一，也是最难理解的一个设计。 在版本库（.git）目录下，有一个 index 文件，相信大家在上图中已经看到了。下面我们好好说一说他们之间关系，同样的我们先看一张图：



在上图中，我们可以看到部分 Git 命令是如何影响工作区和暂存区（stage/index）的。

* 图中左侧为工作区，右侧为版本库。在版本库中标记为 "index" 的区域是暂存区（stage/index），标记为 "master" 的是 master 分支所代表的目录树（关于分支问题在下面的文章中会详解）。

图中我们可以看出此时 "HEAD" 实际是指向 master 分支的一个“指针”。所以，图示的命令中出现 HEAD 的地方可以用 master 来替换（HEAD的概念我们在后面的文章中也会详解）。

图中的 objects 标识的区域为 Git 的对象库，实际位于 ".git/objects" 目录下，我们会在后面的文章中将重点介绍，嘿嘿！。

当对工作区新增或修改的文件执行 "git add" 命令时，暂存区的目录树被更新，同时工作区新增或修改的文件内容被写入到对象库中的一个新的对象中，而该对象的ID被记录在暂存区的文件索引中。（如上图）

当执行提交操作 "git commit" 时，暂存区的目录树写到版本库的对象库（objects）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树。（如上图）

当执行 "git reset HEAD" 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。 当执行 "git rm --cached <file>" 命令时，会直接从暂存区删除文件，工作区则不做出改变。

当执行 "git checkout ." 或者 "git checkout -- <file>" 命令时，会用暂存区全部或指定的文件替换工作区的文件。这个操作很危险，会清除工作区中未添加到暂存区的改动。

当执行 "git checkout HEAD ." 或者 "git checkout HEAD <file>" 命令时，会用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件。这个命令也是极具危险性的，因为不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改动。