代码整理：

创建本地管理仓库

git init

添加上传文件(多个文件)到暂存区

git add 文件

提交

git commit -m “注释”

HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit\_id。

穿梭前，用git log可以查看提交历史，以便确定要回退到哪个版本

要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

克隆部分

git clone https://github.com/zhangjincong99/mic.git

Git鼓励大量使用分支：

查看分支：git branch

创建分支：git branch <name>

切换分支：git checkout <name>或者git switch <name>

创建+切换分支：git checkout -b <name>或者git switch -c <name>

合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name>

上传命令整理

git config --global user.email "你自己的github邮箱"

git config --global user.name "你自己的github名字"

ssh-keygen -t ed25519 -C "你自己的github邮箱"

然后在github右上角下拉框中点击setting-->点击SSH and GPG keys -->New SSH key创建自己的title和key，

注意：key在C盘用户的.ssh文件中的id\_rsa.pub

git remote add new-origin git@github.com:zhangjincong99/mic.git

git remote set-url new-origin git@github.com:zhangjincong99/mic.git

git add 概要设计.docx

git commit -m 'add wenzi'

git pull new-origin master

git push new-origin master

注意：

Git命令必须在Git仓库目录内执行（git init除外），在仓库目录外执行是没有意义的。

创建仓库

在某个路径创建一个空文件夹

然后右键点击Git Bash Here

在命令窗中输入

git init

通过git init命令把这个目录变成Git可以管理的仓库当前目录下多了一个.git的目录，这个目录是Git来跟踪管理版本库的，没事千万不要手动修改这个目录里面的文件，不然改乱了，就把Git仓库给破坏了。

把一个文件放到Git仓库只需要两步

第一步，用命令git add告诉Git，把文件添加到仓库：

$ git add readme.txt

第二步，用命令git commit告诉Git，把文件提交到仓库：

$ git **commit** -m "wrote a readme file"

简单解释一下git commit命令，-m后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录。

为什么Git添加文件需要add，commit一共两步呢？因为commit可以一次提交很多文件，所以你可以多次add不同的文件，比如：

$ git add file1.txt

$ git add file2.txt file3.txt

$ git commit -m "add 3 files."

注意创建文本时，不要使用电脑自带的记事本。可直接使用

vi readme.txt

在Git中，我们用git log命令查看：

我们要把当前版本append GPL回退到上一个版本add distributed，就可以使用git reset命令：

$ git re**set** --hard HEAD^

看看readme.txt的内容是不是版本add distributed：

$ cat readme.txt

还想再回去怎么办？

办法其实还是有的，只要上面的命令行窗口还没有被关掉，你就可以顺着往上找啊找啊，找到那个append GPL的commit id是1094adb...，于是就可以指定回到未来的某个版本：

$ git re**set** --hard 1094a

版本号没必要写全，前几位就可以了，Git会自动去找。

记不得id了怎么办？

Git提供了一个命令git reflog用来记录你的每一次命令：

$ git reflog

总结一下：

* HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit\_id。
* 穿梭前，用git log可以查看提交历史，以便确定要回退到哪个版本。
* 要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

创建本地管理仓库

git init

添加上传文件(多个文件)到暂存区

git add 文件

提交

git commit -m “注释”

HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit\_id。

穿梭前，用git log可以查看提交历史，以便确定要回退到哪个版本

要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

工作区（Working Directory）

就是你在电脑里能看到的目录，比如我的learngit文件夹就是一个工作区：

#### 版本库（Repository）

工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。

Git的版本库里存了很多东西，其中最重要的就是称为stage（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。

前面讲了我们把文件往Git版本库里添加的时候，是分两步执行的：

第一步是用git add把文件添加进去，实际上就是把文件修改添加到暂存区；

第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支。

因为我们创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以，现在，git commit就是往master分支上提交更改。

你可以简单理解为，需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

先用git status查看一下状态：查看未使用commit的文件

$ git status

Git非常清楚地告诉我们，readme.txt被修改了，而LICENSE还从来没有被添加过，所以它的状态是Untracked。

管理修改

如果修改了了文本—添加到了暂存区—但是如果又修改了文本—没有再次添加到暂存区—那么最后commit的时候啥也没有

所以在第二次的时候也要使用git add 将其添加到暂存区

撤销修改

Git会告诉你，git checkout -- file可以丢弃工作区的修改：

$ git checkout -- readme.txt

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

一种是readme.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；

一种是readme.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。

总之，就是让这个文件回到最近一次git commit或git add时的状态。

删除文件

在Git中，删除也是一个修改操作，我们实战一下，先添加一个新文件test.txt到Git并且提交：

一般情况下，你通常直接在文件管理器中把没用的文件删了，或者用rm命令删了：

远程仓库

在继续阅读后续内容前，请自行注册GitHub账号。由于你的本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以，需要一点设置：

第1步：创建SSH Key。在用户主目录下，看看有没有.ssh目录，如果有，再看看这个目录下有没有id\_rsa和id\_rsa.pub这两个文件，如果已经有了，可直接跳到下一步。如果没有，打开Shell（Windows下打开Git Bash），创建SSH Key：

$ ssh-keygen -t rsa -C [youremail@example.com](mailto:youremail@example.com)

ssh-keygen -t rsa -b 4096 -C "2934471851@qq.com"

你需要把邮件地址换成你自己的邮件地址，然后一路回车，使用默认值即可，由于这个Key也不是用于军事目的，所以也无需设置密码。

如果一切顺利的话，可以在用户主目录里（C:\Users\86158\.ssh）找到.ssh目录，里面有id\_rsa和id\_rsa.pub两个文件，这两个就是SSH Key的秘钥对，id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

第2步：登陆GitHub，打开“Account settings”，“SSH Keys”页面：

然后，点“Add SSH Key”，填上任意Title，在Key文本框里粘贴id\_rsa.pub文件的内容：



点“Add Key”，你就应该看到已经添加的Key：

当然，GitHub允许你添加多个Key。假定你有若干电脑，你一会儿在公司提交，一会儿在家里提交，只要把每台电脑的Key都添加到GitHub，就可以在每台电脑上往GitHub推送了。

在github上创建仓库

目前，在GitHub上的这个learngit（上面创建的仓库名）仓库还是空的，GitHub告诉我们，可以从这个仓库克隆出新的仓库，也可以把一个已有的本地仓库与之关联，然后，把本地仓库的内容推送到GitHub仓库。

现在，我们根据GitHub的提示，在本地的learngit仓库下运行命令：

$ git remote add origin git@github.com:michaelliao/learngit.git

请千万注意，把上面的michaelliao替换成你自己的GitHub账户名，否则，你在本地关联的就是我的远程库，关联没有问题，但是你以后推送是推不上去的，因为你的SSH Key公钥不在我的账户列表中。

添加后，远程库的名字就是origin，这是Git默认的叫法，也可以改成别的，但是origin这个名字一看就知道是远程库。

git remote add new-origin git@github.com:zhangjincong99/music.git

下一步，就可以把本地库的所有内容推送到远程库上：

$ git push -u origin master

git push -u new-origin master

把本地库的内容推送到远程，用git push命令，实际上是把当前分支master推送到远程。

由于远程库是空的，我们第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令。

现在，远程库已经准备好了，下一步是用命令git clone克隆一个本地库：

$ git clone [git@github.com:zhangjincong99/gitskills.git](mailto:git@github.com:zhangjincong99/gitskills.git)

Cloning into 'gitskills'...

remote: Counting objects: 3, done.

remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 3

Receiving objects: 100% (3/3), done.

注意把Git库的地址换成你自己的，然后进入gitskills目录看看，已经有README.md文件了：

$ cd gitskills

$ ls

README.md

git clone [git@github.com:zhangjincong99/mic.git](mailto:git@github.com:zhangjincong99/mic.git)

如果有多个人协作开发，那么每个人各自从远程克隆一份就可以了。

你也许还注意到，GitHub给出的地址不止一个，还可以用https://github.com/michaelliao/gitskills.git这样的地址。实际上，Git支持多种协议，默认的git://使用ssh，但也可以使用https等其他协议。

现在有了分支，就不用怕了。你创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性合并到原来的分支上，这样，既安全，又不影响别人工作。

其他版本控制系统如SVN等都有分支管理，但是用过之后你会发现，这些版本控制系统创建和切换分支比蜗牛还慢，简直让人无法忍受，结果分支功能成了摆设，大家都不去用。

但Git的分支是与众不同的，无论创建、切换和删除分支，Git在1秒钟之内就能完成！无论你的版本库是1个文件还是1万个文件。

比如新提交一次后，dev指针往前移动一步，而master指针不变：

假如我们在dev上的工作完成了，就可以把dev合并到master上。Git怎么合并呢？最简单的方法，就是直接把master指向dev的当前提交，就完成了合并：

首先，我们创建dev分支，然后切换到dev分支：

$ git checkout -b dev

Switched to a new branch 'dev'

git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：

$ git branch dev

$ git checkout dev

Switched to branch 'dev'

然后，用git branch命令查看当前分支：

$ git branch

\* dev

master

git branch命令会列出所有分支，当前分支前面会标一个\*号。

然后，我们就可以在dev分支上正常提交，比如对readme.txt做个修改，加上一行：

现在，dev分支的工作完成，我们就可以切换回master分支：

$ git checkout master

现在，我们把dev分支的工作成果合并到master分支上：

$ git merge dev

git merge命令用于合并指定分支到当前分支。合并后，再查看readme.txt的内容，就可以看到，和dev分支的最新提交是完全一样的。

注意到上面的Fast-forward信息，Git告诉我们，这次合并是“快进模式”，也就是直接把master指向dev的当前提交，所以合并速度非常快。

当然，也不是每次合并都能Fast-forward，我们后面会讲其他方式的合并。

合并完成后，就可以放心地删除dev分支了：

$ git branch -d dev

删除后，查看branch，就只剩下master分支了：

$ git branch

switch

我们注意到切换分支使用git checkout <branch>，而前面讲过的撤销修改则是git checkout -- <file>，同一个命令，有两种作用，确实有点令人迷惑。

实际上，切换分支这个动作，用switch更科学。因此，最新版本的Git提供了新的git switch命令来切换分支：

创建并切换到新的dev分支，可以使用：

$ git switch -c dev

直接切换到已有的master分支，可以使用：

$ git switch master

使用新的git switch命令，比git checkout要更容易理解。

Git鼓励大量使用分支：

查看分支：git branch

创建分支：git branch <name>

切换分支：git checkout <name>或者git switch <name>

创建+切换分支：git checkout -b <name>或者git switch -c <name>

合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name>

准备合并dev分支，请注意--no-ff参数，表示禁用Fast forward：

$ git merge --no-ff -m "merge with no-ff" dev