新建本地仓库：

创建一个目录，Git Bash here打开命令行，**通过git init命令把这个目录变成仓库**

（注：所有的版本控制系统，其实只能跟踪文本文件的改动，图片、视频这些二进制文件无法跟踪变化，Windows的Word就是二进制格式，此外强烈建议文本统一使用UTF-8编码）

不要使用Windows自带的记事本编辑任何文本文件，建议使用Notepad++，把Notepad++的默认编码设置为UTF-8 without BOM即可

**git add**：告诉Git，**把文件添加到仓库提交列表**

**git add . 一次性添加所有改变了的文档，也可以git add后面跟多个文件名，用空格隔开**

**git commit：**告诉Git，**把刚才add的文件提交到仓库**，注意**commit的时候一定要加上-m “xxx”**

其中-m后面双引号里是对于本次提交的说明

**git status：查看当前仓库状态**，可以知道哪些文件被修改过了，哪些文件还未被跟踪

**git diff：查看具体修改了什么内容**，后面跟上你要查看的文件的名字，-开头的行代表只在源文件出现，+开头的行代表只在目标文件出现，空格开头的行代表在两个文件都出现

**git log：**查看我们版本控制的历史记录，从最近到最远的提交日志（如果觉得输出的信息太多，可以在后面加上 --pretty=oneline），显示出来的每一次提交都对应着一个commit id，可以通过这个id回退版本（输入前几位即可）

在Git中，HEAD代表当前最新的版本，HEAD^代表上一个版本，HEAD^^代表着上上一个版本，依此类推

**git reset：**把当前版本的文件**回退到上一个版本**，例如：git reset --hard HEAD^

（如果修改只是添加到了暂存区，还没有提交，git reset HEAD <file>可以把暂存区的修改撤销掉（unstage），重新放回工作区）

**git reflog：记录你的每一次关键命令**（提交，回退，删除）

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。

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实际就是往master分支上提交更改

Git跟踪并管理的是修改，而非文件

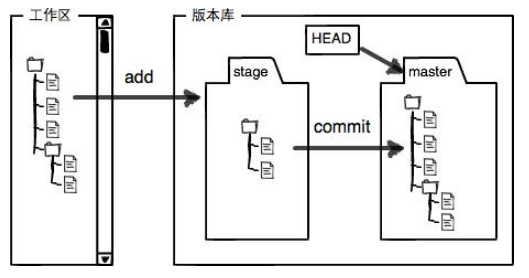
**git checkout -- <file>：丢弃工作区的修改**

工作区（Working Directory）是指在电脑里能看到的目录，文件夹就是一个工作区

此时有两种情况：

1.文件自修改后还未被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态

2.文件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



注：git checkout -- file命令中的--很重要，没有--，就变成了“切换到另一个分支”的命令

**git rm <file>：**从版本库中删除文件，删除以后要commit

（如果是不小心删错了文件，可以把误删的文件恢复到最新的版本，使用

git checkout -- <file>命令，该命令其实是用版本库里的版本替换工作区的版本）

本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以需要设置SSH：

打开Git Bash，创建SSH Key：$ ssh-keygen -t rsa -C "youremail@example.com"

（填写自己的邮箱地址，然后会在User目录下的主用户目录下看到.ssh文件夹）

然后Add SSH Key，填上任意密钥的名称，在文本框里粘贴id\_rsa.pub文件的内容即可添加

SSH Key主要是为了让GitHub可以识别出是你本人而不是别人冒充你推送的，你也可以在多个电脑中添加多个Key，这样每台电脑都可以往GitHub上推送了

在GitHub上找到“Create a new repo”按钮，创建一个新的远程仓库

在本地仓库下运行命令：**$ git remote add origin git@github.com:michaelliao/learngit.git**

**从而把本地库和远程库关联起来**（注意换成自己的GitHub账户名）

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

**$ git push -u origin master：把本地库的内容推送到远程库**（origin是仓库名，master是分支名），-u参数是第一次推送master分支时使用的，这样次Git会把本地的master分支和远程的master分支关联起来，以后的推送或者拉取就可以简化命令，只要本地做了提交就可以直接$ git push origin master

**$ git clone git@github.com:michaelliao/gitskills.git：把远程库的内容克隆一份到本地**（注意换成自己的Git库的地址，地址尽量使用ssh，因为比https地址速度要快）

**$ git checkout -b dev：创造一个dev的分支并且切换到dev**，其相当于两条指令：

1. $ git branch dev

2. $ git checkout dev

**git branch：查看当前分支**，会列出所有的分支，前面带\*号的就是当前所处的分支

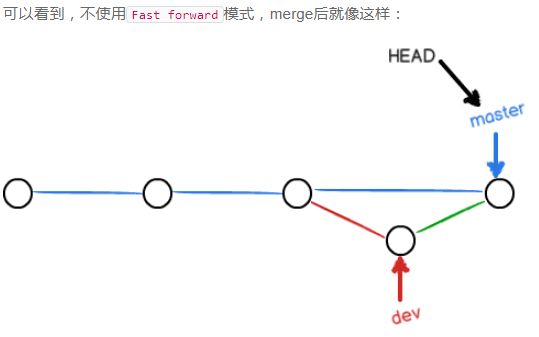
**git merge <branch>：把指定的分支的成果合并到当前的分支上**

（Fast-forward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）

**$ git branch -d <branch>：删除指定分支**

**$ git log --graph --pretty=oneline --abbrev-commit：查看分支历史**

**$ git merge --no-ff -m "merge with no-ff" <branch>：禁用快速合并模式，此时合并会提交一个新的commit，所以要加上-m参数**



分支策略：



标签是为了让每一个版本有一个更加好记的名字

**git tag <name>：创建一个新标签**

**git tag：查看所有标签**

**git show <tagname>：查看某一个标签的具体信息**

**$ git tag -d <tagname>：删除一个标签**

**git push origin <tagname>：推送一个标签到远程库**

**$ git push origin --tags：一次性推送全部未推送的标签**

如果忘了打标签，用git log找到历史提交的commit id，然后$ git tag <tag> <commit id>即可

还可以创建带有说明的标签，用-a指定标签名，-m指定说明文字：

例如$ git tag -a v0.1 -m "version 0.1 released" 1094adb

高级功能：

$ git stash：把当前工作现场“储藏”起来，等以后恢复现场后继续工作

（当手头工作没有完成时，先把工作现场git stash一下，然后去修复bug，修复后，再git stash pop，回到工作现场）

每添加一个新功能，最好新建一个feature分支，在上面开发，完成后合并，最后删除该feature分支

当从远程库clone时，默认情况下，你只能看到本地的master分支，如果要在dev分支上进行开发，使用命令：$ git checkout -b dev origin/dev，创建本地dev分支

git pull：把最新的提交从origin/dev抓下来，在本地合并，解决冲突，再推送

拉取的时候要先指定本地dev分支与远程origin/dev分支的链接：

$ git branch --set-upstream-to=origin/dev dev

多人协作的工作模式通常是这样：

1. 首先，可以试图用git push origin <branch-name>推送自己的修改；

2. 如果推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并；

3. 如果合并有冲突，则解决冲突，并在本地提交；

4. 没有冲突或者解决掉冲突后，再用git push origin <branch-name>推送就能成功！

5. 如果git pull提示no tracking information，则说明本地分支和远程分支的链接关系没有创建，用命令git branch --set-upstream-to <branch-name> origin/<branch-name>

使用github：Fork用来克隆别人的仓库到自己的远程库，然后clone到自己的本地库，修改了以后推送到自己的远程库，然后在github上发起一个pull request，看对方是否愿意接受你的合并请求