# 初始化 / 新建工作区

`git init`：初始化一个 Git **仓库**（repository），即把当前所在目录变成 Git 可以管理的仓库。

`git clone git@github.com:username/<repo name>.git`：从 GitHub 克隆一份远程库到本地，使用的是 SSH 协议

`git clone https://github.com/username/repo name.git`：使用 https 协议拉取远程仓库

就通常来说，如果只是在本地使用git进行版本控制，一般创建新的文件夹然后`git init`就好，如果是要连到GitHub上，就在GitHub创建好repo之后clone到本地文件夹，之后用`git push`和`git pull`进行与远程仓库的交互。

# 添加（add）文件，把文件添加到暂存区（stage）

`git add $filename`：$filename表示一个文件名。把文件添加到**暂存区**（stage），可被追踪纪录下来。**可多次使用这个方法来添加多个文件。**

`git add -A`：暂存所有的文件，包括新增加的、修改的和删除的文件。

`git add .`：暂存新增加的和修改的文件，**不包括已删除的文件。**即当前目录下所有文件。

`git add -u`：暂存修改的和删除的文件，**不包括新增加的文件。**

# 提交（commit） 文件

`git commit -m "本次提交说明"`：把暂存区所有文件修改提交到仓库的当前分支。

`git commit -am "本次提交说明"`：把所有**已经跟踪过的文件**暂存起来一并提交，从而跳过`git add`步骤。

**注意：如果**`git add <file>`某个文件，然后对这个文件进行修改了，再`git commit -m`是无法提交最新版本的文件的，但这时该文件**已经被跟踪过了，**因此可以使用`git commit -am`来提交最新版本。 它相当于`git add -u`与`git commit –m`两句操作合并为一句进行使用。
**

`git commit --amend`：**重新提交**，第二次提交将**代替**第一次提交的结果。最终只会有一个提交。

# Branch & checkout 分支。分支包括远程分支和本地分支

`git branch`： 列出**本地**当前所有分支。当前分支前面会标有一个 * 号。本地分支不一定包含项目的全部分支。

`git branch $branchName`：创建新分支

`git branch -r`：查看**远程**分支列表。

`git branch -a`：显示项目的**全部分支**，包含本地分支和远程分支。

`git branch -v`： 查看每一个分支的最后一次提交。

`git checkout $branchName`：切换分支。

`git checkout -b $branchName`： **创建**并**切换**到新的分支，相当于下面两条命令： `git branch $branchName` + `git checkout $branchName`。

`git checkout $branchName <file1> <file2>`：将分支下的 file1、file2 文件 合并到本分支（注意此时处在的分支**不是**$branchName） 。**这样做的话相当于只 merge 一部分文件**

`git checkout -- <file>`：取消对 filename 文件做的修改（ 建议使用`git restore`命令来代替）

`git rm --cached <file>`：将已经保存在暂存区的文件（已经add过的）丢弃（unstage）。不会删除本地文件。

`git rm <file>`：从版本库中删除该文件

**注意：新版本的 Git 引入了两个新命令`git switch`和`git restore`，用以替代现在的`git checkout`。换言之，`git checkout`将逐渐退出历史舞台。**

**在 Git 的新版本2.23中，有以下几种新用法：**

`git switch $branchName`：切换分支，等价于 `git checkout $branchName`

`git switch -c $branchName`.**切换并创建新的分支** 等价于 `git checkout -b $branchName`

`git restore $filename`： 取消该文件所做的修改，注意不能再恢复回来哦，要么就用`git stash`保留

`git merge $branchName` ： 合并**指定分支到当前**所在的分支。

`git branch -d $branchName` ： 删除分支（一般要在合并完分支之后）

`git branch -D $branchName`：强行删除分支，尤其适用于分支内容有了新的修改但还没有被合并的情况。

# Pull & push

`git push origin $branchName` ：$branchName 是分支名；把该分支上的所有本地提交推送到远程库对应的远程分支上。

`git checkout <分支> origin/<分支>`： 如果远程有某一个分支而本地没有，该命令将远程的这个分支迁到本地。

`git pull`： 从远程获取最新版本同时合并（merge）到本地

`git fetch`： 从远程分支拉取代码。

**注意：`git fetch`是将远程主机的最新内容拉到本地，用户在检查了以后决定是否合并到工作本机分支中。会拉取当前项目的所有分支的commit**

**而`git pull`则是将远程主机的最新内容拉下来后直接合并，即：`git pull = git fetch + git merge`，这样可能会产生冲突，需要手动解决。**

# Stash

**当前修改还没完成，或者暂时不想commit，然而需要做 pull，merge或者 checkout / switch到其他分支的操作，一般会用stash将修改暂时储藏。**

**stash操作是不随分支的，无论在哪个分支的stash，都可以应用到任何分支。**

`git stash`：把当前分支的工作现场**储存**起来，等以后恢复现场后继续工作。一般适用于还没有 commit 的分支代码。

`git stash save -a "message"`：功能同上， 同时添加一条 message。

`git stash list`：查看储存的工作现场纪录列表。

`git stash clear`：清空所有暂存区的 stash 纪录。

`git stash apply`：恢复最近 stash 过的工作现场，恢复后，stash 内容并不删除，可以用 git stash drop 命令来删除。apply 和 drop 后面都可以加上某一指定的 stash_id。

`git stash drop`：如上

`git stash pop`：相当于上面两条命令（`git stash apply + git stash drop`），恢复回到工作现场的同时把 stash 内容也删除了。

# Diff 比较差异

`git diff <file>`：比较当前文件和暂存区文件差异

`git diff <branch1> <branch2>`： 在两个分支之间比较，会详细地逐个列出文件的差异

`git diff <branch1> <branch2> --stat`：只列出两个分支间有差异的文件的文件名（相比于上一个命令了来说）

`git diff <branch1> <branch2> <file>`：比较两个分支中某一文件的差异，写在左边的分支是旧的，写在右边的是新的

`git diff <commit_id1> <commit_id2> --stat`： 比较两次提交之间的文件差异

在比较的界面用上下箭头前进、后退，按 q 退出比较

# Remote 远程仓库操作

`git remote`： 查看已经配置的远程仓库服务器，效果同 `git remote show`

`git remote -v`： 显示需要读写远程仓库使用的 Git 保存的简写与其对应的 URL。

`git push -u origin master`：关联后，使用该命令第一次推送 master 分支的所有内容，后续再推送的时候就可以省略后面三个参数了，其中参数 u 代表 **上游（upstream**）的意思。

# Log 查看日志

`git log`：查看项目的提交历史，会列出提交人和提交时间。常用来找到想要回退的版本号

`git reflog`：查看每一次命令操作记录。常用来找到未来的版本号（版本回退后）

`git log --graph`：输出分支合并图。

# Reset 版本回退

`git reset --hard <commit_id>`：将版本回退到对应的这个 commit_id 上。（只需要输入 commit id 的一部分让git知道是那次commit的即可）

**commit id 是每次执行 commit 命令后给项目生成的一个 unique 的 id 。不会出现多次 commit 属于同一 id 的情况。因此可以用来标识每次的提交。**

**可以通过`git log`命令查看以前提交的 commit id.**

**在版本回退以后是无法通过`git log`查看回退到的版本之后的版本的commit id，这时候要使用git reflog**