总结：

安装：git -version

# Git高层命令（CRUD）：

* 初始化新仓库 git init

解析：要对现有的某个项目开始用Git管理，只需到此项目所在的目录,执行git init

作用：初始化后，在当前目录下会出现一个名为.git的目录，所有Git需要的数据和资源都存放在这个目录中。不过目前，仅仅是按照既有的结构框架初始化好了里边所有的文件和目录，但我们还没有开始跟踪管理项目中的任何一个文件。

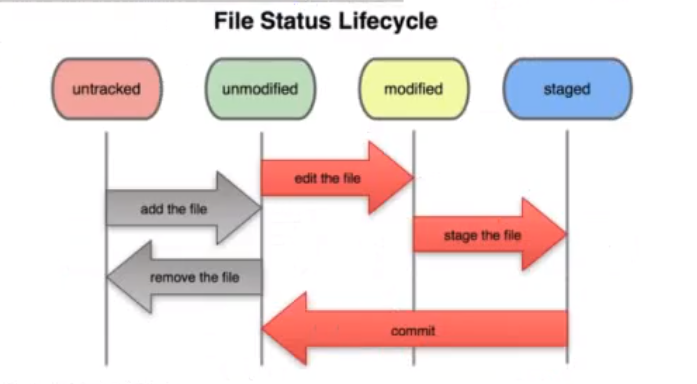
* 记录每次更新到仓库

工作目录下面的所有文件状态：**已跟综**或**未跟踪**

已跟综的文件是指本来就被纳入版本控制的文件，在上次快照中有它们的记录，工作一段时间后，它们的状态可能是**已提交**、**已修改**或**已暂存**。

所有其他文件都属于未跟踪文件。它们既没有上次更新时的快照，也不在当前的暂存区域。

初次克隆某个仓库时，工作目录中的所有文件都属于已跟综文件，且状态为已提交；在编辑某些文件之后，Git将这些文件标为已修改。我们逐步把这些修改过的文件放到暂存区域，直到最后一次性提交所有这些暂存起来的文件。使用Git时的文件状态变化周期如下所示：



注：已修改为红色、已暂存为绿色、已提交为空

* 检查当前文件状态 git status
* 克隆仓库后的文件

如果在克隆仓库之后立即执行此命令，会看到类似这样的输出：

On branch master

nothing to commit, working directory clean

这说明你现在的工作目录相当干净。换句话说，所有已跟综文件在上次提交后都未被更改过。此外，上面的信息还表明，当前目录下没有出现任何处于未跟踪的新文件，否则Git会在这里列出来。最后，该命令还显示了当前所在的分支时master，这是默认的分支名称，实际是可以修改的，现在先不用考虑。

* 未跟踪文件
* 基本操作
* 跟踪新文件（暂存）

命令：git add文件

作用：跟踪一个新文件

再次运行git status命令，会看到README文件已被跟踪，并处于暂存状态：

Changes to be committed

(use “git reset HEAD<file>…”to unstage)

new file README

只要在“Changes to to committed”这行下面的，就说明是已暂存状态。如果此时提交，那么该文件此时此刻的版本将被留存在历史记录中。在git add后面可以指明要跟踪的文件或目录路径。如果是目录的话，就说明要递归跟踪该目录下的所有文件。（译注：其实git add的潜台词就是把目标文件快照放入暂存区域，也就是add file into staged area，同时未曾跟踪的文件标记为已跟综。）

* 暂存已修改文件

现在README文件都已暂存，下次提交时就会一并提交到仓库。假设此时，你想要README里再加条注释，重新编辑存盘后，准备好提交。不过且慢，再运行git status看看：

On branch master

Changes to be committed:

(user “git reset HEAD <file>…” to unstage)

new file README

Changes not staged for commit:

(use “git add <file>…”to update what will be committed)

(use “git checkout -- <file>…”to discard changes in working directory)

Modified: README

README文件出现了两次！一次算已修改，一次算已暂存。Git暂存了git add的版本，已修改的文件需要再次git add。

* 查看已暂存和未暂存的更新

实际上git status的显示比较简单，仅仅是列出了修改过的文件，如果要查看具体修改了什么地方，可以用git diff命令。

git diff作用：①当前那些更新尚未暂存；②当前哪些已暂存准备提交（git diff –cached或git diff –staged【1.6.1以上版本】）

* 提交更新

当暂存区域已准备提交时，一定要确认将需要版本控制的文件添加（git add）到暂存区。

git commit：

这种方式会启动文本编辑器以便输入本次提交的说明：默认的提交信息及最后一次运行git status的输出，会放在注释里。另外开头还有一空行，用于输入提交说明。“#”号注释行可以删除，但留着易于“提交注释信息“的完整。

另外，也可以选择git commit -m “注释”。

注意：提交时的记录实际是暂存区的快照，任何未暂存（git add操作）的数据保持已修改状态，可以留作下一版本的管理。

* 跳过使用暂存区域

尽管使用暂存区的可以精心准备要提交的细节，但有时候这么做略显繁琐。Git提供了一个跳过使用暂存区的方法，只要在提交时给git commit加上-a选择，Git就会自动把所有跟踪过的文件暂存起来一并提交，从而跳过手动git add的步骤

注意：git commit -a 会提交已跟综数据（可以已修改但尚未暂存，但echo的未跟踪数据不能git commit -a）

* 移除文件

要从Git中移除某个文件，就必须要从已跟综文件清单中注册删除（从暂存区删除），然后提交。可以用git rm，此命令连带删除工作目录中指定的文件，这样有“删除的文件不会出现在未跟踪文件清单“的好处。

提交后，会记录删除操作。新版本删除了某文件，但老版本未变。

流程：*rm 文件→git add→git commit*或*git rm 文件→git commit*

* 文件改名

git mv相当于删除rm再新增echo再暂存git add，但git add时会显示rename；

* 查看历史纪录

git log：默认不带任何参数的话，git log会按提交时间列出所有的更新，从上往下显示从最新到最老的信息。

该命令会列出每个提交的SHA-1校验（哈希）、作者名、邮件名、提交时间、提交说明

* git init 初始化仓库
* git add ./ 将修改添加到暂存区【暂存区根据文件名识别，重名会覆盖】
* git commit 提交暂存区数据，会进入vim编辑器编辑注释信息
* git commit -a 提交包括未暂存的数据，会进入vim编辑器编辑注释信息
* git commit -m "" 提交暂存区数据，-m直接携带注释，不进入vim
* git commit -a -m “” 提交
* git status 查看文件的状态
* git ls-files -s 查看暂存区数据
* git diff 查看哪些修改该还没有暂存
* git diff --staged 查看哪些修改已暂存但还未提交
* git rm 路径+文件名 删除并添加到暂存区【单独rm 文件是指删除操作】
* git mv 老文件名 新文件名 改名并添加到暂存区【单独mv 文件是指改名操作】
* git log 打印历史
* git log –oneline 查看简化哈希+注释（带分支）
* git log –pretty=oneline 查看哈希+注释（带分支）

# Git高层命令（分支）：

* git branch 显示分支
* git branch 分支名 新建分支
* git branch 分支名 commitHash(哈希对象名) 新建分支并指向对应提交对象
* git checkout 分支名 切换分支
* git checkout -b 分支名 新建并指向该分支
* git branch -v 查看“每一个分支”的最后提交
* git branch -d 分支名 删除分支（不能自己删自己）
* git branch -D 分支名 强制删除分支（不能自己删自己）
* git log --oneline --decorate --graph --all 查看分支历史（注意双杠）
* git config --global alias.别名 命令名【注意双引号】
  + 例如：git config alias.lol “log --oneline --decorate --graph –all”

效果：执行后查看历史分支的命令为：git lol

* git merge：快进合并，将次分支与主分支快速合并【具体操作：将指针切换到主分支，从主分支快进到次分支(因为次分支是最新的操作)】
  + 假如合并时“版本冲突”：Vim或还原某个数据等修改、合并冲突文件后，重新提交commit就可以了

# Git高层命令（存储--栈）注意该功能仅用于本地，不会上传到git server

* 假如需要紧急修改某个BUG时，可以将目前的工作区操作加入到“栈”中，改完紧急问题后再将其拿出来：
  1. git stash list：查看当前栈中保存的数据【串联保存，遵循先进(保存)后出(取出)，后进先出的原则】；
  2. git stash：将未完成的修改保存到一个栈上【注：git stash save “注释”是带注释的加入】；
  3. git stash apply stash@{0}：取出（重新加载）某个栈数据【不跟指定储藏，将默认最新】；
  4. git stash drop stash@{0}：删除栈中的第0个数据
  5. git stash pop：重新加载栈数据到工作区并删除栈数据

# Git高层命令（撤销）

* git checkout –文件名 撤销工作区的操作
* git reset HEAD 文件名 撤销暂存区的操作