1.安装、配置环境变量；

2.git init在当前目录下创建本地空仓库（.git文件夹隐藏的）

3.修改.git下的config文件，配置上remote地址例如

[remote "origin"]

        url = <https://github.com/AimeeLee23/main-station.git>

4.git pull从远程仓库拉取代码到本地空仓库

拉取远程仓库：$ git pull [remoteName] [localBranchName]

推送远程仓库：$ git push [remoteName] [localBranchName]

查看远程仓库：$ git remote -v

如果想把本地的某个分支test提交到远程仓库，并作为远程仓库的master分支，或者作为另外一个名叫test的分支，如下：

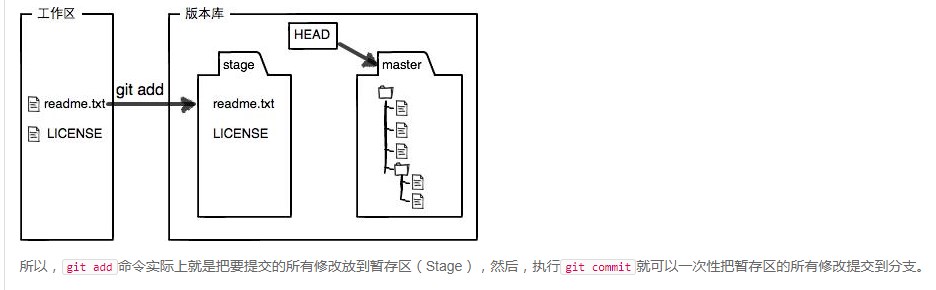
$git push origin test:master         // 提交本地test分支作为远程的master分支

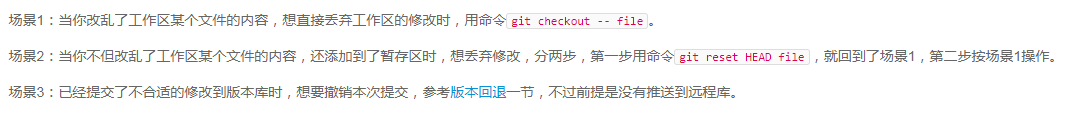
$git push origin test:test              // 提交本地test分支作为远程的test分支

Git本地仓库包含代码库还有历史库，在本地的环境开发就可以记录历史 而SVN的历史库存在于中央仓库，每次对比与提交代码都必须连接到中央仓库才能进行

这样的好处在于： 1、自己可以在脱机环境查看开发的版本历史 2、多人开发时如果充当中央仓库的Git仓库挂了，任何一个开发者的仓库都可以作为中央仓库进行服务

所有的版本控制系统，其实只能跟踪文本文件的改动，比如TXT文件，网页，所有的程序代码等等，Git也不例外。版本控制系统可以告诉你每次的改动，比如在第5行加了一个单词“Linux”，在第8行删了一个单词“Windows”。而图片、视频这些二进制文件，虽然也能由版本控制系统管理，但没法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是只知道图片从100KB改成了120KB，但到底改了啥，版本控制系统不知道，也没法知道。





版本回退

 git reset --hard HEAD^  退到上一个版本

 git reset --hard 00898f332 退到指定版本号的版本

1.只在工作区修改了文件内容，尚未add  直接用git checkout -- readme.txt

2.在工作区修改了，已经add到暂存区，但是还没commit，可以先git reset HEAD readme.txt,然后再git checkout -- readme.txt

3.修改后，已经add且已经commit，然后可以通过git reset --hard HEAD^ 回退到上一个版本

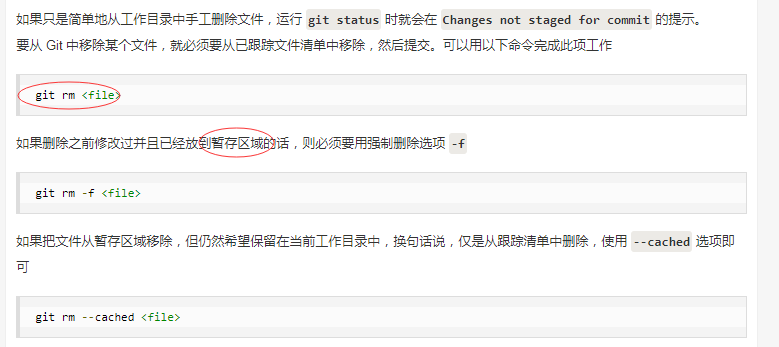
--------------------------------------------------------------------------

在工作区新增一个文件，且add+commit了，然后想要删除

git rm test.txt  删除一个文件

git commit -m "remove a test.txt"

在工作区新增一个文件，尚未add，然后可以直接rm test.txt删除即可；



在工作区新增一个文件，已经add，尚未commit，然后要删除，git rm -f file，然后git commit -m "delete file"，即可实现删除文件。这一步是要commit的

-------------------------------------------------------------------------------

git checkout其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原

git remote add origin git@github.com:anzhenzhenA123/learngit.git

$ git config --global user.name "John Doe"

$ git config --global user.email johndoe@example.com

git clone git@github.com:anzhenzhenA123/learngit.git(速度快建议使用)

git clone <https://github.com/michaelliao/gitskills.git>

（本地库命名也是gitskills）

git clone http://git.oschina.net/yiibai/git-start.git mygit-start

（本地库命名为mygit-start）

git clone是clone到当前命令的运行的目录，所以在clone前，先cd到要clone到的目录，然后再执行语句。 



查看分支：git branch

创建并切换到分支dev：git checkout -b dev

创建分支：git branch dev

切换分支：git checkout dev

合并分支dev到master：git merge dev

删除分支：git branch -d dev

当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。

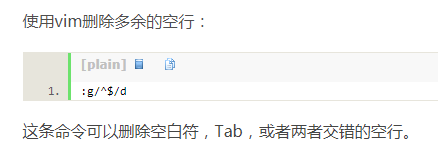
用git log --graph命令可以看到分支合并图

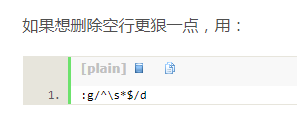
修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除；

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



* 命令git tag <name>用于新建一个标签，默认为HEAD，也可以指定一个commit id；
* git tag -a <tagname> -m "blablabla..."可以指定标签信息；
* git tag -s <tagname> -m "blablabla..."可以用PGP签名标签；
* 命令git tag可以查看所有标签。







git status –short或git status –s

$ git status -s

M README.md

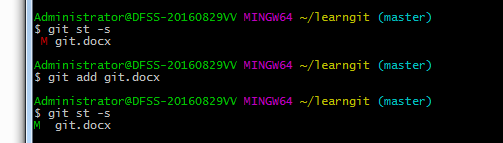
MM Rakefile

A lib/git.rb

M lib/simplegit.rb

?? LICENSE.txt

Shell



新添加的未跟踪文件前面有 ?? 标记（新添加的未git add的），新添加到暂存区中的文件前面有 A 标记，修改过的文件前面有 M 标记。 你可能注意到了 M 有两个可以出现的位置，出现在右边的 M 表示该文件被修改了但是还没放入暂存区（尚未执行git add），出现在靠左边的 M 表示该文件被修改了并放入了暂存区（执行过git add了）。 例如，上面的状态报告显示： README 文件在工作区被修改了但是还没有将修改后的文件放入暂存区,lib/simplegit.rb 文件被修改了并将修改后的文件放入了暂存区。 而 Rakefile 在工作区被修改并提交到暂存区后又在工作区中被修改了，所以在暂存区和工作区都有该文件被修改了的记录。

git diff，可能通常会用它来回答这两个问题：当前做的哪些更新还没有暂存？ 有哪些更新已经暂存起来准备好了下次提交？

Git 提供了一个跳过使用暂存区域的方式， 只要在提交的时候，给 git commit 加上 -a 选项，Git 就会自动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：git commit -a -m 'added new benchmarks'

