本质 管理文档都在.git中存放 通过git init命令来初始化

没有提交过的数据请慎重删除

Git 保存的不是文件的差异或变化量，而是一系列的文件快照

git config --global user.name "your name"  
git config --global user.email “[your\_email@youremail.com](mailto:your_email@youremail.com)”

git config --list 查看配置信息

去掉内容显示是否添加成功

两种方式进行初始化：

1.git clone:这是较为简单的一种初始化方式，当你已经有一个远程的Git版本库，只需要在本地克隆一份，例如'git clone git://github.com/someone/some\_project.git some\_project'命令就是将'git://github.com/someone/some\_project.git'这个URL地址的远程版 本库完全克隆到本地some\_project目录下面

2.Git init、git remote 初始化本地仓库。（第一次需要，检查代码）

如果你需要将它放到远程服务器上，可以在远程服务器上创建一个目录，并把可访问的URL记录下来，此时你就可以利用'git remote add'命令来增加一个远程服务器端，例如'git remote add origin git://github.com/someone/another\_project.git'这条命令就会增加URL地址为'git: //github.com/someone/another\_project.git'，名称为origin的远程服务器，以后提交代码的时候只需要使用 origin别名即可

基本命令：

1： git pull 从其他的版本库(既可以是远程的也可以是本地的)将代码更新到本地，例如：'git pull origin master'就是将origin这个版本库的代码更新到本地的master主枝

2：git add . 整个

3：git add file 单独文件

4：git commit –m “tag”（标签名）

5：git rm从当前的工作空间中和索引中删除文件，例如'git rm app/model/user.rb'

6：git push：将本地commit的代码更新到远程版本库中，例如'git push origin'就会将本地的代码更新到名为orgin的远程版本库中

7：git revert：还原一个版本的修改，必须提供一个具体的Git版本号，例如'git revert bbaf6fb5060b4875b18ff9ff637ce118256d6f20'，Git的版本号都是生成的一个哈希值

\* git revert HEAD 撤销前一次 commit

\* git revert HEAD^ 撤销前前一次 commit

8：git log 我们常用 -p 选项展开显示每次提交的内容差异，用 -2 则仅显示最近的两次更新：(有很多命令，具体查看)

修改：

修改最后一次提交

有时候我们提交完了才发现漏掉了几个文件没有加，或者提交信息写错了。想要撤消刚才的提交操作，可以使用 --amend 选项重新提交：

$ git commit --amend

此命令将使用当前的暂存区域快照提交。如果刚才提交完没有作任何改动，直接运行此命令的话，相当于有机会重新编辑提交说明，但将要提交的文件快照和之前的一样。

启动文本编辑器后，会看到上次提交时的说明，编辑它确认没问题后保存退出，就会使用新的提交说明覆盖刚才失误的提交。

如果刚才提交时忘了暂存某些修改，可以先补上暂存操作，然后再运行 --amend 提交：

$ git commit -m 'initial commit'

$ git add forgotten\_file

$ git commit --amend

上面的三条命令最终只是产生一个提交，第二个提交命令修正了第一个的提交内容。

取消修改

Git reset HEAD [filename] 已经提交到缓存区

Git checkout -- [filename] 在工作区

git status (检查当前操作到哪里了)

git diff 此命令比较的是工作目录中当前文件和暂存区域快照之间的差异，也就是修改之后还没有暂存起来的变化内容。

git diff --staged 若要看已经暂存起来的文件和上次提交时的快照之间的差异

git rm [-f][--cached] filename

另外一种情况是，我们想把文件从 Git 仓库中删除（亦即从暂存区域移除），但仍然希望保留在当前工作目录中。换句话说，仅是从跟踪清单中删除。比如一些大型日志文件或者一堆.a 编译文件，不小心纳入仓库后，要移除跟踪但不删除文件，以便稍后在 .gitignore 文件中补上，用 --cached 选项即可：

$ git rm --cached readme.txt

分支：Git 中的分支，其实本质上仅仅是个指向 commit 对象的可变指针。

git branch –a 查看分支

git branch version1.0 创建分支

HEAD指向当前工作的分支

git checkout version1.0转换分支 （git checkout –b version 创建并切换）

git merge version1.0 合并分支（在master分支下）

git branch –d version1.0 删除分支

分支工作方式：

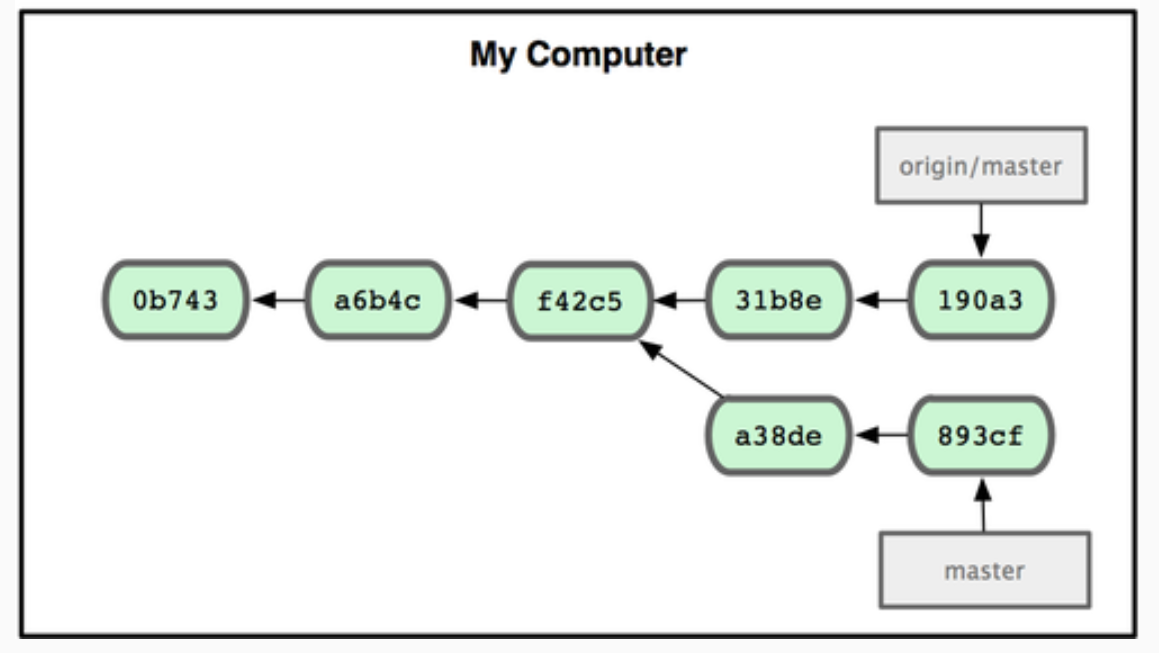
长期分支

特性分支

git help

远程分支：

git fetch origin master来同步远程服务器上的数据到本地。（如果跟你本地不一样会产生分支）



git rebase master

还有另外一个选择：你可以把在 C3 里产生的变化补丁在 C4 的基础上重新打一遍。在 Git 里，这种操作叫做\_衍合（rebase）\_。有了 rebase 命令，就可以把在一个分支里提交的改变移到另一个分支里重放一遍。

git pull origin master

远程

git remote add origin [git@github.com:linan../learn.git](mailto:git@github.com:linan../learn.git) 把远程仓库命名为origin

git remote show origin 显示和哪个远程的库链接

git remote –v

$ git fetch [remote-name]

此命令会到远程仓库中拉取所有你本地仓库中还没有的数据。

git push origin master

问题（Non-fast-forward）的出现原因在于：git仓库中已经有一部分代码，所以它不允许你直接把你的代码覆盖上去。于是你有2个选择方式：

1，强推，即利用强覆盖方式用你本地的代码替代git仓库内的内容

git push -f

2，先把git的东西fetch到你本地然后merge后再push

$ git fetch

$ git merge

这2句命令等价于

1.$ git pull

可是，这时候又出现了如下的问题：

上面出现的 [branch "master"]是需要明确(.git/config)如下的内容

[branch "master"]

remote = origin

merge = refs/heads/master

这等于告诉git2件事:

1，当你处于master branch, 默认的remote就是origin。

2，当你在master branch上使用git pull时，没有指定remote和branch，那么git就会采用默认的remote（也就是origin）来merge在master branch上所有的改变

如果不想或者不会编辑config文件的话，可以在bush上输入如下命令行：

1.$ git config branch.master.remote origin

2.$ git config branch.master.merge refs/heads/master

之后再重新git pull下。最后git push你的代码吧。it works now~

Github gitlab 需要配置ssh

$ ssh-keygen -t rsa -C "user\_test@user.test"

一路回车即可

密钥生成成功！密钥分成两个文件，一个私钥（id\_rsa）、一个公钥（id\_rsa.pub）。私钥保存在你的电脑上，公钥交项目负责人添加到服务器上。用户必须拥有与服务器公钥所配对的私钥，才能访问服务器上的代码库。

【注意！】为了项目代码的安全，请妥善保管你的私钥！因为一旦私钥外泄，将可能导致服务器上的代码被泄漏！

接下来将公钥提交给gitlab服务器即可。打开C:\Users\Administrator\.ssh\id\_rsa.pub，将里面的内容复制到“add an SSH key”网页上的key文本框中，Title填写为你的邮箱账号，这里为user\_test@user.test，保存后即添加成功。

忽略某些文件

一般我们总会有些文件无需纳入 Git 的管理，也不希望它们总出现在未跟踪文件列表。通常都是些自动生成的文件，比如日志文件，或者编译过程中创建的临时文件等。我们可以创建一个名为 .gitignore 的文件，列出要忽略的文件模式。来看一个实际的例子：

$ cat .gitignore

\*.[oa]

\*~

第一行告诉 Git 忽略所有以 .o 或 .a 结尾的文件。一般这类对象文件和存档文件都是编译过程中出现的，我们用不着跟踪它们的版本。第二行告诉 Git 忽略所有以波浪符（~）结尾的文件，许多文本编辑软件（比如 Emacs）都用这样的文件名保存副本。此外，你可能还需要忽略 log，tmp 或者 pid 目录，以及自动生成的文档等等。要养成一开始就设置好 .gitignore 文件的习惯，以免将来误提交这类无用的文件。

文件 .gitignore 的格式规范如下：

•所有空行或者以注释符号 ＃ 开头的行都会被 Git 忽略。

•可以使用标准的 glob 模式匹配。 \* 匹配模式最后跟反斜杠（/）说明要忽略的是目录。 \* 要忽略指定模式以外的文件或目录，可以在模式前加上惊叹号（!）取反。

所谓的 glob 模式是指 shell 所使用的简化了的正则表达式。星号（\*）匹配零个或多个任意字符；[abc] 匹配任何一个列在方括号中的字符（这个例子要么匹配一个 a，要么匹配一个 b，要么匹配一个 c）；问号（?）只匹配一个任意字符；如果在方括号中使用短划线分隔两个字符，表示所有在这两个字符范围内的都可以匹配（比如[0-9] 表示匹配所有 0 到 9 的数字）。

我们再看一个 .gitignore 文件的例子：

# 此为注释 – 将被 Git 忽略

\*.a # 忽略所有 .a 结尾的文件

!lib.a # 但 lib.a 除外

/TODO # 仅仅忽略项目根目录下的 TODO 文件，不包括 subdir/TODO

build/ # 忽略 build/ 目录下的所有文件

doc/\*.txt # 会忽略 doc/notes.txt 但不包括 doc/server/arch.txt

eclipse git:

安装问题，不要随便删除security的content，如果删除了重启eclipse

Pull的问题就是在Git Repositories View中的Remotes中新建一个Remote（fetch）

.git就建立在父包中就ok，不要到处换地方

最后，eclipse的这个插件真难用。

Head表示当前版本，上一个版本是head^,上上个版本是head^^ ,上一百个版本head~100

回溯到上一个版本 git reset -- hard HEAD^

如果还想该回来，只要上面的命令行窗口没关，找到commit id 3653243如，git reset - -hard 365…版本号没必要写全

如果命令行关了，可以通过git reflog来查看，reflog记录你的每一次命令

工作区，版本库--暂存区(stage，head->master)

Commit只能提交在add到stage中的修改

Git checkout - - file 丢弃工作区的修改：如果是没有add 回到和版本库一致的状态

如果已经add到暂存区，又做了修改，撤销到add到暂存区后的状态

总之回到最近一次commit或add的状态

Git reset Head file 可以把暂存区的修改撤销掉，重新放回工作区。当我们用HEAD的时候表示是最新版本

删除文件: git rm file

如果是删错了，用git checkout - - file

创建SSHKEY ssh-keygen –t rsa –C “email”

在用户主目录中找到.ssh目录，里面有id\_rsa and id\_rsa.pub 前一个是私钥，不透露，后者是私钥

Git内部原理

<http://www.open-open.com/lib/view/open1328070620202.html>

http://rongjih.blog.163.com/blog/static/33574461201121611154627