**1 Git简介**

Git是用C语言开发的分布版本控制系统。版本控制系统可以保留一个文件集合的历史记录，并能回滚到另外一个状态（历史记录状态）。对 于任何一个文件,在 Git 内都只有三种状态:已提交(committed),已修改 (modified)和已暂存(staged)。已提交表示该文件已经被安全地保存在本地数据库中了;已修改表示修改了某个文件,但还没有提交保存;已暂 存表示把已修改的文件放在下次提交时要保存的清单中。

**2 Git命令基本格式**

一般情况下，Git的命令都是git [command] [option] [argument]格式。其中 command指的是某种操作命令，比如config,mv,rm,pull,push等等。而option和 argument指的是操作命令后面的具体参数，比如：

git config --golobal user.name="hic"

其中，config指的就是配置命令,接着后面跟上配置的具体参数，其中–global指的是对全局配置生效，后面的user.name就是对用户名进行设置。

**3 配置文件**

配置文件是从事所有所有工作的基础，在这里配置文件决定了工作时候的用户，邮箱，以及默认编辑器等等常用的配置参数，下面就先说一说配置文件。

在Git中，配置文件有三大类，分别放在不同的位置。

**/etc/gitconfig**

系统中所有用户都普遍适用的配置。git config –system就是用来配置这个文件的。

**~/.gitconfig**

用户目录下的配置文件，该文件只适用于该用户。git config –global就是用来配置这个文件的。

**~/\*/.git/config**

这是当前项目的配置文件，仅对该项目适用。

注意：每一级别的配置，都会覆盖上层的相同配置。

**3.1 配置信息**

用户名设置

git config --system user.name "hic"

git config --global user.name "hic" git config user.name "hic"

电子邮件

git config --global user.email "hic@gmail.com"

设置文本编辑器，默认为vi，这里设置为emacs

git config --global core.editor emacs

查看配置信息

git config --list

如果要查看某个特定的环境变量，只要把它的名字放到最后即可

git config user.name

**4 选择工作目录**

选择一个工作目录，开始工作，同时我们添加一些文件和目录。

~$ mkdir test

~$ cd test/

~/test$ ls

~/test$ echo "Hello world! " > README

~/test$ echo "This is a test! " > file

**5 初始化**

~/test$ git init Initialized empty Git repository in /home/hic/test/.git/

~/test$ la file .git README

执行这条命令之后，在test目录下会出现一个隐藏的.git目录，该目录会保存工程中所有的信息。

**6 Git中文件的三种状态**

在Git中，文件只有三种状态，已修改，已暂存和已提交。这三个文件分别放在工作目录，暂存区域和git目录。当一 个文件修改完毕之后，它仍然在工作目录；只有当它被add之后，才会进入暂存区域，文件状态变为已暂存；最后，当文件被 commit之后，它就会进去git目录区域。这三个文件状态非常重要，请慢慢理解。

**7 add命令暂存已修改文件**

接着上面的初始化。当初始化之后，在test目录下，有README，file和.git/三个文件或目录，其中.git/是在初始化之后生成的。

此时在test中的文件都是存在于工作目录区域，尚未添加到暂存区。使用下面的命令可以查看到如果此时我要修改README文件。

~/test$ git status

# On branch master #

# Initial commit #

# Untracked files:

# (use "git add ..." to include in what will be committed) #

# README

# file nothing added to commit but untracked files present (use "git add" to track)

你会发现，README和file文件均是Untracked。接着使用add命令将它们添加到暂存区。

~/test$ git add .

~/test$ git status

# On branch master #

# Initial commit #

# Changes to be committed:

# (use "git rm --cached ..." to unstage) #

# new file: README

# new file: file #

此时的文件均已放在暂存区，其中git add . 就是将本目录下所有文件暂存。

**8 commit命令提交已暂存文件**

暂存区其实更像是一个缓冲区域，只有将文件commit到git目 录区域才表示真正的提交。其中commit命令提交的时候，需要对提交内容进行说明，有两种方法，一种是使用 -m "Message" 指令，这适用与比较短小的信息；另外一种就是使用编辑器进行编辑，前面设定的编辑器就是为它服务的，默认编辑器是vi，不过我这里设定的是emacs。

在这里，我使用-m参数。其中-a选项，表示提交所有在暂存区的文件，当然你也可以使用 git commit file README这样的方式来提交。

~/test$ git commit -a -m "hahah"

[master (root-commit) 81fde76] hahah 2 files changed, 3 insertions(+) create mode 100644 README create mode 100644 file

这是再查看文件状态，就会发现没有可以提交的文件了。

~/test$ git status

# On branch master nothing to commit (working directory clean)

**9 分支**

如果事先没有作出任何的更改，一个项目是没有分支的，它只有一条主线，例如：

~/test$ git branch \* master

这里master就是主干，就是整个开发的流程，它前面的星号表示当前开发的流程。

如果在开发过程中，我们突然需要添加某个功能，或者打上某个补丁，可以在 master主干上添加一个分支，比如host，例如:

~/test$ git checkout -b host

Switched to a new branch 'host'

~/test$ git branch

\* host master

这个时候，你可以清楚的看到星号已经在host头上，表示当前已经切换到host上，你可以开发你的补丁，而不会打乱主干的开发。

如果你在开发过程中想要回到主干，可以使用checkout命令进行切换。

~/test$ git checkout master

Switched to branch 'master'

此时，你又回到主干了。

**10 合并**

当某个分支的开发结束后，你会需要将其合并到主干上，从而集中精力进行主干的开发。合并前只需要将指针切换到主干，即master，然后使用merge命令。

~/test$ git checkout host Switched to branch 'host'

~/test$ echo "lsdknl" >> file

~/test$ git add file

~/test$ git commit file -m "add somthing"

[host 0dacfd9] add somthing 1 file changed, 1 insertion(+)

~/test$ git checkout master Switched to branch 'master'

~/test$ git merge host Updating 81fde76..0dacfd9 Fast-forward file | 1 + 1 file changed, 1 insertion(+)

此时的host分支已经失去作用了，可以将删除。

~/test$ git branch -d host Deleted branch host (was 0dacfd9).

**11 远程库操作**

**11.1 查看当前远程库**

git remote

**11.2 添加远程库**

git remote add emacs git://github.com/lishuo/emacs

**11.3 从远程库抓取数据**

git fetch [remote-name]

**11.4 推送数据到远程仓库**

git push [remote-name] [branch-name]

**11.5 查看远程仓库**

git remote [remote-name]

**11.6 远程仓库的删除和重命名**

git remote rm [remote-name] git remotw rename form-name to-name