My Git:

邮箱：2805325601@qq.com

密码：h\*7

用户名：Hertz11

**Git 教程**

<http://www.runoob.com/git/git-tutorial.html>

**Git 工作流程**

直接采用**vim**的指令 。

一般工作流程如下：

克隆 Git 资源作为工作目录。

在克隆的资源上添加或修改文件。

如果其他人修改了，你可以更新资源。

在提交前查看修改。

提交修改。

在修改完成后，如果发现错误，可以撤回提交并再次修改并提交。

**Git 工作区、暂存区和版本库**

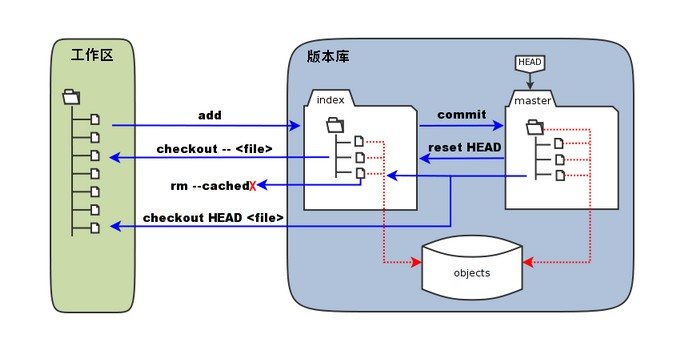
我们先来理解下Git 工作区、暂存区和版本库概念

**工作区：**就是你在电脑里能看到的工作目录。

**暂存区：**英文叫stage, 或index。一般存放在 ".git目录下" 下的index文件（.git/index）中，所以我们把暂存区有时也叫作索引（index）。

**版本库：**工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。

下面这个图展示了工作区、版本库中的暂存区和版本库之间的关系：



图中左侧为工作区，右侧为版本库。在版本库中标记为 "index" 的区域是暂存区（stage, index），标记为 "master" 的是 master 分支所代表的目录树。

图中我们可以看出此时 "HEAD" 实际是指向 master 分支的一个"游标"。所以图示的命令中出现 HEAD 的地方可以用 master 来替换。

图中的 objects 标识的区域为 Git 的对象库，实际位于 ".git/objects" 目录下，里面包含了创建的各种对象及内容。

当对工作区修改（或新增）的文件执行 "git add" 命令时，暂存区的目录树被更新，同时工作区修改（或新增）的文件内容被写入到对象库中的一个新的对象中，而该对象的ID被记录在暂存区的文件索引中。

当执行提交操作（git commit）时，暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树。

当执行 "git reset HEAD" 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。

当执行 "git rm --cached <file>" 命令时，会直接从暂存区删除文件，工作区则不做出改变。

当执行 "git checkout ." 或者 "git checkout -- <file>" 命令时，会用暂存区全部或指定的文件替换工作区的文件。这个操作很危险，会清除工作区中未添加到暂存区的改动。

当执行 "git checkout HEAD ." 或者 "git checkout HEAD <file>" 命令时，会用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件。这个命令也是极具危险性的，因为不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改动。

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

**Git环境搭建**

**Git 安装**

Windows 平台上安装：

安装包下载地址：<http://msysgit.github.io/>

<https://git-scm.com/download/win>

**准备工作**

在开始菜单里找到"Git"->"Git Bash"，会弹出 Git 命令窗口，你可以在该窗口进行 Git 操作。

**Git 配置**

Git 提供了一个叫做 git config 的工具，专门用来配置或读取相应的工作环境变量。

这些环境变量，决定了 Git 在各个环节的具体工作方式和行为。这些变量可以存放在以下三个不同的地方：

/etc/gitconfig 文件：系统中对所有用户都普遍适用的配置。若使用 git config 时用 --system 选项，读写的就是这个文件。

~/.gitconfig 文件：用户目录下的配置文件只适用于该用户。若使用 git config 时用 --global 选项，读写的就是这个文件。

当前项目的 Git 目录中的配置文件（也就是工作目录中的 .git/config 文件）：这里的配置仅仅针对当前项目有效。每一个级别的配置都会覆盖上层的相同配置，所以 .git/config 里的配置会覆盖 /etc/gitconfig 中的同名变量。

**用户信息**

配置个人的用户名称和电子邮件地址：

$ git config --global user.name "Hertz"

$ git config --global user.email 2805325601@qq.com

如果要在某个特定的项目中使用其他名字或者电邮，只要去掉 --global 选项重新配置即可，新的设定保存在当前项目的 .git/config 文件里。

**文本编辑器**

设置Git默认使用的文本编辑器, 一般可能会是 Vi 或者 Vim。

 Emacs可以重新设置：$ git config --global core.editor emacs

**差异分析工具**

还有一个比较常用的是，在解决合并冲突时使用哪种差异分析工具。比如要改用 vimdiff 的话：

$ git config --global merge.tool vimdiff

**查看配置信息**

要检查已有的配置信息，可以使用 git config --list 命令：

$ git config --list

这些配置我们也可以在 **~/.gitconfig** 或 **/etc/gitconfig** 看到，如下所示：

vim ~/.gitconfig

也可以直接查阅某个环境变量的设定

$ git config user.name

检查版本

[root@Git git-2.7.3]# git --version

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

**Git 基本操作**

**Git 创建仓库**

使用当前目录作为Git仓库，我们只需使它初始化。

git init

使用我们指定目录作为Git仓库。

git init <file\_name>

查看：ls -a //

**基本快照**

如果当前目录下有几个文件想要纳入版本控制，需要先用 git add 命令告诉 Git 开始对这些文件进行跟踪

$ git add \*.c
$ git add readme.txt

然后提交并添加信息：
$ git commit -m '<message>'

**从仓库中拷贝项目**

使用 git clone 从现有 Git 仓库中拷贝项目

git clone <repo>

需要克隆到指定的目录

git clone <repo> <dest\_directory>

**参数说明：**

**repo:**Git 仓库。

**directory:**本地目录。

比如，要拷贝某个 Git 代码仓库 Grit，可以用下面的命令：

$ git clone git://github.com/schacon/grit.git

执行该命令后，会在当前目录下创建一个名为grit的目录，其中包含一个 .git 的目录，用于保存下载下来的所有版本记录。

如果要自己定义要新建的项目目录名称，可以在上面的命令末尾指定新的名字：

$ git clone git://github.com/schacon/grit.git my\_name

**查看项目状态**

查看缓冲区状态，查看是否存在需要commit的内容。

查看在你上次提交之后是否修改或增加过文件：

git status

git status -s // s表示简短输出

**查看区别**

显示已写入缓存与已修改但尚未写入缓存的改动的区别

尚未缓存的改动：git diff

查看已缓存的改动： git diff --cached

查看已缓存的与未缓存的所有改动：git diff HEAD

显示摘要而非整个 diff：git diff --stat

**提交缓冲到仓库**

缓存区内容添加到仓库中

git commit -m 'commit\_message'

Git 也允许用 -a 选项跳过git add 提交缓存这一步

git commit -a ‘commit\_message’

git commit -am ‘commit\_message’

Git 为你的每一个提交都记录你的名字与电子邮箱地址，所以第一步需要配置用户名和邮箱地址。

$ git config --global user.name 'runoob'

$ git config --global user.email [test@runoob.com](mailto:test@runoob.com)

**取消缓冲区内容**

取消已缓存的内容

$ git reset HEAD -- file\_name

**移除某个文件**

要从 Git 中移除工作目录某个文件，就必须要从已跟踪文件清单中移除，然后提交。

git rm <file>

如果删除之前修改过并且已经放到暂存区域的话，则必须要用强制删除选项 -f

git rm -f <file>

如果把文件从缓冲区域移除，但仍然希望保留在当前工作目录中，使用

git rm --cached <file>

可以递归删除

git rm –r \*

**移动文件**

移动或重命名一个文件、目录、软连接

git mv old\_name new\_name

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

**分支管理**

**创建分支**

创建一个分支

git branch (branch\_name)

列出当前的分支

git branch

**切换分支**

git checkout (branch\_name)

git checkout -b (branch\_name) //创建新分支并立即切换到该分支下

当你切换分支的时候，Git 会用该分支的最后提交的快照替换你的工作目录的内容， 所以多个分支不需要多个目录。

将某个文件恢复到某个版本时的状态

git checkout id序列号 文件名

将所有的文件都恢复到某个版本

git reset --hard id序列号

**合并分支**

将某个分支合并到主分支上：

git merge (branch\_name)

你可以多次合并到统一分支， 也可以选择在合并之后直接删除被并入的分支。

**合并冲突**

将某一个分支合并到 "master" 分支，可能会出现合并冲突，接下来我们需要手动去修改它。改完后可以用 git add 主动提交到缓冲区，要告诉 Git 文件冲突已经解决，最后git commit。

**删除分支**

git branch -d (branch\_name)

**删除文件后如何恢复**

git status

用reset命令，先将abbr\_data这个文件找回来

git reset HEAD 'file-name' // 将它从缓冲区撤销到工作区

git status

再chekout一次即可

git checkout -- 'file-name'

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

**查看提交历史**

使用 Git 提交了若干更新之后，又或者克隆了某个项目，想回顾下提交历史。

git log

查看历史记录的简洁信息

git log --oneline

查看历史记录的拓扑图结构

git log --oneline --graph

--reverse参数来逆向显示所有日志

$ git log --reverse --oneline

--author=Linus查找指定用户的提交日志

$ git log --author=Linus --oneline -5

--since 和 --before、--until 和 --after指定日期

--no-merges 选项以隐藏合并提交

git log --oneline --before={3.weeks.ago} --after={2010-04-18} --no-merges

--decorate查看详细信息（tag信息）

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

**打标签tag**

当达到一个重要的阶段，可以给特别的快照打上标签。

创建带注解的标签

git tag -a <tag-name>

$ git tag -a v1.0

给已有快照追加标签

$ git tag -a v0.9 85fc7e7

查看所有标签

git tag

指定标签信息命令

git tag -a <tag\_name> -m "标签信息"

PGP签名标签命令

git tag -s <tag\_name> -m "w3cschool.cc标签"

删除标签

git tag -d <tag\_name>

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

**查看快照修改的具体内容**

通过序列号：

git show 91388f0883903ac9014e00

通过标签：

git show v1.0

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

**创建Git 远程仓库**

1.先到github网站注册账号

然后创建一个项目, Create a new repository

2.之后再在Repository name 填入你的远程仓库名，其他保持默认设置

3.点击"Create repository"按钮，就成功地创建了一个新的Git仓库

4.打开终端进入你的文件目录

ssh-keygen -t rsa -C "hertzzhang@outlook.com"

之后会要求确认路径和输入密码，用默认配置的一路回车(回车三次)就行。成功的话会在指定目录下生成.ssh文件夹。

5.进去，打开 id\_rsa.pub，复制里面的 key

vi /Users/jackey/.ssh/id\_rsa.pub

6.复制里面的所有内容, 然后回到github网页选择Settings

进入settings后选择SSH and GPG keys

选择New SSH Key

输入完title和key后选择Add SSH key按钮完成添加。

7.现在回到终端, 测试下是否能连接到github服务器

ssh -T git@github.com

看到这样的提示就说明设置好了

bogon:ZXTabBarController jackey$ ssh -T git@github.com Hi zhouxihi! You've successfully authenticated, but GitHub does not provide shell access.

8.然后就可以将本地文件push到远程仓库了

首先建立本地仓库

git init

添加文件到缓冲区

git add .

写入本地版本仓库

git commit -m 'first upload'

在本地通过vim在远程服务器上添加一个新的远程仓库

//git remote add [shortname] [url]

git remote add origin git@github.com:zhouxihi/ZXTabBarController.git

推送文件到远程仓库

git push [alias] [branch]

git push -u origin master

显示100%. done就可以了

若要将标签也推送到远程服务器，则运行

git push origin [tagname]

如果要一次推送所有本地新增的标签上去

git push origin --tags

以后每次有修改本地文件就只要下面三个步骤就可以推送到服务器了

->git add.

->git commit -m 'second upload'

->git push -u origin master

要查看当前有哪些远程仓库

git remote

git remote -v // 查看每个别名的实际链接地址

如果想从服务器拷贝一份代码

git clone XXX.git

**提取远程仓库**

1、从远程仓库下载新分支与数据：

git fetch

该命令执行完后需要执行git merge 远程分支到你所在的分支。

2、从远端仓库提取数据并尝试合并到当前分支：

git merge

假设你配置好了一个远程仓库，并且你想要提取别人更新的数据，你可以首先执行 git fetch [alias] 告诉 Git 去获取它有你没有的数据，然后你可以执行 git merge [alias]/[branch] 以将服务器上的任何更新（假设有人这时候推送到服务器了）合并到你的当前分支。

**推送到远程仓库**

git push [alias] [branch]

**删除远程仓库**

git remote rm [别名]

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

**Git私有服务器搭建**

上一章节中我们远程仓库使用了 Github，Github 公开的项目是免费的，但是如果你不想让其他人看到你的项目就需要收费。

这时我们就需要自己搭建一台Git服务器作为私有仓库使用。

接下来我们将以 Centos 为例搭建 Git 服务器。

**1、安装Git**

$ yum install curl-devel expat-devel gettext-devel openssl-devel zlib-devel perl-devel
$ yum install git

接下来我们 创建一个git用户组和用户，用来运行git服务：

$ groupadd git
$ useradd git -g git

**2、创建证书登录**

收集所有需要登录的用户的公钥，公钥位于id\_rsa.pub文件中，把我们的公钥导入到/home/git/.ssh/authorized\_keys文件里，一行一个。

如果没有该文件创建它：

$ cd /home/git/
$ mkdir .ssh
$ chmod 755 .ssh
$ touch .ssh/authorized\_keys
$ chmod 644 .ssh/authorized\_keys

**3、初始化Git仓库**

首先我们选定一个目录作为Git仓库，假定是/home/gitrepo/runoob.git，在/home/gitrepo目录下输入命令：

$ cd /home
$ mkdir gitrepo
$ chown git:git gitrepo/
$ cd gitrepo
$ git init --bare runoob.git
Initialized empty Git repository in /home/gitrepo/runoob.git/

以上命令Git创建一个空仓库，服务器上的Git仓库通常都以.git结尾。然后，把仓库所属用户改为git：

$ chown -R git:git runoob.git

**4、克隆仓库**

$ git clone git@192.168.45.4:/home/gitrepo/runoob.git
Cloning into 'runoob'...
warning: You appear to have cloned an empty repository.
Checking connectivity... done.

192.168.45.4 为 Git 所在服务器 ip ，你需要将其修改为你自己的 Git 服务 ip。

这样我们的 Git 服务器安装就完成。