# 前言

Git的定义是：分布式版本控制系统，用于项目开发中的版本控制。从本质上来讲Git是一个内容寻址(content-addressable)文件系统，并在此之上提供了一个版本控制系统的用户界面。 Git的核心部分是一个简单的键值对数据库(key-value data store)。 你可以向该数据库插入任意类型的内容，它会返回一个键值，通过该键值可以在任意时刻再次检索(retrieve)该内容。Git管理的项目工作目录下的每一个文件都不外乎这两种状态：已跟踪或未跟踪。

* 已跟踪的文件是指那些被纳入了版本控制的文件，在上一次快照中有它们的记录，在工作一段时间后，它们的状态可能处于未修改，已修改或已放入暂存区。
* 工作目录中除已跟踪文件以外的所有其它文件都属于未跟踪文件，它们既不存在于上次快照的记录中，也没有放入暂存区。
* 初次克隆某个仓库的时候，工作目录中的所有文件都属于已跟踪文件，并处于未修改状态。

Git保存的不是文件的变化或者差异，而是一系列不同时刻的文件快照。在进行提交操作时，Git会保存一个提交对象(commit object)。该提交对象会包含一个指向暂存内容快照的指针。但不仅仅是这样，该提交对象还包含了作者的姓名和邮箱、提交时输入的信息以及指向它的父对象的指针。

# Git存储

Git将项目的存储分为4部分，每部分有自己作用，见下图：

杯子里有饮料

描述已自动生成

* Workspace：工作区(当前用户操作修改的区域)
* Index / Stage：暂存区 (add后的区域)
* Repository：仓库区或本地仓库(commit后的区域)
* Remote：远程仓库(push后的区域)

整体过程可以简述为：

* 工作区–>add–>暂存区–>commit–>本地仓库区–>push–>远程仓库区
* 远程仓库区–>fetch–>使用refs\remotes下对应分支文件记录远程分支末端commit\_id 和 本地仓库区 -->merge–>工作区
* 远程仓库区–>pull–>使用refs\remotes下对应分支文件记录远程分支末端commit\_id and 本地仓库区 and 工作区

git pull和git fetch有什么不同呢？下面简单说一下。想要知道他们得不同，我们需要先了解两个概念：

* FETCH\_HEAD：可以看做是一个版本链接，记录在本地的refs\remotes下对应分支文件中，指向着目前已经从远程仓库取下来的分支的最新版本的commitid。
* commit-id：在每次本地commit来保存当前工作到本地仓库区后， 会产生一个commit-id，这是一个能唯一标识一个版本的序列号。在使用git push后，这个序列号还会同步到远程仓库。

所以他们之间的不同在于：

* git pull直接将远程分支的修改更新到本地仓库区和本地工作区，我们就可以在本地工作区中看到最新代码
* git fetch只将远程分支的修改拉取到本地仓库，并更新到FETCHHEAD，记录远程分支最新的commitid，不会更新本地工作区代码，只有使用了git merge才会将提交更新到本地仓库区和工作区

其他想要了解更多Git内部消息请移步我的另一篇博文：https://liyangyang.blog.csdn.net/article/details/101035399。在git status中的体现，见下图：

电脑屏幕截图

描述已自动生成

* Changes to be committed：代表被add的文件，被加载到了暂存区
* Changes not staged for commit：代表在当前分支中被修改的文件，还没有被add，存储在工作区
* Untracked files：代表不被Git追踪的文件，可以理解为不被Git管理的文件
* 如果没有Changes to be committed和Changes not staged for commit说明现阶段所有的修改已经被commit到本地仓库
* 如果git status后出现下述情况，说明还有已经的commit到本地仓库的还未被push到远程仓库

$ git status

On branch master

Your branch is ahead of 'origin/master' by 2 commits.

(use "git push" to publish your local commits)

Git作为一个系统，是以它的一般操作来管理并操纵(HEAD 版本库、index 暂存区、Working Directory 工作区)三棵树的。

* HEAD：是当前分支引用的指针，它总是指向该分支上的最后一次提交。这表示HEAD将是下一次提交的父结点。通常，理解HEAD的最简方式，就是将它看做 你的上一次提交的快照。
* index：index索引是你的 预期的下一次提交。我们也会将这个概念引用为Git的 “暂存区域”，这就是当你运行git commit时Git看起来的样子。Git将上一次检出到工作目录中的所有文件填充到索引区，它们看起来就像最初被检出时的样子。之后你会将其中一些文件替换为新版本，接着通过git commit将它们转换为树来用作新的提交。
* Working Directory：最后，你就有了自己的工作目录。另外两棵树以一种高效但并不直观的方式，将它们的内容存储在.git文件夹中。工作目录会将它们解包为实际的文件以便编辑。

图片包含 日程表

描述已自动生成

# Git提交规则

首先，有个问题需要确认一下，提交信息是使用中文还是英文呢？如果你的项目是开源项目并且面向国际的开源项目，类似于阿里的Druid\PingCAP的TiDB等，那么一定要是用英文提交信息的！如果你的项目是公司内部使用或者只会被公司内部开发修改，那么中文也是不错的，更加便于查看和管理。当然，开发组中的英文能力都不错的话，用英文也是可以的。

分支Branch管理：如果没有一个好的Branch管理的话，可能会有下述图的情况，刺不刺激~

图片包含 电子, 电路, 游戏机, 电脑

描述已自动生成

推荐的分支管理：

* master分支为主分支(保护分支)，禁止直接在master上进行修改代码和提交，此分支的代码可以随时被发布到线上；
* develop分支为测试分支或者叫做合并分支，所有开发完成需要提交测试的功能合并到该分支，该分支包含最新的更改；
* feature分支为开发分支，大家根据不同需求创建独立的功能分支，开发完成后合并到develop分支；
* fix分支为bug修复分支，需要根据实际情况对已发布的版本进行漏洞修复。

标签Tag管理：Tag采用三段式：v版本.里程碑.序号(v2.3.1)

* 架构升级或架构重大调整，修改第1位
* 新功能上线或者模块大的调整，修改第2位
* bug修复上线，修改第3位

当然，可以根据实际情况来设计，比如项目特别大，可以使用四段表达Tag，项目比较小也可以使用二段式Tag，只要符合场景并有实际意义即可。提交信息格式：下面只是提供一种建议格式，大家可以根据自己的项目实际情况来定格式，只要能把当前提交表达清楚，格式统一，方便快速阅读和定位即可！

* 建议中文示例：
* (urllAnalyz)添加解析url功能
* (TestServiceImpl)修改某功能的某个实现为另一个实现
* (TestUnti)修复url特殊情况下解析失败问题(issue#12)
* (getData)重构获取数据的方法
* (getDataTest)添加(修改、删除)获取数据的单元测试代码
* (doc)修改(添加、删除)文档
* 对应到英文：
* feat：新功能(feature)
* style：格式
* fix：修补bug
* refactor：重构
* test：测试相关
* docs：文档(documentation)
* 格式(type：scope：body：issue)

(影响模块)提交描述信息(#issue?)

* 优点作用：
* 与github数据issue关联，便于通过issue获取更多信息
* commit 提交时，格式统一，便于后续快速准确定位提交
* 可以更好的将此次提交表述清楚

# Git操作过程

****1、初始化项目，并上传到Git服务器****

基本过程：创建远程仓库、初始化本地Git仓库、将本地仓库与远程仓库关联起来、添加本地仓库想要提交的代码到本地Git缓冲区，将本地仓库的本地分支与远程仓库的远程分支关联起来、提交代码。在Git服务器上创建同名Git项目，并获取http地址。

本地Git初始化项目Git仓库，在项目目录下：

git init

将本地Git仓库和远程仓库关联起来，并设置远程仓库名称：

git remote add <http地址>

其中http地址为上述第一步获取的远程仓库的地址，name一般为origin，当然也可以设置其他的名字。

例如：git remote add origin <http://igit.corp.com/my/test.git> 或者 git remote add origin git@github.com:my/test.git

添加项目文件到本地Git缓冲区：

git add -A

提交本地Git缓冲区内容到本地版本库：

git commit -m ‘初始化项目’ 或者 git commit -a -m ‘初始化项目’

将本地分支关联远程分支并提交，Git默认在远程分支上创建与本地分支同名的分支：

git push --set-upstream origin master

这就是将本地的master分支与origin远程仓库关联起来并在远程仓库创建同名master分支，以后本地master都提交到远程仓库中的origin/master分支上。upstream：上游的意思。至此，应该就可以了，我们可以在Git服务器上刷新看看是否提交上去了。

2、提交某一分支的修改

查看当前分支的修改：

git status

查看想要查看的文件的修改：

git diff

确认正确后，提交修改到暂存区：

git add -A 或者 git add

提交到本地仓库：

git commit -m '提交信息'

提交到远程仓库：

git push origin master

3、拉取远程分支修改到本地分支

当远程分支别人推了一版新的代码时，我们想要将代码拉下来，可以采用两种方式pull和fetch+merge：(他们的不同点文章上面已经解释)

使用pull：将远程分支最新代码更新合并到本地仓库区和工作区：

git pull

使用fetch：将远程所有分支最新的commitid更新到FETCHHEAD，记录远程分支最新的commit\_id 和 本地仓库区：

git fetch origin

将最新的代码合并到工作区：

git merge orgin/master

4、取消track某一文件

git rm -r --cache

untrack后，使用commit -a时，不会将其添加到暂存区中。之后会在.ignore文件中将该untrack的文件添加进去，完成。

5、保存账号密码，避免每次push都要输入(简单方法)

确保在Git中手动输入过账号和密码。输入下面语句即可：

git config --global credential.helper store

6、暂存自己的修改，便于接着工作(特别有用)

如果你正在一个分支上工作修改，leader让你改另外的分支的BUG或者对其他的分支做一些操作。我们知道如果一个分支上有还没有commit的修改的话，不可以切换分支，但是又因为自己的工作还未完成，不想commit，此时git stash 就起作用了。你要把现在正在工作的分支保存下来，等处理完其他的再回来接着当前分支的修改工作。

将当前分支的修改暂存起来(此处不等于add+commit)：

git stash

备份当前的工作区的内容，从最新的一次提交中读取相关内容，让工作区保证和上次提交的内容一致。同时，将当前的工作区修改的内容保存到Git栈中暂存起来。

切换到别的分支工作，完成后切换回原来的工作分支，查看暂存列表：

git stash list

显示Git栈内的所有备份，可以利用这个列表来决定从那个地方恢复。恢复暂存的修改到工作区：git stash apply恢复暂存之后不删除暂存。从Git栈中读取最新一次保存的内容，恢复工作区的相关内容。git stash pop恢复暂存之后删除暂存。从Git栈中读取最新一次保存的内容，恢复工作区的相关内容。之后pop会删除最新的暂存。

删除“暂存”：

git stash drop

从Git栈删除最旧的一个暂存。

结束。

其他：git stash pop = git stash apply + git stash drop 版本的回溯与前进有时候需要回溯或前进到以前的版本或回溯前进到以前的commit。只要记住commit\_id就可以在版本之间来回的穿梭，注意是可以“来回”穿梭。获取需要回溯到版本的commitid：git log --> 复制所需的版本commitid回退到该版本：git reset --hard如果想让服务器也回退到该版本的话：git push origin HEAD --force

7、Git常用命令

初始化项目为Git项目：

git init

clone服务器代码到本地：

git clone

添加修改文件到暂存区：

git add

git add -A添加所有修改文件到暂存区

提交修改到本地仓库：

git commit -m '提交信息'

git commit -a -m '提交信息' 相当于git add -A + git commit -m的整合

提交本地仓库的修改到远程仓库：

git push origin <branch-name>

将远程库的远程分支作为当前分支的上游分支：

git push --set-upstream

为远程仓库的别名，一般为origin。

查看文件本次的修改：

* git diff显示本次所有被修改文件的修改
* git diff 显示该文件本次的修改

查看当前分支下当前状态：

git status

显示出被修改的文件和提交的次数等。

查看提交历史：

git log

git log --graph查看分支合并图。

merge其他分支到当前分支：

git merge

在merge过程如果出现冲突，在解决冲突后会产生一个新的commit，并且HEAD指向该commit。如果没有冲突，HEAD会在分支的最新commit上。

切换到上一个分支：

git checkout -

切换到其他分支：

git checkout

在某一分支基础上创建新分支：

* git checkout 切换到基础分支
* git checkout -b 在当前分支基础上创建新分支

显示分支：

* git branch所有本地分支
* git branch -r所有远程分支
* git branch -a所有分支，本地和远程

删除本地分支：

git branch -D <branch-name>

删除远程分支：

git push origin --delete <branch-name>

添加一个新的远程仓库：

git remote add orgin2 git@github.com:remoWorkSpace/QtProj1.git

在同一个项目的git url中可以添加多个远程仓库，每个仓库相互隔离有自己的分支管理。

将本地分支与远程仓库中分支联系起来：

git push --set-upstream

以后本地的该分支的push，会默认提交到设置的远程仓库中远程分支中。

删除远程仓库：

git remote rm <name>

删除本地tag：

git tag -d

删除远程tag：

git push origin :refs/tags/<tag\_name>

查看tag信息：

git show

暂存当前修改：

git stash

查看暂存列表：

git stash lsit

恢复暂存的修改：

git stash apply(恢复后不删除暂存)

git stash pop(恢复后删除暂存)

删除暂存：

git stash drop

撤回已经add到暂存区的文件到本地工作区：

git reset HEAD

git reset HEAD 回退所有add。add会被标识为Changes to be committed，取消add后标识为Changes not staged for commit(不等于untrack)。

取消track某一文件：

git rm -r --cache

untrack后，使用commit -a 时，不会将其添加到暂存区中。

只merge某一个分支上的某一个commit：

git cherry-pick merge分支中的某一个commit的commit-id>

撤销在本地工作区的文件的修改：撤销修改就回到和版本库一模一样的状态，即用版本库里的版本替换工作区的版本。

git checkout --

删除某些commit，将head重定位到某一commit(回溯到以前的版本)：

git reset --hard

git push origin HEAD --force此步骤将服务器方也设置为相commit。

删除本地在远程服务器上不存在的分支：

git remote prune origin

拉取远程分支到本地：

git fetch origin

新建一个tag到指定commit：

git tag

git tag 当前commit

取消当前合并，重建合并前状态：

git merge --abort

# 总结

本文介绍了Git是什么、Git的存储结构、Git的提交规则和一些工作中常会用到的Git操作的过程，最后总结了常用的命令。Git在工作中的团队开发中起着至关重要的作用，希望本篇文章可以对大家有些许帮助。原文链接：https://liyangyang.blog.csdn.net/article/details/100939749Kubernetes入门与实战培训