目录

[一、Git 常用命令 1](#_Toc11560915)

[git 简介 1](#_Toc11560916)

[集中式 2](#_Toc11560917)

[分布式 2](#_Toc11560918)

[git 命令流 3](#_Toc11560919)

[Git基本说明 4](#_Toc11560920)

[git commit -am "修改的记录注视" 5](#_Toc11560921)

[git push 6](#_Toc11560922)

[git log 6](#_Toc11560923)

[git checkout 6](#_Toc11560924)

[git status 7](#_Toc11560925)

[git add 7](#_Toc11560926)

[git diff 7](#_Toc11560927)

[Clear 8](#_Toc11560928)

[git stash 8](#_Toc11560929)

[强制将远程分支覆盖本地 8](#_Toc11560930)

[二、vim使用 8](#_Toc11560931)

# 一、Git 常用命令

## git 简介

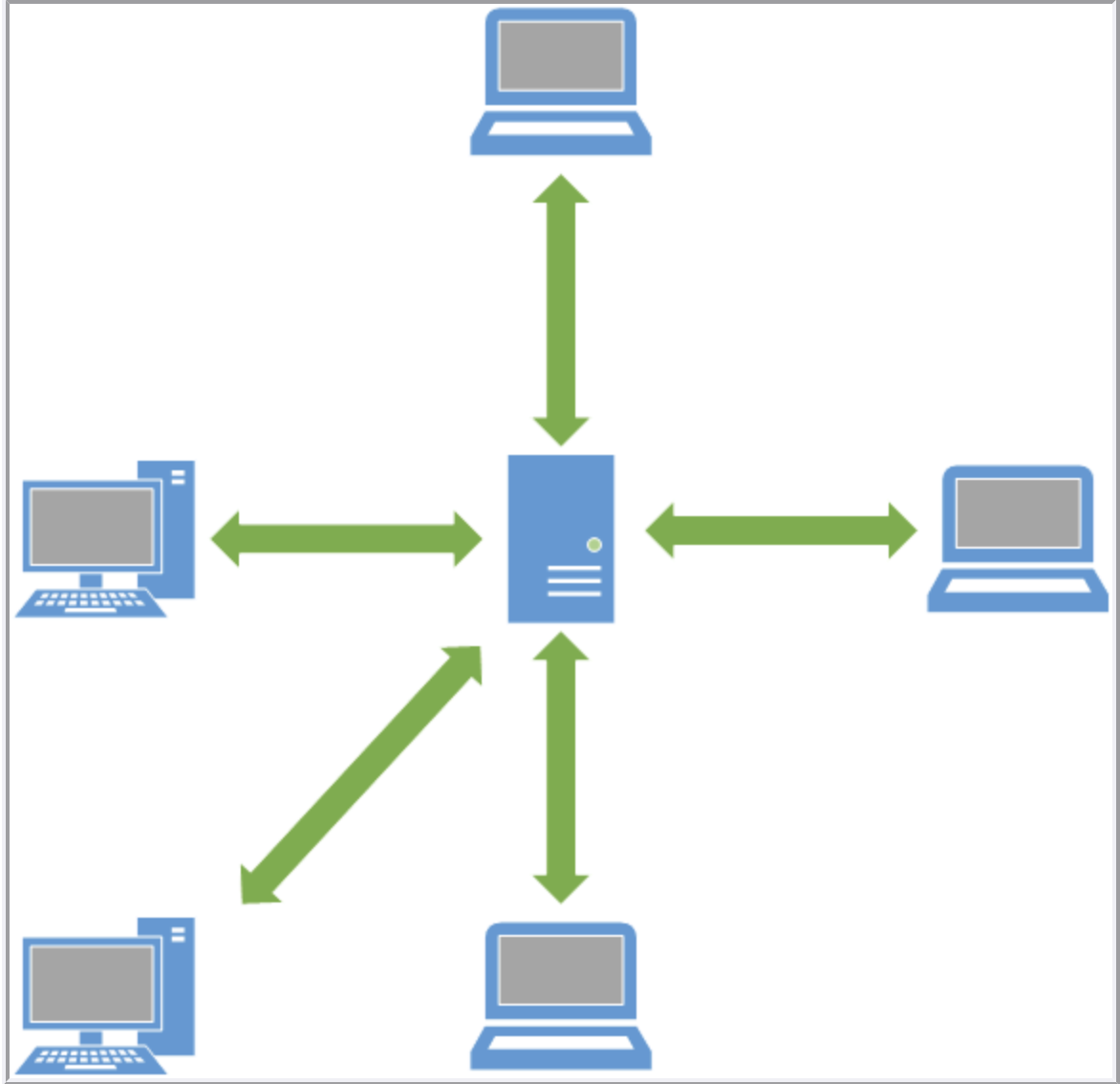
转自 https://www.cnblogs.com/wilber2013/p/4185643.html

Git是Linus用C实现的一个分布式版本控制工具，注意这里对分布式的强调。不同于Git，像Perforce、SVN和CVS这类版本控制工具都是集中式的。

下面大概介绍下集中式版本控制工具和分布式版本控制工具的区别

### 集中式

：所谓集中式的版本控制，就是在一个系统中只有一个机器是服务端，其他机器全是客户端。



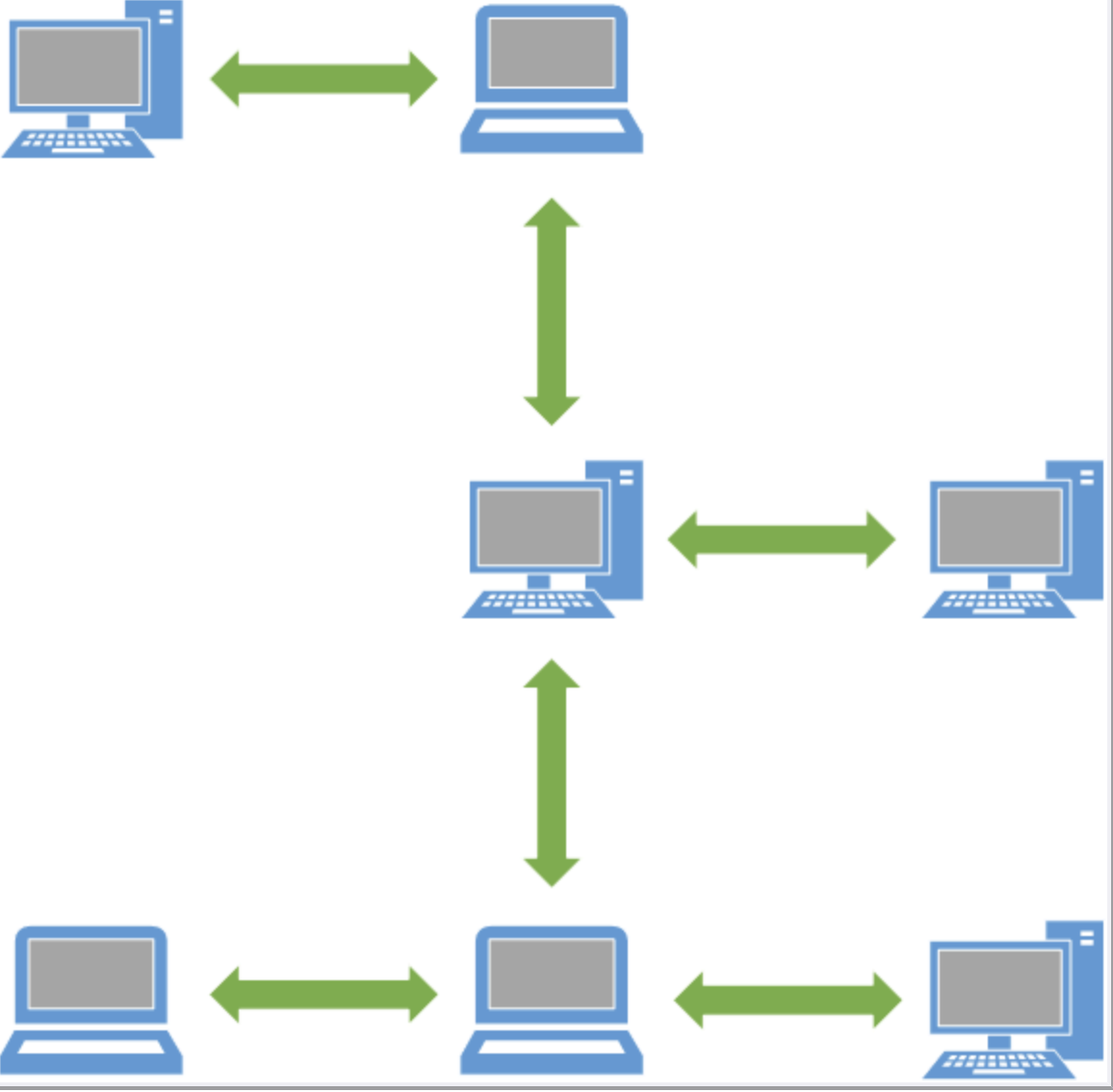
以Perforce版本控制为例，在一个系统中会有一个Perforce服务器，所有的代码以及版本信息都保存在这个服务器上。每个客户端可以从服务器get下来一份代码，然后在本地修改，最后submit修改的代码。

可以看到集中式的版本控制还是有一些问题的：

* 网络依赖性强，工作环境保持网络连接，如果网络断掉了，所有的客户端就无法工作了。
* 安全性较弱，所有的代码以及版本信息保存在服务器中，一旦服务器挂掉了，代码和版本控制信息就丢失了。

### 分布式

而在分布式版本控制系统中，没有服务端/客户端的概念，每台机器都是一个服务器。也就是说，在分布式本版控制系统中，每台机器都有一份代码，并且有代码的版本信息。



所以可以看到Git的优势：

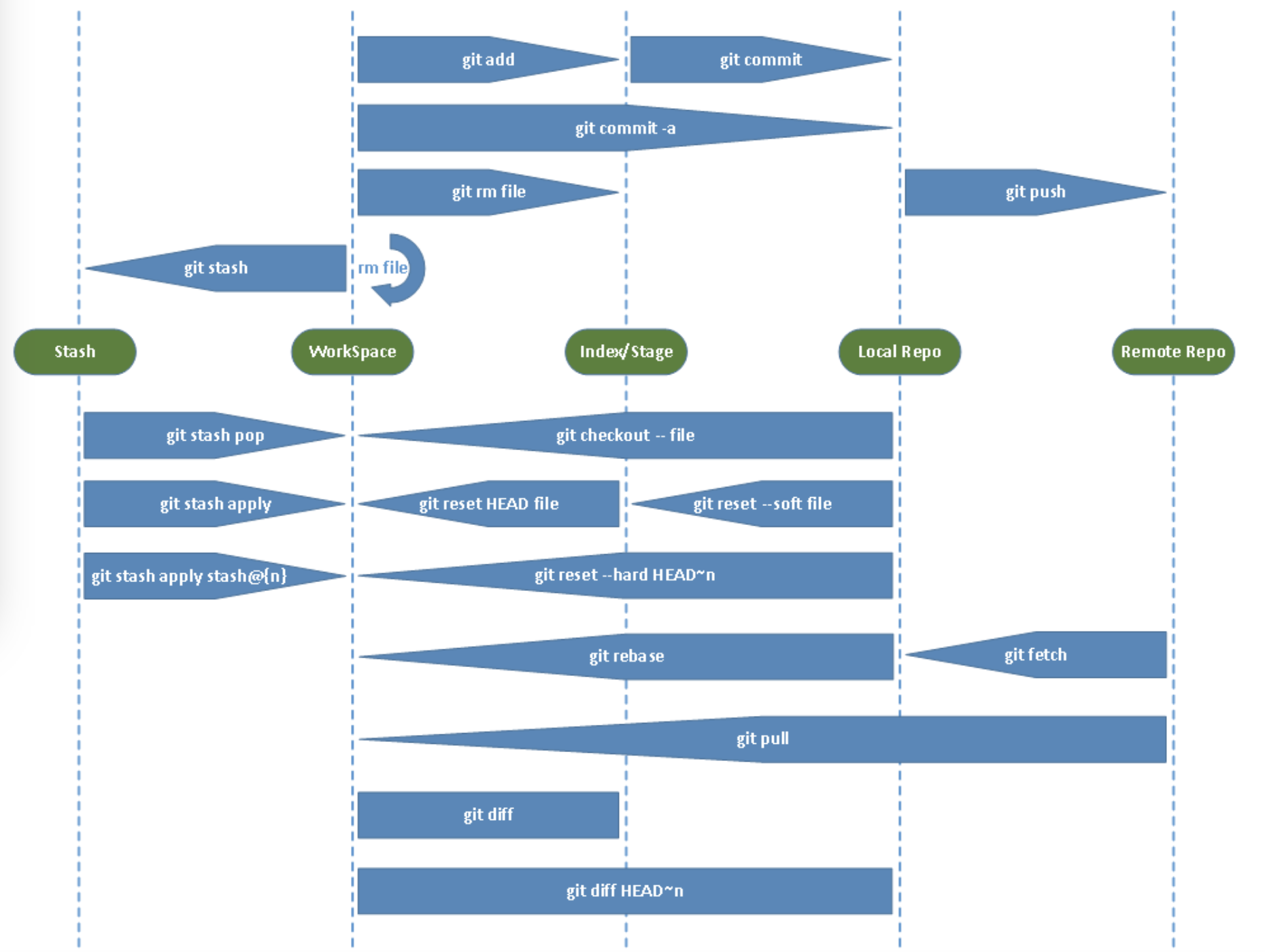
* 每台机器都是一台服务器，无需依赖网络就可以帮自己的更新提交到本地服务器，支持离线工作。当有网络环境的时候，就可以把更新推送给其他服务器。
* 安全性高，每台机器都有代码以及版本信息的维护，所有即使某些机器挂掉了，代码依然是安全的。

在Git中，同步更新的方式有很多种，可以把自己的更新推送给别人；也可以生成一个diff的patch，通过邮件方式把这个patch发送给别人。这些都将会在后面的文章中介绍。

Zs

### git 命令流

在Git中支持上百个命令，每个命令又有很多的选项。其实，真正接触过Git一段时间后，会慢慢的发现我们会经常使用的命令也就十几二十个，掌握这些命令之后就可以满足我们大部分的日常工作了



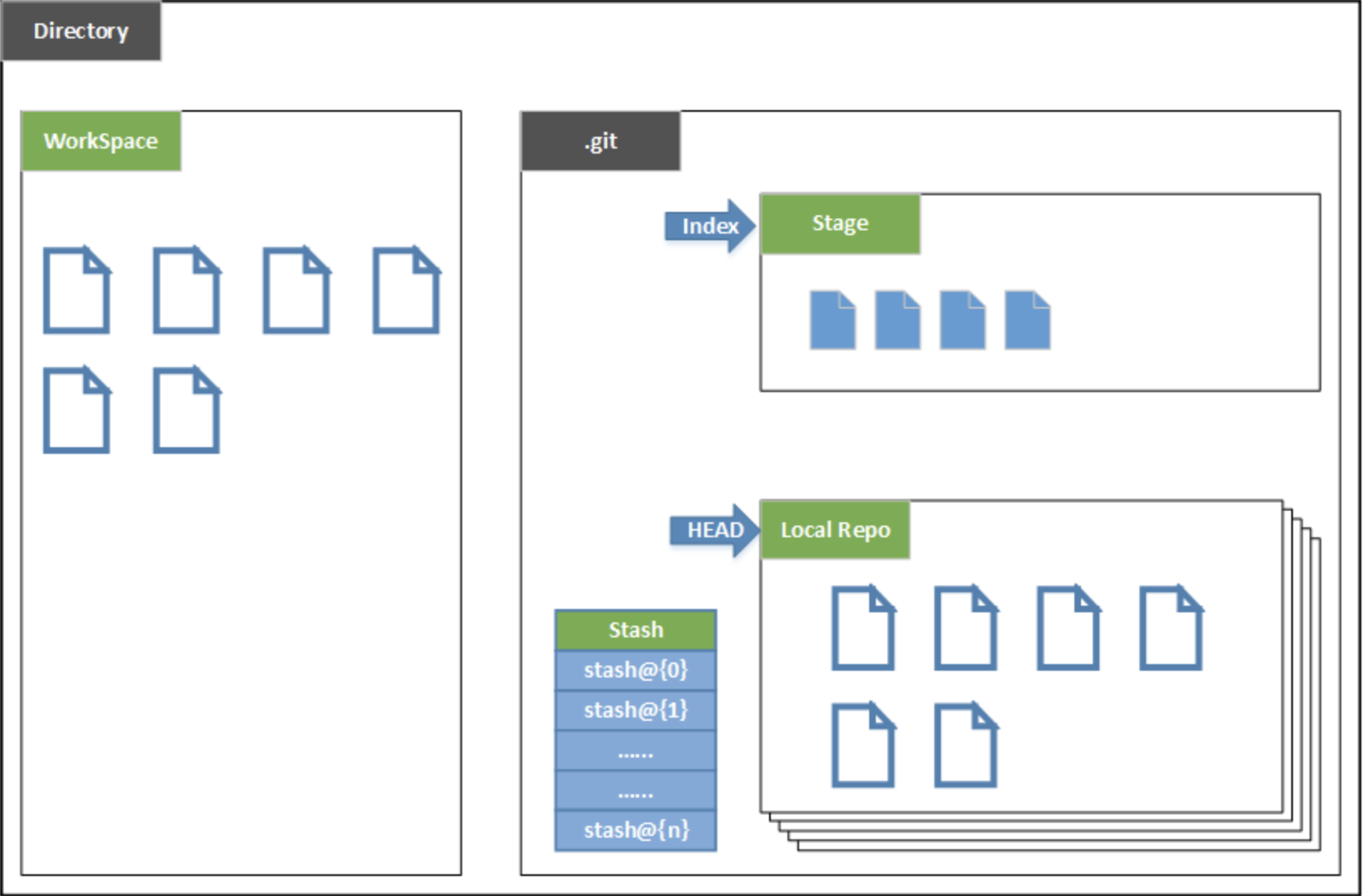
## Git基本说明

基本概念：摘自（https://www.cnblogs.com/wilber2013/p/4189920.html）

在Git中，我们将需要进行版本控制的文件目录叫做一个**仓库（repository）**，每个仓库可以简单理解成一个目录，这个目录里面的所有文件都通过Git来实现版本管理，Git都能跟踪并记录在该目录中发生的所有更新。

现在我们已经知道什么是repository（缩写repo）了，假如我们现在建立一个仓库（repo），那么在建立仓库的这个目录中有一个“.git”的文件夹。这个文件夹非常重要，所有的版本信息，更新记录，以及Git进行仓库管理的相关信息全部保存在这个文件夹里面。所以，不要修改/删除其中的文件，以免造成数据的丢失。

进一步的讲解请参考下面一张图，大概展示出了我们需要了解的基本知识



根据上面的图片，下面给出了每个部分的简要说明：

* Directory：使用Git管理的一个目录，也就是一个仓库，包含我们的工作空间和Git的管理空间。
* WorkSpace：需要通过Git进行版本控制的目录和文件，这些目录和文件组成了工作空间。
* .git：存放Git管理信息的目录，初始化仓库的时候自动创建。
* Index/Stage：暂存区，或者叫待提交更新区，在提交进入repo之前，我们可以把所有的更新放在暂存区。
* Local Repo：本地仓库，一个存放在本地的版本库；HEAD会只是当前的开发分支（branch）。
* Stash：是一个工作状态保存栈，用于保存/恢复WorkSpace中的临时状态。

 有了上面概念的了解，下面就开始在本地repo上进行Git操作了。

## git commit -am "修改的记录注视"

该命令会将本地工作区中修改后，还未使用git add . 命令添加到暂存区中的文件也一并提交上去。相当于git add . 与git commit –m “本次提交描述”两句操作合并为一句进行使用。

git commit –m

该命令会将git add .存入暂存区修改内容提交至本地仓库中，若文件未添加至暂存区，则提交时不会提交任何修改。

## git push

这个。命令一般和git commit 一起使用git push 将当前最新的缓存上传到远端,

git push 是将本地的缓存推送到远程端，只有一个分支的时候，默认将本地master推送到远端master

如果远端存在多个分支的时候需要添加具体的远端分支名称

例如

git push origin master

git push origin branch\_name

git push origin master如果远端没有branch\_name 这个分支，远端会自动建立一个这个分支

注意：使用git push 只是将缓存的stage的推送到远端，和本机在那个分支没有关系

git push origin

如果当前分支与远程分支存在追踪关系，则本地分支和远程分支都可以省略，将当前分支推送到origin主机的对应分支

git push --set-upstream origin 分支名

将当前分支推送到远端并建立跟踪关系

即便本机上的多个分支分别和远端有跟踪关系，单独的git push 只能将当前的分支分支推送到远端，不能将所有的分支同时推送到远端。

## git log

多次使用git commit -am ""之后，可以通过git log 查看提交记录。

## git checkout

多次使用git. Commit -am 但是没有git push ,如果想恢复到之前git commit 的版本。 可以使用git log查看版本号，然后

git checkout d6c287483827fa8be9879b915441502ad57e6ab2

这样就切回到之前的分支

注意：执行上述操作后，在使用git log 只能看到当前版本之前git commit 记录。 如果想看所有的需要git check master 切回到缓存区的主分支。

容易混洗的地方：不同的分支可以有不同的文件，每次切换分支前只要将修改的文件放到缓存stage上，切换分支后本地文件夹会自动给切换。

git checkout -f 撤销本地电脑上的修改

git checkout -b 新分支的名称

新建一个分支，但是这个新建的分支是在缓存stage上，还要将其推到远端。可以使用过如下命令

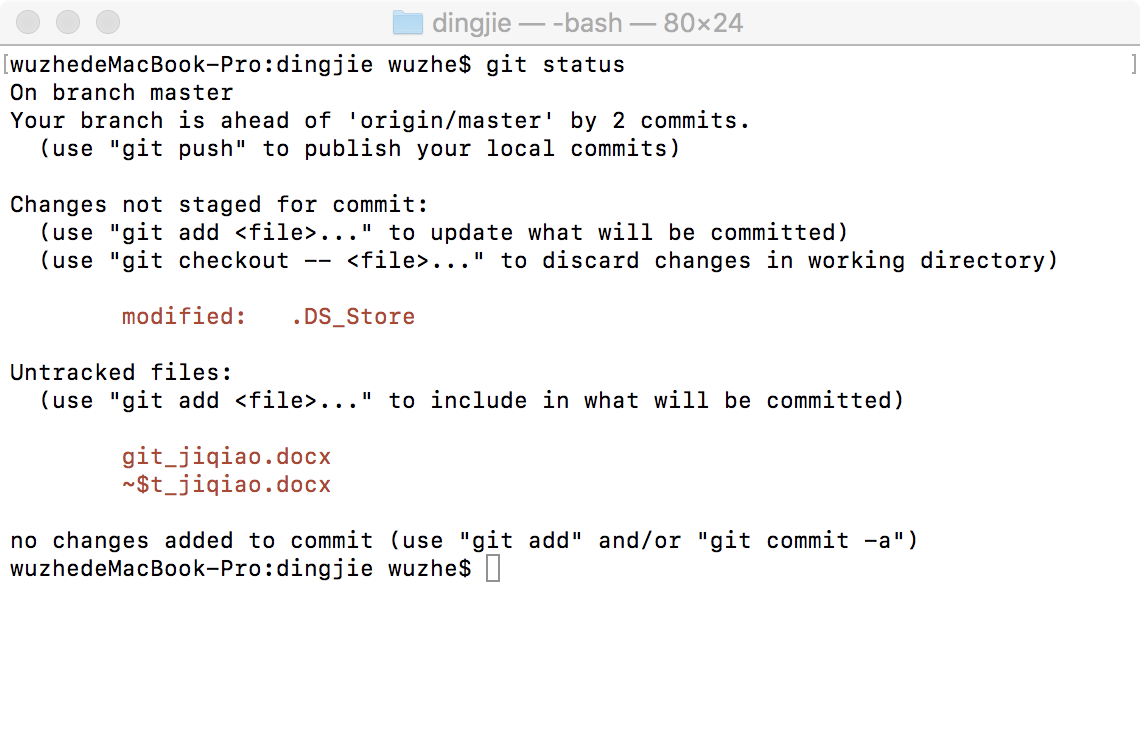
## git status

如果想看当前git情况，可以使用git status.

On branch 当前的分支

Changes not staged for commit 当前本机的修改和当前分支的不同（注意此处的当前分支指的是缓存中的分支，并不是远端）

Untracked files. 本机新添加的文件，缓存上没有，需要添加到缓存进行跟踪



## git add

git add git\_jiqiao.docx 本机git\_jiqiao.docx 会添加到缓存stage中。 然后便git. Commit 和git push了

## git diff

本机修改了一些，git diff 可以看本机和所在分支的不同，所在分支既不是远端,是缓存上的文件，可能是master也可能是别的分支.

git diff origin/master master 查看远端master和本机master 的不同

## Clear

终端内容很多是，可以clear. 清空终端

## git stash

如果当前正在写代码，临时要切换到之前的版本，现在的正在谢的又不想上传，可以使用git stash 将当前的代码放在缓存区，之后可以在用git stash pop 回到这个版本

## 强制将远程分支覆盖本地

以此执行以下三条：

git fetch --all

git reset --hard origin/master

git pull

# 二、vim使用