git学习

# 1.Git用途

Git与svn一样是代码管理工机具。Git不仅能管理代码版本，还能管理文件版本，为开发的版本带来了极大的便利。

Git是分布式管理工具，而svn不是

Git的主要作用：

1. 备份
2. 代码还原
3. 协同开发
4. 代码追责-查找代码是由谁提交的

# 2.git工作流程

如下是git工作流程图



上图的步骤可以分为如下几步：

1.克隆 Git 资源作为工作目录。

2.在克隆的资源上添加或修改文件。

3.如果其他人修改了，你可以更新资源。

4.在提交前查看修改。

5.提交修改。

6.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。

# 3.一些基本概念

**工作区：**就是你在电脑里能看到的目录。

**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repository：**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。



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。

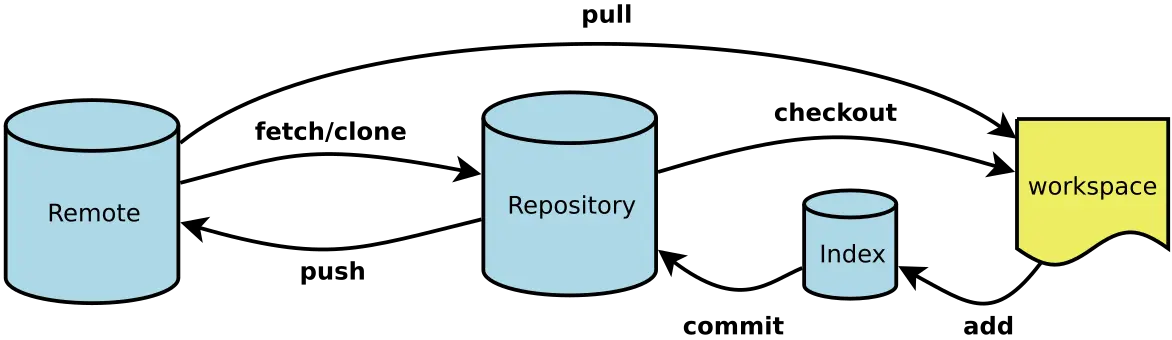
当执行 "git reset HEAD" 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。

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。

远程仓库: 与svn的集中式服务器类似，所有的项目参与人员都可以从该仓库中拉取推送代码



本地仓库：

Git GUI：git的图形化界面

Git Bash： git的命令行界面

分支：在多人协同的项目开发中，每个人都希望开发自己的代码而互不影响，则需要使用分支来进行操作

# 4 基本操作

## 4.1 基本配置

设置个人信息：

git config --global user.name “steven”

git config --global user.email [“1144873128@qq.com”](mailto:\“1144873128@qq.com\”)

查看个人信息

git config --global user.name

git config --global user.email

linux中为命令指定别名

在用户目录下创建 .bashrc 文件写入命令的缩写别名

alias git-log=‘git log --pretty=oneline --all -graph --abbrev-commit’

alias ll=’ls -al’

解决gitbash乱码问题

1. 执行命令

git config --global core.quotepath false

1. 在.bashrc文件最后加入下面两行

export LANG=”zh\_CN.UTF-8”

export LC\_ALL=”zh\_CN.UTF-8”

## 4.2 常用基本操作

### 创建仓库

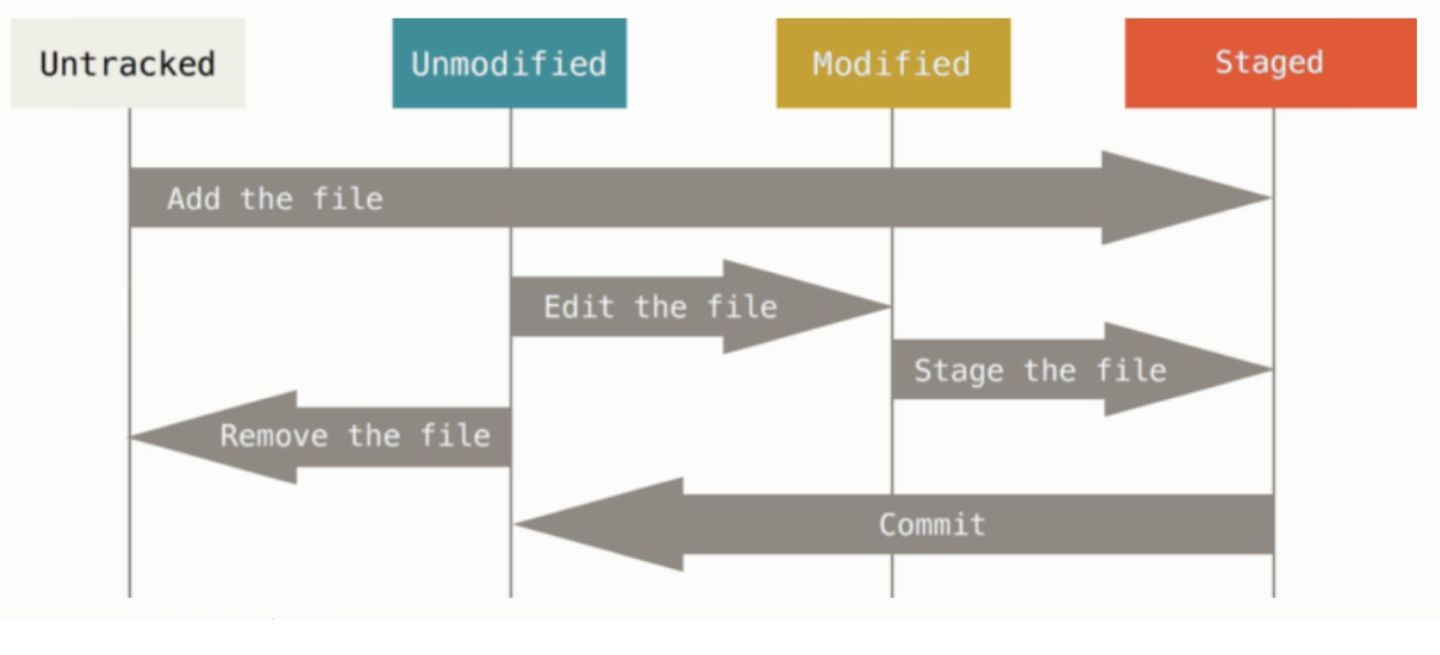
在当前目录生成一个 .git 目录：

git init

指定目录：

git init newrepo

### 文件状态



文件的状态

未跟踪状态：新创建一个文件（使用touch命令创建或其他命令创建），这时文件处于未跟踪状态

已暂存：使用 git add filename ，将文件放入git的index区（暂存区）

已提交：使用git commit filename,将文件放入仓库区（repository）

### add操作

git add 命令告诉 Git 开始对这些文件进行跟踪

$ git add \*.c

$ git add README

### 4.3 commit操作

将文件提交到版本库中：

$ git commit -m '初始化项目版本'

跳过add操作直接提交：

git commit -a -m '这是一次版本提交'

### 查看git 提交日志

git log

参数

--all 显示所有分支提交记录

--pretty=oneline 将信息显示到一行

--abbrev-commit 优化提交记录的hash

--graph 在命令行中展示分支的合并信息

### 版本回退

git reset --hard <commitID>

git reflog ：显示所有的操作记录，在版本回退时使用比较方便

### .gitignore文件来指定不需要git管理的文件

在git的工作区中如果由不希望git来管理，创建.gitignore的文件，如果a.txt不需要git管理

可以在文件中添加 a.txt指定git忽略该文件

### 分支操作

HEAD 指的是当前分支

git branch 查看分支

git branch dev01 创建一个分支

git log --all 查看所有分支提交记录

git status 查看工作区文件状态

git checkout dev01 切换分支

git checkout -b dev02 创建并切换到分支dev02

git merge dev02 将dev02分支合并到当前所在的分支

git branch -d dev01 删除分支

git branch -D dev01 强制删除分支

### 分支冲突解决

当两个人同时操作同一个文件，在进行分支合并时会引发冲突，导致自动合并分支失败。

这时git会在冲突文件中展示各个分支中的修改，这时需要对文件进行修改，然后再次add、commit，冲突解决

### 开发中的操作流程

master分支：生产分支

dev分支：开发分支，所有的开发代码都提交到dev的分支

### 绑定远程仓库

git remote add origin <https://fsdfasdfas/test>

### 推送代码到远程仓库

git push origin dev[本地分支名]:dev[远端分支名]

git push -f origin dev:dev 强制推送

### 4.4 clone操作

1.从现有仓库拷贝到当前目录

git clone <repo>

2.克隆到指定的目录:

git clone <repo> <directory>

**repo:**Git 仓库。

**directory:**本地目录。

## 4.5查看当前状态

git status [-s(获取简短结果)]

## 4.6 查看详细信息

一行一行地显示具体的改动

git diff [--cached]

## 4.7取消已缓存内容

$ git reset HEAD hello.php

执行 git reset HEAD 以取消之前 git add 添加，但不希望包含在下一提交快照中的缓存。

## 4.8移除文件

要从 Git 中移除某个文件，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，使用 **--cached** 选项即可

git rm --cached <file>

递归删除，即如果后面跟的是一个目录做为参数，则会递归删除整个目录中的所有子目录和文件：

git rm –r \*

## 4.9 mv操作

git mv 命令用于移动或重命名一个文件、目录、软连接。

## 4.10 reset操作

git reset HEAD 命令用于取消已缓存的内容。（取消add的内容）

git reset HEAD hello.php

常见版本管理工具：svn、cvs、bitkeeper

概念：

工作区：版本控制的目录（用户编辑的文件）

暂存区：未提交的代码 .git/index（进行了跟踪但未提交的文件，add操作产生的文件）

版本库：.git（提交后的文件，commit产生的文件）

特点：开源分布式版本控制器

## 4.11常见操作：

git init 在任意文件夹下执行该初始化版本库，会产生一个.git的隐藏文件夹，该文件夹（即为版本库）下index文件对应暂存区

git add filename 添加文件到暂存区

git commit -m “message” 提交暂存区文件到版本库

git reset --hard HEAD^ 重置版本未上次提交的版本（取消最近一次提交）

git status 查看暂存区的状态

git diff 查看版本改动

git commit –amend修改上一次提交

git reset HEAD filename 取消暂存文件

git checkout – filename 取消对文件修改（危险操作）

git remote -v 查看远程仓库

git remote add name remoteurl 创建远程仓库别名

git clone remote -url 从远程仓库拉取数据

git fetch origin 从远程仓库抓取本地没有地的数据

git push origin master 将本地代码提交到远程仓库主分支上

git tag 列出版本标签

git tag -a tagname -m ‘message’ 添加标签

git show tagname 显示标签详细信息

git tag tagname-lw 创建轻量级标签，轻量级标签实际上就是一个保存着对应提交对象的校验和信息的文件

git branch branchname 创建分支

git branch 查看分支

git branch -v 查看分支最后一个提交信息

git checkout branchname 切换到其他分支上

git checkout -b branchname 创建分支并在切换到该分支

git merge branchname 合并分支到当前分支

git branch -d branchname （合并完成之后）删除分支

git mergetool 合并产生冲突时使用该命令引导解决冲突

git branch –merged 查看当前已合并分支

git branch –no-merged 查看未合并分支进度

git push origin remotebranchname 推送本地分支到远程仓库（在已经进行clone操作的情况下）

git pull origin master从远程主分区拉取数据

git push origin branchname 将本地分支推送到远程服务器上

git checkout --track origin/branchname 跟踪远程分支

git checkout -b localbranchname origin/remotebranchname 跟踪远程分支并在本地创建别名分支

git push origin :remotebranchname 合并远程分支到master

git rebase master 分支基变，与merge操作类似，将当前分支作为master分支的下游分支

git rebase --onto master server client 将client、server合并到master分支上

git rm -r –cached foldername 删除文件夹

git rm –cached filename 删除文件

1. 在idea中的配置使用
2. GitHub的使用
3. GitLab