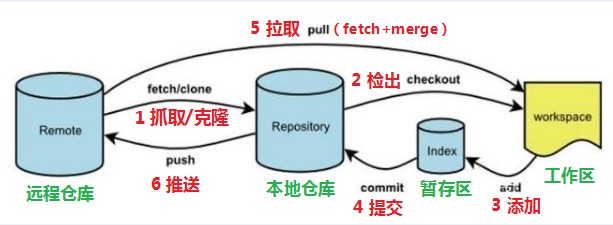
**Git分布式版本控制工具**

# 1、概述

## 1.1、Git工作流程图



命令如下：

* + 1. clone（克隆）: 从远程仓库中克隆代码到本地仓库
    2. checkout （检出）:从本地仓库中检出一个仓库分支然后进行修订
    3. add（添加）: 在提交前先将代码提交到暂存区
    4. commit（提交）: 提交到本地仓库。本地仓库中保存修改的各个历史版本
    5. fetch (抓取) ： 从远程库，抓取到本地仓库，不进行任何的合并动作，一般操作比较少。
    6. pull (拉取) ： 从远程库拉到本地库，自动进行合并(merge)，然后放到到工作区，相当于

fetch+merge

* + 1. push（推送） : 修改完成后，需要和团队成员共享代码时，将代码推送到远程仓库

# 2、常用命令

## 2.1、 Git环境配置

### 2.1.1 基本配置

* + - 1. 打开Git Bash
      2. 设置用户信息

git conﬁg --global user.name “itcast”

git conﬁg --global user.email “[hello@itcast.cn](mailto:hello@itcast.cn)”

查看配置信息

git conﬁg --global user.name git conﬁg --global user.email

### 2.1.2 为常用指令配置别名（可选）

有些常用的指令参数非常多，每次都要输入好多参数，我们可以使用别名。

1. 在.bashrc 文件中输入如下内容：

#用于输出git提交日志

alias git-log='git log --pretty=oneline --all --graph --abbrev-commit' #用于输出当前目录所有文件及基本信息

alias ll='ls -al'

## 2.2、获取本地仓库

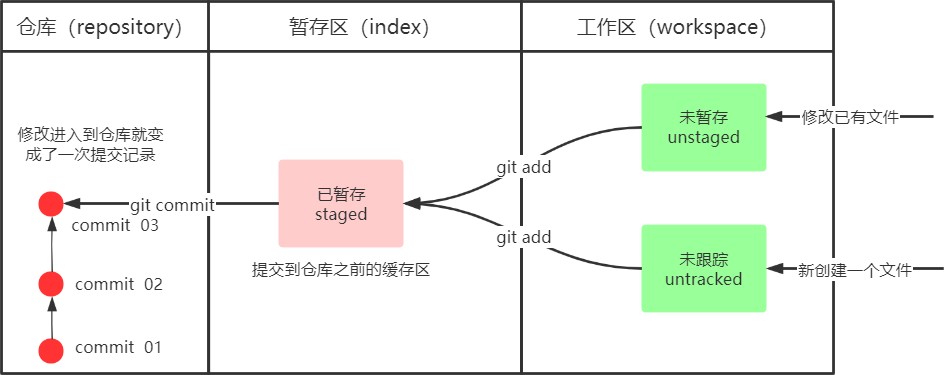
要使用Git对我们的代码进行版本控制，首先需要获得本地仓库

1. 在电脑的任意位置创建一个空目录（例如test）作为我们的本地Git仓库
2. 进入这个目录中，点击右键打开Git bash窗口
3. 执 行 命 令 git init
4. 如果创建成功后可在文件夹下看到隐藏的.git目录。

## 2.3、基础操作指令

Git工作目录下对于文件的**修改**(增加、删除、更新)会存在几个状态，这些**修改**的状态会随着我们执行Git

的命令而发生变化。



本章节主要讲解如何使用命令来控制这些状态之间的转换：

1. git add (工作区 --> 暂存区)
2. git commit (暂存区 --> 本地仓库)

### 2.3.1 \*查看修改的状态（status）

作用：查看的修改的状态（暂存区、工作区） 命令形式：git status

### 2.3.2 \*添加工作区到暂存区(add)

作用：添加工作区一个或多个文件的修改到暂存区命令形式：git add 单个文件名|通配符

将所有修改加入暂存区：git add .

### 2.3.3 \*提交暂存区到本地仓库(commit)

作用：提交暂存区内容到本地仓库的当前分支命令形式：git commit -m '注释内容'

### 2.3.4 \*查看提交日志(log)

**配置的别名** git**-**log **就包含了这些参数，所以后续可以直接使用指令** git**-**log

作用:查看提交记录

命令形式：git log [option] options

--all 显示所有分支

--pretty=oneline 将提交信息显示为一行

--abbrev-commit 使得输出的commitId更简短

--graph 以图的形式显示

### 2.3.5 版本回退

作用：版本切换

命令形式：git reset --hard commitID

commitID 可以使用git-log 或git log 指令查看如何查看已经删除的记录？

git reﬂog

这个指令可以看到已经删除的提交记录

## 2.4、分支

几乎所有的版本控制系统都以某种形式支持分支。 使用分支意味着你可以把你的工作从开发主线上分离开来进行重大的Bug修改、开发新的功能，以免影响开发主线。

### 2.4.1 查看本地分支

命令：git branch

### 2.4.2 创建本地分支

命令：git branch 分支名

### 2.4.3 切换分支(checkout)

命令：git checkout 分支名

我们还可以直接切换到一个不存在的分支（创建并切换） 命令：git checkout -b 分支名

### 2.4.4 \*合并分支(merge)

一个分支上的提交可以合并到另一个分支命令：git merge 分支名称

### 2.4.5 删除分支

**不能删除当前分支，只能删除其他分支**

git branch -d b1 删除分支时，需要做各种检查

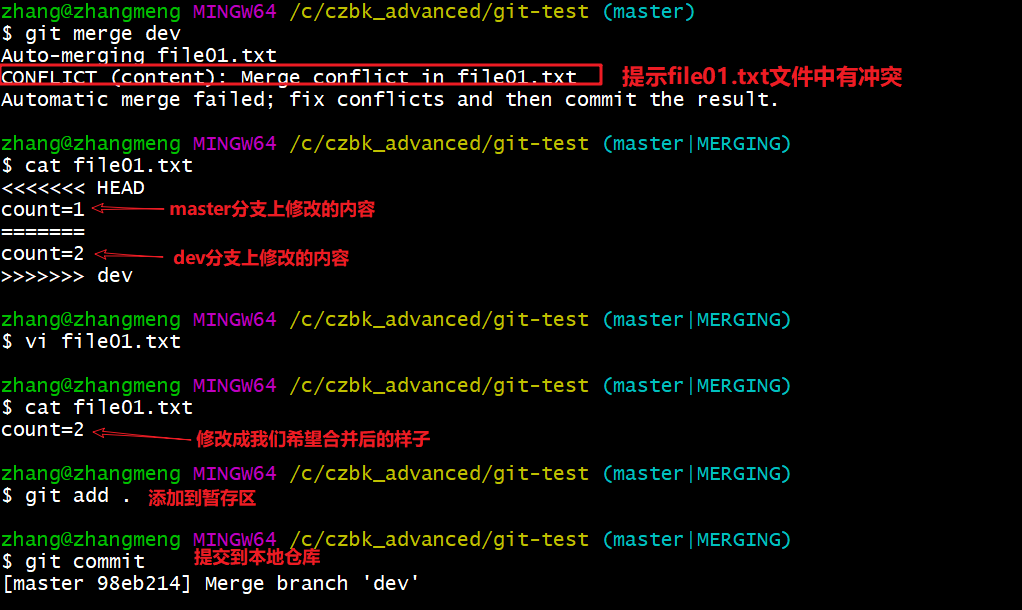
git branch -D b1 不做任何检查，强制删除

### 2.4.6 解决冲突

当两个分支上对文件的修改可能会存在冲突，例如同时修改了同一个文件的同一行，这时就需要手动解 决冲突，解决冲突步骤如下：

* + - 1. 处理文件中冲突的地方
      2. 将解决完冲突的文件加入暂存区(add)
      3. 提交到仓库(commit)

冲突部分的内容处理如下所示：



### 2.4.7 开发中分支使用原则与流程

几乎所有的版本控制系统都以某种形式支持分支。 使用分支意味着你可以把你的工作从开发主线上分离开来进行重大的Bug修改、开发新的功能，以免影响开发主线。

在开发中，一般有如下分支使用原则与流程：

master （生产） 分支

线上分支，主分支，中小规模项目作为线上运行的应用对应的分支；

develop（开发）分支

是从master创建的分支，一般作为开发部门的主要开发分支，如果没有其他并行开发不同期上线 要求，都可以在此版本进行开发，阶段开发完成后，需要是合并到master分支,准备上线。

feature/xxxx分支

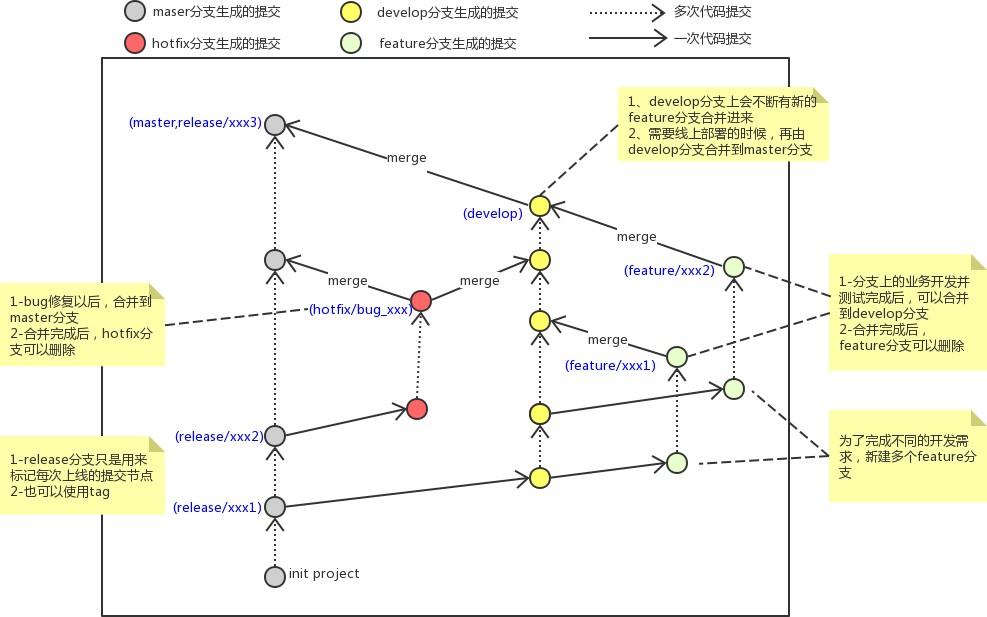
从develop创建的分支，一般是同期并行开发，但不同期上线时创建的分支，分支上的研发任务完 成后合并到develop分支。

hotﬁx/xxxx分支，

从master派生的分支，一般作为线上bug修复使用，修复完成后需要合并到master、test、

develop分支。

还有一些其他分支，在此不再详述，例如test分支（用于代码测试）、pre分支（预上线分支）等 等。



# 3、Git远程仓库

## 3.1、 常用的托管服务[远程仓库]

前面我们已经知道了Git中存在两种类型的仓库，即本地仓库和远程仓库。那么我们如何搭建Git远程仓库

呢？我们可以借助互联网上提供的一些代码托管服务来实现，其中比较常用的有GitHub、码云、GitLab等。gitHub（ 地址：https://github.com/ ）是一个面向开源及私有软件项目的托管平台，因为只支持Git 作为唯一的版本库格式进行托管，故名gitHub

码云（地址： https://gitee.com/ ）是国内的一个代码托管平台，由于服务器在国内，所以相比于

GitHub，码云速度会更快

GitLab （地址： https://about.gitlab.com/ ）是一个用于仓库管理系统的开源项目，使用Git作为代码管理工具，并在此基础上搭建起来的web服务,一般用于在企业、学校等内部网络搭建git私服。

## 3.2、 创建远程仓库



仓库创建完成后可以看到仓库地址，如下图所示:

## 3.3、配置SSH公钥

生成SSH公钥

ssh-keygen -t rsa

不断回车

如果公钥已经存在，则自动覆盖

Gitee设置账户共公钥

获取公钥

cat ~/.ssh/id\_rsa.pub



验证是否配置成功

ssh -T [git@gitee.com](mailto:git@gitee.com)

## 3.4、操作远程仓库

### 3.4.1 添加远程仓库

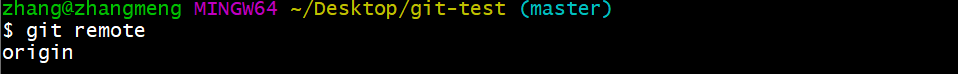
**此操作是先初始化本地库，然后与已创建的远程库进行对接**。命令： git remote add <远端名称> <仓库路径>

远端名称，默认是origin，取决于远端服务器设置

仓库路径，从远端服务器获取此URL

例如: git remote add origin [git@gitee.com](mailto:git@gitee.com):czbk\_zhang\_meng/git\_test.git

### 3.4.2 查看远程仓库

命令：git remote

### 3.4.3 推送到远程仓库

命令：git push [-f] [--set-upstream] [远端名称 [本地分支名][:远端分支名] ]

如果远程分支名和本地分支名称相同，则可以只写本地分支

git push origin master

-f 表示强制覆盖

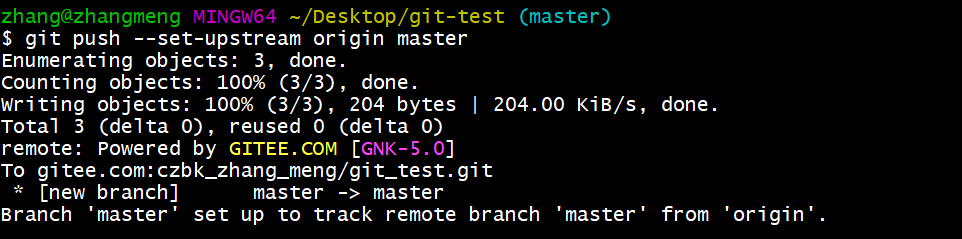
推送到远端的同时并且建立起和远端分支的关联关系。

--set-upstream

git push --set-upstream origin master

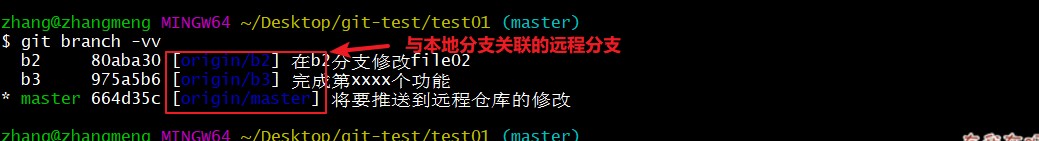
如果**当前分支已经和远端分支关联**，则可以省略分支名和远端名。

git push 将master分支推送到已关联的远端分支。



### 3.4.4 本地分支与远程分支的关联关系

查看关联关系我们可以使用 git branch -vv 命令



### 3.4.5 从远程仓库克隆

如果已经有一个远端仓库，我们可以直接clone到本地。命令: git clone <仓库路径> [本地目录]

本地目录可以省略，会自动生成一个目录



### 3.4.6 从远程仓库中抓取和拉取

远程分支和本地的分支一样，我们可以进行merge操作，只是需要先把远端仓库里的更新都下载到本 地，再进行操作。

抓取 命令：git fetch [remote name] [branch name]

抓取指令就是将仓库里的更新都抓取到本地，不会进行合并

如果不指定远端名称和分支名，则抓取所有分支。拉取 命令：git pull [remote name] [branch name]

拉取指令就是将远端仓库的修改拉到本地并自动进行合并，等同于fetch+merge

如果不指定远端名称和分支名，则抓取所有并更新当前分支。

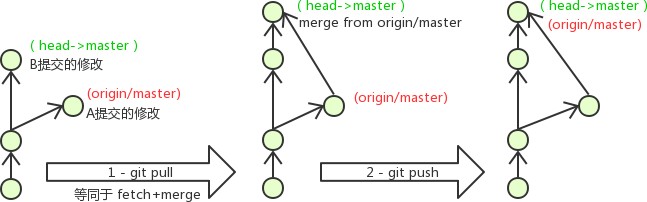
* + - 1. 在test01这个本地仓库进行一次提交并推送到远程仓库
      2. 在另一个仓库将远程提交的代码拉取到本地仓库



### 3.4.7 解决合并冲突

在一段时间，A、B用户修改了同一个文件，且修改了同一行位置的代码，此时会发生合并冲突。

A用户在本地修改代码后优先推送到远程仓库，此时B用户在本地修订代码，提交到本地仓库后，也需要 推送到远程仓库，此时B用户晚于A用户，**故需要先拉取远程仓库的提交，经过合并后才能推送到远端分 支**,如下图所示。



在B用户拉取代码时，因为A、B用户同一段时间修改了同一个文件的相同位置代码，故会发生合并冲 突。

**远程分支也是分支，所以合并时冲突的解决方式也和解决本地分支冲突相同相同**。

# Notes

切换分支前先提交本地的修改