目录

[What is git? 2](#_Toc529429621)

[Why use git? 3](#_Toc529429622)

[版本控制系统 3](#_Toc529429623)

[集中式版本控制系统： 3](#_Toc529429624)

[分支： 4](#_Toc529429625)

[How to use git? 5](#_Toc529429626)

[安装流程： 5](#_Toc529429627)

[初始化: 8](#_Toc529429628)

[Git commit规范: 10](#_Toc529429629)

[编写符合格式的commit message的工具： 13](#_Toc529429630)

[Git flow（工作流）规范 14](#_Toc529429631)

[什么是工作流 14](#_Toc529429632)

[Git Flow： 15](#_Toc529429633)

[码云规范 17](#_Toc529429634)

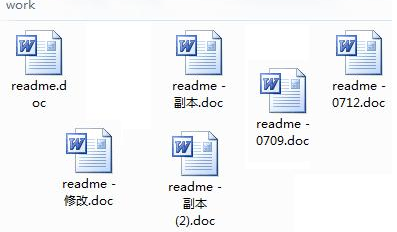
# What is git?

Git是由linus(创造Liunx系统的那个大神)用一个月时间写出的开源免费的分布式版本控制系统。具体细节请点击：[Git简史](https://git-scm.com/book/zh/v2/%E8%B5%B7%E6%AD%A5-Git-%E7%AE%80%E5%8F%B2)

# Why use git?

版本控制系统：记录文件或项目在何时由何人改了什么东西。

演变过程：言情小说作家写小说，想把某一段苦逼剧情给删除，但是以后可能又会加上这段剧情，没办法，只能把文件复制一份，然后继续创作。就这样一个月后，作家想把那段剧情加上去，然后去寻找那个复制的文件，这时候打开目录发现如下结果：



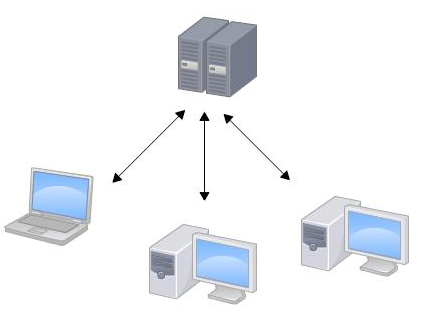
然后要在这些文件里面找到需要的文件。

缺点：费时费力，不易管理，不易查找改变的文件。

版本控制系统：作家把那一段苦逼剧情提交到系统中，记录时间，改变内容的简介，由谁提交的等等，这样作家在会过来看时，便可以直接找到文件所在位置。效果大致如下;

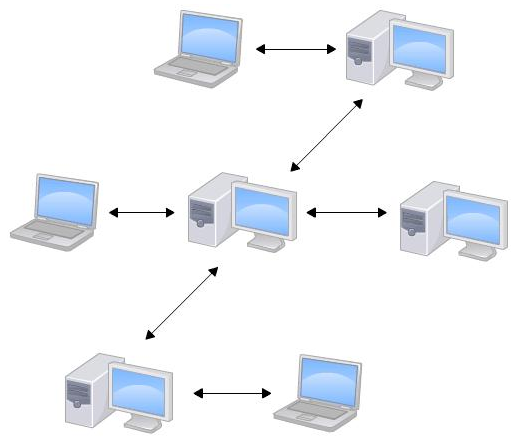


## 集中式版本控制系统：



工作方式：本地电脑只保存本次版本，对网络要求高，如果没有网络基本不能做任何版本管理的事情，只能在本地开发。代表性产品：SVN

分布式版本控制系统：



工作方式：每个电脑都是一个版块库，不用担心服务器坏了后，代码会消失。对版本管理需要的网络要求不高。代表性产品：Git

优点：安全性高（每个电脑都由版本库）。

网络依赖少（比如差异比较，查看历史及分支等本地即可做到）

速度快，简单易用。

## 分支：

事例：两个言情小说作家联合创作了一个小说（名称：办公室之恋）的男女主角到了相互爱慕，即将表白的时候，问题是：A作家想男的先表白，B作家想女的先表白。两人的解决方案是各写各的表白剧情（创建分支），然后再回到主体的剧情中（分支合并，解决两人代码冲突部分）。

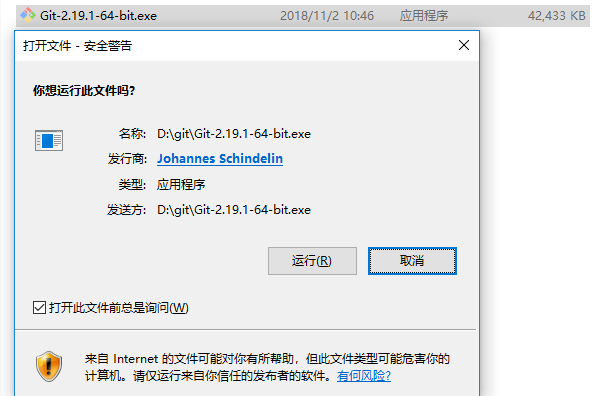
优点：便于项目不同环境下的管理（测试，生产等）及多人协同合作。

# How to use git?

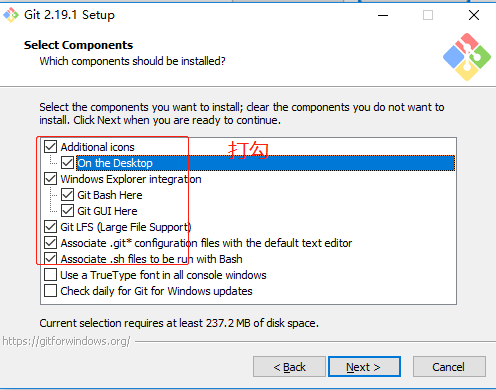
## 安装流程：

windows下:

下载地址：[Git for windows 下载地址](https://git-scm.com/download/win)



点击运行。然后一路点击next到如下界面



一路next直到安装完成。

## 初始化:

**设置用户信息：**

$ git config --global user.name “汉字姓名” #设置用户名

$ git config --global user.email “工作邮箱” #设置邮箱

# --global 参数用来设置全局变量（任何项目都是这些信息）

**配置本地SSH（**[**什么是SSH**](https://www.cnblogs.com/ftl1012/p/ssh.html)**）：**

$ ssh-keygen #开始进入生成公钥秘钥相关文件的流程

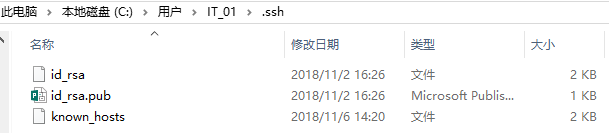


输入文件路径（要用提示的路径）：/c/Users/IT\_01/.ssh/id\_rsa #输入此路径然后按回车键

**查看.ssh文件：**

进入到 /c/Users/电脑用户名/.ssh/id\_rsa 目录。

id\_ras.pub为公钥文件



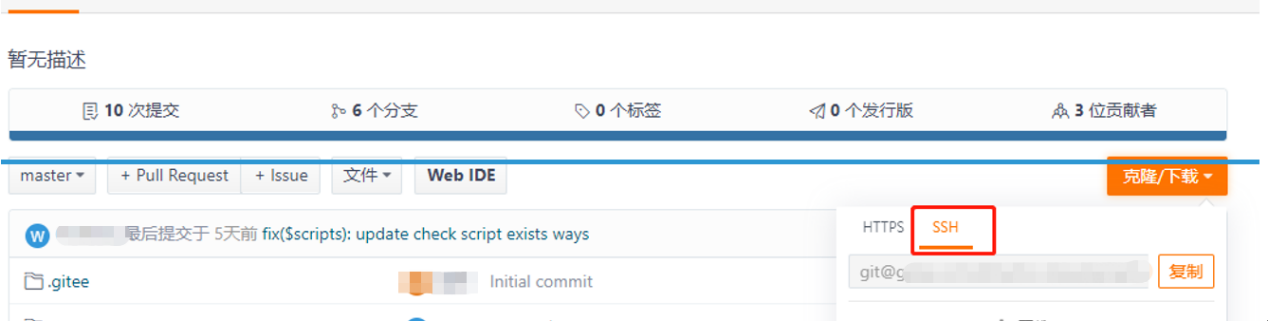
**下拉项目：**

把自己的公钥放在项目中。在码云中，进入个人设置>安全设置>SSH公钥。把公钥粘贴上去，并写上标题，即可。详细信息请点击：[码云中SSH 公钥设置](https://gitee.com/help/articles/4191)

双击打开Git Bash.

$ cd /d #进入到D磁盘中（也可以是E盘等等）

$ git init #在当前目录新建一个Git代码库

$ git clone [url] #url为项目所在SSH地址,地址如下图

项目中常用使用方法:

**下拉代码：**

$ git pull #下拉全部分支的代码

$ git pull origin 分支名 #下拉某一分支的代码

**提交代码（三步按顺序进行）：**

$ git add filename #添加某个文件到预提交阶段，git add \* 表示添加所有已改变文件（或第一次的所有文件）

$ git commit -m ‘提交信息’ #输入提交的备注信息

$ git push origin 远程分支名 #提交代码到版本控制服务器（如果断网等，可以提交到本地版本控制中心，输入 git push 即可）。

**创建分支：**

$ git checkout -b 本地分支名 远程分支名 #拉取远程分支并创建本地分支

强行下拉代码：(两步按顺序进行)

$ git reset --hard #在本地改变的文件但是不想上传到服务器时使用，本地未提交的代码被覆盖掉

$ git pull

**切换分支：**

$ git checkout 分支名

查看分支及当前所在分支：

$ git branch #高亮显示的为当前所在分支，其他是其他分支

**删除本地分支：**

$ git branch -D 本地需要删除的分支名称 #先切换到其他非删除的分支

**删除远程分支：**

$ git push origin --delete 远程分支名 #慎用

**合并分支：**

$ git merge 分支名 # 把另一个分支合并到当前分支中

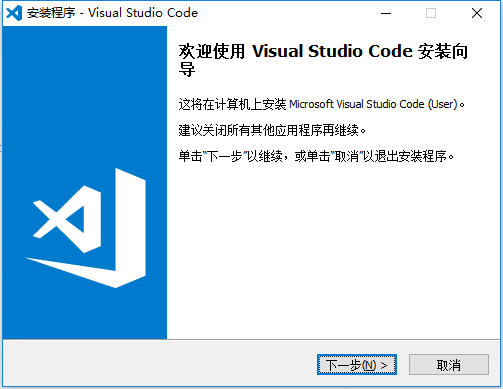
**进入某个提交版本(两步按顺序进行)：**

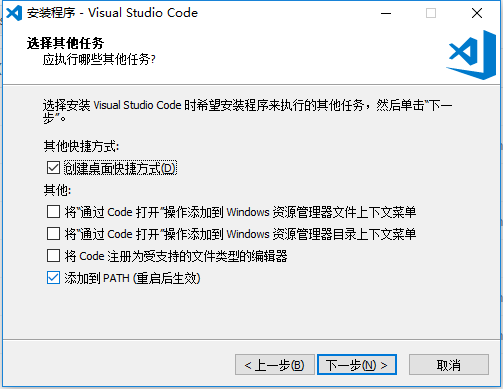
$ git log #查看需要进入的版本的hash值（40位的16进制数据【0-9/a-f】）

$ git reset –hard 那个版本的hash值 # 别忘了再切换回来

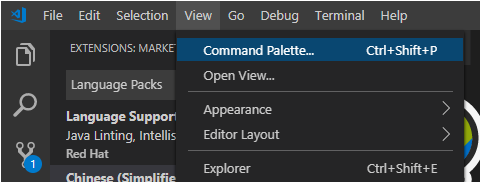
其他命令：[码云git命令大全](https://gitee.com/all-about-git)

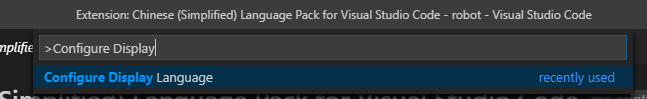
# VS CODE使用：

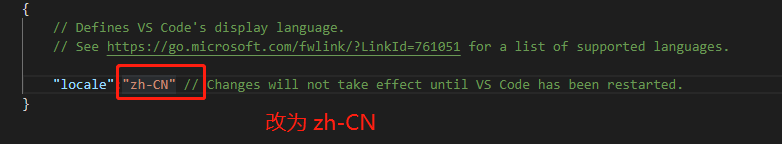


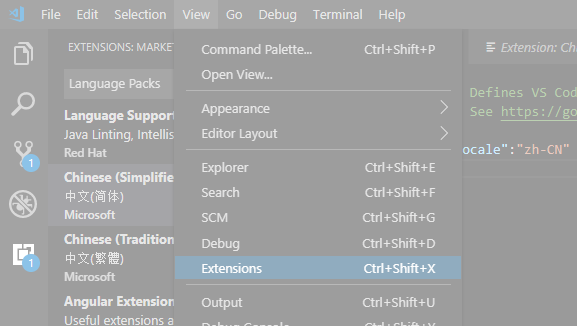


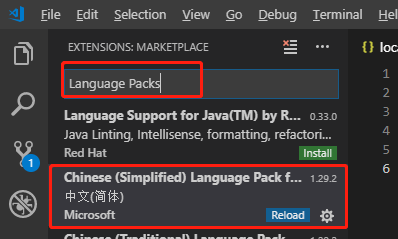
切换语言：详情点击：[URL](https://code.visualstudio.com/docs/getstarted/locales)

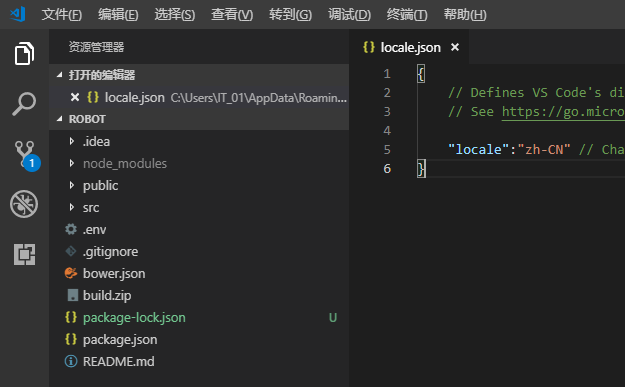








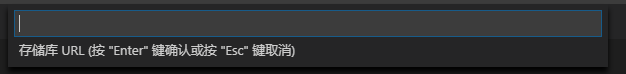
输入后，点击安装，重启



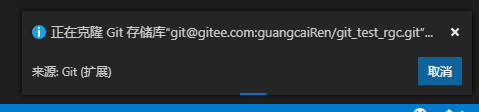
**下拉项目：**

Ctrl+shift+p 或者 F1

输入 git clone

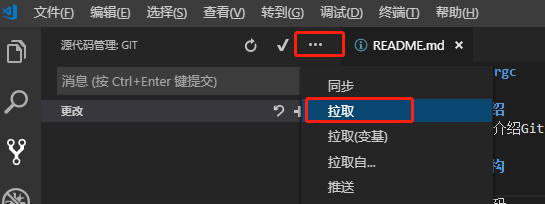


输入ssh地址



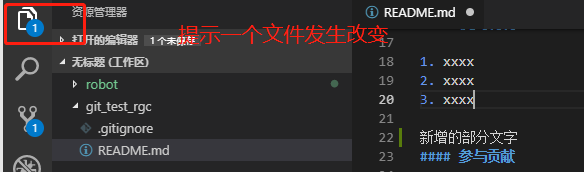
然后添加到工作区

**下拉代码：**



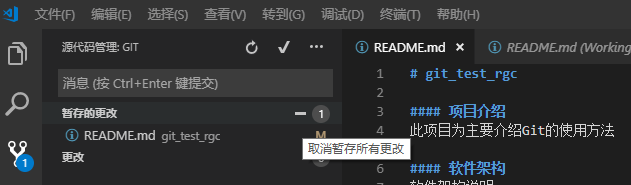
**查看当前所在分支：**

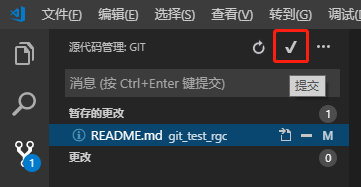
**提交代码：**

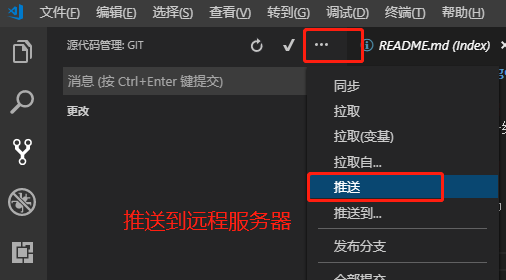






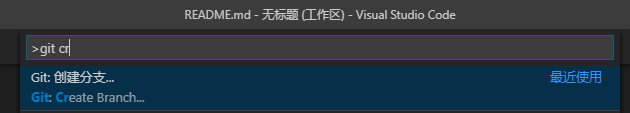


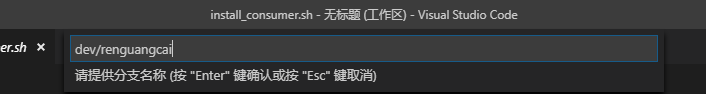
提交到本地仓库



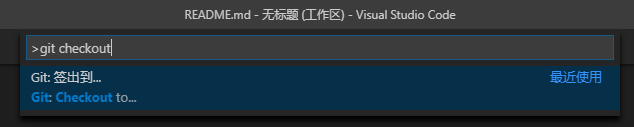
点击对勾，提交代码

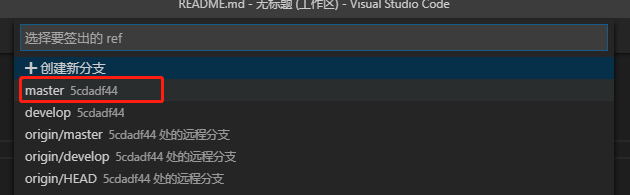
**创建分支：**



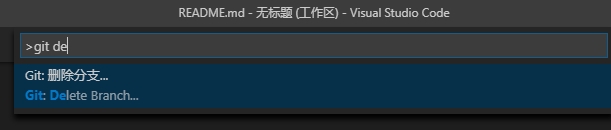


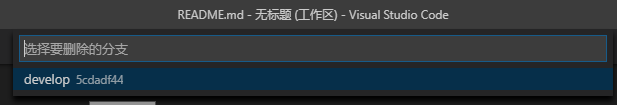
**切换分支：**



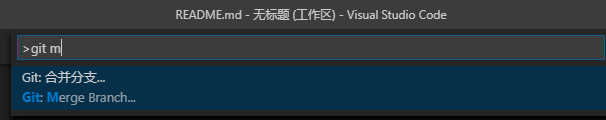


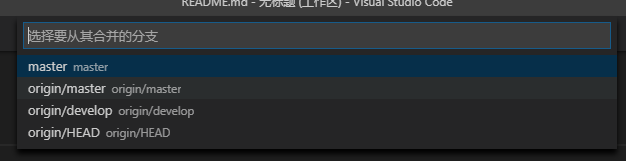
**删除分支：**





**合并分支：**





**进入某个提交版本(两步按顺序进行)：**

$ git log #查看需要进入的版本的hash值（40位的16进制数据【0-9/a-f】）

$ git reset –hard 那个版本的hash值 # 别忘了再切换回来

# Git commit规范:

一般写法：

$ git commit -m “我今天做了什么事情”

缺点：不知道我没什么这样改，不知道具体改了哪些东西，会对代码有哪些影响。

格式化的commit message（公司严格要求）：

格式如下：

Header部分: <type>(<scope>): <subject>

// 空一行

Body部分: <body>

// 空一行

Footer部分:<footer>

**Header**：大致描述（简洁扼要）

*type:* 用于说明commit的类别，只允许使用下面8个标识。

* feat：开发新功能（feature）
* fix：修补bug
* docs：文档修改（documentation）
* style： code review时修改代码格式（不影响代码运行的变动）
* refactor：重构（即不是新增功能，也不是修改bug的代码变动）
* test：增加缺少的测试或者纠正已有的测试
* chore：构建过程或辅助工具的变动
* perf: 改变代码用来提升性能

如果type为feat和fix，则该 commit 将肯定出现在 Change log 之中。其他情况（docs、chore、style、refactor、test）由你决定，要不要放入 Change log，建议是不要。

*scope:* 用于说明 commit 影响的范围，比如数据层、控制层、视图层等等，视项目不同而不同。如：$location, $browser, $compile, $rootScope, ngHref, ngClick, ngView, etc...

*subject:* 是 commit 目的的简短描述，不超过50个字符。

注意点：

* 以动词开头，使用第一人称现在时，比如change，而不是changed或changes
* 第一个字母小写
* 结尾不加句号（.）

**Body:**本次commit的详细描述，可以分为多行。如：

More detailed explanatory text, if necessary. Wrap it to about 72 characters or so.

Further paragraphs come after blank lines.

- Bullet points are okay, too

- Use a hanging indent

注意点：

* 使用第一人称现在时，比如使用change而不是changed或changes。
* 应该说明代码变动的动机，以及与以前行为的对比。

**Footer**:只有两种情况时使用

不兼容变动（Breaking changes）：如果当前代码与上一个版本不兼容，则 Footer 部分以BREAKING CHANGE开头，后面是对变动的描述、以及变动理由和迁移方法。如;

BREAKING CHANGE: isolate scope bindings definition has changed.

To migrate the code follow the example below:

Before:

scope: {

myAttr: 'attribute',

}

After:

scope: {

myAttr: '@',

}

The removed `inject` wasn't generaly useful for directives so there should be no code using it.

关闭BUG:如果当前 commit 针对某个bug，那么可以在 Footer 部分关闭这个 bug。如：

Closes #234 #关闭一个bug

Closes #123, #245, #992 #关闭多个bug

**Revert:** 如果当前 commit 用于撤销以前的 commit，则必须以revert:开头，后面跟着被撤销 Commit 的 Header。如：

revert: feat(pencil): add 'graphiteWidth' option

This reverts commit 667ecc1654a317a13331b17617d973392f415f02.

Body部分的格式是固定的，必须写成This reverts commit &lt;hash>.，其中的hash是被撤销 commit 的 SHA 标识符

## 编写符合格式的commit message的工具：

[Commitizen使用](https://github.com/commitizen/cz-cli)：

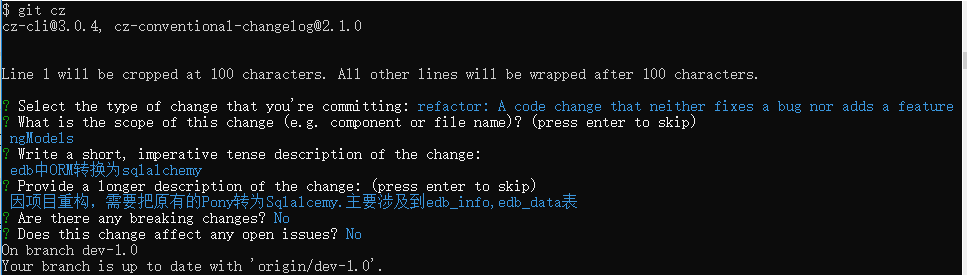
安装如下(按顺序执行)：

$ npm install -g commitizen #需要先[安装npm](https://www.cnblogs.com/interdrp/p/6779973.html)(node.js包管理器)

$ npm install -g cz-conventional-changelog

$ echo '{ "path": "cz-conventional-changelog" }' > ~/.czrc

以后，凡是用到git commit命令，一律改为使用git cz。这时，就会出现选项，用来生成符合格式的 Commit message。



依次需要操作：选择type类型，输入scope类型（没有就跳过），输入subject,输入body,输入是否有 break changes(如果是Yes，进入继续输入),输入是否有bug(issues)。然后提交。

其他相关知识：[Git Commit编写指南](http://www.ruanyifeng.com/blog/2016/01/commit_message_change_log.html)

Git flow（工作流）规范：

什么是工作流：在多人开发维护一个项目时，需要有一个规范的工作流程。使其在版本迭代，版本发布，分支合并，bug修复等工作中能够有条不紊，循序渐进，更好的管理代码。

工作流有如下三个流行:

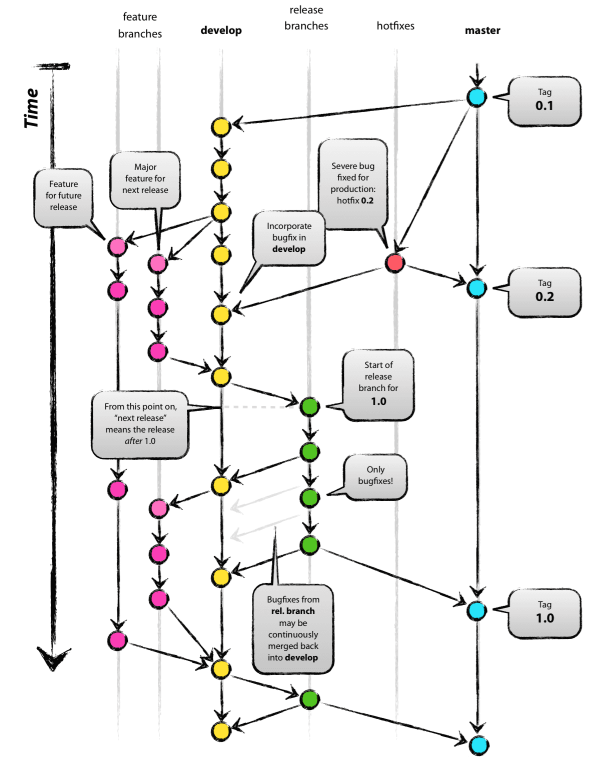
Git Flow

GitHub Flow

GitLab Flow

## [Git Flow](https://nvie.com/posts/a-successful-git-branching-model/)：

主要工作流如下图：



主要分支如下：

**master分支**: 主要负责发布到生存环境中的代码(production-ready)，每次提交到master的代码都必须打上标签标示版本号。

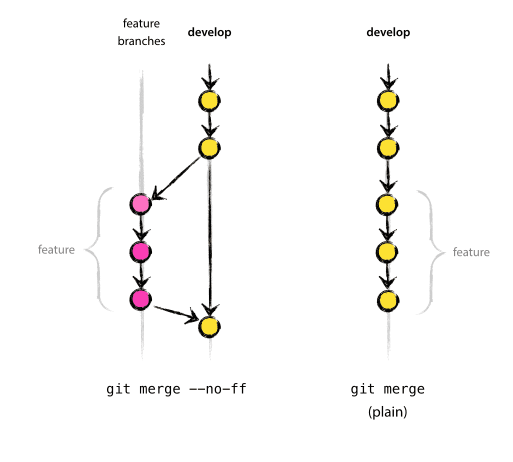
**develop分支**：开发分支，主要是辅助分支在开发完成后合并到develop分支，在功能开发（包括自测）完，测试结束（感觉没有bug），就要把develop分支合并到release分支。

release分支：负责预发布环境的分支（只能进行bug修复，不能进行新功能开发，并且在修复bug后要合并到develop和master分支）。在产品验收后，合并到master分支。

**Hotfixs分支**：负责线上bug修复。当线上出现bug(测不完的bug)时，从master分支拉取一个hotfixs分支，修复完bug后，合并到master和develop分支。

**Feature分支**：主要负责新功能模块的开发，开发自测完后合并到develop分支。命名比如：dev/myname(公司采用此方式)

注意点：在进行分支合并时，使用 git merge –no-ff 命令合并。可以记录被删除的分支。图如下：



除了master,develop的其他分支，在完成其任务后需要删除。

码云规范:

git每日提交，每个最小颗粒度的任务pull request一次，原则上每日都需要pull request

git 个人分支名称格式为：dev/youname

git账号全部为公司邮箱

git本地username 全部为 中文名