Software

Develop/Integration/Release

Guideline

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Data** | **Author** | **Status** | **Remarks** |
| V0.0.1 | 2021.04.21 | Mingfen XIAO | Init | Init Version |
| V0.0.2 | 2021.04.24 | Mingfen XIAO | Draff | Added Chapter 3.1.4 notes  Added Chapter 3.1.5  Added Chapter 6  Added Chapter 9 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Contents

[1.Branch Management 4](#_Toc101814069)

[1.1 master branch 4](#_Toc101814070)

[1.2 develop branch 4](#_Toc101814071)

[1.3 release branch 4](#_Toc101814072)

[1.4 feature branch 5](#_Toc101814073)

[2.Software Version Definition 5](#_Toc101814074)

[2.1 MCU Software Version Number Definition 5](#_Toc101814075)

[2.1.1 MCU Software Version Number Management Measures 5](#_Toc101814076)

[2.2.2 Software Components Version Number Management Measures 6](#_Toc101814077)

[2.2 SOC Software Version Number Definition 7](#_Toc101814078)

[3.Project Develop Workflow 7](#_Toc101814079)

[3.1 Get Develop Base 7](#_Toc101814080)

[3.1.1 Clone Repositories 7](#_Toc101814081)

[3.1.2 Checkout to Develop Branch 8](#_Toc101814082)

[3.1.3 Pull newest code form git Server 8](#_Toc101814083)

[3.1.4 Get and Updated the submodule[option] 8](#_Toc101814084)

[3.1.5 Config the commit rules 9](#_Toc101814085)

[3.2 Create Work Branch 9](#_Toc101814086)

[3.3 Development 9](#_Toc101814087)

[3.4 Merge develop Branch to Work Branch 10](#_Toc101814088)

[3.5 Fixed Conflict 10](#_Toc101814089)

[3.6 Push to Git Server 11](#_Toc101814090)

[3.7 Create Pull Request 11](#_Toc101814091)

[3.8 Code Review 12](#_Toc101814092)

[3.9 Merge feature branch to develop branch 14](#_Toc101814093)

[4.Submodule Develop Workflow 15](#_Toc101814094)

[4.1 Submodule Branch Mangament 15](#_Toc101814095)

[4.2 Platform Components Files Struct 16](#_Toc101814096)

[5.Integration&Release Workflow 16](#_Toc101814097)

[5.1 Updated Code from Git Server 17](#_Toc101814098)

[5.2 Create Release Branch 17](#_Toc101814099)

[5.3 Modified the Version Number 17](#_Toc101814100)

[5.4 Modified the Debug Switch/Config Info 17](#_Toc101814101)

[5.5 Compiling 17](#_Toc101814102)

[5.6 Smoke Test 17](#_Toc101814103)

[5.7 Review Smoke Test Result 17](#_Toc101814104)

[5.8 Push Release Branch to Git Server 18](#_Toc101814105)

[5.9 Create Tag and Release 18](#_Toc101814106)

[5.10 Merge Release Branch to Develop Branch 20](#_Toc101814107)

[6.Create and Configuration Project Code Repositories 20](#_Toc101814108)

[6.1 Create Repositories 20](#_Toc101814109)

[6.2 Change the main branch to master branch 22](#_Toc101814110)

[6.3 Add Collaborators 22](#_Toc101814111)

[6.4 Link exernal Links 23](#_Toc101814112)

[6.5 Create Autolinks 23](#_Toc101814113)

[6.6 Clone Repository 23](#_Toc101814114)

[6.7 Create develop branch 24](#_Toc101814115)

[6.8 Definition The Commit Message Format 24](#_Toc101814116)

[6.9 Create Init Version Code & Create Compilation[Option] 24](#_Toc101814117)

[6.10 Added Submodules[Option] 25](#_Toc101814118)

[6.11 Commit & Push 25](#_Toc101814119)

[6.12 Set the develop branch to default branch 26](#_Toc101814120)

[7.CICT Buliding 26](#_Toc101814121)

[8.Git Command Reference 26](#_Toc101814122)

[9.Commit Message Rules 26](#_Toc101814123)

此文档目的是为了规范软件开发、集成、发布的流程和保证软件质量。同时也是为使用git进行代码管理的软件工程师提供一定的指导。

# 1.Branch Management

Diagram

Description automatically generated 当前CSC的代码都是基于git来进行管理，下图为代码分支管理的示意图。

如上图所示，在软件开发的全生命周期主要存在四种类型的分支：

## master branch

**master**，名字也可能是**main**。主分支，当软件达到一定成熟度时，将软件从其他的分支合并到主分支作为存档。软件成熟度的衡量可以根据客户的交付节点或者根据MCL的节点。MLC可以通过process过程中的check list来评估软件的成熟度；

## 1.2 develop branch

**develop**，开发分支，也是软件开发过程中的默认分支。develop分支用于存放各个软件工程师的所有软件开发和修改，同时也是持续集成（CI）的工作分支。当软件工程师对软件有更新（功能新增或者问题修复）时，可以通过Pull Request（PR）的方式将开发好的代码合并到develop分支进行集成。

## 1.3 release branch

release，软件释放分支，用于软件释放的分支，由集成工程师从develop分支branch出来，在完成软件集成测试或者软件冒烟测试后进行释放。集成工程师在release分支上增加Tag的形式来完成软件释放的活动。

当需要比较长的时间来执行集成测试，可以先从develop分支branch出一个release分支进行一次pre release，在集成测试完成并且修复完集成测试发现的问题后，在进行一次正式的release；

当需要基于release的软件版本进行维护和开发的时候，则可以直接在release分支上进行维护，在达到一定成熟度或者合适的时间时在合并的develop；

release分支的命名如下所示：

**release/v<version number>**

比如说：**release/v1.1.0**

## 1.4 feature branch

feature分支为软件工程师的工作分支。当软件工程师需要实现新的功能或者进行问题修复时，需要从develop分支上branch出一个feature分支进行功能开发或者问题修复，在功能开发或者问题修复完成后，需要将feature分支push到服务器，并且通过Pull Request来请求合并到develop分支。

feature分支的命名规则如下：

**feature/<JIRA-ID>/<descriptive-name>**

例如：**feature/MFC5J3-415/update-rte-interface**

***Notes :***

1. ***当遇到feature开发时涉及比较多的关联模块时，建议及时将更新合并到develop分支，以减小合并过程中解决冲突的工作量（基于AUTOSAR架构的配置，很多配置项是相互关联的，所以对于MCU的开发工程师需要特别的注意）;***

# 2.Software Version Definition

## 2.1 MCU Software Version Number Definition

### 2.1.1 MCU Software Version Number Management Measures

* The software shall contain version number and separator.
* The version number part consists of three numbers which there contains MAJOR\_VERSION ,MINOR\_VERSION and PATCH\_VERSION;
* The Mcu/Ecu software number add the prefix of "MCU\_SW\_",the version number identifier is MCU\_SW\_MAJOR\_VERSION,MCU\_SW\_MINOR\_VERSION and MCU\_SW\_PATCH\_VERSION.
* The three number use the uint8 data type and the range is 00-99;
* The separator use the '.';
* The sample of version information is 01.00.00;
* The VerInfo components shall provide API to get the version number;
* The version number have two ways to upgrade the version number;
* When the project starts, The software team(SPM,SW Integrator/SW Arch/SW Developer) should define a way to upgrade the version;
* General, the software version number shall mapping to DID(e.g.ECU software version,0xF189) from Dcm.

**Notes : The MCU Software Version Number same of the ECU Software Version Number.**

The MCU Software Version number have two ways following:

One :

* When developer updated any code,the PATCH\_VERSION number shall increase 1(One commit,the PATCH\_VERSION will increase);
* When the Integrator released a version software,the MINOR\_VERSION shall increase 1 and the PATCH\_VERSION shall clear to 0;
* When the software achieve the gate or milestone(may be from customer),the MAJOR\_VERSION shall increase 1,then the MINOR\_VERSION and PATCH\_VERSION shall clear to 0;
* The developer just modified the PATCH\_VERSION number;
* The MINOR\_VERSION and MAJOR\_VERSION number only modified by integrator.

Two :

* When the software detail design(the code implemented),the PATCH\_VERSION number shall increase 1;
* When the software arch updated(interface updated[RTE interface]/new feature/schedule table updated/e.t.c),the MINOR\_VERSION shall increase 1 and the PATCH\_VERSION shall clear to 0;
* When the Integrator release software,the MAJOR\_VERSION shall increase 1,then the MINOR\_VERSION and PATCH\_VERSION shall clear to 0;

### 2.2.2 Software Components Version Number Management Measures

* The software shall contain version number and separator;
* The version number part consists of three numbers which there contains MAJOR\_VERSION ,MINOR\_VERSION and PATCH\_VERSION;
* The software components number add the prefix of "<SWCs>\_SW\_",the version number identifier is <SWCs>\_SW\_MAJOR\_VERSION,<SWCs>\_SW\_MINOR\_VERSION and <SWCs>\_SW\_PATCH\_VERSION.
* The three number use the uint8 data type and the range is 00-99;
* The separator use the '.';
* The sample of version information is 01.00.00;
* The VerInfo components shall provide API to get the version number;
* The version number have one ways to upgrade the version number;
* The software components shall definition a component id,like the "<SWCs>\_MODULE\_ID",the component may be use for CICT;
* The Module ID start for 0x8001,For each new addition to the software module ID,increase 1;

***Notes : The module ID is determined by the software architecture.***

The Software Components version number have one ways following:

* When the software components detail design (the code implemented or bug fixed),the PATCH\_VERSION number shall increase 1;
* When the software components arch updated (interface updated/state machine updated),the MINOR\_VERSION shall increase 1 and the PATCH\_VERSION shall clear to 0;
* When the software components added new feature (alse arch updated),the MAJOR\_VERSION shall increase 1,then the MINOR\_VERSION and PATCH\_VERSION shall clear to 0;

## 2.2 SOC Software Version Number Definition

TBD

# 3.Project Develop Workflow

软件开发工程师可以按照以下的步骤来获取仓库，切换分支，提交代码以及提交PR直至代码合并到develop分支。

**Notes: 以下步骤是通过git bash命令行来执行的。**

## 3.1 Get Develop Base

### 3.1.1 Clone Repositories

*git clone <url> [repo name]*

<url>:git repositories的链接

[repo name]:git repositories的名称，如果不填写则默认和git repositories一致

**Notes: Clone操作只有在需要从git server拉取仓库的时候才需要进行操作。**

### 3.1.2 Checkout to Develop Branch

通过以下命令切换到develop分支，一般情况下，所有的软件开发的base都是从develop分支branch出去的。

*git checkout develop*

### 3.1.3 Pull newest code form git Server

在软件功能开发之前，需要通过3.1.2的命令切换在develop分支，并通过以下命令拉去git server最新的代码

*git pull origin develop*

or

*git fetch origin develop*

**Notes:**

1. **为了避免开发完成合并的难度，每次开发都需要从最新的develop进行分支的创建。**
2. **为了保证后续软件开发的效率，在拉取develop最新代码后，建议先进行一次编译和运行，以确认开发基线是否存在问题。**

### 3.1.4 Get and Updated the submodule[option]

如果仓库使用了submodule的形式，则需要使用以下命令将submodule的代码拉取到本地。

初始化仓库的submodule:

*git submodule init*

更新仓库所有submodule的代码

*git submodule update*

如果需要对某一个submodule进行操作，也可以直接进入到submodule的文件夹，通过以下命令在操作

*git pull origin <branch name>*

or

*git fetch origin <branch name>*

**Notes:**

1. **通过上述命令来获取的submodule的前提是，开发者需要有相关submodule仓库的访问权限；**
2. **如果有部分仓库没有访问权限，可能会影响其他仓库的操作，即通过git submodule update命令后，即使有访问权限的子仓库也没有pull下来。**
3. **对于2中描述的情况，建议进入到子仓库的文件夹目录在开启git bash的命令来单独操作。当进入到子仓库文件夹的命令行后的操作和普通的操作就一致了，即可以通过git pull/git fetch等命令来讲目标版本的内容拉取到本地的主仓库。**

### 3.1.5 Config the commit rules

此步骤为了规范commit message的格式，所以在配置仓库的时候要求集成工程师需要将代码提交的格式存放在一个.gitcommit的文本文件中（此文件放置在仓库的根目录下）

软件工程师可以通过操作以下命令来使能commit message格式的规则：

*git config commit.templat .gitcommit*

Notes：代码的Commit message的说明请参考第9章节的说明；

## 3.2 Create Work Branch

通过以下命令基于develop分支创建软件工程师的开发工作分支。

*git checkout -b feature/<JIRA-ID>/<descriptive-name>*

Or

*git branch feature/<JIRA-ID>/<descriptive-name>*

*git checkout feature/<JIRA-ID>/<descriptive-name>*

## 3.3 Development

在此步骤中进行软件开发的活动。代码编码，配置修改等

软件工程师开发完功能或者完成问题修复后，需要进行编译和测试后，在通过以下命令进行提交代码。

将代码修改提交到暂存区：

*git add .*

将代码从暂存区提交到本地仓库：

*git commit -m “<message>”*

## 3.4 Merge develop Branch to Work Branch

由于是多分支并行开发的，所以当一个feature开发完成时，可能有其他的feature合并到develop，为了增加提交PR的成功率。在将feature分支合并到develop分支之前并且develop分支有更新的时候，需要通过以下操作来进行一次合并。

切换到develop分支：

*git checkout develop*

更新develop分支的代码：

*git pull origin develop*

切换到当前的开发分支：

*git checkout feature/<JIRA-ID>/<descriptive-name>*

将develop分支的更新merge到feature分支

*git merge –no-ff develop*

Or

*git merge develop*

## 3.5 Fixed Conflict

如果merge出现了冲突，需要更新提示进行冲突的修改。之后在通过以下命令将提交解决冲突的代码进行提交。

将代码修改提交到暂存区：

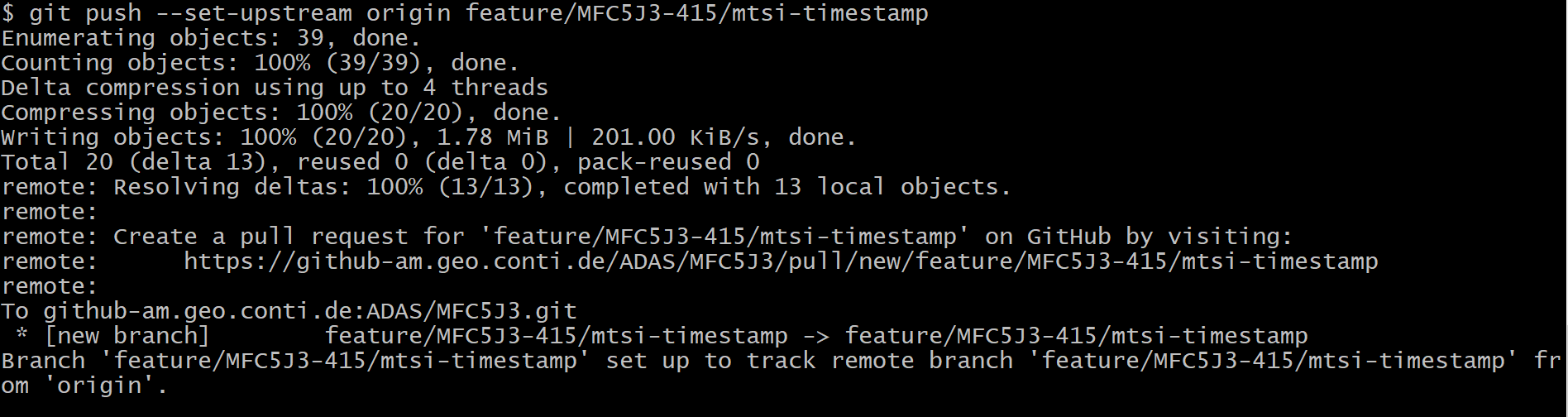
*git add .*

将代码从暂存区提交到本地仓库：

*git commit -m “<message>”*

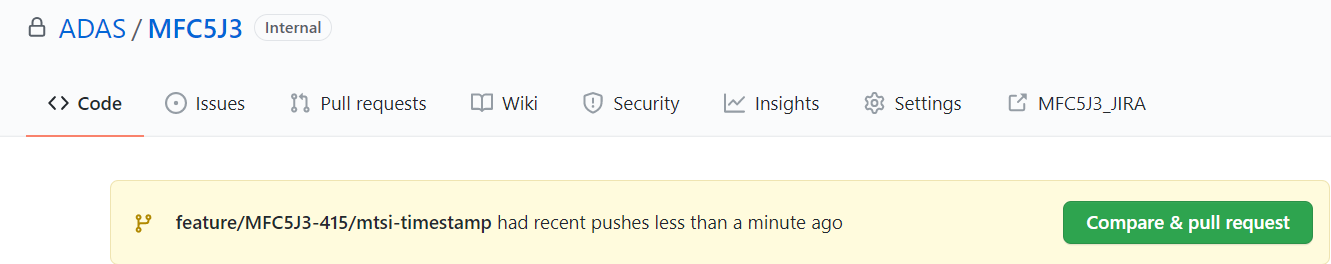
## 3.6 Push to Git Server

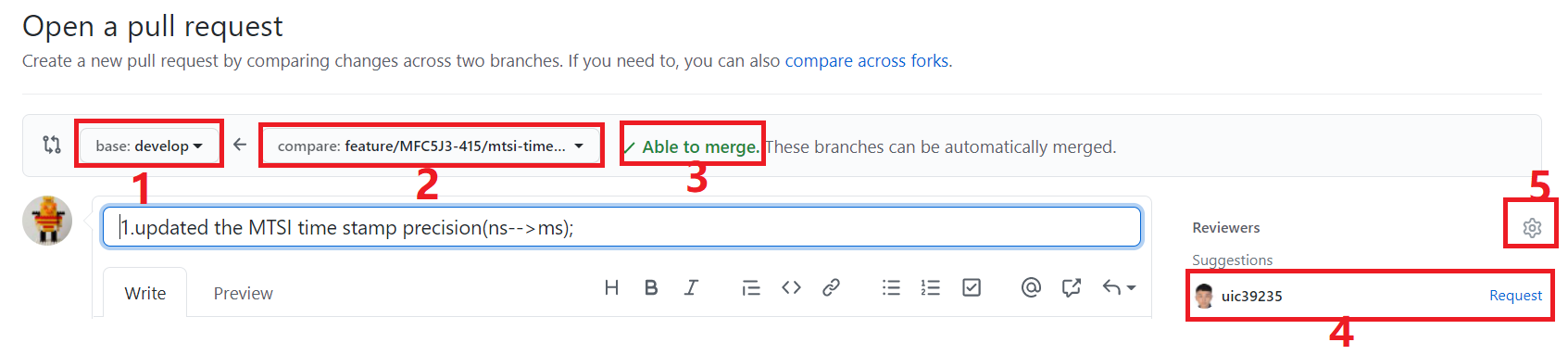
在代码编译测试没有问题后，需要通过以下命令将代码提交到git server。

*git push --set-upstream origin feature/<JIRA-ID>/<descriptive-name>*

**Notes: 在进行push之前，一定需要保证代码能够编译通过以及相关的测试没有问题。**

## 3.7 Create Pull Request

当执行完push之后，通过网页进入git仓库，可以出现以下界面：

通过点击绿色的“Compare & pull request”按钮来创建一个PR。出现如下的界面：

红色框1，设置为develop;

红色框2，设置为需要合并的feature分支

红色框3，**一定需要显示为Able to merge**，否则就存在冲突，需要章节3.4-3.6进行更新并且重新提交；

红色框4，选择推荐的人员进行code review

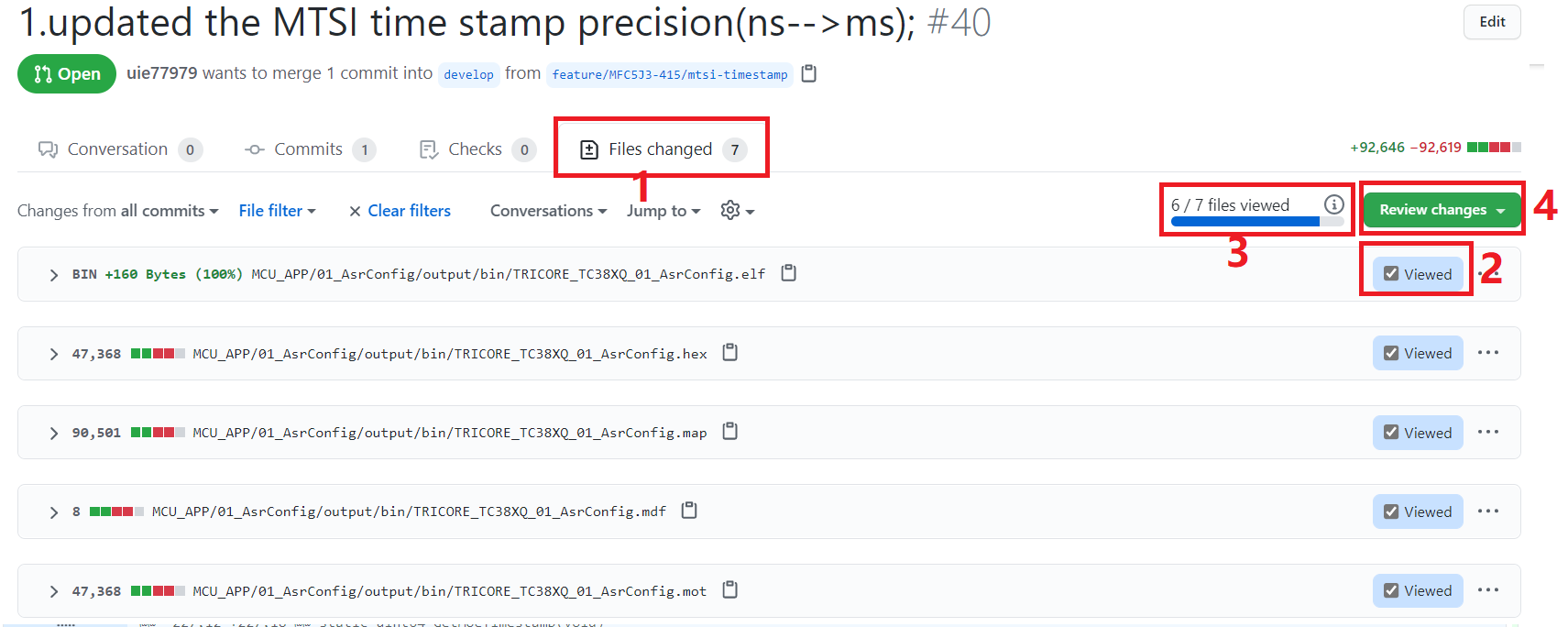
红色框5，如果需要红色框4之外的人进行code review,则可以通过红色框5来进行选择。

选择上述选项后，往下拉，有一个绿色的”Create pull request”的按钮，点击即可发起code review。

## 3.8 Code Review

软件工程师发起PR后，reviewer会收到一个邮件。Reviewer可以通过邮件进入到review的界面进行Code Review过程，也可以直接在git的网页进入。

Code Review的界面如下图所示：



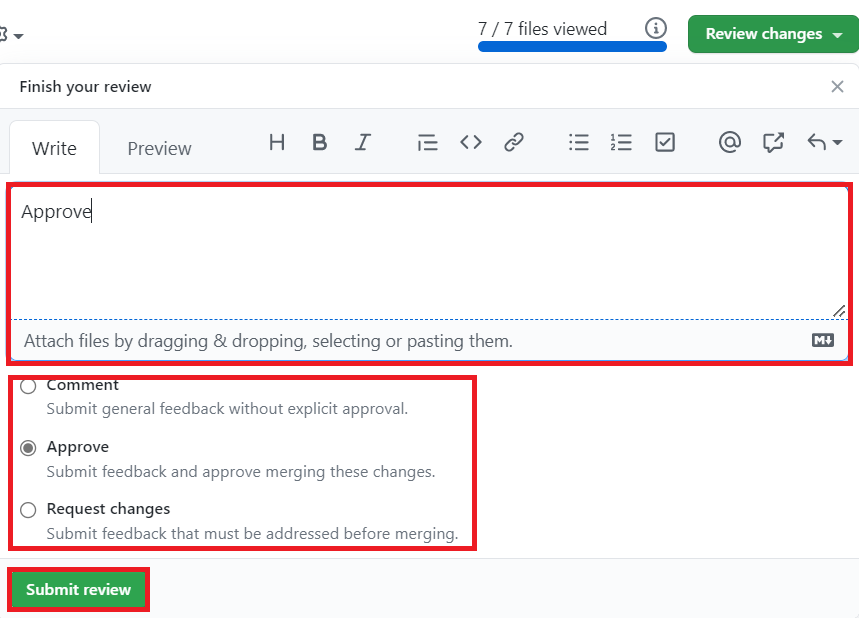
Reviewer需要按照如下的操作进行review:

红色框1，为本次提交变化的文件列表，Reviewer需要进入到此标签页针对变化的文件进行review;

红色框2，当Reviewer确认文件审核完毕，则需要在红色框2的位置打一个勾（点击即可）；

红色框3，显示review文件的进度；

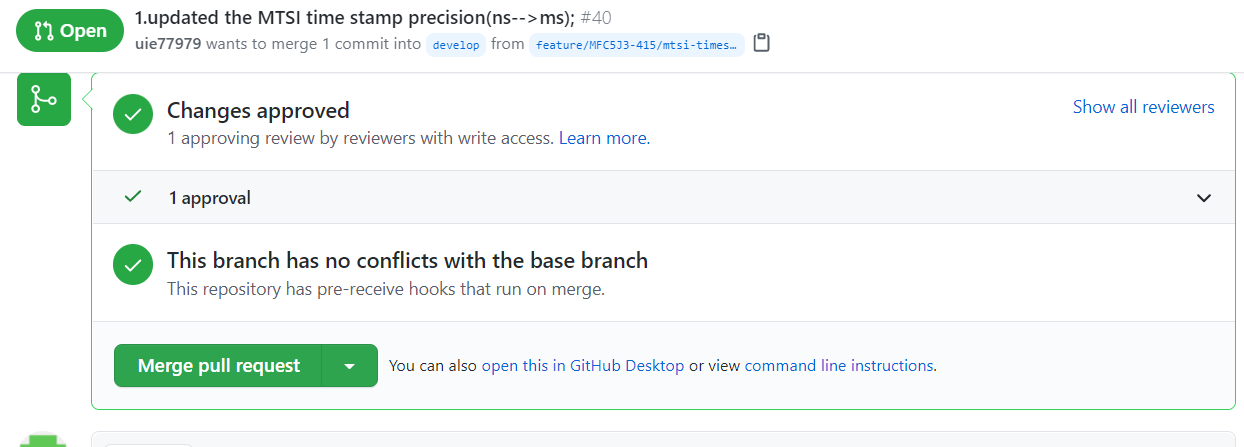
红色框4，当全部文件review完成后，在红色框4中增加review comments，即完成review。根据实际的review情况如下图所示，增加comments.



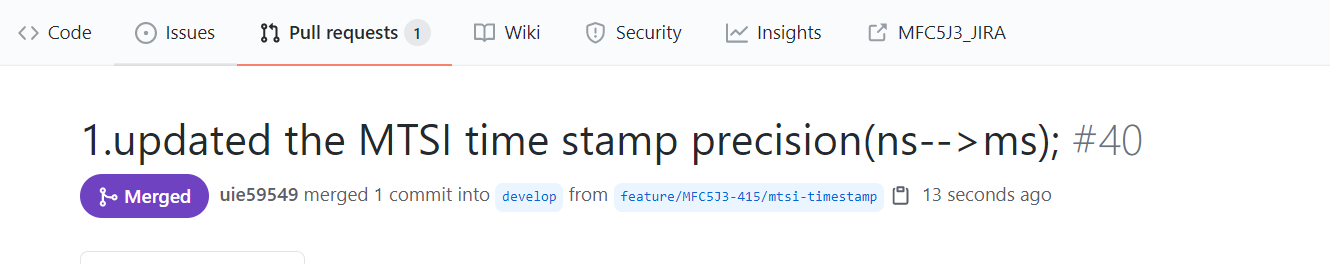
当点击“Submit review”后，review工作就完成了。

## 3.9 Merge feature branch to develop branch

如下图所示，当review完成也没有冲突需要解决后，就可以通过如下图所示的按钮将feature分支的代码合并到develop上。



当合并完成后，PR的状态就会冲Open状态变成merge状态，如下图所示：



# 4.Submodule Develop Workflow

## 4.1 Submodule Branch Mangament

当存在多个项目共用一个模块的话即平台模块，可以单独建立feature仓库，项目仓库就通过submodule的方式将平台模块引入到项目中使用。

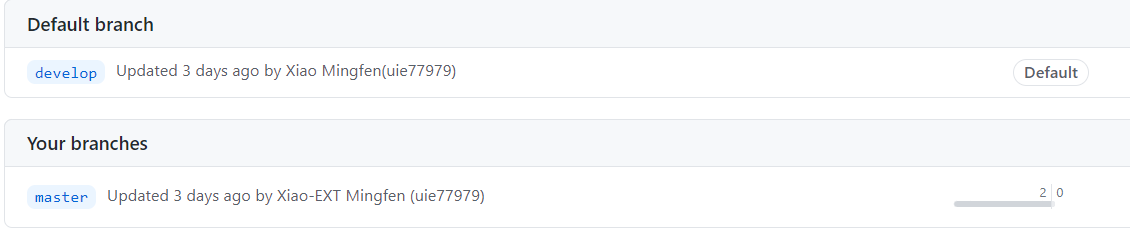
当feature仓库的更新需要合并到项目仓库时，还是需要按照第3章节的wrokflow提交PR的方式来合并；

Feature仓库的提交模式，建议参考正常的项目仓库的开发流程，即第3章节，但是不做强制要求，由feature开发团队自己协商决定；

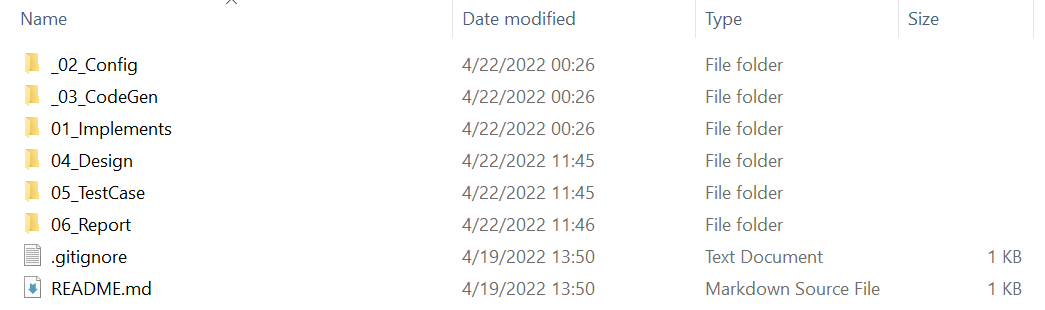
Submodule的分支可以参考以下两张图：

1.如下图所示用于放置外部供应商提供的协议栈，所以直接使用release分支。

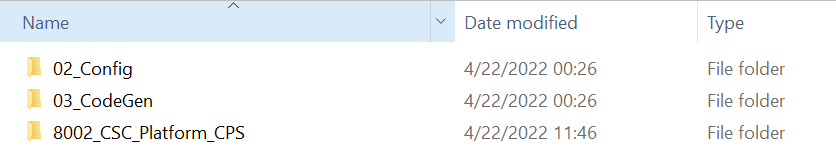
2.如下图所示用于放置自己开发的platform component的模块，按照正常的项目的分支管理进行



## 4.2 Platform Components Files Struct

对于自主开发的平台的软件模块，可以参考如下图所示的文件结构进行设计

* + \_02\_Config: 配置文件，可以是手动维护也可以是通过Code Generation生成的，跟着项目走
  + \_03\_CodeGen: 代码生成器，当CodeGen里面的脚本需要跟着项目修改，则此文件夹跟着项目走；如果不修改，则作为平台固有的文件
  + 01\_Implements: 静态代码
  + 04\_Design: 放置设计文档，包括架构设计，详细设计以及使用指导等
  + 05\_TestCase: 放置测试用例，包括但不限于单元测试，集成测试用例等
  + 06\_Report: 放置测试报告，包括但不限于单元测试报告，集成测试报告等

集成到项目中的文件结构如下所示：

# 5.Integration&Release Workflow

以下工作主要是集成工程师来执行和操作。

## 5.1 Updated Code from Git Server

此步骤是将最新的代码从Git Server拉下来，包括Submodule的更新（如果有的话）;

## 5.2 Create Release Branch

基于最新的develop分支，切换一个release分支用来执行软件释放的活动；

Release branch的命名应该符合1.3章节定义的命名规则。

## 5.3 Modified the Version Number

更新软件的版本号；

MCU的版本号: VerInfo.h

SOC的版本号：TBD

**Notes: 版本号的更新应该遵循第2章节的规则**

## 5.4 Modified the Debug Switch/Config Info

此步骤是为了检查调试开关或者一些配置信息的情况；

## 5.5 Compiling

此步骤是编译，生成可执行文件；

## 5.6 Smoke Test

为了保证软件释放的质量，建议每次释放之前至少要执行一次冒烟测试。

具体的测试项有软件团队进行商量后确定，下面的附件可以作为一个参考模板。



## 5.7 Review Smoke Test Result

如果存在冒烟测试fail的项，则需要软件团队对冒烟测试的结果进行review，如果确认fail项可以不影响此次释放，则可以进行有条件放行的软件释放。

## 5.8 Push Release Branch to Git Server

在第4.6或者4.7确认可以正式释放后，则需要将release分支push到Git Server上。步骤如下：

将更改添加到暂存区：

*git add .*

将更改从暂存区提交到本地仓库：

*git commit*

Or

*git commit -m “<message>”*

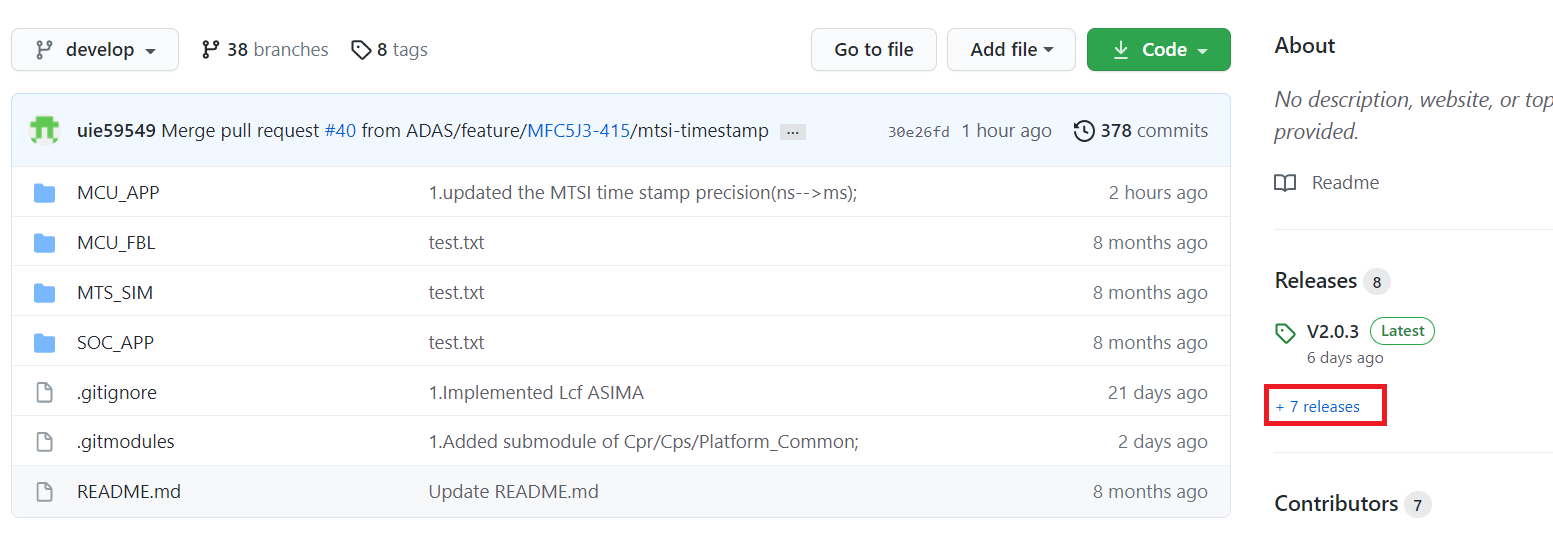
将更改从本地仓库提交到远程仓库（Git Server）

*git push --set-upstream origin release/v<x>.<y>.<z>*

## 5.9 Create Tag and Release

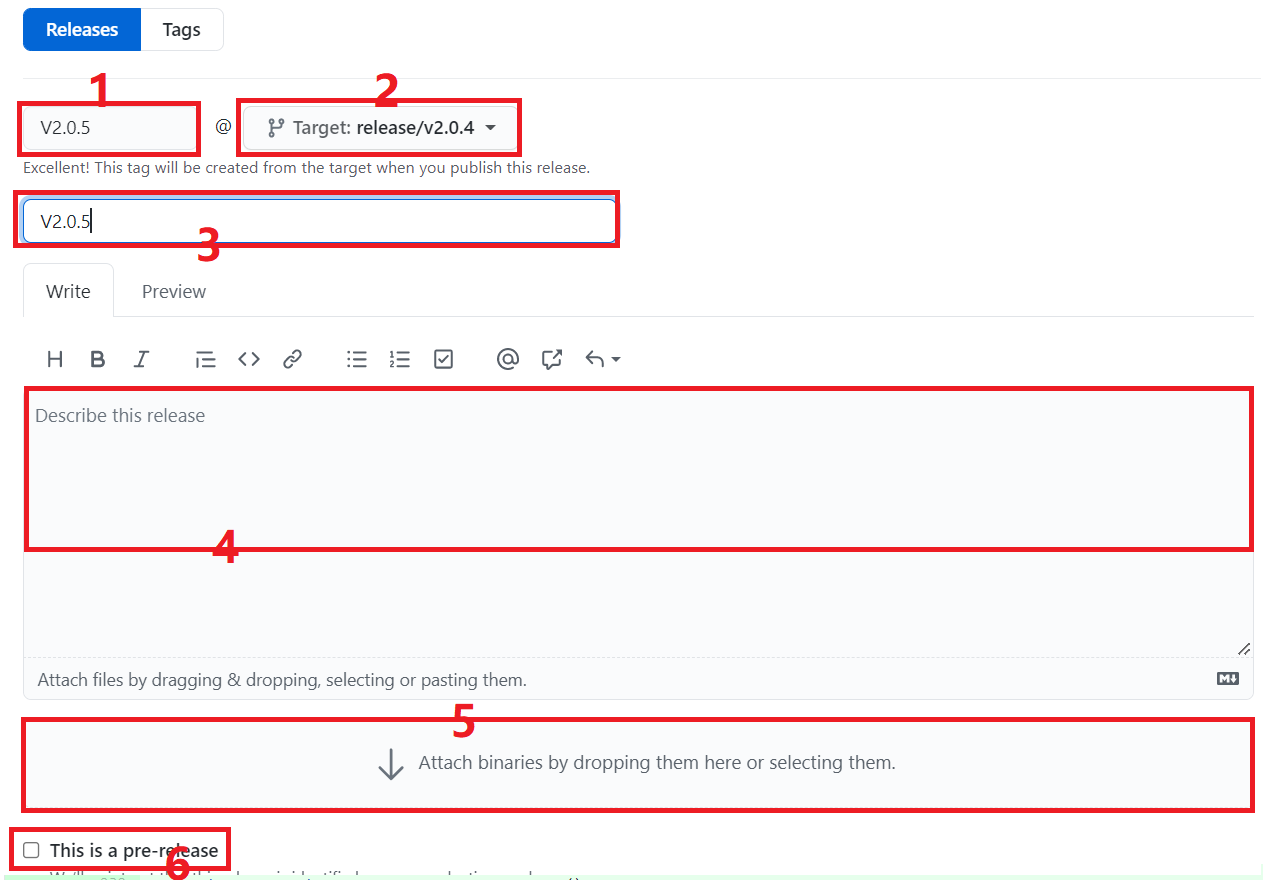
当release分支提交到Git Server后，则可以在git 网页上进行release的操作：

如下图所示，在git项目的主页上找到如红色框的release的按钮，进入release的界面。



 进入release界面如下图所示

可以通过上图红色框的按钮可创建新的release，创建的界面



红框1：填写release的版本，TAG的名称

红框2：选择对应的release的branch

红框3：填写release title

红框4：填写release notes

红框5：上传Smoke Test Report/release notes

红框6：如果需要将此个版本的作为pre release，则将红框6的功能使能起来

填好上述内容后，点击”Publish release”进行释放。

**Notes:当需要将pre release转换为正式release，则可以通过编辑pre release来将其转换为正式的release.即只需要将红框6中关闭即可。**

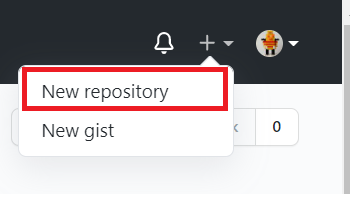
## 5.10 Merge Release Branch to Develop Branch

当需要将release分支合并到develop分支时，则也需要按照PR流程进行合并。具体操作可以参考第3章节的描述。

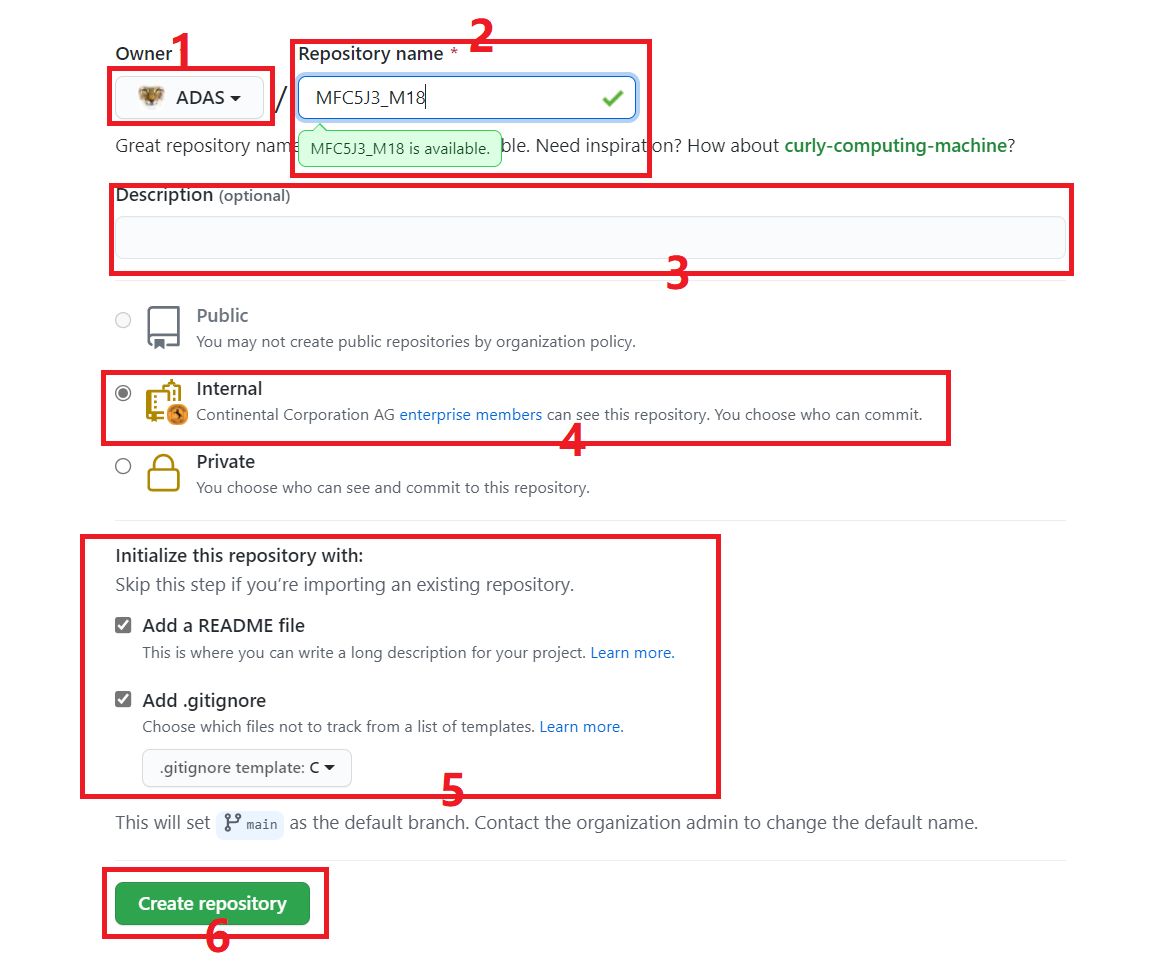
# 6.Create and Configuration Project Code Repositories

此章节用来指导创建和配置代码仓库。

## 6.1 Create Repositories

 如下图所示，在网页的github上通过如下图所示的红色框中的按钮来进行创建仓库。

点击后就会出现如下图所示的界面：



如上图所示：

红框1：设置仓库的Owner，项目仓库的Owner需要设置为ADAS；

红框2：设置仓库的Name，根据实际的项目名字设置即可；

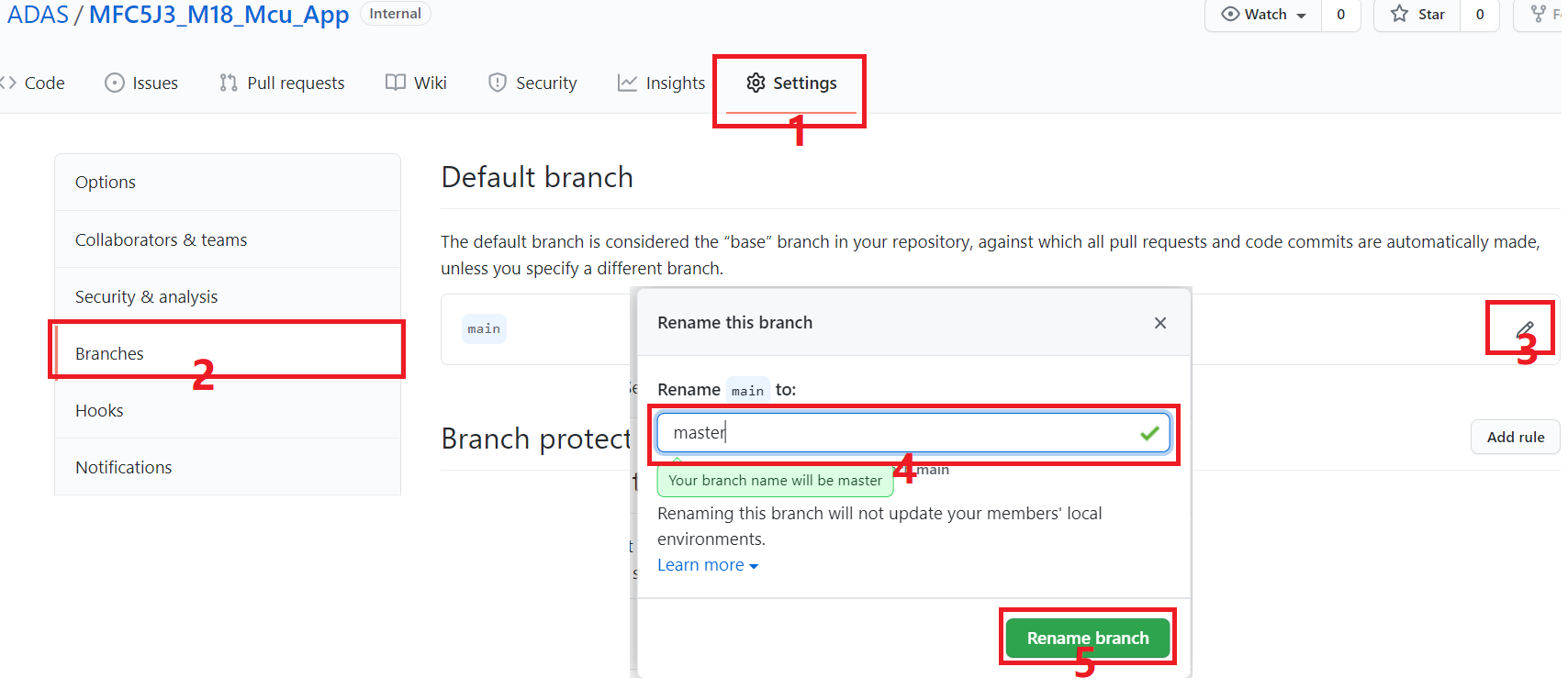
红框3：填写项目仓库的描述，主要描述仓库使用的项目以及跑在什么芯片上等信息；

红框4：选择仓库的类型，此项必须建议选择为Internal；

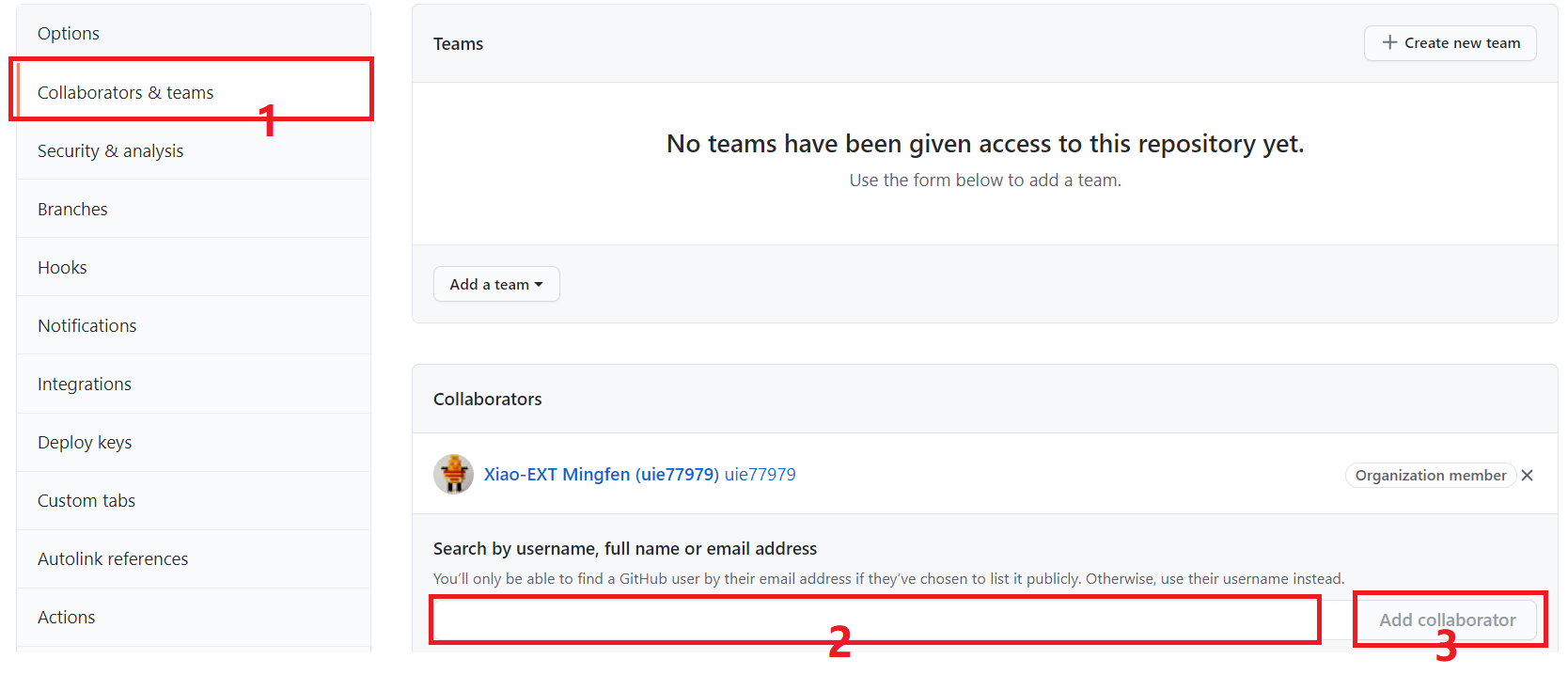
红框5：根据实际的情况选择是否添加README文件以及gitignore文件

红框6：确认以上信息后，点击“Create repository”即可完成仓库的创建；

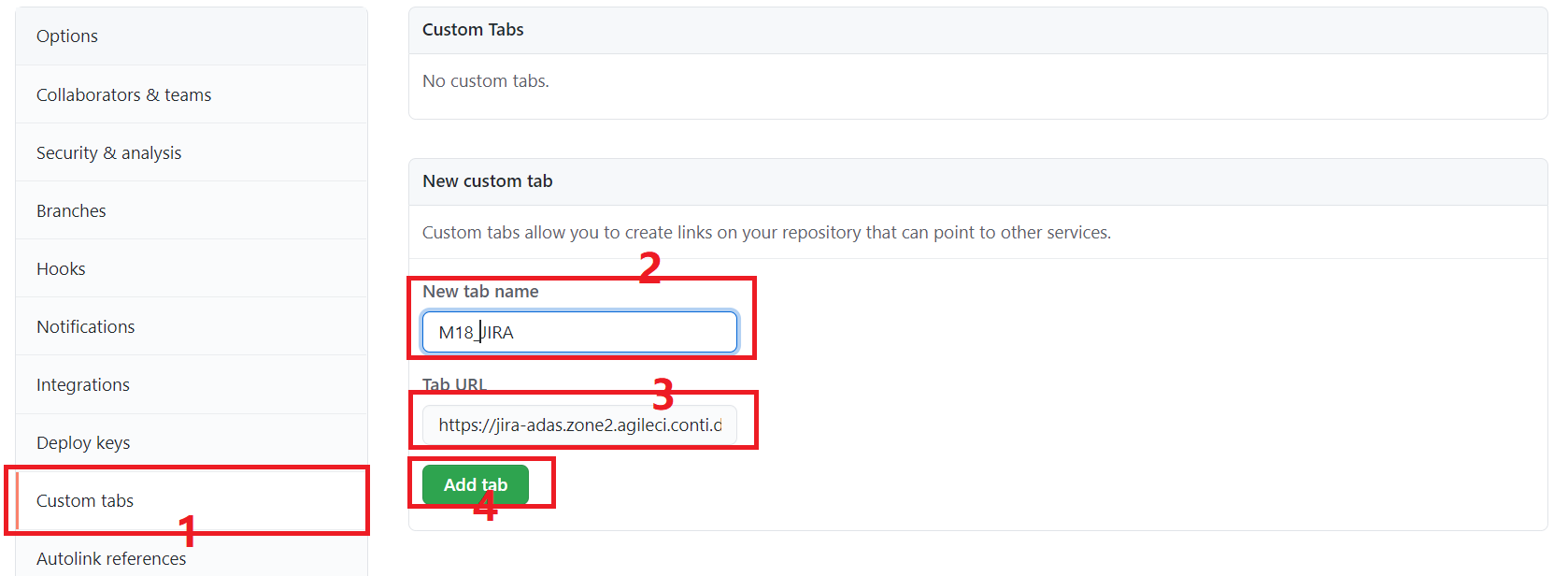
## 6.2 Change the main branch to master branch

如下图所示，根据数字的顺序将branch的名字从main改成master.

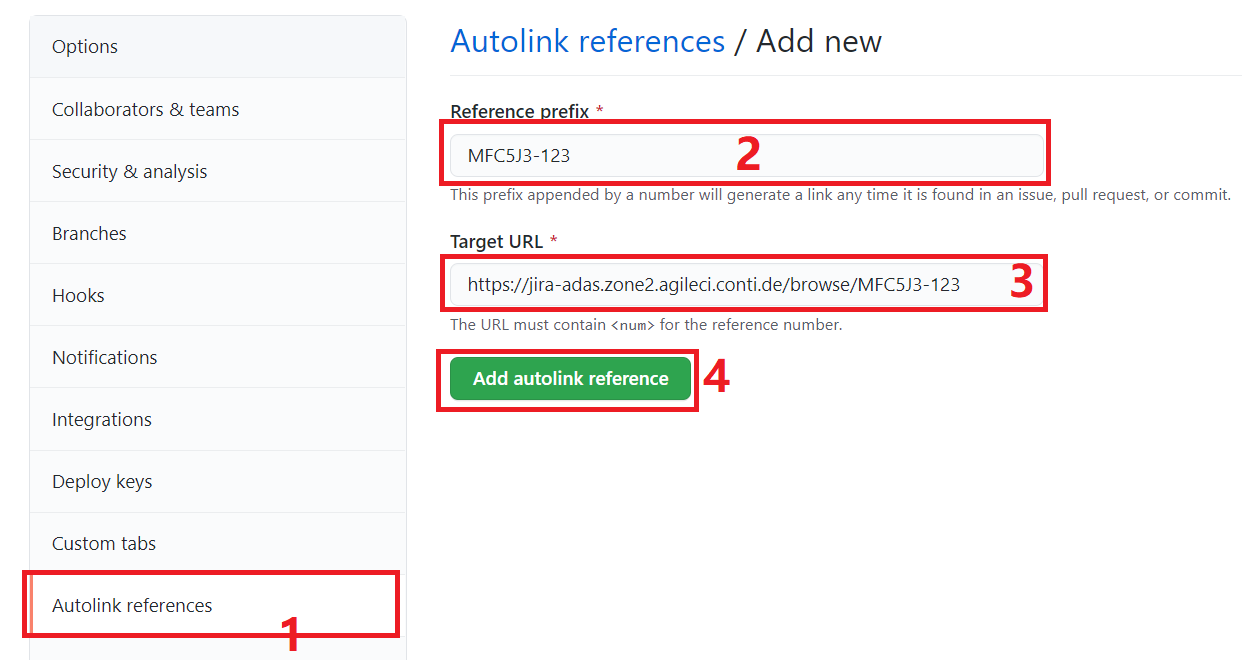
## 6.3 Add Collaborators

 如下图所示，根据数字的顺序来添加Collaborators.

## 6.4 Link exernal Links

 此步骤是为了将项目管理的主页，类似于JIRA或者Confluence和git 仓库进行管理，就可以在git上直接进入到JIRA或者Confluence。

## 6.5 Create Autolinks

 此步骤是为了将git commit和JIRA的task或者issue进行链接，可以通过如下图所示的顺序进行设置。

## 6.6 Clone Repository

使用如下命令来克隆新创建的仓库；

*git clone <url>*

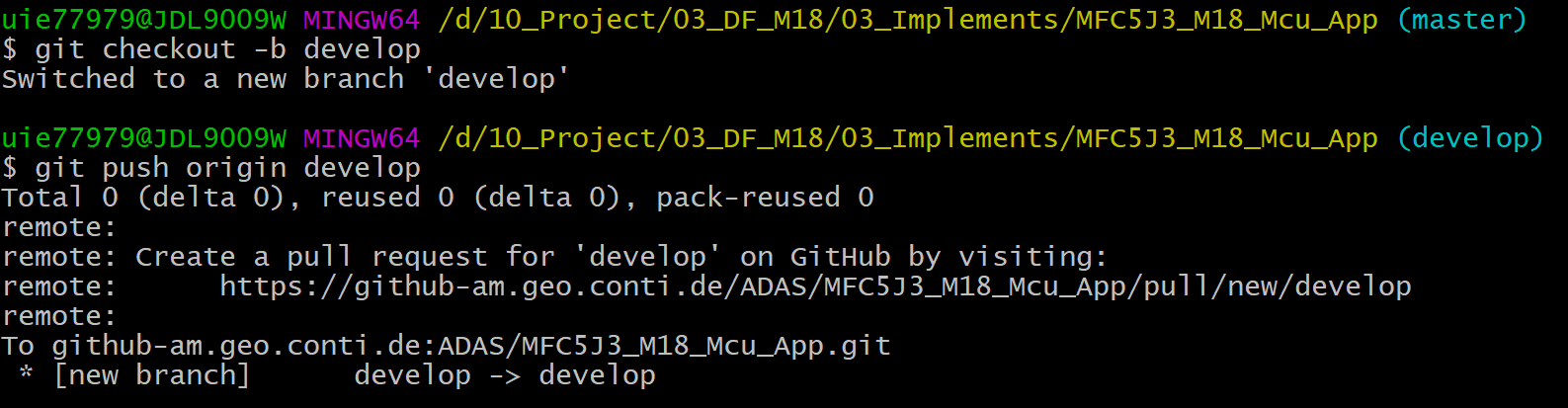
## 6.7 Create develop branch

如下图所示，通过以下命令来创建develop分支，并且push到git server.

创建并且切换到develop分支：

*git checkout -b develop*

将本地仓库同步到远程仓库：

 *git push origin develop*

## 6.8 Definition The Commit Message Format

为了规范代码的提交，此步骤的目的是定义代码提交记录的规则。

在仓库的根目录下创建一个.gitcommit的文件，将commit messag需要包含的一些信息填入到.gitcommit文件中。

在通过以下指令使其生效：

*git config commit.template .gitcommit*

**Notes:**

1. **此规则不是强制的，并且也不会触发检查，但是为了规范代码提交，建议还是按照规则进行操作；**
2. **Commit Message的规则请参考第九章节**

## 6.9 Create Init Version Code & Create Compilation[Option]

此步骤是将此项目的初始代码加入到仓库的文件夹中;

对于有base的项目，只需要将base项目的代码和配置文件拷贝到新的仓库中，并且验证好编译环境即可；

对于没有base的项目，尽量做到第一版的代码能够正常的编译；

## 6.10 Added Submodules[Option]

对于需要通过submodule方式拉取平台模块或者其他项目的子模块时，则需要按照如下操作来设置submodule，在设置submodule时，需要保证有submodules仓库的访问权限；

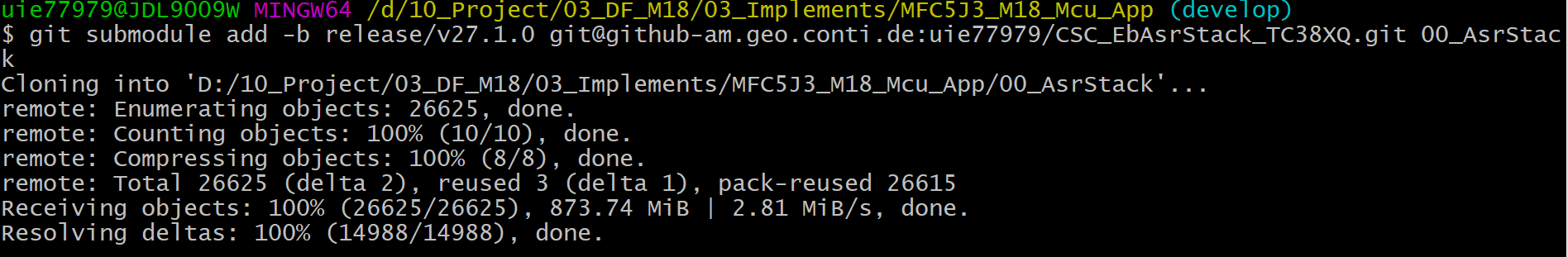
添加submodule的命令如下

*git submodule add [-b] [branch\_name] <url> [folder\_name]*

[-b]:可选参数，当拉取子仓库的非默认分支，则需要增加此参数

[branch\_name]:可选参数，当拉取子仓库的非默认分支，则需要增加此参数为拉取子仓库的分支名

[folder\_name]:可选参数，当需要将子仓库放置在特定的文件夹名称下时，需要启用才参数；如果不启用的话，则放置在与子仓库同名字的文件夹下；

如下图所示为添加完submodule后的截图：

**Notes:**

1. **需要在submodule放置的文件夹的根目录下开启git bash命令并且执行上述命令；**
2. **当拉取错误或者需要删除，则可以通过手动删除仓库根目录下.gitmodule文件的内容以及/.git/config和/.git/module文件夹下对应以submodule命名的文件夹；**
3. **如果分支设置错误，可以通过git config -f .gitmodules submodule.<submodule\_name>.branch <new\_branch>和git submodule update –remote 来进行设置和更新；**

## 6.11 Commit & Push

在这一步骤就通过以下命令进行commit和push

从工作空间添加到暂存区

*git add .*

从暂存区添加到本地仓库

*git commit -m “<commit\_message>”*

Or

*git commit*

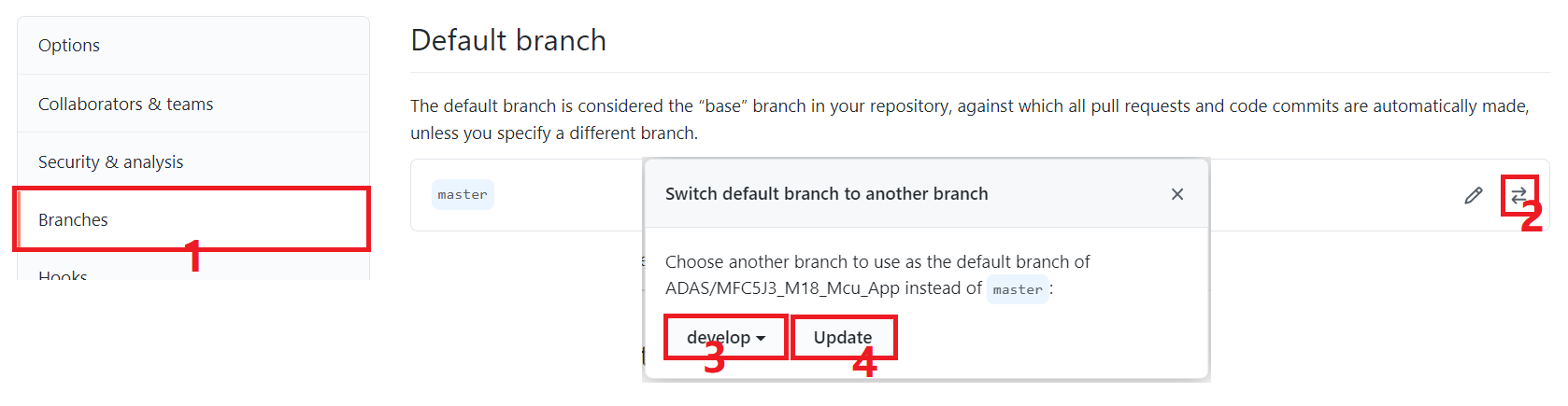
从本地仓库提交到远程仓库

git push origin develop:develop

**Notes:**

1. **在此步骤建议设置好.gitignore文件，此文件主要是用于设置哪些文件不上传到到git server，比如说编译的中间文件以及一些软件自动保存到备份文件就可以设置成不上传到git server。**

## 6.12 Set the develop branch to default branch

 如第一章节所描述，软件开发的默认分支为develop，所以需要将develop设置为默认分支，通过如下图所示的顺序即可完成设置。

# 7.CICT Buliding

TBD

# 8.Git Command Reference

如果想要更加深入的研究git bash的命令行，请参考：[Git - Reference (git-scm.com)](https://git-scm.com/docs)

# 9.Commit Message Rules

此章节是为了规范提交以及便于查看代码提交的状态，要求必须包含以下信息：

* **Version Number：**根据第二章节版本管理的规则，建议每一次提交都有一个唯一的版本号；
* **Change Description**：用于描述本次提交的更改点
* **Related Ticket(Task/Issue)：**用于填写本次提交修复的issue id或者task id。这样的目的是为了更好追踪项目管理系统中的ticket状态，另外commit也可以作为ticket的交付物。

如下图所示，为.gitcommit的内容：