# 软件包引入流程

# 前言

在这里，我们以软件cloc为例进行说明。

在openEuler社区中，基本的“组织”单元是SIG组，也就是special interest group。软件本身必须是要属于某一个SIG的。很多人在写完程序以后，都总会执行一下cloc这个命令，看一下今天又新增了多少行代码，以期获得一下码农与生俱来的成就感。显然cloc是一个开发类的工具，帮助码农统计代码行，幸运的是，拥有同样兴趣的人并不在少数，他们建立了一个[dev-utils](https://gitee.com/openeuler/community/tree/master/sig/dev-utils)的SIG组，我们可以将cloc这个软件归属于这个SIG组。openEuler所有的SIG组都在https://gitee.com/openeuler/community/tree/master/sig列出来，大家可以参考。

一般来说，增加一个软件包到openEuler中，需要如下的几个大步骤：

* + 1. 让系统为你的cloc软件包建立一个“仓”，也就是git仓。
    2. 上传制作cloc软件包所需要的“零件”。
    3. 将这个软件系统加入到openEuler的自动化编译系统中，由系统自动化构建出软件。

# 提交PR基本操作

提交PR是对项目上一次合入申请，为保证合入的质量，这个动作是需要小心谨慎的。在您提交PR之前，请先完成本地验证，以便在一定程度上保证在提交PR后的持续集成测试的通过。

Gitee上工作流的基本操作，请参考[工作流](https://gitee.com/openeuler/community/blob/master/zh/contributors/Gitee-workflow.md)。

## 提交PR

## 提交PR请求

为了更快的得到响应，您可以给PR添加标签，或提供更详细的信息，这里需要特别提示您的是：

* **关联Issue**：如果您提交的PR是针对某个Issue的，请您在提交的描述框内添加“#”字符，此时机器人会自动关联出当前存在的Issue，你可以通过此种方式快速的链接到关联的Issue。
* **标注优先级**：您可以在创建PR的时候，选择PR的优先级，或者在评论区通过/priority high给PR添加高优先级标签。
* **标注是解决bug的合入**：您可以在描述框通过输入/kind bug来标注该PR是合入解决问题的代码，以便于大家更快的回应您的PR请求。
* **标注所属sig**：为了方便查找，您也可以在描述框通过输入sig sig-name来标识该PR所属的sig。

## 分配评审人

您提交PR以后，社区机器人会自动给您分配评审人，您也可以指定评审人。制定评审人有两种方式，您可以在创建PR的时候，在右侧的下拉框中选择评审人；也可以在评论框中输入/assign @reviewer，从而把该PR分配给对应的人。如果您想把PR提交给项目的核心成员评审，以便于更快的获得批准，可以有两种方式获取到核心成员的信息：

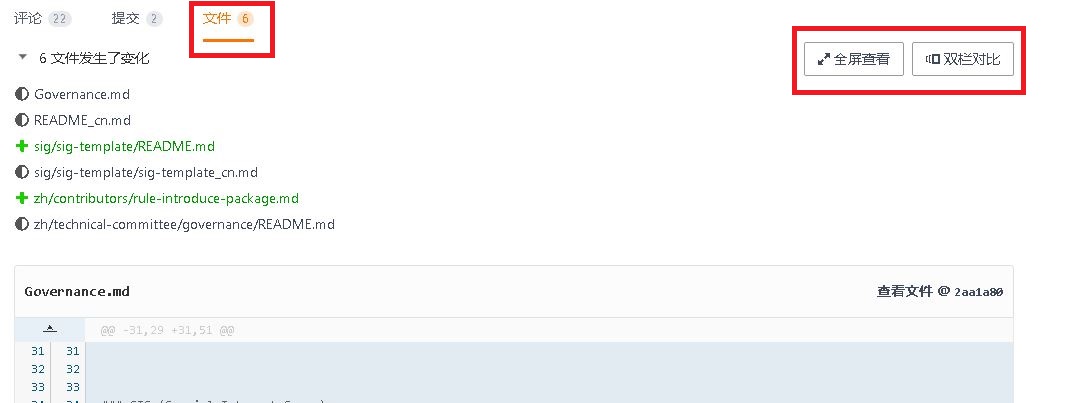
* **方式一**：该Repository的owners文件（该文件通常在该repository的根目录下）中查看，此文件保存的是所有该Repository的评审人列表。
* **方式二**：可以到该项目所属的SIG的首页内查看README.md文件，此文件会列出该sig的负责人，所有项目以及项目的负责人。

## 审核PR

* 如果审核人通过您的审核，会在评论区添加/lgtm以及/approve，以表示对本次PR提交的认同
* 审核人可以在评论区发表意见，也可以在审核文件的时候，在发现问题处添加审核意见。无论哪种方式，都会在评论区显示出来。区别是，后者的评论会显示出“代码评论”，您可以通过“详情”查看评论具体指向的出处。



* 为了表示对评审人意见的尊重，如果对意见有异议，请您通过回复该意见来说明原因；如果接纳评审人意见，也请您做出简单的回应，便于确认后继的提交是否已按照所有接纳意见完成修改。
* 您也可以点击“文件”，直接在文件区查看具体的代码评审意见和对比结果。在此处，您也可以直接回复评审意见。



* 如果项目组的核心成员拒接了您的PR，会给出拒接意见。请不要气馁，您可以根据意见调整后继续和项目组成员沟通，相信在持之以恒的沟通调整后，您的能力和意见都会得到大家的认同。
* 请注意，在使用/approve前至少要有一个/lgtm。

# 建仓

## 3.1 SIG组相关

openEuler社区按照不同的SIG来组织，以便于更好的管理和改善工作流程。SIG组是开放的，欢迎任何人加入并参与贡献。SIG都是针对特定的一个或多个技术主题而成立的。SIG内的成员推动交付成果输出，并争取让交付成果成为openEuler社区发行的一部分。SIG的核心成员主导SIG的治理，开发者可以在贡献的同时积累经验和提升影响力。

每一个SIG在Gitee上都会拥有一个或多个项目，这些项目会拥有一个或多个Repository。SIG的交付成果会保存在这些Repository内。可以在SIG对应的Repository内提交Issue、针对特定问题参与讨论，提交和解决问题，参与评审等。也可以通过邮件列表、IRC或视频会议和SIG内的成员进行交流。

引入软件包时，需要找到这个软件包所归属的SIG组，如果您知道感兴趣的项目名称，可以在openEuler的Repository列表下进行模糊搜索，从而快速定位到对应项目的首页地址。通常情况下，在该项目首页地址的README.md文件中，可以找到该项目所属的SIG信息、交流方式、成员和联系方式等。

每个SIG内的项目使用的编码语言、开发环境、编码约定等都可能存在差异的。如果想了解并参与到编码类贡献，可以先找到该项目给开发者提供的贡献者指南——这个指南一般是在该SIG的首页地址内，以CONTRIBUTING.md文件的形式提供，或者就直接在该项目的README.md内。

除了这些文件外，SIG可能还会提供其他指南信息。这些信息位于SIG或其项目的特定社区目录中。如果未找到相关信息，或者对相关信息有疑问，可以在该SIG内提交Issue，或者把问题或疑问发送到该项目所属SIG的邮件列表。如果您认为长时间没有得到回应，可以向community@openeuler.org求助。

接下来建仓其实就是提交PR来修改三个文件。以当前软件包为例，其所在sig组为dev-utils， 需要修改的文件如下：[README.md](https://gitee.com/openeuler/community/blob/master/sig/dev-utils/README.md)，[sigs.yaml](https://gitee.com/openeuler/community/blob/master/sig/sigs.yaml)，[src-openeuler.yaml](%09https:/gitee.com/openeuler/community/blob/master/repository/src-openeuler.yaml)。

修改第一个文件README.md将你要加入的cloc软件的名字和地址放上去。

修改sigs.yaml文件，将cloc软件增加到dev-utils这个SIG分组下面。

修改src-openeuler.yaml将cloc增加到src-openeuler里。

## 3.2 修改步骤

通过修改gitee中的openeuler/community仓库实现新增软件包的具体操作步骤如下：

1. Fork到自己的仓库

点击进入[community仓库](https://gitee.com/openeuler/community.git)，将其fork到自己的仓库。

1. clone到本地

将fork完成的community仓库clone到本地。

git clone https://gitee.com/"$username"/community.git

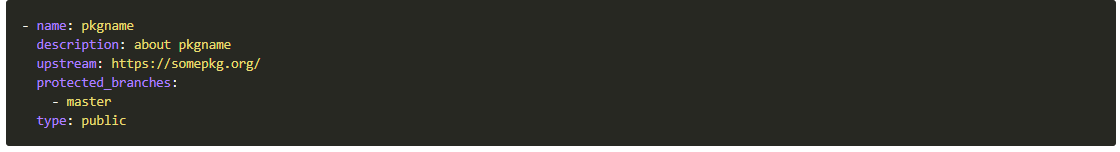
1. 修改community仓库

以在gitee新增zip软件包，并且同步在openEuler:Factory工程项目下添加该包为例。

1. cd community/sig；
2. 明确软件包所属的sig组，如zip属于Base-service组；
3. 修改所属sig文件夹下的内容，如项目清单等；
4. 修改sig/sigs.yaml，将要新增的软件包以"- src-openeuler/zip"的形式添加到对应的sig组列表下；以zip为例：



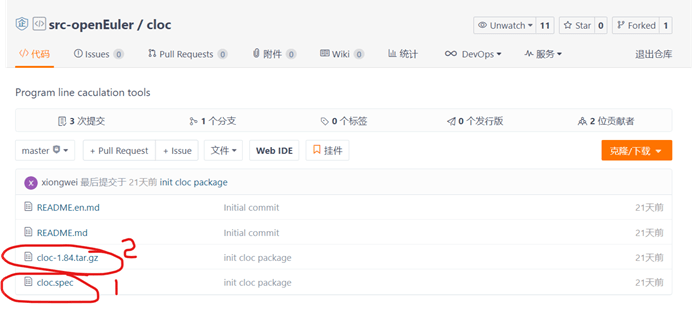
1. openeuler.yaml；其他社区引入包: src-openeuler.yaml。



提交PR（PR合入后将会在gitee建立同名仓库。当申请的结果被批准以后，你所需要的“仓”就会被系统自动建立起来，查看地址是[src-openeuler](https://gitee.com/src-openeuler)。对于cloc来说，它的代码仓的位置在<https://gitee.com/src-openeuler/cloc>，这是PR被批准以后由系统自动为我建立的。同时在OBS上建立同名仓库，可在[OBS网站](https://build.openeuler.org/project/show/openEuler:Factory)查看； 剩下的时间，你就可以开始真正上传制作cloc软件的原材料了。

# 上传软件包

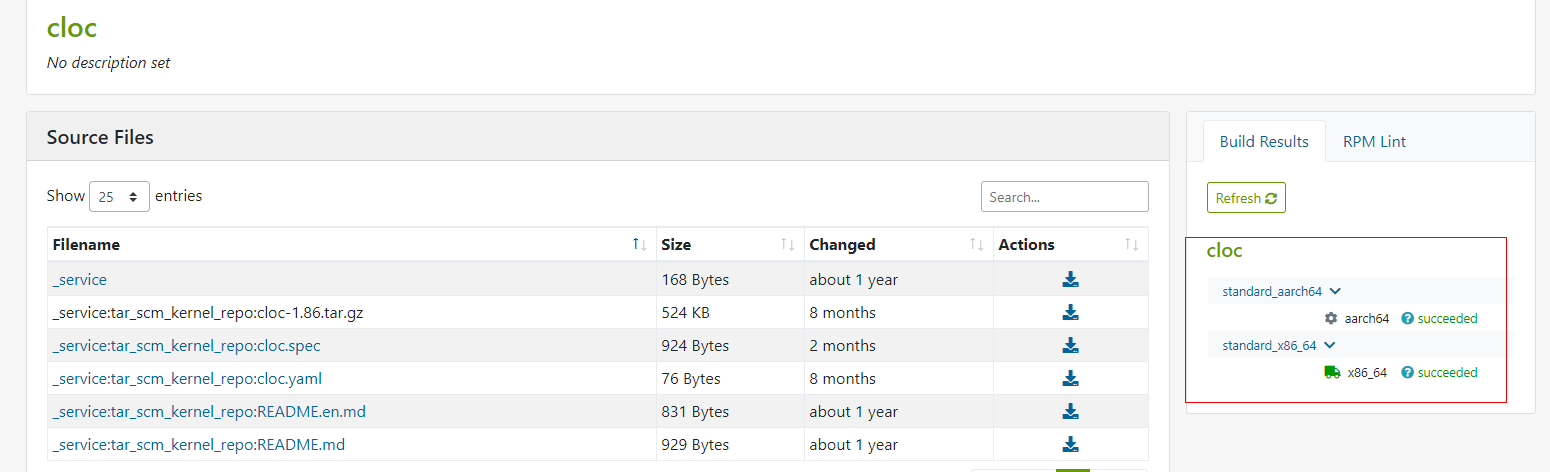
一般来说，一个软件只需要上传两个“原材料”就足够制作一个软件包了。如下图所示：



* **第一个材料**：首先要上传这个软件包的SPEC文件，也就是告诉构建系统如何编译，制作cloc这个软件包。
* **第二个材料**：cloc的源代码压缩包。
* **其它零件**：如果SPEC中需要有Patch，那么也需要把相关的Patch文件上传到仓中。

上传完毕一个软件包所需要的原材料， 这里我们需要在OBS上的Factory工程中确认软件包是否构建成功。在Factory工程下软件包，会分别显示在aarch64与x86环境中的构建结果。

以软件cloc为例，如图所示即为构建成功：



如果构建不成功，则状态为failed，unresolveable，broken等状态， 需要查看编译日志中的具体错误信息，定位出错原因。

# 加入构建系统

当我们在Factory工程上面确认好了软件包已经构建成功，当前的构建工程还是在Factory上面，我们需要将它移动到最终归属的构建工程。

openEuler现在使用OBS作为构建工具系统，因此，引入的软件包需要加入到OBS构建系统中 。加入到构建系统中，就意味着你的软件可以被系统自动编译，自动生成RPM包，继而在后续的openEuler发行版本中出现。

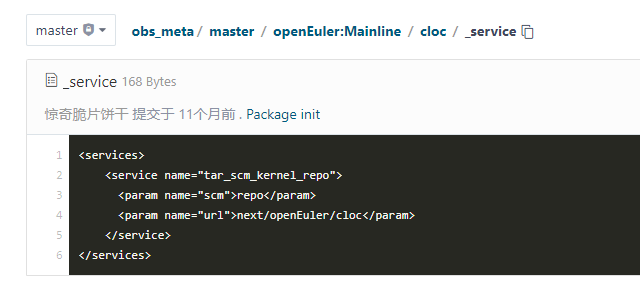
OBS使用源服务获取源码，使用源服务，就需要\_service文件。新增软件包时，openEuler自动使用了tar\_scm\_kernel\_repo插件拉取代码。可通过修改src-openeuler/obs\_meta仓库下对应软件包的\_service文件实现自定义更改，具体步骤如下：

1. 点击进入obs\_meta仓库，将其fork到自己的仓库。
2. 将fork完成的obs\_meta仓库clone到本地。

git clone https://gitee.com/"$username"/obs\_meta.git

1. 修改obs\_meta仓库。
2. cd obs\_meta/master/openEuler:Factory/
3. vim "package\_name"/\_service根据自己的需求修改\_service 文件。（package\_name是需要自定义\_service文件的软件包名称）
4. 提交PR，PR合入后将会把修改后的\_service文件同步到该仓库。

\_service文件是一个xml格式的文件，主要功能是拉取代码库上的代码，对代码库的代码再组织打包进行编译前的源码准备。以软件包cloc为例：



* <service name="tar\_scm\_kernel\_repo">这行指定插件名称为tar\_scm\_kernel\_repo。
* <param name="scm">repo </param>这行指定插件操作的代码管理工具为repo。
* <param name="url">next/openEuler/cloc</param>这行指定repo源码的地址。

在本次赛题中，我们需要将代码提交到mainline工程下，因此需要将obs\_meta/master/openEuler:Factory/下面的\_service文件移除，重新添加到obs\_meta/master/openEuler:MainLine /目录下，然后提交PR，这样，构建工程就从Factory转移到Mainline下面了。