

# Mining Challenge 介绍

今年的 mining challenge 挑战是关于全局软件供应链（Global Software Supply Chain, GSSC）数据和被称为代码世界（World of Code, WoC）的基础设施，该基础设施从几乎所有公共版本控制系统中收集、策划和构建数据的交叉引用结构。

GSSC 的数据版本 U 是根据 GitHub、GitLab、Bitbucket 和在 2021 年 10 月 20 日至 30 日期间确定的数十个其他 forge 的更新和新 repository 收集的，这些 forge 在 11月28日之前(?)检索到的 git 对象。173M git repository 包含超过 3.1B commit、12.6B tree 和 12.5B blobs。整个数据集占用超过250TB，mining challenge 的参与者应该使用集群进行分析或进行预过滤，以对数据集的一个子集进行研究。

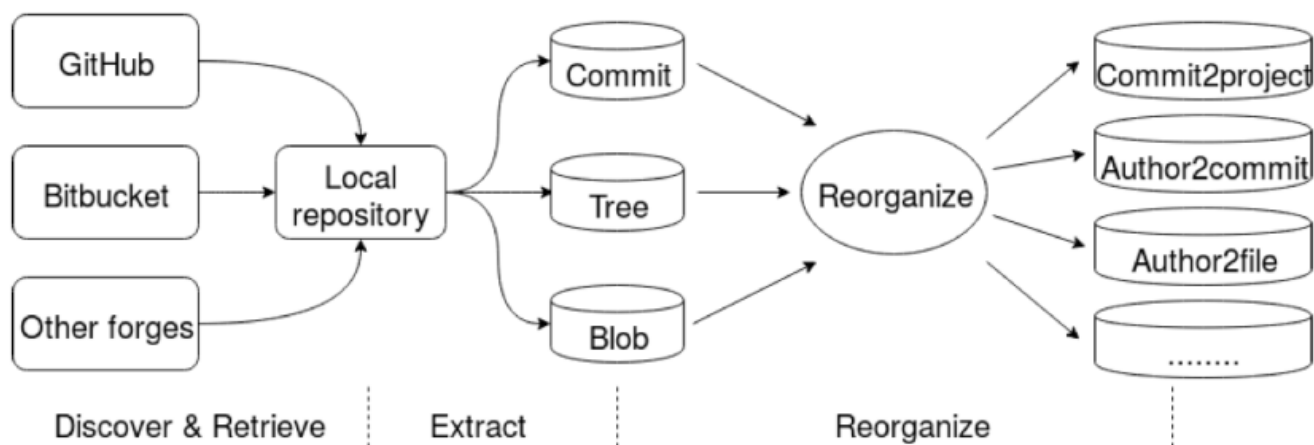
## WoC 介绍

WoC 的项目论文为 [《World of Code: Enabling a Research Workflow for Mining and Analyzing the Universe of Open Source VCS data》](#)。

总结一下这个论文的整体逻辑：

1. 现有的 OSS 生态系统的研究方法多是对少数 repository 的研究，对整体的研究也缺少对供应链/代码联系/文件作者联系的研究，所以 WoC 应运而生：
  1. 在整个 FLOSS（Free/Libre Open Source Software）生态系统中创建一个非常大且频繁更新的版本控制数据集合，命名为代码世界（WoC），可以完全交叉引用FLOSS系统的作者、项目、提交、blob、依赖关系和历史；
  2. 提供有效的更正、扩充、查询和分析这些数据的能力。当前的 WoC 能够实现每月更新一次，每次更新超过24B git objects。
2. 整体的 WoC 使用一种类似微服务的架构思想，为每个特定任务使用最简单的技术和组件；这种类似微服务架构的松散耦合组件的设计和性能可以独立评估，每个服务都可以利用最适合其需求的数据库，计算密集型组件非常可移植，以确保它们在任何高性能平台上运行；
3. 现有的技术方法只支持 WoC 的子集，而 WoC 的设计让它可以保持完整的同时使其更新和检索更加高效；

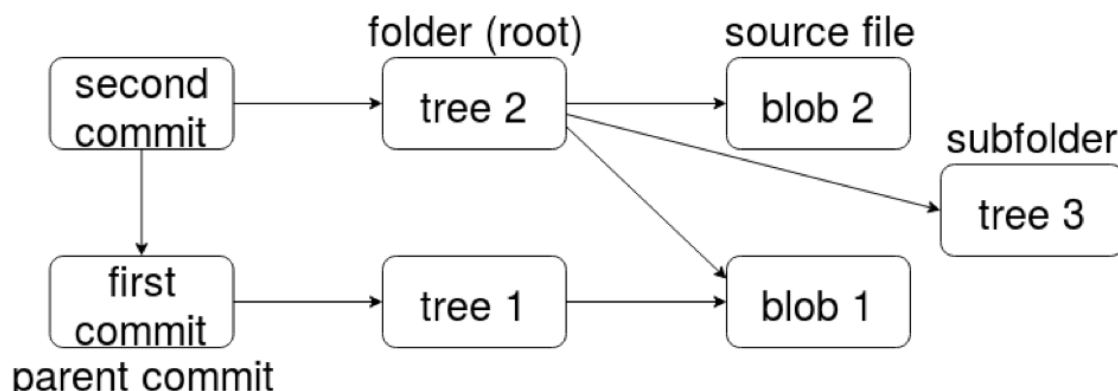
#### 4. WoC 的原形框架如下：



1. 将 Search API、Search Engine、Keyword Search 这三种以及其它一些特定的方法一起来保证所发现的 repository 是接近完整的；之后使用多线程以及子集等优化手段来提升获取代码以及更新的速度；

#### 2. 数据模型：

包含以下commit、tree、blob、tag四个对象



3. 数据存储：由于很多 git repository 会出现极多的重复文件，所以根据四种 git 对象将数据库分为四部分，每一部分都包含一个缓存和一个主体内容：cache 用于决定一个对象是否在数据库中、是否必要被解压缩以及是否都需要被 clone（如果他的最新 commit 已经在我们的数据库中，就没有必要clone。其中：

1. Cache 数据库是 key-value 数据库，对象 SHA1 压缩编码的前20位作为 key 而对象在对应的 value 数据库中的位置作为 value；
2. Value 数据库由一个 offset 查找表组成，该表提供 git 对象压缩后的偏移量和大小；

3. 为了加快搜索效率，于是为 **commit** 和 **tree** 分别多构建了一个 **key-value** 表，其中 **key** 为 **SHA1** 而 **value** 为所对应的文件；

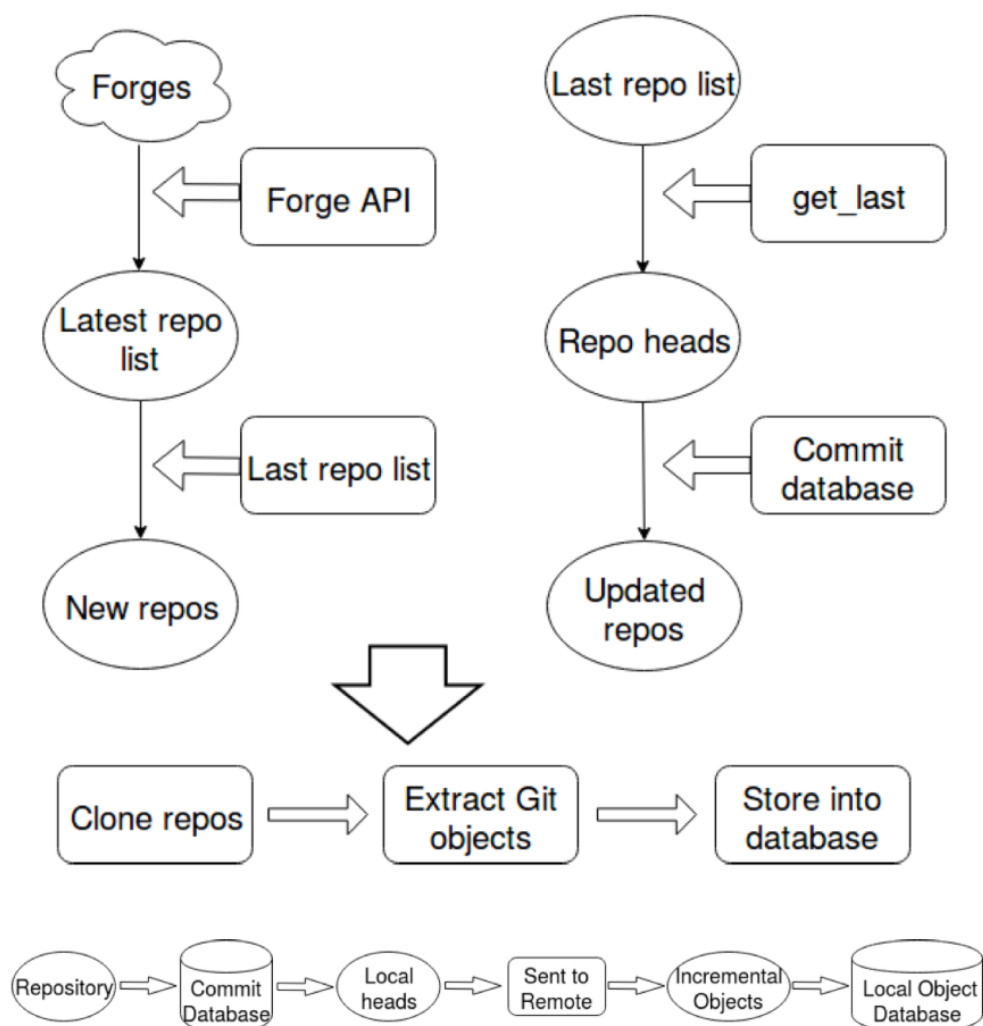
为了方便存储以及提高检索效率，将数据库并行化为 **128** 个切片数据库，于是总共有  $128 \times (4 + 4 + 2)$  个数据库，单个大小在 **20 MB ~ 0.5 TB** 之间；为了增强鲁棒性和可靠性并避免永久性数据丢失，使用了 **2N + 1** 策略，一共维护了数据库的三个副本；在流程的每个阶段验证数据完整性的技术是必要的。因此，我们包括许多测试，以确保只有有效的数据才能传播到下一阶段；

#### 4. 数据更新：

包含两个方式：

- 识别新的存储库、克隆和提取Git对象
- 识别更新的存储库并仅检索新添加的Git对象

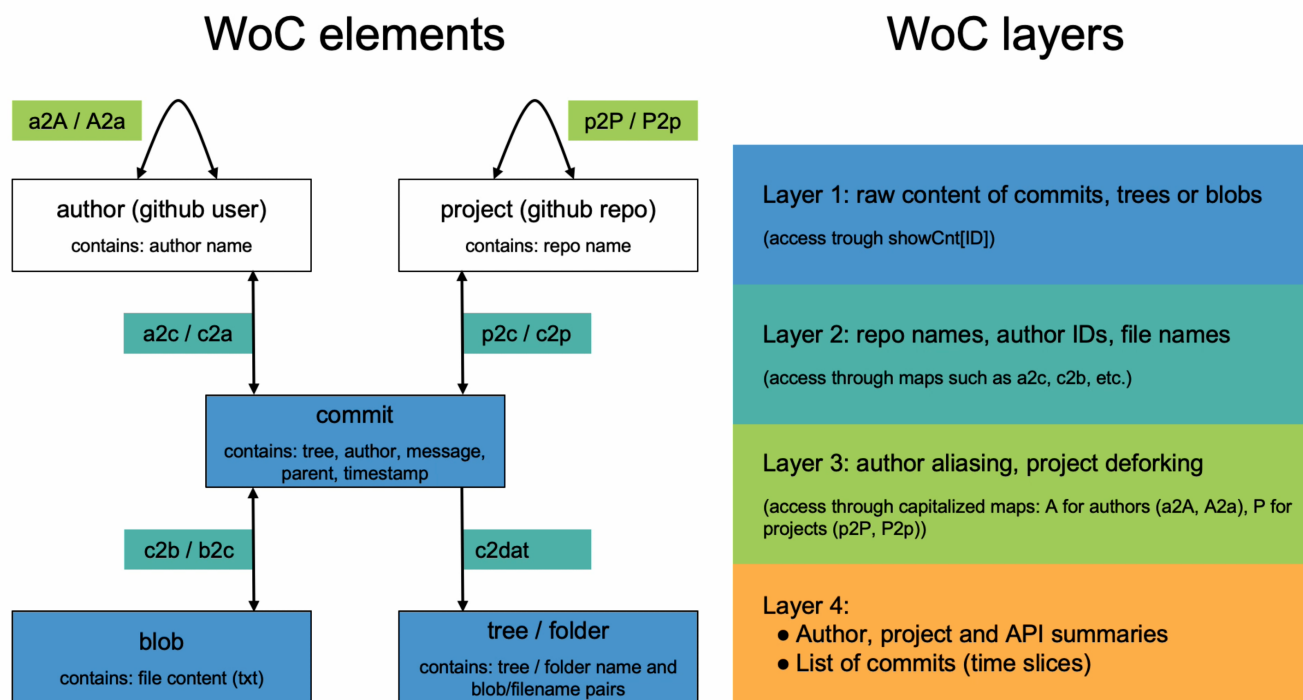
#### Identify new repository      Identify updated repository



5. 每次更新只需要在文件数据库中添加新的文件即可；

5. 分析数据库：现有的主流数据库难以支持这一量级的数据存储，所以使用 TokyoCabinet 进行存储，可以提供十倍快于 MongoDB 的查询速度

1. maps：

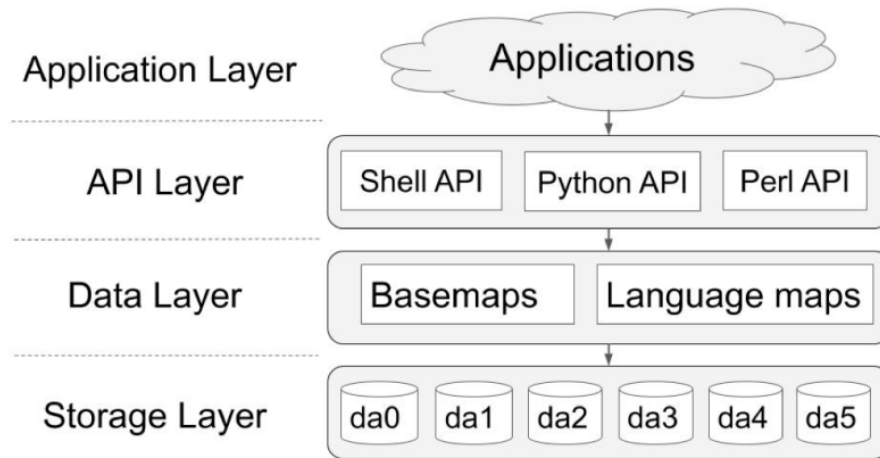


其中：

**Layer 2 (Cross-reference Layer)** 是根据 git 对象的内容或与其他对象的交互构建出来的，包括 commit 和项目之间的正向和向后链接、commit 创建的 blob、commit 作者、时间等。这些 map 代表了各种软件供应链的边；

**Layer 3 (Curation Layer)** 去重并通过解析文件内容以识别17种编程语言中的依赖项来计算blob的其他属性，以及使用领先的商业服务（Namsor）推断开发人员的性别。  
**a2A/A2a** 表示识别某一个开发者的多个账号/账号名；**p2P/P2p**表示识别一个仓库的多个 fork 、复制和多个仓库名；

**Layer 4 (Summary Layer)** 数据集专注于三个实体：项目、开发人员和API，并在 mongoDB 中总结每个实体，以简化查询。例如，每个项目、作者和API都有活动日期范围、每月活动、核心团队（开发人员负责80%的提交）以及预先计算和存储的许多其他属性。该摘要级别的目的是对项目、作者和API进行自然实验和代表性采样。



6. 研究流程结构:

7. API: 提供了三个 API

1. Shell API:

– Checking the content of a Git object given a SHA:

```

1  # (on da3) e.g., show a commit SHA's content:
2  echo e4af89166a17785c1d741b8b1d5775f3223f510f | showCnt commit
3  # Output Formatting:
4  # Commit SHA;Tree SHA;Parent Commit SHA;Author;Committer;Author Time;
   Commit Time
5  e4af89166a17785c1d741b8b1d5775f3223f510f;
   f1b66dcca490b5c4455af319bc961a34f69c72c2;
   c19ff598808b181f1ab2383ff0214520cb3ec659;Audris Mockus <audris@utk.
   edu>;Audris Mockus <audris@utk.edu>;1410029988 -0400;1410029988 -0400
  
```

– Given an object, check its related objects:

```

1  # (on da3) e.g., show the names of the projects associated with a given
   commit SHA:
2  # "getValue" command takes a database name as an argument and keys
   presented as standard input and produces key-value pairs as output.
3  echo e4af89166a17785c1d741b8b1d5775f3223f510f | getValue /da0_data/
   basemaps/c2pFullP
4  # Output Formatting: Commit SHA;ProjectNames
5  e4af89166a17785c1d741b8b1d5775f3223f510f;W4D3_news;chumekaboom_news;
   fdac15_news;fdac_syllabus;igorwiese_syllabus;jaredmichaelsmith_news;
   jking018_news;milanjpatel_news;rroper1_news;tapjdey_news;
   taurytang_syllabus;tennisjohn21_news
  
```

## 2. Python API:

1. Author('...') - initialized with a combination of name and email, e.g. "Albert Krawczyk <pro-logic@optusnet.com.au>"
  - .commit\_shas/commits - all commits by this author
  - .project\_names - all projects this author has committed to
2. Blob('...') - initialized with SHA of blob
  - .commit\_shas/commits - commits creating or modifying (but not removing) this blob
3. Commit('...') - initialized with SHA of commit
  - .blob\_shas/blobs - all blobs in the commit
  - .child\_shas/children - the commit that follows this commit
  - .changed\_file\_names/files\_changed
  - .parent\_shas/parents - the commit that this commit follows
  - .project\_names/projects - projects this commit appears in
4. Commit\_info('...') - initialized like Commit()
  - .head
  - .time\_author - the commit time and its author
5. File('...') - initialized with a path, starting from a commit root tree. This represents a filename, regardless of content or repository; e.g. File(".gitignore") represents all .gitignore files in all repositories.
  - .commit\_shas/commits - All commits that include a file with this name
6. Project('...') - initialized with project name/URI
  - .author\_names - all author names in this project
  - .commit\_shas/commits - all commits in this project

## 3. Perl API: 略

8. 针对 dependency，构建了 12 种针对不同语言的 map：C、C#、Java、JavaScript、Python、R、Rust、Go、Swift、Scala 和 Fortran。存储形式为 ``

```
commit;repository_name;timestamp;author;blob;module1;module2;...
```

## 教程

1. 在 WoC 集群（<https://forms.gle/vixXjmjocBzX3BBB6>）上注册帐户。创建帐户的周转时间大概为一个工作日；
2. 本机操作参考 <https://github.com/woc-hack/tutorial>。


## 挑战

挑战是开放式的：参与者可以选择他们认为最有趣的研究问题。我们的建议是考虑不以特定项目或一组项目为中心的问题，而是利用WoC的完整性、策划和交叉引用能力。下面有一些想

法。

1. 软件供应链是许多热门问题的基础，如漏洞、代码来源、材料清单等。针对这些，WoC 旨在衡量三种类型的软件供应链（代码依赖性、代码复制和作者-代码联系）。这三种链均有各自独特的风险和优点；
2. 研究那些需要构建、采样或分析源代码、API、人员和项目的全局网络的问题，并按时间或内容过滤子集。例如，你可以调查特定代码来自哪里，何时引入特定API，哪些项目或人员使用它，以及特定开发人员从事哪些项目？
3. 确定全局背景。传统的 MSR 分析倾向于专注于一组特定的项目，因为只需要获得特定于项目的数据。通常会丢失此类数据集中元素的关键上下文，例如开发人员的行动、与代码相关的活动以及特定项目集之外的API的使用。WoC 允许恢复和量化此类全局环境；
4. 避免方便取样。Layer 4提供了项目、API 和开发人员的详细 summary，可以基于此进行多种更加自然的实验。
5. 在全球范围内利用策展。WoC 中的 Curation Layer 通过根据共享 commit 来合并别名化作者 ID 和分叉项目来解决常见的MSR问题；
6. 链接/增强 WoC 数据集本身。鼓励参与者将 WoC 与其他数据相结合，或加入收集和链接外部数据的代码，以及关于如何将这些数据永久集成到WoC的建议；
7. 与静态数据库不同，WoC可以重建整个开源软件的过去状态。许多当代质量、准备时间、努力、任务预测模型需要重建过去状态的能力，以避免普遍存在的“数据泄露”问题；
8. 该数据集为开发人员为什么以及如何决定重用现有软件、他们选择哪种类型的供应链（技术依赖性、基于复制或重用想法）等问题提供了现成的数据；
9. 重建 OSS 过去状态的能力允许找到“如何生成广泛使用的框架或库？”等问题的答案。或“如何降低上游项目变化的风险，以及如何降低下游项目的风险。”

## 时间线

Important Dates		 AoE (UTC-12h)
Fri 3 Feb 2023	Abstract Deadline	
Sun 5 Feb 2023	Paper Deadline	
Tue 21 Feb 2023	Author Notification	
Mon 13 Mar 2023	Camera Ready Deadline	
Thu 27 Oct 2022 02:00	Live Webinar & Kickoff Session (2hrs)	new
Mon 15 Aug 2022	Call for Challenge Papers Published	
Thu 28 Jul 2022	Mining Challenge Proposals Notification	
Mon 18 Jul 2022	Deadline for Mining Challenge Proposals	