

# Lab0 编译原理实验环境搭建

---

本文档用于指导编译原理实验环境搭建。

本课程使用的核心软件版本为：

- Ubuntu 20.04
- LLVM 10.0.1
- Flex 2.6.4
- Bison 3.5.1

首先，搭建实验环境，安装后续实验必要的软件。

此外，后续将使用 GitLab 提交代码，需要配置 Git。

不接受因软件版本不一致而导致的任何迟交、错交、误交理由。

## 搭建实验环境

---

推荐使用 [Docker 镜像](#) 方式搭建实验环境，也可以在 Ubuntu 下用 [包管理器](#) 安装，自行编译安装也是可行的。

推荐第一种方式，因为希冀平台上是使用 Docker 镜像进行测试的，使用其他方式可能会因为软件版本出现问题。

如果选择 Docker 的话建议直接在实体机上安装 Docker，因为 Docker 相当于一个虚拟机。当然装在虚拟机里也并非不可。

### 使用 Docker 镜像搭建（方式一，推荐）

Docker 是流行的容器管理器，且借助于虚拟化技术，可以在多种操作系统下运行，十分方便。

本课程提供的 Docker 镜像主要用于交互式使用，类似于轻量级的虚拟机。本文档只提供与本课程相关的 Docker 基础配置，对 Docker 感兴趣的同学可以联系助教或者自行搜索相关资料。

#### 概念

最重要的两个概念是“容器”和“镜像”。

- 容器：类比为“虚拟机”。（也可类比作“进程”。）
- 镜像：类比为“装某个虚拟机用的 ISO 文件”。（也可类比作“可执行文件”。）

技术上说，容器和虚拟机有不少差别，不过对我们的实验来说了解到这里就足够了。

#### 安装并启动 Docker

Docker 的安装可参考官方文档 [Get Docker](#)。

#### Windows

下载并安装 [Docker Desktop for Windows](#)

如果选择 WSL2 作为后端，启动 Docker Desktop 可能会提示缺少 [WSL2](#)，从此链接下载安装即可。

## macOS

下载并安装 [Docker Desktop for Mac](#)。

注：由于 Macbook 系列电脑 2020 年后使用 arm/v8 架构的芯片，直接使用我们的实验镜像会有警告，助教团队已经测试可正常使用该镜像完成实验 lab1-4。详细请移步 [镜像与容器配置](#)

## Linux (Ubuntu 20)

用包管理器安装 Docker 可能比官方文档方便一些。

```
sudo apt update
sudo apt install docker.io
sudo usermod -aG docker $USER # 使当前用户不需要 sudo 就可以使用 docker 命令
sudo chmod a+rw /var/run/docker.sock
sudo systemctl restart docker
```

### 验证是否安装成功

使用 `docker run hello-world` 命令验证 Docker 是否安装成功，如果出现以下内容则说明安装成功。

```
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

如果不成功，请确认 Docker daemon 是否正在运行。

- Windows, macOS：打开 Docker Desktop，确认看到提示 Docker Desktop is running。
- Linux：参考相应发行版的文档。以 systemd 为例，参考文档 [Control Docker with systemd](#)。

## Docker 常用命令

```
docker version          # 查看 Docker 版本
docker run hello-world  # 运行 hello-world 容器
docker ps -a            # 查看所有容器
docker images            # 查看所有镜像（也可以用 docker image ls）
docker pull ksqsf/llvm  # 拉取 ksqsf/llvm 镜像
docker create --name compiler-labs ksqsf/llvm # 创建一个名为 compiler-labs 的容器
docker start -ai compiler-labs # 启动 compiler-labs 容器，并进入交互模式
docker rm compiler-labs # 删除 compiler-labs 容器
```

## 具体用法

### 下载镜像

```
docker pull ksqsf/llvm
```

注：校内网络可用 [USTC 镜像](#) 加速。

下载后，可用 `docker image ls` 命令确认。

### 创建与查看容器

选择共享文件夹，将下列命令中的“绝对路径”替换为此文件夹的绝对路径

```
docker create --name compiler-labs -v "绝对路径:/labs" -it ksqsf/llvm
```

例如

```
# Windows
docker create --name compiler-labs -v "D:\Compiler\labs:/labs" -it ksqsf/llvm
# Linux
docker create --name compiler-labs -v "$(pwd):/labs" -it ksqsf/llvm
```

上述命令将创建该容器。如果一切顺利，你将可以看到一串十六进制数（容器 ID）打印出来。此时，使用 `docker ps -a` 可以看到出现了一个新的容器。（`ps` 默认只输出正在运行的容器，因此使用 `-a` 强制输出所有容器。）

如果输错了路径，用 `docker rm compiler-labs` 删除再重新 `docker create`

注 1：使用 Windows 系统的同学如果发现报错 `Error response from daemon: error creating overlay mount to ...` 请核查 Docker Desktop 是否安装完整，同时请注意 Windows 和 Linux 路径分隔符的区别。

注 2：使用 macOS 系统且为 M1 或 M2 芯片的同学会得到一个警告：

```
WARNING: The requested image's platform (linux/amd64) does not match the
detected host platform (linux/arm64/v8) and no specific platform was
requested
```

这是因为原始实验环境镜像没有 buildx 至 arm/v8 架构，但这不影响后续实验。助教团队确认过 M1 芯片的 Macbook 可以使用该镜像正常完成实验。如果你想忽略该警告可添加 `--platform` 参数，使用如下命令：

```
docker create --name compiler-labs -v "绝对路径:/labs" -it --platform
linux/amd64 ksqsf/llvm
```

参数的含义是：

- `--name` 赋予该容器一个名字。
- `-v $DIR_PATH:/labs` 将创建一个“共享文件夹”，路径在容器的 `/labs`。可以通过这个目录来与宿主机交换文件。
- `-i` 表示创建交互式容器。
- `-t` 表示分配伪 TTY。

## 进入容器环境

```
docker start -ai compiler-labs
```

你将可以看到一个 bash 命令行，表示已经在 Ubuntu 环境中了。退出 shell 后，可以再次使用该命令回到容器环境中，并且对环境作出的修改仍将保留着（除非你删除了这个容器，或者不小心进入了另一个容器，或者在创建容器时错误地使用了 `--rm` 参数）。

如果报错 `Error response from daemon: dial unix docker.raw.sock: connect: connection refused`，请检查是否启动了 Docker 引擎。

参数说明：

- `compiler-labs` 是创建时的 `--name` 参数。
- `-a` 表示 attach。
- `-i` 表示 interactive。

用于实验的 LLVM Docker 镜像提供以下内容：

- LLVM 10.0.1
  - `/llvm`: 头文件和已编译安装好的二进制文件等
  - `/llvm-src`: LLVM 源代码
- Flex 2.6.4
- Bison 3.5.1
- CMake 3.16.3
- GCC, G++ 9.3.0

此外，默认 `PATH` 含有 `/llvm/bin`，故可以直接使用 `clang` 和 LLVM 提供的工具。

### 删除容器和镜像

```
docker rm compiler-labs
docker rmi ksqs/llvm
```

可用 `docker container ls -a` 列出所有容器，用 `docker image ls -a` 列出所有镜像。

### 其他事项

- Windows 下，对于共享文件夹中的文件，文件换行符尽量选择 **LF 格式** (`\n`)，而不是 CRLF 格式 (`\r\n`)，否则在进行执行脚本等操作时可能会出现问题。
- macOS 和 Windows 的 Docker 默认对内存等资源有所限制，如果要在容器环境中编译 LLVM，请注意调整资源限制 (Preferences → Resources → Advanced)。

## Ubuntu 下包管理器安装（方式二）

通过虚拟机、WSL 等方式配置 linux 环境（建议 Ubuntu 20.04）。然后用以下命令安装实验所需的软件

```
sudo apt update
sudo apt install cmake xz-utils build-essential wget git
sudo apt install llvm clang flex bison
```

## 验证是否成功搭建实验环境

创建文件 `fibonacci.c`（如果使用 Docker，在本地的共享文件夹中创建此文件）

```
// fibonacci.c
int fibonacci(int n) {
    if (n <= 1)
        return n;
    else
        return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    int n;
    n = 10;
    return fibonacci(n);
}
```

在命令行中输入（如果使用 Docker，进入容器后在 `/labs` 目录下执行以下命令）

如果觉得每次进入 Docker 环境都要输入 `cd /labs` 很麻烦，可以用 `echo "cd /labs" >> ~/.bashrc`，这样每次进入容器环境时都会自动进入 `/labs` 目录。

```
clang -S -emit-llvm fibonacci.c
lli fibonacci.ll; echo $?
```

若打印 `55`，表示实验环境搭建成功

## 配置 Git

后续实验将使用 Git 进行版本控制，使用 GitLab 进行代码托管。

Git 是一个分布式版本控制系统，可以用来管理代码。GitLab 是一个基于 Git 的代码托管平台，可以用来存储代码。

## 安装及初始化 Git

方式一中的 Docker 已经安装了 Git；如果按照方式二，也安装了 Git。如果打算在这些平台下使用，可跳过安装过程。

如果想在宿主机上使用 Git，可以参考 [Git 官方文档](#) 安装 Git，或者安装 GUI 界面的 Git 客户端，如 [GitHub Desktop](#)。

使用 Git 首先需要设置用户名和邮箱，这些信息会被记录在每次提交的 commit 中，用于标识提交者。

```
git config --global user.name "Your Name"
git config --global user.email "Your Email"
```

## 常用的 Git 命令

```
git status # 查看当前仓库的状态
git add . # 将所有修改添加到暂存区
git commit -m "commit message" # 将暂存区的内容提交到本地仓库
git push # 将本地仓库的内容推送到远程仓库
git pull # 将远程仓库的内容拉取到本地仓库
git log # 查看当前仓库的提交历史
git reset --hard HEAD~ # 回退到上一个版本（注意：这会丢失当前未提交的修改）
git checkout abcd1234 # 切换到某个 commit 或 branch
```

更多内容可以参考 [官方文档](#) 或者 [廖雪峰的教程](#)。

## 登录 GitLab

登录希冀平台，点击 GitLab，用与平台相同的用户名和密码登录 GitLab

## 配置 ssh 密钥

ssh 密钥能让用户在不输入密码的情况下，使用 `git push` 等命令与远程仓库进行交互。

## 生成 ssh 密钥对

如果你已经有了 ssh 密钥对，可以跳过此步骤

在想要运行 Git 的平台（Docker，虚拟机或实体机）执行以下命令，生成 ssh 密钥对。一路回车即可

```
ssh-keygen
```

## 将公钥添加到 GitLab

查看公钥：

- Windows 下，公钥文件位于 `C:\Users\用户名\.ssh\id_rsa.pub`，可以使用记事本打开查看。
- Linux 下使用以下命令查看公钥：

```
cat ~/.ssh/id_rsa.pub
```

然后添加公钥：

在 GitLab 中，点击右上角头像 → Settings → SSH Keys，将整个公钥粘贴进去，点击 Add key 即可。

复制时注意把开头的 `ssh-rsa` 和结尾的 `用户名@主机名` 都复制进去。

## Fork 实验仓库并克隆

将 [compiler\\_staff/2022fall-compiler\\_cminus](#) 仓库 Fork 到自己的帐户下。

然后克隆到本地：

```
git clone git@202.38.79.174:PB*****/2022fall-compiler_cminus.git
```

## 添加上游仓库及同步（后续实验会用到）

进入仓库目录：

```
cd 2022fall-compiler_cminus
```

### 1. 添加上游仓库

```
git remote add upstream \  
git@202.38.79.174:compiler_staff/2022fall-compiler_cminus.git
```

由于长度限制，上面的命令被分成了两行，建议在一行中输入（删去 `\`）。

使用 `git remote -v` 可以查看当前仓库的远程仓库信息。出现以下内容表示添加成功：

```
origin git@202.38.79.174:PB*****/2022fall-compiler_cminus.git (fetch)  
origin git@202.38.79.174:PB*****/2022fall-compiler_cminus.git (push)  
upstream git@202.38.79.174:compiler_staff/2022fall-compiler_cminus.git (fetch)  
upstream git@202.38.79.174:compiler_staff/2022fall-compiler_cminus.git (push)
```

如果想要移除名为 `upstream` 的远程仓库，可以使用 `git remote remove upstream`

## 2. 与上游仓库同步

```
git pull upstream master
```

这会将上游仓库的 master 分支合并到本地的 master 分支。

然后把更新推送到自己的仓库：

```
git push origin master
```

## 实验要求

---

### 提交内容

在希冀平台上提交 PDF 文件，包含以下内容：

- 成功运行 `fibonacci.ll` 的截图
- 简述对 Docker 的理解

### 提交时间

Deadline: 2022-09-19 23:59:59 (UTC+8)