# Cmake入门

## 1.1第一个Cmake

1. //first\_cmake.cpp
2. #include <iostream>
3. **using** **namespace** std;
4. **int** main(**int** argc,**char** \*argv[])
5. {
6. cout<<"first CMake TEST"<<endl;
7. **return** 0;
8. }
9. # CMakeLists.txt cmake文件名大小写不敏感
11. #指定cmake最低版本
12. cmake\_minimum\_required(VERSION 3.20)
14. #构建项目的名称
15. project(first\_cmake)
17. #构建执行程序
18. add\_executable(first\_cmake first\_cmake.cpp)

cmake -S . -B build 当前路径

Cmaek --build build

## 1.2 Cmake基本概念

Cmake用于构建、测试和软件打包的开源跨平台工具

1.为什么需要构建工具？

* 避免硬编码路径
* 在多台计算机上构建一个包
* 持续集成
* 支持不同操作系统
* 使用IDE，但不是所有情况
* 想描述程序逻辑结构，而不是标志和命令
* 使用库
* 使用其他工具帮助编写代码
* 想使用单元测试

注：

持续集成 ：

每次集成都通过自动化的制造（包括提交、发布、自动化测试）来验证，准确地发现集成错误

快速错误，没完成一点更新，就集成到主干，可以快速发现错误，定位错误也比较容易

各种不同的更新主干，如果不经常集成，会导致集成成本变大

让产品可以快速的通过，同时保持关键测试合格

自动化测试，只要有一个测试用例不通过就不集成

集成并不能删除发现的错误，而是让他们很容易改正

2.为什么用Cmake?

* 自动搜素可能需要的程序、库和头文件的能力
* 独立的构建目录，可以安全清理
* 创建复杂的自定义命令
* 配置时选择可选组件能力
* 从简单的文本文件CMakeLists.txt自动生成工作区和项目能力
* 在静态和共享构建之间轻松切换能力

## 1.3 Cmake工作原理

