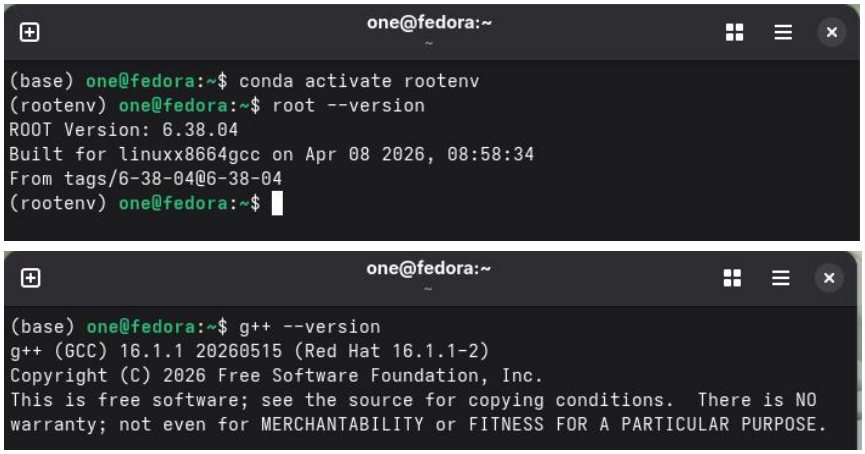
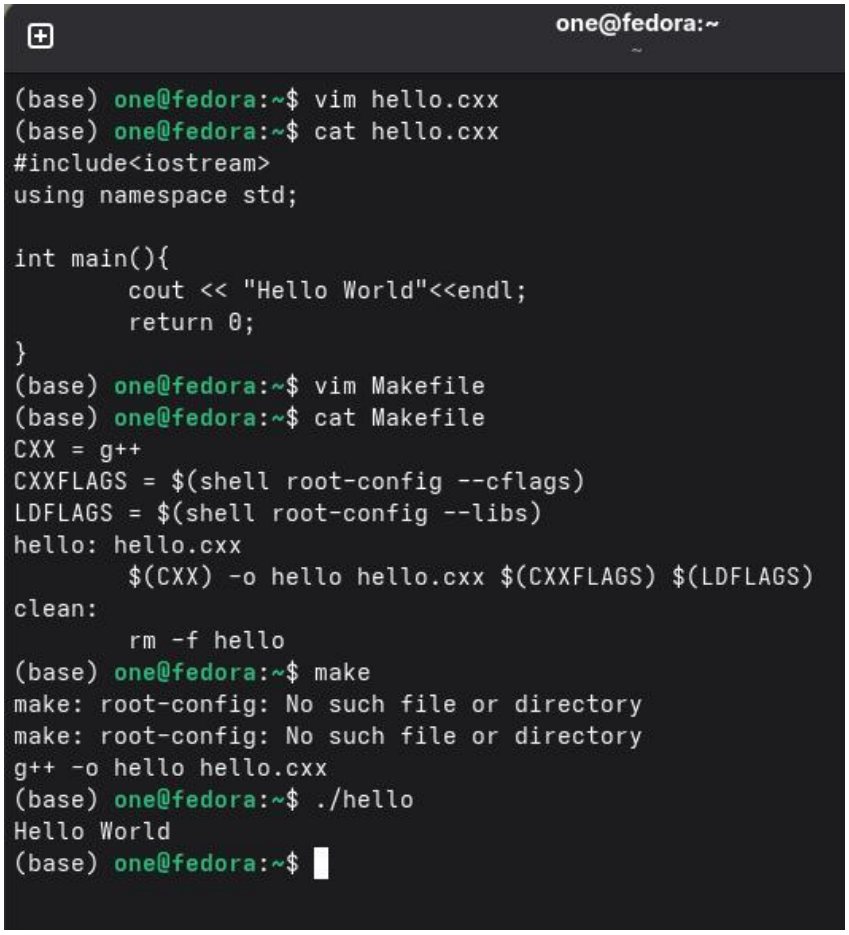
5.11 科创

1、安装 g++、ROOT

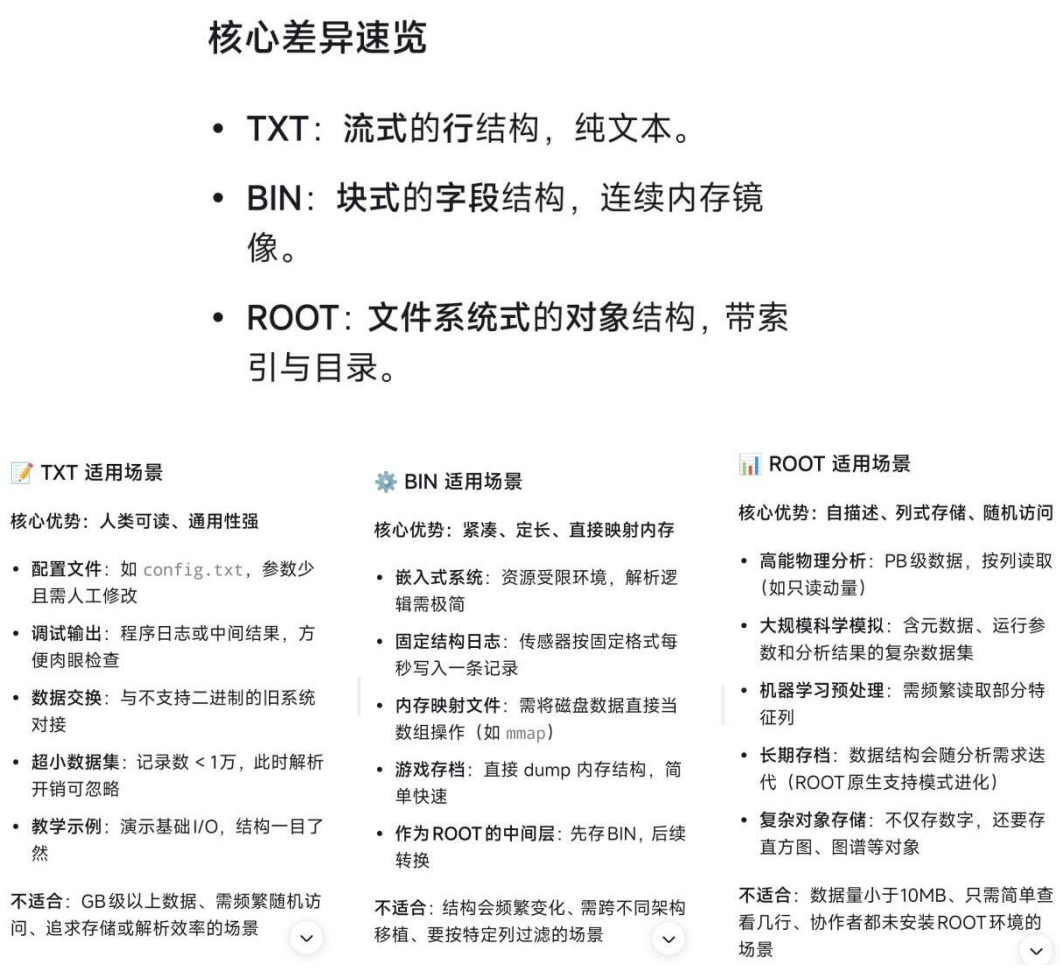


2、写一个 Makefile，编译第一个 root 程序，输出“Hello World”即可



3、拓展 1：（为什么要用 ROOT？）了解 ROOT 软件的存储结构，为什么用 ROOT 存储，而不是 txt、bin？

ROOT 的储存结构：头文件、数据记录和尾部；调用文件内容时，检索目录，只读取对应内容



高能物理分析，数据量大，txt、bin 调用不方便。ROOT 列式储存，可快速读取指定数据，并且可压缩，内存小。

4、拓展 2：（为什么要学 Shell 语言？）已知有一个文件夹的 M 个文件需要提交，分配得到的 CPU 核数为 N，执行文件为 executable 变量，如何批量实现提交作业？

学习Shell语言是为了：

1. **批量化**：处理大量（M个）文件/作业时，循环自动化
2. **智能化**：根据实际情况（N个CPU核）动态调整策略
3. **流程化**：构建“提交→监控→重试→合并”完整流水线
4. **可复用**：写一次脚本，反复使用，避免重复劳动

在HPC集群、Condor、Slurm等环境中，**Shell就是胶水语言**，把各种工具粘合成自动化工作流。

理由：批量处理重复操作，提升效率

实现思路：准备文件列表——编写 Condor 提交文件（定义变量，遍历）——控制同时处理数量小于 N

核心目标：
有 M 个数据文件要处理，分配给用户的 CPU 核数是 N，需要把这些文件全部处理完，但不能让同时运行的作业数超过 N（否则会被系统限制或排队）。

解决思路分三步：

1. **准备文件列表**
   先把所有需要处理的 M 个输入文件的完整路径，全部写到一个文本文件里，每行一个文件。这样 Condor 可以一行一行地读取。
2. **编写 Condor 提交描述文件**
   - 在这个文件里，不要写死具体的文件名，而是定义一个变量（比如叫 `indata`），代表“当前要处理的这个输入文件”。
   - 在 `executable`（你要运行的程序或脚本）后面，用 `$(indata)` 把这个文件路径传进去。
   - 输入、输出、日志文件的名字也要用 `$(indata)` 或者 Condor 自带的作业ID（`$(ClusterId).$(ProcId)`）来区分，防止不同作业意思就是 从 filelist.txt 里读取每一行，每次把这一行的值赋给 `indata`，然后提交一个作业”。
3. **控制并发数量**
   - 默认 Condor 会尽可能多地同时跑作业，但这样可能超过你分配的 N 个核，导致作业被挂起或挤占别人资源。
   - 所以提交时要加限制：`condor_submit -maxjobs N 你的提交文件`。这样 Condor 最多同时运行 N 个作业，跑完一个再启动下一个，直到 M 个全部完成。

总结一句话：
把 M 个文件路径写进列表，让 Condor 用 `queue from` 语句为每个文件生成一个作业，再用 `-maxjobs N` 限制同时运行的作业数不超过 CPU 核数 N。