1、 1) 安装 g++：

```
(base) ksty@fedora:~$ g++ --version
g++ (GCC) 16.1.1 20260515 (Red Hat 16.1.1-2)
Copyright (C) 2026 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There
is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PU
RPOSE.

(base) ksty@fedora:~$
```

2) 安装 ROOT：

```
(rootenv) ksty@fedora:~$ root --version
ROOT Version: 6.38.04
Built for linuxx8664gcc on Apr 08 2026, 08:58:41
From tags/6-38-04@6-38-04
(rootenv) ksty@fedora:~$
```

2、 写一个 Makefile，编译第一个 root 程序，输出"Hello World"即可.

ksty@fedora:~ — nano HelloWorld.C

```
 GNU nano 8.7.1                 HelloWorld.C
#include <iostream>
#include <TROOT.h>
using namespace std;

int main(){
    cout << "Hello World ROOT!" << endl;
    return 0;
}
```

```
(rootenv) ksty@fedora:~$ ls
Desktop    Downloads     Makefile    Music     Public     Videos
Documents  HelloWorld.C  miniconda3  Pictures  Templates
(rootenv) ksty@fedora:~$ make
g++ HelloWorld.C -o HelloWorld -fvisibility-inlines-hidden -fmessage-
length=0 -march=nocona -mtune=haswell -ftree-vectorize -fPIC -fstack-
protector-strong -fno-plt -O2 -ffunction-sections -pipe -isystem /hom
e/ksty/miniconda3/envs/rootenv/include -pthread -std=c++20 -m64 -fsiz
ed-deallocation -I/home/ksty/miniconda3/envs/rootenv/include -L/home/
ksty/miniconda3/envs/rootenv/lib -lCore -lImt -lRIO -lNet -lHist -lGr
af -lGraf3d -lGpad -lROOTVecOps -lTree -lTreePlayer -lRint -lPostscri
pt -lMatrix -lPhysics -lMathCore -lThread -lROOTNTuple -lROOTNTupleUt
il -lMultiProc -lROOTDataFrame -Wl,-rpath,/home/ksty/miniconda3/envs/
rootenv/lib -pthread -lm -ldl -rdynamic
./H(rootenv) ksty@fedora:~$ ./HelloWorld
Hello World ROOT!
(rootenv) ksty@fedora:~$
```

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

①ROOT采用.root树状二进制格式存储高能粒子数据。——海量数据以二进制存储，且可分层归类，便于查找读取，且压缩极强，文件超小。
②高能对撞实验产出的数据量极大，txt纯文本格式冗余多，大批量处理很卡顿；通用bin文件缺少标准化结构，不同程序难以互通
ROOT是专为高能物理开发的工具，其文件格式不仅压缩高效、检索迅速，还配套了完整的数据处理与绘图功能，跨平台、跨实验室都能通用，自然成为首选。

4、拓展2：(为什么要学Shell语言?)
①自动化批量处理：大量数据文件拆分、分配任务，替代手动操作，提升效率，减少失误.
②目录和文件管理：自动创建分类目录，生成任务配置文件，规整日志与结果
③环境初始化：加载ROOT、实验软件环境变量，保证程序正常运行。
④对接集群调度：调用condor_submit提交作业，是本地和集群交互的主要方式.
⑤灵活可控：通过传入参数，自由指定程序、文件、核数、输出路径，方便调试和复用。

2) 批量实现提交作业的解释.
①参数读取与初始化：脚本首先读取用户传入的可执行程序路径、文件列表、输出目录、CPU核数等参数，完成基础配置。
②文件均匀分配：根据文件总数M和核数N，计算每个核需要处理的文件数(mini_files_per_core = M//N，剩余文件均匀分给前M%N个核)，确保每个核的负载平衡。
③生成任务文件：为每个核生成对应的输入文件，文件中包含该核需要处理的所有数据文件路径。
④生成Condor提交脚本：根据提供的模板，自动生成.sub提交文件配置作业的运行环境、资源需求，日志路径，以及每个任务对应的执行脚本
⑤提交作业到集群：调用condor_submit命令，将所有任务提交到Condor集群，实现批量并行处理。