1 Introducción

Éste manual pretende introducir a los usuarios en el manejo del software **Dune**, que se puede ubicar en la dirección `https://dune-project.org/`, software de simulación numérica DUNE (Distributed and Unified Numerics Environment), proyecto de código abierto que proporciona una infraestructura flexible y modular para la solución de ecuaciones diferenciales parciales.

2 Instalación

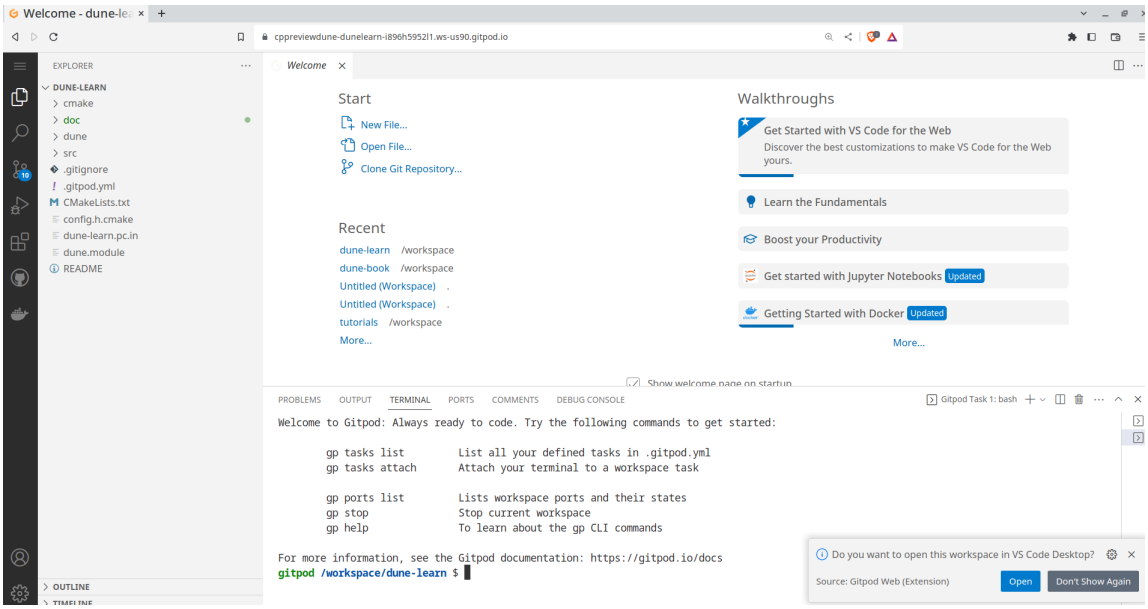
Para la correr el software DUNE, vamos a utilizar el repositorio `https://github.com/cpp-review-dune/dune-learn`, diseñado para funcionar en la nube, utilizando las ventajas que nos ofrece `https://www.gitpod.io/`, en el que su puede utilizar una imagen docker, que contiene las dependencias y repositorios necesarios para su funcionamiento.

En la sección de C++, se presentará **dunepdelab**, que fue estudiada en el curso presentado en el año 2022, de manera remota y cuya link está en: `https://dune-pdelab-course.readthedocs.io/en/latest/intro.html`, y en la sección de Python, utilizaremos el Jupyter Notebook para introducir el uso de **dunefem**.

3 C++

3.1 Primer proyecto en Dune

Para iniciar la ejecución, una vez hemos ingresado a la plataforma y estamos en la nube, debemos ver una imagen similar a la siguiente:



Es muy similar a Vscode, cómo se puede apreciar. Para hacer mi primer proyecto, en la terminal debo escribir: gitpod /workspace $ duneproject

```
== Dune project/module generator ==

duneproject will assist you in the creation of a new Dune application.
During this process a new directory with the name of your project will be
created. This directory will hold all configuration and Makefiles and a
simple example application.

1) Name of your new Project? (e.g.: dune-grid):
```

En ésta parte, debo completar la información de mi proyecto, el nombre vamos a poner por ejemplo **dune-prueba**, luego

```
2) Which modules should this module depend on?
   The following modules have been found:
   dune-common dune-geometry dune-uggrid dune-grid dune-localfunctions dune-istl
   dune-typetree dune-functions dune-alugrid dune-pdelab dune-fem dune-learn
   Enter space-separated list:
```

En éste caso, vamos a usar **dune-pdelab**, luego pregunta:

```
3) Project/Module version?
```

Por ejemplo puede ser 01, luego nos pide el email:

```
4) Maintainer's email address? maintainer@unal.edu.co
```

Finalmente, nos pregunta si los datos están correctos:

```
creating Project "dune-prueba", version 01
which depends on "dune-pdelab"
with maintainer "maintainer@unal.edu.co"
Is this information correct? [y/N] y
```

Si los datos son correctos, asignamos "y", de lo contrario "N", y finalmente enter para iniciar la configuración. Una vez iniciada la configuración, se crea una carpeta con el nombre **dune-prueba**, que fue el nombre del proyecto que asignamos. Una vez que el proyecto se ha construido, podemos utilizar el comando

```
gitpod /workspace $ ls
```

Dentro del listado debe aparecer la carpeta dune-prueba, que si explora dentro de ella, encontrará varios archivos y carpetas. Se puede dirigir a la siguiente dirección:

```
cd /workspace/dune-prueba/src $
```

Cuando liste, encontrará dos archivos

```
CMakeLists.txt  dune-prueba.cc
```

En el primer archivo, usted encuentra lo siguiente:

```
add_executable("dune-prueba" dune-prueba.cc)
target_link_dune_default_libraries("dune-prueba")
```

Significa que se ha creado un código fuente, que se llama "dune-prueba.cc", tiene el mismo nombre del proyecto que creamos, y en el que está escrito nuestro primer programa, el "Hola mundo de DUNE". A continuación se puede apreciar el contenido completo del primer programa :

3.2 Compilación

Para compilar el programa, vamos a crear una carpeta

```
gitpod /workspace $ mkdir build
```

Build, se utiliza para construir allí los aspectos necesarios para la compilación del proyecto, y aparecerá después de compilar el archivo ejecutable. Una vez creada la carpeta, es necesario utilizar el comando **Cmake**. En éste caso se indica a cmake que utilice como fuente ("S" de source) la carpeta **dune-prueba** y que la construcción de la compilación se va a guardar en la carpeta **build**, por eso se escribe ("-B" build)

```
gitpod /workspace $ cmake -S dune-prueba -B build
```

Luego de hacer el proceso, si hacemos una lista, tendremos:

```
gitpod /workspace $ build dune-prueba
```

Es necesario aclarar, que nuestro código estará en la carpeta **dune-prueba**, mientras que el ejecutable y todos los archivos necesarios para la compilación están en la carpeta **build**. A continuación, ingresamos a la carpeta **build** y usamos el comando **make**:

```
gitpod /workspace/build $ make
```

Con lo que se iniciará el proceso de generación del archivo ejecutable, que estará ubicado en éste caso, en la dirección y falta ejecutarlo de la siguiente forma:

```
gitpod /workspace/build/src $ ./dune-prueba
```

```
// -*- tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 2 -*-
// vi: set et ts=4 sw=2 sts=2:

#ifdef HAVE_CONFIG_H
# include "config.h"
#endif
#include <iostream>
#include <dune/common/parallel/mpihelper.hh> // An initializer of MPI
#include <dune/common/exceptions.hh> // We use exceptions
#include <dune/learn/learn.hh>

int main(int argc, char** argv)
{
  try{
    // Maybe initialize MPI
    Dune::MPIHelper& helper = Dune::MPIHelper::instance(argc, argv);
    std::cout << "Hello World! This is dune-prueba." << std::endl;
    if(Dune::MPIHelper::isFake)
      std::cout<< "This is a sequential program." << std::endl;
    else
      std::cout<<"I am rank "<<helper.rank()<<" of "<<helper.size()
        <<" processes!"<<std::endl;
    return 0;
  }
  catch (Dune::Exception &e){
    std::cerr << "Dune reported error: " << e << std::endl;
  }
  catch (...){
    std::cerr << "Unknown exception thrown!" << std::endl;
  }
}
```

```
Hello World! This is dune-prueba.
I am rank 0 of 1 processes!
```

4 Python