Linux admite multitud de lenguajes de programación (*Java, C/C++, C#, Perl, Python, Lisp,* ...), editores con reconocimiento de código (*Gedit, Emacs, Geany, **Visual Studio Code**, Atom,*...), diseñadores de interfaces (***Glade**, QTDesigner*) y entornos de desarrollo (*Anjuta, Eclipse, Netbeans, MonoDevelop, Gambas, KDevelop,* ...) que permiten desarrollar aplicaciones para este sistema operativo.

Si decides utilizar ***Visual Studio Code*** es buena idea instalar el idioma español desde *Extensiones*  y buscar *Spanish Language Pack*.

Con el comando **code .** se abre la carpeta actual en VS Code.

https://code.visualstudio.com/docs/editor/codebasics

C.

Para crear/editar el código fuente lo haremos con un editor de texto como *Gedit o **Geany***, asignando como extensión .c

```
gedit ejercicio1.c
```

Para compilar lo haremos con *gcc* que normalmente estará instalado en cualquier distribución.

```
gcc ejercicio1.c –o ejercicio
```

Si no está instalado el compilador lo instalaremos con: `sudo apt install gcc`

Aunque si queremos que instale también librerías adicionales:

```
sudo apt install build-essential gdb
```

Si utilizas *VS Code* cuando nombras un archivo nuevo con extensión **.c**, te propondrá instalar la *extensión C/C++*. Using C++ on Linux in VS Code

Podemos ejecutar el archivo *ejercicio* para probar su funcionamiento: `./ejercicio1`

```
// ejercicio1.c
#include <stdio.h>
int main(){
    printf("Hola mundo\n");
    return 0;
}
```

```
// ejercicio2.c
// sintaxis: ./ejercicio2 argumento1 argumento2 …
#include <stdio.h>
#include <stdlib.h>
int main(int argc,char *argv[]) {
  if (argc>1) //argumento 0 => nombre del archivo ejecutable
  {
          int i;
          for (i=1; i<argc; i++) {
          printf("argumento: %i = %s \n", i, argv[i]);
          }
  }
  system("ls -l"); //ejecuta el comando del sistema
  return 0;
}
```

El compilador de C obtiene código no administrado por lo que el resultado es un binario ejecutable directamente en la plataforma destino (sistema y procesador concreto).

Si el resultado de la compilación o ejecución no es correcto, debes volver a editar y compilar.

Para recoger información del usuario puedes utilizar la función *scanf*:

https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C/Interacci%C3%B3n_con_el_usuario

Manual on-line de C → https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C

Sencillo manual de C++ → https://upload.wikimedia.org/wikipedia/commons/a/ab/Manual_C%2B%2B.PDF

Java.

Podemos comprobar si tenemos instalado java en el equipo con el comando:

```
java –versión
```

➔ nos informará de la versión y si tenemos instalado la JVM con su JRE que nos permitirá ejecutar *bycode* de java (archivos *.class*). Si no tenemos el JDK que nos permite compilar los programas podemos instalarlo con el comando:

```
sudo apt-get install default-jdk                    (openjdk-11-jdk)
```

Para editar un programa podemos utilizar el editor *gedit* u otro editor con *Geany* :

```
gedit saludo.java &
```

➔ El *&* ejecutará en segundo plano el editor y así será más fácil volver al programa para modificarlo si se encuentran errores.

```
import java.util.Scanner;
public class saludo{
  public static void main(String[] args){
          System.out.println("Hola...");
          for (String nombre: args){
System.out.println(nombre);}
String nombre="";
Integer numero;
Scanner escaner = new Scanner(System.in);
System.out.print ("Introduzca su nombre: ");
          nombre = escaner.nextLine ();
          System.out.print ("Introduzca un número entero: ");
          numero=escaner.nextInt();
          escaner.close();
          System.out.println ("Nombre introducido: \"" +nombre+ "\"");
System.out.println("El número "+número);
  }
}
```

Compilamos con *javac*:

```
javac saludo.java
```

- ➔ Si tenemos errores debemos editar el programa, corregirlos y volver a compilar.
- ➔ Si todo ha ido bien podemos ejecutar el programa (en realidad la *Máquina Virtual de Java* junto con el *Java Runtime Environment* lanzará el *bytecode* contenido en *saludo.class*):

`java saludo` *(sin extensión)*

Es posible conseguir aplicaciones con interfaces gráficas sencillas usando los componentes *AWT* o *Swing*. *AWT* se adapta al aspecto de cada entorno de escritorio y *Swing* permite elegir el *look & feel*.

Ejemplo con AWT:

```
import java.awt.*;
import java.awt.event.*;
public class pruebaAWT extends Frame {
  Button boton=new Button("botón");
Label etiqueta=new Label("este texto se muestra en el frame...");
    public pruebaAWT(){
        super("Demo AWT");
        setSize(600, 400);
        setLayout(new FlowLayout());
        add(boton);
        add(etiqueta);
          addWindowListener(new WindowAdapter()
          {
                  public void windowClosing(WindowEvent e)
                  { System.exit(0); }
          });
          boton.addActionListener(new ActionListener()
          {
                  public void actionPerformed(ActionEvent e)
                  { etiqueta.setText("Has pulsado el botón!"); }
          });
    }
    public static void main(String args[]){
      pruebaAWT frame = new pruebaAWT();
      frame.setVisible(true);
    }
}
```

Ejemplo con Swing:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.UIManager.*;
public class holaSwing {
          public static void main(String args[]){
      try{
  JFrame.setDefaultLookAndFeelDecorated(true);
  UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
  }
  catch(Exception e)
  {
          e.printStackTrace();
  }
          javax.swing.SwingUtilities.invokeLater(new Runnable() {
             public void run() {
                 mostrarVentana();
              }
```

```
        });
          }
private static void mostrarVentana() {
        //Crea y configura la ventana.
        JFrame frame = new JFrame("swingDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JLabel etiqueta = new JLabel("texto mostrado");
        etiqueta.setPreferredSize(new Dimension(400, 300));
        frame.getContentPane().add(etiqueta, BorderLayout.CENTER);

        //Muestra la ventana.
        frame.pack(); //ajusta el tamaño de la ventana a su contenido
        frame.setVisible(true);
          }
}
```

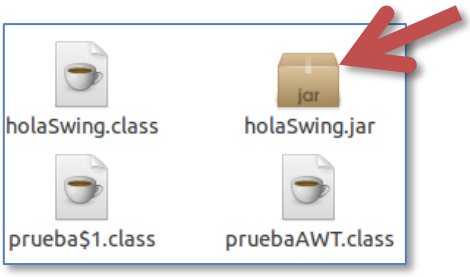
➔ Se puede empaquetar una o varias clases java en un único archivo .jar que se puede ejecutar directamente desde el entorno gráfico asociándolo al *runtime* de Java.

`jar cf nombreArchivoDestino.jar clase1.class clase2.class …`

→*`jar cf holaSwing.jar holaSwing.class 'holaSwing$1.class'`*

Una vez hecho esto habrá que editar el archivo de manifiesto abriendo el archivo .jar con el gestor de archivadores y dentro de la carpeta META-INF está el archivo MANIFEST.MF, lo editamos e incluimos la línea: `Main-Class: holaSwing`

→*holaSwing* es el nombre de la clase que será el punto de entrada a la aplicación.



→ *`java -jar holaSwing.jar`* para ejecutar el paquete creado.

Puedes asignar el permiso de ejecución con *`chmod +x holaSwing.jar`*

Comprueba que también se pueden ejecutar desde *Windows* si tienes la **máquina virtual de Java** instalada (con el JRE o el JDK).

GTK

GTK es una biblioteca de componentes gráficos multiplataforma para desarrollar interfaces gráficas de usuario. Existen múltiples implementaciones para diversos lenguajes de programación, como *C, Java, Python o Perl*.

Suele ser la opción elegida para el entorno de escritorio *Gnome*.

Aunque en *Ubuntu* el *runtime* está instalado por defecto, para instalar las librerías de desarrollo:
apt install libgtk-3-dev

Hola Mundo en C

Si tienes problemas al compilar:
apt install pkg-config

Para compilar:

```
gcc -o example-1 example-1.c `pkg-config --cflags --libs gtk+-3.0`
```

- The GTK Project - A free and open-source cross-platform widget toolkit
- GTK+ 3 Reference Manual: GTK+ 3 Reference Manual (gnome.org)

Para crear la interfaz gráfica es una buena opción ***Glade***: primeros pasos, GJS con Glade
O ***GtkBuilder***: https://developer.gnome.org/gtk3/stable/GtkBuilder.html

GTK está *basado en eventos*. Los *widgets* escuchan eventos, como un clic en un botón o un cambio de tamaño de la ventana, y los notifican a la aplicación.

➔ Solo tendrás que compilar *C* y *Java*, sin embargo, *Perl, Javascript* o *Python* son scripts ejecutables directamente.

➔ Más enlaces: Empezando con GTK, Wiki Gnome, PerlDoc, Guía JavaScrtipt, Python-GTK