

Curso Linux – Nivel I

Trabajo Practico Final

“Alternativas de Programación en Linux”

*Copyright © 2009, Basiliquiotis, Santiago; Lussa, Leandro Gabriel; Quirós, Nadia Soledad
Permission is granted to copy, distribute and/or modify this document under the terms of
the GNU Free Documentation License, Version 1.2 or any later version published by the
Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-
Cover Texts. A copy of the license is included in the section entitled "GNU Free
Documentation License .*

Integrantes:

**Basiliquiotis, Santiago – santiagobasi@hotmail.com
Lussa, Leandro Gabriel – leo.g.lu@hotmail.com
Quirós, Nadia Soledad – nadia_quiros@hotmail.com**

Índice

INTRODUCCIÓN.....	3
CONCEPTOS TEÓRICOS.....	4
LENGUAJE “C”.....	5
INTRODUCCIÓN Y DEFINICIÓN.....	5
ENTORNO DE PROGRAMACIÓN.....	6
LENGUAJE “PYTHON”.....	8
INTRODUCCIÓN Y DEFINICIÓN.....	8
Multiplataforma.....	8
Lenguaje Interpretado o Lenguaje Script.....	8
Tipado Dinámico.....	9
Fuertemente Tipado.....	9
INSTALACIÓN.....	9
CONFIGURACIONES BÁSICAS PARA COMENZAR A PROGRAMAR.....	10
LENGUAJE “JAVA”.....	14
HISTORIA.....	14
CARACTERÍSTICAS.....	15
ENTORNO DE PROGRAMACIÓN E INSTALACIÓN.....	15
EL J2SE.....	16
UN ENTORNO VISUAL DE DESARROLLO.....	16
NetBeans.....	17
La Plataforma NetBeans.....	17
Eclipse.....	18
INSTALACIÓN DE LOS PAQUETES DEBIAN NECESARIOS.....	18
Instalación.....	18
CREAMOS EL PROGRAMA.....	20
EL MÉTODO MAIN.....	21
SACAR TEXTO POR LA PANTALLA.....	21
EJECUTAR EL PROGRAMA.....	22
CLASSPATH.....	23
OPCIÓN DEL COMANDO JAVA.....	23
LENGUAJE “RUBY”.....	24
DEFINICIÓN e INTRODUCCIÓN.....	24
INSTALACIÓN.....	25
PRIMEROS PASOS EN RUBY.....	26
SINTAXIS BÁSICA.....	28
CONCLUSIÓN.....	29
BIBLIOGRAFÍA.....	30

INTRODUCCIÓN

Este trabajo esta orientado a todos aquellos programadores que están incursionando en el mundo del desarrollo de Software Libre, en entorno a un Sistema Operativo tan completo y que posee una gran diversidad de herramientas disponibles, como lo es GNU/Linux.

Nuestra idea se basa en exponer diferentes lenguajes de desarrollo de software; con el fin de interiorizar a todos aquellos programadores inexpertos o no experimentados, para que puedan evaluar y decidir de acuerdo a las aplicaciones que quieran lograr, cual es el lenguaje más apropiado para cumplir sus objetivos.

Luego de una investigación y análisis, incorporamos a esta exposición cuatro lenguajes importantes como lo son: C, Java, Python y Ruby. Basándonos primordialmente en las características principales de los mismos, como son: Funcionalidad, Flexibilidad, Fácil Uso, Portabilidad, Potencialidad y Popularidad con respecto a documentación y herramientas existentes.

CONCEPTOS TEÓRICOS

A lo largo del documento se utilizarán conceptos específicos; por lo cual hemos decidido definirlos brevemente con el fin de lograr un mejor y fácil entendimiento por parte del lector.

SOFTWARE LIBRE: Es todo aquel software que puede ser usado, modificado y redistribuido sin restricción alguna, solo nombrando en ellos a su autor o autores. Esto no significa que es gratis, puede ser vendido.

GPL: Licencia que cuida la libre distribución, modificación y uso de software.

GNU: Es un proyecto iniciado por Richard Stallman con el objetivo de crear un sistema operativo completamente libre: el sistema GNU.

CLASES: Es un conjunto de objetos agrupados de acuerdo a características que poseen en común

OBJETOS: son estructuras en la cual se agrupan datos, y métodos o funciones relacionadas. Con la característica de ocultar su contenido a usuarios y programadores.

MÉTODOS: Son funciones contenidas en los objetos que utilizan los datos que existen en él, con el fin de ejecutar alguna acción.

COMPILADORES: es un programa que traduce otro programa hecho en un lenguaje de alto nivel, a otro de mas bajo nivel para que la maquina pueda interpretarlo; generalmente el lenguaje mas bajo es lenguaje assembler.

LENGUAJE “C”



INTRODUCCIÓN Y DEFINICIÓN

C es un lenguaje de programación creado en 1972 por Ken Thompson y Dennis M. Ritchie en los Laboratorios Bell como evolución del anterior lenguaje B, a su vez basado en BCPL. Al igual que B, es un lenguaje orientado a la implementación de Sistemas Operativos, concretamente Unix. C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones.

Se trata de un lenguaje débilmente tipificado de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos.

Aunque a evolucionado modestamente a través del tiempo, la creciente popularidad de C y la creación de compiladores por grupo no involucrados en su diseño, se combinaron para demostrar la necesidad de una definición del lenguaje mas precisa y contemporánea. Por eso en 1983, el American National Standards Institute (ANSI) estableció un comité cuyos propósitos eran producir una definición no ambigua del lenguaje C, independiente de la maquina y su resultado es el estándar ANSI para el lenguaje C.

ANSI C es un lenguaje de programación de propósito general que ofrece como ventajas economía de expresión, control de flujo, estructuras de datos modernos y un rico conjunto de operadores. No esta basado en ningún área especial de aplicación; pero su ausencia de restricciones y su generalidad lo hacen mas conveniente y efectivo para muchas tareas que otros lenguajes mas poderosos.

ENTORNO DE PROGRAMACIÓN

GNU Compiler Collection -Colección de Compiladores GNU- es un conjunto de compiladores creados por el proyecto GNU. GCC es software libre y lo distribuye la FSF bajo la licencia GPL. Estos compiladores se consideran estándar para los sistemas operativos derivados de UNIX, de código abierto. GCC requiere el conjunto de aplicaciones conocido como *binutils* para realizar tareas como identificar archivos objeto u obtener su tamaño para copiarlos, traducirlos o crear listas, enlazarlos , o quitarles símbolos innecesarios.

Originalmente GCC significaba *GNU C Compiler (compilador GNU para C)*, porque sólo compilaba el lenguaje C .

GCC es parte del proyecto GNU, el cual tiene como objetivo mejorar el compilador usado en los sistemas GNU incluyendo la variante GNU/Linux. El desarrollo de GCC usa un entorno de desarrollo abierto y soporta muchas otras plataformas con el fin de fomentar el uso de un compilador-optimizador de clase global, para atraer muchos equipos de desarrollo, para asegurar que GCC y los sistemas GNU funcionen en diferentes arquitecturas y diferentes entornos, y más aún, para extender y mejorar las características de GCC.

Este compilador como se dijo anteriormente viene incluido en todos los sistemas operativos derivados de UNIX. En extraño caso que no se encuentra en la versión de GNU/Linux Debian

Se lo puede instalar siguiendo los pasos que se detallan a continuación:

1.Como primer paso vamos a encargarnos de editar nuestro archivo `/etc/apt/sources.list` y agregar como repositorio de paquetes: deb <http://ftp.debian.org> etch main contrib non-free.

2.Ahora desde el interprete de comando y como el superusuario escribir los siguientes comandos

- a. `apt-get update` → Esto es para que actualice los paquetes disponibles
- b. `apt-get install gcc` → esta linea lo que hace es instalar GCC en la maquina
- c. `apt-get install libc-dev` → Bibliotecas de desarrollo y archivos de cabecera

Ahora ya esta instalado y listo para usarse.

HOLA MUNDO!!!

Para probar el GCC vamos a realizar el famoso programa “Hola mundo!!!”. Lo primero que debemos hacer es abrir un editor de texto, creando un archivo con extensión “.c”. En nuestro caso y a modo de ejemplo lo llamaremos “hola.c”. Luego copiamos el siguiente código.

```

nadia@nadia: ~
Archivo Editar Ver Terminal Ayuda
GNU nano 2.0.9 Fichero: hola.c

#include <stdio.h>

main ()
{
    printf("Hola Mundo\n");
}

```

[Leer 8 líneas]

^G Ver ayuda ^O Guardar ^R Leer Fich ^Y Pág Ant ^K Cortar Tex ^C Pos actual
 ^X Salir ^J Justificar ^W Dónde Está ^V Pág Sig ^U PegarTxt ^T Ortografía

La línea `#include <stdio.h>` significa que incluye información acerca de la biblioteca estándar de entrada y salida. `Main ()` define una función llamada `main` que no recibe valores de argumentos. Las proposiciones de `main` están encerradas entre llaves. Y por último `printf("Hola mundo\n")` es llamada por la función `main` para imprimir en pantalla lo que está entre comillas, “/n” representa el carácter nueva línea.

Una vez creado y editado el archivo `hola.c` guardamos los cambios y salimos. En el intérprete de comando:

```
–gcc hola.c -o nombre
```

Lo que hace esta línea es compilar `hola.c`; la opción `–o` especifica que la salida se va a guardar en el archivo `nombre`. Esto es opcional ya que si no se ingresara, el `gcc` automáticamente generaría un salida bajo el nombre `a.out`.

Por último para probar nuestro programa lo que tenemos que hacer es ejecutar en el intérprete:

```

usuario@usuario:/home/usuario# ./nombre
Hola mundo
usuario@usuario:/home/usuario#

```

LENGUAJE “PYTHON”



INTRODUCCIÓN Y DEFINICIÓN

El lenguaje que vamos a ver a continuación es una de las mejores opciones que tiene Linux en materia de programación, tanto por su sencillez, rapidez, y su completa librería y herramientas disponibles.

Es uno de los lenguajes elegidos por la mayoría de los usuarios (incluyéndome) a la hora de comenzar con el mundo de la programación, y donde, desarrollar una aplicación se torna, como dije anteriormente, muy simple (creo que es lo que buscamos como principal característica) y desde mi punto de vista divertido a la vez.

Para que vean que es muy popular, y mayor a muchos de los programas disponibles, puedo citarles a empresas como Google, Yahoo, la misma NASA y todas las distribuciones de Linux, usándolo con éxito en sus funciones.

Fue creado por Guido van Rossum a comienzos de los años 90 y su nombre se inspira en el grupo cómico de origen inglés llamado “Monty Python”.

Lo que hay que tener en cuenta es que Python no es recomendable para programación de bajo nivel y tampoco para aplicaciones en las que el rendimiento sea crítico.

Ahora voy a nombrar algunas de las características principales con las que cuenta:

Multiplataforma

Esta disponible en cantidad de plataformas (UNIX, Solaris, Linux, DOS, Windows, OS/2, entre otros.), por lo que vamos a poder correrlo sin problemas en cualquiera de las mencionadas plataformas.

Lenguaje Interpretado o Lenguaje Script

Quiere decir que se ejecuta mediante un programa intermedio llamado Interprete, en lugar de ser compilado el código a lenguaje máquina.

La ventaja que tienen los lenguajes compilados es que su ejecución es más rápida. Sin embargo los lenguajes interpretados son más portables y más flexibles.

Se podría decir que es un lenguaje semi-interpretado, ya que tiene muchas de las características de los lenguajes compilados. En Python, tanto como en Java y en muchos otros lenguajes, el código fuente es traducido a un pseudo código máquina intermedio llamado bytecode

cuando es ejecutado por primera vez.

Tipado Dinámico

Se refiere a que no es necesario declarar el tipo de dato que va a contener una variable. Lo va a determinar en su ejecución según el tipo de valor al que se asigne, pudiendo cambiar el tipo de de la variable si se le asigna otro valor.

Fuertemente Tipado

Quiere decir que no se puede tratar a una variable como si fuera de un tipo distinto al que tiene, es necesario convertirlo anteriormente. En el caso de una variable string por ejemplo, no podemos tratarle como una variable numérica. En otros lenguajes, cambiaría dicha variable para adaptarse al comportamiento, pero es probable que posteriormente se produzcan algunos errores inesperados.

Y como ultima principal característica es un lenguaje Orientado a Objetos. Permite la introducción de objetos y clases en todo su entorno y en donde al momento de su ejecución en nuestro programa, dichos objetos interactúan entre si.

Finalizando con un poco de Introducción, definición y algunas características de Python voy a comentar brevemente los Dos entornos de Programación con los que cuenta.

El *primero* es cuando escribimos líneas de código en el interprete, y donde obtenemos una respuesta por cada línea, y por otra parte, podemos escribir el código de un programa en un archivo de texto, guardarlo y ejecutarlo.

Siguiendo con el tema voy a hablar ahora de como instalar Python, tiene una forma sencilla y a la vez muy clara a la hora de entenderla.

INSTALACIÓN

Comenzando con que existen varias implementaciones distintas de Python: Cpython, Jython, IronPython, PyPy, entre otras.

Donde *Cpython* es la mas rápida, la mas estable y por ende, la mas utilizada. Es la implementación que nos referimos gralmente los usuarios cuando hablamos de Python y donde los interpretes y módulos están escritos íntegramente en C.

Jpython es la implementación de Java en Python, mientras que IronPython es su contrapartida en C# (.NET).

Y por ultimo PyPy, es la implementación en Python de Python.

Cpython esta instalado por defecto en casi todas las distribuciones Linux y en las ultimas versiones de Mac Os. Para comprobar si esta instalado, abris una terminal y escribis Python, si esta instalado se va a iniciar la consola interactiva de Python y vamos a obtener los siguiente:

```
Python 2.5.4 (r254:67916, Feb 17 2009, 20:16:45)
[GCC 4.3.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

La primera línea nos va a indicar la versión de Python que tenemos instalada. Al final podemos ver el prompt (>>>), que nos indica que el intérprete está esperando que el usuario ingrese código. Podemos salir escribiendo *exit()* o pulsando Control + D.

Si no te indica nada parecido al tipear Python, no hay que preocuparse, es muy sencillo instalarlo. Puedes descargar la versión correspondiente a tu sistema operativo desde la web de Python, en <http://www.python.org/download/> o sino en Linux usando la herramienta de gestión de paquetes de tu distribución. También existen instaladores para Windows y Mac OS.

CONFIGURACIONES BÁSICAS PARA COMENZAR A PROGRAMAR

Es aconsejable instalar y utilizar iPython a la hora de empezar con una sesión interactiva, en lugar de la consola de Python. La encontramos en la web <http://ipython.scipy.org/>. Tiene características muy interesantes, como ser el *auto-completado* o el *operador* ?.

La función de *autocompletado* se ejecuta pulsando la tecla Tabulador. Por ejemplo, si escribimos *fi* y presionamos Tab nos va a mostrar una lista de las palabras que comienzan con *fi* (file, filter y finally). Si escribimos *file.* y pulsamos Tab nos va a mostrar una lista de los métodos y propiedades del objeto *file*.

La función *operador* nos muestra información sobre los objetos. Y se utiliza añadiendo el símbolo de interrogación al final del nombre del objeto del cual queremos más información. Por ejemplo,

```
In [3]: str?

Type: type
Base Class:
String Form:
Namespace: Python builtin
Docstring:
str(object) -> string
Return a nice string representation of the object.
If the argument is a string, the return value is the same object.
```

Nombrando algunas otras configuraciones básicas a tener en cuenta, como ser las entradas de datos, las cuales se distinguen por la presencia del indicador ('>>>'). Y las líneas que no empiezan por un indicador son la salida del intérprete.

Los comentarios en Python empiezan con carácter '#' y se extienden hasta el final de la línea. El comentario se puede iniciar al principio de una línea o tras un espacio en blanco o código.

```
# éste es un comentario

palabra = 1 # y éste también

cadena = "# Esto no es un comentario."
```

Podemos encontrar también otra funcionalidad interesante que es la de usar Python como una calculadora. Se teclea una expresión y el muestra un resultado. Y se puede usar paréntesis para agrupar operaciones, como vemos en el ejemplo:

```
>>> 1+1
2
>>> # Esto es un comentario
... 2+2
4
>>> 2+2 # Esto es un comentario junto al código
4
>>> (15-5*6)/4
15
>>> # La división entera redondea hacia abajo:
... 7/3
2
>>> 7/-3
-3
```

De igual manera que en C, se usa el signo de igualdad '=' para asignar un valor a una variable. El valor de una asignación no se escribe:

```
>>> alto = 20
>>> ancho = 5*9
>>> alto * ancho
900
También se puede asignar un valor simultáneamente a varias variables:
>>> x = y = z = 0 # Poner a cero 'x', 'y' y 'z'
>>> x
0
>>> y
0
>>> z
0
```

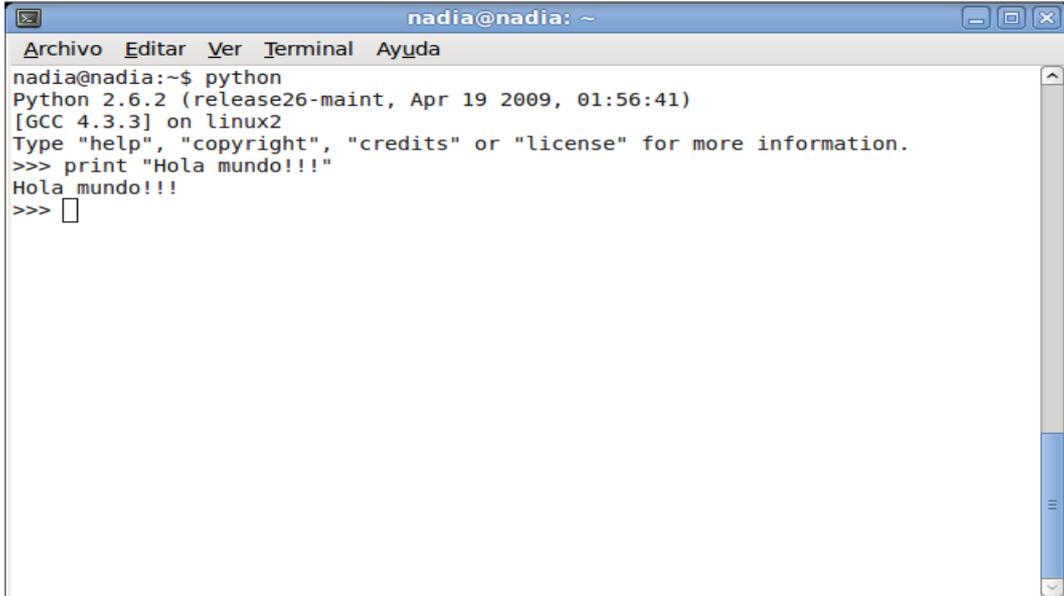
Y muchas funciones que van a descubrir ustedes mismos a la hora de recorrer con mas profundidad este lenguaje.

Como ultima instancia de esta presentación de Python, vamos a escribir el primer programa.

Primeros pasos ('Hola Mundo!')

El primer programa que vamos a escribir en Python es el clásico Hola Mundo y en este lenguaje es tan simple como:

```
print "Hola Mundo"
```



```
nadia@nadia: ~  
Archivo Editar Ver Terminal Ayuda  
nadia@nadia:~$ python  
Python 2.6.2 (release26-maint, Apr 19 2009, 01:56:41)  
[GCC 4.3.3] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print "Hola mundo!!!"  
Hola mundo!!!  
>>> □
```

Primero lo vamos a probar en el interprete. Ejecuta Python o iPython según lo prefieran, escribimos la línea mencionada antes y apretamos Enter. El interprete va a responder mostrando en la consola el texto *Hola Mundo*.

Lo que vamos a hacer ahora es crear un archivo de texto con el código anterior. Abrimos el editor de texto (el que uds prefieran) y copiamos la línea anterior, por ultimo guardamos el archivo como *hola.py* por ejemplo.

Ahora ejecutamos el programa, indicándole el nombre del archivo que queremos abrir, así de sencillo:

```
Python hola.py
```

Si usamos Windows los archivos *.py* ya van a estar asociados al interprete de Python, por lo que al hacer doble click sobre el archivo ya estamos ejecutando el programa. Sin embargo, como este programa no hace mas que imprimir un texto en la consola, la ejecución es demasiado rápida para poder verla. Para remediarlo, lo que vamos a hacer es añadir una nueva línea que espere la entrada de datos por parte del usuario.

```
print "Hola Mundo"  
raw_input()
```

De esta forma, se va a mostrar una consola con el texto *Hola Mundo* hasta que presionemos Enter.

Si usamos Linux (u otro Unix) para abrir el archivo *.py* con el interprete adecuado, es necesario añadir una nueva línea al principio del archivo:

```
#!/usr/bin/Python
sprint "Hola Mundo"
rae_input()
```

A esta línea se la conoce en el mundo Unix como *deshebrar*, hasaní o charango. Los dos caracteres `#!` indican al sistema operativo que dicho inscripto se debe ejecutar utilizando el intérprete especificado a continuación. De esto se desprende, que si esta no es la ruta en la que está instalado nuestro intérprete de Python, es necesario cambiarla.

Por supuesto además de añadir el deshebrar, vamos a tener que dar permisos de ejecución al programa.

```
cusrdbinhola.py
```

Y listo, si hacemos doble clic el programa se ejecutará, mostrando una consola con el texto *Hola Mundo*, como en el caso de Windows.

También podríamos correr el programa desde la consola como si tratara de un ejecutable cualquiera:

```
./hola.py
```

De ahora en mas esta en uds introducirse a todas las funcionalidades que posee Python y comenzar a programar en este fabuloso lenguaje de programación del software libre.

LENGUAJE "JAVA"



HISTORIA

Para apreciar el significado e importancia de Java, es muy importante conocer su lugar de origen y cuales fueron sus propósitos. En abril de 1991, JAVA fue creado por Sun Microsystems, como parte de un proyecto de investigación para desarrollar software para

comunicación entre aparatos electrónicos de consumo. Durante la investigación surgió el problema de que cada aparato tenía un microprocesador diferente y poco espacio de memoria; el equipo decidió introducir sistemas de software con aplicaciones para consumidores smart como plataforma de lanzamiento para su proyecto. James Gosling escribió el compilador original y lo denominó "Oak", y con la ayuda de los otros miembros del equipo desarrollaron un decodificador que más tarde se convertiría en lenguaje Java.

Al examinar las dinámicas de Internet, lo realizado por el equipo se adecuaba a este nuevo ambiente ya que cumplía con los mismos requerimientos de las set-top box OS que estaban diseñadas con un código de plataforma independiente pero sin dejar de ser pequeñas y confiables.

Patrick Naughton procedió a la construcción del lenguaje de programación Java que se accionaba con un browser prototipo, más tarde se le fueron incorporando algunas mejoras y el browser Hot Java fue dado a conocer al mundo en 1995.

Una de las características más atractivas del Hot Java fue su soporte para los "applets", que son las partes del código Java que pueden ser cargadas mediante una red de trabajo para después ejecutarlo localmente y así lograr o alcanzar soluciones dinámicas en computación acordes al rápido crecimiento del ambiente Web.

Para dedicarse al desarrollo de productos basados en la tecnología Java, Sun formó la empresa Java Soft en enero de 1996, de esta forma se dio continuidad al fortalecimiento del programa del lenguaje Java y así trabajar con terceras partes para crear aplicaciones, herramientas, sistemas de plataforma y servicios para aumentar las capacidades del lenguaje.

Durante ese mismo mes, Java Soft dio a conocer el Java Development Kit (JDK) 1.0, una rudimentaria colección de componentes básicos para ayudar a los usuarios de software a construir aplicaciones de Java. Dicha colección incluía el compilador Java, un visualizador de applets, un debugger prototipo y una máquina virtual Java (JVM), necesaria para correr programas basados en Java, también incluía paquetería básica de gráficos, sonido, animación y trabajo en red.

Asimismo el Netscape Communications Inc, mostró las ventajas de Java y rápidamente se asoció con Java Soft para explotar su nueva tecnología. No pasó mucho tiempo antes de que Netscape Communications decidiera apoyar a los Java applets en Netscape Navigator 2.0. Este fue el factor clave que lanzó a Java a ser reconocido y famoso, y que a su vez forzó a otros vendedores para apoyar el soporte de applets en Java.

Como parte de su estrategia de crecimiento mundial y para favorecer la promoción de su nueva tecnología, Java Soft otorgó permisos a otras compañías para que pudieran tener acceso al código fuente de Java y al mismo tiempo mejorar sus navegadores, dicha licencia también les permitía crear herramientas de desarrollo para programación Java y los facultaba para acondicionar Máquinas Virtuales Java (JVM), a varios sistemas operativos.

Muy pronto las licencias o permisos contemplaban a prestigeadas firmas como IBM, Microsoft, Symantec, Silicon Graphics, Oracle, Toshiba y por supuesto Novell.

Desde su aparición, Java se ha ganado una impresionante cantidad de apoyo. Virtualmente cada vendedor importante de software ha obtenido autorización de Java y ahora ha sido incorporado en los principales sistemas operativos base de PC's de escritorio hasta estaciones de trabajo UNIX.

<http://distritos.telepolis.com/java/lib/documentos/historia.htm>

CARACTERÍSTICAS

Java es un lenguaje de alto nivel con el que se puede escribir tanto programas

convencionales como para Internet.

Una de las ventajas más significativas es su independencia de plataforma. En caso de que tengan que ejecutarse en sistemas diferentes.

Otra característica importante de Java es que es un lenguaje de programación orientado a objetos (POO).

Además de ser portable y orientado a objetos, Java es un lenguaje fácil de aprender. Tiene un tamaño pequeño que favorece el desarrollo y reduce las posibilidades de cometer errores; a la vez es potente y flexible.

ENTORNO DE PROGRAMACIÓN E INSTALACIÓN

Lo primero que tenemos que decidir es qué "edición" de java necesitamos. Hay tres disponibles: **J2ME**, **J2SE** y **J2EE**

J2ME es la versión "Micro". Es una versión "reducida" de java para aparatos pequeños. Los más típicos son los teléfonos móviles.

J2SE es la versión "Standard". Esta es la versión para las aplicaciones normales que pueden correr en un PC. Normalmente, esta es la versión que debemos descargar.

J2EE es la versión "Enterprise". Esta versión es para el desarrollo de aplicaciones web. Es útil para gente que programa en servidores web y hacen páginas web complejas, con accesos a bases de datos, etc.

EL J2SE

Suponemos que nos hemos decidido por el **J2SE**. Dentro tenemos varias opciones para descargarnos. Vamos a verlas:

JRE Java Runtime Environment, o Entorno en Tiempo de Ejecución de Java es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java. El usuario final usa el JRE como parte de paquetes software o plugins (conectores) en un navegador Web. *Sun* ofrece también el SDK de Java 2, o JDK (Java Development Kit) en cuyo seno reside el JRE, e incluye herramientas como el compilador de Java, Javadoc para generar documentación o el depurador. Puede también obtenerse como un paquete independiente, y puede considerarse como el entorno necesario para ejecutar una aplicación Java, mientras que un desarrollador debe además contar con otras facilidades que ofrece el JDK.

SDK es el entorno de desarrollo. Aquí está incluido el compilador de java, el debugger y otras herramientas. Esta es la opción adecuada para los que quieren programar en java sus propias aplicaciones.

SDK + netbeans además incluye **netbeans**, un entorno visual de desarrollo.

UN ENTORNO VISUAL DE DESARROLLO

El **SDK** nos proporciona las herramientas necesarias para hacer nuestros programas, pero todas estas herramientas son a base de comandos. Esto quiere decir que nuestro programa java debemos escribirlo con un editor de nuestro sistema operativo (**vi o nano** en Linux).

El **netbeans** es un entorno visual de desarrollo. Cuando arrancamos **netbeans**, nos aparece un árbol con todas las clases de nuestra aplicación o proyecto (por supuesto, la primera vez que lo arranquemos saldrá vacío), un editor en el que podemos escribir nuestro programa, con opciones para ejecutar, para depurar, etc.

El desarrollo siempre es más fácil con una herramienta de este tipo, sin embargo su gran ventaja es también su gran inconveniente: Nos solucionan automáticamente muchos de los trabajos que haríamos a mano de no tenerla. Esto hace que haya cosas que no aprendamos. Es bastante habitual que programadores acostumbrados a estas herramientas no sepan luego compilar o incluso ejecutar un programa java fuera de ella.

Netbeans tiene su propia página <http://www.netbeans.org> en la que puedes descargar el mismo **netbeans** (es gratuito) y módulos adicionales para el mismo (algunos gratuitos y otros no).

Hay otros entornos de desarrollo visuales gratuitos, como **eclipse**, que pueden descargarse de

<http://www.eclipse.org>.

NetBeans

Se refiere a una plataforma para el desarrollo de aplicaciones de escritorio usando Java y a un entorno de desarrollo integrado (IDE) desarrollado usando la Plataforma NetBeans.

Ésta plataforma permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados *módulos*. Un módulo es un archivo Java que contiene clases escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

Sun Microsystems fundó el proyecto de código abierto NetBeans en Junio de 2000 y continúa siendo el patrocinador principal de los proyectos.

La Plataforma NetBeans

Durante el desarrollo del NetBeans IDE ocurrió una cosa interesante. La gente empezó a construir aplicaciones usando el NetBeans core runtime con sus propios plug-ins, de hecho, esto se convirtió en un mercado bastante grande.

La Plataforma NetBeans es una base modular y extensible usada como una estructura de integración para crear aplicaciones de escritorio grandes. Empresas independientes asociadas, especializadas en desarrollo de software, proporcionan extensiones adicionales que se integran fácilmente en la plataforma y que pueden también utilizarse para desarrollar sus propias herramientas y soluciones.

La plataforma ofrece servicios comunes a las aplicaciones de escritorio, permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación. Entre las características de la plataforma están:

- Administración de las interfaces de usuario (ej. menús y barras de herramientas)
- Administración de las configuraciones del usuario
- Administración del almacenamiento (guardando y cargando cualquier tipo de dato)
- Administración de ventanas
- Framework basado en asistentes (diálogos paso a paso)

Instalar NetBeans en debian

A través del apt

su

apt-get update

apt-get install netbeans-ide

Esto nos pedirá además instalar netbeans-platform.

Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent Azureus.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Es ahora desarrollado por la Fundación Eclipse, una organización independiente sin fines de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

La versión actual de Eclipse dispone de las siguientes características:

- * Editor de texto
- * Resaltado de sintaxis
- * Compilación en tiempo real
- * Pruebas unitarias con JUnit
- * Control de versiones con CVS
- * Integración con Ant
- * Asistentes (wizards): para creación de proyectos, clases, tests, etc.
- * Refactorización

INSTALACIÓN DE LOS PAQUETES DEBIAN NECESARIOS

Instalación.

```
usuario:~# apt-get install eclipse-jdt
usuario:~# apt-get install eclipse eclipse-rls
```

Solo queda lanzar el entorno:

```
usr@usuario:~$ eclipse &
```

```
Sun-java5-jdk
```

Pero como ya hemos mencionado con anterioridad para aquellos que quieren aprender a programar en java, es mejor que lo hagan en un entorno de desarrollo, como lo es Sun-java5-jdk. Aquí está incluido el compilador de java, el debugger y otras herramientas.

Recordemos que este trabajo se desarrolla en entorno a GNU/Linux Debian Lenny.

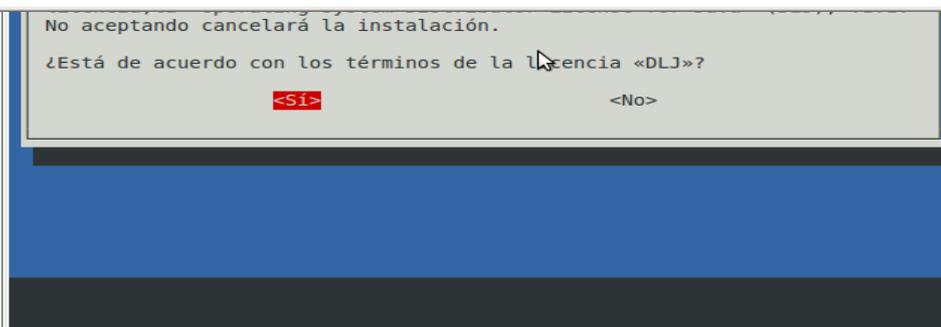
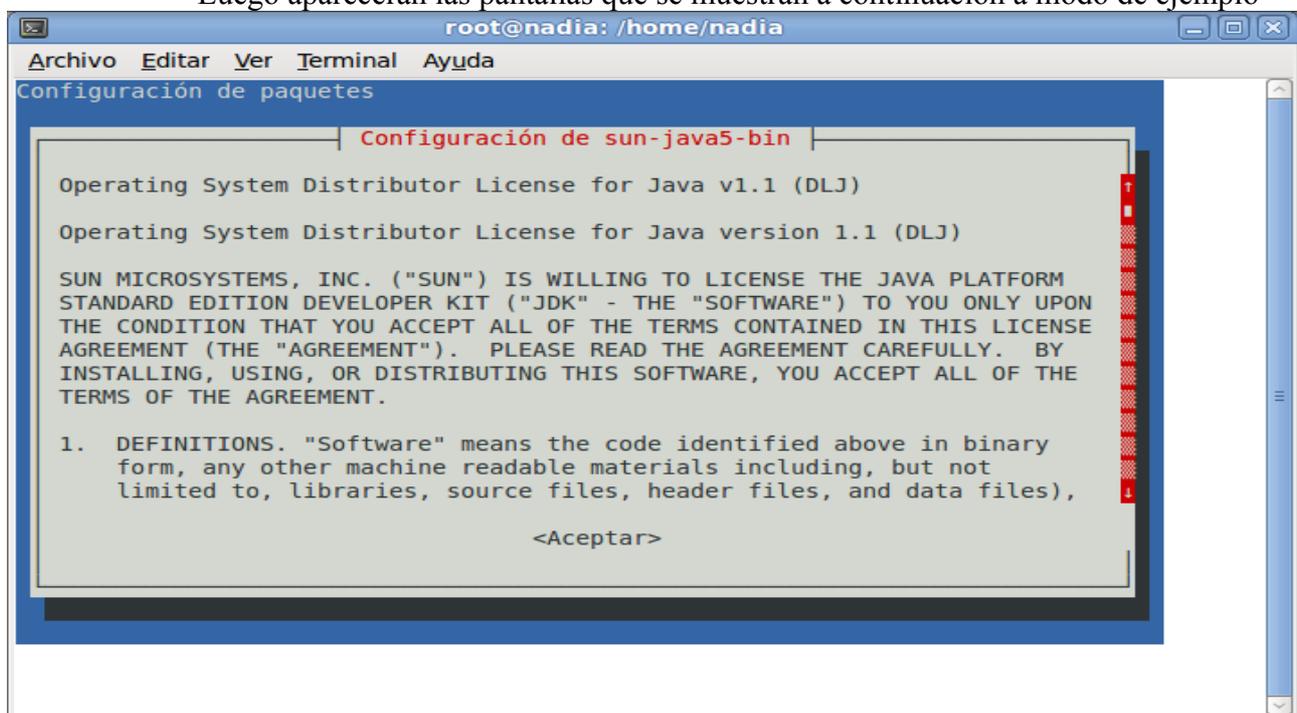
Esta versión dispone de OpenJDK, el compilador Java de GNU, el intérprete de bytecodes Java de GNU, Classpath, y otras versiones libre de la tecnología Java de Sun hace posible la distribución de aplicaciones basadas en Java dentro del repositorio principal (main) de Debian.

El Kit de Desarrollo de Java de Sun está disponible como paquete Debian . El KDJ (JDK en inglés) permite ejecutar programas en Java y "applets", y escribirlos. Si el núcleo instalado está correctamente configurado, el KDJ permitirá ejecutar programas en Java igual que otro tipo de ejecutables. El KDJ también incluye varios programas de demostración.

Para la instalación de dicho entorno basta con hacer las siguientes instrucciones en el Shell:

```
apt-get install sun-java5-jdk
```

Luego aparecerán las pantallas que se muestran a continuación a modo de ejemplo



Una vez terminada la instalación, podemos empezar con nuestro primer programa “Hola mundo!!!”

CREAMOS EL PROGRAMA

Lo primero que hacemos, es crear un archivo de texto con el editor **nano**. Atención a las mayúsculas y minúsculas en el nombre del fichero y el nombre de la clase.



```

nadia@nadia: ~
Archivo Editar Ver Terminal Ayuda
GNU nano 2.0.9 Fichero: hola.c

#include <stdio.h>

main ()
{
    printf("Hola Mundo\n");
}
[ Leer 8 líneas ]
^G Ver ayuda ^O Guardar ^R Leer Fich ^Y Pág Ant ^K Cortar Tex ^C Pos actual
^X Salir ^J Justificar ^W Dónde Está ^V Pág Sig ^U PegarTxt ^T Ortografía
  
```

```
$ nano HolaMundo.java
```

Ahora escribimos nuestro programa

```

public class HolaMundo
{
    public static void main (String [ ] args)
    {
        System.out.println ("Hola mundo");
    }
}
  
```

Salvamos el programa y salimos del editor.

LA CLASE HolaMundo

Veamos qué es lo que hemos escrito:

Hemos hecho una clase que hemos llamado **HolaMundo**.

En java es habitual meter cada clase en un fichero distinto. **Es obligatorio que la clase se llame igual que el fichero. Importan las mayúsculas y las minúsculas**, así que no es lo

mismo HolaMundo que holaMundo ni que Holamundo. Como nuestra clase se llama **HolaMundo**, debemos llamar a nuestro fichero **HolaMundo.java**.

Si hubiera varias clases en el mismo fichero, java sólo permite que se pueda usar una de ellas desde código que esté fuera del fichero. Esa clase que se puede ver desde todos lados debe llevar el modificador **public**. Además, la clase **public** debe ser obligatoriamente la que se llama igual que el fichero.

En resumen, si hacemos un fichero **HolaMundo.java**, debemos meter dentro obligatoriamente y para que todo vaya bien una clase pública que se llame **HolaMundo**. Eso es lo que tenemos en la primera línea.

```
public class HolaMundo
```

¿Desde dónde hasta dónde va la clase?. En java lo marcamos con una llaves.

Inmediatamente después del nombre de la clase, abrimos unas llaves. Luego hacemos el código de nuestra clase y finalmente la cerramos.

```
public class HolaMundo
```

```
{
...
}
```

EL MÉTODO MAIN

Dentro de la clase podemos poner código de muchas maneras distintas. Una forma habitual es hacer "**métodos**". En java hay un nombre especial de método que es **main**. Cuando a java le decimos que debe ejecutar el código de una clase, busca en dicha clase el método con nombre **main** y es el método que ejecuta.

Para que java entienda el método **main**, este debe estar declarado de una forma muy concreta, que es la siguiente

```
public static void main (String [ ] args)
```

```
{
...
}
```

Debe ser **public**. Eso quiere decir que se le puede llamar desde cualquier lado.

Debe ser **static**. Esto quiere decir que se le puede llamar sin necesidad de instanciar

la clase

Es **void**. Eso quiere decir que ese método no devuelve ningún resultado.

El parámetro del método, que va entre paréntesis detrás de él, es un array de String (cadenas de texto). Como no lo vamos a usar, de momento lo ignoramos y no detallo exáctamente qué quiere decir eso. Baste saber que hay que ponerlo siempre que hagamos un método main en una clase.

SACAR TEXTO POR LA PANTALLA

En java tenemos disponible la clase **System** que está accesible directamente en cualquier sitio, es decir, podemos usarla sin hacer nada especial siempre que queramos. Esta clase contiene cosas relacionadas con nuestro sistema, con nuestro ordenador. Dentro, entre otras cosas, tiene un atributo **out** que es público. Al ser público podemos acceder a él libremente. Para acceder a este atributo debemos indicar que está dentro de System. Eso se hace poniendo **System.out**.

Este atributo **System.out** esta ligado a la pantalla y es otra clase que a su vez tiene métodos. Tiene método para poder enviarle cadenas de caracteres que saldrán en pantalla. Los dos

métodos más usados son **print()** y **println()**. Ambos sacan el texto que pongamos entre los paréntesis por pantalla, sólo que el segundo añade además un "nueva línea", de forma que lo siguiente que escribamos saldrá en una línea nueva. Si más adelante queremos escribir en la misma línea, debemos usar **print()**.

El código dentro del **main** será entonces

```
System.out.println ("Hola Mundo");
COMPILAMOS EL PROGRAMA
```

Para compilar usamos el programa **javac**. Dicho programa está en el subdirectorio **bin** de donde hayamos instalado java. Es importante para que todo vaya bien que ese directorio **bin** esté en el **path** de búsqueda de ejecutables.

Para comprobar si está, probamos a ejecutarlo

```
$ javac
```

Si obtenemos un error del estilo "comando no se reconoce" o "command not found" o similar, es que **javac** no está en la variable de entorno **PATH** y debemos ponerlo.

```
$ PATH=$PATH:/directorio_java/bin
```

Si la ejecutar **javac** obtenemos un mensaje de "Usage: javac" y la lista de opciones, es que todo está correcto y podemos compilar nuestro programa.

Para compilar el programa, estando en el mismo directorio en el que está el fichero **HolaMundo.java** escribimos

```
javac HolaMundo.java
```

Si todo va bien, no obtendremos ninguna salida de error y habrá aparecido un fichero **HolaMundo.class**. Este fichero es nuestro programa compilado y lo que se puede ejecutar.

Si obtienes algún error, es que hay algo mal en algún sitio. Revisa que has escrito bien el comando **javac** y el nombre del fichero. Revisa también que tu fichero contiene lo mismo (mayúsculas y minúsculas incluidas) que el de este ejemplo.

EJECUTAR EL PROGRAMA

Una vez que tenemos **HolaMundo.class**, podemos ejecutarlo y ver el resultado. Para ejecutar este **HolaMundo.class** necesitamos un programa llamado **java** que está también en el **bin** de donde tengamos instalado java. Si hemos conseguido compilarlo, es que este directorio está en el **PATH** y no deberíamos tener problemas.

Si tienes un java más o menos moderno, bastará con hacer esto estando situados en el directorio donde esté el fichero **HolaMundo.class**

```
$ java HolaMundo
```

```
Hola Mundo
```

El comando **java** admite como parámetro el nombre de la clase, no el del fichero. La clase se llama **HolaMundo**, el fichero se llama **HolaMundo.class**. Por eso NO HAY QUE PONER .class AL EJECUTAR. Este despiste suele ser habitual en la gente que empieza y llega a dar bastantes quebraderos de cabeza hasta que se cae en la cuenta.

Si tienes un java más antiguo, es posible que no te funcione. Si obtienes un error del estilo "Class Not Found", verifica que estás en el mismo directorio que **HolaMundo.class** y que has escrito bien las mayúsculas y minúsculas. Si todo está bien y te sigue fallando, no queda más remedio que liarnos con el **CLASSPATH**...

CLASSPATH

java busca los ficheros `.class` en determinados sitios. Las versiones más antiguas de java los buscan en el directorio en el que está instalado java. Las versiones más modernas los buscan en los mismos directorios y en el directorio actual. Si estás con un java moderno, java encontrará tu fichero **HolaMundo.class** y no tendrás problemas. Si estás con un java antiguo, entonces no lo encontrará y te dará el error de que no encuentra la clase HolaMundo.

Afortunadamente, **java** se puede configurar para que busque clases en otros directorios de los de defecto. Hay dos formas de hacerlo.

VARIABLE DE ENTORNO CLASSPATH

Una es por medio por la variable de entorno **CLASSPATH**. Simplemente debemos definir la variable diciéndole en qué directorios están los ficheros `.class`.

Podemos dar este path **relativo** o **absoluto**.

```
$ CLASSPATH=.
```

```
$ CLASSPATH=/home/usuario/mi_proyecto_HolaMundo
```

```
$ export CLASSPATH
```

En Linux suele ser necesario poner "**export CLASSPATH**" después de definir la variable. En Linux, cuando se define una variable nueva por primera vez, si queremos que los demás procesos (por ejemplo java cuando lo ejecutemos) la vean, es necesario ejecutar un **export**.

Una vez hecho esto, deberíamos poder ejecutar el programa **HolaMundo** sin problemas.

OPCIÓN DEL COMANDO JAVA

La otra opción para definir el **CLASSPATH** es hacerlo como opción en la línea de comandos de java. Se haría así

```
$ java -cp . HolaMundo
```

```
$ java -cp /home/usuario/mi_proyecto_HolaMundo HolaMundo
```

Si usas esta opción ("-cp"), se ignora el valor de la variable **CLASSPATH**, así que por norma general y para evitar errores, elige qué forma te gusta más y usa sólo esa.

LENGUAJE "RUBY"



DEFINICIÓN e INTRODUCCIÓN

Ruby es un lenguaje de programación de alto nivel, interpretado, reflexivo y orientado a objetos. También llamado *lenguaje de script*.

Este lenguaje corre o se ejecuta sobre un *intérprete*, éste es un programa que ejecuta órdenes de *alto nivel* y las traduce a otro lenguaje de más bajo nivel para ejecutarlas, como por ejemplo C. Esto hace mas fácil y rápida la programación.

Cabe la diferenciación entre lenguajes compilados y lenguajes interpretados, los primeros usan un programa compilador que crea un ejecutable y traduce las sentencias de alto nivel a lenguaje máquina para su rápida ejecución, a diferencia de éstos los lenguajes interpretados traducen una a una las líneas de código y generalmente no guardan un estado anterior, haciéndolos unas 10 veces más lentos que los anteriores. Resumiendo, en los compilados escribís, guardas, compilas y ejecutas; en los interpretados escribís, guardas y ejecutas.

La *reflexión* es la capacidad que tiene el lenguaje de modificar en tiempo de ejecución su estructura de alto nivel, como por ejemplo evaluar una cadena de caracteres como si fuera código fuente en tiempo de ejecución.

Ruby es totalmente *orientado a objetos* (POO o Programación Orientada a Objetos), esto quiere decir que todo en él es un objeto; las clases y también los datos primitivos como los enteros, flotantes, booleanos, string, etc.

Ruby también se caracteriza, además de ser libre, por ser muy flexible. Lo queremos decir con esto, es que este lenguaje nos permite cambiar a gusto y placer sus partes más esenciales. Por ejemplo, estoy haciendo un programa en el que ocupamos muchas veces el + pero como tenemos el teclado en inglés en nuestra notebook y nos cuesta muchísimo presionar la tecla shift para hacer el +, podemos 'decirle' a Ruby que a partir de ahora usaremos la palabra *suma* en vez de el signo +, y de esa forma vamos a poder programar más a gusto y rápido:

```
class Numeric
  def suma(x)
    self.+(x)
  end
end
```

```
X = 5.suma 6
```

y ahora *X* vale 11.

Otras características más específicas de Ruby son:

- No hace falta declarar las variables.
- Hace llamadas directas al SO (Sistema Operativo).
- Tiene soporte para manejo de excepciones.
- Es altamente portable; capás de correr en **GNU/Linux**, varios tipos de UNIX, Windows 9x/NT, MacOS X, DOS, etc.
- Posee carga dinámica de bibliotecas, si lo permite el sistema operativo.

Además de ser gratuito, Ruby es completamente libre para utilizarlo, copiarlo,

modificarlo y distribuirlo. Esta licenciado bajo dos licencias inseparables libres y de código abierto: GPL y Licencia Ruby.

Fue creado por un programador japonés llamado **Yukihiro Matsumoto** en 1993 y presentado en 1995. “*Matz*” para los amigos, después de muchos años de programación orientada a objetos y de no lograr programar a gusto, decidió crear su propio lenguaje. Buscando un lenguaje más poderoso que Perl, más orientado a objetos que Python, y que a su vez sea de fácil escritura, rápida programación, y aún más fácil su lectura; sin dejar de '*divertirse*' al programar.



INSTALACIÓN

A partir de este momento pasaremos a explicar paso por paso la instalación:

Abrimos una consola y entramos como superusuario o root y allí escribimos lo siguiente:

```
# apt-get install ruby irb rdoc
```

Y listo, eso es todo para instalar Ruby.

Nota: Podríamos descargar el código fuente y compilarlo desde la siguiente dirección <ftp://ftp.ruby-lang.org/pub/ruby/1.9/ruby-1.9.1-p0.tar.gz>, pero se recomienda usar nuestro gestor de paquetes, ya que éste nos resuelve todo tipo de dependencias.

La *IRB* (*Interactive Ruby Shell*), es la *Consola Interactiva de Ruby*, en la cual podemos ir probando nuestras sentencias antes de plasmarlas en un archivo definitivo de nuestro programa.

En la *IRB* es donde podemos 'jugar' con las sentencias de Ruby para poder aprenderlas bien y no tener que ejecutar el archivo cada vez que modifiquemos algo de nuestro programa.

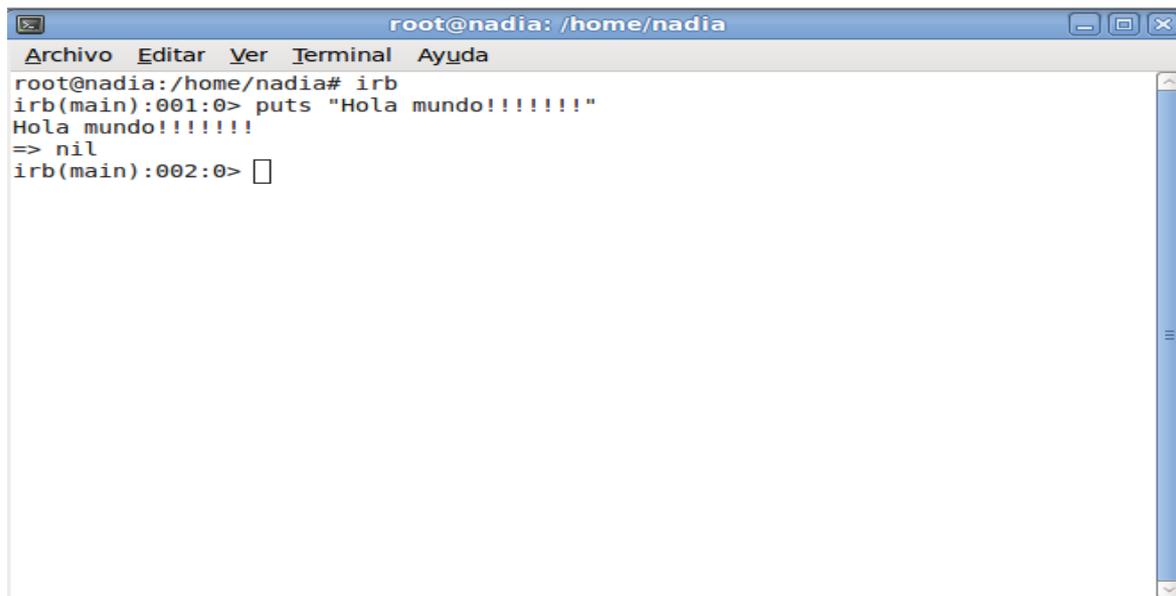
Para utilizar la *IRB* sólo basta con abrir una consola y ejecutar la siguiente línea:

```
# irb
```

y cuando queremos salir:

```
# exit
```

PRIMEROS PASOS EN RUBY



```
root@nadia: /home/nadia
Archivo Editar Ver Terminal Ayuda
root@nadia:/home/nadia# irb
irb(main):001:0> puts "Hola mundo!!!!!!!!"
Hola mundo!!!!!!!!
=> nil
irb(main):002:0> □
```

Para empezar a probar Ruby tenemos varias opciones:

– Podemos 'decirle' a Ruby que ejecute una línea como si lo ejecutara la *IRB*, anteponiendo a la línea que queremos ejecutar la palabra *ruby* y el parámetro *-e* y posteriormente entre comillas simples (' ') la línea a ejecutar propiamente dicha:

```
# ruby -e 'puts "Mi primera línea en Ruby"'
```

– También podemos crear un archivo con extensión *.rb* con cualquier editor de texto y lo ejecutamos en una consola anteponiendo la palabra *ruby* al nombre del archivo:

```
# ruby primerArchivoRuby.rb
```

– O la más recomendable, la más fácil y rápida que es hacerlo en la *IRB*. Para hacer esto sólo tenemos que abrir una consola y tipear *irb* y allí ejecutamos todos los comandos o partes de código que deseamos probar:

```
# irb(main):001:0> puts 'Hola mundo Ruby'
Hola mundo Ruby
=> nil
```

Nota: Si a pesar de todo esto no te animas a instalarlo en tu pc, Ruby te da otra opción. Si contás con una conexión de internet, puedes ejecutar una *Consola Interactiva de Ruby* en línea sólo con poner en la barra de tu navegador la siguiente dirección y allí puedes probar este gran lenguaje:

<http://tryruby.hobix.com/>

Try Ruby!

a hands-on tutorial

```
Interactive ruby ready.
>>
```

Got 15 minutes? Give Ruby a shot right now!

Ruby is a programming language from Japan (available at ruby-lang.org) which is revolutionizing the web. The beauty of Ruby is found in its balance between simplicity and power.

Try out Ruby code in the prompt above. In addition to Ruby's builtin methods, the following commands are available:

- help** Start the 15 minute interactive tutorial. Trust me, it's very basic!
- help 2** Hop to chapter two.
- clear** Clear screen. Useful if your browser starts slowing down. Your command history will be remembered.
- back** Go back one screen in the tutorial.
- reset** Reset the interpreter if you get too deep. (or **Ctrl-D**!)
- time** A stopwatch. Prints the time your session has been open.

If you happen to leave or refresh the page, your session will still be here for unless it is left inactive for ten minutes.

Trapped in double dots? A quote or something was left open. Type: reset or hit Ctrl-D.

SINTAXIS BÁSICA

Numérica:

uma	S	n):002:0>	<pre style="font-family: monospace; font-size: small;">irb(mai > 3+4 => 7</pre>
esta	R	n):003:0>	<pre style="font-family: monospace; font-size: small;">irb(mai > 9-3 => 6</pre>
roducto	P	n):004:0>	<pre style="font-family: monospace; font-size: small;">irb(mai > 4*5 => 20</pre>
otencia	P	n):005:0>	<pre style="font-family: monospace; font-size: small;">irb(mai > 3**2 => 9</pre>

División Entera (Módulo)	R E D	irb(main):006:0> 5%2 => 1
División Entera	D	irb(main):007:0> 5/2 => 2

Cadenas:

Impresión en Pantalla	Mundo Ruby'	irb(main):001:0> puts 'Hola Mundo Ruby' Hola Mundo Ruby
Concatenación	Mundo '+'Ruby'	irb(main):002:0> "Hola Mundo '+'Ruby' => "Hola Mundo Ruby"
Repetición	Ruby"*3	irb(main):003:0> "Hola Ruby"*3 => "Hola RubyHola RubyHola Ruby"

CONCLUSIÓN

Al desarrollar este trabajo nos encontramos con distintas dificultades. Una de ellas fue el elegir los lenguajes adecuados desde nuestros puntos de vista. Para la elección nos basamos

en los más utilizados.

Si bien nosotros programamos, nunca hemos utilizado los mismos. Este trabajo nos dio la posibilidad de aprender y conocer aspectos de la programación que hasta el momento desconocíamos. Ha juzgar por lo que hicimos con estos lenguajes en el entorno del Sistema Operativo GNU/Linux descubrimos que es mucho más amigable y las herramientas disponibles son fáciles de instalar e implementar y muy accesibles. Entre estos lenguajes podemos destacar que los más fáciles de usar han sido Ruby y Python, por la sencillez de su sintaxis, funciones y su entorno de desarrollo. Con respecto a C nos dimos cuenta que a pesar de ser un poco mas complejo es muy útil, debido a que todos los demás lenguajes se basan en él.

El programar en JAVA nos resulto más complicado, por la estructura que utiliza, pero a pesar de ello; es un lenguaje que casi no posee límites en cuanto a lo que se puede desarrollar.

BIBLIOGRAFÍA

-<http://www.ruby-lang.org/es/>

-<http://www.softwarelibre.net>

-<http://www.maestrosdelweb.com/editorial/ruby/>

-<http://es.wikipedia.org/wiki/Ruby>

www.python.com.ar/

-es.wikipedia.org/wiki/Python

-*JAVA 2 – Francisco Javier Ceballos- Editorial RA-MA – Año 2000*

-*Aprenda JAVA como si estuviera en primero- Editorial TECNUN- Año 2000*

-*Como programar en Java- Deitel and Deitel- Editorial: Prentice Hall Hispanoamerica- Año: 2008*