Vim para Programadores

Jorge Emanuel Capurro

¿Programas en varios lenguajes? ¿Buscas un IDE que se adapte a tus necesidades?¿Quieres unificar todas las características distintas de cada lenguaje en un solo programa que te sea cómodo y eficiente de usar a la hora de programar? Si a todas estas preguntas todavía no le encontraste respuestas, este artículo te las va a dar. Esta vez, nos concentraremos en la forma de usar, configurar y personalizar el magnifico editor de texto Vim de modo que podamos programar en él bajo prácticamente cualquier lenguaje de programación sin tener que envidiarle nada a los grandes Entornos de Desarrollo Integrado (IDE). Veamos cómo hacerlo...

linux@software.com.pl

El editor vi es de uso obligado para cualquier Administrador de Sistemas, ya que es el único editor de texto que viene preinstalado en prácticamente todos los sistemas UNIX. Esto ha sido así siempre y seguramente lo seguirá siendo por mucho tiempo más. Desde la llegada de su versión mejorada llamada Vim (Vi Improved), este editor ha traspasado las barreras de ser solamente un "simple" editor de texto. Con sus mejoras, se han incluido características que jamás habían sido pensadas que podrían ser de suma utilidad para la gran mayoría de la gente que usa este editor a diario. Entre las prestaciones que se incluyeron, éstas son las principales:

- Autocompletado de texto,
- Navegación por Tags,
- Ventanas múltiples,
- Resaltado de sintaxis dependiente del lenguaje de programación,
- Comprensión de más de 200 sintaxis diferentes,
- Completado de comandos, palabras y nombres de ficheros,
- Reconocimiento de formatos de fichero y conversión entre los mismos,
- Historial de comandos ejecutados,
- Grabación y reproducción de macros,
- Guardado de la configuración entre sesiones,
- Plegado automático y manual de código,
- Uso de plugins (extensiones).

Como vemos, muchas de estas prestaciones son muy útiles para los programadores. Ente otra de las características interesantes para mencionar, es la de poder utilizar el modo "editar, compilar, y corregir". De la misma forma que los entornos de desarrollo integrados, Vim puede editar el código fuente además llamar a un compilador externo, e interpretar sus resultados. Muchos dicen que Vim es un "editor hecho por programadores para programadores" y evaluando todas sus características, no existen posibilidades en que estemos en desacuerdo con esta idea.

Como comentario adicional, Vim posee en sus últimas versiones la inclusión de la herramienta `Vimdiff` que fusiona el viejo y poderoso comando `diff` de UNIX con las

Figura 1. Logotipo del Editor Vim

ventajas de Vim para poder interpretar de una manera más amena las diferencias entre dos ficheros. Muchas de las características mencionadas aprenderemos a utilizarlas y configurarlas, en el caso que sea necesario, en el desarrollo de este artículo.

Estructura

La estructura del artículo se dividirá en dos partes, en función a la implementación de las características mencionadas. La división será: Básica y Avanzada. En primer lugar, empezaremos dando un bosquejo acerca de los comandos útiles para los programadores, cómo implementarlos y en qué situación. Entre los temas a tratar estarán el coloreado de sintaxis, indentación de código automática, trabajar con múltiples ficheros, etc.

Con respecto a la segunda parte del artículo, comprenderá el uso del editor Vim con plugins específicos que harán de la tarea del programador una tarea mucho más fácil y llevadera. En esta sección también configuraremos a Vim lo suficiente como para tratarlo como nuestro IDE universal.

Vale aclarar que la estructura del artículo comprende y está pensada para ser utilizada como una guía de aprendizaje y referencia al mismo tiempo, siendo útil en ambos casos.

Prerrequisitos

Para poder seguir este artículo sin ninguna dificultad, se asume que el lector ya está familiarizado con el uso de Vim. Es decir, que conoce sus modos de funcionamiento y sus comandos básicos. De igual modo, como el lector ha podido observar, se incluyen en el artículo tablas que indican las funcionalidades básicas de Vim, que sirven a modo de recordatorio, para que cuando quiera aplicar lo aprendido en este artículo, pueda hacerlo sin necesidad de recurrir a otras fuentes de información si es que no recuerda algún comando. Recuerde que Vim trae un muy buen tutorial de iniciación al uso del editor, llamado Vimtutor. Puede invocarlo abriendo la terminal y tecleando `Vimtutor es`.

También es requisito fundamental el poseer una conexión de internet, ya que necesitaremos descargar algunos plugins para poder aplicar lo visto en la sección "Avanzada" del artículo. De no poseerla, puede buscar alguna fuente de conexión a internet, descargar los archivos necesarios, y luego seguir con el recorrido del artículo.

Todo lo aplicado en este artículo ha sido probado con el Vim - Vi IMproved 7.2, usando el Sistema Operativo GNU/Linux Ubuntu 9.04. Es válido aclarar que todos los temas vistos aquí, pueden ser llevados sin ningún cambio a los editores Vim que sean ejecutados en otras plataformas, como por ejemplo, el editor "Vim for Windows".

Prestaciones

Una vez realizada la introducción y mencionadas todas las características de este magnífico

Tip: Abreviando Texto con Vim

Por supuesto, Vim tiene un comando que nos permite abreviar texto. Por ejemplo, si quisiéramos que cada vez que escribamos la palabra "Nombre" nos apareciera "Jorge Emanuel Capurro", solamente tendríamos que tipear el siguiente comando *:ab Nombre Jorge Emanuel Capurro.* ¿Útil no?

Jugando con Vim

Muchos dicen que al ser tan grande la curva de aprendizaje de Vim, se vuelve tedioso y aburrido de aprender. Es por ello que existen algunos juegos para Linux que pueden ayudarte a aprender Vim. Estos juegos te ayudarán principalmente con lo básico. Aunque no conozco ningún juego que te ayude con todos los comandos de vi, conozco algunos que te enseñarán a a moverte con el cursor por el documento, entre otras cosas. Uno de ellos es `NetHack`, un juego que te ayudará concretamente a hacerlo, es bastante completo y te puede entretener y divertir durante mucho tiempo. Se puede instalar empleando el comando `sudo apt-get install nethack-console`. Algunos otros son: rouge, moria, omega, worm, y snake.

Figura 2. Una eterna batalla: Vim vs Emacs

Comando	Acción
"vi archivo"	Edita el fichero en modo pantalla completa
"vi –r archivo"	Recupera la última versión guardada del fichero
"vi + archivo"	Edita el fichero y se situa en la última línea
"vi +numLinea archivo"	Edita el fichero y se sitúa en la línea indicada
"vi archivo1… archivoN"	Va editando todos los archivos especificados. Para saltar de uno a otro deberemos escribir en modo comando ":n". Con ":n" no guardamos las modificaciones.
"vi +/string archivo"	Edita el fichero situando el cursor en la primera ocurrencia del "string" indicado.

Figura 3. Comandos para la Apertura de Archivos

editor de texto, empecemos a aplicarlo a la práctica. En un principio, como comenté con anterioridad, veremos los comandos básicos aplicados a situaciones que nos ocurrirán cuando estemos programando, como así también, las configuraciones básicas para que nuestro editor sea "más decente" a la hora de utilizarlo para programar. ¡Comencemos!

Parte 1
Configuración Básica

Los temas a desarrollar en esta primera parte del artículo serán los que figuran a continuación:

- Coloreado de Sintaxis,
- Numerado de Líneas,
- Indentación Automática y Manual,
- Comandos Específicos para Programadores,
- Creación de Marcas,
- Autocompletado Manual,
- Aplicar Folding al Código,
- Uso del Explorador de Archivos,
- Trabajar con Múltiples Ficheros,
- Compilación desde Vim,
- Ejecución de Comandos de la shell.

Coloreado de Sintaxis

Desde la versión 5.0 (1998) Vim dispone de coloreado de sintaxis. En las distribuciones de Debian o derivadas como es Ubuntu, el coloreado de sintaxis en el editor Vim no se incluye. Por más que la queramos incluir con los comandos correspondientes, no surgirá ningún efecto. Estas distros traen un paquete de Vim reducido en características y utilidades, esto hace que afecte al tamaño y que sea mucho más reducido, por ello si queremos un Vim con color debemos desinstalar el Vim actual e instalar `vim-full` como vemos a continuación.

Primero desinstalamos el paquete correspondiente:

```
# apt-get remove vim-tiny
```

Luego, instalamos Vim con todas sus prestaciones:

```
# apt-get install vim-full
```

Una vez instalada la versión full de Vim, simplemente debemos indicarle que queremos activar el resaltado de sintaxis. Esto lo haremos mediante el comando `:syntax on`. Una vez activado el resaltado de sintaxis, éste estará habilitado por la sesión de Vim que tengamos en el momento. Si queremos que cada vez que abramos Vim con un archivo específico, automáticamente resalte las palabras reservadas según sea su tipo, debemos añadir `syntax on` en el archivo de configuración de Vim llamado `.vimrc`. Cómo configurar el archivo .vimrc lo veremos en detalle en la sección "Configurando el archivo .vimrc".

Como todo buen programador sabe, el coloreado de sintaxis es una herramienta fundamental para el trabajo diario. Vim soporta más de 300 lenguajes de programación para el resaltado de sintaxis, por lo que lo convierte en un programa muy extensible a la hora de escribir código en distintos lenguajes, sin tener que cambiar de herramienta.

Numerado de Líneas

El numerado de líneas en un archivo de código fuente puede resultar muy útil a la hora de depurar código de errores y en general para mejorar la legibilidad de un programa. En Vim podemos modificar muchos parámetros del editor mientras editamos los ficheros. Por

Operaciones con archivos	
":w"	Guarda el fichero
":w nombreArchivo"	Guarda el fichero con el nombre indicado
":wq"	Guarda el fichero y sale del editor
"zz"	Guarda el fichero y sale del editor
":q"	Sale si no ha habido cambios en el fichero
":q!"	Sale sin guardar los cambios en el fichero
":e archivo"	Edita el fichero indicado si no hay cambios en el actual
":e! archivo"	Edita el fichero indicado perdiendo los cambios en el actual, si hubiera
":r archivo"	Añade el archivo indicado después de la línea actual
":Nr archivo"	Añade el archivo indicado después de la línea indicada
":N,Mw!"	Guarda desde la línea "N" a la "M" descartando las otras
":N,M>>archivo"	Añade desde la línea "N" a la "M" en el archivo indicado
":="	Muestra la línea actual
"CTRL+G"	Muestra el estado del fichero

Figura 4. Comandos para las distintas Operaciones con Archivos

Moviéndonos por el fichero	
"j" (o cursor abajo)	Siguiente línea
"k" (o cursor arriba)	Línea anterior
"l" (L minúscula) (o cursor derecho)	Siguiente carácter
"h" (o cursor izquierdo)	Carácter anterior
"[["	Inicio del archivo
"]]"	Final del archivo
"nG"	Ir a la línea "n"
"G"	Ir al final del archivo
RETURN	Siguiente línea
CTRL+F	Pantalla siguiente
CTRL+B	Pantalla anterior
CTRL+D	Media pantalla siguiente
CTRL+U	Media pantalla anterior

Figura 5. Comandos para los Movimientos Básicos dentro de Vim

Tecla	Acción
"a"	Insertar después de carácter donde estamos situados
"i"	Insertar antes del carácter donde estamos situados
"A"	Añade al final de la línea actual
"I" (i mayúscula)	Añade al principio de la línea actual
"R"	Entra en el modo inserción reemplazando caracteres
"o" (o minúscula)	Añade una línea en blanco debajo de la nuestra y pasa a modo inserción
"O" (o mayúscula)	Añade una línea en blanco encima de la actual y pasa a modo inserción

Figura 6. Comandos para la Inserción de Texto

Comando	Estructura
\sc	case in ... esac
\sf	for in do done
\sfo	for ((...)) do done
\sw	while do done
\si	if then fi
\sie	if then else fi
\sl	elif then
\ss	select in do done
\st	until do done
\se	ech -e "\n"
\sp	printf "\n"

Tabla 1. Comandos de desplazamiento útiles para programar

ejemplo tecleando el comando `:set number` todas las líneas del fichero del lado izquierdo estarán numeradas y dicha numeración aparecerá en pantalla. Si queremos desactivar esta opción, simplemente tecleamos el comando : `set nonumber`. Esta función básica pero potente es algo que sin duda puede ser útil para programar.

Indentación Automática y Manual

Existen varias formas distintas que Vim nos propone a la hora de indentar código. En principio, abordaremos las técnicas automáticas, mostrando sus diferencias entre sí. Luego, veremos cómo arreglar la indentación de forma manual.

Las principales formas de automatizar el proceso de indentación de código son mediante tres comandos, dos de ellos internos de Vim (`:set smartindent` y `: set autoindent`) y uno externo, que es una utilidad que nos provee la shell, llamada `indent`. A modo de aclaración, esta última utilidad puede instalarse mediante el comando `sudo apt-get install indent`. El comando smartindent es una versión mejorada del comando autoindent, en donde el primero tiene la capacidad de reconocer de manera inteligente las estructuras del lenguaje que estemos utilizando y así aplicar la indentación en llaves, bloques, etc., según corresponda, siempre y cuando tengamos el archivo de configuración de indentación para ese lenguaje. Esto lo podemos chequear ubicándonos en el directorio */usr/share/Vim***/*indent* (El ** debe ser reemplazado por la versión de Vim que estemos utilizando sin el punto separador. En mi caso, es la versión 7.2, es por eso que la ruta quedaría así */usr/share/Vim72/indent*). De no poseer el archivo de indentación, podemos descargarlo de internet en la página oficial de Vim, en la sección de Scripts: *http://www.vim.org/scripts/index.php*.

Con respecto a la indentación manual, podemos mencionar el uso de los comandos << y >>. Estos comandos se encargan de "tabular" nuestro código, dependiendo de la configuración de la opción ShiftWidth. Por ejemplo, si nuestro ShiftWidth está seteado en 10, cuando apliquemos el comando << o >> se nos indentará la línea actual hacia la derecha o la izquierda, respectivamente. Para cambiar el valor de ShiftWidth, invocamos al seteador mediante `:set shiftwidth=X` , donde X es el valor de la tabulación, que por lo general, es tres.

Otro comando interesante es el comando =. Este comando nos indentará la línea actual, si ésta no fue indentada. Es muy útil ya que permite el uso de *counts* o repetidores. Por ejemplo, si queremos asegurarnos que la totalidad de nuestro código esté indentado correctamente, simplemente llamamos al comando gg=G. Este comando nos situará en la primer línea del archivo (gg), luego indentará todo el archivo (=) hasta el final (G), algo que sin duda es de suma utilidad. Otras formas de utilizar este comando para indentar partes específicas de código es:

- `=iB` Tabula el bloque de código entre {y} pero no estos caracteres.
- `=aB` Tabula el bloque de código entre {y} incluyendo estos caracteres.
- `=ib` Tabula el bloque de código entre (y) pero no estos caracteres.
- `=ab` Tabula el bloque de código entre (y) incluyendo estos caracteres.

Por ultimo, en el tema de indentado, podemos recurrir a una poderosa y vieja herramienta de

Comando	Acción
"vi archivo"	Edita el fichero en modo pantalla completa
"vi –r archivo"	Recupera la última versión guardada del fichero
"vi + archivo"	Edita el fichero y se situa en la última línea
"vi +numLinea archivo"	Edita el fichero y se sitúa en la línea indicada
"vi archivo1... archivoN"	Va editando todos los archivos especificados. Para saltar de uno a otro deberemos escribir en modo comando ":n". Con ":n" no guardamos las modificaciones.
"vi +/string archivo"	Edita el fichero situando el cursor en la primera ocurrencia del "string" indicado.

Figura 7. Comandos de Copiado, Pegado, Borrado, Búsqueda y Reemplazo de texto

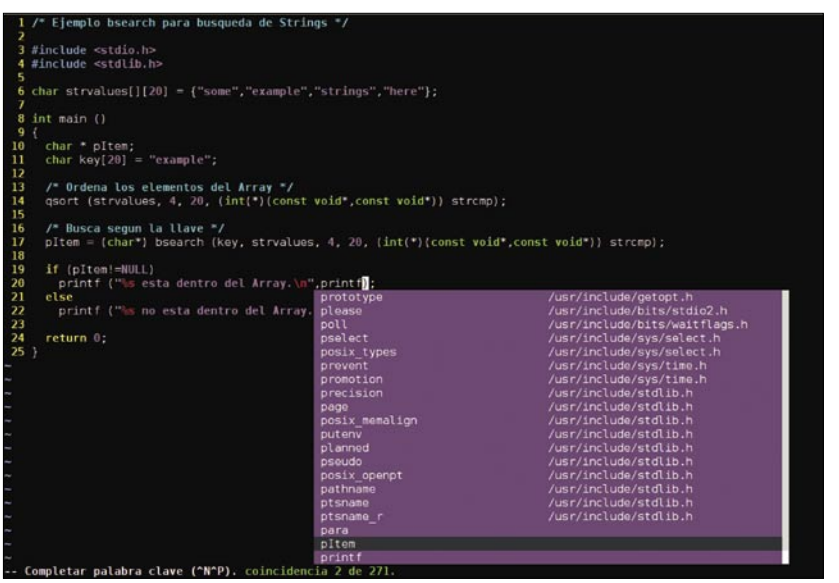

Figura 8. Ejemplo del uso de Coloreado de Sintaxis, Numerado de Líneas, Indentación y Autocompletado manual

Figura 9. Marcas para poder plegar el texto

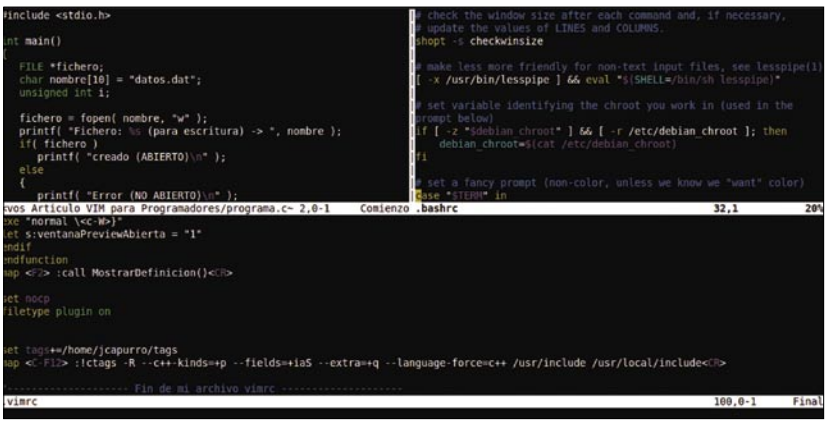

Figura 10. Texto plegado mediante el método marker

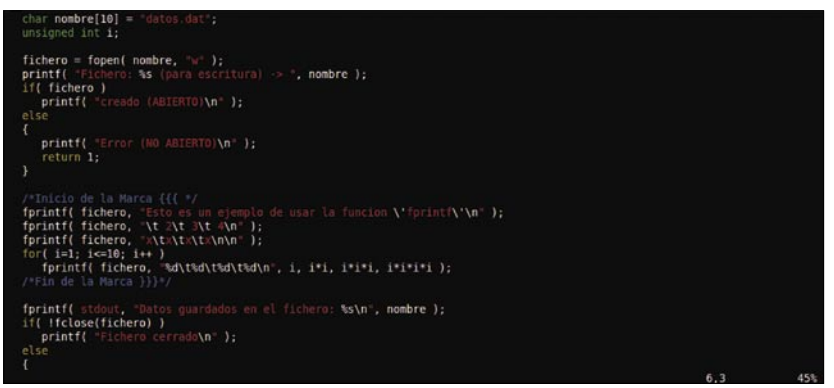

Figura 11. Uso del Explorador de Archivos :Vex. A la izquierda, el explorador de archivos, a la derecha, un código fuente

UNIX: la herramienta `indent`. Este utilitario sirve, como su nombre lo indica, para corregir indentaciones de códigos fuente. Solamente, para su uso desde Vim, la invocamos mediante el comando `:!indent %`. De no tener instalada la herramienta, lo podemos hacer mediante `sudo apt-get install indent`. También existe un seteo de indentación específico para el lenguaje C. Este seteo se activa mediante el comando `:set cindent` y puede desactivarse con `:set nocindent`.

Comandos Específicos para Programadores

En esta sección, se mostrará una recopilación de comandos útiles para los programadores. Estos comandos se aplican en base al texto que muestra Vim, es decir, no son comandos del tipos "seters" (técnicamente llamados *comandos ex*, en referencia al viejo editor de texto), como los que estuvimos viendo hasta el momento. Todos estos comandos tienen múltiples usos, pero daremos ejemplos en donde aplicados al entorno de la programación, resultarán de suma utilidad. La mayoría de los comandos tratan sobre el desplazamiento eficiente y rápido sobre el código fuente en Vim.

La Tabla 1 resume los comandos de movimiento que nos serán de mucha utilidad una vez que los tengamos bien incorporados. Por ejemplo, en referencia a la Tabla 1, si utilizamos el comando `[{d}]` dentro de un bucle while, lo borraremos por completo, sin necesidad de fijarnos donde comienza y termina el bucle ¿Demasiado rápido no? Combinaciones de este tipo son muy utilizadas y se vuelven casi automáticas con la práctica.

Dos desplazamientos rápidos que incluye Vim pura y exclusivamente para los programadores, son los comandos `gD y gd`. El primero, cuando estamos situados en el nombre de una variable, nos lleva a la declaración global de la misma. El segundo, funciona de la misma forma, solamente que nos lleva a la declaración local. Por ejemplo, si tenemos una porción de código que es `i=10`, y nos posicionamos sobre la variable `i`, cuando presionemos el comando gd nos podrá llevar hasta la línea que dice, por ejemplo, `int i;`. Es incalculable la comodidad de usar este comando en las tareas diarias.

El comando % resulta muy cómodo también. Éste, situado sobre un delimitador de bloque, nos mueve a su pareja. O sea, si estamos sobre el { nos lleva al } y viceversa. También se puede utilizar para encontrar las parejas de `#if, #ifdef, #else #elif and #endif`, por ejemplo.

Otro comando que vale la pena mencionar es el comando '. (comilla simple y punto) que nos permitirá posicionarnos en el lugar en que estábamos la última vez que editamos este fichero. Muy útil a la hora de corregir errores.

Por último, el comando `<NUMERO>G` posicionará el cursor en la línea número `NUMERO` del documento. Por ejemplo, "100G" nos llevaría a la línea número 100. Es especialmente útil a la hora de programar, cuando tenemos que ir a líneas concretas del programa donde el compilador nos ha reportado errores.

Una característica muy interesante incluida en las ultimas versiones de Vim, es la del uso de identificadores. La desventaja de usar los comandos gd y gD es que éstos no tienen la posibilidad de buscar en ficheros de inclusión, solamente buscan declaraciones de variables, estructuras, etc., sobre el fichero actual. Los identificadores fueron creados para corregir esa problemática. Para hacer uso de ellos, simplemente nos posicionamos sobre una palabra, y mediante la combinación de teclas [I se nos aparecerá una lista de todos los identificadores encontrados en el archivo actual y los incluidos junto con el nombre del archivo y el número de linea de la definición.

Creación de Marcas

El comando '`m`' es el utilizado para crear marcas en nuestros archivos. Pero primero, ¿qué es una marca? La respuesta es simple. Una marca sir-

Tip: Ver Archivo en Hexadecimal

Si queremos visualizar nuestro archivo en hexadecimal, podríamos hacerlo bien saliendo de Vim y ejecutando la poderosa herramienta *od*. Pero si deseamos ver y modificar nuestro archivo en hexadecimal sin salir de Vim, podemos hacerlo mediante un simple comando: `:%!xxd`.

Tip: Utilizar sudo dentro de Vim para dar permisos de escritura

Muchas veces sucede que abrimos un fichero con Vim y lo editamos, y cuando queremos guardarlo, nos damos cuenta que no tenemos permisos de escritura sobre él. ¿Qué hacer en esa situación? Para no tener que descartar las modificaciones realizadas, nos remitimos a introducir el comando `:w !sudo tee %` y listo, problema solucionado.

ve para poder volver a una línea (determinada por la marca) en cualquier momento de manera rápida mediante una combinación de teclas. Esto es muy útil, por ejemplo, cuando estamos programando y necesitamos con frecuencia volver a una determinada función. En vez de ir buscando la función cada vez que queramos acceder a su código, simplemente le asignamos una marca y volveremos a ella de manera rápida.

Las *marcas* se guardan en buffers, desde [a-z], [A-Z] y [0-9]. Para crear una marca, simplemente presionamos la combinación de teclas mx, siendo "x" el nombre del buffer en donde se guardará la marca. Para acceder a una marca, debemos presionar 'x , siendo "x" el nombre del buffer asignado. Es decir, `ma` sería una *marca* a la que se vuelve pulsando 'a. Es importante que el buffer de marcas se vacía una vez cerrado el archivo.

Una marca interna de Vim que es muy útil es la marca `gd`. Esta marca tiene como característica que al poner el cursor encima de una variable podemos ir directamente a la línea donde se declaró. Si la variable está declarada como local y global, podremos ir a la declaración como global pulsando `gD`.

Autocompletado Manual

Aparte del autocompletado de sentencias automático que posee Vim agregándole plugins, también posee un sistema de autocompletado que viene incorporado en la configuración de Vim por defecto. Esta funcionalidad es la que veremos ahora, dejando los plugins específicos para más adelante. El autocompletado no solamente sirve para los programadores, sino que se aplica también a cualquier archivo de texto en el que queramos autocompletar alguna palabra. Este mecanismo que Vim trae incorporado funciona básicamente indexando las palabras que vayamos escribiendo en el archivo actual y en los que hayamos incluido (en el caso del lenguaje C por ejemplo, los archivos que estén referenciados en la directiva `#include`) para que, cuando queramos autocompletarla, la busque de manera rápida y nos arroje los posibles resultados. El autocompletado de Vim se activa mediante la siguiente combinación de teclas:

- *Ctrol + N*: Busca las concordancias en el fichero desde la posición actual hacia adelante (si se llega al final, se sigue buscando desde el principio).
- *Ctrol + P*: Busca las concordancias en el fichero desde la posición actual hacia atrás (si se llega al principio, se sigue buscando desde el final hacia atrás).

Por ejemplo, supongamos que en una parte de nuestro código aparece `funcion1()`. Más adelante, para poder autocompletar la instrucción, tendríamos que escribir una parte de la palabra a autocompletar y presionar alguna de las combinaciones de teclas que se describen anteriormente. Por ejemplo, podríamos poner "func" (sin comillas) y luego presionar *Ctrol + P*. Se nos desplegará un listado debajo de la palabra a completar con todas las posibles coincidencias, si es que las hubiera (por ejemplo, se podría desplegar `funcion1()`, `funcion2()`, `funcion_bsearch()`, etc.).

En la Figura 8, podemos ver un ejemplo de implementación de la función estándar de búsqueda binaria `bsearch` en C, en la cual estamos utilizando Vim con las características de numerado de línea, resaltado de sintaxis, indentación y autocompletado manual de sentencias. Nótese en el ejemplo cómo en el autocompletado aparecen como posibles coincidencias, tanto las funciones del lenguaje (printf, pathname,

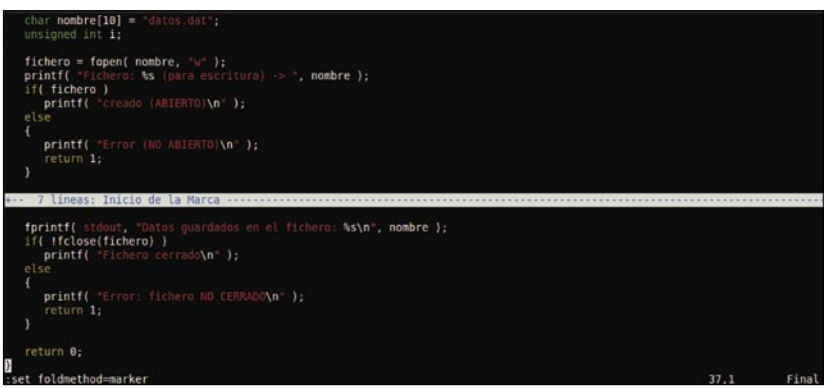

Figura 12. Uso de Tabs para editar múltiples ficheros simultáneamente

Figura 13. Uso de Split para editar múltiples ficheros simultáneamente

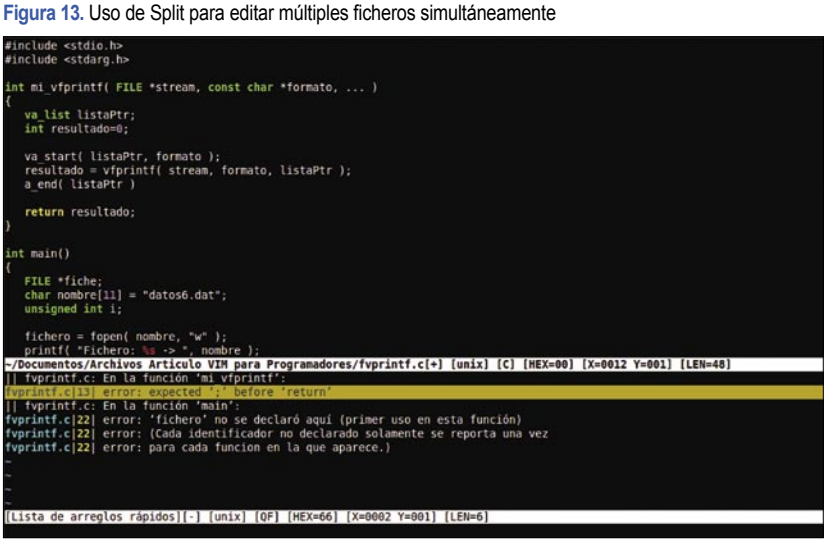

Figura 14. Uso de QuickFix

Tabla 2. Listado de algunos de los más famosos plugins que Vim ofrece a la hora de programar

Comando	Descripcion/Accion
[{	Ir al { que abre el bloque de llaves
}]	Ir al } que cierra el bloque de llaves
[(	Ir al (que abre el bloque de paréntesis
)]	Ir al) que cierra el bloque de paréntesis
[*	Ir al /* que abre el comentario del tipo /**/
*]	Ir al */ que cierra el comentario del tipo /**/

etc.) como también los identificadores propios del programa, como en este caso, la variable `pItem`, algo realmente muy útil.

También podemos autocompletar nombres de archivos o directorios donde se encuentre nuestro fichero a editar, presionando la combinación de teclas Ctrol + X + F. Por último, una funcionalidad muy interesante es la que se logra presionando las teclas Ctrol + X + O, la cual mostrará una lista de las funciones que puedan coincidir en el autocompletado, además de una ventana superior especificando diversa información como la librería donde se encuentra, su prototipo, etc.

Aplicar Folding al Código

El plegado (o folding) de código fuente le da al programador la posibilidad de concentrarse en las partes del código que realmente quiere analizar o modificar, ofreciéndole un ambiente de trabajo mucho más ordenado. Cuando pleguemos código, éste se reducirá a una sola línea informando la cantidad de líneas ocultas y mostrando la primer línea del contenido del pliegue.

De esta característica no escapa Vim, y es por ello que nos ofrece distintos métodos de plegado automático como manual, indent, expr, marker, syntax y diff. Entre los que se destacan por ser usados con más popularidad, los tres siguientes:

- `indent`: Utiliza los niveles de indentados configurados para determinar qué líneas indentar.
- `syntax`: Utiliza una inteligencia similar a la del resaltado de sintaxis a la hora de plegar las líneas.
- `marker`: Realiza el plegado de código en base a una marca predefinida por el usuario.

Para activar el método de plegado mediante `indent`, invocamos al comando `:set fold method=indent` y veremos cómo se pliegan todas nuestras líneas, según el indentado que tengan. Este método resulta sencillo de aplicar, pero a veces se vuelve incómodo, ya que en ciertas ocasiones que el editor pliegue un *if* de dos líneas crea más problemas de lo que ayuda.

Con respecto al método syntax, debemos setear los delimitadores que queramos que se usen para plegar el código Por ejemplo, cuando programamos en C, nos puede resultar útil la siguiente expresión de configuración:

```
:syntax region myFold start "{" end
"}" trasparent fold
:set foldmethod=syntax
```

ya que provocará que se pliegue el código según las llaves que se encuentren en el archivo, que se encargan de delimitar los bloques de las estructuras del lenguaje.

Por último, el método más eficiente a mi parecer, es utilizar plegado mediante `marker`. En primer lugar, tendremos que establecer una marca que se encargará de delimitar el texto a plegar. Estas marcas son {{{ para la marca de comienzo, y }}} para delimitar el fin de la zona de pliegue. En la Figura 9 podemos ver esta acción. Luego, seteamos el método de plegado con :set foldmethod=marker y por último nos queda manipular los pliegues con los comandos correspondientes. La siguiente lista de comandos es universal para cualquier método de pliegue:

- `zo`: Abre el pliegue bajo el cursor,
- `zc`: Cierra el pliegue bajo el cursor,
- `za`: Cierra el pliegue si está abierto y lo abre si está cerrado,
- `zR`: Abre todos los pliegues,
- `zM`: Cierra todos los pliegues,
- `zj`: Mueve el cursor al pliegue siguiente,
- `zk`: Mueve el cursor al pliegue anterior,
- `zd`: Elimina el pliegue que se encuentra debajo del cursor,
- `zE`: Elimina todos los pliegues del archivo.

La Figura 10 muestra como queda el plegado el texto, según la marca definida anteriormente. Puede consultar la ayuda de Vim si quiere ver cómo se utilizan los demás métodos de plegado. Simplemente tipee : `h foldmethod`

Uso del Explorador de Archivos

Entre otras de las características a destacar de Vim, es la inclusión de un explorador de ficheros/directorios propio. El mismo se llama con los comandos `:Vex` y `:Sex` (o bien, de la forma larga :Vexplore y :Explore respectivamente). Las diferencias de estos dos comandos es en qué forma abrirán el nuevo archivo, es decir, si lo harán dividiendo la pantalla en forma vertical u horizontal (ver apartado siguiente "Trabajar con Múltiples Ficheros"). En el caso de :Vex, lo hace de forma vertical, y :Sex de forma horizontal. En la Figura 11, vemos el uso de :Vex para explorar un árbol de directorios determinado.

En la parte superior del explorador de ficheros, podemos cambiarle el orden en que queremos que nos muestre los archivos, como también renombrarlos, eliminarlos, etc. Si no queremos que el explorador de archivos se abra en una división aparte, podemos usar el comando `:edit .` el cual abrirá al explorador de archivos con ruta principal en el directorio actual (.).

Vale aclarar que existen plugins específicos que realizan la misma tarea, pero de manera (tal vez) más eficiente. El uso de estos plugins lo veremos en la segunda parte del artículo.

Trabajar con Múltiples Ficheros

Anteriormente, el editor vi no traía soporte para el trabajo con múltiples ficheros. Podríamos llegar a resolver este problema usando GNU Screen, pero no era de lo más recomendable, ya que había que saber hacer uso de una aplicación externa.

El trabajo con múltiples ficheros en el editor vi tiene una historia anecdótica. Una de

Tip: Explorando archivos con Wildmenu

Wildmenu es una característica simple e interesante de Vim. Nos permite navegar por el sistema de archivos donde estemos parados, de manera rápida y sencilla. Por ejemplo, si queremos editar un archivo utilizamos el comando :e. Si luego de esto, presionamos la combinación de teclas CTRL + D, wildmenu aparecerá, dejándonos navegar por nuestro file system en busca del archivo requerido.

las historias más antiguas y preferidas sobre vi es que aparentemente Bill Joy (creador y programador de Vi) estaba trabajando sobre una versión multiventanas de vi, pero el código fuente se perdió, y por eso vi nunca llegó a desarrollarse con aplicaciones multiventanas hasta la aparición de las versiones modernas, como es Vim. En palabras de Joy, citadas en una entrevista aparecida en el número de agosto de 1984 de la revista *Unix Review*, dice lo siguiente acerca de este hecho: *"...lo que ocurrió realmente fue que estaba en pleno proceso de desarrollo para añadir multiventanas a Vi y entonces instalamos nuestro* VAX*, allá por diciembre de 1978. No habíamos hecho copias de respaldo y la unidad de cinta falló. Seguí trabajando a pesar de no poder hacer copias de seguridad. Entonces el código fuente se corrompió y no tenía hecha copia del listado. Casi había rehecho todo el código de visualización de las ventanas, y eso fue lo que me hizo desistir. Después de aquello, recuperé la versión anterior y me limité a documentar el código, terminar el manual y archivarlo definitivamente. Si no se hubiera fastidiado ese código, Vi tendría múltiples ventanas, y le hubiera añadido algún tipo de programabilidad... quién sabe...*".

Por suerte, la gente de Vim resolvió el problema que Bill habría querido resolver allá por los años '70. Este problema fue subsanado añadiendo a Vim dos funcionalidades distintas a la hora de trabajar con múltiples ficheros a la vez: el uso de tabs y split. Con respecto a los tabs, se intentó añadir la funcionalidad de edición múltiple con pestañas que separen los distintos archivos, al mejor estilo Firefox. Con respecto al split, éste divide la pantalla permitiendo ver simultáneamente los ficheros que estamos editando.

Para utilizar tabs en Vim, simplemente necesitamos aprender un par de comandos, que son sumamente mnemónicos:

- `:tabfind` *archivo*: Abre *archivo* en un nuevo tab,
- `:tabnext` : Avanza al tab siguiente,
- `:tabprevious` : Avanza al tab anterior
- `:tabnew` : Abre un nuevo tab,
- `:tabmove n` -: Mover el tab a la posición *n*.

En la Figura 12, se puede ver como podemos editar dos archivos usando tabs. En primer lugar, se está editando el archivo *main.c* y luego el archivo *funciones.h*. Puede resultar tedioso escribir todos estos comandos cada vez que queramos cambiar de tab o agregar alguno

Tabla 3. Comandos para insertar estructuras con Bash-Support

Plugin	Descripcion
NERD tree	NERD tree es un plugins que nos brindara un arbol de directorio mucho mas flexible y com mas opciones que :Vex
Nerd Commenter	Un plugin que permitecomentar de código de manera facilpara muchos (casi todos) los archivos.
VCS Command	Este plugins nos permitira manipular archivos controlados por CVS, SVN, GIT y en SVK , incluyendo la posibilidad de realizar los cambios y diferencias mediante la herramienta vimdiff.
SQLComplete	Este plugin incluye las declaraciones, las funciones, las palabras clave, los operadores, ect de SQL. Incluye 9 ficheros diferentes de sintaxis de SQL (Oracle, Informix, MySQL, etc). Puede elegir diferentes variaciones SQL usando el comando (: h sql-dialects):
Align	Este plugin permite alinear codigo de acuerdo con una lista de caracteres predefinidos. Muy util a la hora de querer dejar el codigo prolijo
Doxygen-Support	Este plugin permite comentar codigo siguiendo las normas para poder crear documentacion de manera automatica mediante la potente herramienta *doxygen*. Muy recomedable.
Xdebug	Este plugin nos permitirá conectarnos al modulo XDebug de PHP , podremos poner Breakpoints, ver el contenido de las variables en un momento dado.
Check Syntax	Este plugin permitira que cada vez que guardemos un archivo PHP o cliquemos la tecla F5 se ejecutará la comprobación de la sintaxis PHP, indicándonos los errores si los hubiera.
DBExt	Con este plugin podremos conectarnos a varias BBDD y hacer consultas. La configuración de la conexión se puede hacer al momento o tener un pool de conexiones en nuestro .vimrc
FindMate	Este plugin nos permite tener un buscador de archivos dentro del árbol de directorios.
SQLUtils	Este plugin nos permite darle un formato a nuestra consulta SQL. También pude generar la lista de columnas de una tabla si en cualquiera de los buffers tenemos el CREATE TABLE.
Vimspell	¡Simpre es bueno escribir nuestros programas sin faltas de ortografia!

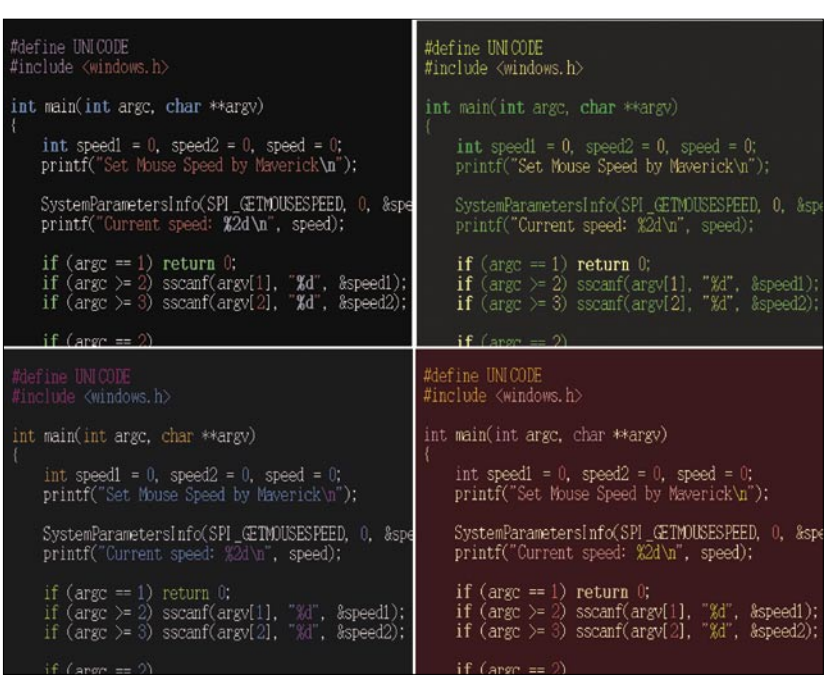

Figura 15. Esquemas de Colores (de izquierda a derecha): CandyCode, CRT, PaintBox y Red

nuevo. Para ello, veremos cómo mapear teclas y configurarlas a nuestro gusto en el archivo .vimrc, que lo veremos en detalle en "Configurando el archivo .vimrc".

Para utilizar `split` en Vim, solamente nos tendremos que acordar algunos comandos (o bien, configurarlos a nuestro gusto). Los comandos típicos para el uso de split en Vim son los siguientes:

- `:hide` : Cierra la ventana actual.
- `:only` : Cierra todas las ventanas excepto la actual.
- `:split` *archivo o* `:new` *archivo* : Divide la ventana horizontalmente, carga *archivo* y lo muestra en la nueva ventana.
- `:sview` *archivo*: Igual que :split, pero muestra el archivo en modo de sólo lecura.
- `:vsplit` *archivo o* `:vnew` *archivo*: Divide la ventana verticalmente, carga *archivo* y lo muestra en la nueva ventana.

También puedes iniciar Vim en modo multiventana utilizando la opción `-o` seguida por la lista de archivos que deseas abrir, como en el siguiente ejemplo:

```
$vim -o programa.c fuciones.h
texto.txt
```

Ahora solamente nos faltan los comandos para movernos entre las distintas ventanas. Nótese que todos empiezan con la combinación de teclas CTRL+ :

- *CTRL+W* : Cambia el cursor cíclicamente entre las ventanas,
- *CTRL+J* : Cambia a la ventana de abajo,
- *CTRL+K* : Cambia a la ventana de arriba,
- *CTRL+H* : Cambia a la ventana de la izquierda,
- *CTRL+l* : Cambia a la ventana de la derecha,
- *CTRL+=* : Iguala los tamaños de las ventanas,
- *CTRL+_* : Maximiza la ventana actual.

Podemos mencionar como otros de los comandos útiles el uso del comando `:ls` para listar los buffers abiertos y el comando `:b` *numerodebuffer* que abriría el buffer elegido en la ventana que está actualmente enfocada. Puede ver un ejemplo del uso de split en la Figura 13, donde se está programando un programa en C y editando los ficheros de configuración `.vimrc` y `.bashrc` al mismo tiempo.

Sencillo, ¿verdad? Si necesita más ayuda o quiere investigar más sobre estos comandos, consulte `:help split`.

Compilación desde Vim

Existen diferentes formas de poder compilar desde Vim. En esta ocasión, nos concentraremos en 3 métodos enfocados al lenguaje C, por su gran popularidad y uso. Vale la pena aclarar que para compilar otros lenguajes, como lo puede ser Java, estos métodos sufren mínimas variaciones. Los métodos referidos para poder compilar, son los siguientes:

- Uso de comandos externos de shell: Este método es el más sencillo, pero el más ineficiente también si se trata de proyectos grandes. Simplemente, llamamos al compilador gcc mediante una llamada externa a un comando. Es decir, lo hacemos mediante `:!gcc %`. Podemos dirigirnos a la línea y obtener directamente una descripción del error ocurrido invocando al comando `:cn`. Veremos más comandos a lo largo de este apartado.
- Uso de Plugin CVim: Como se mencionó con anterioridad, el plugin *CVim* tiene la posibilidad de compilar directamente dentro de Vim. Para ver su uso, diríjase al apartado “Plugins de Utilidad”.
- Comando :make : Vim tiene la posibilidad de reconocer un archivo *MakeFile* y poder utilizarlo para poder compilar el programa en cuestión. Simplemente, dentro de nuestro archivo fuente, introduciendo la directiva `:make`, Vim compilará automáticamente utilizando nuestro MakeFile anteriormente creado.
- También, podemos decirle a Vim que mediante el comando :make utilice a GCC para la compilación, en vez de nuestro MakeFile. Esto lo logramos con el comando `:set makeprg=gcc`. Luego, para invocarlo usamos `:make %`.

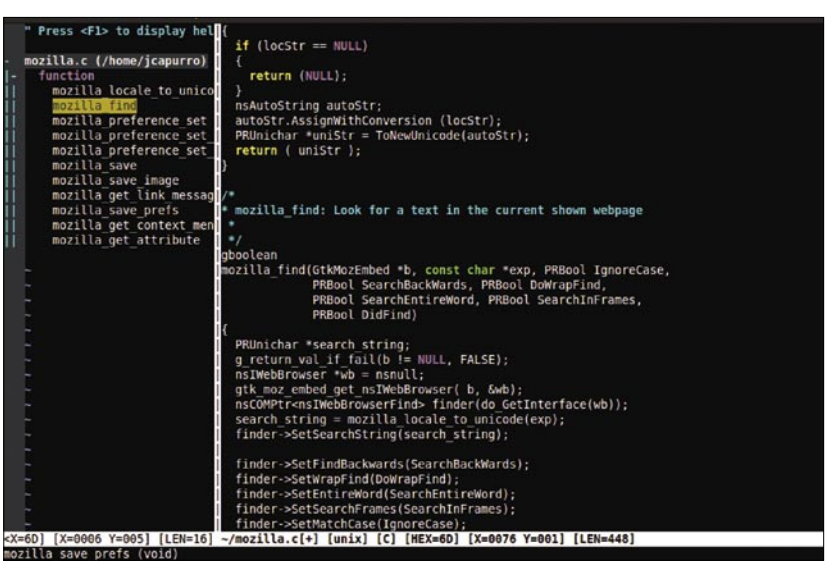

Figura 16. Muestra del uso de TagList

Algo realmente novedoso en Vim, es la inclusión de la ventana de QuickFix. Esta ventana, como su nombre nos indica, tiene la tarea de facilitarnos el debugging de errores. Es decir, cuando compilamos con algunos de los métodos descriptos anteriormente, los errores son informados en la misma terminal de Shell, es decir, no se ven dentro de Vim. QuickFix vino a subsanar esta “incomodidad”. Para invocar a QuickFix utilizamos el comando `:cope`. En la Figura 14 podemos ver a QuickFix en acción, luego de compilar con errores. Los comandos útiles de QuickFix son:

- `:cl` Listar los errores,
- `:cc` Mostrar el mensaje completo del error actual,
- `:cn` Moverse al siguiente error,
- `:cp` Moverse al error anterior,
- `:ccl` Cerrar la ventana de QuickFix.

Estos comandos también pueden usarse cuando compilamos con :make, y por ejemplo, no queremos tener la ventana de QuickFix activada.

Como última sugerencia, tal vez puede resultar más cómodo mapear las teclas de los comandos :cn y :cp, ya que se usarán con frecuencia. Para ello podemos incluir en nuestro `.vimrc` (ver apartado “Confi-

gurando el Archivo `.vimrc`", el siguiente código de mapeo, para poder desplazarnos entre los errores con F7 y F8:

```
map <F7> :cp<CR>
map <F8> :cn<CR>
```

Ejecución de Comandos de la Shell

Vim posee la característica de poder ejecutar comandos de Shell dentro de él, pegar su salida en el editor, o bien abrir una nueva shell sin la necesidad de salir de Vim. Todo esto se hace mediante simples *comandos del tipo ex*. Primero, si queremos ejecutar una nueva shell sin salir de Vim, simplemente tecleamos el siguiente comando `:sh` , donde se ejecutará una nueva shell interactiva. Si queremos salir de ella, tecleamos `exit`. Dentro de esta shell, podrías si quisieses, ejecutar Vim de nuevo. Esto es particularmente útil, por ejemplo, cuando estás editando makefiles y/o ficheros de configuración de programas en un intento de hacer que un programa compile correctamente.

Ahora bien, para ejecutar comandos de Shell sin tener que salir de Vim, solamente tendremos que poner `:!comando`, siendo "comando" el nombre del comando.de shell a ejecutar. Por ejemplo, si ponemos :!ls, veremos como se ejecuta el comando `ls` del directorio actual. Luego, presionando la tecla Enter volvemos a Vim como si nada hubiese ocurrido. Si necesitamos especificar en nuestro comando el nombre del archivo que tenemos abierto con Vim, lo hacemos mediante el comodín %. Por ejemplo, si queremos saber qué tipo de archivo estamos editando, empleamos el siguiente comando `:!file %`. O bien, podemos llamar a gcc para que compile nuestro código, por ejemplo, mediante `:!gcc % -o Salida` y luego ejecutar el programa mediante `:!./Salida` ¿Que fácil no?

Por último, algo muy útil es incluir la salida de un comando de Shell en el archivo que estamos editando. Por ejemplo, supongamos que estamos editando un código fuente al que le estamos realizando modificaciones. Estas modificaciones tienen que ser documentadas, por ejemplo, al principio del archivo de código fuente. Si queremos cambiar la fecha en la cual se ha modificado el archivo, simplemente recurrimos al comando `:r !date` y veremos como la nueva fecha se pega en nuestro código ¡Esto es algo muy útil a la hora de programar!

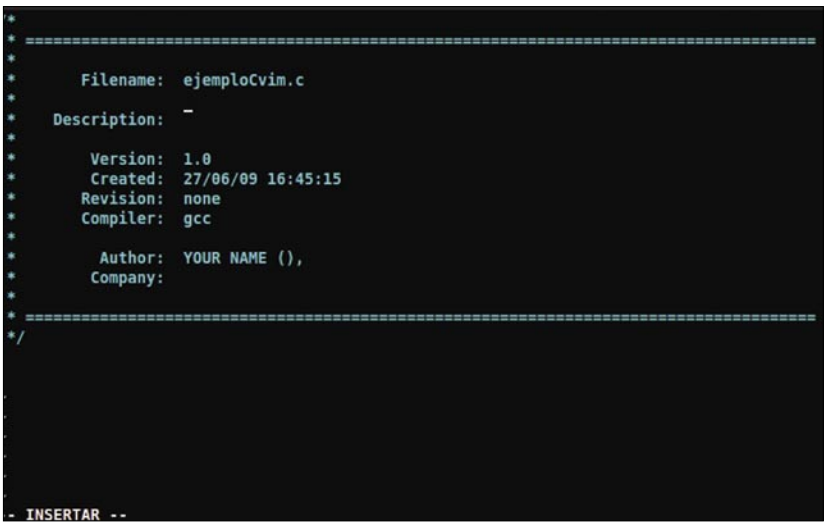

Figura 17. Template de Comentarios principal de CVim

Parte 2
Configuración Avanzada

¡Excelente! Hemos finalizado la primer parte del artículo. De ahora en más, nos dedicaremos a configurar al extremo nuestro editor de texto agregándole plugins y elementos personalizados, para poder tener todo un ambiente de desarrollo productivo en menos de 7MB. Los temas a tratar de aquí en adelante son:

- Esquemas de Colores,
- Plugins de utilidad,
- Uso de Snippets,
- La Herramienta *vimdiff,*
- Configurando el archivo *.vimrc.*

Esquemas de Colores

Los esquemas de colores, o *colorschemes* son temas visuales que se aplican a Vim para que éste tenga un mejor aspecto. Se encargan de modificar el color de fondo de pantalla, y los distintos colores que corresponden al resaltado de sintaxis. Existen cientos de distintos esquemas, además de poder el usuario crear uno personalizado, simplemente modificando un par de líneas de un archivo típico de esquema. Pueden ver todos los esquemas oficiales de Vim en el siguiente enlace *http://www.vi-improved.org/colors.php*. Si desea, puede descargar el que le guste, o bien, si desea tener todo el pack, puede descargarlo de *http://www.busindre.com/wp-content/uploads/2007/06/Pack_colores_vim.rar*.

¿Cómo se instalan? Basta con copiar a la carpeta *~/.vim/colors el/los tema/s* (si no existe, crearla). Para seleccionar el esquema a utilizar debe estar previamente activado el resaltado de sintaxis con *:syntax on*, y luego, mediante el comando *:colorscheme nombreEsquema* elegimos el de nuestra preferencia. Si deseamos que siempre que

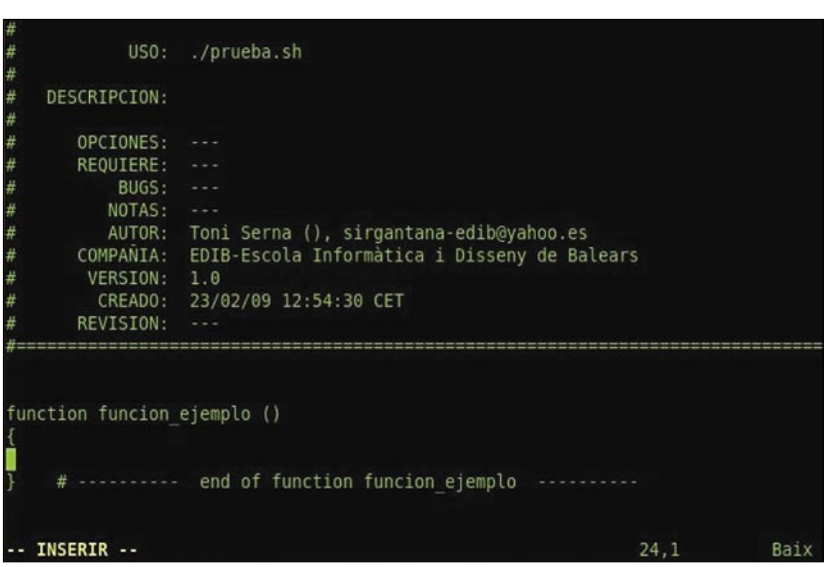

Figura 18. Generación automática del Esqueleto de una Función mediante Bash-Support

abramos Vim esté nuestro esquema presente, debemos modificar nuestro .vimrc, como se verá más adelante.

En la Figura 15 podemos ver cuatro esquemas de colores distintos, extraídos de la web oficial de Vim.

Plugins de Utilidad

Existen infinidades de plugins o extensiones que harán que nuestro Vim sea más fácil de usar, facilitándonos la realización de tareas y demás cosas que hacemos a diario. Tal como lo indica el nombre de este artículo, nos concentraremos en los plugins específicos que nos serán de ayuda a la hora de programar. Debido a la gran cantidad de lenguajes de programación existentes, existen muchos plugins que son específicos para algunos o muchos de ellos. Por motivos de espacio, presentaremos solamente el uso de tres plugins: `TagList` ,`CVim`, `Bash-Support`. A modo de referencia, en la Tabla 2, podemos ver otros plugins existentes para que el lector investigue su uso.

Empecemos con TagList. *TagList* es un plugin que nos proporciona un explorador de código con el cual podemos desplazarnos fácilmente entre diferentes funciones, métodos, clases, variables dentro del código de los ficheros que tengamos abiertos, al igual que los grandes IDEs.

Antes de poder usarlo, es necesario crear el archivo de tags. Para ello, usaremos el programa ctags, que puede instalarse mediante `sudo apt-get install exuberant-ctags`. Luego, crearemos el fichero de tags posicionándonos en el directorio donde tenemos nuestro archivo fuente e introduciendo el comando `ctags *.c *h`, para que genere todos los tags que utilizará TagList. Ahora, descargaremos e instalaremos el plugin. Para descargar el plugin, nos dirigimos a http://www.vim.org/scripts/script.php?script_id=273. Luego, lo instalamos descomprimiendo el archivo descargado en el directorio ~/.vim. ¡Listo! Ya podemos usarlo. En la Figura 16, podemos ver una parte de la función de búsqueda de texto del Código Fuente de Mozilla Firefox, donde se ve a la izquierda a TagList en acción.

Para abrir el árbol de TagList, introducimos el comando `:TlistOpen`. Luego, podemos navegar sobre él como si se tratara de otra ventana, aunque con algunas características más. Por ejemplo, si nos paramos en el nombre de una función y luego presionamos la tecla *space-bar*, veremos como se nos informa en la parte inferior de la pantalla el prototipo de esa función. Si queremos ir a la definición de una función, simplemente nos

Listado 1. Fichero vimrc orientado a la programación

```
"-------------------------------------------------"
"  Fichero de configuración de .vimrc [Revista Linux+]
"
"-------------------------------------------------"
"Detecta el Tipo de Archivo Automaticamente
filetype on
"Activamos soporte plugins
filetype plugin on
" Coloreado de Sintaxis
syntax on
"Esquema de Colores
colorscheme seoul
"Numerado de Lineas
set number
"Resalta la { 0 ) que estamos cerrando
set sm
"Mostrar la posicion del cursor en todo momento
set ruler
"Indentado Automatico
set smartindent
"Ignora si la palabra esta en mayusculas o minusculas
set ignorecase
"Busca la Palabra a medida que la tecleamos
set incsearch
"Resalta las palabras encontradas
set hlsearch
" Ocultar el mouse cuando escribimos
set mousehide
"Tabulacion de 3 caracteres
set ts=3
"Indentado de 3 caracteres
set sw=3
"Cambia los tabs por espacios
set expandtab
"Realiza una Copia de Seguridad del Fichero
set backup
"Directorio de la Copia de Seguridad
set backupdir=~/.tmp
"Todo lo que copiemos en el registro se copiará
también en el portapapeles
set clipboard=unnamed
"Para guardar los cambios y compilar automáticamente
con <F5>
map <F5> :wall<CR>:make<CR>
imap <F5> <ESC>:wall<CR>:make<CR>
"Modifica la Barra de estado
set statusline=%F%m%r%h%w\ [%{&ff}]\ [%Y]\ [HEX=\
%02.2B]\ [X=%041\ Y=%03v]\ [LEN=%L]
"Desactiva el sonido de error y lo sustituye
 por un parpadeo
"blanco en la pantalla
set visualbell
"Mapeo para los Tabs
map ,t :tabnew
map ,1 :tabprevious
map ,2 :tabnext
map ,f :tabfind
map ,m :tabmove
map ,n :tabnew
"Pone una linea horizontal donde se encuentra
 el cursor
"set cursorline
"Pone una linea vertical donde se encuentra el cursor
"set cursorcolumn
"--------------- Fin de archivo vimrc --------------"
```

posicionamos en el nombre de la función y presionamos la tecla ENTER. Por último, si nos interesa saber el número de funciones que contiene nuestro código fuente, vamos hacia el TagList, nos posicionamos sobre la palabra `function` y presionamos la tecla *space-bar*.

También podemos navegar mediante los tags usando el modo comando. Por ejemplo, si ponemos *:ta nombreFuncion*, nos dirigiremos directamente al cuerpo de la función que hayamos especificado. Para más información, podemos apretar F1 mientras la TagList tiene foco, o bien dirigirnos a *http://vim-taglist.sourceforge.net/manual.html.*

Ahora pasemos a otro excelente plugin: `CVim`. Como el lector se lo imaginará, este plugin está realizado pura y exclusivamente para facilitarnos la programación mediante el lenguaje C. Como todos los plugins, lo descargamos (*http://www.vim.org/scripts/script.php?script_id=213*) y lo instalamos. Sin más que acotar, estudiemos sus características, las cuales nos permitirán:

- Agregar archivos de cabecera,
- Incluir fragmentos de código predeterminados,
- Realizar comprobación de sintaxis,
- Leer documentación sobre una determinada función,
- Convertir un bloque de código a comentario, o viceversa,
- Indentación Automática,
- Etc.

Éstas son solamente algunas de las tareas que nos ofrece CVim. Veamos cómo realizar algunas de ellas.

Algo con lo cual nos encontramos al abrir un archivo de extensión .c con Vim y nos sorprende, es la inclusión automática de cabeceras para el código. Esto lo podemos observar en la Figura 17. Sin duda, es muy interesante y útil para los programadores, ya que permite tener documentado (mínimamente) el código a desarrollar. Si queremos modificar el *template* que genera esta cabecera, nos dirigimos al directorio *~/.vim/c-support/templates/* y editamos el archivo *Templates.* El archivo es muy intuitivo para su edición, solamente tiene claves del tipo "*par-valor*", por lo cual no merece una explicación.

Otra de las características que nos provee CVim es la posibilidad de generarnos código automático para la realización de una función. Por ejemplo, entramos en el modo comando, y luego presionamos \if, veremos como Vim nos preguntará el nombre de la función a generar. Una vez introducido el nombre, se autogenerará el cuerpo de la función. Características como éstas tenemos a montones, las cuales numero y describo algunas a continuación:

- Insertar cuerpo de la Función main. Para que nos genere automáticamente el cuerpo de la función main, utilizamos el comando `\im`.
- *Comentario de Descripción de una Función*. Esto lo podemos lograr con el comando `\cfu`. Al igual que `\if`, nos preguntará el nombre de la función.
- *Insertar comentarios simples*. Para ello, existe el comando `\cfr`.
- *Guardar, Compilar y Ejecutar*. Sí, todos estos pasos los realizamos con solo introducir el comando `\rc`. Para ejecutar el programa, existe el comando `\rr`.

Existen muchas más características que posee CVim. Para más información, visite la documentación oficial en *http://lug.fh-swf.de/vim/vim-doc/csupport.html*, o bien el manual interno `:h csupport`.

Por último, nos queda describir a otro excelente plugin. Esta vez, nos enfocaremos a la programación de Shell Scripts, usando Bash. El plugin en cuestión se denomina Bash-Support, y se puede descargar e instalar desde *http://www.vim.org/scripts/script.php?script_id=365*, y es el que estudiaremos a continuación.

Bash.Support nos provee básicamente las mismas facilidades que nos provee CVim, es decir, nos brinda la posibilidad de:

- Añadir automáticamente cabeceras a los ficheros,
- Añadir comentarios de diversos tipos,
- Escribir los esqueletos de los distintos comandos de control (if, case, for, while...),
- Verificar la sintaxis,
- Consultar la documentación de un comando de Bash,
- Convertir un bloque de código en comentario o viceversa,
- Escribir rápidamente expresiones regulares.

Este plugin ha sido programado por *Fritz Mehner,* el mismo autor que el CVim. Es por ello, que las formas de realizar las acciones automáticas en Bash-Support son similares a las de CVim, por lo cual no merece una explicación tan exhaustiva. A continuación se numeran algunas características de Bash-Support y la forma de llevarlas a cabo:

- *Cabeceras Automáticas*. Cuando crees un archivo con la extensión .sh, éste comenzará con una cabecera predefinida por Bash-Support, el cual contendrá campos a completar por el programador, como puede ser "Descripción del Script", "Autor", "Fecha", "Modificaciones", "Bugs", "Opciones", etc.
  Esta cabecera puede personalizarse fácilmente editando el fichero: *~/.vim/bash-support/templates/bash-file-header.*
- *Esqueletos de Funciones.* Si en el editor en modo Normal pulsamos la secuencia *\sfu* (statement function – declaración de función) nos aparece un prompt en el que se pide el nombre de la función. Una vez tecleado ("función_ejemplo") veremos que en el lugar del cursor se habrá insertado el esqueleto básico de la función, tal y como puede verse en la Figura 18.

```
#===============================================================================



#===  FUNCTION  ================================================================
#          NAME:  assert
#   DESCRIPTION:  Abort the script if assertion is false.
#    PARAMETERS:  expression     : assertion
#                 [linenumber]   : use $LINENO
#                 [functionname] : use $FUNCNAME
#       RETURNS:  99             : exit error status
#===============================================================================
function assert ()
{
        if [ ! $1 ]
                then
                        local linenumber=""
                        local functionname=""
                        [ -n "$2" ] && linenumber=": line $2"
                        [ -n "$3" ] && functionname=": function '$3'"
                        echo "File '$0' $linenumber$functionname: assertion '$1' failed."
                        exit 99
                        fi
}
~                                                                    43,1          Baix
```

Figura 19. Inclusión de la función *assert* de manera automática

Vim Online

Si estás en una PC editando un archivo de texto y no puedes aguantar la desesperación de no tener a Vi en tus manos, puedes recurrir a alternativas online. Con el furor de la "nueva era" de la Cloud *Computing*, cada vez son más las aplicaciones que se encuentran "en la nube" para poder ser usadas por cualquier usuario. Vim no hace excepción. En el sitio *http://gpl.internetconnection.net/vi/* se ofrece una versión online de este magnifico editor de texto, al que han apodado `jsvi`. El único requisito es tener un navegador web que soporte JavaScript. Este simulador de Vi funciona con casi todas las claves y comandos de sustitución habituales, y puedes copiar y pegar desde el portapapeles de tu ordenador.

- *Comentarios de funciones*. De un modo análogo a las cabeceras del script, podéis personalizar las cabeceras de comentarios de las funciones editando el archivo `bash-function-description`. Después de personalizar y traducir la cabecera con la secuencia `\cfu` podrás insertar un bloque de comentarios a la función.
- *Insertar Estructuras del Lenguaje de manera automática*. Mediante el comando `\s` (de statement) podemos insertar estructuras del lenguaje. La Tabla 3 resume su uso.
- *Insertar fragmentos de código predefinidos*. Para insertar rápidamente porciones de código que tecleamos muy a menudo, tenemos la combinación \nr que debemos usar desde el modo de Comando. Al pulsarla nos aparece un prompt pidiéndonos el nombre del fichero que contiene el pedazo de código que queremos insertar. Fácilmente podemos recorrerlos uno a uno pulsando la tecla de tabulación varias veces. Por defecto en el sistema se incluyen unos cuantos fragmentos (snippets), algunos de ellos muy interesantes. Los ficheros que almacenan estos fragmentos de código se encuentran en *~/.vim/bash-support/codesnippets/* de modo que se pueden modificar fácilmente para adaptarlos a nuestras preferencias, también podemos crear nuevos fragmentos o incluso crear fragmentos a partir del código que estemos creando.
  Para crear un snippet a partir de nuestro código primero deberemos marcar la porción de texto que queremos guardar (usando + v para entrar en el modo "Visual" y desplazando el cursor para seleccionar). A continuación teclearemos \nw. Se nos solicitará un nombre con el que guardará ese fragmento y que luego con \nr recuperaremos todas las veces que queramos.
  Un ejemplo del uso de Snippet puede verse en la Figura 19, donde usamos el comando `\nr assert` para incluir la función.
- *Obtener ayuda rápida sobre comandos internos de Bash*. Para poder lograr esto, nos situamos sobre el nombre de la función que queremos recibir ayuda y apretamos el comando `\hh`. Veremos en la parte superior de Vim la ayuda del manual sobre ese comando.

Muchas cosas más pueden lograrse con este fantástico plugin. Si desea aprender más, puede dirigirse a *http://lug.fh-swf.de/vim/vim-doc/bashsupport.html*.

Uso de Snippets

Haciendo una definición exacta, podemos decir que un snippet es "*una pequeña porción de* código *o texto de programación. Son utilizados generalmente para minimizar la repetición de códigos, hacer más claros los* algoritmos *o permitir que una aplicación genere el código automáticamente. Muchos editores de texto, editores de* códigos fuente, IDEs *y programas similares, permiten administrar snippets para facilitar las tareas al programador, especialmente en aquellas rutinarias. Los snippets pueden ser estáticos o dinámicos.*

Un snippet estático es simplemente texto que se utiliza una y otra vez sin cambiar nunca. Un snippet dinámico, depende de diferentes parámetros que coloca el programador, y se genera el código dependiendo de éstos.".

Siguiendo la clasificación de esta definición, en este apartado veremos un ejemplo de snippet dinámico. Un snippet estático es, por ejemplo, las abreviaciones (ver cuadro "*Tip: Abreviando Texto con Vim*").

Ahora bien, utilizaremos como ejemplo el snippet *snipMate*, que se encargará de autocompletar secciones repetitivas del código tales como ciclos, condicionales y constructores de funciones de manera sencilla. Algo muy útil a la hora de programar. Es válido aclarar que estos tipos de snippets nos brindan la posibilidad de crear los nuestros personalizados, cosa que también aprenderemos a realizar en este apartado.

En primer lugar, tenemos que descargar el snipMate. Podemos hacerlo desde esta URL *http://www.vim.org/scripts/script.php?script_id=2540*. Luego, procedemos a instalarlo, descomprimiendo el archivo descargado en el directorio *~/.vim*. ¡Listo! Ya podemos utilizar sus funcionalidades. Para ver la potencia de snipMate, creamos un archivo de ejemplo, que podría llamarse *main.c*, haciendo referencia al programa principal de un archivo programado en C. Una vez en él, entramos al modo inserción, y luego escribimos la palabra *main* y presionamos la tecla <TAB>. Veremos como automáticamente se nos completa la estructura de un programa en C. También, podemos escribir la palabra *for* y luego presionamos la tecla <TAB>, y veremos como se autocompleta la estructura for. ¡Excelente!

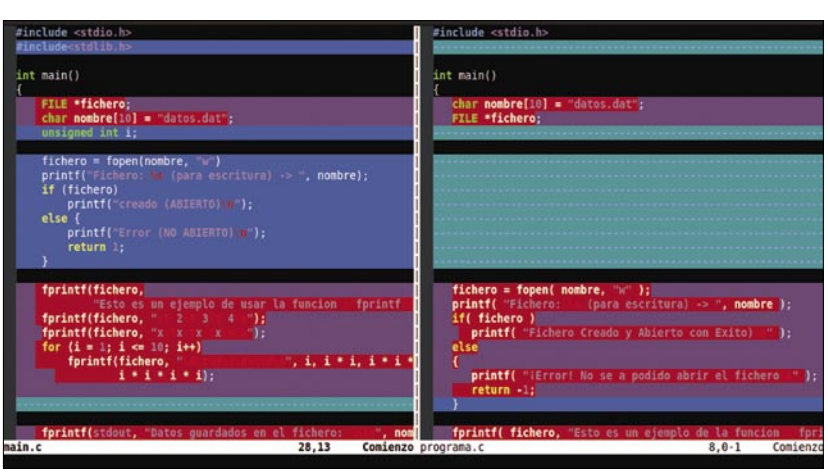

Figura 20. La herramienta vimdiff

Podemos ver un excelente screencast sobre snipMate en *http://vimeo.com/3535418*.

La Herramienta vimdiff

Todo buen programador sabe apreciar lo útiles que son las herramientas de comparación de archivos. Conociendo a *vimdiff*, lo sabrá apreciar aun más. Vimdiff es una herramienta basada en Vim que permite editar dos o más ficheros simultáneamente en tiempo real en busca de diferencias entre ellos. En concreto, abrirá en dos ventanas verticales dentro de vim los dos ficheros (o más) que se le pasen como parámetros, marcando con distintos colores las diferencias debidas a añadidos, borrados o similitudes entre ambos. También, permite el scroll sincronizado de ambas ventanas. Para usarlo, simplemente lo debemos invocar desde la shell con `vimdiff archivo1 archivo2 [archivo3 [archivo4]]` o de forma equivalente, podemos llamarlo directamente desde vim con la opción -d asi `vim -d file1 file2 [file3 [file4]]`.

Cuando llamemos a vimdiff, éste marcará las diferencias entre los archivos con tres colores diferentes. Por defecto, el color rosa son líneas que contienen alguna diferencia, las líneas con color azul oscuro indican que se han añadido, las líneas con azul claro muestran las que faltan, y las de color rojo las que cambian.

Como vimdiff es una herramienta que se empotra en Vim, podemos utilizar los mismos comandos de siempre, como por ejemplo para el movimiento entre ventanas que explicamos con anterioridad.

Eclim, integrando Eclipse con Vim

Eclipse es un IDE multiplataforma y libre para crear aplicaciones de cualquier tipo. Eclim es un impresionante proyecto que permite integrar las funcionalidades de Eclipse, con el mucho más modesto pero omnipresente editor de textos VIM. De manera que tan sólo ejecutando VIM podrás disponer de todas las ventajas que te proporciona Eclipse. Se instala de manera similar a cualquier plugin de Vim. De manera similar también tenemos jVim, pero desde la mano de NetBeans (*http://jvi.sourceforge.net/*) aunque no está tan potente.

Para probar Eclim, podemos descargarlo de *http://eclim.sourceforge.net/*.

Básicamente, los comandos propios de vimdiff que necesitamos saber para defendernos son:

- `[c` : Salta a la diferencia anterior
- `]c` : Salta a la diferencia siguiente

La Figura 13 muestra un ejemplo del uso de la herramienta vimdiff en acción.

Configurando el Archivo vimrc

En el archivo .vimrc podemos poner nuestras configuraciones específicas y concretas de Vim, tales como el resaltado de sintaxis, el numerado de línea, para no tenerlas que cargar manualmente cada vez que iniciamos Vim. Muchas cosas más que hemos visto a lo largo de este artículo podemos configurarlas en él. Si queremos que la configuración impacte sólo para nuestro usuario, modificamos el archivo *$HOME/.vimrc* o bien, si queremos afectar a todos los usuarios que usen Vim en el sistema, modificamos el archivo */etc/vim/vimrc*. De no existir estos ficheros, puede crearlos mediante el comando `touch`.

El fichero .vimrc no sólo permite especificar parámetros y opciones de arranque para Vim: es mucho más que eso. En él puedes programar en un lenguaje de programación interno de Vim nuestras propias funciones, macros, filtros de texto, etc., haciéndolo más productivo y personalizable a la hora de usarlo.

Las opciones de configuración son las mismas que vimos hasta el momento pero sin anteponer el signo de dos puntos (:). Es decir, si queremos activar el resaltado de sintaxis cada vez que se inicie Vim, tendremos que añadir al vimrc el comando `syntax on`.

A continuación se explican algunas de las opciones clásicas. Luego, en el Listado 1, podemos ver un ejemplo de vimrc orientado a la programación con su correspondiente explicación.

- `filetype plugin on`: Habilita el uso de plugins discriminándolos por tipo de archivo.
- `syntax on`: Habilita el resaltado de sintaxis.
- `set hlsearch`: Habilita el coloreado de las palabras encontradas en las búsquedas, en un color diferente del color del texto.
- `set backup`: Si está activada esta opción, cada vez que grabemos el fichero se almacenará una copia de la versión anterior como fichero~ (con el carácter '~' detrás).

Los comentarios se aplican con las comillas dobles (“), para conocer la totalidad de opciones de Vim y una explicación de cada una de ellas, puedes hacerlo mediante la ayuda incluida al respecto en Vim, que se despliega tecleando `:options`

Conclusión

Como puede ser fácilmente visto anteriormente, Vim es un potente y flexible editor de texto con un montón de características que pueden ser fácilmente añadidas a la distribución por defecto. No tiene absolutamente nada que envidiarle a los grandes IDEs, es más, muchas características de Vim son seguramente envidiadas por ellos. Como programadores pasamos el 90% del tiempo editando un archivo de código fuente, y el otro 10%, en el mejor de los casos, corrigiendo errores. Es por ello, que no existe excusa para no aprender a usar este magnifico “editor” de texto. Espero que el artículo les sea de utilidad, y puedan exprimirlo al 100%. Cualquier duda o consulta, pueden realizarla a *jorge.capurro@linuxmail.org*, que se las responderé lo antes posible. ¡Hasta la Próxima!

Sobre el autor

Jorge Emanuel Capurro es estudiante de la Tec. Superior en Programación, carrera dictada en la Universidad Tecnológica Nacional – Facultad Regional Haedo, provincia de Buenos Aires, Argentina. Principalmente, su área de investigación se centra en los Sistemas Operativos de tipo UNIX y de la programación bajo dicha plataforma.

Es el creador del proyecto IDEas (*http://ideasc.sourceforge.net*), que es el primer frontend desarrollado bajo Gambas del compilador gcc (*http://gcc.gnu.org*), que se utiliza con fines didácticos. Actualmente se desempeña como programador para la empresa argentina VATES S.A. “Ingeniería de Software – CMMI 5” (*http://www.vates.com/*) participando activamente en la Software Factory. Actualmente, se encuentra en el proyecto de edición de un libro dedicado a la Programación de Sistemas GNU/Linux.