Manual de uso del editor de texto `Vim`

Gabriel López

1 de octubre de 2022

1. `Vim`, un editor de texto minimalista.

`Vim` es un editor de texto creado por Brian Moolenaar. Es una versión mejorada del editor de texto `vi` creado por el programador Bill Joy. El editor de texto `Vim` es un editor de texto pensando para ser un programa de edición de texto minimalista y extensible, lo que significa que con ayuda de programas auxiliares hechos por la comunidad conocidos como *plugins*, se puede utilizar el editor `Vim` como un entorno de programación (conocido como *IDE*) o un entorno minimalista de edición de documentos hechos en LaTeX.
Si bien, aprender a usar `Vim` es un proyecto con un curva de aprendizaje un poco pronunciada, este editor ofrece ciertas ventajas por sobre otros editores, sobre todo a nivel de personalización, configuración y uso de recursos, lo que lo hace ídoneo en *hardware* de capacidad limitada. Además de esto, con la correcta configuración y el correcto uso de *plugins*, la escritura (sobre todo de documentos de tipo científico) puede ser más fluida a un nivel muy similar a la toma de notas manuscritas.

2. Entornos en `Vim`

`Vim` es un editor de texto poco ortodoxo comparado con el resto de los editores de texto que están disponibles para uso público, para empezar, `Vim` usa un esquema de entornos *entornos*, lo que quiere decir, que para las acciones que usualmente se ven relegadas al uso del ratón en otros editores, en `Vim` están relegadas a un *entorno*. Los entornos de uso básico son los siguientes:

- *NORMAL*: Este entorno es el entorno básico de `Vim`. En este entorno se utiliza para navegar entre líneas de texto, borrar contenido y introducir comandos dentro del editor. La utilidad del modo *NORMAL* radica en el uso de comandos para usar la terminal y el editor al mismo tiempo.
- *INSERTAR*: Este entorno se utiliza para la inserción de texto dentro de líneas en el editor. Para entrar a este entonro basta con introducir la tecla `i` mediante el teclado. Una vez que la tecla se ha introducido, el editor colocará el texto `INSERTAR` o `INSERT` en la parte inferior de la pantalla, denoatando que el modo de inserción está activado.
- *VISUAL*: Este entorno se encarga de función de copiar y pegar contenido de las lineas. Para entrar a este modo, se debe introducir `v` con el teclado. Una vez que el modo *VISUAL* está activado debe aparecer en pantalla la leyenda `VISUAL` en la esquina inferior izquierda.

Es importante destacar que para salir de un entorno para entrar de nuevo al entorno *NORMAL* se introduce la tecla `Esc` con el teclado. Para ilustrar los entornos de `Vim` colocamos la siguiente imagen. En dicha imagen es notorio que debido al idioma con el que se tiene el ordenador, aparece la leyenda `INSERTAR` en la parte inferior, denotando que el modo de inserción de texto está activo.

Figura 1: `Vim` en el modo *INSERTAR*

A pesar de todo esto, existen muchos más entornos dentro del editor, sin embargo, consideramos que estos tres son los más útiles para un usuario nuevo de `Vim` debido a que facilitan la escritura de documentos y permiten la introducción de comandos dentro del mismo.

3. ¿Cómo salir de `Vim`?

Es normal que el usuario novato de GNU-Linux o cualquier sistema operativo derivado de UNIX. De modo que el sistema operativo a veces deja el usuario dentro de `Vim` sin ningún aviso de cómo salir del editor de texto. Por esta razón en esta sección se explicará como salir de `Vim`.
Para empezar, la imagen inicial que el usuario tiene al entrar a `Vim` es la siguiente:

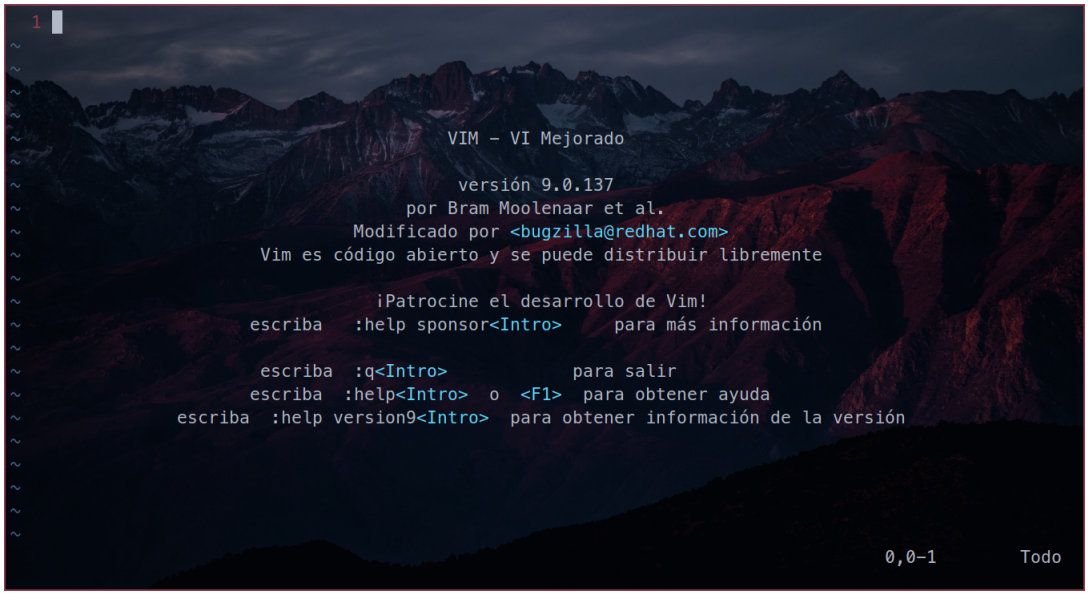

Figura 2: Primera impresión de `Vim`

Podría parecer que `Vim` es complicado, pero en realidad, salir del editor es relativamente sencillo, la secuencia de pasos para salir de `Vim` es la siguiente:

- Primero, si usted desea guardar el contenido del archivo, es conveniente guardar el contenido del archivo primero, esto se logra mediante el comando `:w` dentro del entorno *NORMAL*.
- Ahora que el archivo está guardado, es posible salir del editor sin problemas, por lo que se procede a colocar el comando `:q` dentro del mismo entorno *NORMAL*.

Si bien es posible que queramos guardar el contenido de nuestro archivo de texto, también tenemos la opción de salir sin guardar (no recomendada a menos que se desee salir de `Vim` si se entró por error), en este caso, basta con introducir el comando `:q!` en el modo *NORMAL* del editor.

4. ¿Existe alguna forma de aprender `Vim` mediante ejemplos?

Si bien `Vim` es un editor de texto que puede parecer poco ortodoxo al principio, existen formas para aprender su uso de una forma interactiva mediante ejemplos. En el caso de `Vim` existe el complemento denominado `vimtutor`, que es un complemento de consola que suele estar instalado

en cualquier máquina UNIX. En el caso de querer acceder a dicho complemento, basta con introducir el comando `vimtutor` dentro de la terminal de GNU-Linux o su equivalente en UNIX.

5. Navegación en `Vim`

`Vim` es un editor de texto que tiene un esquema de navegación diferente al resto de editores de texto que podemos encontrar en el mercado. Esto es debido a cuestiones de convención. Si bien, las teclas direccionales del teclado funcionan para navegar entre texto dentro del editor, su uso no es recomendado por la mayoría de guias de uso de `Vim`.

5.1. Navegación básica en `Vim`

Esto se entiende mejor con los ejemplos proporcionados por el complemento de consola llamado `vimtutor`. Para nuestros efectos, se busca introducir al usuario a los conceptos básicos de los que `Vim` para que sea capaz de redactar documentos simples dentro del editor. Dicho esto, podemos mostrar el esquema básico de navegación del que hace uso `Vim`.

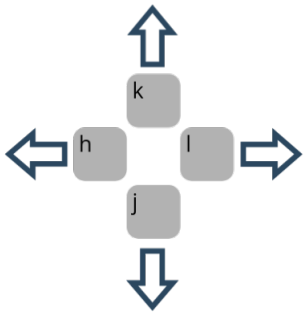

Figura 3: Esquema de navegación básico dentro de `Vim`

Para resumir, podemos ver que las teclas de navegación son las siguientes:

- `k` y `j` son usadas para subir y bajar de linea, respectivamente.
- `h` y `l` son usadas para desplazarse de izquierda a derecha, respectivamente.

Como se puede ver, el esquema de navegación dista de los editores convencionales. Esto es por diseño, ya que `Vim` hace uso de estas mecánicas de movimiento para tratar dicho movimiento como si fuera un *comando de consola* y esto permite aplicar ciertos comandos, como el de *borrar* al movimiento para alcanzar cierta velocidad a la hora de borrar contenido dentro de una línea de texto.

5.2. Otras opciones de movimiento en `Vim`

Como se mencionó anteriormente, `Vim` trata el movimiento como si fuera un comando de consola, esto permite aplicar ciertos efectos al movimiento. Si bien, el movimiento básico se resume con las teclas, `h,j,k` y `l`, existen también cuatro comandos útiles que deben ser mostrados para que este manual se pueda considerar una referencia útil para el uso de `Vim`. En general, `Vim` suele trabajar por *palabras*cuando se habla de navegación dentro de una linea de texto, entonces existen formas para desplazarse en una linea de texto de palabra en palabra. En este caso, existen cuatro comandos destinados para esta función:

- El comando `w` permite al usuario desplazarse desde su localización actual hacia el *inicio* de la siguiente palabra en la linea de texto.
- El comando `e` permite al usuario desplazarse desde su actual ubicación, hacia el *final* de la siguiente palabra en la linea actual de texto.
- Se usa el número `0` para desplazarse hacia el *inicio* de la linea.
- Finalmente, el comando `$` se usa para desplazarse hacia el *final* de la linea.

6. La función de *Borrar* en `Vim`

Como cualquier editor de texto plano, usualmente queremos editar el contenido de nuestro archivo, ya sea cambiando algunas lineas dentro del archivo o *borrando* contenido dentro de las líneas. En `Vim` la función de *borrar* dentro del documento está relativamente ligada al uso de movimiento.
Esto es debido a que podemos pensar en lo siguiente: la función de borrar se activa con el comando `d` y la cantidad de contenido a borrar se determina usando los comandos de movimiento vistos anteriormente. Por lo que, para ilustrar algunos de los usos de este mecánica, es conveniente colocar algunos de los comandos más utilizados con esta modalidad.

- El comando `dw` permite borrar todo lo que está desde *la ubicación actual hasta el inicio de la siguiente palabra.* Como ven, el comando de *borrar* se antepone a la opción de movimiento.
- Introduciendo `de` podemos borrar todo aquello que está comprendido desde *la localización del puntero en el editor, hasta el final de la siguiente palabra en la linea.*

Como podemos observar, el comando de *borrar* en `Vim` usualmente está antes de la opción de movimiento. Sin embargo, debido a razones que los creadores de `Vim` comentan en `vimtutor` es común que se requiera *borrar una o más lineas completas.*
En este caso, la función de borrar es mucho más sencilla, solamente basta con introducir el comando `dd`. Es en este punto en donde es conveniente introducir una figura que está presente también en `Vim` y que al día de hoy, no hemos visto en otros editores de texto y que, al día de hoy, no hemos visto en otros editores de texto, esta característica se conoce como *contadores.*

6.1. Contadores

Los *contadores* son una característica de `Vim` que permite al usuario repetir un comando un determinado número de veces, usualmente `vimtutor` relega la lección de esta característica hasta

que ya se han visto los comandos básicos de movimiento y borrado y en este caso, es conveniente explicar un poco el porqué. Supongamos lo siguiente: Se desea borrar un número determinado de líneas en el editor, si se usaran otros editores, se podría pensar en usar el ratón para seleccionar las líneas y borrarlas con la tecla `backspace`.
Sin embargo, en `Vim` se debe aplicar el uso de los comandos vistos anteriormente. Un ejemplo simple sería el borrado de 4 lineas en un archivo de texto. En este caso, el comando a utilizar es: `4dd`. La explicación del comando es la siguiente:

- `4` es el número de veces que se repetirá el siguiente comando.
- `dd` es el comando usado para borrar una linea completa en el documento de texto.

Este es el uso más sencillo de los contadores en `Vim`, sin embargo, su uso no se limita solamente a esto, y es recomandable revisar la sección de contadores en el complemento `vimtutor` para una mejor comprensión del uso de esta característica del editor.

7. Configuración del archivo `.vimrc`

`Vim` es un programa de UNIX, lo que significa que como una gran mayoría de los programas de terminal que se usan en esta plataforma, este usa un archivo de configuración. Este archivo se encuentra en el directorio `$HOME` de la terminal de GNU-LINUX, con el nombre de `.vimrc`. Como se puede observar `.vimrc` es un archivo oculto, por lo que es necesario cambiar las opciones por defecto en su gestor de archivos o introducir el comando `ls -a` en la terminal.
El archivo `.vimrc` es util para configurar opciones globales de `Vim`, estas sobre todo se refieren a *qué* hacer con ciertos archivos, sobre todo a aquellos que tienen un tipo de extensión particular como ser `.tex, .c, .py`, etc. Sobre todo, `Vim` facilita ciertas cosas útiles para el usuario al momento de editar estos archivos, como ser:

- *Syntax-highlighting*: que es básicamente colocar colores a ciertas expresiones para diferenciar entre comandos y texto plano. Esto es particularmente útil con archivos de LaTeX, dado que permite una mejora en la legibilidad del archivo.
- *Numeración de las líneas*: como en los editores de texto modernos, `Vim` tiene la opción de asignar números a las líneas de texto. Esto facilita el *depuración* del código independientemente del lenguaje de programación, porque permite revisar la línea que tiene errores dentro del archivo de reporte de errores.
- *Instalación de Plugins*: `Vim` permite configurar el editor para mejorar ciertas funcionalidades referentes a ciertos archivos de texto. Los plugins que serán usados en este manual son referentes a la integración de LaTeX con `Vim`. Para la instalación de estos plugins, se suele incluir una línea dentro del `.vimrc` para indicar qué plugin se procede a instalar.

Para mostrar algunas de las posibles configuraciones que pueden hacerse en el archivo `.vimrc` sin hacer instalaciones avanzadas (eso será tratado en un momento posterior en este manual) se pasa a mostrar una configuración básica del archivo `.vimrc`. Esta es la configuración que el autor de este documento usa hasta el momento, sin embargo, es necesario que se comprenda que muchas de las lineas que aparecen en este documento están relacionadas a los *plugins* que el autor del documento mostrará más adelante.

7.1. Configuración básica de un archivo `.vimrc`

```
"Esto es un comentario en el archivo .vimrc"

"El siguiente comando es para colocar numero de línea al lado del editor"
:set number
"Comando para hacer ajuste del texto al ancho de ventana"
:set wrap
"Comando usado para la instalación de plugins"
call plug#begin('~/.vim/plugged')

"INSTALACION DE PLUGINS"
"Se instalan multiples plugins con el comando Plug #nombre
"seguido del caracter |"
Plug 'SirVer/ultisnips' | Plug 'honza/vim-snippets'
Plug 'xuhdev/vim-latex-live-preview', { 'for': 'tex' }
Plug 'lervag/vimtex'
"Mas adelante se aprenderá para qué sirven los comandos situados"
"Despues del :set wrap"
```

Como podemos ver, en realidad, las posibilidades de configurar el archivo `.vimrc` son infinitas, sobre todo porque ocultamos (por brevedad) el resto de la configuración de *plugins* que se hacen dentro del archivo mencionado. Considerando esto, podemos ver que el archivo `.vimrc` es un archivo fundamental para poder configurar a `Vim` para poder realizar tareas productivas tanto como un *IDE* (para programación) o como editor de documentos de LaTeX. Hay que destacar que la configuración anterior es una configuración sumamente básica, es común que los usuarios avanzados de `Vim` tengan archivos de configuración de más de cien líneas.

8. Instalación de *Plugins*

`Vim` es un editor de texto muy básico, pero su principal ventaja es su capacidad de ser *extensible* lo que permite agregar funciones a `Vim` que pueden ser:

1. *Syntax Highlighting* para los distintos lenguajes de programación que se usen para desarrollo de software en `Vim`.

2. Configuración de atajos de teclado *para cada uno de los lenguajes de programación usados en* `Vim`, un buen ejemplo de esto puede ser el que, si tomamos a LaTeX como un ejemplo, es muy común tener que escribir `"begin"` para colocar un entorno como `equation` o `align`. Por lo que es posible configurar un atajo de teclado para evitar la escritura completa de este comando.

Los *Plugins* en `Vim` se pueden instalar de dos formas: manual y utilizando un *plugin-manager*. Debido a cuestiones más prácticas, vamos a centrarnos en la última forma de instalar los *plugins*. En este caso, consideramos que es más conveniente revisar la documentación del *plugin-manager* que el autor de este manual recomienda, en este caso, se recomienda empezar usando el *manager* conocido como **Vim-Plug**.
Comentario: Como muchas de las instrucciones de instalación de software en Linux

suelen cambiar con el tiempo, consideramos más conveniente que el lector revise las instalaciones de instalación de Vim-Plug en el repositorio de GitHub de este componente software. El repositorio de **Vim-Plug** se encuentra en la siguiente dirección web.

`https://github.com/junegunn/vim-plug`

Una vez que se ha instalado el gestor de plugins *Vim-Plug*, es posible que si requiere instalar plugins de utilidad para este manual, usted requiera cambiar algunas cosas en el archivo `.vimrc`. Los plugins de interés que el autor usa en este manual se encuentran en la subsección 7.1 de este manual. Mostramos aquí cómo es el proceso de instalación de los plugins que el autor de este manual usó para la creación de este documento.

```
"Este comando es para inicializar a Vim-Plug"
call plug#begin('~/.vim/plugged')
"Aquí le damos a Vim-Plug una lista de plugins a instalar"

"PLUGIN DE ATAJOS DE TECLADO PARA LaTeX"
Plug 'SirVer/ultisnips' | Plug 'honza/vim-snippets'

"PLUGIN PARA COMPILAR DOCUMENTOS EN TIEMPO REAL"
Plug 'xuhdev/vim-latex-live-preview', { 'for': 'tex' }

"ESTE ULTIMO PLUGIN TIENE SU PROPIA SECCION EN EL MANUAL"
Plug 'lervag/vimtex'
```

Una vez que tengamos lo que se mostró en la sección de código anterior, es posible instalar todos estos plugins usando el comando `:PlugInstall` en el modo `NORMAL` de `Vim`, esto considerando que se requiere de una correcta conexión a internet para poder instalar de dichos plugins. Además de esto, se le recomienda al usuario investigar por su cuenta acerca de los otros plugins que existen para otros lenguajes de programación, debido a que este manual solo se centra en aspectos básicos de `Vim` y su integración con LaTeX.

9. Uso básico de `Vimtex`

Hemos hablado de algunos plugins que son usados para integrar el editor de `Vim` con LaTeX. En este caso, podemos hacer uso del *Plugin* conocido como `Vimtex`, que es una suite de plugins para `Vim` que está orientado a integrar a `Vim` con LaTeX. Las instrucciones de instalación de `Vimtex` de nuevo, cambian con las constantes actualizaciones que se le hacen al software, por lo que recomendamos revisar el repositorio de GitHub para `Vimtex`. El repositorio de GitHub de `Vimtex` se encuentra en la siguiente dirección web:

`https://github.com/lervag/vimtex`

Una vez que se instale `Vimtex` en su ordenador, consideramos que es útil explicar los siguientes comandos y carácteristicas de `Vimtex` que pueden ser de utilidad para el usuario.

9.1. Configuración del archivo `.vimrc` orientado al uso de `Vimtex`

Como `Vimtex` es una suite de paquetes útiles para integrar `Vim` con LaTeX, es necesario realizar algunos cuantos cambios al archivo `.vimrc` para que el plugin de `Vimtex` esté cargado cuando se ejecuta el comando `Vim` en la terminal y se requiere editar un archivo con extensión `*.tex` en la terminal. En realidad, muchas de las instrucciones acerca del cómo se debe colocar esta configuración están en elrepositorio de `Vimtex`, sin embargo, el autor de este manual si encontró algunos inconvenientes al principio para poder integrar `Vim`con esta suite, por lo que se considera conveniente explicar a fondo esta sección.
La configuración que presenta el repositorio de `Vimtex` es la siguiente:

```
1  " This is necessary for VimTeX to load properly. The "indent" is optional.
2  " Note that most plugin managers will do this automatically.
3  filetype plugin indent on
4
5  " This enables Vim's and neovim's syntax-related features. Without this, some
6  " VimTeX features will not work (see ":help vimtex-requirements" for more
7  " info).
8  syntax enable
9
10 " Viewer options: One may configure the viewer either by specifying a built-in
11 " viewer method:
12 let g:vimtex_view_method = 'zathura'
13
14 " Or with a generic interface:
15 let g:vimtex_view_general_viewer = 'okular'
16 let g:vimtex_view_general_options = '--unique file:@pdf\#src:@line@tex'
17
18 " VimTeX uses latexmk as the default compiler backend. If you use it, which is
19 " strongly recommended, you probably don't need to configure anything. If you
20 " want another compiler backend, you can change it as follows. The list of
21 " supported backends and further explanation is provided in the documentation,
22 " see ":help vimtex-compiler".
23 let g:vimtex_compiler_method = 'latexrun'
24
25 " Most VimTeX mappings rely on localleader and this can be changed with the
26 " following line. The default is usually fine and is the symbol "\".
27 let maplocalleader = ","
```

Esta configuración puede copiarse y pegarse al archivo `.vimrc` sin ningún problema, el tema aquí es que la configuración aparenta tener algunas cosas que no suelen venir instaladas de fábrica en una instalación común de GNU-LINUX, como ser:

- La configuración hace uso de un lector de pdf conocido como `zathura`, este lector suele ser usado por entusiastas de Linux que buscan un lector minimalista con atajos de `Vim`, sin embargo, no viene instalado de fábrica en muchas de las instalaciones de Linux, por lo que se le recomienda al usuario de este manual que comente o borre la línea 12 a conveniencia.

- Además de esto, si utiliza distribuciones de Linux como Ubuntu o alguna distribución de UNIX como MacOS, se recomienda cambiar opción de la linea 15. Se recomienda cambiar la opción de `okular` por `evince`. Estos dos últimos son lectores de archivos pdf con interfaz gráfica que suelen estar instalados de fábrica tanto en Ubuntu como en MacOS.

La configuración está toda en inglés, pero el detalle de importancia es el siguiente: La línea que habla del *compilador* de LaTeXpuede presentar problemas para el principiante, esto es porque muchas de las instalaciones de LaTeX en GNU-LINUX suelen venir con el comando `latexmk` por defecto, este es un compilador más maduro y de uso general, por lo que es muy probable que se tenga que cambiar la linea 23 del pesudocódigo anterior con la siguiente línea:

```
"Esta es la linea que debe ser cambiada."
let g:vimtex_compiler_method = 'latexmk'
```

9.2. Comandos básicos de `Vimtex`

`Vimtex` es un plugin que funciona como una extensión de comandos utiles para el usuario de `Vim`, estos comandos van relacionados al uso de `Vim` como un editor completo de LaTeX. Es por eso que consideramos que es prudente explicar algunos de los comandos básicos de `Vimtex`. Algunos de estos comandos son de utilidad, sobre todo para el usuario de `Vim` y LaTeX que tienen que ver con la compilación del documento.
Tenemos los siguientes comandos básicos:

- Se usa `,ll` para compilación en *modo continuo*, lo que quiere decir que cada vez que se guarde el documento de texto con el comando `:w` se procede a compilar el contenido completo del documento y mostrar una previsualización de este documento en un lector de archivos pdf de su elección.
- Se utiliza el comando `,lk` para cerrar el modo de compilación en *modo continuo*.
- Es común que cuando se compile un archivo de extensión `*.tex` aparezcan muchos archivos que se le denominan *archivos auxiliares*, si lo desea, el usuario puede borrar dichos archivos usando el comando `,lc`.
- Existe también la opción de ver el contenido compilado (por ejemplo, texto compilado) de una linea del `*.tex`, en este caso, se encuentra dicho contenido con el comando `,lv`.

9.3. Revisión de errores con `Vimtex`

Algo muy común con el usuario principiante de LaTeX es que al cambiar un par de lineas en su documento se encuentre en la penosa situación de obtener un `*.tex` que no compila. Muchas de las veces, en la humilde opinión del autor de este manual, estos errores pueden solventarse fácilmente si uno mira los errores que le arroja la consola al usuario.
`Vimtex` es particularmente util para este tipo de corrección de errores y *bugs* dentro del archivo de texto debido a que tiene integrada la capacidad de mostarle al usuario los errores en tiempo real mediante un menú que viene por defecto en `Vim`, el menú `QuickFix` este menú es simplemente una pestaña de `Vim` que se abre de forma paralela a la ventana principal y le permite al usuario tener una vista rápida de los errores que tiene el documento (si es que los tiene).

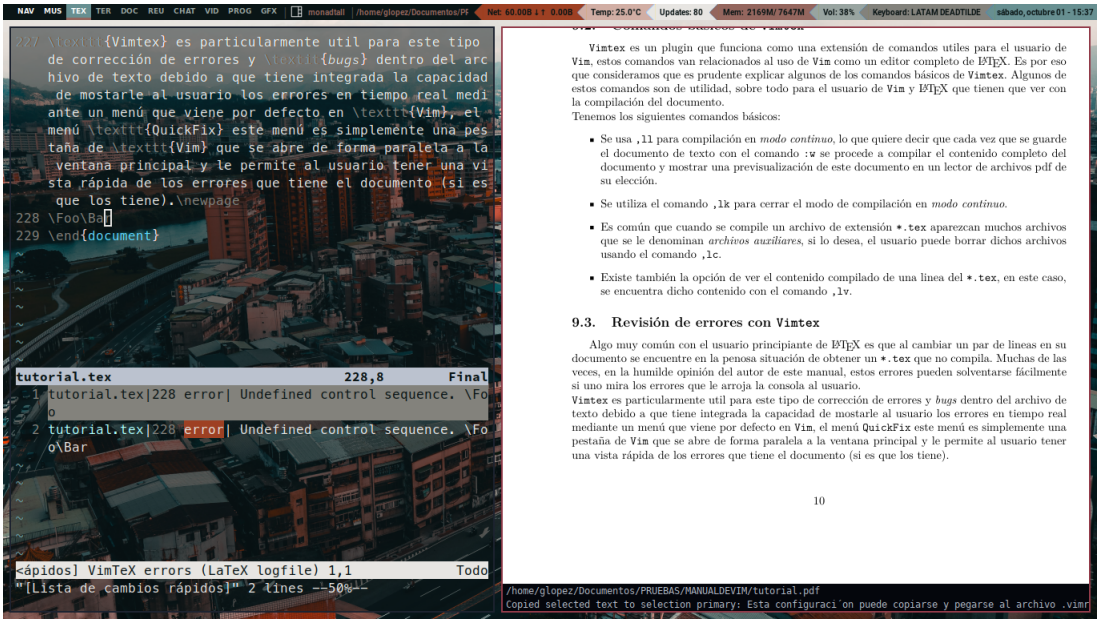

Figura 4: Vista del menú quickmenu, note que el error de compilación es muy simple. Resulta que los comandos `Foo` y `Bar` que se encuentran en la línea 228 del documento no están definidos.

Puede observar que el menú está en la parte baja de la ventana de `Vim` y nos comenta que existe un error en la linea 228 del documento `tutorial.tex` que estamos editando. Además de esto, se nota que en la Figura 4. se observan las bondandes de usar `Vimtex` para la compilación de estos documentos,se nota que la ventana de la derecha es el documento siendo compilado en tiempo real, lo que le permite al usuario ser más productivo dado que la compilación suele ser un elemento de trabajar en LaTeX que suele tomar su tiempo.
Finalmente, le recomendamos al usuario revisar la documentación de `Vimtex` en su respectivo repositorio de GitHub. Esto con el objetivo de que el lector pueda realizar operaciones mucho más complicadas con este plugin.

9.4. ¿Es posible moverse entre los diferentes errores de LaTeX en un documento?

Es normal que un documento de LaTeXtenga errores en distintas partes del mismo. Por eso, `Vimtex` agrega al menú *QuickFix* la posibilidad de moverse entre diferentes partes del documento, localizar los errores y corregirlos conforme se vaya necesitando. Para lograr esto, se hace lo siguiente:

- Al crearse un error en el documento, se entra al menu *QuickFix* con el comando `:copen`, esto permite saltar de la venta principal al menú y poder desplazarse en el mismo.
- Presionando la tecla `Enter` en el error situado dentro del menú *QuickFix* se saltará a la línea que requiera corrección. Esto facilita al usuario revisar si ha escrito algo mal y depurar su

archivo más rápido.

- Una vez en la linea de destino, cierre la selección del menú *QuickFix* usando el comando `:cw` de `Vim`.
- Haga la corrección pertinente.

Esto se realiza por cada línea que se desea corregir, es interesante observar que cuando se trabaja así, se exige que al trabajar en un `*.tex` se manejen buenas prácticas de creación de documentos para tener la menor cantidad de errores posibles. Por lo que el usuario se entrena para poder escribir documentos de LaTeXque contengan pocos errores, lo que mejora sus capacidades de creación.

10. Atajos de Teclado para comandos de LaTeX

Hasta ahora, tenemos una instalación de `Vim` que permite hacer dos cosas principales:

1. Editar archivos de texto plano.
2. Crear, editar y compilar en tiempo real archivos de extensión `*.tex`.

Ahora bien, todo esto es bastante sólido en cuento a productividad, pero quizá queremos aumentar la cantidad de trabajo que se desea producir. Una forma de hacer esto es evitar tener que escribir el comando de LaTeX completo. Para ilustrar esto, consideremos el siguiente ejemplo:

```
\begin{figure}
\begin{center}
        \includegraphics[scale=]{Figures/}
\end{center}
\caption{}
\label{fig:}
\end{figure}
```

Es evidente pensar que escribir todo esto es particularmente ineficiente, entonces nos preguntamos *¿Existe alguna forma para poder evitar escribir toda la sintaxis del comando?* Resulta que sí, en este caso, existe un plugin de `Vim` que nos permite usar lo que se conoce en la comunidad como los *snippets*, en este caso, los snippets son atajos de teclado que permiten escribir todo el contenido de un comando o frase usando pocas letras.
De nuevo, el plugin para usar los snippets debe instalarse ya sea usando **Vim-Plug** o de forma manual. El presente manual presenta la siguiente ventaja: si usted colocó la configuración dada en la sección 8 en su `.vimrc`, ya debería de tener instalado el plugin para poder hacer uso de los snippets. Sin embargo, para poder usar los snippets (con el plugin *Ultisnips*) es necesario que su sistema cumpla los siguientes requisitos:

1. Su sistema debería de tener una instalación de `Python3` funcional.
2. Su versión de `Vim` debería de tener la opción de compilado de `Python` activada. Para saber si su versión tiene esta característica, coloque el siguiente comando dentro de `Vim`:

   ```
   :echo has("Python3")
   ```

   En el caso de que su versión de `Vim` sea la adecuada, el comando le debería de arrojar un `1` como respuesta en la parte inferior de su ventana.

Una vez que tenga todo listo, primero, debe insertar lo siguiente en su `.vimrc`

```
"La tecla para activar los snippets"
let g:UltiSnipsExpandTrigger       = '<Tab>'

"Para usar tab y moverse entre snippets "
let g:UltiSnipsJumpForwardTrigger  = '<Tab>'

"Para usar shift-tab y moverse hacia atrás entre snippets"
let g:UltiSnipsJumpBackwardTrigger = '<S-Tab>'

"Para colocar el directorio de los snippets"
let g:UltiSnipsSnippetDirectories=[$HOME.'/.vim/UltiSnips']
```

Para `Vim`, es común que los snippets se activen con una sola tecla del teclado, en este caso, se recomienda comenzar usando la tecla `tab` para el uso inicial de los snippets, si luego se desea cambiar el uso de estos, basta con remover la instrucción `<Tab>` que se ve anteriormente por la tecla de su preferencia. Ya dicho esto, podemos pasar a la parte de configuración de snippets de uso general.

10.1. Configuración de Snippets

Para configurar los Snippets de uso local (para su instalación de Linux) se recomienda primero ir al siguiente directorio.

```
${HOME}/.vim
```

Una vez en el directorio oculto de `Vim` debe observar que tiene que existir el siguiente árbol de directorios en esta carpeta:

```
${HOME}
└─ .vim
   └─ UltiSnips
      └─ tex
```

Si no está alguna de esas carpetas, se le recomienda crearlas usando los comandos de creación de carpetas de Linux. Una vez que tenga este árbol de directorios, deberá crear un archivo en la carpeta `.tex`, que puede llamar como guste, seguido de la extensión `.snippets`. Este archivo se utiliza para editar los snippets que el usuario estime convenientes para su flujo de trabajo. Al final debería de tener algo como esto en su arbol de directorios:

```
${HOME}
└─ .vim
   └─ UltiSnips
      └─ tex
         └─ archivo.snippets
```

Con todo esto hecho, solo resta editar (usando `Vim`) el archivo `archivo.snippets` de modo que se pueda editar un snippet muy básico y se muestren las capacidades del plugin *UltiSnips*. Dicho esto, se procede a mostrar la estructura básica de un snippet de `Vim`.

```
snippet {bandera} ["descripción" [opciones]]
{cuerpo snippet}
endsnippet
```

Esto puede parecer un poco confuso, por lo que se procede a explicar cuales son los parámetros de importancia.

- La *bandera* es la combinación de letras que seguidas de la tecla `tab`, permite correr el snippet.
- El *cuerpo snippet* es la pieza de codigo/frase/texto que se quiere escribir mediante el atajo de teclado.

Consideramos conveniente ilustrar esta teoría mediante un ejemplo, por lo que el autor de este manual comparte algunos de los snippets que ha creado para su uso personal:

```
snippet tt "El comando de texto de terminal"
\texttt{$1}$0
endsnippet

snippet ti "El texto en italica"
\textit{$1}$0
endsnippet

snippet tb "El texto en negrita"
\textit{$1}$0
endsnippet

snippet ff "El comando de fraccion"
\frac{$1}{$2}$0
endsnippet

snippet sec "El comando de seccion"
\section{$1}$0
endsnippet

snippet ssec "El comando de subseccion"
\subsection*{$1}$0
endsnippet

snippet env "el comando para entornos"
\begin{$1}
        $2
\end{$1}
$0
endsnippet
```

```
snippet fig "El comando para figura"
\begin{figure}
        \centering
        \includegraphics[$1]{$2}
        \caption{$0}
\end{figure}
endsnippet
```

El lector ahora quizá se esté preguntando *¿Qué son esos simbolos de dolar seguidos de un número?* En sí, esos símbolos pueden entenderse como la zona en la que se desea que quede el cursor cuando se active el snippet. Es decir, utilizando la tecla `tab` podemos movernos entre el comando de LaTeXpara colocar los parámetros que queremos editar. Un buen ejemplo es el del snippet con *bandera* `env`, que visto más de cerca es el siguiente:

```
snippet env "el comando para entornos"
\begin{$1}
        $2
\end{$1}
$0
endsnippet
```

En este caso, se nota que el símbolo `$1` está presente tanto en el `begin` como en el `end` del comando. Esto tiene el siguiente uso: podemos escribir el entorno de LaTeXque queremos *al mismo tiempo* si configuramos el snippet de la forma que se mostró con anterioridad, por lo que no es necesario escribir el entorno en el `begin` como en el `end`, dado que esta configuración permite escribir la frase al mismo tiempo en los dos lugares marcados por el símbolo `$1`.

11. Algunas recomendaciones finales

Si bien `Vim` es un editor de texto potente, este tiene una curva de dificultad relativamente pronunciada. Esto no es malo, simplemente se necesita de tiempo para poder acostumbrarse al flujo de trabajo que este software requiere del usuario para poder ser utilizado a su máximo potencial. Se recomienda antes de realizar todo el trabajo de configuración visto en este manual, primero, revisar el complemento `vimtutor` para poder tener cierta soltura con los comandos básicos de `Vim` y la navegación básica que este software ofrece. Además de esto, si bien es cierto que este manual va orientado a integrar `Vim` con LaTeX, es necesario recalcar que en realidad, `Vim` es una herramienta de uso general y puede configurarse para usarse como *IDE* de cualquier plataforma de programación, por lo que es de principal utilidad a aquellos desarrolladores que desean un *IDE* minimalista y veloz. Por lo comentado anteriormente, el autor de este manual exhorta al lector a revisar otros usos de `Vim` orientados más bien al desarrollo software.