



EBook Gratis

APRENDIZAJE

vim

Free unaffiliated eBook created from
Stack Overflow contributors.

#vim

Tabla de contenido

Acerca de.....	1
Capítulo 1: Empezando con vim.....	2
Observaciones.....	2
Versiones.....	3
Examples.....	4
Instalación.....	4
Instalación en Linux / BSD.....	4
Distribuciones basadas en arco y arco.....	4
Distribuciones basadas en Debian y Debian.....	4
Distribuciones basadas en Gentoo y Gentoo.....	4
RedHat y distribuciones basadas en RedHat.....	4
Fedora.....	4
Distribuciones basadas en Slackware y Slackware.....	5
Distribuciones basadas en OpenBSD y OpenBSD.....	5
Distribuciones basadas en FreeBSD y FreeBSD.....	5
Instalación en Mac OS X.....	5
Instalación regular.....	5
Gerente de empaquetación.....	5
Instalación en Windows.....	6
Chocolatey.....	6
Construyendo Vim desde la fuente.....	6
Saliendo de Vim.....	6
Explicación:.....	7
Tutoriales interactivos de Vim (como vimtutor).....	7
Guardando un archivo de solo lectura editado en Vim.....	8
Explicación del comando.....	8
Suspendiendo vim.....	8
Lo esencial.....	9
Qué hacer en caso de accidente.....	11
Capítulo 2: :global.....	13
Sintaxis.....	13

Observaciones.....	13
Examples.....	13
Usó básico del Comando Global.....	13
Tirar de cada línea que coincide con un patrón.....	13
Mover / recoger líneas que contienen información clave.....	14
Capítulo 3: Ahorro.....	15
Examples.....	15
Guardar un búfer en un directorio no existente.....	15
Capítulo 4: Asignaciones clave en Vim.....	16
Introducción.....	16
Examples.....	16
Mapeo básico.....	16
mapa general.....	16
operador de mapas.....	16
comando de mapa.....	17
Ejemplos.....	17
Combinación de teclas de líder de mapa.....	17
Ilustración de mapeo básico (atajos prácticos).....	18
Capítulo 5: Autocomandos.....	19
Observaciones.....	19
Examples.....	19
Fuente automática .vimrc después de guardar.....	19
Actualice las vistas de vimdiff si un archivo se guarda en otra ventana.....	20
Capítulo 6: Buscando en el buffer actual.....	21
Examples.....	21
Buscando un patrón arbitrario.....	21
Buscando la palabra debajo del cursor.....	21
Ejecutar comando en líneas que contienen texto.....	22
Capítulo 7: Código de auto-formato.....	23
Examples.....	23
En modo normal:.....	23

Capítulo 8: Comandos de modo normal	24
Syntaxis.....	24
Observaciones.....	24
Examples.....	24
Ordenar texto.....	24
Clasificación normal	24
Clasificación inversa	24
Clasificación insensible a mayúsculas	24
Clasificación numérica	24
Eliminar duplicados después de ordenar	24
Combinando opciones	25
Capítulo 9: Comandos en modo normal (Edición)	26
Examples.....	26
Introducción - Nota rápida sobre el modo normal.....	26
Deshacer y rehacer básico.....	26
Deshacer.....	26
Rehacer.....	26
Repetir el ultimo cambio.....	27
Copiar, Cortar y Pegar.....	27
Registros.....	28
Mociones.....	28
Copiado y corte.....	28
Paliza.....	29
Entonces, ¿cómo realizo un corte y pegado realmente simple?.....	29
Terminación.....	30
Capítulo 10: Cómo compilar Vim	32
Examples.....	32
Compilando en Ubuntu.....	32
Capítulo 11: Complementos	33
Examples.....	33
Fugitivo vim.....	33

NERD Tree.....	33
Capítulo 12: Complementos de tipo de archivo.....	34
Examples.....	34
¿Dónde poner los plugins de filetype personalizados?.....	34
Capítulo 13: Configuraciones útiles que se pueden poner en .vimrc.....	35
Syntax.....	35
Examples.....	35
Mover hacia arriba / abajo las líneas mostradas al envolver.....	35
Habilitar la interacción del ratón.....	35
Configurar el registro predeterminado para ser utilizado como portapapeles del sistema.....	35
Capítulo 14: Configurando Vim.....	37
Examples.....	37
El archivo vimrc.....	37
¿Qué opciones puedo usar?.....	37
Archivos y directorios.....	38
Opciones.....	39
Configuración de opciones booleanas.....	39
Configuración de opciones de cadena.....	39
Configuración de opciones de número.....	39
Usando una expresión como valor.....	39
Mapeos.....	40
Mapeos recursivos.....	40
Asignaciones no recursivas.....	40
Ejecutando un comando desde un mapeo.....	40
Ejecutando múltiples comandos desde un mapeo.....	40
Llamar a una función desde un mapeo.....	41
Mapeando un mapeo <Plug>.....	41
Variables.....	41
Comandos.....	41
Ejemplos.....	41
Funciones.....	42
Ejemplo.....	42

Funciones de script.....	42
Usando s: funciones de mapeos.....	42
Grupos de autocomando.....	43
Ejemplo.....	43
Condicionales.....	43
Opciones de configuración.....	43
Resaltado de sintaxis.....	44
Esquemas de color.....	44
Cambio de esquemas de color.....	44
Instalación de combinaciones de colores.....	44
Enumerar línea de enumeración.....	45
Complementos.....	45
Capítulo 15: Consejos y trucos para aumentar la productividad.....	46
Sintaxis.....	46
Observaciones.....	46
Examples.....	46
Macros rápidos "desechables".....	46
Usando la función de finalización de ruta dentro de Vim.....	46
Activar números de línea relativos.....	47
Visualización de números de línea.....	48
Mapeos para salir del modo Insertar.....	48
j k.....	48
Bloq Mayús.....	49
Linux.....	49
Windows.....	49
Mac OS.....	49
Cómo saber el directorio y / o la ruta del archivo que está editando.....	50
Buscar dentro de un bloque de función.....	50
Copiar, mover o eliminar la línea encontrada.....	51
Escribe un archivo si te olvidas de `sudo` antes de iniciar vim.....	52
Recargar automáticamente vimrc al guardar.....	52
Completar la línea de comando.....	52

Capítulo 16: Construyendo desde vim	54
Examples	54
Comenzando una construcción	54
Capítulo 17: Convertir archivos de texto de DOS a UNIX con vi	55
Observaciones	55
Examples	55
Convertir un archivo de texto de DOS en un archivo de texto de UNIX	55
Usando el formato de archivo de VIm	55
Capítulo 18: Corrector ortográfico	57
Examples	57
Corrección ortográfica	57
Establecer lista de palabras	57
Capítulo 19: De desplazamiento	58
Examples	58
Desplazándose hacia abajo	58
Desplazándose hacia arriba	58
Desplazamiento relativo a la posición del cursor	58
Capítulo 20: Diferencias entre Neovim y Vim	60
Examples	60
Archivos de configuración	60
Capítulo 21: El operador de puntos	61
Examples	61
Uso básico	61
Establecer sangría	61
Capítulo 22: Encontrar y reemplazar	63
Examples	63
Comando sustituto	63
Reemplazar con o sin expresiones regulares	64
Capítulo 23: Espacio en blanco	66
Introducción	66
Observaciones	66

Examples.....	66
Eliminar espacios finales en un archivo.....	66
Eliminar líneas en blanco en un archivo.....	66
Convertir pestañas en espacios y espacios en pestañas.....	67
Capítulo 24: Expresiones regulares.....	68
Observaciones.....	68
Examples.....	68
Palabra.....	68
Capítulo 25: Expresiones regulares en modo Ex.....	69
Examples.....	69
Editar una expresión regular en modo Ex.....	69
Capítulo 26: Extendiendo Vim.....	72
Observaciones.....	72
Examples.....	72
Cómo funcionan los complementos.....	72
El principio.....	72
El metodo manual.....	73
Plugin de un solo archivo.....	73
Haz.....	73
VAM.....	73
Vundle.....	74
Instalando Vundle.....	74
Formatos de complementos compatibles.....	74
El futuro: los paquetes.....	75
Patógeno.....	75
Instalación de patógeno.....	75
Usando patogeno.....	76
Beneficios.....	76
Capítulo 27: Huevos de Pascua.....	77
Examples.....	77
¡Ayuda!.....	77

Cuando te sientes triste.....	77
La respuesta.....	77
Buscando el Santo Grial.....	77
Ceci n'est pas une pipe.....	77
Cuando un usuario se aburre.....	78
Cuchara.....	78
Caballeros que dicen Ni!.....	78
nunmap.....	78
Capítulo 28: Insertar texto.....	80
Examples.....	80
Saliendo del modo insertar.....	80
Diferentes maneras de entrar en el modo de inserción.....	80
Métodos abreviados del modo Insertar.....	81
Ejecutando comandos normales desde el modo de inserción.....	81
Ejemplo.....	81
Insertar texto en varias líneas a la vez.....	82
Pegar texto usando el comando "pegar" del terminal.....	82
Cómo pegar desde un registro mientras está en modo inserción.....	83
Comandos y accesos directos de inserción avanzados.....	83
Deshabilitar la sangría automática para pegar el código.....	84
Capítulo 29: Macros.....	86
Examples.....	86
Grabando una macro.....	86
Edición de una macro vim.....	86
Macros recursivas.....	87
¿Qué es una macro?.....	87
Grabar y reproducir la acción (macros).....	89
Capítulo 30: Manipulando texto.....	90
Observaciones.....	90
Examples.....	90
Convertir caso de texto.....	90
En modo normal:.....	90

En modo visual:.....	90
Incremento y decremento de números y caracteres alfabéticos.....	90
Números crecientes y decrecientes.....	91
Incremento y decremento de caracteres alfabéticos.....	91
Incremento y decremento de números cuando se habilita el incremento / decremento alfabético.....	91
Código de formato.....	92
Usando "verbos" y "sustantivos" para la edición de texto.....	92
Capítulo 31: Mejora deshacer y rehacer con un undodir.....	94
Examples.....	94
Configurando tu vimrc para usar un undodir.....	94
Capítulo 32: Mociones y objetos de texto.....	95
Observaciones.....	95
Examples.....	95
Cambiando los contenidos de una cadena o lista de parámetros.....	95
Capítulo 33: Modos - insertar, normal, visual, ex.....	96
Examples.....	96
Lo básico sobre los modos.....	96
Modo normal (o modo de comando).....	96
Modo de inserción.....	96
Modo visual.....	96
Seleccionar modo.....	96
Modo de reemplazo.....	97
Modo de línea de comandos.....	97
Modo ex.....	97
Capítulo 34: Movimiento.....	98
Examples.....	98
buscando.....	98
Saltando a los personajes.....	98
Buscando cuerdas.....	98
Movimiento basico.....	99
Observaciones.....	99

Flechas	99
Movimientos basicos	99
Buscando patrón.....	101
Navegando al principio de una palabra específica.....	102
Usando marcas para moverse.....	104
TLDR	104
Establecer una marca	104
Saltar a una marca	104
Marcas globales	104
Marcas especiales	105
Saltar a la línea específica.....	106
Capítulo 35: Objetos de texto VIM	107
Examples.....	107
Seleccione una palabra sin espacios en blanco circundantes.....	107
Seleccione una palabra con espacio en blanco circundante.....	107
Seleccionar texto dentro de una etiqueta.....	107
Capítulo 36: Obtener: ayuda (usando el manual incorporado de Vim)	109
Introducción.....	109
Sintaxis.....	109
Parámetros.....	109
Examples.....	109
Empezando / Navegando archivos de ayuda.....	109
Buscando el manual.....	110
Capítulo 37: Opciones de Vim	112
Sintaxis.....	112
Observaciones.....	112
Examples.....	112
Conjunto.....	112
Sangría.....	112
Anchura	112
Espacios	112

Pestañas	113
Sangría automática	113
Descripciones de instrucciones	113
Personajes invisibles.....	113
Mostrar u ocultar personajes invisibles	113
Caracteres de símbolo por defecto	114
Personalizar símbolos	114
Capítulo 38: Pida crear directorios no existentes al guardar un archivo nuevo	115
Introducción.....	115
Examples.....	115
Indique la creación de directorios con: w, o créelos lentamente con: w!.....	115
Capítulo 39: Plegable	116
Observaciones.....	116
Examples.....	116
Configurando el método de plegado.....	116
Creación de un pliegue manual.....	116
Apertura, cierre y cambio de pliegues.....	116
Mostrando la línea que contiene el cursor.....	117
Bloques plegables en C.....	117
Capítulo 40: Rangos de línea de comando	118
Examples.....	118
Números de línea absolutos.....	118
Números de línea relativos.....	118
Atajos de línea.....	118
Marcas.....	118
Buscar.....	119
Compensaciones de línea.....	119
Rangos mixtos.....	119
Capítulo 41: Recursos Vim	120
Observaciones.....	120
Examples.....	120

Aprendiendo Vimscript de la manera difícil.....	120
Capítulo 42: Registros Vim.....	121
Parámetros.....	121
Examples.....	121
Eliminar un rango de líneas en un registro con nombre.....	121
Pegue el nombre del archivo en el modo de inserción usando el registro de nombre de archiv.....	122
Copiar / pegar entre Vim y el portapapeles del sistema.....	122
Anexar a un registro.....	122
Capítulo 43: Saliendo de Vim.....	123
Parámetros.....	123
Observaciones.....	123
Examples.....	123
Salir con guardar.....	123
Salir sin guardar.....	123
Salir con fuerza (sin guardar).....	123
Salir con fuerza (con guardar).....	124
Salir con fuerza de todas las ventanas abiertas (sin guardar).....	124
si se abren múltiples archivos.....	124
Capítulo 44: Sangría.....	125
Examples.....	125
Sangre un archivo completo utilizando el motor de sangría incorporado.....	125
Líneas sangradas o caducas.....	125
Capítulo 45: Sustitución.....	127
Sintaxis.....	127
Parámetros.....	127
Observaciones.....	127
Ejemplo.....	127
Examples.....	127
Reemplazo simple.....	127
Refactoriza rápidamente la palabra debajo del cursor.....	128
Reemplazo con aprobación interactiva.....	128
Teclado de método abreviado para reemplazar la palabra resaltada.....	128

Capítulo 46: Tampones	129
Examples.....	129
Gestionando buffers.....	129
Buffers ocultos.....	129
Cambio de búfer usando parte del nombre de archivo.....	129
Cambie rápidamente al búfer anterior o a cualquier búfer por número.....	130
Capítulo 47: Usando ex desde la línea de comando	131
Examples.....	131
Sustitución desde la línea de comando.....	131
Capítulo 48: Usando Python para scripts Vim	132
Sintaxis.....	132
Examples.....	132
Compruebe la versión de Python en Vim.....	132
Ejecute los comandos del modo normal de Vim a través de la declaración de Python.....	132
Ejecutando código multilínea de Python.....	132
Capítulo 49: Ventajas de vim	134
Examples.....	134
Personalización.....	134
Ligero.....	134
Capítulo 50: Ventanas divididas	136
Sintaxis.....	136
Observaciones.....	136
Examples.....	136
Abriendo múltiples archivos en splits desde la línea de comando.....	136
Horizontalmente.....	136
Verticalmente.....	136
Abriendo una nueva ventana dividida.....	136
Cambiando el tamaño de una división o vsplit.....	137
Atajos	137
Cierra todas las splits pero la actual.....	137
Administrar ventanas divididas abiertas (atajos de teclado).....	137
Moverse entre divisiones.....	138

Sane split split.....	138
Capítulo 51: vglobal: ejecuta comandos en líneas que no coinciden globalmente.....	139
Introducción.....	139
Examples.....	139
: v / patrón / d.....	139
Capítulo 52: Vim Solarizado.....	141
Introducción.....	141
Examples.....	141
.vimrc.....	141
Capítulo 53: Vimscript.....	142
Observaciones.....	142
Examples.....	142
Hola Mundo.....	142
Usando Comandos de Modo Normal en Vimscript.....	142
Creditos.....	144

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [vim](#)

It is an unofficial and free vim ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official vim.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con vim

Observaciones

Vim (o "Vi IMproved") es un editor de texto *multimodo* (*modal*) basado en consola. Se usa ampliamente y está disponible de forma predeterminada en todos los sistemas Unix, Linux y Apple OS X. Vim tiene una gran comunidad activa y una amplia base de usuarios. El editor es compatible con todos los lenguajes de programación populares, y muchos complementos están disponibles para ampliar sus funciones.

A los desarrolladores les gusta el editor por su velocidad, muchas opciones de configuración y una potente edición basada en expresiones. En el modo "comando", el editor está controlado por los comandos del teclado, por lo que el usuario no se distrae con una GUI o el puntero del mouse.

Vim se basa en el anterior editor "vi" de Unix creado en los años setenta y ha estado en continuo desarrollo desde 1991. Con las macros y los complementos, el editor ofrece la mayoría de las características de un IDE moderno. También tiene una capacidad única para procesar grandes cantidades de texto con su lenguaje de scripting (vimscript) y expresiones regulares.

Temas principales:

- [instalación](#)
- modos de edición
- [navegación](#)
- [edición básica](#)
- edición avanzada
- [configuración](#)
- [plugins](#)
- [vimscript](#)
- [macros](#)
- comunidad
- Proyectos relacionados

Versión	Fecha de lanzamiento
1.14	1991-11-02

Examples

Instalación

El Vim en su máquina, si lo hay, es muy probable que sea una compilación "pequeña" que carece de características útiles como soporte del portapapeles, resaltado de sintaxis o incluso la capacidad de usar complementos.

Esto no es un problema si todo lo que necesita es una forma rápida de editar archivos de configuración, pero pronto llegará a una serie de muros si pretende hacer de Vim su editor principal.

Por lo tanto, generalmente se recomienda instalar una compilación completa.

Instalación en Linux / BSD

En esos sistemas, el truco es simplemente instalar la versión de GUI que viene con un comando `gvim` para iniciar la GUI y un comando `vim` para iniciar la TUI.

Distribuciones basadas en arco y arco.

```
$ sudo pacman -R vim
$ sudo pacman -S gvim
```

Distribuciones basadas en Debian y Debian

```
$ sudo apt-get update
$ sudo apt-get install vim-gtk
```

Distribuciones basadas en Gentoo y Gentoo

```
$ sudo emerge --sync
$ sudo emerge app-editors/gvim
```

RedHat y distribuciones basadas en RedHat

```
$ sudo yum check-update
$ sudo yum install vim-X11
```

Fedora

```
$ sudo dnf check-update
$ sudo dnf install vim-X11
```

Distribuciones basadas en Slackware y Slackware

```
$ sudo slackpkg update
$ sudo slackpkg install-new vim-gvim
```

Distribuciones basadas en OpenBSD y OpenBSD

```
$ sudo pkg_add vim-x11
```

Distribuciones basadas en FreeBSD y FreeBSD

```
$ sudo pkg install editors/vim
```

Instalación en Mac OS X

La estrategia es similar a Mac OS X: instalamos la versión GUI para obtener tanto la GUI como la TUI. Al final, deberíamos ser capaces de:

- haga doble clic en el icono de MacVim en el Finder,
- haga clic en el icono de MacVim en el Dock,
- emita `$ mvim` en el shell para abrir la GUI de MacVim,
- emita `$ mvim -v` en el shell para abrir la TUI de MacVim.

Instalación regular

Descargue e instale [una instantánea oficial](#) como lo haría con cualquier otra aplicación de Mac OS X.

Coloque el script `mvim` que viene incluido con MacVim en algún lugar de su `$PATH`.

Gerente de empaquetación

MacPorts:

```
$ sudo port selfupdate
$ sudo port install macvim
```

Homebrew

```
$ brew install macvim
```

Para hacer de MacVim la consola predeterminada Vim:

```
$ brew install macvim --with-override-system-vim
```

Instalación en Windows

No hay Vim en los sistemas Windows por defecto. Puede descargar e instalar Vim desde el [sitio de Tuxproject](#) para [compilaciones](#) más actualizadas y completas, o puede descargar e instalar Vim desde el [sitio](#) oficial de [Vim](#) .

Chocolatey

```
> choco install vim
```

Construyendo Vim desde la fuente

Si los métodos anteriores no satisfacen sus necesidades, todavía es posible crear Vim usted mismo, con *solo* las opciones que necesita.

Este tema será discutido en su propia sección (actualmente en borrador).

Saliendo de Vim

Para salir de Vim, primero asegúrese de estar en modo *Normal* presionando `Esc` .

- `:q` Ingresar (evitará que salga si tiene cambios no guardados, abreviatura de: salir)

Para *descartar* cambios y salir de Vim:

- `:q!` Entrar para forzar la salida y descartar los cambios (abreviatura de `:quit!` No debe confundirse con `:!q`),
- `ZQ` es un atajo que hace lo mismo que `:q!` ,
- `:cq` Ingrese quit y devuelva el error (descarte todos los cambios para que el compilador no vuelva a compilar este archivo)

Para *guardar los cambios* y salir de Vim:

- `:wq` Enter (abreviatura de `:write` y `:quit`),
- `:x` Ingrese (igual que `:wq` , pero no escribirá si el archivo no se modificó),
- `ZZ` es un acceso directo que hace lo mismo que `:x` (Guardar espacio de trabajo y salir del editor),
- `:[range]wq!` Ingrese (escriba las líneas en [rango])

Para cerrar varios buffers a la vez (incluso en varias ventanas y / o pestañas), agregue la letra `a` a cualquiera de los *Comandos* anteriores (los que comienzan con `:` . Por ejemplo, para escribir y salir de todas las ventanas puede usar:

- `:wqa` Entrar `O`
- `:xa` Enter - Escribe todos los buffers modificados y sale de Vim. Si hay buffers sin un nombre

de archivo, que son de solo lectura o que no se pueden escribir por otra razón, Vim no se cerrará

- `:xa!` Ingresar `:` escriba todos los búferes modificados, incluso los que son de solo lectura, y salga de Vim. Si hay buffers sin un nombre de archivo o que no se pueden escribir por otra razón, Vim no se cerrará
- `:qa` Enter `:` intente salir, pero deténgase si hay archivos no guardados;
- `:qa!` Ingresar - salir *sin guardar* (descartar los cambios en los archivos no guardados)

Si ha abierto Vim sin especificar un archivo y desea guardar ese archivo antes de salir, recibirá un mensaje `E32: No file name`. Puedes guardar tu archivo y salir usando:

- `:wq filename` Entrar `O`;
- `:x filename` Introduzca

Explicación:

La pulsación de tecla `:` en realidad abre el modo *Comando*. El comando `q` es una abreviatura de `quit`, `w`, de `write` y `x`, de `exit` (también puede escribir `:quit`, `:write` y `:exit` si lo desea). Los accesos directos que *no* comienzan con `:` como `zz` y `zQ` refieren a las asignaciones de teclas del modo *Normal*. Puedes pensar en ellos como atajos.

El `!` La pulsación de tecla a veces se usa al final de un comando para forzar su ejecución, lo que permite descartar cambios en el caso de `:q!`. Colocando el `!` Al comienzo del comando tiene un significado diferente. Por ejemplo, uno puede escribir `mal:!` `:!q` lugar de `:q!` y vim terminaría con un error 127.

Una manera fácil de recordar esto es pensar `!` Como una forma de insistir en ejecutar algo. Al igual que cuando escribes: "¡Quiero dejar de fumar!"

Tutoriales interactivos de Vim (como vimtutor)

`vimtutor` es un tutorial interactivo que cubre los aspectos más básicos de la edición de texto.

En un sistema similar a UNIX, puede iniciar el tutorial con:

```
$ vimtutor
```

En Windows, "Vim tutor" se puede encontrar en el directorio "Vim 7.x" en "Todos los programas" en el menú de Windows.

Ver `:help vimtutor` para más detalles.

Otros tutoriales interactivos incluyen estos basados en navegador:

- [Vim Adventures](#) - Una versión interactiva del juego de vimtutor disponible en la web. Solo los primeros niveles son gratuitos.
- [Open Vim](#) : un terminal interactivo que le enseña los comandos básicos con comentarios.
- [Vim Genius](#) : otro terminal interactivo que también incluye lecciones intermedias y

avanzadas que incluyen macros y arglist.

Guardando un archivo de solo lectura editado en Vim

A veces, podemos abrir un archivo que no tenemos permiso para escribir en Vim sin usar `sudo`.

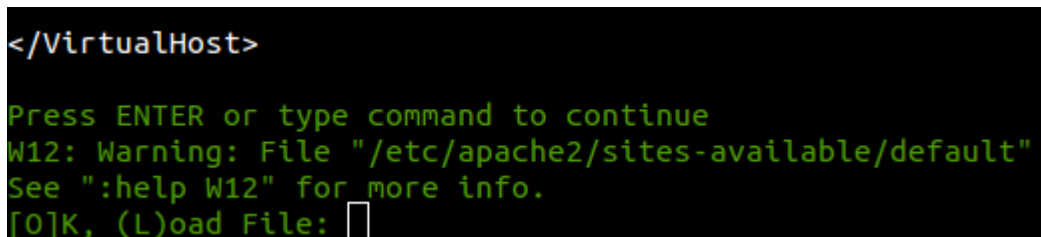
Utilice este comando para guardar un archivo de solo lectura editado en Vim.

```
:w !sudo tee > /dev/null %
```

Que puedes mapear a `:w!!` en tu `.vimrc`:

```
cmap w!! w !sudo tee > /dev/null %
```

Se le presentará un aviso como se muestra en la imagen.



```
</VirtualHost>
Press ENTER or type command to continue
W12: Warning: File "/etc/apache2/sites-available/default"
See ":help W12" for more info.
[O]K, (L)oad File: 
```

Presiona `o` y el archivo se guardará. Permanece abierto en `vi` / `vim` para más ediciones o lecturas y puede salir normalmente escribiendo `:q!` ya que el archivo todavía está abierto como de solo lectura.

Explicación del comando

```
:w ..... isn't modifying your file in this case,
..... but sends the current buffer contents to
..... a substituted shell command
!sudo ..... call the shell 'sudo' command
tee ..... the output of the vi/vim write command is redirected
using the 'tee' command
> /dev/null ..... throws away the standard output, since we don't need
to pass it to other commands
% .... expands to the path of the current file
```

Fuentes:

- [El blog de tecnología de Adam Culp](#)
- [Stackoverflow, ¿Cómo funciona el truco vim "write with sudo"?](#)

Suspendiendo vim

Cuando use `vim` desde la línea de comandos, puede suspender `vim` y volver a su indicador, sin tener que salir de `vim`. Por lo tanto, más tarde podrá recuperar su sesión de `vim` desde el mismo

indicador.

Cuando esté en el *modo Normal* (si no lo está, presione `esc` para llegar allí), emita cualquiera de estos comandos:

```
:st enter
```

```
:sus enter
```

```
:stop entrar
```

```
:suspend entrar
```

Alternativamente, en algunos sistemas, cuando se encuentra en modo *Normal* o *Visual*, la emisión de `Ctrl + Z` tendrá el mismo efecto.

Nota: Si se establece la `autowrite` se `autowrite` búferes con cambios y nombres de archivos.

Añadir un `!` antes de `entrar` para evitar, por ejemplo. `:st! entrar`

Más adelante, cuando desee volver a su sesión de `vim`, si no ha suspendido ningún otro trabajo, emitir lo siguiente restaurará `vim` como su trabajo de primer plano.

```
fg enter
```

De lo contrario, tendrá que encontrar el ID de trabajo de las sesiones de `vim` emitiendo los `jobs` ingresados y, luego, poniendo en primer plano los trabajos coincidentes `fg %[job ID] ingrese`, por ejemplo. `fg %1 entrar`.

Lo esencial

Ejecute [tutoriales](#) interactivos de `vim` tantas veces como sea necesario para sentirse cómodo con lo básico.

Vim presenta varios modos, por ejemplo, **modo normal**, **modo de inserción** y **modo de línea de comandos**.

El modo normal es para editar y navegar texto. En este modo, `h`, `j`, `k` y `l` corresponden a las teclas de cursor `←`, `↓`, `↑` y `→`. La mayoría de los comandos en el modo normal se pueden prefijar con un "conteo", por ejemplo, `3j` baja 3 líneas.

El modo de inserción es para insertar el texto directamente, en este modo `vim` es similar a otros editores de texto más simples. Para ingresar al modo de inserción, presione `i` en el modo normal. Para dejarlo presione `<ESC>` (tecla de escape).

El modo de línea de comandos es para ejecutar comandos más complejos como guardar el archivo y salir de `vim`. Presione `:` para iniciar el modo de línea de comando. Para salir de este modo también puede presionar `<ESC>`. Para guardar los cambios en el archivo use `:w` (o `:write`). Para salir de `vim` sin guardar tus cambios usa `:q!` (o `:quit!`).

Estos son algunos de los comandos más útiles en `vim`:

Mando	Descripción
i	(insertar) ingresa al modo de inserción <i>antes de</i> la posición actual del cursor
I	ingresa al modo de inserción <i>antes</i> del primer carácter imprimible de la línea actual
a	(añadir) ingresa al modo de inserción <i>después de</i> la posición actual del cursor
A	ingresa al modo de inserción <i>después</i> del último carácter imprimible de la línea actual
x	eliminar caracteres en la posición actual del cursor
X	Eliminar el carácter a la izquierda a la posición actual del cursor.
w	pasar a la siguiente palabra
b	pasar a la palabra anterior
0	mover al principio de la línea
\$	mover al final de la línea
r	reemplazar: ingresa al modo de reemplazo para un personaje. El siguiente carácter que escriba reemplazará el carácter debajo del cursor.
R	Entra en modo de reemplazo indefinidamente. Cada carácter que escriba reemplazará el carácter debajo del cursor y avanzará el cursor en uno.
s	sustituto: elimina el carácter en la posición actual del cursor y luego ingresa al modo de inserción
S	borre la línea actual en la que se encuentra actualmente el cursor e ingrese al modo de inserción
<Esc> , <Cc>	Salir del modo insertar y volver al modo normal.
u	deshacer
<Cr>	rehacer
dd , dw , dl , d\$	corte la línea actual, desde el cursor hasta la siguiente palabra, o el carácter, la posición actual hasta el final de la línea actual respectivamente, nota: D es el equivalente de d\$
cc , cw , cl	cambiar la línea actual, desde el cursor a la siguiente palabra, o el carácter, respectivamente
YY , yw , yl , y\$	tirar ("copiar") la línea actual, desde el cursor hasta la siguiente palabra, o el carácter, la posición actual hasta el final de la línea actual, respectivamente

Mando	Descripción
p , P	poner ("pegar") después, o antes de la posición actual, respectivamente
o , O	para crear una nueva línea vacía, después o antes de la actual e ingresar al modo de inserción
:w	escribe el búfer actual en el disco
:q! , ZQ	dejar de fumar sin escribir
:x :wq , ZZ	escribe y deja
:help	abrir una ventana con el archivo de ayuda
:help {subject}	Mostrar ayuda para un tema específico.
qz	comience a grabar acciones para registrar z , q para finalizar la grabación, @z para reproducir las acciones. z puede ser cualquier letra: q se usa a menudo por conveniencia. Leer más: Macros

Qué hacer en caso de accidente.

Vim guarda todas sus ediciones no guardadas en un *archivo de intercambio* , un archivo adicional que se borra una vez que se comprometen los cambios al guardar. El nombre del archivo de intercambio suele ser el nombre del archivo que se está editando precedido por un . y con un sufijo .swp (se puede ver con :sw).

Entonces, en caso de que su proceso de vim finalice antes de que haya tenido la oportunidad de guardar sus ediciones, puede recuperar su trabajo aplicando los cambios contenidos en el archivo de intercambio a su archivo actual usando la opción de línea de comandos -r . Por ejemplo, si myFile es el archivo que estaba editando, use:

```
$ vi -r myFile
```

Para recuperar los cambios no comprometidos.

Si existe un archivo de intercambio, vim debería solicitarle de todos modos opciones de recuperación

```
$ vi myFile
E325: ATTENTION
Found a swap file by the name ".myFile.swp"
...
Swap file ".myFile.swp" already exists!
[O]pen Read-Only, (E)dit anyway, (R)ecover, (D)elete it, (Q)uit, (A)bort:
```

Si elige (R) ecover, se aplicarán los cambios del archivo swp pero el archivo de intercambio no se eliminará, así que no olvide eliminar el archivo de intercambio posteriormente si está satisfecho

con la recuperación.

Lea Empezando con vim en línea: <https://riptutorial.com/es/vim/topic/879/empezando-con-vim>

Capítulo 2: :global

Sintaxis

- : [<range>]g[lobal]/{<pattern>}/[<command>]
- : [<range>]g[lobal]!/{<pattern>}/[<command>] (invertido)
- : [<range>]v[lobal]/{<pattern>}/[<command>] (invertido)

Observaciones

El comando "global" de Vim se usa para aplicar un comando ex a cada línea donde coincida una expresión regular.

Examples

Uso básico del Comando Global

```
:g/Hello/d
```

Se eliminarán todas las líneas que contengan el texto "Hola". **Nota importante** : este no es el comando de modo normal `d` , este es el comando ex `:d` .

Puede usar el comando global para aplicar pulsaciones de teclado en modo normal en lugar de comandos ex al añadir `normal` comando `normal` o `norm` al comando. Por ejemplo:

```
:g/Hello/norm dw
```

Se eliminará la primera palabra de cada línea que contenga el texto "Hola".

El comando global también admite el modo visual [y los rangos](#) .

Tirar de cada línea que coincida con un patrón

El comando

```
:g/apples/y A
```

será un tirón todas las líneas que contengan la palabra *manzanas* en el `a` registro, que se puede pegar con `"ap` . Cualquier expresión regular puede ser utilizado.

Tenga en cuenta el espacio antes de la `A` y la mayúscula de la letra de registro. Si se utiliza una letra mayúscula como registro de tirón, las coincidencias se *agregarán* a ese registro. Si se usa una letra minúscula, solo se colocará la *última* coincidencia en ese registro.

Mover / recoger líneas que contienen información clave

Un comando simple pero muy útil:

```
:g/ending/m$
```

Mueve las líneas que contienen `ending` hasta el final del búfer.

`m` significa mover

`$` significa fin de búfer, mientras que `0` significa comienzo de búfer.

Lea `:global` en línea: <https://riptutorial.com/es/vim/topic/3861/-global>

Capítulo 3: Ahorro

Examples

Guardar un búfer en un directorio no existente

```
:!mkdir -p %:h
```

para crear los directorios que faltan, entonces

```
:w
```

Lea Ahorro en línea: <https://riptutorial.com/es/vim/topic/6440/ahorro>

Capítulo 4: Asignaciones clave en Vim

Introducción

La actualización de las asignaciones de teclas Vim le permite resolver dos tipos de problemas: reasignar comandos de teclado a letras que sean más fáciles de recordar o accesibles, y crear comandos de tecla para funciones que no tienen ninguno. Aquí aprenderá sobre las diversas formas de [re] mapear comandos de teclado y el contexto al que se aplican (*es decir, los modos vim*)

Examples

Mapeo basico

mapa general

Una secuencia de teclas puede volver a asignarse a otra secuencia de teclas utilizando una de las variantes de `map`.

Como ejemplo, el siguiente `map` típico saldrá del *modo Insertar* cuando presione `jk` en secuencia rápida:

```
:inoremap jk <Esc>
```

operador de mapas

Existen múltiples variantes de `:map` para diferentes modos.

Comandos	Modos
<code>:map</code> <code>:noremap</code> <code>:unmap</code>	Modo normal, visual y pendiente de operador.
<code>:map!</code> <code>:noremap!</code> <code>:unmap!</code>	Insertar y modo de línea de comandos
<code>:nmap</code> <code>:nnoremap</code> <code>:nunmap</code>	Modo normal
<code>:imap</code> <code>:inoremap</code> <code>:iunmap</code>	Insertar y reemplazar el modo
<code>:vmap</code> <code>:vnoremap</code> <code>:vunmap</code>	Modo visual y de selección.
<code>:xmap</code> <code>:xnoremap</code> <code>:xunmap</code>	Modo visual
<code>:smap</code> <code>:snoremap</code> <code>:sunmap</code>	Seleccionar modo
<code>:cmap</code> <code>:cnoremap</code> <code>:cunmap</code>	Modo de línea de comandos

Comandos	Modos
<code>:omap</code> <code>:onoremap</code> <code>:ounmap</code>	Modo operador pendiente

Por lo general, **debe utilizar las variantes** `:noremap` ; hace que el mapeo sea inmune a la reasignación y la recursión.

comando de mapa

- Puede visualizar todas las asignaciones utilizando `:map` (o una de las variaciones anteriores).
- Para mostrar la asignación actual para una secuencia de teclas específica, use `:map <key>` donde `<key>` es una secuencia de teclas
- Las teclas especiales como `Esc` se asignan mediante la notación especial `<>` , como `<Esc>` . Para obtener la lista completa de códigos de clave, consulte <http://vimdoc.sourceforge.net/html/doc/intro.html#keycodes>
- `:nmapclear` - Borrar todos los mapas de modo normal
- `:nunmap` - Desasignar un mapa de modo normal
- Puede configurar el tiempo máximo entre las teclas de una secuencia cambiando las variables `timeout` y `tttimeout`

Ejemplos

- `imap jk <Esc>` : escribir `jk` en el modo de inserción lo regresará al modo normal
- `nnoremap tt :tabnew<CR>` : escribir `tt` en modo normal abrirá una nueva página de pestaña
- `nnoremap <Cj> <Cw>j` : escribir `<Cj>` en modo normal te hará saltar a la ventana de abajo y a la izquierda
- `vmap <Cc> \cc` : escribir `<Cc>` en modo visual ejecutará `\cc` (comando NERDCommitter para comentar la línea). Como esto se basa en un mapeo de complementos, no puede usar `:vnoremap` aquí!

leer más [aquí](#)

Combinación de teclas de líder de mapa

La clave de líder podría utilizarse como una forma de crear una asignación con un enlace de clave que el usuario final puede anular.

El líder es la clave `\` por defecto. Para anularlo, el usuario final tendría que ejecutar `:let g:mapleader='somekey(s)'` antes de definir la asignación.

En un escenario típico, el `mapleader` se establece en el `.vimrc` , y los complementos usan `<Leader>` en la parte de enlace de teclas de sus asignaciones para que sean personalizables.

En el plugin, definiríamos mapeos con:

```
:nnoremap <Leader>a somecomplexaction
```


Esto sería asignar la acción a la `somecomplexaction` una combinación de teclas `\ +`.

La acción sin un líder no cambia.

También es posible usar `<Plug>Mappings` para dejar más espacio para personalizar los enlaces de teclas.

Ilustración de mapeo básico (atajos prácticos).

En la mayoría de los editores de texto, el acceso directo estándar para guardar el documento actual es `Ctrl + S` (o `Cmd + S` en macOS).

Vim no tiene esta función de forma predeterminada, pero se puede asignar para facilitar las cosas. Agregar las siguientes líneas en el archivo `.vimrc` hará el trabajo.

```
nnooremap <c-s> :w<CR>
inoremap <c-s> <c-o>:w<CR>
```

El comando `nnooremap` asigna `Ctrl + S` a `:w` (escribir el contenido actual en el archivo) mientras que el comando `inoremap` asigna `Ctrl + S` al comando `:w` y regresa al modo de inserción (`<co>` pasa al modo normal para un comando y luego vuelve al modo de inserción, sin alterar la posición del cursor que otras soluciones como `<esc>:w<cr>a` no pueden garantizar).

Similar,

```
" This is commented, as Ctrl+Z is used in terminal emulators to suspend the ongoing
program/process.
" nnooremap <c-z> :u<CR>

" Thus, Ctrl+Z can be used in Insert mode
inoremap <c-z> <c-o>:u<CR>

" Enable Ctrl+C for copying selected text in Visual mode
vnoremap <c-c> <c-o>:y<CR>
```

PD: Sin embargo, debe tenerse en cuenta que `Ctrl + S` puede no funcionar como se espera al usar ssh (o PuTTY). La solución a esto no está dentro del alcance de este documento, pero se puede encontrar [aquí](#).

Lea Asignaciones clave en Vim en línea: <https://riptutorial.com/es/vim/topic/3535/asignaciones-clave-en-vim>

Capítulo 5: Autocomandos

Observaciones

`autocmd` **envolventes** `autocmd`

`autocmd` es un comando aditivo, y probablemente no desee este comportamiento de forma predeterminada.

Por ejemplo, si redistribuye su `.vimrc` varias veces mientras lo edita, vim puede disminuir la velocidad.

Aquí está la prueba:

```
:autocmd BufWritePost * if &diff | diffupdate | endif " update diff after save
:autocmd BufWritePost * if &diff | diffupdate | endif " update diff after save
```

Si ahora escribe `:autocmd BufWritePost *`, verá **ambas** líneas en la salida, no solo una. Ambos son ejecutados.

Para evitar este comportamiento, rodee todos sus `autocmd` s de la siguiente manera:

```
if has ('autocmd')      " Remain compatible with vi which doesn't have autocmd
  augroup MyDiffUpdate " A unique name for the group. DO NOT use the same name twice!
    autocmd!           " Clears the old autocommands for this group name
    autocmd BufWritePost * if &diff | diffupdate | endif " Update diff after save
    " ... etc ...
  augroup END
endif
```

Examples

Fuente automática `.vimrc` después de guardar

Agregue esto a su `$MYVIMRC` :

```
" Source vim configuration file whenever it is saved
if has ('autocmd')      " Remain compatible with earlier versions
  augroup Reload_Vimrc " Group name. Always use a unique name!
    autocmd!           " Clear any preexisting autocommands from this group
    autocmd! BufWritePost $MYVIMRC source % | echom "Reloaded " . $MYVIMRC | redraw
    autocmd! BufWritePost $MYGVIMRC if has('gui_running') | so % | echom "Reloaded " .
$MYGVIMRC | endif | redraw
  augroup END
endif " has autocmd
```

características:

- `echom` le dice al usuario lo que ha sucedido (y también se registra en `:messages`).

- `$MYVIMRC` y `$MYGVIMRC` manejan nombres específicos de la plataforma para los archivos de configuración,
- y solo coinciden con los archivos de configuración reales (ignorando copias en otros directorios, o un `fugitive:// diff`)
- `has()` evitará un error si utiliza versiones incompatibles, como `vim-tiny`.
- `autocmd!` evita la acumulación de múltiples autocomandos idénticos si este archivo se obtiene de nuevo. (Borra todos los comandos del grupo nombrado, por lo que el nombre del grupo es importante).

Actualice las vistas de vimdiff si un archivo se guarda en otra ventana

```
:autocmd BufWritePost * if &diff | diffupdate | endif
```

Lea Autocomandos en línea: <https://riptutorial.com/es/vim/topic/4887/autocomandos>

Capítulo 6: Buscando en el buffer actual

Examples

Buscando un patrón arbitrario

Los comandos de búsqueda estándar de Vim son / para búsqueda hacia adelante y ? para búsqueda hacia atrás.

Para iniciar una búsqueda desde el modo normal:

1. pulse / ,
2. escribe tu patrón,
3. presione <CR> para realizar la búsqueda.

Ejemplos:

```
/foobar<CR>      search forward for foobar
?foo\bar<CR>     search backward for foo/bar
```

n y N se pueden usar para saltar a la siguiente y anterior ocurrencia:

- Al presionar n después de una búsqueda hacia adelante, el cursor se posiciona en la siguiente aparición, *hacia adelante* .
- Presionando N después de una búsqueda hacia adelante coloca el cursor en la siguiente aparición, *hacia atrás* .
- Presionando n después de una búsqueda hacia atrás coloca el cursor en la siguiente aparición, *hacia atrás* .
- Presionando N después de una búsqueda hacia atrás posiciona el cursor en la siguiente aparición, *hacia adelante* .

Buscando la palabra debajo del cursor

En el modo normal, mueva el cursor a cualquier palabra, luego presione * para buscar la siguiente aparición de la palabra debajo del cursor, o presione # para buscar hacia atrás.

* o # busca la palabra exacta debajo del cursor: buscar big solo encontraría big y no bigger .

Bajo el capó, Vim utiliza una búsqueda simple con los *límites de las palabras* átomos:

- /\<big\> para * ,
- ?\<big\> para # .

g* o g# no busca la palabra exacta debajo del cursor: la búsqueda de big buscaría bigger .

Bajo el capó, Vim utiliza una búsqueda simple sin *límites de palabras en los átomos*:

- `/\<big\> para *` ,
- `?\<big\> para #` .

Ejecutar comando en líneas que contienen texto.

El comando `:global` ya tiene su propio tema: [el comando global](#)

Lea [Buscando en el buffer actual en línea](#): <https://riptutorial.com/es/vim/topic/3269/buscando-en-el-buffer-actual>

Capítulo 7: Código de auto-formato

Examples

En modo normal:

En modo normal:

gg *ir arriba* = entonces G

Lea Código de auto-formato en línea: <https://riptutorial.com/es/vim/topic/7931/codigo-de-auto-formato>

Capítulo 8: Comandos de modo normal

Sintaxis

- : [range] sor [t] [!] [b] [f] [i] [n] [o] [r] [u] [x] [/ {pattern} /]
- Nota: Las opciones [n] [f] [x] [o] [b] se excluyen mutuamente.

Observaciones

Ver [clasificación](#) en el manual de vim para la explicación canónica

Examples

Ordenar texto

Clasificación normal

Resalte el texto que desea ordenar y escriba:

```
:sort
```

Si no resalta texto o no especifica un rango, todo el búfer se ordena.

Clasificación inversa

```
:sort!
```

Clasificación insensible a mayúsculas

```
:sort i
```

Clasificación numérica

Ordenar por el primer número que aparecerá en cada línea:

```
:sort n
```

Eliminar duplicados después de ordenar

```
:sort u
```

(u significa único)

Combinando opciones

Para obtener un orden inverso que no distinga entre mayúsculas y minúsculas y que se eliminen los duplicados:

```
:sort! iu
```

Lea Comandos de modo normal en línea: <https://riptutorial.com/es/vim/topic/6005/comandos-de-modo-normal>

Capítulo 9: Comandos en modo normal (Edición)

Examples

Introducción - Nota rápida sobre el modo normal

En el Modo normal, los comandos pueden ingresarse mediante combinaciones de teclas directas (escribiendo `u` para deshacer el último cambio, por ejemplo). Estos comandos a menudo tienen equivalentes en el modo 'ex', al que se accede escribiendo dos puntos `:`, lo que lo coloca en un búfer de una sola línea en la parte inferior de la ventana de Vim.

En el modo 'ex', después de escribir los dos puntos, escribe un nombre de comando o su abreviatura seguido de `Enter` para ejecutar el comando. Así, `: undo` *Introduzca* logra lo mismo que escribir directamente `u` en el modo normal.

Puede ver que los comandos directos a menudo serán más rápidos (una vez que se aprenden) que los comandos 'ex' para una edición simple, pero de manera completa, siempre que sea posible en la documentación que sigue, si ambos están disponibles para su uso, se mostrarán ambos.

La mayoría de estos comandos también pueden ir precedidos de un *recuento* prefijando o entremezclando un `3dd` tecleando *números* en el Modo Normal, por ejemplo, elimina tres líneas (comenzando desde la posición actual del cursor).

Deshacer y rehacer básico

Deshacer

Mando	:	Descripción
<code>tu</code>	<code>u</code> , <code>undo</code>	Deshacer el cambio más reciente
<code>5u</code>		Deshacer los <i>cinco</i> cambios más recientes (usar cualquier número)

Tenga en cuenta que en Vim, el 'cambio más reciente' varía según el modo en el que se encuentre. Si ingresa el Modo Insertar (`i`) y escribe un párrafo completo antes de volver al Modo Normal (`Esc`), ese *párrafo completo* es Considerado el cambio más reciente.

Rehacer

Mando	:	Descripción
Ctrl-R	red, redo	Rehacer el cambio deshecho más reciente.
2Ctrl-R		Rehace los <i>dos</i> cambios deshechos más recientes (use cualquier número)

Hay otra forma de deshacer y rehacer los cambios en Vim que se maneja de manera un poco diferente. Cuando deshace un cambio con `u`, realiza una copia de seguridad de los nodos en un 'árbol' de sus cambios, y al presionar `Ctrl-R` retrocede esos nodos en orden. (El árbol de deshacer es un tema aparte y es demasiado complejo para cubrirlo aquí).

También puede utilizar `U` (es decir, en mayúsculas) para eliminar todos los últimos cambios en una sola línea (la línea donde se hicieron los últimos cambios). Esto *no* atraviesa los nodos del árbol de la misma manera que `u`. El uso de `U` en realidad cuenta como un cambio en sí mismo (otro nodo *hacia adelante* en el árbol), de modo que si presiona `U` por segunda vez inmediatamente después del primero, actuará como un comando Rehacer.

Cada uno tiene sus usos, pero `u` / `:undo` debería cubrir los casos más simples.

Repetir el ultimo cambio

El comando Repetir, ejecutado con la tecla de punto o punto (`.`), Es más útil de lo que parece. Una vez aprendido, te encontrarás usándolo a menudo.

Mando	:	Descripción
.		Repite el ultimo cambio
10.		Repite el último cambio 10 veces.

Entonces, para un ejemplo muy simple, si realiza un cambio en la línea 1 escribiendo `i I Esc`, con el siguiente resultado:

```
1 I made a mistake
2 made a mistake
3 made a mistake
```

Su cursor estará en la posición 1 de la línea 1, y todo lo que necesita hacer para arreglar las siguientes dos líneas es presionar `j`. dos veces, es decir, `j` para bajar una línea y `.` Para repetir el último cambio, que fue la adición del `I` No es necesario volver al modo Insertar dos veces para corregir esas líneas.

Se vuelve mucho más potente cuando se usa para repetir [macros](#) .

Copiar, Cortar y Pegar

En Vim, estas operaciones se manejan de manera diferente a lo que podría estar acostumbrado

en casi cualquier otro editor o procesador de textos moderno (`Ctrl-C` , `Ctrl-X` , `Ctrl-V`). Para entender, necesita saber un poco acerca de los registros y movimientos.

Nota: esta sección no cubrirá la copia y corte del Modo Visual o el desplazamiento de rango, ya que están fuera del alcance del Modo Normal y la edición básica.

Registros

Vim utiliza el concepto de *registros* para manejar el movimiento de texto dentro del propio programa. Windows tiene un solo portapapeles para este propósito, que es análogo a un solo registro en Vim. Al copiar, cortar y pegar en Vim, hay formas de utilizar un flujo de trabajo de edición similar (donde no tiene que pensar en los registros), pero también hay posibilidades mucho más complejas.

Un registro está dirigido a la entrada / salida de un comando prefijando el comando con " y un nombre de letra minúscula.

Mociones

Un movimiento en Vim es cualquier comando que mueve la posición del cursor a otra parte. Al copiar, cortar y pegar en el Modo normal, las posibilidades de selección de texto para el movimiento solo están limitadas por su conocimiento de los movimientos. Algunos serán ilustrados a continuación.

Copiado y corte

Las operaciones básicas de copiar y cortar comandos se basan en `y` ('yank', para copiar) `yd` ('delete', para cortar). Verás las similitudes en la siguiente tabla.

Mando	:	Descripción
<code>y</code> {movimiento}		Copie ('yank') el texto indicado por el movimiento en el registro predeterminado
<code>yy</code>		Copie la línea actual en el registro predeterminado, en <i>línea</i>
<code>Y</code>		Copie la línea actual en el registro predeterminado (sinónimo para <code>yy</code>)
<code>"ayiw</code>		Copie la palabra en la que se encuentra el cursor en el registro 'a'
<code>20 "byy</code>		Copie veinte líneas, comenzando desde el cursor, en el registro 'b'
<code>d</code> {movimiento}		Cortar ('eliminar') el texto indicado por el movimiento en el registro predeterminado
<code>dd</code>		Cortar la línea actual en el registro predeterminado, en <i>línea</i>

Mando	:	Descripción
re		Corte desde el cursor hasta el final de la línea en el registro predeterminado (NO un sinónimo para dd)
"adiw		Corta la palabra donde está el cursor en el registro 'a'
20 "bdd		Cortar veinte líneas, comenzando desde el cursor, en el registro 'b'

Nota: cuando algo se copia o corta en *línea* , el comportamiento de pegado que se muestra a continuación colocará el texto antes o después de la *línea* actual (en lugar del cursor). Ejemplos para aclarar.

Paliza

Hay varias formas de pegar en Vim, dependiendo de lo que estés tratando de lograr.

Mando	:	Descripción
pag		Pegue lo que esté en el registro predeterminado <i>después</i> del cursor
PAG		Pegue lo que esté en el registro predeterminado <i>antes</i> del cursor
"ap		Pegue el contenido del registro 'a' después del cursor.
"cP		Pegue el contenido del registro 'c' antes del cursor.

Entonces, ¿cómo realizo un corte y pegado realmente simple?

Si tengo el siguiente texto:

```
1 This line should be second
2 This line should be first
```

Puedo hacer el corte y pegado más simple colocando mi cursor en algún lugar de la línea 1 y escribiendo `ddp` . Aquí están los resultados:

```
1 This line should be first
2 This line should be second
```

¿Que pasó? `dd` 'Corta' la primera línea (línea) en el registro predeterminado, que solo contendrá una cosa a la vez, como el portapapeles de Windows, `yp` pega la línea después de la actual, que acaba de cambiar debido al comando `dd` .

Aquí hay un ejemplo no tan simple. Necesito mover un par de palabras. (Esto es artificial e innecesario, pero puede aplicar este principio a trozos más grandes de código).

```
1 These words order out are of
```

Puedo repetir `w` para llegar a la 'o' al frente de 'order' (o `b` si lo escribí y me di cuenta de mi error).

Entonces `"adaw` para poner 'orden' en el registro 'a'.

Entonces `w` para llegar a la 'a' en 'son'.

Después de esto, escribiría `"bdaw` para poner 'are' en el registro 'b'. Ahora tengo esto desplegado:

```
1 These words out of
```

Para ser claros, ahora 'orden' está en el registro 'a' y 'son' está en el registro 'b', como dos portapapeles separados.

Para ordenar las palabras correctamente, escribo `b` para llegar a la 'o' en 'out', y luego `"bP` para poner 'son' del registro 'b' delante de 'out':

```
1 These words are out of
```

Ahora escribo `A` para llegar al final de la línea, seguido de `Space Esc` (asumiendo que no había espacio después de 'de') y `"ap` para poner "orden" donde pertenece.

```
1 These words are out of order
```

Terminación

La finalización se puede utilizar para hacer coincidir las palabras utilizadas en un documento. Al escribir una palabra, `Ctrl p` o `Ctrl n` coincidirán con las palabras anteriores o siguientes similares en el documento.

Esto incluso se puede combinar con el modo `Ctrl-x` para completar líneas completas. Por ejemplo, escriba algo como:

```
This is an example sentence.
```

luego ve a la siguiente línea y comienza a escribir la misma oración:

```
Thi
```

y luego presione `Ctrl p` que resultará en:

```
This
```

Ahora, aún en el modo de inserción, presione `Ctrl x Ctrl p` y luego la siguiente palabra se completará, lo que resultará en:

```
This is
```

Continúe presionando `Ctrl x Ctrl p` hasta completar toda la línea.

Si sabes que quieres completar una línea completa, escribe esto:

```
This is an example sentence.
```

luego en la siguiente línea escribe:

```
Thi
```

y pulsa `x Ctrl l` para completar la línea.

Si la finalización que se está haciendo es un nombre de archivo `Ctrl x Ctrl f` puede usarse para completar ese directorio. Tipo:

```
~/Deskt
```

luego presione `Ctrl x Ctrl f` y:

```
~/Desktop
```

será completado (si está en esa ubicación). `Ctrl x Ctrl f` se puede usar repetidamente para enumerar los archivos en el escritorio.

Lea Comandos en modo normal (Edición) en línea:

<https://riptutorial.com/es/vim/topic/5250/comandos-en-modo-normal--edicion->

Capítulo 10: Cómo compilar Vim

Examples

Compilando en Ubuntu

Para construir vim desde la fuente en Ubuntu:

1. Obtenga una copia del código fuente descargando desde el [repositorio oficial de Vim en GitHub](#) .
2. Obtenga las dependencias ejecutando `$ sudo apt-get build-dep vim-gnome` o similar.
3. Vaya al directorio del código fuente de Vim: `cd vim/src`
4. Ejecutar `$./configure` . Puede personalizar la compilación (y habilitar las integraciones de lenguaje Perl, Python, etc.) al pasar las opciones de configuración. Vea `src/INSTALL` para una descripción general.
5. Ejecutar `$ make` .
6. Finalice la instalación ejecutando `$ sudo make install` . Como su administrador de paquetes no administra su Vim autocompilado, se colocará en `/usr/local/bin/vim` , en lugar de `/usr/bin/vim` . Por lo tanto, para ejecutarlo, debe especificar la ruta completa o asegurarse de que `/usr/local/bin` esté antes de `/usr/bin` en su `PATH` (por lo general lo está).
7. (Opcional) Elimine la versión de Vim provista por la distribución si ya la tenía instalada: `$ sudo apt-get remove vim vim-runtime gvim` .

Para verificar la instalación, puede ejecutar `$ which vim` que debería devolver algo como `/usr/local/bin/vim` si la instalación fue exitosa.

Lea [Cómo compilar Vim en línea](https://riptutorial.com/es/vim/topic/3737/como-compilar-vim): <https://riptutorial.com/es/vim/topic/3737/como-compilar-vim>

Capítulo 11: Complementos

Examples

Fugitivo vim

Fugitive Vim es un complemento de Tim Pope que proporciona acceso a los comandos git que puedes ejecutar sin dejar vim.

Algunos comandos comunes incluyen:

```
:Gedit - edite un archivo en el índice y escríbalo para poner en escena los cambios
:Gstatus - equivalente al git status de git status
:Gblame - trae división vertical de la producción de git blame
:Gmove - para git mv
:Gremove - para git rm
:Git - ejecuta cualquier comando
```

También agrega elementos a la `statusline` como indicación de la rama actual.

Por favor, consulte su [GitHub](#) para más detalles e instrucciones de instalación.

NERD Tree

NERD TREE es un complemento de scroolouse que te permite explorar el sistema de archivos mientras usas vim. Puede abrir archivos y directorios a través de un sistema de árbol que puede manipular con el teclado o el mouse.

Agregue esto a su `.vimrc` para iniciar NERDTree automáticamente cuando se inicie vim:

```
autocmd vimenter * NERDTree
```

Para cerrar NERDTree automáticamente si es la única ventana que queda, agregue esto a su `.vimrc`:

```
autocmd bufenter * if (winnr("$") == 1 && exists("b:NERDTree") && b:NERDTree.isTabTree()) | q
| endif
```

Se recomienda asignar una combinación de teclas al comando `NERDTreeToggle`. Agregue esto a su `.vimrc` (este ejemplo usa `Ctrl + N`)

```
map <C-n> :NERDTreeToggle<CR>
```

Los detalles completos y las instrucciones de instalación se pueden ver en su [Github](#) .

Lea Complementos en línea: <https://riptutorial.com/es/vim/topic/9976/complementos>

Capítulo 12: Complementos de tipo de archivo

Examples

¿Dónde poner los plugins de filetype personalizados?

Los complementos de tipo de archivo para `foo` filetype se originan en ese orden:

1. `$HOME/.vim/ftplugin/foo.vim` . Tenga cuidado con lo que ponga en ese archivo, ya que puede ser reemplazado por `$VIMRUNTIME/ftplugin/foo.vim` - debajo de windows, será en su lugar `$HOME/vimfiles/ftplugin/foo.vim`
2. `$VIMRUNTIME/ftplugin/foo.vim` . Como todo en `$VIMRUNTIME` , este archivo no debe modificarse.
3. `$HOME/.vim/after/ftplugin/foo.vim` . Este archivo es el último, por lo que es el lugar ideal para la configuración específica de *su* tipo de archivo.

Lea Complementos de tipo de archivo en línea:

<https://riptutorial.com/es/vim/topic/7734/complementos-de-tipo-de-archivo>

Capítulo 13: Configuraciones útiles que se pueden poner en .vimrc

Sintaxis

- establecer mouse = a
- establecer envoltura
- nmap j gj
- nmap k gk

Examples

Mover hacia arriba / abajo las líneas mostradas al envolver

Por lo general, `J` y `K` mueven hacia arriba y hacia abajo las líneas de archivo. Pero cuando haya terminado, puede que desee que suban y bajen las líneas **mostradas** .

```
set wrap " if you haven't already set it
nmap j gj
nmap k gk
```

Habilitar la interacción del ratón

```
set mouse=a
```

Esto permitirá la interacción del mouse en el editor `vim` . El ratón puede

- cambiar la posición actual del cursor
- seleccionar texto

Configurar el registro predeterminado para ser utilizado como portapapeles del sistema.

```
set clipboard=unnamed
```

Esto hace posible copiar / pegar entre Vim y el portapapeles del sistema sin especificar ningún registro especial.

`yy` coloca la línea actual en el portapapeles del sistema

`p` pega el contenido del portapapeles del sistema en Vim

Esto solo funciona si su instalación de Vim tiene soporte para el portapapeles. Ejecute el siguiente comando en el terminal para verificar si la opción del portapapeles está disponible: `vim --version`

| grep clipboard

Lea Configuraciones útiles que se pueden poner en .vimrc en línea:

<https://riptutorial.com/es/vim/topic/6560/configuraciones-utiles-que-se-pueden-poner-en--vimrc>

Capítulo 14: Configurando Vim

Examples

El archivo vimrc

El archivo `.vimrc` (pronunciado Vim-wreck) es un archivo de configuración de Vim. Contiene comandos que serán ejecutados por Vim cada vez que se inicie.

Por defecto, el archivo está vacío o no existe; Puedes usarlo para personalizar tu entorno Vim.

Para averiguar dónde espera Vim que se almacene el archivo vimrc, abra Vim y ejecute:

```
:echo $MYVIMRC
```

Unix: en un sistema Unix como Mac o Linux, su vimrc se llamará `.vimrc` y, por lo general, estará ubicado en su directorio de inicio (`$HOME/.vimrc`).

Windows: en Windows se llamará `_vimrc` y se ubicará en su directorio de inicio (`%HOMEPATH%/_vimrc`).

En el inicio, Vim buscará en varios lugares un archivo vimrc. Lo primero que existe se usa, los otros se ignoran. Para una referencia completa vea el artículo de documentación `:h $MYVIMRC` .

¿Qué opciones puedo usar?

Si no sabe qué opciones debe usar, es posible que le interese el comando `:options` .

Esto abrirá una división con todas las opciones de Vim listadas y con su valor actual mostrado. Hay 26 secciones para mostrar todas las opciones que puedes probar.

p.ej

```
4 displaying text

scroll    number of lines to scroll for CTRL-U and CTRL-D
          (local to window)
          set scr=20
scrolloff  number of screen lines to show around the cursor
          set so=5
wrap      long lines wrap
          set nowrap    wrap

...

```

En una línea de valor (por ejemplo, `set nowrap`) puede presionar `CR` para alternar el valor (si es un valor binario). En una línea de opción (por ejemplo, `wrap long line wrap`) puede presionar `CR` para acceder a la documentación de esta opción.

Archivos y directorios

Independientemente de lo que haga para personalizar Vim, NUNCA debe suceder fuera de `$HOME` :

- en Linux, BSD y Cygwin, `$HOME` es usualmente `/home/username/` ,
- en Mac OS X, `$HOME` es `/Users/username/` ,
- en Windows, `$HOME` suele ser `C:\Users\username\` .

La ubicación canónica de `vimrc` y su directorio `vim` encuentra en la raíz de ese directorio `$HOME` :

- en sistemas similares a Unix

```
$HOME/.vimrc      <-- the file
$HOME/.vim/      <-- the directory
```

- en Windows

```
$HOME\_vimrc      <-- the file
$HOME\vimfiles\  <-- the directory
```

El diseño anterior está garantizado para funcionar, ahora y en el futuro.

Vim 7.4 hizo posible mantener su encantador `vimrc` **dentro de** su directorio `vim` . Realmente es una buena idea, aunque solo sea porque hace que sea más fácil mover la configuración.

Si usa 7.4 exclusivamente, lo siguiente será suficiente:

- en sistemas similares a Unix

```
$HOME/.vim/vimrc
```

- en Windows

```
$HOME\vimfiles\vimrc
```

Si desea los beneficios de un `vim/` autónomo `vim/` pero usa tanto la versión 7.4 como una versión anterior, o solo una versión anterior, la solución más simple y preparada para el futuro es poner esta línea *y solo esta*:

```
runtime vimrc
```

en este archivo:

- en sistemas similares a Unix

```
$HOME/.vimrc
```

- en Windows

```
$HOME\_vimrc
```

y realice su configuración en `$HOME/.vim/vimrc` o `$HOME/vimfiles/vimrc`.

Opciones

Hay tres tipos de opciones:

- opciones booleanas,
- opciones de cadena,
- opciones de números.

Para comprobar el valor de una opción,

- uso `:set option?` para comprobar el valor de una opción,
- uso `:verbose set option?` Para ver también dónde se colocó por última vez.

Configuración de opciones booleanas

```
set booloption      " Set booloption.
set nobooloption    " Unset booloption.

set booloption!     " Toggle booloption.

set booloption&     " Reset booloption to its default value.
```

Configuración de opciones de cadena

```
set stroption=baz   " baz

set stroption+=buzz " baz,buzz
set stroption^=fizz " fizz,baz,buzz
set stroption-=baz  " fizz,buzz

set stroption=      " Unset stroption.

set stroption&     " Reset stroption to its default value.
```

Configuración de opciones de número

```
set numoption=1     " 1

set numoption+=2    " 1 + 2 == 3
set numoption-=1    " 3 - 1 == 2
set numoption^=8    " 2 * 8 == 16
```

Usando una expresión como valor

- utilizando concatenación:

```
execute "set stroption=" . my_variable
```

- utilizando `:let`

```
let &stroption = my_variable
```

Ver `:help :set` y `:help :let`.

Mapeos

- No ponga comentarios después de las asignaciones, se romperán las cosas.
- Use `:map <F6>` para ver qué se asigna a `<F6>` y en qué modo.
- Utilice `:verbose map <F6>` para ver también dónde se mapeó por última vez.
- `:map` y `:map!` son demasiado genéricos Utilice `:n[nore]map` para las asignaciones de modo normal, `:i[nore]map` para el modo de inserción `:x[nore]map` para el modo visual, etc.

Mapeos recursivos

Use asignaciones *recursivas* **solo** si pretende usar otras asignaciones en sus asignaciones:

```
nnoremap b      B
nmap    <key> db
```

En este ejemplo, `b` está hecho para funcionar como `B` en modo normal. Como usamos `b` en un mapeo *recursivo*, presionar `<key>` funcionará como `dB`, no como `db`.

Asignaciones no recursivas

Use asignaciones *no recursivas* **solo** si pretende usar comandos predeterminados en sus asignaciones, que es casi siempre lo que desea:

```
nnoremap <key> db
```

En este ejemplo, usamos `b` en un mapeo *no recursivo*, por lo que al presionar la tecla siempre funcionará como `db`, ya sea que reasignemos `b` o no.

Ejecutando un comando desde un mapeo

```
nnoremap <key> :MyCommand<CR>
```

Ejecutando múltiples comandos desde un mapeo

```
nnoremap <key> :MyCommand <bar> MyOtherCommand <bar> SomeCommand<CR>
```

Llamar a una función desde un mapeo

```
nnoremap <key> :call SomeFunction()<CR>
```

Mapeando un mapeo <Plug>

```
map <key> <Plug>name_of_mapping
```

Ve `:help map-commands` `:help key-notation` y `:help <plug>`.

Ver [mapeos clave en Vim](#) para leer más

Variables

Como la mayoría de los lenguajes de scripting, vimscript tiene variables.

Uno puede definir una variable con el comando: `:let` :

```
let variable = value
```

y eliminarlo con `:unlet` :

```
unlet variable
```

En Vim, las variables se pueden incluir en el ámbito anteponiendo una sola letra y dos puntos a su nombre. Los autores de complementos usan esa característica para imitar las opciones:

```
let g:plugin_variable = 1
```

Ver `:help internal-variables`.

Comandos

- No olvide la explosión para permitir que Vim sobrescriba ese comando la próxima vez que vuelva a cargar su vimrc.
- Los comandos personalizados deben comenzar con un carácter en mayúscula.

Ejemplos

```
command! MyCommand call SomeFunction()  
command! MyOtherCommand command | Command | command
```

- Ver `:help user-commands`.

Funciones

- No olvide el bang para permitir que Vim sobrescriba esa función la próxima vez que vuelva a cargar el script donde se define la función.
- Las funciones personalizadas deben comenzar con un carácter en mayúscula (funciones globales) o con `s:` (funciones locales de script), o deben tener un prefijo con el nombre asociado al complemento de carga automática donde están definidas (por ejemplo, en `{&rtp}/autoload/foo/bar.vim` podríamos definir `foo#bar#functionname()`).
- Para poder usar los parámetros en la función, use `a:parameter_name`. Las funciones variables se pueden definir con puntos suspensivos `...`, para acceder a los parámetros use `a:000` (lista de todos los parámetros), o `a:0` (número de parámetros igual a `len(a:000)`), `a:1` primero sin nombre parámetros, y así sucesivamente.
- Las funciones pueden llamarse así `:call MyFunction(param1, param2)`
- Cada línea en una función comienza implícitamente con `a :` tanto, todos los comandos son comandos de dos puntos
- Para evitar que la función continúe su ejecución en caso de error, es mejor anotar la firma de la función con la `abort`

Ejemplo

```
function! MyFunction(foo, bar, ... ) abort
    return a:foo . a:bar . (a:0 > 0 ? a:1 : '')
endfunction
```

Funciones de script

Si solo planea usar su función en el archivo donde está definida (ya sea porque ha roto una función más grande en partes más pequeñas o porque la usará en un comando, una asignación, ...), puede prefijar con `s:` evitando ensuciar su espacio de nombres global con funciones internas inútiles:

```
function! s:my_private_function() " note we don't need to capitalize the first letter this time
    echo "Hi!"
endfunction
```

Usando `s:` funciones de mapeos

Si su función local de secuencia de comandos se va a utilizar en una asignación, debe hacer referencia a ella con el prefijo especial `<SID>` :

```
nnoremap <your-mapping-key> :call <SID>my_private_function()<CR>
```

Ver `:help user-functions`.

Sin embargo, tenga en cuenta que desde Vim 7, se considera una práctica recomendada definir

las abreviaturas, los comandos y los menús de los complementos (ft), y definir funciones en los complementos de carga automática, excepto las funciones que los complementos deben usar cuando se cargan. Esto significa que hoy en día la necesidad de llamar a los scripts funciones locales desde mapeos no es tan pertinente como solía ser.

Grupos de autocomando

- Los grupos de autocomando son buenos para la organización, pero también pueden ser útiles para la depuración. Piense en ellos como pequeños espacios de nombres que puede habilitar / deshabilitar a voluntad.

Ejemplo

```
augroup MyGroup
  " Clear the autocmds of the current group to prevent them from piling
  " up each time you reload your vimrc.
  autocmd!

  " These autocmds are fired after the filetype of a buffer is defined to
  " 'foo'. Don't forget to use 'setlocal' (for options) and '<buffer>'
  " (for mappings) to prevent your settings to leak in other buffers with
  " a different filetype.
  autocmd FileType foo setlocal bar=baz
  autocmd FileType foo nnoremap <buffer> <key> :command<CR>

  " This autocmd calls 'MyFunction()' everytime Vim tries to create/edit
  " a buffer tied to a file in '/path/to/project/**/'.
  autocmd BufNew,BufEnter /path/to/project/**/* call MyFunction()
augroup END
```

Ver :help autocommand .

Condicionales

```
if v:version >= 704
  " Do something if Vim is the right version.
endif

if has('patch666')
  " Do something if Vim has the right patch-level.
endif

if has('feature')
  " Do something if Vim is built with 'feature'.
endif
```

Ver :help has-patch y :help feature-list .

Opciones de configuración

Comúnmente usaría `:set` para configurar opciones a su gusto en su `.vimrc` . Hay muchas opciones que se pueden cambiar.

Por ejemplo, para usar espacios para la sangría:

```
:set expandtab
:set shiftwidth=4
:set softtabstop=4
```

Resaltado de sintaxis

Active el resaltado de sintaxis cuando el terminal tenga colores.

```
if &t_Co > 2 || has("gui_running")
  syntax on
end
```

Mostrar espacios en blanco finales y pestañas. Mostrar pestañas puede ser especialmente útil cuando se buscan errores en Makefiles.

```
set list listchars=tab:\|_,trail:.
highlight SpecialKey ctermfg=DarkGray
```

Esquemas de color

Vim viene con varios esquemas de color preinstalados. En Linux, los esquemas de color que vienen con Vim se almacenan en `/usr/share/vim/vim74/colors/` (donde 74 es su número de versión, sin períodos); MacVim los almacena en

`/Applications/MacVim.app/Contents/Resources/vim/runtime/colors .`

Cambio de esquemas de color

El comando `colorscheme` cambia la combinación de colores actual.

Por ejemplo, para establecer el esquema de color en "robokai":

```
:colorscheme robokai
```

El esquema de color predeterminado se denomina creativamente `default` , por lo tanto, para volver a usarlo

```
:colorscheme default
```

Para ver todos los esquemas de color instalados actualmente, escriba `:colorscheme` seguido de espacio y luego `tab` o `ctrl d` .

Instalación de combinaciones de colores

Los esquemas de color instalados por el usuario se pueden colocar en `~/.vim/colors/` . Una vez que se agrega un esquema de color a este directorio, aparecerá como una opción para el comando `colorscheme` .

Para encontrar nuevas **combinaciones de** colores, hay sitios como [vimcolors](#) que contienen una variedad de **combinaciones** de colores. También hay herramientas como [vim.ink](#) y [Vivify](#) para ayudarlo a crear sus propios esquemas de color, o puede crearlos a mano.

Enumerar línea de enumeración

Para habilitar - escriba:

```
:set number 0 :set nu .
```

Para deshabilitar - escriba:

```
:set nonumber 0 :set nonu .
```

Para habilitar la enumeración *relativa* a la ubicación del cursor, escriba:

```
:set relativenumber .
```

Para deshabilitar la enumeración *relativa* a la ubicación del cursor, escriba:

```
:set norelativenumber .
```

Nota: para cambiar si la línea actual muestra el número de línea real o 0 , use los comandos `:set number` y `:set nonumber` mientras el atributo `relativenumber` esté activo.

Complementos

Los complementos de Vim son complementos que se pueden usar para cambiar o mejorar la funcionalidad de vim.

Hay una buena lista de complementos en [vimawesome](#)

Lea **Configurando Vim en línea:** <https://riptutorial.com/es/vim/topic/2235/configurando-vim>

Capítulo 15: Consejos y trucos para aumentar la productividad.

Sintaxis

- : set relativenumber
- : establecer número
- : set nonumber /: set nonu
- : pwd

Observaciones

Esta recarga automática solo ocurrirá si edita su `vimrc` en una versión completa de vim que admita `autocmd`.

Examples

Macros rápidos "desechables"

Añade esto a tu `vimrc`:

```
nnoremap Q @q
```

Para comenzar a grabar la macro "desechable", use `qq`. Para finalizar la grabación pulsa `q` (en modo normal para ambos).

Para ejecutar la macro grabada, use `Q`

Esto es útil para las macros que necesita repetir muchas veces seguidas, pero que probablemente no se volverán a utilizar después.

Usando la función de finalización de ruta dentro de Vim

Esto es muy común, memoriza una ruta a un archivo o carpeta, abre Vim y trata de escribir lo que acaba de memorizar, pero no está 100% seguro de que sea correcto, así que cierra el editor y comienza de nuevo.

Cuando desea la función de finalización de ruta, tiene un archivo `/home/ubuntu/my_folder/my_file` y está editando otro archivo que hace referencia a la ruta de la anterior:

Ingrese al modo insertar: `inserte` o hágalo de la manera que desee. A continuación, escribe `/h`. Cuando el cursor está debajo de `h`, presione `Ctrl x` y luego `Ctrl f` para que el editor lo complete en `/home/` porque el patrón `/h` es único

Ahora, suponga que tiene dos carpetas dentro de `/home/ubuntu/` llamado `my_folder_1` `my_folder_2`

y quieres la ruta `/home/ubuntu/my_folder_2`

como siempre:

Entrar en modo de inserción

escribe `/h` presiona `Ctrl x` y luego `Ctrl f` . Ahora tiene `/home/` Next add `u` después de `/ home /` y presione `Ctrl x` y luego `Ctrl f` . Ahora tienes `/home/ubuntu/` porque esa ruta es única. Ahora, escriba `my` after `/home/ubuntu/` y presione `Ctrl x` y luego `Ctrl f` . El editor completará su palabra hasta `my_folder_` y verá el árbol de directorios, así que use las teclas de flecha para elegir la que desee.

Activar números de línea relativos

Para eliminar algunas líneas de texto cuando no conoce el número exacto de líneas a eliminar, pruebe con `10dd` , `5dd` , `3dd` hasta que elimine todas las líneas.

Los números de línea relativos resuelven este problema, supongamos que tenemos un archivo que contiene:

```
sometimes, you see a block of
text. You want to remove
it but you
cannot directly get the
exact number of
lines to delete
so you try
10d , 5d
3d until
you
remove all the block.
```

Entra en modo NORMAL: `Esc`

Ahora, ejecute `:set relativenumber` . Una vez hecho esto, el archivo se mostrará como:

```
3 sometimes, you see a block of
2 text. You want to remove
1 it but you
0 cannot directly get the
1 exact number of
2 lines to delete
3 so you try
4 10d , 5d
5 3d until
6 you
7 remove all the block.
```

donde `0` es el número de línea de la línea actual y también muestra el número de línea real delante del número relativo, por lo que ahora no tiene que estimar los números de líneas de la línea actual para cortar o eliminar o, peor aún, contarlos uno por uno.

Ahora puede ejecutar su comando habitual como `6dd` y está seguro del número de líneas.

También puede usar la forma abreviada del mismo comando `:set rnu` para activar los números relativos y `:set nonu` para desactivar la misma.

Si también `:set number` o tiene `:set number` ya `:set number`, obtendrá el número de línea de la línea en la que se encuentra el cursor.

```
3 sometimes, you see a block of
2 text. You want to remove
1 it but you
4 cannot directly get the
1 exact number of
2 lines to delete
3 so you try
4 10d , 5d
5 3d until
6 you
7 remove all the block.
```

Visualización de números de línea

Para ver los números de línea desde la vista predeterminada ingrese

```
:set number
```

Para ocultar números de línea

```
:set nonumber
```

También hay un atajo para arriba. `nu` es igual que el `number`.

```
:set nonu
```

Para ocultar números de línea, también podemos usar

```
:set nu!
```

Mapeos para salir del modo Insertar

Muchos usuarios de Vim encuentran que la `Esc` es demasiado difícil de alcanzar y terminan encontrando otro mapeo al que es fácil acceder desde la fila de casa. Tenga en cuenta que `Ctrl - [` puede ser equivalente a `Esc` en un teclado inglés, y es mucho más fácil de alcanzar.

j k

```
inoremap jk <ESC>
```

Este es realmente fácil de activar; solo aplasta tus dos primeros dedos en la fila de casa al mismo tiempo. También es difícil de activar accidentalmente ya que "jk" nunca aparece en ninguna palabra en inglés, y si estás en modo normal no hace nada. Si no escribes demasiado "blackjack", considere agregar también `inoremap kj <ESC>` para no tener que preocuparse por el tiempo de las dos teclas.

Bloq Mayús

Linux

En Linux, puedes usar `xmodmap` para hacer que `Caps Lock` haga lo mismo que `Esc` . Pon esto en un archivo:

```
!! No clear Lock
clear lock
!! make caps lock an escape key
keycode 0x42 = Escape
```

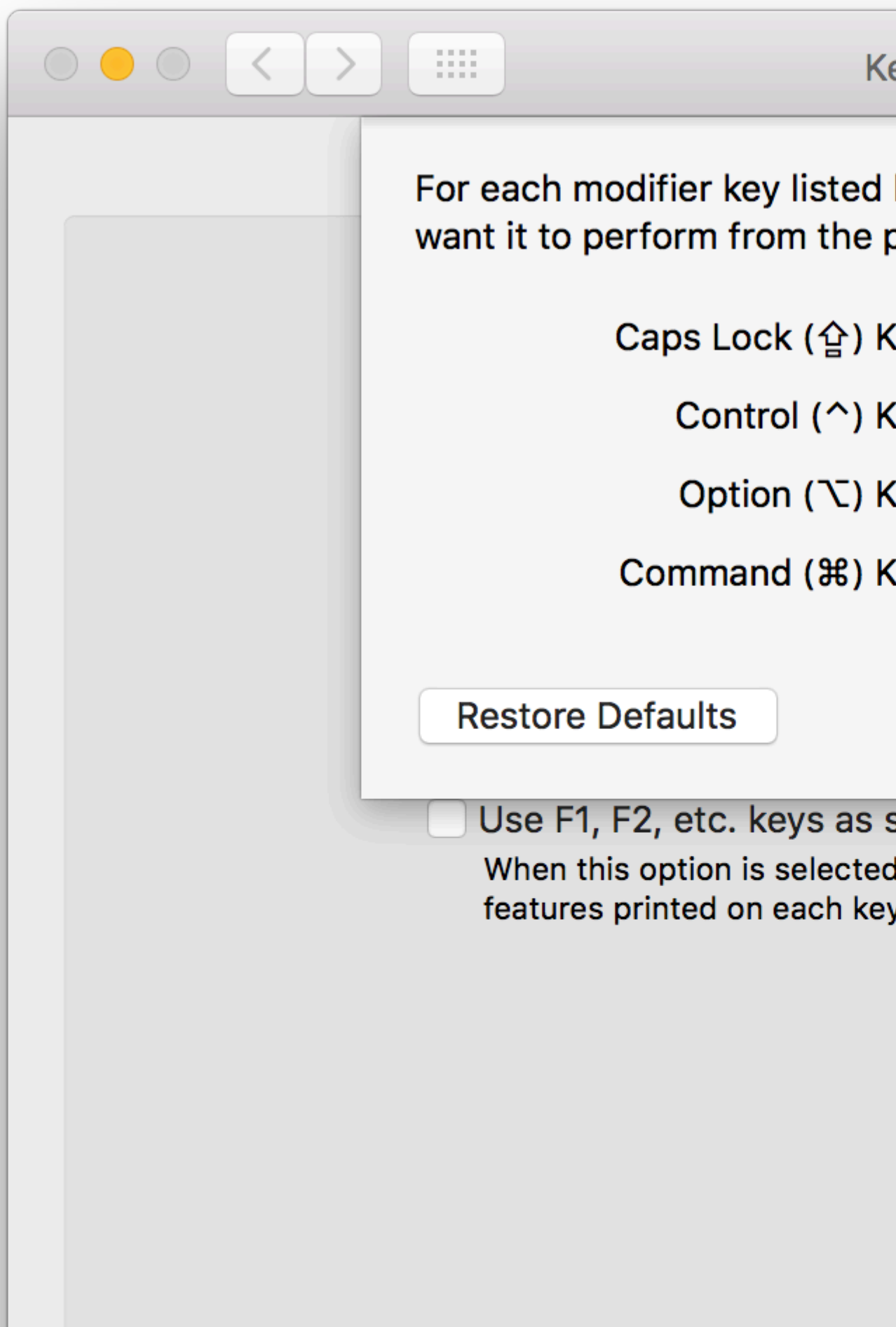
A continuación, ejecute el `xmodmap file` . Esto remapsula `Caps Lock` a `Esc` .

Windows

En Windows puedes usar [SharpKey](#) o [AutoHotkey](#) .

Mac OS

Si tiene macOS 10.12.1 o posterior, puede volver a asignar `Caps Lock` to `Escape` usando las Preferencias del sistema. Seleccione Teclado, vaya a la pestaña Teclado y haga clic en Teclas modificadoras.



- tecla de escape, <CR> - tecla enter):

```
vi{<ESC>/\%Vfoo<CR>
```

ahora puedes saltar entre las coincidencias dentro del bloque presionando `n p`. Si tiene la opción `hlsearch` habilitada, esto resaltará todas las coincidencias. `\%V` es una parte especial del patrón de búsqueda, que le dice a vim que busque solo en el área seleccionada visualmente. También puedes hacer un mapeo como este:

```
:vnoremap g/ <ESC>/\%V
```

Después de esto, el comando anterior se reduce a lo siguiente:

```
vi{g/foo<CR>
```

Otro truco útil es imprimir todas las líneas que contienen el patrón:

```
vi{  
: '<, '>g/foo/#
```

El rango `'<, '>` se inserta automáticamente.

Ver `:help range` y `:help :g`.

Copiar, mover o eliminar la línea encontrada.

Muchos usuarios se encuentran en una situación en la que solo quieren copiar, mover o eliminar una línea rápidamente y regresar a donde estaban.

Por lo general, si desea mover una línea que contiene la palabra `lot` debajo de la línea actual, escriba algo como:

```
/lot<Esc>dd<C-o>p
```

Pero para aumentar la productividad, puede utilizar este acceso directo en estos casos:

```
" It's recommended to turn on incremental search when doing so  
set incsearch  
  
" copy the found line  
cnoremap $t <CR>:t''<CR>  
" move the found line  
cnoremap $m <CR>:m''<CR>  
" delete the found line  
cnoremap $d <CR>:d<CR>``
```

Así que una búsqueda como esta:

```
/lot$m
```

movería la línea que contiene el `lot` debajo de la línea en la que estaba su cursor cuando comenzó la búsqueda.

Escribe un archivo si te olvidas de `sudo` antes de iniciar vim

Este comando guardará el archivo abierto con derechos sudo

```
:w !sudo tee % >/dev/null
```

También puedes mapear `w!!` escribir un archivo como root

```
:cnoremap w!! w !sudo tee % >/dev/null
```

Recargar automáticamente vimrc al guardar

Para volver a cargar `vimrc` automáticamente al guardar, agregue lo siguiente a su `vimrc` :

```
if has('autocmd') " ignore this section if your vim does not support autocommands
  augroup reload_vimrc
    autocmd!
    autocmd! BufWritePost $MYVIMRC,$MYGVIMRC nested source %
  augroup END
endif
```

y luego, por última vez, escriba:

```
:so $MYVIMRC
```

La próxima vez que guarde su `vimrc` , se volverá a cargar automáticamente.

`nested` es útil si está utilizando `vim-airline`. El proceso de carga de la aerolínea desencadena algunos autocomandos, pero ya que estás en el proceso de ejecutar un autocomando se omiten. `nested` permite activar autocomandos anidados y permite que la línea aérea se cargue correctamente.

Completar la línea de comando

configura `wildmenu` para activar las sugerencias de finalización para la línea de comando. Ejecuta lo siguiente

```
set wildmenu
set wildmode=list:longest,full
```

Ahora si dices, pestaña `:color` ,

Obtendrás

```
256-jungle Benokai BlackSea C64 CandyPaper Chasing_Logic ChocolateLiquor
```

```
:color 0x7A69_dark
```

Lea [Consejos y trucos para aumentar la productividad.](https://riptutorial.com/es/vim/topic/3382/consejos-y-trucos-para-aumentar-la-productividad-) en línea:

<https://riptutorial.com/es/vim/topic/3382/consejos-y-trucos-para-aumentar-la-productividad->

Capítulo 16: Construyendo desde vim

Examples

Comenzando una construcción

`:mak[e][!] [arguments]` iniciará el programa al que hace referencia la opción `makeprg`. De forma predeterminada, `makeprg` se establece en "make", pero se puede configurar para invocar cualquier programa apropiado.

Todos los `[arguments]` (pueden ser varios) se pasan a `makeprg` como si se hubiera invocado con `!{makeprg} [arguments]`.

La salida del programa invocado se analiza en busca de errores de acuerdo con la opción `'errorformat'`. Si se encuentra algún error, la ventana de revisión rápida se abre para mostrarlos.

`:cnext` `:cprev` se puede usar para `:cprev` entre los errores mostrados en la ventana de revisión rápida. `:cc` saltará al error debajo del cursor.

Se debe tener en cuenta que en los sistemas donde `gnumake` está instalado y configurado correctamente, generalmente no hay necesidad de definir `&makeprg` para nada más que su valor predeterminado para compilar proyectos de un solo archivo. Por lo tanto, en C, C ++, Fortran ... simplemente escriba `:make %<` para compilar el archivo actual. Según el archivo fuente en el directorio actual, `!./%<` lo ejecutará. Las opciones de compilación se pueden controlar a través de `$CFLAGS`, `$CXXFLAGS`, `$LDFLAGS`, etc. Consulte la documentación de la `make` respecto a las *reglas implícitas*.

Lea *Construyendo desde vim en línea*: <https://riptutorial.com/es/vim/topic/3321/construyendo-desde-vim>

Capítulo 17: Convertir archivos de texto de DOS a UNIX con vi

Observaciones

El carácter `^M` representa un retorno de carro en Vim (`<cm>` o simplemente `<CR>`). Vim muestra este carácter cuando al menos en línea en el archivo utiliza finales de línea `LF`. En otras palabras, cuando Vim considera que un archivo tiene `fileformat=unix` pero algunas líneas sí tienen retornos de carro (`CR`), los retornos de carro se muestran como `^M`.

Un archivo que tiene una sola línea con final de línea `LF` y varias líneas con finales de línea `CRLF` se crea con mayor frecuencia editando incorrectamente un archivo creado en un sistema basado en MSDOS. Por ejemplo, creando un archivo bajo un sistema operativo MSDOS, copiándolo a un sistema basado en UNIX y luego preparando una picadura de hash-bang (por ejemplo, `#!/bin/sh`) usando herramientas en el sistema operativo basado en UNIX.

Examples

Convertir un archivo de texto de DOS en un archivo de texto de UNIX

Muy a menudo tienes un archivo que se editó dentro de DOS o Windows y lo estás viendo bajo UNIX. Esto puede verse como el siguiente cuando ve el archivo con vi.

```
First line of file^M
Next Line^M
And another^M
```

Si desea eliminar el `^M`, puede ser que elimine cada `^M` a mano. Alternativamente, en vi después de pulsar `Esc`, puede ingresar lo siguiente en el indicador de modo de comando:

```
:1,$s/^M//g
```

Donde `^M` se ingresa con `Ctrl y v` juntos y luego `Ctrl y m` juntos.

Esto ejecuta el comando desde la primera línea '1' hasta la última línea '\$', el comando es sustituir 's' la '^M' por nada '' y para esto globalmente 'g'.

Usando el formato de archivo de Vim

Cuando Vim abre un archivo con `<CR><NL>` finales de línea (común en los sistemas operativos basados en MSDOS, también llamado `CRLF`), configurará el `fileformat` de `fileformat` a `dos`, puede verificar con:

```
:set fileformat?
```

```
fileformat=dos
```

O solo

```
:set ff?  
fileformat=dos
```

Para convertirlo en `<NL>` finales de línea (comunes en la mayoría de los sistemas operativos basados en UNIX, también llamados `LF`), puede cambiar la configuración del `fileformat` y Vim cambiará el búfer.

```
:set ff=unix
```

Lea [Convertir archivos de texto de DOS a UNIX con vi en línea](https://riptutorial.com/es/vim/topic/3827/convertir-archivos-de-texto-de-dos-a-unix-con-vi):

<https://riptutorial.com/es/vim/topic/3827/convertir-archivos-de-texto-de-dos-a-unix-con-vi>

Capítulo 18: Corrector ortográfico

Examples

Corrección ortográfica

Para activar la ejecución del corrector ortográfico vim `:set spell`. Para apagarlo, ejecute el `:set nospell`. Si siempre desea que el corrector ortográfico esté `set spell`, agregue `set spell` a su `vimrc`. Puede activar la ortografía solo para ciertos tipos de archivos con un comando automático.

Una vez que el corrector ortográfico esté activado, se resaltarán las palabras mal escritas. Escriba `]s` para pasar a la siguiente palabra mal escrita y `[s` para moverse a la anterior. Para ver una lista de ortografías corregidas, coloque el cursor en una palabra mal escrita y escriba `z=`. Puede escribir el número de la palabra con la que desea reemplazar la palabra mal escrita y presionar `<enter>` para reemplazarla, o puede presionar `enter` para dejar la palabra sin cambiar.

Con el cursor en una palabra mal escrita, también puede escribir `<number>z=` para cambiar a la `<number>` corrección sin ver la lista. Normalmente, `1z=` si crees que la primera suposición de vim probablemente sea la palabra correcta.

Establecer lista de palabras

Para establecer la lista de palabras que vim utilizará para la corrección ortográfica, configure la opción de `spelllang`. Por ejemplo

```
:set spelllang=en      # set to English, usually this is the default
:set spelllang=en_us   # set to U.S. English
:set spelllang=en_uk   # set to U.K. English spellings
:set spelllang=es      # set to Spanish
```

Si desea establecer el `spelllang` y activar la corrección ortográfica en un comando, puede hacer:

```
:setlocal spell spelllang=en
```

Lea Corrector ortográfico en línea: <https://riptutorial.com/es/vim/topic/3653/corrector-ortografico>

Capítulo 19: De desplazamiento

Examples

Desplazándose hacia abajo

Mando	Descripción
Ctrl + E	Desplácese una línea hacia abajo.
Ctrl + D	Desplazar media pantalla hacia abajo (configurable mediante la opción de <code>scroll</code>).
Ctrl + F	Desplácese una pantalla completa hacia abajo.
z +	Dibuja la primera línea debajo de la ventana en la parte superior de la ventana.

Desplazándose hacia arriba

Mando	Descripción
Ctrl + Y	Desplácese una línea hacia arriba.
Ctrl + U	Desplazar media pantalla hacia arriba (configurable mediante la opción de <code>scroll</code>).
Ctrl + B	Desplazar una pantalla completa hacia arriba.
z ^	Dibuja la primera línea sobre la ventana en la parte inferior de la ventana.

Desplazamiento relativo a la posición del cursor

Mando	Descripción
z	Vuelva a dibujar la línea actual en la parte superior de la ventana y coloque el cursor en el primer carácter que no esté en blanco en la línea.
zt	Como <code>z</code> pero deja el cursor en la misma columna.
z .	Vuelva a dibujar la línea actual en el centro de la ventana y coloque el cursor en el primer carácter que no esté en blanco en la línea.
zz	Como <code>z .</code> pero deja el cursor en la misma columna.
z-	Vuelva a dibujar la línea actual en la parte inferior de la ventana y coloque el cursor en el primer carácter que no esté en blanco en la línea.
zb	Como <code>z-</code> pero deja el cursor en la misma columna.

Lea De desplazamiento en línea: <https://riptutorial.com/es/vim/topic/3000/de-desplazamiento>

Capítulo 20: Diferencias entre Neovim y Vim

Examples

Archivos de configuración

En Vim, su archivo de configuración es `~/.vimrc`, con más archivos de configuración en `~/.vim`.

En Neovim, los archivos de configuración se encuentran en `~/.config/nvim`. Tampoco hay `~/.nvimrc` archivo `~/.nvimrc`. En su lugar, use `~/.config/nvim/init.vim`.

Puede importar su configuración de Vim directamente en Neovim usando este comando de Unix:

```
ln -s ~/.vimrc ~/.config/nvim/init.vim
```

Lea [Diferencias entre Neovim y Vim en línea](https://riptutorial.com/es/vim/topic/7848/diferencias-entre-neovim-y-vim): <https://riptutorial.com/es/vim/topic/7848/diferencias-entre-neovim-y-vim>

Capítulo 21: El operador de puntos

Examples

Uso básico

El operador de punto repite la última acción realizada, por ejemplo:

archivo `test.tx`

```
helo, World!  
helo, David!
```

(cursor en [1] [1])

Ahora realizar una `cwHello<Esc>` (Cambiar Palabra `helo` a `Hello`)

Ahora el búfer se ve así:

```
Hello, World!  
helo, David!
```

(cursor en [1] [5])

Después de escribir `j_` el cursor está en [2] [1].

Ahora ingresa el `.` y la última acción se realiza de nuevo:

```
Hello, World!  
Hello, David!
```

(cursor en [2] [5])

Establecer sangría

Esto es muy útil al establecer sangría de su código

```
if condition1  
if condition2  
# some commands here  
endif  
endif
```

mueva el cursor a la segunda línea, luego `>>` , el código se sangrará a la derecha.

Ahora puedes repetir tu acción al continuar en la tercera línea, luego presionar `.` dos veces, el resultado será

```
if condition1  
    if condition2
```

```
    # some commands here  
endif  
endif
```

Lea El operador de puntos en línea: <https://riptutorial.com/es/vim/topic/3665/el-operador-de-puntos>

Capítulo 22: Encontrar y reemplazar

Examples

Comando sustituto

Este comando:

```
:s/foo/bar/g
```

sustituye cada aparición de `foo` con `bar` en la línea actual.

```
fool around with a foodie
```

se convierte en

```
barl around with a bardie
```

Si omite la última `/g`, solo reemplazará la primera aparición en la línea. Por ejemplo,

```
:s/foo/bar
```

En la línea anterior se convertiría

```
barl around with a foodie
```

Este comando:

```
:5,10s/foo/bar/g
```

Realiza la misma sustitución en las líneas 5 a 10.

Este comando

```
:5,$s/foo/bar/g
```

realiza la misma sustitución desde la línea 5 hasta el final del archivo.

Este comando:

```
:%s/foo/bar/g
```

Realiza la misma sustitución en todo el búfer.

Si estás en modo visual y presionas los dos puntos, aparecerá el símbolo `'<, '>`. Entonces

puedes hacer esto

```
: '<', '>'s/foo/bar/g
```

y haga que la sustitución ocurra dentro de su selección de modo visual.

Este comando:

```
:%s/foo/bar/gc
```

es equivalente al comando anterior, pero solicita confirmación en cada aparición gracias a la marca `/c` (para "confirmación").

Ver `:help :s` y `:help :s_flags`.

Véase también [esta sección sobre rangos de línea de comando](#).

Reemplazar con o sin expresiones regulares

Este comando de sustitución puede usar [expresiones regulares](#) y coincidirá con cualquier instancia de `foo` seguida de cualquier (uno) carácter desde el período `.` en Expresiones regulares coincide con cualquier carácter, por lo que el siguiente comando coincidirá con todas las instancias de `foo` seguidas de cualquier carácter en la línea actual.

```
:s/foo./bar/g
```

```
1 fooinf fooes fool foobar foosup
```

se convertirá

```
1 barnf bars bar barar barup
```

Si quieres hacer coincidir el literal `.` período puede escapar de él en el campo de búsqueda con una barra invertida `\`.

```
:s/foo\./bar/g
```

```
1 fooinf fooes foo.l foo.bar foosup
```

se convertirá

```
1 fooinf fooes barl barbar foosup
```

O deshabilite todas las coincidencias de patrones siguiendo el comando `s` con `no`.

```
:sno/foo./bar/g
```

```
1 fooing fooes foo.l foo.bar foosup
```

generará un error

```
E486: Pattern not found
```

Lea **Encontrar y reemplazar en línea**: <https://riptutorial.com/es/vim/topic/3533/encontrar-y-reemplazar>

Capítulo 23: Espacio en blanco

Introducción

Aquí es cómo puedes limpiar los espacios en blanco.

Observaciones

Ver [transcripción de vimcast 4](#)

Examples

Eliminar espacios finales en un archivo

Puede eliminar los espacios finales con el siguiente comando.

```
:%s/\s\+$//e
```

Este comando se explica de la siguiente manera:

- entrar en modo Comando con `:`
- haga esto a todo el archivo con `%` (el valor predeterminado sería para la línea actual)
- acción sustitutiva `s`
- `/` inicio del patrón de búsqueda
- `\s` caracter del espacio en blanco
- `\+` escape + signo, uno o más espacios deben coincidir
- antes de que la línea termine `$`
- `/` Fin del patrón de búsqueda, inicio del patrón de reemplazo.
- `/` Fin del patrón de reemplazo. Básicamente, reemplazar con nada.
- `e` suprimir mensajes de error si no hay coincidencia encontrada

Eliminar líneas en blanco en un archivo

Puede eliminar todas las líneas en blanco en un archivo con el siguiente comando: `g / ^ $ / d`

Este comando se explica de la siguiente manera:

- entrar en modo Comando con `:`
- `g` es un comando global que debe ocurrir en todo el archivo
- `/` inicio del patrón de búsqueda
- el patrón de búsqueda de la línea en blanco es `^g`
- `/` final del patrón de búsqueda
- Ex comando `d` borra una línea

Convertir pestañas en espacios y espacios en pestañas

Puedes convertir las pestañas en espacios haciendo lo siguiente:

Primero verifica que la [barra de expansión](#) esté apagada

```
:set noexpandtab
```

Entonces

```
:retab!
```

El cual reemplaza espacios de cierta longitud con pestañas.

Si habilita [expandtab](#) nuevamente `:set expandtab` entonces y ejecute `:retab!` comando entonces todas las pestañas se convierten en espacios.

Si desea hacer esto para el texto seleccionado, primero seleccione el texto en [modo visual](#) .

Lea [Espacio en blanco en línea](https://riptutorial.com/es/vim/topic/8288/espacio-en-blanco): <https://riptutorial.com/es/vim/topic/8288/espacio-en-blanco>

Capítulo 24: Expresiones regulares

Observaciones

ejecute `:h pattern` para ver mucha información relacionada con expresiones regulares

Examples

Palabra

Vim tiene operadores especiales para hacer coincidir el principio de palabra, palabra, final, etc. `\<` representa el principio de una palabra y `\>` representa el final de una palabra.

Buscar `/\<foo\>` en el siguiente texto solo devolverá el último foo.

el fútbol no es tonto foo

Lea Expresiones regulares en línea: <https://riptutorial.com/es/vim/topic/6533/expresiones-regulares>

Capítulo 25: Expresiones regulares en modo Ex

Examples

Editar una expresión regular en modo Ex

Supongamos que está buscando un patrón de `Title Case` minúsculas en un archivo de texto grande y desea editar una expresión regular incorrecta:

1. Primero, vaya al modo `Ex` escribiendo `q`:
2. Ahora verá todos los comandos que escribió en el modo de `commandline` de `commandline`, presione `j` para pasar a la expresión regular que desea editar (`/\v[AZ]\w+\s[AZ]\w+`)
3. Una vez hecho esto, presione `ESC` para ir al modo normal
4. Luego presiona `Enter` para ejecutar la búsqueda

Aquí hay una captura de pantalla que muestra una búsqueda de `Title Case`

```
1 Lorem Ipsum is simply dummy text  
0 Lorem Ipsum has been the industry  
>\galley of type and scrambled  
1 It has survived not only five  
>\unchanged.  
2 It was popularised in the 1960s  
>\recently with desktop publishing
```

<https://riptutorial.com/es/vim/topic/6472/expresiones-regulares-en-modo-ex>

Capítulo 26: Extendiendo Vim

Observaciones

Un complemento es una secuencia de comandos o un conjunto de secuencias de comandos que cambia el comportamiento predeterminado de Vim, ya sea agregando características no existentes o extendiendo las funciones existentes.

Las "características no existentes" añadidas a menudo incluyen:

- comentando
- detección de sangría,
- autocompletar,
- coincidencia difusa,
- soporte para un lenguaje específico,
- etc.

A menudo, las "características existentes" extendidas incluyen:

- omni-finalización,
- texto-objetos y movimientos,
- tirando y poniendo
- línea de estado,
- buscar y reemplazar,
- búfer / ventana / pestaña de cambio de página,
- plegable,
- etc.

Examples

Cómo funcionan los complementos

Un complemento podría presentarse como un solo archivo que contiene 30 líneas de vimscript o como 20 MB de vimscript / python / ruby / lo que sea dividido en muchos archivos en una docena de directorios que dependen de una serie de herramientas externas.

El primero es obviamente fácil de instalar y administrar, pero el último podría detener un gran desafío.

El principio

La opción '`runtimepath`' le dice a Vim dónde buscar los scripts de tiempo de ejecución. El valor predeterminado hace que Vim busque los scripts en los siguientes directorios *en orden* :

- en sistemas similares a UNIX

- **\$HOME/.vim/**
- \$VIM/vimfiles/
- \$VIMRUNTIME/
- \$VIM/vimfiles/after/
- **\$HOME/.vim/after/**

- en Windows

- **\$HOME/vimfiles/**
- \$VIM/vimfiles/
- \$VIMRUNTIME/
- \$VIM/vimfiles/after/
- **\$HOME/vimfiles/after/**

De los directorios anteriores, solo instale los complementos en negrita. Los otros causarán inestabilidad por ninguna buena razón. La instalación de un complemento se reduce a colocar cada uno de sus componentes en el directorio correcto debajo de `$HOME/.vim/` o `$HOME/vimfiles/`.

El metodo manual

Plugin de un solo archivo

Coloque el archivo en `$HOME/.vim/plugin` o `$HOME/vimfiles/plugin`

Esto generaría el complemento en el inicio de Vim. Ahora el usuario podría usar todo lo definido en él. Sin embargo, si el complemento necesita activación, el usuario debe ejecutar el comando cada vez que quiera usarlo o agregar el comando a `.vimrc`

Haz

Un paquete es una estructura de directorio que utiliza el complemento. Se compone de todos los archivos del complemento en los subdirectorios correspondientes.

Para instalar dicho complemento, los subdirectorios deben fusionarse con sus equivalentes en `$HOME/.vim/plugin`. Sin embargo, este enfoque conduce a la mezcla de los archivos de diferentes complementos en los mismos directorios y posiblemente podría dar lugar a problemas de espacio de nombres.

Otro enfoque es copiar todo el directorio en `$HOME/.vim/bundle`.

Al usar este enfoque, debe haber al menos un archivo `.vim` directorio `$HOME/.vim/bundle/autoload`. Estos archivos serían originados por vim en el inicio.

Nota: Dependiendo del sistema operativo del usuario, el prefijo de todas las rutas puede ser `$HOME/vimfiles`. Para más detalles vea [cómo funcionan los complementos](#).

VAM

Vundle

Vundle es un administrador de plugins para Vim.

Instalando Vundle

(Los detalles completos de la instalación se pueden encontrar en el [Inicio rápido de Vundle](#))

1. Instale [Git](#) y clone Vundle en `~/.vim/bundle/Vundle.vim` .
2. Configure los complementos agregando lo siguiente en la parte superior de su `.vimrc` , agregando o eliminando los complementos según sea necesario (los complementos en la lista son meramente ilustrativos)

```
set nocompatible          " be iMproved, required
filetype off              " required

" set the runtime path to include Vundle and initialize
set rtp+=~/.vim/bundle/Vundle.vim
call vundle#begin()
" alternatively, pass a path where Vundle should install plugins
"call vundle#begin('~/.vim/bundle')

" let Vundle manage Vundle, required
Plugin 'VundleVim/Vundle.vim'

" All of your Plugins must be added before the following line
call vundle#end()          " required
filetype plugin indent on  " required
" To ignore plugin indent changes, instead use:
"filetype plugin on

"place non-Plugin stuff after this line
```

3. Instalar complementos: iniciando Vim y ejecutando `:PluginInstall` .

Formatos de complementos compatibles

Los siguientes son ejemplos de diferentes formatos soportados. Mantenga los comandos del complemento entre `vundle#begin` y `vundle#end` .

Ubicación del complemento	Uso
plugin en GitHub	Plugin 'tpope/vim-fugitive'
complemento de http://vim-scripts.org/vim/scripts.html	Plugin 'L9'

Ubicación del complemento	Uso
El complemento de Git no está alojado en GitHub	Plugin 'git://git.wincent.com/command-t.git'
git repos en su máquina local (es decir, cuando trabaja en su propio complemento)	Plugin 'file:///home/gmarik/path/to/plugin'
El script sparkim vim se encuentra en un subdirectorio de este repositorio llamado vim. Pase el camino para establecer el runtimepath correctamente.	Plugin 'rstacruz/sparkup', {'rtp': 'vim/'}
Instala L9 y evita un conflicto de nombres si ya has instalado una versión diferente en otro lugar.	Plugin 'ascenator/L9', {'name': 'newL9'}

Trabajar en una cuenta compartida, por ejemplo, en el nodo principal del clúster puede generar problemas desde el punto de uso del disco por el directorio `.vim`. Hay un par de paquetes que ocupan una cantidad considerable de espacio en disco, por ejemplo, [YCM](#). Por lo tanto, elija sabiamente su directorio de complementos de `Vundle`, y es muy fácil hacerlo configurando `rtp`. Y también si planea instalar cualquier complemento vim, no haga directamente `git clone` en el directorio del `bundle`. Utilice la forma `Vundle`.

El futuro: los paquetes.

Ver `:help packages`.

Patógeno

[vim-pathogen](#) es un administrador de tiempo de `runtimepath` creado por Tim Pope para facilitar la instalación de complementos y archivos de tiempo de ejecución en sus propios directorios privados.

Instalación de patógeno

1. Coloque el patógeno en `~/vim/bundle` (aquí con Git, pero no es obligatorio):

```
git clone https://github.com/tpope/vim-pathogen.git
```

2. Agrega las siguientes líneas a la parte superior de tu `.vimrc`:

```
" enable vim-pathogen
runtime bundle/vim-pathogen/autoload/pathogen.vim
execute pathogen#infect()
```

- la directiva de `runtime` especifica la ruta al script de carga automática de `vim-pathogen`;
- `execute pathogen#infect()` inicia.

Una vez iniciado, Pathogen iniciará automáticamente un barrido a través de las carpetas en `~/ .vim/bundle` y cargará el complemento desde cada una de ellas.

Usando patogeno

1. Coloque el directorio de nivel superior de su complemento en `~/ .vim/bundle/` para que esté disponible la próxima vez que inicie Vim.
2. Ejecutar `:Helptags` para indexar la documentación de su nuevo complemento.

Beneficios

- Cada complemento reside en su propio directorio en `~/ .vim/bundle/` .
- Su `.vimrc` mantiene limpio de la configuración necesaria para cargar complementos.

El esfuerzo necesario para "administrar" un complemento se reduce así a:

- **ponga** su directorio de nivel superior bajo `~/ .vim/bundle/` para *instalarlo* ,
- **reemplazar** su directorio de nivel superior para *actualizarlo* ,
- **eliminar** su directorio de nivel superior para *desinstalarlo* .

La forma en que realice esas tres acciones (manualmente, a través de una herramienta de automatización, con Git / Svn / Hg / lo que sea ...) depende completamente de usted.

Lea *Extendiendo Vim en línea*: <https://riptutorial.com/es/vim/topic/3659/extendiendo-vim>

Capítulo 27: Huevos de Pascua

Examples

¡Ayuda!

Para el usuario angustiado, vim proporciona palabras de sabiduría.

```
:help!
```

Cuando te sientes triste

Problema: los usuarios de Vim no siempre están contentos.

Solución: Hazlos felices.

7.4

```
:smile
```

Nota: Requiere versión de parche \geq [7.4.1005](#)

La respuesta

Vim sabe la respuesta:

```
:help 42
```

Vim abrirá el documento `usr_42.txt` del manual y mostrará el siguiente texto:

¿Cuál es el significado de la vida, el universo y todo? **42**

Desafortunadamente, Douglas Adams, la única persona que sabía de qué se trataba realmente esta pregunta, ahora está muerto. Así que ahora te preguntarás cuál es el significado de la muerte ...

Buscando el Santo Grial

Mira esto:

```
:help holy-grail
```

Ceci n'est pas une pipe

Si buscas la sección de ayuda de `| o bar : :h bar` puedes ver:

```
bar
```

```
| To screen column [count] in the current line.  
exclusive motion. Ceci n'est pas une pipe.
```

Esta es una referencia a la pintura *La trahison des images* de René Magritte.



Quando un usuario se aburre

Buscar :h UserGettingBored

```
UserGettingBored      *UserGettingBored*  
                        When the user presses the same key 42 times.  
                        Just kidding! :-)
```

Cuchara

En lugar de buscar la ayuda del `fork` , puede buscar la ayuda de la `spoon` :

```
:h spoon  
  
fork spoon  
For executing external commands fork()/exec() is used when possible, otherwise  
system() is used, which is a bit slower. The output of ":version" includes ...
```

Caballeros que dicen Ni!

Mira esto:

```
:Ni!
```

Monty Python y el Santo Grial

nunmap

```
:help map-modes
```

:nunmap can also be used outside of a monastery.

Lea Huevos de Pascua en línea: <https://riptutorial.com/es/vim/topic/4656/huevos-de-pascua>

Capítulo 28: Insertar texto

Examples

Saliendo del modo insertar

Mando	Descripción
<Esc>	Deja el modo insertar, activa autocomandos y abreviaturas.
<C-[>	Exacto sinónimo de <Esc>
<Cc>	Deja el modo de inserción, no dispara autocommands.

A algunas personas les gusta usar un par de caracteres relativamente poco comunes como `jk` como acceso directo para `<Esc>` o `<C-[>` que puede ser difícil de alcanzar en algunos teclados:

```
inoremap jk <Esc>l
```

Diferentes maneras de entrar en el modo de inserción

Mando	Descripción
<code>a</code>	Añadir texto siguiendo la posición actual del cursor
<code>A</code>	Añadir texto al final de la línea actual
<code>i</code>	Insertar texto antes de la posición actual del cursor
<code>I</code>	Insertar texto antes del primer carácter no en blanco de la línea actual
<code>gI</code>	Insertar texto en la primera columna de la línea del cursor
<code>gi</code>	Insertar texto en la misma posición donde se dejó la última vez en el modo Insertar
<code>O</code>	Abra una nueva línea sobre la línea actual y agregue texto allí (CAPITAL <code>O</code>)
<code>o</code>	Abra una nueva línea debajo de la línea actual y agregue texto allí (minúscula <code>o</code>)
<code>s O c l</code>	Eliminar el carácter debajo del cursor y cambiar al modo de inserción
<code>s O c c</code>	Eliminar toda la línea y cambiar al modo Insertar
<code>C</code>	Eliminar desde la posición del cursor hasta el final de la línea e iniciar el modo de inserción

Mando	Descripción
<code>c{motion}</code>	Eliminar <code>{motion}</code> e iniciar el modo de inserción (ver Movimiento básico)

Métodos abreviados del modo Insertar

Mando	Descripción
<code><Cw></code>	Borrar palabra antes del cursor
<code><Ct></code>	Sangrar la línea actual con un <code>shiftwidth</code>
<code><Cd></code>	Línea de corriente unindent con un <code>shiftwidth</code>
<code><Cf></code>	reindente la línea, (mueva el cursor a la posición de sangría automática)
<code><Ca></code>	Insertar texto previamente insertado
<code><Ce></code>	Inserta el caracter abajo
<code><Ch></code>	Eliminar un carácter hacia atrás
<code><Cy></code>	Inserta el caracter arriba
<code><Cr>{register}</code>	Insertar el contenido de <code>{register}</code>
<code><Co>{normal mode command}</code>	ejecute <code>{normal mode command}</code> sin salir del modo de inserción
<code><Cn></code>	Siguiente opción de autocompletar para la palabra actual
<code><Cp></code>	Opción de autocompletar anterior para la palabra actual
<code><Cv></code>	Insertar un carácter por su valor ASCII en decimal
<code><Cv>x</code>	Insertar un carácter por su valor ASCII en hexadecimal
<code><Cv>u</code>	Insertar un carácter por su valor Unicode en hexadecimal.
<code><Ck></code>	Ingrese un digraph

Ejecutando comandos normales desde el modo de inserción

Mientras está en el modo de inserción, presione `<Co>` para abandonar temporalmente el modo de inserción y ejecutar un comando normal de una sola vez.

Ejemplo

`<Co>2w` salta a la segunda palabra a la izquierda y regresa al modo de inserción.

Nota: Repitiendo con `.` solo repetirá las acciones de volver al modo insertar

Esto permite algunas asignaciones útiles, por ejemplo,

```
inoremap <C-f> <Right>
inoremap <C-b> <Left>
inoremap <C-a> <C-o>^
inoremap <C-e> <C-o>$
```

Ahora `ctrl + a` pondrá el cursor al principio de la línea y `ctrl + e` - al final de la línea. Estas asignaciones se utilizan de forma predeterminada en `readline`, por lo que pueden ser útiles para las personas que desean coherencia.

Insertar texto en varias líneas a la vez

Presione `Ctrl + v` para entrar en el modo de bloqueo visual.

Use `↑ / ↓ / j / k` para seleccionar múltiples líneas.

Presiona `Shift + i` y comienza a escribir lo que quieras.

Después de presionar `Esc`, el texto se insertará en todas las líneas que seleccionó.

¡Recuerde que `Ctrl + c` no es 100% equivalente a `Esc` y no funcionará en esta situación!

Existen leves variaciones de `Shift + i` que puede presionar mientras se encuentra en el modo de bloque visual:

Llave	Descripción
<code>c o s</code>	Eliminar el bloque seleccionado y entrar en el modo de inserción
<code>do</code>	Eliminar las líneas seleccionadas (desde el cursor hasta el final) e ingresar al modo de inserción
<code>R</code>	Eliminar las líneas seleccionadas y entrar en el modo de inserción
<code>UNA</code>	Anexar a las líneas seleccionadas, con la columna al final de la primera línea

También tenga en cuenta que presionando `.` ¡Después de una operación de bloqueo visual se repetirá esa operación donde está el cursor!

Pegar texto usando el comando "pegar" del terminal

Si usa el comando `pegar` de su programa emulador de terminal, Vim interpretará la secuencia de caracteres como si estuvieran escritos. Eso causará todo tipo de efectos indeseables, particularmente malas críticas.

Para arreglar eso, desde el modo de comando:

```
:set paste
```

Luego pase al modo de inserción, con `i`, por ejemplo. Observe que el modo es ahora `-- INSERT` (paste) `--`. Ahora pegue con su comando de emulador de terminal, o con el mouse. Cuando haya terminado, vaya al modo de comando, con `Esc` y ejecute:

```
:set nopaste
```

Hay una forma más simple, cuando uno quiere pegar solo una vez. Ponga esto en su `.vimrc` (o use el complemento [unimpaired.vim](#)):

```
function! s:setup_paste() abort
  set paste
  augroup unimpaired_paste
    autocmd!
    autocmd InsertLeave *
      \ set nopaste |
      \ autocmd! unimpaired_paste
  augroup end
endfunction

nnoremap <silent> yo :call <SID>setup_paste()<CR>o
nnoremap <silent> yO :call <SID>setup_paste()<CR>O
```

Ahora se puede simplemente presionar `yo` para pegar el código debajo del cursor, y luego `<Esc>` para volver al modo normal / nopaste.

Cómo pegar desde un registro mientras está en modo inserción

Mientras está en el modo de inserción, puede usar `<Cr>` para pegar desde un registro, que se especifica en la siguiente pulsación de tecla. `<Cr>"` por ejemplo, las pastas del registro sin nombre (`"`).

Ver `:help registers`.

Comandos y accesos directos de inserción avanzados

Aquí hay una referencia rápida para comandos avanzados de inserción, formateo y filtrado / accesos directos.

Comando / Atajo	Resultado
<code>g +? + m</code>	Realizar codificación rot13, en movimiento <code>m</code>
<code>n + ctrl + a</code>	+ <code>n</code> al número debajo del cursor
<code>n + ctrl + x</code>	- <code>n</code> al número debajo del cursor

Comando / Atajo	Resultado
<code>g + q + m</code>	Formato de líneas de movimiento <i>m</i> a ancho fijo.
<code>: r ce w</code>	Líneas centrales en el rango <i>r</i> al ancho <i>w</i>
<code>: r le i</code>	Alinear a la izquierda las líneas en el rango <i>r</i> con sangría <i>i</i>
<code>: r ri w</code>	Alinear a la derecha las líneas en el rango <i>r</i> al ancho <i>w</i>
<code>! mc</code>	Filtrar líneas de movimiento <i>m</i> mediante comando <i>c</i> .
<code>n !! do</code>	Filtra <i>n</i> líneas a través del comando <i>c</i>
<code>: r do</code>	Rango de filtro <i>r</i> líneas a través del comando <i>c</i>

Deshabilitar la sangría automática para pegar el código

Al pegar texto a través de un emulador de terminal, la función de sangría automática *puede* destruir la sangría del texto pegado.

Por ejemplo:

```
function () {
  echo 'foo'
  echo 'bar'
  echo 'baz'
}
```

se pegará como

```
function () {
  echo 'foo'
  echo 'bar'
  echo 'baz'
}
```

En estos casos, use la opción `paste / nopaste` para deshabilitar / habilitar la función de sangría automática:

```
:set paste
:set nopaste
```

Además de esto, hay un enfoque más simple del problema: agregue la siguiente línea en su `.vimrc`:

```
set pastetoggle=<F3>
```

Y si quieres pegar como está desde el portapapeles. Solo presiona `F3` en el modo de `insert`, y pega. Presiona `F3` otra vez para salir del modo de `paste`.

Lea Insertar texto en línea: <https://riptutorial.com/es/vim/topic/953/insertar-texto>

Capítulo 29: Macros

Examples

Grabando una macro

Una forma de crear una macro es *grabarla* .

Comience a grabar una macro y guárdela en un registro (en este ejemplo, usaremos `a` , pero puede ser cualquier registro al que normalmente pueda copiar texto):

```
qa
```

Luego, ejecute los comandos que desea grabar en la macro (aquí, rodearemos el contenido de una línea con etiquetas ``):

```
I<li><ESC>A</li>
```

Cuando hayamos terminado con los comandos que queremos grabar en la macro, detenga la grabación:

```
q
```

Ahora, cada vez que deseamos ejecutar la secuencia grabada de comandos almacenados en `a` , use:

```
@a
```

y vim repetirá la secuencia grabada.

La próxima vez que desee repetir la última macro que se usó, puede escribir doble `@` :

```
@@
```

Y como un bono extra, es bueno recordar que si coloca un número antes de un comando, lo repetirá muchas veces. Por lo tanto, se repite la macro guardada en el registro `a` 20 veces con:

```
20@a
```

Edición de una macro vim

Algunas veces cometerá un error con una macro larga, pero preferiría editarla en lugar de regrabarla por completo. Puedes hacerlo usando el siguiente proceso:

1. Coloque la macro en una línea vacía con "`<register>p` .

Si su macro está guardada en el registro `a` , el comando es `"ap` .

2. Edite la macro según sea necesario.

3. Mueva la macro al registro correcto moviendo el cursor al principio de la línea y usando

`"<register>y$` .

Puedes reutilizar el registro original o usar otro. Si desea utilizar el registro `b` , el comando es `"by$` . O mediante `"<register>d$` (elimina la línea no utilizada)

Macros recursivas

Las macros de Vim también pueden ser recursivas. Esto es útil para cuando necesita actuar en cada línea (u otro objeto de texto) hasta el final del archivo.

Para grabar una macro recursiva, comience con un registro vacío. (Un registro se puede vaciar usando `q<register>q` .)

Elija un punto de inicio consistente en cada línea para comenzar y terminar.

Antes de finalizar la grabación, invoca la macro como el último comando. (Esta es la razón por la que el registro debe estar vacío: para que no haga nada, ya que la macro aún no existe).

Ejemplo, dado el texto:

```
line 1
line 2
line 3
foo bar
more random text
.
.
.
line ???
```

En el modo normal, con el cursor en la primera línea y un registro vacío `a` , se podría grabar esta macro:

```
qaI"<Esc>A"<Esc>j@a
```

Luego, con una sola invocación de `@a` , todas las líneas del archivo estarán ahora entre comillas dobles.

¿Qué es una macro?

Una macro es una serie de pulsaciones de teclado destinadas a ser "reproducidas" por Vim sin demora. Las macros pueden almacenarse en registros o variables, vincularse a teclas o ejecutarse en la línea de comandos.

Aquí hay una macro simple que pone en mayúscula la tercera `word` en una línea:

```
0wwgUiw
```

Esa macro podría *grabarse* en el registro `q` :

```
qq      start recording into register q
0wwgUiw
q       stop recording
```

o guardado directamente en el registro `q` :

```
:let @q = '0wwgUiw'
```

para ser reproducido con:

```
@q
```

Pero también se puede escribir directamente en la línea de comandos:

```
:normal 0wwgUiw
```

para reproducción instantánea a través del `:normal` comando `:normal .`

O poner en una variable:

```
:let myvar = '0wwgUiw'
```

para ser reproducido con:

```
@=myvar
```

O guardado como un mapeo:

```
nnoremap <key> 0wwgUiw
```

para reproducir presionando `<key>` .

Si desea almacenar una macro para su reutilización posterior, puede escribir en el modo de inserción:

```
<C-r>q
```

Esto inserta la macro en el registro `q` (en este ejemplo: `0wwgUiw`). Puede usar esta salida, por ejemplo, para definir la macro en su `vimrc` :

```
let @q='0wwgUiw'
```

Al hacerlo, el registro `q` se inicializa con esta macro cada vez que inicia vim.

Grabar y reproducir la acción (macros)

Con el comando `q` podríamos simplificar mucho trabajo tedioso en vim.

Ejemplo 1. Generar secuencia de matriz (1 a 20).

PASO 1. presione `i` para ingresar al modo de inserción, ingrese `1`

```
1
```

PASO 2. Registre la siguiente acción: "agregue el último número a la siguiente línea e incremente el número"

1. escriba `esc` para salir del modo de entrada
2. escriba `qa` para ingresar al modo de grabación, usando el búfer `a`
3. escriba `yy` y `p` para copiar la línea actual y péguela como la siguiente línea
4. escriba `ctrl + a` para incrementar el número
5. Escriba `q` nuevo para terminar el registro

```
1  
2
```

PASO 3. Repetir la acción 18 veces.

escriba `18@a` para reproducir la acción 3 y la acción 4 en el paso 2.

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20
```

Lea Macros en línea: <https://riptutorial.com/es/vim/topic/1447/macros>

Capítulo 30: Manipulando texto

Observaciones

Para incrementar y disminuir cosas como `11:59AM`, `3rd` y `XVIII`, use el complemento [vim-speeddating](#)

Examples

Convertir caso de texto

En modo normal:

- `~` invierte el caso del personaje debajo del cursor,
- `gu{motion}` minúscula el texto cubierto por `{motion}`,
- `gU{motion}` mayúsculas el texto cubierto por `{motion}`

Ejemplo (`^` marca la posición del cursor):

```

Lorem ipsum dolor sit amet.
      ^
Lorem ipSum dolor sit amet.    ~
Lorem IPSUM DOLOR sit amet.    gU2w
Lorem IPsum DOLOR sit amet.    gue
```

En modo visual:

- `~` invierte el caso del texto seleccionado,
- `u` minúsculas el texto seleccionado,
- `U` mayúsculas el texto seleccionado

Ejemplo (`^^^` marca la selección visual):

```

Lorem ipsum dolor sit amet.
      ^^^^^^^^^^^^^^^
Lorem ipSUM DOLOR SIT amet.    ~
Lorem ipSUM DOLOR SIT amet.    U
Lorem ipsum dolor sit amet.    u
```

Incremento y decremento de números y caracteres alfabéticos.

En el modo normal, podemos incrementar el número más cercano en la línea en o después del cursor con `<Ca>` y disminuirlo con `<Cx>`. En los siguientes ejemplos, la posición del cursor se indica mediante `^`.

Números crecientes y decrecientes

```
for i in range(11):  
    ^
```

<Cx> disminuye el número:

```
for i in range(10):  
    ^
```

10<Ca> incrementa en 10 :

```
for i in range(20):  
    ^
```

Incremento y decremento de caracteres alfabéticos.

Para hacer que el incremento y el decremento también funcionen con letras, use el comando `ex :set nrformats+=alpha` o agregue `set nrformats+=alpha` a su `.vimrc`.

Ejemplo de incremento:

```
AAD  
^
```

<Ca> incrementa a B :

```
ABD  
^
```

Ejemplo de decremento:

```
ABD  
^
```

<Cx> disminuye D a C :

```
ABC  
^
```

Incremento y decremento de números cuando se habilita el incremento / decremento alfabético

Tenga en cuenta que habilitar el incremento / decremento para trabajar con caracteres alfabéticos significa que debe tener cuidado de no modificarlos cuando realmente solo quiere modificar los números. Puede desactivar el incremento / decremento alfabético usando el comando `ex :set nrformats-=alpha` o simplemente puede ser consciente de ello y asegurarse [de pasar](#) al número

antes del incremento o decremento. Aquí está el ejemplo de " `for i in range(11):` " desde arriba para volver a trabajar mientras se establece el incremento / decremento alfabético:

Digamos que desea disminuir de 11 a 10 y el incremento / decremento alfabético está activo.

```
for i in range(11):  
    ^
```

Como el incremento / decremento alfabético está activo, para evitar modificar el carácter debajo del cursor, primero avance hacia el primer 1 usando el comando de movimiento de *modo normal* `f1` (que es `f` minúscula seguido del número 1, para no confundirse con una tecla de función):

```
for i in range(11):  
    ^
```

Ahora, como el cursor está en el número, puede disminuirlo con `<Cx>`. Al disminuir, el cursor se vuelve a colocar en el último dígito del número:

```
for i in range(10):  
    ^
```

Código de formato

En modo normal:

`gg` ir arriba

= entonces `G`

Usando "verbos" y "sustantivos" para la edición de texto

Una de las maneras de pensar acerca de los comandos que deben ejecutarse, de editar un texto de cierta manera, es como oraciones completas.

Un comando es una acción realizada en un objeto. Por eso tiene un verbo:

```
:normal i    " insert  
:normal a    " append  
:normal c    " overwrite  
:normal y    " yank (copy)  
:normal d    " delete
```

Algunas de estas palabras trabajan con un objeto como `d`, `c`, `y`. Dichos objetos pueden ser **palabra, línea, oración, párrafo, etiqueta**. Uno puede usar estos en combinación:

```
:normal dw    " deletes the text from the position of the cursor to the end of the next word  
:normal cw    " deletes the text from the cursor to the end of the next word and  
              " enters insert mode
```

También se podría usar un **modificador** para especificar con precisión dónde se debe ejecutar la acción:

```
:normal diw      " delete inside word. I.e. delete the word in which is the cursor.
:normal ciw      " removes the word, the cursor points at and enters insert mode
:normal ci"      " removes everything between the opening and closing quotes and
                 " enters insert mode
:normal cap      " change the current paragraph
:normal ct8      " remove everything until the next number 8 and enter insert mode
:normal cf8      " like above but remove also the number
:normal c/goal   " remove everything until the word 'goal' and enter insert mode
:normal ci{      " change everything inside the curly braces
```

Más recursos:

[Aprende a hablar vim - verbos, sustantivos y modificadores!](#)

[Aprendiendo Vim en 2014: Vim como lenguaje](#)

[Edición de VimSpeak utilizando gramática de voz](#)

[Lea Manipulando texto en línea: https://riptutorial.com/es/vim/topic/1707/manipulando-texto](https://riptutorial.com/es/vim/topic/1707/manipulando-texto)

Capítulo 31: Mejora deshacer y rehacer con un undodir

Examples

Configurando tu vimrc para usar un undodir

Desde la versión 7.3 de vim, se admite la función 'persistent_undo', lo que hace posible deshacer / rehacer cambios, incluso después de cerrar vim o reiniciar su computadora.

Es posible configurarlo agregando lo siguiente a su vimrc, pero primero cree un directorio, donde se guardarán sus archivos sin perfil. Puede crear el archivo en cualquier lugar, pero le recomiendo usar el directorio ".vim".

```
if has('persistent_undo')           "check if your vim version supports
  set undodir=$HOME/.vim/undo       "directory where the undo files will be stored
  set undofile                       "turn on the feature
endif
```

Después de agregar esto a vimrc y volver a obtener vimrc, puede usar la función usando los [comandos básicos de deshacer / rehacer](#)

Lea [Mejora deshacer y rehacer con un undodir en línea](#):

<https://riptutorial.com/es/vim/topic/7875/mejora-deshacer-y-rehacer-con-un-undodir>

Capítulo 32: Mociones y objetos de texto

Observaciones

Un objeto de texto en Vim es otra forma de especificar una porción de texto para operar. Se pueden utilizar con operadores o en modo visual, en lugar de movimientos.

Examples

Cambiando los contenidos de una cadena o lista de parámetros.

Digamos que tienes esta línea de código:

```
printf("Hello, world!\n");
```

Ahora diga que desea cambiar el texto a "Salir del programa".

Mando	Buffer	Mnemotécnico
ci"	printf(" ");	c hange i n el " .
Program exiting.\n<esc>	printf("Program exiting.\n");	

Lea Mociones y objetos de texto en línea: <https://riptutorial.com/es/vim/topic/4107/mociones-y-objetos-de-texto>

Capítulo 33: Modos - insertar, normal, visual, ex

Examples

Lo básico sobre los modos.

`vim` es un editor modal. Esto significa que en cualquier momento dentro de una sesión `vim`, el usuario estará en uno de los modos de operación. Cada uno de ellos ofrece un conjunto diferente de comandos, operaciones, enlaces de teclas ...

Modo normal (o modo de comando)

- El modo `vim` comienza en.
- Desde otros modos, normalmente accesibles por `Esc`.
- **Tiene la mayoría de los comandos de navegación y manipulación de texto.**

Ver `:help normal-mode`.

Modo de inserción

- Se accede comúnmente por: `a`, `i`, `A`, `I`, `c`, `s`.
- **Para insertar texto**.

Ver `:help insert-mode`.

Modo visual

- Se accede comúnmente por: `v` (carácter), `V` (línea), `<Cv>` (bloque).
- Básicamente, para **selección de texto**; La mayoría de los comandos normales están disponibles, más los adicionales para actuar sobre el texto seleccionado.

Ver `:help visual-mode`.

Seleccionar modo

- Accesible desde el modo de inserción con `<Cg>`.
- Similar al modo visual pero con muchos menos comandos disponibles.
- Contrariamente al modo de inserción, es posible escribir de inmediato.
- Raramente usado.

Ver `:help select-mode`.

Modo de reemplazo

- Accesible desde el modo normal con `R`
- Permite sobrescribir el texto existente.

Ver `:help replace-mode` .

Modo de línea de comandos

Ver `:help command-line-mode` .

Modo ex

Ver `:help Ex-mode` .

Lea Modos - insertar, normal, visual, ex en línea: <https://riptutorial.com/es/vim/topic/2231/modos---insertar--normal--visual--ex>

Capítulo 34: Movimiento

Examples

buscando

Saltando a los personajes

`f {char}` : pasa a la siguiente aparición de `{char}` a la derecha del cursor en la misma línea

`F {char}` : se mueve a la siguiente aparición de `{char}` a la izquierda del cursor en la misma línea

`t {char}` : muévete a la izquierda de la siguiente aparición de `{char}` a la derecha del cursor en la misma línea

`T {char}` : muévete a la derecha de la siguiente aparición de `{char}` a la izquierda del cursor en la misma línea

Salta hacia adelante / hacia atrás entre los 'resultados' a través de `; y, .`

Además, puede buscar palabras completas a través de `/<searchterm> Enter .`

Buscando cuerdas

`*` - pasar a la siguiente aparición de la palabra debajo del cursor

`#` - pasar a la aparición anterior de la palabra debajo del cursor

`/ searchterm Enter` te lleva a la siguiente coincidencia (búsqueda hacia adelante). Si usas `?` En lugar de `/`, la búsqueda va hacia atrás.

Salta entre los partidos vía `n` (siguiente) y `N` (anterior).

Para ver / editar sus búsquedas anteriores, escriba `/` y presione la tecla de flecha `hacia arriba` .

También son útiles estas configuraciones: (nota `:se` es igual a `:set`)

- `:se hls` HighLightSearch, resalta todas las coincidencias de búsqueda; use `:noh` para desactivar temporalmente la búsqueda / marca resaltada (`:set noh 0 :set nohls se` desactiva).
- `:se is 0 :set incs` activa la Búsqueda incremental, el cursor salta a la siguiente coincidencia automáticamente. (`:se nois` se apaga.)
- `:se ic` IgnoreCase, desactiva la sensibilidad a mayúsculas. (`:se noic` enciende de nuevo.)
- `:se scs` SmartCaSe, se puede usar cuando se establece IgnoreCase; hace que la sensibilidad de la caja (in) sea **inteligente** ! por ejemplo, `/the` buscará `the , The , ThE , etc.`

mientras que /The only buscará The .

Movimiento basico

Observaciones

- Cada movimiento puede usarse después de un comando de operador, por lo que el comando opera en el texto que comprende el alcance del movimiento.
- Al igual que los comandos del operador, los movimientos pueden incluir un conteo, por lo que puede moverse por 2w ords, por ejemplo.

Flechas

En Vim, las teclas de flecha / cursor normales (← ↓ ↑ →) funcionan como se espera. Sin embargo, para los tecladores táctiles, es más fácil usar las teclas alternativas de h j k l . En un teclado típico, están ubicados uno al lado del otro en la misma fila y son fácilmente accesibles con la mano derecha. La técnica mnemotécnica para recordar cuál es cuál de ellas va así:

- h / l : están ubicadas "más a la izquierda / derecha" entre las cuatro letras del teclado, por lo que son equivalentes a "ir a la izquierda / derecha" respectivamente;
- j - en minúscula "j" tiene su cola "abajo" debajo de las letras típicas, como una pequeña flecha, por lo que es equivalente a "bajar";
- k : a la inversa, la "k" minúscula tiene su "ascendente" arriba "encima de las letras típicas, como un pequeño puntero - por lo que es equivalente a" subir ".

Movimientos basicos

Todos los comandos a continuación deben hacerse en **modo normal** .

Mando	Descripción
h O izquierda	ir [contar] caracteres a la izquierda
j O abajo	ir [contar] los caracteres a continuación
k O arriba	ir [contar] los caracteres de arriba
l O derecho	ir [contar] caracteres a la derecha
gg	ir a la primera línea, o [contar] 'th línea, si se da
H	Ir a la primera línea en la pantalla visible.
METRO	ir a la línea media en la pantalla visible

Mando	Descripción
L	Ir a la última línea en la pantalla visible.
sol	ir a la última línea, o [contar] 'th línea, si se da
Casa O 0	ir al primer carácter de la línea
^	ir al primer carácter no en blanco de la línea
+	bajar una línea hasta el primer carácter no en blanco
-	subir una línea al primer carácter que no esté en blanco
\$ O fin	ir al final de la línea (si se da [cuenta] , vaya [contar - 1] líneas hacia abajo)
	vaya al carácter [count] 'th o vaya al principio de la línea si no se ha especificado count
f {char}	vaya a [contar] 'la ocurrencia de {char} a la derecha <i>inclusive</i>
F {char}	vaya a [contar] 'la ocurrencia de {char} a la izquierda <i>inclusive</i>
t {char}	vaya a [contar] 'la ocurrencia de {char} a la derecha <i>exclusiva</i>
T {char}	ir a [contar] 'la ocurrencia de {char} a la izquierda <i>exclusiva</i>
;	repetir los últimos tiempos f , t , F O T [contar]
,	repita la última f , t , F O T , en la dirección opuesta, [cuenta] veces
w	ir al comienzo de la siguiente palabra
segundo	ir al principio de la palabra anterior
mi	ir al final de la siguiente palabra
ge	ir al final de la palabra anterior
%	ir a pares coincidentes, por ejemplo (), [], {}, /* */ O #if, #ifdef, #else, #elif, #endif
{ }	párrafo anterior / siguiente
[{ }]	principio / final del bloque
'{carbonizarse}	Ir a la marca (marcar con m {char})
<CB> <CF>	página anterior / siguiente
<CO> <CI>	Regresa o avanza en la "lista de salto" (requiere la función <code>jumplist</code> , ver

Mando	Descripción
	<code>:help jumps</code>)

Nota: `b`, `e`, y `w` consideramos que una palabra son letras, números y guiones bajos de forma predeterminada (esto se puede configurar con la configuración de `iskeyword`). Cada uno de estos también puede ser capitalizado, haciendo que se salte todo lo que no sea un espacio en blanco también.

Nota: Vim reconoce dos tipos de movimiento: movimiento del operador (`:help movement`) y saltos (`:help jumplist`). Los movimientos como los ejecutados con `g` (`gg`, `G`, `g,`) cuentan como saltos, al igual que los cambios. Los cambios obtienen su propio jumplist, que es navegable como se mencionó anteriormente a través de `g, g;` (ver `:help changelist`). Los saltos no son tratados como comandos de movimiento por Vim

Cuando se mueve hacia arriba o hacia abajo a través de las líneas, el cursor retiene su columna como se esperaría. Si la nueva línea es demasiado corta, el cursor se mueve al final de la nueva línea. Si la columna está más allá del final de la línea, el cursor se muestra al final de la línea. El número de columna inicial aún se conserva hasta que se realiza una acción para modificarlo (como editar texto o mover la columna explícitamente).

Si la longitud de una línea excede el ancho de la pantalla, el texto se ajusta (bajo la configuración predeterminada, se puede configurar este comportamiento). Para moverse a través de líneas como se muestra en la pantalla, en lugar de líneas dentro del archivo, agregue `g` delante del comando usual. Por ejemplo, `gj` moverá el cursor a la posición que se muestra una línea debajo de su posición actual, incluso si está en la misma línea del archivo.

Buscando patrón

Vim admite el uso de expresiones regulares al buscar en un archivo.

El carácter para indicar que desea realizar una búsqueda es `/`.

La búsqueda más simple que puedes realizar es la siguiente

```
/if
```

Esto buscará el archivo completo para todas las instancias de `if`. Sin embargo, nuestra búsqueda `if` es en realidad una expresión regular que coincidirá con cualquier ocurrencia de la palabra `if` incluyendo los que están dentro de otras palabras.

Por ejemplo, nuestra búsqueda diría que todas las siguientes palabras coinciden con nuestra búsqueda: `if`, `spiffy`, `endif`, etc.

Podemos hacer búsquedas más complicadas usando expresiones regulares más complicadas.

Si nuestra búsqueda fuera:

```
/\if\>
```

entonces nuestra búsqueda solo devolvería coincidencias exactas a la palabra completa `if` . El `spiffy` y el `endif` antedichos no serían devueltos por la búsqueda, solamente `if` .

También podemos utilizar rangos. Dado un archivo:

```
hello1
hello2
hello3
hello4
```

Si queremos buscar aquellas líneas que contienen "hola" seguidas de un dígito entre 1 y 3, diríamos:

```
/hello[1-3]
```

Otro ejemplo:

```
/(?:\d*\.)?\d+
```

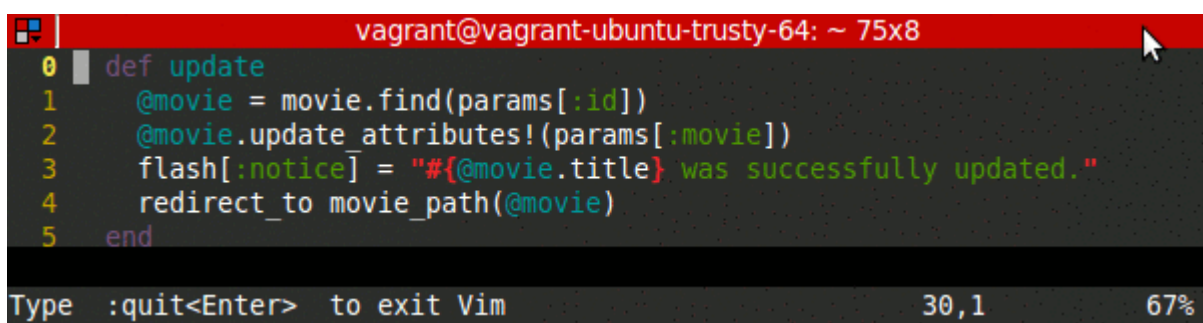
encontraría todos los números enteros y decimales en el archivo.

Navegando al principio de una palabra específica

Al editar texto, una tarea común es navegar a una palabra en particular en la pantalla. En estos ejemplos, exploramos cómo podemos navegar hasta la palabra `updated` . En aras de la coherencia en los ejemplos, nuestro objetivo es aterrizar en la primera letra de la palabra.

Salto a mitad de pantalla

M \$ B



```
vagrant@vagrant-ubuntu-trusty-64: ~ 75x8
0 | def update
1 |   @movie = movie.find(params[:id])
2 |   @movie.update_attributes!(params[:movie])
3 |   flash[:notice] = "#{@movie.title} was successfully updated."
4 |   redirect_to movie_path(@movie)
5 | end
Type :quit<Enter> to exit Vim          30,1          67%
```

Este enfoque es rápido, utilizando solo 3 pulsaciones. La desventaja, sin embargo, es que no es muy general, ya que no es común que nuestra línea objetivo quede justo en el centro de la pantalla. Aún así, es un movimiento útil cuando se realizan movimientos menos granulares.

Usando una cuenta

3j f u ; ;

```
vagrant@vagrant-ubuntu-trusty-64: ~ 75x8
0 def update
1   @movie = movie.find(params[:id])
2   @movie.update_attributes!(params[:movie])
3   flash[:notice] = "#{@movie.title} was successfully updated."
4   redirect_to movie_path(@movie)
5 end

1 change; before #116 2016/07/27 13:48:34 30,1 67%
```

A primera vista, esto puede parecer un paso atrás desde el primer enfoque debido a la cantidad de pulsaciones de teclas. Pero como usamos un conteo aquí en lugar de `M`, es más flexible. Podemos identificar rápidamente el recuento correcto para usar si `relativenumber` está habilitado. Para moverse a la palabra objetivo, usando `f` en combinación con `;` puede ser sorprendentemente efectivo, y ciertamente mejor que presionar `w` repetidamente. Si sobrepasas tu objetivo con `;`, se puede ir hacia atrás con `,`.

Búsqueda explícita

`/` arriba Entrar `n n`

```
vagrant@vagrant-ubuntu-trusty-64: ~ 75x8
0 def update
1   @movie = movie.find(params[:id])
2   @movie.update_attributes!(params[:movie])
3   flash[:notice] = "#{@movie.title} was successfully updated."
4   redirect_to movie_path(@movie)
5 end

Type :quit<Enter> to exit Vim 30,1 67%
```

Navegar a través de `/` puede ser muy potente. A menudo podemos saltar directamente a nuestra palabra objetivo al escribirla. Aquí solo escribimos los dos primeros caracteres con la esperanza de que coincidan de forma única con nuestra palabra. Desafortunadamente, hay varias coincidencias, pero podemos saltar rápidamente a la próxima coincidencia con `n`.

Búsqueda implícita

`/` y Espacio ingrese `w`

```
vagrant@vagrant-ubuntu-trusty-64: ~ 75x8
0 def update
1   @movie = movie.find(params[:id])
2   @movie.update_attributes!(params[:movie])
3   flash[:notice] = "#{@movie.title} was successfully updated."
4   redirect_to movie_path(@movie)
5 end

/up 30,1 67%
```

En algunos casos, puede ser más eficiente saltar *cerca de* nuestro objetivo en lugar de ir directamente a él. Aquí observamos que hay una letra que aparece con poca frecuencia, `y`, justo al lado del objetivo. Podemos añadir un `espacio` en nuestro término de búsqueda para disminuir

las posibilidades de que hubiera ganado algún otro `y` carácter a lo largo del camino. Esto también se puede usar con gran efecto con `f {char}`, como en el ejemplo *Usando un conteo*.

Usando marcas para moverse

Las marcas son como marcadores; Te ayudan a encontrar lugares en los que ya has estado.

TLDR

Configúrelos en modo normal con `m{a-zA-Z}`, y salte a ellos en modo normal o visual con `'{a-zA-Z}` (comilla simple) o ``{a-zA-Z}` (marca inversa). Las letras minúsculas son para las marcas dentro de un búfer, y las letras mayúsculas y los dígitos son globales. Vea sus marcas establecidas actualmente con `:marks`, y para más información vea `:help mark`.

Establecer una marca

La ayuda incorporada de Vim dice:

```
m{a-zA-Z}          Set mark {a-zA-Z} at cursor position (does not move
                   the cursor, this is not a motion command).
```

La marca mantendrá un registro de en qué línea y columna se colocó. No hay confirmación visual de que se haya establecido una marca, o si una marca tenía un valor anterior y se ha sobrescrito.

Saltar a una marca

La ayuda incorporada de Vim dice:

```
Jumping to a mark can be done in two ways:
1. With ` (backtick):    The cursor is positioned at the specified location
                        and the motion is exclusive.
2. With ' (single quote): The cursor is positioned on the first non-blank
                        character in the line of the specified location and
                        the motion is linewise.
```

Backtick usa la posición de la columna, mientras que Single-quote no lo hace. La diferencia entre simplemente le permite ignorar la posición de la columna de su marca si así lo desea.

Puede saltar entre marcas no globales en modo visual además del modo normal, para permitir la selección de texto basado en marcas.

Marcas globales

Las marcas globales (letras mayúsculas) permiten saltar entre archivos. Lo que eso significa es

que si, por ejemplo, la marca `A` se establece en `foo.txt`, luego desde `bar.txt` (en cualquier lugar de mi sistema de archivos), si salto a la marca `A`, mi búfer actual se reemplazará con `foo.txt`. Vim le pedirá que guarde los cambios.

Saltar a una marca en otro archivo **no se** considera un movimiento, y las selecciones visuales (entre otras cosas) no funcionarán como saltar a marcas dentro de un búfer.

Para volver al archivo anterior (`bar.txt` en este caso), use `:b[uffer] #` (es decir `:b[uffer] # :b# o :buffer#`).

Nota:

Marcas especiales

Hay ciertas marcas que Vim establece automáticamente (que puede sobrescribir usted mismo, pero probablemente no será necesario).

Por ejemplo (parafraseado de la ayuda de Vim):

```
`[` and `]`: jump to the first or last character of the previously changed or yanked text. {not in Vi}

`<` and `>`: jump to the first or last line (with ``) or character (with <code>`</code>) of the last selected Visual area in the current buffer. For block mode it may also be the last character in the first line (to be able to define the block). {not in Vi}.
```

Más, de la ayuda incorporada de Vim:

```
'' `` To the position before the latest jump, or where the last "m'" or "m`" command was given. Not set when the :keepjumps command modifier was used. Also see restore-position.

'" `"' To the cursor position when last exiting the current buffer. Defaults to the first character of the first line. See last-position-jump for how to use this for each opened file. Only one position is remembered per buffer, not one for each window. As long as the buffer is visible in a window the position won't be changed. {not in Vi}.

'. `.` To the position where the last change was made. The position is at or near where the change started. Sometimes a command is executed as several changes, then the position can be near the end of what the command changed. For example when inserting a word, the position will be on the last character. {not in Vi}

'" `"' To the cursor position when last exiting the current buffer. Defaults to the first character of the first
```



```
line. See last-position-jump for how to use this
for each opened file.
Only one position is remembered per buffer, not one
for each window. As long as the buffer is visible in
a window the position won't be changed.
{not in Vi}.
```

```
'^ `^
To the position where the cursor was the last time
when Insert mode was stopped. This is used by the
gi command. Not set when the :keepjumps command
modifier was used. {not in Vi}
```

Además, los caracteres (,) , { y } son marcas que saltan a la misma posición que sus comandos en modo normal, es decir, ' } hace lo mismo en modo normal que } .

Saltar a la línea específica.

Para saltar a una línea específica con número de dos puntos. Para saltar a la primera línea del uso de un archivo.

```
:1
```

Saltar a la línea 23

```
:23
```

Lea Movimiento en línea: <https://riptutorial.com/es/vim/topic/1117/movimiento>

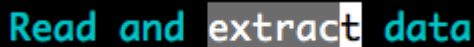
Capítulo 35: Objetos de texto VIM

Examples

Seleccione una palabra sin espacios en blanco circundantes

Supongamos que queremos seleccionar una palabra sin los espacios en blanco circundantes, use el objeto de texto `iw` para la palabra interna usando el modo visual:

1. Llegó al modo normal presionando `ESC`
2. Escribe `viw` al comienzo de una palabra
3. Esto seleccionará la palabra interior




Read and **extract** data

Seleccione una palabra con espacio en blanco circundante

Supongamos que queremos seleccionar una palabra con un espacio en blanco circundante, use el objeto de texto `aw` para una palabra usando el modo visual:

1. Llegó al modo normal presionando `ESC`
2. Escriba `vaw` al principio de una palabra
3. Esto seleccionará la palabra con espacio en blanco.



Read and **extract data**

Seleccionar texto dentro de una etiqueta

Podemos seleccionar un texto dentro de un `html` o `xml` etiqueta mediante el uso de selección visual `v` y el objeto de texto `it`.

1. Ir al modo normal py presionando `ESC`
2. Escriba `vit` desde cualquier lugar dentro de la sección `html` o `xml`
3. Esto seleccionará visualmente todo el texto dentro de la `tag`

```
11      <head>
10
9      <!-- Latest compiled and
8      <link rel="stylesheet"
7      <link rel="stylesheet"
6
5      <!-- Optional theme -->
4      <link rel="stylesheet"
3      >\css">
2      <!-- for datepicker -->
1      <link rel="stylesheet"
14     </head>
1
```

Todos los demás objetos de texto también se pueden utilizar para operar en el texto dentro de la etiqueta

1. `cit` : elimina el texto dentro de la etiqueta y la coloca en el modo de `insert`
2. `dit` : elimina el texto dentro de la etiqueta y permanece en modo `normal`
3. `cat` - eliminar alrededor de la etiqueta y colocar en modo de `insert`
4. `dat` : borra el texto alrededor de la etiqueta y permanece en modo `normal`

Lea Objetos de texto VIM en línea: <https://riptutorial.com/es/vim/topic/4050/objetos-de-texto-vim>

Capítulo 36: Obtener: ayuda (usando el manual incorporado de Vim)

Introducción

El manual incorporado de Vim es la fuente autorizada de información y documentación sobre cada característica de Vim, incluidas las configuraciones, las funciones integradas e incluso Vimscript. Si bien no es la interfaz más amigable para los principiantes, si sabe cómo examinarla, puede encontrar lo que necesita.

Comience a buscar ejecutando `:help :help [subject]`, o `:help :help`.

Sintaxis

- `:h[elp] [keyword]`

Parámetros

Parámetros	Detalles
keyword	Configuración, nombre de la función o cualquier otra palabra clave que tenga importancia para Vim. Palabras clave con dos puntos iniciales <code>:</code> búsqueda de comandos Vim; por ejemplo <code>:help :split</code> produce el comando de división de ventanas, y <code>:help split</code> produce la función de Vimscript <code>split()</code> .

Examples

Empezando / Navegando archivos de ayuda

Desde cualquier lugar en Vim, ejecuta `:help :help`. Esto abrirá una ventana dividida horizontalmente con la página de manual para el comando `:help . :help` por sí misma lo llevará a la Tabla de contenido del propio manual.

Los archivos de ayuda de Vim son navegables como archivos normales (puede buscar palabras clave dentro de un archivo como normal, con `/`), y además están vinculados por etiquetas. Salta al destino de una etiqueta con `CTRL-]`.

Las etiquetas son palabras rodeadas de tubo `|` caracteres. Las versiones 7.3 y versiones posteriores "ocultan" esos caracteres de tubería (`:help conceal`) y los resaltan.

Por ejemplo, la página Tabla de contenido muestra lo siguiente. Todas las palabras resaltadas en azul son etiquetas y están rodeadas por caracteres de canalización. Escribir `CTRL-]` con el cursor en `quickref` lo llevará a una página útil con una lista de etiquetas con características útiles de Vim.

```

doc-file-list Q_ct
BASIC:
quickref      Overview of the most common commands you will use
tutor         30 minutes training course for beginners
copying       About copyrights
iccf          Helping poor children in Uganda
sponsor       Sponsor Vim development, become a registered Vim user
www           Vim on the World Wide Web
bugs          Where to send bug reports

USER MANUAL: These files explain how to accomplish an editing task.

usr_toc.txt   Table Of Contents

```

Buscando el manual

`:help [subject]` intenta encontrar la "mejor" coincidencia para el argumento que proporciona. El argumento "puede incluir comodines como `* ? Y [az]` (cualquier letra).

También puede usar la terminación de línea de comandos de Vim con `CTRL+D` :

`:help spli<Ctrl-D>` mostrará una lista de temas de ayuda que coinciden con el patrón `spli` , incluyendo `split()` y `:split` .

Para buscar comandos basados en `Ctrl` , como `Ctrl-V` , escriba:

`:help ^v` con un carácter de letra literal, o incluso más específicamente,

`:help i_^V` para obtener ayuda en `Ctrl-V` en modo de inserción.

Como puede ver, vim tiene una nomenclatura para sus temas de ayuda. Por ejemplo, las opciones están citadas (consulte `:h 'sw'`), los comandos comienzan con dos puntos (consulte `:h :split`), las funciones terminan con corchetes vacíos (consulte `:h split()`), las asignaciones de modo de inserción comienzan con `i_` , comando las asignaciones de modo comienzan con `c_` , y así sucesivamente, excepto las asignaciones de modo normal que no tienen prefijo.

Término de búsqueda	Página de ayuda
<code>:help textwidth</code>	Configuración para longitud de línea / ancho de texto
<code>:help normal</code>	<code>:normal</code> comando <code>:normal</code> , para ejecutar comandos en modo normal desde la línea de comandos
<code>:help cursor</code>	Comando Vimscript para mover el cursor alrededor
<code>:help buffers</code>	Trabajando con buffers; igual que <code>:help windows</code>
<code>:help :buffer</code>	El comando <code>:buffer</code>
<code>:help :vs</code>	División vertical

Lea [Obtener: ayuda \(usando el manual incorporado de Vim\) en línea:](#)

<https://riptutorial.com/es/vim/topic/8837/obtener--ayuda--usando-el-manual-incorporado-de-vim->

Capítulo 37: Opciones de Vim

Sintaxis

- `:set [no] (option|shortcut)`
- `:set (option|shortcut)=value`
- `:set (option|shortcut) (?|&)`
- no usar `:` en el archivo `vimrc`

Observaciones

Ver [video de vimcast 1](#)

Ver [transcripción vimcast 1](#)

Examples

Conjunto

Para configurar las opciones - use `:set` instrucción. Ejemplo:

```
:set ts=4
:set shiftwidth=4
:set expandtab
:set autoindent
```

Para ver el valor actual de la opción, escriba `:set {option}?`. Ejemplo:

```
:set ts?
```

Para restablecer el valor de la opción a su valor predeterminado, escriba `:set {option}&`. Ejemplo:

```
:set ts&
```

Sangría

Anchura

Para hacer hendiduras de 4 espacios de ancho:

```
:set shiftwidth=4
```

Espacios

Para utilizar espacios como sangrías, 4 espacios de ancho:

```
:set expandtab  
:set softtabstop=4
```

`softtabstop` y `sts` son equivalentes:

```
:set sts=4
```

Pestañas

Para usar pestañas como sangrías, 4 espacios de ancho:

```
:set noexpandtab  
:set tabstop=4
```

`tabstop` y `ts` son equivalentes:

```
:set ts=4
```

Sangría automática

```
:set autoindent
```

Descripciones de instrucciones

Instrucción	Descripción	Defecto
tabulación	ancho del caracter de la pestaña	8
expandtab	hace que se usen espacios en lugar de caracteres de tabulación	apagado
softtabstop	afinar el espacio en blanco	0
ancho de turno	determina la cantidad de espacios en blanco cuando está en modo normal	8

Personajes invisibles

Mostrar u ocultar personajes invisibles

Para mostrar caracteres invisibles:

```
:set list
```

Para ocultar personajes invisibles:

```
:set nolist
```

Para alternar entre mostrar y ocultar caracteres invisibles:

```
:set list!
```

Caracteres de símbolo por defecto

Símbolo	Personaje
^ Yo	Lengüeta
PS	Nueva línea

Personalizar símbolos

Para configurar el carácter de tabulación en ****> **** y el nuevo carácter de línea en **↵**

```
set listchars=tab:>\ ,eol:↵
```

Para configurar los espacios a **_**

```
set listchars=spaces
```

Para ver una lista de opciones de personajes.

```
:help listchars
```

Lea Opciones de Vim en línea: <https://riptutorial.com/es/vim/topic/2407/opciones-de-vim>

Capítulo 38: Pida crear directorios no existentes al guardar un archivo nuevo

Introducción

Si edita un archivo nuevo: `vim these/directories/dont/exist/newfile`, usted no será capaz de guardar el archivo como el directorio de `vim` está tratando de salvar a que no existe.

Examples

Indique la creación de directorios con: `w`, o créelos lentamente con: `w!`

Este código le pedirá que cree el directorio con `:w`, o simplemente lo haga con `:w!`:

```
augroup vimrc-auto-mkdir
  autocmd!
  autocmd BufWritePre * call s:auto_mkdir(expand('<afile>:p:h'), v:cmdbang)
  function! s:auto_mkdir(dir, force)
    if !isdirectory(a:dir)
      \   && (a:force
      \     || input("'" . a:dir . "' does not exist. Create? [y/N]") =~? '^y\|[es]$')
      call mkdir(iconv(a:dir, &encoding, &termencoding), 'p')
    endif
  endfunction
augroup END
```

Lea [Pida crear directorios no existentes al guardar un archivo nuevo en línea](https://riptutorial.com/es/vim/topic/9470/pida-crear-directorios-no-existentes-al-guardar-un-archivo-nuevo):

<https://riptutorial.com/es/vim/topic/9470/pida-crear-directorios-no-existentes-al-guardar-un-archivo-nuevo>

Capítulo 39: Plegable

Observaciones

El *plegado* hace que varias líneas de texto se contraigan y se muestren como una sola línea. Es útil para ocultar partes de un documento que no se considera importante para la tarea actual. El plegado es simplemente un cambio visual en el documento: las líneas plegadas todavía están presentes, sin cambios.

Un pliegue es persistente. Una vez creado, un pliegue se puede abrir y cerrar sin necesidad de volver a crearlo. Cuando está cerrado, los pliegues se pueden mover o tirar y colocar como si fueran una sola línea, a pesar de que la operación subyacente operará en todo el texto debajo del pliegue.

Examples

Configurando el método de plegado

`:set foldmethod={method}` establece el método de plegado para la ventana actual. Esto determina cómo se manipulan los pliegues dentro de esa ventana. Las opciones válidas para "método" son:

- `manual` (los pliegues son creados y destruidos manualmente por el usuario)
- `indent` (los pliegues se crean para líneas de sangría iguales)
- `marker` (los `marker` subcadena se utilizan para indicar el principio y el final de un pliegue)
- `syntax` (los elementos resaltados de sintaxis definen los pliegues)
- `expr` (una expresión de Vimscript se evalúa por línea para definir su nivel de plegado)
- `diff` (el cambio de texto no se cambia en una vista de diferencia se pliega)

El valor predeterminado es `manual`.

Creación de un pliegue manual

- `zf{motion}` crea un pliegue que cubre el texto que cubriría "motion".
- `{count}zF` crea un pliegue que cubre las líneas "count".
- `{range}fo[ld]` crea un pliegue para las líneas en el rango provisto.

Los tres comandos son válidos solo cuando `foldmethod` se establece en `manual` o `marker`. En el caso del método de pliegue anterior, los pliegues recién creados se cierran inmediatamente.

Apertura, cierre y cambio de pliegues.

- `zo` abre un pliegue debajo del cursor.
- `zO` abre todos los pliegues debajo del cursor, recursivamente.
- `zc` cierra un pliegue debajo del cursor.
- `zC` cierra todos los pliegues debajo del cursor, recursivamente.

- `za` alterna un pliegue debajo del cursor (se abre un pliegue cerrado, se cierra un pliegue abierto).
- `zM` cierra todos los pliegues en el búfer.
- `zR` abre todos los pliegues en el búfer.
- `zm` cierra un nivel de plegado en el búfer.
- `zr` abre un nivel de plegado en el búfer.

Mostrando la línea que contiene el cursor

`zv` asegurará que la línea que contiene el cursor no esté doblada. Se abrirá el número mínimo de pliegues necesarios para exponer la línea del cursor.

Bloques plegables en C

Este es nuestro búfer:

```
void write_buffer(size_t size, char ** buffer)
{
    char * buf = *buffer;
    size_t iter;
    for(iter = 0; iter < size; iter++)
    {
        putchar(*(buf + iter));
    }
}
```

El cursor está en [1] [1] ([línea] [col]). Mueva el cursor al corchete del bucle for:

`/for<Enter>j` cursor es [6] [2].

Ahora ingrese `zf%` (cree plegado, muévase al corchete correspondiente). Has creado con éxito el primer plegado.

Ahora ingrese `:2<Enter>_`, el cursor está ahora en [2] [1] y `zf%`: el cuerpo de la función completa está plegado.

Puede abrir todos los plegados que acaba de crear utilizando `zo` y volver a cerrarlos utilizando `zC`.

Lea Plegable en línea: <https://riptutorial.com/es/vim/topic/3791/plegable>

Capítulo 40: Rangos de línea de comando

Examples

Números de línea absolutos

El siguiente comando ejecuta `:command` en las líneas 23 a 56 :

```
:23,56command
```

NB: Los rangos son *inclusivos* por defecto.

Números de línea relativos

En el siguiente comando, el rango comienza 6 líneas por encima de la línea actual y termina 3 líneas por debajo:

```
:-6,+3command
```

Atajos de línea

- `.` representa *la línea actual*, pero también se puede omitir por completo.
- `$` representa *la última línea* .
- `%` representa *el búfer completo* , es un acceso directo para `1,$` .

Los dos comandos a continuación ejecutan `:command` en cada archivo desde la línea actual hasta la última línea:

```
:.,$command  
:,$command
```

El siguiente comando ejecuta `:command` en todo el búfer:

```
:%command
```

Marcas

El siguiente comando ejecuta `:command` en cada línea desde la que contiene la marca manual `f` a la que contiene la marca manual `t` :

```
:'f','t'command
```

Las marcas automáticas se pueden utilizar también:

```
:'<','>'command " covers the visual selection
```

```
:'{,'}command    " covers the current paragraph
:'[,']command    " covers the last changed text
```

Ver :help mark-motions .

Buscar

Los siguientes comandos ejecutan :command en cada línea desde la primera coincidencia from la primera coincidencia to :

```
:/from/,/to/command    " from next 'from' to next 'to'
:?from?/,to/command    " from previous 'from' to next 'to'
:?from?,?to?command    " from previous 'from' to previous 'to'
```

Ver :help search-commands .

Compensaciones de línea

Las compensaciones de línea se pueden usar para ajustar las líneas de inicio y finalización:

```
:/foo/-,/bar/+4command    " from the line above next 'foo' to 4 lines below next 'bar'
```

Ver :help search-offset .

Rangos mixtos

Es posible combinar todo lo anterior en rangos expresivos:

```
:1267,/foo/-2command
:{,command
:'f,$command
```

Sea creativo y no olvide leer :help cmdline-ranges .

Lea Rangos de línea de comando en línea: <https://riptutorial.com/es/vim/topic/3383/rangos-de-linea-de-comando>

Capítulo 41: Recursos Vim

Observaciones

Este tema trata sobre **espejos de código fuente** , **libros** , **Vim-Wikis** . **NO se** trata de entradas de blog, Wikipedia, Tutoriales. Los recursos no deben estar basados en opiniones.

Examples

Aprendiendo Vimscript de la manera difícil

Un libro que explica cómo funciona Vimscript, lleno de ejemplos. Se puede encontrar en <http://learnvimscriptthehardway.stevelosh.com/>

Lea Recursos Vim en línea: <https://riptutorial.com/es/vim/topic/6383/recursos-vim>

Capítulo 42: Registros Vim

Parámetros

Funcionalidad	Registros
registro predeterminado	" "
registros de historia	" [1-9]
registro de tirón	" 0
registros nombrados	" [az] , " [AZ] igual que " [az] pero se adjunta
recordar el patrón de búsqueda actual	" /
pequeñas eliminaciones (diw, cit, ...)	" -
registros de expresiones para matemáticas simples	" =
registro de agujero negro para eliminar grandes trozos de texto eliminado de mem	" _
último comando	" :
último texto insertado	" .
nombre del archivo	" %
portapapeles	" *
texto seleccionado	" +
texto caído	" ~

Examples

Eliminar un rango de líneas en un registro con nombre

En Normal, escriba lo siguiente para eliminar un rango de líneas en un registro nombrado

```
:10,20d a
```

Esto borrará las líneas 10,20 en el registro "a". Podemos verificar esto escribiendo


```
:reg
```

Esto mostrará el texto que fue eliminado en el registro "a" .

Para pegar el contenido en "a" , simplemente escriba

```
"ap
```

Pegue el nombre del archivo en el modo de inserción usando el registro de nombre de archivo

En el modo Insertar, presione <Cr> y luego % para insertar el nombre del archivo.

Esta técnica es aplicable a todos los registros.

Por ejemplo, si en el modo de inserción, desea pegar el patrón de búsqueda actual, puede escribir <Cr> y luego / .

Copiar / pegar entre Vim y el portapapeles del sistema

Utilice el registro quotestar para copiar / pegar entre Vim y el portapapeles del sistema

"*yy copia la línea actual en el portapapeles del sistema

"*p pega el contenido del portapapeles del sistema en Vim

Anexar a un registro

Tirar todas las líneas que contienen TODO en un registro mediante la operación de adición

```
:global/TODO/yank A
```

Aquí, estamos buscando una palabra clave TODO globalmente, convirtiendo todas las líneas en el registro a (A registro agrega todas las líneas a a registro).

NOTA: En general, es una buena práctica borrar un registro antes de realizar la operación de adición.

Para borrar un registro, en el modo normal, escriba qaq . Confirmar que el a registro está vacío escribiendo :reg y observando que a registro está vacío.

Lea Registros Vim en línea: <https://riptutorial.com/es/vim/topic/4278/registros-vim>

Capítulo 43: Saliendo de Vim

Parámetros

Parámetro	Detalles
:	Entrar en modo de línea de comandos
w	Escribir
q	Dejar
a	Todos
!	Anular

Observaciones

El modo de línea de comando se ingresa a través del modo normal. Sabrá que está en modo de línea de comando cuando hay un `:` en la esquina inferior izquierda de la ventana de su terminal.

El modo normal es el modo predeterminado de `vi` / `vim` y se puede cambiar presionando la tecla `ESC`.

`Vi` / `Vim` tiene controles incorporados para evitar que se pierda el trabajo no guardado y otras características útiles. Para omitir estas comprobaciones, use la anulación `!` en su comando

En `Vi` / `Vim` es posible tener más de un archivo mostrado (en diferentes ventanas) al mismo tiempo. Usa `a` para cerrar todas las ventanas abiertas.

Examples

Salir con guardar

```
: wq
```

```
ZZ
```

Salir sin guardar

```
: q!
```

Salir con fuerza (sin guardar)

```
: q!
```

```
ZQ
```

Salir con fuerza (con guardar)

: wq!

Salir con fuerza de todas las ventanas abiertas (sin guardar)

: qa!

si se abren múltiples archivos

```
:wqall
```

Saliendo de múltiples archivos guardando contenidos.

```
:qall!
```

Saliendo de múltiples archivos sin guardar contenidos

Lea Saliendo de Vim en línea: <https://riptutorial.com/es/vim/topic/5074/saliendo-de-vim>

Capítulo 44: Sangría

Examples

Sangre un archivo completo utilizando el motor de sangría incorporado

En el modo de comando (Esc) ingrese `:gg=G` para usar el motor de sangría incorporado de Vim.

Parte de comando	Descripción
<code>gg</code>	inicio de archivo
<code>=</code>	sangría (cuando <code>equalprg</code> está vacío)
<code>sol</code>	fin del documento

Puede configurar `equalprg` en su `.vimrc` para usar una herramienta de formato automático más sofisticada.

Por ejemplo, para usar `clang-format` en su `.vimrc` para C / C ++, coloque la siguiente línea en su archivo `.vimrc`:

```
autocmd FileType c,cpp setlocal equalprg=clang-format
```

Para otros tipos de archivos, reemplace `c,cpp` con el tipo de archivo que desea formatear y con el formato de `clang-format` con su herramienta de formato preferida para ese tipo de archivo.

Por ejemplo:

```
" Use xmllint for indenting XML files. Commented out.
"autocmd FileType xml setlocal equalprg=xmllint\ --format\ --recover\ -\ 2>/dev/null
" Tidy gives more formatting options than xmllint
autocmd FileType xml setlocal equalprg=tidy\ --indent-spaces\ 4\ --indent-attributes\ yes\ --
sort-attributes\ alpha\ --drop-empty-paras\ no\ --vertical-space\ yes\ --wrap\ 80\ -i\ -xml\
2>/dev/null
```

Líneas sangradas o caducas

Para sangrar nuestra línea actual en [el modo normal](#), presione la tecla mayor que `>` o la menor que `<` dos veces en consecuencia. Para hacer lo mismo en varias líneas, simplemente agregue un número de antemano `6>>`

Mando	Descripción
<code>>></code>	sangrar línea actual
<code><<</code>	línea actual vencida

Mando	Descripción
6>>	sangría las siguientes 6 líneas

También puedes sangrar usando [movimientos](#) . Aquí hay algunos ejemplos útiles.

Mando	Descripción
>gg	sangría desde la línea actual hasta la primera línea en el archivo
>G	sangría desde la línea actual hasta la última línea en el archivo
>{	guión del párrafo anterior
>}	sangría el siguiente párrafo

En [modo visual](#), presionando la tecla mayor o menor que solo una vez. Tenga en cuenta que esto provoca una salida del [modo visual](#) . Entonces puedes usar `.` para repetir la edición si necesita y `u` para deshacer.

Lea [Sangría en línea](#): <https://riptutorial.com/es/vim/topic/6324/sangria>

Capítulo 45: Sustitución

Sintaxis

- `s/<pattern>/<pattern>/optional-flags`
- `<pattern>` es un Regex

Parámetros

Bandera	Sentido
Y	Guarda las banderas del sustituto anterior.
do	Aviso para confirmar cada sustitución.
mi	No reportar errores.
sol	Reemplace todas las ocurrencias en la línea.
yo	Emparejamiento insensible a mayúsculas.
yo	Coincidencia entre mayúsculas y minúsculas
norte	Reporte la cantidad de coincidencias, no sustituya realmente.

Observaciones

Utilice `set gdefault` para evitar tener que especificar el indicador 'g' en cada sustituto.

Ejemplo

Cuando se establece `gdefault`, ejecutar `:s/foo/bar` en la línea `foo baz foo` producirá `bar baz bar` lugar de `bar baz foo`.

Examples

Reemplazo simple

`:s/foo/bar` Reemplace la **primera** instancia de `foo` con `barra` en la línea actual.

`:s/foo/bar/g` Reemplaza cada instancia de `foo` con `barra` en la línea actual.

`:%s/foo/bar/g` Reemplace `foo` con `barra` en todo el archivo.

Refactoriza rápidamente la palabra debajo del cursor.

1. * en la palabra que quieras sustituir.
2. :%s//replacement/g , dejando el patrón de *búsqueda* vacío.

Reemplazo con aprobación interactiva

:s/foo/bar/c Marca la primera instancia de *foo* en la línea y solicita confirmación para la sustitución con *barra*

:%s/foo/bar/gc Marca consecutivamente cada coincidencia de *foo* en el archivo y solicita confirmación de sustitución con *barra*

Teclado de método abreviado para reemplazar la palabra resaltada

Por ejemplo, con el siguiente `nmap` :

```
nmap <expr> <S-F6> ':%s/' . @/ . '//gc<LEFT><LEFT><LEFT>'
```

seleccione una palabra con * , escriba `Shift - F6` , escriba un reemplazo y presione `Enter` para renombrar todas las ocurrencias de manera interactiva.

Lea **Sustitución en línea**: <https://riptutorial.com/es/vim/topic/3384/sustitucion>

Capítulo 46: Tampones

Examples

Gestionando buffers

Puedes usar buffers para trabajar con múltiples archivos. Cuando abres un archivo usando

```
:e path/to/file
```

se abre en un nuevo búfer (el comando significa editar el archivo). Nuevo búfer que contiene una copia temporal del archivo.

Puede ir al búfer anterior con `:bp[rev]` y al siguiente búfer con `:bn[ext]` .

Puede ir a un búfer en particular con `b{n}` para ir al enésimo búfer. `b2` va al segundo búfer.

Utilice `:ls` o `:buffers` para enumerar todos los buffers

Buffers ocultos

Alejarse de un búfer con cambios no guardados causará este error:

```
E37: No write since last change (add ! to override)
```

Puede deshabilitar esto agregando un `set hidden` a su archivo `.vimrc`. Con esta opción establecida, sus cambios persistirán en el búfer, pero no se guardarán en el disco.

Cambio de búfer usando parte del nombre de archivo

Para seleccionar fácilmente un búfer por nombre de archivo, puede utilizar:

```
:b [part_of_filename]<Tab><Tab><Tab>...<Enter>
```

La primera `pestaña` expandirá la palabra a un nombre de archivo completo, y las presiones subsiguientes de la `pestaña` recorrerán la lista de posibles coincidencias.

Cuando hay varias coincidencias disponibles, puede ver una lista de coincidencias *antes de* la expansión de la palabra configurando esta opción:

```
:set wildmode=longest:full:list,full
```

Esto le permite refinar su palabra si la lista de coincidencias es demasiado larga, pero requiere presionar una `tecla` adicional para realizar la expansión. Agregue la configuración a su `MYVIMRC` si desea mantenerla.

A algunas personas les gusta iniciar este proceso con un mapa de teclado que primero enumera los buffers:

```
:nnoremap <Leader>b :set nomore <Bar> :ls <Bar> :set more <CR>:b<Space>
```

Eso hace que sea fácil seleccionar un búfer por su número:

```
:b [buffer_num]
```

Cambie rápidamente al búfer anterior o a cualquier búfer por número

<C-^> cambiará hacia y desde el archivo editado anteriormente. En la mayoría de los teclados <C-^> está CTRL-6.

3<C-^> cambiará al número de búfer 3. Esto es muy rápido, pero solo si conoce el número de búfer.

Puede ver los números de búfer desde `:ls` o desde un complemento como [MiniBufExplorer](#) .

Lea Tampones en línea: <https://riptutorial.com/es/vim/topic/2317/tampones>

Capítulo 47: Usando ex desde la línea de comando

Examples

Sustitución desde la línea de comando.

Si desea usar vim de una manera similar a `sed`, puede usar el indicador `-c` para ejecutar un comando ex desde la línea de comandos. Este comando se ejecutará automáticamente antes de presentarte el archivo. Por ejemplo, para reemplazar `foo` con `bar`:

```
vim file.txt -c "s/foo/bar"
```

Esto abrirá el archivo con todas las instancias de `foo` reemplazadas con `bar`. Si desea realizar cambios en el archivo *sin* tener que guardarlos manualmente, puede ejecutar varios comandos ex y hacer que el último comando escriba y salga. Por ejemplo:

```
vim file.txt -c "s/foo/bar" -c "wq"
```

Nota IMPORTANTE:

No se puede ejecutar varios comandos ex separadas por una barra `|`. Por ejemplo

```
vim file.txt -c "s/foobar | wq"
```

No es correcto Sin embargo, se puede hacer si usas `ex`.

```
ex -c ":%s/this/that/g | wq" file.txt
```

Lea Usando ex desde la línea de comando en línea:

<https://riptutorial.com/es/vim/topic/6819/usando-ex-desde-la-linea-de-comando>

Capítulo 48: Usando Python para scripts Vim

Sintaxis

- `: [range] py [thon] {instrucción}`

Examples

Compruebe la versión de Python en Vim

Vim tiene su propio intérprete de Python incorporado. Por lo tanto, podría utilizar una versión diferente del intérprete predeterminado para el sistema operativo.

Para verificar con qué versión de Python Vim se compiló, escriba el siguiente comando:

```
:python import sys; print(sys.version)
```

Esto importa el módulo `sys` e imprime su propiedad de `version`, que contiene la versión del intérprete de Python utilizado actualmente.

Ejecute los comandos del modo normal de Vim a través de la declaración de Python

Para poder usar los comandos vim en Python, se debe importar el módulo `vim`.

```
:python import vim
```

Después de importar este módulo, el usuario tiene acceso a la función de `command`:

```
:python vim.command("normal iTxt to insert")
```

Este comando ejecutaría `i` en modo normal, luego escriba `Text to insert` y volver al modo normal.

Ejecutando código multilínea de Python

Cada declaración de Python en Vim debe tener un prefijo con el comando `:python`, para indicar a Vim que el siguiente comando no es Vimscript sino Python.

Para evitar escribir este comando en cada línea, al ejecutar el código de Python multilínea, es posible indicar a Vim que interprete el código entre dos expresiones de marcador como Python.

Para lograr esto, usa:

```
:python << {marker_name}  
a = "Hello World"
```

```
print(a)
{marker_name}
```

donde {marker_name} es la palabra que desea usar para designar el final del bloque de python.

P.ej:

```
:python << endpython
surname = "Doe"
forename = "Jane"
print("Hello, %s %s" % (forename, surname))
endpython
```

imprimiría:

```
Hello, Jane Doe
```

Lea Usando Python para scripts Vim en línea: <https://riptutorial.com/es/vim/topic/5604/usando-python-para-scripts-vim>

Capítulo 49: Ventajas de vim

Examples

Personalización

La ventaja de usar **vim** sobre un simple editor de texto como **notepad** o **gedit** es que le permite al usuario personalizar casi todo sobre sí mismo. Si alguna vez te encuentras realizando algún tipo de acción una y otra vez, vim tiene una multitud de funciones que te ayudarán a realizar esta acción de manera más rápida y fácil.

La mayoría de los IDE populares, como **MS Visual Studio** o **IntelliJ IDEA**, brindan a sus usuarios accesos directos útiles e incluso cierta cantidad de personalización, pero generalmente están relacionados con acciones específicas que son comunes en un contexto particular, mientras que vim permite personalizar para diferentes Situaciones, sin chocar entre sí. Puede ser cómodo desarrollar programas c ++ en Visual Studio y Java en IntelliJ, pero no escribiría el código de Python allí, y por supuesto hay otro IDE, pero en vim puede editar prácticamente cualquier idioma que desee sin perdiendo la conveniencia.

Por supuesto, hay otros editores personalizables, y no soy yo quien dice que vim es el mejor para todos. Esta es una cuestión de preferencia personal. No creo que alguien discuta que **emacs** permite un nivel de personalización inferior al de vim (y algunos dirían lo contrario), pero realmente tienes que probarlo por ti mismo, para encontrar lo que más te convenga.

Algunas personas dicen que no quieren pasar meses aprendiendo a usar un editor, solo para poder trabajar en él. Pero los que lo hacen, en su mayoría están de acuerdo, que valió la pena. Para mí, personalmente, nunca fue un problema, aprender cosas nuevas sobre vim y ser más eficiente es simplemente divertido. Y hay mucho que aprender.

Ligero

Vim es (como GNU Nano o GNU emacs) ligero. No necesita ningún tipo de interfaz gráfica (como x11, wayland & co).

Esto hace que vim sea el mejor amigo de un mantenedor del sistema. Puede usarlo usando ssh y, esto es realmente importante, en dispositivos realmente pequeños que no tienen algún tipo de interfaz gráfica.

La programación y el mantenimiento de servidores remotos se volvieron cada vez más importantes durante los últimos años y el uso de vim (o emacs) es la mejor manera de hacerlo.

A diferencia de muchos IDE, vim ofrece la capacidad de trabajar con muchos tipos de archivos de forma inmediata y escribir sus propios comandos y sintaxis hl es fácil.

Y por último, pero no menos importante, un usuario vim debería poder usar vi, que está preinstalado en la mayoría de los sistemas UNIX.

Lea Ventajas de vim en línea: <https://riptutorial.com/es/vim/topic/9653/ventajas-de-vim>

Capítulo 50: Ventanas divididas

Sintaxis

- `:split <file>`
- `:vsplit <file>`
- `:sp <-` taquigrafía para dividir
- `:vsp <-` taquigrafía para vsplit

Observaciones

Cuando se llama desde la línea de comandos, se pueden proporcionar varios archivos en el argumento y vim creará una división para cada archivo. Cuando se llama desde el modo `ex`, solo se puede abrir un archivo por invocación del comando.

Examples

Abriendo múltiples archivos en splits desde la línea de comando

Horizontalmente

```
vim -o file1.txt file2.txt
```

Verticalmente

```
vim -O file1.txt file2.txt
```

Opcionalmente, puede especificar el número de splits para abrir. El siguiente ejemplo abre dos divisiones horizontales y carga `file3.txt` en un búfer:

```
vim -o2 file1.txt file2.txt file3.txt
```

Abriendo una nueva ventana dividida

Puede abrir una nueva división en Vim con los siguientes comandos, en modo *normal* :

Horizontalmente:

```
:split <file name>  
:new
```

Verticalmente

```
:vsplit <file name>
```

```
:vnew
```

split abrirá el archivo en una nueva división en la parte superior o izquierda de la pantalla (o la división actual) `:sp` y `:vs` son accesos directos convenientes.

nuevo abrirá una división vacía

Cambiando el tamaño de una división o vsplit

Es posible que a veces desee cambiar el tamaño de una división o vsplit.

Para cambiar el tamaño de la división activa actualmente, use `:resize <new size>`. `:resize 30` por ejemplo haría que la división de 30 líneas de alto.

Para cambiar el tamaño de la vsplit actualmente activa, use `:vertical resize <new size>` **resize** `:vertical resize <new size>`. `:vertical resize 80` por ejemplo, el `:vertical resize 80` haría que el vsplit 80 caracteres de ancho.

Atajos

- `Ctrl + wy +` aumentan el tamaño de la ventana dividida
- `Ctrl + wy -` disminuir el tamaño de la ventana dividida
- `Ctrl + wy =` establecer un tamaño igual a las ventanas divididas

Cierra todas las splits pero la actual.

Modo normal

```
Ctrl-w o
```

Modo ex

```
:only
```

o corto

```
:on
```

Administrar ventanas divididas abiertas (atajos de teclado)

Después de abrir una ventana dividida en vim (como lo demuestran muchos ejemplos en esta etiqueta), es probable que desee controlar las ventanas rápidamente. Aquí es cómo controlar ventanas divididas usando atajos de teclado.

Mover para dividir arriba / abajo:

- `Ctrl + wy k`
- `Ctrl + wy j`

Mover para dividir izquierda / derecha:

- `Ctrl + w y h`
- `Ctrl + w y l`

Mover para dividir Arriba / Abajo (envolver):

- `Ctrl + w y w`

Crear nueva ventana vacía:

- `Ctrl + w y n -o--`: nuevo

Crear nueva división horizontal / vertical:

- `Ctrl + w , s` (mayúsculas)
- `Ctrl + w , v` (minúsculas)

Haz la división activa actual en la pantalla:

- `Ctrl + w y o -o--`: en

Moverse entre divisiones

Para moverse para dividir a la izquierda, use `<Cw><Ch>`

Para pasar a la división a continuación, use `<Cw><Cj>`

Para mover para dividir a la derecha, use `<Cw><Ck>`

Para pasar a la división anterior, use `<Cw><Cl>`

Sane split split

Es una mejor experiencia abrir split abajo y a la derecha

configurarlo usando

```
set splitbelow
set splitright
```

Lea Ventanas divididas en línea: <https://riptutorial.com/es/vim/topic/1705/ventanas-divididas>

Capítulo 51: vglobal: ejecuta comandos en líneas que no coinciden globalmente

Introducción

:vglobal o: v es el opuesto de: global o: g que opera en líneas que no coinciden con el patrón especificado (inverso).

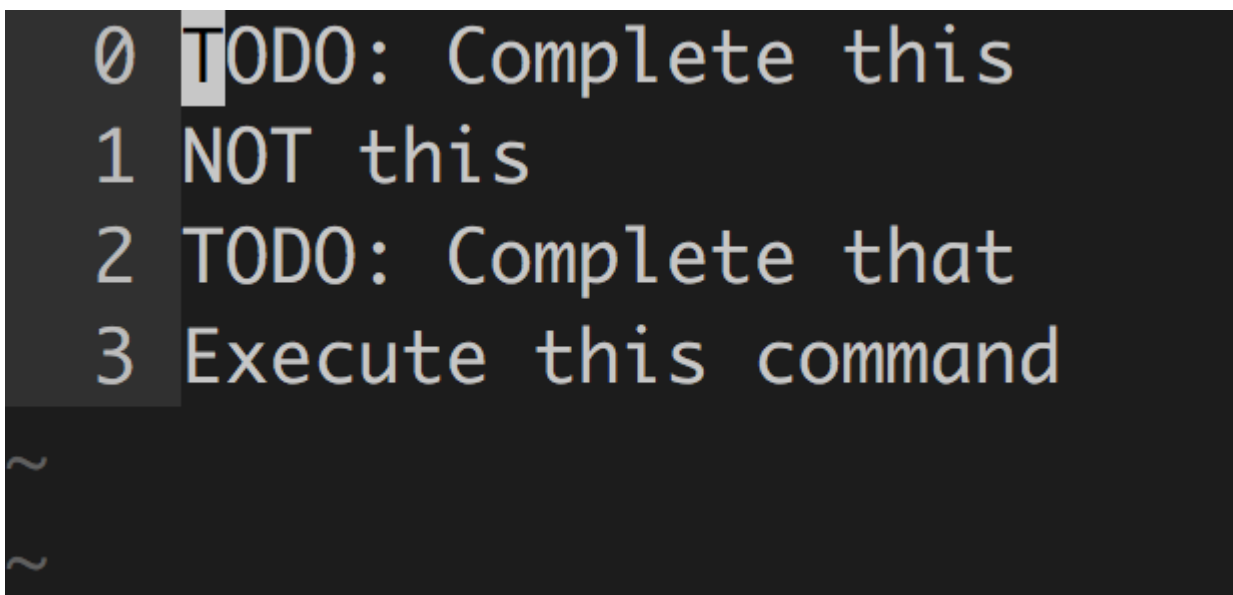
Examples

:v / patrón / d

Ejemplo:

```
> cat example.txt
TODO: complete this
NOT this
NOT that
TODO: Complete that
```

Abra el `example.txt` usando `vim` y escriba `:v/TODO/d` en el modo `Ex` . Esto eliminará todas las líneas que no contengan el patrón `TODO` .



```
0 TODO: Complete this
1 NOT this
2 TODO: Complete that
3 Execute this command
~
~
```

```
1 TODO: Complete this
0 TODO: Complete that
~
~
~
```

Lea vglobal: ejecuta comandos en líneas que no coinciden globalmente en línea:
<https://riptutorial.com/es/vim/topic/9867/vglobal--ejecuta-comandos-en-lineas-que-no-coinciden-globalmente>

Capítulo 52: Vim Solarizado

Introducción

Pasar la mayor parte del tiempo en la terminal puede ser un gran problema para los ojos. Elegir sabiamente el esquema de color puede beneficiar a sus ojos de muchas maneras. Recientemente me encontré con [Solarized ColorScheme for Vim](#) . Agregar este pequeño complemento puede hacer una gran diferencia en la apariencia del texto en el terminal. Muchas gracias a Ethan Schoonover por desarrollar este paquete. Los `howtos` se explican bastante bien [aquí](#) . ¡Disfrutar!

Examples

`.vimrc`

Solarized tiene dos opciones: modo claro y modo oscuro.

Modo de luz :

```
syntax enable
set background=light
colorscheme solarized
```

Modo oscuro :

```
syntax enable
set background=dark
colorscheme solarized
```

Lea Vim Solarizado en línea: <https://riptutorial.com/es/vim/topic/9500/vim-solarizado>

Capítulo 53: Vimscript

Observaciones

Los comandos en un archivo Vimscript se ejecutan en `command mode` por defecto. Por lo tanto, todas las directivas de `command mode no- command mode` deben ser prefijadas

Examples

Hola Mundo

Cuando se intenta imprimir algo para depurar en vimscript, es tentador simplemente hacer lo siguiente.

```
echo "Hello World!"
```

Sin embargo, en el contexto de un complemento complejo, a menudo suceden muchas otras cosas inmediatamente después de que intenta imprimir su mensaje, por lo que es importante agregar la `sleep` después de su mensaje para que pueda verlo antes de que desaparezca.

```
echo "Hello World!"  
sleep 5
```

Usando Comandos de Modo Normal en Vimscript

Dado que un archivo Vimscript es una colección de acciones en modo Comando, el usuario debe especificar que las acciones deseadas deben ejecutarse en modo normal.

Por lo tanto, la ejecución de un comando en modo normal como `i`, `a`, `d`, etc. en Vimscript se realiza anteponiendo el comando con `normal`:

Ir a la parte inferior del archivo y seleccionar las últimas 5 filas:

```
normal GV5k
```

Aquí, la `G` indica a vim que cambie la posición del cursor a la última fila, la `v` para ir al modo visual en línea y la `de 5k` para ir 5 filas hacia arriba.

Insertando su nombre al final de la fila:

```
normal ABoris
```

donde `A` coloca el editor en el modo de inserción al final de la fila y el resto es el texto para insertar.

Lea Vimscript en línea: <https://riptutorial.com/es/vim/topic/5136/vimscript>

Creditos

S. No	Capítulos	Contributors
1	Empezando con vim	A. Raza , akavel , Ashok , carrdelling , Christian Rondeau , Community , Cows quack , Daniel , Daniel Käfer , Daniel Margosian , Deborah V , depperm , ericdwang , ExistMe , GiftZwergrapper , gmoshkin , HerrSerker , James , Js Lim , KerDam , LittleByBlue , liuyang1 , LotoLo , Marek Skiba , Mattias , Miljen Mikic , mnoronha , Nasreddine , Nhan , Nick Weseman , pktangyue , redBit Device , Romain Vincent , romainl , ropata , Rory O'Kane , Sardathrion , sascha , SeekAndDestroy , sjas , sudo bangbang , Sumner Evans , tbodt , Tejus Prasad , TheMole , timss , Tom Gijselinck , Tom Lord , user2314737 , user45891 , Vin , Vishnu Kumar , vvnraman , Wieland , Wojciech Kazior , zarak , Zaz
2	:global	cmlaverdiere , DJMcMayhem , LittleByBlue , tbodt , Vin
3	Ahorro	abidibo
4	Asignaciones clave en Vim	Christian Rondeau , Ingo Karkat , KerDam , Luc Hermitte , madD7 , New Alexandria , Nikola Geneshki , RamenChef , romainl
5	Autocomandos	joeytwiddle , tbodt , Tom Hale
6	Buscando en el buffer actual	Abdelaziz Dabebi , DJMcMayhem , LittleByBlue , romainl
7	Código de auto-formato	Philip Kirkbride
8	Comandos de modo normal	Tom Hale
9	Comandos en modo normal (Edición)	A. Raza , rodericktech , romainl , The Nightman
10	Cómo compilar Vim	Ingo Karkat , Josh Petrie , romainl
11	Complementos	Nick Weseman
12	Complementos de tipo de archivo	Luc Hermitte , romainl
13	Configuraciones útiles que se pueden	Cows quack , maniacmic , Tomh

	poner en .vimrc	
14	Configurando Vim	Aaron Thoma , bn. , Christian Rondeau , Cometsong , Cows quack , Daniel , Johnathan Andersen , KerDam , Luc Hermitte , lwassink , mezzode , nobe4 , romainl , SnoringFrog , sudo bangbang , Sumner Evans , timss , Wojciech Kazior , Yosh
15	Consejos y trucos para aumentar la productividad.	Abdelaziz Dabebi , adelarsq , Chris Midgley , depperm , GanitK , gath , gmoshkin , Hotschke , KerDam , LittleByBlue , LotoLo , mash , naveen.panwar , RamenChef , rjmill , romainl , Simone Bronzini , soupono majumder , Stryker , sudo bangbang , tbodt , Tom Hale
16	Construyendo desde vim	grochmal , Josh Petrie , LittleByBlue , Luc Hermitte
17	Convertir archivos de texto de DOS a UNIX con vi	grochmal , LazyBrush
18	Corrector ortográfico	andipla , Johnathan Andersen , lwassink
19	De desplazamiento	Delapouite , evuez
20	Diferencias entre Neovim y Vim	still_dreaming_1 , tbodt
21	El operador de puntos	Js Lim , LittleByBlue
22	Encontrar y reemplazar	DJMcMayhem , Kara , naveen.panwar , ncmathsadist , romainl , sudo bangbang , zzz
23	Espacio en blanco	dallyingllama
24	Expresiones regulares	4444 , sudo bangbang
25	Expresiones regulares en modo Ex	UNagaswamy
26	Extendiendo Vim	baptistemm , LittleByBlue , Nikola Geneshki , romainl , satyanarayan rao , Sumner Evans , void
27	Huevos de Pascua	Aaron Thoma , Andrea Romagnoli , Christian Rondeau , cmlaverdiere , Daniel Käfer , Gerard Roche , LittleByBlue , Mateusz Piotrowski , Mattias , mcarton , nobe4 , NonlinearFruit , sudo bangbang , tbodt , Tejus Prasad
28	Insertar texto	Batsu , Boysenb3rry , Christopher Bottoms , cmlaverdiere , codefly

		, DJMcMayhem , Eric Bouchut , GiftZwergrapper , gmoshkin , Johnathan Andersen , Kent , lazysoundsystem , Mahmood , omul , Promarbler , RamenChef , rodrigo , romainl , satyanarayan rao , Scroff , SnoringFrog , sudo bangbang , Sundeeep , timss , Tom Lord , UNagaswamy , Xavier Nicollet
29	Macros	Johan , Johnathan Andersen , lazysoundsystem , LittleByBlue , Oliver Wespi , rjmill , romainl , TheMole , Victor Schröder , vielmetti , Wenzhong
30	Manipulando texto	Chris Nager , Christopher Bottoms , LittleByBlue , Nikola Geneshki , Philip Kirkbride , romainl , till , Tom Hale , zarak
31	Mejora deshacer y rehacer con un undodir	GiftZwergrapper
32	Mociones y objetos de texto	tbodt
33	Modos - insertar, normal, visual, ex	rgoliveira , romainl
34	Movimiento	Aidan Miles , akavel , Boysenb3rry , Caek , Chris H , Cows quack , depperm , fedorqui , Georgi Dimitrov , gmoshkin , JacobLeach , jamessan , KerDam , Madis Pukkonen , Noam Hacker , rgoliveira , sjas , SnoringFrog , Sundeeep , timss , Tyler , Vin , Wazam , zarak
35	Objetos de texto VIM	UNagaswamy
36	Obtener: ayuda (usando el manual incorporado de Vim)	Aidan Miles , Luc Hermitte
37	Opciones de Vim	dallyingllama , LittleByBlue , mezzode , Wojciech Kazior , Yosh
38	Pida crear directorios no existentes al guardar un archivo nuevo	Tom Hale
39	Plegable	Josh Petrie , LittleByBlue , sudo bangbang
40	Rangos de linea de comando	RamenChef , romainl
41	Recursos Vim	Nikola Geneshki
42	Registros Vim	maniacmic , romainl , UNagaswamy

43	Saliendo de Vim	Arulpandiyan Vadivel , asclepix , AWippler , Nick Weseman , Yosef Nasr
44	Sangría	dallyingllama , Daniel , RamenChef , toto21
45	Sustitución	cmlaverdiere , LittleByBlue , Nikola Geneshki , Timur
46	Tampones	Chris Jones , eli , joeytwiddle , sudo bangbang
47	Usando ex desde la línea de comando	DJMcMayhem , Matt Clark
48	Usando Python para scripts Vim	Nikola Geneshki
49	Ventajas de vim	gmoshkin , LittleByBlue
50	Ventanas divididas	beardc , Boysenb3rry , Downgoat , Goluxas , grenangen , HerrSerker , Johnathan Andersen , KerDam , madD7 , Sachin Divekar , sudo bangbang , timss , Victor Schröder , zarak
51	vglobal: ejecuta comandos en líneas que no coinciden globalmente	UNagaswamy
52	Vim Solarizado	Luc Hermitte , Nick Weseman , satyanarayan rao
53	Vimscript	merlin2011 , Nikola Geneshki