Una introducción rápida a GNU Emacs

Joaquín Ataz López*

Índice

Sobre el presente documento

Inicialmente, cuando me propuse escribir el documento «*Creación de ficheros LaTeX con GNU Emacs*» (que hoy día se puede descargar de ftp://ftp.dante.de/tex-archive/info/spanish/guia-atx), me planteé la posibilidad de dividirlo en dos partes. La primera contendría una introducción rápida a Emacs y la segunda se referiría al

uso de sus funciones avanzadas para la creación de ficheros LaTeX. Sin embargo, conforme fui redactando dicho documento me di cuenta de que, de un lado, era demasiado largo y convenía aligerarlo y, sobre todo, había una evidente descompensación entre ambas partes: la primera era muy superficial y la segunda muy detallada.

En consecuencia decidí suprimir la primera parte, ya que, en mi opinión, un buen documento debe tener un «*lector tipo*» y me era muy difícil imaginar al mismo *lector tipo* en ambas partes del documento. En la primera parte se resumía una información mucho más amplia, y en la segunda se profundizaba sobre una información que no suele estar disponible.

En las páginas que siguen se reproduce la "primera parte" de aquel documento, que fue eliminada de la versión definitiva. En ella se contiene una introducción a Emacs en la que se explica el uso general de los comandos de Emacs, las distintas partes de su pantalla, el significado de sus «*modos*» mayores y menores, así como las operaciones básicas de edición y una introducción a la sintaxis de las expresiones regulares.

Al respecto téngase en cuenta que se trata de reproducir el documento "tal y como estaba". Tan solo he escrito esta introducción explicativa, y he ajustado las referencias internas que pudiera haber a otras partes del documento que ya no están en él. Pero no lo he reescrito, por lo que se sigue presuponiendo en él que habrá una segunda parte centrada en la creación de documentos LaTeX y un apéndice dedicado a la personalización de Emacs.

Asimismo, como no terminé de escribirlo, no he llegado tampoco a depurarlo. Y por "depuración" entiendo realizar una comprobación de que lo que aquí digo funciona siempre y no sólo en mi concreta distribución. Al respecto debe tenerse en cuenta que muchas distribuciones de Linux introducen cambios en ciertos paquetes y que, además, en el caso concreto de Emacs, suele haber ciertas diferencias entre su versión en modo gráfico y su versión en modo texto. También hay ciertas diferencias entre Emacs propiamente dicho y Xemacs (una versión gráfica de Emacs). Pues bien: lo que aquí se dice está comprobado exclusivamente para la versión 21.3 de Emacs funcionando en modo gráfico e instalado de manera estándar con la versión 9.0 de Linux SuSE.

No me he molestado en reescribir el documento, porque mi proyecto para el futuro, es el de escribir un auténtico manual de Emacs que sea completo. Eso lo haré cuando tenga tiempo. De momento, no obstante, ofrezco este documento al público, tal y como está, por si a alguien le fuera útil. Por lo tanto ruego que no se me hagan observaciones sobre su posible «mejora», ya que no es mi intención mejorarlo en el futuro, sino más bien jubilarlo.

Cuestión distinta es que se detecte alguna afirmación incorrecta. En tal caso si agradeceré que se me comunique a jal@um.es.

1. Panorámica general de Emacs

GNU Emacs posiblemente sea el *editor de textos* más potente que exista para sistemas Unix-Linux, lo cual es tanto como decir que se trata del editor de textos más potente que existe en términos absolutos.

Las características que le hacen único son las siguientes:

Reconocimiento de formatos: Es lo primero que llama la atención de Emacs, y lo que hace que una vez que nos hemos acostumbrado a él se convierta en una herramienta insustituible.

Por reconocimiento de formatos hay que entender la posibilidad de detectar que cierto fichero sigue determinadas convenciones de sintaxis (correspondientes normalmente a un lenguaje de programación o a un lenguaje de marcas) de tal modo que, una vez reconocido el formato al que se ajusta el texto del documento Emacs pueda proporcionar ciertos mandatos especialmente útiles para ese tipo de documentos, así como resaltar mediante procedimientos gráficos la sintaxis del documento, distinguiendo entre las *instrucciones* y los *datos*, e incluso diferenciando entre distintas categorías de instrucciones.

Entre los formatos que Emacs es capaz de reconocer se encuentran numerosos lenguajes de programación y —para lo que nos interesa en este documento— los formatos TEX y LATEX.

Por otra parte, es cierto que hoy día muchos otros editores han incorporado esta habilidad. Pero Emacs —que fue de los primeros en implementarla— sigue siendo capaz de reconocer más formatos que la mayoría, y su diseño, basado en "modos mayores de edición" se ajusta de manera más natural al manejo de diferentes tipos de ficheros.

Facilidad de configuración y personalización: En Emacs casi todo es *personalizable*. Podemos crear mandatos nuevos, asignar combinaciones de teclas diferentes a las previstas por defecto, alterar ciertas variables de las que depende el funcionamiento del propio Emacs, etc. De hecho el propio nombre *Emacs* procede de una contracción de la expresión **E**ditor de **Mac**ro**s**[2], porque fue concebido como eso: como un editor de macros; y recordemos que el término *macro* suele significar

[2] Aunque los detractores de Emacs —que también los hay— suelen hacer numerosos juegos de palabras en los que el nombre de Emacs asume otros significados tales como, por ejemplo, ***E**macs **M**akes **A**ll **C**omputers **S**lower* (= Emacs convierte a cualquier ordenador en más lento), o ***E**macs **M**anuals **A**re **C**riptic and **S**ubreal* (= Los manuales de Emacs son crípticos y subrealistas).

La oposición a Emacs se debe al hecho de que esta aplicación en cierto modo es contraria a la filosofía dominante en el mundo Unix que se basa en aplicaciones pequeñas y compactas que hagan una sola cosa pero lo hagan bien. Y Emacs es una contradicción a dicha proclama, porque no es un

(aunque no siempre) mandato definido por el usuario de una determinada aplicación dirigido a personalizarla.

Extensibilidad: La facilidad para configurar Emacs queda ampliamente superada por su extensibilidad.

Que Emacs es extensible significa que cualquiera que sepa Emacs Lisp (un dialecto de Lisp en el que está escrita la mayor parte del propio Emacs) puede escribir nuevos mandatos de Emacs en dicho lenguaje de programación e incorporarlos al sistema, incluso sobre la marcha, sin necesidad de reinstalar o reiniciar el propio Emacs.

Esta facilidad para extender y ampliar el sistema se traduce en que existen numerosos *paquetes* de ampliación de Emacs que le permiten reconocer nuevos formatos, o le dotan de mandatos específicos para ciertos formatos o para ciertos usos, haciendo que, en definitiva, Emacs sirva para casi todo. Y así hay paquetes para convertir a Emacs en un lector de correo electrónico, en un lector de noticias, en un entorno integrado de desarrollo, ¡e incluso en un calendario!

***

Emacs, sin embargo, no es un programa sencillo. Por un lado su extremada potencia hace que sean muchísimas las funciones o mandatos que hay que aprender. Pero, por otra parte, Emacs es muy peculiar y su terminología y conceptos implicados no siempre se asemejan a los de otras aplicaciones de finalidad similar.

Esta última observación ha constituido para mí un verdadero mar de dudas a la hora de redactar este texto ya que no sabía si, al referirme a Emacs, utilizar su terminología específica o utilizar la terminología estándar que puede ser más familiar para el lector y ayudarle a entender mejor de qué se trata. Al final he optado por un criterio intermedio: siempre explico la terminología propia de Emacs. Pero cuando esta difiere de la más extendida, y la diferencia es *meramente terminológica* (no conceptual) a la tarea en cuestión me refiero, indistintamente, con la terminología normal o con la de Emacs con lo que creo que facilitaré que el lector se vaya familiarizando poco a poco con ella. Por el contrario, si la diferencia terminológica encierra una diferencia conceptual, tras explicarla, procuraré usar sólo la terminología de Emacs.

---

simple editor de textos, sino mucho más: una aplicación que hace muchas cosas (y si las hace bien o mal es opinable)

2. Los mandatos de Emacs

Emacs funciona mediante la ejecución de mandatos: abrir o cerrar un fichero, insertar texto en él o borrarlo, desplazar el cursor por el texto, realizar una búsqueda... todo se hace mediante mandatos. En teoría incluso escribir una letra es fruto de un mandato[3].

Desde el punto de vista interno los mandatos de Emacs son funciones —normalmente escritas en el lenguaje Elisp, que es como suele abreviarse el nombre de Emacs-Lisp— pero desde el punto de vista del usuario son *mandatos*, es decir: acciones que se realizan cuando pulsamos determinada combinación de teclas, o cuando seleccionamos una opción del menú.

2.1. Formas de invocar a los mandatos

Hay básicamente tres maneras de ejecutar un mandato, aunque sólo la primera de ellas funciona siempre y para cualquier mandato:

1. Invocar al mandato por su nombre, indicándole a Emacs que queremos ejecutar un mandato de esa manera.
2. Pulsar en el teclado la combinación de teclas a la que cierto mandato esté asociada.
3. Seleccionar el mandato del menú o de la barra de herramientas de Emacs.

2.1.1. Invocación de mandatos por su nombre:

El primero de los procedimientos es el único que funciona siempre, pues no todos los mandatos tienen una combinación de teclas asociada a ellos, ni constituyen una opción de la barra de menús o de la barra de herramientas.

Para ejecutar un mandato invocándolo por su nombre, lo primero que hay que hacer es decirle a Emacs que queremos introducir el nombre de un mandato para ejecutarlo. Eso lo podemos hacer de dos maneras:

- Pulsando simultáneamente la combinación de teclas Alt y “x”.
- Pulsando primero la tecla ESC y luego, tras soltar la tecla ESC, pulsando la tecla “x”.

[3] Aunque casi nunca se piensa en ello de esa manera. Pero que se trata de un mandato es claro, y se comprueba, por ejemplo, con el hecho de que a la pulsación de una tecla podemos aplicarle un *prefijo numérico* del mismo modo que a cualquier otro mandato (véase la sección 2.4, página 11) y así, por ejemplo, la secuencia de teclado «`C-u 5 g`» escribirá cinco veces la letra “g”, o lo que es lo mismo: ejecutará cinco veces el mandato consistente en escribir la letra “g”

En Emacs ambas acciones se consideran la misma: en ambos casos habremos pulsado la tecla a la que Emacs llama *meta* (en seguida se verá qué significa eso). Tras ello se activará una parte de la interfaz de Emacs llamada "*minibuffer*" y el cursor saltará automáticamente allí para que podamos escribir el nombre del mandato a ejecutar. A la hora de hacerlo recuérdese que Emacs distingue entre mayúsculas y minúsculas. Si en un momento dado tenemos dudas sobre cómo se llama la función deseada, pulsando la tecla SPC obtendremos una lista de todos los mandatos de Emacs cuyo nombre comienza igual que la parte que hemos tecleado; función esta última a la que se denomina "*autocompletado*". Tras terminar de escribir el nombre del mandato y pulsar la tecla RET, éste se ejecutará.

2.1.2. Ejecución de mandatos mediante una combinación o secuencia de teclado:

Pero pocas veces se ejecutan los mandatos de esa manera. Lo más normal para ejecutar un mandato en Emacs es pulsar la combinación de teclas a la que esté asociado. Aunque, claro está, no todos los mandatos están asociados a alguna combinación de teclas: sólo lo están los más frecuentemente utilizados.

Atendiendo al número de teclas necesarias para lanzar un mandato, éstos se clasifican en las siguientes categorías:

- Mandatos asociados a *una sola tecla*. Son muy pocos. Básicamente los mandatos de movimiento del cursor, que están —como es lógico— asociados a las teclas de movimiento del cursor[4].
- Mandatos asociados a la pulsación simultánea de dos o tres teclas. A estos mandatos se les llama *mandatos simples*, porque en ellos hay un solo golpe de teclado. Normalmente exigen que se pulsen simultáneamente dos teclas y sólo excepcionalmente implican la pulsación simultánea de tres teclas. De las teclas a pulsar una de ellas (a veces dos de ellas) debe necesariamente ser la tecla Control o la tecla Meta. A estas dos teclas se las suele denominar «teclas de cambio» porque su efecto consiste en alterar de algún modo el significado del resto de las teclas y su uso indica a Emacs que queremos introducir un mandato, y no escribir un carácter en el texto.
- Mandatos asociados a varias pulsaciones sucesivas, cada una de las cuales puede implicar simultáneamente a dos o tres teclas. Es decir: si los mandatos *simples* implican un solo *golpe de teclado*, este otro grupo de mandatos, a los que podríamos llamar *complejos*, suponen *dos o más golpes de teclado*.

[4] En este grupo habría también que incluir a los mandatos para insertar letras, dígitos y símbolos, lo que ocurre es que, como antes señalé, aunque insertar una letra en el *buffer* sea en sentido estricto un mandato, casi nunca se piensa en ello de esa manera.

En estos mandatos, se llama *prefijo* a la secuencia de teclado inicial, es decir: al conjunto de pulsaciones que no llegan a constituir el mandato completo. La utilidad de los prefijos estriba en que normalmente los mandatos que hacen tareas parecidas o relacionadas entre sí comparten el mismo *prefijo*, lo que hace que las distintas secuencias de teclado sean más sencillas de recordar, y así, por ejemplo, todas las secuencias de teclado referidas a funciones de ayuda empiezan por la pulsación simultánea de las teclas Control y "h", lo que se suele transcribir como «`C-h`».

Pero a todo esto: ¿Qué es la tecla Meta?

Vengo diciendo que las secuencias de teclado implican siempre a la tecla Control o a la tecla Meta. Y todo el mundo sabe qué es la tecla Control y dónde está. Pero en la mayoría de los teclados no hay ninguna tecla que se llame *meta*. Y es que la tecla meta, en realidad, más que una tecla física es una especie de concepto o *tecla virtual*.

Esto es así porque, de un lado, existen y han existido —y sin duda existirán— numerosos tipos de teclados y ello no tanto por la existencia de teclados para los distintos idiomas sino, fundamentalmente, atendiendo a la *arquitectura* del ordenador y al sistema operativo utilizado. Y así, por ejemplo, el teclado de los ordenadores Apple es distinto del de los Pcs, y en estos últimos la famosa tecla "win" no apareció hasta el lanzamiento del sistema operativo Windows 95.

Pues bien: Emacs fue diseñado para funcionar en cualquier ordenador, con cualquier teclado. Por ello sus autores decidieron que necesitaban *dos teclas de cambio*, es decir: dos teclas que se pudieran pulsar simultáneamente con otras alterando su significado. A estas dos teclas las llamaron, respectivamente, Control y Meta, y es la implementación o compilación de Emacs para cada sistema operativo y arquitectura de hardware la encargada de ocuparse de que haya una tecla que funcione como *tecla control* y otra que funcione como *tecla meta*.

En la práctica la *tecla control* se asigna a la tecla del mismo nombre que suele haber en casi todos los teclados, y la tecla Meta se asigna —en los teclados españoles ordinarios— a dos teclas físicas: La tecla Alt-izquierda y la tecla Escape. Una y otra funcionan de modo diferente:

- Para que la tecla Alt-izquierda funcione como tecla Meta debe ser pulsada *simultáneamente* con otra tecla, de modo similar a como funciona la tecla Control.
- Para que la tecla Escape funcione como tecla Meta debe ser pulsada *antes* que la tecla con la que se vaya a combinar.

Como ejemplo de lo dicho recordemos lo que antes señalé sobre la invocación de mandatos por su nombre (página 7): dije que había que pulsar M-x (Meta-x) para indicar a Emacs que se iba a introducir un mandato, y después añadí que ese efecto lo podíamos conseguir de dos maneras: pulsando simultáneamente Alt y "x", o pulsando primero ESC y luego "x".

> Pudiera pensarse que sería más fácil dejarse de zarandajas y, al menos en manuales como este, dirigidos a absolutos novatos, decir Alx-x en lugar de M-x. Sin embargo creo que es útil seguir hablando de la tecla Meta ya que:
>
> - Eso nos recuerda que las combinaciones de teclado en las que está presente la tecla Meta pueden alcanzarse por dos vías distintas (con la tecla Alt o con la tecla ESC).
> - Hay ocasiones en las que la combinación *sólo puede obtenerse* con la tecla ESC. Concretamente la combinación M-TAB cuando Emacs se ejecuta en modo gráfico[5].
> - Pero, sobre todo, en secuencias largas de teclado, Emacs nos informa en el área de eco de la parte que llevamos tecleada, y para ello a la tecla Meta la abrevia como M. Si en la documentación nos refiriéramos a ella habitualmente como tecla Alt, al usuario le costaría más aprovecharse de la ayuda que Emacs ofrece ya que no estaría tan acostumbrado a identificar la tecla Meta con la tecla Alt.
> - Razones todas estas a las que habría que añadir la circunstancia —casi nunca mencionada— de que *existen dos teclas llamadas Alt* Una situada a la izquierda de la barra espaciadora y otra a la derecha, y —a diferencia de la tecla control, de la que también hay dos, pero ambas son equivalentes— las dos teclas Alt son distintas: una (Alt-derecha) se usa para generar caracteres poco corrientes y otra (Alt-izquierda) se utiliza como *metatecla*. Con lo que resulta que hablar de *tecla Alt* es muy poco preciso.
> - Aparte de que en sistemas Linux podemos personalizar las teclas de cambio mediante la utilidad "xmodmap", lo que deja aun más claro que la tecla Meta es más un concepto que una tecla real.

2.2. Forma de transcribir las secuencias de teclado

Dado que Emacs informa (en el área de eco) sobre la secuencia de teclado que se está introduciendo, en la documentación de Emacs siempre se acoje la misma convención para transcribir las secuencias de teclado. Esta convención es la siguiente:

[5] Esto es porque el modo gráfico implica muchos programas cada uno en su propia *ventana gráfica* y en él la función de cambiar de aplicación se asocia a la combinación de teclas Alt-TAB. Pero como esa asociación la hace el sistema operativo, es prioritaria respecto a las asociaciones hechas por Emacs, de modo que cuando el sistema operativo detecta esa pulsación, cree que se le quiere dar una orden a él, por lo tanto la ejecuta sin comunicarle nada a Emacs, que no llega a enterarse de que se ha producido tal pulsación.

- La tecla Control se abrevia en C.
- La tecla Meta se abrevia en M.
- Las teclas que representan letras se escriben siempre en minúsculas o en mayúsculas según cómo deban pulsarse (recordemos que Emacs distingue entre mayúsculas y minúsculas), aunque la mayoría de las veces el mandato funcionará pulsándola de cualquiera de las dos formas, en cuyo caso la letra se transcribirá siempre en minúsculas.
- Las teclas que imprimen símbolos o dígitos se representan por el carácter que cada una de ellas imprime. Si para obtenerlo hay que pulsar alguna otra tecla (mayúsculas o Alt-derecha) ello no se indica en la transcripción, sobre todo porque depende del teclado que se esté utilizando. Y así por ejemplo la combinación «`C-;`» exige, en un teclado español, que se pulsen simultáneamente TRES teclas (control, mayúsculas y punto y coma) pero la tecla mayúsculas no se menciona explícitamente porque es obvio que no se puede generar el carácter ";" sin pulsarla[6].
- Las teclas que no representan caracteres imprimibles o visibles se representan siempre por su nombre o por una abreviatura de su nombre. Así se usa ESC para la tecla Escape (cuando no se transcribe como tecla Meta), TAB para el tabulador, SPC para la barra espaciadora, y RET para la tecla RETURN, INTRO o ENTER (que de las tres maneras es conocida), RETRO para la tecla de RETROCESO, etc.
- Un guión uniendo dos o más letras indica que deben ser pulsadas *simultáneamente*[7].
- Un espacio en blanco dentro de una secuencia de teclas significa que antes de pulsar la próxima tecla hay que soltar la tecla anterior. Así, por ejemplo, «`C-x u`» significa que primero hay que pulsar «`C-x`» y luego, tras soltar ambas teclas, la tecla "u". No obstante, en ciertas secuencias complejas como «`C-x C-f`» la tecla Control la podemos —si queremos— mantener pulsada todo el rato.

[6] Esta consideración matiza lo que antes dije respecto de que las combinaciones de teclas en Emacs son siempre de *dos o tres teclas*, Control, Meta y una tercera tecla; porque la obtención de esa tercera tecla puede implicar, a su vez, la pulsación de dos teclas. De modo que hay combinaciones de teclado que, en un teclado español, pueden requerir la habilidad de un pianista por exigir hasta *cuatro* teclas distintas pulsadas simultáneamente. Por ejemplo: Control + Alt-izquierda, + Alt-derecha + "@" (pues en un teclado español nose puede generar el carácter "@" sin pulsar también Alt-derecha). En estos casos puede ser una buena idea usar, como tecla Meta, la tecla ESC en lugar de Alt, ya que así sustituiríamos la pulsación simultánea de *cuatro* teclas, por dos pulsaciones sucesivas: la primera de una tecla (ESC) y la segunda de tres teclas (que es más fácil).

[7] Cuando una de esas teclas es la tecla Meta, ya sabemos que puede usarse ESC *antes* de la otra tecla. Si la combinación de teclas incluye el uso simultáneo de Meta, Control y una tercera tecla, podemos pulsar primero ESC y luego *simultáneamente*, Control y la tercera tecla que fuera

2.3. Uso, en este manual, de las secuencias de teclas y de los nombres de mandatos

Las secuencias de teclas son más fáciles de recordar que los nombres de mandatos[8], y desde luego más rápidas de teclear. Por ello es corriente que en la documentación sobre Emacs se haga referencia a los mandatos no por su verdadero nombre, sino por la combinación de teclas a que están asociados.

Sin embargo, el nombre de los mandatos, aunque no siempre haya que recordarlo, es importante; sobre todo para la personalización de Emacs, en la que a los mandatos hay que denominarlos por su nombre.

Esta es la razón de que en este manual aunque muchas veces me refiera a los mandatos por sus teclas asociadas, siempre se recoja el nombre real del mandato. Al menos así se hace la primera vez que el mandato es mencionado.

2.4. Prefijos numéricos para los mandatos

Hay mandatos de Emacs que *necesitan* algún argumento el cual normalmente se introduce en el minibuffer. Pero casi todos los mandatos de Emacs *aceptan* un argumento numérico que se puede introducir en la secuencia de teclado que provoca la ejecución del mandato. Estos argumentos numéricos normalmente se interpretan como un *factor de repetición*, es decir: provocan que el mandato se repita cierto número de veces; pero a veces tienen otro significado.

A este tipo de argumentos se les denomina *argumento prefijo*, porque cuando el mandato se ejecuta mediante una secuencia de teclado, deben introducirse *antes* que el mandato propiamente dicho.

Para introducir el argumento numérico hay dos procedimientos:

- Pulsar «C-u» seguido del número en que consista el argumento, y a continuación, la combinación de teclas asociada al mandato que se desea ejecutar.
- Pulsar simultáneamente la tecla Meta y el número que constituya el argumento y, a continuación, la combinación de teclas asociada al mandato que se desea ejecutar.

Así, por ejemplo, las siguientes tres secuencias de teclado son equivalentes:

```
C-u 3 C-p
ESC 3 C-p
Alt-3 C-p
```

[8] Sobre todo si se sabe inglés, porque en la mayoría de los casos se elige, como secuencia de teclado, la tecla correspondiente a la inicial del nombre del mandato a ejecutar; nombre este que casi siempre está en inglés

En los tres casos estamos indicando al mandato «C-p» que se ejecute con el argumento numérico 3.

Por otra parte se dice que existe un *argumento numérico vacío* cuando se pulsa «C-u» seguido inmediatamente del mandato que sea, pero sin introducir ningún *número* concreto. En estos casos Emacs envía como argumento numérico por defecto el número 4 (¡Vaya usted a saber por qué!).

También es posible *anidar* los argumentos numéricos. Así, por ejemplo, si introducimos el mandato «C-u C-u C-n», Emacs interpretaría lo siguiente:

- En primer lugar hay un argumento numérico vacío, lo cual significa que el resto del mandato se debe ejecutar cuatro veces.
- El resto del mandato empieza por un segundo argumento numérico vacío, que ordena la repetición, cuatro veces, del resto. O sea: el resto del mandato se ejecutará un total de 16 veces (cuatro por cuatro).
- El resto del mandato es la orden C-n que produce un desplazamiento del cursor a la línea inferior. La consecuencia es que el cursor se desplazará 16 líneas hacia abajo.

2.5. Cancelar y deshacer mandatos

Aunque acabamos de iniciar el aprendizaje de Emacs, hay dos mandatos que conviene que sean los primeros en ser aprendidos: el mandato de cancelación y el mandato para deshacer la última acción.

Cancelar un mandato: Cuando empezamos a introducir un mandato —cosa que se detecta por el uso de las teclas Control o Meta— Emacs espera hasta que hayamos escrito un mandato existente. Mientras no lo hayamos hecho considera que cualquier pulsación de teclas que hagamos forma parte del mandato que se está introduciendo. Pero puede ocurrir que hayamos cambiado de idea, o que nos hayamos equivocado en la secuencia de teclas y queramos *anular* la introducción del mandato. En estos casos lo mejor que podemos hacer, para recuperar el control, es pulsar «C-g» que ejecuta un mandato cuya utilidad está en cancelar el mandato que estuviera ejecutándose o escribiéndose.

El mandato ejecutado por «C-g» se denomina «keyboard-quit» y su utilidad es más general de lo que se acaba de decir: Siempre que nos encontremos en el minibuffer, pulsando «C-g» cancelamos la tarea que se esté ejecutando y volvemos a la anterior o, si estábamos en el primer nivel de ejecución, a la ventana de edición. Por eso a veces, en ediciones recursivas, para recuperar totalmente el control es preciso pulsar «C-g» varias veces.

Deshacer la acción de un mandato: Si tras ejecutar un mandato cambiamos de idea, podemos deshacer su acción mediante el mandato «`C-x u`» («`undo`», también asociado a «`C-_`»). Ejecutando sucesivas veces este mandato iremos deshaciendo las últimas acciones en el orden inverso a aquel en el que se ejecutaron; y para deshacer exclusivamente acciones que afecten a *la región*, hay que pulsar «`C-u C-x u`» (sobre la región véase la sección 5.3, página 5.3).

3. Elementos de la pantalla de Emacs

En un sistema Unix-Linux, Emacs puede ejecutarse en modo de sólo texto o en modo gráfico, según esté o no activo el sistema X-Window. Dependiendo de la manera en que Emacs se ejecute su aspecto será muy diferente. Sin embargo las partes de la pantalla de Emacs siempre serán las mismas y, en el fondo, la diferencia fundamental entre ejecutar o no Emacs en modo gráfico se encuentra en si se podrá o no hacer uso del ratón.

Las partes de la pantalla de Emas son las siguientes, contadas desde arriba hacia abajo:

- Barra de menús.
- Barra de herramientas (sólo en modo gráfico).
- Área de edición.
- Barra de modo.
- Área de eco y minibuffer.

Veamos estas partes por separado.

3.1. La barra de menús y la barra de herramientas

La barra de menús se encuentra en la línea superior del *marco* ocupado por Emacs. En ella se agrupan varios mandatos que se supone que son de utilización más frecuente que los demás, y que además cambian según el *modo mayor* en que Emacs esté funcionando (Véase, sobre modos mayores, la sección 4.1 en pág, 17).

Como todos los mandatos existentes en la barra de menús son ejecutables por la vía ordinaria, en este documento, a efectos de simplificarlo, en general no mencionaré —al hablar de un concreto mandato— si forma parte o no de la barra de menús, la cual, al final no es sino una forma *alternativa* para ejecutar ciertos mandatos y, por ello, prescindible.

El usuario interesado simplemente tiene que ir abriendo los menús y comprobar las opciones que encierran y los mandatos que ejecutan. Algunos de ellos se explican en este documento y otros no.

Y exactamente lo mismo hay que decir de la barra de herramientas, añadiendo que, además, esta sólo existe cuando Emacs se ejecuta en modo gráfico: En ella existen una serie de *botones* cada uno de los cuales ejecuta un mandato *normal* cuando se hace click sobre él con el ratón. Por lo tanto también es un *modo alternativo* para la ejecución de mandatos y no volveré a referirme a ella en este documento.

3.2. El área de edición o *ventana*

Se trata de la parte más amplia de la pantalla de Emacs. En ella se realizará la edición. Esta zona puede a su vez dividirse en áreas más pequeñas a cada una de las cuales se la denomina *ventana*. En tal caso cada una de esas *ventanas* puede editar un *buffer* distinto.

> Debe hacerse notar que en este punto la terminología de Emacs difiere de la terminología más corriente. En los sistemas gráficos una *ventana* es una zona de la pantalla en la que cierta aplicación produce su salida. A eso Emacs le llama *marco*. En modo de sólo texto Emacs sólo puede tener un *marco* pero en modo gráfico puede tener muchos *marcos* diferentes (a los que otras aplicaciones llamarían *ventanas*). Ahora bien: un *marco* puede dividirse en *ventanas* que son zonas diferenciadas dentro del marco en donde se editan *buffers* distintos. A lo que Emacs llama *ventana* otras aplicaciones llamarían *paneles*

La división de la ventana principal en *subventanas* puede ser realizada automáticamente por Emacs para llevar a cabo ciertas tareas. Fundamentalmente cuando tiene que enviar al usuario un mensaje que no cabe en el *área de eco* (véase la sección 3.4, pág. 16). En tales casos normalmente la ventana adicional será automáticamente cerrada por Emacs cuando termine la tarea que provocó su apertura. En los demás casos habrá que cerrarla expresamente.

Las órdenes más importantes en relación con las ventanas son las siguientes:

Teclado	Mandato	Acción
C-x 0	delete-window	Borra la ventana activa
C-x 1	delete-other-windows	Borra todas las ventanas menos la activa
C-x 2	split-window-verticaly	Divide horizontalmente la ventana activa en dos ventanas
C-x 3	split-window-horizontaly	Divide verticalmente la ventana activa en dos ventanas
C-x o	other-window	Lleva el cursor a la próxima ventana
C-M-v	scroll-other-window	Desplaza hacia abajo el texto de la próxima ventana (como si en ella se hiciera AvPág)

3.3. La línea de modo

La línea de modo se encuentra en la parte inferior de la ventana de edición. Por lo tanto si hay varias *ventanas* cada una de ellas dispondrá de su propia línea de modo, la cual, en modo de sólo texto se ve en vídeo inverso, y en modo gráfico tiene un aspecto tridimensional y un color gris.

Esta línea de modo ofrece información variada sobre el *buffer* que se está editando en la ventana. Esta información se muestra en el siguiente formato:

sc:mod nombrebuf (mayor menores) - Pos—

Cada una de estas partes significa lo siguiente:

sc: Se refiere al sistema de codificación utilizado por el fichero que se está editando. El sistema de codificación se representa mediante un código numérico representativo[9].

mod: Aquí se recoge un código indicador de si el fichero que se está editanto ha sido o no modificado desde la última vez que se almacenó en disco. Puede asumir distintos valores de los que los más importantes son dos guiones (indicador de que el fichero no ha sido modificado) o dos asteriscos, indicador de que hay modificaciones del fichero que aun no se han guardado en el disco.

nombrebuf: Nombre del *buffer* que se está editando. Si estamos en modo gráfico ese nombre se ve en negrita.

(mayor menores): Aquí se informa (entre paréntesis) sobre el *modo mayor* de Emacs que está activado y algunos de los *modos menores* que puedan estarlo. De los modos de Emacs se habla más adelante (en pág. 17) por lo tanto de momento sólo diré que de esos modos dependen muchas de las funciones de Emacs.

[9] Para documentos escritos en español el sistema de codificación resulta fundamental, porque es el que determina cómo se verán los caracteres no anglosajones (vocales acentuadas, letra eñe, apertura de interrogantes y admiraciones, etc). Además, un sistema de codificación incorrecto puede provocar errores al compilar los documentos LaTeX. Como aquí no puedo extenderme demasiado simplemente diré sobre este punto que para evitar problemas lo mejor es activar para cada documento el mismo sistema de codificación que hayamos indicado como opción para el paquete "inputenc". Así, por ejemplo, si usamos «latin1», el sistema de codificación debe ser latin-1-unix y si usamos «utf8» el sistema de codificación debe ser «utf-8-unix».

En Emacs se asume que el sistema de codificación dependerá del "idioma por defecto"; pero para ficheros ya almacenados en disco, se intenta detectar el sistema de codificación, usando para ello varias "opciones de usuario" cuya explicación excede este documento.

Podemos, no obstante, variar el sistema de codificación para un fichero concreto. Eso lo podemos hacer ejecutando el mandato «`C-x RET f`» («`set-buffer-file-coding-system`»), o estableciendo la variable de fichero «`coding`» (véase al respecto el apéndice sobre la personalización de Emacs).

Esto último puede ser útil si en alguna ocasión tenemos interés en mantener cierta opción para el paquete `inputenc` y esa opción no es acorde con nuestro sistema general de codificación. Bastaría entonces con escribir, al final del fichero, entre las sentencias "Local Variables:" y "End:" una línea con el texto: «`coding: xxx`» donde xxx representa el sistema de codificación deseado.

pos: En este lugar se informa sobre la posición actual del cursor dentro del *buffer* (línea, columna y tanto por ciento). Téngase no obstante en cuenta que depende de la configuración de Emacs el que esa información esté activada o no. Es posible que solo se muestre información sobre el número de línea, o sobre el número de columna.

3.4. El área de eco y el minibuffer

La última línea de la pantalla de Emacs se usa para dos cosas distintas. En ella se muestra el área de eco y en ella se ubica también el minibuffer.

El área de eco es la zona en la que Emacs muestra los mensajes que tiene que dirigir al usuario. Estos mensajes se generan normalmente después de la ejecución de un mandato, y pueden advertir de algún error, o también pueden informar del resultado de la ejecución del mandato. Cuando tecleamos secuencias de teclado complejas que están asociadas a mandatos, esta zona de la pantalla también nos puede servir para ver qué parte del mandato hemos tecleado ya.

No obstante este último *eco* es un «*eco retardado*» es decir: cuando empezamos a escribir un mandato no se refleja inmediatamente, sino que es preciso hacer una pausa momentánea hasta que se produzca el eco. Pero una vez que éste ha empezado, ya no hay más retardos.

La existencia de este retardo determina, por su parte, que no haya nunca eco en los mandatos simples, que constan de un solo golpe de teclado.

El minibuffer es una pequeña ventana que Emacs utiliza cuando necesita que el usuario introduzca cierta información adicional para la ejecución de un mandato. Por ejemplo: si ejecutamos el mandato que abre un fichero, habrá que indicar cómo se llama el fichero que queremos abrir, y ello se hará en el *minibuffer*.

Como el minibuffer y el área de eco residen en la misma parte de la pantalla de Emacs, es el propio Emacs el que decide cuando activar uno y cuando activar otro. Cuando se active el minibuffer el cursor se colocará automáticamente en él, y cuando se desactive volverá a la ventana de edición.

4. Los modos de Emacs

4.1. Los modos mayores de Emacs

4.1.1. Modos mayores en general

Para entender lo que son los *modos mayores* de Emacs hay que partir de la idea de que Emacs es altamente configurable. En él podemos con facilidad añadir mandatos o modificar la forma en la que trabaja un mandato. Podemos cambiar el tipo de letra o el color de la misma para todo el texto o sólo para parte de él, el sangrado, etc.

Pues bien: partiendo de lo anterior los diseñadores de Emacs se preguntaron si no sería conveniente preparar *configuraciones alternativas* que optimizaran el funcionamiento de Emacs para ciertos tipos de ficheros especialmente conocidos, y el día en que se contestaron a sí mismos que esa era una buena idea, inventaron los *modos mayores* de Emacs.

Un *modo mayor* de Emacs no es sino una especie de *proconfiguración* o *personalización* de Emacs que lo optimiza para trabajar con ciertos tipos de ficheros. Por ejemplo: con ficheros de texto normal, o con ficheros que contengan programas escritos en C, o en Java, o con ficheros HTML... Y por supuesto, con ficheros escritos en TEX o en LATEX.

¿En qué consiste la optimización para trabajar con cierto tipo de ficheros? Pues eso depende del tipo de ficheros que sea. En todo caso normalmente el ajuste que los distintos modos mayores realizan se traduce en lo siguiente:

- **Reconocimiento de la sintaxis:** Emacs, para los tipos de ficheros que conoce, puede identificar en el texto lo que constituyen *instrucciones* y resaltarlas gráficamente. Por ejemplo: en ficheros LATEX Emacs identifica y muestra en un color distinto el texto normal, las marcas de comentario (y los comentarios propiamente dichos), las macros que generan unidades estructurales de LATEX (parte, capítulo, sección, etc), las que abren y cierran entornos, las que afectan a la paginación y el resto de las macros... Cada uno de estos textos se mostrará de un color distinto y, además, las órdenes que dan formato a las fuentes son reconocidas y algunas de ellas provocan que en pantalla se muestre el texto *preformateado*, es decir: se verá en negrita el texto que en el documento final debe ir en negrita, en cursiva el que irá en cursiva, etc. Todo ello, sin duda, contribuye muchísimo a facilitar la lectura del documento fuente.
- **Reajuste del funcionamiento de ciertos mandatos y teclas:** Hay mandatos de Emacs que funcionan de forma distinta según el modo en el que Emacs se

encuentre. Asimismo algunas teclas funcionan de modo distinto dependiendo del modo. Hay concretamente tres teclas que se suelen ver afectadas por el cambio de modo: TAB, RETRO y C-j.

En particular, el reajuste de la tecla TAB se debe a que Emacs procura siempre realizar un sangrado *inteligente*, ajustado a la sintaxis del tipo de documento que sea, y dirigido a facilitar la legibilidad del mismo. Esto, lógicamente, depende del *lenguaje* empleado por dicho documento (si es un lenguaje de programación o similar).

- **Inclusión de algunos mandatos adicionales:** Estos mandatos están dirigidos a realizar ciertas tareas específicas para el tipo de fichero de que se trate en cada caso. Por ejemplo, en el modo LaTeX, estando activo el paquete AUC TeX, se facilitan cerca de cien comandos nuevos específicamente dirigidos a la escritura de ficheros TeX.

Es decir: mediante los modos mayores podemos convertir a Emacs en un editor *especializado* para numerosos tipos de ficheros: tantos como modos de Emacs hay.

Todos los modos mayores se agrupan en tres categorías: Modos relacionados con ficheros de texto plano o de texto con marcas (en donde se incluyen los modos TeX y LaTeX), modos relativos a lenguajes de programación (y aquí Emacs es capaz de manejar cerca de treinta lenguajes de programación diferentes), y modos para ficheros de propósitos específicos (ficheros de correo electrónico, ficheros dirigidos a comunicarse con la *shell*, etc).

4.1.2. Seleccionar un modo mayor

Como el *modo mayor* en el que se esté afecta a numerosos aspectos del funcionamiento de Emacs, algunos de ellos básicos, estos son incompatibles entre sí, es decir: en cada momento sólo puede estar activo un modo mayor: cuando activamos uno, desactivamos el anterior. En la línea de modo (véase sección 3.3, pág. 15) se nos informa siempre del nombre del modo activo.

Cuando se abre un fichero, normalmente Emacs selecciona correctamente el *modo* mayor adecuado para ese fichero atendiendo a su extensión. Pero a veces la extensión de los ficheros es ambigua. Un fichero de extensión .tex, por ejemplo, puede albergar un texto escrito en LaTeX, en plain TeX, en AmsTeX, en infoTeX, etc.

Por ello si Emacs no ha seleccionado correctamente el modo adecuado, podemos ser nosotros mismos quienes indiquemos directamente el modo mayor en el que queremos trabajar. Para ello hay que pulsar «`M-x`» y escribir el nombre del modo seguido de «`-mode`». Por ejemplo: «`tex-mode`» o «`latex-mode`». Más tarde,

si queremos volver al modo que Emacs asignaría por defecto a dicho fichero, nos bastará con ejecutar el mandato «`normal-mode`».

También podemos escribir en el mismo fichero la información necesaria para que Emacs automáticamente cargue el modo adecuado con independencia de la extensión del fichero. Eso lo podemos hacer de dos maneras:

- Escribiendo en la primera línea del fichero que no esté en blanco, el texto «`-*-`*NombreModo*`-*-`». Si el modo en cuestión es un lenguaje de programación o, en general, un lenguaje sometido a convenciones estrictas de sintaxis (como LaTeX) debemos anteponer a dicho texto el carácter adecuado para que el compilador ignore dicha línea, es decir: una marca de comentario. Por ejemplo: en un fichero escrito en formato LaTeX la primera línea no en blanco debería decir: «`% -*-latex-*-`», resultando indiferente cuantas marcas de comentario pongamos.
- Estableciendo la *variable de documento* «`mode`». Al respecto consúltese sobre la creación de este tipo de variables en el apéndice relativo a la personalización de Emacs.

4.1.3. Listas de tareas al iniciar un modo («`Hooks`»)

En Emacs se denomina *Hook* a un tipo especial de variable cuyo significado es el de contener dentro de sí una lista de tareas que deben realizarse por orden. Estas tareas pueden ser de muy distinta naturaleza: ejecución de funciones de Elips, asignación de valor a ciertas variables, ejecución de aplicaciones externas, etc.

Ciertos modos mayores cuentan con una variable de este tipo que se ejecuta cuando el modo se activa. Por lo tanto, modificando el contenido de esta variable podremos *personalizar* el funcionamiento del propio modo. Para ello suele usarse una función denominada «`add-hook`» que añade una tarea a la lista de tareas. Así por ejemplo la sentencia lisp

```
(add-hook 'LaTeX-mode-hook 'turn-on-reftex)
```

Añade a la lista de tareas que se ejecuta cuando se activa el modo LaTeX una tarea consistente en "activar" reftex, que es un *modo menor* especialmente útil para ficheros LaTeX.

4.2. Los modos menores

A diferencia de los modos mayores, que son incompatibles entre sí, en Emacs se llama *modo menor* al modo de funcionamiento en el que cierta utilidad se encuentra activada. Cada modo menor activa una manera especial de comportarse,

pero estas son compatibles entre sí, es decir: podemos funcionar simultáneamente con numerosos modos menores.

Los modos menores se activan o desactivan escribiendo «`M-x`» seguido del nombre del modo y de la palabra «`-mode`» aunque también podemos hacer que al activar un modo mayor determinado, se activen automáticamente ciertos modos menores. Para ello debe tratarse de un modo mayor cuya activación admita una lista de tareas (es decir, en donde se pueda usar el mandato «`add-hook`» y que los modos menores a activar admitan el método llamado «`turn-on`». En tales casos la sentencia elips

```
(add-hook 'NombreMayor-mode-hook 'turn-on-NombreMenor)
```

producirá la activación automática del modo menor llamado *NombreMenor* cada vez que se entre en el modo mayor denominado *NombreMayor*.

De otro lado, el mandato de activación de los modos menores, además, funciona como un conmutador: si el modo estaba activado lo desactiva y si estaba desactivado lo activa. En un momento dado podemos saber si cierto modo menor está o no activado comprobándolo en la línea de modo (véase sección 3.3 en pág. 15), si bien hay algunos modos menores que no se ven reflejados en ella, y respecto de otros, en lugar del verdadero nombre del modo se muestra una abreviatura.

Por lo demás los modos menores son muy diferentes entre sí. Algunos son globales, es decir: afectan a cualquier *buffer* editado en la sesión de trabajo actual. Otros son locales: se deben activar o desactivar para cada *buffer* individual. Los hay que pueden funcionar como modo mayor y como modo menor, y en muchos de ellos el nombre del modo es también el nombre de una variable que controla su funcionamiento.

5. Edición básica con Emacs

5.1. Operaciones con ficheros y con *buffers*

5.1.1. Ficheros y *buffers*:

La precisión terminológica de Emacs es francamente admirable, aunque contribuye a hacer que parezca más diferente a los demás programas de edición de lo que realmente es.

Uno de los ejemplos más claros de lo que acabo de decir lo tenemos en lo que es objeto de edición. Casi todas las aplicaciones de edición hablan de abrir, cerrar,

modificar, imprimir o guardar *ficheros*. Emacs, sin embargo, no aplica todas estas nociones, a los ficheros, sino que alguna de ellas se aplica a los *buffers*.

Se entiende por *buffer* una zona de memoria en la que se encuentra el texto objeto de edición, mientras que se entiende por fichero un conjunto de datos almacenado en el disco o en algún soporte similar que lo conserve en su estado actual. Es decir: un fichero es —como tal fichero— estable e inmodificable (aunque podemos borrarlo y crear otro fichero con el mismo nombre y distinto contenido); un *buffer*, por el contrario, se encuentra en la memoria RAM, es volátil y esencialmente modificable.

Cuando leemos un fichero, lo que hacemos es copiar su contenido a una zona de memoria (a un *buffer*), y durante la edición lo que modificamos es esa zona de memoria, el fichero como tal permanece inalterado hasta que explícitamente demos la orden de grabar en el disco las modificaciones.

> Además, no todos los *buffers* que hay en Emacs corresponden a ficheros. Cuando Emacs se inicia se abren dos *buffers* que no están asociados a ningún fichero: «`*scratch*`» y «`*messages*`» (obsérvese que el nombre de ambos *buffers* empieza y acaba por un asterisco, lo que indica que no es un *buffer* asociado a un fichero). Asimismo, cuando desde Emacs se ejecuta un mandato externo, se creará un *buffer* para recoger la posible salida de dicho mandato, o cuando se llama a alguna función de ayuda, esta se muestra en un *buffer*, etc.

Es decir: en Emacs abrimos ficheros, y trabajamos con *buffers* los cuales podemos, finalmente, grabar como ficheros.

5.1.2. Visitar (abrir) ficheros:

Otra de las peculiaridades terminológicas de Emacs —esta menos justificada que la anterior— es que a la operación de abrir un fichero se le llama *visitar* al fichero.

El mandato para *visitar* un fichero es «`C-x C-f`» («`find-file`»). Mediante este mandato Emacs leerá desde el disco un fichero y copiará su contenido en un *buffer* donde podremos editarlo. Tras introducir el mandato habrá que facilitar el nombre del fichero, el cual debe ser escrito en el minibuffer. Podemos aquí ayudarnos de la función de auto-completado del *shell*, es decir: conforme vayamos *navegando* por nuestro árbol de directorios, pulsando la barra espaciadora haremos que —en un *buffer* aparte— Emacs nos muestre cuántos ficheros hay en el directorio actual cuyo nombre coincida con la parte que hayamos escrito, y pulsando la tecla TAB haremos que se complete el nombre escrito con el primer fichero que coincida.

Al introducir el nombre del fichero recuérdese que Unix-Linux distingue entre mayúsculas y minúsculas.

Por otra parte Emacs carece de órdenes específicas para la creación de ficheros nuevos. Un fichero nuevo se creará simplemente cuando ordenamos a Emacs abrir un fichero inexistente. Es decir: si yo quiero crear un fichero llamado "prueba.tex" debo pulsar «`C-c C-f`» y a continuación escribir «`prueba.tex`» y pulsar RET: Emacs creará el fichero indicado, de la misma manera que, si hubiera existido, lo habría abierto.

Una operación parecida a la anterior es la de abrir un directorio, lo que se puede hacer con «`C-x d`» («`dired`»). Esta orden abre, en un *buffer* independiente un listado con el contenido del directorio que se le indique, permitiendo luego abrir, renombrar, borrar, imprimir, etc., cualquiera de los ficheros contenidos en dicho directorio.

5.1.3. Grabar en disco:

Para grabar en disco las modificaciones de un fichero, la orden fundamental es «`C-x C-s`» («`save-buffer`»). Tras la ejecución de este mandato, el fichero original será sustituido por el contenido de nuestro *buffer*. Si intentamos grabar un *buffer* que originariamente no era un fichero, Emacs nos preguntará por el nombre con el que queremos almacenar el *buffer*.

Si queremos preservar el contenido del fichero original, podemos grabar el *buffer* con otro nombre. Para ello debemos usar la orden «`C-x C-w`» («`write-file`»). Así, tras indicar el nuevo nombre, conservaremos la versión original del fichero.

Asimismo, si estamos editando simultáneamente varios *buffers*, podemos pulsar «`C-x s`» («`save-some-buffer`») para que Emacs compruebe, para cada uno de los *buffers*, si ha sido modificado y en caso afirmativo nos preguntará si deseamos guardar los cambios.

5.2. Desplazamiento por el texto

Aunque Emacs contiene un completo juego de mandatos para facilitar el movimiento del cursor —y los fanáticos de Emacs dicen, además, que esos mandatos son más efectivos que las teclas del cursor[10]— lo cierto es que las teclas del cursor funcionan y, por lo tanto, en una introducción *brevísima* como la actual pretende

[10]Cosa con la que yo no estoy de acuerdo. Primero porque aunque «`C-n`», por ejemplo, esté más cerca de los dedos que la flecha abajo, también hay que tener en cuenta otras circunstancias que afectan a la velocidad. Una de ellas es la de si estamos en modo gráfico (que siempre implica mayor lentitud, con la mano desplazándose del teclado al ratón), pero la principal es la de la automatización: La velocidad depende de lo automatizadas que tengamos las tareas, y es muy difícil que una tarea se

ser, no tiene sentido extenderme más en este punto. Por lo tanto asumase que las teclas de flecha mueven el cursor en la dirección indicada por cada una de ellas, las teclas inicio y fin nos desplazan, respectivamente, al principio o al final de la línea, y las teclas RePág y AvPág nos desplazan de pantalla en pantalla. Además de estos movimientos básicos, disponemos de los siguientes mandatos:

Desplazarse a	**Teclado**	**Alternativo**	**Mandato**
Principio del *buffer*	C-Inicio	M-<	beginning-off-buffer
Final del *buffer*	C-Fin	M->	end-off-buffer
Próxima palabra	M-Derecha	M-f	forward-word
Palabra anterior	M-Izquierda	M-b	backward-word
Próximo párrafo	C-Arroba		forward-paragraph
Párrafo anterior	C-Abajo		backward-paragrap
Línea central	M-r		move-to-window-line
Idem	C-l		recenter

Las dos últimas funciones son en realidad antitéticas, y si en el cuadro resumen he puesto que ambas hacen lo mismo, es por la dificultad que tiene explicar en una sola línea de texto la sutil diferencia entre ellas: «`M-r`» coloca el cursor en el extremo izquierdo de la línea central de la pantalla, mientras que «`C-l`» coloca la línea sobre la que esté el cursor en el centro de la pantalla. O sea: la primera mueve el cursor sin producir desplazamiento de texto (*scroll*), la segunda deja el cursor en su lugar pero produce desplazamiento de texto.

En fin: por último debo decir que las funciones M-Izquierda y M-Derecha también funcionan con la tecla Control (aunque sólo en modo gráfico y exclusivamente en Emacs, no así en Xemacs).

> A estas funciones se pueden añadir los mandatos no asociados a ninguna combinación de teclas: «`goto-char`» y «`goto-line`» que desplazan el cursor, respectivamente, al carácter o a la línea que se le indique. Esta última función es bastante útil cuando se está depurando un documento LaTeX, ya que el fichero .log generado por el compilador informa —en caso de fallo o de advertencia—, acerca de en qué línea se produjo el fallo.

automatice si, de un lado, las demás aplicaciones no utilizan la misma secuencia de teclado (salvo que sólo usemos esa aplicación) y, por otro lado, la secuencia utilizada *no nos dice nada*, como ocurre, en general, con las secuencias de Emacs a quienes no tengan por lengua materna el inglés. Esa fracción de segundo que tarda uno en recordar que está ejecutando Emacs y no otra aplicación, y que para ir a la próxima línea, en Emacs hay que pulsar *Control Next* («`C-n`»), es suficiente para que no pueda afirmarse que llevando la mano derecha a la tecla de flecha hacia abajo hayamos tardado más. Sin contar, además, con que, normalmente, cuando movemos el cursor *no estamos escribiendo nada*, por lo que el que la tecla que mueve el cursor esté más o menos lejos del resto de las teclas para escribir no es tan importante y que, por último, un comando de movimiento del cursor exige pulsar DOS teclas lo que también contribuye a que se tarde una fracción de segundo más. Pero en todo caso la cuestión es —como suele pasar— muy opinable y poco importante.

5.3. Selección de texto (la región)

La selección de texto es uno de los puntos en los que Emacs se mantiene al margen del resto de las aplicaciones. En terminología y en secuencias de teclado.

> Aunque la diferencia terminológica está basada en una diferencia conceptual:
>
> En realidad para Emacs en cada *buffer* hay dos indicadores internos, uno que se llama *marca* y otro que es el cursor (al que Emacs denomina *punto*). Se llama *región* al bloque de texto que hay entre la marca y el cursor. La región, por lo tanto, siempre existe, lo que ocurre es que, normalmente, no es visible y cuando no lo es está inactiva, salvo para ciertos mandatos.
>
> La región, por otra parte, sólo está visible —como regla general— en el periodo de tiempo que transcurre desde que se fija la marca en algún nuevo lugar hasta que se ejecuta un mandato que no sea de movimiento del cursor (incluyendo en este grupo los de búsqueda incremental.).
>
> Lo anterior se ve más claro con ejemplos. Supongamos que ejecutamos el mandato «`M-h`» que *selecciona* todo el párrafo, en realidad lo que habremos hecho es, primero mover el cursor al final del párrafo, establecer allí la marca y luego llevar el cursor al principio del párrafo, de modo que la región se extiende desde la marca hasta el cursor. Si inmediatamente después de ejecutado el mandato movemos el cursor veremos que la región va siguiendo al cursor desde el final del párrafo, lugar en donde fue fijada la marca por el mandato «`M-h`». Ahora bien, si ejecutamos cualquier mandato que no sea de movimiento del cursor (por ejemplo, insertamos un espacio en blanco, o pulsamos «`C-g`»), aparentemente la región desaparece; en realidad sigue allí pero deja de estar visible y activa.
>
> Si luego llevamos el cursor a otro lugar —por ejemplo tres párrafos más abajo— y pulsamos «`C-x C-x`» («`exchange-point-and-mark`»), veremos que la región vuelve a aparecer y se extiende del lugar en donde estaba el cursor, hasta el lugar en donde *antes habíamos fijado la marca*, ello es porque el último mandato ejecutado lo que ha ello es poner la marca donde estaba el cursor, y llevar el cursor a donde estaba la marca, haciendo visible la región como ocurre siempre que se fija la marca en un nuevo lugar mientras no se ejecute un mandato que no consista en mover el cursor.
>
> Eso de que la selección siga al cursor vaya a donde vaya puede llegar a hacerse molesto. Para detenerlo basta con ejecutar cualquier mandato o pulsar «`C-g`» (lo que también es *ejecutar un mandato*).

La principal diferencia entre la forma de funcionar que tiene lo que Emacs llama *región* y la forma en la que normalmente funciona la *selección* en otras aplicaciones, se encuentra en que normalmente cuando se realiza una acción y hay un bloque de texto seleccionado, esta acción —cualquiera que sea— afecta a toda la selección. En Emacs por el contrario la regla general es que los mandatos que trabajan sobre la región y los que no trabajan sobre ella son distintos. Y aunque esa regla tiene excepciones, estas no son muchas. Normalmente los mandatos que trabajan sobre la región se identifican porque en su nombre aparece la palabra “region”.

En particular en Emacs, el hecho de insertar un carácter, o pulsar la tecla SUPR o la tecla RETRO cuando un bloque de texto está seleccionado, no implica el que el texto seleccionado sea sustituido por el carácter pulsado, cosa que sí suele ocurrir en otras aplicaciones.

En fin: los mandatos más importantes para fijar la región son los siguientes:

Teclado	Mandato	Efecto
«C-SPC»	«set-mark-command»	Fija el principio de la región en el lugar donde se encuentre el cursor
«M-@»	«mark-word»	Selecciona una palabra hacia delante
«M-h»	«mark-paragraph»	Selecciona el párrafo actual
«C-x h»	«mark-whole-buffer»	Selecciona todo el *buffer*
«C-M-h»	«mark-defun»	Selecciona todo lo que esté encerrado en los paréntesis más próximos

El primero de los mandatos recogidos es el más corriente y está asociado, además de a «C-SPC», a «C-@»: tras ejecutarlo, la selección se extenderá allá a donde llevemos el cursor, con independencia del modo en que lo traslademos: con alguno de los mandatos para mover el cursor, haciendo click con el ratón, o ejecutando alguna de las funciones de búsqueda.

> «mark-word» no es un mandato demasiado útil, pues selecciona una sola palabra, aunque lo ejecutemos varias veces seguidas. Para seleccionar más de una palabra hay que indicarle —como parámetro numérico— el número de palabras a seleccionar.
>
> Asimismo, «mark-defun» lo que hace es seleccionar un bloque de texto encerrado entre determinados símbolos de apertura y ciere que pueden variar según el *modo mayor* en el que se esté ejecutando Emacs. Para los modos TEX y LATEX esos símbolos son los paréntesis.
>
> Por otra parte, en Emacs *la marca* es útil, no sólo como punto de partida para establecer la región, sino también para señalar un lugar del *buffer* al que pretendemos volver más tarde de manera rápida. Esto es así porque cada *buffer* de Emacs recuerda los 16 últimos lugares en los que se colocó la marca. A esto se le denomina *el círculo de marcas* y nos permite movernos con rapidez mediante los mandatos «C-u C-SPC» y «C-u C-@», o sea: los mandatos para establecer la posición de la marca, pero con un prefijo numérico indicador del número de marca al que queremos retroceder. Cuando por este procedimiento saltamos a la posición que en algún momento anterior tuvo la marca, esta posición se coloca en la última posición dentro del *círculo de marcas*.

5.4. Borrar, eliminar y recuperar texto

Esta es una de las operaciones más corrientes en la edición de textos, y supongo que a estas alturas a nadie le extrañará que la terminología de Emacs, una vez más,

sea diferente pero que, al mismo tiempo, Emacs sea más potente que la mayor parte de las aplicaciones.

Empecemos por recordar la terminología cada vez más extendida, para así poder señalar con claridad las diferencias con Emacs.

Numerosas aplicaciones de manejo de texto distinguen las siguientes tres acciones posibles:

- **Copiar**: Implica que en una zona de memoria intermedia se haga una copia de un bloque de texto, previamente seleccionado, con vistas a luego poderlo insertar en algún otro lugar del *buffer*.
- **Cortar**: Es una operación similar a la anterior, pero con la diferencia de que, tras copiar el texto a dicha zona de memoria, este es eliminado de su localización original; equivale por lo tanto a copiar y luego borrar..
- **Pegar**: Así se suele llamar a la operación por la que el texto que se encontraba en la zona intermedia de memoria, como consecuencia de una operación de cortar o copiar, se inserta en algún lugar del *buffer*.

Las principales características de estas operaciones son dos: La primera es que el texto existente en esa zona de memoria intermedia, puede insertarse tantas veces como queramos, y la segunda es que cada vez que efectuamos una operación de copiar o cortar, el contenido de esa zona de memoria es sustituido, de modo que a partir de ahora ya sólo podremos insertar (pegar) el nuevo texto.

Pues bien: en Emacs, además de cambiar la terminología, cambia también la funcionalidad, pues cada operación equivalente a lo que otras operaciones llaman *copiar* o *pegar*, genera un nuevo bloque de memoria intermedia que no sustituye al anterior, sino que se añade a él.

Pero veamos bien como funciona eso, y hagámoslo con la terminología propia de Emacs.

5.4.1. Borrar y eliminar texto

En Emacs se distingue entre lo que he traducido como *borrar* texto («*erase*») y *eliminar* texto («*kill*»)[11]. En el primer caso el texto desaparece del *buffer* y no se puede recuperar (salvo volviéndolo a escribir). Por el contrario cuando eliminamos

[11] Literalmente “erase” significa *borrar* o *eliminar*, y “kill” *matar* o *asesinar*. En mi traducción, que no es literal, he asignado a cada una de ambas acciones el término que se usa en el “tutorial español de Emacs”, incluido en sus distribuciones.

texto lo que hacemos es quitarlo del *buffer*, enviándolo a una zona independiente de memoria desde la que podremos rescatarlo («*yank*»[12]) cuando queramos.

Para borrar texto (sin poderlo recuperar) se utilizan las teclas RETRO y SUPR: La primera borra el carácter que esté a la izquierda del cursor y desplaza el cursor un carácter a la izquierda, y la segunda borra el carácter que esté debajo del cursor y desplaza el texto hacia la izquierda, de modo que, bajo el cursor queda el carácter que antes estaba a su derecha. O dicho de modo más gráfico: RETRO borra hacia detrás y SUPR hacia delante.

El borrado, por lo tanto, se hace siempre carácter a carácter. No hay ningún mandato para "borrar" de golpe más de un carácter.

Ahora bien: el grupo de mandatos que provocan la *eliminación* de texto es mucho más numeroso. Los más importantes son los siguientes:

Teclado	**Mandato**	**Acción**
«C-k»	«kill-line»	Elimina el texto hasta el final de la línea
«M-RETRO»	«backward-kill-word»	Elimina una palabra hacia atrás
«M-d»	«kill-word»	Elimina la próxima palabra
«C-w»	«kill-region»	Elimina el texto seleccionado
«M-z»	«zap-to-char»	Elimina hasta la próxima ocurrencia del carácter que se teclee inmediatamente detrás del mandato
«M-w»	«kill-ring-save»	Añade el texto seleccionado a la memoria intermedia, como si se hubiera eliminado, pero sin eliminarlo

Hay que aclarar que para eliminar palabras hacia atrás en mi distribución de Emacs funciona tanto «M-RETRO» como «C-RETRO» (cuando Emacs se ejecuta en modo gráfico).

Como dije al principio, en Emacs las distintas eliminaciones se guardan en bloques de memoria diferentes de modo que eliminar un texto no impide que podamos luego recuperar el texto de alguna eliminación anterior.

¿Cómo organiza Emacs los distintos bloques de memoria que contienen texto eliminado? Para ello, cuando se produce un mandato de eliminación (o de copiar sin eliminar, que para estos efectos es igual), Emacs comprueba si el mandato inmediatamente anterior fue también de eliminación. En caso afirmativo, guarda el texto que se acaba de eliminar en el mismo bloque de memoria en donde guardó el texto anteriormente eliminado. En caso negativo, abre un nuevo bloque de memoria para guardar el nuevo texto.

[12]Resulta muy difícil realizar una traducción exacta del término utilizado por Emacs: *yank* significa tirar de algo hacia nosotros, jalar o arrastrar una cosa. Yo lo he traducido por *recuperar* o *rescatar*, que transmite la idea pero que no es tan descriptivo. El tutorial de Emacs en español habla de "*reinsertar*".

Es decir: si, por ejemplo, eliminamos sucesivamente tres parabras contiguas, las tres se almacenarán en el mismo bloque de memoria y serán recuperadas conjuntamente, siempre y cuando los tres mandatos de eliminación sean consecutivos, es decir: entre ellos no se haya ejecutado ningún otro mandato, incluidos los mandatos de movimiento del cursor.

En el caso de que hayamos eliminado un texto y, más tarde, queramos añadir otro texto al mismo bloque, pero tras la última eliminación hayamos ejecutado algún mandato intermedio, podemos indicarle a Emacs que *reactive* el bloque correspondiente a la última eliminación pulsando «`C-M-v`» («`append-next-kill`») *inmediatamente antes* de la siguiente eliminación. El efecto de este mandato será que el nuevo texto eliminado se añadirá al último bloque como si entre la anterior eliminación y la nueva no hubiera habido otros mandatos.

5.4.2. Recuperar texto previamente eliminado

Para recuperar el texto que ha sido eliminado o salvado, la orden fundamental es «`C-y`» («`yank`»). Mediante ella insertaremos, en el lugar donde se encuentre el cursor, el texto más recientemente eliminado o salvado.

Inicialmente, cuando se ejecuta este mandato, se inserta el texto correspondiente a la última eliminación, pero —como ya se señaló— una eliminación no hace desaparecer a las demás, por lo tanto, si lo que queremos es insertar el texto correspondiente a alguna eliminación anterior, debemos, tras haber recuperado el texto de la *última* eliminación, ir pulsando sucesivamente «`M-y`» («`yank-pop`») para irnos moviendo por lo que Emacs llama *el círculo de textos eliminados*. Cada pulsación de «`M-y`» sustituirá el texto insertado por el de la eliminación anterior.

Por otra parte «`M-y`» no funciona si el mandato inmediatamente anterior a él no ha sido un mandato de recuperación de texto (incluido el propio «`M-y`»)

> Los anteriores mandatos son los *más corrientes*, pero la verdad es que la riqueza de Emacs en punto a almacenar texto para luego recuperarlo, supera con creces lo que se ha visto. El lector curioso puede consultar, en la ayuda de Emacs, los temas correspondientes a rectángulos (*rectangles*) y registros (*registers*). En particular los registros de texto resultan extremadamente útiles.

6. Operaciones de búsqueda y reemplazo

6.1. Búsqueda en general

6.1.1. Búsqueda incremental

Esta es otra de las operaciones corrientes durante la edición. Se trata de buscar dentro del *buffer* un determinado texto para desplazarnos a él (búsqueda), o de sustituir en el *buffer* de forma automática todas las apariciones de una cadena de texto por otra cadena distinta.

En Emacs la búsqueda es, en principio, incremental, y toma como punto de partida la posición del cursor.

El que la búsqueda sea incremental significa que el cursor se va desplazando en la dirección indicada conforme vamos tecleando el texto a buscar. Para realizar esta operación los pasos a dar son los siguientes:

1. En primer lugar hay que ejecutar el mandato «`C-s`» («`isearch-forward`») o «`C-r`» («`isearch-backward`») según queramos realizar una búsqueda hacia delante o hacia detrás.

2. El cursor saltará al minibuffer, donde podremos ir tecleando poco a poco la cadena buscada. En la zona de edición, conforme vayamos tecleando, se irá mostrando la próxima ocurrencia del texto buscado. Para cambiar la dirección de la búsqueda podemos teclear el mandato correspondiente a la dirección deseada.

3. Cuando se ha encontrado un texto, podemos añadir otro carácter para afinar la búsqueda, o volver a pulsar «`C-s`» o «`C-r`» para localizar la próxima aparición (hacia delante o hacia detrás) del mismo texto localizado.

4. Si durante la búsqueda se localiza una palabra que es exactamente la que estábamos buscando pero aun no habíamos terminado de teclear, «`C-w`» hace que esa palabra entera se incorpore a la cadena de búsqueda. De manera similar «`C-y`» incorpora a la cadena de búsqueda toda la línea de la ventana de edición a partir del lugar donde se encuentra el texto localizado.

La búsqueda incremental cesa cuando se pulsa ESC, RET o se ejecuta cualquier mandato que no tenga algún significado especial en la búsqueda, incluido cualquier movimiento del cursor.

Durante la búsqueda podemos ademas usar la tecla RETRO para borrar el último carácter introducido y en su lugar escribir cualquier otro; eso no afectará a la

búsqueda, pero si hará que en la zona de edición se deje de mostrar el texto que se había localizado y se vuelva al lugar en donde se encontraba el texto que se localizó cuando la cadena de búsqueda no tenía todavía el carácter que se acaba de borrar.

6.1.2. Repetir búsquedas anteriores

Después de haber realizado alguna búsqueda, pulsando «`C-s C-s`» la repetiremos. El primer C-s indica que se inicie una nueva búsqueda, y el segundo indica que se debe rescatar la última búsqueda.

Y es que Emacs va almacenando las cadenas objeto de búsqueda en zonas de memoria independientes que constituyen lo que se llama *el círculo de búsquedas*. Estas búsquedas almacenadas pueden ser rescatadas si, en una nueva búsqueda, pulsamos «`M-p`» o «`M-n`» para irnos moviendo entre las sucesivas búsquedas almacenadas.

6.1.3. Búsqueda no incremental y búsqueda por palabras

Si queremos que nuestra búsqueda no sea incremental, basta con, inmediatamente después de haber arrancado una búsqueda incremental, y antes de haber introducido el primer carácter de búsqueda, pulsar RET. Ello provoca que el cursor salte al minibuffer donde podremos escribir el texto a buscar y luego pulsar RET otra vez para iniciar la búsqueda.

Si queremos que la búsqueda localice palabras completas, tras pulsar «`C-s RET`» o «`C-r RET`» hay que pulsar, antes de introducir el texto de búsqueda, «`C-w`». Ello provocará que la búsqueda se haga sobre palabras, sin tener en cuenta cómo se separen. Es decir: si buscamos por ejemplo, la cadena "*tararí que te ví mariví*", la búsqueda por palabras localizará esa cadena aunque el número de espacios en blanco varíe, o aunque entre dos de esas palabras haya un salto de línea, o incluso si hay signos de puntuación entre las palabras.

Esta forma de búsqueda es en Emacs más útil que en otros editores debido a su modo menor auto-fill (que introduce automáticamente saltos de línea al final de cada línea de texto) y a su sistema de "sangrado inteligenge" que provoca el que en ciertos casos se añadan automáticamente espacios en blanco adicionales. Es decir: en Emacs no siempre podemos estar seguros de que la separación entre dos palabras consista exactamente en un espacio en blanco. Pero este inconveniente se suprime mediante la función de búsqueda por palabras.

6.2. Reemplazar texto

6.2.1. Reemplazo normal de texto

Una operación de reemplazo significa que a partir de la posición del cursor se sustituirán todas las ocurrencias de un texto por otro texto distinto,

El mandato para lograr esto es «`replace-string`». Este mandato no está asociado por defecto a ninguna secuencia de teclas, por lo tanto para ejecutarlo hay que pulsar M-x y escribirlo en el minibuffer. Tras ello se nos preguntará la cadena de búsqueda, y tras introducirla y pulsar RET se nos preguntará por la cadena de reemplazo. Respecto de la introducción de la cadena de búsqueda y de reemplazo téngase en cuenta lo que a propósito de la búsqueda incremental se dijo para localizar ciertos caracteres especiales (tabuladores y marcas de fin de línea).

Por otra parte este mandato tiene la especialidad de que si en el momento de ejecutarse hay una zona seleccionada (o sea: una región visible), su efecto se limitará a dicha zona. En el resto de los casos el mandato provocará la sustitución desde el cursor hasta el final del *buffer*.

Con un argumento numérico provocaremos que la búsqueda se limite a palabras completas (con las matizaciones hechas en el epígrafe anterior sobre la búsqueda por palabras).

En cuanto a la sensibilidad entre mayúsculas y minúsculas, depende del texto a buscar: si todo él es en minúsculas, en principio la operación se hará sin distinguir entre mayúsculas y minúsculas. Y del mismo modo: si la cadena de reemplazo está toda en minúsculas, Emacs alterará la cadena de reemplazo para ajustarla al patrón mayúsculas-minúsculas del texto localizado.

Con un ejemplo se entendrá mejor: Si pedimos que se reemplace el texto “hola” por el texto “adios”, Emacs, en primer lugar, considerará que la búsqueda es insensible y por lo tanto reemplazará con “adiós” tanto las ocurrencias de “hola” como las de “Hola” o las de “HOLA”. Ahora bien, donde originalmente había “hola” pondrá “adiós”, donde había “Hola” pondrá “Adiós” y donde había “HOLA” pondrá “ADIÓS”.

6.2.2. Reemplazo selectivo

Si no queremos sustituir todas las ocurrencias de cierto texto, sino sólo algunas, podemos usar el mandato «`M-%`» («`query-replace`»), el cual va localizando las ocurrencias del texto buscado y pregunta al usuario, para cada ocurrencia, si desea efectuar el reemplazo.

Cada vez que se nos consulte si queremos efectuar el reemplazo, podemos pulsar alguna de las teclas que a continuación se exponen:

SPC Reemplaza la ocurrencia actual y busca la próxima.

RETRO No reemplaza la ocurrencia actual, y busca la próxima.

, (Coma) Reemplaza la ocurrencia actual, muestra el resultado, y vuelve a preguntar (por si queremos mantenerlo o anularlo)

RET No realiza el reemplazo y termina la operación.

. (Punto). Reemplaza la ocurrencia actual y termina la operación.

! Deja de preguntar y realiza todos los reemplazos.

^ Vuelve a la ocurrencia anterior y vuelve a preguntarnos por ella (deshace la última operación).

C-r Nos permite una edición recursiva, es decir: podemos modificar el texto y luego volver a esta operación pulsando «C-M-c» cuando queramos volver.

e Permite editar la cadena de reemplazo para modificarla de aquí en adelante.

6.3. Cuestiones adicionales relativas a las operaciones de búsqueda

6.3.1. Caracteres especiales (como insertarlos)

En la cadena de búsqueda podemos incluir cualquier carácter; sin embargo algunos caracteres hay que introducirlos de forma especial, ya que si son pulsados en el minibuffer se interpretarán de otra manera. Por ejemplo, para buscar saltos de línea, no podemos teclear RET, ya que Emacs interpretaría que ese carácter quiere indicar que hemos terminado la búsqueda.

En particular:

- Para buscar marcas de tabulación debemos escribir en la cadena de búsqueda como carácter a buscar «C-q TAB».
- Para buscar saltos de línea debemos escribir en la cadena de búsqueda, como carácter a buscar «C-q C-j»

En los dos casos anteriores se utiliza el mandato «C-q» («quoted insert»), el cual imprime el carácter no gráfico que se pulse a continuación

6.3.2. Distinguir (o no) entre mayúsculas y minúsculas

La sensibilidad o no a la distinción entre mayúsculas y minúsculas, funciona de modo distinto en la búsqueda incremental y en el resto de operaciones de búsqueda.

Sensibilidad a las mayúsculas en la búsqueda incrementa:l Por defecto la búsqueda no distingue entre minúsculas y mayúsculas siempre y cuando, como cadena de búsqueda *usemos solamente letras minúsculas*. En cuanto introduzcamos una letra mayúscula, la búsqueda será sensible a la distinción entre ambos tipos de letras.

Es decir: una búsqueda que empezó como *insensible* a la distinción, puede convertirse, en cualquier momento, en sensible. Para ello hay que teclear un carácter de búsqueda en mayúsculas. Y cuando digo *teclear* quiero decir exactamente eso: el carácter en mayúsculas debe ser expresamente introducido; pero si, por ejemplo, en una operación de búsqueda, se localiza una palabra que decidimos incorporar a la cadena de búsqueda pulsando «`C-w`», y dicha palabra tenía alguna letra mayúscula, eso no provocará que la búsqueda que no era sensible a la distinción entre ambos tipos de letras pase a serlo, sino que la palabra (o la línea si se usó «`C-y`») se incorporará a la cadena de búsqueda, en minúsculas (en el caso de que todavía no se hubiera tecleado ninguna tecla mayúscula).

También podemos indicar de forma manual si queremos que la búsqueda sea sensible o insensible a la distinción entre mayúsculas y minúsculas. Para ello basta con, durante la búsqueda, pulsar «`M-c`» Esta secuencia cambia la sensibilidad de la búsqueda: Si era insensible la hace sensible y viceversa[13]. Téngase en cuenta, no obstante, que el efecto de «`M-c`» no va más allá de la operación de búsqueda actual.

Fijar la sensibilidad a las mayúsculas de modo general para un *buffer* concreto: Los procedimientos anteriores sólo sirven en la búsqueda incremental y, en parte, en la búsqueda que se hace para una operación de reemplazo (cuando en la cadena a buscar haya letras en mayúsculas), pero no funciona en la búsqueda por palabras, ni en la búsqueda por expresiones regulares (véase más adelante), así como en la búsqueda para una operación de reemplazo en la que queramos sustituir sólo las apariciones en minúsculas del texto a cambiar.

En estos otros casos la regla es la de que la búsqueda no será sensible a la distinción entre mayúsculas o minúsculas, a no ser que se le haya asignado el valor “nil” a la variable «`case-fold-search`», que por defecto vale “t”.

Esta variable es una variable de “*buffer*”, es decir: una modificación de su valor se entiende realizada exclusivamente para el *buffer* activo. Por lo tanto, es una gran candidata a convertirse en variable de fichero (véase el apéndice sobre la personalización de Emacs), para aquellos ficheros en los que queramos que la búsqueda distinga entre mayúsculas y minúsculas siempre.

[13] En realidad la secuencia «`M-c`» lo que hace es ejecutar el comando «`capitalize-word`», que provoca que la próxima palabra se convierta en mayúsculas. Lo que ocurre es que la ejecución de ese comando durante una operación de búsqueda incremental, afecta a la sensibilidad de la búsqueda respecto de la distinción entre mayúsculas y minúsculas.

OJO: Si cambiamos el valor de esta variable mediante un "*buffer de personalización*" y le indicamos a Emacs que guarde los cambios para siempre, lo que provocaremos es que un cambio en el valor "por defecto" para todos los documentos. Eso produce un efecto distinto del que obtendríamos cambiando el valor de la variable para el documento actual. Por ello en el menú de Emacs se ofrece la posibilidad de alterar el valor de la variable "para el documento actual" (en "Options" y luego en "Case-insensitive search").

6.4. Otras órdenes de búsqueda y reemplazo

Además de los mandatos examinados Emacs dispone de mandatos para buscar o reemplazar *expresiones regulares*.

Los mandatos que se refieren a expresiones regulares son los siguientes:

«C-M-s» («`isearch-forward-regexp`») Realiza una búsqueda incremental hacia adelante de una expresión regular.

«C-M-r» («`isearch-backward-regexpe`») Realiza una búsqueda hacia atrás de una expresión regular de modo incremental.

«`replace-regexp`» Realiza una operación de reemplazo empleado como patrón de búsqueda una expresión regular.

«C-M-%» Realiza una operación de reemplazo selectivo con una expresión regular como patrón de búsqueda.

A todo esto ¿qué son las expresiones regulares? Responder a esta pregunta es fácil. Lo difícil es saber cómo se usan. Para ello necesitamos toda la próxima sección y no estoy seguro de que sea suficiente

7. Expresiones regulares

7.1. Concepto

Un concepto simple de «expresión regular» diría que éstas consisten en una especie de patrón de búsqueda. Con ellas no se trata tanto de localizar un texto exacto (aunque eso puede hacerse), como de localizar un texto que se ajuste a determinadas pautas o patrones. Un buen ejemplo serían los llamados caracteres comodín de MS-DOS (o de las distintas *shells* de unix). Todos sabemos que, por ejemplo, la orden «`dir *.doc`» (o «`ls *.doc`») devuelve una lista que contiene el nombre de los documentos del directorio activo cuya extensión sea "doc". Es decir: «`*.doc`» es un patrón de búsqueda que devuelve todos los nombres que acaben en ".doc".

Pero las expresiones regulares son mucho más que eso y de una potencia infinitamente mayor. De hecho mediante ellas podemos realizar búsquedas extremadamente precisas, como, por ejemplo: en un documento LaTeX, buscar una frase que contenga la palabra "hola", comprobar si dicha frase está en un entorno de tipo "enumerate" y, en caso afirmativo, cambiar el entorno por otro de tipo "itemize". Otro ejemplo de más interés: Asegurarse mediante una sola operación de que todos los párrafos de nuestro documento terminan con un punto y aparte salvo los que terminen con el carácter de los dos puntos o un cierre de interrogación o de exclamación.

Eso y más puede hacerse mediante una sola operación[14]. Si bien es cierto que la potencia tiene el precio de la complejidad. Las expresiones regulares son complejas. En ellas es fácil cometer errores y para el profano tienen el aspecto de cadenas de texto ininteligibles.

Esa complejidad se debe a varias razones. Una es la de que en estas expresiones se combinan caracteres normales de búsqueda con otros que tienen un significado especial y que reciben distintos nombres, de los que personalmente considero que el más claro es el de *operadores*. Pero es que, además, no hay que olvidar que las expresiones regulares utilizan la ténica denominada "*backtracking*" o "vuelta atrás" que es un tipo de programación compleja usado, por ejemplo, en el lenguaje PROLOG.

De hecho las expresiones regulares no sólo se usan en Emacs sino también en otras aplicaciones que incluyen editores potentes (como vi), *shells* de sistemas operativos, ciertas aplicaciones (grep, egrep) y algunos lenguajes de programación como sed, awk o Perl, que es el lenguaje que hace un uso más potente y depurado de este tipo de expresiones.

El problema es que la sintaxis de las expresiones regulares varía en todos estos casos. Y aunque en Internet se puede obtener bastante información, en la mayor parte de los casos esta se refiere a las expresiones regulares en Perl. Sobre su uso en Emacs se encuentran bastantes menos cosas.

Esta es la razón de que en las líneas que siguen me haya atrevido a intentar una introducción al uso de estas expresiones en Emacs.

[14] De hecho cuando veamos la creación de documentos LaTeX con Emacs veremos que las expresiones regulares se usan para cosas tan diferentes como para determinar donde acaba el preámbulo de un documento, qué tipo de información hay que almacenar de él, qué líneas, de un fichero LaTeX cabe considerar constitutivas de "títulos", qué tipo de caracteres no deben admitirse en una etiqueta, etc. Incluso el mismo Emacs utiliza internamente este tipo de expresiones para saber cómo dividir el documento en párrafos, frases o palabras

7.2. Operadores en las expresiones regulares (caracteres especiales)

Una expresión regular es una cadena de texto y, como tal, se compone de caracteres. Esos caracteres pueden ser *normales* o *especiales*. Los normales se interpretan tal cual; pero son los caracteres especiales los que dotan de potencia a las expresiones regulares.

En Emacs los caracteres con un significado especial en expresiones regulares son:

```
$ ^ . * + ? [ ] - \
```

Alguien pudiera pensar que si los caracteres especiales son sólo esos, las expresiones regulares no pueden ser tan difíciles. Pero el reducido número de caracteres especiales es engañoso, ya que, de un lado, el significado de estos caracteres especiales cambia según el contexto, y de otro, el último de los caracteres que he mencionado (la barra invertida) provoca que el carácter inmediatamente detrás asuma un sentido especial; y a veces la combinación de la barra invertida con ciertos caracteres funciona como si fuera un solo «carácter especial».

Por ello, aunque la terminología normal de Emacs en materia de expresiones regulares habla de "caracteres especiales", a mí me parece más clarificador hablar de "*operadores*"[15] y decir que hay operadores que constan de un sólo carácter y otros que constan de más de un carácter, y que ciertos operadores sólo funcionan en determinados contextos.

Para explicar los operadores lo mejor es, en mi opinión, agruparlos por tipo de tarea realizada:

7.2.1. Operadores básicos de reemplazo

Este grupo de operadores funcionan como una especie de comodín que puede asumir distintos valores, es decir: que puede ser reemplazado por distintos caracteres. Cuando están presentes provocan que la expresión regular pueda coincidir con varias cadenas de texto diferentes.

Los operadores de reemplazo fundamentales son el punto y los corchetes:

El operador punto (.) Es el más general de los operadores de reemplazo: en una expresión regular un punto coincidirá con cualquier carácter excepto el de

[15] Prefiero el término "operadores" al término "*metacaracteres*" que también en ocasiones se utiliza.

nueva línea. Así, por ejemplo, «`a.b`» devolverá cualquier secuencia de tres caracteres que empiece por “a” y termine por “b” salvo aquellas en las que el carácter central sea un salto de nueva línea.

Uso de los corchetes Los corchetes permiten indicar un conjunto de caracteres *alternativos*, cualquiera de los cuales nos interesa. Así, por ejemplo, la expresión «`ca[srm]a`» coincide con “casa”, “cara” y “cama”, pero no con “cana” ni con “capa”.

Algunos caracteres, cuando se incluyen dentro de unos corchetes, tienen un significado especial salvo si se introducen en algún lugar concreto. En particular:

- El guión, dentro de unos corchetes, normalmente se interpreta como “*operador de rango”*, es decir: se asume que el carácter a su izquierda indica el “rango inferior” y el carácter a su derecha el rango superior. Así, por ejemplo, escribir «`[1-8]`» significa que buscamos cualquier carácter en el rango del uno al ocho. O, dicho con otras palabras. La frase «`[1-8]`» es equivalente a la frase «`[12345678]`».

  Los rangos pueden ser también alfabéticos. Así «`[a-z]`» representa cualquier carácter en minúsculas entre la a y la z (OJO: según el alfabeto anglosajón), y «`[A-Z]`» lo mismo, pero referido a letras mayúsculas[16].

  En fin: si lo que queremos no es indicar un rango, sino incluir entre los caracteres que queremos localizar a un guión, entonces este debe colocarse dentro de los corchetes en un lugar en el que quede claro que no está funcionando como un operador de rango, sino como un guión, lo cual se obtiene sólo si el guión es el primero o el último de los caracteres incluidos en los corchetes. Así, «`a-z`» localizará cualquier letra minúscula, pero «`-az`» localizará cualquier guión, cualquier “a” o cualquier “z”

] El carácter de cierre de corchetes se puede incluir dentro de unos corchetes como uno más de los caracteres que se buscan, con la condición de que sea el primero. En otro caso se interpretará como *cierre de los corchetes* y no como un carácter a buscar. Así, por ejemplo, «`b[]a`» es una expresión correcta que buscará alternativamente las secuencias “b]” o “ba”.

^ El carácter utilizado en español para indicar el acento circunflejo, dentro de unos corchetes tiene un significado especial en el caso de que sea el primero. Cuando así sea se interpretará como “*operador de negación*”, es decir: se buscarán las cadenas que contengan *cualquier carácter salvo los que aparecen entre corchetes*. O sea, mientras «`[a-z]`»

[16] Téngase en todo caso en cuenta que la sensibilidad de la expresión regular a la distinción entre mayúsculas o minúsculas, depende de lo que se dijo en la sección 6.3.2.

busca cualquier letra minúscula, «`[^a-z]`» buscará cualquier carácter que no sea una letra minúscula. Por el contrario, el carácter "^" situado en cualquier lugar de los corchetes que no sea el principio se interpreta como un carácter más de los que están dentro de los corchetes.

Como «^» al principio de los corchetes tiene un significado especial, no se considera propiamente un carácter y, en consecuencia, cualquier carácter que vaya detrás de él será tomado como "el primer carácter de los corchetes". Así, por ejemplo, la expresión «`[^]ab]`» se interpretará como cualquier carácter salvo "]", "a" ó "b".

Aparte de esos caracteres, el resto de los caracteres, cuando están entre corchetes, se interpretan siempre como caracteres ordinarios. Es decir: Los caracteres que normalmente funcionan como operadores, dentro de unos corchetes pierden su significado especial y pasan a ser considerados simples caracteres.

7.2.2. Otros operadores de reemplazo

Una de las características de los modos mayores de Emacs es que en cada uno de ellos, a cada carácter se le asigna cierta categoría la cual, además de tener un nombre, tiene un "símbolo" que se usa para representarla. Así, por ejemplo, las letras suelen asignarse a la categoría "caracteres que forman parte de palabras", la cual se representa con la letra "w".

Todas estas asignaciones se hacen en lo que se suele llamar una "tabla de sintaxis". Cada modo mayor tiene su propia tabla de sintaxis, y podemos leerla para comprobar a qué categoría se asigna cada carácter, ejecutando el comando «`C-h s`» («`describe-sintax`»). Este comando nos muestra tres columnas. En la columna izquierda se contienen los caracteres concretos; en la derecha se contiene el nombre de la categoría a la que dicho carácter se asigna, y en la columna central, el símbolo representativo de dicha categoría.

Pues bien: podemos usar los operadores «`\sC`» y «`\SC`» para buscar caracteres que pertenezcan (o no pertenezcan, si usamos el segundo operador) a cierta categoría, teniendo en cuenta que en la sintaxis anterior "C" representa el símbolo identificativo de la categoría.

Así, por ejemplo, si el símbolo representativo de la categoría "puntuación" es un punto, la expresión «`\s.`» localizará cualquier signo de puntuación (o lo que es lo mismo: cualquier carácter asignado a tal categoría en la tabla de sintaxis correspondiente al modo mayor activo), mientras que «`\S.`» localizará cualquier carácter que no esté asignado a tal categoría.

De todas estas "categorías" hay una en la que es tan normal buscar, que existe un operador específico para ella. Se trata de los caracteres "constitutivos de palabras", es decir: que pueden formar parte de una palabra. Para buscar estos caracteres el operador es «`\w`» y para buscar caracteres que no sean constitutivos de palabras se usa el operador «`\W`».

Es decir: el operador «\w» equivale al operador «\sw», mientras que «\W» equivale a «\Sw».

De otro lado, de la misma manera que a cada carácter se le asigna una categoría "sintáctica", también cada carácter pertenece a una segunda categoría relacionada con la página de códigos y conjunto de caracteres a donde pertenece. Para ver estas asignaciones (que funcionan de modo similar a lo que se ha dicho de las tablas sintácticas) se usa el comando «describe-categories».

Pues bien, hay dos operadores que permite buscar caracteres asignados (o no asignados) a alguna de estas otras categorías. Se trata de «\cC» y «\CC», donde, una vez más, la "C" final representa el símbolo identificativo de una de las categorías. Así, por ejemplo, la expresión «\cg», localizará cualquier carácter asignado al grupo de caracteres denominado "caracteres griegos".

7.2.3. Operadores de repetición

Este grupo de operadores indica que el carácter o la expresión que se encuentra a su izquierda debe o puede repetirse cierto número de veces. Así:

***** Significa que el carácter anterior a él (o la expresión regular anterior) puede repetirse tantas veces como sea posible (incluso ninguna). Así por ejemplo la expresión "fo*" coincidirá con "f", "fo", "foo", "fooo", etc.

+ Similar a * salvo en que el carácter anterior debe aparecer al menos una vez. Así "fo" coincidirá con "fo", "foo", etc., pero no con "f".

? Indica que el carácter anterior puede encontrarse una o ninguna vez, pero no más de una. Así "flo?r" coincidirá con "flr" y con "flor", pero no con "floor".

\{N\} Es cierto si el carácter o expresión previo se repite exactamente N veces. Así por ejemplo "x\{4\}" coincidirá con "xxxx" y con nada más.

\{N,M\} Indica repetición entre N y M veces. Es decir: la expresión previa debe repetirse al menos N veces y no más de M veces. Si M se omite entonces no hay límite superior.

En condiciones normales los operadores de repetición se aplican a la letra inmediatamente anterior a ellos. Pero si esta es a su vez otro operador (o un grupo encerrado entre corchetes), la repetición se aplicará al resultado de ese otro operador, lo que hace que cuando se combinan dos operadores de repetición seguidos, se obtengan resultados cuya predicción exige un gran esfuerzo mental; en particular si el segundo carácter de repetición es "?".

Esto último es debido a que los operadores "*", "+" y "?" tienen lo que se podríamos llamar "*naturaleza ávida*" o "*carácter voraz*", es decir intentan extenderse lo

más posible a la derecha; pero al añadirles el operador "?" pierden dicho carácter "ávido" y buscarán la menor coincidencia posible.

Por ejemplo: si tenemos el texto "¡goooooooool!" y buscamos en él con la expresión regular «go+» se nos devolverá como resultado de la búsqueda "gooooooooo", lo que quiere decir que el operador "+" ha buscado el mayor número de repeticiones posibles. Pero si usamos la expresión regular «go+?» el operador final hace que el anterior pierda su carácter ávido y pase a conformarse con la menor coincidencia posible, es decir: se conformará con "go" y será ese texto el que se pase al siguiente operador "?" que busca ninguna o una repetición del carácter a su izquierda, la "o", y como está repetida, se devolverá como resultado "goo".

7.2.4. Operadores que indican posición del texto buscado

Este grupo de operadores nos permiten indicar en qué lugar debe encontrarse el texto que buscamos. En ciertos contextos a estos operadores se les denomina «*anclas*» (*anchors*).

Al respecto disponemos de las siguientes posibilidades:

Carácter	Busca sólo en	Se coloca
^	Principio de línea	Precediendo a la expresión
$	Final de línea	Al final de la expresión
\‘	Principio del *buffer*	Precediendo a la expresión
\’	Final del *buffer*	Al final de la expresión
\b	Principio de palabra	Precediendo a la expresión
\<	Principio de palabra	Precediendo a la expresión
\b	Final de palabra	Al final de la expresión
\>	Final de palabra	Al final de la expresión
\B	No principio de palabra	Precediendo a la expresión
\B	No final de palabra	Al final de la expresión

En la tabla anterior, cuando he escrito "precediendo a la expresión" o "Al final de la expresión", lo que pretendo decir, en el primer caso, es que el operador que sea debe estar al principio de la expresión regular y en el segundo caso lo contrario. Así, por ejemplo, «`adios$`» localizará todas las apariciones del texto "adios" que se encuentren en final de línea.

Se observará, por otra parte, que en lo relativo a las palabras hay dos juegos de instrucciones: «`\b`» y «`\<`» o «`\>`». La diferencia entre unas y otras se encuentra en cómo actúan cuando la aparición se produce al principio o al final del *buffer*. «`\<`» y «`\>`» sólo consideran que hay coincidencias al principio o al final del *buffer* si el carácter que sigue o que precede a la cadena buscada entra dentro de los que coincidirían con el operador de reemplazo «`\w`» (véase la sección 7.2.2).

7.3. Incluir, como carácter de búsqueda, el identificativo de un operador

Ya hemos visto que los caracteres especiales son en realidad operadores y se interpretan como tales. Ahora bien: como además de operadores se trata de caracteres, en alguna ocasión nos puede interesar buscar una cadena de texto que contenga alguno de ellos. Para lograrlo podemos usar dos procedimientos alternativos:

- Colocar el carácter especial en un contexto en el que no tenga sentido su uso como operador, en cuyo caso Emacs interpretará que el uso que se hace de ese carácter es como tal y no como operador.
- Usar el operador barra invertida antes del carácter en cuestión.

Un ejemplo del primer procedimiento lo hemos tenido cuando explicaba el uso de los corchetes. Allí dije que un guión se interpreta como operador de rango, salvo si es el primer o el último carácter de los corchetes, ya que en tal caso es evidente que ahí el guión no indica ningún rango. De la misma manera, si usamos los operadores de repetición (*+?) como primer carácter de la cadena de búsqueda, al no haber ninguna letra a su izquierda, es evidente que no pueden estar siendo usados como operadores de repetición.

En cuanto al segundo procedimiento, es el más corriente: El operador barra invertida se aplica al carácter que se encuentre a su derecha, y provoca que ese carácter se interprete de forma literal (OJO, esto se refiere al operador barra invertida en sí mismo considerado, no a los operadores que constan de más de un carácter, el primero de los cuales es una barra invertida).

Así, por ejemplo, la cadena «\$\\» provocará que se busque la secuencia «$\».

Por otra parte, como poner una barra invertida a un carácter normal no lo convierte en especial (salvo los casos ya vistos de operadores que constan de más de un carácter), lo mejor, en caso de duda, es hacer uso de la barra invertida, salvo cuando estemos dentro de unos corchetes, en donde, como ya sabemos, los caracteres especiales se convierten en normales, por lo que la propia barra invertida, deja de ser especial.

7.4. Combinar expresiones regulares

Podemos combinar dos o más expresiones regulares creando una expresión nueva la cual puede buscar, bien cadenas de texto que coincidan simultáneamente con las dos expresiones constituyentes, bien cadenas de texto que coincidan con cualquiera de las expresiones constituyentes.

Para la primera posibilidad no hay sintaxis especial, basta con escribir una expresión regular detrás de la otra. Así, por ejemplo, si tenemos la expresión regular «`ho`» y la combinamos con «`la`», generamos una expresión regular nueva que es «`hola`».

Esto parece trivial y lo es.

Ahora bien: para combinar dos o más expresiones regulares de tal manera que la expresión resultante sea cierta cuando coincida con cualquiera de las previas, aunque no coincida con todas ellas, debemos usar el operador «`\|`».

Este operador podríamos traducirlo por "*operador OR lógico*", y significa que debe coincidir bien la expresión a su izquierda, bien la expresión a su derecha. Si esas expresiones no son simples, debemos usar los caracteres de agrupamiento. Y así, por ejemplo: «`\(foo\|bar\)x`» encontrará "foox" o "barx".

7.5. Operadores de agrupación

La pareja de operadores «`\(`» y «`\)`» funcionan como operadores de agrupación, es decir: todo lo que haya entre ellos se interpreta como un grupo.

En las expresiones regulares de Emacs los grupos se emplean para tres finalidades:

1. Hacer que varios caracteres sean tratados como uno solo a efectos de aplicarles algún operador de repetición. Así, por ejemplo «`Bana\(na\)*`» localizará "Bana", "Banana", "Bananana", etc.

2. Encerrar dentro de ellos varias expresiones alternativas separadas por el operador «`\|`».

3. Definir la secuencia encerrada dentro del grupo y asignarle una denominación mediante la que nos podamos referir a ella más tarde.

Los dos primeros casos son bastante claros y no requieren mayor explicación. Me ocupo ahora del tercero. En él el término "*agrupar*" se usa en un sentido distinto a los anteriores, ya que lo que pretendemos es almacenar en una especie de registro el resultado del grupo, para poderlo usar más tarde. Los registros se llaman 1, 2, 3, ...9. En una expresión regular no se pueden usar más de nueve grupos con esta finalidad.

Es decir: cuando Emacs encuentra un grupo cuya finalidad no es ninguna de las dos primeras finalidades de los grupos, lo numera y asigna a dicho número el resultado que ese grupo arroje, de modo que más adelante, en la misma expresión

regular (o en otra expresión relacionada con ella), podemos incluir *ese mismo resultado* mediante el operador «\N», donde "N" representa un número de grupo, contado de izquierda a derecha, para el caso de que en la misma expresión regular haya más de un grupo.

Es decir:«\N» se interpreta como "insertar aquí el mismo texto que devolvió el grupo número N". Esto no es lo mismo que volver a escribir el grupo propiamente dicho, porque no se trata de utilizar por segunda vez la misma expresión regular, sino el resultado que dicha expresión ha arrojado.

Así, por ejemplo, las siguientes dos expresiones regulares:

```
\(am.r\) +\1
\(am.r\) +\(am.r\)
```

no son equivalentes. La primera exige que la frase acabe exactamente igual que empieza, y la segunda no. Si en nuestro texto tenemos la cadena "amor amar", la segunda expresión regular la localizará, y la primera no, pues aunque tanto "amor" como "amar" coinciden con la expresión regular «am.r», en el primer caso se exige que se coincidencia al final de la frase sea la misma que la del principio.

El inconveniente de lo anterior es que como los grupos se pueden usar para varias finalidades, sólo se considerará que se usa para esta finalidad un grupo cuando sea obvio que no se usa para cualquiera de las otras dos. Es decir: un grupo que vaya seguido de un operador de repetición, o que contenga el operador «\|» no podrá emplearse de esta manera.

En estos casos podemos empezar el contenido del grupo con los caracteres "?:" que indican que queremos que ese grupo se use para hacer después referencia a su resultado.

Un uso muy corriente de este tipo de agrupamiento se da en las operaciones de reemplazo basadas en expresiones regulares, que son materia del próximo epígrafe.

7.6. Operaciones de reemplazo basadas en expresiones regulares

Una operación de reemplazo en realidad se traduce en dos operaciones distintas: una primera consistente en la búsqueda de cierto texto, y otra consistente en la sustitución del texto una vez localizado por una segunda cadena de texto. Es evidente que para la primera parte podemos usar una expresión regular. Pero cabe preguntarse si la segunda cadena de texto debe necesariamente ser constante.

Es obvio que en la cadena de sustitución no puede usarse una expresión regular en el mismo sentido en el que hasta ahora nos venimos refiriendo a ellas, porque la

expresión regular ofrece muchos resultados posibles, y en el momento de sustituir texto hay que saber *qué texto* escribir, es decir: necesitamos un texto constante y no un patrón de texto.

Pero lo que sí podemos hacer es usar en nuestro texto de sustitución algunas partes variables cuyo resultado dependa del resultado que haya ofrecido la expresión regular con la que se localizó el texto a cambiar.

En concreto podemos usar:

\& Representa todo el texto original que será sustituido.

\N Representa el texto devuelto por el grupo número N de la expresión regular usada para localizar el texto a sustituir (véase el epígrafe anterior).

Así, por ejemplo, la expresión

```
M-x replace-regexp <RET> coches*  <RET> guarda-\& <RET>
```

reemplazará “coche” y “coches” por “guarda-coche” y “guarda-coches” respectivamente, mientras que

```
M-x replace-regexp <RET> coche\(s*\)  <RET> vehículo\1 <RET>
```

cambiará todas las apariciones de “coche”y “coches” por “vehículo” y “vehículos” respectivamente.

Pero el uso de esa posibilidad no significa que la cadena de reemplazo sea una expresión regular. No lo es y, por ello, podemos utilizar en ella cualquiera de los operadores de una expresión regular con la seguridad de que será interpretado como un carácter, salvo el operador barra invertida, que sí deberá ser precedido de otra barra invertida para ser utilizado como carácter.

8. GNU Free Documentation License (Licencia GNU para Documentación Libre)

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free"in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of çopyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as τ**ou"**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **"Modified Version"** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **"Secondary Section"** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with

the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **"Invariant Sections"** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **Çover Texts"** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent"is called **.Opaque"**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **"Title Page"** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page"means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **.Acknowledgements"**, **"Dedications"**, **"Endorse-**

ments", or **"History"**.) To **"Preserve the Title"** of such a section when you modify the Document means that it remains a section "Entitled XYZ.ªccording to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with

each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History"in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History"section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled .Acknowledgements.or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements.or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History"in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled .Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into

the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an .aggregate"if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled .Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your

rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License .ºr any later version.ªpplies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright ©YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts."line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.