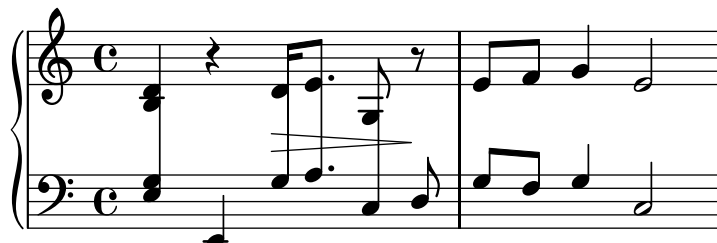


## Funcionalidades nuevas de la versión 2.16 desde la 2.14

- Se contemplan las plicas de pentagrama cruzado en acordes, utilizando `crossStaff` y el grabador `Span_stem_engraver`. Éste efectúa el cálculo de la longitud de las plicas de pentagrama cruzado de forma automática.



- La sintaxis de las palabras (secuencias de caracteres que se reconocen sin encerrarlas entre comillas) y las instrucciones (que ahora son siempre una barra invertida ‘\’ seguida de una palabra) se ha unificado para todos los modos: ahora consiste en caracteres alfabéticos, posiblemente comprendiendo guiones aislados ‘-’ y guiones bajos ‘\_’.

Una consecuencia es que la utilización de guiones de texto sin entrecomillar como (literalmente)

```
{ c-script c\f_script }
```

ahora tienden a producir una música no válida. La omisión de las comillas para texto arbitrario en lugar de palabras clave nunca ha sido una buena práctica o se ha documentado, y es poco probable que se haya utilizado mucho.

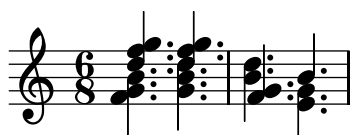
Quedarse con las convenciones establecidas (como no utilizar guiones o guiones bajos para los nombres de instrucción orientados a ser utilizados dentro de la música) sigue siendo recomendable. El motivo de este cambio es el reconocimiento más robusto de las unidades léxicas de LilyPond para sí mismo así como para las herramientas externas que interpretan su sintaxis.

- Se contempla el canto kievano en notación cuadrada:

```
\new KievianVoice {  
  \cadenzaOn  
  c d e f g a bes  
  \bar "kievan"  
}
```



- Los puntillos ahora evitan a las otras voces en la polifonía a dos partes, de manera que los usuarios pueden trasladar el grabador `Dot_column_engraver` para ajustar los puntillos de forma independiente para cada voz.



- Ahora se incluye en LilyPond una función de Scheme, desarrollada por varios usuarios, para ajustar los puntos de control de las curvas como ligaduras de unión y de expresión.

```
g8->( bes,-.) d4
\shape Slur #'((-0.5 . 1.5) (-3 . 0) (0 . 0) (0 . 0))
g8->( bes,!-.) d4
```



- El uso de las especificaciones de `\tempo` en los bloques `\midi` (eliminados en la versión 2.9.16 en favor del ajuste explícito de `tempoWholesPerMinute`) ha resurgido: ahora cualquier clase de música que efectúe ajustes de propiedades se convierte en definiciones de contexto dentro de las especificaciones de salida, permitiendo declaraciones como

```
\layout { \accidentalStyle modern }
\midi { \tempo 4. = 66 }
```

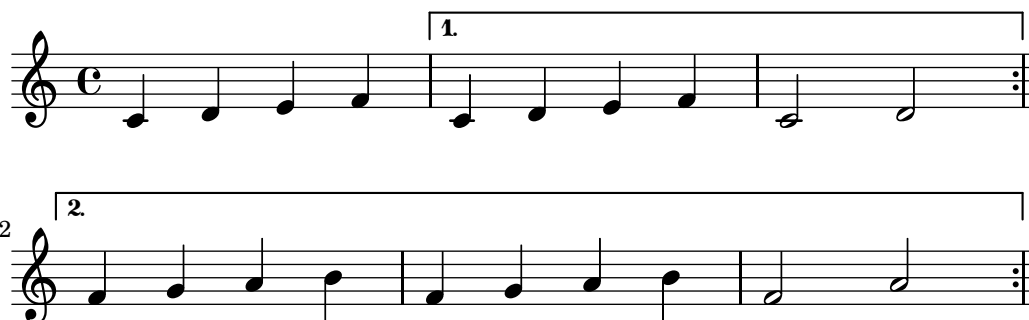
- Se ha rediseñado la clave de Sol de LilyPond: el bucle superior está más equilibrado, el gancho inferior sobresale menos y la línea vertical principal ("spine") está curvada de forma más regular. Pueden compararse las versiones vieja y nueva consultando la documentación: [versión antigua](#), [versión nueva](#).
- Se han simplificado las instrucciones de los sellos de elementos gráficos para permitir una menor duplicación de código y mejores aproximaciones de altura de los objetos gráficos. Se han eliminado las siguientes instrucciones de sello:

- beam
- bezier-sandwich
- bracket
- dashed-slur
- dot
- oval
- repeat-slash
- zigzag-line

- Ahora se tratan los corchetes como objetos diferenciados y no como parte de la plica.



- Se puede elegir entre dos métodos de numeración de compases, en especial para cuando se emplean repeticiones:





- Lo que sigue es un cambio fundamental en la representación que LilyPond hace de la música: los eventos de duración como `LyricEvent` y `NoteEvent` ya no se encuentran envueltos dentro de elementos `EventChord` a no ser que se hayan escrito realmente como parte de un acorde. Si manipulamos expresiones musicales en Scheme, el nuevo comportamiento puede necesitar cambios en nuestro código. Las llamadas a la función musical `\eventChords` o a la función de Scheme `event-chord-wrap!` convierten a la representación anterior; la utilización de una cualquiera de ellas puede ser la vía más sencilla para mantener operativo el código tradicional.

Los tres siguientes elementos de la lista son consecuencia de este cambio.

- Se ha reimplementado la abreviatura de ayuda para intrucción repetitiva de acordes, `q`. Ahora los acordes repetidos se sustituyen justo antes de la interpretación de la expresión musical. En caso de que el usuario quiera retener ciertos eventos del acorde original, puede ejecutar manualmente la función `\chordRepeats` de sustitución de acordes de repetición.
- Los números de cuerdas y digitaciones de la mano derecha sobre notas individuales aparecen ahora sin tener que escribirlas dentro de corchetes de acorde.
- Ahora las funciones musicales funcionan igual cuando se usan dentro o fuera de los acordes, incluidas todas las posibilidades de la exploración de argumentos. Las variables musicales se pueden usar dentro de acordes: una construcción como

```
tonic=fis'
{ <\tonic \transpose c g \tonic> }
```



ahora funciona como se espera. Puede utilizarse `#{...#}` para la construcción de componentes de acordes. Ahora `\tweak` funciona sobre notas individuales sin necesidad de incluirlas dentro de un acorde. Ahora es posible usarla dentro de eventos de instrucciones y letra de canciones, pero aún no es probable que ofrezca resultados.

- `\tweak` toma ahora opcionalmente la especificación de un objeto de presentación. Se puede usar para trucar objetos de presentación que están causados sólo indirectamente por el evento trucado, como alteraciones accidentales, plicas y corchetes:

```
<\tweak Accidental #'color #red cis4
\tweak Accidental #'color #green es
g>
```



- Las expresiones de Scheme dentro de fragmentos de código de LilyPond incrustados (`{...#}`) se ejecutan ahora dentro de la cerradura léxica del código de Scheme circundante. El símbolo `$` ya no es especial dentro del código de LilyPond incrustado. Se puede utilizar de forma incondicional dentro de código de LilyPond para la evaluación inmediata de expresiones de Scheme, de forma parecida a la forma en que se utilizaba anteriormente `ly:export`. Se ha suprimido `ly:export`. Como consecuencia, ahora `#` está libre para diferir la evaluación de su argumento hasta que el analizador sintáctico reduzca efectivamente la expresión contenida, reduciendo significativamente el potencial de la evaluación prematura. También están los operadores de ‘división de cadenas’ `$@` y `#@` para la interpretación de los miembros de una lista de forma individual.
- Para reducir la necesidad de utilizar `$`, las expresiones de Scheme escritas con `#` se interpretan como música dentro de las listas de música, y como elementos de marcado o de listas de marcado dentro de los elementos de marcado.
- Se ha mejorado el soporte de acordes de tipo jazz: se reconocen los acordes lidios y alterados; ahora se tratan los separadores entre modificadores de acorde de forma independiente de los separadores entre acordes invertidos y sus notas de bajo (y por omisión, la barra inclinada se usa ahora solamente para el último tipo de separador); las notas adicionales ya no van prefijadas por "add" de forma predeterminada; y la "m" en los acordes menores se puede personalizar. Consulte [Sección “Nombres de acorde personalizados” in Referencia de la Notación](#) para más información.
- Se ha cambiado el nombre de la instrucción `\markuplines` por `\markuplist` para conseguir una mejor correspondencia con su semántica y con la nomenclatura general de LilyPond.
- Se ha simplificado considerablemente la interfaz para especificar afinaciones en las tablaturas y se emplea la función de Scheme `\stringTuning` para la mayor parte de los propósitos.
- Las barras ahora pueden preservar la inclinación por encima de los saltos de línea.



Para hacerlo, se han hecho obsoletas varias funciones de "callback".

- `ly:beam::calc-least-squares-positions`
- `ly:beam::slope-damping`
- `ly:beam::shift-region-to-valid`

Además, `ly:beam::quanting` ahora acepta un argumento adicional para ayudar a los cálculos sobre los cambios de línea. Todas estas funciones se llaman automáticamente cuando se ajusta el parámetro `positions`.

- En los argumentos de función, la música, los elementos de marcado y las expresiones de Scheme (así como algunas otras entidades sintácticas) se han hecho mayormente intercambiables y se diferencian solamente mediante la evaluación del predicado respectivo. En ciertos casos, el analizador sintáctico consulta este predicado, como cuando se decide si interpretar `-3` como un número o como un evento de digitación.
- Ahora se pueden definir las funciones musicales (y sus parientes cercanos) con argumentos opcionales.
- Para definir instrucciones que se ejecutan solamente por sus efectos secundarios, ahora está disponible `define-void-function`.
- Hay una instrucción nueva `define-event-function` en analogía con `define-music-function` que se puede usar para definir funciones musicales que actúan como post-eventos sin que se requiera un especificador de dirección como `(-, ^ o _)` antes de ellos.

```
dyn=#(define-event-function (parser location arg) (markup?)
      (make-dynamic-script arg))
\relative c' { c\dyn pfsss }
```



- Se puede incluir una lista de alias en ASCII para caracteres especiales.

```
\paper {
  #(include-special-characters)
}
\markup "&bull; &dagger; &copyright; &OE; &ss; &para;"
```

• † © Œ ß ¶

- Hay una instrucción nueva `define-scheme-function` en analogía con `define-music-function` que puede usarse para definir funciones que se evalúan a expresiones de Scheme pero aceptan argumentos en la sintaxis de LilyPond.
- Ahora se puede utilizar la construcción `#{ ... #}` no solo para crear listas secuenciales de música, sino también para alturas (que se distinguen de los eventos de nota sencillos por la ausencia de duración u otra información que no puede formar parte de una altura), eventos musicales únicos, expresiones musicales vacías, post-eventos, elementos de marcado (sobre todo para liberar a los usuarios de la necesidad de usar la macro `markup`), listas de marcado, expresiones numéricas, definiciones y modificaciones de contextos y algunas otras cosas. Si no contiene nada o contiene un único evento musical, ya no devuelve una lista secuencial de música, sino una expresión musical vacía o simplemente el propio evento musical, respectivamente.
- Se pueden usar alturas en la parte derecha de las asignaciones. Las alturas se diferencian de los eventos de una sola nota en que no tienen duración ni otras informaciones que no pueden formar parte de una altura.

- Nueva opción de la línea de órdenes ‘`--loglevel=level`’ para controlar el volumen de datos que LilyPond produce en la salida. Los valores posibles son ERROR (errores), WARN (advertencias), BASIC\_PROGRESS (progreso básico), PROGRESS (progreso) y DEBUG (depuración).
- `\once \set` ahora reinicia correctamente el valor de la propiedad al valor previo.



- La alineación de los elementos de matiz dinámico extensos (reguladores, crescendi textuales, etc.) se divide automáticamente si se da explícitamente una dirección distinta.



- Ahora las apoyaturas y mordentes funcionan también dentro de una ligadura de expresión, y no solo dentro de una ligadura de fraseo. Asimismo, se ha añadido la función `\slashedGrace` que no imprime ninguna ligadura partiendo de la nota del mordente.



- Para suprimir a línea en un elemento de crescendo extenso (y otros elementos extensos similares), LilyPond contempla ahora de forma plena la propiedad `#'style = #'none`.



- LilyPond.app está disponible ahora para MacOS X 10.7. ¡Gracias, Christian Hitz!
- Los glissandos pueden abarcar varias líneas.