LilyPond

Le système de gravure musicale

Nouveautés

L'équipe de développement de LilyPond

Ce document recense les modifications et les nouvelles fonctionnalités de LilyPond pour la version 2.24.1 (depuis la 2.22).

Pour connaître la place qu'occupe ce manuel dans la documentation, consultez la page Section "Manuels" dans *Informations générales*.

Si vous ne disposez pas de certains manuels, la documentation complète se trouve sur `https://lilypond.org/`.

Pour LilyPond version 2.24.1

Note : Chaque nouvelle version de LilyPond peut comporter des changements de syntaxe, ce qui peut exiger de modifier les fichiers que vous avez écrits avec des vesrions précédentes, de telle sorte qu'ils restent fonctionnels avec les nouvelles versions. Afin de mettre à jour vos fichiers, il est **fortement conseillé** d'utiliser l'utilitaire `convert-ly` distribué avec LilyPond et qui est abordé dans Voir Section "Mise à jour avec convert-ly" dans *Utilisation des programmes*. `convert-ly` est capable de réaliser la plupart des modifications de syntaxe automatiquement. Les utilisateurs de Frescobaldi peuvent lancer `convert-ly` directement à partir du menu de Frescobaldi en faisant « Outils > Mettre à jour avec convert-ly. . . ». D'autres environnements prenant en charge LilyPond sont susceptibles de fournir un moyen graphique de lancer `convert-ly`.

Modifications majeures de LilyPond

- LilyPond requiert désormais Guile 2.2. Même si vous n'êtes pas personnellement l'auteur de code Scheme, vous pourriez avoir recours à des bibliothèques de personnalisation évoluée. Si elles venaient à ne pas fonctionner avec la version 2.24.1 de LilyPond, veuillez vous adresser aux développeurs de ces bibliothèques. Si vous êtes vous-même développeur de bibliothèques Scheme, veuillez lire [Notes sur Guile 2.2], page 21, ci-dessous.
- L'infrastructure permettant de générer les binaires officiels a été complètement réécrite, de concert avec la bascule sur Guile 2.2. Nous fournissons, à compter de cette version, des binaires 64 bits pour macOS et Windows. Par ailleurs, tous les paquetages sont disponibles sous forme de simples archives qui peuvent être dépliées dans n'importe quel endroit pour « installation ». Il suffira, pour désinstaller, de simplement supprimer le dossier en question. Est aussi abandonné l'éditeur de base, LilyPad, qui était installé sur macOS et Windows. Nous recommandons l'utilisation de solutions externes telles que l'éditeur bien connu Frescobaldi (`https://frescobaldi.org`) ou l'un de ceux mentionnés dans Section "Facilités d'édition" dans *Informations générales*. Pour de plus amples informations, veuillez consulter les instructions détaillées dans Section "Installation" dans *Manuel d'initiation*.

  Dans les paquetages officiels, Ghostscript ne contient plus son propre jeu de polices par défaut, ceci afin de réduire la taille des archives à télécharger. LilyPond n'a pas besoin de ces polices (l'application contient elle-même les polices dont elle a besoin). Cependant, quelques utilisateurs y faisaient appel avec des instructions comme `/Arial findfont` dans du code PostScript inséré avec `\markup \postscript`. Il est recommandé d'utiliser la syntaxe `\markup` normale pour le texte plutôt que du code PostScript écrit à la main.

Problèmes connus de LilyPond 2.24.1

Sur Windows, la compilation de partitions volumineuses (quelques centaines de pages) peut entraîner un crash.[1] Nous espérons résoudre ce problème dans une future version 2.24.x.

Notes à propos de la compilation des sources et à l'attention des empaqueteurs

La présente section s'adresse aux entousiastes désirant compiler LilyPond à partir des sources et aux empacteurs qui préparent LilyPond pour les différentes distributions. Si vous ne faites partie d'aucun de ces groupes, vous pouvez passer cette section.

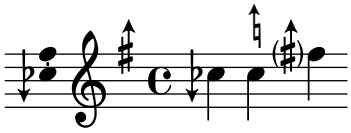
- Comme annoncé plus haut, LilyPond requiert désormais Guile 2.2. Selon les exigences de la distribution, on peut aussi compiler avec Guile 3.0 en passant `GUILE_FLAVOR=guile-3.0` au script `configure`. Ceci n'est toutefois pas recommandé et n'est pas officiellement pris en charge.
- L'évaluateur de code Scheme est plus lent avec Guile 2.2 qu'il ne l'était avec Guile 1.x. Pour atténuer cette baisse de performance, nous recommandons de compiler les fichiers `.scm` en bytecode ajoutant `make bytecode` à la compilation, puis `make install-bytecode` en plus de `make install`.
- À compte de cette version stable, le système de construction de LilyPond n'installe plus de fonte textuelle. Nous vous remercions d'en fournir en tant que paquetages indépendants, tout en vérifiant leur licence et les fichiers de notice.

[1] Il est à noter que d'imposantes partitions pouvaient déjà crasher avec LilyPond 2.22 en raison de restrictions inhérentes à l'architecture 32 bits.

Nouveautés en matière de notation musicale

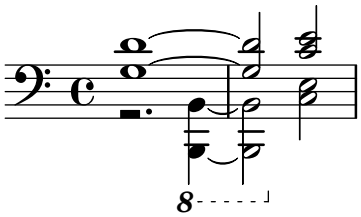
Améliorations de la représentation des hauteurs

- La prise en charge d'altérations alternatives s'améliore. Grâce à la propriété `alterationGlyphs` des contextes de niveau portée, peuvent se définir globalement les glyphes à utiliser pour tous les objets – voir Section "Glyphes d'altération alternatifs" dans *Manuel de notation*.



- Les crochets d'octaviation peuvent s'appliquer à une seule voix plutôt qu'à l'intégralité de la portée. Ceci demandait auparavant quelques circonvolutions.

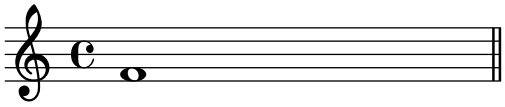
```
\layout {
  \context {
    \Staff
    \remove Ottava_spanner_engraver
  }
  \context {
    \Voice
    \consists Ottava_spanner_engraver
  }
}
```

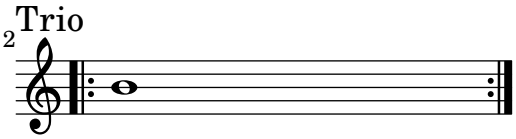


Améliorations en matière de rythme

- La nouvelle commande `\section` insère une double barre qui interagit de manière optimale avec les barres de reprise. Un passage peut être nommé à l'aide de la commande `\sectionLabel`.

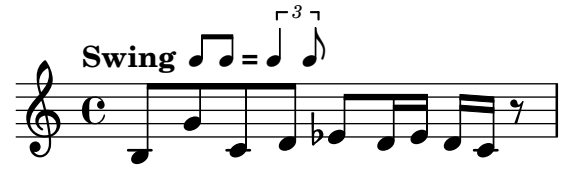
```
\fixed c' {
  f1
  \break
  \section
  \sectionLabel "Trio"
  \repeat volta 2 {
    b1
  }
}
```





- `\numericTimeSignature` et `\defaultTimeSignature` s'appliquent désormais en même temps à toutes les portées – plus précisément à toutes les portées d'un même contexte `Timing` – afin de correspondre au comportement de `\time`.
- La commande de *markup* `\rhythm` fait son apparition. Il s'agit d'un moyen simple de mélanger du texte et du rythme comme, par exemple, pour afficher une indication de « swing ».

```
\relative {
  \tempo \markup {
    Swing
    \hspace #0.4
    \rhythm { 8[ 8] } = \rhythm { \tuplet 3/2 { 4 8 } }
  }
  b8 g' c, d ees d16 ees d c r8
}
```



- La commande `\enablePolymeter` agit comme un raccourci lors de la saisie pour déplacer les graveurs selon le besoin et permettre d'avoir différentes métriques en parallèle. Le code :

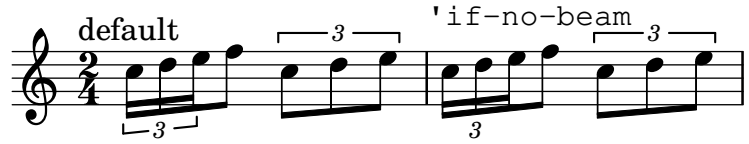
```
\layout {
  \context {
    \Score
    \remove Timing_translator
    \remove Default_bar_line_engraver
  }
  \context {
    \Staff
    \consists Timing_translator
    \consists Default_bar_line_engraver
  }
}
```

  peut ainsi s'abréger en :

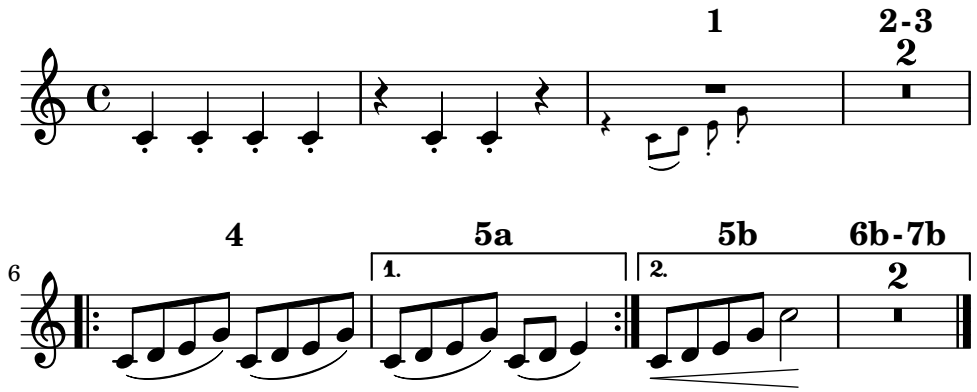
```
\layout {
  \enablePolymeter
}
```

  Par ailleurs, le `Default_bar_line_engraver` a été supprimé.

- La nouvelle option `visible-over-note-heads` permet aux crochets de n-olets de toujours s'afficher lorsqu'ils sont placés du côté des têtes de note. Cette option peut s'utiliser avec le style par défaut de visibilité des crochets de n-olet ou avec `#'if-no-beam`.

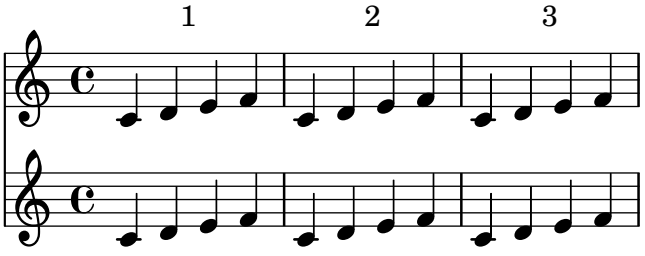


- Les compteurs de mesures prennent en considération les mesures à compter et les alternatives.

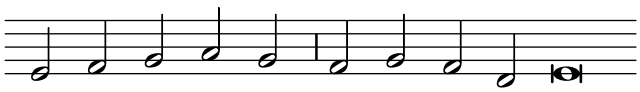


- Les numéros de mesure peuvent se présenter au centre de la mesure comme il est d'usage dans les musiques de film.

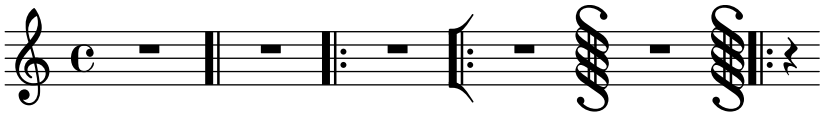
```
\layout {
  \context {
    \Score
    centerBarNumbers = ##t
    barNumberVisibility = #all-bar-numbers-visible
  }
}

<<
  { \repeat unfold 3 { c'4 d' e' f' } }
  { \repeat unfold 3 { c'4 d' e' f' } }
>>
```



- L'alignement des numéros de mesure présents en cours ou en fin de ligne s'effectue désormais sur leur extrémité gauche. Il s'agit là de suivre la recommandation d'Elaine Gould (*Behind Bars*, p. 237) et le consensus entre les développeurs lors de la discussion de ce problème. Aucun changement n'intervient aux numéros portés en début de ligne.
- L'instruction `\bar ","` crée une barre de mesure raccourcie.



- Les types de barre prédéfinie qui suivent n'apparaissent plus sous forme de simple barre lorsqu'ils surviennent en fin de ligne. Des types de barre annotée, tel que `\bar "S-|"`, ont été ajoutés en conséquence.



- L'instruction `\bar ""` n'est plus un préalable à l'affichage du numéro de la première mesure. Il suffit désormais de régler la propriété `barNumberVisibility` à `all-bar-numbers-visible` ou l'un des autres réglages pour lesquels le premier numéro de mesure est visible.

  Notez bien qu'il s'agit d'un changement de comportement pour les partitions dans lesquelles `barNumberVisibility` est réglé à `all-bar-numbers-visible` ou équivalent et `BarNumber.break-visibility` activé sans avoir de `\bar ""`. Un numéro de mesure est désormais affiché au début. Il s'agit bien du comportement attendu (**tous** les numéros devraient être visibles) mais, en raison d'une documentation probablement pas assez explicite, certains utilisateurs ont opté pour ces réglages afin d'afficher les numéros de mesure en cours de portée sauf pour la première mesure. En pareil cas, il convient de tout simplement supprimer la clause `\set Score.barNumberVisibility = #all-bar-numbers-visible` puisque `\override BarNumber.break-visibility = ##t` est le réglage approprié et suffisant.

- Désormais, la commande `\break` insère toujours un saut, outrepassant toutes les décisions par défaut quant aux points de rupture. Par exemple, il n'est désormais plus nécessaire d'ajouter `\bar ""` pour obtenir une rupture en cours de mesure.

  La nouvelle commande `\allowBreak` insère un point de rupture potentiel, sans le forcer, mais outrepasse les décisions par défaut à l'instar de `\break`.

- Le type de barre de mesure `"-"` a été supprimé. `convert-ly` le convertit en `""`. Il en résulte une légère différence dans l'espacement horizontal lors d'un saut de ligne.

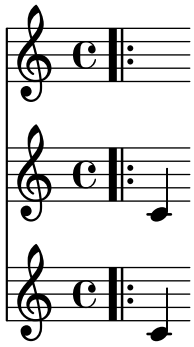
- `automaticBars` a été supprimé. `convert-ly` convertit `automaticBars = ##f` en `measureBarType = #'()`.

- Au lieu de répéter le nom de glyphe en cours de ligne, `\defineBarLine` accepte désormais la valeur `#t`.

- Le `Bar_engraver` interdisait jusqu'à présent un saut de ligne entre deux barres de mesure ; il ne le fait dorénavant que lorsque la propriété de contexte `forbidBreakBetweenBarLines` est activée (réglée à `#t`), ce qui est le cas par défaut.

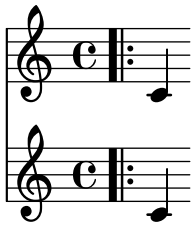
- En raison de modifications dans le fonctionnement interne de l'instruction `\bar`, il n'est désormais plus possible de l'instancier avant la création de contextes inférieurs par un `\new` sous peine de créer une portée intempestive comme lors de l'utilisation de commandes telles que `\override Staff...` (voir Voir Section "Apparition d'une portée supplémentaire" dans *Utilisation des programmes*).

```
{
  \bar ".|:"
  <<
    \new Staff { c' }
    \new Staff { c' }
  >>
}
```

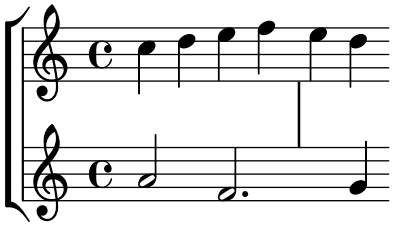


La solution consiste à placer le `\bar` au sein même de la musique dans chaque portée, comme c'est généralement le cas pour la plupart des commandes.

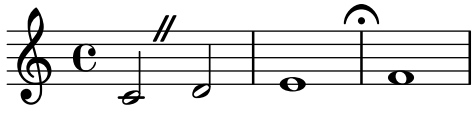
```
<<
  \new Staff { \bar ".|:" c' }
  \new Staff { \bar ".|:" c' }
>>
```



- Le type de barre `"-span|"` crée un *mensurstrich* – impression des lignes de mensuration.



- Les contextes `Staff` utilisent désormais le nouveau `Caesura_engraver` pour réaliser la commande `\caesura`.

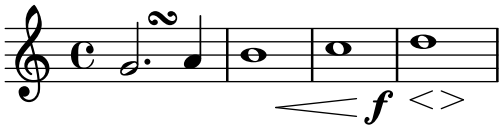


Améliorations en matière d'expressivité

- Les événements attachés à des notes, tels que nuances ou articulations, peuvent être différés d'une durée arbitraire à l'aide de `\after`. Ceci permet de simplifier de nombreuses situations qui jusqu'alors demandaient de recourir à une polyphonie et des silences invisibles.

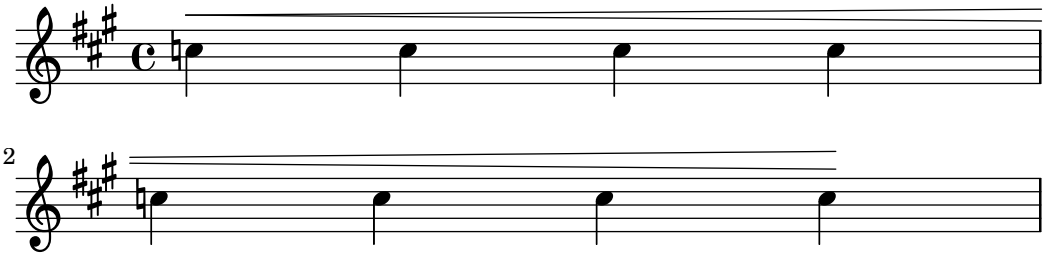
```
{
  \after 2 \turn g'2. a'4
  \after 2 \< b'1
  \after 2. \f c''
  <>\< \after 4 \> \after 2\! d''
```

```
}
```

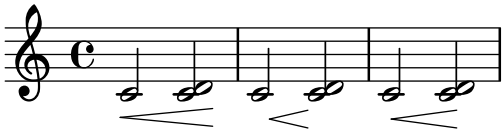


- La deuxième portion des soufflets tronqués bénéficie maintenant d'un rembourrage à gauche. Ceci est conforme aux canons de la gravure et règle le problème de décallage vertical qui pouvait intervenir en raison de l'armure.

```
\relative {
  \key a \major
  c''4^\< c c c \break c c c c\! |
}
```

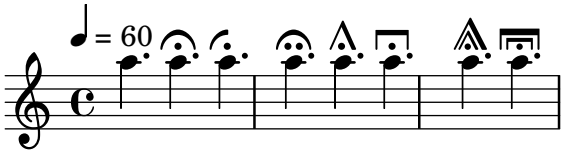


- Les terminaisons des soufflets peuvent désormais s'aligner sur la gauche, au centre ou sur la droite des *grobs* `NoteColumn` à l'aide d'une dérogation à la propriété `endpoint-alignments`.

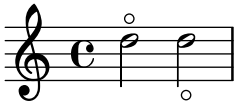


- Le positionnement des extensions de trille peut désormais se spécifier comme pour toute indication d'articulation, autrement dit à l'aide de `_\startTrillSpan` ou `^\startTrillSpan`.
- L'apparence des extensions de trille a été modifiée pour mieux correspondre aux conventions classiques de la gravure. L'extension s'arrête désormais juste avant la note qui suit, sans la chevaucher. Ce sera juste avant l'altération si la note qui suit en est pourvue, et au niveau de la barre si la prochaine note est dans la mesure suivante.

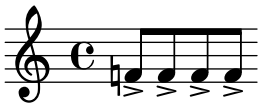
- Le décalage par défaut des points d'orgue a été élargi. Ceci permet de pallier certains cas où, en présence d'une note pointée, ils étaient trop proches des points et des autres objets.



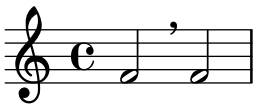
- Le symbole de flageolet est désormais plus petit et un peu plus épais. Ceci correspond plus aux partitions que l'on peut trouver et rend inutile la retouche `\tweak font-size -3 \flageolet` jusqu'ici recommandée.



- Le glyphe d'accent est un peu plus petit qu'auparavant. Ceci permet de pallier certains cas où, par exemple, la présence d'un bécarre aurait déplacé l'accent verticalement.

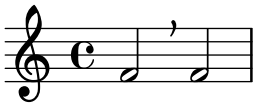


- Le glyphe de virgule stylisé, tel que celui utilisé par la commande `\breathe`, adopte une forme plus habituelle.



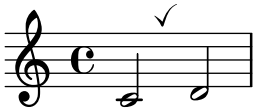
L'ancienne forme reste accessible sous le nom de « raltcomma » :

```
{
  \override BreathingSign.text =
    \markup { \musicglyph "scripts.raltcomma" }
  f'2 \breathe f' |
}
```



- La nouvelle propriété de contexte `breathMarkType` sélectionne le marqueur produit par la commande `\breathe` parmi plusieurs types prédéfinis.

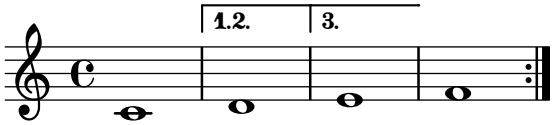
```
\fixed c' {
  \set breathMarkType = #'tickmark
  c2 \breathe d2
}
```



Améliorations en matière de reprises

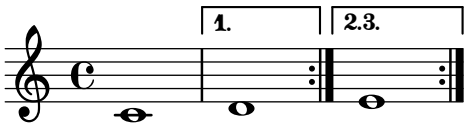
- Les alternatives peuvent se placer au sein même du bloc de répétition.

```
\repeat volta 3 { c'1 \alternative { d' e' } f' }
```



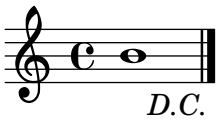
- Les numéros d'alternative peuvent se définir à l'aide de la commande `\volta`.

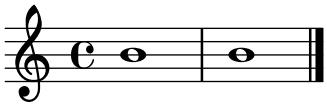
```
\repeat volta 3 c'1 \alternative { \volta 1 d' \volta 2,3 e' }
```



- La nouvelle commande `\repeat segno` permet de gérer automatiquement un certain nombre de formes *da-capo* et *dal-segno*.

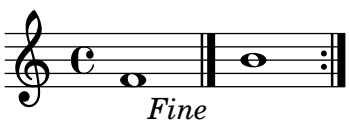
```
music = \fixed c' {
  \repeat segno 2 {
    b1
  }
  \fine
}

\score { \music }
\score { \unfoldRepeats \music }
```

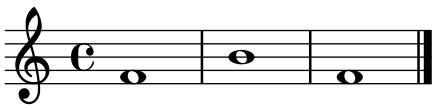




- La nouvelle commande `\fine` insère une barre de mesure finale qui interagit de façon optimale avec les barres de reprise. Placée à l'intérieur d'une reprise, elle ajoute une instruction *Fine* et termine la musique lorsque les reprises sont expansées.

```
music = \fixed c' {
  \repeat volta 2 {
    f1
    \volta 2 \fine
    \volta 1 b1
  }
}

\score { \music }
\score { \unfoldRepeats \music }
```

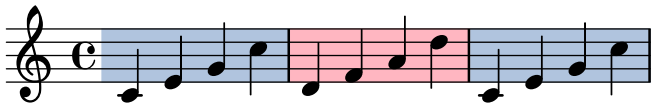




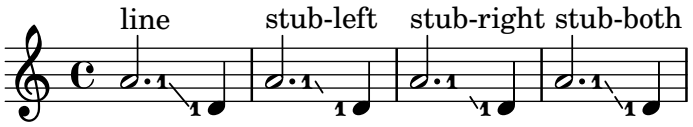
- La commande `\volta` suprime la musique lorsque la répétition est expansée.
- La commande `\unfolded` ajoute la musique lorsque la répétition est expansée.

Améliorations en matière d'annotations éditoriales

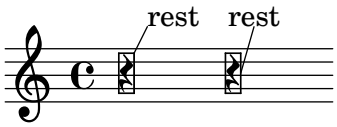
- Les nouvelles commandes `\staffHighlight` et `\stopStaffHighlight` permettent de colorier des fragments musicaux.



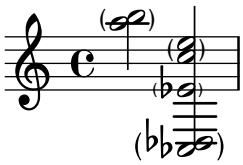
- Le nouvel objet graphique `FingerGlideSpanner` permet d'indiquer le glissé d'un doigt sur une corde passant d'une position à une autre. Il peut se présenter sous différentes formes selon le style adopté ; l'image ci-dessous affiche les styles `line`, `stub-left`, `stub-right` et `stub-both`.



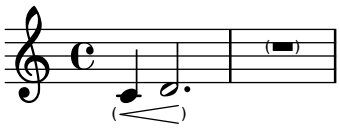
  D'autres styles sont disponibles : `dashed-line`, `dotted-line`, `zigzag`, `trill`, `bow` et `none`.
- La mise en forme des info-bulles est désormais modifiable.



- Il est désormais possible de placer un accord entre parenthèses. Toutefois, la taille de la fonte utilisée pour les parenthèses devra être ajustée manuellement.

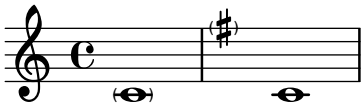


- Les objets étendus ou bandeaux peuvent être mis entre parenthèses.



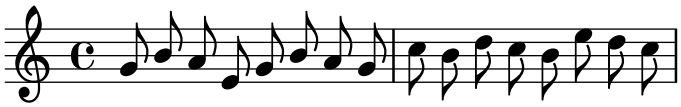
- Une version « temporelle » de la commande `\parenthesize` est disponible. Elle prend en considération le chemin vers un objet graphique : `\parenthesize` *`NomGrob`* ou `\parenthesize` *`ContextName.GrobName`*. Cette commande se comporte comme un `\once \override`. Cette interface vient en complément de la forme déjà disponible `\parenthesize` *`événement`*, de manière identique à `\footnote`.

```
{
  \parenthesize NoteHead
  c'1
  \parenthesize Staff.KeySignature
  \key g \major
  c'1
}
```



- L'ajout du `Melody_engraver` à un contexte `Voice` prend désormais en compte le sens de la mélodie pour orienter la hampe de la note sur la ligne médiane. Il fallait auparavant apporter une dérogation spécifique à la propriété `Stem.neutral-direction`.

```
\new Voice \with {
  \consists Melody_engraver
}
\relative c'' {
  \autoBeamOff
  g8 b a e g b a g |
  c b d c b e d c |
}
```



La propriété de contexte `suspendMelodyDecisions` permet de désactiver temporairement ce comportement, tout comme le faisait `\override Stem.neutral-direction = #DOWN`.

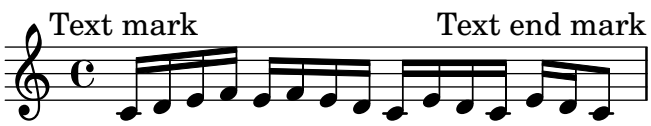
- Le nouveau `Mark_tracking_translator` prend le pas sur le `Mark_engraver` en matière de décision quant à la création d'une marque. Le `Mark_engraver` se charge toutefois de contrôler la mise en forme et le positionnement vertical du repère.

  Par défaut, des `Mark_engravers` dans de multiples contextes créent une séquence commune de repères. Lorsque des séquences indépendantes sont préférables, il faut utiliser plusieurs `Mark_tracking_translator`.

Améliorations en matière de mise en forme du texte

- Les nouvelles commandes `\textMark` et `\textEndMark` permettent d'ajouter entre les notes un texte arbitraire que l'on appelle indication textuelle. Ces commandes constituent une amélioration de la syntaxe utilisée précédemment avec la commande `\mark` à laquelle on adjoignait un *markup* – autrement dit libeller `\mark "..."` ou `\mark \markup ...`.

```
\fixed c' {
  \textMark "Text mark"
  c16 d e f e f e d c e d c e d c8
  \textEndMark "Text end mark"
}
```



`\textMark` et `\textEndMark` sont désormais les moyens recommandés de créer des indications textuelles. Le propos n'est pas ici de décourager l'utilisation de `\mark`, mais de plutôt la réserver à la création des marques-repères – avec `\mark \default` ou `\mark` *nombre* – auxquelles il est souvent fait référence en répétition.

Ces nouvelle commandes diffèrent de `\mark` *markup* par quelques aspects. Alors que plusieurs indications peuvent survenir en un même endroit, il ne peut y avoir qu'une seule

occurrence de \mark (un seul repère à la fois). Elles génèrent des objets graphiques du type TextMark particulier, alors que \mark génère un objet RehearsalMark que ce soit pour créer un repère ou une indication textuelle. Cette distinction permet alors de gérer des réglages différents pour les repères et les indications textuelles. L'alignement engendré par ces nouvelles commandes est différent : l'indication créée par \textMark sera toujours alignée par la gauche et celle créée par \textEndMark le sera toujours par la droite ; l'alignement d'un RehearsalMark, par contre, dépend du point d'ancrage de l'objet sur lequel il doit s'aligner.

Voir Section "Indications textuelles" dans *Manuel de notation* pour de plus amples détails.

- Sont désormais disponibles dans la fonte Emmentaler, des variantes textuelles pour les glyphes de dièse, bémol, bécarre, double dièse et double bémol. En mode *markup*, ils sont accessibles par leur valeur Unicode standard.

1 ♯ 2 ♭ 3 ♮ 4 𝄫 5 𝄪 6

- Il est désormais possible de contrôler la largeur et l'aspect de certains chiffres Emmentaler à l'aide de fonctionnalités OpenType.

- \smallCaps est maintenant opérationnel sur tout *markup*, non plus uniquement sur des chaînes brutes.
- La syntaxe permettant de gérer des conditions pour les *markups* a gagné en flexibilité et en simplicité. Elle utilise les nouvelles commandes de *markup* \if et \unless. Voici quelques uns des changements apportés :

Syntaxe version 2.22	**Syntaxe version 2.24**
`\on-the-fly #first-page ...`	`\if \on-first-page ...`
`\on-the-fly #not-part-first-page ...`	`\unless \on-first-page-of-part ...`
`\on-the-fly #(on-page n) ...`	`\if \on-page #n ...`

- La nouvelle commande de *markup* string-lines permet de couper une chaîne à un caractère donné. La coupure intervient par défaut au saut de ligne, et les éventuels espaces sont ignorés. La liste résultante de *markups* peut faire l'objet d'une mise en forme particulière.

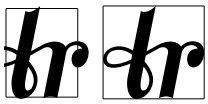
Cette fonctionnalité constitue un moyen tout à fait adapté pour ajouter des couplets à une chanson.

Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky.
Twinkle, twinkle, little star,
How I wonder what you are!

- La nouvelle commande de *markup* `\align-on-other` déplace un *markup* comme s'il était aligné sur un autre.

1
12
12345
123

- Sont maintenant disponibles les deux commandes de *markup* `\with-dimension` et `\with-dimension-from`. Elles sont similaires à `\with-dimensions` et `\with-dimensions-from`, à ceci près qu'elles permettent de ne jouer que sur une seule des deux dimensions.
- Sont maintenant disponibles les commandes de *markup* `\with-true-dimension` et `\with-true-dimensions`. Elles procurent l'étendue réelle de la surface encrée, qui peut être quelque peu différente de l'étendue par défaut pour certains glyphes en raison des contraintes de régularité du texte.



- Des textes de remplacement peuvent désormais changer des chaînes dans tout *markup*, au lieu de modifier une chaîne en particulier.

```
\markup
  \replace #`(("2nd" . ,#{ \markup \concat { 2 \super nd } #}))
  "2nd time"
```

2nd time

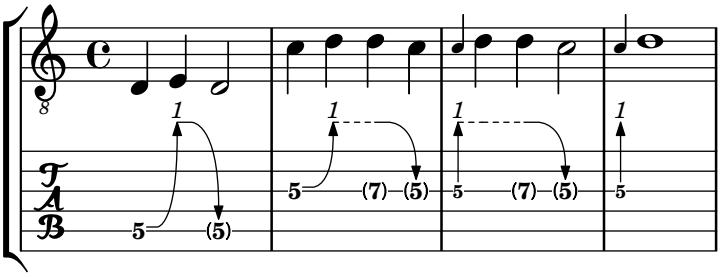
- La commande de *markup* `\with-string-transformer` est désormais disponible. Elle interprète un *markup* avec l'un des « transformateurs de chaîne » installés. Le transformateur est appelé lorsque l'interprétation du *markup* requiert l'interprétation d'une chaîne et permet d'opérer des modifications sur cette chaîne, comme par exemple en basculer la casse.
- La fonction `markup->string` convertit un *markup* dans une réprésentation approximative de chaîne de caractères. Ceci est utile pour générer des métadonnées PDF ainsi que les paroles et repères en MIDI. Il est donc possible de créer des commandes pour *markups* personnalisées afin de retraiter les chaînes converties par `markup->string`, comme par exemple

```
#(define-markup-command (upcase layout props arg) (string?)
   #:as-string (string-upcase arg)
   (interpret-markup layout props (string-upcase arg)))
```

Nouveautés en matière de notation spécialisée

Améliorations pour les cordes frettées

- Ajout des accordages pour banjo `banjo-double-c` et `banjo-double-d`.
- Le nouvel objet graphique `BendSpanner` permet, dans le cadre d'un `TabStaff`, d'indiquer une désinance. En plus de l'apparence par défaut sont disponibles les styles `'hold`, `'pre-bend` et `'pre-bend-hold`.



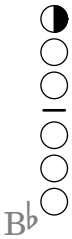
Améliorations pour la notation des percussions

- Ajout du style de notation pour percussions `weinberg-drums-style` basé sur les travaux de standardisation de Norman Weinberg.

Améliorations pour la notation des instruments à vent

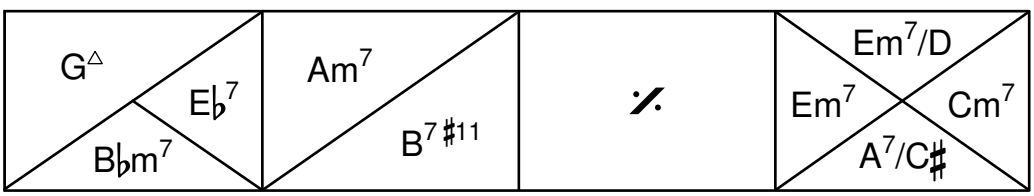
- Des détails peuvent désormais s'ajouter à un `\woodwind-diagram`, y compris l'angle d'un trou partiellement bouché ou l'indication non graphique d'une clé de trille.

```
\markup {
  \override #'(graphical . #f)
  \override #'(woodwind-diagram-details . ((fill-angle . 90)
                                           (text-trill-circled . #f)))
  \woodwind-diagram #'flute #'((cc . (one1h))
                               (lh . ())
                               (rh . (besT)))
}
```



Améliorations pour la notation des accords

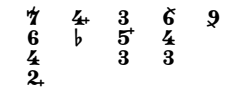
- Les grilles harmoniques sont désormais prises en charge.



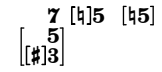
- Dans un `ChordNames`, les silences multimesure forcent désormais l'apparition du symbole « N.C. » comme pour des silences normaux.
- En matière de basse chiffrée, un `_` crée désormais un un chiffre vide occupant de l'espace.

```
\figures {
  <8 _ 4]> <_ 5+ 3>
}
```

- La mise en forme de la basse chiffrée s'est améliorée. En particulier, sa taille par défaut est réduite à une valeur que l'on retrouve dans nombre d'éditions *Urtext* de musique baroque.
- La basse chiffrée utilise désormais par défaut des glyphes dessinés spécifiquement dans ce cadre, à savoir les `6\\`, `7\\` et `9\\`. De même, des glyphes spécifiques seront utilisés par défaut en présence d'un signe plus après le chiffre pour les symboles `2\+`, `4\+` et `5\+`.



  L'utilisation de ces glypes dans le cadre d'un *markup* s'obtient à l'aide de la commande `\figured-bass`.
- En matière de basse chiffrée, il est désormais possible de placer les altérations entre crochets.

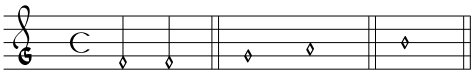


Améliorations pour les notations anciennes

- Le nouveau contexte `VaticanaLyrics` est similaire au contexte `Lyrics`, à ceci près qu'il fournit un style d'hyphénation utilisé communément dans le style des éditions vaticanes, à savoir un trait d'union simple accolé à la syllabe de gauche.
- Les commandes prédéfinies pour les divisions grégoriennes ne sont plus des variantes de `\breathe`. `\divisioMinima`, `\divisioMaior`, `\divisioMaxima` et `\virgula` sont des variantes d'un `\caesura` de base. `\finalis` équivaut à `\section`.

  `MensuralStaff` et `VaticanaStaff` ont recours au `Divisio_engraver` pour interpréter ces commandes, ainsi que `\repeat volta` et `\fine`.

```
\new MensuralStaff \fixed c' {
  \repeat volta 2 { f2 f }
  g1
  a1 \section
  b1 \fine
}
```



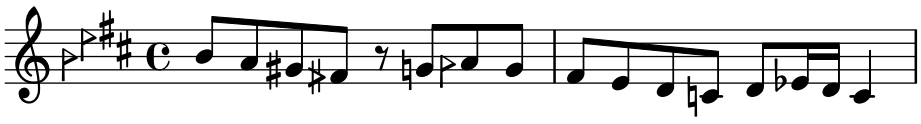
- `KievanStaff`, `MensuralStaff`, `PetrucciStaff` et `VaticanaStaff` acceptent désormais qu'un saut de ligne intervienne n'importe où, sans qu'une barre de mesure `""` soit créée.
- Dans un `GregorianTranscriptionStaff`, les *divisiones* sont désormais gravées comme des objets graphiques *(grobs)* `BarLine`. Pour les modifier en *grobs* `Divisio`, il faut mentionner `\EnableGregorianDivisiones`.

- `GregorianTranscriptionStaff` autorise désormais un saut de ligne apèrès n'importe quelle note et n'a plus recours au `Time_signature_engraver`.
- `GregorianTranscriptionVoice` n'a désormais plu recours au `Stem_engraver`.

Améliorations pour les musiques du monde

- LilyPond prend désormais en charge la musique persane. Deux glyphes d'altération ont été ajoutés à cet effet : *sori* et *koron*.

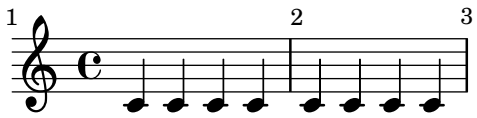
```
\include "persian.ly"

\relative c' {
  \key d \chahargah
  bk'8 a gs fo r g ak g |
  fs ek d c d ef16 d c4 |
}
```



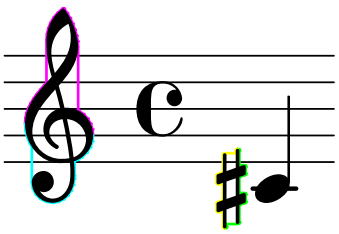
Autres améliorations diverses

- Dans le cadre de la fonte Emmentaler, les têtes de notes dont l'aspect est identique et pour lesquelles la seule différence réside dans l'orientation de leur hampe ont été consolidées en un glyphe unique. Par exemple, les glyphes `noteheads.u2triangle` et `noteheads.d2triangle` ont été remplacés par le seul glyphe `noteheads.s2triangle`. Les paires de tête de note ayant un aspect différent selon l'orientation de la hampe conservent leur distinction.

  Par ailleurs, la propriété `stem-attachment` des objets graphiques `NoteHead` renvoie le point d'attachement réel selon l'orientation de hampe en lieu et place d'un hypothétique point d'attachement d'une hampe ascendante.
- Deux glyphes redondants ont été supprimés des fontes Emmentaler : `scripts.trillelement` – utilisez `scripts.trill_element` en remplacement – et `scripts.augmentum` – à remplacer par `dots.dotvaticana`.
- L'utilisation de `\paper { bookpart-level-page-numbering = ##t}` permet d'individualiser la numérotation de page des parties d'un ouvrage. Dans le cas où l'instruction s'applique à toutes les parties, chacune d'entre elles aura sa propre séquence, en partant de 1 par défaut. Cette instruction peut aussi apparaître dans une partie spécifique, comme par exemple un propos liminaire dont la pagination peut accessoirement se présenter en chiffres romains grâce à l'instruction `page-number-type = #'roman-lower`.
- La nouvelle fonction de rappel pour objet graphique `break-alignment-list` permet de renvoyer une valeur différente selon le positionnement d'un *grob* par rapport à une rupture. Elle permet, par exemple, de fournir un alignement du *grob* différent selon qu'il est positionné en début, en cours de ligne ou à la fin.

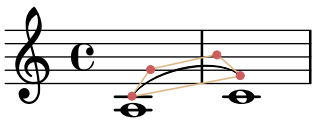


- Le nouveau `Mark_performer` crée des événements MIDI à l'instar du `Mark_engraver` pour les sorties imprimables.
- Les propriétés de `PaperColumn` et `NonMusicalPaperColumn` telle que `NonMusicalPaperColumn.line-break-system-details` acceptent désormais des dérogations au fil de la musique à l'aide d'une simple instruction `\once \override`. Elles consituaient jusqu'alors une exception, requérant l'utilisation de la commande `\overrideProperty`.
- Les nouvelles propriétés `show-horizontal-skylines` et `show-vertical-skylines` permettent d'afficher les lignes d'horizon d'un objet. Ces propriétés sont beaucoup plus flexibles que l'utilisation de l'option `debug-skylines` dans la mesure où elles sont opérationnelles avec tous les objets graphiques. Bien que prévu à l'origine pour le débogage de LilyPond, ceci s'avère fort utile lorsque l'on tente de comprendre les décisions en matière d'espacement ou la modification de stencils en Scheme.



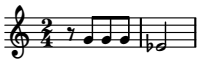
- La nouvelle commande `\vshape` se comporte comme `\shape`, à ceci près qu'elle affiche les points de contrôle et le polygone qui les enveloppe, afin de faciliter les ajustements.

```
{ a1\vshape #'((0 . 0) (0 . 0.5) (0 . 0.9) (0 . 0.4))^( c'1) }
```



- L'instruction \markup \path est désormais également opérationnelle pour une sortie SVG même si le chemin n'est pas initialisé par une commande moveto ou rmoveto. Elle accepte par ailleurs les raccourcis équivalents SVG (moveto = M, etc.).
- set-default-paper-size et set-paper-size acceptent désormais des tailles de papier personnalisées.

```
#(set-default-paper-size '(cons (* 100 mm) (* 50 mm)))
```

- lilypond-book prend en charge deux nouvelles options pour le traitement des extraits : paper-width et paper-height permettent de déterminer une taille particulière de papier.
- lilypond-book prend en charge la nouvelle option inline pour le traitement d'extraits musicaux. Ceci permet de faire apparaître des bribes de partition, comme  au sein même d'un paragraphe de texte.
- Le script lilypond-book autorise désormais l'utilisation d'accolades dans les arguments des commandes \lilypond (pour LaTeX) et @lilypond (pour Texinfo).
- lilypond-book ajoute désormais le répertoire courant comme dernière entrée pour la recherche des fichiers inclus, au lieu de le placer en premier dans la liste des chemins spécifiés. Ainsi, les fichiers situés dans les dossiers d'inclusion prennent le pas sur ceux du même nom dans le répertoire courant. L'effet ne sera visible que dans le cas où existent des fichiers du même nom de part et d'autre.
- La nouvelle fonction Scheme universal-color procure une palette de huit couleurs conçue pour lever certaines ambiguïtés dont souffrent les gens atteints de dyschromatopsie.

black
orange
skyblue
bluegreen
yellow
blue
vermillion
redpurple

- L'option -dembed-source-code permet maintenant d'embarquer également les images ajoutées par \epsfile ainsi que les fichiers inclus par \verbatim-file.
- La valeur par défaut de l'option aux-files est désormais fixée à #f. Lorsque LilyPond est lancé avec l'argument -dbackend=eps et que sont nécessaires les fichiers .tex et .texi, il faut désormais specifier explicitement l'option -daux-files. Les formats pour les images lilypond-book peuvent se définir séparément selon qu'il s'agit de la page (donc une sortie PNG pour du format HTML) ou des images indépendantes par système (pour des formats imprimables EPS ou PDF) à l'aide respectivement des sous-options -dtall-page-formats et -dseparate-page-formats.
- L'unité « big point » (1 bp = 1/72 in) est désormais accessible en ajoutant \bp à la dimension saisie.
- Les traducteurs définis en Scheme et utilisables à la fois dans le cadre d'un '\layout' et d'un '\midi' peuvent désormais être créés à l'aide de make-translator. Les exécutants définis en

Scheme, qui ne peuvent s'utiliser que dans un '`\midi`' se créent à l'aide de `make-performer`. Ces macros fontionnent tout comme la macro préexistante `make-engraver` qui permet de créer un graveur, uniquement utilisable dans un '`\layout`'.

- Les traducteurs définis en Scheme peuvent désormais définir un nouveau connecteur dénommé `pre-process-music`. Celui-ci sera appelé par tous les traducteurs, après tous les *listeners* mais préalablement aux connecteurs `process-music`. Ceci peut s'utiliser pour un traitement qui dépend de l'intégralité des événements entendus mais nécessite de définir des propriétés de contexte avant que d'autres traducteurs les lisent.
- Les traducteurs Scheme peuvent désormais contenir des *listeners* libellés ainsi :

```
(listeners
 ((event-class engraver event #:once)
  ...))
```

  Ils ne sont jamais déclenchés plus d'une fois par pas dans le temps. Ils émettront un avertissement lorsqu'ils recevront deux événements dans un même pas, sauf si ces événements sont équivalents.
- Une même définition d'objet graphique peut désormais s'utiliser pour créer des *grobs* de classe différente (`Item`, `Spanner`, `Paper_column`, `System`). Dans le cadre de cette évolution, les types de *grob* `FootnoteItem` et `FootnoteSpanner` ont été rassemblés en un unique type `Footnote`. De la même manière, `BalloonTextSpanner` et `BalloonTextItem` ont été unifiés en `BalloonText`.

  Lorsque la définition du *grob* ne requiert pas de classe, les graveurs doivent décider de la classe à utiliser pour créer le *grob*. Pour ceux qui créent leurs propres graveurs en Scheme, cela signifie qu'il faudra utiliser soit `ly:engraver-make-item`, soit `ly:engraver-make-spanner`. La fonction utilitaire `ly:engraver-make-sticky` permet de prendre en charge le cas des *sticky grobs* tels que notes de bas de page et infobulles. Elle crée un *grob* de la même classe qu'un autre et gère la parenté et les attachements.
- La nouvelle option `-dcompile-scheme-code` pour la ligne de commande – on peut aussi l'inclure en tête de fichier LilyPond par une ligne `#(ly:set-option 'compile-scheme-code)` – permet un meilleur diagnostic lorsque l'exécution de code Scheme provoque une erreur. En interne, ceci utilise le compilateur de byte fournit par guile, en lieu et place de l'interpréteur.

  Néanmoins, en raison d'une limitation de Guile, il n'est pas possible à ce jour de traiter au-delà de quelques milliers d'expressions Scheme. Il faut aussi tenir compte du fait que le compilateur de Guile diffère quelque peu de l'interpréteur. Par exemple, les parties constantes des quasicitations sont traduites en constantes réelles de manière plus agressive, ce qui a pour conséquence qu'un code comme `(let ((x 4)) (sort! `(,x 3 2 1)))` produira une erreur puisque le « cdr » de la liste quasicitée est une constante, et faire subir une mutation à une donnée littérale est une erreur pour Scheme. (Dans ce cas d'espèce, le code pourrait éviter l'erreur s'il avait été utilisé un simple `sort` – qui n'est pas destructif – ou en créant une nouvelle liste à chaque fois avec un `(list x 3 2 1)`).

  De plus, cette option n'est à ce jour pas fonctionnelle sur Windows.

Notes sur Guile 2.2

LilyPond utilise désormais Guile 2.2 au lieu de Guile 1.8. Cette section liste quelques unes des incompatibilités que l'on rencontre le plus fréquemment en portant du code écrit pour Guile 1.8 vers Guile 2.2.

Un liste détaillée des changements apportés à Guile est disponible dans le fichier `NEWS` (`https://git.savannah.gnu.org/cgit/guile.git/tree/NEWS`) des sources de Guile.

- La fonction `format` requiert un premier argument qui est un booléen ou un port. Cet argument était facultatif en Guile 1.8. Pour que la fonction renvoie la sortie formatée comme une chaîne de caractères, il faut passer `#f` pour cet argument, c'est à dire utiliser `(format #f "`*`chaîne`*`" `*`arguments`*` ...)` au lieu de `(format "`*`chaîne`*`" `*`arguments`*` ...)`.
- Les règles sur les définitions internes (les `define` qui ne sont pas au plus haut niveau du fichier mais à l'intérieur d'autres expressions) sont devenues plus strictes. En particulier, on ne peut plus utiliser `define` dans certains contextes où une expression est attendue. À titre d'exemple, ce code n'est plus valide :

  ```
  (if (not (defined? 'variable))
      (define variable 'valeur))
  ```

  Dans ce cas précis, une solution est

  ```
  (define variable
          (if (not (defined? 'variable))
          'valeur
          variable))
  ```

- Les chaînes de caractères sont désormais constituées de caractères Unicode. Auparavant, un caractère Unicode s'étendait sur plusieurs caractères. De plus, les fonctions de Guile dédiées au traitement des chaînes de caractères prennent maintenant en charge l'Unicode.
- Certaines fonctions numériques renvoient des résultats exacts dans des cas où le résultat était auparavant inexact. Par exemple, `(sqrt 4)` renvoie maintenant `2` (un entier) au lieu de `2.0`.