

# Tema 12. Notificaciones

## Índice

1. [Introducción](#)
2. [Elementos de una notificación](#)
3. [Canales de notificación](#)
4. [Crear una notificación](#)
5. [Estilos de notificación](#)
6. [Botones](#)
7. [Algunos extras de notificaciones](#)

# Introducción

Las **notificaciones** son mensajes que se muestran en el sistema para proporcionar al usuario información adicional de la app (mensajes, avisos, etc). Los usuarios pueden presionar la notificación para abrir la app o realizar una acción directamente desde la notificación.

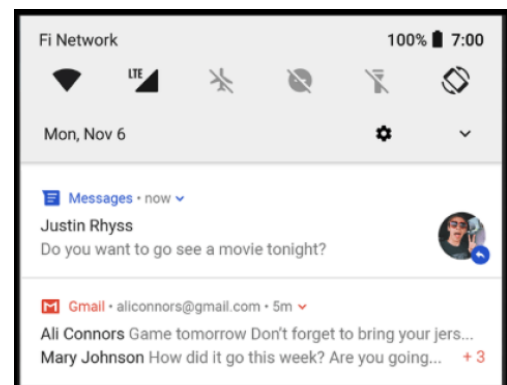
Las notificaciones se muestran de diferentes maneras:

- Iconos en la barra de estado.
- Entrada más detallada en el panel desplegable de notificaciones.
- Notificaciones emergentes (normalmente para notificaciones importantes).
- Notificaciones de pantalla de bloqueo.
- Distintivo en el ícono de la app, en versiones más recientes.

## Notificaciones en la Barra de Estado

La barra de estado de Android se encuentra situada en la parte superior de la pantalla. La parte izquierda de esta barra está reservada para visualizar notificaciones. Cuando se crea una nueva notificación, aparece un texto desplazándose en la barra, y a continuación, un pequeño icono permanecerá en la barra para recordar al usuario la notificación.

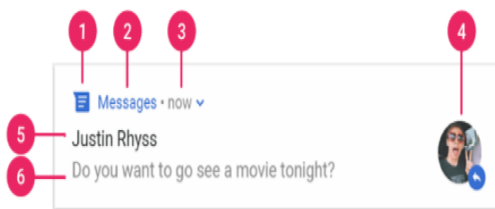
El usuario puede arrastrar la barra de notificaciones hacia abajo, para mostrar el listado de las notificaciones por leer.



## Elementos de una notificación

Las notificaciones podemos verlas de dos formas distintas, la vista normal antes de que el usuario la despliegue y la vista ampliada.

|  |  |
|--|--|
|  | 1. Icono pequeño: se establece con <code>setSmallIcon()</code> . |
|--|--|



2. Nombre de la aplicación: proporcionado por el sistema.
3. Hora a la que se emitió la notificación. Se puede establecer un valor explícito con `setWhen()` u ocultarlo `setShowWhen(false)`; si no se hace se mostrará por defecto la hora del sistema en el momento de recepción de la notificación.
4. Icono grande: es opcional (generalmente se usa solo para fotos de contacto; no lo use para el icono de su aplicación) se establece con `setLargeIcon()`.
5. Título: Esto es opcional y se establece con `setContentTitle()`.
6. Texto: esto es opcional y se establece con `setContentText()`.

La vista ampliada sólo aparece cuando se expande la notificación, lo que sucede cuando la notificación está en la parte superior del buzón de notificaciones, o cuando el usuario amplía la notificación con un gesto.

## Canales de notificación

A partir de Android 8.0 (nivel de API 26), todas las notificaciones deben asignarse a un canal o directamente no serán visibles. Los canales de notificación permiten a los desarrolladores agrupar las notificaciones en categorías (canales). Esto permitirá al usuario la habilidad de modificar los ajustes de notificación para el canal entero a la vez. Por ejemplo, para cada canal, los usuarios pueden bloquear completamente todas las notificaciones, anular el nivel de importancia, o permitir que la insignia de la notificación se muestre. Los usuarios también pueden presionar una notificación para cambiar los comportamientos del canal asociado. Todas las notificaciones publicadas en el mismo canal de notificación tienen el mismo comportamiento.

⚠ la interfaz de usuario se refiere a los canales de notificación como "categorías".

En los dispositivos que ejecutan Android 7.1 (nivel de API 25) e inferior, los usuarios pueden administrar las notificaciones solo por aplicación (de hecho, cada aplicación


solo tiene un canal). Por lo tanto antes crear la notificación, deberemos crear el canal o los canales a los que queremos añadir nuestras notificaciones.

## Crear canales de notificación

Antes de publicar una notificación se debe crear el canal, por lo que es importante que al iniciarse la aplicación o antes de crear la notificación se ejecute el código de creación. Veamos un ejemplo para crear dos canales de notificación:

```
0 private fun crearCanal(
    idCanal: String,
    nombreCanal: String,
    descripcion: String,
    importancia: Int
): NotificationChannel? {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        val canal = NotificationChannel(idCanal, nombreCanal, importancia)
        canal.description = descripcion
        return canal
    }
    return null
}

private fun crearCanalesNotificacion() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        val notificationManager = getSystemService(
            NotificationManager::class.java
        )
        var canal = crearCanal(
            CHANNELAVISOS_ID,
            "Avisos",
            "Avisos importantes",
            NotificationManager.IMPORTANCE_HIGH
        )
        canal!!.vibrationPattern = longArrayOf(400, 600, 100, 300, 100)
        // Registrando el canal en el sistema. Después de esto no se
        // podrá cambiar las características del canal
        // (importancia u otras propiedades del)
        notificationManager.createNotificationChannel(canal)
        canal = crearCanal(
            CHANNELMENSAJES_ID,
            "Mensajes",
            "Mensajes",
            NotificationManager.IMPORTANCE_LOW
        )
        notificationManager.createNotificationChannel(canal!!)
    }
}
```

 Los pasos a seguir son:

1. Construir un objeto del tipo **NotificationChannel** , con un ID de canal único, un nombre visible para el usuario, una descripción y un nivel de importancia.
2. Si en el constructor no especificamos la descripción, se puede especificar después con setDescription().
3. Podemos aplicar otros comportamientos visuales y sonoros a nuestro canal, en el ejemplo hemos activado vibración para el canal de Avisos.
4. Por último registrar el canal de notificación pasándolo al sistema con el método **createNotificationChannel()** .

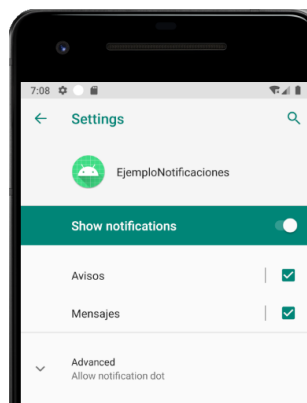
En nuestro ejemplo definimos un método que crea un canal y que es llamado por el método que asigna los canales creados a la aplicación.

La prioridad de nuestras notificaciones se ajusta a las siguientes opciones:

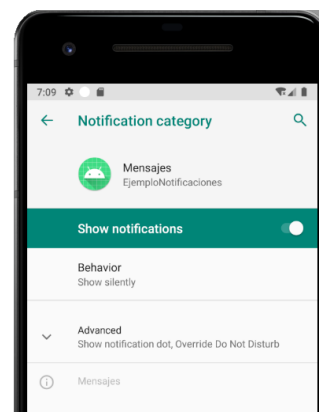
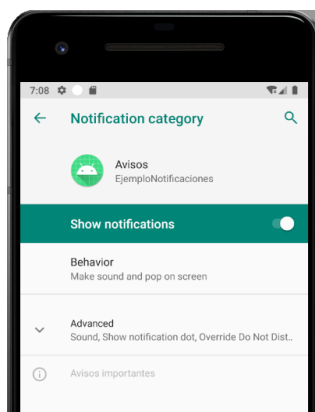
| User-visible importance level   | Importance (Android 8.0 and higher) | Priority (Android 7.1 and lower) |
|---|-------------------------------------|----------------------------------|
| <b>Urgent</b><br>Makes a sound and appears as a heads-up notification | IMPORTANCE_HIGH                     | PRIORITY_HIGH or PRIORITY_MAX    |
| <b>High</b><br>Makes a sound  | IMPORTANCE_DEFAULT                  | PRIORITY_DEFAULT                 |
| <b>Medium</b><br>No sound   | IMPORTANCE_LOW                      | PRIORITY_LOW                     |
| <b>Low</b><br>No sound and does not appear in the status bar          | IMPORTANCE_MIN                      | PRIORITY_MIN                     |

Evidentemente, a todas las notificaciones para un canal se les dará el mismo nivel de importancia.

Las categorías creadas aparecerán en las opciones de configuración de nuestro dispositivo. *Ir a Ajustes->Notificaciones->Seleccionamos Aplicación y nos aparecerá la categoría creada. También se mostrará si se pulsa largamente sobre la notificación desplegada y después sobre el icono de información.*



Si entramos dentro de la categoría veremos su definición detallada:



Es en este panel donde el usuario activa o no las notificaciones de una determinada categoría, también puede cambiar la acción que ocurrirá al lanzarse la notificación asociada a la categoría.

## Crear una notificación

Para crear notificaciones se utiliza la clase `NotificationCompat.Builder`. Para crear una notificación solamente crearemos un objeto de este tipo y le asignaremos el contexto de la aplicación.

```
val notificacion: NotificationCompat.Builder
notificacion = NotificationCompat.Builder(applicationContext, CHANNELMENSAJES_ID
```

se usa como en el siguiente ejemplo:

```
val notificacion: NotificationCompat.Builder
notificacion = NotificationCompat.Builder(applicationContext, CHANNELMENSAJES_ID
notificacion.setContentText("Este es el texto de mi notificación")
notificacion.setTitle("Mi notificación")
notificacion.setSmallIcon(R.mipmap.ic_launcher)
notificacion.setLargeIcon(
    (ContextCompat.getDrawable(
        this,
        android.R.drawable.sym_action_email
    ) as BitmapDrawable?)!!.bitmap
)
notificacion.setTicker("Optional ticker")
notificacion.setWhen(System.currentTimeMillis())
notificacion.setAutoCancel(true) //elimina notificación una vez visualizada
```

✂ Los métodos `setTitle()` y `setContentText()`, permiten colocar el título y el texto de la notificación. Para mostrar los iconos se utilizarán los métodos `setSmallIcon()` y `setLargeIcon()` que se corresponden con los iconos mostrados a la derecha y a la izquierda del contenido de la notificación en

versiones recientes de Android, en versiones más antiguas tan sólo se mostrará el icono pequeño a la izquierda de la notificación. Además, el icono pequeño también se mostrará en la barra de estado superior.

El ticker (texto que aparece por unos segundos en la barra de estado al generarse una nueva notificación) lo añadiremos mediante `setTicker()` y el texto auxiliar (opcional) que aparecerá a la izquierda del icono pequeño de la notificación mediante `setContentInfo()`.

La fecha/hora asociada a nuestra notificación se tomará automáticamente de la fecha/hora actual si no se establece nada, o bien puede utilizarse el método `setWhen()` para indicar otra marca de tiempo.

El siguiente paso, una vez tenemos completamente configuradas las opciones de nuestra notificación, será el de generarla llamando al método `notify()`. Para ello nos crearemos un objeto de tipo `NotificationManagerCompat` al que le pasaremos el contexto, a través de este llamaremos al método `notify()`. `notify` necesitará un entero que identifique nuestra notificación y el resultado del builder que hemos construido antes.

```
val idNotificacion: Int = 0
val notificationManager = NotificationManagerCompat.from(this)
notificationManager.notify(idNotificacion, notificacion.build())
```

El último paso será establecer la actividad a la cual debemos dirigir al usuario automáticamente si éste pulsa sobre la notificación. Para ello debemos construir un objeto `PendingIntent`, que será el que contenga la información de la actividad asociada a la notificación y que será lanzado al pulsar sobre ella. Para ello definiremos en primer lugar un objeto `Intent`, indicando la clase de la actividad a lanzar, que en nuestro caso será una búsqueda por `URL`. Este intent lo utilizaremos para construir el `PendingIntent` final mediante el método

`PendingIntent.getActivity()`. Por último asociaremos este objeto a la notificación mediante el método `setContentIntent()` de la notificación creada.

Veamos cómo quedaría esta última parte comentada:

```
val intent = Intent(Intent.ACTION_VIEW)
intent.data = Uri.parse("http://www.ua.es")
val intent2 = Intent(this@MainActivity, MainActivity::class.java)
notificacion.setContentIntent(PendingIntent.getActivity(this@MainActivity, 0, intent2, 0))
notificacion.setDeleteIntent(PendingIntent.getActivity(this@MainActivity, 0, intent2, 0))
```

⚠ En este caso, además, hemos añadido más funcionalidad al lanzar otro `pendingIntent` una vez borrada la notificación. Este pending lo añadiremos con el método `setDeleteIntent()` sobre la notificación, y se ejecutará al deslizar la notificación para eliminarla.

## Estilos de notificación

Las notificaciones nos permiten [personalizarlas](#) creando layouts propios, pero nosotros nos centraremos en los 3 estilos que vienen de serie. Recordad que estos estilos se expanden y contraen, por lo que tenemos que definir unos datos para la notificación normal y otros para la expandida. Además, por defecto sólo aparecerá expandida la primera notificación de la bandeja, el resto estarán contraídas y el usuario podrá abrirlas manualmente con un gesto. Los estilos posibles son:

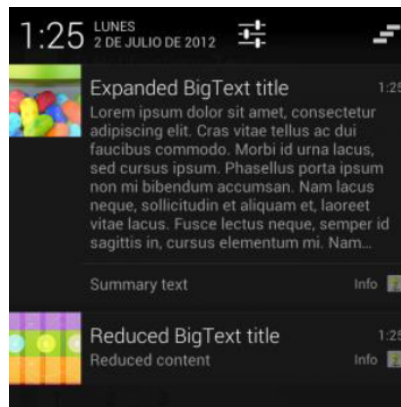
### Estilo BigText

Se usa a partir de un objeto de tipo Builder, pasándolo como parámetro a la clase `NotificationCompat.BigTextStyle` para poder aplicarle algunos métodos opcionales. Queda más claro con un ejemplo:

```
NotificationCompat.BigTextStyle(notificacion)
    .bigText("Lorem ipsum dolor sit amet, consectetur adipiscing
    Donec id ex hendrerit, ullamcorper lacus quis, auctor odio.
    Curabitur in feugiat elit, ut laoreet ex. Nunc in tristique
    at posuere magna. Curabitur augue lectus, tristique a consec
    vitae, luctus a nisi. Quisque nec nulla neque. Proin grvida
    lorem in tempus. Interdum et malesuada fames ac ante ipsum
    in faucibus.")
    .setBigContentTitle("Expanded BigText title")
    .setSummaryText("Summary text")
notificationManager.notify(2, notificacion.build())
```

✎ Como se puede suponer al crear el estilo, lo haremos sobre una notificación base creada con anterioridad y que la pasaremos al objeto `BigTextStyle`. Al crear el nuevo estilo estableceremos el texto largo, como puede ser el contenido de un correo electrónico o de un SMS, el título que llevará en su forma expandida si queremos que sea diferente, y un resumen opcional que se mostrará en 1 línea al final del texto. En la captura podéis ver un ejemplo de cómo quedaría la misma notificación expandida y contraída (la imagen no coincide exacta con el código).





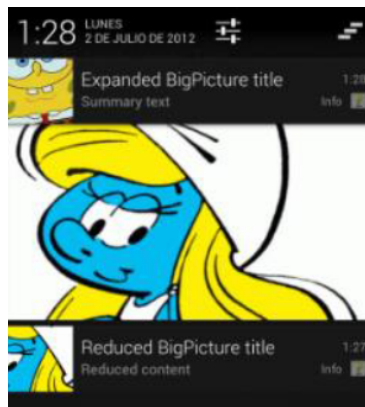
## Estilo BigPicture

Si la notificación es relativa a una imagen también podemos usar una notificación expandida para mostrar dicha imagen en la bandeja gracias a

`NotificationCompat.BigPictureStyle`. Su uso es casi idéntico al anterior, cambiando la clase y métodos del estilo.

```
NotificationCompat.BigPictureStyle(notificacion)
    .bigPicture(
        ContextCompat.getDrawable(
            this,
            R.drawable.ic_launcher_foreground
        ).toBitmap()
    )
    .bigLargeIcon(
        ContextCompat.getDrawable(
            this,
            R.drawable.ic_launcher_foreground
        ).toBitmap()
    )
    .setBigContentTitle("Expanded BigPicture title")
    .setSummaryText("Summary text")
notificationManager.notify(7, notificacion.build())
```

✎ En este caso en vez de un texto, se añadirá imagen grande que queremos mostrar con `bigPicture()` y si queremos que la imagen "pequeña" a la izquierda de la notificación sea distinta en la forma expandida también la cambiamos con `bigLargeIcon()`.

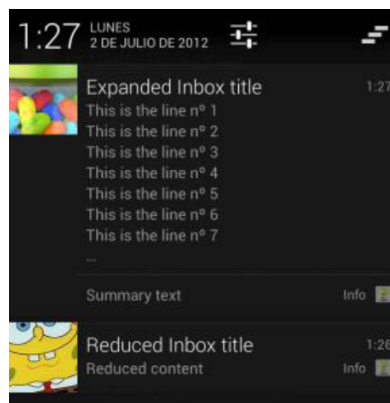


## Estilo Inbox, línea a línea

El último estilo que proporciona Android es el `Notification.InboxStyle`, que muestra una lista de texto. Como varios correos recibidos, o lo que nos escribe alguien por mensajería instantánea. Muy similar también a los anteriores.

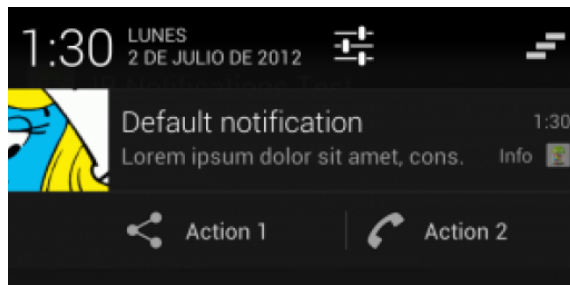
```
val notificacionInboxStyle = NotificationCompat.InboxStyle(notificacion)
    .setBigContentTitle("Expanded Inbox title")
    .setSummaryText("Summary text")
notificacionInboxStyle.addLine("Mensaje 1")
notificacionInboxStyle.addLine("Mensaje 2")
notificationManager.notify(3, notificacionInboxStyle.build())
```

En esta ocasión añadimos las líneas una a una con `addLine()`, aunque nos aconsejan un máximo de 5. Nos quedaría algo así (código e imagen no coinciden).



## Botones

Otra novedad introducida en Jelly Bean es la posibilidad de añadir hasta 3 botones a las notificaciones con sus acciones asociadas. Se hace sobre el objeto de tipo `Notification.Builder` con el método `addAction()`, así que es compatible con cualquiera de los 4 tipos de notificaciones anteriores. En la notificación normal los botones se considerarán como parte "expandida", por lo que tendrán el mismo comportamiento que estas.



En el ejemplo vemos un método que añade 2 botones a cualquier builder que pasemos como parámetro.

```
val action = NotificationCompat.Action.Builder(  
    android.R.drawable.ic_menu_delete,  
    "Delete", PendingIntent.getActivity(this@MainActivity, 0, i,  
        .build()  
    notificacion.addAction(action)  
    notificacion.addAction(  
        android.R.drawable.ic_menu_share,  
        "Share",  
        PendingIntent.getActivity(this@MainActivity, 0, i, 0))  
    notificationManager.notify(4, notificacion.build())
```

✎ Le pasamos como parámetros un drawable del icono que llevará el botón a la izquierda del texto, el texto que llevará, y un PendingIntent que se activará cuando pulsemos el botón.

## Algunos extras de notificaciones

Por ahora todo lo que hemos visto ha sido para crear la notificación, pero aún tenemos que mostrarla y quizá hacer que el móvil suene y/o vibre. También podemos hacer estas cosas con los métodos de la clase `NotificationCompat.Builder`. Veamos unos ejemplos:

### Sonido

Podemos reproducir un sonido al activar la notificación. El la imagen se reproduce el sonido por defecto, puede ser otro.

```
val uri=RingtoneManager.getDefaultUri(RingtoneManager.TYPE_ALARM)  
notificacion.setSound(uri)
```

### Vibración

También podemos hacer que el teléfono vibre. Para eso lo primero que necesitamos es declarar el permiso de vibración

```
<uses-permission android:name="android.permission.VIBRATE" />
```

Luego creamos un patrón de vibración. Consiste en un array de long alternando el tiempo de vibración y pausa en milisegundos. Y se lo asignamos a la notificación.

```
val pattern = longArrayOf(1000, 500, 1000)
notificacion.setVibrate(pattern)
```

La vibración por defecto la podemos conseguir de la siguiente manera:

```
notificacion.setDefaults(Notification.DEFAULT_VIBRATE)
```

## getActiveNotifications

A partir del Api 23 se ha agregado a la clase `NotificationManager` el método `getActiveNotifications()`. Muy útil para consultar el estado de las notificaciones, este método devuelve un array con el estado de todas las notificaciones activas en el momento.

## StatusBarNotification [] getActiveNotifications ()

Recupera una lista de notificaciones activas: aquellas publicadas por la aplicación que aún no han sido descartadas por el usuario o canceladas por la aplicación. Se puede identificar la notificación a partir del tag o del id suministrado al `notify()`, así como una copia del original Notification objeto a través `getNotification()`.

 **Ejercicio Resuelto Notificaciones Expandibles**

 **Ejercicio Propuesto Temporizador Huevo**