Práctica 1 (a): entorno de trabajo

Antonio Bonafonte – profesores de la asignatura

septiembre de 2021

Resumen

Este documento presenta las herramientas básicas que se usarán durante las prácticas de PAV:

- Sistema operativo: Linux o similar.
  - Versiones de Linux y alternativas de instalación.
  - Instalación de paquetes en Ubuntu.
- Manejo de ficheros de audio.
  - Edición de audio con `wavesurfer` y/o `audacity`.
  - Formato de ficheros WAVE.
    - Visualización/edición de ficheros binarios con `bless`.
  - Manejo de ficheros de audio con `SoX`.
- Generación y visualización de gráficos usando `Python/Matplotlib`.
  - Otras alternativas como `gnuplot`, `octave`...
- Editor de textos.

Índice

1. Sistema operativo.

Las prácticas se realizarán en el sistema operativo GNU/Linux, que es una versión libre y de código abierto del sistema operativo Unix. GNU/Linux es el producto de la convergencia de dos iniciativas independientes: por un lado, Richard Stallman y otros crearon la *Free Software Foundation* (FSF) con la idea de desarrollar un sistema operativo de código abierto inspirado en UNIX, el GNU (que, curiosamente, significa *GNU's Not Unix*); por otro, Linus Torvalds decidió programar el núcleo (*kernel*) de un sistema operativo, que acabaría siendo el núcleo usado por la FSF para su sistema. Así pues, el nombre correcto del sistema operativo es GNU/Linux, aunque casi todo el mundo lo conoce como *Linux* a secas (y así lo haremos nosotros en lo que queda de curso, que es mucho).

Siendo un sistema operativo de código abierto y gratuito, muchas empresas han desarrollado *distribuciones* del mismo. Una distribución es un paquete completo con el núcleo y las aplicaciones necesarias para tener un sistema operativo operativo[1] (desarrollo de software, entornos gráficos, etc.). Entre las más importantes, tenemos Ubuntu, Fedora, Debian, SUSE... En general, podemos decir que todas ellas son el mismo S.O., pero con distinto aroma (en inglés, *flavour*). Estas diferencias afectan sobre todo a los mecanismos de instalación y actualización, a los entornos gráficos, y a los paquetes de software instalados por defecto.

Aunque, en la actualidad, Linux suele utilizarse en un entorno gráfico, el modo fundamental de trabajo continua siendo orientado a texto, utilizando un intérprete de comandos o *shell*. El shell más extendido en los sistemas Linux es el `Bash`. Este intérprete es una evolución del shell estándar en los sistemas Unix originales, el *Bourne Shell* o `sh`. De hecho, Bash significa *Bourne again shell*, jugando con la homofonía entre *Bourne*, el nombre del programador que desarrolló `sh` y *born*, nacer en inglés. Así pues, Bash vendría a ser ***`sh`*** *renacido*.

1.1. Instalación de un sistema compatible con Linux.

En los puestos de trabajo del laboratorio está instalada una versión reciente de la distribución Ubuntu de Linux. Como buena parte del trabajo se realizará fuera del horario de prácticas, es altamente aconsejable, casi imprescindible, que todos los alumnos dispongan de un ordenador dotado de este sistema operativo, o semejante. Hay diveras opciones para conseguir esto:

1. `MacOS`: este sistema operativo es altamente compatible con Linux, así que puede usarse directamente la consola del S.O., que ejecuta una versión del intérprete de comandos `Bash`.
   - Algunas aplicaciones pueden dar problemas de instalación, pero siempre es posible encontrar alternativas y, en las prácticas más importantes, (casi) todo parece funcionar bien.
2. Tanto en máquinas PC como Apple es posible reservar una partición del disco duro para ejecutar Linux en ella.
   - Solución óptima desde el punto de vista de ejecución del Linux...
   - ... pero tiene todas las ventajas e inconvenientes de tener dos máquinas distintas que no pueden estar en marcha simultáneamente.
3. Uso de una máquina virtual, como `VirtualBox`, `VMware` o semejante.
   - La principal ventaja es que permite instalar una distribución Linux auténtica sin necesidad de realizar modificaciones en el S.O. original.
   - Pueden haber problemas de compatibilidad con el hardware (teclado, ratón, pantalla...).
   - Mala gestión de los recursos de RAM y CPU
     - No puede disponer de todos los recursos disponibles porque hay que dejar los suficientes para el S.O. anfitrión.

[1] Valga la redundancia

    - Captura los recursos que le asignemos, con lo que el S.O. anfitrión no puede acceder a ellos aunque la máquina virtual no los utilice.
  - Baja eficiencia, sobre todo si no se dispone de virtualización por hardware.

4. Cygwin es un port de Linux a Windows

   - Colección de aplicaciones típicas de Linux recompiladas para ser ejecutadas desde Windows
     - Aunque se consigue la apariencia de ejecutar un Linux, en el fondo sólo estamos ejecutando programas Wimdows.
     - Al principio sólo incorporaba un conjunto limitado de comandos y programas, pero en la actualidad instala un Linux casi completo.
     - En concreto, incorpora el sistema de desarrollo completo de GNU.
   - Permite ejecutar tanto aplicaciones de Cygwin como de Windows, desde el `bash.exe`. Pero recordemos que éste es un programa Windows, que no es lo mismo que el `bash` auténtico de Linux.
     - Podemos ejecutar aplicaciones Cygwin o Windows, pero no podremos ejecutar programas Linux nativos.

5. WSL (Windows Subsystem for Linux).

   - Es una aplicación desarrollada por Microsoft que permite ejecutar un Linux nativo como una DLL de Windows.
     - Es un Linux auténtico, con lo que la compatibilidad es total, tanto con Linux como con Windows.
     - Es una DLL, con lo que sólo se usan los recursos realmente necesarios.
   - Aún está en un nivel muy experimental (al menos, en el verano de 2019).
     - Tiene problemas aún no resueltos: usuarios, permisos...
     - Es peligroso (muy peligroso) acceder a los ficheros y directorios Linux desde Windows.
     - Está prevista una primera gran actualización para otoño de 2019.
   - Permite ejecutar desde el `bash` programas tanto de Linux como de Windows.
   - En principio, sólo permite la ejecución en modo texto (consola `bash`).
     - Para ejecutar aplicaciones gráficas hay que instalar un servidor X en el Windows e informar a Linux de su uso.
       - Un servidor X bastante bueno es `VcXsrv`.
         - ◇ Al ejecutarlo por primera vez, Windows preguntará si permitimos es acceso a del programa a las redes privadas y/o públicas. Hay que decirle que sí a todo tipo de red.
         - ◇ Si se usa un antivirus distinto al propio de Windows (Microsoft Defender), hay que permitir el acceso del programa a través de su *firewall*.
       - Desde `bash` hay que ejecutar:

```
export DISPLAY=$(awk '/nameserver / {print $2; exit}' /etc/resolv.conf 2>/dev/null):0.0
export LIBGL_ALWAYS_INDIRECT=1
```

       Suele ser una buena idea poner esta orden en el script de arranque de sesión (`~/.bashrc`, `~/.profile` u otro, en función de la versión de Linux y/o el shell empleado)[2].

La opción recomendada para los usuarios de Windows 10 es WSL con una versión estable y reciente de Ubuntu (en la actualidad, la última versión *long term support* es Ubuntu LTS 20.04).

[2] En Ubuntu ejecutado desde WSL, el fichero es `~/.bashrc`.

Actualización de Ubuntu

Se recomienda a todos los usuarios de Ubuntu, tanto nuevos como antiguos, actualizar el sistema. Para ello deben ejecutarse las órdenes siguientes:

```
usuario:~/PAV/P1$ sudo apt-get update
usuario:~/PAV/P1$ sudo apt-get dist-upgrade
```

1.2. Instalación de aplicaciones en Ubuntu.

Antes de entrar en detalle de cómo instalar programas en Linux, conviene entender la diferencia entre gestión de paquetes e instalación de aplicaciones. Un sistema de gestión de paquetes sirve para automatizar los procesos de instalación, configuración, actualización y eliminación de paquetes de software. Suele estar vinculado al sistema operativo, que mantiene una base de datos centralizada con todos los paquetes instalados y sus versiones. Aunque muchas aplicaciones externas al S.O. pueden, y es recomendable que lo hagan, utilizar el gestor de paquetes del S.O., otras aplicaciones son gestionadas con sus propios programas de instalación.

Cada S.O. tiene su propio gestor de paquetes. Por ejemplo, en Windows tenemos los menús de *aplicaciones y características* dentro del menú de configuración del sistema. Aparte de las aplicaciones de Windows y de Microsoft, otras muchas aplicaciones utilizan este gestor de paquetes (Adobe Acrobat, Matlab, etc.). Sin embargo, hay otras aplicaciones que utilizan sus propios sistemas de instalación, como Cygwin o AutoCAD.

En Linux hay casi tantos gestores de aplicaciones como distribuciones distintas del S.O.: Fedora usa `yum`, Debian usa `dpkg`, MacOS X usa `fink`, etc. En principio, en las prácticas de PAV se usará la distribución de Linux Ubuntu. En Ubuntu, el gestor de aplicaciones es `apt`. Este gestor de paquetes está formado por diversas herramientas. Las dos más importantes son `apt-get`, que permite gestionar la instalación, actualización y desinstalación de paquetes, y `apt-cache`, que permite gestionar la base de datos de aplicaciones instaladas y disponibles para instalación.

Cuando una aplicación está disponible en los repositorios estándar de Ubuntu, la orden a ejecutar para instalarla es `apt-get install paquete`. Como, por defecto, `apt-get` intentará instalar la aplicación en los directorios del sistema y, en general, no dispondremos de permisos de escritura en esos directorios, lo habitual es tener que ejecutar la instalación adquiriendo privilegios de administrador. Eso se consigue con la orden siguiente, que será la orden que primero se tendrá que probar cada vez que necesitemos un paquete de software:

```
usuario:~/PAV/P1$ sudo apt-get install paquete
```

Nótese que, al intentar ejecutar comandos con privilegio de administrador (`sudo`), el sistema nos solicitará la contraseña, que deberá ser la del usuario. Para que el sistema permita la ejecución de comandos con privilegios de administrador, el usuario deberá estar incluido en el fichero `/etc/sudoers`. Para obtener más información de cómo añadir un usuario al fichero, se puede ejecutar `man sudoers`.

Desgraciadamente, la aplicación deseada no siempre estará disponible en los repositorios controlados por nuestra distribución. En ese caso, cada paquete tendrá sus propias herramientas de configuración e instalación. Lo ideal es acudir a la página web de la aplicación para ver qué pasos hemos de seguir.

Un caso particular muy frecuente, y de gran importancia, es cuando el paquete está disponible en código fuente mantenido con las herramientas del *GNU Build System*, también conocidas como `autotools`. En este caso, una secuencia típica para instalar un paquete sería:

```
usuario:~/PAV/P1$ tar xvzf paquete.tgz
usuario:~/PAV/P1$ cd paquete
usuario:~/PAV/P1/paquete$ ./configure
usuario:~/PAV/P1/paquete$ make
usuario:~/PAV/P1/paquete$ sudo make install
```

Otro caso particular de interés es cuando disponemos del fichero de instalación para distribuciones Debian (en las cuales se basa Ubuntu). Estos ficheros tienen la extensión `.deb` y pueden instalarse con el comando:

```
usuario:~/PAV/P1$ sudo dpkg -i fichero-del-paquete.deb
```

1.3. Algunos consejos para trabajar en Linux.

1.3.1. Obtener información.

Internet está lleno de libros, tutoriales, vídeos, etc. con información acerca de todos los aspectos del trabajo en Linux. Por ejemplo, poniendo `linux tutorial for beginners` en la barra de búsqueda de Google, obtenemos 27 millones de resultados, y refinando la búsqueda para que sea en catalán, se obtiene más de medio millón de resultados.

Algo semejante ocurre al buscar algún concepto concreto. Algunos sitios internet suelen proporcionar respuestas de bastante calidad a gran cantidad de las dudas que podamos tener desarrollando software en Linux. Stack Overflow y Source Forge son dos de los más populares. En ellos, programadores experimentados dan su respuesta a las dudas planteadas por otros usuarios con menos experiencia. Mediante un sistema de votaciones se consigue que, habitualmente, las primeras respuestas en la lista sean interesantes y sólidas desde un punto de vista técnico.

En otros sitios, como Medium Daily Digest o Quora, también aparecen a menudo cuestiones relacionadas con la programación y los sistemas Linux. Aunque el tipo de cuestión y su tratamiento suele ser mucho más superficial (un poco *cuñado-oriented*), las discusiones acerca de temas complejos como Git, lenguajes de programación, entornos de trabajo, etc. pueden ser bastante interesantes (y, a menudo, hay algún cuñado que sabe de lo que habla).

Los comandos `man` e `info` y la opción `--help`.

El comando `man` proviene de los orígenes del Unix, cuando no existían entornos gráficos ni la WWW. Proporciona rápidamente información bastante completa de la mayor parte de los comandos, programas, funciones de librería, etc. de un entorno Unix. Su sintaxis es sencilla: si queremos obtener información acerca de un concepto determinado, sólo hemos de invocar `man concepto` y, si tal concepto está documentado en el sistema `man`, se nos mostrará en pantalla un documento, de formato estandarizado, con la información más relevante del concepto.

```
usuario:~/PAV/P1$ man which
WHICH(1)                    General Commands Manual                   WHICH(1)

        NAME
       which - locate a command

SYNOPSIS
       which [-a] filename ...
```

```
DESCRIPTION
       which  returns  the  pathnames of the files (or links) which would be
       executed in the current environment, had its arguments been given as
       commands in a strictly POSIX-conformant shell.  It does this  by searching
       the PATH for executable files matching the names of the arguments. It does
       not canonicalize path names.

OPTIONS
       -a     print all matching pathnames of each argument

EXIT STATUS
       0      if all specified commands are found and executable

       1      if one or more specified commands is nonexistent or not executable

       2      if an invalid option is specified

Debian                              29 Jun 2016                              WHICH(1)
 Manual page which(1) line 1/25 (END) (press h for help or q to quit)
```

Aunque se dispone de información en formato `man` para la mayor parte de los conceptos relacionados con la distribución Linux que estemos ejecutando, los programas externos no siempre tienen lo que se denomina *página man*. En ese caso, muchas veces el programa incorpora una opción que muestra una pantalla de ayuda. Las opciones de ayuda varían de programa en programa, aunque `-h`, `-help` y `-?` están entre las más habituales. También suele funcionar el invocar el programa con un número erróneo de argumentos (típicamente, sin ningún argumento) o con una opción errónea (tal y como se realiza el procesado de opciones en programas C, la opción `-?` suele ser, o bien la opción de ayuda, o bien errónea).

```
usuario:~/PAV/P1$ more --help

        Usage:
 more [options] <file>...

A file perusal filter for CRT viewing.

Options:
 -d          display help instead of ringing bell
 -f          count logical rather than screen lines
 -l          suppress pause after form feed
 -c          do not scroll, display text and clean line ends
 -p          do not scroll, clean screen and display text
 -s          squeeze multiple blank lines into one
 -u          suppress underlining
 -<number>   the number of lines per screenful
 +<number>   display file beginning from line number
 +/<string>  display file beginning from search string match

     --help     display this help
 -V, --version  display version
```

```
For more details see more(1).
```

Finalmente, algunos programas, como el compilador C de GNU, `gcc`, o el sistema de composición de documentos LaTeX, también proporcionan información usando el comando `info concepto`. Ejecutando, simplemente, `info` apareceremos en el nodo superior del sistema de información, permitiéndonos fisgar por todos los programas y conceptos cubiertos.

1.3.2. Ejecución de programas.

La variable de entorno `PATH` y el fichero `~/.profile`.

En Linux, y casi todos el resto de sistemas operativos en modo texto, el modo de ejecutar un programa es escribir su nombre junto con sus argumentos en la línea de comandos y pulsar la tecla *retorno de carro* (intro, enter, ↩, ...).

Por otro lado, en los sistemas Unix-*like*, sólo podremos ejecutar el programa si escribimos la ruta para localizar el programa en la forma `directorio/nombre`, o bien si el programa está alojado en uno de los directorios indicados por la variable `PATH`. Esta variable incluye los directorios en los que se alojan los programas más habituales separados por dos puntos (`:`). Podemos ver su valor ejecutando la orden `echo $PATH`:

```
usuario:~/PAV/P1$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
```

Fijémonos en que el directorio actual, que se nombra con un punto (`.`), no aparece en el `PATH` por defecto, así que no se podrá ejecutar un programa en él salvo que mencionemos explícitamente su ubicación:

```
usuario:~/PAV/P1$ p1
p1: command not found
usuario:~/PAV/P1$ ./p1
Empleo: ../P1/p1 inputfile [outputfile]
```

Este comportamiento es debido a cuestiones de seguridad, aunque más por motivos históricos que realmente prácticos. Para hacer que los programas en el directorio actual se ejecuten sin necesidad de escribir el engorroso `./`, podemos redefinir la variable `PATH` para añadir el punto (directorio actual) a los directorios de búsqueda:

```
usuario:~/PAV/P1$ export PATH+=:.
```

Esta orden suele meterse en el fichero `~/.bashrc`. Este fichero y su compañero `~/.profile` son ficheros de ejecución de comandos por lotes (*scripts*) que se ejecutan automáticamente cada vez que se inicia una sesión de Bash[3]. A su vez, un script es un fichero de texto con una secuencia de órdenes que se ejecutará del mismo modo que si hubiera sido escrita en el terminal. De este modo, al añadir el punto a la variable `PATH` en el fichero `~/.profile`, podemos ejecutar un programa del directorio actual con sólo escribir su nombre, sin preocuparnos de añadir la ruta o redefinir la variable `PATH` cada vez.

[3] De hecho, hay varias diferencias entre ellos, pero su uso no es siempre consistente. En principio, `.profile` se ejecuta sólo al iniciar una sesión en el sistema (cuando se pide el nombre de usuario y contraseña), mientras que `.bashrc` se ejecuta cada vez que se inicia el programa `bash`. Suele convenirse que las órdenes que no hacen referencia explicita a Bash deben ir en `.profile`. Un ejemplo es, justamente, el contenido de la variable `PATH`. Pero no todos los sistemas inspirados en Unix usan los dos ficheros y/o lo hacen de este modo.

El fichero de arranque en MacOS X

En MacOS X ejecutando Bash, el fichero de arranque es `~/.bash_profile`.

Las versiones más recientes de MacOS X no usan Bash sino `zsh`. En estos sistemas el fichero de arranque de sesión es el `~/.zshrc`, y la sintaxis no permite el uso del operador `+=`, así que la orden a añadir es `export PATH=$PATH:.`

Trabajo en primer y segundo plano.

Como es habitual en todos los sistemas operativos en modo texto, cuando invocamos un programa, el terminal queda bloqueado durante su ejecución. Esto es: la salida estándar del programa se envía a la pantalla del terminal, y el programa aceptará datos escritos con el teclado. Es lo que se denomina *ejecución en primer plano* o *foreground*.

Muchas veces, este modo de funcionamiento no nos interesa. A menudo queremos lanzar la ejecución de un programa, pero queremos que seguir disponiendo del terminal para hacer cualquier otra cosa. Por ejemplo: si queremos editar un fichero con un editor gráfico en una ventana emergente, no queremos que el terminal se quede esperando a que acabe el editor para aceptar nuevas órdenes. Para conseguir esto, Linux permite la llamada *ejecución en segundo plano* o *background*. Para ello es necesario terminar la orden de llamada con el carácter *ampersand* (`&`). Por ejemplo, para editar el fichero `.profile` con el programa Visual Studio Code en segundo plano, tendríamos que escribir:

```
usuario:~/PAV/P1$ code ~/.profile &
```

1.3.3. Algunas teclas importantes.

Evidentemente, la tecla más importante en un sistema Linux es el retorno de carro (intro), ya que es la que se utiliza para finalizar una línea e indicar a Bash que debe ejecutar las órdenes contenidas en ella. Existen, además, otras teclas que son importantes:

\ La barra invertida tiene muchas misiones en Linux. En general, sirve para anular el significado especial de algunos caracteres. Por ejemplo, el espacio se usa para delimitar palabras; asi, 'tercera guerra mundial' sería una frase de tres palabras. Usando barra invertida delante de cada espacio podemos eliminar el significado especial del espacio, de manera que 'tercera\ guerra\ mundial' es una sola palabra. También es útil para dividir líneas muy largas en otras más cortas, sin que Bash interprete cada línea corta como una orden diferente. Para ello, basta con poner un carácter de barra invertida al final de cada línea que queramos unir con la siguiente.

Ctrl-C La combinación del carácter control (Ctrl) con la tecla de la letra `C` provoca la finalización del programa en curso.

Ctrl-Z Esta combinación provoca la interrupción del programa. Éste deja de ejecutarse, pero no se termina, sino que se mantiene *a la espera*. El shell nos informará de ello y nos mostrará un número que indica la tarea que se ha parado:

```
usuario:~/PAV/P1$ code ~/.profile
^Z
[13]+  Stopped                 /usr/bin/Code.exe ~/.profile
```

A continuación, podemos reemprender la ejecución con la orden `fg %num`, para continuar en el primer plano, o `bg %num`, para hecerlo en el segundo:

```
albino:~/PAV/Prácticas/enunciados/P1$ bg %13
[13]+  /usr/bin/Code.exe ~/.profile &
```

tab Aunque no tan importante como las teclas anteriores, el tabulador es una de las más útiles, que todo usuario de Linux debe usar intensiva y extensivamente. Su uso lanza la función *autocompletar*. Apretando el tabulador en cualquier momento hace que el sistema busque las mejor manera de completar la palabra:

- Si los caracteres ya escritos identifican unívocamente el inicio de una palabra, el sistema completará tal palabra silenciosamente. Si el sistema no encuentra ninguna palabra que empiece por ellos, lanzará un mensaje sonoro y/o visual, y no hará nada más.
- Si los caracteres ya escritos pueden corresponder al inicio de unas pocas palabras, en principio el sistema no hará nada; pero, si volvemos a apretar la tecla tab, nos mostrará las distintas continuaciones disponibles. Añadiendo caracteres, e insistiendo en el uso del tabulador, el sistema nos permite ir refinando la búsqueda hasta completar la palabra deseada.
- Si, en lugar de unas pocas palabras cuyo inicio pueda coincidir con lo ya escrito, tenemos un montón, el sistema nos indicará cuántas posibles continuaciones hay y nos pedirá confirmación para mostrarlas en pantalla

El sistema de autocompletado se ha ido perfeccionando con el tiempo. En la actualidad, es habitual que muchos programas, entre ellos Git, la soporten de manera contextual. Por ejemplo, si tecleamos '`git c` tab tab', `git` nos informará de que hay ocho comandos distintos que empiezan por `c`. Si añadimos una `h` y apretamos el tabulador, el sistema añade una `e` y se queda a la espera. Si volvemos a apretar dos veces el tabulador, nos informará de que hay tres comandos que empiezan por `che`. Finalmente, si añadimos la letra `c`, ya sólo existe el comando `checkout` que empiece por `chec`, y el sistema completará su nombre y escribirá un espacio al final, indicando que está a la espera de que se escriba otra palabra o el retorno de carro.

El interés del autocompletado es doble: por un lado nos permite escribir menos; incluso, pensar menos. Pero, salvo para los más perezosos, éste es un beneficio menor comparado con el verdadero interés del autocompletado: todo lo que se escriba de este modo está garantizado frente a errores ortográficos o tipográficos.

Desgraciadamente, parece como si para su desarrollo hubieran contratado a alguien de cierta empresa radicada en Redmond, Washngton, EE.UU., porque, a veces, parece que el sistema sepa mejor que el usuario con qué ficheros debe trabajar. Así, si hemos tecleado `gunzip` y apretamos el tabulador, el sistema de autocompletado sólo expandirá el nombre de los ficheros cuya extensión coincida con una de las reconocidas por `gunzip`. Mientras que esta característica es útil en multitud de ocasiones, en otras muchas resulta un verdadero incordio.

2. Manejo de ficheros de audio.

2.1. Editores de audio: `wavesurfer` o `audacity`.

Existen múltiples programas que permiten el manejo de ficheros de audio en Linux. Dos de los más extendidos son `wavesurfer` y `audacity`.

wavesurfer: Es un sencillo, pero bastante potente, editor de audio, especialmente orientado al tratamiento de la señal de voz. Además de permitir grabar, visualizar y reproducir señales de audio, tiene otras posibilidades que serán de interés durante el curso:

- Edición de la señal, con funciones de corte, copia, pegado, etc.
- Análisis de voz: obtención del contorno de potencia, pitch, formantes, etc.
- Etiquetado de señales.
- Procesado de señal, con efectos como la adición de eco, normalización de la señal, eliminación de la componente continua, etc.

Para evitar problemas con el sistema de audio en sistemas WSL y cygwin, se recomienda utilizar el programa de Windows e invocarlo desde la consola `bash`.

audacity: Es un programa de grabación y edición de audio multipista con muchas más posibilidades que el anterior, pero también algo más complicado de usar. Sin embargo, para apasionados del sonido y la edición musical puede ser una opción más interesante que aquél.

Ambos programas son de software libre y están disponibles, en versiones recientes de Ubuntu, con el comando `sudo apt-get install ...` En WSL hay problemas para acceder al sistema de audio, así que es mejor instalar la versión de Windows y usar `wavesurfer.exe`, indicando la ruta completa en la que lo instalemos.

Tarea

Usando `wavesurfer`, grabe un párrafo que empiece por su nombre y el de su compañero de prácticas y, a continuación, una o dos frases cortas con pausas intermedias. Ajuste el nivel de grabación para que la señal tenga un nivel suficiente pero sin saturar. Colóquese adecuadamente el micrófono para que no aparezcan ruidos por roces o golpes.

Guarde la señal en un fichero Microsoft Wave con el formato siguiente: un solo canal (mono), codificado con PCM lineal de 16 bits y una frecuencia de muestreo de 16 kHz. Para ello, puede seguir dos procedimientos distintos:

1. Fijar el formato por defecto en la pestaña `Sound I/O` de la opción `Preferences` del menú `File`. A partir de ese momento, todas las señales que se graben tendrán ese formato y serán guardadas con él.

2. Convertir la señal al formato deseado usando la opción `Convert` del menú `Transform` y, a continuación, guardar el fichero.

3. Formato de ficheros WAVE.

WAVE es un formato estándar de ficheros de audio propiedad de IBM y Microsoft (a menudo se le conoce como Microsoft WAVE). Es el principal formato usado en Microsoft Windows para el almacenamiento de señales de audio. Aunque soporta distintos esquemas de compresión, es habitualmente usado si ella.

La estructura de los ficheros se basa en los denominados trozos (*chunk*). Cada chunk es identificado por una cadena de cuatro caracteres, que indica el tipo de chunk del que se trata, seguida por un número entero de cuatro bytes, que indica el tamaño del chunk en bytes. El primer chunck siempre se identifica como `"RIFF"`, que es el formato base de los ficheros WAVE, y en el se especifica el formato del fichero, que en nuestro caso es `"WAVE"`. Además, este chunk debe tener dos subchunks: el primero, identificado con la cadena `"fmt "` indica el tipo de codificación empleado (número de canales, esquema

de codificación, número de bytes, etc.); el segundo, identificado con `"data"`, contiene las muestras de la señal.

Puede verse una descripción completa del fichero en formato de ficheros WAVE. Puede observarse que, en los ficheros WAVE, la cabecera ocupa un total de 44 Bytes, a partir de los cuales comienzan los datos de la propia señal.

3.1. Visualización/edición de ficheros binarios con `bless`.

Por definición, un fichero binario no está preparado para ser visualizado o editado directamente. Si lo abrimos con un editor de texto, lo que nos aparece en pantalla es una retahíla de símbolos totalmente incomprensible. `bless` (*binary less*, a su vez chanza de la utilidad clásica de Unix `more`... *less is more*) es una herramienta gráfica que permite visualizar y/o editar ficheros binarios.

Al abrir un fichero binario con `bless` nos encontramos con dos ventanas y una barra de estado en la línea inferior. Esta última nos indica tres cosas: el offset en hexadecimal desde el inicio del fichero, tanto en bytes como en bits; el rango seleccionado, si lo hay; y si el modo de edición es inserción o sobreescritura.

La ventana superior está dividida en tres zonas:

Izquierda: Offset en bytes desde el inicio del fichero.

Centro: Cada uno de los bytes del fichero tal y como aparecen en él. Cada byte está representado por un número hexadecimal de dos dígitos. Esta es la parte más importante de `bless`, en la que podemos usar el ratón para posicionarnos en cualquier punto del fichero, seleccionar un grupo de bytes, editarlos, etc.

Derecha: Contenido del fichero interpretado como caracteres de texto. Sólo en el caso de que el contenido del fichero sea texto veremos algo con pies y cabeza. En el caso de que el byte correspondiente no se pueda interpretar como un carácter válido, se muestra un punto. También permite posicionarse, seleccionar o editar uno o más bytes.

Finalmente, la ventana inferior nos muestra el contenido del fichero en la posición del cursor y representando los bytes siguientes de diversas maneras: como enteros con o sin signo de 8, 16 y 32 bits; como reales de 32 y 64 bits; como cadenas de texto; etc.

Es importante tener en cuenta el modo cómo los datos son almacenados en el fichero, lo que en inglés se conoce como *endianness* y, a menudo, se traduce al castellano como *sexo*. En las máquinas big-endian, los datos son almacenados tal y como se escriben. Así, si el sexo es big-endian, el entero de cuatro bytes $0x01234567$ es almacenado como 01 23 45 67 (empezando por el extremo más grande). En las máquinas little-endian, los datos son almacenados en orden inverso: 67 45 23 01. El formato big-endian es típico de los procesadores de Motorola, mientras que el little-endian lo es de los procesadores de Intel. En cualquier caso, fijémonos en que, en el estándar de los ficheros WAVE, los valores numéricos siempre se representan en formato little-endian.

Tarea

Localice los siguientes campos de la cabecera del fichero WAVE grabado anteriormente:

- Descriptores de los chunks y subchunks: `"RIFF"`, `"fmt "` y `"data"`.
- Número de canales del fichero (será un entero de dos bytes que empieza en el byte 22).
- Frecuencia de muestreo (será un entero de cuatro bytes con offset 24 bytes).

Determine el número de muestras del fichero a partir de la lectura del tamaño en bytes del subchunk `"data"`.

3.2. Manejo de ficheros de audio con SoX.

SoX, *Sound eXchange, the Swiss Army knife of audio manipulation*, es un programa de software libre disponible en múltiples plataformas (incluyendo Windows, Linux y MacOS X), que permite realizar multitud de tareas con ficheros de audio:

- Conversión de formatos de señal y de fichero. Soporta casi cualquier tipo de fichero y codificación de la señal tanto en lectura como en escritura.
- Reproducción de audio en múltiples sistemas, incluyendo streaming por internet.
- Distintas operaciones de procesado de señal, incluyendo, entre otros:
  - Amplificación y normalización, filtrado, modificación de la frecuencia de muestreo, etc.
  - Análisis espectral usando espectrograma.
  - Transformada de Hilbert.
  - Reducción de ruido.
- Distintos efectos de audio, tanto para aplicaciones de voz como musicales, incluyendo:
  - Modificación del tempo y/o tono.
  - Compresión y saturación (*overdrive*).
  - Eco, chorus, flanger, phaser.
  - Ecualización.
  - Síntesis de audio básica.

El programa se ejecuta siempre desde la línea de comandos. Pueden verse las opciones más importantes ejecutando `sox -help`, pero son tantas que se recomienda ejecutar `sox -help | less`.

Tarea:

Estudie las opciones proporcionadas por `sox` y use el programa para convertir el fichero grabado anteriormente al formato de fichero `au` usando codificación ley-mu de 8 bits y una frecuencia de muestreo de 8 kHz. Para ello, ejecute la orden siguiente:

```
sox nombre_fichero.wav -e mu-law -r 8000 nombre_fichero.au
```

Verifique el uso de las opciones de `sox` y compruebe que el fichero resultante cumple con las especificaciones señaladas (por ejemplo, usando el comando `file` de Linux). ¿En cuánto se ha reducido el tamaño del fichero?

Escuche la señal resultante y compruebe la diferencia de calidad con el original.

4. Generación y visualización de gráficos usando Python y Matplotlib.

Existen distintas posibilidades para la representación gráfica de datos o señales. En un entorno Unix o Linux, la opción clásica es el programa `gnuplot`. Pero ésta es una opción probablemente demasiado clásica, aunque de resultados realmente buenos.

A principios del siglo XXI, el neurobiólogo estadounidense John D. Hunter se encontró con que no había ningún sistema de generación de gráficos que visualizara el electrocorticograma de enfermos de epilepsia con la calidad que él deseaba. Así que, ni corto ni perezoso, desarrolló un sistema nuevo que denominó `Matplotlib` (https://matplotlib.org/3.1.0/index.html). Con los años, `Matplotlib` se ha convertido en una opción de referencia para obtener gráficos de alta calidad y configurabidad.

`Matplotlib` es un módulo de Python que está concebido como una librería orientada a objetos, en la que se definen las distintas clases necesarias para la representación de todo tipo de gráficos. Aunque el uso de `Matplotlib` a partir de sus clases y objetos es la manera más potente de acceder a todas sus características, también se incluye una interfaz semejante al sistema de representación gráfica de MATLAB, `matplotlib.pyplot`, y que es el modo más sencillo de iniciarse en el uso de `Matplotlib`, al menos para quien esté familiarizado con MATLAB.

4.1. Python.

Python es un lenguaje de programación creado en la década de los 90 por Guido van Rossum con el explícito propósito de ser un lenguaje *bonito* que permita (incluso obligue) la creación de programas *bonitos*, fáciles de entender y mantener, pero, al mismo tiempo, de altas prestaciones. El resultado es un lenguaje interpretado multi-paradigma que permite, entre otras, programación procedimental, orientada a objeto y funcional. Con los años, Python ha visto crecer su popularidad hasta convertirse en uno de los lenguajes más usados y demandados (y mejor pagados).

Aunque su mayor limitación es su pobre eficiencia computacional (justamente, la eficiencia nunca se ha planteado como un objetivo del lenguaje), la facilidad con la que interactúa con rutinas escritas en otros lenguajes, particularmente C, hace que tal limitación sea mucho menos grave de lo que pueda parecer. De hecho, uno de los objetivos de los creadores de Python es evitar la denominada *optimización prematura*, esto es: sacrificar la legibilidad y mantenibilidad del código en aras a obtener un beneficio marginal en términos de eficiencia computacional. Según el criterio de los *pythonisos* del *pythoniverso* lo más *pythónico* es escribir código lo más legible posible para, a continuación, analizar en qué partes se emplean más recursos computacionales y, sólo si es necesario, optimizarlas o reescribirlas en C.

Una de las claves del éxito de Python radica en la abundancia de librerías, también denominadas *módulos* que aumentan las capacidades del lenguaje. A menudo, estas librerías están escritas en C y, por tanto, son muy eficientes, Entre las más importantes, al menos para los propósitos de este curso, cabe mencionar:

`NumPy y SciPy:` son dos librerías de cálculo numérico y procesado de señal con funcionalidades semejantes a MATLAB.

`Matplotlib:` librería para generación de gráficas de alta calidad dotada de un interfaz semejante al sistema de gráficos de MATLAB.

`PyTorch y TensorFlow:` librerías de altas prestaciones para aplicaciones de *deep learnig* y redes neuronales.

`SoundFile:` librería para lectura/escritura de ficheros de audio.

Una cuestión importante es la existencia de dos versiones de Python, no del todo compatibles. La primera versión ampliamente extendida fue Python 2, cuya versión 2.0 fue lanzada en 2000. A lo largo de los años se fue concibiendo una gran revisión del lenguaje que dio lugar a Python 3 en 2008. Fieles a la filosofía de Python, se decidió no *ensuciar* el lenguaje manteniendo compatibilidad, por lo que muchas características de Python 2 se modificaron o eliminaron completamente en Python 3. Sin embargo, y debido a lo extendido de Python 2, se decidió continuar con el mantenimiento de éste en paralelo con la nueva versión. Está previsto que este mantenimiento finalice en 2020.

Instalación de `python3` y sus librerías.

En Ubuntu, Python se instala del modo habitual con la orden `sudo apt-get install python3` (repárese en el 3). Una vez instalado, las librerías se instalan usando el comando `pip3`[a]. Así, para disponer de las librerías necesarias para representar gráficas en Python, ejecutaremos:

```
usuario:~/PAV/P1$ pip3 install soundfile numpy matplotlib
```

[a]Es posible que sus sistema carezca del comando `pip3`. En ese caso, puede instalarlo con el comando `sudo apt-get install python3-pip`.

También es conveniente instalar `IPython3`, usando `apt-get`. `IPython3` es un shell interactivo de altas prestaciones originalmente diseñado para Python y un acompañante habitual de `Matplotlib`. Usando `IPython3` y las librerías adecuadas, Python se convierte en una alternativa sumamente potente a programas como MATLAB, con la ventaja añadida de que se trata de software libre.

4.2. Empleo de `Matplotlib`.

En https://matplotlib.org/users/pyplot_tutorial.html se dispone de un tutorial completo para el manejo de `matplotlib.pyplot`. A modo de ejemplo, el código siguiente muestra cómo podemos visualizar la señal grabada anteriormente usando `Matplotlib`:

```
import matplotlib.pyplot as plt
import soundfile as sf
import numpy as np

senyal, fm = sf.read('nombre_fichero.au')
t = np.arange(0, len(senyal)) / fm
plt.plot(t, senyal)
plt.show()
```

La función `plt.plot()` es el equivalente en `Matplotlib` del comando `plot()` de MATLAB, así que (prácticamente) las mismas opciones son válidas en los dos casos. Eso sí, la versión de `Matplotlib` permite una mayor configurabilidad de los gráficos y el resultado es mejor en términos de calidad.

También es posible representar gráficamente los valores contenidos en un fichero de texto. Para ello, el fichero debe contener los datos organizados en columnas separadas por un carácter delimitador. Por ejemplo, consideremos que `polynomial.txt` es un fichero de texto con tres columnas de datos, en el que la primera es el valor de la variable independiente x y las otras dos son este valor al cuadrado y al cubo, respectivamente. Deseamos representar tres curvas que tengan el valor de x en el eje de abscisas, y los valores de x, x^2 y x^3 en el eje de ordenadas. La orden necesaria para ello es:

```
import matplotlib.pyplot as plt

plt.plotfile('polynomial.txt', cols=(0, 0, 1, 2), delimiter='\t', subplots=False,
             names=('$x$', '$x$', '$x^2$', '$x^3$'))
plt.show()
```

Fijémonos en algunas cuestiones referidas a esta última orden:

- Las columnas a utilizar se indican con la opción `cols=(0, 0, 1, 2)`. Al usar esta opción, el primer valor se corresponde con los valores del eje de abscisas y el resto con los datos de las curvas. Si el fichero sólo contiene una columna, podemos omitir esta opción y `Matplotlib` representará los valores del fichero usando el número de línea como valor de la abscisa.
- Por defecto, `Matplotlib` representará cada curva en gráficos separados y apilados verticalmente. Para obtener todas las curvas en una misma gráfica ha de especificarse la opción `subplots=False`.

- La tupla[4] `names` indica el nombre dado a las distintas secuencias de valores. El primero se mostrará como el título del eje x, los siguientes se mostrarán en un recuadro denominado *leyenda*. La leyenda puede incluir, como es el caso, expresiones LaTeX, que serán interpretadas adecuadamente. En este caso, por ejemplo, "x^2" se representa como x^2.

4.3. Otras alternativas para visualización de gráficos.

Aunque se recomienda encarecidamente el uso de Python y `Matplotlib`, más que nada como introducción a Python para quienes no lo conozcan ya, existen otras opciones que pueden ser interesantes para quienes no tengan interés en iniciarse en Python todavía (aunque, más adelante en el curso, será imprescindible hacerlo).

4.3.1. `gnuplot`

Como se comentó al principio de esta sección, el programa clásico para generar gráficos en Linux es `gnuplot`. Este programa, que es software libre, aunque no tiene nada que ver con GNU o la FSF, está disponible en la mayor parte de distribuciones de Unix y Linux, y puede instalarse también en máquina MacOS X y Windows. `gnuplot` es un interpréte de comandos con amplias capacidades para la representación de señales a partir de sus valores y/o su representación analítica.

Por ejemplo, para representar los valores del fichero usado en la descripción de `Matplotlib`, podemos usar las órdenes siguientes:

```
plot "polynomial.txt" using 1:1 with lines title "x", \
                   '' using 1:2 with lines title 'x2',\
                   '' using 1:3 with lines title 'x3'
```

Las gráficas obtenidas con `gnuplot` tienen también una alta calidad. Además permite configurar diferentes dispositivos de salida, incluso puede usarse en terminales en modo texto.

4.3.2. MATLAB/Octave.

`MATLAB` (abreviatura de *MATrix LABoratory* es un sistema de cómputo numérico que ofrece un entorno de desarrollo integrado dotado de un lenguaje de programación denominado `M`. Con sus pros y sus contras, `MATLAB` ha constituido durante muchos años el sistema de referencia en áreas como el procesado de señal, análisis numérico y estadístico, etc. En la actualidad, puede decirse que es una de las principales víctimas del auge de Python. Al contrario que este último, `MATLAB` es un sistema propietario de pago (y no precisamente barato, aunque existen versiones domésticas y docentes mucho más asequibles).

`Octave` es un programa libre y gratuito que emula `MATLAB`, aunque más limitado que éste. Su sistema de representación de gráficos es compatible, a nivel de sintaxis, con `MATLAB`, pero internamente usa `gnuplot`, con lo que la calidad obtenida es similar a la obtenida por éste (aunque con muchas menos opciones de configuración). La representación de los valores de `polynomial.txt` usando `MATLAB` u `Octave` sería:

```
load "polynomial.txt"
x = polynomial;
plot(x(:,1), x)
legend('x', 'x2', 'x3')
```

[4] Una *tupla* en Python es un conjunto ordenado de valores y se representa mediante estos valores (en este caso, cadenas de texto), separadas por comas y encerradas entre paréntesis.

4.3.3. `wavesurfer`

También es posible realizar las gráficas necesarias usando el propio editor de señales `wavesurfer` y copiando las gráficas a base de *pantallazos*. Aunque el resultado es el peor de todas las opciones contempladas, su uso es el más inmediato de todos. Además, puede ser una buena opción para representar la señal junto a su segmentación en unidades fonéticas (algo que se hará en la segunda práctica).

5. Editor de textos.

Una decisión a tomar al montar un sistema de desarrollo en Linux es la elección del editor de textos. Éste es necesario para multitud de tareas. Por ejemplo: para editar el fichero `polynomial.txt`, pequeños ficheros de comandos (*scripts*), ficheros de configuración del sistema o del usuario (`~/.bashrc`, `~/.vimrc`, ...), el código fuente, etc.

En el caso del código fuente, son populares los llamados `IDE` (*Integrated Development Environment*) como `eclipse`, `codeblocks`, etc., que permiten editar, compilar, ejecutar y depurar programas desde la propia aplicación. Sin embargo, en esta asignatura invocaremos explícitamente las herramientas básicas para familiarizarnos con ellas.

Probablemente, la peor opción posible para editar textos en Linux (y en otros sistemas operativos) es utilizar el editor que viene configurado por defecto al doble-clickar en un fichero desde el navegador del sistema de archivos. Entre las carencias de estos editores, (`gedit` en Linux, o `notepad` en Windows), podemos destacar la ausencia de realce del código fuente o de sangrado automático, la excesiva dependencia del ratón, etc. No obstante, estos editores han mejorado notablemente en los últimos años y, mediante las opciones de configuración adecuadas, permiten superar algunas de estas limitaciones.

Entre las opciones disponibles en Linux para editar ficheros de texto, podemos destacar dos grandes familias: la de los editores clásicos, orientados al trabajo en terminales de texto; y la de los editores en modo gráfico, cuyo funcionamiento se basa en el uso del ratón, menús y ventanas.

5.1. Editores clásicos: `vim` y `emacs`.

Entre los editores clásicos orientados a terminales de texto, destacan dos: el `emacs` y el `vim`. Los dos son editores muy potentes que permiten automatizar tareas complejas a partir de sus respectivos lenguajes de programación (un dialecto del LISP, en el caso de `emacs`, y un lenguaje más o menos semejante al Python, en el caso de `vim`).

La verdad es que ambos editores son ásperos de aprender y anticuados a los ojos de los neófitos. El motivo es que están fundamentalmente orientados a terminales en modo texto, con modos de funcionamiento extraños (`vim`), o combinaciones de teclas destroza meñiques y tendones (`emacs`).

Aunque pocos editores modernos pueden realizar todo lo que estos editores permiten, la curva de aprendizaje es tan larga, que cabe hablar de años para empezar a sentirse a gusto con ellos. Sin embargo, la característica de no requerir entornos gráficos los hace especialmente útiles cuando éstos no están disponibles: recuperación de sistemas caídos, terminales remotos, etc. Además, en el caso de `vim`, su puesta en marcha y operación es tan rápida que un usuario experto puede abrir, modificar y guardar un fichero antes de que cualquiera de los otros ni siquiera haya empezado a pensar en cargarse.

Durante el siglo pasado se desarrolló la llamada *guerra de los editores* entre los defensores de uno y otro. En la actualidad esa guerra ya no tiene sentido[5]; todos los grandes programadores usan `emacs`: Richard Stallman (su creador), Linus Torvalds, Guido van Rossum, etc. Sin embargo, entre los desarrolladores de a pie, `vim` supera en preferencia a `emacs` en más de cinco a uno (ver https://insights.stackoverflow.com/survey/2019#technology). Lo cual es especialmente cierto entre los administradores de sistema, donde `vim` ocupa un destacado segundo lugar, con un 43.7 % de aceptación (frente al décimo cuarto puesto, con un 6.5 % de aceptación, de `emacs`).

[5] ¿`vim` o `emacs`? No tiene sentido perpetuar esta guerra de editores, ... pero `emacs` es mejor ;-) Como dice Richard Hendricks, fundador de *Pied Piper*: ¿`vim`? *God, help us!*. https://www.youtube.com/watch?v=3r1z5NDXU3s

5.2. Editores con interfaz gráfica.

Entre los editores más modernos, que usan el ratón, menús, ventanas y demás, dos opciones son:

`Visual Studio Code:`
: Es el editor de texto de Microsoft Visual Studio, pero sin su entorno de desarrollo. Aunque desarrollado por Microsoft, es gratuito y de código libre. Tiene todas las características deseables en un editor de texto para desarrollo de software, y figura en el primer lugar de las preferencias de los desarrolladores en todas las aplicaciones (ver el estudio de stackoverflow).

`Atom:`
: *A hackable text editor for the 21st Century*, es un programa de código abierto semejante en sus capacidades a Visual Studio Code, pero especialmente adaptado para interactuar con Git y GitHub (de hecho, ha sido desarrollado por estos últimos). Además, tiene un modo de emulación `vim`.