El entorno de trabajo del científico de datos

Guía de lecturas

PID_00270630

Julià Minguillón Alfonso

Tiempo mínimo de dedicación recomendado: 2 horas

Universitat Oberta de Catalunya

Julià Minguillón Alfonso

Doctor ingeniero en Informática por la Universidad Autónoma de Barcelona (UAB). Profesor agregado de los Estudios de Informática, Multimedia y Telecomunicación de la Universitat Oberta de Catalunya (UOC) en el ámbito de la ciencia de datos. Sus intereses docentes incluyen la programación, la minería de datos y la visualización, entre otros. En investigación, se dedica a analizar el comportamiento de los usuarios en entornos virtuales de aprendizaje y redes sociales, como por ejemplo Wikipedia, con el objetivo de mejorar los procesos de apoyo al aprendizaje y la interacción con el entorno.

El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por el profesor: Julià Minguillón Alfonso (2020)

Primera edición: febrero 2020

Avda. Tibidabo, 39-43, 08035 Barcelona
Realización editorial: FUOC

Índice

Introducción

Siguiendo el ciclo de vida de los datos, es bien sabido que antes de poder construir y evaluar un modelo que resuelva un problema en el ámbito de la ciencia de datos, en la mayoría de casos hay que **recopilar y procesar los datos**, los cuales pueden provenir de diferentes fuentes, en diferentes formatos, etc. Este proceso acostumbra a ser costoso, poco automatizable y seguramente tiene que prever un número elevado de excepciones que hay que resolver después de una inspección detallada. *Grosso modo*, se puede considerar que todo el proceso de creación de un modelo sigue una distribución típica 80-20, donde un 80 % del tiempo se dedica a la preparación de los datos, y el 20 % restante, a la construcción de los modelos y a la evaluación e interpretación de los modelos construidos. Obviamente, cada proyecto o experimento tendrá una configuración diferente, pero normalmente la preparación de los datos exige un proceso a veces muy artesanal que solo hay que ejecutar una vez y que, una vez listo, se puede intentar automatizar para posteriores ocasiones, cuando haya que obtener nuevos datos siguiendo el mismo procedimiento, por ejemplo, y reproducir todos los pasos seguidos.

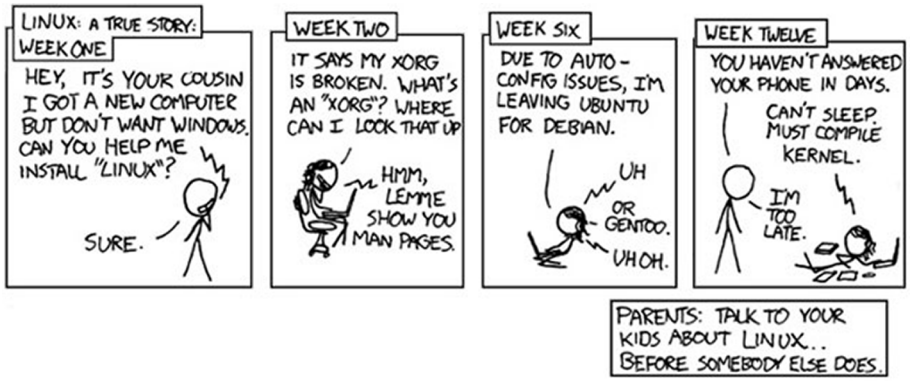
Desgraciadamente (o seguramente, por suerte), no hay una herramienta general que resuelva todos los problemas en todo momento, de forma que para cada experimento o proyecto que debamos resolver tendremos que utilizar un conjunto de herramientas diferente. Este hecho, conocido como *no-free-lunch theorem* en otros ámbitos, también se puede aplicar a la ciencia de datos. Así, algunas de estas herramientas necesarias para resolver un problema pueden venir impuestas por la naturaleza del mismo o de los datos, otras pueden formar parte de la infraestructura tecnológica disponible en el entorno de trabajo y otras pueden ser sencillamente las favoritas que tiene cada científico de datos para trabajar en su día a día para resolver los problemas típicos a los que se enfrenta, como parte de su entorno de trabajo. Por lo tanto, uno de los primeros aspectos que se debe resolver es disponer de un **entorno de trabajo adecuado** para la manipulación de los datos, la creación de modelos y otras tareas típicas del ciclo de vida de los datos. Una vez más, este entorno puede venir impuesto por la manera en que se trabaja en una cierta empresa, organización, la Administración, etc., pero también puede tener que ver con la perspectiva personal del científico de datos, que dispone de sus propios recursos (ordenador personal, acceso a otros servidores, software, etc.) que quiere utilizar para esta finalidad.

En esta guía nos centraremos en el entorno de trabajo necesario para las fases de captura y, especialmente, de preparación de los datos, con el objetivo de generar los conjuntos de datos que serán utilizados en la fase de análisis siguiente. Esto determinará en cierto modo los criterios que seguiremos a la hora de utilizar unas herramientas u otras y la configuración del entorno de trabajo

adecuado. Una vez tengamos una máquina lista para manipular y analizar los datos, veremos cómo usar la línea de comandos, también llamada *consola*, como instrumento que permite acceder al corazón del sistema, y que nos permite realizar tareas de mantenimiento para asegurar que el entorno de trabajo se mantiene actualizado y operativo, y también del centro de operaciones desde el cual llevaremos a cabo las tareas que pueden ser automatizadas mediante las capacidades del lenguaje de *scripting* que nos ofrece el sistema operativo.

Así, pues, el objetivo final de esta guía es ofrecer una visión de los diferentes elementos que proporciona el sistema operativo mediante el uso de la línea de comandos, los cuales pueden ser combinados en pequeños programas o *scripts* que automatizan las tareas repetitivas que son habituales en cualquier proyecto de ciencia de datos. Primero, definiremos los componentes del entorno de trabajo y una propuesta general de cómo se podría configurar. Después, describiremos los comandos típicos que se realizan desde la consola del sistema operativo y algunos ejemplos de problemas que se podrían resolver de este modo. Finalmente, la guía acaba con unas lecturas históricas recomendadas para entender los orígenes de todos los componentes que conforman el universo actual de los entornos mencionados a lo largo de este texto.

Figura 1. *Linux: a true story*



Fuente: xkcd.

Objetivos

1. Introducir los conceptos básicos de la asignatura.

2. Definir los componentes del entorno de trabajo del científico de datos.

3. Presentar diferentes herramientas para la manipulación de datos.

4. Introducir la línea de comandos y la programación en *scripting*.

5. Proponer casos de uso típicos de la línea de comandos.

6. Conocer las diferentes lecturas clave en este ámbito.

1. El entorno de trabajo del científico de datos

Actualmente, hay muchas opciones a la hora de configurar lo que debe ser un **entorno de trabajo optimizado** para la preparación de los datos usados en la fase de análisis, pero de forma simplificada debemos empezar por decidir los tres niveles típicos de cualquier sistema informático: el **hardware**, el **sistema operativo** y el **conjunto de herramientas o software**, los cuales no son realmente independientes, sino que las decisiones en un nivel afectan a los otros dos, estableciendo fuertes vínculos, como, por ejemplo, que cierto software solo se ejecuta sobre un cierto sistema operativo.

1.1. El hardware

Como siempre, cuanta más capacidad de computación y de disco tenga el ordenador que usaremos para hacer experimentos, mejor, puesto que la ciencia de datos consiste precisamente en manipular grandes volúmenes de datos y construir modelos complejos con un coste computacional elevado. Sin entrar en el tema de las infraestructuras para datos masivos, es muy habitual que se tengan que manipular ficheros con millones de registros, lo que genera muchos ficheros intermedios en el proceso, de forma que la capacidad de disco disponible tiene que ser del orden de centenares de gigabytes (GB) o, incluso, de terabytes (TB). Los discos de estado sólido (SSD), usados típicamente en los ordenadores portátiles, son más eficientes, puesto que no requieren partes móviles, pero su capacidad es limitada (centenares de GB). Para grandes capacidades de almacenamiento, hay que usar discos duros convencionales, los cuales son más lentos y consumen más energía, pero llegan a capacidades de varios TB. Otra opción es utilizar el almacenamiento en la nube, con una capacidad virtualmente ilimitada, pero entonces habrá que tener en cuenta la velocidad de transferencia de los ficheros, lo cual puede generar un cuello de botella a la hora de realizar experimentos, puesto que los ficheros del orden del GB o superior no se tendrían que ir moviendo o copiando continuamente.

Un tuit

Un tuit capturado con todos sus metadatos en formato JSON puede ocupar unos 10 KB, de forma que una captura de un millón de tuits puede ocupar unos 10 GB de disco.

El *dump* de Wikipedia que contiene el detalle de todas las ediciones realizadas[1], sin el contenido de las páginas ni de la propia edición, es un fichero comprimido de unos 7 GB para la Wikipedia en castellano, 1,5 GB para la Wikipedia en catalán y de más de 60 GB para la Wikipedia en inglés.

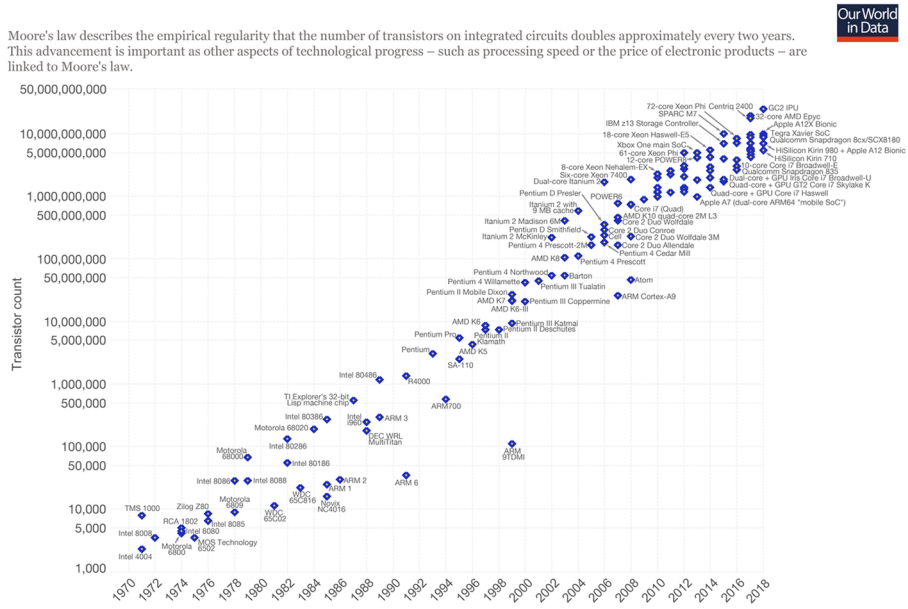
(1)Llamado `<idioma>wiki-latest-stub-meta-history.xml.gz` donde `<idioma>` es «en», «es», «ca», etc.

En cuanto a la capacidad de cómputo, hay dos aspectos a tener en cuenta. El primero es la **cantidad de memoria RAM disponible**, la cual se mueve actualmente en el orden de GB o decenas de GB. Aunque hablar de cifras absolutas es complicado, dada la rapidez con que estas quedan obsoletas, actualmente se considera que 8 GB es el mínimo recomendable para poder afrontar experimentos de cierta envergadura, teniendo en cuenta que muchas de las herramientas para manipular y analizar los datos se cargan en memoria antes de empezar a realizar ningún proceso. En este caso, la decisión es sencilla: cuan-

ta más cantidad de RAM disponible, mejor, y también cuanto más rápida sea y más capacidad de transferencia tenga, mejor; la que se conoce como DDR SDRAM es la más habitual actualmente.

Otro aspecto a tener en cuenta es **la potencia y la velocidad del procesador** (la CPU), elemento que incluye el número de núcleos (también denominados *cores*) que tenga, los cuales se pueden usar para paralelizar tareas, algo muy habitual en las operaciones en las que se manipulan datos en formato de vectores, tablas o tensores, aprovechando al máximo sus capacidades. También aquí las cifras cambian rápidamente, de acuerdo con la famosa ley de Moore, tal como muestra la figura 2. Actualmente, los fabricantes principales de CPU, como por ejemplo AMD o Intel, cuentan en el mercado con procesadores con decenas de núcleos, de forma que es posible hacer tareas en paralelo de forma sencilla, siempre por medio del software adecuado, obviamente.

Figura 2. Ley de Moore sobre el número de transistores en los procesadores (1971-2018)



Fuente: Max Roser. Wikipedia (commons.wikimedia.org/wiki/transistor_count).

Finalmente, desde hace ya un cierto tiempo, ha aparecido un nuevo actor en el escenario de la computación, el que se conoce como **unidades de procesamiento gráfico o GPU**. Originalmente, las GPU estaban dedicadas a procesar de forma muy eficiente los datos necesarios para la visualización de las imágenes en la pantalla del sistema gráfico, pero actualmente se utilizan para hacer cálculos aprovechando su arquitectura paralela que permite aumentar el rendimiento para el tipo de cálculos que son habituales en ciencia de datos. De hecho, con el auge de las redes neuronales y lo que se conoce como *deep learning*, el uso de las GPU se ha visto reforzado por la aparición de hardware dedicado a tal efecto y también por la disponibilidad de librerías que aprovechan al máximo sus capacidades.

1.2. El sistema operativo

El nivel intermedio que conecta el hardware con el software que lo utiliza es el **sistema operativo**, que en realidad puede ser considerado también software en sí mismo. El sistema operativo es una capa que aísla al usuario de todos los detalles necesarios para realizar operaciones sencillas, como, por ejemplo, copiar un fichero, sin importar la estructura del sistema de ficheros subyacente u otros parámetros relacionados con la geometría de los discos. De hecho, el sistema operativo es un software especial que permite al usuario ejecutar otros programas mientras le proporciona servicios básicos para acceder a todo el hardware disponible. En realidad, cualquier hardware contiene de una forma u otra un sistema operativo que lo hace funcionar, como, por ejemplo, los teléfonos inteligentes, las consolas de videojuegos, los automóviles, etc.

Normalmente, cuando arranca el ordenador y se carga el sistema operativo, este proporciona una línea de comandos con la que el usuario puede ejecutar acciones relacionadas con los elementos típicos que componen un sistema informático, pero hace ya tiempo que prácticamente la totalidad de sistemas operativos proporcionan además una interfaz de usuario gráfica que permite ejecutar la mayoría de estas acciones de forma más visual, que es la que se carga por defecto. Esto ha propociado que la mayoría de usuarios desconozca incluso la existencia de la consola o línea de comandos, y también las operaciones que se pueden realizar desde la misma, reduciendo su conocimiento sobre el hardware y el propio sistema operativo a unas cuantas operaciones básicas relacionadas con el sistema de ficheros. Esto es percibido como positivo por muchos usuarios, que solo desean usar su ordenador sin mayores preocupaciones, pero no así por los usuarios que pretenden extraer el máximo provecho de su entorno de trabajo.

De hecho, en muchos casos la elección del hardware implica en cierta forma escoger el sistema operativo, puesto que este va muy ligado a los detalles de los dispositivos físicos de que dispone el ordenador. En el caso de los primeros ordenadores personales esto era prácticamente así, ya que las combinaciones disponibles se reducían a solo unas cuantas (mencionamos las más habituales):

- Un ordenador personal compatible con el IBM PC con sistema operativo MS-DOS/Windows (lo más habitual).

- Un ordenador personal compatible con el IBM PC con sistema operativo GNU/Linux (para los más osados).

- Un ordenador personal de la marca Apple con el sistema operativo Mac OS.

Sin pretender ser exhaustivos, había otras opciones, como, por ejemplo, las estaciones de trabajo de la marca Solaris o Silicon Graphics, que ejecutaban diferentes versiones del sistema operativo Unix, los ordenadores como, por ejemplo, el NeXT o los sistemas operativos como el OS/2 y otras variantes de

Unix, pero no eran tan populares como las tres mencionades anteriormente. También había otros sistemas operativos para servidores de red, pero no eran populares entre los usuarios típicos.

Hay que destacar que los ordenadores de la marca Apple traen un núcleo parecido a GNU/Linux pero que en realidad no lo es, que se denomina *XNU*, y deriva de un sistema anterior llamado *Mach*, usado conjuntamente con BSD para crear el primer sistema operativo de los ordenadores NeXT, marca creada por Steve Jobs que fue posteriormente comprada por Apple, que dio pie al sistema actual. En la práctica, los ordenadores Apple se pueden considerar equivalentes a los ordenadores que ejecutan GNU/Linux, a pesar de que no son exactamente idénticos y no todas las herramientas están disponibles (en el caso de Apple).

En la actualidad, cuando compramos un ordenador, este ya incorpora el sistema operativo y no es posible elegirlo, como pasa con los ordenadores Apple. Sin embargo, en el caso de los ordenadores PC, algunos fabricantes permiten elegir entre Windows y GNU/Linux y, así, se puede descontar el coste de la licencia del primero, un hecho conocido como *Windows tax*.

1.3. El software

El último nivel es el **software** que aprovechará todas las capacidades del sistema informático resultantes de combinar el hardware y el sistema operativo. Obviamente, el sistema operativo determina qué se podrá ejecutar y qué no, aunque muchas veces los desarrolladores de software ofrecen versiones para diferentes sistemas operativos, excepto en el caso del software propietario, dado que es posible que un desarrollador decida optar por una plataforma (hardware + sistema operativo) para explotar al máximo sus capacidades, olvidándose del resto. De hecho, en algunos casos, el hecho de disponer de un software concreto era lo que establecía cómo debía ser el sistema informático donde se tenía que ejecutar, como, por ejemplo, 3D Studio MAX, que solo se podía ejecutar con el sistema operativo Windows (y continúa siendo así), o el Final Cut Pro, que solo se puede encontrar para Mac OS.

Actualmente, excepto en casos muy específicos, todo el software se puede encontrar para versiones de diferentes sistemas operativos, ya sea porque se trata de software abierto con el código fuente disponible y que se puede compilar para tener una versión ejecutable en cualquier plataforma (hardware + sistema operativo), o porque los fabricantes de software se aprovechan de los nuevos entornos de desarrollo multiplataforma que permiten generar binarios ejecutables para cada plataforma entre las más comunes.

De hecho, en un entorno de trabajo dentro del ámbito de la ciencia de datos, será muy habitual tener que compilar aplicaciones distribuidas solo como código fuente, de forma que habrá que disponer de un conjunto de herramientas (editores, compiladores, depuradores, etc.). No es extraño, pues, que uno

de los primeros softwares en abierto disponible fuera un compilador llamado *GCC* (acrónimo de GNU *Compiler Collection*) creado por el MIT en 1987, capaz de compilar lenguaje C. A partir de ese momento, fue posible hacer crecer la familia de herramientas GNU, dado que se disponía de una herramienta que permitía convertirlas en código ejecutable. El propio compilador podía usarse para crear nuevas versiones del mismo compilador para otras plataformas y arquitecturas, y también para otros lenguajes (entre ellos, Fortran), impulsando así la diseminación de las herramientas GNU/Linux y las aplicaciones basadas en estas.

1.4. Una propuesta

Combinando los elementos introducidos en los apartados anteriores, es fácil ver que puede haber tantos entornos de trabajo como científicos de datos, atendido al amplio abanico de posibilidades, limitadas casi únicamente por el presupuesto disponible. Sin embargo, para iniciarse y desarrollar competencias propias del ámbito de la ciencia de datos desde una perspectiva personal, una buena propuesta es optar por un entorno lo más flexible posible, basado en el uso de software abierto, con una distribución GNU/Linux. La opción de hardware más económica es un ordenador PC compatible, especialmente si lo comparamos con los ordenadores de la marca Apple, dado que así podemos disponer de más prestaciones por el mismo precio (o más económico para prestaciones similares), tal como muestra el ejercicio de la tabla siguiente:

	Dell* XPS	**MacBook Pro**
Pantalla	13 pulgadas	13 pulgadas
Procesador	Intel Core i5 a 3,9 GHz	Intel Core i5 a 3,9 GHz
RAM	8 GB	8 GB
Disco duro	256 GB SSD	256 GB SSD
Sistema operativo	Ubuntu Linux 18.04	Mac OS (XNU)
Precio (octubre 2019)	979 $	1.499 $

* Hemos elegido Dell por su popularidad, pero hay otras muchas marcas (Acer, Lenovo, Asus, etc.).

Obviamente, hay otros aspectos que hay que tener en cuenta (peso, duración de la batería, etc.) que no se han incluido en la tabla anterior. Por suerte, la mayoría de fabricantes de hardware permiten simular en línea la configuración de un ordenador, de manera que se pueden hacer comparaciones entre diferentes marcas y modelos.

En cuanto al sistema operativo, en este caso nos decantamos claramente por GNU/Linux (o XNU en el caso de los ordenadores con Mac OS), dado que permite sacar provecho de toda la colección de herramientas y participar de forma activa en el movimiento de software libre. En cuanto a la distribución

elegida, las opciones también son casi innumerables, pero de entre todas podemos destacar Ubuntu (Desktop) por muchas razones. Entre otras, tiene una interfaz gráfica muy amigable, es altamente personalizable, de acuerdo con las necesidades de cada usuario, hay una gran comunidad de usuarios y desarrolladores detrás que aseguran una evolución constante, y funciona en casi cualquier hardware, incluyendo los ordenadores antiguos, además de contar con versiones Lite para máquinas con capacidades limitadas. Aunque hay otras distribuciones que *a priori* parecen orientadas hacia el ámbito de la ciencia de datos, como, por ejemplo, Scientific Linux o Fedora Scientific, generalmente es mejor optar por una distribución genérica como Ubuntu y adaptarla a las necesidades de cada usuario, instalando los paquetes y módulos necesarios.

Herramientas

Algunas de estas herramientas están descritas en datascience.recursos.uoc.edu.

Entre otras cosas, habrá que disponer como mínimo de las herramientas siguientes para poder descargar y manipular ficheros:

- **Compiladores** de C, C++ y Fortran, para compilar librerías científicas y otras herramientas que usan computación numérica o simbólica. El compilador GCC (de hecho, un *front-end*) proporciona compiladores para la mayoría de los lenguajes típicos.

- Un **editor de texto** adecuado para la edición de código fuente. Las opciones también son innumerables: desde las herramientas del sistema operativo, como, por ejemplo, Vi o el editor Emacs, pasando por Nano o Joe, hasta editores más visuales que usan el sistema de ventanas para ofrecer entornos de visualización WYSIWYG (*what you see is what you get*). Resulta interesante que el editor de texto incluya un modo para la edición de código fuente, teniendo en cuenta el lenguaje que se use, ayudando así con la sintaxis de los comandos, variables, estructuras de control, etc.

- Herramientas para la **manipulación de ficheros** CSV, JSON, XML, RDF, etc., es decir, ficheros de texto con un formato interno orientado a la descripción de datos en formato tabular o jerárquico, con o sin metadatos. Entre otras, destacan CSVKit o Jq, por ejemplo.

- Herramientas para acceder a **recursos web**, puntos de entrada SPARQL o API en general, como, por ejemplo, Wget o cURL, que permiten descargar páginas web y realizar otras operaciones mediante el protocolo HTTP. Otras herramientas parecidas, pero con características adicionales que simplifican su uso, son Aria2 y HTTPie.

R o Python

Por ejemplo, sobre la «falsa» dicotomía entre el uso de R o Python, ved www.kdnuggets.com/tag/python-vs-r.

Y en cuanto a las herramientas más allá de las primeras fases del ciclo de vida de los datos, últimamente ha habido una especie de «falsa» dicotomía entre el uso de R o Python como motor del sistema empleado para analizar y visualizar los datos. Decimos «falsa» porque realmente no hay que elegir entre los dos mundos como si fueran incompatibles, dado que pueden convivir razonablemente bien en un mismo sistema. Sí que es cierto, no obstante, que R y Python muestran diferencias muy importantes que pueden hacer optar por

uno o el otro, puesto que parten de premisas muy diferentes. Se puede decir que **R** es un entorno parecido a una «supercalculadora», con la que se pueden hacer cálculos complejos y obtener los resultados inmediatamente, usando un lenguaje con una sintaxis y semántica particulares pero típica del software científico-matemático. Por otro lado, **Python** es un lenguaje de programación general que permite establecer los pasos que hacen falta para resolver un problema y obtener la solución deseada, usando las librerías adecuadas. Sea como fuere, cualquier científico de datos tendría que conocer los dos mundos, usando cada uno en función de sus objetivos y de las circunstancias, sin tener que elegir entre uno o el otro. Un entorno habitual en la caja de herramientas del científico de datos es **Anaconda**. Se trata de una distribución que incluye todo lo necesario para manipular datos, crear modelos, evaluarlos e implementarlos, y también generar informes y gráficos, tanto para R como para Python. De hecho, Anaconda es tan completo que podríamos decir que lo incluye casi todo (la instalación ocupa más de medio gigabyte), por lo que puede no ser necesario en muchos casos en que solo se quiere trabajar con un entorno más concreto, como, por ejemplo, la dicotomía antes mencionada, R o Python:

- R + R Studio + R markdown + Shiny
- Python + Numpy + SciPy

1.5. Entornos virtuales

Una vez escogido el entorno de trabajo, la opción natural es instalar el sistema operativo elegido, hacer las configuraciones oportunas y, después, instalar todo el software deseado. Esto inicia un ciclo continuo de instalaciones y actualizaciones, especialmente por temas de seguridad que siempre hay que tener presentes, además de las mejoras en las prestaciones del hardware, del sistema operativo y del software. Dada la diversidad de hardware, de sistemas operativos (y de distribuciones en el caso de GNU/Linux) y de la combinación de software y librerías, en la práctica muchas veces lo que pasa es que lo que funciona en una máquina puede no funcionar en otra «idéntica», especialmente por un problema de versiones del software necesario y sus dependencias, es decir, del software requerido por otro software, como, por ejemplo, las librerías del sistema. De hecho, la figura 3 muestra en clave de humor la diversidad de comandos usados con GNU/Linux para la instalación de nuevos paquetes o módulos de software, en función de la distribución usada.

Figura 3. Instalador «universal»

Fuente: xkcd.com/1654.

Normalmente, en un entorno personal esto no representa ningún problema, dado que se pueden instalar todas las dependencias y resolver este tipo de errores. Pero en un entorno de producción de una empresa, un cambio en una librería o versión puede tener resultados catastróficos, provocando que muchas dependencias se rompan y se tengan que resolver, dejando todo el sistema en un estado inconsistente.

Una posible solución a este problema es el uso de **entornos virtuales**, los cuales permiten reproducir un escenario concreto de forma que se pueda ejecutar sobre cualquier combinación de hardware y sistema operativo (quizá con algunas restricciones). Una vez obtenida una instalación que se considera adecuada, esta se encapsula en forma de máquina virtual que después puede ser ejecutada por un software, de forma que podemos asegurar su funcionamiento independientemente de la plataforma subyacente. Obviamente, esto tiene un coste en cuanto al rendimiento y en algunos casos en que se requiera un hardware muy específico, no será posible, pero es una solución muy popular, por ejemplo, en una distribución de GNU/Linux entre los estudiantes de una asignatura concreta donde hacen falta unas herramientas y configuraciones específicas.

Hay diferentes niveles:

1) Máquinas virtuales como cajas negras, usando herramientas como, por ejemplo, Parallels Desktop, VirtualBox o anteriormente VMware, con las cuales es posible crear una máquina virtual que se ejecuta de forma aislada, sin interactuar con el resto de procesos que se están ejecutando en la máquina que hace de huésped ejecutando el software adecuado.

2) Herramientas como Docker, que permiten ejecutar en un mismo servidor diferentes máquinas virtuales que se pueden comunicar entre ellas mediante canales. Dado que se comparte el mismo sistema operativo de base, se trata de «contenedores» más ligeros que las máquinas virtuales anteriores, que tienen que contener todos los elementos necesarios, incluyendo el sistema operativo.

3) Herramientas como Kubernetes, que permiten gestionar una «orquesta» de contenedores que se pueden ejecutar en recursos distribuidos y en la nube, añadiendo un nivel de abstracción respecto a las anteriores, puesto que no dependen de una máquina concreta en la que se hospedan las máquinas virtuales o contenedores. De hecho, Kubernetes permite usar Docker como contenedores de las aplicaciones que se quieren distribuir.

A nivel personal no es habitual usar el tercer nivel y raramente el segundo. En cambio, el uso de máquinas virtuales puede ser útil para hacer pruebas y asegurar que el código desarrollado se puede ejecutar correctamente independientemente de la plataforma elegida. En entornos de producción reales, usar herramientas del tercer nivel es cada vez más habitual, dado que los desarrolladores se pueden desentender de los detalles de la plataforma en que se acabará ejecutando su código, creando herramientas que se ejecutan como un servicio, lo que se denomina *SAAS*[2].

[(2)]Acrónimo de *Software As A Service*.

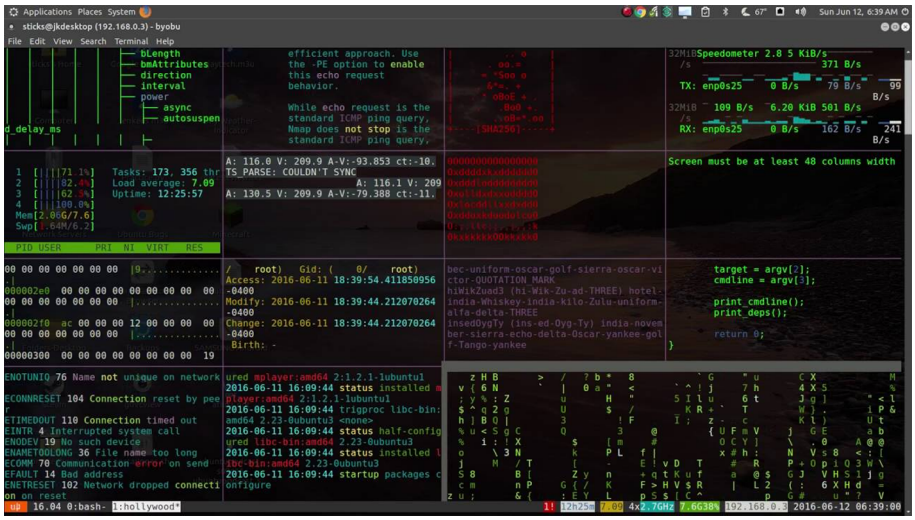
2. La línea de comandos

Resulta un tanto paradójico que una vez configurado un sistema informático con unas capacidades de computación, gráficas y de almacenamiento punteras y que aprovechan el estado del arte en los tres niveles descritos anteriormente, ahora nos centremos en uno de los elementos más primigenios, casi un vestigio, que todavía está disponible (pero en muchos casos escondido) en los sistemas operativos actuales; nos referimos a la **consola o línea de comandos**.

Hay que tener en cuenta que las interfaces gráficas de usuario actuales utilizadas por los sistemas operativos modernos son relativamente recientes, dado el coste del hardware necesario para visualizar con una resolución adecuada los elementos que las componen. Por ejemplo, Windows 3.0, que se puede considerar la primera versión popular del sistema operativo de Microsoft para ordenadores personales, fue lanzado en 1990, casi diez años después de la aparición del ordenador personal de IBM en 1981, el cual incorporaba MS-DOS como sistema operativo. Anteriormente ya había otras propuestas, como la de Apple Macintosh, del año 1984, o la de Commodore Amiga, de 1985. En la actualidad, todos los sistemas operativos modernos ofrecen una interfaz gráfica por defecto, relegando la consola a un segundo término para tareas muy concretas o consideradas solo por administradores expertos.

De hecho, el uso de la consola parece que se ha relacionado, erróneamente, con un uso limitado a expertos, seguramente por su utilización en películas, donde un *hacker* teclea rápidamente comandos incomprensibles y resuelve un problema clave para el desarrollo de la narración. Algunos ejemplos muy conocidos son *Matrix*, con la pantalla de fósforo verde llena de símbolos, *Jurassic Park* o *Tron Legacy*, entre otras. De hecho, la popularidad de la consola es tal que incluso hay una aplicación que simula una pantalla digna del mejor *hacker*, denominada, como no podría ser de otro modo, *Hollywood*, tal como muestra la figura 4.

Figura 4. Ejemplo de pantalla típica de *hackers*



Fuente: elaboración propia.

Sea como fuere, la consola es una herramienta que permite realizar operaciones básicas de mantenimiento del sistema operativo, especialmente cuando estas comprenden muchos pasos distintos o afectan a muchos elementos diferentes (ficheros, directorios, procesos, etc.). No se trata de elegir entre la interfaz gráfica y la consola, pero esta última no puede ser desconocida para cualquier persona que utilice su ordenador de forma intensiva y avanzada.

No pretendemos con esta guía ser exhaustivos en cuanto a lo que se puede hacer desde la línea de comandos; para eso ya hay manuales y otros materiales docentes que describen todas sus capacidades. Nos limitaremos a describir las operaciones más típicas para sacar el máximo provecho del sistema operativo y el hardware disponible más directamente relacionadas con la ciencia de datos.

Ved también

Los manuales y materiales docentes se proporcionan en el apartado «Bibliografía».

2.1. Manipulación de ficheros

Además del mantenimiento del propio sistema operativo, una de las tareas habituales que se realizan desde la línea de comandos es la **manipulación de ficheros**. Esta tarea tiene diferentes niveles en función de las operaciones realizadas:

- **A nivel del sistema de ficheros**: esto incluye listar, crear, renombrar, mover y borrar directorios del árbol de directorios, y también las mismas operaciones con los ficheros que están en dichos directorios. El sistema operativo proporciona comandos básicos para hacer todas estas operaciones.

- **A nivel del contenido de los ficheros**: esto incluye operaciones para detectar patrones mediante expresiones regulares, sustituir caracteres o cadenas, o convertir formatos, tanto a bajo nivel (la codificación usada, por ejemplo UTF-8) como a alto nivel (por ejemplo, entre CSV y JSON), aunque en algunos casos esto requerirá el apoyo de alguna herramienta adicional.

Como muestra de las posibilidades de la línea de comandos del sistema operativo, el siguiente comando busca todos los ficheros con extensión «.zip» desde el directorio raíz del usuario y los mueve a un directorio llamado «zips» dentro del directorio de ficheros temporales.

```
find /home/usuario -iname '*.zip' -exec mv '{}' /tmp/zips/ \;
```

Esto también se puede hacer desde la interfaz gráfica, haciendo una búsqueda y después moviendo todos los ficheros encontrados, pero seguramente con un coste mayor, especialmente si el número de ficheros encontrados es elevado, además de que este comando se puede automatizar para ejecutarlo automáticamente cada día a una hora concreta, por ejemplo.

2.2. Gestión de procesos

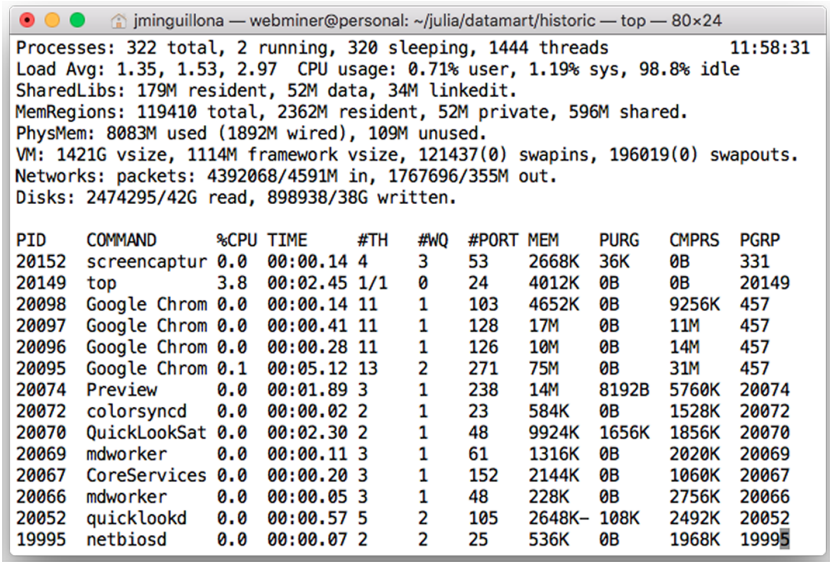
Otra tarea típica que se realiza desde la línea de comandos es la **supervisión de los procesos** que están en ejecución. En este caso, algunas de las operaciones más habituales son las siguientes:

- **Saber qué procesos se están ejecutando**: el comando *ps* (*process status*) muestra los procesos en ejecución en ese momento exacto; el comando *top* lo hace de forma dinámica, actualizando los datos en cada momento y también mostrando el consumo de recursos, como, por ejemplo, la carga de cada CPU y la memoria RAM usada, tal como muestra la figura 5. Una opción más amigable es *htop*, que añade más funcionalidades una vez instalada. Una vez identificado un proceso por su identificador, es posible cambiar su estado con el comando *kill* que, como su nombre indica, permite interrumpir la ejecución de procesos. Cuando un proceso ejecuta varios subprocesos, es posible eliminarlos todos con el comando *killall*.

- **Ejecutar un comando en *background***: si en el momento de ejecutar un comando añadimos & al final, este pasa a ejecutarse en segundo plano, liberando la línea de comandos para seguir ejecutando otras operaciones. Con el comando *nohup* se puede indicar que el proceso que se quiere ejecutar en segundo plano no debe ser interrumpido, asegurando su ejecución completa.

La ejecución del comando *top* genera un resultado parecido al que muestra la figura 5.

Figura 5. Ejemplo de ejecución del comando *top*



Fuente: elaboración propia.

Entonces, si queremos cancelar o suspender uno de los procesos en ejecución, solo hay que conocer su PID y mediante el comando *kill* cambiar su estado; por ejemplo, para matar el proceso llamado *preview* haríamos:

```
kill -9 20074
```

2.3. Automatización de tareas

Finalmente, una de las características más interesantes de la línea de comandos es que, de hecho, se trata de un entorno integrado que permite desarrollar y ejecutar conjuntos de comandos agrupados en lo que se conoce como ***scripts***. Los *scripts* (ficheros de texto con código fuente) permiten crear secuencias de comandos que aprovechan las posibilidades que ofrece el sistema operativo para concatenar operaciones, como, por ejemplo, los *pipes* o el uso de ficheros temporales. Además, el lenguaje que interpreta y ejecuta los *scripts* también proporciona los elementos típicos de cualquier lenguaje de programación, como, por ejemplo, el uso de parámetros, variables, estructuras de control (bifurcación y bucles), funciones, etc. Esto permite crear pequeños programas que automatizan la resolución de los problemas típicos con los que se encuentra un científico de datos cuando tiene que resolver un nuevo reto.

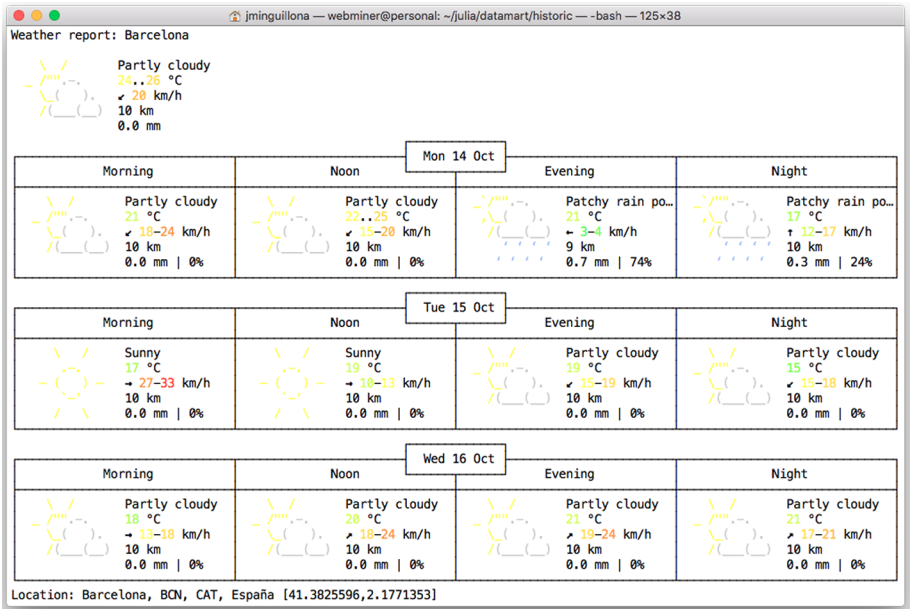
El sistema operativo proporciona una línea de comandos o ***shell***, que es un programa que permite ejecutar operaciones y otros programas de forma interactiva. De hecho, el primer *shell* en entornos Unix se llamó *sh* y desde entonces otros muchos entornos han usado estas siglas, como por ejemplo, Ksh, el popular Bash o Zsh. Aunque hay una *shell* por defecto, normalmente es posible usar diferentes *shells*, cosa que se indicará en la primera línea del fichero de texto que contiene el *script*, tal como mostraba el ejemplo de la figura 3, que usaba Bash como intérprete de los comandos.

Como ejemplos avanzados de lo que se puede llegar a hacer con los *scripts*, hay una colección llamada *Bash Snippets* que incluye herramientas para hacer cambios de moneda con el cambio actual, acceder a la predicción meteorológica o encriptar y desencriptar ficheros, entre otros ejemplos, combinando las capacidades de Bash con otras herramientas como las descritas con anterioridad. Así, una vez instalados los Bash Snippets, disponemos de un comando para obtener la predicción meteorológica en una ciudad concreta de la forma siguiente:

```
weather Barcelona
```

El resultado obtenido por pantalla, usando solo caracteres ASCII, es el siguiente:

Figura 6. Resultado de la ejecución de un *script Bash (weather)*



Fuente: elaboración propia.

Aunque queda fuera del alcance y objetivos de este material docente, recomendamos echar un vistazo a los Bash Snippets para ver cómo aprovechan al máximo los recursos que el sistema operativo nos proporciona para trabajar desde la línea de comandos.

3. Escenarios de uso

Como hemos comentado anteriormente, el uso de la consola permite ejecutar *scripts*, los cuales resuelven problemas concretos combinando comandos y herramientas diferentes. Sin entrar en detalles técnicos sobre cómo hacerlo, algunos posibles problemas que se pueden resolver usando las posibilidades que ofrece la línea de comandos son como los que se describen a continuación.

3.1. Datos en abierto

Cada vez es más habitual que la Administración publique datos en abierto, mediante portales en los que es posible encontrar conjuntos de datos que describen muchos de los aspectos relacionados con nuestra sociedad, como, por ejemplo, datos sociodemográficos, del tránsito, contaminación, etc., tal como muestra el ejemplo de la figura 7. En muchos casos, el periodismo de datos empieza con el análisis de uno o más conjuntos de datos que pueden arrojar luz para responder alguna pregunta relacionada con hechos relevantes de nuestra sociedad. Descargar estos ficheros desde su origen, limpiarlos y filtrar la información deseada y fusionarlos en uno solo para su análisis posterior son tareas que se pueden automatizar usando *scripts* desde la línea de comandos.

Figura 7. Categorías del catálogo de datos.gob.es

Fuente: datos.gob.es.

Por ejemplo, cada día se publica la lista de precios de los carburantes en todas las estaciones terrestres (también existe el equivalente de las marítimas), la cual se puede descargar en diferentes formatos, entre ellos como fichero JSON, mediante el comando siguiente (es una sola línea):

```
curl "https://sedeaplicaciones.minetur.gob.es/serviciosrestcarburantes/precioscarburantes/
EstacionesTerrestres/" | jq
```

Si nos quedamos con los primeros elementos de la lista retornada por el comando anterior procesado con *Jq*, una herramienta orientada a manipular documentos JSON, obtenemos:

```
{
 "Fecha": "14/10/2019 16:18:57",
 "ListaEESSPrecio": [
 {
 "C.P.": "01240",
 "Dirección": "CALLE GASTEIZBIDEA, 59",
 "Horario": "L-D: 24H",
 "Latitud": "42,842917",
 "Localidad": "ALEGRIA-DULANTZI",
 "Longitud (WGS84)": "-2,519194",
 "Margen": "D",
 "Municipio": "Alegría-Dulantzi",
 "Precio Biodiesel": null,
 "Precio Bioetanol": null,
 "Precio Gas Natural Comprimido": null,
 "Precio Gas Natural Licuado": null,
 "Precio Gases licuados del petróleo": null,
 "Precio Gasoleo A": "1,279",
 "Precio Gasoleo B": null,
 "Precio Gasolina 95 Protección": "1,359",
 "Precio Gasolina 98": null,
 "Precio Nuevo Gasoleo A": null,
 "Provincia": "ÁLAVA",
 "Remisión": "dm",
 "Rótulo": "ES DULANTZI REPSOL",
 "Tipo Venta": "P",
 "% BioEtanol": "0,0",
 "% Éster metílico": "0,0",
 "IDEESS": "14209",
 "IDMunicipio": "1",
 "IDProvincia": "01",
 "IDCCAA": "16"
 },
 …
 ]
}
```

Como se puede ver, para cada estación tenemos su localización (y geolocalización) y también el precio de diferentes carburantes, por lo que se puede usar para hacer un seguimiento y análisis de los precios de una zona a lo largo del tiempo, por ejemplo.

3.2. Generación de corpus de datos

Wikipedia es el recurso digital colaborativo más grande construido por la humanidad: contiene millones de artículos sobre temas diversos en más de trescientos idiomas diferentes. Es decir, es una base de conocimiento inconmensurable, que crece cada día. Además, proyectos como **Wikidata** añaden una capa semántica que permite establecer relaciones entre los elementos y hacer

búsquedas más complejas, habilitando lo que se conoce como **web semántica**. Muchos proyectos de investigación usan Wikipedia como base para la construcción de corpus de palabras, usados para entrenar sistemas de traducción automática, sistemas inteligentes de pregunta/respuesta, etc., para lo que es necesario descargar grandes ficheros de datos que habrá que procesar posteriormente.

De forma periódica (cada pocos días) se publican vertidos de memoria (*dumps*) de todo lo que hay en la Wikipedia, incluyendo los contenidos, pero también las ediciones realizadas, datos de los usuarios, etc. Los *dumps* son ficheros XML o SQL que se pueden procesar para extraer la información deseada. En paralelo, la Wikipedia dispone de una API que permite acceder a datos de forma dinámica, mediante consultas parametrizables. Además, Wikidata incluye un punto SPARQL para hacer consultas más avanzadas, no solamente del contenido, sino también de las relaciones entre las entidades representadas en las páginas de la Wikipedia.

Por ejemplo, desde la línea de comandos podemos ejecutar el comando siguiente (es una sola línea):

```
curl "https://www.wikidata.org/w/api.php?format=json&action=
wbgetentities&ids=Q1492" | jq '.entities[].claims.P625'
```

Con el comando *curl* hacemos una petición al API de Wikidata, con el objetivo de obtener todas las entidades (*wbgetentities*) relacionadas con el elemento Q1492 (que corresponde a la página de Wikipedia sobre Barcelona). El resultado es un fichero JSON que es formateado adecuadamente mediante el comando *jq*, con el cual nos quedamos con la propiedad *P625*, que corresponde a la geolocalización del elemento, en este caso la ciudad de Barcelona. El resultado obtenido es:

```
[
 {
 "mainsnak": {
 "snaktype": "value",
 "property": "P625",
 "hash": "608b0ebef62ccb5a0b98f9f5c8953fa949ede0e1",
 "datavalue": {
 "value": {
 "latitude": 41.414802777778,
 "longitude": 2.1335555555556,
 "altitude": null,
 "precision": null,
 "globe": "http://www.wikidata.org/entity/Q2"
 },
 "type": "globecoordinate"
 },
 "datatype": "globe-coordinate"
 },
 "type": "statement",
 "id": "q3042433$A2E9939B-838B-4D6F-B383-14C5F49B0F03",
 "rank": "normal"
 }
]
```

Es cierto que este tipo de tareas también se podrían hacer usando Python, por ejemplo, pero la línea de comandos nos ofrece un entorno de trabajo en el que se pueden ejecutar consultas como esta de forma sencilla, combinando comandos y herramientas típicas de un entorno de trabajo de un científico de datos.

4. Lecturas relacionadas

Como cierre de esta introducción general a la programación en *scripting*, existen unos cuantos artículos que, aunque son muy antiguos, resultan muy interesantes para entender el razonamiento lógico que ha seguido el desarrollo de todas las herramientas relacionadas con la consola del sistema operativo y que permiten entender mejor por qué algunas cosas todavía son como son en la actualidad.

4.1. El lenguaje de programación de comandos de Unix

Publicado en 1978 por Stephen Bourne, creador de la primera consola de comandos completa para Unix (*shell*), este artículo describe el lenguaje de programación (según palabras del autor) que permitía gestionar los recursos de un ordenador ejecutando el sistema operativo Unix.

El artículo se limita a describir la sintaxis y los comandos básicos que se pueden ejecutar desde la línea de comandos, y también agruparlos en *scripts* incluyendo toda la parafernalia típica de los lenguajes de programación, como, por ejemplo, variables, estructuras de control, operaciones y la gestión de errores. Nos obstante, resulta muy interesante, pues introduce conceptos como la concatenación de operaciones (*pipe*) o la redirección de la entrada/salida; realmente es un buen ejemplo de cómo esconder detalles del sistema operativo para hacer operaciones más complejas de forma sencilla, pero siempre desde un planteamiento de lenguaje de programación, no simplemente de *scripting*.

Posteriormente (en 1983), el mismo autor publicó una versión más actualizada en una de las revistas más importantes de la época en cuanto a la diseminación de la informática entre el público general, denominada *Byte*, en un monográfico sobre el sistema operativo Unix. La revista también es un muy buen ejemplo del boom que se vivió durante los años ochenta en cuanto a la informática de consumo, los anuncios que aparecen son un escaparate perfecto de lo que estaba sucediendo.

Bourne, S. R. (1978). «UNIX time-sharing system: The UNIX shell». *The Bell System Technical Journal* (vol. 57, n.º 6, págs. 1971-1990). Disponible en: ieeexplore.ieee.org/abstract/document/6770407.

Bourne, S. R. (1983). «The Unix shell». *Byte Magazine* (vol. 10, n.º 8). Disponible en: archive.org/stream/byte-magazine-1983-10/1983_10_byte_08-10_unix#page/n187/modo/2up

4.2. El proyecto GNU

Este artículo de Richard Stallman (conocido por *rms*) del año 1998, gran defensor del software libre y desarrollador de gran parte del software que forma parte del universo GNU/Linux, describe cómo fue diseñado e ideado a partir de la necesidad creada por la imposibilidad de compartir software con los ordenadores disponibles en el laboratorio del MIT donde trabajaba, repleto de máquinas ejecutando UNIX pero con licencias propietarias. Así nació el proyecto GNU, acrónimo recursivo de *GNU is not Unix*, nombre típico en este contexto.

El artículo presenta la línea argumental de Stallman y otros desarrolladores a la hora de decidir los aspectos ligados a las licencias, los primeros desarrollos (era especialmente importante disponer de un compilador para poder crear programas libres a partir de código abierto), y también la creación de la Free Software Foundation (FSF) y el impulso al software libre. También explica por qué hablamos de GNU/Linux, que no es más que el uso de Linux como núcleo en el que ejecutar toda la familia de herramientas que componen GNU.

El artículo acaba listando los peligros del movimiento, teniendo en cuenta que se basa en el trabajo voluntario de miles de desarrolladores, aunque son pocos los que participan en proyectos críticos. Entre otros, Stallman menciona las patentes, los detalles ocultos en el hardware (cosa que dificulta el desarrollo de software para los nuevos dispositivos), el software propietario, con Microsoft al frente como enemigo número uno, y la dificultad de documentar todo de forma adecuada.

Stallman, R. (1998). «The GNU project». *Open Sources*. Disponible en: noemalab.eu/org/sections/ideas/ideas_articles/pdf/stallman_eng.pdf.

4.3. Los lenguajes de *scripting*

Este artículo, elaborado por John Ousterhout en 1998, es una compilación de las particularidades de los lenguajes de *scripting* que los hacen diferentes (e interesantes) respecto a los lenguajes de programación tradicionales. El autor destaca que los lenguajes de *scripting* se basan en la existencia de otros componentes ya existentes (quizá programados en otros lenguajes de programación) y que, por lo tanto, son más una especie de pegamento que permite unir operaciones para conseguir el resultado obtenido, favoreciendo la resolución de pequeños problemas y la experimentación.

El autor hace una compilación de diferentes lenguajes de *scripting* (obviamente, un poco obsoleto hoy en día) y compara diferentes herramientas desarrolladas con lenguajes de programación tradicionales (C, C++, Java, etc.) con versiones más ligeras programadas con un lenguaje de *scripting* como es Tcl. El argumento principal para usar lenguajes de *scripting* es cuando se tienen que combinar tecnologías diferentes, como, por ejemplo, la creación de interfaces

gráficas, el acceso a recursos de internet (en aquella época algo todavía muy poco desarrollado) y el uso de diferentes *frameworks* para la creación de aplicaciones.

Ousterhout, J. K. (1998). «Scripting: Higher level programming for the 21st century». *Computer* (vol. 31, n.º 3, págs. 23-30). Disponible en: ieeexplore.ieee.org/abstract/document/660187.

4.4. Los orígenes del lenguaje AWK

En un artículo del año 1979, Alfred V. Aho, Brian W. Kernighan y Peter J. Weinberger presentaron el lenguaje llamado *AWK* (no de forma fortuita, como se puede deducir), orientado a la detección y procesamiento de patrones. Según los autores, AWK es una extensión de las funcionalidades del comando *grep*, tanto en cuanto al uso de expresiones regulares para detectar patrones como las acciones que se pueden realizar para procesarlos.

El artículo describe los elementos que caracterizan el lenguaje: su estructura, orientada a analizar líneas de ficheros de texto, con un bloque que se ejecuta una vez antes de procesar ninguna línea, un bloque que se ejecuta por cada línea de entrada y un bloque final que se ejecuta una vez después de procesar todas las líneas; el concepto de registro (de hecho, la línea) y de campo; la salida con las operaciones *print* y *printf* (como en el lenguaje C); las expresiones regulares para analizar patrones; las funciones incorporadas (también muy parecidas a las del lenguaje C) y las estructuras de control.

Los autores finalizan el artículo con las consideraciones de diseño, teniendo en cuenta los objetivos del lenguaje, orientado a resolver pequeños problemas relacionados con los ficheros de texto, y una comparación del coste de diferentes operaciones realizadas con AWK y otros comandos del sistema operativo o lenguajes de programación. Aunque hoy en día estas cifras están del todo obsoletas, sirven para mostrar uno de los objetivos de los autores: la facilidad de uso de AWK (en un sentido de coherencia con todas sus posibilidades), por encima de su eficiencia para operaciones concretas como contar líneas, cuando existe un comando específico (wc) para realizar esta tarea de forma mucho más eficiente.

Aho, A. V.; Kernighan, B. W.; Weinberger, P. J. (1979). «Awk—a pattern scanning and processing language». *Software: Practice and Experience* (vol. 9, n.º 4, págs. 267-279). Disponible en: onlinelibrary.wiley.com/doi/abs/10.1002/spe.4380090403.

4.5. Espacio de recursos de ciencia de datos

Finalmente, recomendamos a los estudiantes explorar el espacio de recursos de ciencia de datos de la UOC, donde se pueden encontrar descripciones y pequeños ejemplos de uso de herramientas relacionadas con estos materiales (y otras propias del ámbito). El enlace a este recurso es: «Espacio de recursos de ciencia de datos».

Bibliografía

Albing, C.; Vossen, J. P.; Newham, C. (2007). *Bash Cookbook: Solutions and Examples for bash Users*. California: O'Reilly Media.

Dougherty, D.; Robbins, A. (1997). *Sed & Awk, UNIX Power tools* (2.ª ed.). California: O'Reilly Media.

Friedl, J. E. F. (2006). *Mastering regular expressions*. California: O'Reilly Media.

Janssens, J. (2014). *Data Science at the Command Line: Facing the Future with Time-tested Tools*. California: O'Reilly Media. Disponible en: www.datascienceatthecommandline.com.

Shotts, W. (2012). *The Linux command line: a complete introduction*. San Francisco: No Starch Press. Disponible en: linuxcommand.org/tlcl.php.