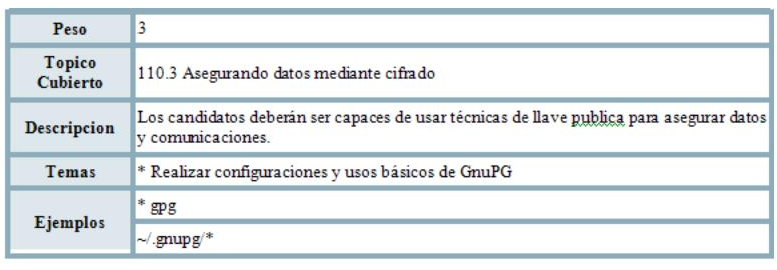
Asegurando datos mediante cifrado 



Peso: Indica el valor de importancia que tiene este tópico en la certificacion.

Tópico Cubierto: Indica según el programa de certificacion LPI que topico le corresponde a este tema.

Descripción: Un resumen de lo que se verá.

Temas: Un resumen de los conceptos primordiales que están cubiertos.

Ejemplos: Palabras claves que se tienen que tener en cuenta.

Introducción

En este topico vamos a explicar sobre como usar la herramienta GnuPG y su funcionamiento.

GnuPG

GnuPG es una aplicación que sirve para encriptar mensajes, documentos … GnuPG usa un sistema de claves públicas lo que quiere decir que cada usuario tiene una clave privada y una clave pública.  
La clave privada es la que se usa para desencriptar aquello que nos envían encriptado con nuestra clave pública, La clave privada es una clave que solo ha de conocer el propietario ya que si alguien más la conociese podría desencriptar lo que nos mandan encriptado.  
La clave pública es la que se da a la gente para que nos manden cosas encriptadas y usarán para encriptar aquello que nos quieran pasar.

Funcionamiento

GNU Privacy Guard (GPG) es una implementación de código abierto de la norma OpenPGP (RFC 4880) ,que le permite cifrar y firmar digitalmente los datos y las comunicaciones.

Por ejemplo, usted puede cifrar los archivos en el sistema de archivos Linux. También puede cifrar y firmar digitalmente mensajes de correo electrónico.

GPG cifra los mensajes usando pares de claves individuales asimétricas generadas por los usuarios. Las claves públicas pueden ser compartidas con otros usuarios de muchas maneras, un ejemplo de ello es depositándolas en los servidores de claves. Siempre deben ser compartidas cuidadosamente para prevenir falsas identidades por la corrupción de las claves públicas. También es posible añadir una firma digital criptográfica a un mensaje, de esta manera la totalidad del mensaje y el remitente pueden ser verificados en caso de que se desconfíe de una correspondencia en particular.

GnuPG también soporta algoritmos de cifrado simétricos, por ejemplo CASTS.

GPG no usa algoritmos de software que están restringidos por patentes, entre estos se encuentra el algoritmo de cifrado IDEA que está presente en PGP casi desde sus inicios. En su lugar usa una serie de algoritmos no patentados como ElGamal, CAST5, Triple DES (3DES), AES y Blowfish. También es posible usar IDEA en GPG descargando un plugin extra, sin embargo este puede requerir una licencia para usuarios de algunos países en donde esté patentada IDEA.

GPG es un software de cifrado híbrido que usa una combinación convencional de criptografía de claves simétricas para la rapidez y criptografía de claves públicas para el fácil compartimiento de claves seguras, típicamente usando recipientes de claves públicas para cifrar una clave de sesión que es usada una vez. Este modo de operación es parte del estándar OpenPGP y ha sido parte del PGP desde su primera versión.[2]

GPG utiliza muchos algoritmos de encriptación, incluyendo los siguientes:

Cifrado simétrico

§         AES

§         3DES

§         Blowfish

Cifrado asimétrico

§         Elgamal

§         RSA

§         Hashes:

§         MD5

§         SHA-1 and -2

§         RIPEMD-160

Las firmas digitales

§         DSA

§         RSA

Conceptos Básicos

Sistemas de Claves Públicas

Para poder entender mejor el sistema de codificación usado por los sistemas de claves asimétricas (ie. claves públicas y privadas), es necesario entender las diferencias con los sistemas de claves simétricas (ie. claves secretas).

Los sistemas de cifrado con clave simétrica son aquéllos en los que la clave que se usa para cifrar una serie de datos, es la misma que la que se usará para descifrar estos datos. En el caso del correo electrónico, el remitente cifraría el mensaje con una clave secreta, y para que el destinatario pueda descifrarlo, necesitaría haber obtenido previamente esta misma clave de un modo «seguro», o sea de modo que la clave no haya podido ser interceptada durante la entrega. Si no tenemos la completa seguridad de que el intercambio de la clave ha sido seguro, la validez de este sistema es nula.

Por el contrario, los sistemas de cifrado con claves asimétricas usan claves distintas para el cifrado y posterior descifrado de los datos. En un caso como el anterior, el remitente usaría la clave pública del destinatario para cifrar el mensaje, y el destinatario descifraría el mensaje con su propia clave privada. Así pues, la clave privada no debe ser accesible para nadie que no sea el propio dueño de la misma, mientras que la clave pública, puede ser entregada a cualquier persona. En un sistema de cifrado bien implementado, la clave privada no debe derivar nunca de la clave pública.

Firmas Digitales

El concepto de la firma digital se basa en la verificación de la autoría de un mensaje. Esto quiere decir que se puede comprobar que el destinatario del mensaje puede comprobar que el «supuesto» remitente es quien afirma ser. Para ello, el remitente, una vez compuesto el mensaje, lo firma usando su propia clave privada. El destinatario, una vez ha recibido el mensaje, comprobará la veracidad de éste, esto es, lo verificará usando la clave pública del remitente.

Este método es de especial utilidad para reducir riesgos de seguridad en nuestros sistemas (nos podrían enviar un supuesto parche para un programa, y éste en realidad ser un virus o un troyano); también podrían enviarnos información o datos, como provenientes de una fuente lícita o fiable. En ambos casos, no sería muy difícil falsificar la dirección y  nombre del remitente, pero sí imposible falsificar la firma digital de éste.

Como ya hemos dicho, la verificación de un mensaje firmado digitalmente se lleva a cabo mediante el uso de la clave pública del remitente sobre el texto del propio mensaje. De este modo no sólo podemos verificar la identidad del autor, sino que también podemos comprobar la integridad del mensaje, ya que la firma digital ha sido generada con el texto y la clave privada. Así pues, una alteración o modificación del texto «a posteriori», o cualquier manipulación del mensaje (especialmente si hacemos uso de las especificaciones MIME/PGP), daría como resultado un error en la verificación.

Anillos de Confianza

Un punto flaco en los algoritmos de clave asimétrica es la transmisión del código público. Es posible que una persona ponga en circulación código con un identificador de usuario falso. Si se codifican mensajes con este pseudo código, el intruso los puede decodificar y leerlos.

La solución PGP (y por consiguiente la solución GnuPG) está en firmar los códigos. La clave pública de un usuario puede estar firmada con las claves de otros usuarios. El objetivo de estas firmas es el de reconocer que el UID (identificador de usuario) de la clave pertenece al usuario a quien dice pertenecer. A partir de ahí es un problema de cada usuario de GnuPG el decidir hasta qué punto se puede fiar de la firma. Una clave se puede considerar fiable cuando se confía en el remitente y cuando se sabe con seguridad que dicha clave pertenece a éste. Sólo cuando se puede confiar plenamente en la clave del firmante, se puede confiar en la firma que acompaña a la clave de un tercero.  Para tener la certeza de que la clave es correcta hay que compararla con la huella digital por medio de canales fiables (por ejemplo, podríamos buscar el teléfono en la guía y llamarle, y que nos la dijera de palabra para poder compararla), antes de darle una confianza absoluta.

Límites de Seguridad

Si lo que se desea es mantener la confidencialidad de los datos que se poseen, no basta con determinar qué algoritmo de cifrado se va a usar; también es necesario pensar en la seguridad general del sistema. En principio, PGP está considerado como suficientemente seguro, y hasta el momento no se sabe que haya habido ningún incidente en el que una clave PGP haya sido descodificada. Pero eso no significa que todo lo cifrado sea seguro; si la NSA (Agencia de  Seguridad Nacional de los EE.UU.) hubiera conseguido descodificar una clave PGP mediante criptoanálisis, análisis del código, o cualquier otro modo, no es probable que lo hicieran público. Pero aún en el caso de que las claves PGP fueran a todas luces imposibles de decodificar, otros tipos de ataques a la seguridad pueden ser utilizados. A principios de Febrero fue detectado un troyano que buscaba las claves PGP en el disco duro, y las transfería al atacante mediante FTP. Si en este caso hubiéramos escogido una contraseña débil o fácil, un simple análisis que consistiera en un «ataque de diccionario» la descubriría en poco tiempo.

Otra posibilidad técnica, aunque más difícil, es la de los troyanos que recogen entradas de teclado y las transmiten al asaltante. También es posible, aunque muy difícil, pasar el contenido de una pantalla a otra. En este último caso no sería necesario ningún análisis sobre datos cifrados, ya que se obtendrían «pre-cifrados».

Por todo esto es necesaria una planificación de la seguridad que esté bien prevista y que minimice los riesgos.

La idea no es la de recrear una atmósfera de paranoia entre la gente, sino dejar claro que para implementar un sistema seguro no basta con la instalación de un programa criptográfico, que si bien es un paso hacia un sistema más seguro, no es una solución completa. Troyanos como el aparecido en Marzo de 1999 (Melissa) probaron que muchas compañías no se encuentran preparadas en temas de seguridad.

Uso

Creación de claves

Lo primero que hay que hacer una vez que se tiene GnuPG instalado es crear nuestra clave pública y privada. Para hacerlo hay usar el comando gpg --gen-key.

# gpg --gen-key

gpg (GnuPG) 1.4.12; Copyright (C) 2012 Free Software Foundation, Inc.

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:

   (1) RSA and RSA (default)

   (2) DSA and Elgamal

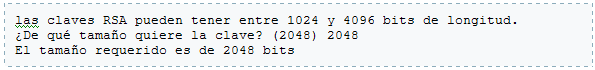
   (3) DSA (sign only)

   (4) RSA (sign only)

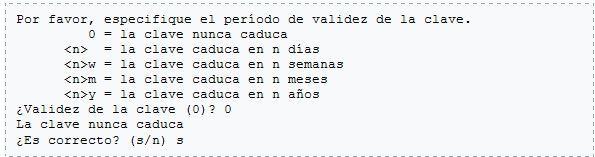
Your selection?

Al ser la primera vez que se ejecuta nos crea un directorio en el que guardará el fichero de configuración así como los archivos secring.gpg y pubring.gpg. En el primero se almacenarán las claves privadas y en el segundo las claves públicas.

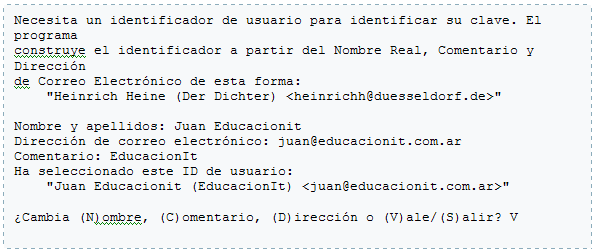
La primera pregunta que hace es qué tipo de clave queremos. Lo normal suele ser seleccionar la primera opción (DSA and ElGamal) que nos permite encriptar y firmar.



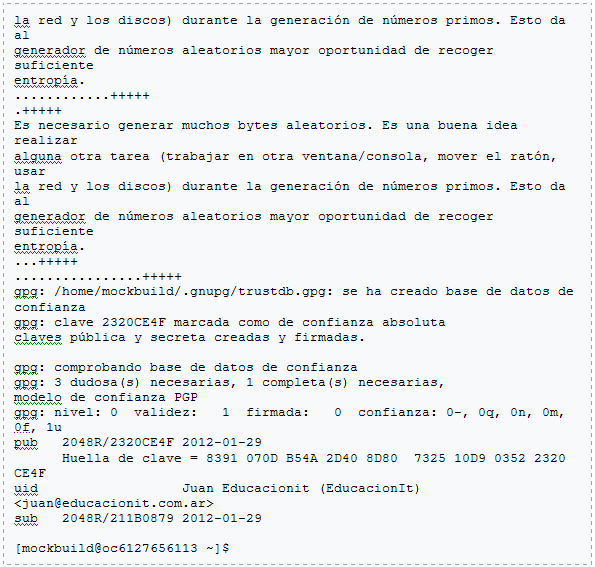
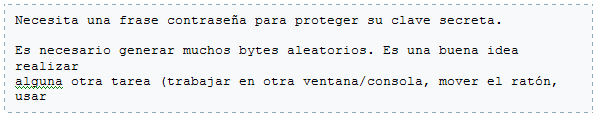
La siguiente pregunta es el tamaño de las claves que se puede elegir entre 1024 y 4096 bits. Por defecto se recomienda 2048, a mayor tamaño más segura es la clave. También a mayor tamaño más tiempo lleva encriptar y desencriptar.



La siguiente pregunta es cuanto tiempo de validez queremos que tenga la clave. La periodicidad se puede poner que no caduque nunca, que dure ciertos semanas, meses o años. En el caso de poner que caduque al cabo de cierto tiempo habrá que volver a generar las claves y volver a mandar la nueva clave pública a aquellos que usaban la que ha caducado. Por defecto viene la opción 0 que es que no caduque nunca.



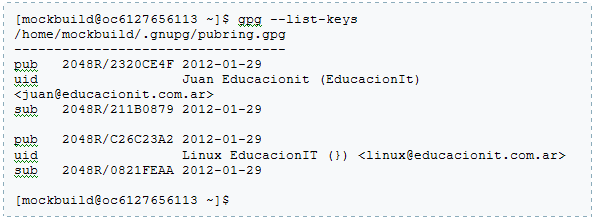
Ahora pregunta nuestro nombre y apellidos, dirección de correo y un comentario para la llave. Una vez introducidos todos los datos nos muestra cual es nuestro ID de usuario que lo crea a partir de los datos que le hemos introducido antes. Luego pregunta si queremos cambiar algún dato o si están bien los datos. Si están correctos respondemos “V” y sigue adelante el proceso.



Por último se pregunta cuál va a ser el password para nuestra clave privada. Al introducir el password no se ve nada de lo que se escribe ni se ve avanzar el cursor. Después de introducirla nos vuelve a preguntar el password y si coincide con el primer password comienza la generación de las claves. Cuando se produce el proceso de generación de las claves es buena idea reproducir mp3, mover el ratón … para que se generen números aleatorios y se creen antes las claves.

Ver claves públicas disponibles

Para ver las claves públicas que tenemos disponibles hay que hacerlo con el comando gpg –list-keys. Esto lo que haces listar las claves que hay disponibles dentro del fichero pubring.gpg.

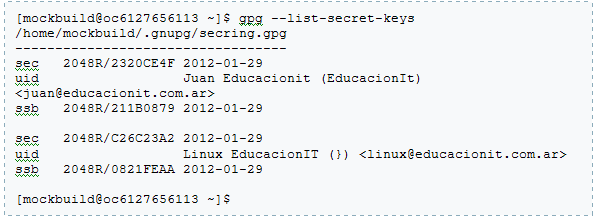


El identificador de las claves es lo que hayamos metido en el nombre, en el apellido, en la dirección de correo o el número que aparece después del 1024D al hacer gpg –list-keys. Si en algún caso coincide el ID se mostrarán los que coinciden.



Ver claves privadas disponibles

Para ver las claves privadas que tenemos disponibles hay que hacerlo con el comando gpg –list-secret-keys. Esto lo que haces listar las claves que hay disponibles dentro del fichero secring.gpg.



Borrar claves de los anillos

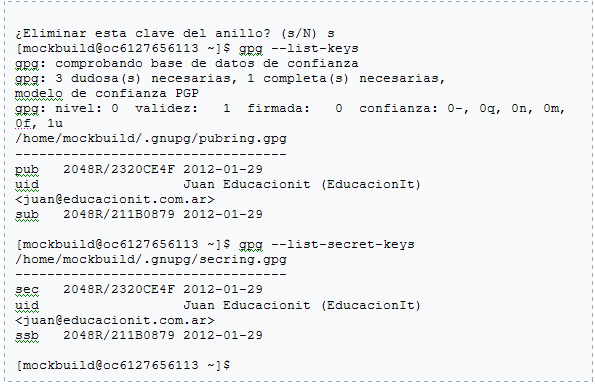
Se llama anillos a los archivos en los que se guardan las claves públicas y las privadas. Generalmente donde se guardan las claves públicas es el archivo pubring.gpg y en el que se guardan las claves secretas secring.gpg. Si se quiere borrar alguna clave primero hay que borrar la clave privada y después la pública. Si se intenta borrar primero la clave pública y esta tiene asociada una clave privada da un mensaje de error.



Para borrar claves privadas se hace con el comando gpg –delete-secret-key ClaveID

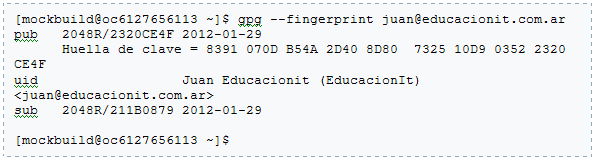
Para borrar claves públicas se hace con el comando gpg –delete-key ClaveID





Ver huella de la clave

Las claves están identificadas por lo que se llama huella. La huella es una serie de números que se usa para verificar si una clave pertenece realmente al propietario. Si se recibe una clave podemos ver cuál es su huella y luego pedirle a su propietario que nos diga su huella. Si ambas coinciden la clave es correcta y no ha sido manipulada. Si no fuera igual es que ha sido modificada. La huella es como el md5 que verifica que un archivo no ha sido manipulado.



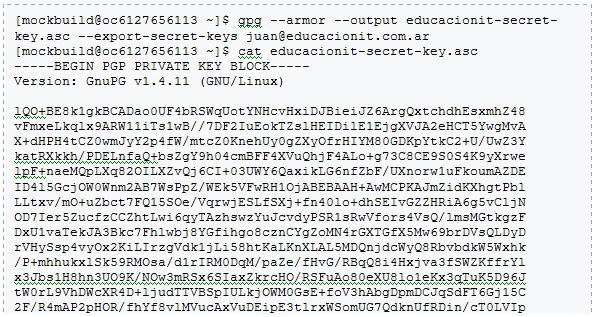
Exportar claves

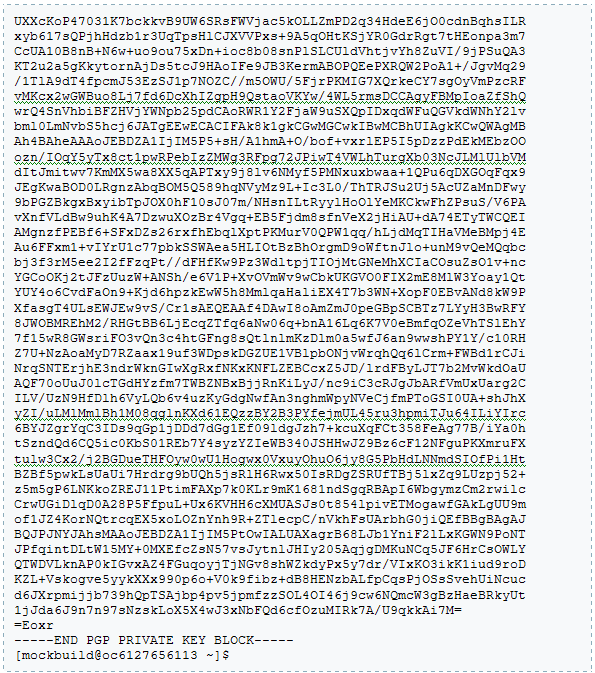
Las claves se pueden exportar a ficheros para que las podamos distribuir entre la gente que queremos que nos encripte o firme cosas o bien porque vamos a formatear el equipo y necesitamos salvarlas.

Para exportar la clave pública se hace poniendo gpg –armor –output fichoeDeSalida –export ClaveID



Para exportar la clave privada se haría poniendo gpg –armor –output fichoeDeSalida –export-secret-key ClaveID

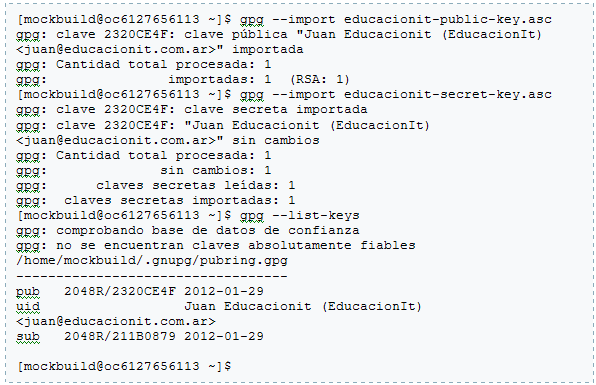




Si quisiéramos salvar todas las claves que tenemos valdría con copiar los archivos pubring.gpg y secring.gpg y luego cuando vayamos al nuevo equipo ponerlas en el directorio de GnuPG.

Importar claves

Si se quiere importar claves nuevas porque por ejemplo hemos formateado el equipo y queremos volver a tener nuestras claves las importamos con el comando gpg –import ClaveID. En el apartado anterior se han salvado las claves pública y privada pues ahora vamos a importarlas. Primero importamos la pública y luego la privada.



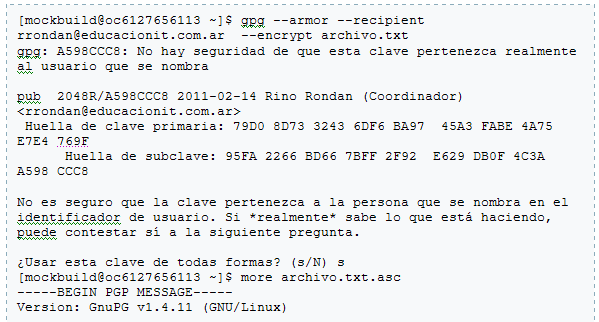
Ahora si queremos importar la clave de una amigo pues se haría igual.



Encriptar mensajes

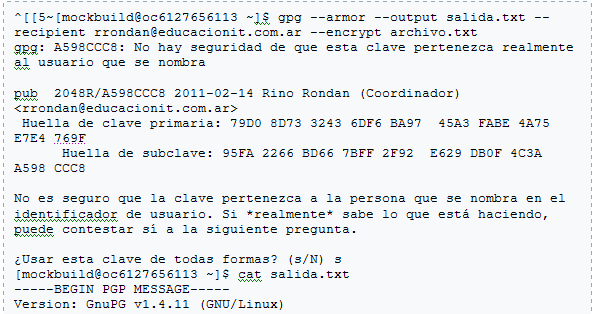
Si se quiere encriptar mensajes se puede hacer poniendo gpg –armor –recipient ClaveID –encrypt mensaje. Si por ejemplo queremos encriptar el archivo a.txt habría que poner

# gpg --armor --recipient prueba@prueba.com --encrypt a.txt





También se puede encriptar a un fichero en concreto con la opción –output nombreFichero



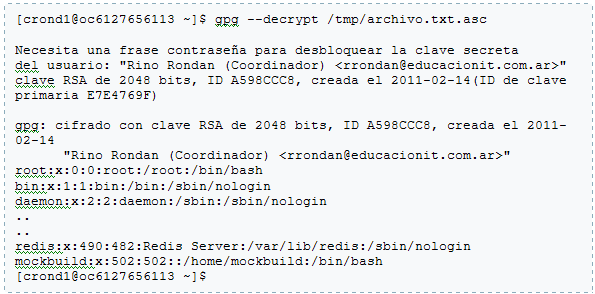


https://alumni.educacionit.com/content/289/1007/images/image38.png

Desencriptar mensajes

Para desencriptar el mensaje que hemos encriptado antes hay que poner gpg –decrypt archivo. Para el caso anterior sería

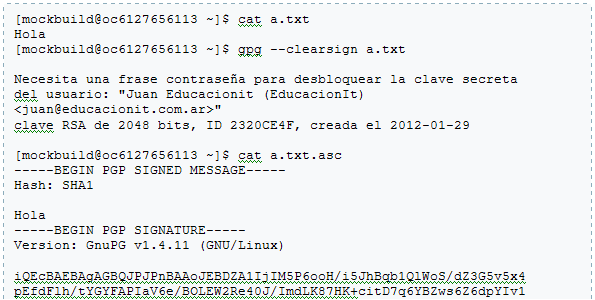
# gpg --decrypt a.txt.asc.

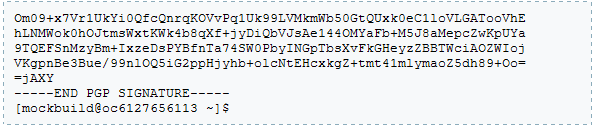


Cuando desencriptamos algo se pide la password de nuestra clave para poder desencriptarlo. Para nuestro caso tenemos el archivo a.txt.asc encriptado al desencriptarlo nos deja el archivo a.txt y nos muestra su contenido.

Firmar mensajes

Firmar mensajes sirve para que cuando a alguien le llegue un mensaje que hemos firmado la persona que lo ha recibido verifique con GnuPG que la firma es buena y que entonces hemos sido nosotros quien le ha enviado el mensaje. Por ejemplo vamos a firmar el archivo a.txt para ello se pondría gpg --clearsign a.txt. Esto nos creará el archivo a.txt.asc con el contenido que se ve en la imagen.





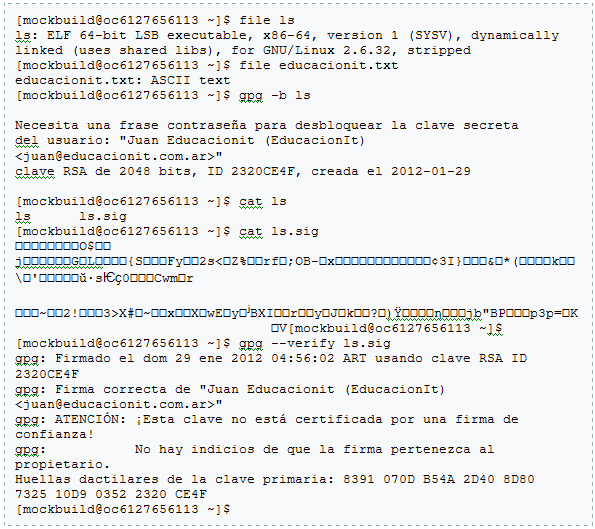
Para firmar algo se pide la contraseña para poder firmarlo. Como se ve en la imagen lo que se ha hecho en el fichero firmado es añadir unas líneas que contienen la firma.

A la hora de firmar si se firma con el parámetro –sign en lugar de –clearsign nos generará un fichero de salida en binario con extensión .gpg. Para validar la firma y ver el contenido hay desencriptarlo con la opción --decrypt.



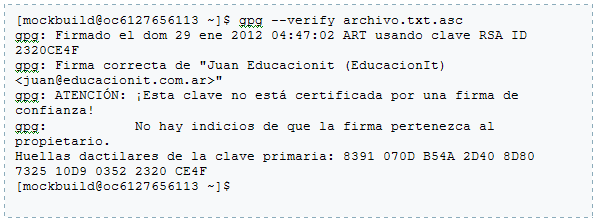
La firma también se puede hacer que se muestre en un fichero aparte con la opción -b. Esta opción se suele usar para firmar archivos

binarios.

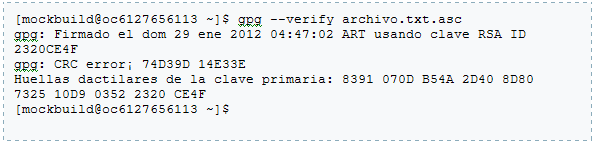


Verificar mensajes firmados

Para verificar mensajes firmados se hace poniendo gpg --verify mensaje. Para el caso anterior sería poner gpg –verify a.txt.asc



Si la firma no fuese correcta podríamos ver un mensaje como el siguiente:



Trabajar con claves en servidores

Podemos buscar claves públicas de gente a la que queremos enviar mensajes cifrados en servidores de claves. Para hacer una búsqueda se hace poniendo los parámetros --keyserver NombreDelServidor --search-keys ClaveID. Si encuentra claves que coinciden con esa ID nos las muestras, luego tenemos tres opciones mostrar los siguientes si es que se han encontrado muchos, poner el número del registro que nos interesa (en ese caso nos importa la clave al anillo de claves públicas) o salir.

Buscar claves públicas en servidores

Si queremos importar una clave en concreto se hace con los parámetros --keyserver NombreDelServidor --recv-keys ClaveID .

Importar una clave desde un servidor

Si queremos subir una clave a un servidor para que esté disponible para la gente se hace con los parámetros --keyserver NombreDelServidor --send-keys ClaveID

Exportar una clave a un servidor

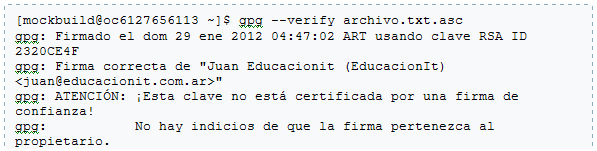
GnuPG subshell

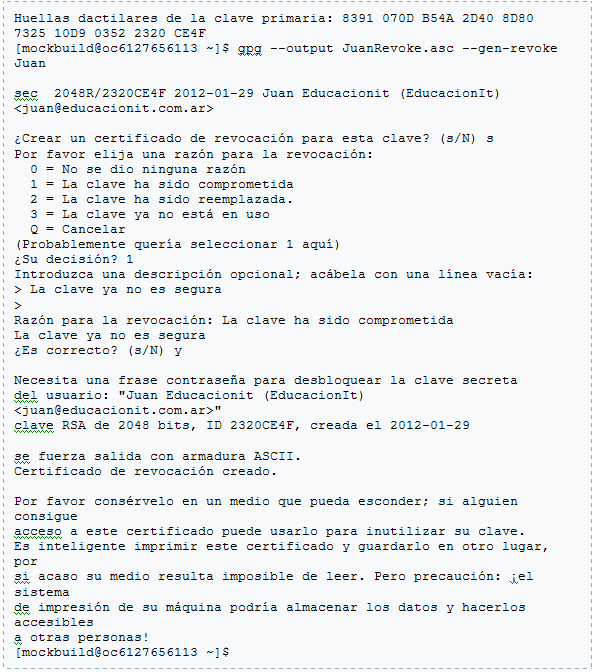
GnuPG viene con una especie de shell que nos da multitud de opciones para trabajar con la clave, Nos permite firmar la clave, cambiar la contraseña, cambiar la fecha de expiración de la llave …

Para acceder a esta shell hay que poner el parámetro --edit-key ClaveID.

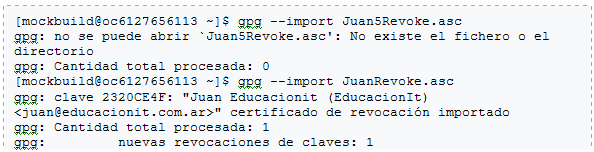
Clave de revocación

La clave de revocación es una clave que lo que hace es que cuando la importemos a nuestro anillo de claves invalide esa clave. Para generarla se hace con la opción --gen-revoke. Esta clave se puede crear nada más generar las llaves o bien cuando se haya comprometido. Hay gente que lo crea nada más crear las claves porque si por ejemplo ha olvidado  la contraseña no podrá generar la clave de revocación ya que al final del proceso de generación se pide la contraseña. Esta clave ha de guardarse en un lugar seguro ya que si alguien la obtuviese podría revocar nuestras claves y dejarnos las claves inutilizadas.

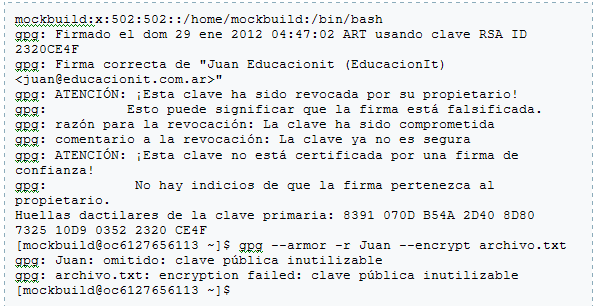




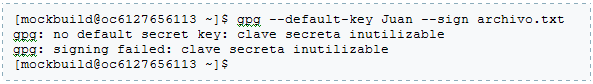
Si queremos revocar la clave hay que importar el fichero que tiene la clave de revocación y ya está. Una vez revocada la clave ya no podemos cifrar mensajes aunque sí se pueden desencriptar, aunque al desencriptar se avisa de que la clave ha sido revocada..







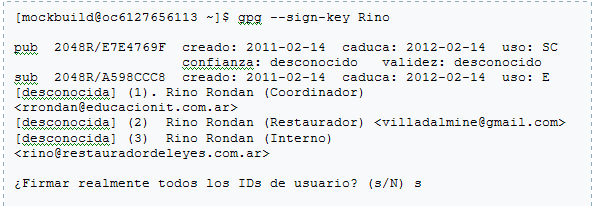
Al invalidar la clave tampoco se pueden firmar mensajes.

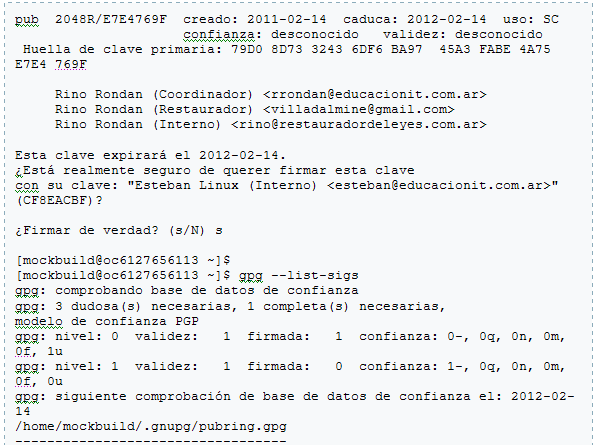


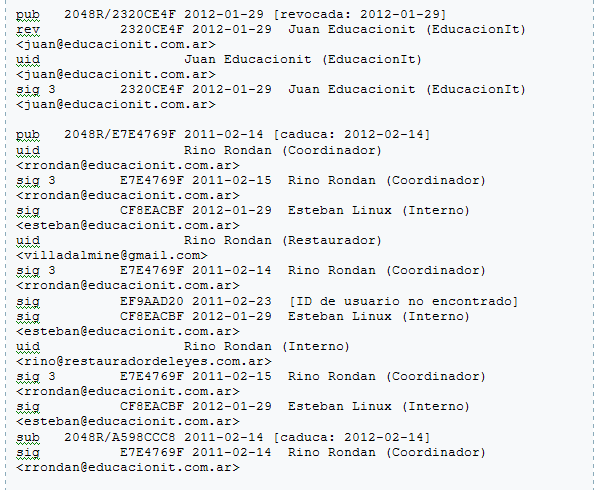
Anillo de confianza

Crear un anillo de confianza consiste en tener claves de gente firmada por otra gente que la han firmado y que con su firma aseguran que esa clave es realmente de quien dice ser y no ha sido alterada.

Si por ejemplo tenemos las persona A y B. Las personas A y B son amigas y se intercambian entre ellas las claves públicas, verifican sus fingerprints para ver que las claves son las correctas y quedan para ver las claves que se han pasado son correctas. Entonces una vez verificado que todo es correcto cada uno firma la clave de su amigo. Ahora si por ejemplo yo obtengo la clave de B y veo que está firmada por A (que es una persona que conozco y en la que confió) entonces me fio de que esa clave es la clave correcta de B y la puedo usar. Si por un casual quedaría firmar la clave de B con mi firma para avalar que su clave es buena sería bueno que me pusiese en contacto con él y verificáramos la clave.







Programas que soportan GnuPG

Para utilizar GnuPG de forma gráfica y no tener que estar escribiendo los comandos podemos encontrar para KDE el programa KGPG y para Gnome el programa SeaHorse.

Hay algunos clientes de correo que también soportan GnuPG como Evolution o Mozilla Thunderbird al que hay que ponerle el plugin Enigmail para que tenga soporte de GnuPG.

Bibliografía

Libros:

[LPI Linux Certification in a Nutshell, Third Edition, June 2010](https://www.google.com/url?q=http://wiki.itrestauracion.com.ar/doku.php/http_:oreilly.com_catalog_9780596804879&sa=D&ust=1464643730532000&usg=AFQjCNFV8qQ-xQtdiWoPGql4SSGqsODPDw)

[LPIC-1: Linux Professional Institute Certification Study Guide: (Exams 101 and 102), 2nd Edition, February 2009](https://www.google.com/url?q=http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470404833.html&sa=D&ust=1464643730533000&usg=AFQjCNElOtEZMPSOxUgqVPji_8W8EO73-Q)

Páginas:

[1] [GnuPG](https://www.google.com/url?q=http://www.lostscene.com/manuales/gnupg.php&sa=D&ust=1464643730534000&usg=AFQjCNEzR5hoqcl8uZh1E25ojFp7CaEaWg" \t "_blank)

[2] [Wikipedia GNUPG](https://www.google.com/url?q=http://es.wikipedia.org/wiki/GNU_Privacy_Guard&sa=D&ust=1464643730534000&usg=AFQjCNGo4y-PwT2gHPBpjIXASXznqU3klA" \t "_blank)

[3] [GnuPG](https://www.google.com/url?q=http://www.dewinter.com/gnupg_howto/spanish/gpgminicomo-1.html%23ss1.1&sa=D&ust=1464643730535000&usg=AFQjCNEECP0cSgfn_iwFM5UY03o2qKrytA" \t "_blank)

[4] [Ejemplos at](https://www.google.com/url?q=http://www.thegeekstuff.com/2010/06/at-atq-atrm-batch-command-examples/&sa=D&ust=1464643730536000&usg=AFQjCNEz7FiMD4uLAe34GkaZkqukItj9Tg)

[5] [Checksum](https://www.google.com/url?q=http://www.thegeekstuff.com/2012/07/cksum-command-examples/?utm_source%3Dfeedburner%26utm_medium%3Demail%26utm_campaign%3DFeed%253A%2BTheGeekStuff%2B%2528The%2BGeek%2BStuff%2529&sa=D&ust=1464643730536000&usg=AFQjCNGXNH_zN1IWxB7d7wWdsNrXjQ9L-w" \t "_blank)

[6] [Teoria Encriptacion](https://www.google.com/url?q=http://www.thegeekstuff.com/2012/07/cryptography-basics/?utm_source%3Dfeedburner%26utm_medium%3Demail%26utm_campaign%3DFeed%253A%2BTheGeekStuff%2B%2528The%2BGeek%2BStuff%2529&sa=D&ust=1464643730537000&usg=AFQjCNFr2ko20O2MJWz-srfNfCPmP5VLdw" \t "_blank)

Laboratorio Compilación   

Kernel

Kernel Monolítico

Un núcleo monolítico es un tipo de núcleo o kernel de un sistema operativo. Como ejemplo de sistema operativo de núcleo monolítico están UNIX, Linux y FreeBSD.[3]

Estos sistemas tienen un núcleo grande y complejo, que engloba todos los servicios del sistema. Está programado de forma no modular, y tiene un rendimiento mayor que un micronúcleo. Sin embargo, cualquier cambio a realizar en cualquier servicio requiere la recopilación del núcleo y el reinicio del sistema para aplicar los nuevos cambios.

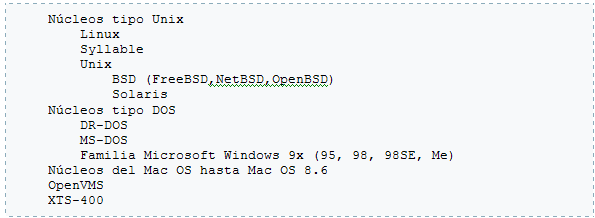
Hay diversas ramificaciones de este diseño, que se han ido amoldando a nuevas necesidades. Podemos citar el sistema de módulos ejecutables en tiempo de ejecución, que le brinda al modelo de núcleo monolítico algunas de las ventajas de un micronúcleo. Dichos módulos pueden ser compilados, modificados, cargados y descargados en tiempo de ejecución, de manera similar a los servicios de un micronúcleo, pero con la diferencia de que se ejecutan en el espacio de memoria del núcleo mismo (anillo 0). De esta forma, un bloqueo del módulo, es probable que bloquee todo el núcleo. Además, el módulo pasa a formar un todo con el núcleo, usando la API del mismo, y no se emplea un sistema de mensajes como en los micronúcleos. Este es el esquema usado por, entre otros, Linux, FreeBSD y varios derivados de UNIX. Cabe resaltar que el paso constante de mensajes entre los servicios del micronúcleo, es en parte responsable del pobre rendimiento de los micronúcleos.

Un sistema operativo con núcleo monolítico concentra todas las funcionalidades posibles (planificación, sistema de archivos, redes, controladores de dispositivos, gestión de memoria, etc) dentro de un gran programa. El mismo puede tener un tamaño considerable, y deberá ser recopilado por completo al añadir una nueva funcionalidad. Todos los componentes funcionales del núcleo tienen acceso a todas sus estructuras de datos internas y a sus rutinas. Un error en una rutina puede propagarse a todo el núcleo.

La alternativa es tener una estructura de micronúcleo, donde las partes funcionales están divididas en unidades separadas con mecanismos de comunicación estrictos entre ellos.

Sistemas operativos con núcleos monolíticos

Entre los sistemas operativos que cuentan con núcleos monolíticos se encuentran:



Arquitectura

Diagrama del núcleo

Actualmente Linux es un núcleo monolítico híbrido. Los controladores de dispositivos y las extensiones del núcleo normalmente se ejecutan en un espacio privilegiado conocido como anillo 0 (ring 0), con acceso irrestricto al hardware, aunque algunos se ejecutan en espacio de usuario. A diferencia de los núcleos monolíticos tradicionales, los controladores de dispositivos y las extensiones al núcleo se pueden cargar y descargar fácilmente como módulos, mientras el sistema continúa funcionando sin interrupciones. También, a diferencia de los núcleos monolíticos tradicionales, los controladores pueden ser prevolcados (detenidos momentáneamente por actividades más importantes) bajo ciertas condiciones. Esta habilidad fue agregada para gestionar correctamente interrupciones de hardware, y para mejorar el soporte de multiprocesamiento simétrico.

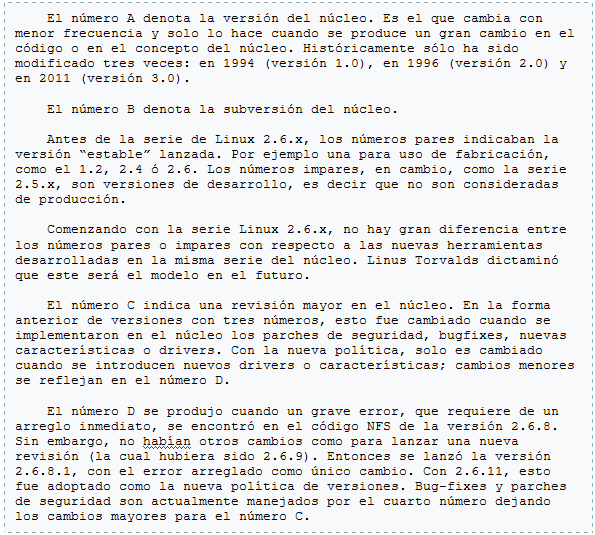
El hecho de que Linux no fuera desarrollado siguiendo el diseño de un micronúcleo (diseño que, en aquella época, era considerado el más apropiado para un núcleo por muchos teóricos informáticos) fue asunto de una famosa y acalorada discusión entre Linus Torvalds y Andrew S. Tanenbaum.[2]

Versiones

Más allá de haber desarrollado su propio código y de integrar los cambios realizados por otros programas, Linus Torvalds continúa lanzando nuevas versiones del núcleo Linux. Estos son llamados núcleos “vanilla”, lo que significa que no han sido modificados por nadie. Muchos desarrolladores de distribuciones Linux modifican dicho núcleo en sus productos, principalmente para agregarle soporte a dispositivos o herramientas que no fueron oficialmente lanzadas como estables, mientras que algunas distribuciones, como Slackware, mantienen el núcleo vanilla.

Numeración

La versión del núcleo Linux actualmente consta de cuatro números. Por ejemplo, asumamos que el número de la versión está compuesta de esta forma: A.B.C[.D] (ej.: 2.2.1, 2.4.13 ó 2.6.12.3).



También, algunas veces luego de las versiones puede haber algunas letras como “rc1” o “mm2”. El “rc” se refiere a release candidate e indica un lanzamiento no oficial. Otras letras usualmente (pero no siempre) hacen referencia a las iniciales de la persona. Esto indica una bifurcación en el desarrollo del núcleo realizado por esa persona, por ejemplo ck se refiere a Con Kolivas, ac a Alan Cox, mientras que mm se refiere a Andrew Morton.

El modelo de desarrollo para Linux 2.6 fue un cambio significativo desde el modelo de desarrollo de Linux 2.5. Previamente existía una rama estable (2.4) donde se habían producido cambios menores y seguros, y una rama inestable (2.5) donde estaban permitidos cambios mayores. Esto significó que los usuarios siempre tenían una versión 2.4 a prueba de fallos y con lo último en seguridad y casi libre de errores, aunque tuvieran que esperar por las características de la rama 2.5. La rama 2.5 fue eventualmente declarada estable y renombrada como 2.6. Pero en vez de abrir una rama 2.7 inestable, los desarrolladores de núcleos eligieron continuar agregando los cambios en la rama “estable” 2.6. De esta forma no había que seguir manteniendo una rama vieja pero estable y se podía hacer que las nuevas características estuvieran rápidamente disponibles y se pudieran realizar más test con el último código.

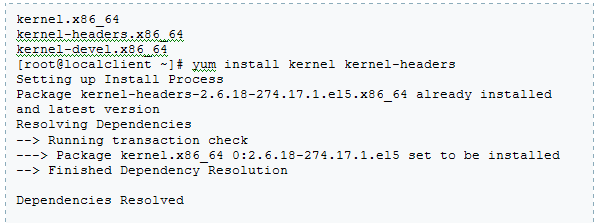
Sin embargo, el modelo de desarrollo del nuevo 2.6 también significó que no había una rama estable para aquellos que esperaban seguridad y bug fixes sin necesitar las últimas características. Los arreglos solo estaban en la última versión, así que si un usuario quería una versión con todos los bug fixed conocidos también tendría las últimas características, las cuales no habían sido bien testeadas. Una solución parcial para esto fue la versión ya mencionada de cuatro números (y en 2.6.x.y), la cual significaba lanzamientos puntuales creados por el equipo estable (Greg Kroah-Hartman, Chris Wright, y quizás otros). El equipo estable solo lanzaba actualizaciones para el núcleo más reciente, sin embargo esto no solucionó el problema del faltante de una serie estable de núcleo. Distribuidores de Linux, como Red Hat y Debian, mantienen los núcleos que salen con sus lanzamientos, de forma que una solución para algunas personas es seguir el núcleo de una distribución.

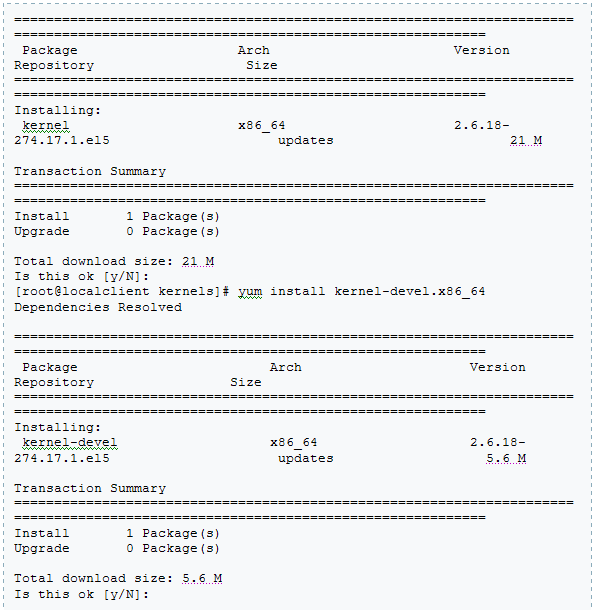
Como respuesta a la falta de un núcleo estable y de gente que coordinara la colección de corrección de errores, en diciembre de 2005 Adrian Bunk anunció que continuaría lanzando núcleos 2.6.16 aun cuando el equipo estable lanzara 2.6.17. Además pensó en incluir actualizaciones de controladores, haciendo que el mantenimiento de la serie 2.6.16 sea muy parecido a las viejas reglas de mantenimiento para las serie estables como 2.4. El núcleo 2.6.16 será reemplazado próximamente por el 2.6.27 como núcleo estable en mantenimiento durante varios años.

Compilando el Kernel

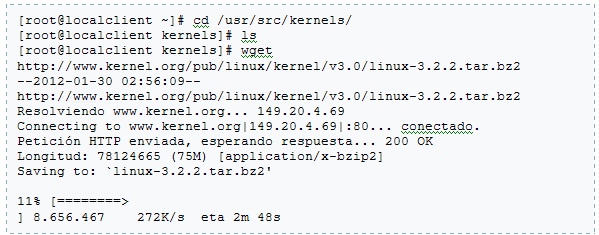
Primero antes que nada vamos a listar paquetes que necesitaremos para poder compilar nuestro Kernel.

Si quisiéramos el kernel que nos provee nuestra distribución

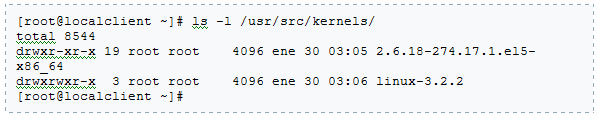




Si quisiéramos el kernel oficial de Kernel.org.



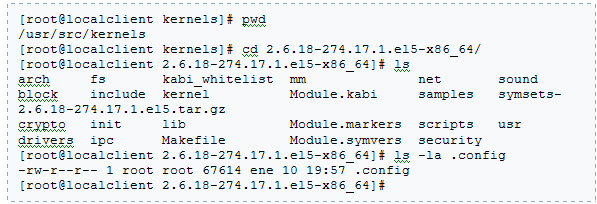
Ahora vamos a listar el directorio donde se encuentra el kernel .



Tenemos el kernel del paquete de fedora y el kernel oficinal de kernel.org

Menú Compilación

Para poder empezar a compilar nuestro kernel debemos ir al directorio donde se encuentra este.



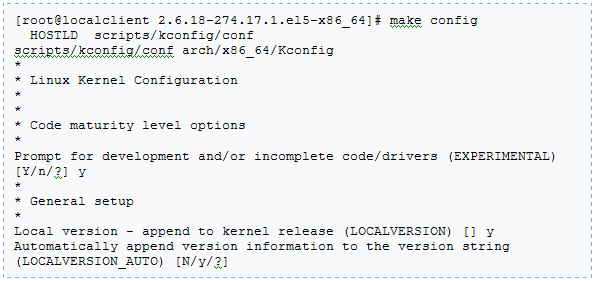
El archivo .config es el que contiene la data de todo lo que va a contener nuestro kernel, tanto lo que está totalmente compilado o lo que va a estar como módulo.

Luego los comandos esenciales para poder empezar a tocar de una manera amigable las muchas opciones que hay son los siguientes:

Tener en cuenta que el archivo .config me sirve para luego copiarlo a otra pc que quiera compilar con las mismas opciones, o sino una vez compilado el kernel podemos generar un paquete rpm e instalar el kernel en otro equipo sin tener que compilar devuelta. También si bajamos algún kernel y queremos usar las opciones que teníamos antes podemos copiarlo en la carpeta donde bajamos el kernel.

make config

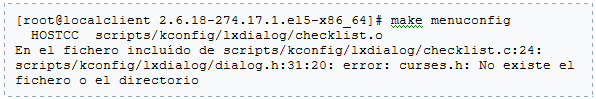
Probemos algunos:



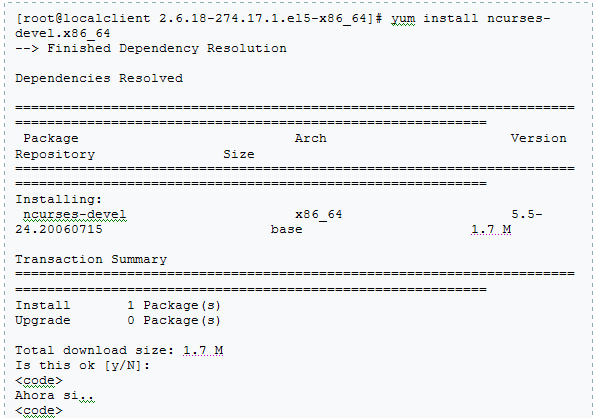
Así nos va a preguntar por cada opción existente.

make menuconfig

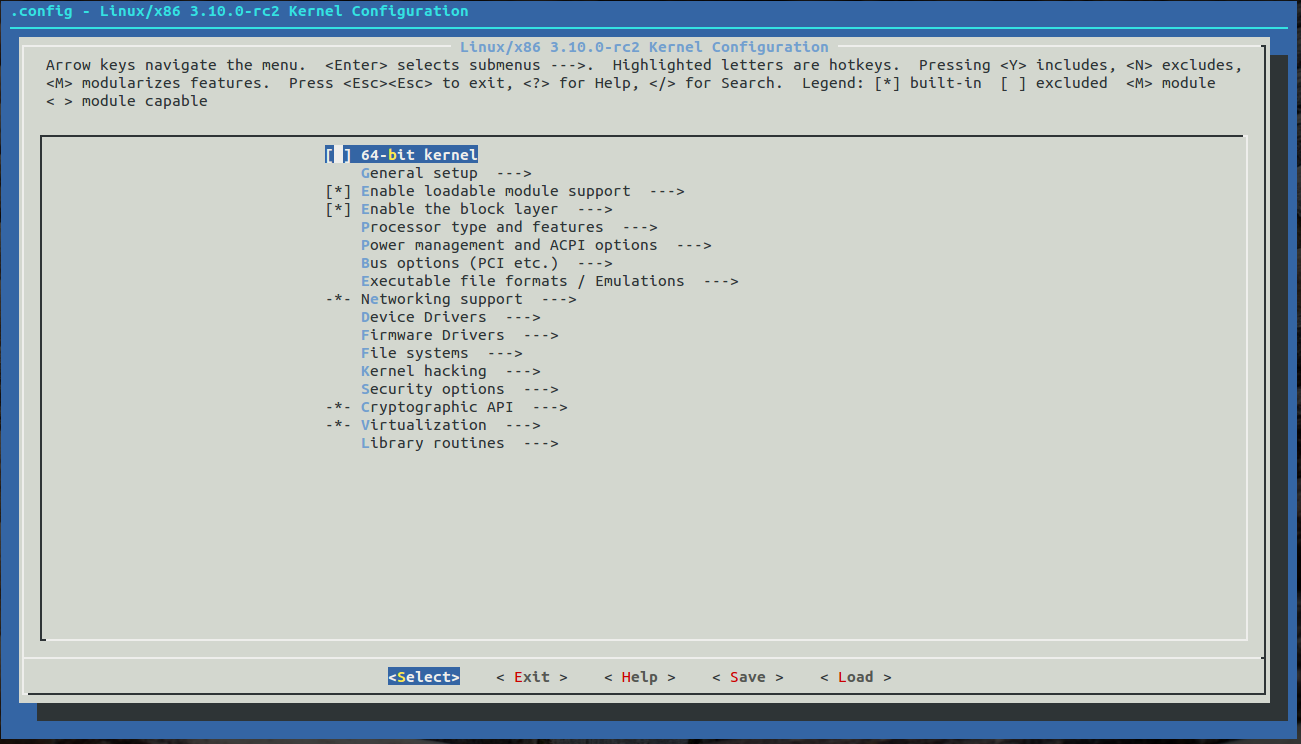
Otra forma más amigable con menuconfig:



 Si tenemos ese error es porque no tenemos el paquete ncurses que es el que se encarga de armar en nuestro equipo el menú.







make gconfig

Ahora la siguiente imagen muestra la utilización de gconfig

make bzImage

Una vez terminado el proceso anterior, previo a este podríamos haber ejecutado un make clean para que elimine todo lo que tenga de una compilación previa.

Este paso nos va a crear el archivo del kernel.

bzImage es una versión comprimida del kernel, similar a vmlinuz. Este fue creado para evitar ciertas limitaciones técnicas que tenia la creación de un kernel zimage.

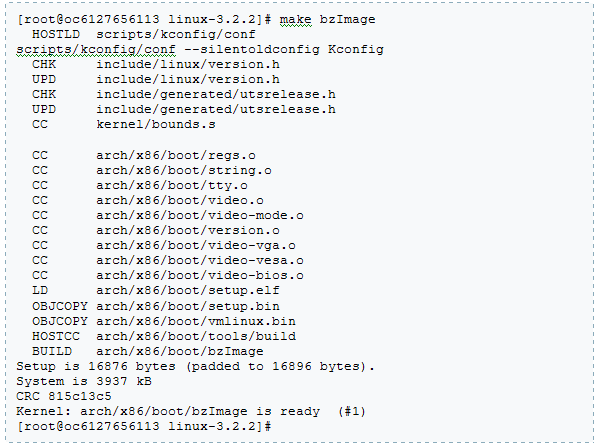
zImage ya no se usa más es similar al formato vmlinuz. Anteriormente cuando se compilaba se usaba ese nombre, como el kernel fue creciendo en tamaño este último para los sistemas modernos quedo inadecuado dado que está limitado en tamaño en 512KB.

vmlinux es una versión descomprimida del kernel de linux, generalmente es usada como paso intermedio y no es copiada en /boot. Este kernel no esta preparado para arrancar dado que algunas opciones no están habilitadas.

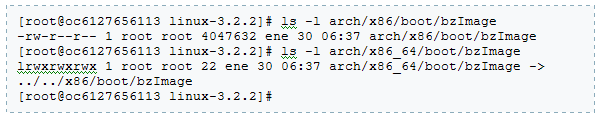
vmlinuz es una variante de vmlinux en donde este se encuentra comprimido y con diversas herramientas que habilitan su uso para arrancar el sistema agregando otras características más. Las distribuciones de linux proveen generalmente un kernel vmlinuz como nombre del kernel precompilado de sus paquetes.

make bzImage

El siguiente cuadro muestra la forma de que se creara el kernel. En las versiones 2.6 en adelante se puede ejecutar make sin parámetros, creará el bzImage de manera predeterminada.

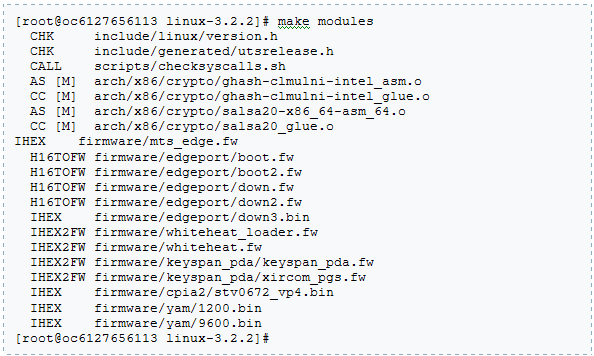


La imagen del kernel fue generada.



make modules

Ahora el próximo paso es crear los módulos del kernel y esto va a tardar más que lo anterior. En la versión 2.6 en adelante se puede ejecutar make modules\_install directamente.



make modules\_install

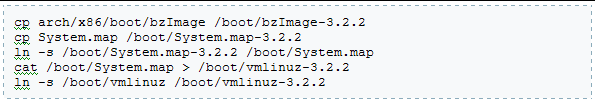
Para poder instalar los módulos y generar los archivos correspondientes faltaria este paso.



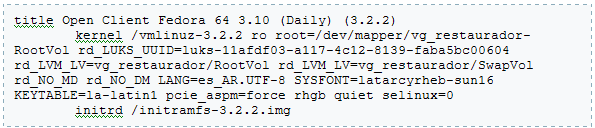
Preparar Arranque   

Manualmente

Para lograrlo de forma manual deberíamos hacer lo siguiente.



Luego tendríamos que mirar alguna entrada vieja del grub y editarla para que siga el mismo patrón que el kernel nuevo.

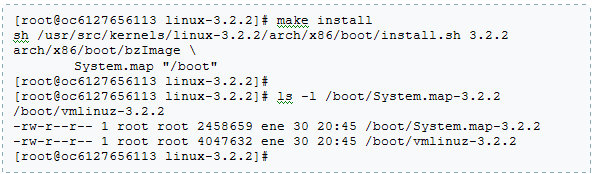


Luego generamos la imagen de RAM disk.

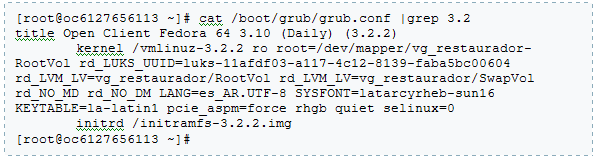
https://alumni.educacionit.com/content/289/1007/images/image51.png

make install

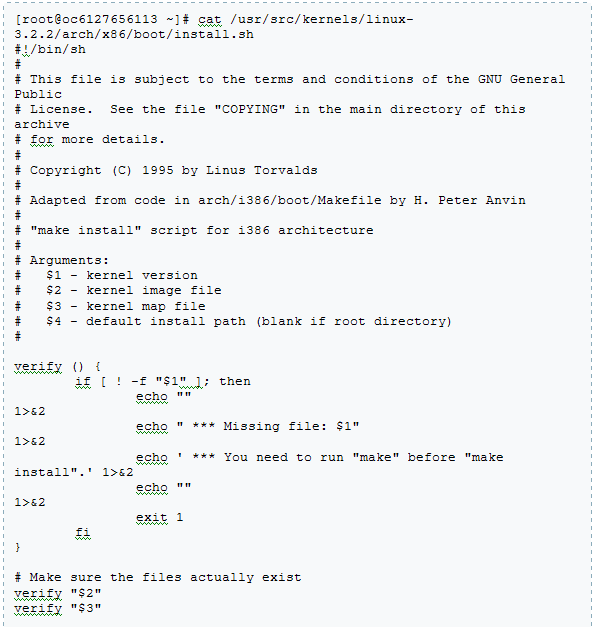
De esta forma nos instala el kernel y editar el grub dejando todo preparado.

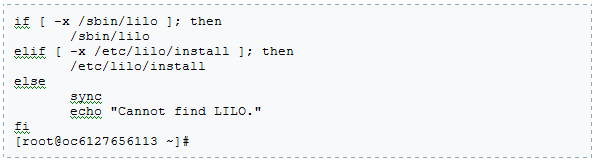


El grub ya posee la entrada nueva.



Podríamos ver que el script hace lo siguiente.





Herramientas de RAM disk

Estas herramientas lo que me van a permitir es tener un set módulos y herramientas críticas esenciales para el arranque de la máquina que van a ser utilizadas en el inicio de arranque del kernel previo al montaje del kernel en el sistema.

El kernel es cargado en memoria como si estuviera en un disco, cargando los módulos y herramientas desde la RAM disk para luego poder montar el sistema raíz.

El comando es mkinitrd :

Podria usarse de una manera basica: mkinitrd /boot/initrd-3.2.2.img 3.2.2

Instalando desde las fuentes

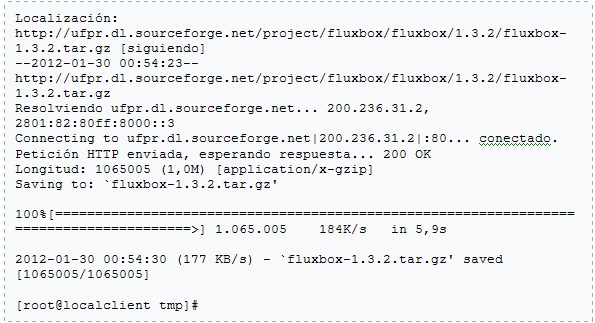
En este laboratorio vamos a ver como instalar fluxbox desde la fuente.

Instalando Fluxbox

Lo primero que vamos hacer es bajar fluxbox desde la página oficinal.

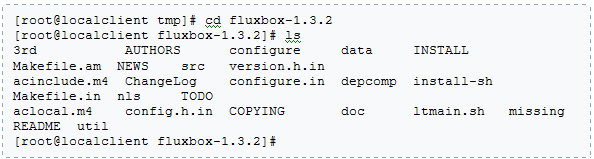
–> [http://sourceforge.net/projects/fluxbox/files/fluxbox/1.3.2/fluxbox-1.3.2.tar.gz](https://www.google.com/url?q=http://sourceforge.net/projects/fluxbox/files/fluxbox/1.3.2/fluxbox-1.3.2.tar.gz&sa=D&ust=1464643730562000&usg=AFQjCNEtXMngG_MAwYR8AbL91wRvsHJqZQ" \t "_blank)



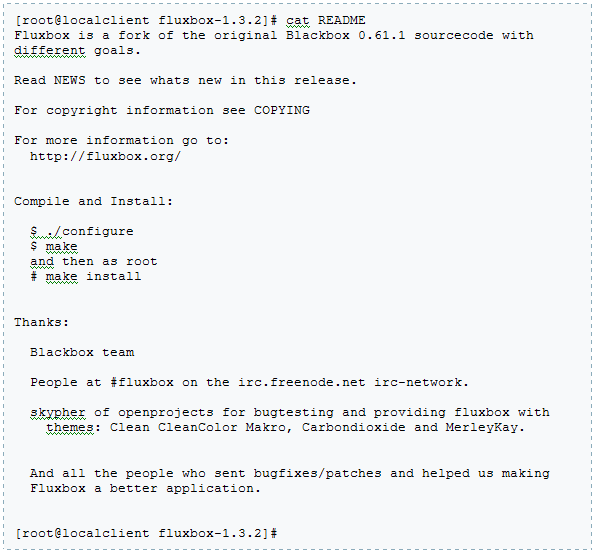


Ahi lo bajamos.

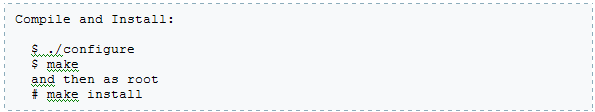
Una vez que lo descomprimimos.



Siempre es una buena idea tratar de mirar toda la documentación posible para ver como se instala desde la fuente. Generalmente se encontrará un archivo README o un archivo INSTALL con las instrucciones.

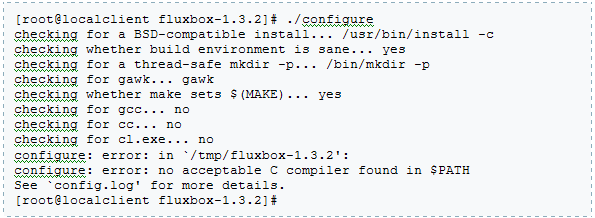


Vamos a seguir los pasos que nos recomienda.



./configure

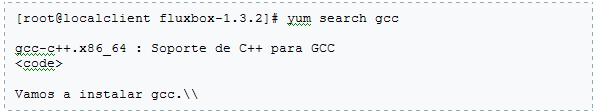
El script configure es el encargado de verificar si tenemos todo lo necesario para compilar. También al configure se le pasan parámetros como --prefix para indicarle donde se instalará el programa.



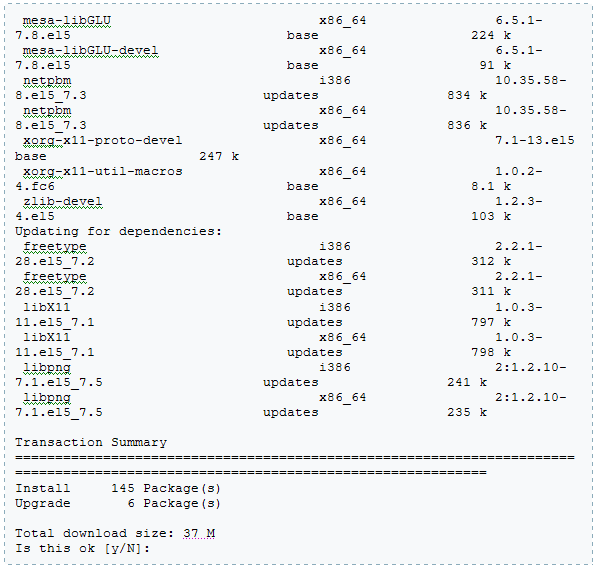
En este caso muestra es que no tenemos un compilador que compile el código fuente que estamos intentando instalar.

# yum groupinstall “Bibliotecas de desarrollo”

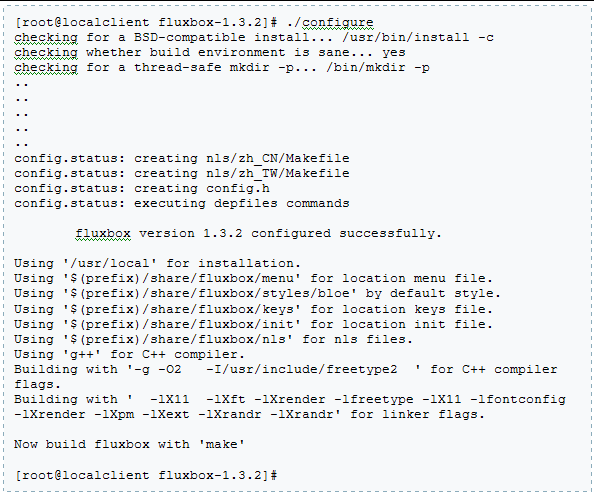
Si no, instalar herramienta por herramienta



# yum install gcc-c++.x86\_64  
# yum grouplist “Desarrollo de software para X”



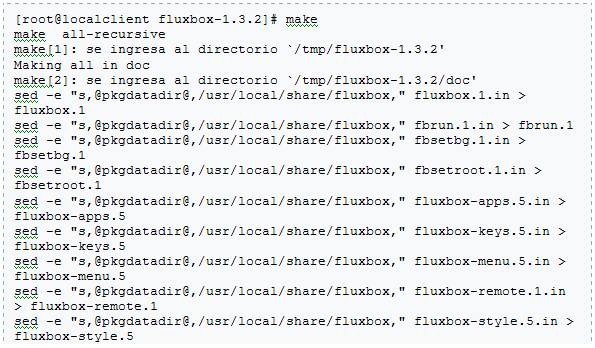
Podríamos haber instalado el paquete necesario pero prefiero instalar esto en este caso dado que es para fines educativos.

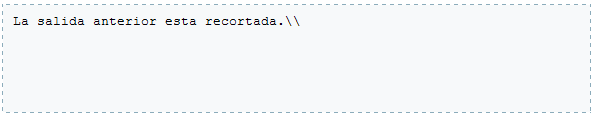


Como vemos ahi termino y nos dice cual es el siguiente paso.

Make

El comando make leerá un archivo llamado Makefile, el cual contendrá todo el listado de archivos a compilar generado por el configure.

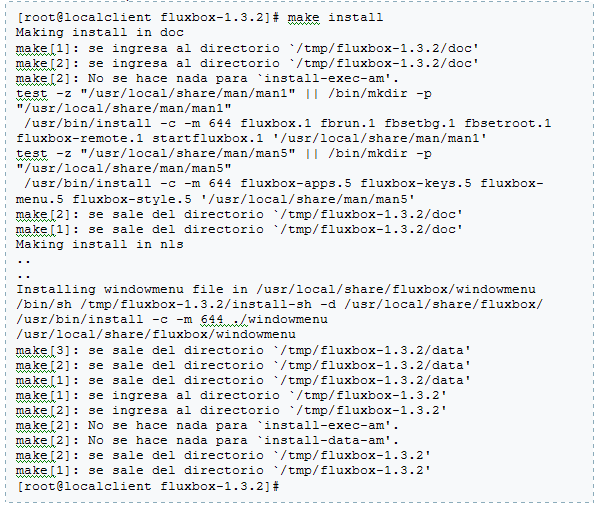




Los pasos desarrollados hasta ahora lo podríamos haber ejecutado sin necesidad de ser root.

Make install

Este último paso moverá todo lo compilado al directorio predeterminado o al que se haya definido con --prefix dependiendo de esto lo podremos ejecutar como usuario común o como root.

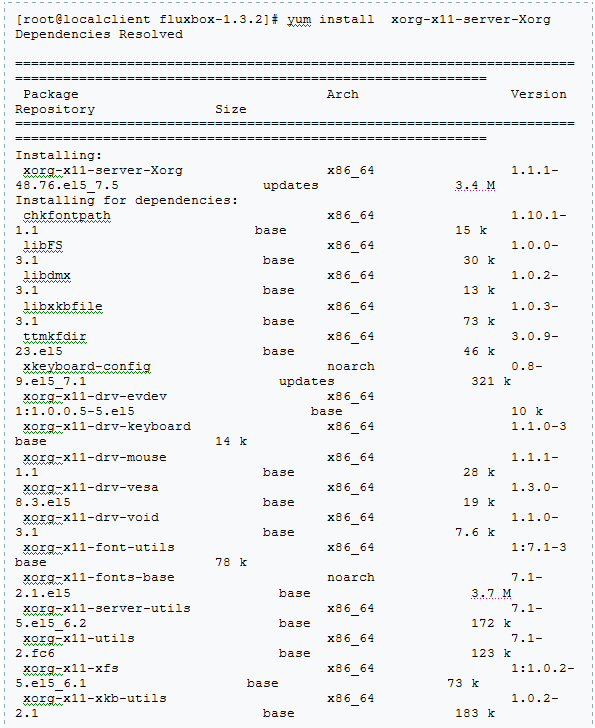


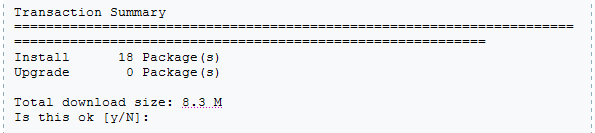
La salida fue recortada.

Hasta acá compilo todo correctamente pero nos falta instalar el Servidor X que fluxbox va a necesitar para poder iniciar.

Instalando X Server

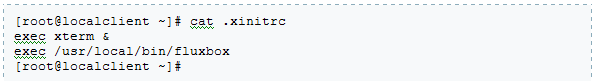
Para instalar el X server vamos a instalar el paquete del server X11.



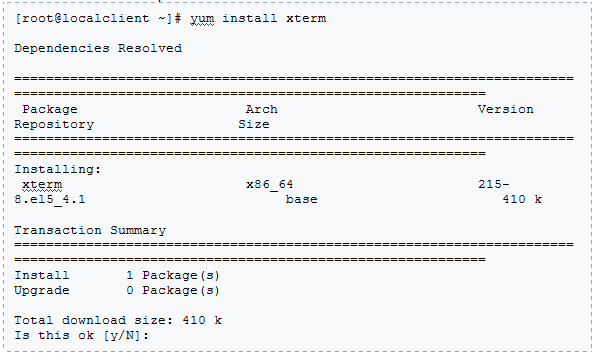


# yum install xorg-x11-xinit.x86\_64

Generamos este archivo:



Para eso tendremos que instalar un xterm.



Ahora si en la máquina tipeamos xinit iniciara fluxbox y abrirá una terminal xterm, dado que el xinit lee el archivo xinitrc.