Guía para los Committers

Resumen

Este documento proporciona información para la comunidad de committers de FreeBSD. Todos los committers nuevos deben leer este documento antes de empezar, y se recomienda encarecidamente a los committers actuales que lo revisen de vez en cuando.

Casi todos los desarrolladores de FreeBSD tienen derecho de acceso a uno o más repositorios. Sin embargo, algunos desarrolladores no lo tienen, y cierta información aquí expuesta también les afecta. (Por ejemplo, algunas personas sólo tienen derecho a trabajar con la base de datos de reporte de problemas.) Por favor lea Problemas Específicos para Desarrolladores que No Son Committers para más información.

Este documento también puede ser de interés para los miembros de la comunidad de FreeBSD que quieran saber más sobre el funcionamiento del proyecto.

Tabla de contenidos

1. Detalles administrativos . . . 2
2. Claves OpenPGP de FreeBSD . . . 3
3. Kerberos y contraseña web LDAP para el clúster de FreeBSD . . . 4
4. Tipos de Commit Bits . . . 5
5. Introducción a Git . . . 7
6. Histórico del Control de Versiones . . . 40
7. Configuración, Convenciones y Tradiciones . . . 41
8. Revisión previa al commit . . . 46
9. Mensajes de Commit . . . 47
10. Licencia preferida para los nuevos archivos . . . 55
11. Seguimiento de las licencias concedidas al proyecto FreeBSD . . . 56
12. Etiquetas SPDX en el árbol . . . 57
13. Relaciones con los desarrolladores . . . 57
14. Si tienes dudas 58
15. Bugzilla . . . 59
16. Phabricator . . . 59
17. Quien es Quien . . . 60
18. Guía de inicio rápido de SSH . . . 61
19. Disponibilidad de Coverity® para los Committers de FreeBSD . . . 62
20. La gran lista de reglas de los Committers de FreeBSD . . . 62
21. Soporte para múltiples arquitecturas . . . 71
22. Preguntas frecuentes sobre ports específicos . . . 75

23. Problemas Específicos para Desarrolladores que No Son Committers . . . 82
24. Información sobre Google Analytics . . . 82
25. Preguntas misceláneas . . . 83
26. Beneficios y Ventajas para los committers de FreeBSD . . . 83

1. Detalles administrativos

Métodos de inicio de sesión	ssh(1), sólo protocolo 2
Host Shell Principal	`freefall.FreeBSD.org`
Máquinas de Referencia	`ref*.FreeBSD.org`, `universe*.freeBSD.org` (consulta también Máquinas del Proyecto FreeBSD)
SMTP Host	`smtp.FreeBSD.org:587` (consulta también Configuración de acceso SMTP).
`src/` Repositorio Git	`ssh://git@gitrepo.FreeBSD.org/src.git`
`doc/` Repositorio Git	`ssh://git@gitrepo.FreeBSD.org/doc.git`
`ports/` Repositorio Git	`ssh://git@gitrepo.FreeBSD.org/ports.git`
Listas de Correo Internas	developers (técnicamente llamada all-developers) doc-developers, doc-committers, ports-developers, ports-committers, src-developers, src-committers. (Cada repositorio del proyecto tiene su propia lista de correo terminada en -developers y -committers. Se pueden encontrar archivos para estas listas en los ficheros /local/mail/repository-name-developers-archive y /local/mail/repository-name-committers-archive en `freefall.FreeBSD.org`.)
Informes mensuales del Core Team	/home/core/public/reports en el clúster `FreeBSD.org`.
Informes mensuales del Ports Management Team	/home/portmgr/public/monthly-reports en el clúster `FreeBSD.org`.
Notablemente Ramas de Git de `src/`:	`stable/n` (`n`-STABLE), `main` (-CURRENT)

Se requiere ssh(1) para conectarse a los servidores del proyecto. Para más información, lea Guía de inicio rápido de SSH.

Enlaces de interés:

- Páginas Internas del Proyecto FreeBSD
- Servidores del Proyecto FreeBSD

- Grupos Administrativos del Proyecto FreeBSD

2. Claves OpenPGP de FreeBSD

Claves criptográficas que siguen al estándar OpenPGP (*Pretty Good Privacy*) son utilizadas por el Proyecto FreeBSD para autentificar a los colaboradores. Mensajes que contengan información importante como claves SSH públicas pueden ser firmadas con una clave OpenPGP para demostrar que provienen realmente del colaborador. Véase PGP & GPG: Email for the Practical Paranoid by Michael Lucas y http://en.wikipedia.org/wiki/Pretty_Good_Privacy para más información.

2.1. Creando una clave

Se pueden utilizar claves ya existentes, pero primero deberían ser comprobadas primero con documentation/tools/checkkey.sh. En este caso, comprueba que la clave tiene un identificador de usuario de FreeBSD.

Para aquellos que todavía no tengan una clave OpenPGP, o necesiten una nueva para reunir los requerimientos de seguridad de FreeBSD, se mostrará a continuación como generarla.

1. Instala security/gnupg. Inserta las siguientes líneas en ~/.gnupg/gpg.conf para establecer valores aceptables por defecto:

```
fixed-list-mode
keyid-format 0xlong
personal-digest-preferences SHA512 SHA384 SHA256 SHA224
default-preference-list SHA512 SHA384 SHA256 SHA224 AES256 AES192 AES CAST5
BZIP2 ZLIB ZIP Uncompressed
verify-options show-uid-validity
list-options show-uid-validity
sig-notation issuer-fpr@notations.openpgp.fifthhorseman.net=%g
cert-digest-algo SHA512
```

2. Genera una clave:

```
% gpg --full-gen-key
gpg (GnuPG) 2.1.8; Copyright (C) 2015 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.

Warning: using insecure memory!
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? 1
```

```
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 2048 ①
Requested keysize is 2048 bits
Please specify how long the key should be valid.
     0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 3y ②
Key expires at Wed Nov  4 17:20:20 2015 MST
Is this correct? (y/N) y
GnuPG needs to construct a user ID to identify your key.

Real name: Chucky Daemon ③
Email address: notreal@example.com
Comment:
You selected this USER-ID:
"Chucky Daemon <notreal@example.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.
```

① Claves de 2048 bits con una expiración de tres años proporcionan una protección adecuada actualmente (202-10).

② Tres años de vida útil para una clave hacen que sea lo suficientemente corta como para hacer que quede obsoleta por el avance de la potencia de los ordenadores, pero lo suficientemente larga como para reducir los problemas de administración de claves.

③ Utiliza tu nombre real aquí, preferiblemente coincidente con el nombre de tu documento de identificación oficial para ayudar a otros a verificar tu identidad. En la sección `Comment` se puede introducir texto que ayude a otros a identificarte.
Después de introducir la dirección de correo electrónico, se solicita una contraseña. Los métodos para crear una contraseña segura son bastante polémicos. En lugar de sugerir una única forma, aquí hay algunos enlaces a sitios que describen varios métodos: https://world.std.com/~reinhold/diceware.html, https://www.iusmentis.com/security/passphrasefaq/, https://xkcd.com/936/, https://en.wikipedia.org/wiki/Passphrase.

Protege la clave privada y la contraseña. Si la clave privada o la contraseña fueran comprometidas o reveladas, notifícalo de forma inmediata a accounts@FreeBSD.org y revoca la clave.

Los pasos para enviar la nueva clave se muestran en Pasos para los Nuevos Committers.

3. Kerberos y contraseña web LDAP para el clúster de FreeBSD

El clúster de FreeBSD requiere una contraseña de Kerberos para acceder a ciertos servicios. La

contraseña de Kerberos también sirve como contraseña web LDAP, ya que LDAP hace de proxy a Kerberos en el clúster. Algunos de los servicios que requieren esto incluyen:

- Bugzilla
- Jenkins

Para crear una nueva cuenta de Kerberos en el clúster de FreeBSD, o para restablecer una contraseña de Kerberos para una cuenta existente utilizando un generador de contraseñas aleatorias:

```
% ssh kpasswd.freebsd.org
```

Esto debe hacerse desde una máquina fuera del clúster de FreeBSD.org.

Una contraseña de Kerberos también puede ser establecida manualmente iniciando sesión en `freefall.FreeBSD.org` y ejecutando:

```
% kpasswd
```

A menos que los servicios autentificados con Kerberos del clúster de FreeBSD.org hayan sido usados previamente, se mostrará `Client unknown`. Este error significa que el método de `ssh kpasswd.freebsd.org` mostrado previamente tendrá que ser usado para inicializar la cuenta de Kerberos.

4. Tipos de Commit Bits

El repositorio de FreeBSD tiene una serie de componentes que, cuando se combinan, integran el código fuente del sistema base del sistema operativo, la documentación, la infraestructura de ports de las aplicaciones de terceros y varias utilidades mantenidas. Cuando se asignan los commit bits, se especifican las áreas del árbol donde se tiene permiso. Generalmente, las áreas asociadas con un commit bit reflejan quién autorizó la asignación del commit bit. Se pueden agregar más áreas de autoridad posteriormente: cuando esto ocurre, el committer debe seguir los procedimientos normales de asignación de commit bit para esa área del árbol, buscar la aprobación de la entidad apropiada y posiblemente obtener un mentor para esa área durante un cierto periodo de tiempo.

Tipos de Commiters	*Responsable*	*Componentes del Árbol*
src	core@	src/
doc	doceng@	doc/, ports/, src/ documentación
ports	portmgr@	ports/

Los commit bits asignados antes de que se desarrollara la idea de áreas de autoridad, pueden ser apropiados para su uso en muchas partes del árbol. Sin embargo, el sentido común dicta que un committer que no haya trabajado previamente en esa área del árbol busque una revisión antes de realizar el commit, busque la aprobación del equipo responsable, y/o trabaje con un mentor. Dado

que las reglas con respecto al mantenimiento del código difieren según el área del árbol, esto beneficiará tanto a quién trabaja en un área del árbol con la que no está muy familiarizado como a quienes trabajan en el árbol.

Se anima a los committers a buscar la revisión de su trabajo como parte del proceso natural del desarrollo, independientemente del área del árbol en la cual se esté realizando el trabajo.

4.1. Política para la actividad de los Committers en otros árboles

- Todos los committers pueden modificar src/share/misc/committers-*.dot, src/usr.bin/calendar/calendars/calendar.freebsd, y ports/astro/xearth/files.
- Los committers de documentación pueden realizar commits en la documentación de src, como las páginas del manual, READMEs, bases de datos de fortune, archivos de calendario y correcciones de comentarios sin la aprobación de un src committer, teniendo en cuenta las normas requeridas para la correcta realización de los commits.
- Cualquier committer puede realizar cambios en cualquier otro árbol con un "Approved by" de un committer que no esté tutelado y dispone del commit bit apropiado. Los committers con mentor pueden proporcionar un comentario "Reviewed by" pero no un "Approved by".
- Los committers pueden adquirir commit bit adicionales mediante el proceso habitual de encontrar a un mentor que lo proponga a core, doceng o portmgr, según sea el caso. Una vez aprobados, se añadirán al "acceso" y se producirá el periodo normal de tutoría, que implicará una continuación de "Approved by" durante algún tiempo.

4.1.1. Aprobación Implícita (Blanket) de Documentación

Algunos arreglos tienen "blanket approval" por parte de Grupo de Ingeniería de Documentación <doceng@FreeBSD.org>, permitiendo a cualquier committer arreglar ese tipo de problemas en cualquier parte del árbol de documentación. Estos arreglos no necesitan aprobación o revisión por parte de un committer de documentación si el autor no tiene un commit bit de documentación.

El blanket approval aplica en estos tipos de arreglos:

- Faltas de ortografía
- Arreglos triviales

  Puntuación, URLs, fechas, rutas y nombres de fichero con información desactualizada o incorrecta, y otros errores comunes que puedan confundir a los lectores.

A lo largo de los años, se han concedido algunas aprobaciones implícitas en el árbol de documentación. Esta lista muestra los casos más comunes:

- Cambios en documentation/content/en/books/porters-handbook/versions/_index.adoc

  __FreeBSD_version Values (Porter’s Handbook), utilizado principalmente por committers de src.

- Cambios en doc/shared/contrib-additional.adoc

Mantenimiento de Colaboradores Adicionales de FreeBSD.

- Todo Pasos para los Nuevos Committers, relacionado con documentación
- Avisos de seguridad; Notas de Errata; Releases;

  Utilizado por Grupo Responsables de Seguridad <security-officer@FreeBSD.org> y Grupo de Ingeniería de Releases <re@FreeBSD.org>.

- Cambios en website/content/en/donations/donors.adoc

  Utilizado por el Responsable de Donaciones <donations@FreeBSD.org>.

Antes de un commit, es necesario comprobar la compilación; consulta las secciones de 'Overview' y 'The FreeBSD Documentation Build Process' de Introducción al Proyecto de Documentación de FreeBSD para Nuevos Voluntarios para más detalles.

5. Introducción a Git

5.1. Git básico

Cuando uno busca "Introducción a Git" aparecen unos cuantos buenos las introducciones de Daniel Miessler A git primer y de Willie Willus Git - Quick Primer son ambas buenas. El libro de Git también es completo, pero mucho más largo https://git-scm.com/book/en/v2. También hay un sitio web https://dangitgit.com/ para errores comunes y problemas de Git, en caso de que necesites ayuda para arreglar algo. Por último una introducción dirigida a científicos computacionales ha demostrado ser útil para algunos a la hora de explicar cómo Git ve el mundo.

Este documento asumirá que lo has leído y tratará de no insistir en lo básico (aunque lo cubrirá brevemente).

5.2. Mini Introducción a Git

Esta introducción tiene un ámbito menos ambicioso que la antigua Introducción a Subversion, pero debería cubrir lo básico.

5.2.1. Ámbito

Si quieres descargar FreeBSD, compilarlo desde las fuentes, y en general mantenerte actualizado de ese modo, esta introducción es para ti. Cubre cómo obtener las fuentes, actualizarlas, hacer bisección y trata brevemente cómo lidiar con unos pocos cambios locales. Cubre lo básico y trata de dar buenos consejos para un tratamiento más en profundidad para cuando el lector encuentre lo básico insuficiente. Otras secciones de esta guía cubren temas más avanzados relacionados con cómo contribuir al proyecto.

El objetivo de esta sección es resaltar aquellas partes de Git que se necesitan para seguir la pista a las fuentes. Asumen un conocimiento básico de Git. Hay muchas introducciones de Git en la web, pero el Git Book proporciona una de las mejores.

5.2.2. Primeros Pasos Para Desarrolladores

Esta sección describe el acceso de lectura-escritura para que los committers hagan push de los commits de los desarrolladores o colaboradores.

Uso diario

> In the examples below, replace `${repo}` with the name of the desired FreeBSD repository: `doc`, `ports`, or `src`.

- Clona el repositorio:

```
% git clone -o freebsd --config remote.freebsd.fetch='+refs/notes/*:refs/notes/*'
https://git.freebsd.org/${repo}.git
```

  Después deberías tener tu remote apuntando a los mirrors oficiales:

```
% git remote -v
freebsd  https://git.freebsd.org/${repo}.git (fetch)
freebsd  https://git.freebsd.org/${repo}.git (push)
```

- Configura los datos del committer de FreeBSD:

  El commit hook en repo.freebsd.org comprueba que el campo "Commit" coincide con la información del committer en FreeBSD.org. La forma más fácil de conseguir la configuración sugerida es ejecutar el script `/usr/local/bin/gen-gitconfig.sh` en freefall:

```
% gen-gitconfig.sh
[...]
% git config user.name (your name in gecos)
% git config user.email (your login)@FreeBSD.org
```

- Establece la URL para hacer push:

```
% git remote set-url --push freebsd git@gitrepo.freebsd.org:${repo}.git
```

  Después deberías tener URLs separadas para fetch y push que es la configuración más eficiente:

```
% git remote -v
freebsd  https://git.freebsd.org/${repo}.git (fetch)
freebsd  git@gitrepo.freebsd.org:${repo}.git (push)
```

  De nuevo, date cuenta de que `gitrepo.freebsd.org` ha sido convertido a su forma canónica `repo.freebsd.org`.

- Instala el hook para la plantilla del mensaje de commit:

```
% fetch https://cgit.freebsd.org/src/plain/tools/tools/git/hooks/prepare-commit-msg
-o .git/hooks
% chmod 755 .git/hooks/prepare-commit-msg
```

rama "admin"

Los ficheros `access` y `metors` se almacenan en una rama huérfana, `internal/admin`, en cada repositorio.

El siguiente ejemplo muestra cómo obtener la rama `internal/admin` en una rama local `admin`:

```
% git config --add remote.freebsd.fetch '+refs/internal/*:refs/internal/*'
% git fetch
% git checkout -b admin internal/admin
```

De forma alternativa, puedes añadir un árbol de trabajo (worktree) para la rama `admin`:

```
git worktree add -b admin ../${repo}-admin internal/admin
```

Para visualizar la rama `internal/admin` en la web: https://cgit.freebsd.org/${repo}/log/?h=internal/admin

For pushing, specify the full refspec:

```
git push freebsd HEAD:refs/internal/admin
```

5.2.3. Mantenerse Actualizado Con el Árbol src de FreeBSD

Primer paso: clonar un árbol. Esto descarga el árbol completo. Hay dos formas de hacerlo. La mayoría de la gente quiere hacer un clonado profundo del repositorio. Sin embargo, hay momentos en los que quieres hacer un clonado superficial.

Nombres de las Ramas

FreeBSD-CURRENT utiliza la rama `main`.

`main` es la rama por defecto.

Para FreeBSD-STABLE, los nombres de las ramas incluyen `stable/12` y `stable/13`.

Para FreeBSD-RELEASE, los nombres de las ramas de ingeniería de versiones incluyen `releng/12.4` y `releng/13.2`.

https://www.freebsd.org/releng/ muestra:

- ramas `main` y `stable/⋯` abiertas
- ramas `releng/⋯` , cada una de las cuales es congelada cuando se etiqueta una versión.

Ejemplos:

- etiqueta release/13.1.0 en la rama releng/13.1
- etiqueta release/13.2.0 en la rama releng/13.2.

Repositorios

Por favor consulta Detalles Administrativos para la última información sobre dónde obtener las fuentes de FreeBSD. El $URL que se muestra abajo se puede obtener en esa página.

Nota: El proyecto no utiliza submódulos ya que no encajan en nuestro flujo de trabajo y modelo de desarrollo. Cómo seguimos la pista a los cambios en las aplicaciones de terceros se discute en otro sitio y en general no es de interés para un usuario casual.

Clonado Profundo

Un clonado profundo se trae el árbol entero, así como las ramas y toda la historia. Es lo más fácil de hacer. También te permite usar la característica de los árboles de trabajo para tener todas tus ramas activas en directorios separados pero con una sola copia del repositorio.

```
% git clone -o freebsd $URL -b branch [<directory>]
```

— creará un clonado profundo. `branch` debería ser una de las ramas listadas en la sección anterior. Si no se proporciona `branch` se usará la rama por defecto (`main`). Si no se proporciona `<directory>` se usará como nombre del nuevo directorio el que coincida con el nombre del repositorio (doc, ports o src).

Querrás un clonado profundo si estás interesado en el histórico, planeas hacer cambios locales, o planeas trabajar en más de una rama. Es la forma más fácil también de mantenerse actualizado. Si estás interesado en el histórico pero vas a trabajar sólo con una rama y andas corto de espacio, también puedes usar --single-branch para descargar la rama (aunque algunos commits de merge no referenciarán la rama desde la que se mergearon lo que podría ser importante para algunos usuarios interesados en versiones detalladas del histórico).

Clonado Superficial

Un clonado superficial sólo copia el código más actual, pero nada o poco del histórico. Esto puede ser útil cuando necesitas construir una revisión específica de FreeBSD o cuando simplemente estás empezando y planeas seguir la pista al árbol de forma más completa. También puedes usarlo para limitar el histórico a un número determinado de revisiones. Sin embargo, lee más abajo para una limitación importante a esta aproximación.

```
% git clone -o freebsd -b branch --depth 1 $URL [dir]
```

Esto clona el repositorio, pero sólo la versión más reciente. El resto del histórico no se descarga. Si cambiaras de opinión más tarde, puedes hacer `git fetch --unshallow` para obtener el histórico antiguo.

Cuando haces un clonado superficial, pierdes el contador de commits en la salida de uname. Esto puede hacer más difícil determinar si tu sistema necesita ser actualizado cuando se notifica un aviso de seguridad.

Compilando

Una vez que has descargado, la compilación se hace como se describe en el manual, por ejemplo.:

```
% cd src
% make buildworld
% make buildkernel
% make installkernel
% make installworld
```

de forma que no lo cubriremos en profundidad.

Si quieres construir un kernel personalizado, la sección de configuración del kernel del FreeBSD Handbook recomienda crear un fichero MYKERNEL bajo sys/${ARCH}/conf con tus cambios contra GENERIC. Para que Git ignore MYKERNEL, se puede añadir a .git/info/exclude.

Actualización

Para actualizar ambos tipos de árbol utilizan los mismos comandos. Esto se trae todas las revisiones desde tu última actualización.

```
% git pull --ff-only
```

actualizará el árbol. En Git, un merge tipo 'fast forward' es aquel que sólo necesita establecer el puntero a una rama nueva y no necesita recrear los commits. Haciendo siempre un merge/pull de tipo 'fast forward', te asegurarás de que tienes una copia exacta del árbol de FreeBSD. Esto será importante si quieres mantener parches locales.

Lee más abajo para saber cómo gestionar cambios locales. Lo más sencillo es utilizar `--autostash` con el comando `git pull`, pero hay disponibles opciones más sofisticadas.

5.2.4. Seleccionando una Versión Específica

En Git, `git checkout` se trae tanto ramas como versiones específicas. Las versiones de Git son hashes largos en lugar de números secuenciales.

Cuando te traes una versión específica, simplemente especifica en la línea de comando el hash que quieres (el comando git log te ayudará a decidir cuál es el hash que quieres):

```
% git checkout 08b8197a74
```

y ya te lo has traído. Se te saludará con un mensaje como el siguiente:

```
Note: checking out '08b8197a742a96964d2924391bf9fdfeb788865d'.

You are in a 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 08b8197a742a hook gpiokeys.4 to the build
```

donde la última línea es generada a partir del hash que te has traído y la primera línea del mensaje de commit de esa revisión. El hash se puede abreviar a la longitud única más corta que exista. Git es inconsistente acerca de cuántos dígitos muestra.

5.2.5. Bisección

A veces, algo va mal. La última versión funcionó pero la última a la que te has actualizado no. Un desarrollador podría pedirte que bisecciones el problema para localizar qué commit causó la regresión.

Git hacer fácil biseccionar cambios con un potente comando `git bisect`. Aquí hay una breve introducción a cómo usarlo. Para más información, puedes ver https://www.metaltoad.com/blog/beginners-guide-git-bisect-process-elimination o https://git-scm.com/docs/git-bisect para más detalles. La página de manual de git-bisect es buena describiendo lo que puede salir mal, qué hacer cuando las versiones no compilan, cuándo quieres usar otros términos diferentes de 'bueno' y 'malo', etc, nada de lo cual se cubrirá aquí.

`git bisect start --first-parent` comenzará el proceso de bisección. Después necesitarás decirle un rango para que trabaje. `git bisect good XXXXXX` le dirá la revisión que funciona y `git bisect bad XXXXX` le dirá la revisión mala. La revisión mala casi siempre será HEAD (un tag especial para lo que te has traído). La versión buena será la última que te trajiste. El argumento `--first-parent` es necesario para que llamadas siguientes a `git bisect` no intenten traerse una rama externa que carece de las fuentes completas de FreeBSD.

Si quieres saber la última versión que te trajiste, deberías usar `git reflog`:

```
5ef0bd68b515 (HEAD -> main, freebsd/main, freebsd/HEAD) HEAD@{0}: pull
--ff-only: Fast-forward
a8163e165c5b (upstream/main) HEAD@{1}: checkout: moving from
b6fb97efb682994f59b21fe4efb3fcfc0e5b9eeb to main
```

```
...
```

me muestra moviendo el directorio de trabajo a la rama `main` (a816...) y después actualizando desde el origen (a 5ef0...). En esta caso, malo sería HEAD (o 5rf0bd68) y bueno sería a8163e165. Como puedes ver en la salida, HEAD@{1} también funciona, pero no es a prueba de fallos si has hecho otras cosas en tu árbol después de actualizar, pero antes de que descubrieras que tenías que hacer bisección.

Primero establece la versión 'good', luego la mala (aunque el orden no importa). Cuando establezcas la versión mala, te dará algunas estadísticas sobre el proceso:

```
% git bisect start --first-parent
% git bisect good a8163e165c5b
% git bisect bad HEAD
Bisecting: 1722 revisions left to test after this (roughly 11 steps)
[c427b3158fd8225f6afc09e7e6f62326f9e4de7e] Fixup r361997 by balancing parens.  Duh.
```

Después deberías compilar/instalar esa versión. Si es buena, teclearías `git bisect good` si no `git bisect bad`. Si la versión no compila, teclea `git bisect skip`. Recibirás un mensaje similar al de arriba para cada paso. Una vez que hayas terminado, informa al desarrollador de la versión mala (o arregla el fallo tú mismo y envía un parche). `git bisect reset` terminará el proceso y te devolverá a donde empezaste (normalmente a la punta de `main`). De nuevo, el manual de git-bisect (enlazado arriba) es un buen recurso para cuando las cosas van mal o en casos inusuales.

5.2.6. Firmando los commits, tags, y pushes, con GnuPG

Git sabe cómo firmar commits, tags y pushes. Cuando firmas un commit o tag de Git, puedes probar que el código que enviaste vino de ti y que no fue alterado mientras lo transferías. También puedes probar que tú enviaste el código y no otra persona.

Se puede encontrar documentación más en profundidad sobre cómo firmar commits y tags en el capítulo Git Tools - Signing Your Work del libro de Git.

El motivo tras la firma de pushes se puede encontrar en el commit que introdujo esta característica.

La mejor forma es simplemente decirle a Git que siempre quieres firmar commits, tags y pushes. Puedes hacerlo estableciendo unas pocas variables de configuración:

```
% git config --add user.signingKey LONG-KEY-ID
% git config --add commit.gpgSign true
% git config --add tag.gpgSign true
% git config --add push.gpgSign if-asked
```

Para evitar posibles colisiones, asegúrate de darle a Git una id de clave que sea largo. Puedes obtenerlo con: `gpg --list-secret-keys --keyid-format LONG`.

Para utilizar subclaves específicas y no hacer que GnuPG resuelva la subclave a una clave primaria, añade `!` a la clave. Por ejemplo, para encriptar la subclave `DEADBEEF`, usa `DEADBEEF!`.

Verificando firmas

Las firmas de los commits se pueden verificar ejecutando `git verify-commit <commit hash>`, o `git log --show-signature`.

Las firmas de los tags se pueden verificar con `git verity-tag <tag name>`, o `git tag -v <tag name>`.

5.2.7. Consideraciones para Ports

El árbol de ports funciona de la misma forma. Los nombres de las ramas son diferentes y los repositorios están en diferentes lugares.

La interfaz web cgit del repositorio para ser usada desde navegadores web está en https://cgit.FreeBSD.org/ports/. El repositorio Git de producción está en https://git.FreeBSD.org/ports.git y en ssh://anongit@git.FreeBSD.org/ports.git (o anongit@git.FreeBSD.org:ports.git).

También hay un mirror en GitHub, lee Mirrors externos para un resumen. La rama más actual es 'main'. Las ramas *trimestrales* se llaman `yyyyQn` para el año 'yyyy' y el trimestre 'n'.

Formatos de mensaje de commits

El repositorio de ports tiene disponible en .hooks/prepare-commit-message un hook para ayudarte a escribir tus mensajes de commit. Se puede activar ejecutando `git config --add core.hooksPath .hooks`.

La razón principal es que un mensaje de commit se debería formatear de la siguiente forma:

```
category/port: Summary.

Descripción de por qué se han hecho los cambios.

PR:     12345
```

La primera línea es el título del commit, contiene por qué el port ha cambiado, y un resumen del commit. Debería ser de no más de 50 caracteres.

Se debería utilizar una línea en blanco para separarlo del resto del mensaje de commit.

El resto del mensaje se debería limitar a no más de 72 caracteres por línea.

Si hay campos de metadatos se debería utilizar otra línea en blanco, de forma que se distingan fácilmente del mensaje de commit.

5.2.8. Gestionando Cambios Locales

This section addresses tracking local changes. If you have no local changes you can skip this section.

Un punto que es importante para todos ellos: todos los cambios son locales hasta que se hace push. A diferencia de Subversion, Git utiliza un modelo distribuido. Para la mayoría de los usuarios y los casos, hay poca diferencia. Sin embargo, si tienes cambios locales, puedes usar la misma herramienta para gestionarlos que la que usara para traerte los cambios de FreeBSD. Todos los cambios para los que no has hecho push son locales y se pueden cambiar fácilmente (git rebase, discutido más abajo hace esto).

Manteniendo cambios locales

La forma más sencilla de mantener cambios locales (especialmente si son triviales) es usar `git stash`. En su forma más simple, utilizas `git stash` para grabar los cambios (lo que los empuja a la pila del stash). La mayoría de la gente utiliza esto para guardar cambios antes de actualizar un árbol como se describe arriba. Después utilizan `git stash apply` para reaplicarlos al árbol. El stash es una pila de cambios que se puede examinar con `git stash list`. La página del manual de git-stash (https://git-scm.com/docs/git-stash) tiene todos los detalles.

Este método va bien cuando tienes pequeños cambios en el árbol. Cuando tienes algo no trivial, probablemente sea mejor mantener una rama local y rebasarla. Guardar los cambios (stashing) también es algo integrado en el comando `git pull`: simplemente añade `--autostash` en la línea de comando.

Manteniendo una rama local

Es mucho más fácil mantener una rama local con Git que con Subversion. En Subversion necesitas mergear el commit, y resolver los conflictos. Esto es manejable, pero puede llevar a un histórico complicado que es difícil de mover al origen (upstream) si fuera necesario, o difícil de replicar si lo necesitas. Git también permite mergear, con los mismos problemas. Esa es una forma de gestionar la rama, pero es la menos flexible.

Además de hacer merging, Git soporta el concepto de rebase que evita estos problemas. El comando `git rebase` rehace todos los commits de una rama en un lugar nuevo de la rama padre. Cubriremos los casos más comunes que surgen al usarlo.

Crear una rama

Digamos que quieres hacer un cambio en el comando ls de FreeBSD para que nunca use colores. Hay muchas razones para hacer esto, pero en este ejemplo usaremos esto como punto de partida. El comando ls de FreeBSD cambia de cuándo en cuándo y necesitarás lidiar con esos cambios. Afortunadamente, con Git rebase esto es algo normalmente automático.

```
% cd src
% git checkout main
% git checkout -b no-color-ls
% cd bin/ls
% vi ls.c     # hack the changes in
% git diff    # check the changes
```

```
diff --git a/bin/ls/ls.c b/bin/ls/ls.c
index 7378268867ef..cfc3f4342531 100644
--- a/bin/ls/ls.c
+++ b/bin/ls/ls.c
@@ -66,6 +66,7 @@ __FBSDID("$FreeBSD$");
 #include <stdlib.h>
 #include <string.h>
 #include <unistd.h>
+#undef COLORLS
 #ifdef COLORLS
 #include <termcap.h>
 #include <signal.h>
% # these look good, make the commit...
% git commit ls.c
```

El commit te llevará a un editor para que describas lo que has hecho. Una vez hecho esto, tienes tu propia rama **local** en el repo de Git. Compila e instala como harías normalmente, siguiendo las instrucciones del manual. Git es diferente a otros sistemas de control de versiones en cuanto que tienes que decirle explícitamente qué ficheros quieres incluir en el commit. He optado por hacerlo en la linea de comando pero también puedes hacerlo con `git add` que se cubre en muchos de los tutoriales más detallados.

Momento de actualizar

Cuando es momento de sacar una nueva versión, es casi lo mismo que sin ramas. Actualizarías como se ha hecho arriba, pero hay un comando extra antes de que actualices y uno después. Lo que sigue asume que empiezas con un árbol sin modificar. Es importante empezar las operaciones de rebase con un árbol limpio (es un requisito en Git).

```
% git checkout main
% git pull --ff-only
% git rebase -i main no-color-ls
```

Eso arrancará un editor que lista todos los commits. Para este ejemplo, no lo cambies. Esto es típicamente lo que haces mientras actualizas la base (aunque también puedes utilizar el comando rebase de Git para filtrar los commits que quieres en la rama).

Una vez que has terminado con lo de arriba, tienes que avanzar los commits de ls.c desde la versión vieja de FreeBSD a la nueva.

A veces hay conflictos al fusionar. Está bien. No te asustes. En lugar de eso, trátalos como cualquier otro conflicto de merge. Para hacerlo sencillo, simplemente describiré un problema común que puede aparecer. Se puede encontrar un enlace a un tratamiento más completo al final de esta sección.

Digamos que los includes cambian en el proyecto origen de una forma radical para terminfo así como también un cambio de nombre para la opción. Cuando te actualizaste, podrías haber visto algo como esto:

```
Auto-merging bin/ls/ls.c
CONFLICT (content): Merge conflict in bin/ls/ls.c
error: could not apply 646e0f9cda11... no color ls
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply 646e0f9cda11... no color ls
```

que da miedo. Si abres un editor, verás que es una resolución de conflicto típica de 3 vías con la que podrías estar familiarizado de otros sistemas de control de código (el resto de ls.c se ha omitido):

```
 <<<<<<< HEAD
 #ifdef COLORLS_NEW
 #include <terminfo.h>
 =======
 #undef COLORLS
 #ifdef COLORLS
 #include <termcap.h>
 >>>>>>> 646e0f9cda11... no color ls
....
El código nuevo está primero, y tu código segundo.
El arreglo correcto aquí es añadir simplemente #undef COLORLS_NEW ante de @ifdef y
después borrar los cambios antiguos:
[source,shell]
....
#undef COLORLS_NEW #ifdef COLORLS_NEW #include <terminfo.h>
....
guarda el fichero.
El rebase fue interrumpido, así que tienes que completarlo:
[source,shell]
....
% git add ls.c % git rebase --continue
....
```

que le dice a Git que ls.c ha sido arreglado y que puede continuar con el rebase. Puesto que hubo un conflicto, se te dirigirá al editor para actualizar el mensaje de commit si es necesario. Si el mensaje sigue siendo preciso, simplemente sal del editor.

Si te atascas durante el rebase, no te asustes. git rebase --abort te llevará de nuevo a un estado limpio. Sin embargo, es importante empezar con un árbol sin modificar. Una nota: el `git reflog` mencionado arriba es útil aquí ya que tendrá una lista de todos los commits (intermedios) que puedes ver, inspeccionar o seleccionar con cherry-pick.

Para saber más sobre esto, https://www.freecodecamp.org/news/the-ultimate-guide-to-git-merge-and-git-rebase/ proporciona un tratamiento bastante amplio. Es un buen recursos para problemas que puedan surgir de forma ocasional pero que son muy oscuros para esta guía.

Cambiando a una Rama Diferente de FreeBSD

Si quieres cambiar de stable/12 a la rama current. Si tienes un clonado profundo, lo siguiente es suficiente: [source,shell]

```
% git checkout main % # build and install here...
```

Sin embargo, si tienes una rama local, hay algún problema. Primero, rebase sobreescribirá el histórico de forma que querrás hacer algo para salvarlo. Segundo, saltar entre ramas suele causar más conflictos. Si imaginamos que el ejemplo anterior era relativo a stable/12, entonces para moverlo a `main`, sugeriría lo siguiente:

```
% git checkout no-color-ls
% git checkout -b no-color-ls-stable-12   # create another name for this branch
% git rebase -i stable/12 no-color-ls --onto main
```

Lo anterior se trae no-color-ls. Luego le da un nombre nuevo (no-color-ls-stable-12) en caso de que necesites volver a ella. Después rebase sobre la rama `main`. Esto encontrará todos los commits de la rama no-color-ls actual (hacia atrás hasta donde se encuentra con la rama stable/12) y después los aplicará de nuevo sobre la rama main creando una nueva rama no-color-ls allí (para lo cual te hice crear un nombre tipo place holder).

5.3. Procedimientos MFC (Merge From Current)

5.3.1. Resumen

El flujo de trabajo de MFC se puede resumir como `git cherry-pick -x` más `git commit --amend` para ajustar el mensaje de commit. Para múltiples commits, usa `git rebase -i` para refundirlos juntos y editar el mensaje de commit.

5.3.2. MFC de un sólo commit

```
% git checkout stable/X % git cherry-pick -x $HASH --edit
```

Para commits MFC, por ejemplo una importación externa, necesitarías especificar un padre para cherry-pick. Normalmente, sería el "primer padre" de la rama de la que estás haciendo cherry-pick, así que:

```
% git checkout stable/X % git cherry-pick -x $HASH -m 1 --edit
```

Si algo va mal, necesitarás abortar el cherry-pick con `git cherry-pick --abort` o arreglarlo y hacer un `git cherry-pick --continue`.

Una vez terminado el cherry-pick, empuja con `git push`. Si recibes un error por haber perdido una

carrera por el commit, utiliza `git pull --rebase` y prueba a empujarlo de nuevo.

5.3.3. MFC a una rama RELENG

Se necesita más cuidado para hacer MFCs a ramas para las cuales se necesita aprobación. El proceso es el mismo tanto para un merge típico como para un commit directo excepcional.

- Integra o hace commit directamente a la rama `stable/X` apropiada antes de integrarlo en la rama `releng/X.Y`.
- Utiliza el hash que está en la rama `stable/X` para el MFC a la rama `releng/X.Y`.
- Deja ambas líneas "cherry picked from" en el mensaje de commit.
- Asegúrate de añadir la línea `Approved by:` cuando estés en el editor.

```
% git checkout releng/13.0 % git cherry-pick -x $HASH --edit
```

Si se te olvida añadir la línea `Approved by:`, puedes hacer un `git commit --amend` para editar el mensaje de commit antes de empujar los cambios.

5.3.4. MFC de varios commits

```
% git checkout -b tmp-branch stable/X % for h in $HASH_LIST; do git cherry-pick -x $h;
done % git rebase -i stable/X # mark each of the commits after the first as 'squash' #
Actualiza el mensaje de commit para reflejar todos los cambios del mismo, si fuera
necesario. # Asegúrate de mantener las líneas "cherry picked from". % git push freebsd
HEAD:stable/X
```

Si el push falla por perder la carrera del commit, haz rebase y prueba de nuevo:

```
% git checkout stable/X % git pull % git checkout tmp-branch % git rebase stable/X %
git push freebsd HEAD:stable/X
```

Una vez que el MFC se ha completado, puedes borrar la rama temporal:

```
% git checkout stable/X % git branch -d tmp-branch
```

5.3.5. Haciendo MFC de una importación externa

Las importaciones externas son lo único en el árbol que crean un commit tipo merge en la rama `main`. Seleccionar commits tipo merge en stable/XX representa una dificultad adicional porque hay dos padres para un commit tipo merge. En general, querrás la diferencia del primer padre ya que es la diferencia con `main` (aunque podría haber algunas excepciones).

```
% git cherry-pick -x -m 1 $HASH
```

es normalmente lo que quieres. Esto le dirá a cherry-pick que aplique el diff correcto.

Hay algunos pocos casos (con suerte) donde es posible que la rama `main` haya sido mergeada hacia atrás por el script de conversión. Si ese fuera el caso (y todavía no hemos encontrado ninguno), cambiarías lo de arriba por '-m 2' para escoger el padre adecuado. Simplemente haz:

```
% git cherry-pick --abort % git cherry-pick -x -m 2 $HASH
```

para hacerlo. El `--abort` limpiará el primer intento fallido.

5.3.6. Rehaciendo un MFC

Si haces un MFC y va terriblemente mal y quieres empezar de nuevo, lo más fácil es usar `git reset --hard` así: [source,shell]

```
% git reset --hard freebsd/stable/12
```

aunque si tienes algunas revisiones que quieres mantener, y otras que no,es mejor usar `git rebase -i`.

5.3.7. Consideraciones cuando se hace un MFC

Cuando se hace commit the commits the código fuente a las ramas stable y releng, tenemos los siguientes objetivos:

- Señala claramente los commits directos de aquellos que introducen un cambio desde otra rama.
- Evita introducir errores en las ramas stable y releng.
- Permite a los desarrolladores determinar qué cambias han sido o no traídos desde otra rama.

Con Subversion, usábamos las siguientes prácticas para conseguir estos objetivos:

- Usar las etiquetas `MFC` y `MFS` para marcar los commits que integran cambios desde otra rama.
- Compactar los commits de correcciones en el commit principal cuando se integra un cambio.
- Grabar mergeinfo de forma que `svn mergeinfo --show-revs` funcionara.

Con Git, necesitaremos usar diferentes estrategias para conseguir los mismos objetivos. Este documento trata de definir las mejores prácticas para conseguir estos objetivos con Git cuando se mergean cambios de código fuente. En general, tratamos de usar el soporte nativo de Git para conseguir los objetivos en lugar de forzar a realizar las prácticas construidas sobre el modelo de Subversion.

Una nota general: debido a las diferencias técnicas con Git, no utilizaremos los "merge commits" de Git (creados mediante `git merge`) en las ramas stable o releng. En su lugar, cuando este documento

habla de "merge commits", significa el commit original hecho en `main` que es replicado o "aterrizado" (landed) en una rama stable, o un commit de una rama stable que es replicado a una rama releng con alguna variación de `git cherry-pick`.

5.3.8. Encontrando Hashes Seleccionables para MFC

Git proporciona algo de soporte para esto mediante los comandos `git cherry` y `git log --cherry`. Estos comandos comparan los diffs en crudo de los commits (pero no otros metadatos como los mensajes de log) para determinar si dos commits son idénticos. Esto funciona bien cuando cada commit de `main` se lleva como un sólo commit a la rama stable, pero falla si múltiples commits de `main` se compactan juntos como un sólo commit en la rama stable. El proyecto utiliza mucho `git cherry-pick -x` preservando todas las líneas para evitar estas dificultades y funciona con herramientas automatizadas.

5.3.9. Estándares para los mensajes de commit

Marcar MFCs

El proyecto ha adoptado las siguientes prácticas para marcar MFCs:

- Usa el flag `-x` con `git cherry-pick`. Esto añade una línea al mensaje de commit que incluye el hash del commit original cuando se hace el merge. Puesto que Git lo añade directamente, los committers no tienen que editar manualmente el log cuando hacen el merge.

Cuando se mergean varios commits, mantén todas las líneas "cherry picked from".

¿Recortar Metadatos?

Un área que no estaba documentada de forma clara con Subversion (ni con CVS) era cómo formatear los metadatos en los mensajes de log para los commits tipo MFC. ¿Debería incluir los metadatos del commit original sin modificar o se debería modificar para reflejar la información acerca del propio commit MFC?

Históricamente la práctica ha variado, aunque parte de la variación es por campo. Por ejemplo, MFCs relativos a un PR normalmente incluyen el campo PR en el MFC de forma que los commits MFC se incluyen en el log de autoría del sistema de reportes de error (bug tracker). Con otros campos está menos claro. Por ejemplo, Phabricator muestra la diferencia entre el último commit etiquetado a una revisión, de forma que incluir URLs de Phabricator reemplaza el commit principal con los commits "aterrizados". La lista de revisores tampoco está clara. Si un revisor ha aprobado un cambio a `main`, ¿significa eso que han aprobado el commit MFC? ¿Es cierto si el código es idéntico o con sólo cambios triviales? Claramente no es cierto para trabajos más extensivos. Incluso para código idéntico ¿qué pasa si el commit no tiene conflicto pero introduce un cambio en el ABI? Un revisor podría haber dado el visto bueno para un commit en `main` debido al rompimiento del ABI pero podría no aprobar el mergeado del mismo commit tal cual. Cada uno tiene que usar su mejor juicio hasta que acordemos unas directrices claras.

Para MFCs que están regulados por re@, se añaden nuevos campos de metadatos como la etiqueta Approved by para commits aprobados. Estos nuevos metadatos se tendrán que añadir con `git commit --amend` o similar después de que el commit original haya sido revisado y aprobado.

También podríamos querer reservar algunos campos en los metadatos de los commtis MFC como las URLs de Phabricator para uso futuro por parte de re@.

Preservar los metadatos existentes proporciona un flujo de trabajo sencillo. Los desarrolladores usan `git cherry-pick-x` sin tener que editar el mensaje de log.

Si por el contrario escogemos ajustar los metadatos en los MFCs, los desarrolladores tendrán que editar los mensajes de log de forma explícita mediante el uso de `git cherry-pick --edit` o `git commit --amend`. Sin embargo, comparado con svn, al menos el mensaje de commit existente se puede precargar y los campos de metadatos se pueden añadir o eliminar sin tener que reescribir el mensaje de commit entero.

La conclusión es que los desarrolladores seguramente tengan que refinar los mensajes de commit para los MFCs que no sean triviales.

5.4. Importaciones Externas con Git

Esta sección describe en detalle el procedimiento para hacer importaciones de terceros con Git.

5.4.1. Convenciones en el nombrado de ramas

Todas las ramas de terceros y etiquetas comienzan con `vendor/`. Estas ramas y etiquetas son visibles por defecto.

[NOTE] ==== Este capítulo sigue la convención de que el origen `freebsd` es el nombre del origen del repositorio Git oficial de FreeBSD. Si usas otra convención, en los ejemplos de abajo reemplaza `freebsd` con el nombre que uses en su lugar. ====

Exploraremos un ejemplo para actualizar el mtree de NetBSD que está en nuestro árbol. La rama externa para esto es `vendor/NetBSD/mtree`.

5.4.2. Actualizando una importación externa antigua

Los árboles externos normalmente tienen sólo un subconjunto del software de terceros que es apropiado para FreeBSD. Estos árboles son muy pequeños en comparación con el árbol de FreeBSD. Los worktrees de Git son por lo tanto bastante pequeños y rápidos y el método preferido a usar. Asegúrate de que el directorio que escojas debajo (el `../mtree`) no existe.

```
% git worktree add ../mtree vendor/NetBSD/mtree
```

5.4.3. Actualizar las Fuentes en la Rama Vendor

Prepara un árbol limpio, completo con las fuentes externas. Importa todo pero mergea sólo lo que es necesario.

Este ejemplo asume que las fuentes de NetBSD se han traído de su mirror de GitHub en `~/git/NetBSD`. Date cuenta de que "upstream" podría haber añadido o eliminado ficheros, por lo que queremos asegurarnos de que los borrados también se propagan. Normalmente net/rsync está

instalado así que lo usaremos.

```
% cd ../mtree
% rsync -va --del --exclude=".git" ~/git/NetBSD/usr.sbin/mtree/ .
% git add -A
% git status
...
% git diff --staged
...
% git commit -m "Vendor import of NetBSD's mtree at 2020-12-11"
[vendor/NetBSD/mtree 8e7aa25fcf1] Vendor import of NetBSD's mtree at 2020-12-11
 7 files changed, 114 insertions(+), 82 deletions(-)
% git tag -a vendor/NetBSD/mtree/20201211
```

Nota: Ejecuto los comandos `git diff` y `git status` para asegurarme de que no hay nada raro. También usé `-m` de forma ilustrativa, pero tú deberías componer un mensaje apropiado en un editor (usando una plantilla para el mensaje de commit).

También es importante crear una etiqueta anotada utilizando `git tag -a`, de lo contrario el push será rechazado. Sólo se permite hacer push de etiquetas anotadas. Las etiquetas anotadas te dan una oportunidad de introducir un mensaje de commit. Introduce la versión que estás importando así como cualquier característica que resalte o arreglos que lleve la versión.

5.4.4. Actualizando la Copia de FreeBSD

En este momento puedes empujar la importación a `vendor` en nuestro propio repo.

```
% git push --follow-tags freebsd vendor/NetBSD/mtree
```

`--follow-tags` le dice a `git push` que también empuje las etiquetas asociadas con la revisión local de la que se ha hecho commit.

5.4.5. Actualizando el árbol de fuentes de FreeBSD

Ahora necesitas actualizar el mtree en FreeBSD. Las fuentes están en `contrib/mtree` ya que es software de terceros.

```
% cd ../src % git subtree merge -P contrib/mtree vendor/NetBSD/mtree
```

Esto generaría un commit merge para el subárbol `contrib/mtree` contra la rama local `vendor/NetBSD/mtree`. Si hubiera conflictos, necesitarías arreglarlos antes de hacer el commit. Incluye detalles en el mensaje de commit acerca de los cambios que se están mergeando.

5.4.6. Rebasando to cambio contra lo último del árbol de fuentes de FreeBSD

Puesto que la política actual no recomienda utilizar meges, si el `main` de FreeBSD remoto avanzó antes de que tuvieras oportunidad de hacer el push, tendrías que rehacer el merge.

Los `git rebase` o `git pull --rebase` habituales no saben cómo rebasar un commit tipo merge **como un commit merge**, así que tendrías que recrear el commit.

Se deberían seguir los siguientes pasos para facilitar recrear el commit tipo merge como si `git rebase --merge-commits` hubiese funcionado adecuadamente:

- Muévete al directorio raíz del repositorio
- Crea una rama `XXX` con el **contenido** del árbol mergeado.
- Actualiza este lado de la rama `XXX` para mergearla y tenerla actualizada respecto a la rama `main` de FreeBSD.
  - En el peor caso, tendrías que resolver conflictos, si hubiera alguno, pero esto debería ser raro.
  - Resuelve los conflictos, y compacta varios commits en uno si es necesario (si no hay conflictos, no hay necesidad de compactar)
- Haz checkout de `main`
- crea una rama `YYY` (permite deshacer los cambios si algo va mal)
- Rehaz el merge del subárbol
- En lugar de resolver conflictos en el subárbol mergeado, haz un checkout del contenido de XXX encima de él.
  - El último `.` es importante, igual que lo es estar en el directorio raíz del repositorio.
  - En lugar de cambiar a la rama XXX, pone el contenido de XXX sobre el repositorio.
- Haz commit del repositorio con el mensaje de commit anterior (el ejemplo asume que sólo hay un merge en la rama XXX).
- Asegúrate de que las ramas son iguales.
- Haz las revisiones que necesites, incluyendo involucrar a otros si crees que es necesario.
- Empuja el commit, si has 'perdido la carrera' otra vez, simplemente haz otra vez estos pasos (lee más abajo para una receta)
- Borra las ramas una vez que el commit está en el repositorio. Son desechables.

Los comandos que uno usaría, siguiendo el ejemplo de mtree, sería como esto (el símbolo `#` marca un comentario para ayudar y enlazar los comandos con las descripciones de arriba):

```
% cd ../src         # cambiar a la raíz del árbol
% git checkout -b XXX       # crea la rama XXX de usar y tirar para hacer el merge
% git fetch freebsd     # Obtiene los datos de upstream
% git merge freebsd/main    # Mergea los cambios y resuelve conflictos
% git checkout -b YYY freebsd/main # Crea una nueva rama de usar y tirar YYY para
```

```
rehacer
% git subtree merge -P contrib/mtree vendor/NetBSD/mtree # Redo subtree merge
% git checkout XXX .        # La rama XXX tiene la resolución del conflicto
% git commit -c XXX~1       # -c reutiliza el mensaje de commit del commit anterior al
rebase
% git diff XXX YYY      # Debería estar vacío
% git show YYY          # Sólo debería tener los cambios que quieres, y ser un commit
merge desde la rama del vendor
```

Nota: si algo va mal con el commit, puedes resetear la rama `YYY` para comenzar de nuev volviendo a ejecutar el comando checkout que la creó con -B :

```
% git checkout -B YYY freebsd/main # Crea una nueva rama YYY de usar y tirar si
empezar desde cero es más sencillo
```

5.4.7. Empujando los cambios

Una vez que crees que tienes un conjunto de diferencias que es bueno, puedes empujarlo a un fork de GitHub o Gitlab para que otros lo revisen. Una cosa buena de Git es que te permite publicar borradores de tu trabajo para que otros lo revisen. Mientras que Phabricator es bueno para revisión de contenido, publicar una rama externa actualizada y los commits tipo merge permite a otros comprobar los detalles tal y como aparecerán eventualmente en el repositorio.

Después de la revisión, cuando estás seguro de que es un buen cambio, puedes empujarlo al repo de FreeBSD:

```
% git push freebsd YYY:main # put the commit on upstream's 'main' branch % git branch
-D XXX        # Throw away the throw-a-way branches. % git branch -D YYY
```

Nota: He usado `XXX` y `YYY` para que sea obvio que son nombres horribles que no deberían abandonar tu máquina. Si usas esos nombres para otro trabajo, necesitarás escoger nombres diferentes, o arriesgarte a perder el otro trabajo. No hay nada mágico sobre estos nombres. Upstream no te permitirá empujarlos, pero de todas formas, por favor presta atención a los comandos exactos de arriba. Algunos comandos usan sintaxis que es algo diferente respecto de los casos típicos y ese comportamiento diferente es crítico para que esta receta funcione.

5.4.8. Como rehacer cosas si es necesario

Si has intentado empujar los cambios de la sección anterior y ha fallado, entonces deberías hacer lo siguiente para 'rehacer' las cosas. Esta secuencia mantiene el commit cno el mensaje de commit simpre en XXX~1 para que sea más fácil.

```
% git checkout -B XXX YYY   # recreate that throw-away-branch XXX and switch to it %
git merge freebsd/main # Merge the changes and resolve conflicts % git checkout -B YYY
freebsd/main # Recreate new throw-away YYY branch for redo % git subtree merge -P
contrib/mtree vendor/NetBSD/mtree # Redo subtree merge % git checkout XXX .       #
```

```
XXX branch has the conflict resolution % git commit -c XXX~1      # -c reuses the
commit message from commit before rebase
```

Después haz el checkout como arriba y empuja los cambios como arriba cuando estén listos.

5.5. Crear una nueva rama externa

Hay varias formas de crear una nueva rama externa. La forma recomendada es crear un nuevo repositorio y después mergearlo con FreeBSD. Supongamos que se importa `glorbnitz` en el árbol de FreeBSD, release 3.1415. Por simplicidad, no recortaremos esta release. Es un simple comando de usuario que pone el dispositivo nitz en diferentes estados mágicos glorb y es suficientemente pequeño como para que recortarlo no ahorre demasiado.

5.5.1. Crear el repo

```
% cd /some/where % mkdir glorbnitz % cd glorbnitz % git init % git checkout -b
vendor/glorbnitz
```

En este momento, tienes un nuevo repo, donde irán todos los commits de la rama `vendor/glorbnitz`.

Los expertos en Git pueden hacer esto directamente en su clon de FreeBSD usando `git checkout --orphan vendor/glorbnitz` si así se sienten más cómodos.

5.5.2. Copia las fuentes

Puesto que es una nueva importación, puedes simplemente usar cp, o tar o incluso rsync como se muestra arriba. Y añadiremos todo, asumiendo que no hay ficheros dot.

```
% cp -r ~/glorbnitz/* . % git add *
```

En este punto, deberías tener una copia prístina de glorbnitz lista para hacer commit.

```
% git commit -m "Import GlorbNitz frobnosticator revision 3.1415"
```

Como arriba, he usado `-m` por simplicidad, pero seguramente deberías crear un mensaje de commit que explica qué es un Glorb y por qué usarías un Nitz para conseguirlo. No todo el mundo lo sabrá así que para tu commit de verdad, deberías seguir la sección mensaje de log del commit en lugar de emular el estilo corto utilizado aquí.

5.5.3. Ahora importa en nuestro repositorio

Ahora necesitas importar la rama en nuestro repositorio.

```
% cd /path/to/freebsd/repo/src % git remote add glorbnitz /some/where/glorbnitz % git
```

```
fetch glorbnitz vendor/glorbnitz
```

Fíjate que la rama vendor/glorbnitz está en el repo. En este momento puedes borrar `/some/where/glorbnitz` si quieres. Ha cumplido su labor.

5.5.4. Etiquetas y push

Los pasos desde aquí en adelante son básicamente los mismos que en el caso de la actualización de una rama externa, aunque sin el paso de actualizar la rama externa.

```
% git worktree add ../glorbnitz vendor/glorbnitz % cd ../glorbnitz % git tag
--annotate vendor/glorbnitz/3.1415 # Make sure the commit is good with "git show" %
git push --follow-tags freebsd vendor/glorbnitz
```

Por 'bueno' nos referimos a:

1. Todos los ficheros están presentes
2. Ninguno de los ficheros erróneos está presente
3. La rama vendor apunta a algo que tiene sentido
4. La etiqueta tienen buena pinta, y está anotada
5. El mensaje de commit para la etiqueta tiene un resumen con las novedades respecto de la última etiqueta

5.5.5. Momento de mergear finalmente en el árbol base

```
% cd ../src
% git subtree add -P contrib/glorbnitz vendor/glorbnitz
# Make sure the commit is good with "git show"
% git commit --amend   # one last sanity check on commit message
% git push freebsd
```

Aquí 'bueno' significa:

1. Todos los ficheros correctos, y ninguno de los incorrectos, se mergearon en contrib/glorbnitz.
2. No hay otros cambios en el árbol.
3. Los mensajes de commit están bien. Debería contener un resumen de lo que ha cambiado desde el último merge a la rama `main` de FreeBSD así como cualquier problema.
4. Se debería actualizar UPDATING si hay algo que reseñar, como cambios visibles por el usuario, preocupaciones sobre la actualización, etc.

Todavía no hemos conectado `glorbnitz` a la construcción. Hacerlo es específico al software que se importa y está fuera del alcance de este tutorial.

Mantenerse actualizado

El tiempo pasa. Es momento de actualizar el árbol con los últimos cambios. Cuando haces un checkout de `main` asegúrate de que no tienes diferencias. Es mucho más fácil hacer commit de esos cambios en una rama (o utilizar `git stash`) antes de hacer lo siguiente.

Si estás acostumbrado a `git pull` recomendamos encarecidamente el uso de la opción `--ff-only` y además establecerla como la opción por defecto. De forma alternativ, `git pull --rebase` es útil si tienes cambios guardados en la rama `main`.

```
% git config --global pull.ff only
```

Podrías necesitar omitir el --global si quieres que esta configuración sólo aplique en este repositorio.

```
% cd freebsd-src % git checkout main % git pull (--ff-only|--rebase)
```

Hay un problema habitual, que la combinación del comando `git pull` intentará hacer un merge, que algunas veces creará un commit de tipo merge que no existía antes. Esto puede ser difícil de arreglar.

La forma larga también se recomienda.

```
% cd freebsd-src % git checkout main % git fetch freebsd % git merge --ff-only
freebsd/main
```

Estos comandos restauran tu árbol a la rama `main` y después lo actualizan desde donde hiciste el pull originalmente. Es importante cambiarse a `main` antes de hacer esto de forma que avance. Ahora es momento de avanzar los cambios:

```
% git rebase -i main working
```

Esto traerá un pantalla interactiva para cambiar los valores por defecto. Por ahora, simplemente sal del editor. Todo debería aplicar. Si no, necesitarás resolver los diffs. Este documento de github te puede ayudar en el proceso.

Momento de empujar los cambios

Primero, asegúrate de que la URL de push está correctamente configurada para el repositorio remoto.

```
% git remote set-url --push freebsd ssh://git@gitrepo.freebsd.org/src.git
```

Después, verifica que el usuario y el email están correctamente configurados. Requerimos que coincidan exactamente con la entrada del fichero passwd del clúster de FreeBSD.

Usa

```
freefall% gen-gitconfig.sh
```

en freefall.freebsd.org para obtener un texto que puedes usar directamente, asumiendo que /usr/local/bin está en el PATH.

El comando de abajo integra la rama `working` en la línea principal. Es importante que filtres tus cambios para que sean justo lo que quieres en el repo de fuentes de FreeBSD antes de hacer esto. Esta sintaxis empuja la rama `working` a `main`, avanzando la rama `main`. Sólo podrás hacer esto si resulta en un cambio lineal a `main`(es decir, no merges).

```
% git push freebsd working:main
```

Si se rechaza tu push debido a que perdiste una carrera, haz un rebase de tu rama antes de intentarlo de nuevo:

```
% git checkout working % git fetch freebsd % git rebase freebsd/main % git push
freebsd working:main
```

Momento de empujar los cambios (alternativa)

Algunas personas encuentran más fácil mergear sus cambios a su `main` local antes de empujarlos al repositorio remoto. También `git arc stage` mueve los cambios de una rama al `main` local cuando necesitas hacer un subconjunto de una rama. Las instrucciones son similares a las de la sección anterior: [source,shell]

```
% git checkout main % git merge --ff-only `working` % git push freebsd
```

Si pierdes la carrera, inténtalo de nuevo con

```
% git pull --rebase % git push freebsd
```

Estos comandos recuperarán el `freebsd/main` más reciente y después rebasará los cambios del `main` local encima, que es lo que quieres cuando pierdes una carrera por el commit. Nota: integrar commits de ramas externas no funcionará con esta técnica.

Encontrar la Revisión de Subversion

Tendrás que asegurarte de que has recuperado las notas (lee Uso diario para más detalles). Una vez que las tengas, las notas se mostrarán el comando git log de la siguiente forma:

```
% git log
```

Si tienes una versión específica en mente, puedes utilizar esto:

```
% git log --grep revision=XXXX
```

para encontrar la revisión específica. El número hexadecimal después de 'commit' es el hash que puedes usar para referirte a este commit.

5.6. Git FAQ

Esta sección proporciona un número de respuestas para usuarios y desarrolladores a preguntas que suelen surgir a menudo.

> Usamos la convención habitual de tener el origen del repositorio de FreeBSD en 'freebsd' en lugar del 'origin' por defecto para permitir que la gente use ese para sus propios desarrollo y para minimizar los pushes "ooops" al repositorio incorrecto.

5.6.1. Usuarios

Cómo puedo monitorizar -current y -stable con una sola copia del repositorio?

Q: Aunque el espacio en disco no es un asunto importante, es más eficiente usar sólo una copia del repositorio. Con SVN podía tener varios árboles del mismo repositorio. ¿Cómo hago esto con Git?

A: Puedes usar worktrees. Hay varias formas de hacer esto, pero la más sencilla es utilizar un clone para monitorizar -current, y un worktree para hacer lo mismo con las releases stables. Aunque usar un 'repositorio desnudo' se ha propuesto como una forma de lidiar con esto, es más complicado y no se documentará aquí.

Primero, necesitas un clon de un repositorio de FreeBSD, mostrado aquí en `freebsd-current` para reducir la confusión. $URL es el mirror que mejor que funcione:

```
% git clone -o freebsd --config remote.freebsd.fetch='+refs/notes/*:refs/notes/*' $URL
freebsd-current
```

que una vez clonado, puedes simplemente crear un worktree a partir de él:

```
% cd freebsd-current % git worktree add ../freebsd-stable-12 stable/12
```

esto se traerá `stable/12` a un directorio llamado `freebsd-stable-12` que es un análogo al directorio `freebsd-current`. Una vez creado se actualiza de forma similar a como cabría esperar:

```
% cd freebsd-current % git checkout main % git pull --ff-only # changes from upstream
now local and current tree updated % cd ../freebsd-stable-12 % git merge --ff-only
```

```
freebsd/stable/12 # now your stable/12 is up to date too
```

Recomiendo usar `--ff-only` porque es más seguro y evita que te metas accidentalmente en una 'pesadilla de integraciones' donde tienes un cambio extra en tu árbol, forzándote a una integración complicada en lugar de hacer uno sencillo.

Aquí hay un buen texto que tiene más detalles.

5.6.2. Desarrolladores

¡Ooops! He hecho commit en `main` en lugar de en otra rama.

Q: De vez en cuando meto la pata y hago un commit en `main` en lugar de una rama. ¿Qué hago?

A: Primero, que no te entre el pánico.

Segundo, no hagas push. De hecho, puedes arreglar casi cualquier cosa si no has hecho push. Todas las respuestas en esta sección asumen que no se ha hecho push.

La siguiente respuesta asume que has hecho commit en `main` y quieres crear una rama llamada `issue`:

```
% git branch issue                # Create the 'issue' branch
% git reset --hard freebsd/main    # Reset 'main' back to the official tip
% git checkout issue               # Back to where you were
```

¡Ooops! ¡He hecho commit de algo en la rama equivocada!

Q: Estaba trabajando en una característica en la rama `wilma`, pero accidentalmente he hecho commit de un cambio relacionado con la rama `fred` en la rama `wilma`. ¿Qué hago?

A: La respuesta es similar a la anterior pero escogiendo cambios (cherry picking). Se asume que sólo hay un commit en wilma, pero lo generalizaremos a situaciones más complicadas. También se asume que es el último commit en wilma (por lo tanto se usa wilma en el comando `git cherry-pick`), pero también se puede generalizar.

```
# We're on branch wilma % git checkout fred     # move to fred branch % git cherry-
pick wilma       # copy the misplaced commit % git checkout wilma        # go back to
wilma branch % git reset --hard HEAD^  # move what wilma refers to back 1 commit
```

Los expertos en Git primero rebobinarían la rama wilma en 1 commit, cambiarían a la rama fred y después usarían `git reflog` para ver cuál era el commit borrado para poder hacer cherry-pick sobre él.

Q: Pero ¿Y si quiero hacer commit de unos cuantos cambios a `main`, pero dejar el resto en `wilma` por algún motivo?

A: La misma técnica de arriba funciona si quieres llevar partes de la rama en la que estás

trabajando a `main` antes de que el resto de la rama está listo (digamos que has visto un error ortográfico no relacionado, o has arreglado un bug puntual). Puedes usar seleccionar esos cambios y llevarlos a `main`, luego empuja al repositorio padre. Una vez hecho esto, limpiar no podría ser más fácil: simplemente `git rebase -i`. Git se dará cuenta de que has hecho esto y omitirá los cambios comunes automáticamente (incluso si tienes que cambiar el mensaje de commit o modificar el commit ligeramente). No hay necesidad de cambiar de nuevo a wilma para ajustarlo: ¡simplemente rebásalo!

Q: Quiero separar algunos cambios de la rama `wilma` y llevarlos a una rama `fred`

A: La respuesta más general sería la misma que previamente. Crearías la rama `fred`, escogerías los cambios que quieres de `wilma` uno a uno, luego rebasa `wilma` para eliminar esos cambios que has seleccionado. `git rebase -i main wilma` te llevará a un editor, luego elimina las líneas `pick` que se corresponden con los cambios que has llevado a `fred`. Si todo va bien y no hay conflictos, has terminado. Si no, necesitarás resolver los conflictos sobre la marcha.

La otra forma de hacer esto sería hacer un checkout de `wilma` y luego crear la rama `fred` apuntando al mismo punto del árbol. Después puedes hacer `git rebase -i` en ambas ramas, seleccionando los cambios que quieres en `fred` o `wilma` manteniendo las líneas "pick" y eliminando el resto en el editor. Algunas personas crearían una etiqueta/rama llamada `pre-split` antes de empezar por si algo va mal. Puedes deshacerlo con la siguiente secuencia:

```
% git checkout pre-split    # Go back % git branch -D fred      # delete the fred
branch % git checkout -B wilma        # reset the wilma branch % git branch -d pre-
split  # Pretend it didn't happen
```

El último paso es opcional. Si vas a intentar hacer el split de nuevo, lo omitirías.

Q: Pero lo he hecho todo como he leído que se hacía y no he visto tu consejo al final para crear una rama y ahora `fred` y `wilma` están hechas un lío. ¿Cómo sé cuál era el estado de `wilma` antes de que empezara? No sé cuántas veces he movido las cosas de sitio.

A: No todo está perdido. Puedes averiguarlo, siempre que no haya pasado mucho tiempo o haya habido muchos commits (cientos).

Creé una rama wilma e hice commit de un par de cosas, luego decidí que quería dividirla en fred y wilma. No pasó nada raro cuando lo hice, pero digamos que hubiera sido así. La forma de ver lo que has hecho es con `git reflog`:

```
% git reflog 6ff9c25 (HEAD -> wilma) HEAD@{0}: rebase -i (finish): returning to
refs/heads/wilma 6ff9c25 (HEAD -> wilma) HEAD@{1}: rebase -i (start): checkout main
869cbd3 HEAD@{2}: rebase -i (start): checkout wilma a6a5094 (fred) HEAD@{3}: rebase -i
(finish): returning to refs/heads/fred a6a5094 (fred) HEAD@{4}: rebase -i (pick):
Encourage contributions 1ccd109 (freebsd/main, main) HEAD@{5}: rebase -i (start):
checkout main 869cbd3 HEAD@{6}: rebase -i (start): checkout fred 869cbd3 HEAD@{7}:
checkout: moving from wilma to fred 869cbd3 HEAD@{8}: commit: Encourage contributions
... %
```

Aquí vemos los cambios que he hecho. Puedes utilizarlo para averiguar dónde han empezado a ir mal las cosas. Señalaré unas pocosas cosas. La primera es que HEAD@{X} es algo relacionado con los commits de forma que lo puedes usar como argumento para algunos comandos. Aunque si ese comando hace commit de algo en el repositorio, la X cambia. También puedes usar el hash (primera columna).

Luego, 'Encourage contributions' fue el último commit que hice en `wilma` antes de que decidiera separar las ramas. Puedes ver ahí el mismo hash que cuando creé la rama `fred`. Empecé rebasando `fred` y puedes ver el 'start', cada paso y el 'finish' para ese proceso. Aunque no sea necesario ahora, puedes averiguar exactamente lo que pasó. Afortunadamente, para arreglar esto, puedes seguir los pasos de la respuesta anterior pero con el hash `869cbd3` en lugar de `pre-split`. Aunque puede parecer un poco verboso, es fácil de recordar ya que haces una cosa cada vez. También puedes apilar:

```
% git checkout -B wilma 869cbd3 % git branch -D fred
```

y ya estás listo para probar de nuevo. El 'checkout -B' con el hash combina hacer checkout y crear una rama. El -B en lugar de -b fuerza el movimiento de una rama pre-existente. De cualquiera de las maneras funciona, lo que está genial (y también es horrible) en Git. Un motivo por el que suelo usar `git checkout -B xxxx hash` en lugar de hacer checkout del hash y después crear / mover la rama es simplemente para evitar el mensaje ligeramente angustioso sobre los 'detached heads':

```
% git checkout 869cbd3 M    faq.md Note: checking out '869cbd3'.

You are in 'detached HEAD' state. You can look around, make experimental changes and
commit them, and you can discard any commits you make in this state without impacting
any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or
later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 869cbd3 Encourage contributions % git checkout -B wilma
```

esto produce el mismo efecto, pero tengo que leer mucho más y las cabezas cortadas (detached heads) no es una imagen que me guste contemplar.

¡Ooops! He hecho un `git pull` y he creado un commit tipo merge, ¿qué hago?

Q: Estaba con el piloto automático y he hecho `git pull` desde mi árbol de desarrollo y eso ha creado un commit tipo merge en la rama `main`. ¿Cómo lo recupero?

A: Esto puede pasar cuando invocas el pull con un checkout de tu rama de desarrollo.

Justo después del pull, tendrás en el checkout el nuevo commit tipo merge. Git soporta la sintaxis `HEAD^#` para examinar los padres de un commit tipo merge:

```
git log --oneline HEAD^1   # Look at the first parent's commits
git log --oneline HEAD^2   # Look at the second parent's commits
```

A partir de esos logs, puedes identificar fácilmente qué commit es tu trabajo de desarrollo. Después simplemente restaura tu rama al `HEAD^#` correspondiente:

```
git reset --hard HEAD^2
```

Q: Pero también necesito arreglar mi rama `main`. ¿Cómo lo hago?

A: Git controla las ramas del repositorio remoto en el espacio de nombres `freebsd/`. Para arreglar tu rama `main`, simplemente ponla apuntando al `main` de tu remoto:

```
git branch -f main freebsd/main
```

No hay nada mágico en las ramas de Git: tan sólo son etiquetas en un grafo que se mueven automáticamente hacia adelante cuando se hacen commits. Así que lo de arriba funciona porque tan sólo estamos moviendo una etiqueta. Debido a ello, no hay metadatos de la rama que se necesiten preservar.

Mezclando y combinando ramas

Q: Digamos que tengo dos ramas `worker` y `async` que me gustaría combinar en una rama llamada `feature` a la vez que mantengo los commits de ambas.

A: Esto es trabajo para cherry pick.

```
% git checkout worker % git checkout -b feature # create a new branch % git cherry-
pick main..async # bring in the changes
```

Ahora tienes una nueva rama llamada `feature`. Esta rama combina commits de ambas ramas. Puedes filtrar más utilizando `git rebase`.

Q: Tengo una rama llamada `driver` y me gustaría partirla en `kernel` y `userland` de forma que pueda hacerlas evolucionar por separado y hacer commit en cada rama cuando estén listas.

A: Esto necesita un poco de trabajo preparatorio, pero `git rebase` hará todo el trabajo duro.

```
% git checkout driver       # Checkout the driver % git checkout -b kernel  # Create
kernel branch % git checkout -b userland   # Create userland branch
```

Ahora tienes dos ramas idénticas. Es momento de separar los commits. Asumiremos inicialmente que todos los commits de `driver` van en las ramas `kernel` o en `userland` pero no en ambas.

```
% git rebase -i main kernel
```

y simplemente incluye los cambios que quieres (con una línea 'p' o 'pick') y borra los commits que no quieres (da miedo, pero si sucede lo peor, puedes tirar todo esto a la basura y empezar de nuevo con la rama `driver` ya que todavía no la has movido).

```
% git rebase -i main userland
```

y haz lo mismo que hiciste con la rama `kernel`.

Q: ¡Oh, genial! Seguí las instrucciones de arriba y me olvidé de hacer commit en la rama `kernel`. ¿Cómo lo arreglo?

A: Puedes usar la rama `driver` para encontrar el hash del commit que falta y seleccionarlo con cherry pick.

```
% git checkout kernel % git log driver % git cherry-pick $HASH
```

Q: OK. Tengo la misma situación que arriba, pero mis commits están todos mezclados. Necesito que partes de un commit vayan a una rama y el resto a otra. De hecho, tengo varias. Tu método basado en rebase suena complicado.

A: En esta situación, lo mejor sería filtrar la rama original para separar los commits y luego usar el método descrito arriba para separar las ramas.

Asumamos que sólo hay un commit con un árbol limpio. Puedes usar `git rebase` con una línea `edit` o puedes usarlo con el commit en el extremo (tip). Los pasos son los mismos de cualquiera de las dos formas. Lo primero que tenemos que hacer es echar atrás un commit mientras dejamos los cambios en el árbol sin hacer commit:

```
% git reset HEAD^
```

Nota: No añadas, repito no añadas `--hard` aquí porque esto también elimina los cambios de tu árbol.

Ahora, si tienes suerte, el cambio que necesita partirse cae completamente en las líneas del fichero. En ese caso puedes hacer el `git add` habitual para los ficheros de cada grupo y luego hacer `git commit`. Nota: cuando hagas esto, perderás el mensaje de commit al hacer el reset, así que si lo necesitas por algún motivo, deberías guardar una copia (aunque `git log $HASH` puede recuperarlo).

Si no tienes suerte, tendrás que partir ficheros. Hay otra herramienta para hacer eso que puedes aplicar en cada fichero.

```
git add -i foo/bar.c
```

iterará por los diffs, preguntándote a cada paso si quieres incluir o excluir un trozo del cambio.

Cuando hayas terminado, haz `git commit` y tendrás lo que quede en tu árbol. Puedes ejecutarlo varias veces también o incluso en varios ficheros (aunque encuentro más fácil hacerlo en un fichero cada vez y después utilizar `git rebase -i` para agrupar juntos commits que están relacionados).

5.6.3. Clonar y Duplicar (crear un mirror)

Q: Me gustaría crear un mirror de todo el repositorio Git, ¿cómo lo hago?

A: Si todo lo que quieres es un mirror, entonces

```
% git clone --mirror $URL
```

hará lo que quieres. Sin embargo, hay dos desventajas si quieres utilizar esto para algo más que hacer un mirror del cual crearás un clon.

Primero, esto es un 'repositorio desnudo' que tiene la base de datos del repositorio, pero no tiene ningún worktree. Esto es genial para crear un mirror, pero es terrible para el trabajo del día a día. Hay maneras de solventar esto con 'git worktree':

```
% git clone --mirror https://git.freebsd.org/ports.git ports.git % cd ports.git % git
worktree add ../ports main % git worktree add ../quarterly branches/2020Q4 % cd
../ports
```

Pero si no estás usando tu mirror para hacer más clones locales, entonces esta es una alternativa algo pobre.

La segunda desventaja es que Git normalmente sobrescribe las refs (nombres de ramas, etiquetas, etc) del repositorio remoto de forma que tus refs locales pueden evolucionar de forma independiente. Esto significa que perderás los cambios si haces commit a este repositorio en cualquier sitio que no sean ramas de proyectos privados.

Q: ¿Qué puedo hacer entonces?

A: Puedes agrupar todas las refs del repositorio remoto en un espacio de nombres privado en tu repositorio local. Git clona todo mediante un 'refspec' y el refspec por defecto es:

```
        fetch = +refs/heads/*:refs/remotes/freebsd/*
```

que le dice que se traiga las refs de la rama.

Sin embargo, el repositorio de FreeBSD tiene otras cosas. Para verlas, puedes añadir refspects de forma explícita para cada espacio de nombres o puedes traértelo todo. Para configurar tu repositorio para que haga eso:

```
git config --add remote.freebsd.fetch '+refs/*:refs/freebsd/*'
```

que pondrá todo el repositorio remoto en tu espacio de nombres 'refs/freebsd/' de tu repositorio local. Por favor, date cuenta de que esto también se trae ramas externas sin convertir y el número de refs que tienen asociadas es bastante grande.

Necesitarás hacer referencia a estas 'refs' con su nombre completo porque no son espacios de nombres regulares de Git.

```
git log refs/freebsd/vendor/zlib/1.2.10
```

mostraría el log de la rama externa para zlib comenzando en 1.2.10.

5.7. Colaborando con otros

Una de las claves para un buen desarrollo de software en un proyecto tan grande como FreeBSD es la habilidad para colaborar con otros antes de que empujes tus cambios al árbol. Los repositorios Git del proyecto FreeBSD todavía no permiten la creación de ramas de usuario que puedan ser empujadas al repositorio y por lo tanto si quieres compartir tus cambios con otros debes usar otro mecanismo como GitLab o GitHub, para compartir los cambios en una rama generada por el usuario.

Las siguientes instrucciones muestran cómo preparar una rama de usuario, basada en la rama `main` de FreeBSD y cómo empujarla a GitHub.

Antes de empezar, asegúrate de que tu repo local de Git está actualizado y tiene los orígenes correctos como se muestra arriba.

` % git remote -v freebsd https://git.freebsd.org/src.git (fetch) freebsd ssh://git@gitrepo.freebsd.org/src.git (push) `

El primer paso es crear un fork de FreeBSD en GitHub siguiendo estas instrucciones. El destino del fork debería ser tu propia cuenta personal de GitHub (en mi caso gvnn3).

Ahora añade un remoto a tu sistema local que apunte a tu fork: [source,shell]

```
% git remote add github git@github.com:gvnn3/freebsd-src.git % git remote -v github
git@github.com:gvnn3/freebsd-src.git (fetch) github git@github.com:gvnn3/freebsd-
src.git (push) freebsd https://git.freebsd.org/src.git (fetch) freebsd
ssh://git@gitrepo.freebsd.org/src.git (push)
```

Una vez hecho esto puedes crear una rama como se muestra arriba.

```
% git checkout -b gnn-pr2001-fix
```

Haz las modificaciones que quieras en tu rama. Compila, prueba y una vez que estés listo para colaborar con otros es momento de empujar tus cambios a la rama. Antes de que puedas hacerlo, deberás establecer el upstream apropiado, ya que Git te lo pedirá la primera vez que intentes empujar a tu remoto en github:

```
% git push github fatal: The current branch gnn-pr2001-fix has no upstream branch. To
push the current branch and set the remote as upstream, use

    git push --set-upstream github gnn-pr2001-fix
```

Establecer el push como git recomienda hace que se pueda completar con éxito:

```
% git push --set-upstream github gnn-feature
Enumerating objects: 20486, done.
Counting objects: 100% (20486/20486), done.
Delta compression using up to 8 threads
Compressing objects: 100% (12202/12202), done.
Writing objects: 100% (20180/20180), 56.25 MiB | 13.15 MiB/s, done.
Total 20180 (delta 11316), reused 12972 (delta 7770), pack-reused 0
remote: Resolving deltas: 100% (11316/11316), completed with 247 local objects.
remote:
remote: Create a pull request for 'gnn-feature' on GitHub by visiting:
remote:      https://github.com/gvnn3/freebsd-src/pull/new/gnn-feature
remote:
To github.com:gvnn3/freebsd-src.git
[new branch]                gnn-feature -> gnn-feature
Branch 'gnn-feature' set up to track remote branch 'gnn-feature' from 'github'.
```

Los siguientes cambios en la rama se podrán empujar correctamente con el comando por defecto:

```
% git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 314 bytes | 1024 bytes/s, done.
Total 3 (delta 1), reused 1 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:gvnn3/freebsd-src.git
   9e5243d7b659..cf6aeb8d7dda  gnn-feature -> gnn-feature
```

En este momento tu trabajo está en tu rama de GitHub y puedes compartir el enlace con otros colaboradores.

5.8. Traer al proyecto una pull request de github

Esta sección documenta cómo traerse una pull request de GitHub que se ha hecho contra los mirros de Git de FreeBSD en GitHub. Aunque en este momento esta no es una forma oficial de enviar parches, a veces buenos arreglos vienen de esta forma y es más fácil cogerlos del árbol de un committer que hacerles que lo empujen al árbol de FreeBSD desde ahí. Se pueden usar pasos similares para traerse ramas de otros repositorios. Cuando se hace commit de pull requests de

otros, se debe tener especial cuidado en examinar todos los cambios para asegurar que son exactamente lo que representan.

Antes de empezar, asegúrate de que tu repo local de Git está actualizado y de que tiene el origen correctamente establecido como se muestra arriba. Además, asegúrate de tener los siguientes orígenes: [source,shell]

```
% git remote -v freebsd https://git.freebsd.org/src.git (fetch) freebsd
ssh://git@gitrepo.freebsd.org/src.git (push) github
https://github.com/freebsd/freebsd-src (fetch) github
https://github.com/freebsd/freebsd-src (fetch)
```

Muchas veces las pull requests son sencillas: peticiones que contienen un sólo commit. En este caso, se puede utilizar una aproximación directa, aunque la aproximación de la sección anterior también funciona. Aquí se crea una rama, se selecciona el cambio con cherry pick, se ajusta el mensaje de commit y se hacen controles de calidad antes de empujar el cambio. En este ejemplo se usa la rama `staging` pero podría utilizarse cualquier nombre. Esta técnica funciona para cualquier número de commits que haya en la pull request, especialmente cuando el cambio se puede aplicar limpiamente al árbol de FreeBSD. Sin embargo, cuando hay varios commits, especialmente cuando se necesitan pequeños ajustes, `git rebase -i` funciona mejor que `git cherry-pick`. Brevemente, estos comandos crean una rama; seleccionan los cambios de la rama del pull request; los prueban; ajustan los mensajes de commit; y lo mergean de vuelta a `main` haciendo un fast forward. El número de PR abajo es `$PR`. Cuando se ajusta el mensaje, añade `Pull Request: https://github.com/freebsd-src/pull/$PR`. Todas las pull requests enviadas al repositorio de FreeBSD deberían ser revisadas por al menos una persona. No es necesario que sea la persona que hace el commit, pero en ese caso la persona que lo hace debería confiar en la competencia de los otros revisores para revisar el commit. Los committers que hacen revisión de código de una pull request antes de empujarla al repo deberían añadir una línea `Reviewed by:` al commit, porque en este caso no es implícito. Añade también a la línea `Reviewed by:` a cualquiera que revise y apruebe el commit en github. Como siempre, se debe poner cuidado para asegurar que el código hace lo que se supone que hace y que no hay código malicioso.

> Además, por favor asegúrate de que el nombre del autor de la pull request no es anónimo. El interfaz web de edición de GitHub genera nombres como:
>
> ```
> Author:     github-user <38923459+github-user@users.noreply.github.com>
> ```
>
> Se debería hacer una solicitud educada al autor para que proporcione un nombre mejor y/o un email. Se debería poner cuidado para asegurar de que no hay problemas de estilo ni se introduce código malicioso.

```
% git fetch github pull/$PR/head:staging % git rebase -i main staging   # to move the
staging branch forward, adjust commit message here <do testing here, as needed> % git
checkout main % git pull --ff-only      # to get the latest if time has passed % git
checkout main % git merge --ff-only staging <test again if needed> % git push freebsd
```

```
--push-option=confirm-author
```

Para pull requests complicadas que tienen varios commits con conflictos, sigue el siguiente esquema.

1. haz checkout de la pull request `git checkout github/pull/XXX`
2. crea una rama para hacer un rebase `git checkout -b staging`
3. rebasa la rama `staging` con lo último de `main` con `git rebase -i main staging`
4. resuelve conflictos y haz las pruebas que sean necesarias
5. haz fast forward de la rama `staging` in la rama `main` como arriba
6. últimas comprobaciones de cambios para asegurarse de que todo está bien
7. empuja al repositorio Git de FreeBSD.

Esto también funcionará cuando nos traigamos ramas desarrolladas en otros sitios hasta el árbol local para hacer commit.

Una vez que hayas terminado con la pull request, ciérrala usando el interfaz web de GitHub. Merece la pena mencionar que si tu origen `github` utiliza `https://`, el único paso para el que necesitas una cuenta de GitHub es para cerrar la pull request.

6. Histórico del Control de Versiones

El proyecto se ha movido a git.

El repositorio fuente de FreeBSD pasó de CVS a Subversion el 31 de Mayo de 2008. El primer commit real de SVN es *r179447*. El repositorio fuente cambió de Subversion a Git el 23 de Diciembre de 2020. El último commit real de svn es *r368820*. El hash del primer commit real en git es *5ef5f51d2bef80b0ede9b10ad5b0e9440b60518c*.

El repositorio `doc/www` de FreeBSD cambió de CVS a Subversion el 19 de Mayo de 2012. El primer commit real de SVN es *r38821*. El repositorio de documentación cambió de Subversion a Git el 8 de Diciembre de 2020. El último commit de SVN es *r54737*. El has del primer commit real de git es *3be01a475855e7511ad755b2defd2e0da5d58bbe*.

El repositorio de `ports` de FreeBSD cambió de CVS a Subversion el 14 de Julio de 2012. El primer commit real de SVN es *r300894*. El repositorio de ports cambió de Subversion a Git el 6 de Abril de 2021. El último commit de SVN es *r569609*. El hash del primer commit de git es *ed8d3eda309dd863fb66e04bccaa513eee255cbf*.

7. Configuración, Convenciones y Tradiciones

Hay una serie de cosas que hacer como nuevo desarrollador. La primera serie de pasos es específica solamente para los committers. Estos pasos deben ser realizados por un mentor para aquellos que no son committers.

7.1. Para los Nuevos Committers

Aquellos a los que se les han concedido derechos de envío a los repositorios de FreeBSD deben seguir estos pasos.

- ¡Obtén aprobación de tu mentor para hacer commit de cada uno de estos cambios!
- Todos los commits de src van primero a FreeBSD-CURRENT antes de llevarse a FreeBSD-STABLE. La rama FreeBSD-STABLE debe mantener la compatibilidad de ABI y API con versiones anteriores de esa rama. No lleves cambios que rompan esta compatibilidad.

Pasos para los Nuevos Committers

1. Añade una Entidad de Autor

   doc/shared/authors.adoc - Añade una entidad de autor. Los pasos posteriores dependen de esta entidad, y saltarse este paso provocará que la construcción de doc/ falle. Esta es una tarea relativamente sencilla, pero sigue siendo una buena primera tarea de prueba de las habilidades de control de versiones.

2. Actualiza la Lista de Desarrolladores y Colaboradores

   doc/shared/contrib-committers.adoc - Añade una entrada, la cual aparecerá en la sección "Developers" de la ection of the Lista de Colaboradores. Las entradas están ordenadas por apellido.

   doc/shared/contrib-additional.adoc - *Elimina* la entrada. Las entradas están ordenadas por nombre.

3. Añade un Ítem a las Noticias

   doc/website/data/en/news/news.toml - Añade una entrada. Busca otras entradas que anuncien nuevos committers y sigue el formato. Usa la fecha del correo de aprobación del commit bit.

4. Añade una Clave PGP

   `Dag-Erling Smørgrav <des@FreeBSD.org>` ha escrito un shell script (doc/documentation/tools/addkey.sh) para hacerlos más fácil. Lee el fichero README para más información.

   Usa doc/documentation/tools/checkkey.sh para verificar que la clave cumple con el mínimo

de las buenas prácticas estándar.

Después de añadir y comprobar la clave, añade ambos ficheros actualizados al control de código y luego haz commit. Las entradas en este fichero están ordenadas por apellido.

> Es muy importante tener una clave PGP/GnuPG actualizada en el repositorio. Se podría requerir la clave para identificar a un committer. Por ejemplo, el `Administradores de FreeBSD <admins@FreeBSD.org>` podría necesitarlo para recuperar una cuenta. Hay un llavero completo de usuarios de `FreeBSD.org` disponible para descarga desde https://docs.FreeBSD.org/pgpkeys/pgpkeys.txt.

5. Actualiza la información del Mentor y el Alumno

   src/share/misc/committers-<repository>.dot - Añade una entrada a la sección de committers actuales, donde *repository* es `doc`, `ports`, o `src`, dependiendo de los privilegios de commit concedidos.

   Añade una entrada para cada relación mentor/alumno individual al final de la sección.

6. Genera una Contraseña de Kerberos

   Lee Kerberos y contraseña web LDAP para el clúster de FreeBSD para generar o establecer una cuenta de Kerberos para utilizarla con otros servicios de FreeBSD como la base de datos de bugs (obtienes una cuenta en la base de datos como parte de ese paso).

7. Opcional: Activa la Cuenta de la Wiki

   FreeBSD Wiki Account - Una cuenta en la wiki permite compartir proyectos e ideas. Aquellos que todavía no tienen una cuenta pueden seguir las instrucciones en Wiki/About page para obtener una. Contacta con wiki-admin@FreeBSD.org si necesitas ayuda con tu cuenta Wiki.

8. Opcional: Actualiza la Información de la Wiki

   Información en la Wiki - Después de obtener acceso a la wiki, algunas personas añaden entradas a las páginas Cómo Hemos Llegado Aquí, Nicks de IRC, Perros de FreeBSD, y o Gatos de FreeBSD.

9. Opcional: Actualiza los Ports con Información Personal

   ports/astro/xearth/files/freebsd.committers.markers y src/usr.bin/calendar/calendars/calendar.freebsd - Algunas personas añaden entradas para ellos mismos a estos ficheros para mostrar dónde viven o su fecha de cumpleaños.

10. Opcional: Evita Correos Duplicados

    Los subscriptores de Mensajes de commit para todas la ramas del repositorio doc, Mensajes de commit para todas las ramas del repositorio de ports o Mensajes de commit para todas las ramas del repositorio src podrían querer darse de baja para evitar recibir copias duplicadas de los mensajes de commit y de sus continuaciones.

7.2. Para Todos

1. Preséntate ante los otros desarrolladores, de otro modo nadie tendrá ni idea de quién eres o en qué trabajas. La presentación no tiene que ser una biografía completa, tan sólo escribe un párrafo o dos acerca de quién eres, en qué piensas trabajar como desarrollador de FreeBSD, y quién será tu mentor. Envía este correo a Lista de correo de desarrolladores de FreeBSD y habrás terminado. Entra en `freefall.FreeBSD.org` y crea un fichero /var/forward/usuario (donde *usuario* es tu nombre de usuario) que contenga la dirección de correo donde quieres que se reenvíen los correos dirigidos a *tunombredeusuario*@FreeBSD.org. Esto incluye todos los mensajes de commit así como cualquier otro correo enviado a Lista de correo para 'committers' de FreeBSD y a Lista de correo de desarrolladores de FreeBSD. Los buzones de correo realmente grandes que están en `freefall` podrían ser truncados sin previo aviso si se necesita liberar espacio, así que reenvíalo o sálvalo en otra parte.

   > Si tu sistema de correo electrónico usa SPF con reglas estrictas, deberías excluir `mx2.FreeBSD.org` de las comprobaciones de SPF.

   Debido a la severa carga que tratar con SPAM produce en los servidores centrales de correo que hacen el procesamiento de las listas de correo, el servidor front-end hace algunas comprobaciones básicas y eliminará algunos mensajes basándose en estas comprobaciones. En este momento sólo se comprueba la que la información de DNS para el host que se conecta es la adecuada, pero esto podría cambiar. Algunas personas culpan a estas comprobaciones de bloquear correo válido. Para deshabilitar estas comprobaciones para tu correo, crea un fichero llamado ~/.spam_lover en `freefall.FreeBSD.org`.

   > Aquellos que sean desarrolladores pero no committers no estarán suscritos a las listas de committers o desarrolladores. Las suscripciones se derivan de los permisos de acceso.

7.2.1. Configuración de acceso SMTP

Para aquellos que deseen enviar mensajes de correo electrónico a través de la infraestructura de FreeBSD.org, sigan las siguientes instrucciones:

1. Apunta tu cliente de correo a `smtp.FreeBSD.org:587`. Activa STATTLS. Asegúrate de que tu dirección `From:` está establecida a `tunombredeusuario@FreeBSD.org`. Para la autenticación puedes usar tu nombre de usuario de Kerberos y tu contraseña (lee Kerberos y contraseña web LDAP para el clúster de FreeBSD). Se prefiere el `tunombredeusuario/mail` principal, ya que sólo se usa para validar recursos de correo

   > No incluyas `FreeBSD.org` cuando introduzcas tu nombre de usuario

2. Notas adicionales

- Sólo se aceptará correo desde `tunombredeusuario@FreeBSD.org`. Si estás autenticado como un usuario, no se te permite enviar correo como otro.
- Se añadirá una cabecera con el nombre de usuario SASL: (`Authenticated sender: username`).
- La máquina tiene varios límites de velocidad para cortar los intentos de fuerza bruta.

Uso de un MTA local para reenviar correos electrónicos al servicio SMTP de FreeBSD.org

También es posible utilizar un MTA local para reenviar emails enviados localmente a los servidores SMTP de FreeBSD.org.

Ejemplo 1. Usando Postfix

Para decirle a una instancia local de Postfix que se debería reenviar a los servidores FreeBSD.org cualquier cosa que venga de `tunombredeusuario@FreeBSD.org`, añade esto a tu main.cf:

```
sender_dependent_relayhost_maps = hash:/usr/local/etc/postfix/relayhost_maps
smtp_sasl_auth_enable = yes smtp_sasl_security_options = noanonymous
smtp_sasl_password_maps = hash:/usr/local/etc/postfix/sasl_passwd smtp_use_tls =
yes
```

Crea /usr/local/etc/postfix/relayhost_maps con el siguiente contenido:

```
tunombredeusuario@FreeBSD.org [smtp.freebsd.org]:587
```

Crea /usr/local/etc/postfix/sasl_passwd con el siguiente contenido:

```
[smtp.freebsd.org]:587          tunombredeusuario:tucontraseña
```

Si otras personas utilizan el servidor de correo electrónico, es posible que quieras evitar que envíen correos electrónicos desde tu dirección. Para lograr esto, agrega esto a tu main.cf:

```
smtpd_sender_login_maps = hash:/usr/local/etc/postfix/sender_login_maps
smtpd_sender_restrictions = reject_known_sender_login_mismatch
```

Crea /usr/local/etc/postfix/sender_login_maps con el siguiente contenido:

```
tunombredeusuario@FreeBSD.org tunombredeusuariolocal
```

Donde *tunombredeusuariolocal* es el nombre de usuario SASL utilizado para conectar a la instancia local de Postfix.

Ejemplo 2. Usando OpenSMTPD

Para decirle a una instancia local de OpenSMTPD que se debería reenviar a los sevidores FreeBSD.org cualquier cosa que venga de `tunombredeusuario@FreeBSD.org`, añade esto a tu smtpd.conf:

```
action "freebsd" relay host smtp+tls://freebsd@smtp.freebsd.org:587 auth <secrets>
match from any auth yourlocalusername mail-from "_yourusername_@freebsd.org" for
any action "freebsd"
```

Donde *tunombredeusuariolocal* es el nombre de usuario SASL utilizado para conectar a la instancia local de OpenSMTPD.

Crea /usr/local/etc/mail/secrets con el siguiente contenido:

```
freebsd tunombredeusuario:tucontraseña
```

Ejemplo 3. Usando Exim

Para decirle a una instancia local de Exim que se debería reenviar a los sevidores FreeBSD.ORG cualquier cosa que venga de `example@FreeBSD.org` añade esto a tu configuration de Exim:

```
Routers section: (at the top of the list):
freebsd_send:
   driver = manualroute
   domains = !+local_domains
   transport = freebsd_smtp
   route_data = ${lookup {${lc:$sender_address}} lsearch
{/usr/local/etc/exim/freebsd_send}}

Transport Section:
freebsd_smtp:
        driver = smtp
  tls_certificate=<local certificate>
  tls_privatekey=<local certificate private key>
  tls_require_ciphers =
EECDH+ECDSA+AESGCM:EECDH+aRSA+AESGCM:EECDH+ECDSA+SHA384:EECDH+ECDSA+SHA256:EECDH+a
RSA+SHA384:EECDH+aRSA+SHA256:EECDH+AESGCM:EECDH:EDH+AESGCM:EDH+aRSA:HIGH:!MEDIUM:!
LOW:!aNULL:!eNULL:!LOW:!RC4:!MD5:!EXP:!PSK:!SRP:!DSS
  dkim_domain = <local DKIM domain>
  dkim_selector = <local DKIM selector>
  dkim_private_key= <local DKIM private key>
```

```
  dnssec_request_domains = *
  hosts_require_auth = smtp.freebsd.org

Authenticators:
fixed_plain:
  driver = plaintext
  public_name = PLAIN
  client_send = ^example/mail^examplePassword
```

Crea /usr/local/etc/exim/freebsd_send con el siguiente contenido:

```
example@freebsd.org:smtp.freebsd.org::587
```

7.3. Mentores

Todos los nuevos desarrolladores tienen un mentor asignado durante los primeros meses. Un mentor es responsable de enseñar a los aprendices las reglas y convenciones del proyecto y de guiar sus primeros pasos en la comunidad de desarrolladores. El mentor también es personalmente responsable de las acciones de los aprendices durante este período inicial.

Para los committers: no envíes nada sin obtener primero la aprobación del mentor. Documenta esa aprobación con una línea `Approved by:` en el mensaje de commit.

Cuando el mentor decide que un aprendiz ha aprendido las reglas y está listo para hacer envíos por su cuenta, el mentor lo anuncia con un commit en mentors. Este archivo está en la rama huérfana admin de cada repositorio. Se puede encontrar información detallada sobre cómo acceder a estas ramas en rama "admin".

8. Revisión previa al commit

La revisión de código es una forma de incrementar la calidad del software. Las siguientes guías aplican a los commits a la rama `main`(-CURREN) del repositorio `src`. Otras ramas y los árboles `ports` y `docs` tienen sus propias políticas, pero estas directrices aplican generalmente a commits que necesitan revisión:

- Todos los cambios no triviales deberían ser revisados antes de hacer commit en el repositorio.
- Las revisiones se pueden realizar por email, en Bugzilla, en Phabricator, o por otro mecanismo. Cuando sea posible, las revisiones deberían ser públicas.
- El desarrollador responsable de un cambio de código también es responsable de hacer todos los cambios relacionados con la revisión.
- La revisión de código puede ser un proceso iterativo, que continúa hasta que el parche está listo para ser comprometido. Específicamente, una vez que se envía un parche para su revisión, debes recibir un "looks good" explícito antes de hacer commit. Siempre que sea explícito, esto puede tomar cualquier forma que tenga sentido para el método de revisión.

- Los timeouts no sustituyen una revisión.

A veces las revisiones de los códigos tardan más de lo que se espera, especialmente para las funciones más grandes. Las formas aceptadas de acelerar los tiempos de revisión de tus parches son:

- Revisa los parches de otras personas. Si tú ayudas, todo el mundo estará más dispuesto a hacer lo mismo por ti; la buena voluntad es nuestra moneda.
- Avisa del parche. Si es urgente, proporciona razones por las que es importante que este parche sea incluido y avisa cada dos días. Si no es urgente, la cortesía habitual es llamar la atención sobre el parche una vez a la semana. Recuerda que estás pidiendo tiempo valioso de otro desarrollador profesional.
- Pide ayuda en las listas de correo, IRC, etc. Otros podrían ser capaces de ayudarte directamente, o de sugerir un revisor.
- Parte tu parche en varios parches más pequeños que se apliquen uno sobre otro. Cuanto más pequeño sea tu parche, más alta será la probabilidad de que alguien le eche un vistazo.

  Cuando hagas cambios grandes, es útil tener en cuenta esto desde el comienzo ya que romper cambios en trozos más pequeños es normalmente difícil al hacerlo más tarde.

Los desarrolladores deben participar en las revisiones de código como revisores y revisados. Si alguien tiene la amabilidad de revisar tu código, deberías devolverle el favor a otra persona. Ten en cuenta que aunque cualquiera es bienvenido a revisar y dar su opinión sobre un parche, sólo un experto en la materia puede aprobar un cambio. Normalmente será un especialista que trabaje con el código en cuestión de forma regular.

En algunos casos, es posible que no se disponga de un experto en la materia. En esos casos, basta con un examen por parte de un desarrollador experimentado cuando se combina con las pruebas apropiadas.

9. Mensajes de Commit

Esta sección contiene algunas sugerencias y tradiciones sobre cómo se formatean los mensajes de commit.

9.1. ¿Por qué son importantes los mensajes de commit?

Cuando haces commit en Git, Subversion, o cualquier otro sistema de control de versiones (VCS), se te pide un texto que describa el cambio — un mensaje de commit. ¿Cómo de importante es este mensaje? ¿Deberías dedicar un esfuerzo significativo escribiéndolo? ¿Realmente importa si escribes simplemente "arregla un bug"?

La mayoría de los proyectos tienen más de un desarrollador y duran un tiempo determinado. Los mensajes de commit son un método muy importante de comunicación con otros desarrolladores, en el presente y para el futuro.

FreeBSD tiene cientos de desarrolladores activos y cientos de miles de commits a lo largo de décadas de historia. Durante ese tiempo la comunidad de desarrolladores ha aprendido cómo de valiosos son los buenos mensajes de commit; a veces se ha tenido que aprender a la fuerza.

Los mensajes de commit sirven al menos tres propósitos:

- Comunicándote con otros desarrolladores

  Los commits en FreeBSD generan emails en varias listas de correo. Estos incluyen el mensaje de commit junto con una copia del propio parche. Los mensajes de commit también se visualizan a través de comandos como git log. Esto sirve para que otros desarrolladores sean conscientes de los cambios que se están produciendo; que otro desarrollador podría querer probar el cambio, podría tener un interés en el asunto en cuestión y querrá revisarlo en más detalle, o que podría tener sus propios proyectos en curso que se beneficiarían de una posible interacción entre ambos.

- Haciendo que los Cambios sean Descubribles

  En un proyecto grande con mucha historia podría ser difícil encontrar cambios de interés cuando se está investigando un problema o un cambio de comportamiento. Los mensajes de commit largos y detallados permiten buscar cambios que podrían ser relevantes. Por ejemplo, `git log --since 1year --grep 'USB timeout'`.

- Proporcionando documentación histórica

  Los mensajes de commit se utilizan para documentar los cambios para los futuros desarrolladores, quizás años o décadas más tardes. Este desarrollador futuro podrías ser tú, el autor original. Un cambio que hoy podría resultar obvio, podría no serlo mucho tiempo después.

El comando `git blame` anota cada línea de un fichero fuente con el cambio (hash y línea de título) que lo incorporó.

Habiendo establecido su importancia, aquí hay algunos ejemplos de buenos mensajes de commit en FreeBSD:

9.2. Comienza con una línea para el título

Los mensajes de commit deberían empezar con una sola línea para el título que resume brevemente el cambio. El título, por sí mismo, debería permitir al lector determinar de forma rápida si el cambio tiene algún interés o no.

9.3. Mantén las líneas de título cortas

La línea de título debería ser lo más corta posible a la vez que mantiene la información requerida. Esto hace que navegar el log de Git sea más eficiente, y también que git log --oneline pueda mostrar el hash corto y el título en una línea de 80 columnas. Una buena regla básica es mantenerse por debajo de 63 caracteres, e intentar hacerlo en 50 o menos si es posible.

9.4. Añade al título un prefijo para el componente si aplica

Si el cambio está relacionado con un componente específico, se puede añadir ala línea del título un prefijo con el nombre del componente y dos puntos (:).

✓ `foo: Add -k option to keep temporary data`

Incluye el prefijo en el límite de 63 caracteres sugerido arriba, de forma que `git log --oneline` evite partir la línea.

9.5. Usa mayúsculas para la primera letra del título

Utiliza mayúscula en la primera letra del título. El prefijo, si lo hay, no utiliza mayúsculas a menos que sea necesario (por ejemplo, `USB:` va en mayúsculas).

9.6. No termines el título con punto

No termines en punto o con cualquier otro signo de puntuación. En este aspecto la línea de título es como el titular de un periódico.

9.7. Separa el título y el cuerpo con una línea en blanco

Separa el cuerpo del título con una línea en blanco.

Algunos commits triviales no necesitan cuerpo y tendrán sólo un título.

✓ `ls: Fix typo in usage text`

9.8. Limita los mensajes a 72 columnas

`git log` y `git format-patch` tabulan el mensaje de commit utilizando cuatro espacios. Cortar en 72 columnas proporciona un margen en el borde derecho. Limitar los mensajes a 72 caracteres también mantiene el mensaje de commit en parches formateados bajo el límite de longitud de línea de email de 78 caracteres fijado en el RFC 2822. Este límite funciona bien con un buen número de herramientas que podrían mostrar mensajes de commit; el cortado de líneas podría ser inconsistente con longitudes de línea más largas.

9.9. Usa el modo presente en imperativo

Esto favorece las líneas de título cortas y proporciona consistencia, incluyendo la generación automática de mensajes de commit (ejemplo, como los generados por git revert). Esto es importante cuando se lee una lista de títulos de commit. Piensa en los títulos como las partes finales de la frase "cuando se aplica, este cambio...".

✓ `foo: Implement the -k (keep) option`
 `foo: Implemented the -k option`
 `This change implements the -k option in foo`
 `-k option added`

9.10. Céntrate en el qué y el por qué, no en el cómo

Explica qué hace el cambio y por qué se ha hecho, en lugar de cómo lo hace.

No asumas que el lector está familiarizado con el asunto. Explica los antecedentes y la motivación para el cambio. Incluye datos de pruebas si los tienes.

Si hay limitaciones o aspectos incompletos del cambio, descríbelos en el mensaje de commit.

9.11. Considera si hay partes del mensaje de commit que podrían ser en realidad comentarios de código

A veces mientras escribes un mensaje de commit puedes ver que estás escribiendo un par de frases explicando algún aspecto confuso del cambio. Cuando esto suceda considera si sería valioso tener esa explicación también en el código en forma de comentario.

9.12. Escribe mensajes de commit para tu yo del futuro

Mientras escribes un mensaje de commit para un cambio tienes todo el contexto en la cabeza - qué motivó el cambio, aproximaciones alternativas que se consideraron y fueron rechazadas, limitaciones del cambio y demás. Imagínate a ti mismo revisitando el cambio en uno o dos años y escribe el mensaje de commit de forma que proporcione el contexto necesario.

9.13. Los mensajes de commit deberían ser autocontenidos

Puedes incluir referencias a mensajes de la lista de correo, resultados de pruebas en sitios web, o enlaces a revisiones de código. Sin embargo, los mensajes de código deberían contener toda la información relevante en caso de que estas referencias ya no estén disponibles en el futuro.

De forma similar, un commit podría referenciar un commit anterior, por ejemplo en el caso de un arreglo y una marcha atrás. Además del identificador del commit (revisión o hash), incluye la línea de título del commit referenciado (u otra referencia breve que sirva). Con cada migración de VCS (de CVS a Subversion a Git) los identificadores de revisión de los sistemas previos podrían ser difíciles de seguir.

9.14. Incluye los metadatos apropiados al pie

Además de incluir un mensaje informativo con cada envío, es posible que se necesite información adicional.

Esta información consta de una o más líneas que contienen la palabra o frase clave, dos puntos, pestañas para formatear y, a continuación, la información adicional.

Las palabras o frases clave son:

`PR:`	El informe de error (si lo hay) que se ve afectado (típicamente, cerrándolo) por este commit. Se pueden especificar varios PRs en una línea, separados por comas o espacios.
`Reported by:`	El nombre y dirección de correo de la persona que reportó el problema: para desarrolladores sólo el nombre de usuario en el clúster de FreeBSD. Típicamente utilizando cuando no hay PR, por ejemplo si el problema fue reportado en una lista de correo.
`Submitted by:` (deprecated)	Esto es obsoleto con git; los parches enviados deberían tener el autor establecido usando `git commit --author` con un nombre completo y una dirección de email válida.

`Reviewed by:`	El nombre y dirección de correo de la persona o personas que revisaron el cambio; para los desarrolladores tan solo el nombre de usuario en el clúster de FreeBSD. Si se envió un parche a la lista de correo para ser revisado y la revisión fue favorable, entonces simplemente incluye el nombre de la lista. Si el revisor no es un miembro del proyecto, proporciona el nombre, email y si es el caso de ports un rol externo como el de mantenedor:
	
	Revisado por un desarrollador:
	
	`Reviewed by: username`
	
	Revisado por un mantenedor de ports que no es un desarrollador:
	
	`Reviewed by: Full Name <valid@email> (maintainer)`

`Tested by:`	El nombre y dirección de correo de la persona o personas que probaron el cambio; para desarrolladores, sólo el nombre de usuario en el clúster de FreeBSD.

`Approved by:`	El nombre y la dirección de correo de la persona o personas que aprobaron el cambio; para desarrolladores el nombre de usuario en el clúster de FreeBSD.
	
	Hay varios casos donde se suele necesitar aprobación:
	
	• cuando un committer todavía está bajo tutorización
	• commits en un are del árbol cubierto bajo el fichero LOCKS (srv)
	• durante el ciclo de liberación
	• hacer commit a un repo en el que no tienes commit bit (por ejemplo un committer de src haciendo commit en docs)
	• hacer commit a un port que mantenga otra persona
	
	Mientras estés aprendiendo, obtén aprobación de tu mentor antes de hacer commit. Introduce el nombre de usuario del mentor en este cambio y haz referencia a que es un mentor:
	
	`Approved by: username-of-mentor (mentor)`
	
	Si los commits los aprueba un grupo incluye el nombre del grupo seguido del nombre de usuario entre paréntesis de la persona que aprobó. Por ejemplo:
	
	`Approved by: re (username)`

`Obtained from:`	El nombre el proyecto (si aplica) del que se obtuvo el código. No uses esta línea para el nombre de una persona individual.
`Fixes:`	El hash corto de Git y la línea de título del commit que se arregla con este cambio tal y como lo devuelve `git log -n 1 --oneline GIT-COMMIT-HASH`.
`MFC after:`	Para recibir un correo con un recordatorio para hacer MFC posteriormente, especifica el número de días, semanas o meses después de los cuales se planea hacer el MFC.
`MFC to:`	Si el commit se debe mergear a un subconjunto de ramas estables, especifica los nombres de las ramas.
`MFH:`	Si el commit se debe mergear a una rama trimestral de ports, especifica la rama trimestral. Por ejemplo `2021Q2`.
`Relnotes:`	Si el cambio es candidato para inclusión en las notas de la versión para la siguiente versión de la rama, establece el campo a `yes`.
`Security:`	Si el cambio está relacionado con una vulnerabilidad de seguridad o riesgo de seguridad, incluye una o más referencias o una descripción del problema. Si es posible incluye una URL de VuXML o un ID de CVE.

`Event:`	La descripción del evento donde se hizo este commit. Si es un evento recurrente, añade el año o incluso el mes. Por ejemplo, podría ser `FooBSDcon 2019`. La idea de esta línea es darle reconocimiento a las conferencias, reuniones y otro tipo de encuentros y mostrar que son útiles. Por favor no utilices la línea `Sponsored by:` para esto ya que se utiliza para organizaciones que son patrocinadores de ciertas características o de desarrolladores que trabajan en ellas.
`Sponsored by:`	Organizaciones que patrocinan este cambio, si aplica. Separa varias organizaciones con comas. Si sólo se patrocinó una parte del trabajo, o distintos autores patrocinaron a distintos niveles, por favor, da el crédito apropiado entre paréntesis después de cada nombre de los patrocinadores. Por ejemplo, `Example.com (alice, refactorización de código), Wormulon (bob), Momcorp (cindy)` muestra que Alice fue patrocinada por Example.com para hacer refactorización de código, mientras que Wormulon patrocinó el trabajo de Bob y Momcorp patrocinó el trabajo de Cindy. Otros autores o no fueron patrocinados o escogieron no listar dicho patrocinio.
`Pull Request:`	Este cambio fue enviado como una pull request o merge request contra uno de los repositorios Git de sólo lectura de FreeBSD. Debería incluir la URL completa de la pull request, ya que normalmente sirve para hacer la revisión del código. Por ejemplo: `https://github.com/freebsd/freebsd-src/pull/745`
`Co-authored-by:`	The name and email address of an additional author of the commit. GitHub has a detailed description of the Co-authored-by trailer at https://docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/creating-a-commit-with-multiple-authors.
`Signed-off-by:`	El ID certifica que cumple con https://developercertificate.org/
`Differential Revision:`	La URL completa de la revisión de Phabricator. Esta línea *debe ser la última línea*. Por ejemplo: `https://reviews.freebsd.org/D1708`.

Ejemplo 4. Registro de compromiso para un compromiso basado en un PR

El commit se basa en un parche en un PR enviado por John Smith. El cambio "PR" del mensaje de commit está relleno.

```
...

PR:     12345
```

El committer establece el autor del parche con `git commit --author "John Smith <John.Smith@example.com>"`.

Ejemplo 5. Confirmar registro para una confirmación que necesita revisión

Se está cambiando el sistema de memoria virtual. Después de publicar los parches en la lista de correo correspondiente (en este caso, `freebsd-arch`) y los cambios han sido aprobados.

```
...

Reviewed by:    -arch
```

Ejemplo 6. Registro de compromiso para un compromiso que necesita aprobación

HAcer un commit de un port, después de trabajar con el MAINTAINER, quien dio el visto bueno para hacer el commit.

```
...

Approved by:    abc (maintainer)
```

Donde *abc* es el nombre de la cuenta de la persona que lo aprobó.

Ejemplo 7. Commit Log para una confirmación que trae código desde OpenBSD

Hacer commit de código basado en el trabajo realizado en el proyecto OpenBSD.

```
...

Obtained from:  OpenBSD
```

Ejemplo 8. Commit Log para un cambio en FreeBSD-CURRENT con un compromiso planificado en FreeBSD-STABLE para seguir en una fecha posterior.

Haciendo commit de un código que se fusionará de FreeBSD-CURRENT en la rama FreeBSD-STABLE después de dos semanas.

```
...

MFC after:  2 weeks
```

Donde *2* es el número de días, semanas, o meses después de los cuales se planea hacer un MFC. La opción *weeks* podría ser `day`, `days`, `week`, `weeks`, `month`, `months`.

A menudo es necesario combinarlos.

Considera la situación en la que un usuario ha enviado un PR que contiene código del proyecto NetBSD. Mirando el PR, el desarrollador ve que no es un área del árbol en la que trabaja habitualmente de forma que se solicita que el cambio sea revisado por la lista de correo `arch`. Como el cambio es complejo, el desarrollador opta por hacer MFC después de un mes para permitir que se

hagan las pruebas adecuadas.

La información adicional para incluir en la confirmación sería algo así como

Ejemplo 9. Ejemplo de Registro de Commit Combinado

```
PR:     54321
Reviewed by:    -arch
Obtained from:  NetBSD
MFC after:  1 month
Relnotes:   yes
```

10. Licencia preferida para los nuevos archivos

La política completa de licencias del Proyecto FreeBSD se puede encontrar en https://www.FreeBSD.org/internal/software-license. El resto de esta sección está pensada para ponerte en funcionamiento. Como regla, cuando tengas dudas, pregunta. Es mucho más fácil dar consejo que arreglar el árbol de fuentes.

El Proyecto FreeBSD sugiere y usa este texto como el esquema de licencia preferido:

```
/*-
 * SPDX-License-Identifier: BSD-2-Clause
 *
 * Copyright (c) [year] [your name]
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
```

```
 * SUCH DAMAGE.
 *
 * [id for your version control system, if any]
 */
```

El proyecto FreeBSD desaconseja rotundamente la denominada "cláusula publicitaria" en el nuevo código. Debido a la gran cantidad de colaboradores al proyecto FreeBSD, cumplir con esta cláusula para muchos proveedores comerciales se ha vuelto difícil. Si tienes código en el árbol con la cláusula publicitaria, considera eliminarlo. De hecho, considera usar la licencia anterior para tu código.

El proyecto FreeBSD desaconseja licencias completamente nuevas y variaciones de las licencias estándar. Las nuevas licencias requieren la aprobación del core@FreeBSD.org para que se añadan al repositorio `src`. Cuantas más licencias diferentes se utilicen en el árbol, más problemas ocasionará a quienes deseen utilizar este código, por lo general debido a las consecuencias no deseadas de una licencia mal redactada.

La política del proyecto dicta que el código de algunas licencias que no sean BSD debe colocarse solo en secciones específicas del repositorio y, en algunos casos, la compilación debe ser condicional o incluso deshabilitada de forma predeterminada. Por ejemplo, el núcleo GENERIC debe compilarse únicamente bajo licencias idénticas o sustancialmente similares a la licencia BSD. El software con licencia GPL, APSL, CDDL, etc., no debe compilarse en GENERIC.

Se recuerda a los desarrolladores que en el código abierto, conseguir "abrir" correctamente es tan importante como conseguir una "fuente" correcta, ya que el manejo inadecuado de la propiedad intelectual tiene graves consecuencias. Cualquier pregunta o inquietud debe comunicarse inmediatamente al Core Team.

11. Seguimiento de las licencias concedidas al proyecto FreeBSD

Existen varias piezas de software y datos en los repositorios para los cuales se ha concedido al proyecto FreeBSD una licencia especial de uso. Un caso de ejemplo es la fuente Terminus para utilizar con vt(4). Aquí el autor Dimitar Zhekov nos ha permitido utilizar la fuente "Terminus BSD Console" bajo una licencia BSD de dos cláusulas en lugar de las licencia regular Open Font License que utiliza é normalmente.

Conviene claramente mantener un registro de dichas concesiones de licencias. Para tal fin, core@FreeBSD.org ha decidido mantener un archivo de ellas. Cuando se le otorga al proyecto FreeBSD una licencia especial, obligamos a que se notifique a core@FreeBSD.org. A cualquier desarrollador involucrado en acordar dichas concesiones de licencia, por favor, envía los detalles a core@FreeBSD.org incluyendo:

- Datos de contacto de personas u organizaciones que otorgan la licencia especial.
- Qué archivos, directorios, etc. de los repositorios están cubiertos por la concesión de licencia, incluidos los números de revisión donde se comprometió cualquier material con licencia especial.

- La fecha en que la licencia entra en vigor. A menos que se acuerde lo contrario, esta será la fecha en que la licencia fue emitida por los autores del software en cuestión.
- El texto de la licencia.
- Una nota de cualquier restricción, limitación o excepción que se aplique específicamente al uso de FreeBSD del material licenciado.
- Cualquier otra información relevante.

Una vez que core@FreeBSD.org está satisfecho con todos los detalles necesarios que se han recopilado y que estos son correctos, el secretario enviará un acuse de recibo firmado con PGP que incluye los detalles de la licencia. Este recibo se almacenará de forma persistente y servirá como registro permanente de la concesión de la licencia.

El archivo de licencias sólo debería contener detalles de las concesiones de licencias; no es lugar para discusiones acerca de licencias en sí u otros asuntos. El acceso a los datos del fichero de licencias estará disponible bajo petición al core@FreeBSD.org.

12. Etiquetas SPDX en el árbol

El proyecto utiliza etiquetas SPDX en nuestra base de fuentes. En este momento, las etiquetas están indentadas para ayudar a las herramientas automáticas a reconstruir los requisitos de las licencias mecánicamente. Todas las etiquetas *SPDX-License-Identifier* en el árbol deberían considerarse informativas. Todos los ficheros en el árbol de fuentes de FreeBSD con estas etiquetas también tienen una copia de la licencia de gobierna el uso de dicho fichero. En el caso de alguna discrepancia, la licencia literal es la que domina. El proyecto intenta seguir la SPDX Specification, Version 2.2. Se puede ver cómo crear ficheros fuente y expresiones algebraicas válidas en Appendix IV y Appendix V. El proyecto extrae identificadores de la lista de identificadores cortos de licencias de SPDX. El proyecto sólo utiliza la etiqueta *SPDX-License-Identifier*.

A fecha de Marzo de 2021, se han marcado aproximadamente 25,000 de los 90,000 ficheros en el árbol.

13. Relaciones con los desarrolladores

Cuando trabajes directamente en tu propio código o en un código que ya está bien establecido como tu responsabilidad, entonces probablemente haya poca necesidad de verificar con otros committers antes de hacer un commit. Cuandoo trabajes en un arreglo para un error en un área del sistema que está claramente huérfana (y hay algunas áreas de este tipo, para nuestra vergüenza), se aplica lo mismo. Cuando modifiques partes del sistema que se mantienen, formal o informalmente, considera solicitar una revisión tal como lo haría un desarrollador antes de convertirse en un committer. Para ports, contacta con el `MAINTAINER` que aparece listado en el Makefile.

Para determinar si se mantiene un área del árbol, consulta el archivo MAINTAINERS en la raíz del árbol. Si no aparece nadie, escanea el historial de revisiones para ver quién ha realizado cambios en el pasado. Para listar los nombres y direcciones de correo de todos los autores de commits de un fichero concreto en los dos últimos años y el número de commits de cada autor, ordenado por número descendente de commits, usa:

```
% git -C /path/to/repo shortlog -sne --since="2 years" -- relative/path/to/file
```

Si las consultas quedan sin respuesta o el committer de otro modo indica una falta de interés en el área afectada, continúa adelante y realiza el commit.

Evita enviar correos electrónicos privados a los mantenedores. Otras personas podrían estar interesadas en la conversación, no sólo en el resultado final.

Si hay alguna duda sobre un commit por cualquier motivo, hazlo revisar antes de realizar el commit. Es mejor que reciba críticas en ese mismo momento que cuando es parte del repositorio. Si un commit da lugar a que surja una controversia, puede ser aconsejable considerar deshacer el cambio hasta que se resuelva el asunto. Recuerda, con un sistema de control de versiones siempre podemos volver a cambiarlo.

No impugnes las intenciones de los demás. Si ven una solución diferente a un problema, o incluso un problema diferente, probablemente no sea porque sean estúpidos, porque tienen una paternidad cuestionable o porque están tratando de destruir el trabajo duro, la imagen personal o FreeBSD, sino básicamente porque tienen una perspectiva diferente del mundo. Diferente es bueno.

Discrepa de forma honesta. Argumenta tu posición desde sus méritos, sé honesto acerca de cualquier deficiencia que puedas tener y mantente abierto a ver su solución, o incluso su visión del problema, con una mente abierta.

Acepta la corrección. Todos cometemos errores. Cuando hayas cometido un error, discúlpate y sigue con tu vida. No te castigues a ti mismo, y ciertamente no castigues a otros por tu error. No pierdas el tiempo en vergüenza o recriminación, simplemente soluciona el problema y sigue adelante.

Pide ayuda. Busca (y proporciona) revisiones de pares. Una de las formas en que se supone que el software de código abierto sobresale es en la cantidad de ojos que se le aplican; esto no se aplica si nadie revisa el código.

14. Si tienes dudas ...

Cuando no estés seguro de algo, ya sea un problema técnico o una convención del proyecto, asegúrate de preguntar. Si te quedas en silencio, nunca progresarás.

Si se relaciona con un problema técnico, pregunta en las listas de correo públicas. Evita la tentación de enviar un correo electrónico a la persona que conoce la respuesta. De esta manera, todos podrán aprender de la pregunta y la respuesta.

Para preguntas administrativas o específicas del proyecto, pregunta, en orden:

- Tu mentor o ex mentor.
- Un cometer experimentado en IRC, correo electrónico, etc.
- Cualquier equipo con "sombrero", ya que pueden darte una respuesta definitiva.

- Si aún así no estás seguro, pregunta en Lista de correo de desarrolladores de FreeBSD.

Una vez que se responda tu pregunta, si nadie te indicó la documentación que detalla la respuesta a tu pregunta, documenta, ya que otros tendrán la misma pregunta.

15. Bugzilla

El proyecto FreeBSD utiliza Bugzilla para rastrear errores y solicitudes de cambio. Si haces un commit de un arreglo o una sugerencia que está en la base de datos de PR asegúrate de cerrarlo. También se considera bueno si te tomas tiempo para cerrar cualquier PR asociado con tus commits, si corresponde.

Committers sin una cuenta `FreeBSD.org` en Bugzilla pueden fusionar la antigua cuenta con su cuenta `FreeBSD.org` siguiendo los siguientes pasos:

1. Inicie sesión con su cuenta anterior.
2. Abre un nuevo bug. Escoge `Services` como Product y `Bug Tracker` como Component. En la descripción del bug lista las cuentas que quieres fusionar.
3. Haz login utilizando la cuenta `FreeBSD.org` y haz un comentario en el bug recién abierto para confirmar la propiedad. Visita Kerberos y contraseña web LDAP para el clúster de FreeBSD para más detalles sobre cómo generar o establecer una contraseña para tu cuenta `FreeBSD.org`.
4. Si hay más de dos cuentas para fusionar, publique comentarios de cada una de ellas.

Puedes encontrar más acerca de Bugzilla en:

- FreeBSD Problem Report Handling Guidelines
- https://www.FreeBSD.org/support

16. Phabricator

El Proyecto FreeBSD utiliza Phabricator para las solicitudes de revisión de código. Visita la página de la wiki de Phabricator para más detalles.

Committers sin una cuenta `FreeBSD.org` en Phabricator pueden fusionar la antigua cuenta con su cuenta `FreeBSD.org` siguiendo los siguientes pasos:

1. Cambia tu cuenta de correo de Phabricator a tu dirección `FreeBSD.org`.
2. Abre un nuevo informe de error en nuestra base de datos usando tu cuenta `FreeBSD.org`, visita Bugzilla para más información. Escoge `Services` como Product y `Code Review` como Component. En la descripción del bug solicita que se renombre tu cuenta de Phabricator y proporciona un enlace a tu usuario de Phabricator. Por ejemplo, `https://reviews.freebsd.org/p/bob_example.com/`

Las cuentas de Phabricator no se pueden fusionar, por favor no abras una cuenta nueva.

17. Quien es Quien

Además de los meisters del repositorio, hay otros miembros y equipos del proyecto FreeBSD a los que probablemente conocerá en su rol de committer. Brevemente, y de ninguna manera todo incluido, estos son:

`Grupo de Ingeniería de Documentación <doceng@FreeBSD.org>`

doceng es el grupo responsable de la infraestructura de construcción de documentación, aprobar nuevos committers de documentación, y asegurar que el sitio web de FreeBSD y la documentación en el sitio FTP están actualizados respecto del árbol de Subversion. No es un órgano de resolución de conflictos. La mayoría de las discusiones relacionadas con documentación tienen lugar en Lista de correo del proyecto de documentación de FreeBSD. Se pueden encontrar más detalles acerca del equipo de doceng en su charter. Los committers interesados en contribuir a la documentación se deberían familiarizar con el Documentation Project Primer.

`Sergio Carlavilla Delgado <carlavilla@FreeBSD.org>, Dave Cottlehuber <dch@FreeBSD.org>, Marc Fonvieille <blackend@FreeBSD.org>, Craig Leres <leres@FreeBSD.org>, Xin Li <delphij@FreeBSD.org>, Ed Maste <emaste@FreeBSD.org>, Colin Percival <cperciva@FreeBSD.org>, Muhammad Moinur Rahman <bofh@FreeBSD.org>, Vladlen Popolitov <vladlen@FreeBSD.org>, Lexi Winter <ivy@FreeBSD.org>, Alexander Ziaee <ziaee@FreeBSD.org>`

Estos son los miembros del equipo de ingeniería de versiones `Grupo de Ingeniería de Releases <re@FreeBSD.org>`. Este equipo es responsable de establecer los plazos de publicación y controlar el proceso de publicación. Durante la congelación del código, los ingenieros de versiones tienen la autoridad final sobre todos los cambios en el sistema para cualquier rama que tenga el estado de versión pendiente. Si hay algo que quieras incluir de FreeBSD-CURRENT a FreeBSD-STABLE (independientemente de los valores que puedan tener en un momento dado), estas son las personas con las que hablar al respecto.

`Gordon Tetlow <gordon@FreeBSD.org>`

`Gordon Tetlow` es el FreeBSD Security Officer y supervisa el `Grupo Responsables de Seguridad <security-officer@FreeBSD.org>`.

Lista de correo para 'committers' de FreeBSD

{dev-src-all}, {dev-ports-all} y {dev-doc-all} son las listas de correo que utiliza el sistema de control de versiones para mandar mensajes de commit. *Nunca* envíes correo directamente a esas listas. Envía sólo respuestas a esta lista cuando son cortas y directamente relacionadas con un commit.

Lista de correo de desarrolladores de FreeBSD

Todos los committers están suscritos a -developers. Esta lista se creó como un foro para los asuntos relacionados con la "communidad" de committers. Ejemplos son las votaciones de Core, anuncios, etc.

La Lista de correo de desarrolladores de FreeBSD es de uso exclusivo de los committers de FreeBSD. Para desarrollar FreeBSD, los committers deben tener la capacidad de discutir asuntos abiertamente que se resolverán antes de que sean anunciados públicamente. Discusiones con franqueza sobre el trabajo en curso no son aptas para la publicación abierta y podrían dañar a FreeBSD.

Se espera que todos los committers de FreeBSD no publiquen ni reenvíen mensajes de la lista de correo de desarrolladores de FreeBSD fuera de la membresía de la lista sin el permiso de todos los autores. Los infractores serán eliminados de la lista de correo de desarrolladores de FreeBSD, lo que resultará en la suspensión de los privilegios de commit. Las violaciones repetidas o flagrantes pueden resultar en la revocación permanente de los privilegios de commit.

Esta lista *no* está pensada como un sito para hacer revisiones de código o para otras cuestiones técnicas. De hecho utilizarla para eso daña el Proyecto FreeBSD ya que le da un aire de lista cerrada donde las decisiones que afectan a toda la comunidad que usa FreeBSD no se hacen de forma "abierta". Por último, pero no menos importante, nunca, nunca, nunca, mandes un correo a {developers-mail} y pongas en CC:/BCC: a otra lista de FreeBSD. Nunca, nunca envíes correo a otra lista de correo de FreeBSD con CC:/BCC: a la Lista de correo de desarrolladores de FreeBSD. Hacerlo puede disminuir los beneficios de esta lista.

18. Guía de inicio rápido de SSH

1. Si no quieres escribir tu contraseña cada vez que uses ssh(1), y utilizas claves para autenticar, ssh-agent(1) está aquí para ayudarte. Si quieres usar ssh-agent(1), asegúrate de ejecutarlo antes que otras aplicaciones. Los usuarios de X, por ejemplo, normalmente hacen esto en su .xsession o .xinitrc. Lee ssh-agent(1) para más detalles.
2. Genera un par de claves con ssh-keygen(1). El clave de pares terminará en tu directorio $HOME/.ssh/.

   Sólo se soportan claves ECDSA, Ed25519 o RSA.

3. Envía tu clave pública ($HOME/.ssh/id_ecdsa.pub, $HOME/.ssh/id_ed25519.pub, o $HOME/.ssh/id_rsa.pub) a la persona que te está dando de alta como committer de forma que la pueda poner en yourlogin en /etc/ssh-keys/ en `freefall`.

Ahora se puede usar ssh-add(1) para autenticarse una vez por sesión. Solicita la frase de paso de la clave privada y después la almacena en el agente de autenticación (ssh-agent(1)). Utiliza `ssh-add -d` para eliminar las claves almacenadas en el agente.

Pruébalo con un comando remoto sencillo: `ssh freefall.FreeBSD.org ls /usr`.

Para más información, lee security/openssh-portable, ssh(1), ssh-add(1), ssh-agent(1), ssh-keygen(1), y scp(1).

Para información sobre cómo añadir, cambiar o eliminar claves ssh(1), lee este artículo.

19. Disponibilidad de Coverity® para los Committers de FreeBSD

Todos los desarrolladores de FreeBSD pueden obtener acceso a los resultados de análisis de Coverity para todo el software del Proyecto FreeBSD. Todo aquel que esté interesado en el acceso a los resultados de análisis de las ejecuciones automáticas de Coverity, pueden registrarse en Coverity Scan.

La wiki de FreeBSD incluye una mini-guía para desarrolladores interesados en trabajar con los informes de análisis de Coverity®: https://wiki.freebsd.org/CoverityPrevent. Por favor, ten en cuenta que esta mini-guía sólo es accesible para los desarrolladores de FreeBSD, así que si no puedes acceder a esta página, tendrás que pedirle a alguien que te añada a la lista de acceso apropiada de la Wiki.

Por último, a todos los desarrolladores de FreeBSD que vayan a usar Coverity® se les anima a preguntar por más detalles e información de uso, mediante el envío de preguntas a la lista de correo de desarrolladores de FreeBSD.

20. La gran lista de reglas de los Committers de FreeBSD

Todo aquel involucrado en el proyecto FreeBSD debe seguir el *Código de Conducta* disponible en https://www.FreeBSD.org/internal/code-of-conduct. Como committer, tú eres la cara visible del proyecto y cómo te comportas tiene un impacto vital en la percepción pública del mismo. Esta guía expande las partes del *Código de Conducta* específicas para committers.

1. Respeta a los demás committers.
2. Respeta a otros colaboradores.
3. Discute cualquier cambio significativo *antes* de hacer commit.
4. Respeta los mantenedores que existan (si están listados en el campo `MAINTAINER` en Makefile o en MAINTAINER en el directorio raíz).
5. Cualquier cambio en disputa debe ser anulado en espera de la resolución de la disputa si lo solicita un mantenedor. Los cambios relacionados con la seguridad pueden anular los deseos del mantenedor a discreción del oficial de seguridad.
6. Los cambios van a FreeBSD-CURRENT antes de FreeBSD-STABLE a menos que el ingeniero de versiones lo permita específicamente o que no sean aplicables a FreeBSD-CURRENT. Cualquier cambio no trivial o no urgente que sea aplicable también debe permitirse que permanezca en FreeBSD-CURRENT durante al menos 3 días antes de fusionarse para que se puedan realizar las pruebas suficientes. El ingeniero de versiones tiene la misma autoridad sobre la rama FreeBSD-STABLE que se describe para el mantenedor en la regla # 5.
7. No luches en público con otros committers; se ve mal.
8. Respeta la congelación de código y lee las listas de correo de `committers` y `developers` regularmente de forma que sepas que hay una congelación de código en marcha.

9. En caso de duda sobre cualquier procedimiento, ¡pregunta primero!
10. Prueba tus cambios antes de realizarlos.
11. No hagas commit en software contribuido sin aprobación *explícita* de los respectivos mantenedores.

Como se señaló anteriormente, romper algunas de estas reglas puede ser motivo de suspensión o, en caso de reincidencia, eliminación permanente de los privilegios de committer. Los miembros individuales de core tienen el poder de suspender temporalmente los privilegios de commit hasta que core en su conjunto tenga la oportunidad de revisar el problema. En caso de "emergencia" (un committer que daña el repositorio), los meisters del repositorio también pueden realizar una suspensión temporal. Solo una mayoría de 2/3 de core tiene la autoridad para suspender los privilegios de commit durante más de una semana o para eliminarlos permanentemente. Esta regla no existe para que core se convierta en un grupo de dictadores crueles que pueden deshacerse de los responsables de manera tan casual como las latas de refresco vacías, sino para darle al proyecto una especie de mecanismo de seguridad. Si alguien está fuera de control, es importante poder lidiar con esto de inmediato en lugar de quedar paralizado por el debate. En todos los casos, un comitter cuyos privilegios se suspenden o revocan tiene derecho a una "vista" ante core, determinándose en ese momento la duración total de la suspensión. Un committer cuyos privilegios estén suspendidos también puede solicitar una revisión de la decisión después de 30 días y cada 30 días a partir de entonces (a menos que el período total de suspensión sea inferior a 30 días). Un committer cuyos privilegios hayan sido revocados por completo puede solicitar una revisión después de que haya transcurrido un período de 6 meses. Esta política de revisión es "estrictamente informal" y, en todos los casos, core se reserva el derecho de actuar o ignorar las solicitudes de revisión si sienten que su decisión original es la correcta.

En todos los demás aspectos de la operación del proyecto, core es un subconjunto de committers y está vinculado por las *mismas reglas*. El hecho de que alguien esté en core no significa que tenga una dispensación especial para salir de cualquiera de las líneas pintadas aquí; los "poderes especiales" de core solo se activan cuando actúa como grupo, no de forma individual. Como individuos, los miembros del equipo central son todos committers primero y miembros de core en segundo lugar.

20.1. Detalles

1. Respeta a los demás committers.

   Esto significa que debes tratar a los demás committers como los desarrolladores de grupos de iguales que son. A pesar de nuestros ocasionales intentos de demostrar lo contrario, uno no llega a committer siendo estúpido y nada irrita más que ser tratado de esa manera por uno de sus compañeros. Si siempre sentimos respeto por los demás o no (y todos tienen días libres), todavía tenemos que *tratar* a otros committers con respeto en todo momento, en foros públicos y en correos privados.

   Poder trabajar juntos a largo plazo es el mayor activo de este proyecto, uno mucho más importante que cualquier conjunto de cambios en el código, y convertir los argumentos sobre el código en problemas que afectan nuestra capacidad a largo plazo para trabajar juntos en armonía simplemente no vale la pena. -abandonado por cualquier tramo concebible de la

imaginación.

Para cumplir con esta regla, no envíes correos electrónicos cuando estés enfadado o te comportes de una manera que pueda parecer a los demás como una confrontación innecesaria. Primero cálmate, luego piensa en cómo comunicarte de la manera más efectiva para convencer a las otras personas de que tu lado del argumento es correcto, no te desahogues un poco para sentirte mejor en el corto plazo a costa de una guerra de llamas a largo plazo. No solo esto es una mala "economía energética", sino que las demostraciones repetidas de agresión pública que perjudican nuestra capacidad para trabajar bien juntos serán tratadas severamente por el liderazgo del proyecto y pueden resultar en la suspensión o terminación de tus privilegios de commit. El liderazgo del proyecto tendrá en cuenta tanto las comunicaciones públicas como las privadas que se le presenten. No buscará la divulgación de comunicaciones privadas, pero la tendrá en cuenta si es voluntaria por parte de los autores de la denuncia.

Todo esto nunca es una opción de la que disfrute en lo más mínimo el liderazgo del proyecto, pero la unidad es lo primero. Ninguna cantidad de código o buen consejo se puede cambiar por esta unidad.

2. Respeta a otros colaboradores.

   No siempre fuiste un committer. Hubo un tiempo en que contribuiste. Recuerda eso en todo momento. Recuerda lo que fue tratar de obtener ayuda y atención. No olvides que tu trabajo como colaborador fue muy importante para ti. Recuerda cómo fue. No desanimes, menosprecies o hagas de menos a los voluntarios. Trátalos con respeto. Son nuestros committers en espera. Son tan importantes para el proyecto como los committers. Sus contribuciones son tan válidas e importantes como las tuyas. Después de todo, hiciste muchas contribuciones antes de convertirse en committer. Recuerda eso siempre.

   Considera los puntos mencionados en Respeta a otros committers y aplícalos también a los voluntarios.

3. Discute cualquier cambio significativo *antes* de hacer commit.

   El repositorio no es donde los cambios se envían inicialmente para su corrección o para ser discutidos, eso ocurre primero en las listas de correo o mediante el uso del servicio Phabricator. El commit solo ocurrirá una vez que se haya alcanzado algo parecido al consenso. Esto no significa que se requiera permiso antes de corregir todos los errores de sintaxis obvios o errores ortográficos de la página del manual, solo que es bueno desarrollar una idea de cuándo un cambio propuesto no es tan obvio y requiere algunos comentarios primero. A la gente realmente no le importan los cambios radicales si el resultado es claramente mejor que lo que tenían antes, simplemente no les gusta ser *sorprendidos* por esos cambios. La mejor manera de asegurarse de que todo va por buen camino es hacer que el código sea revisado por uno o más committers.

   En caso de duda, ¡solicite una revisión!

4. Respeta a los mantenedores existentes si están listados como tales.

   Muchas partes de FreeBSD no tienen "dueño" en el sentido de que cualquier individuo saltará sobre ti y te gritará si haces un cambio en "su" área, pero aún así merece la pena comprobarlo

primero. Una convención que usamos es poner una linea "maintainer" en el Makefile para cualquier paquete o subárbol que es mantenido de forma activa por una o más personas; visita see Directrices y Políticas del Árbol de Fuentes para obtener documentación sobre esto. Donde hay secciones de código con varios mantenedores, los commits efectuados por un mantenedor en dicha área deben ser revisados por al menos otro mantenedor. En los casos donde el mantenimiento de algo no está claro, mira los logs del repositorio para los ficheros en cuestión y mira si alguien ha estado trabajando recientemente o de forma predominante en esa área.

5. Cualquier cambio en disputa debe ser anulado en espera de la resolución de la disputa si lo solicita un mantenedor. Los cambios relacionados con la seguridad pueden anular los deseos del mantenedor a discreción del oficial de seguridad.

   Esto puede ser difícil de aceptar en tiempos de conflicto (cuando cada parte está convencida de que tienen razón, por supuesto), pero un sistema de control de versiones hace innecesario tener una disputa en curso cuando es mucho más fácil simplemente revertir el cambio, para calmar a todos nuevamente y luego intentar averiguar cuál es la mejor manera de proceder. Si el cambio resulta ser lo mejor después de todo, se puede recuperar fácilmente. Si resulta que no es así, entonces los usuarios no tenían que vivir con el falso cambio en el árbol mientras todos debatían afanosamente sus méritos. La gente *muy* raramente pide deshacer cambios en el repositorio, ya que la discusión generalmente expone cambios malos o controvertidos incluso antes de que ocurra la confirmación, pero en ocasiones tan raras, el retroceso debe hacerse sin discutir para que podamos pasar inmediatamente al tema de resolver si era falso o no.

6. Los cambios van a FreeBSD-CURRENT antes de FreeBSD-STABLE a menos que el ingeniero de versiones lo permita específicamente o que no sean aplicables a FreeBSD-CURRENT. Cualquier cambio no trivial o no urgente que sea aplicable también debe permitirse que permanezca en FreeBSD-CURRENT durante al menos 3 días antes de fusionarse para que se puedan realizar las pruebas suficientes. El ingeniero de versiones tiene la misma autoridad sobre la rama FreeBSD-STABLE como se describe en la regla # 5.

   Este es otro problema de tipo "no discutas sobre eso", ya que es el ingeniero de versiones el responsable en última instancia (y recibe una paliza) si un cambio resulta ser malo. Respeta esto y brinda al ingeniero de versiones tu total cooperación cuando se trata de la rama FreeBSD-STABLE. El manejo de FreeBSD-STABLE puede parecer con frecuencia demasiado conservador para el observador casual, pero también ten en cuenta el hecho de que se supone que el conservadurismo es el sello distintivo de FreeBSD-STABLE y que se aplican reglas diferentes a las de FreeBSD-CURRENT. Tampoco tiene sentido que FreeBSD-CURRENT sea un campo de pruebas si los cambios se fusionan con FreeBSD-STABLE inmediatamente. Los cambios necesitan la oportunidad de ser probados por los desarrolladores de FreeBSD-CURRENT, así que deja pasar un tiempo antes de fusionarlos, a menos que la corrección de FreeBSD-STABLE sea crítica, urgente o tan obvia como para hacer innecesarias más pruebas (corrección de errores / errores tipográficos, etc.) En otras palabras, aplica el sentido común.

   Los cambios a las ramas de seguridad (por ejemplo, `releng/9.3`) deben ser aprobados por un miembro de `Grupo Responsables de Seguridad <security-officer@FreeBSD.org>`, o en algunos casos, por un miembro de `Grupo de Ingeniería de Releases <re@FreeBSD.org>`.

7. No luches en público con otros committers; se ve mal.

Este proyecto tiene una imagen pública que defender y esa imagen es muy importante para todos nosotros, especialmente si queremos seguir atrayendo nuevos miembros. Habrá ocasiones en las que, a pesar de los mejores intentos de autocontrol de todos, se pierden los ánimos y se intercambian palabras de enojo. Lo mejor que se puede hacer en tales casos es minimizar los efectos de esto hasta que todos se hayan calmado de nuevo. No transmitas palabras de enojo en público y no reenvíes correspondencia privada u otras comunicaciones privadas a listas de correo públicas, alias de correo, canales de mensajería instantánea o sitios de redes sociales. Lo que la gente dice cara a cara a menudo está menos suavizado que lo que dirían en público y, por lo tanto, tales comunicaciones no tienen cabida allí; solo sirven para inflamar una situación que ya es mala. Si la persona que envía un mensaje incendiario al menos tuvo el detalle de enviarlo en privado, entonces ten el detalle de mantenerlo en privado. Si sientes que otro desarrollador te está tratando injustamente y te está causando angustia, plantea el asunto a Core en lugar de hacerlo público. Core hará todo lo posible para pacificar y hacer que las cosas vuelvan a la cordura. En los casos en que la disputa implique un cambio en la base de código y los participantes no parezcan estar llegando a un acuerdo amistoso, Core puede designar a un tercero de mutuo acuerdo para resolver la disputa. Todas las partes involucradas deben aceptar quedar vinculadas por la decisión tomada por este tercero.

8. Respeta todas las congelaciones de código y lee las listas de correo de `committers` y `developers` regularmente de forma que sepas cuando una congelación está en curso.

   Realizar cambios no aprobados durante una congelación de código es un gran error y se espera que los committers se mantengan actualizados sobre lo que está sucediendo antes de saltar después de una larga ausencia y hacer commit de 10 megabytes de material acumulado. A las personas que abusen de esto de forma regular se les suspenderán sus privilegios de commit hasta que regresen del Happy Reeducation Camp de FreeBSD que llevamos a cabo en Groenlandia.

9. En caso de duda sobre cualquier procedimiento, ¡pregunta primero!

   Cuando se tiene prisa se cometen muchos errores y simplemente asume que sabe la forma correcta de hacer algo. Si no lo has hecho antes, es muy probable que no sepas realmente la forma en que hacemos las cosas y realmente necesites preguntar primero o te avergonzarás por completo en público. No hay vergüenza en preguntar "¿Cómo diablos hago esto?" Ya sabemos que eres una persona inteligente; de lo contrario, no serías un committer.

10. Prueba tus cambios antes de realizarlos.

    Si tus cambios son en el kernel, asegúrate de que aún puedes compilar tanto GENERIC como LINT. Si tus cambios están en cualquier otro lugar, asegúrate de que aún puedes compilar el resto del sistema (make world). Si tus cambios son en una rama, asegúrate de que la prueba se realice con una máquina que ejecute ese código. Si tienes un cambio que también puede romper otra arquitectura, asegúrate de probar en todas las arquitecturas compatibles. Por favor asegúrate de que tu cambio funciona para supported toolchains. Por favor dirígete a FreeBSD Internal Page para obtener una lista de los recursos disponibles. A medida que se agregan otras arquitecturas a la lista de plataformas compatibles con FreeBSD, los recursos de prueba compartidos apropiados estarán disponibles.

11. No hagas commit en software contribuido sin aprobación *explícita* de los respectivos

mantenedores.

Código contribuido es cualquier cosa bajo los árboles src/contrib, src/crypto, o src/sys/contrib.

Los árboles mencionados anteriormente son para software contribuido generalmente importado a una rama de un proveedor. Hacer commit allí puede causar dolores de cabeza innecesarios al importar versiones más nuevas del software. En general, considera enviar parches directamente al proveedor. Los parches se pueden enviar a FreeBSD primero con el permiso del desarrollador.

Las razones para modificar el software en el proyecto original van desde querer un control estricto sobre una dependencia estrechamente acoplada hasta la falta de portabilidad en la distribución del código del repositorio canónico. Independientemente de la razón, el esfuerzo por minimizar la carga de mantenimiento de nuestra copia es útil para los compañeros mantenedores. Evita realizar cambios triviales o estéticos en los archivos, ya que hace que cada merge a partir de entonces sea más difícil: dichos parches deben volver a verificarse manualmente en cada importación.

Si un trozo particular de software no tienen mantenedor, se te anima a que tomes propiedad del mismo. Si no estás seguro del estado actual del mantenimiento del código envía un correo a Lista sobre arquitectura y diseño de FreeBSD y pregunta.

20.2. Política sobre arquitecturas múltiples

FreeBSD ha añadido varias arquitecturas nuevas durante los últimos ciclos de lanzamiento y ya no es en realidad un sistema operativo centrado en i386™. En un esfuerzo por hace más fácil el poder mantener FreeBSD portable en las distintas plataformas que soportamos, Core ha desarrollado esta exigencia:

Nuestra plataforma de referencia de 32 bits es i386 y nuestra plataforma de referencia de 64 bits es amd64. El trabajo de diseño importante (incluidos los cambios importantes de API y ABI) debe demostrar su valía en al menos una plataforma de 32 bits y al menos una de 64 bits, preferiblemente las plataformas de referencia primarias, antes de que se pueda hacer commit en el árbol de fuentes.

Los desarrolladores también deben conocer nuestra Política de Niveles para el soporte a largo plazo de arquitecturas de hardware. Las reglas aquí están destinadas a proporcionar una guía durante el proceso de desarrollo y son distintas de los requisitos para las características y arquitecturas enumeradas en esa sección. Las reglas de nivel para el soporte de características en arquitecturas en el momento del lanzamiento son más estrictas que las reglas de cambios durante el proceso de desarrollo.

20.3. Política sobre Múltiples Compiladores

FreeBSD compila tanto con Clang como con GCC. El proyecto hace esto de forma cuidadosa y controlada para maximizar los beneficios de este trabajo extra, a la vez que mantiene el trabajo extra en mínimos. Suportar tanto Clang como GCC mejora la flexibilidad que tienen nuestros usuarios. Estos compiladores tienen distintas fortalezas y debilidades, y soportar ambos permite a

los usuarios escoger el que mejor se adapta a sus necesidades. Clang y GCC soportan dialectos similares de C y C++, necesitándose una cantidad relativamente pequeña de código condicional. El proyecto gana más cobertura de código y mejora la calidad del código usando características de ambos compiladores. El proyecto es capaz de compilar en más entornos de usuario y aprovechar más entornos de CI al soportar este rango, incrementando las ventajas para los usuarios y dándoles más herramientas con las que probar. Mediante la restricción cuidadosa de las versiones modernas soportadas en estos compiladores, el proyecto evita incrementar la matriz de pruebas sin necesidad. Los compiladores más viejos y oscuros, así como dialectos más antiguos de los lenguajes, tienen un soporte extremadamente limitado que permite a los programas de usuarios compilar con ellos, pero sin limitar a que el sistema base se compile con ellos. El equilibro exacto está en constante evolución para asegurar que los beneficios del trabajo extra son mayores que la carga que imponen. El proyecto solía soportar compiladores de Intel realmente antiguos o versiones antiguas de GCC, pero cambiamos soportar esos compiladores obsoletos por una selección cuidadosas de compiladores modernos. Esta sección documenta dónde usamos los diferentes compiladores, y las expectativas al respecto.

El proyecto FreeBSD incorpora el compilador Clang. Debido a que está en el árbol, este es el compilador mejor soportado. Todos los cambios tienen que compilar con él, antes de hacer el commit. Las comprobaciones completas, como sean apropiadas para el cambio, se deberían hacer con este compilador.

En cualquier momento, el proyecto FreeBSD también soporta uno o más compiladores fuera del árbol. En este momento, esto es GCC 12.x. Idealemente, los committers deberían compilar con este compilador, especialmente para cambios grandes o arriesgados. El compilador está disponible como el paquete `${TARGET_ARCH}-gcc${VERSION}` como devel/freebsd-gcc12@aarch64 o devel/freebsd-gcc12@riscv64. El proyecto ejecuta trabajos automáticos de CI para compilar todo con estos compiladores. Se espera que los committers arreglen los trabajos que se rompan con sus cambios. Los committers pueden probar la compilación con, por ejemplo `CROSS_TOOLCHAIN=aarch64-gcc12` o `CROSS_TOOLCHAIN=llvm15` cuando sea necesario.

El proyecto FreeBSD también tiene algunos pipelines de CI en github. Para las pull requests en github y algunas ramas empujadas a los forks de github, se ejecutan algunos trabajos de compilación cruzada. Estos comprueba la compilación de FreeBSD usando una versión de Clang que a veces durante un tiempo está una versión por delante de la versión incluida en el árbol.

El proyecto FreeBSD también actualiza los compiladores. Tanto Clang como GCC se cambian constantemente. Algunos cambios en el árbol, por ejemplo eliminando las declaraciones y definiciones de funciones en estilo antiguo K&R, se introducirán en el árbol antes de cambiar el compilador. Los committers deberían tratar de ser conscientes de esto y ser receptivos a la hora de analizar problemas con su código o cambios con estos nuevos compiladores. Además, justo después de que se ha introducido una nueva versión del compilador en el árbol, la gente necesita compilar con la versión antigua si se sospecha que ha habido una regresión no detectada.

Además del compilador, el compilador usa directamente LDD de LLVM y las binutils de GNU. Los committers deberían ser conscientes de las diferencias en la sintaxis de ensamblador y las características de los enlazadores y asegurarse de que ambas variantes funcionan. Estos componentes se comprobarán como parte de los trabajos de CI de FreeBSD para Clang o GCC.

El proyecto FreeBSD proporciona cabeceras y librerías que permiten que se puedan usar otros

compiladores que no estén en el sistema base. Estas cabeceras tienen soporte para hacer que el entorno sea tan estricto como el estándar, soportando dialectos anteriores a ANSI-C hasta C89, y otros casos esquina que la colección de ports ha dejado al descubierto Este soporte limita la retirada de estándares antiguos en sitios como ficheros de cabecera, pero no limitan la actualización del sistema base a nuevos dialectos. Tampoco requiere que el sistema base compile con estos estándares antiguos. Romper el soporte causaría fallos en los paquetes de la colección de ports, de forma que se debería evitar en la medida de lo posible, y arreglarlo rápidamente cuando sea fácil hacerlo.

El sistema de compilación de FreeBSD actualmente soporta estos entornos diferentes. Conforme se añaden nuevos avisos a los compiladores, el proyecto intenta arreglarlos. Sin embargo, a veces estos avisos requieren un trabajo extensivo, de forma que se silencian de alguna forma usando variables que evalúen a lo que sea apropiado dependiendo de la versión del compilador. Los desarrolladores deberían ser conscientes de esto, y asegurar que cualquier flag específico de un compilador debería ser usado condicionalmente.

20.3.1. Versiones Actuales de los Compiladores

El compilador en el sistema base es actualmente Clang 15.x. Actualmente, se prueban GCC 12 y Clang 12, 13, 14 y 15 en los trabajos de CI de jenkins en github. Se está trabajando para preparar el árbol para Clang 16. La rama soportada más antigua del proyecto tiene Clang 12, así que las porciones del build que hacen el arranque deben funcionar con Clang desde la versión 12 hasta la 15.

20.4. Otras sugerencias

Al realizar cambios en la documentación, utiliza un corrector ortográfico antes de realizar el commit. Para todos los documentos XML, verifica que las directivas de formato sean correctas ejecutando `make lint` y textproc/igor.

Para páginas de manual, ejecuta sysutils/manck y textproc/igor sobre las páginas de manual para verificar que todas las referencias cruzadas y las referencias de ficheros son correctas y que la página del manual tiene instalados todos los `MLINKS` apropiados.

No mezcles arreglos de estilo con nuevas funciones. Una corrección de estilo es cualquier cambio que no modifica la funcionalidad del código. La combinación de los cambios confunde el cambio de funcionalidad al solicitar diferencias entre las revisiones, lo que puede ocultar cualquier error nuevo. No incluyas cambios de espacios en blanco con cambios de contenido en los commits de doc/. El desorden adicional en las diferencias hace que el trabajo de los traductores sea mucho más difícil. En su lugar, realiza cambios de estilo o espacios en blanco en commits separados que estén claramente etiquetados como tales en el mensaje de commit.

20.5. Funciones obsoletas

Cuando sea necesario eliminar la funcionalidad del software en el sistema base, sigue estas pautas siempre que sea posible:

1. En la página del manual y posiblemente en las notas de la versión se menciona que la opción,

utilidad o interfaz está obsoleta. El uso de la función obsoleta genera una advertencia.

2. La opción, utilidad o interfaz se conserva hasta la próxima versión principal (punto cero).
3. La opción, utilidad o interfaz se elimina y ya no se documenta. Ahora está obsoleto. También es generalmente una buena idea anotar su eliminación en las notas de la versión.

20.6. Privacidad y confidencialidad

1. La mayoría de los negocios de FreeBSD se realizan en público.

   FreeBSD es un proyecto *abierto*. Lo cual significa no solo que cualquiera puede usar el código fuente, sino que la mayoría del proceso de desarrollo está abierto para el escrutinio público.

2. Ciertos asuntos delicados deben permanecer privados o mantenidos bajo embargo.

   Lamentablemente, no puede haber una transparencia total. Como desarrollador de FreeBSD, tendrás un cierto grado de acceso privilegiado a la información. En consecuencia, se espera que respetes ciertos requisitos de confidencialidad. A veces la necesidad de confidencialidad proviene de colaboradores externos o tiene un límite de tiempo específico. Sin embargo, sobre todo, se trata de no liberar comunicaciones privadas.

3. El oficial de seguridad tiene el control exclusivo sobre la publicación de avisos de seguridad.

   Mientras que hay problemas de seguridad que afectan a muchos sistemas operativos diferentes, FreeBSD frecuentemente depende del acceso temprano para poder preparar avisos para el lanzamiento coordinado. A menos que se pueda confiar en que los desarrolladores de FreeBSD mantendrán la seguridad, dicho acceso temprano no estará disponible. El oficial de seguridad es responsable de controlar el acceso previo al lanzamiento a la información sobre vulnerabilidades y de programar el lanzamiento de todos los avisos. Puedes solicitar ayuda bajo condición de confidencialidad de cualquier desarrollador con conocimientos relevantes para preparar soluciones de seguridad.

4. Las comunicaciones con Core se mantienen confidenciales durante el tiempo que sea necesario.

   Las comunicaciones con Core inicialmente se tratarán de forma confidencial. Sin embargo, con el tiempo, la mayor parte del negocio de Core se resumirá en informes básicos mensuales o trimestrales. Se tendrá cuidado de no hacer públicos los detalles sensibles. Es posible que los registros de algunos temas particularmente sensibles no se informen en absoluto y se conservarán solo en los archivos privados de Core.

5. Es posible que se requieran acuerdos de no divulgación para acceder a ciertos datos comercialmente sensibles.

   El acceso a ciertos datos comercialmente sensibles solo puede estar disponible bajo un Acuerdo de Confidencialidad. Se debe consultar al personal legal de la Fundación FreeBSD antes de firmar cualquier acuerdo vinculante.

6. Las comunicaciones privadas no deben hacerse públicas sin permiso.

   Más allá de los requisitos específicos anteriores, existe una expectativa general de no publicar

comunicaciones privadas entre desarrolladores sin el consentimiento de todas las partes involucradas. Pide permiso antes de reenviar un mensaje a una lista de correo pública o publicarlo en un foro o sitio web al que puedan acceder otras personas que no sean los corresponsales originales.

7. Las comunicaciones en canales de acceso restringido o solo para proyectos deben mantenerse privadas.

   De manera similar a las comunicaciones personales, ciertos canales de comunicación internos, incluidas las listas de correo de FreeBSD Committer y los canales de IRC de acceso restringido, se consideran comunicaciones privadas. Se requiere permiso para publicar material de estas fuentes.

8. Core puede aprobar la publicación.

   Cuando no sea práctico obtener permiso debido a la cantidad de corresponsales o cuando el permiso para publicar se niegue sin razón, Core puede aprobar la divulgación de tales asuntos privados que merecen una publicación más general.

21. Soporte para múltiples arquitecturas

FreeBSD es un sistema operativo altamente portable destinado a funcionar en muchos tipos diferentes de arquitecturas de hardware. Mantener una separación limpia del código dependiente de la máquina (MD) y el código independiente de la máquina (MI), así como minimizar el código MD, es una parte importante de nuestra estrategia para permanecer ágiles con respecto a las tendencias actuales de hardware. Cada nueva arquitectura de hardware soportada por FreeBSD aumenta sustancialmente el coste del mantenimiento del código, el soporte de la cadena de herramientas y la ingeniería de versiones. También aumenta drásticamente el coste de las pruebas efectivas de los cambios del kernel. Como tal, existe una fuerte motivación para diferenciar entre clases de soporte para varias arquitecturas mientras se mantiene fuerte en algunas arquitecturas clave que se ven como FreeBSD "Público objetivo".

21.1. Declaración de intención general

El proyecto FreeBSD tiene como objetivo "estaciones de trabajo comerciales listas para usar (COTS) de calidad de producción, servidores y sistemas integrados de alta gama". Al mantener un enfoque en un conjunto estrecho de arquitecturas de interés en estos entornos, el Proyecto FreeBSD puede mantener altos niveles de calidad, estabilidad y rendimiento, así como minimizar la carga en varios equipos de soporte en el proyecto, como el equipo de ports, equipo de documentación, oficial de seguridad y equipos de ingenieros de versiones. La diversidad en el soporte de hardware amplía las opciones para los consumidores de FreeBSD al ofrecer nuevas características y oportunidades de uso, pero estos beneficios siempre deben considerarse cuidadosamente en términos del coste de mantenimiento del mundo real asociado con el soporte de plataforma adicional.

El Proyecto FreeBSD diferencia los objetivos de la plataforma en cuatro niveles. Cada nivel incluye una lista de garantías en las que los consumidores pueden confiar, así como las obligaciones del Proyecto y los desarrolladores para cumplir con esas garantías. Estas listas definen las garantías mínimas para cada nivel. El Proyecto y los desarrolladores pueden proporcionar niveles

adicionales de soporte más allá de las garantías mínimas para un nivel determinado, pero dicho soporte adicional no está garantizado. Cada objetivo de plataforma se asigna a un nivel específico para cada rama estable. Como resultado, a una plataforma de destino podría asignarsele diferentes niveles en ramas estables concurrentes.

21.2. Objetivos de plataforma

El soporte para una plataforma de hardware consta de dos componentes: el soporte del kernel y las interfaces binarias de aplicaciones (ABI) del área de usuario. El soporte de la plataforma del kernel incluye las cosas necesarias para ejecutar un kernel FreeBSD en una plataforma de hardware, como la administración de memoria virtual dependiente de la máquina y los controladores de dispositivo. Una ABI de área de usuario especifica una interfaz para que los procesos de usuario interactúen con un núcleo de FreeBSD y bibliotecas del sistema base. Una ABI de área de usuario incluye interfaces de llamada al sistema, el diseño y la semántica de las estructuras de datos públicas y el diseño y la semántica de los argumentos que se pasan a las subrutinas. Algunos componentes de una ABI pueden definirse mediante especificaciones como el diseño de objetos de excepción de C ++ o convenciones de llamada para funciones de C.

Un kernel de FreeBSD también usa una ABI (a veces denominada interfaz binaria del kernel (KBI)) que incluye la semántica y los diseños de las estructuras de datos públicas y el diseño y la semántica de los argumentos de las funciones públicas dentro del propio kernel.

Un kernel de FreeBSD puede admitir múltiples ABI de usuario. Por ejemplo, el kernel amd64 de FreeBSD es compatible con las ABI de área de usuario amd64 e i386 de FreeBSD, así como con las ABI de área de usuario de Linux x86_64 e i386. Un kernel de FreeBSD debería admitir un ABI "nativo" como ABI predeterminado. El "ABI" nativo generalmente comparte ciertas propiedades con la ABI del kernel, como la convención de llamadas de C, tamaños de tipos básicos, etc.

Los niveles se definen tanto para los núcleos como para las ABI del área de usuario. En el caso común, el kernel de una plataforma y las ABI de FreeBSD se asignan al mismo nivel.

21.3. Nivel 1: Arquitecturas totalmente compatibles

Las plataformas de nivel 1 son las plataformas FreeBSD más maduras. Están respaldados por el oficial de seguridad, la ingeniería de versiones y el Equipo de Gestión de Ports. Se espera que las arquitecturas de nivel 1 sean de calidad de producción con respecto a todos los aspectos del sistema operativo FreeBSD, incluidos los entornos de instalación y desarrollo.

El Proyecto FreeBSD ofrece las siguientes garantías a los consumidores de plataformas Tier 1:

- Las imágenes oficiales de lanzamiento de FreeBSD serán proporcionadas por el equipo de ingenieros de lanzamiento.
- Se proporcionarán actualizaciones binarias y parches de origen para avisos de seguridad y avisos de erratas para las versiones compatibles.
- Se proporcionarán parches de origen para avisos de seguridad para las sucursales admitidas.
- Las actualizaciones binarias y los parches de origen para los avisos de seguridad multiplataforma se proporcionarán normalmente en el momento del anuncio.

- Los cambios en las ABI del área de usuario generalmente incluirán ajustes de compatibilidad para garantizar el funcionamiento correcto de los binarios compilados en cualquier rama estable donde la plataforma sea de nivel 1. Es posible que estos ajustes no estén habilitados en la instalación predeterminada. Si no se proporcionan calzas de compatibilidad para un cambio de ABI, la falta de calzas se documentará claramente en las notas de la versión.
- Los cambios en ciertas partes de la ABI del kernel incluirán ajustes de compatibilidad para garantizar el funcionamiento correcto de los módulos del kernel compilados con la versión compatible más antigua de la rama. Tenga en cuenta que no todas las partes de la ABI del kernel están protegidas.
- El equipo de ports proporcionará paquetes binarios oficiales para software de terceros. Para las arquitecturas integradas, estos paquetes pueden construirse de forma cruzada a partir de una arquitectura diferente.
- Los ports más relevantes deberían construir o tener los filtros apropiados para evitar que se construyan otros inapropiados.
- Las nuevas características que no son inherentemente específicas de la plataforma serán completamente funcionales en todas las arquitecturas de Nivel 1.
- Las características y las correcciones de compatibilidad utilizadas por los binarios compilados contra ramas estables más antiguas pueden eliminarse en versiones principales más recientes. Dichas eliminaciones se documentarán claramente en las notas de la versión.
- Las plataformas de nivel 1 deben estar completamente documentadas. Las operaciones básicas se documentarán en el manual de FreeBSD.
- Las plataformas de nivel 1 se incluirán en el árbol de fuentes.
- Las plataformas de nivel 1 deben ser auto contenidas, ya sea a través de la cadena de herramientas en árbol o una cadena de herramientas externa. Si se requiere una cadena de herramientas externa, se proporcionarán paquetes binarios oficiales para una cadena de herramientas externa.

Para mantener la madurez de las plataformas de Nivel 1, el Proyecto FreeBSD mantendrá los siguientes recursos para apoyar el desarrollo:

- Crea y prueba el soporte de automatización, ya sea en el clúster de FreeBSD.org o en alguna otra ubicación fácilmente disponible para todos los desarrolladores. Las plataformas integradas pueden sustituir un emulador disponible en el clúster de FreeBSD.org por hardware real.
- Inclusión en los objetivos `make universe` y `make tinderbox`.
- Hardware dedicado en uno de los clústeres de FreeBSD para la construcción de paquetes (ya sea de forma nativa o mediante qemu-user).

En conjunto, los desarrolladores deben proporcionar lo siguiente para mantener el estado de Nivel 1 de una plataforma:

- Los cambios en el árbol de fuentes no deben romper conscientemente la construcción de una plataforma de Nivel 1.
- Las arquitecturas de nivel 1 deben tener un ecosistema maduro y saludable de usuarios y desarrolladores activos.

- Los desarrolladores deberían poder crear paquetes en sistemas de Nivel 1 no integrados y comúnmente disponibles. Esto puede significar compilaciones nativas si los sistemas no integrados están comúnmente disponibles para la plataforma en cuestión, o puede significar compilaciones cruzadas alojadas en alguna otra arquitectura de Nivel 1.
- Los cambios no pueden romper la ABI del área de usuario. Si se requiere un cambio de ABI, la compatibilidad de ABI para binarios existentes debe proporcionarse mediante el uso de versiones de símbolos o cambios de versión de biblioteca compartida.
- Los cambios combinados en ramas estables no pueden romper las partes protegidas de la ABI del kernel. Si se requiere un cambio de ABI del kernel, el cambio debe modificarse para preservar la funcionalidad de los módulos del kernel existentes.

21.4. Nivel 2: Arquitecturas de desarrollo y de nicho

Las plataformas de nivel 2 son plataformas FreeBSD funcionales, pero menos maduras. No cuentan con el apoyo del oficial de seguridad, la ingeniería de versiones y los equipos de administración de ports.

Las plataformas de nivel 2 pueden ser candidatas a plataformas de nivel 1 que aún se encuentran en desarrollo activo. Las arquitecturas que llegan al final de su vida útil también pueden pasar del estado de Nivel 1 al estado de Nivel 2 a medida que disminuye la disponibilidad de recursos para continuar manteniendo el sistema en un estado de Calidad de Producción. Las arquitecturas especializadas bien soportadas también pueden ser de Nivel 2.

El Proyecto FreeBSD proporciona las siguientes garantías a los consumidores de plataformas Tier 2:

- La infraestructura de ports debe incluir soporte básico para arquitecturas de Nivel 2 suficiente para soportar la construcción de ports y paquetes. Esto incluye soporte para paquetes básicos como ports-mgmt / pkg, pero no hay garantía de que los ports arbitrarios sean compilables o funcionales.
- Las nuevas características que no son inherentemente específicas de la plataforma deberían ser factibles en todas las arquitecturas de Nivel 2 si no se implementan.
- Las plataformas de nivel 2 se incluirán en el árbol de fuentes.
- Las plataformas de nivel 2 deben auto alojarse a través de la cadena de herramientas en árbol o una cadena de herramientas externa. Si se requiere una cadena de herramientas externa, se proporcionarán paquetes binarios oficiales para una cadena de herramientas externa.
- Las plataformas de nivel 2 deben proporcionar kernels funcionales y áreas de usuario incluso si no se proporciona una distribución de lanzamiento oficial.

Para mantener la madurez de las plataformas Tier 2, el Proyecto FreeBSD mantendrá los siguientes recursos para apoyar el desarrollo:

- Inclusión en los objetivos `make universe` y `make tinderbox`.

En conjunto, los desarrolladores deben proporcionar lo siguiente para mantener el estado de Nivel 2 de una plataforma:

- Los cambios en el árbol de fuentes no deberían romper a sabiendas la construcción de una

plataforma de Nivel 2.

- Las arquitecturas de nivel 2 deben tener un ecosistema activo de usuarios y desarrolladores.
- Si bien se permite que los cambios rompan la ABI del área de usuario, la ABI no debe romperse gratuitamente. Los cambios significativos en la ABI del área de usuario deben restringirse a las versiones principales.
- Las nuevas funciones que aún no se han implementado en las arquitecturas de nivel 2 deberían proporcionar un medio para desactivarlas en esas arquitecturas.

21.5. Nivel 3: Arquitecturas experimentales

Las plataformas de nivel 2 son plataformas FreeBSD funcionales, pero menos maduras. No cuentan con el apoyo del oficial de seguridad, la ingeniería de versiones y el Equipo de Gestión de Ports.

Las plataformas de nivel 3 son arquitecturas en las primeras etapas de desarrollo, para plataformas de hardware no convencionales, o que se consideran sistemas heredados con pocas probabilidades de tener un uso amplio en el futuro. El soporte inicial para las plataformas de Nivel 3 puede existir en un repositorio separado en lugar del repositorio de origen principal.

El Proyecto FreeBSD no ofrece garantías a los consumidores de plataformas de Nivel 3 y no se compromete a mantener los recursos para apoyar el desarrollo. Es posible que las plataformas de nivel 3 no siempre sean compilables, ni ningún núcleo o ABI de área de usuario se considera estable.

21.6. Arquitecturas No Soportadas

Otras plataformas no están soportadas en absoluto por el proyecto. El proyecto antes las describía como sistemas de Nivel 4.

Después de que una plataforma pase a ser no soportada, se elimina de los árboles de fuentes, ports y documentación todo su soporte. Nótese que el soporte en ports debe permanecer mientras la plataforma esté soportada en una rama todavía soportada por los ports.

21.7. Política sobre el cambio de nivel de una arquitectura

Los sistemas solo se pueden mover de un nivel a otro con la aprobación del Core Team de FreeBSD, que tomará esa decisión en colaboración con el Oficial de Seguridad, la Ingeniería de Versiones y el Equipo de Gestión de Ports. Para que una plataforma sea promovida a un nivel superior, las garantías de soporte que falten deben cumplirse antes de que se complete la promoción.

22. Preguntas frecuentes sobre ports específicos

22.1. Agregar un port nuevo

22.1.1. ¿Cómo agrego un nuevo port?

Añadir un port al árbol es algo relativamente sencillo. Una vez que el port está listo para ser añadido, como se explica en aquí, necesitas añadir la entrada al directorio de port en el Makefile de la categoría correspondiente. En este Makefile, los ports están listados en orden alfabético y añadidos a la variable `SUBDIR`, de este modo:

```
SUBDIR += newport
```

Una vez que el port y el Makefile de su categoría están listos, se puede hacer commit del nuevo port:

```
% git add category/Makefile category/newport
% git commit
% git push
```

No te olvides de establecer los hooks de git para el árbol de ports como se explica aquí; se ha desarrollado un hook específico para verificar la categoría del Makefile.

22.1.2. ¿Alguna otra cosa que deba saber cuando agregue un nuevo port?

Verifica el port, preferiblemente para asegurarse de que se compila y empaqueta correctamente.

The Porters Handbook's Testing Chapter contains more detailed instructions. See the Portclippy / Portfmt and the poudriere sections.

No necesitas eliminar todos los avisos pero asegúrate de haber corregido los más simples.

Si el port viene de alguien que no ha contribuido anteriormente al Proyecto, añade el nombre de esa persona a la sección Additional Contributors de la Lista de Colaboradores de FreeBSD.

Si el port vino a través de un PR, ciérralo. Para cerrar un PR, cambia el estado a `Issue Resolved` y la resolución a `Fixed`.

If for some reason using poudriere to test the new port is not possible, the bare minimum of testing includes this sequence:

```
# make install
# make package
# make deinstall
# pkg add package you built above
# make deinstall
# make reinstall
```

```
# make package
```

Date cuenta de que poudriere es la referencia para la construcción de paquetes, si el paquete no compila en poudriere, será eliminado.

22.2. Eliminar un port existente

22.2.1. ¿Cómo elimino un port existente?

Primero, lea la sección sobre copias del repositorio. Antes de eliminar el port, debe verificar que no haya otros ports que dependan de él.

- Asegúrese de que no haya dependencia del port en la colección de ports:
  - El PKGNAME del port aparece exactamente en una línea en un archivo INDEX reciente.
  - Ningún otro port contiene ninguna referencia al directorio del port o PKGNAME en sus Makefiles

Cuando uses Git, considera utilizar git-grep(1), es mucho más rápido que `grep -r`.

- Luego, quita el port:
  - Elimina los ficheros del port y el directorio con `git rm`.
  - Elimina la entrada `SUBDIR` del port en el Makefile del directorio padre.
  - Añade una entrada en ports/MOVED.
  - Elimina el port de ports/LEGAL si estuviera ahí.

Como alternativa, puedes utilizar el script rmport, de ports/Tools/scripts. Este script fue escrito por Vasil Dimov <vd@FreeBSD.org>. Cuando envíes preguntas acerca de este script a Lista de correo sobre los ports de FreeBSD, por favor, pon en copia a Chris Rees <crees@FreeBSD.org>, el actual mantenedor.

22.3. ¿Cómo muevo un port a un lugar nuevo?

1. Realiza una comprobación exhaustiva de la colección de ports buscando cualquier dependencia de la localización/nombre antiguo del port y actualízalos. Ejecutar `grep` en INDEX no es suficiente porque algunos ports tienen dependencias activadas a través de opciones de tiempo de compilación. Se recomienda hacer un git-grep(1) completo sobre la colección de ports.
2. Elimina la entrada `SUBDIR` del Makefile de la categoría antigua y añade una entrada `SUBDIR` en el Makefile de la nueva categoría.
3. Añade una entrada en ports/MOVED.

4. Busca entradas en los ficheros xml de ports/security/vuxml y ajústalos en consecuencia. En particular, verifica los paquetes anteriores con el nuevo nombre cuya versión podría incluir el nuevo port.
5. Mueve el port con `git mv`.
6. Haz commit de los cambios.

22.4. ¿Cómo copio un port a un lugar nuevo?

1. Copia el port con `cp -R old-cat/old-port new-cat/new-port`.
2. Añade el nuevo port a new-cat/Makefile.
3. Cambia lo que se necesite en new-cat/new-port.
4. Haz commit de los cambios.

22.5. Congelación de ports

22.5.1. ¿Qué es una "congelación de ports"?

Una "Congelación de ports" era un estado restringido en el que se colocaba el árbol de ports antes de un lanzamiento de versión. Se utilizó para garantizar una mayor calidad de los paquetes enviados con una versión. Solía durar un par de semanas. Durante ese tiempo, se solucionaban los problemas de compilación y se compilaban los paquetes para dicha versión. Esta práctica ya no se utiliza, ya que los paquetes para las versiones se crean a partir de la rama trimestral estable actual.

Para más información sobre cómo mergear commits en la rama trimestral, lee ¿Cuál es el procedimiento para solicitar autorización para fusionar un compromiso con la sucursal trimestral?.

22.6. Sucursales trimestrales

22.6.1. ¿Cuál es el procedimiento para solicitar autorización para fusionar un compromiso con la sucursal trimestral?

Desde el 30 de Noviembre de 2020 no es necesario buscar aprobación explícita para hacer commit en la rama trimestral.

22.6.2. ¿Cuál es el procedimiento para mergear commits con la rama trimestral?

Mergear commits a la rama trimestral (un proceso que llamamos MFH por razones históricas) es muy similar a hacer un commit MFC en el repositorio de src, así que básicamente:

```
% git checkout 2021Q2
% git cherry-pick -x $HASH
```

```
(verify everything is OK, for example by doing a build test)
% git push
```

donde `$HASH` es el hash del commit que quieres copiar a la rama trimestral. El parámetro `-x` asegura que se incluye el hash `$HASH` de la rama `main` en el nuevo mensaje de commit de la rama trimestral.

22.7. Crear una nueva categoría

22.7.1. ¿Cuál es el procedimiento para crear una nueva categoría?

Por favor, lee Proposing a New Category en el Porter's Handbook. Una vez que se ha seguido el procedimiento y que se ha asignado el PR a Grupo de Administración de ports <portmgr@FreeBSD.org>, es su decisión si se aprueba o no. Si lo hacen, es su responsabilidad:

1. Realiza los movimientos necesarios. (Esto solo se aplica a las categorías físicas.)
2. Actualiza la definición de `VALID_CATEGORIES` en ports/Mk/bsd.port.mk.
3. Asígnate el PR de nuevo.

22.7.2. ¿Qué debo hacer para implementar una nueva categoría física?

1. Actualizar cada Makefile de los ports movidos. No conectes todavía la nueva categoría a la compilación.

   Para hacer esto, necesitarás:

   1. Cambiar `CATEGORIES` del port (este era el objetivo del ejercicio, ¿recuerdas?) Primero se lista la nueva categoría. Esto ayudará a que el PKGORIGIN sea correcto.
   2. Ejecutar un `make describe`. Puesto que el `make index` de nivel raíz que ejecutarás en unos pocos pasos es una iteración de un `make describe` realizado sobre toda la jerarquía de ports, detectar cualquier error aquí te evitará tener que volver a ejecutar ese paso más adelante.
   3. Si quieres ser realmente concienzudo, ahora podría ser un buen momento para ejecutar portlint(1).

2. Comprueba que los `PKGORIGIN` son correctos. El sistema de ports utiliza la entrada `CATEGORIES` de cada port para crear su `PKGORIGIN`, el cual se usa para conectar los paquetes instalados con el directorio de port a partir del cual fue construido. Si esta entrada es incorrecta, herramientas habituales de los prots como pkg-version(8) y portupgrade(1) fallarán.

   Para hacer esto, utiliza la herramienta chkorigin.sh: `env PORTSDIR=/path/to/ports sh -e /path/to/ports/Tools/scripts/chkorigin.sh`. Esto comprobará cada port en el árbol, incluso aquellos que no estén conectados a la compilación, de forma que puedes ejecutarlo

directamente después de la operación de mover el port. Truco: ¡no te olvides de mirar los `PKGORIGIN` de los ports esclavos en los ports que acabas de mover!

3. En tu propio sistema local, comprueba los cambios propuestos: primero, comenta las entradas SUBDIR en los Makefiles de las categorías de los ports antiguos; luego activa la construcción de la nueva categoría en ports/Makefile. Ejecuta `make checksubdirs` en los directorios de las categorías afectadas para comprobar las entradas SUBDIR. Después en el directorio ports/ ejecuta `make index`. Esto puede durar más de 40 minutos incluso en sistemas modernos; sin embargo, es un paso necesario para evitar que otra gente tenga problemas.
4. Una vez hecho esto, puedes hacer commit del ports/Makefile actualizado para conectar la nueva categoría a la compilación y también hacer commit de los cambios en el Makefile para la(s) categoría(s) nueva(s).
5. Añade las entradas apropiadas a ports/MOVED.
6. Actualiza la documentación modificando:
   - el list of categories en el Porter's Handbook
7. Solo una vez que se haya hecho todo lo anterior, y ya no se informe de problemas con los nuevos ports, los ports antiguos deben eliminarse de sus ubicaciones anteriores en el repositorio.

22.7.3. ¿Qué debo hacer para implementar una nueva categoría virtual?

Esto es mucho más simple que una categoría física. Solo se necesitan algunas modificaciones:

- el list of categories en el Porter's Handbook

22.8. Preguntas misceláneas

22.8.1. ¿Hay cambios de los que se pueda hacer commit sin pedir la aprobación del mantenedor?

La aprobación general para la mayoría de los ports se aplica a estos tipos de arreglos:

- La mayoría de los cambios de infraestructura sobre un port (es decir, modernizarlo, pero no cambiar la funcionalidad). Por ejemplo, el "blanket" cubre convertir ports para que utilicen una nueva macro `USES`, habilitar compilación con más información de log y cambiar a una nueva sintaxis en el sistema de ports.
- Arreglos triviales *y probados* en compilación y tiempo de ejecución.
- Cambios de documentación y metadatos en los ports, como pkg-descr o `COMMENT`.

> Cualquier cosa mencionada por Grupo de Administración de ports <portmgr@FreeBSD.org> o el Grupo Responsables de Seguridad <security-officer@FreeBSD.org> pueden ser excepciones a estas reglas. Nunca se pueden hacer commits no autorizados en ports mantenidos por esos grupos.

22.8.2. ¿Cómo sé si mi port se está construyendo correctamente o no?

Los paquetes se construyen varias veces por semana. Si un port falla, el mantenedor recibe un email de `pkg-fallout@FreeBSD.org`.

Informes de todos las construcciones de paquetes (oficiales, experimentales y de no-regresión) se agregan en pkg-status.FreeBSD.org.

22.8.3. He añadido un nuevo port. ¿Necesito añadirlo al INDEX?

No. El fichero se puede generar bien ejecutando `make index`, o se puede descargar una versión pre-generada con `make fetchindex`.

22.8.4. ¿Hay otros archivos que no pueda tocar?

Cualquier fichero bajo ports/, o cualquier fichero bajo un subdirectorio que empieza con una letra mayúscula (Mk/, Tools/, etc.). En concreto, Grupo de Administración de ports <portmgr@FreeBSD.org> es muy protector con ports/Mk/bsd.port*.mk así que no hagas commit de cambios en esos ficheros a menos que quieras enfrentarte a su ira.

22.8.5. ¿Cuál es el procedimiento adecuado para actualizar la suma de comprobación de un archivo distfile de un pport cuando el archivo cambia sin un cambio de versión?

Cuando la suma de comprobación (checksum) de un archivo de distribución se actualiza debido a que el autor actualizó el archivo sin cambiar la revisión del port, el mensaje de confirmación incluye un resumen de las diferencias relevantes entre el archivo de distribución original y el nuevo para garantizar que el archivo de distribución no haya sido dañado o alterado maliciosamente. Si la versión actual del port ha estado en el árbol de ports durante un tiempo, una copia del antiguo archivo de distribución estará disponible en los servidores ftp; de lo contrario, se debe contactar con el autor o el encargado del mantenimiento para averiguar por qué ha cambiado el archivo de distribución.

22.8.6. ¿Cómo se puede solicitar una construcción experimental (exp-run) del árbol de ports?

Se debe completar una ejecución de exp-run antes de que se haga commit de parches con un impacto significativo en los ports. El parche puede ser contra el árbol de ports o el sistema base.

Se hará una construcción completa con los parches proporcionados por el peticionario, y éste es responsable de corregir los problemas detectados *(fallout)* antes de hacer commit.

1. Visita la página Bugzilla new PR page.
2. Selecciona el producto relacionado con tu parche.
3. Completa el informe de error como de costumbre. Recuerda adjuntar el parche.
4. Si arriba dice “Show Advanced Fields”, haz click en el enlace. Ahora dirá “Hide Advanced Fields”. Habrá disponibles muchos más campos. Si ya dice “Hide Advanced Fields”, no se

necesita hacer nada.

5. En la sección “Flags”, establece “exp-run” a ?. Respecto a los otros campos, pasando el ratón por encima de cualquier campo hace que se muestren más detalles.
6. Envía. Espera a que se ejecute la compilación.
7. El Grupo de Administración de ports <portmgr@FreeBSD.org> contestará con los posibles errores detectados.
8. Dependiendo del resultado:
   - Si no hay errores, el procedimiento se detiene aquí y se puede hacer commit del cambio, pendiente de cualquier otra aprobación requerida.
     i. Si hay errores, *deben* ser corregidos, bien arreglando los ports directamente en el árbol de ports, o añadiéndolo al parche enviado.
     ii. Una vez hecho esto, vuelve al paso 6 y di que los errores se han solucionado y espera a que se vuelva a ejecutar el exp-run. Repite mientras haya ports rotos.

23. Problemas Específicos para Desarrolladores que No Son Committers

Algunas personas que tienen acceso a las máquinas FreeBSD no tienen commit bits. Casi todo este documento también aplicará a estos desarrolladores (excepto los aspectos específicos de los commits y las pertenencias a las listas de correo que las acompañan). En particular, te recomendamos que leas:

- Detalles administrativos
- Para Todos

> Pídele a tu mentor que te añada al "Additional Contributors" (doc/shared/contrib-additional.adoc), si todavía no estás en la lista.

- Relaciones con los desarrolladores
- Guía de inicio rápido de SSH
- La gran lista de reglas de los Committers de FreeBSD

24. Información sobre Google Analytics

A partir del 12 de diciembre de 2012, se habilitó Google Analytics en el sitio web del Proyecto FreeBSD para recopilar estadísticas de uso anónimas con respecto al uso del sitio.

> El 3 de Marzo de 2022, Google Analytics fue eliminado del Proyecto FreeBSD.

25. Preguntas misceláneas

25.1. ¿Cómo accedo a people.FreeBSD.org para incluir algo de información personal o información acerca de un proyecto?

`people.FreeBSD.org` es lo mismo que `freefall.FreeBSD.org`. Simplemente crea un directorio public_html. Cualquier cosa que dejes en ese directorio será automáticamente visible bajo https://people.FreeBSD.org/.

25.2. ¿Dónde se almacenan los archivos de la lista de correo?

Las listas de correo se archivan en /local/mail en `freefall.FreeBSD.org`.

25.3. Me gustaría ser mentor de un nuevo committer. ¿Qué proceso debo seguir?

Lee el documento New Account Creation Procedure en las páginas internas.

26. Beneficios y Ventajas para los committers de FreeBSD

26.1. Reconocimiento

El reconocimiento como ingeniero de software competente es el valor más duradero. Además, tener la oportunidad de trabajar con algunas de las mejores personas con las que todo ingeniero soñaría conocer ¡es una gran ventaja!

26.2. Centro comercial FreeBSD

Los committers de FreeBSD pueden obtener gratis en las conferencias un conjunto de 4-CDs o DVD de FreeBSD Mall, Inc..

26.3. `Gandi.net`

Gandi proporciona hospedaje web, computación en la nube, registro de dominios y servicios de certificados X.509.

Gandi oferta una tarifa E-rate de descuento a todos los desarrolladores de FreeBSD. Para facilitar el proceso de obtener el descuento, primero crea una cuenta en Gandi, rellena la información de

facturación y selecciona la moneda. Después envía un email a non-profit@gandi.net usando tu dirección `@freebsd.org` e indica tu identificador de Gandi.

26.4. rsync.net

rsync.net proporciona almacenamiento en la nube para backup que está optimizado para usuarios UNIX. Su servicio funciona en su totalidad con FreeBSD y ZFS.

rsync.net oferta una cuenta de 500 GB gratis para siempre para los desarrolladores de FreeBSD. Simplemente regístrate en https://www.rsync.net/freebsd.html usando tu dirección `@freebsd.org` para recibir esta cuenta gratuita.

26.5. JetBrains

JetBrains es una compañía de desarrollo de software que crea herramientas para desarrolladores de software y gestores de proyectos. La compañía ofrece varios entornos integrados de desarrollo (IDEs) para distintos lenguajes de programación.

JetBrain oferta 100 licencias anuales de forma gratuita para todos sus IDE. Simplemente regístrate en https://account.jetbrains.com/a/322tl3z7 usando tu dirección `@freebsd.org` y la cuenta tendrá una licencia asociada a ella automáticamente. Una vez que la cuenta esté activa úsala en cualquiera de los productos para activarlos y ya has terminado.

> **!** Por favor, utiliza estas licencias sólo para uso personal y no las compartas con nadie fuera del proyecto FreeBSD, ya que eso sería una violación de los términos de donación.