Руководство коммиттера

Аннотация

В этом документе представлена информация для сообщества коммиттеров FreeBSD. Все новые коммиттеры должны прочитать этот документ перед началом работы, а существующим коммиттерам настоятельно рекомендуется периодически его пересматривать.

Почти все разработчики FreeBSD имеют права на коммит в один или несколько репозиториев. Однако некоторые разработчики не имеют таких прав, и часть информации здесь применима и к ним. (Например, некоторые люди имеют права только для работы с базой данных отчётов о проблемах.) Дополнительную информацию можно найти в Вопросы.

Этот документ также может быть интересен участникам сообщества FreeBSD, которые хотят узнать больше о том, как работает проект.

Содержание

1. Административные детали

Способ логина	ssh(1), только протокол версии 2
Главный хост для входа в оболочку	`freefall.FreeBSD.org`
Референсные машины	`ref*.FreeBSD.org`, `universe*.freeBSD.org` (см. также ссылку Хосты проекта FreeBSD)
Узел SMTP	`smtp.FreeBSD.org:587` (см. также Настройка доступа SMTP).
`src/` Git-репозиторий	`ssh://git@gitrepo.FreeBSD.org/src.git`
`doc/` Git-репозиторий	`ssh://git@gitrepo.FreeBSD.org/doc.git`
`ports/` Git-репозиторий	`ssh://git@gitrepo.FreeBSD.org/ports.git`
Внутренние списки рассылки	developers (технически называемый all-developers), doc-developers, doc-committers, ports-developers, ports-committers, src-developers, src-committers. (Каждый репозиторий проекта имеет свои собственные списки рассылки -developers и -committers. Архивы этих списков можно найти в файлах /local/mail/repository-name-developers-archive и /local/mail/repository-name-committers-archive на `freefall.FreeBSD.org`.)
Ежемесячные отчёты основной команды (Core Team)	/home/core/public/reports на кластере `FreeBSD.org`.
Ежемесячные отчёты команды управления портами	/home/portmgr/public/monthly-reports в кластере `FreeBSD.org`.
Важные ветки Git в `src/`:	`stable/n` (`n`-STABLE), `main` (-CURRENT)

Для подключения к хостам проекта требуется ssh(1). Дополнительную информацию можно найти в Руководстве по быстрому началу работы с SSH.

Полезные ссылки:

- Внутренние страницы проекта FreeBSD
- Узлы проекта FreeBSD

- Административные группы проекта FreeBSD

2. Ключи OpenPGP для FreeBSD

Криптографические ключи, соответствующие стандарту OpenPGP (*Pretty Good Privacy*), используются проектом FreeBSD для аутентификации коммиттеров. Сообщения, содержащие важную информацию, такие как публичные SSH-ключи, могут быть подписаны с помощью OpenPGP-ключа, чтобы доказать, что они действительно отправлены коммиттером. Дополнительную информацию можно найти в PGP & GPG: Email for the Practical Paranoid от Michael Lucas и https://en.wikipedia.org/wiki/Pretty_Good_Privacy.

2.1. Создание ключа

Существующие ключи можно использовать, но сначала их следует проверить с помощью documentation/tools/checkkey.sh. В этом случае убедитесь, что у ключа есть FreeBSD user ID.

Для тех, у кого ещё нет ключа OpenPGP или требуется новый ключ, соответствующий требованиям безопасности FreeBSD, здесь показано, как его сгенерировать.

1. Установите security/gnupg. Добавьте следующие строки в ~/.gnupg/gpg.conf, чтобы задать минимально приемлемые настройки для подписи и предпочтений новых ключей (подробнее см. в документации по опциям GnuPG):

```
# Sorted list of preferred algorithms for signing (strongest to weakest).
personal-digest-preferences SHA512 SHA384 SHA256 SHA224
# Default preferences for new keys
default-preference-list SHA512 SHA384 SHA256 SHA224 AES256 CAMELLIA256 AES192
CAMELLIA192 AES CAMELLIA128 CAST5 BZIP2 ZLIB ZIP Uncompressed
```

2. Сгенерировать ключ:

```
% 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 2048-битные ключи с трёхлетним сроком действия обеспечивают достаточную защиту на данный момент (2022-10).

② Срок действия ключа в три года достаточно мал, чтобы устаревшие ключи, ослабленные растущей мощностью компьютеров, перестали использоваться, но достаточно велик, чтобы уменьшить проблемы управления ключами.

③ Используйте здесь своё настоящее имя, предпочтительно совпадающее с указанным в удостоверении личности, выданном государством, чтобы другим было проще подтвердить вашу личность. Текст, который может помочь другим идентифицировать вас, можно ввести в раздел Комментарий.
После ввода адреса электронной почты запрашивается парольная фраза. Методы создания безопасной парольной фразы вызывают споры. Вместо того чтобы предлагать один способ, вот несколько ссылок на сайты, описывающие различные методы: https://world.std.com/~reinhold/diceware.html, https://www.iusmentis.com/security/passphrasefaq/, https://xkcd.com/936/, https://en.wikipedia.org/wiki/Passphrase.

Защитите закрытый ключ и парольную фразу. Если закрытый ключ или парольная фраза могли быть скомпрометированы или раскрыты, немедленно уведомите accounts@FreeBSD.org и отзовите ключ.

Фиксация нового ключа показана в Шаги для новых коммиттеров.

3. Kerberos и LDAP веб-пароль для кластера FreeBSD

Кластер FreeBSD требует пароль Kerberos для доступа к определённым сервисам. Пароль Kerberos также служит веб-паролем LDAP, поскольку LDAP проксирует запросы к Kerberos в

кластере. Некоторые из сервисов, требующих его, включают:

- Bugzilla

Для создания новой учётной записи Kerberos в кластере FreeBSD или сброса пароля Kerberos для существующей учётной записи с использованием генератора случайных паролей:

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

Это должно быть выполнено с компьютера за пределами кластера FreeBSD.org.

Пароль Kerberos также можно установить вручную, войдя в `freefall.FreeBSD.org` и выполнив:

```
% kpasswd
```

Если ранее не использовались аутентифицированные через Kerberos службы кластера FreeBSD.org, будет отображено сообщение `Client unknown`. Эта ошибка означает, что сначала необходимо использовать метод `ssh kpasswd.freebsd.org`, показанный выше, для инициализации учётной записи Kerberos.

4. Типы битов коммита (прав на коммит)

Репозиторий FreeBSD содержит ряд компонентов, которые в совокупности поддерживают исходный код базовой операционной системы, документацию, инфраструктуру портов сторонних приложений и различные поддерживаемые утилиты. При выделении прав на коммит (commit bits) в FreeBSD указываются области дерева, где эти права могут быть использованы. Как правило, области, связанные с правами, отражают, кто авторизовал их выделение. Дополнительные области полномочий могут быть добавлены позже: в этом случае коммиттер должен следовать стандартной процедуре выделения прав на коммит для соответствующей области дерева, получив одобрение от соответствующей инстанции и, возможно, наставника для этой области на некоторый период времени.

Тип коммиттера	*Ответственный*	*Компоненты дерева исходного кода*
src	srcmgr@	src/
doc	doceng@	документация doc/, ports/, src/
ports	portmgr@	ports/

Биты коммитов, выделенные до разработки концепции областей ответственности, могут быть подходящими для использования во многих частях дерева. Однако здравый смысл подсказывает, что коммиттер, ранее не работавший в определённой области дерева,

должен перед коммитом получить рецензирование (review), согласование от соответствующего ответственного лица и/или работать с наставником. Поскольку правила поддержки кода различаются в зависимости от области дерева, это важно как для самого коммиттера, работающего в менее знакомой области, так и для других участников, работающих с деревом.

Коммиттерам рекомендуется запрашивать рецензирование своей работы в рамках обычного процесса разработки, независимо от области дерева, в которой происходит работа.

4.1. Политика активности коммиттеров в других деревьях

- Все коммиттеры могут изменять файлы src/share/misc/committers-*.dot, src/usr.bin/calendar/calendars/calendar.freebsd и ports/astro/xearth/files.
- Документационные коммиттеры могут вносить изменения в документацию в файлы src, такие как руководства, README, базы данных fortune, календарные файлы и исправления комментариев, без одобрения коммиттера src, при условии соблюдения обычных правил и внимания к коммитам.
- Любой коммиттер может вносить изменения в любое другое дерево с пометкой "Approved by" от некурируемого коммиттера с соответствующими правами. Курируемые коммиттеры (имеющие наставника) могут предоставлять пометку "Reviewed by", но не "Approved by".
- Коммиттеры могут получить дополнительный бит по обычному процессу: найти наставника, который предложит их srcmgr, doceng или portmgr, в зависимости от ситуации. После одобрения их добавят в 'access', и начнётся стандартный период наставничества, который будет включать продолжение отметки "Approved by" в течение некоторого времени.

4.1.1. Неявное (по умолчанию) одобрение для документации

Некоторые типы исправлений имеют "одобрение по умолчанию" от Группа Менеджеров Дерева Документации <doceng@FreeBSD.org>, что позволяет любому коммиттеру исправлять эти категории проблем в любой части дерева документации. Эти исправления не требуют одобрения или проверки от коммиттера документации, если у автора нет прав на коммит в документацию.

Общее одобрение применяется к следующим типам исправлений:

- Опечатки
- Тривиальные исправления

  Пунктуация, URL-адреса, даты, пути и имена файлов с устаревшей или некорректной информацией, а также другие распространённые ошибки, которые могут ввести читателей в заблуждение.

За годы в дереве документации были неявно одобрены некоторые случаи. Этот список

показывает наиболее распространённые из них:

- Изменения в documentation/content/ru/books/porters-handbook/versions/_index.adoc

  Значения __FreeBSD_version (Руководство по созданию портов), в основном используется коммиттерами src.

- Изменения в doc/shared/contrib-additional.adoc

  Сопровождение раздела Дополнительные участники FreeBSD.

- Все Шаги для новых коммиттеров, связанные с документацией
- Рекомендации по безопасности; Уведомления об ошибках; Релизы;

  Команда Офицера Безопасности <security-officer@FreeBSD.org> и Группа Выпуска Релизов FreeBSD <re@FreeBSD.org> используют эти разделы.

- Изменения в website/content/ru/donations/donors.adoc

  Координатор пожертвований для проекта FreeBSD <donations@FreeBSD.org> использует этот документ.

Перед любым коммитом необходимо выполнить тестовую сборку; подробности см. в разделах «Обзор» и «Процесс сборки документации FreeBSD» Руководства для новых участников проекта документации FreeBSD.

5. Руководство по Git

5.1. Основы Git

При поиске по ключевым словам "Git Primer" можно найти множество хороших материалов. Страницы Дэниела Милера Введение в Git и Вилли Виллуса Git - Краткое введение являются хорошими обзорами. Книга по Git также полная, но гораздо длиннее: https://git-scm.com/book/en/v2. Также обратите внимание на сайт https://dangitgit.com/, посвящённый распространённым ловушкам и подводным камням Git, на случай, если вам нужно исправить ошибки. Наконец, введение, ориентированное на компьютерных учёных, оказалось полезным для некоторых в объяснении мировоззрения Git.

Этот документ предполагает, что вы уже читали про это, и постарается не повторять основы (хотя кратко их рассмотрит).

5.2. Мини-руководство по Git

Это руководство имеет менее амбициозные цели, чем старое руководство по Subversion, но должно охватить основы.

5.2.1. Область применения

Если вы хотите загрузить FreeBSD, собрать его из исходных кодов и в целом поддерживать актуальность таким способом, это руководство для вас. Оно охватывает получение исходных кодов, их обновление, бинарный поиск (bisect) и кратко затрагивает способы работы с локальными изменениями. В нём изложены основы, а также даны полезные ссылки на более глубокие материалы для случаев, когда читателю будет недостаточно базовой информации. Другие разделы этого руководства посвящены более сложным темам, связанным с участием в проекте.

Цель этого раздела — выделить те аспекты Git, которые необходимы для отслеживания исходных кодов. Предполагается базовое понимание Git. В интернете есть множество вводных руководств по Git, но Книга по Git предлагает одно из лучших изложений.

5.2.2. Начало работы для разработчиков

Этот раздел описывает доступ на чтение и запись для коммиттеров, чтобы отправлять коммиты от разработчиков или контрибьюторов.

5.2.2.1. Повседневное использование

В приведённых ниже примерах замените `${repo}` на имя нужного репозитория FreeBSD: `doc`, `ports` или `src`.

- Клонировать репозиторий:

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

  В результате у вас в качестве удалённых (remote) должны быть официальные зеркала:

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

- Настройка данных коммиттера FreeBSD:

  Хук для коммита в repo.freebsd.org проверяет, что поле "Commit" соответствует информации о коммиттере в FreeBSD.org. Самый простой способ получить предлагаемую конфигурацию — выполнить скрипт `/usr/local/bin/gen-gitconfig.sh` на freefall:

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

- Установите URL для отправки (push URL):

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

В таком случае у вас должны быть раздельные URL для извлечения (fetch) и отправки (push) как наиболее эффективная настройка:

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

Еще раз обратите внимание, что `gitrepo.freebsd.org` является псевдонимом для `repo.freebsd.org`.

- Установка хука для шаблона сообщения коммита:

Для репозитория документации:

```
% cd .git/hooks
% ln -s ../../.hooks/prepare-commit-msg
```

Для репозитория портов:

```
% git config --add core.hooksPath .hooks
```

Для репозитория src:

```
% cd .git/hooks
% ln -s ../../tools/tools/git/hooks/prepare-commit-msg
```

5.2.2.2. Ветка "admin"

Файлы `access` и `mentors` хранятся в отдельной (orphan) ветке `internal/admin` в каждом репозитории.

Следующий пример показывает, как переключиться (check out) на ветку `internal/admin` в локальной ветке с именем `admin`:

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

В качестве альтернативы вы можете добавить рабочее дерево для ветки `admin`:

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

Для просмотра ветки `internal/admin` в веб-интерфейсе: `https://cgit.freebsd.org/${repo}/log/?h=internal/admin`

Для отправки (push) укажите полную спецификацию ссылки:

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

5.2.3. Как поддерживать актуальную копию дерева исходных кодов FreeBSD src

Первый шаг: клонирование дерева. Это загружает всё дерево целиком. Существует два способа загрузки. Большинству пользователей потребуется глубокое клонирование репозитория. Однако бывают случаи, когда может потребоваться поверхностное клонирование.

5.2.3.1. Названия веток

FreeBSD-CURRENT использует ветку `main`.

`main` — это ветка по умолчанию.

Для FreeBSD-STABLE названия веток включают `stable/12` и `stable/13`.

Для FreeBSD-RELEASE, названия веток разработки выпусков включают `releng/12.4` и `releng/13.2`.

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

- ветки `main` и `stable/⋯` открыты
- ветки `releng/⋯`, каждая из которых замораживается при создании тега релиза.

Примеры:

- тег release/13.1.0 на ветке releng/13.1
- тег release/13.2.0 на ветке releng/13.2.

5.2.3.2. Репозитории

Пожалуйста, обратитесь к разделу Административные детали для получения актуальной информации о том, где взять исходные коды FreeBSD. Значение $URL ниже можно получить с этой страницы.

Примечание: Проект не использует подмодули, так как они плохо подходят для наших рабочих процессов и модели разработки. То, как мы отслеживаем изменения в сторонних приложениях, обсуждается в другом месте и, как правило, мало интересует обычного пользователя.

5.2.3.3. Полный клон

Полный клон загружает всё дерево целиком, включая всю историю и ветки. Это самый простой способ. Он также позволяет использовать функцию Git `worktree`, чтобы все активные ветки были извлечены в отдельные каталоги, но с единственной копией репозитория.

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

— создаст полную копию. `branch` должна быть одной из веток, перечисленных в предыдущем разделе. Если параметр `branch` не указан: будет использоваться ветка по умолчанию (`main`). Если параметр `<directory>` не указан: имя нового каталога будет соответствовать имени репозитория (doc, ports или src).

Вам понадобится полный клон, если вас интересует история, вы планируете вносить локальные изменения или работать более чем с одной веткой. Это также самый простой способ поддерживать актуальность. Если вас интересует история, но вы работаете только с одной веткой и у вас мало места, вы также можете использовать `--single-branch`, чтобы загрузить только одну ветку (хотя некоторые коммиты слияния не будут ссылаться на ветку, из которой произошло слияние, что может быть важно для некоторых пользователей, интересующихся детальными версиями истории).

5.2.3.4. Частичный клон

Частичный клон копирует только самый актуальный код, но не включает или включает лишь малую часть истории. Это может быть полезно, когда вам нужно собрать определённую ревизию FreeBSD или когда вы только начинаете и планируете более полно отслеживать дерево. Также вы можете использовать его, чтобы ограничить историю только определённым количеством ревизий. Однако обратите внимание на существенное ограничение этого подхода, описанное ниже.

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

Это клонирует репозиторий, но будет содержать только самую последнюю версию в репозитории. Остальная история не загружается. Если позже вы передумаете, вы можете выполнить `git fetch --unshallow`, чтобы получить старую историю.

При частичном клонировании вы потеряете счетчик коммитов в выводе команды uname. Это может затруднить определение необходимости обновления системы при выпуске бюллетеня безопасности.

5.2.3.5. Сборка

После загрузки сборка выполняется так, как описано в Руководстве, например:

```
% cd src
% make buildworld
```

```
% make buildkernel
% make installkernel
% make installworld
```

так что здесь это не будет рассматриваться подробно.

Если вы хотите собрать собственное ядро, в разделе конфигурации ядра руководства FreeBSD рекомендуется создать файл MYKERNEL в sys/${ARCH}/conf с вашими изменениями на основе GENERIC. Чтобы Git игнорировал MYKERNEL, его можно добавить в .git/info/exclude.

5.2.3.6. Обновление

Для обновления обоих типов деревьев используются одинаковые команды. Это загружает (pull) все изменения, сделанные после последнего обновления.

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

обновит дерево. В Git 'перемотка' (fast forward) — это слияние, которое только перемещает указатель ветки и не требует пересоздания коммитов. Если всегда выполнять слияние/загрузку (merge/pull) с перемоткой, это гарантирует точную копию дерева FreeBSD. Это важно, если вы хотите поддерживать локальные патчи.

См. ниже, как управлять локальными изменениями. Самый простой способ — использовать `--autostash` в команде `git pull`, но доступны и более сложные варианты.

5.2.4. Выбор конкретной версии

В Git команда `git checkout` используется для переключения как между ветками, так и между конкретными версиями. Версии в Git представляют собой длинные хеши, а не последовательные номера.

Когда вы извлекаете конкретную версию, просто укажите нужный хэш в командной строке (команда `git log` может помочь вам определиться, какой хэш выбрать):

```
% git checkout 08b8197a74
```

и вам это скопировано (checkout). Вы увидите сообщение, похожее на следующее:

```
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
```

где последняя строка формируется из хэша, который вы использовали для извлечения рабочей копии, и первой строки сообщения коммита из этой ревизии. Хэш может быть сокращен до минимальной уникальной длины. Сам Git не всегда последователен в том, сколько цифр он отображает.

5.2.5. Бинарный поиск (bisect)

Иногда что-то идёт не так. Последняя версия работала, но только что обновлённая — нет. Разработчик может попросить вас провести бинарный поиск проблемы, чтобы определить, какой коммит вызвал регрессию.

Git упрощает поиск изменений с помощью мощной команды `git bisect`. Вот краткое описание, как её использовать. Для получения дополнительной информации вы можете посмотреть https://www.metaltoad.com/blog/beginners-guide-git-bisect-process-elimination или https://git-scm.com/docs/git-bisect. Страница man git-bisect хорошо описывает, что может пойти не так, что делать, когда версии не собираются, в каких ситуациях вам лучше использовать другие условия поиска, а не 'хорошо (good)' и 'плохо (bad)', и так далее, но это здесь не будет рассматриваться.

`git bisect start --first-parent` запустит процесс бинарного поиска. Далее необходимо указать диапазон для проверки. `git bisect good XXXXXX` укажет рабочую версию, а `git bisect bad XXXXX` — нерабочую версию. Нерабочая версия почти всегда будет HEAD (специальный тег для текущего состояния). Рабочая версия будет последней, которую вы проверяли. Аргумент `--first-parent` необходим, чтобы последующие команды `git bisect` не пытались переключиться на ветку вендора, в которой отсутствует полное дерево исходников FreeBSD.

Если вы хотите узнать последнюю версию, которую вы извлекли, используйте `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
...
```

показывает, как я перемещаю рабочее дерево в ветку `main` (a816...) и затем обновляю его из вышестоящего репозитория (до 5ef0...). В этом случае, bad будет HEAD (или 5ef0bd68b515), а good — a8163e165c5b. Как видно из вывода, HEAD@{1} также часто работает, но не является безошибочным, если вы выполняли другие действия с деревом Git после обновления, но до того, как обнаружили необходимость в бинарном поиске.

Установите сначала «хорошую» версию, затем установите «плохую» (хотя порядок не имеет значения). При установке «плохой» версии вы получите некоторую статистику по процессу:

```
% 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.
```

Затем вы собираете и устанавливаете эту версию. Если она работает корректно, введите `git bisect good`, в противном случае — `git bisect bad`. Если версия не компилируется, введите `git bisect skip`. После каждого шага вы будете получать сообщение, аналогичное приведённому выше. По завершении сообщите о проблемной версии разработчику (или исправьте ошибку самостоятельно и отправьте патч). Команда `git bisect reset` завершит процесс и вернёт вас туда, откуда вы начали (обычно на вершину ветки `main`). Ещё раз, руководство по git-bisect (ссылка выше) — это хороший ресурс на случай возникновения проблем или нестандартных ситуаций.

5.2.6. Подписание коммитов, тегов и отправок с помощью GnuPG

Git умеет подписывать коммиты, теги и отправки (push). Когда вы подписываете Git-коммит или тег, вы можете доказать, что отправленный код действительно принадлежит вам и не был изменён во время передачи. Вы также можете подтвердить, что именно вы отправили код, а не кто-то другой.

Более подробная документация по подписанию коммитов и тегов доступна в главе Инструменты Git - Подписание вашей работы книги по Git.

Обоснование подписания отправок можно найти в коммите, где представлена эта функция.

Лучший способ — просто указать Git, что вы всегда хотите подписывать коммиты, теги и отправки. Это можно сделать, установив несколько переменных конфигурации:

```
% 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
```

Чтобы избежать возможных конфликтов, убедитесь, что вы указали длинный идентификатор ключа для Git. Вы можете получить длинный идентификатор с помощью команды: `gpg --list-secret-keys --keyid-format LONG`.

Для использования конкретных подключей, без разрешения GnuPG подключей в первичный ключ, добавьте `!` к ключу. Например, для шифрования с подключом `DEADBEEF` используйте `DEADBEEF!`.

5.2.6.1. Проверка подписей

Подпись коммита можно проверить, выполнив команду `git verify-commit <хэш коммита>` или `git log --show-signature`.

Подписи тегов можно проверить с помощью `git verify-tag <имя тега>` или `git tag -v <имя тега>`.

5.2.7. Особенности портов

Дерево портов работает аналогичным образом. Названия ветвей отличаются, и репозитории расположены в других местах.

Веб-интерфейс cgit для работы в браузере доступен по адресу https://cgit.FreeBSD.org/ports/ . Рабочий репозиторий Git находится по адресу https://git.FreeBSD.org/ports.git и ssh://anongit@git.FreeBSD.org/ports.git (или `anongit@git.FreeBSD.org:ports.git`).

Также доступно зеркало на GitHub, см. Внешние зеркала для обзора. Ветка *latest* называется `main`. Ветки *quarterly* именуются `yyyyQn`, где 'yyyy' — год, а 'n' — квартал.

5.2.7.1. Форматы сообщений коммитов

В репозитории портов доступен перехватчик, который помогает оформлять сообщения коммитов в .hooks/prepare-commit-message. Его можно включить, выполнив команду `git config --add core.hooksPath .hooks`.

Основная идея заключается в том, что сообщение коммита должно быть оформлено следующим образом:

```
category/port: Summary.

Description of why the changes where made (Объяснение, почему были сделаны изменения
).

PR:     12345
```

Первая строка — это тема коммита, в ней указывается, какой порт был изменён, и краткое описание коммита. Она должна содержать 50 символов или меньше.

Пустая строка должна отделять его от остальной части сообщения о коммите.

Остальная часть сообщения о коммите должна быть перенесена на границе 72 символов.

Еще одна пустая строка должна быть добавлена, если есть какие-либо поля метаданных, чтобы их можно было легко отличить от сообщения о коммите.

5.2.8. Управление локальными изменениями

Этот раздел посвящен отслеживанию локальных изменений. Если у вас нет локальных изменений, вы можете пропустить этот раздел.

Один важный момент для всех: все изменения остаются локальными, пока они не будут отправлены (push). В отличие от Subversion, Git использует распределённую модель. Для пользователей в большинстве случаев разница невелика. Однако, если у вас есть локальные изменения, вы можете использовать тот же инструмент для управления ими, что и для получения изменений из FreeBSD. Все изменения, которые вы не отправили, являются локальными и могут быть легко изменены (это делает git rebase, обсуждаемый ниже).

5.2.8.1. Сохранение локальных изменений

Самый простой способ сохранить локальные изменения (особенно незначительные) — использовать `git stash`. В простейшем случае вы используете `git stash`, чтобы записать изменения (что помещает их в стек stash). Большинство людей используют это для сохранения изменений перед обновлением дерева, как описано выше. Затем они используют `git stash apply`, чтобы повторно применить изменения к дереву. Stash представляет собой стек изменений, который можно просмотреть с помощью `git stash list`. Подробности приведены в man-странице git-stash (https://git-scm.com/docs/git-stash).

Этот метод подходит, когда у вас есть небольшие изменения в дереве. Если у вас что-то более сложное, вероятно, лучше будет создать локальную ветку и выполнять перебазирование. Сохранение изменений также интегрировано в команду `git pull`: просто добавьте `--autostash` в командную строку.

5.2.8.2. Сохранение локальной ветки

С помощью Git гораздо проще поддерживать локальную ветку, чем в Subversion. В Subversion необходимо выполнять коммит слияния и разрешать конфликты. Это выполнимо, но может привести к запутанной истории изменений, которую будет сложно передать в вышестоящий репозиторий, если это потребуется, или сложно воспроизвести при необходимости. Git также позволяет выполнять коммит слияния, но с теми же проблемами. Это один из способов управления веткой, но наименее гибкий.

В дополнение к слиянию, Git поддерживает концепцию «перебазирования» (rebase), которая позволяет избежать этих проблем. Команда `git rebase` воспроизводит все коммиты ветки в конце родительской ветки. Мы рассмотрим наиболее распространённые сценарии, возникающие при её использовании.

5.2.8.2.1. Создать ветку

Предположим, вы хотите внести изменение в команду `ls` FreeBSD, чтобы она никогда не использовала цветовое выделение. Существует множество причин для этого, но в данном примере мы будем использовать это в качестве базового сценария. Команда `ls` в FreeBSD периодически изменяется, и вам нужно будет адаптироваться к этим изменениям. К счастью, с помощью `git rebase` это обычно происходит автоматически.

```
% 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
```

Коммит откроет редактор, чтобы описать выполненные изменения. После ввода описания у вас будет собственная **локальная** ветка в Git-репозитории. Соберите и установите изменения, как обычно, следуя указаниям в руководстве. Git отличается от других систем контроля версий тем, что нужно явно указывать, какие файлы коммитить. Я предпочитаю делать это в командной строке коммита, но также можно использовать `git add`, как описано в более подробных руководствах.

5.2.8.2.2. Время обновить

Когда приходит время внедрить новую версию, процесс почти такой же, как и без веток. Вы выполняете обновление, как описано выше, но перед обновлением нужно выполнить одну дополнительную команду и одну после. Ниже предполагается, что вы начинаете с неизменённого дерева. Важно начинать операции перебазирования с чистого дерева (Git требует этого).

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

Это откроет редактор, в котором будут перечислены все коммиты. В данном примере не изменяйте его содержимое. Обычно это делается при обновлении базовой версии (хотя также можно использовать команду Git rebase для управления коммитами в ветке).

После завершения вышеуказанных действий необходимо перенести коммиты для `ls.c` со старой версии FreeBSD на новую.

Иногда возникают конфликты слияния. Это нормально. Не паникуйте. Вместо этого

решайте их так же, как и любые другие конфликты слияния. Чтобы упростить, я опишу лишь распространённую проблему, которая может возникнуть. Ссылка на полное руководство приведена в конце этого раздела.

Допустим, изменения включают переход на terminfo в вышестоящем коде, а также изменение названия опции. При обновлении вы можете увидеть следующее:

```
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
```

что выглядит пугающе.

Если открыть редактор, вы увидите типичное разрешение конфликта с трёхсторонним слиянием, с которым вы могли сталкиваться в других системах управления исходным кодом (остальная часть ls.c опущена):

```
<<<<<<< HEAD
#ifdef COLORLS_NEW
#include <terminfo.h>
=======
#undef COLORLS
#ifdef COLORLS
#include <termcap.h>
>>>>>>> 646e0f9cda11... no color ls
```

Новый код идёт первым, а ваш код - вторым.

Правильное решение здесь — просто добавить `#undef COLORLS_NEW` перед `#ifdef`, а затем удалить старые изменения:

```
#undef COLORLS_NEW
#ifdef COLORLS_NEW
#include <terminfo.h>
```

и сохранить файл.

Ребазирование было прервано, поэтому вам необходимо завершить его:

```
% git add ls.c
% git rebase --continue
```

что указывает Git, что файл `ls.c` исправлен и можно продолжить операцию перебазирования. Поскольку возник конфликт, система откроет редактор для обновления сообщения коммита (если это необходимо). Если сообщение коммита по-прежнему корректно, просто закройте редактор.

Если вы застряли в процессе перебазирования, не паникуйте. Команда `git rebase --abort` вернёт вас к исходному состоянию. Однако важно начинать с неизменённого дерева. Примечание: упомянутая выше команда `git reflog` будет полезна в этой ситуации, так как она содержит список всех (промежуточных) коммитов, которые вы можете просмотреть, изучить или выбрать через `cherry-pick`.

Для более подробного ознакомления с этой темой, см. довольно обширное руководство по адресу: https://www.freecodecamp.org/news/the-ultimate-guide-to-git-merge-and-git-rebase/. Этот ресурс будет полезен для решения проблем, которые возникают нечасто, но слишком специфичны для данного руководства.

5.2.8.3. Переключение на другую ветку FreeBSD

Если вы хотите перейти с ветки stable/12 на ветку current и у вас есть полный клон репозитория, будет достаточно выполнить следующую команду:

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

Однако, если у вас есть локальная ветка, есть несколько важных замечаний. Во-первых, перебазирование переписывает историю, поэтому вам, скорее всего, захочется сохранить её каким-либо образом. Во-вторых, переключение между ветками часто вызывает больше конфликтов. Если предположить, что приведённый выше пример относился к ветке stable/12, то для перехода на ветку `main` я бы рекомендовал следующее:

```
% 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
```

Что делает вышеописанное: извлекается ветка no-color-ls. Затем для неё создаётся новое имя (no-color-ls-stable-12) на случай, если потребуется вернуться к ней. После этого выполняется перебазирование на ветку `main`. Это позволит найти все коммиты в текущей ветке no-color-ls (вплоть до точки её ответвления от stable/12) и затем воспроизвести их на ветке `main`, создав там новую ветку no-color-ls (поэтому я и попросил вас создать резервное имя).

5.3. Процедуры MFC (Merge From Current)

5.3.1. Краткое содержание

Рабочий процесс MFC можно обобщить как `git cherry-pick -x` плюс `git commit --amend` для

корректировки сообщения коммита. Для нескольких коммитов используйте `git rebase -i`, чтобы объединить их вместе и отредактировать сообщение коммита.

5.3.2. Одиночный коммит MFC

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

Для коммитов MFC, например, импорта от вендора, вам потребуется указать одного родителя для целей выборочного применения (cherry-pick). Обычно это будет «первый родитель» ветки, из которой вы применяете изменения, то есть:

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

Если что-то пойдет не так, вам потребуется либо прервать выборочное применение с помощью `git cherry-pick --abort`, либо исправить проблему и выполнить `git cherry-pick --continue`.

После завершения выборочного применения выполните отправку с помощью `git push`. Если возникнет ошибка из-за проигрыша в гонке коммитов, используйте `git pull --rebase` и повторите попытку отправки.

5.3.3. MFC в ветку RELENG

MFC в ветки, требующие одобрения, требуют немного больше внимания. Процесс одинаков как для обычного слияния, так и для прямого коммита в исключительной ситуации.

- Сначала выполните слияние или прямой коммит в соответствующую ветку `stable/X`, прежде чем сливать в ветку `releng/X.Y`.
- Используйте хэш из ветки `stable/X` для MFC в ветку `releng/X.Y`.
- Оставляйте обе строки "cherry picked from" в сообщении коммита.
- Обязательно добавьте строку `Approved by:` при работе в редакторе.

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

Если вы забыли добавить строку `Approved by:`, вы можете выполнить `git commit --amend`, чтобы отредактировать сообщение коммита перед отправкой изменения.

5.3.4. Множественный коммит MFC

```
% git checkout -b tmp-branch stable/X
% for h in $HASH_LIST; do git cherry-pick -x $h; done
```

```
% git rebase -i stable/X
# отметить каждый коммит после первого как 'squash'
# При необходимости обновить сообщение коммита, чтобы отразить все его элементы.
# Обязательно сохранить строки "cherry picked from".
% git push freebsd HEAD:stable/X
```

Если отправка не удалась из-за проигрыша в гонке коммитов, выполните перебазирование и повторите попытку:

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

После завершения MFC вы можете удалить временную ветку:

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

5.3.5. MFC — импорт от вендора

Импорты от вендоров — это единственное в дереве, что создаёт коммит слияния в ветке `main`. Коммиты слияния выборочным переносом (cherry-pick) в stable/XX представляют дополнительную сложность, поскольку у коммита слияния два родителя. Как правило, вам понадобится разница с первым родителем, так как это разница с `main` (хотя могут быть и исключения).

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

обычно это то, что вам нужно. Это укажет выборочному переносу применить правильный diff.

Бывают, надеюсь, редкие случаи, когда возможно, что ветка `main` была объединена в обратном порядке скриптом преобразования. Если это произойдет (а мы пока таких случаев не обнаружили), вам следует изменить указанное выше на `-m 2`, чтобы выбрать правильного родителя. Просто выполните:

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

для этого. Опция `--abort` выполнит очистку после неудачной первой попытки.

5.3.6. Переделка MFC

Если вы делаете MFC, и всё идёт ужасно неправильно, и вы хотите начать сначала, то самый простой способ — использовать `git reset --hard`, как показано ниже:

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

хотя если у вас есть некоторые ревизии, которые вы хотите сохранить, а другие — нет, использование `git rebase -i` предпочтительнее.

5.3.7. Некоторые соображения о MFC

При внесении исходных коммитов в стабильные и релизные ветки мы преследуем следующие цели:

- Четко обозначить прямые коммиты, отличая их от коммитов, вносящих изменения из другой ветки.
- Избегать внесения известных нарушений в стабильные ветки и ветки выпуска релизов (releng).
- Позволить разработчикам определять, какие изменения были или не были перенесены из одной ветки в другую.

С помощью Subversion мы использовали следующие практики для достижения этих целей:

- Использование тегов `MFC` и `MFS` для пометки коммитов, которые сделали слияние изменения из другой ветки.
- Объединение (squash) исправляющих (fixup) коммитов с основным коммитом при слиянии изменения.
- Запись информации о слияниях для корректной работы `svn mergeinfo --show-revs`.

С помощью Git нам потребуется использовать другие стратегии для достижения тех же целей. Этот документ призван определить лучшие практики для коммитов слияния исходного кода с использованием Git, которые достигают этих целей. В целом мы стремимся использовать встроенные возможности Git для достижения этих целей, а не применять практики, основанные на модели Subversion.

Общее замечание: в связи с техническими различиями в Git, мы не будем использовать "коммиты слияния" Git (созданные через `git merge`) в стабильных ветках или ветках releng. Вместо этого, когда в документе упоминаются "коммиты слияния", имеется в виду коммит, изначально сделанный в `main`, который реплицируется или "переносится" в стабильную ветку, или коммит из стабильной ветки, который реплицируется в ветку releng с помощью вариации команды `git cherry-pick`.

5.3.8. Поиск подходящих хэшей для MFC

Git предоставляет встроенную поддержку этого через команды `git cherry` и `git log --cherry`. Эти команды сравнивают исходные различия коммитов (но не другие метаданные, такие

как сообщения журнала), чтобы определить, идентичны ли два коммита. Это хорошо работает, когда каждый коммит из `main` переносится как отдельный коммит в стабильную ветку, но перестаёт работать, если несколько коммитов из `main` объединяются в один коммит в стабильной ветке. Проект активно использует `git cherry-pick -x` с сохранением всех строк для обхода этих трудностей и разрабатывает автоматизированные инструменты для использования этого подхода.

5.3.9. Стандарты сообщений коммитов

5.3.9.1. Пометка MFC

Проект принял следующую практику для пометки MFC:

- Используйте флаг `-x` с командой `git cherry-pick`. Это добавляет строку в сообщение коммита, которая включает хэш оригинального коммита при слиянии. Поскольку это добавляется непосредственно Git, коммиттерам не нужно вручную редактировать журнал коммитов при слиянии.

При слиянии нескольких коммитов сохраняйте все строки "cherry picked from".

5.3.9.2. Обрезать метаданные?

Одной из областей, которая не была чётко задокументирована в Subversion (и даже в CVS), является форматирование метаданных в сообщениях журнала для коммитов MFC. Должны ли они включать метаданные из исходного коммита без изменений или должны быть изменены для отражения информации о самом коммите MFC?

Исторически практика различалась, хотя некоторые различия зависят от области. Например, MFC, относящиеся к PR, обычно включают поле PR в MFC, чтобы коммиты MFC включались в аудит-трекер системы отслеживания ошибок. В других областях ситуация менее ясна. Например, Phabricator показывает разницу последнего коммита, помеченного для обзора, поэтому включение URL-адресов Phabricator заменяет основной коммит на завершенные коммиты. Список рецензентов также неясен. Если рецензент одобрил изменение для `main`, означает ли это, что он одобрил коммит MFC? Верно ли это только для идентичного кода или лишь с незначительной доработкой? Очевидно, это неверно для более масштабных доработок. Даже для идентичного кода: что, если коммит не конфликтует, но вносит изменение ABI? Рецензент мог одобрить коммит для `main` из-за нарушения ABI, но может не одобрить слияние того же коммита в неизменном виде. Придется полагаться на собственное наилучшее суждение до тех пор, пока не будут согласованы четкие руководящие принципы.

Для MFC, регулируемых re@, добавляются новые поля метаданных, такие как тег Approved by для одобренных коммитов. Эти новые метаданные должны быть добавлены через `git commit --amend` или аналогичные средства после того, как исходный коммит будет проверен и одобрен. Мы также можем захотеть зарезервировать некоторые поля метаданных в коммитах MFC, такие как URL-адреса Phabricator, для использования re@ в будущем.

Сохранение существующих метаданных обеспечивает очень простой рабочий процесс. Разработчики используют `git cherry-pick -x` без необходимости редактировать сообщение

журнала.

Если же мы решим изменять метаданные в MFC, разработчикам придется явно редактировать сообщения журнала с помощью `git cherry-pick --edit` или `git commit --amend`. Однако по сравнению с svn, по крайней мере, существующее сообщение о фиксации может быть предварительно заполнено, а поля метаданных можно добавлять или удалять без необходимости повторного ввода всего сообщения о фиксации.

Суть в том, что разработчикам, вероятно, потребуется тщательно готовить свои сообщения о коммитах для MFC, которые не являются тривиальными.

5.4. Импорт вендоров с Git

В этом разделе подробно описывается процедура импорта вендора с использованием Git.

5.4.1. Соглашение о наименовании веток

Все ветки и теги вендоров начинаются с `vendor/`. Эти ветки и теги видны по умолчанию.

Эта глава следует соглашению, что `freebsd` — это имя источника для официального репозитория Git FreeBSD. Если вы используете другое соглашение, замените `freebsd` на используемое вами имя в приведённых ниже примерах.

Мы рассмотрим пример обновления mtree NetBSD, который находится в нашем дереве. Ветка вендора для него — `vendor/NetBSD/mtree`.

5.4.2. Обновление старого импорта от вендора

Деревья вендоров обычно содержат лишь подмножество стороннего программного обеспечения, подходящего для FreeBSD. Эти деревья, как правило, очень малы по сравнению с деревом FreeBSD. Таким образом, рабочие деревья Git довольно компактны и быстры, что делает их предпочтительным методом использования. Убедитесь, что выбранный вами каталог (`../mtree`) в настоящее время не существует.

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

5.4.3. Обновление исходных кодов в ветке вендора

Подготовьте полное, чистое дерево исходных кодов вендора. Импортируйте всё, но делайте слияние только тому, что необходимо.

Этот пример предполагает, что исходный код NetBSD получен из их зеркала на GitHub в каталоге `~/git/NetBSD`. Учтите, что «вышестоящий репозиторий» мог добавить или удалить файлы, поэтому мы хотим убедиться, что удаления также распространяются. Пакет net/rsync обычно установлен, поэтому я буду использовать его.

```
% 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
```

Крайне важно убедиться, что импортируемый исходный код получен из надёжного источника. Многие проекты с открытым исходным кодом используют криптографические подписи для подписывания изменений кода, тегов git и/или tar-архивов с исходным кодом. Всегда проверяйте эти подписи и используйте механизмы изоляции, такие как клетки, chroot, в сочетании с выделенной непривилегированной учётной записью пользователя, отличной от той, которую вы используете регулярно (подробнее см. в разделе «Обновление дерева исходного кода FreeBSD» ниже), пока вы не будете уверены, что импортируемый исходный код выглядит безопасным. Отслеживание разработки в вышестоящем репозитории и периодический просмотр изменений кода в нём могут значительно помочь в повышении качества кода и принести пользу всем участникам. Также рекомендуется изучать результаты git diff перед импортом их в область вендора.

Всегда выполняйте команды `git diff` и `git status` и внимательно изучайте результаты. В случае сомнений полезно выполнить `git annotate` на ветке вендора или в вышестоящем git-репозитории, чтобы увидеть, кто и почему внес изменение.

В примере выше мы использовали `-m` для иллюстрации, но вам следует составить корректное сообщение в редакторе (используя шаблон сообщения о фиксации).

Также важно создать аннотированный тег с помощью `git tag -a`, иначе отправка будет отклонена. Только аннотированные теги разрешены для отправки. Аннотированный тег даёт вам возможность ввести сообщение коммита. Укажите версию, которую вы импортируете, вместе с любыми важными новыми функциями или исправлениями в этой версии.

5.4.4. Обновление копии FreeBSD

На этом этапе вы можете отправить импорт в `vendor` в наш репозиторий.

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

`--follow-tags` указывает `git push` также отправлять теги, связанные с локально зафиксированной ревизией.

5.4.5. Обновление дерева исходных кодов FreeBSD

Теперь вам нужно обновить mtree в FreeBSD. Исходные коды находятся в `contrib/mtree`, так как это программное обеспечение из внешних источников.

Время от времени нам может потребоваться вносить изменения в предоставленный (contrib) код, чтобы лучше соответствовать потребностям FreeBSD. По возможности, пожалуйста, старайтесь передавать локальные изменения обратно в вышестоящие проекты — это помогает им лучше поддерживать FreeBSD, а также экономит ваше время при разрешении конфликтов в будущем при импорте обновлений.

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

Это создаст коммит слияния поддерева `contrib/mtree` с локальной веткой `vendor/NetBSD/mtree`. Изучите разницу (diff) между результатом слияния и содержимым вышестоящей ветки. Если слияние сократило наши локальные изменения до более тривиальных различий, таких как изменения пустых строк или отступов, попробуйте исправить локальные изменения, чтобы уменьшить разницу с вышестоящей версией, или попытайтесь внести оставшиеся изменения обратно в вышестоящий проект. Если возникли конфликты, вам потребуется устранить их перед коммитом. Включите подробности об объединяемых изменениях в сообщение коммита слияния.

Некоторое открытое программное обеспечение включает скрипт `configure`, который генерирует файлы, используемые для определения процесса сборки кода; обычно эти сгенерированные файлы, такие как `config.h`, должны обновляться как часть процесса импорта. При выполнении этого всегда помните, что эти скрипты являются исполняемым кодом, работающим под учётными данными текущего пользователя. Этот процесс всегда должен запускаться в изолированной среде, в идеале внутри клетки, не имеющей сетевого доступа, и с непривилегированной учётной записью; или, как минимум, с выделенной учётной записью, отличной от той, которую вы обычно используете для повседневных задач или для отправки изменений в репозиторий исходного кода FreeBSD. Это минимизирует риск столкновения с ошибками, которые могут привести к потере данных или, в худших случаях, к выполнению злонамеренно внедрённого кода. Использование изолированной клетки также предотвращает обнаружение скриптами configure локально установленных пакетов программного обеспечения, что может привести к неожиданным результатам.

При тестировании ваших изменений запускайте их сначала в chroot или в клетке, или даже внутри виртуальной машины, особенно для модификаций ядра или библиотек. Этот подход помогает предотвратить неблагоприятное взаимодействие с вашей рабочей средой. Это может быть особенно полезно для изменений в библиотеках, которые используют многие компоненты базовой системы среди прочего.

5.4.6. Перебазирование ваших изменений относительно последней версии исходного дерева FreeBSD

Поскольку текущая политика рекомендует избегать слияний (merge), то, если вышестоящая

ветка FreeBSD `main` продвинулась вперёд до того, как у вас появится возможность отправить изменения, вам придётся переделывать слияние.

Обычный `git rebase` или `git pull --rebase` не умеет перемещать коммит слияния **как коммит слияния**, поэтому вместо этого вам придётся воссоздать коммит.

Следующие шаги следует выполнить, чтобы легко воссоздать коммит слияния, как если бы `git rebase --merge-commits` сработал правильно:

- Перейдите в корень репозитория
- Создайте побочную ветвь `XXX` с **содержимым** дерева со слиянием.
- Обновите эту побочную ветку `XXX` для слияния и актуализации с основной веткой FreeBSD `main`.
    - В худшем случае вам всё равно придётся разрешать конфликты слияния, если они были, но это должно быть крайне редко.
    - Разрешите конфликты и, если необходимо, объедините (collapse) несколько коммитов в один (без конфликтов объединение не требуется)
- извлеките (checkout) ветку `main`
- создайте ветку `YYY` (позволяет проще откатиться, если что-то пойдет не так)
- Повторите слияние поддерева
- Вместо разрешения конфликтов от слияния поддерева, извлеките (checkout) содержимое XXX поверх него.
    - Завершающая `.` важна, как и нахождение на верхнем уровне репозитория.
    - Вместо переключения веток на XXX, он накладывает содержимое XXX поверх репозитория
- Сделайте коммит полученному результату с предыдущим сообщением коммита (пример предполагает, что в ветке XXX была только одна операция слияния).
- Убедитесь, что ветки одинаковые.
- Сделайте любые необходимые проверки, включая привлечение других людей для проверки, если вы считаете, что это необходимо.
- Запишите (push) этот коммит. Если вы «снова проиграли в гонке», просто повторите эти шаги (см. рецепт ниже)
- Удалите ветки после того, как коммит попадёт в вышестоящий репозиторий. Они одноразовые.

Команды, которые можно использовать, следуя приведённому выше примеру с mtree, будут выглядеть следующим образом (символ `#` начинает комментарий, помогающий связать команды с описаниями выше):

```
% cd ../src         # CD to top of tree
% git checkout -b XXX       # create new throw-away XXX branch for merge
% git fetch freebsd     # Get changes from upstream from upstream
% git merge freebsd/main    # Merge the changes and resolve conflicts
```

```
% git checkout -b YYY freebsd/main # Create 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
% git diff XXX YYY      # Should be empty
% git show YYY          # Should only have changes you want, and be a merge commit
from vendor branch
```

Примечание: если что-то пойдет не так с коммитом, вы можете сбросить ветку YYY, повторно выполнив команду checkout, которая создала её, с флагом -B, чтобы начать заново:

```
% git checkout -B YYY freebsd/main # Создать новую временную ветку YYY, если начать
заново будет проще
```

5.4.7. Отправка (push) изменений

Когда вы считаете, что у вас есть набор изменений, и они хорошие, вы можете отправить их в форк на GitHub или GitLab для просмотра и рецензирования другими. Одна из хороших особенностей Git заключается в том, что он позволяет публиковать черновые версии вашей работы для проверки другими. Хотя Phabricator хорош для проверки содержания, публикация обновленной ветки вендора и коммитов слияния позволяет другим проверить детали, которые в конечном итоге появятся в репозитории.

После проверки, когда вы уверены, что это хорошее изменение, вы можете отправить (push) его в репозиторий 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
```

Примечание: Я использовал XXX и YYY, чтобы было очевидно, что это ужасные имена, и они не должны покидать вашу машину. Если вы используете такие имена для другой работы, вам нужно будет выбрать другие имена или рискнуть потерять другую работу. В этих именах нет ничего волшебного. В вышестоящий репозиторий вам не разрешат их отправить, но, тем не менее, обратите внимание на точные команды выше. Некоторые команды используют синтаксис, который лишь незначительно отличается от типичного использования, и это различие в поведении критически важно для работы данного рецепта.

5.4.8. Как переделать вещи, если это необходимо

Если вы попытались выполнить отправку из предыдущего раздела и она завершилась неудачей, то вам следует выполнить следующие действия для «повторного выполнения» операций. Эта последовательность сохраняет коммит с сообщением о коммите всегда на позиции XXX~1 для упрощения коммита.

```
% 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
```

Затем проверьте, как описано выше, и отправьте (push), как описано выше, когда будет готово.

5.5. Создание новой ветки вендора

Существует несколько способов создания новой ветки вендора. Рекомендуемый способ — создать новый репозиторий и затем сделать его слияние с FreeBSD. Например, производится импорт `glorbnitz` версии 3.1415 в дерево FreeBSD. Для простоты мы не будем обрезать этот релиз. Это простая пользовательская команда, которая переводит устройство nitz в различные магические состояния glorb, и она достаточно мала, так что обрезка не сэкономит много.

5.5.1. Создайте репозиторий

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

На этом этапе у вас есть новый репозиторий, в котором все новые коммиты будут направляться в ветку `vendor/glorbnitz`.

Опытные пользователи Git также могут сделать это непосредственно в своей копии FreeBSD, используя `git checkout --orphan vendor/glorbnitz`, если им так удобнее.

5.5.2. Скопируйте себе исходники

Поскольку это новая импортированная копия, вы можете просто скопировать файлы исходного кода командой cp, либо использовать tar или даже rsync, как показано выше. И мы добавим всё, предполагая отсутствие файлов с точкой в начале имени.

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

На этом этапе у вас должна быть чистая копия glorbnitz, готовая к коммиту.

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

Как и выше, я использовал `-m` для простоты, но вам, вероятно, следует создать сообщение коммита, которое объясняет, что такое Glorb и почему для его получения используется Nitz. Не все это знают, поэтому для вашего реального коммита вам следует руководствоваться разделом сообщение коммита, а не имитировать краткий стиль, использованный здесь.

5.5.3. Теперь импортируйте его в наш репозиторий

Теперь необходимо импортировать ветку в наш репозиторий.

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

Обратите внимание, что ветка vendor/glorbnitz находится в репозитории. На этом этапе `/some/where/glorbnitz` можно удалить, если хотите. Это было лишь промежуточным шагом для достижения цели.

5.5.4. Сделайте тег и отправьте (push)

Дальнейшие шаги во многом аналогичны процессу обновления ветки вендора, за исключением самого шага обновления ветки вендора.

```
% git worktree add ../glorbnitz vendor/glorbnitz
% cd ../glorbnitz
% git tag --annotate vendor/glorbnitz/3.1415
# Убедитесь, что коммит хороший, с помощью "git show"
% git push --follow-tags freebsd vendor/glorbnitz
```

Под «хорошим» мы подразумеваем:

1. Все необходимые файлы присутствуют
2. Ни один из неправильных файлов не присутствует
3. Ветка вендора указывает на что-то разумное
4. Тег выглядит хорошо и имеет аннотацию
5. Сообщение коммита для тега содержит краткую сводку о нововведениях с момента предыдущего тега

5.5.5. Время наконец сделать слияние этого в базовое дерево

```
% cd ../src
% git subtree add -P contrib/glorbnitz vendor/glorbnitz
# Убедитесь, что коммит хороший, с помощью "git show"
```

```
% git commit --amend   # one last sanity check on commit message
% git push freebsd
```

Здесь «хороший» означает:

1. Все правильные файлы и ни одного неправильного были объединены слиянием в contrib/glorbnitz.
2. В дереве нет других изменений.
3. Сообщения коммитов должны выглядеть хорошо. Они должны содержать сводку изменений с момента последнего слияния с основной веткой FreeBSD `main` и любые предостережения.
4. `RELNOTES` и `UPDATING` должны быть обновлены, если есть что-то важное, например, заметные пользователям изменения, важные аспекты обновления и т.д.

Это ещё не подключило `glorbnitz` к сборке. Способ сделать это зависит от конкретного импортируемого программного обеспечения и выходит за рамки данного руководства.

5.5.5.1. Сохраняя актуальность

Итак, время идёт. Пришло время обновить дерево до последних изменений из вышестоящего репозитория. При переходе на ветку `main` (при checkout) убедитесь, что у вас нет незакоммиченных изменений. Намного проще закоммитить их в отдельную ветку (или использовать `git stash`) перед выполнением следующих действий.

Если вы привыкли к `git pull`, мы настоятельно рекомендуем использовать опцию `--ff-only` и дополнительно установить её в качестве опции по умолчанию. В качестве альтернативы, `git pull --rebase` полезен, если у вас есть изменения, проиндексированные (stage) в ветке `main`.

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

Возможно, вам потребуется опустить --global, если вы хотите, чтобы эта настройка применялась только к этому репозиторию.

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

Существует распространённая ловушка: команда `git pull` попытается выполнить слияние, что иногда создаёт коммит слияния, которого ранее не существовало. Восстановление после этого может быть затруднительно.

Рекомендуется также использовать расширенную форму.

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

Эти команды сбрасывают ваше дерево к ветке `main`, а затем обновляют его из исходного источника, откуда дерево было первоначально получено (pull). Важно переключиться на `main` перед выполнением этого, чтобы обеспечить продвижение вперёд. Теперь пришло время продвинуть изменения вперёд:

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

Это вызовет интерактивный экран для изменения настроек по умолчанию. Пока просто выйдите из редактора. Всё должно примениться автоматически. Если нет, то вам потребуется разрешить различия. Документация GitHub может помочь вам в этом процессе.

5.5.5.2. Время отправить (push) изменения вверх (upstream)

Сначала убедитесь, что URL для отправки (push) правильно настроен для вышестоящего репозитория.

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

Затем убедитесь, что имя пользователя и адрес электронной почты настроены правильно. Требуется, чтобы они точно соответствовали записи passwd в кластере FreeBSD.

Использование

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

на freefall.freebsd.org (при условии, что /usr/local/bin находится в PATH), чтобы получить готовый шаблон конфигурации, который можно использовать напрямую.

Следующая команда делает слияние ветки `working` в основную ветку `main` вышестоящего репозитория. Важно, чтобы вы подготовили свои изменения именно так, как хотите видеть их в исходном репозитории FreeBSD, перед выполнением этой операции. Данный синтаксис отправляет (push) ветку `working` в `main`, перемещая ветку `main` вперёд. Вы сможете сделать это только в том случае, если результатом будет линейное изменение для `main` (т.е. без слияний).

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

Если ваша отправка (push) отклонена из-за проигрыша в гонке коммитов, перебазируйте вашу ветку перед повторной попыткой:

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

5.5.5.3. Время отправить (push) изменения вверх (альтернативный вариант)

Некоторым удобнее делать слияние своих изменений в локальную ветку `main` перед отправкой в удалённый репозиторий. Кроме того, `git arc stage` перемещает изменения из ветки в локальную `main`, когда требуется выполнить только часть изменений из ветки. Инструкции схожи с предыдущим разделом:

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

Если вы проиграли гонку, попробуйте снова с

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

Эти команды получат (fetch) последние изменения из `freebsd/main`, а затем перебазируют (rebase) локальные изменения `main` поверх них, что и требуется, когда вы проиграли гонку коммитов. Примечание: коммиты слияния ветки вендоров не будет работать с этой техникой.

5.5.5.4. Поиск ревизии Subversion

Вам нужно убедиться, что вы получили (fetch) примечания (подробности смотрите в разделе Ежедневное использование). Как только вы это сделаете, примечания будут отображаться в команде git log следующим образом:

```
% git log
```

Если у вас есть конкретная версия, вы можете использовать эту конструкцию:

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

чтобы найти конкретную ревизию. Шестнадцатеричное число после 'commit' — это хэш, который можно использовать для ссылки на этот коммит.

5.6. Часто задаваемые вопросы о Git

В этом разделе представлены конкретные ответы на вопросы, которые часто возникают у пользователей и разработчиков.

Мы используем общепринятое соглашение, согласно которому источником для репозитория FreeBSD является 'freebsd', а не стандартный 'origin', чтобы позволить людям использовать его для собственной разработки и минимизировать случайные отправки (push) в неправильный репозиторий.

5.6.1. Пользователи

5.6.1.1. Как отслеживать -current и -stable, имея только одну копию репозитория?

В: Хотя место на диске не является большой проблемой, эффективнее использовать только одну копию репозитория. При зеркалировании SVN я мог извлекать несколько деревьев из одного и того же репозитория. Как мне сделать это с помощью Git?

О: Вы можете использовать рабочие деревья Git. Существует несколько способов сделать это, но самый простой — использовать клон для отслеживания -current и рабочее дерево для отслеживания стабильных выпусков. Хотя использование «голого (bare) репозитория» предлагалось как способ справиться с этим, это более сложно и здесь документироваться не будет.

Сначала необходимо клонировать репозиторий FreeBSD, здесь показано клонирование в `freebsd-current` для избежания путаницы. `$URL` — это любой зеркальный сервер, который работает для вас наилучшим образом:

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

после того, как он будет клонирован, вы можете просто создать рабочее дерево из него:

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

это извлечёт `stable/12` в каталог с именем `freebsd-stable-12`, который находится на одном уровне с каталогом `freebsd-current`. После создания он обновляется очень похожим образом, как вы могли бы ожидать:

```
% 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
```

Я рекомендую использовать `--ff-only`, так как это безопаснее и позволяет избежать случайного попадания в 'кошмар слияния', когда в вашем дереве появляются дополнительные изменения, вынуждающие выполнять сложное слияние вместо простого.

Вот хорошая статья, в которой рассматривается этот вопрос более подробно.

5.6.2. Разработчики

5.6.2.1. Ой! Я закоммитил в `main` вместо другой ветки.

В: Время от времени я ошибаюсь и по ошибке делаю коммит в ветку `main`. Что мне делать?

О: Во-первых, не паникуйте.

Во-вторых, не отправляйте (push) изменения. На самом деле, можно исправить почти всё, если изменения не были отправлены. Все ответы в этом разделе предполагают, что отправки не произошло.

Следующий ответ предполагает, что вы закоммитили в `main` и хотите создать ветку с названием `issue`:

```
% git checkout -b issue              # Create the 'issue' branch
% git checkout -B main freebsd/main  # Reset main to upstream
% git checkout issue                 # Back to where you were
```

5.6.2.2. Ой! Я закоммитил что-то не в ту ветку!

В: Я работал над функцией в ветке `wilma`, но случайно сделал коммит изменению, относящемуся к ветке `fred`, в 'wilma'. Что мне делать?

О: Ответ аналогичен предыдущему, но с использованием выборочного применением коммитов (cherry-pick). Предполагается, что в ветке wilma имеется всего один коммит, однако подход можно обобщить и для более сложных ситуаций. Также предполагается, что это последний коммит в ветке wilma (отсюда использование wilma в команде `git cherry-pick`), но и это можно обобщить.

```
# 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
```

Если это не последний коммит, вы можете выборочно применить (cherry-pick) это одно изменение из wilma к fred, затем использовать `git rebase -i`, чтобы удалить изменение из wilma.

```
# Мы находимся на ветке wilma
% git checkout fred         # перейти на ветку fred
% git cherry-pick HASH_OF_CHANGE    # скопировать ошибочно размещённый коммит
% git rebase -i main wilma      # удалить скопированное изменение
```

В: Но что, если я хочу сделать коммит нескольким изменениям в `main`, но оставить остальные в `wilma` по какой-то причине?

О: Тот же метод, описанный выше, также работает, если вы хотите «приземлить» части ветки, над которой работаете, в `main` до того, как вся ветка будет готова (скажем, вы заметили несвязанную опечатку или исправили случайную ошибку). Вы можете выборочно применить (cherry-pick) эти изменения в `main`, а затем отправить их в родительский репозиторий. После того, как вы это сделаете, очистка не может быть проще: просто выполните `git rebase -i`. Git заметит, что вы это сделали, и автоматически пропустит общие изменения (даже если вам пришлось изменить сообщение коммита или слегка подкорректировать коммит). Нет необходимости переключаться обратно на wilma, чтобы её поправить: просто делайте rebase!

В: Я хочу выделить некоторые изменения из ветки `wilma` в ветку `fred`

О: Более общий ответ будет таким же, как и предыдущий. Вы должны выгрузить(checkout)/создать ветку `fred`, выборочно применить (cherry-pick) нужные изменения из ветки `wilma` по одному, а затем перебазировать (rebase) `wilma`, чтобы удалить те изменения, которые вы выборочно применили. `git rebase -i main wilma` перенесёт вас в редактор, где нужно удалить строки `pick`, соответствующие коммитам, которые вы скопировали в `fred`. Если всё пройдёт хорошо и не будет конфликтов, вы закончили. Если нет, вам нужно будет разрешать конфликты по мере их появления.

Другой способ сделать это — выгрузить `wilma`, а затем создать ветку `fred`, указывающую на ту же точку в дереве. Затем вы можете выполнить `git rebase -i` для обеих этих веток, выбирая изменения, которые вы хотите видеть в `fred` или `wilma`, оставляя строки с `pick` и удаляя остальные в редакторе. Некоторые предпочитают создать тег/ветку с названием `pre-split` перед началом, на случай если что-то пойдет не так при разделении. Вы можете отменить это следующей последовательностью:

```
% 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
```

Последний шаг необязателен. Если вы собираетесь повторить попытку разделения, его можно пропустить.

В: Но я делал всё по мере прочтения и не увидел ваш совет в конце создать ветку, и теперь `fred` и `wilma` полностью испорчены. Как мне найти, чем `wilma` была до того, как я начал. Я не знаю, сколько раз я всё переставлял.

О: Не всё потеряно. Вы можете разобраться с этим, если прошло не слишком много времени или не было сделано слишком много коммитов (сотни).

Итак, я создал ветку wilma и закоммитил в неё пару изменений, затем решил разделить её на fred и wilma. Ничего странного при этом не произошло, но предположим, что произошло. Способ посмотреть, что вы сделали, — это использовать `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
...
%
```

Здесь мы видим изменения, которые я внес. Вы можете использовать это, чтобы понять, где что-то пошло не так. Я лишь укажу на несколько моментов. Первый из них — HEAD@{X} является 'коммитоподобной' сущностью, поэтому вы можете использовать это в качестве аргумента команды. Хотя если эта команда вносит что-либо в репозиторий, номера X изменяются. Вы также можете использовать хэш (первый столбец).

Далее, 'Encourage contributions' был последним коммитом, который я сделал в `wilma` перед тем, как решил разделить всё. Вы также можете видеть, что тот же хэш присутствует, когда я создал ветку `fred` для этого. Я начал с перебазирования `fred`, и вы видите 'начало', каждый шаг и 'завершение' этого процесса. Хотя нам это здесь не нужно, вы можете точно понять, что произошло. К счастью, чтобы исправить это, вы можете выполнить шаги из предыдущего ответа, но с хэшем `869cbd3` вместо `pre-split`. Хотя это кажется немного многословным, это легко запомнить, поскольку вы делаете одно действие за раз. Вы также можете последовательно применить команды одну за другой:

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

и вы готовы попробовать снова. Команда `checkout -B` с хэшем объединяет извлечение (checkout) и создание ветки для него. Использование `-B` вместо `-b` принудительно перемещает уже существующую ветку. В любом случае это работает, что и прекрасно (и ужасно) в Git. Одна из причин, по которой я склонен использовать `git checkout -B xxxx хэш` вместо извлечения (checkout) хэша, а затем создания / перемещения ветки, — это чисто чтобы избежать слегка тревожного сообщения об отделённом HEAD:

```
% 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
```

это производит тот же эффект, но мне приходится читать гораздо больше, а отрубленные головы (прим перев.: detached HEAD — отрубленная голова) — не тот образ, который мне нравится созерцать.

5.6.2.3. Ой! Я выполнил `git pull`, и это создало коммит слияния, что мне делать?

В: Я действовал на автопилоте и сделал `git pull` для своего дерева разработки, что создало коммит слияния в `main`. Как мне восстановиться?

О: Это может произойти, когда вы выполняете извлечение (checkout) с выбранной веткой разработки.

Многие разработчики используют `git pull --rebase`, чтобы избежать этой ситуации.

Сразу после получения и слияния (pull) у вас в рабочую копию будет извлечен новый коммит слияния. Git поддерживает синтаксис `HEAD^#` для определения родителей коммита слияния:

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

Из этих журналов вы можете легко определить, какой коммит является вашей разработкой. Затем вы просто сбрасываете свою ветку к соответствующему `HEAD^#`:

```
git reset --hard HEAD^1
```

Кроме того, команда `git pull --rebase` на этом этапе перебазирует ваши изменения из ветки 'main' на последнюю версию 'freebsd/main'.

В: Но мне также нужно исправить мою ветку `main`. Как мне это сделать?

О: Git отслеживает ветки удалённого репозитория в пространстве имён `freebsd/`. Чтобы исправить вашу ветку `main`, просто заставьте её указывать на удалённую ветку `main`:

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

Ветви в Git не имеют ничего магического: они всего лишь метки на графе, которые автоматически перемещаются вперёд при создании коммитов. Так что вышеописанное работает, потому что вы просто перемещаете метку. Из-за этого нет никаких метаданных о ветке, которые нужно было бы сохранять.

5.6.2.4. Смешивание и сопоставление веток

В: Итак, у меня есть две ветки `worker` и `async`, которые я хочу объединить в одну ветку под названием `feature` с сохранением коммитов в обеих.

О: Это задача для выборочного применения (cherry-pick).

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

Теперь у вас есть новая ветка под названием `feature`. Эта ветка объединяет коммиты из обеих веток. Вы можете далее работать с ней с помощью `git rebase`.

В: У меня есть ветка под названием `driver`, и я хочу разделить её на `kernel` и `userland`, чтобы развивать их отдельно и коммитить каждую ветку по мере её готовности.

О: Это требует небольшой подготовительной работы, но `git rebase` выполнит здесь основную часть работы.

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

Теперь у вас есть две идентичные ветки. Значит, пришло время разделить коммиты. Мы предположим, что сначала все коммиты в ветке `driver` попадут либо в ветку `kernel`, либо в ветку `userland`, но не в обе одновременно.

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

и просто включите изменения, которые вы хотите (строкой 'p' или 'pick'), и удалите коммиты, которые не нужны (это звучит пугающе, но в худшем случае вы всегда можете всё отбросить и начать заново с ветки `driver`, если вы только её не переместили).

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

и сделайте то же самое, что вы сделали с веткой `kernel `.

В: Отлично! Я выполнил указанные выше шаги и забыл сделать коммит в ветке `kernel`. Как мне восстановиться?

О: Вы можете использовать ветку `driver`, чтобы найти хэш коммита, который отсутствует, и выборочно применить его (cherry-pick).

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

В: Хорошо. У меня такая же ситуация, как и выше, но мои коммиты все перемешаны. Мне нужно, чтобы части одного коммита попали в одну ветку, а остальные — в другую. На самом деле, у меня их несколько. Ваш метод перебазирования с выбором кажется сложным.

О: В этой ситуации вам лучше обработать исходную ветку, чтобы отделить коммиты, а затем использовать вышеуказанный метод для разделения ветки.

Итак, предположим, что есть всего один коммит с чистым деревом. Вы можете использовать либо `git rebase` со строкой `edit`, либо это с коммитом на острие. В любом случае шаги одинаковы. Первое, что нам нужно сделать, — это откатиться на один коммит назад, оставив изменения незакоммиченными в дереве:

```
% git reset HEAD^
```

Примечание: НЕ, повторяю, НЕ добавляйте здесь `--hard`, так как это также удалит изменения из вашего дерева.

Теперь, если вам повезёт, изменения, которые нужно разделить, полностью укладываются по границам файлов. В этом случае вы можете просто выполнить обычный `git add` для файлов в каждой группе, а затем сделать `git commit`. Примечание: при этом вы потеряете сообщение коммита при выполнении сброса, поэтому если оно вам по какой-то причине нужно, следует сохранить копию (хотя `git log $HASH` может его восстановить).

Если вам не повезло, вам придётся разделять файлы. Для этого существует ещё один инструмент, который можно применять по одному файлу за раз.

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

будет последовательно проходить через различия, предлагая вам включить или исключить каждый фрагмент. После завершения, выполните `git commit`, и оставшиеся изменения окажутся в вашем дереве. Вы также можете запускать эту команду несколько раз, даже для нескольких файлов (хотя я считаю удобнее работать с одним файлом за раз и использовать `git rebase -i` для объединения связанных коммитов).

5.6.2.5. Присоединение к организации FreeBSD на GitHub.

В: Как присоединиться к организации FreeBSD на GitHub?

О: Подробности смотрите на странице нашей вики GitHub. Вкратце, присоединиться могут все коммиттеры FreeBSD. Лица, не являющиеся коммиттерами, которые запрашивают присоединение, будут рассматриваться в индивидуальном порядке.

5.6.3. Клонирование и зеркалирование

В: Я хочу создать полную зеркальную копию репозитория Git, как мне это сделать?

О: Если вам нужно только зеркалирование, то

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

сработает. Однако, у этого есть два недостатка, если вы хотите использовать это для чего-то кроме зеркала, которое вы будете переклонировать.

Во-первых, это 'голый репозиторий', который содержит базу данных репозитория, но не имеет извлеченного рабочего дерева. Это отлично подходит для зеркалирования, но ужасно для повседневной работы. Существует несколько способов обойти это с помощью `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
```

Но если вы не используете свой зеркальный репозиторий для дальнейшего локального клонирования, то это неподходящий выбор.

Второй недостаток заключается в том, что Git обычно перезаписывает ссылки из вышестоящего репозитория (названия веток, теги и т.д.) , чтобы ваши локальные ссылки могли изменяться независимо от вышестоящего репозитория. Это означает, что вы потеряете изменения, если будете коммитить в свой репозиторий куда-либо, кроме веток приватных проектов.

В: Так что же я могу сделать вместо этого?

О: Ну, вы можете поместить все ссылки (refs) вышестоящего репозитория в приватное пространство имён вашего локального репозитория. Git клонирует всё через 'refspec', и refspec по умолчанию выглядит так:

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

что говорит просто получить (fetch) ссылки веток.

Однако в репозитории FreeBSD есть и ряд других элементов. Чтобы увидеть их, вы можете добавить явные спецификации ссылок для каждого пространства имён ссылок или

получить всё. Чтобы настроить ваш репозиторий для этого:

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

что поместит всё из вышестоящего репозитория в пространство имён `refs/freebsd/` вашего локального репозитория. Обратите внимание, что это также захватывает все несконвертированные ветки вендоров, а количество связанных с ними ссылок довольно велико.

Вам потребуется ссылаться на эти ссылки с их полными именами, поскольку они не входят в обычные пространства имён Git.

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

будет просматривать журнал ветки вендора для zlib, начиная с версии 1.2.10.

5.7. Сотрудничество с другими

Одним из ключевых моментов качественной разработки программного обеспечения в таком крупном проекте, как FreeBSD, является возможность сотрудничать с другими участниками до отправки своих изменений в дерево исходного кода. Репозитории Git проекта FreeBSD пока не позволяют отправлять пользовательские ветки в репозиторий, поэтому если вы хотите поделиться своими изменениями с другими, вам необходимо использовать другой механизм, например, репозиторий, размещённый в GitLab или GitHub, для обмена изменениями в ветке, созданной пользователем.

Следующие инструкции показывают, как создать пользовательскую ветку на основе ветки FreeBSD `main` и отправить её в GitHub.

Прежде чем начать, убедитесь, что ваш локальный репозиторий Git актуален и имеет правильно настроенные источники, как показано в разделе выше.

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

Первым шагом является создание форка FreeBSD на GitHub, следуя этим инструкциям. Назначением форка должен быть ваш собственный, личный аккаунт на GitHub (в моём случае gvnn3).

Теперь добавьте удалённый репозиторий в вашей локальной системе, который указывает на ваш форк:

```
% 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)
```

В этом репозитории вы можете создать ветку как показано выше.

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

Вносите любые изменения в своей ветке. Собирайте, тестируйте, и как только будете готовы к совместной работе с другими, настанет время отправить свои изменения в ветку, размещённую в GitHub. Прежде чем отправить изменения, вам нужно будет установить соответствующую вышестоящую ветку (upstream), о чём Git сообщит при первой попытке отправки в ваш удалённый репозиторий 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
```

Если установить параметры отправки (push) так, как советует git, то это позволяет ей успешно выполниться:

```
% 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'.
```

Последующие изменения в той же ветке будут по умолчанию отправляться корректно:

```
% 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
```

На этом этапе ваша работа находится в вашей ветке на GitHub, и вы можете поделиться ссылкой с другими участниками.

5.8. Обработка запросов на принятие изменений (pull request) на github

В этом разделе описано, как интегрировать запрос на принятие изменений (pull request) из GitHub, отправленный на зеркала FreeBSD в Git на GitHub. Хотя на данный момент это не официальный способ отправки исправлений, иногда таким образом приходят хорошие правки, и проще всего просто перенести их в дерево коммиттера и оттуда отправить (push) в дерево FreeBSD. Аналогичные шаги можно использовать для получать и сливать (pull) ветки из других репозиториев и их использовать их. При коммите запросов на принятие изменений от других следует проявлять особую осторожность, чтобы проверить все изменения и убедиться, что они в точности соответствуют заявленным.

Прежде чем начать, убедитесь, что локальный репозиторий Git актуален и имеет правильно настроенные источники, как показано в разделе выше. Кроме того, убедитесь, что у вас есть следующие источники:

```
% 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)
```

Часто запросы на принятие изменений просты: это запросы, содержащие всего один коммит. В этом случае можно использовать упрощённый подход, хотя подход из предыдущего раздела также будет работать. Здесь создаётся ветка, изменения выборочно применяются, сообщение коммита корректируется и проверяется на адекватность перед отправкой. В этом примере используется ветка `staging`, но она может иметь любое имя. Эта техника работает для любого количества коммитов в запросе на принятие изменений, особенно когда изменения чисто применяются к дереву FreeBSD. Однако, когда коммитов несколько, особенно когда требуются незначительные корректировки, `git rebase -i` работает лучше, чем `git cherry-pick`. Вкратце, эти команды создают ветку; выборочно применяют изменения из запроса на принятие изменений; тестируют их; корректируют сообщения коммитов; и выполняют слияние перемоткой (fast-forward) обратно в `main`. Номер PR ниже обозначен как `$PR`. При корректировке сообщения добавьте `Pull Request: https://github.com/freebsd-src/pull/$PR`. Все запросы на принятие изменений, зафиксированные в репозитории FreeBSD, должны быть проверены как минимум одним человеком. Это не обязательно должен быть тот, кто их фиксирует, но в этом случае тот, кто

фиксирует, должен доверять компетентности других рецензентов в проверке коммита. Коммиттеры, которые проводят рецензирование кода у запросов на принятие изменений перед их отправкой в репозиторий, должны добавить строку `Reviewed by:` к коммиту, потому что в этом случае это не подразумевается. Добавьте всех, кто просматривает и одобряет коммит на github, также в `Reviewed by:`. Как всегда, следует позаботиться о том, чтобы изменение делало то, что предполагается, и чтобы в нём не было вредоносного кода.

Кроме того, пожалуйста, убедитесь, что имя автора запроса на принятие изменений не является анонимным. Веб-интерфейс редактирования Github генерирует имена вида:

```
Author:     github-user <38923459+github-user@users.noreply.github.com>
```

Автору следует отправить вежливую просьбу предоставить более подходящее имя и/или адрес электронной почты. Следует проявлять особую осторожность, чтобы не допустить ошибок стиля или внедрения вредоносного кода.

```
% 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
```

Для сложных запросов на принятие изменений с несколькими коммитами, содержащими конфликты, следуйте приведённой ниже схеме.

1. извлеките запрос на принятие изменений `git checkout github/pull/XXX`
2. создайте ветку для перебазирования `git checkout -b staging`
3. перебазируйте ветку `staging` на последнюю версию `main` с помощью `git rebase -i main staging`
4. разрешите конфликты и проведите все необходимые тестирования
5. перемотайте (fast-forward) ветку `staging` в `main`, как описано выше
6. сделайте финальную проверку изменений, чтобы убедиться, что всё в порядке
7. отправьте в репозиторий Git FreeBSD.

Это также будет работать при внесении веток, разработанных в другом месте, в локальное дерево для коммита.

После завершения работы с запросом на принятие изменений (pull request), закройте его с помощью веб-интерфейса GitHub. Стоит отметить, что если ваш источник `github` использует `https://`, единственным шагом, для которого потребуется учётная запись GitHub, будет закрытие запроса на принятие изменений.

6. История системы контроля версий

Проект перешёл на git.

Репозиторий исходных кодов FreeBSD перешёл с CVS на Subversion 31 мая 2008 года. Первый настоящий коммит SVN — *r179447*. Репозиторий исходных кодов перешёл с Subversion на Git 23 декабря 2020 года. Последний настоящий коммит svn — *r368820*. Хеш первого настоящего коммита git — *5ef5f51d2bef80b0ede9b10ad5b0e9440b60518c*.

Репозиторий `doc/www` FreeBSD перешёл с CVS на Subversion 19 мая 2012 года. Первый настоящий коммит SVN — *r38821*. Репозиторий документации перешёл с Subversion на Git 8 декабря 2020 года. Последний коммит SVN — *r54737*. Первый настоящий хэш коммита git — *3be01a475855e7511ad755b2defd2e0da5d58bbe*.

Репозиторий `ports` FreeBSD перешел с CVS на Subversion 14 июля 2012 года. Первый настоящий коммит SVN — *r300894*. Репозиторий ports перешел с Subversion на Git 6 апреля 2021 года. Последний коммит SVN — *r569609*. Первый настоящий хэш коммита git — *ed8d3eda309dd863fb66e04bccaa513eee255cbf*.

7. Настройка, соглашения и традиции

В качестве нового разработчика необходимо выполнить ряд действий. Первый набор шагов относится исключительно к коммиттерам. Эти шаги должны быть выполнены наставником для тех, кто не является коммиттером.

7.1. Для новых коммиттеров

Те, кому предоставлены права на коммит в репозитории FreeBSD, должны выполнить следующие шаги.

- Получите одобрение наставника перед внесением каждого из этих изменений!
- Все коммиты в src сначала попадают в FreeBSD-CURRENT, прежде чем быть слитыми в FreeBSD-STABLE. Ветка FreeBSD-STABLE должна сохранять совместимость ABI и API с предыдущими версиями этой ветки. Не делайте слияние изменений, нарушающих эту совместимость.

Шаги для новых коммиттеров

1. Добавить автора

   doc/shared/authors.adoc - Добавить информацию об авторе. Последующие шаги зависят от этой информации, и пропуск этого шага приведёт к сбою сборки doc/. Это

относительно простая задача, но она остаётся хорошим первым испытанием навыков работы с системой контроля версий.

2. Обновить список разработчиков и участников

   doc/shared/contrib-committers.adoc - Добавьте запись, которая затем появится в разделе "Разработчики" Списка контрибьюторов. Записи сортируются по фамилии.

   doc/shared/contrib-additional.adoc - *Удалить* запись. Записи отсортированы по имени.

3. Добавить статью в Новости

   doc/website/data/en/news/news.toml - Добавьте запись. Найдите другие записи, объявляющие о новых коммиттерах, и следуйте формату. Используйте дату из письма об утверждении коммиттерских прав.

4. Добавить PGP-ключ

   `Dag-Erling Smørgrav <des@FreeBSD.org>` написал сценарий оболочки (doc/documentation/tools/addkey.sh) для упрощения этого процесса. Для получения дополнительной информации обратитесь к файлу README.

   Используйте doc/documentation/tools/checkkey.sh для проверки, что ключи соответствуют как минимум минимальным стандартам лучших практик.

   После добавления и проверки ключа добавьте оба обновленных файла в систему контроля версий и затем зафиксируйте (commit) их. Записи в этом файле отсортированы по фамилии.

   

   Очень важно иметь актуальный PGP/GnuPG ключ в репозитории. Ключ может потребоваться для подтверждения личности коммиттера. Например, `Администраторы FreeBSD.org <admins@FreeBSD.org>` используют его для восстановления учётной записи. Полный набор ключей пользователей `FreeBSD.org` доступен для скачивания по ссылке https://docs.FreeBSD.org/pgpkeys/pgpkeys.txt.

5. Обновить информацию о наставнике и подопечном

   src/share/misc/committers-<repository>.dot - Добавить запись в раздел текущих коммиттеров, где *repository* - это `doc`, `ports` или `src`, в зависимости от предоставленных прав коммита.

   Добавьте запись для каждого дополнительного отношения наставник/подопечный в нижнем разделе.

6. Обновить файл mailmap в git

   src/.mailmap, doc/.mailmap и ports/.mailmap - Добавьте запись для коммитов, созданных вами до получения статуса коммиттера FreeBSD.

Привязка к вашему адресу FreeBSD позволяет нам проще отслеживать внешних коммиттеров, которые могут быть готовы получить права коммиттера. Вы также можете использовать это для исправления старых имен, опечаток в именах и т.д. в стандартном выводе `git log`.

7. Сгенерировать пароль Kerberos

   См. Kerberos и LDAP веб-пароль для кластера FreeBSD для генерации или настройки учётной записи Kerberos для использования с другими сервисами FreeBSD, такими как база данных отслеживания ошибок (вы получаете учётную запись для отслеживания ошибок как часть этого шага).

8. Необязательно: Включить учётную запись Вики

   Учётная запись FreeBSD Wiki — учётная запись на вики позволяет делиться проектами и идеями. Те, у кого ещё нет учётной записи, могут следовать инструкциям на странице Wiki/About, чтобы её получить. Свяжитесь с wiki-admin@FreeBSD.org, если вам нужна помощь с вашей учётной записью на Вики.

9. Необязательно: Обновить информацию в Вики

   Информация в вики — получив доступ к вики, некоторые добавляют записи на страницы Как мы сюда попали, IRC-псевдонимы, Собаки FreeBSD и/или Кошки FreeBSD.

10. Необязательно: Обновить порты с личной информацией

    ports/astro/xearth/files/freebsd.committers.markers и src/usr.bin/calendar/calendars/calendar.freebsd — Некоторые люди добавляют в эти файлы записи о себе, чтобы указать своё местоположение или дату своего дня рождения.

11. Необязательно: Предотвращение повторных рассылок

    Подписчики Коммиты во все ветки репозитория документации, Коммиты во все ветки репозитория портов или Коммиты во все ветки репозитория исходного кода, возможно, захотят отписаться, чтобы избежать получения дубликатов сообщений о коммитах и ответов на них.

7.2. Для всех

1. Представьтесь другим разработчикам, иначе никто не будет иметь представления, кто вы и над чем работаете. Представление не должно быть исчерпывающей биографией — просто напишите абзац или два о том, кто вы, чем планируете заниматься как разработчик в FreeBSD, и кто будет вашим наставником. Отправьте это по электронной почте на Список рассылки разработчиков FreeBSD, и вы начнёте свой путь!
2. Войдите на `freefall.FreeBSD.org` и создайте файл /var/forward/пользователь (где

пользователь — это ваше имя пользователя), содержащий адрес электронной почты, на который должны перенаправляться письма, адресованные на *вашеимяпользователя*@FreeBSD.org. Это включает все сообщения о коммитах, а также любую другую почту, адресованную Список рассылки коммиттеров FreeBSD и Список рассылки разработчиков FreeBSD. Очень большие почтовые ящики, которые заняли постоянное место на `freefall`, могут быть усечены без предупреждения, если потребуется освободить место, поэтому перенаправляйте или сохраняйте их в другом месте.

Если ваша система электронной почты использует SPF со строгими правилами, вам следует исключить `mx2.FreeBSD.org` из проверок SPF.

Из-за высокой нагрузки, которую обработка спама создаёт на центральных почтовых серверах, обрабатывающих почтовые рассылки, фронтенд-сервер выполняет базовые проверки и может отбрасывать некоторые сообщения на их основе. В настоящее время единственной активной проверкой является наличие корректной DNS-информации для подключающегося хоста, но это может измениться. Некоторые пользователи связывают эти проверки с ложным отбрасыванием легитимной почты. Для отключения данных проверок для вашей почты создайте файл с именем ~/.spam_lover на `freefall.FreeBSD.org`.

Те, кто являются разработчиками, но не коммиттерами, не будут подписаны на рассылки коммиттеров или разработчиков. Подписки определяются правами доступа.

7.2.1. Настройка доступа SMTP

Для тех, кто желает отправлять электронные письма через инфраструктуру FreeBSD.org, следуйте приведённым ниже инструкциям:

1. Направьте ваш почтовый клиент на `smtp.FreeBSD.org:587`.
2. Включить STARTTLS.
3. Убедитесь, что ваш адрес `From:` установлен как `вашеимяпользователя@FreeBSD.org`.
4. Для аутентификации можно использовать ваше имя пользователя и пароль FreeBSD Kerberos (см. Kerberos и LDAP веб-пароль для кластера FreeBSD). Предпочтительнее использовать принципал `вашеимяпользователя/mail`, так как он действителен только для аутентификации к почтовым ресурсам.

При вводе имени пользователя не включайте `@FreeBSD.org`.

Дополнительные заметки

- Будет принимать почту только от `вашеимяпользователя@FreeBSD.org`. Если вы аутентифицированы как один пользователь, вам не разрешено отправлять почту от другого.

- Будет добавлен заголовок сообщения с именем пользователя SASL: (`Authenticated sender: имя_пользователя`).
- На хосте действуют различные ограничения по скорости для сокращения попыток взлома перебором паролей.

7.2.1.1. Использование локального MTA для пересылки электронной почты в SMTP-сервис FreeBSD.org

Также возможно использовать локальный MTA для пересылки локально отправленных писем на SMTP-серверы FreeBSD.org.

Пример 1. Использование Postfix

Чтобы сообщить локальному экземпляру Postfix, что любое письмо от `вашеимяпользователя@FreeBSD.org` должно быть перенаправлено на серверы FreeBSD.org, добавьте это в ваш 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
```

Создайте файл /usr/local/etc/postfix/relayhost_maps со следующим содержимым:

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

Создайте /usr/local/etc/postfix/sasl_passwd со следующим содержимым:

```
[smtp.freebsd.org]:587          вашеимяпользователя:вашпароль
```

Если почтовый сервер используется другими людьми, вы можете захотеть предотвратить отправку ими писем с вашего адреса. Для достижения этой цели добавьте это в ваш main.cf:

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

Создайте файл /usr/local/etc/postfix/sender_login_maps со следующим содержимым:

```
вашеимяпользователя@FreeBSD.org вашеимялокальногопользователя
```

Где *вашеимялокальногопользователя* — это имя пользователя SASL, используемое для

подключения к локальному экземпляру Postfix.

Пример 2. Использование OpenSMTPD

Чтобы указать локальному экземпляру OpenSMTPD, что все письма от вашеимяпользователя@FreeBSD.org должны быть перенаправлены на серверы FreeBSD.org, добавьте это в ваш smtpd.conf:

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

Где *вашеимялокальногопользователя* — это имя пользователя SASL, используемое для подключения к локальному экземпляру OpenSMTPD.

Создайте файл /usr/local/etc/mail/secrets со следующим содержимым:

```
freebsd вашеимяпользователя:вашпароль
```

Пример 3. Использование Exim

Чтобы направить локальный экземпляр Exim для пересылки всей почты от example@FreeBSD.org на серверы FreeBSD.org, добавьте это в конфигурацию 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:
freebsd_plain:
  driver = plaintext
  public_name = PLAIN
  client_send = ^example/mail^examplePassword
  client_condition = ${if eq{$host}{smtp.freebsd.org}}
```

Создайте файл /usr/local/etc/exim/freebsd_send со следующим содержимым:

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

7.3. Наставники

Все новые разработчики получают наставника на первые несколько месяцев. Наставник отвечает за обучение подопечного правилам и соглашениям проекта и направляет его первые шаги в сообществе разработчиков. Наставник также несёт личную ответственность за действия подопечного в течение этого начального периода.

Для коммиттеров: не коммитьте ничего без предварительного одобрения ментора. Задокументируйте это одобрение строкой `Approved by:` в сообщении коммита.

Когда наставник решает, что подопечный освоил основы и готов к самостоятельной фиксации изменений, наставник объявляет об этом, выполняя коммит в mentors. Этот файл находится в сиротской ветке admin каждого репозитория. Подробная информация о том, как получить доступ к этим веткам, доступна в ветке "admin".

8. Предварительная проверка перед коммитом

Рецензирование кода — один из способов повышения качества программного обеспечения. Следующие рекомендации применимы к коммитам в ветку `main` (-CURRENT) репозитория `src`. Другие ветки, а также деревья `ports` и `docs` имеют собственные политики проверки и рецензирования, но данные рекомендации в целом применимы к коммитам, требующим рецензии:

- Все нетривиальные изменения должны быть проверены перед их фиксацией в репозитории.
- Рецензирование может проводиться по электронной почте, в Bugzilla, в Phabricator или с помощью другого механизма. По возможности рецензирование должно быть публичным.
- Разработчик, ответственный за изменение кода, также обязан вносить все необходимые изменения, связанные с проверкой.

- Рецензирование кода может быть итеративным процессом, который продолжается до тех пор, пока патч не будет готов к коммиту. В частности, после отправки патча на рецензирование, он должен получить явное подтверждение "выглядит хорошо" перед коммитом. Оно должно быть явным, и это может принимать любую форму, которая имеет смысл для метода рецензирования.
- Тайм-ауты не являются заменой проверке.

Иногда проверка кода занимает больше времени, чем хотелось бы, особенно для большого по объёму функционала. Принятые способы ускорить проверку ваших патчей:

- Проверяйте патчи других людей. Если вы помогаете, все будут более охотно делать то же самое для вас; доброжелательность — наша валюта.
- Пингуйте патч. Если он срочный, укажите причины, почему для вас важно, чтобы этот патч был принят, и пингуйте каждые пару дней. Если он не срочный, общепринятая вежливая частота пинга — одна неделя. Помните, что вы просите у других профессиональных разработчиков их ценное время.
- Обратитесь за помощью в списки рассылки, IRC и т.д. Другие могут либо помочь вам напрямую, либо предложить рецензента.
- Разделите ваш патч на несколько меньших патчей, которые основываются друг на друге. Чем меньше ваш патч, тем выше вероятность, что кто-то бегло его просмотрит.

  При внесении крупных изменений полезно держать это в уме с самого начала работы, поскольку разбиение крупных изменений на более мелкие часто бывает затруднительно постфактум.

Разработчикам следует участвовать в проверках кода как в роли авторов, так и в роли рецензентов. Если кто-то любезно проверил ваш код, вы должны ответить тем же для кого-то другого. Обратите внимание, что хотя любой может проверить и дать обратную связь по патчу, только соответствующий эксперт по теме может одобрить изменение. Обычно это коммиттер, который регулярно работает с рассматриваемым кодом.

В некоторых случаях может не оказаться эксперта по предметной области. В таких случаях достаточно проверки опытным разработчиком в сочетании с соответствующим тестированием.

9. Журнал сообщений о коммитах

В этом разделе содержатся некоторые предложения и традиции по форматированию журналов коммитов.

9.1. Почему важны сообщения коммитов?

При фиксации изменения в Git, Subversion или другой системе контроля версий (СКВ) вам предлагается написать текст с описанием коммита — сообщение о коммите. Насколько важно это сообщение о коммите? Стоит ли прилагать значительные усилия для его написания? Имеет ли значение, если вы просто напишете `исправлена ошибка`?

У большинства проектов более одного разработчика, и они длятся в течение некоторого времени. Сообщения коммитов — это очень важный способ общения с другими разработчиками, как в настоящем, так и в будущем.

В FreeBSD сотни активных разработчиков и сотни тысяч коммитов, охватывающих десятилетия истории. За это время сообщество разработчиков осознало, насколько ценны хорошие сообщения к коммитам; иногда эти уроки давались тяжело.

Сообщения коммитов служат как минимум трем целям:

- Сотрудничество с другими

  Коммиты FreeBSD генерируют письма для различных списков рассылки. Они включают сообщение коммита вместе с копией самого патча. Сообщения коммитов также просматриваются с помощью команд, таких как `git log`. Это служит для информирования других разработчиков об изменениях, которые происходят; другой разработчик может захотеть протестировать изменение, может быть заинтересован в теме и захочет просмотреть более подробно, или может иметь свои собственные проекты, которые выиграют от взаимодействия.

- Обеспечение возможности обнаружения изменений

  В большом проекте с долгой историей может быть сложно найти интересующие изменения при расследовании проблемы или изменения в поведении. Подробные, детальные сообщения о коммитах позволяют искать изменения, которые могут быть релевантны. Например, `git log --since 1year --grep 'USB timeout'`.

- Предоставление исторической документации

  Сообщения о фиксации служат для документирования изменений для будущих разработчиков, возможно, спустя годы или десятилетия. Этот будущий разработчик может оказаться даже вами, первоначальным автором. Изменение, которое кажется очевидным сегодня, может оказаться совсем не таким в будущем.

Команда `git blame` аннотирует каждую строку исходного файла информацией о изменении (хэш и тема коммита), которое её добавило.

Теперь, когда важность хорошего сообщения о коммите в FreeBSD несомненна, вот его элементы:

9.2. Начните со строки темы

Сообщения о коммите должны начинаться с однострочной темы, кратко описывающей изменение. Сама по себе тема должна позволять читателю быстро определить, представляет ли изменение интерес.

9.3. Сохраняйте заголовки краткими

Строка темы должна быть максимально короткой, но при этом сохранять необходимую

информацию. Это повышает эффективность просмотра журнала Git и позволяет команде `git log --oneline` отображать короткий хэш и тему на одной 80-символьной строке. Хорошим эмпирическим правилом является удержание длины ниже 67 символов, а по возможности — около 50 или меньше.

9.4. Добавьте к строке темы префикс с указанием компонента, если это применимо

Если изменение относится к определённому компоненту, строка темы может быть предварена именем этого компонента и двоеточием (:). По возможности используйте тот же префикс, который применялся в предыдущих коммитах к тем же файлам.

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

Включите префикс в лимит 67 символов, чтобы `git log --oneline` избегал переноса.

9.5. Напишите первую букву темы с заглавной буквы

Первая буква темы должна быть заглавной. Префикс, если он есть, с заглавной буквы не пишется, если это не требуется (например, `USB:` пишется с заглавной буквы).

9.6. Не заканчивайте строку темы знаками препинания

Не ставьте точку или другие знаки препинания в конце. В этом отношении строка темы подобна заголовку в газете.

9.7. Разделите тему и тело письма пустой строкой

Отделите тело от темы пустой строкой.

Некоторые тривиальные коммиты не требуют тела и содержат только заголовок.

✓ `ls: Fix typo in usage text`

9.8. Ограничьте сообщения до 72 колонок

`git log` и `git format-patch` делают отступ в сообщении коммита на четыре пробела. Перенос строк на 72-й колонке обеспечивает соответствующий отступ по правому краю. Ограничение сообщений 72 символами также удерживает сообщение коммита в форматированных патчах ниже рекомендованного RFC 2822 ограничения длины строки электронной почты в 78 символов. Это ограничение хорошо работает с различными инструментами, которые могут отображать сообщения коммитов; перенос строк может быть непоследовательным при большей длине строки.

9.9. Используйте настоящее время, повелительное наклонение

Это способствует краткости тем и обеспечивает единообразие, включая автоматически генерируемые сообщения коммитов (например, создаваемые `git revert`). Это важно при чтении списка тем коммитов. Думайте о теме как о завершении фразы «при применении это изменение позволит…(when applied, this change will …)».

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

9.10. Сосредоточьтесь на том, что и почему, а не на том, как

Объясните, чего достигает изменение и почему оно делается, а не как.

Не предполагайте, что читатель знаком с проблемой. Объясните предысторию и мотивацию изменения. Включите данные тестирования производительности, если они у вас есть.

Если в изменениях есть ограничения или неполные аспекты, опишите их в сообщении коммита.

9.11. Подумайте, можно ли части сообщения коммита оформить как комментарии в коде

Иногда при написании сообщения коммита вы можете обнаружить, что пишете одно-два предложения, объясняющих какой-то сложный или запутанный аспект изменения. В таких случаях стоит подумать, будет ли полезно иметь это объяснение в виде комментария в самом коде.

9.12. Напишите сообщения коммитов для себя в будущем

При написании сообщения коммита для изменения у вас есть весь контекст в голове — что вызвало изменение, альтернативные подходы, которые рассматривались и были отклонены, ограничения изменения и так далее. Представьте, что вы возвращаетесь к изменению через год или два, и напишите сообщение коммита таким образом, чтобы оно предоставило этот необходимый контекст.

9.13. Сообщения о коммитах должны быть самодостаточными

Вы можете включать ссылки на сообщения в почтовых рассылках, сайты с результатами тестирования производительности или ссылки на проверки кода. Однако, сообщение о коммите должно содержать всю соответствующую информацию на случай, если эти ссылки станут недоступны в будущем.

Аналогично, коммит может ссылаться на предыдущий коммит, например, в случае исправления ошибки или отката. Помимо идентификатора коммита (ревизии или хеша), включите строку темы из упомянутого коммита (или другую подходящую краткую ссылку). С каждой миграцией системы контроля версий (от CVS к Subversion и затем к Git) идентификаторы ревизий из предыдущих систем могут становиться трудными для отслеживания.

9.14. Укажите необходимые метаданные в нижней части

Помимо включения информативного сообщения с каждым коммитом, может потребоваться некоторая дополнительная информация.

Эта информация состоит из одной или нескольких строк, содержащих ключевое слово или фразу, двоеточие, табуляции для форматирования и затем дополнительную информацию.

Для ключевых слов, где допустимы множественные значения (например, `PR:` со списком PR через запятую), разрешается использовать одно и то же ключевое слово несколько раз, чтобы избежать неоднозначности или улучшить читаемость.

Ключевые слова или фразы:

`PR:`	Номер отчёта о проблеме (если есть), на который влияет данный коммит (обычно путем закрытия). Можно указать несколько номеров PR в одной строке, разделяя их запятыми или пробелами.
`Reported by:`	Имя и адрес электронной почты лица, сообщившего о проблеме; для разработчиков — только имя пользователя в кластере FreeBSD. Обычно используется, когда нет PR, например, если проблема была сообщена в почтовой рассылке.
`Submitted by:` (не рекомендуется)	Имя автора, который отправил изменение без предоставления полного корректного патча, особенно без корректного адреса электронной почты. Отправленные патчи должны иметь указанного автора с помощью команды `git commit --author` с полным именем и корректным адресом электронной почты. До перехода на git, когда появилась возможность разделять поля автора и коммиттера, это использовалось для присланных патчей.

`Reviewed by:`	Имя и адрес электронной почты человека или людей, которые проверили изменение; для разработчиков достаточно указать имя пользователя в кластере FreeBSD. Если патч был отправлен в список рассылки для проверки и получил положительный отзыв, то укажите только название списка. Если рецензент не является участником проекта, укажите имя, электронную почту и, если это порт, внешнюю роль, например, сопровождающего: Проверено разработчиком: [source,shell] …. Reviewed by: username …. Проверено сопровождающим портов, не являющимся разработчиком: [source,shell] …. Reviewed by: Full Name <valid@email> (maintainer) ….
`Tested by:`	Имя и адрес электронной почты человека или людей, которые проверили изменение; для разработчиков — просто имя пользователя в кластере FreeBSD.
`Discussed with:`	Имя и адрес электронной почты человека или людей, которые внесли вклад в исправление, предоставив содержательную обратную связь; для разработчиков — просто имя пользователя в кластере FreeBSD. Обычно используется для упоминания тех, кто не проводил явного рецензирования, тестирования или одобрения изменения, но тем не менее участвовал в обсуждении, связанном с изменением, что привело к улучшениям и лучшему пониманию его влияния на проект FreeBSD.

`Approved by:`	Имя и адрес электронной почты лица или лиц, утвердивших изменение; для разработчиков — просто имя пользователя в кластере FreeBSD. Есть несколько случаев, когда утверждение является обычной практикой: • когда новый коммиттер находится под наставничеством • коммиты в область дерева, указанную в файле LOCKS (src) • во время цикла выпуска • коммиты в репозиторий, где у вас нет права на коммит (например, коммиттер src делает коммит в docs) • коммиты в порт, поддерживаемый кем-то другим Во время наставничества получите одобрение наставника перед коммитом. Укажите имя пользователя наставника в этом поле и отметьте, что он является наставником: ``` Approved by: имя-пользователя-наставника (mentor) ``` Если коммиты были утверждены командой, укажите название команды, за которым в скобках следует имя пользователя утверждающего. Например: ``` Approved by: re (имя-пользователя) ```
`Obtained from:`	Название проекта (если есть), из которого был получен код. Не используйте эту строку для указания имени отдельного человека.
`Fixes:`	Короткий хэш Git и заголовок коммита, который исправлен этим изменением, как возвращается командой `git log -n1 --format:'%h ("%s")' GIT-COMMIT-HASH`. Мы включаем заголовок коммита, чтобы можно было найти указанный коммит даже в случае, если будущая миграция системы контроля версий сделает ссылки по хэшу недействительными.
`MFC after:`	Чтобы получить напоминание по электронной почте о запланированном MFC через определённое время, укажите количество дней, недель или месяцев, после которых планируется выполнить MFC.
`MFC to:`	Если коммиту должно быть сделано слияние в подмножество стабильных веток, укажите названия веток.
`MFH:`	Если коммиту должно быть сделано слияние в квартальную ветку портов, укажите квартальную ветку. Например, `2021Q2`.
`Relnotes:`	Если изменение является кандидатом для включения в примечания к выпуску следующей версии из ветки, установите значение `yes`.

Кандидатами являются изменения, видимые пользователям, новые функции, нарушения совместимости и т.д.	Если вы забыли установить эту строку или хотите предоставить более подробную информацию, добавьте запись в файл `RELNOTES` в корне дерева исходного кода src.
Файл `RELNOTES` используется для формирования примечаний к выпуску для следующего релиза.	Не используйте строку `Relnotes:` для описания изменения: её единственное допустимое значение - `yes`.
`Security:`	Если изменение связано с уязвимостью или угрозой безопасности, укажите одну или несколько ссылок либо описание проблемы. По возможности включите URL VuXML или идентификатор CVE.
`Event:`	Описание события, в рамках которого был выполнен этот коммит. Если это повторяющееся событие, добавьте год или даже месяц. Например, это может быть `FooBSDcon 2019`. Идея этой строки — отдать должное конференциям, встречам и другим подобным мероприятиям, а также показать, что их проведение полезно. Пожалуйста, не используйте строку `Sponsored by:` для этого, так как она предназначена для организаций, спонсирующих определённые функции, или разработчиков, работающих над ними.
`Sponsored by:`	Организации, спонсировавшие это изменение (если есть). Разделяйте несколько организаций запятыми. Если только часть работы была спонсирована или разные суммы спонсорской поддержки были предоставлены разным авторам, укажите соответствующую информацию в скобках после каждого имени спонсора. Например, `Example.com (alice, code refactoring), Wormulon (bob), Momcorp (cindy)` показывает, что Alice была спонсирована Example.com для рефакторинга кода, в то время как Wormulon спонсировал работу Bob, а Momcorp спонсировал работу Cindy. Другие авторы либо не были спонсированы, либо решили не указывать спонсорство.
`Pull Request:`	Это изменение было отправлено как запрос на принятие изменений (pull request) или запрос на слияние (merge request) в один из публичных Git-репозиториев FreeBSD только для чтения. Оно должно включать полный URL запроса на включение изменений, так как они часто выполняют роль рецензий кода. Например: `https://github.com/freebsd/freebsd-src/pull/745`

`Closes:`	Это изменение завершает серию патчей, обсуждавшихся по указанному адресу в pull request на GitHub, и закрывает данный запрос. Оно должно включать полный URL запроса на включение изменений, так как они часто выполняют роль рецензий кода. Например: `https://github.com/freebsd/freebsd-src/pull/745`
`Co-authored-by:`	Имя и адрес электронной почты дополнительного автора коммита. GitHub содержит подробное описание трейлера Co-authored-by по адресу 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:`	ID подтверждает соответствие требованиям https://developercertificate.org/
`Differential Revision:`	Полный URL обзора в Phabricator. Эта строка *должна быть последней строкой*. Например: `https://reviews.freebsd.org/D1708`.

Пример 4. Журнал изменений для коммита на основе PR

Коммит основан на патче из PR, предоставленного Джоном Смитом. Поле "PR" в сообщении коммита заполнено.

```
...

PR:     12345
```

Участник устанавливает автора патча с помощью `git commit --author "John Smith <John.Smith@example.com>"`.

Пример 5. Журнал изменений для коммита, требующего проверки

Вносятся изменения в систему виртуальной памяти. После отправки патчей в соответствующий список рассылки (в данном случае, `freebsd-arch`) и утверждения изменений.

```
...

Reviewed by:    -arch
```

Пример 6. Журнал изменений для коммита, требующего одобрения

Закоммитить порт после согласования с указанным MAINTAINER, который дал добро на коммит.

```
...
```

```
Approved by:    abc (maintainer)
```

Где *abc* — имя учётной записи лица, одобрившего коммит.

Пример 7. Журнал изменений для коммита, вносящего код из OpenBSD

Коммит некоторого кода на основе работы, выполненной в проекте OpenBSD.

```
...

Obtained from:  OpenBSD
```

Пример 8. Журнал коммитов для правки в FreeBSD-CURRENT с запланированным коммитом в FreeBSD-STABLE в последующее время.

Коммит некоторого кода, которому будет сделано слияние из ветки FreeBSD-CURRENT в ветку FreeBSD-STABLE через две недели.

```
...

MFC after:  2 weeks
```

Где *2* — количество дней, недель или месяцев, после которых запланировано MFC. Вариант *weeks* может быть — `day`, `days`, `week`, `weeks`, `month`, `months`.

Часто необходимо комбинировать их.

Рассмотрим ситуацию, когда пользователь отправил PR с кодом из проекта NetBSD. Разработчик, просматривая PR, видит, что это не та часть дерева, с которой он обычно работает, поэтому он отправляет изменение на рецензирование в список рассылки `arch`. Поскольку изменение сложное, разработчик решает отложить MFC на месяц, чтобы обеспечить достаточное тестирование.

Дополнительная информация, которую нужно включить в коммит, будет выглядеть примерно так

Пример 9. Пример объединённого журнала коммитов

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

10. Предпочтительная лицензия для новых файлов

Полная политика лицензирования проекта FreeBSD доступна по ссылке https://www.FreeBSD.org/internal/software-license. Остальная часть этого раздела предназначена для того, чтобы помочь вам начать работу. Как правило, если сомневаетесь — спрашивайте. Дать совет гораздо проще, чем исправлять дерево исходного кода.

Проект FreeBSD предлагает и использует следующий текст в качестве предпочтительной схемы лицензирования:

```
/*
 * Copyright (c) [year] [your name]
 *
 * SPDX-License-Identifier: BSD-2-Clause
 *
 */
```

Проект FreeBSD настоятельно не рекомендует использовать так называемую «рекламную оговорку» в новом коде. Из-за большого числа участников проекта FreeBSD соблюдение этой оговорки стало затруднительным для многих коммерческих вендоров. Если ваш код в дереве содержит рекламную оговорку, пожалуйста, рассмотрите возможность её удаления. Более того, пожалуйста, рассмотрите возможность использования вышеуказанной лицензии для вашего кода.

Проект FreeBSD не приветствует полностью новые лицензии и вариации стандартных лицензий. Новые лицензии требуют одобрения core@FreeBSD.org для размещения в репозитории `src`. Чем больше различных лицензий используется в дереве, тем больше проблем это создаёт для тех, кто желает использовать этот код, обычно из-за непредвиденных последствий плохо сформулированной лицензии.

Политика проекта требует, чтобы код под некоторыми не-BSD лицензиями размещался только в определённых разделах репозитория, а в некоторых случаях компиляция должна быть условной или даже отключена по умолчанию. Например, ядро GENERIC должно компилироваться только под лицензиями, идентичными или существенно схожими с лицензией BSD. Программное обеспечение под лицензиями GPL, APSL, CDDL и т.п. не должно компилироваться в GENERIC.

Разработчикам следует помнить, что в открытом исходном коде правильное понимание «открытости» так же важно, как и правильное понимание «исходного кода», поскольку неправильное обращение с интеллектуальной собственностью имеет серьёзные последствия. Любые вопросы или опасения следует немедленно доводить до сведения основной команды.

11. Отслеживание лицензий, предоставленных проекту FreeBSD

Различное программное обеспечение или данные существуют в репозиториях, где проект FreeBSD получил специальную лицензию для их использования. Примером могут служить шрифты Terminus для использования с vt(4). Здесь автор Димитар Жеков разрешил нам использовать шрифт "Terminus BSD Console" под лицензией BSD с двумя пунктами, а не под обычной Open Font License, которую он обычно применяет.

Очевидно, разумно вести учёт всех подобных разрешений лицензий. С этой целью core@FreeBSD.org решил сохранять их архив. Каждый раз, когда проекту FreeBSD предоставляется специальная лицензия, мы требуем уведомлять core@FreeBSD.org. Разработчики, участвующие в организации такого разрешения, пожалуйста, присылайте детали на core@FreeBSD.org, включая:

- Контактные данные лиц или организаций, предоставляющих специальную лицензию.
- Какие файлы, каталоги и т.д. в репозиториях охватываются предоставлением лицензии, включая номера ревизий, в которые был добавлен любой материал с особыми условиями лицензирования.
- Дата вступления лицензии в силу. Если не согласовано иное, это будет дата выдачи лицензии авторами соответствующего программного обеспечения.
- Текст лицензии.
- Примечание о любых ограничениях, исключениях или особых условиях, которые применяются конкретно к использованию лицензионных материалов в FreeBSD.
- Любая другая соответствующая информация.

Как только core@FreeBSD.org убедится, что собраны все необходимые данные и они корректны, секретарь отправит подтверждение о получении, подписанное PGP, включая детали лицензии. Это подтверждение будет постоянно архивироваться и служить нашим постоянным свидетельством о предоставлении лицензии.

Архив лицензий должен содержать только сведения о предоставленных лицензиях; это не место для обсуждений вопросов лицензирования или других тем. Доступ к данным в архиве лицензий будет предоставляться по запросу в core@FreeBSD.org.

12. Теги SPDX в дереве

Проект использует теги SPDX в нашей исходной базе. На данный момент эти теги выделены отступами, чтобы автоматизированные средства могли программно извлекать лицензионные условия. Все теги *SPDX-License-Identifier* в дереве следует считать информативными. Все файлы в дереве исходных кодов FreeBSD с этими тегами также содержат копию лицензии, регулирующей использование данного файла. Если возникает противоречие, руководствоваться следует дословным текстом лицензии. Проект старается следовать Спецификации SPDX, версия 2.2. Инструкцию, как помечать исходные файлы, и

допустимые алгебраические выражения можно найти в Приложении D и Приложении E. Проект берет идентификаторы из списка допустимых кратких лицензионных идентификаторов SPDX. Проект использует только тег *SPDX-License-Identifier*.

По состоянию на март 2021 года примерно 25 000 из 90 000 файлов в дереве были помечены.

13. Отношения с разработчиками

При работе непосредственно с вашим собственным кодом или с кодом, который уже хорошо зарекомендовал себя как ваша зона ответственности, вероятно, нет особой необходимости согласовывать действия с другими коммиттерами перед внесением изменений. Это же относится и к исправлению ошибок в явно заброшенных частях системы (к сожалению, такие области существуют). При изменении частей системы, которые поддерживаются (формально или неформально), рассмотрите возможность запроса рецензирования, как это делал бы разработчик до получения прав коммиттера. Для портов свяжитесь с сопровождающим, указанным в переменной `MAINTAINER` в файле Makefile.

Чтобы определить, поддерживается ли определённая часть дерева, проверьте файл MAINTAINERS в корне дерева. Если там никого не указано, просмотрите историю изменений, чтобы увидеть, кто вносил изменения в прошлом. Чтобы вывести список имён и адресов электронной почты всех авторов коммитов для заданного файла за последние 2 года, а также количество коммитов каждого автора, отсортированных по убыванию количества коммитов, используйте:

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

Если запросы остаются без ответа или коммиттер иным образом демонстрирует отсутствие интереса к затронутой области, можно смело выполнять коммит.

Избегайте отправки личных писем сопровождающим. Других людей может заинтересовать не только итоговый результат, но и обсуждение.

Если есть какие-либо сомнения относительно коммита по любой причине, необходимо провести его рецензирование перед выполнением. Лучше получить критику сразу, чем когда он станет частью репозитория. Если коммит вызывает споры, возможно, стоит рассмотреть возможность отката изменений до разрешения вопроса. Помните, что с системой контроля версий мы всегда можем вернуть всё обратно.

Не оспаривайте намерения других. Если они видят иное решение проблемы или даже иную проблему, скорее всего, это не потому, что они глупы, имеют сомнительное происхождение или пытаются разрушить чужой труд, личный имидж или FreeBSD, а просто потому, что у них иной взгляд на мир. Разное — это хорошо.

Честно выражайте несогласие. Обосновывайте свою позицию по её достоинствам, будьте честны относительно возможных недостатков и будьте открыты для понимания их решения или даже их видения проблемы.

Примите исправление. Все мы не безгрешны. Когда вы совершили ошибку, извинитесь и продолжайте жить дальше. Не корите себя и уж тем более не вините других за свою ошибку. Не тратьте время на смущение или взаимные обвинения — просто исправьте проблему и двигайтесь дальше.

Попросите о помощи. Ищите (и предоставляйте) рецензии коллег. Одно из преимуществ открытого программного обеспечения заключается в большом количестве проверяющих; но это не работает, если никто не проверяет код.

14. Если сомневаетесь...

Если вы в чем-то не уверены, будь то технический вопрос или соглашение по проекту, обязательно спросите. Если вы промолчите, вы никогда не продвинетесь вперёд.

Если вопрос связан с технической проблемой, задайте его в публичных списках рассылки. Удержитесь от соблазна написать лично тому, кто знает ответ. Так каждый сможет извлечь пользу из вопроса и ответа.

Для административных вопросов и вопросов, связанных с конкретным проектом, обращайтесь в следующем порядке:

- Ваш наставник или бывший наставник.
- Опытный коммиттер — по IRC, электронной почте и т.д.
- Любая команда с ролью ("hat"), так как в ней вам могут дать окончательный ответ.
- Если всё ещё не уверены, спросите на Список рассылки разработчиков FreeBSD.

Когда ваш вопрос будет решён, если никто не указал вам на документацию, содержащую ответ на ваш вопрос, задокументируйте его, так как у других возникнет тот же вопрос.

15. Bugzilla

Проект FreeBSD использует Bugzilla для отслеживания ошибок и запросов на изменения. Если вы исправили проблему или реализовали предложение из базы данных PR, обязательно закройте PR. Также будет вежливо, если вы найдёте время закрыть другие PR, связанные с вашими коммитами.

Коммиттеры с учётными записями Bugzilla не на `FreeBSD.org` могут объединить старую учётную запись с учётной записью `FreeBSD.org`, выполнив следующие шаги:

1. Войдите, используя старую учётную запись.
2. Открыть новую ошибку. Выбрать `Services` в качестве продукта и `Bug Tracker` в качестве компонента. В описании ошибки перечислите аккаунты, которые нужно объединить.
3. Войдите, используя учётную запись `FreeBSD.org`, и оставьте комментарий к только что созданной ошибке, чтобы подтвердить владение. Дополнительные сведения о

том, как сгенерировать или установить пароль для вашей учётной записи FreeBSD.org, см. в Kerberos и LDAP веб-пароль для кластера FreeBSD.

4. Если необходимо объединить более двух учётных записей, оставьте комментарии от каждой из них.

Вы можете узнать больше о Bugzilla на:

- Рекомендации по работе с сообщениями о проблемах FreeBSD
- https://www.FreeBSD.org/support

16. Phabricator

Проект FreeBSD использует Phabricator для запросов на рецензирование кода. Подробности можно найти на странице Phabricator в вики.

Пожалуйста, используйте команду `git arc`, предоставляемую `devel/freebsd-git-arc` (установите порт или пакет, затем введите `git help arc` для получения документации), для создания и обновления рецензий в Phabricator. Это упростит другим процесс рецензирования и тестирования ваших патчей.

Коммиттеры с учётными записями Phabricator не на FreeBSD.org могут переименовать старую учётную запись в FreeBSD.org, выполнив следующие шаги:

1. Измените адрес электронной почты вашей учётной записи Phabricator на ваш FreeBSD.org email.
2. Открыть новую ошибку в нашем трекере ошибок, используя вашу учётную запись FreeBSD.org, см. Bugzilla для получения дополнительной информации. Выберите `Services` в качестве продукта и `Code Review` в качестве компонента. В описании ошибки запросите переименование вашей учётной записи Phabricator и укажите ссылку на вашего пользователя Phabricator. Например, `https://reviews.freebsd.org/p/bob_example.com/`

Учётные записи Phabricator не могут быть объединены, пожалуйста, не создавайте новую учётную запись.

17. Кто есть кто

Помимо хранителей репозиториев, есть и другие участники проекта FreeBSD и команды, с которыми вам, скорее всего, предстоит познакомиться в роли коммиттера. Кратко и далеко не исчерпывающе, вот они:

`Группа Менеджеров Дерева Документации <doceng@FreeBSD.org>`

doceng — это группа, ответственная за инфраструктуру сборки документации, утверждение новых коммиттеров документации и обеспечение актуальности веб-сайта

FreeBSD и документации на FTP-сайте в соответствии с деревом Subversion. Она не является органом по разрешению конфликтов. Подавляющее большинство обсуждений, связанных с документацией, происходит в рассылке Список рассылки Проекта Документации FreeBSD. Подробнее о команде doceng можно узнать в её уставе. Коммиттеры, заинтересованные в работе над документацией, должны ознакомиться с руководством Проект документации FreeBSD: введение для новых участников.

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>

Вот члены команды Группа Выпуска Релизов FreeBSD <re@FreeBSD.org>. Эта команда отвечает за установку сроков выпуска и контроль процесса выпуска. Во время заморозки кода, инженеры выпуска обладают окончательным правом принятия решений по всем изменениям в системе для ветки, ожидающей статуса выпуска. Если у вас есть что-то, что вы хотите влить (merge) из FreeBSD-CURRENT в FreeBSD-STABLE (какими бы ни были их значения в любой момент времени), именно с этими людьми нужно обсудить этот вопрос.

Gordon Tetlow <gordon@FreeBSD.org>

Gordon Tetlow — это ответственный за безопасность FreeBSD, который курирует работу Команда Офицера Безопасности <security-officer@FreeBSD.org>.

Список рассылки коммиттеров FreeBSD

{dev-src-all}, {dev-ports-all} и {dev-doc-all} — это списки рассылки, которые система управления версиями использует для отправки сообщений о коммитах. *Никогда* не отправляйте письма напрямую в эти списки. Отправляйте ответы в этот список только в том случае, если они короткие и непосредственно связаны с коммитом.

Список рассылки разработчиков FreeBSD

Все коммиттеры подписаны на список рассылки -developers. Этот список был создан для обсуждения вопросов, связанных с сообществом коммиттеров. Примеры включают голосования Core, объявления и т.д.

Список рассылки разработчиков FreeBSD предназначен исключительно для использования коммиттерами FreeBSD. Для разработки FreeBSD коммиттеры должны иметь возможность открыто обсуждать вопросы, которые будут решены до их публичного объявления. Откровенные обсуждения работы в процессе не подходят для открытой публикации и могут навредить FreeBSD.

Все коммиттеры FreeBSD обязаны не публиковать и не пересылать сообщения из Список рассылки разработчиков FreeBSD за пределы списка рассылки без разрешения всех авторов. Нарушители будут удалены из Список рассылки разработчиков FreeBSD, что приведёт к приостановке привилегий на коммит. Повторные или вопиющие нарушения могут привести к постоянному лишению привилегий на коммит.

Этот список *не* предназначен для обзора кода или каких-либо технических обсуждений.

На самом деле, использование его в таком качестве вредит проекту FreeBSD, так как создаёт впечатление закрытого списка, где общие решения, затрагивающие всех пользователей FreeBSD, принимаются без "открытости". И последнее, но не менее важное: *никогда, ни при каких обстоятельствах, не отправляйте письмо на Список рассылки разработчиков FreeBSD с копией (CC:/BCC:) на другой список FreeBSD*. Никогда не отправляйте письмо на другой список рассылки FreeBSD с копией (CC:/BCC:) на Список рассылки разработчиков FreeBSD. Это может значительно снизить пользу от данного списка.

18. Руководство по быстрому началу работы с SSH

1. Если вы не хотите каждый раз вводить пароль при использовании ssh(1) и используете ключи для аутентификации, ssh-agent(1) создан для вашего удобства. Если вы хотите использовать ssh-agent(1), убедитесь, что запускаете его перед запуском других приложений. Например, пользователи X обычно делают это в .xsession или .xinitrc. Подробности см. в ssh-agent(1).
2. Сгенерируйте пару ключей с помощью ssh-keygen(1). Пара ключей окажется в вашем каталоге $HOME/.ssh/.

   

   Поддерживаются только ключи ECDSA, Ed25519 или RSA.
3. Отправьте ваш открытый ключ ($HOME/.ssh/id_ecdsa.pub, $HOME/.ssh/id_ed25519.pub или $HOME/.ssh/id_rsa.pub) человеку, который настраивает вас как коммиттера, чтобы его можно было добавить в yourlogin в /etc/ssh-keys/ на `freefall`.

Теперь ssh-add(1) можно использовать для аутентификации один раз за сеанс. Он запрашивает парольную фразу для закрытого ключа и затем сохраняет её в агенте аутентификации (ssh-agent(1)). Используйте `ssh-add -d` для удаления ключей, сохранённых в агенте.

Проверка с помощью простой удалённой команды: `ssh freefall.FreeBSD.org ls /usr`.

Для получения дополнительной информации см. security/openssh-portable, ssh(1), ssh-add(1), ssh-agent(1), ssh-keygen(1) и scp(1).

Для информации о добавлении, изменении или удалении ключей ssh(1), см. эту статью.

19. Доступность Coverity® для коммиттеров FreeBSD

Все разработчики FreeBSD могут получить доступ к результатам анализа Coverity для всего программного обеспечения проекта FreeBSD. Все, кто заинтересован в доступе к результатам автоматического анализа Coverity, могут зарегистрироваться на Coverity Scan.

В вики FreeBSD есть мини-руководство для разработчиков, которые заинтересованы в работе с отчётами анализа Coverity®: https://wiki.freebsd.org/CoverityPrevent. Обратите внимание, что это мини-руководство доступно только разработчикам FreeBSD, поэтому если вы не можете открыть эту страницу, вам нужно будет попросить добавить вас в соответствующий список доступа к вики.

Наконец, все разработчики FreeBSD, которые собираются использовать Coverity®, всегда могут запросить дополнительные детали и информацию об использовании, задав любые вопросы в списке рассылки разработчиков FreeBSD.

20. Большой список правил коммиттеров FreeBSD

Все участники проекта FreeBSD должны соблюдать *Кодекс поведения*, доступный по ссылке https://www.FreeBSD.org/internal/code-of-conduct. Как коммиттеры, вы представляете публичное лицо проекта, и ваше поведение оказывает существенное влияние на его общественное восприятие. Это руководство расширяет разделы *Кодекса поведения*, относящиеся к коммиттерам.

1. Уважайте других коммиттеров.
2. Уважайте других участников.
3. Обсудите любые значительные изменения *перед* коммитом.
4. Уважайте существующих сопровождающих (если они указаны в поле `MAINTAINER` в файле Makefile или в файле MAINTAINER в корневом каталоге).
5. Любое оспариваемое изменение должно быть отменено до разрешения спора, если этого потребует сопровождающий. Изменения, связанные с безопасностью, могут перекрыть пожелания сопровождающего по усмотрению Ответственного за безопасность.
6. Изменения попадают в FreeBSD-CURRENT до FreeBSD-STABLE, если только это явно не разрешено инженером выпуска или если они не применимы к FreeBSD-CURRENT. Любое нетривиальное или не срочное изменение, которое применимо, также должно оставаться в FreeBSD-CURRENT как минимум 3 дня перед слиянием, чтобы его можно было достаточно протестировать. Инженер выпуска имеет те же полномочия в отношении ветки FreeBSD-STABLE, что и сопровождающий, согласно правилу №5.
7. Не устраивайте публичные разборки с другими разработчиками; это выглядит плохо.
8. Соблюдайте все периоды заморозки кода и своевременно читайте рассылки `committers` и `developers`, чтобы знать, когда действует заморозка кода.
9. Если сомневаетесь в каком-либо действии — сначала спросите!
10. Проверьте свои изменения перед их применением.
11. Не вносите изменения в предоставленное программное обеспечение без *явного* одобрения соответствующих сопровождающих.

Как уже отмечалось, нарушение некоторых из этих правил может стать основанием для временного лишения прав на коммиты или, при повторных нарушениях, для их

постоянного отзыва. Отдельные члены Основной Команды (Core Team) имеют право временно приостановить права на коммиты до тех пор, пока основная команда в полном составе не рассмотрит вопрос. В случае «чрезвычайной ситуации» (например, если коммиттер наносит ущерб репозиторию), временное лишение прав может также быть осуществлено хранителями репозитория. Только Основная Команда 2/3 большинства голосов имеет право приостановить права на коммиты сроком более чем на неделю или отозвать их полностью. Это правило существует не для того, чтобы сделать основную команду сборищем жестоких диктаторов, которые могут избавляться от коммиттеров так же легко, как от пустых банок из-под газировки, а чтобы дать проекту своего рода предохранительный клапан. Если кто-то выходит из-под контроля, важно иметь возможность немедленно принять меры, а не быть парализованными дискуссией. Во всех случаях коммиттер, чьи права приостановлены или отозваны, имеет право на «слушание» в основной команде, где определяется общая продолжительность приостановки. Коммиттер, чьи права приостановлены, также может запросить пересмотр решения через 30 дней и каждые 30 дней после этого (если общий срок приостановки не менее 30 дней). Коммиттер, чьи права были полностью отозваны, может запросить пересмотр по истечении 6 месяцев. Эта политика пересмотра *строго неформальна*, и во всех случаях основная команда оставляет за собой право как удовлетворить, так и проигнорировать запрос на пересмотр, если считает первоначальное решение правильным.

Во всех остальных аспектах работы проекта основная команда является подмножеством коммиттеров и связана *теми же правилами*. Тот факт, что кто-то входит в основную команду, не означает, что им позволено выходить за рамки обозначенных здесь границ; «особые полномочия» основной команды активируются только при действиях в качестве группы, а не индивидуально. Как отдельные лица, члены основной команды — это прежде всего коммиттеры, и только во вторую очередь — основная команда.

20.1. Подробности

1. Уважайте других коммиттеров.

   Это означает, что вы должны относиться к другим коммиттерам как к коллегам-разработчикам, каковыми они и являются. Несмотря на наши периодические попытки доказать обратное, коммиттером не становятся по глупости, и ничто не раздражает больше, чем когда коллеги относятся к вам именно так. Независимо от того, испытываем ли мы всегда уважение друг к другу или нет (у всех бывают плохие дни), мы должны *относиться* к другим коммиттерам с уважением всегда — как на публичных форумах, так и в личной переписке.

   Способность долгосрочного сотрудничества — это самое ценное достояние проекта, гораздо более важное, чем любые изменения в коде, и превращение споров о коде в проблемы, влияющие на нашу способность гармонично работать вместе, ни при каких обстоятельствах не стоит того.

   Для соблюдения этого правила не отправляйте электронные письма, когда вы злитесь или ведёте себя таким образом, который может показаться другим излишне конфронтационным. Сначала успокойтесь, затем подумайте, как наиболее эффективно донести свою точку зрения, чтобы убедить других в своей правоте, а не просто

выпустить пар для кратковременного облегчения за счёт долгосрочного конфликта. Иначе это обернётся не только крайне нерациональной тратой сил, но и тем, что повторяющиеся проявления публичной агрессии, мешающие нашей совместной работе, будут строго пресекаться руководством проекта и могут привести к приостановке или лишению ваших прав на коммит. Руководство проекта будет учитывать как публичные, так и приватные сообщения, представленные на его рассмотрение. Оно не будет требовать раскрытия приватной переписки, но примет её во внимание, если участники, вовлечённые в жалобу, предоставят её добровольно.

Всё это никогда не является вариантом, который руководство проекта хоть сколько-нибудь радует, но единство важнее. Никакое количество кода или полезных советов не стоит того, чтобы этим пожертвовать.

2. Уважайте других участников.

   Вы не всегда были коммиттером. Когда-то вы были контрибьютором. Помните об этом всегда. Вспомните, каково это — пытаться получить помощь и внимание. Не забывайте, что ваша работа в качестве контрибьютора была для вас очень важна. Вспомните, каково это. Не отговаривайте, не принижайте и не унижайте контрибьюторов. Относитесь к ним с уважением. Они — наши будущие коммиттеры. Они так же важны для проекта, как и коммиттеры. Их вклад так же важен и значим, как и ваш. В конце концов, вы сами сделали множество вкладов, прежде чем стали коммиттером. Всегда помните об этом.

   Учитывайте моменты, поднятые в разделе Уважайте других коммиттеров, и применяйте их также к контрибьюторам.

3. Обсудите любые значительные изменения *перед* коммитом.

   Репозиторий — это не место, где изменения первоначально отправляются на проверку корректности или обсуждаются, сначала это происходит в почтовых рассылках или с помощью сервиса Phabricator. Коммит произойдет только после того, как будет достигнуто некое подобие консенсуса. Это не означает, что требуется разрешение для исправления каждой очевидной синтаксической ошибки или опечатки на странице руководства, просто полезно развить чутьё, когда предлагаемое изменение не столь очевидно и требует предварительного обсуждения. Люди действительно не против масштабных изменений, если результат явно лучше того, что было раньше, им просто не нравится, когда эти изменения становятся *неожиданностью*. Самый лучший способ убедиться, что всё идёт по правильному пути, — это провести рецензирование кода одним или несколькими другими коммиттерами.

   Если сомневаетесь, запросите рецензию!

4. Уважайте существующих сопровождающих, если они указаны.

   Многие части FreeBSD не "являются чьей-то собственностью" в том смысле, что конкретный человек вскочит и начнёт кричать, если вы внесёте изменения в "его" область, но всё же лучше сначала проверить. Одно из используемых соглашений — добавление строки `maintainer` в Makefile для любого пакета или поддерева, за которые активно отвечает один или несколько людей; см. Рекомендации и политики дерева

исходного кода для документации по этому поводу. Если у участков кода есть несколько сопровождающих, изменения в затронутых областях, внесённые одним сопровождающим, должны быть проверены хотя бы одним другим сопровождающим. В случаях, когда "сопровождение" чего-либо неясно, посмотрите логи репозитория для соответствующих файлов и проверьте, работал ли кто-то недавно или преимущественно в этой области.

5. Любое оспариваемое изменение должно быть отменено до разрешения спора, если этого потребует сопровождающий. Изменения, связанные с безопасностью, могут перекрыть пожелания сопровождающего по усмотрению Ответственного за безопасность.

   Это может быть трудно принять во время конфликтов (когда каждая сторона убеждена, что она права, конечно), но система контроля версий делает ненужным продолжение споров, когда гораздо проще просто отменить спорное изменение, успокоить всех и затем попытаться выяснить, как лучше поступить. Если окажется, что изменение всё-таки было правильным, его можно легко вернуть. Если же нет, то пользователям не придётся мириться с ошибочным изменением в дереве, пока все активно обсуждают его достоинства. Люди *очень* редко требуют отмены изменений в репозитории, поскольку обсуждение обычно выявляет плохие или спорные изменения ещё до коммита, но в таких редких случаях отмена должна производиться без споров, чтобы можно было сразу перейти к выяснению, было ли изменение ошибочным или нет.

6. Изменения вносятся в FreeBSD-CURRENT до FreeBSD-STABLE, если это не разрешено специально инженером выпуска или если они не применимы к FreeBSD-CURRENT. Любое нетривиальное или несрочное изменение, которое применимо, также должно оставаться в FreeBSD-CURRENT как минимум 3 дня перед слиянием, чтобы его можно было достаточно протестировать. Инженер выпуска имеет те же полномочия в отношении ветки FreeBSD-STABLE, как указано в правиле №5.

   Это ещё один вопрос, о котором не стоит спорить, поскольку инженер выпуска несёт окончательную ответственность (и получает по шапке), если изменение окажется неудачным. Пожалуйста, уважайте это и оказывайте инженеру выпуска полное сотрудничество, когда дело касается ветки FreeBSD-STABLE. Управление FreeBSD-STABLE может часто казаться излишне консервативным для случайного наблюдателя, но также стоит помнить, что консерватизм — это отличительная черта FreeBSD-STABLE, и там действуют другие правила, чем в FreeBSD-CURRENT. Также нет смысла в том, чтобы FreeBSD-CURRENT был испытательным полигоном, если изменения немедленно переносятся в FreeBSD-STABLE. Изменения должны быть протестированы разработчиками FreeBSD-CURRENT, поэтому дайте некоторое время перед слиянием, если только исправление в FreeBSD-STABLE не является критическим, срочным или настолько очевидным, что дальнейшее тестирование не требуется (исправления опечаток в руководствах, очевидные исправления ошибок/опечаток и т.д.). Другими словами, руководствуйтесь здравым смыслом.

   Изменения в ветках безопасности (например, `releng/9.3`) должны быть одобрены членом группы — Команда Офицера Безопасности <security-officer@FreeBSD.org> или, в некоторых случаях, членом группы — Группа Выпуска Релизов FreeBSD <re@FreeBSD.org>.

7. Не устраивайте публичные разборки с другими разработчиками; это выглядит плохо.

Этот проект поддерживает публичный имидж, который очень важен для всех нас, особенно если мы хотим продолжать привлекать новых участников. Бывают случаи, когда, несмотря на все усилия сохранять самообладание, люди теряют терпение и обмениваются резкими словами. Лучшее, что можно сделать в таких ситуациях, — минимизировать последствия, пока все не остынут. Не выносите гневные слова на публику и не пересылайте личную переписку или другие частные сообщения в публичные списки рассылки, почтовые алиасы, каналы мгновенных сообщений или социальные сети. То, что люди говорят один на один, часто менее приукрашено, чем их публичные высказывания, и такие сообщения не имеют места в публичном пространстве — они только усугубляют и без того сложную ситуацию. Если человек, отправляющий гневное сообщение, хотя бы проявил такт и отправил его лично, проявите такт и вы — оставьте это в тайне. Если вы чувствуете, что другой разработчик относится к вам несправедливо, и это причиняет вам страдания, обратитесь к Основной команде (Core team), а не выносите это на публику. Основная команда постарается выступить в роли миротворцев и вернуть ситуацию в разумное русло. Если спор касается изменений в кодовой базе и участники не могут прийти к согласию, основная команда может назначить третью сторону, устраивающую всех, для разрешения спора. Все вовлечённые стороны должны согласиться с решением, принятым этой третьей стороной.

8. Соблюдайте все периоды заморозки кода и своевременно читайте рассылки `committers` и `developers`, чтобы знать, когда действует заморозка кода.

   Внесение неподтверждённых изменений во время заморозки кода — это серьёзная ошибка, и от коммиттеров ожидается, что они будут в курсе происходящего, прежде чем, вернувшись после долгого отсутствия, закоммитить 10 мегабайт накопленных изменений. Те, кто злоупотребляет этим на регулярной основе, будут лишены прав на коммиты до возвращения из Лагеря весёлого перевоспитания FreeBSD, который мы проводим в Гренландии.

9. Если сомневаетесь в каком-либо действии — сначала спросите!

   Многие ошибки совершаются из-за того, что кто-то торопится и просто предполагает, что знает правильный способ что-то сделать. Если вы не делали этого раньше, велика вероятность, что вы на самом деле не знаете, как у нас принято делать, и вам действительно нужно сначала спросить, иначе вы рискуете полностью опозориться на публике. Нет ничего постыдного в вопросе «как, черт возьми, это сделать?» Мы уже знаем, что вы умный человек; иначе вы не были бы коммиттером.

10. Проверьте свои изменения перед их применением.

    Если ваши изменения касаются ядра, убедитесь, что вы по-прежнему можете компилировать как GENERIC, так и LINT. Если ваши изменения касаются других частей системы, убедитесь, что вы по-прежнему можете компилировать пользовательское пространство с помощью `make buildworld`. Если ваши изменения относятся к ветке, убедитесь, что тестирование проводится на машине, которая работает с этим кодом. Если ваше изменение может нарушить работу другой архитектуры, обязательно протестируйте его на всех поддерживаемых архитектурах. Пожалуйста, убедитесь, что ваше изменение работает для поддерживаемых инструментов сборки. Пожалуйста,

обратитесь к Внутренней странице FreeBSD для получения списка доступных ресурсов. По мере добавления других архитектур в список поддерживаемых платформ FreeBSD будут предоставлены соответствующие общие ресурсы для тестирования.

11. Не вносите изменения в предоставленное программное обеспечение без *явного* одобрения соответствующих сопровождающих.

    Предоставленное программное обеспечение — это всё, что находится в деревьях каталогов src/contrib, src/crypto или src/sys/contrib.

    Упомянутые выше деревья предназначены для предоставленного программного обеспечения, обычно импортируемого в ветку вендора. Фиксация изменений там может вызвать ненужные проблемы при импорте более новых версий программного обеспечения. В общем случае рекомендуется отправлять исправления напрямую вендору. Исправления могут быть сначала зафиксированы в FreeBSD с разрешения сопровождающего.

    Причины изменения стороннего программного обеспечения варьируются от желания строго контролировать тесно связанную зависимость до отсутствия переносимости в каноническом репозитории с их кодом. Независимо от причины, усилия по минимизации нагрузки на поддержку форка полезны для других сопровождающих. Избегайте внесения тривиальных или косметических изменений в файлы, так как это усложняет каждое последующее слияние: такие патчи необходимо вручную перепроверять при каждом импорте.

    Если конкретное программное обеспечение не имеет сопровождающего, вам предлагается взять на себя ответственность за него. Если вы не уверены в текущем сопровождении, напишите на Список рассылки, посвящённый архитектуре и внутреннему устройству FreeBSD и уточните.

20.2. Политика поддержки нескольких архитектур

В попытке упростить поддержку переносимости FreeBSD на платформы, которые мы поддерживаем, ядро системы разработало следующее предписание:

Основные проектные работы (включая значительные изменения API и ABI) должны подтвердить свою работоспособность как минимум на одной платформе Уровня 1 перед тем, как они могут быть включены в дерево исходного кода.

Разработчики также должны учитывать нашу политику уровней поддержки аппаратных архитектур в долгосрочной перспективе. Эти правила предназначены для предоставления рекомендаций в процессе разработки и отличаются от требований к функциям и архитектурам, указанным в этом разделе. Правила уровней поддержки функций для архитектур на момент выпуска более строгие, чем правила для изменений в процессе разработки.

20.3. Политика по использованию нескольких компиляторов

Базовая система FreeBSD собирается как с Clang, так и с GCC. Проект делает это тщательно и контролируемо, чтобы максимизировать преимущества от этой дополнительной работы, сводя эту работу к минимуму. Поддержка обоих компиляторов повышает гибкость для наших пользователей. У этих компиляторов разные сильные и слабые стороны, и их совместная поддержка позволяет пользователям выбирать наиболее подходящий для их задач. Clang и GCC поддерживают схожие диалекты C и C++, что требует относительно небольшого количества условного кода. Проект получает улучшенное покрытие кода и повышает его качество, используя возможности обоих компиляторов. Поддержка этого диапазона позволяет проекту собираться в большем количестве пользовательских окружений и задействовать больше CI-окружений, увеличивая удобство для пользователей и предоставляя им больше инструментов для тестирования. Тщательно ограничивая диапазон поддерживаемых версий современными версиями этих компиляторов, проект избегает чрезмерного увеличения матрицы тестирования. Устаревшие и малоизвестные компиляторы, а также старые диалекты языков, имеют крайне ограниченную поддержку, позволяющую собирать пользовательские программы, но без ограничений на сборку базовой системы с их помощью. Точный баланс продолжает развиваться, чтобы гарантировать, что преимущества дополнительной работы остаются выше накладываемых ею затрат. Раньше проект поддерживал очень старые компиляторы Intel или старые версии GCC, но мы заменили поддержку этих устаревших компиляторов на тщательно выбранный диапазон современных компиляторов. В этом разделе описано, где мы используем разные компиляторы и какие ожидания с этим связаны.

Базовая система FreeBSD включает в себя компилятор Clang, входящий в дерево исходного кода. Поскольку он является частью дерева, этот компилятор обладает наибольшей поддержкой. Все изменения должны компилироваться с ним перед коммитом. Полное тестирование, соответствующее изменениям, должно проводиться с этим компилятором.

Базовая система FreeBSD также поддерживает различные версии Clang и GCC в качестве компиляторов, не входящих в дерево исходного кода. Для крупных или рискованных изменений коммиттеры должны выполнить тестовую сборку с поддерживаемой версией GCC. Компиляторы, не входящие в дерево исходного кода, доступны в виде пакетов. Компиляторы GCC доступны в виде пакетов `${TARGET_ARCH}-gcc${VERSION}`, например aarch64-gcc14. Компиляторы Clang доступны в виде пакетов `llvm${VERSION}`, например llvm18. Проект запускает автоматизированные задачи CI для сборки всего с использованием этих компиляторов. Ожидается, что коммиттеры исправят задачи, которые они сломали своими изменениями. Коммиттеры могут тестировать сборки пользовательского пространства или отдельных ядер, установив переменную `CROSS_TOOLCHAIN` в имя пакета, например `CROSS_TOOLCHAIN=aarch64-gcc14` или `CROSS_TOOLCHAIN=llvm18`. Для сборки universe или tinderbox, `USE_GCC_TOOLCHAINS=gcc${VERSION}` собирает все архитектуры с использованием соответствующих пакетов компиляторов GCC. Для сборки universe или tinderbox с использованием Clang, не входящего в дерево исходного кода, передайте `CROSS_TOOLCHAIN=llvm${VERSION}`. Обратите внимание, что хотя все архитектуры в базовой системе могут быть скомпилированы с помощью Clang, только несколько архитектур могут быть полностью собраны с помощью GCC.

Проект FreeBSD также имеет несколько CI-пайплайнов на GitHub. Для запросов на принятие изменений (pull request) на GitHub и некоторых веток, отправленных в форки на GitHub, выполняется ряд задач кросс-компиляции. Они тестируют сборку FreeBSD с использованием версий Clang, которые отстают от встроенного компилятора на одну или несколько основных версий.

Проект FreeBSD также обновляет компиляторы. И Clang, и GCC быстро развиваются. Некоторые изменения, такие как удаление устаревших объявлений и определений функций в стиле K&R, будут внесены в дерево до появления нового компилятора. Коммиттерам следует учитывать это и быть готовыми к проверке проблем в своём коде или изменениях с этими новыми компиляторами. Кроме того, сразу после появления новой версии компилятора в дереве, может потребоваться компиляция с использованием старой версии, если есть подозрение на необнаруженную регрессию.

В дополнение к компилятору, LLD от LLVM и binutils от GNU используются компилятором косвенно. Коммиттеры должны учитывать различия в синтаксисе ассемблера и особенностях компоновщиков, а также обеспечивать работоспособность обоих вариантов. Эти компоненты будут тестироваться в рамках CI-задач FreeBSD для Clang или GCC.

Проект FreeBSD предоставляет заголовочные файлы и библиотеки, которые позволяют использовать другие компиляторы для сборки программного обеспечения, не входящего в базовую систему. Эти заголовочные файлы поддерживают создание среды, соответствующей стандартам, включая более ранние диалекты ANSI-C, начиная с C89, а также другие специфичные случаи, выявленные нашей обширной коллекцией портов. Эта поддержка ограничивает отказ от старых стандартов в таких местах, как заголовочные файлы, но не мешает обновлению базовой системы до более новых диалектов. Также она не требует, чтобы базовая система целиком компилировалась с использованием этих старых стандартов. Нарушение этой поддержки приведёт к сбоям в сборке пакетов из коллекции портов, поэтому по возможности этого следует избегать, а при обнаружении проблем — оперативно их исправлять.

Система сборки FreeBSD в настоящее время поддерживает эти различные среды. По мере добавления новых предупреждений в компиляторы проект старается их исправлять. Однако иногда эти предупреждения требуют значительной переработки, поэтому они подавляются каким-либо образом с использованием переменных make, которые вычисляют правильное значение в зависимости от версии компилятора. Разработчики должны учитывать это и обеспечивать правильную условную обработку любых флагов, специфичных для компилятора.

20.3.1. Текущие версии компиляторов

Версии поддерживаемых компиляторов для конкретной ветки, такой как `main` или `stable/X`, со временем меняются. Авторитетным источником информации о поддерживаемых версиях компиляторов являются автоматизированные задачи CI, тестируемые в кросс-сборках GitHub Actions и Jenkins.

Ветка	Компилятор ветки	llvm12	llvm13	llvm14	llvm15	llvm18	amd64-gcc12	amd64-gcc13	amd64-gcc14	amd64-gcc15	riscv64-gcc15
main	llvm 19				Y	Y	Y	Y	Y	Y	Y
stable/15	llvm 19			Y		Y	Y	Y	Y		
stable/14	llvm 19	Y	Y	Y			Y		Y		
stable/13	llvm 19	Y	Y	Y			Y		Y		

Набор инструментов разработки GCC тестируется для amd64 и riscv64 через задания CI в Jenkins. Набор инструментов разработки LLVM тестируется для aarch64 и arm64 в кросс-сборках GitHub.

20.4. Другие предложения

При внесении изменений в документацию перед коммитом проверяйте орфографию. Для всех XML-документов убедитесь в корректности директив форматирования, выполнив `make lint` и textproc/igor.

Для страниц руководства запустите sysutils/manck и textproc/igor на странице руководства, чтобы проверить, что все перекрестные ссылки и ссылки на файлы корректны, а также что страница руководства имеет все необходимые `MLINKS`.

Не смешивайте исправления стиля с новой функциональностью. Исправление стиля — это любое изменение, которое не меняет функциональность кода. Смешивание изменений затрудняет понимание изменений функциональности при запросе различий между версиями, что может скрыть новые ошибки. Не включайте изменения пробелов вместе с изменениями содержимого в коммитах для doc/. Лишний шум в различиях значительно усложняет работу переводчиков. Вместо этого вносите исправления стиля или пробелов в отдельных коммитах, явно обозначенных как таковые в сообщении коммита.

20.5. Устаревающие функции

Когда необходимо удалить функциональность из программного обеспечения в базовой системе, по возможности следуйте этим рекомендациям:

1. Упоминание о том, что опция, утилита или интерфейс устарели, содержится в руководстве и, возможно, в примечаниях к выпуску. Использование устаревшей функции вызывает предупреждение.
2. Опция, утилита или интерфейс сохраняются до следующего мажорного (точка-ноль) релиза.

3. Опция, утилита или интерфейс удалены и больше не документируются. Они считаются устаревшими. Также обычно рекомендуется указать их удаление в примечаниях к выпуску.

20.6. Конфиденциальность и приватность

1. Большая часть работы над FreeBSD выполняется публично.

   FreeBSD — это *открытый* проект. Это означает, что не только любой может использовать исходный код, но и большая часть процесса разработки открыта для публичного изучения.

2. Некоторые конфиденциальные вопросы должны оставаться в тайне или быть под запретом на разглашение.

   К сожалению, полной прозрачности быть не может. Как разработчик FreeBSD, вы будете иметь определённую степень привилегированного доступа к информации. Следовательно, от вас ожидается соблюдение определённых требований конфиденциальности. Иногда необходимость конфиденциальности исходит от внешних сотрудников или имеет конкретный временной лимит. Однако в большинстве случаев это вопрос неразглашения частных переписок.

3. Ответственный за безопасность обладает исключительным правом публикации уведомлений о безопасности.

   Где есть проблемы безопасности, затрагивающие множество различных операционных систем, FreeBSD часто зависит от раннего доступа, чтобы иметь возможность подготовить уведомления для согласованного выпуска. Если разработчики FreeBSD не могут быть доверены в вопросах поддержания безопасности, такой ранний доступ предоставлен не будет. Ответственный за безопасность контролирует доступ к информации об уязвимостях до их публикации и определяет время выпуска всех уведомлений. Он может запросить помощь при условии конфиденциальности у любого разработчика с соответствующими знаниями для подготовки исправлений безопасности.

4. Коммуникации с Основной командой (Core Team) сохраняются в конфиденциальности столько времени, сколько необходимо.

   Коммуникации с Основной командой изначально будут рассматриваться как конфиденциальные. Однако в конечном итоге большая часть деятельности Основной команды будет обобщена в ежемесячных или квартальных отчётах ядра. Будет уделено внимание тому, чтобы избежать разглашения каких-либо чувствительных деталей. Записи по некоторым особо чувствительным темам могут вообще не попадать в отчёты и будут храниться только в частных архивах Основной команды.

5. Соглашения о неразглашении могут потребоваться для доступа к определённой коммерчески чувствительной информации.

   Доступ к определённым коммерчески чувствительным данным может быть

предоставлен только при подписании Соглашения о неразглашении. Перед заключением каких-либо юридически обязывающих соглашений необходимо проконсультироваться с юридическим отделом Фонда FreeBSD.

6. Приватные сообщения не должны становиться публичными без разрешения.

   Помимо указанных выше конкретных требований, существует общее правило не публиковать личную переписку между разработчиками без согласия всех вовлеченных сторон. Перед пересылкой сообщения в публичную рассылку, размещением на форуме или веб-сайте, доступном не только для исходных корреспондентов, необходимо запросить разрешение.

7. Общение в каналах, предназначенных только для проекта или с ограниченным доступом, должно оставаться конфиденциальным.

   Аналогично личным сообщениям, некоторые внутренние каналы связи, включая почтовые рассылки только для коммиттеров FreeBSD и IRC-каналы с ограниченным доступом, считаются частной перепиской. Для публикации материалов из этих источников требуется разрешение.

8. Основная команда может одобрить публикацию.

   В случаях, когда получение разрешения непрактично из-за количества корреспондентов или когда разрешение на публикацию необоснованно отклоняется, Основная команда может одобрить раскрытие таких частных вопросов, которые заслуживают более широкой публикации.

21. Поддержка множественных архитектур

FreeBSD — это высокопортативная операционная система, предназначенная для работы на множестве различных типов аппаратных архитектур. Поддержание четкого разделения между машинозависимым (MD) и машинонезависимым (MI) кодом, а также минимизация MD-кода являются важной частью нашей стратегии по сохранению гибкости в отношении текущих тенденций в аппаратном обеспечении. Каждая новая аппаратная архитектура, поддерживаемая FreeBSD, существенно увеличивает затраты на поддержку кода, инструментальных средств и управление выпусками. Это также значительно повышает стоимость эффективного тестирования изменений в ядре. Таким образом, существует серьезная мотивация для разграничения уровней поддержки различных архитектур, оставаясь при этом сильными в нескольких ключевых архитектурах, которые рассматриваются как «целевая аудитория» FreeBSD.

21.1. Заявление об общих намерениях

Проект FreeBSD ориентирован на "коммерческие готовые рабочие станции, серверы и высокопроизводительные встраиваемые системы производственного уровня". Сохраняя фокус на узком наборе архитектур, актуальных для этих сред, проект FreeBSD способен

поддерживать высокий уровень качества, стабильности и производительности, а также минимизировать нагрузку на различные команды поддержки проекта, такие как команда портов, команда документации, офицер безопасности и команды разработки релизов. Разнообразие в поддержке оборудования расширяет возможности потребителей FreeBSD, предлагая новые функции и варианты использования, однако эти преимущества всегда должны тщательно оцениваться с учётом реальных затрат на поддержку дополнительных платформ.

Проект FreeBSD разделяет целевые платформы на четыре уровня. Каждый уровень включает список гарантий, на которые могут рассчитывать пользователи, а также обязательства проекта и разработчиков по выполнению этих гарантий. Эти списки определяют минимальные гарантии для каждого уровня. Проект и разработчики могут предоставлять дополнительные уровни поддержки, превышающие минимальные гарантии для данного уровня, но такая дополнительная поддержка не гарантируется. Каждая целевая платформа назначается на определённый уровень для каждой стабильной ветви. В результате, целевая платформа может быть назначена на разные уровни в параллельных стабильных ветках.

21.2. Целевые платформы

Поддержка аппаратной платформы состоит из двух компонентов: поддержки ядра и пользовательских двоичных интерфейсов приложений (ABI). Поддержка платформы на уровне ядра включает в себя всё необходимое для запуска ядра FreeBSD на аппаратной платформе, например, зависящее от машины управление виртуальной памятью и драйверы устройств. Пользовательский ABI определяет интерфейс для взаимодействия пользовательских процессов с ядром FreeBSD и базовыми системными библиотеками. Пользовательский ABI включает интерфейсы системных вызовов, расположение и семантику публичных структур данных, а также расположение и семантику аргументов, передаваемых подпрограммам. Некоторые компоненты ABI могут быть определены спецификациями, такими как расположение объектов исключений C++ или соглашения о вызовах для функций C.

Ядро FreeBSD также использует ABI (иногда называемый Kernel Binary Interface (KBI)), который включает семантику и расположение публичных структур данных, а также расположение и семантику аргументов публичных функций внутри самого ядра.

Ядро FreeBSD может поддерживать несколько пользовательских ABI. Например, ядро FreeBSD amd64 поддерживает пользовательские ABI FreeBSD amd64 и i386, а также пользовательские ABI Linux x86_64 и i386. Ядро FreeBSD должно поддерживать "родной" ABI в качестве ABI по умолчанию. "Родной" ABI, как правило, разделяет определённые свойства с ABI ядра, такие как соглашение о вызовах в C, размеры базовых типов и т.д.

Уровни определены как для ядер, так и для пользовательских ABI. В общем случае ядро платформы и ABI FreeBSD назначаются на один и тот же уровень.

21.2.1. Уровень 1: Полностью поддерживаемые архитектуры

Уровень 1 включает наиболее зрелые платформы FreeBSD. Они поддерживаются офицером

безопасности, инженерами выпуска и командой управления портами. Ожидается, что архитектуры Уровня 1 соответствуют производственному качеству во всех аспектах операционной системы FreeBSD, включая среды установки и разработки.

Проект FreeBSD предоставляет следующие гарантии пользователям платформ Уровня 1:

- Официальные образы релизов FreeBSD будут предоставлены командой разработки релизов.
- Двоичные обновления и исправления исходного кода для Security Advisories и Errata Notices будут предоставляться для поддерживаемых выпусков.
- Исходные патчи для бюллетеней безопасности будут предоставлены для поддерживаемых веток.
- Двоичные обновления и исправления исходного кода для кросс-платформенных выпусков Security Advisories обычно предоставляются во время объявления.
- Изменения в пользовательских ABI, как правило, включают прослойки совместимости (shim) для обеспечения корректной работы бинарных файлов, скомпилированных для любой стабильной ветки, где платформа относится к Уровню 1. Эти прослойки могут быть отключены в стандартной установке. Если прослойки совместимости не предоставляются для изменения ABI, их отсутствие будет явно указано в примечаниях к выпуску.
- Изменения в определённых частях ABI ядра будут включать прослойки совместимости для обеспечения корректной работы модулей ядра, скомпилированных для самой старой поддерживаемой версии в ветке. Обратите внимание, что не все части ABI ядра защищены.
- Официальные бинарные пакеты для стороннего программного обеспечения будут предоставлены командой портов. Для встраиваемых архитектур эти пакеты могут быть кросс-собранными на архитектуре другого типа.
- Наиболее подходящие порты должны либо собираться, либо иметь соответствующие фильтры, чтобы предотвратить сборку неподходящих.
- Новые функции, которые не являются специфичными для платформы, будут полностью работоспособны на всех архитектурах Уровня 1.
- Функции и прослойки совместимости, используемые программами, скомпилированными для старых стабильных веток, могут быть удалены в новых основных версиях. Такие изменения будут четко документированы в примечаниях к выпуску.
- Уровень 1 платформ должен быть полностью документирован. Основные операции будут описаны в Руководстве FreeBSD.
- Уровень 1 платформ будет включен в дерево исходных кодов.
- Платформы Уровня 1 должны быть самодостаточными, используя либо встроенный инструментарий, либо внешний инструментарий. Если требуется внешний инструментарий, будут предоставлены официальные бинарные пакеты для него.

Для обеспечения зрелости платформ Уровня 1 проект FreeBSD будет поддерживать следующие ресурсы для разработки:

- Сборка и автоматизация тестирования поддерживаются либо в кластере FreeBSD.org, либо в другом месте, легко доступном для всех разработчиков. Для встраиваемых платформ можно использовать эмулятор, доступный в кластере FreeBSD.org, вместо реального оборудования.
- Включение в цели `make universe` и `make tinderbox`.
- Выделенное оборудование в одном из кластеров FreeBSD для сборки пакетов (нативно или через qemu-user).

Совокупно, разработчики должны обеспечить следующее для поддержания статуса платформы Уровня 1:

- Изменения в дереве исходного кода не должны заведомо нарушать сборку платформы Уровня 1.
- Уровень 1 архитектур должен обладать зрелой, здоровой экосистемой пользователей и активных разработчиков.
- Разработчики должны иметь возможность собирать пакеты на широко доступных, невстраиваемых системах Уровня 1. Это может означать как нативные сборки, если невстраиваемые системы широко доступны для рассматриваемой платформы, так и кросс-сборки, выполняемые на другой архитектуре Уровня 1.
- Изменения не должны нарушать ABI пользовательского пространства. Если изменение ABI необходимо, совместимость ABI для существующих бинарных файлов должна обеспечиваться с помощью версионирования символов или увеличения версий разделяемых библиотек.
- Изменения, которым сделано слияние в стабильные ветки, не должны нарушать защищенные части ABI ядра. Если требуется изменение ABI ядра, изменение должно быть модифицировано для сохранения функциональности существующих модулей ядра.

21.2.2. Уровень 2: Развивающиеся и нишевые архитектуры

Уровень 2 включает в себя функциональные, но менее развитые платформы FreeBSD. Они не поддерживаются офицером безопасности, инженерами выпуска и командой управления портами.

Платформы Уровня 2 могут быть кандидатами в платформы Уровня 1, которые всё ещё находятся в активной разработке. Архитектуры, достигшие конца жизненного цикла, также могут быть переведены из статуса Уровня 1 на Уровень 2 по мере сокращения доступности ресурсов для поддержания системы в состоянии производственного качества. Хорошо поддерживаемые нишевые архитектуры также могут относиться к Уровню 2.

Проект FreeBSD предоставляет следующие гарантии пользователям платформ Уровня 2:

- Инфраструктура портов должна включать базовую поддержку архитектур Уровня 2, достаточную для сборки портов и пакетов. Это включает поддержку базовых пакетов, таких как ports-mgmt/pkg, но нет гарантии, что произвольные порты будут собираемыми или работоспособными.
- Новые функции, которые не являются специфичными для платформы, должны быть

реализуемы на всех архитектурах уровня Уровень 2, если они не реализованы.

- Уровень 2 платформ будет включен в дерево исходных кодов.
- Платформы Уровня 2 должны быть самодостаточными, используя либо встроенный инструментарий, либо внешний инструментарий. Если требуется внешний инструментарий, будут предоставлены официальные бинарные пакеты для него.
- Уровень 2 платформ должен обеспечивать функциональные ядра и пользовательские среды, даже если официальный дистрибутив не предоставляется.

Для поддержания зрелости платформ Уровня 2 проект FreeBSD будет поддерживать следующие ресурсы для разработки:

- Включение в цели `make universe` и `make tinderbox`.

Совместно разработчики должны обеспечить следующее для поддержания статуса платформы Уровня 2:

- Изменения в дереве исходного кода не должны заведомо нарушать сборку платформы Уровня 2.
- Уровень 2 архитектур должен иметь активную экосистему пользователей и разработчиков.
- Хотя изменения, нарушающие ABI пользовательского пространства, допустимы, не следует делать это без веской причины. Значительные изменения ABI пользовательского пространства должны ограничиваться основными версиями.
- Новые функции, которые ещё не реализованы в архитектурах Уровня 2, должны предоставлять возможность их отключения на этих архитектурах.

21.2.3. Уровень 3: Экспериментальные архитектуры

Уровень 3 включает платформы с частичной поддержкой FreeBSD. Они *не* поддерживаются офицером безопасности, инженерами выпуска релизов и командой управления портами.

Уровень 3 включает архитектуры на ранних стадиях разработки, предназначенные для неосновных аппаратных платформ или считающиеся устаревшими системами, которые вряд ли получат широкое применение в будущем. Первоначальная поддержка платформ Уровня 3 может находиться в отдельном репозитории, а не в основном репозитории исходного кода.

Проект FreeBSD не предоставляет никаких гарантий пользователям платформ Уровня 3 и не обязуется выделять ресурсы для поддержки разработки. Платформы Уровня 3 могут не всегда собираться, и ни одно ABI ядра или пользовательского пространства не считается стабильным.

21.2.4. Неподдерживаемые архитектуры

Другие платформы не поддерживаются проектом в какой-либо форме. Ранее проект описывал их как системы Уровня 4.

После перехода платформы в статус неподдерживаемой, вся поддержка платформы удаляется из исходного кода, портов и документации. Обратите внимание, что поддержка в портах должна сохраняться до тех пор, пока платформа поддерживается в ветке, поддерживаемой портами.

21.3. Политика изменения уровня архитектуры

Системы могут быть перемещены с одного уровня на другой только с одобрения Основной команды FreeBSD, которая принимает это решение совместно с Security Officer, Release Engineering и командами управления портами. Для перевода платформы на более высокий уровень все недостающие гарантии поддержки должны быть выполнены до завершения повышения.

22. Специфичные FAQ по портам

22.1. Добавление нового порта

22.1.1. Как добавить новый порт?

Добавление порта в дерево относительно просто. Как только порт готов к добавлению, как описано далее здесь, необходимо добавить запись каталога порта в Makefile соответствующей категории. В этом Makefile порты перечислены в алфавитном порядке и добавлены в переменную `SUBDIR`, например:

```
    SUBDIR += newport
```

После того как порт и Makefile его категории готовы, новый порт можно закоммитить:

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

Не забудьте настроить git-хуки для дерева портов; специальный хук был разработан для проверки Makefile каждой категории.

22.1.2. Есть ли что-то ещё, что мне нужно знать при добавлении нового порта?

Проверьте порт, желательно убедиться, что он компилируется и собирается в пакеты правильно.

В главе Руководства портировщика по тестированию содержатся более подробные инструкции. См. разделы Portclippy / Portfmt и poudriere.

Вам не обязательно устранять все предупреждения, но убедитесь, что исправили простые.

Если порт поступил от отправителя, который ранее не участвовал в Проекте, добавьте имя этого человека в раздел Дополнительные участники списка участников FreeBSD.

Закройте PR, если порт был отправлен как PR. Чтобы закрыть PR, измените состояние на `Issue Resolved` и установите решение как `Fixed`.

Если по какой-то причине использование poudriere для тестирования нового порта невозможно, минимальный набор тестирования включает следующую последовательность:

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

Обратите внимание, что poudriere является эталоном для сборки пакетов. Если порт не собирается в poudriere, он будет удалён.

22.2. Удаление существующего порта

22.2.1. Как удалить существующий порт?

Сначала прочитайте раздел о копиях репозиториев. Перед удалением порта необходимо убедиться, что от него не зависят другие порты.

- Убедитесь, что нет зависимостей от порта в коллекции портов:
  - Имя пакета порта (PKGNAME) появляется ровно в одной строке в последнем файле INDEX.
  - Ни один другой порт не содержит ссылок на каталог порта или PKGNAME в своих Makefiles

При использовании Git рассмотрите возможность использования git-grep(1), так как он значительно быстрее, чем `grep -r`.

- Затем удалите порт:
  - Удалите файлы и каталог порта с помощью `git rm`.
  - Удалите указание `SUBDIR` для порта в Makefile родительского каталога.
  - Добавьте запись в ports/MOVED.
  - Удалите порт из ports/LEGAL, если он там присутствует.

Или вы можете использовать скрипт rmport из ports/Tools/scripts. Этот скрипт был написан Vasil Dimov <vd@FreeBSD.org>. При отправке вопросов об этом скрипте в Список рассылки, посвящённый Портам FreeBSD, пожалуйста, также копируйте Chris Rees <crees@FreeBSD.org>, текущего сопровождающего.

22.3. Как переместить порт в новое место?

1. Выполните тщательную проверку коллекции портов на наличие зависимостей от старого расположения или имени порта и обновите их. Запуск `grep` по файлу INDEX недостаточен, поскольку некоторые порты имеют зависимости, включённые через параметры компиляции. Рекомендуется выполнить полный поиск с помощью git-grep(1) по коллекции портов.
2. Удалите запись `SUBDIR` из Makefile старой категории и добавьте запись `SUBDIR` в Makefile новой категории.
3. Добавьте запись в ports/MOVED.
4. Поищите записи в xml-файлах внутри ports/security/vuxml и скорректируйте их соответствующим образом. В частности, проверьте предыдущие пакеты с новым именем, версия которых может включать новый порт.
5. Переместите порт с помощью `git mv`.
6. Зафиксируйте изменения.

22.4. Как скопировать порт в новое место?

1. Скопируйте порт с помощью `cp -R old-cat/old-port new-cat/new-port`.
2. Добавьте новый порт в файл new-cat/Makefile.
3. Измените содержимое в new-cat/new-port.
4. Зафиксируйте изменения.

22.5. Заморозка портов

22.5.1. Что такое «заморозка портов»?

«Заморозка портов» (ports freeze) — это ограниченное состояние, в которое дерево портов переводилось перед выпуском релиза. Оно использовалось для обеспечения более высокого качества пакетов, поставляемых с релизом. Обычно это длилось несколько недель. В течение этого времени исправлялись проблемы со сборкой, а также строились пакеты для релиза. Эта практика больше не применяется, так как пакеты для релизов теперь собираются из текущей стабильной квартальной ветки.

Для получения дополнительной информации о том, как делать слияние коммитов в квартальную ветку, см. Какова процедура получения разрешения на слияние коммита с

квартальной веткой?.

22.6. Квартальные ветки

22.6.1. Какова процедура получения разрешения на слияние коммита с квартальной веткой?

По состоянию на 30 ноября 2020 года явное одобрение для внесения изменений в квартальную ветку не требуется.

22.6.2. Какова процедура слияния коммитов в квартальную ветку?

Слияние коммитов в квартальную ветку (процесс, который мы по историческим причинам называем MFH) очень похоже на MFC коммитов в репозитории src, поэтому в основном:

```
% git checkout 2021Q2
% git cherry-pick -x $HASH
(verify everything is OK, for example by doing a build test)
% git push
```

где `$HASH` — это хэш коммита, который вы хотите скопировать в квартальную ветку. Параметр `-x` гарантирует, что хэш `$HASH` из ветки `main` будет включён в новое сообщение коммита в квартальной ветке.

22.7. Создание новой категории

22.7.1. Какова процедура создания новой категории?

Пожалуйста, ознакомьтесь с Предложение новой категории в Руководстве FreeBSD по созданию портов. После выполнения этой процедуры и назначения PR в группе — Группа Менеджеров Дерева Портов FreeBSD <portmgr@FreeBSD.org>, решение об одобрении принимается ими. Если решение положительное, их обязанностью является:

1. Выполните все необходимые перемещения. (Это применимо только к физическим категориям.)
2. Обновите определение `VALID_CATEGORIES` в файле ports/Mk/bsd.port.mk.
3. Назначить PR обратно на вас.

22.7.2. Что мне нужно сделать для создания новой физической категории?

1. Обновите Makefile каждого перемещенного порта. Пока не подключайте новую категорию к сборке.

Для этого вам потребуется:

1. Измените `CATEGORIES` порта (это была цель упражнения, помните?). Новая категория указана первой. Это поможет убедиться, что PKGORIGIN указан правильно.
2. Выполните команду `make describe`. Поскольку команда `make index` верхнего уровня, которую вы запустите через несколько шагов, является итерацией `make describe` для всей иерархии портов, обнаружение ошибок на этом этапе сэкономит время, избавив от необходимости перезапускать этот шаг позже.
3. Если вы хотите быть действительно тщательным, сейчас может быть подходящее время запустить portlint(1).

2. Проверьте, что значения `PKGORIGIN` указаны верно. Система портов использует запись `CATEGORIES` каждого порта для создания его `PKGORIGIN`, который служит для связи установленных пакетов с каталогом порта, из которого они были собраны. Если эта запись неверна, такие распространённые инструменты для работы с портами, как pkg-version(8) и portupgrade(1), не будут работать.

   Для этого используйте инструмент chkorigin.sh: `env PORTSDIR=/path/to/ports sh -e /path/to/ports/Tools/scripts/chkorigin.sh`. Это проверит каждый порт в дереве портов, даже те, которые не связаны со сборкой, поэтому вы можете запустить его сразу после операции перемещения. Подсказка: не забудьте проверить `PKGORIGIN` для всех подчинённых портов (slave ports) тех портов, которые вы только что переместили!

3. На вашей локальной системе протестируйте предлагаемые изменения: сначала закомментируйте записи SUBDIR в файлах Makefile старых категорий портов; затем включите сборку новой категории в ports/Makefile. Запустите `make checksubdirs` в затронутых каталогах категорий, чтобы проверить записи SUBDIR. Затем в каталоге ports/ выполните `make index`. Это может занять более 40 минут даже на современных системах; однако это необходимый шаг для предотвращения проблем у других пользователей.
4. После этого можно зафиксировать обновлённый файл ports/Makefile, чтобы подключить новую категорию к сборке, а также сделать коммит изменениям в файле Makefile для старой категории или категорий.
5. Добавьте соответствующие записи в ports/MOVED.
6. Обновите документацию, изменив:
   - список категорий в Руководстве FreeBSD по созданию портов
7. Только после того, как все вышеперечисленное будет выполнено и больше никто не сообщает о проблемах с новыми портами, старые порты следует удалить из их прежних мест в репозитории.

22.7.3. Что мне нужно сделать для создания новой виртуальной категории?

Это намного проще, чем физическая категория. Требуется всего несколько изменений:

- список категорий в Руководстве FreeBSD по созданию портов

22.8. Разные вопросы

22.8.1. Существуют ли изменения, которые можно зафиксировать без запроса одобрения у сопровождающего?

Общее одобрение для большинства портов применяется к следующим типам исправлений:

- Большинство изменений инфраструктуры порта (то есть модернизация без изменения функциональности). Например, это включает переход на новые макросы `USES`, включение подробных сборок и переход на новые синтаксисы системы портов.
- Тривиальные и *проверенные* исправления для сборки и выполнения.
- Документация или изменения метаданных для портов, такие как pkg-descr или `COMMENT`.

Исключениями являются любые компоненты, поддерживаемые Группа Менеджеров Дерева Портов FreeBSD <portmgr@FreeBSD.org> или Команда Офицера Безопасности <security-officer@FreeBSD.org>. Несанкционированные коммиты в порты, которые поддерживаются этими группами, недопустимы.

22.8.2. Как узнать, мой порт собирается правильно или нет?

Пакеты собираются несколько раз в неделю. Если сборка порта завершается неудачно, сопровождающий получит письмо от `pkg-fallout@FreeBSD.org`.

Отчёты по всем сборкам пакетов (официальные, экспериментальные и без регрессии) агрегируются на pkg-status.FreeBSD.org.

22.8.3. Я добавил новый порт. Нужно ли добавлять его в INDEX?

Нет. Файл может быть создан выполнением команды `make index`, или можно загрузить предварительно сгенерированную версию с помощью `make fetchindex`.

22.8.4. Есть ли другие файлы, которые мне нельзя изменять?

Любой файл непосредственно в ports/ или любой файл в подкаталоге, название которого начинается с заглавной буквы (Mk/, Tools/ и т.д.). В частности, Группа Менеджеров Дерева Портов FreeBSD <portmgr@FreeBSD.org> очень ревностно относится к ports/Mk/bsd.port*.mk, поэтому не вносите изменения в эти файлы, если не хотите навлечь на себя их гнев.

22.8.5. Как правильно обновить контрольную сумму для distfile порта, если файл изменился без изменения версии?

Когда контрольная сумма файла дистрибутива обновляется из-за того, что автор обновил файл без изменения ревизии порта, сообщение коммита включает сводку соответствующих различий между оригинальным и новым файлом дистрибутива, чтобы убедиться, что файл дистрибутива не был повреждён или злонамеренно изменён. Если текущая версия порта находится в дереве портов уже некоторое время, копия старого файла дистрибутива обычно доступна на FTP-серверах; в противном случае следует связаться с автором или сопровождающим, чтобы выяснить, почему файл дистрибутива изменился.

22.8.6. Как можно запросить экспериментальную тестовую сборку дерева портов (exp-run)?

Перед коммитом изменений, оказывающих значительное влияние на порты, необходимо выполнить тестовый прогон (exp-run). Исправление может относиться как к дереву портов, так и к базовой системе.

Полные сборки пакетов будут выполнены с патчами, предоставленными отправителем, и отправитель обязан исправить обнаруженные проблемы *(падения сборки и т.п.)* перед коммитом.

1. Перейдите по ссылке Страница создания новой PR в Bugzilla.
2. Выберите продукт, к которому относится ваш патч.
3. Заполните отчёт об ошибке как обычно. Не забудьте приложить исправление.
4. Если вверху написано «Показать дополнительные поля (Show Advanced Fields)», нажмите на это. Теперь будет написано «Скрыть дополнительные поля (Hide Advanced Fields)». Станут доступны многие новые поля. Если уже написано «Скрыть дополнительные поля», ничего делать не нужно.
5. В разделе «Flags» установите флаг «exp-run» в значение ?. Для всех остальных полей при наведении курсора на любое поле отображаются дополнительные сведения.
6. Отправьте. Дождитесь завершения сборки.
7. Группа Менеджеров Дерева Портов FreeBSD <portmgr@FreeBSD.org> ответит с информацией возможных падениях.
8. В зависимости от последствий:
    - Если последствий нет, процедура останавливается здесь, и изменение может быть закоммичено, ожидая любых других необходимых утверждений.
        i. Если возникнут проблемы, их *необходимо* исправить, либо напрямую в дереве портов, либо добавив изменения в предоставленный патч.
        ii. После этого вернитесь к шагу 6, указав, что проблема устранена, и дождитесь повторного запуска exp-run. Повторяйте, пока остаются неисправные порты.

23. Проблемы, характерные для разработчиков, не являющихся коммиттерами

Несколько людей, имеющих доступ к машинам FreeBSD, не обладают правами на коммиты. Почти все положения этого документа применимы и к таким разработчикам (за исключением аспектов, специфичных для коммитов и связанного с ними членства в рассылках). В частности, мы рекомендуем ознакомиться со следующим:

- Административные детали
- Для всех

Попросите вашего наставника добавить вас в список "Дополнительные участники" (doc/shared/contrib-additional.adoc), если вас там ещё нет.

- Отношения с разработчиками
- Руководство по быстрому началу работы с SSH
- Большой список правил коммиттеров FreeBSD

24. Информация о Google Analytics

По состоянию на 12 декабря 2012 года на сайте проекта FreeBSD был включен Google Analytics для сбора анонимной статистики использования сайта.

По состоянию на 3 марта 2022 года Google Analytics был удалён из проекта FreeBSD.

25. Разные вопросы

25.1. Как получить доступ к people.FreeBSD.org для размещения личной или проектной информации?

`people.FreeBSD.org` — это то же самое, что и `freefall.FreeBSD.org`. Просто создайте каталог public_html. Всё, что вы поместите в этот каталог, будет автоматически доступно по адресу https://people.FreeBSD.org/.

25.2. Где хранятся архивы списков рассылки?

Списки рассылки архивируются в /local/mail на `freefall.FreeBSD.org`.

25.3. Я хочу стать наставником нового коммиттера. Какой процесс мне нужно пройти?

См. документ Процедура создания новой учётной записи на внутренних страницах.

26. Преимущества и привилегии для коммиттеров FreeBSD

26.1. Признание

Признание в качестве квалифицированного инженера-программиста — это самая долговечная ценность. Кроме того, возможность работать с одними из лучших специалистов, о встрече с которыми мечтает любой инженер, — это отличный бонус!

26.2. FreeBSD Mall

Коммиттеры FreeBSD могут бесплатно получить набор из 4 CD или DVD на конференциях через FreeBSD Mall, Inc..

26.3. Gandi.net

Gandi предоставляет услуги хостинга веб-сайтов, облачных вычислений, регистрации доменов и сертификатов X.509.

Gandi предоставляет скидку E-rate всем разработчикам FreeBSD. Чтобы упростить процесс получения скидки, сначала создайте учётную запись Gandi, заполните платежные данные и выберите валюту. Затем отправьте письмо по адресу non-profit@gandi.net с вашего адреса `@freebsd.org`, указав ваш идентификатор Gandi.

26.4. rsync.net

rsync.net предоставляет облачное хранилище для резервного копирования вне офиса, оптимизированное для пользователей UNIX. Их сервис полностью работает на FreeBSD и ZFS.

rsync.net предоставляет бесплатный аккаунт на 500 ГБ навсегда разработчикам FreeBSD. Просто зарегистрируйтесь по адресу https://www.rsync.net/freebsd.html, используя ваш адрес `@freebsd.org`, чтобы получить этот бесплатный аккаунт.