Инструкция по работе с системой управления версиями

Версия 1.4

Март 2016 г.

НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

Документ определяет набор правил, которыми следует руководствоваться при работе с используемой в отделе разработки дирекции по производству направления связи системой управления версиями.

В первой части документа приведены общие правила использования системы управления версиями.

Во второй части описаны основные команды и параметры настройки используемой системы управления версиями.

Следование требованиям настоящего документа является обязательным с момента утверждения при работе над существующими и новыми программными проектами.

СОДЕРЖАНИЕ

НОРМАТИВНЫЕ ССЫЛКИ

В настоящей инструкции использованы нормативные ссылки на следующие инструкции отдела разработки дирекции по производству направления связи:

Имя документа	Версия	Расположение на портале ОР ДП НС
Правила оформления исходных кодов программного обеспечения	4.0	EEW3SF4JMF7V-184-3978
Правила сборки и тестирования стабильных версий	1.0	EEW3SF4JMF7V-184-3974

Если ссылочный документ заменён (изменён), то при пользовании настоящей инструкцией следует руководствоваться заменённым (изменённым) документом. Если ссылочный документ отменён без замены, то положение, в котором дана ссылка на него, применяется в части, не затрагивающей эту ссылку.

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Термин	Термин, англ.	Определение
—	HEAD	Ссылка на состояние текущей *ветви* (*вершину текущей ветви* либо некое её произвольное *зафиксированное состояние*), относительно которого в ветвь вносятся текущие изменения
Вершина ветви	Head	*Именованная ссылка* на последнее *зафиксированное состояние ветви*
Ветвь	Branch	Независимое направление разработки
Двоичный нетипизированный объект	Blob object	Содержимое файла
Зафиксированное состояние	Commit	Набор изменений в *ветви*, описывающий отличие текущего состояния содержимого ветви от предыдущего, хранимый в виде *объекта* и формирующий, таким образом, историю разработки
Именованная ссылка	Ref	Хранимое в репозитории *имя объекта*
Имя объекта	Object name	Уникальный идентификатор *объекта*, представляющий собой 40-разрядное шестнадцатеричное значение хэш-функции SHA1 от его содержимого
Индекс	Index	Сохранённое в репозитории в виде *объектов,* подготовленное к *фиксирования*, но ещё не зафиксированное состояние *рабочей копии*
Объект	Object	Неизменяемая уникальная единица хранения в системе управления версиями, однозначно адресуемая *именем объекта*
Перемещение истории разработки	Rebase	Перемещение последовательности *зафиксированных состояний* на вершину другого зафиксированного состояния и обновление *вершины ветви*, в которую было произведено перемещение, таким образом, чтобы та указывала на последнее состояние полученной в результате ветви
Подмодуль		Проект, включаемый в проект более высокого уровня
Получение удалённой ветви	Fetch	Загрузка *вершины соответствующей ветви* из удалённого репозитория, определение недостающих в локальной ветви объектов и загрузка этих объектов в локальную ветвь
Проект		набор файлов, реализующих законченную функциональность.
Рабочая копия	Working tree	Состояние *ветви* в каждый момент времени, представляющее собой совокупность *зафиксированной* истории разработки и незафиксированных изменений
Репозиторий	Repository	Совокупность файлов и каталогов проекта, находящихся под управлением СУВ, и служебных данных СУВ, из которых формируется история разработки, ветви, значимые этапы и отдельные *зафиксированные состояния* проекта

Родитель	Parent	*Зафиксированное состояние*, содержащее список непосредственных логических предшественников другого зафиксированного состояния в истории разработки
Синхронизация локальной ветви	Pull	*Получение* удалённой *ветви* и *слияние* её с локальной
Синхронизация удалённой ветви	Push	Загрузка *вершины удалённой ветви*, определение недостающих в удалённой ветви объектов и загрузка этих объектов в удалённую ветвь
Следящая ветвь	Remote tracking branch	*Ветвь*, отображающая изменения из соответствующей ей ветви *удалённого репозитория*
Слияние ветвей	Merge	1) Процесс объединения содержимого и истории заданной *ветви* с содержимым и историей текущей ветви 2) *Зафиксированное состояние*, представляющее результат слияния двух ветвей и имеющее в качестве *родителей вершины слитых ветвей*
Удалённый («вышестоящий») репозиторий	Origin	Репозиторий, стоящий в логической иерархии выше локального репозитория пользователя
Фиксирование изменений	Commit	Процесс сохранения *зарегистрированных изменений* в истории разработки
Этап в истории разработки	Tag	*Именованная ссылка* на *зафиксированное состояние*, являющееся важным этапов в истории разработки с точки зрения разработчика
Примечание. Курсивом выделены понятия, определения которых также даны в таблице.		

ИСПОЛЬЗУЕМЫЕ СОКРАЩЕНИЯ

Сокращение	Значение	Сокращение, англ.	Значение, англ.
ОР ТД НС	Отдел разработки технической дирекции направления связи	-	-
ОС	Операционная система	OS	Operating System
СУВ	Система управления версиями	VCS	Version Control System

1 ОБЩИЕ ПРАВИЛА

1.1 Фиксирование изменений

1.1.1 Принятие решения о фиксировании изменения

Каждое фиксируемое в репозитории изменение должно быть логически цельным, неделимым и независимым от других. Программный код в любом зафиксированном состоянии репозитория должен быть работоспособным.

Используемая СУВ позволяет выборочно регистрировать и фиксировать отдельные изменения в редактируемых файлах. Это даёт возможность оформить историю разработки закрытой экспериментальной ветви в виде последовательности логически разделённых зафиксированных изменений перед слиянием с публичной ветвью даже в том случае, когда разделить и последовательно фиксировать изменения в процессе работы было невозможно.

Основным критерием при фиксировании изменения является его логическая целостность, связность и неделимость, а не время работы над ним или его размер. Не следует избегать фиксирования состояний, изменяющих один символ в исходном коде, если такое изменение уместно.

Каждое фиксируемое состояние должно вносить в исходный код как можно меньше изменений; все изменения должны быть необходимы и обоснованы.

Не следует фиксировать в репозитории закомментированный исходный код. Такой код должен удаляться из проекта ещё до принятия решения о фиксировании изменений.

В используемой СУВ есть возможность проверки корректности расстановки пробелов, символов табуляции и пустых строк. Следует настроить СУВ на проверку фиксируемых изменений на соответствие положениям 1.1.4.1 и 1.1.6.4 Правил оформления исходных кодов программного обеспечения.

1.1.2 Документирование зафиксированных состояний репозитория

Любое изменение состояния репозитория, требующее документирования согласно логике используемой СУВ, должно быть документировано.

Комментарий должен быть оформлен, по возможности, в виде прозы на русском или английском языке. Длина строк комментария должна быть ограничена 80 символами.

Первая строка комментария должна содержать название системы и подсистемы, затрагиваемых изменением, с кратким описанием вносимого изменения. Перенос описания на вторую строку не допускается.

Основной текст комментария отделяется от первой строки одной пустой строкой. В нём содержится описание изменения, достаточно подробное и развёрнутое для того, чтобы не вызывать вопросов по существу изменения. В основной текст уместно помещать сообщения об ошибках, вывод инструментальных средств, стек вызовов, тексты коротких программ и прочие сведения, способствовавшие выявлению и устранению ошибок и улучшению. Комментарии к истории разработки дополняют пространственное документирование кода временн*ы*м, обосновывая существующий исходный код и подтверждая его комментарии. В особых случаях, при безусловной очевидности и тривиальности вносимых изменений, допускается документирование изменения однострочным комментарием.

В случае, когда в фиксируемом изменении содержится исправление ошибки или реализация функциональности, зарегистрированной в системе отслеживания ошибок, номер закрытого сообщения об ошибке (запроса функциональности) должен быть указан в комментарии к изменению.

Пример оформления комментария к изменению приведён на Рисунок 1.

```
Phoenix shell/GPSGLNS: сигнал/шум 0 дБ в предложениях GSV имеет особое значение

Исправление ошибки r2dtdcd#44444.

Согласно NMEA 0183, значение отношения сигнал/шум в предложениях GSV кодируется
двузначным десятичным числом. При этом значение 0 дБ может указывать на то, что
приёмник получает сигнал от КА, но не использует его в расчётах. МНП-Н7
формирует предложения GSV именно таким образом.

Ранее это не учитывалось, в пользовательском интерфейсе отображались
некорректные данные.
```

Рисунок 1 – Пример оформления комментария к фиксируемому изменению

1.2 Хранение данных

1.2.1 Хранение в репозитории сгенерированных данных

Не следует хранить в репозитории данные, которые могут быть получены из или с помощью других находящихся в репозитории данных. Так, не следует хранить в репозитории исполняемые файлы, которые могут быть получены в результате сборки находящегося в репозитории программного проекта, либо данные, формируемые находящейся в репозитории программой, при условии, что результат работы программы постоянен при одних и тех же входных данных.

1.2.2 Хранение в репозитории двоичных данных

При необходимости хранения в репозитории двоичных файлов по возможности следует настроить СУВ таким образом, чтобы при запросе на отображение изменений между двумя зафиксированными состояниями она выполняла преобразование двоичных файлов в текстовой формат. Это, например, позволит отслеживать изменения содержимого файлов в форматах PDF и Microsoft Word.

1.3 Сопровождение проектов и версий

1.3.1 Управление репозиториями

Используемая СУВ относится к классу распределённых. Центральный репозиторий и репозитории разработчиков технически равноправны и не отличаются друг от друга. Основное назначение центрального репозитория – хранение актуального состояния основной ветви разрабатываемого проекта и публичных ветвей, в которых идёт совместная работа, которая не может быть выполнена силами одного человека, а так же подготовка стабильных и, при необходимости, экспериментальных версий разрабатываемого программного обеспечения.

1.3.2 Управление ветвями

Одним из фундаментальных принципов работы с распределёнными СУВ является активное использование ветвей в процессе разработки.

Использование ветвей позволяет выделять логически связанные последовательные серии изменений, например, исправление архитектурных ошибок или реализация новой функциональности, в независимые направления разработки, удаляя или вливая их в основную ветвь по готовности.

Не следует загружать в центральный репозиторий экспериментальные закрытые ветви, не предназначенные для совместной работы.

Экспериментальные ветви, работа над которыми прекращена, следует своевременно удалять. Стабильные ветви удалению не подлежат.

Следует своевременно проводить промежуточное слияние каждой активной ветви и ветви, являющейся предком по отношению к ней. Это существенно облегчит и ускорит слияние экспериментальной ветви с основной, когда работа будет завершена.

1.3.3 Ведение текущих, экспериментальных и стабильных ветвей разработки

При работе над проектами принимается принцип разделения текущей и стабильных *ветвей разработки*, ведущихся в различных *ветвях репозитория*.

В основной ветви репозитория (`master`) ведётся текущая разработка будущих версий проекта. Изменения в неё попадают слиянием с ней экспериментальной ветви после того, как изменения, реализованные в последней, были признаны годными к включению в состав проекта. Стабилизация значительных версий проекта (1.1, 1.2, 2.0 и т.д.) ведётся в отдельных стабильных ветвях.

Решение об отделении очередной стабильной ветви принимается по готовности в основной ветви репозитория каких-либо значительных изменений, в соответствии с утверждённым планом развития проекта. При достижении состояния готовности к выпуску очередной версии (1.1.0, 2.3.4 и т.д.) в соответствующей стабильной ветви создаётся *этап в истории разработки* (см. п. 2.16) с присвоением ему соответствующего названия (например, v1.1.1, v2.3.4 и т.д.). Формирование двоичных образов программного обеспечения или аппаратуры (для реализации на ПЛИС), предназначенных для установки на сопровождаемых Отделом разработки изделиях, допускается только на основе созданных таким образом этапов.

Ошибки, затрагивающие несколько ветвей (как стабильных, так и основную), одновременно исправляются на всех поддерживаемых ветвях, где они проявляются.

При этом допускается применение к различным ветвям различных исправлений одной и той же ошибки, когда это необходимо.

Процедура выпуска версий программного обеспечения и описания аппаратуры приведена в документе «Правила сборки и тестирования стабильных версий».

1.3.4 Публикация экспериментальных ветвей

Согласно пп. 1.3.4 настоящей Инструкции, единственным способом продвижения изменений в основную ветвь репозитория является слияние содержащей эти изменения экспериментальной ветви с основной ветвью. Согласно документу «Правила сборки и тестирования стабильных версий», процесс обсуждения продвигаемых изменений в отделе является публичным, и самовольное включение изменений в основную ветвь их автором запрещено.

Публикация экспериментальной ветви может выполняться разработчиком для достижения одной из следующих целей:

– немедленное продвижение изменений в основную ветвь репозитория;

– предварительное обсуждение, получение предварительных комментариев коллег для дальнейшей самостоятельной работы над разрабатываемыми изменениями, не готовыми для немедленного продвижения в основную ветвь;

– публикация текущего (промежуточного) состояния проекта при приостановке работы над ним.

Во всех случаях публикация экспериментальной ветви должна сопровождаться комментарием, оформленным в соответствии с требованиями пп. 1.1.2 настоящей Инструкции. Дополнительно:

– в случае публикации ветви с целью немедленного изменения первая строка комментария предваряется префиксом `[MERGE]`;

– в случае публикации ветви с целью получения предварительного обсуждения первая строка комментария предваряется префиксом `[RFC]`;

– в случае публикации ветви с целью фиксирования текущего состояния при приостановке работы над проектом первая строка комментария предваряется префиксом `[WIP]`.

Зачастую, изменения проходят несколько этапов (итераций) рецензирования. Если публикация экспериментальной ветви производится не впервые, к префиксу в первой строке добавляется номер этапа рецензирования; за основным текстом комментария следует список изменений, внесённых в ветвь по результатам каждого этапа рецензирования.

Если экспериментальная ветвь содержит несколько патчей, в целях упрощения обсуждения может быть полезно указать назначение каждого из них (индивидуально или в составе логически выделенных групп).

Пример оформления комментария к экспериментальной ветви приведён на Рисунок 2.

```
[RFC v4] Устранение некорректных случаев использования API Net-SNMP

В настоящее время интерфейс Net-SNMP используется таким образом, что подсистема
опроса состояния оконечного оборудования оказывается подвержена состояниям
гонки (issue #42).

Два первых патча приводят использование интерфейса Net-SNMP в подсистеме опроса
в соответсвтие с документацией, изменяя порядок инициализации Net-SNMP
и добавляя необходимую синхронизацию.

Два последних патча заменяют последовательности вызова процедур Net-SNMP более
простыми там, где это возможно, с целью упрощения сопровождения кода
в дальнейшем; они не вносят функциональных изменений.

После применения этой последовательности следующие тесты больше не проходят,
возвращая XFAIL:

  testsuite/issue4-1.t testsuite/issue4-2.t testsuite/issue4-3.t
  testsuite/issue4-4.t testsuite/foo-1.t testsuite/bar-5.t

Испрвление, решающее проблему с тестами, планируется, но пока не протестировано.

v3 -> v4
- добавлены патчи №№ 3, 4 с упрощением использования API
- добавлены дополнительные тесты

v2 -> v3
- исправлены ошибки во вновь добавленных тестах
- патчи переупорядочены

v1 -> v2
- добавлены дополнительные тесты
- учтены замечания по оформлению
```

Рисунок 2 – Пример оформления комментария к экспериментальной ветви

2 ТИПОВЫЕ ОПЕРАЦИИ

Положения настоящего раздела не являются документацией к используемой СУВ, а лишь иллюстрируют некоторые основные операции, выполняемые в ежедневной работе. Документация поставляется с СУВ, а так же доступна на сайте СУВ git [1].

Сценарии работы иллюстрируются на примере работы с интерфейсом командной строки. Они останутся неизменными при использовании какого-либо графического интерфейса к СУВ.

2.1 Создание репозитория

Для создания локального репозитория в каталоге необходимо выполнить в нём команду `git init`.

2.2 Клонирование удалённого репозитория на локальную машину

Для клонирования удалённого репозитория на локальную машину необходимо в каталоге, в котором будет размещён локальный репозиторий, выполнить следующую команду:

```
git clone <адрес удалённого репозитория>
```

В ОР ТД НС адрес удалённого репозитория имеет вид `gitserver:<имя категории>/<имя репозитория>`.

2.3 Регистрация изменений

2.3.1 Регистрация файлов целиком

```
git add <имена файлов>
```

2.3.2 Выборочная регистрация и редактирование изменений

```
git add --edit <имена файлов>
```

В текстовом редакторе будет открыто описание различий между индексом и текущим состоянием рабочей копии. Для предотвращения вставки определённых строк их следует целиком удалить из описания. Для предотвращения удаления определённых строк следует удалить знак «минус» в начале каждой строки.

2.3.3 Отмена регистрации изменений

Для отмены регистрации изменений необходимо выполнить следующую последовательность действий:

1) удаление из индекса файлов, содержащих изменения:

```
git rm --cached <имена файлов>
```

2) откат содержащих изменения файлов к последнему зафиксированному состоянию:

```
git reset HEAD <имена файлов>
```

Удаляемые из индекса файлы не будут удалены из файловой системы.

Команды git rm и git reset описаны в п.п. 2.4.1 и 2.15 соответственно.

2.4 Операции с файлами

Успешное выполнение файловых операций приводит к регистрации изменений (обновлению индекса). Фиксирование изменений должно быть выполнено разработчиком явно.

2.4.1 Удаление

Команда `git rm` принимает параметры, влияющие на её поведение. Среди них: `--force`, приводящий к принудительному удалению файлов, когда содержимое файловой системы не соответствует содержимому индекса; `-r`, приводящий к рекурсивному удалению каталогов.

Отображение файлов, которые будут удалены, без их фактического удаления:

```
git remove --dry-run <имена файлов>
```

Фактическое удаление файлов, каталогов и символических ссылок:

```
git remove <имена файлов>
```

2.4.2 Перемещение и переименование

Команда `git mv` принимает параметры `--force` и `--dry-run`.

```
git mv <источник> <путь назначения>
```

2.5 Фиксирование изменений

Если процесс фиксирования был запущен, но по какой-либо причине его потребовалось прервать, следует отменить все изменения, сделанные в текстовом редакторе во время написания комментария, либо не сохранять их. По умолчанию СУВ не позволяет сопровождать изменения пустыми комментариями.

2.5.1 Фиксирование зарегистрированных изменений

При необходимости фиксирования всех зарегистрированных в текущей ветви изменений применяется команда `git commit`. Для ограничения области действия команде может передаваться список файлов, изменения в которых требуется зафиксировать.

Для предварительной автоматической регистрации изменений в удалённых и изменённых файлах команде передаётся параметр `--all`. Изменения в новых файлах, ещё не отслеживаемых СУВ, при этом зарегистрированы не будут.

2.5.2 Изменение результата последнего фиксирования

При необходимости изменения результата последнего фиксирования (правки зафиксированного состояния или сопутствующего комментария) команде передаётся параметр `--amend`. Так как выполнение команды с этим параметром приводит к изменению истории разработки, не следует использовать его при работе в ветви, доступной другим разработчикам. При необходимости внесения правки в одно из состояний общедоступной ветви это состояние нужно откатить (см. п. 2.13), а затем зафиксировать изменённую версию.

2.6 Временное сохранение текущей работы

Если имя временного состояния опущено, команда `git stash` выполняется на последнем сохранённом временном состоянии.

2.6.1 Сохранение текущей работы и откат к HEAD

Команда `git stash save` (подкоманда `save` необязательна) выполняет сохранение текущего состояния рабочей копии и индекса и откат к HEAD.

```
git stash [save [<комментарий>]]
```

2.6.2 Просмотр временных сохранённых состояний

а) перечисление сохранённых состояний:

```
git stash list
```

б) просмотр изменений, вносимых выбранным состоянием:

```
git stash show [<имя состояния>]
```

2.6.3 Возврат к сохранённому состоянию

а) применение к HEAD без удаления из списка:

```
git stash apply [<имя состояния>]
```

б) применение к HEAD с удалением из списка:

```
git stash pop [<имя состояния>]
```

в) создание новой ветви и применение сохранённого состояния к её вершине:

```
git stash branch <имя ветви> [<имя состояния>]
```

2.6.4 Удаление сохранённых состояний

а) удаление выбранного состояния:

```
git stash drop [<имя состояния>]
```

б) удаление всех состояний:

```
git stash clear
```

2.7 Обзор состояния рабочей копии

Для получения списков всех добавленных, изменённых, перемещённых и удалённых, незарегистрированных файлов в рабочей копии или выбранном подкаталоге либо для получения сведений о состоянии заданных файлов из рабочей копии используется команда `git status`.

2.8 Навигация по истории разработки

2.8.1 Просмотр журнала истории разработки

Команда `git log` отображает историю разработки. Если помимо имён передать команде перечень имён объектов файловой системы (файлов или каталогов), вывод будет ограничен только записями состояний, в которых эти объекты были изменены.

Основные операции просмотра истории:

а) просмотр полной истории разработки текущей ветви:

```
git log
```

б) просмотр истории разработки текущей ветви между двумя заданными зафиксированными состояниями:

```
git log <имя состояния 1>..<имя состояния 2> [<пути к файлам>]
```

Для выборки записей по заданным условиям команде могут передаваться следующие параметры:

а) `--after` – отображение состояний, зафиксированных после указанной даты;

б) `--before` – отображение состояний, зафиксированных после указанной даты;

в) `--author` – отображение изменений, сделанных разработчиком, имя которого подходит под заданное регулярное выражение;

г) `--committer` – отображение изменений, зафиксированных разработчиком, имя которого подходит под заданное регулярное выражение;

д) `--grep` – отображение изменений, текст комментариев к которым подходит под заданное регулярное выражение;

е) `--all-match` – включение комбинирования условий оператором «логическое И» вместо «логическое ИЛИ»;

ж) `-i` – включение регистро-независимого сопоставления с регулярным выражением;

з) `-F` – поиск вхождений подстрок вместо сопоставления по регулярным выражениям.

2.8.2 Просмотр различий между двумя зафиксированными состояниями

Команда `git diff` отображает описание изменений между двумя произвольными состояниями репозитория. Если помимо имён состояний передать команде перечень имён объектов файловой системы (файлов или каталогов), вывод будет ограничен только изменениями в этих объектах.

Основные варианты просмотра изменений между различными состояниями репозитория:

а) просмотр изменений между двумя произвольными состояниями:

```
git diff <имя состояния 1>..<имя состояния 2> [<пути к файлам>]
```

б) просмотр изменений между текущим состоянием рабочей копии и HEAD (просмотр незарегистрированных изменений):

```
git diff [<пути к файлам>]
```

в) просмотр изменений между текущим состоянием индекса и HEAD (просмотр зарегистрированных изменений):

```
git diff --cached [<пути к файлам>]
```

2.8.3 Общие параметры команд `git log` и `git diff`

а) поиск зафиксированных изменений с вхождением заданной строки: `-S<строка>`;

б) поиск зафиксированных изменений с вхождением строк, подходящих под заданное регулярное выражение: `-G<регулярное выражение>`.

2.8.4 Просмотр различий, внесённых определённым зафиксированным состоянием

```
git show [<имя объекта>]
```

Здесь и далее: если имя состояния не передано команде, принимающей его как необязательный параметр, команда выполняется на HEAD.

2.8.5 Аннотирование каждой строки файла именем автора и временем последнего изменения

Для отображения выбранного файла с метками времени и автора последнего изменения для каждой строки используется команда `git blame`, которой передаётся имя требуемого файла. Факультативно команде может быть передано имя

зафиксированного состояния, которое будет считаться последним в истории изменения файла.

При необходимости аннотирования только фрагмента файла границы фрагмента могут быть переданы как аргументы параметра `-L`. Допускаются следующие варианты задания границ фрагмента:

а) `<номер строки>` – номер сроки в файле;

б) `</регулярное выражение/>` – первая строка, соответствующая заданному регулярному выражению POSIX;

в) `<(+ | -)смещение>` – смещение (число строк) относительно верхней границы фрагмента; допустимо только для нижней границы.

2.9 Управление ветвями в локальном репозитории

2.9.1 Создание ветви

```
git checkout -b <имя ветви> <исходное состояние>
```

2.9.2 Добавление в репозиторий следящей ветви

```
git branch –t <имя локальной ветви> <имя удалённой ветви>
```

2.9.3 Просмотр списка ветвей

а) отображение списка всех локальных ветвей:

```
git branch
```

б) отображение списка всех следящих ветвей:

```
git branch -r
```

в) отображение списка всех локальных и следящих ветвей:

```
git branch -a
```

г) отображение иерархии ветвей и относящихся к ним зафиксированных состояний:

```
git show-branch [<список имён ветвей>]
```

Команда принимает параметры `-a` и `-r`.

2.9.4 Переименование ветви

```
git rename -m [<старое имя ветви>] <новое имя ветви>
```

Если старое имя опущено, переименовывается текущая ветвь.

2.9.5 Переключение между ветвями

```
git checkout <имя состояния>
```

Если в качестве имени состояния, к которому нужно привести репозиторий, указывается имя существующей ветви, происходит переключение на эту ветвь. В общем случае команда позволяет переключиться на любое зафиксированное состояние.

2.9.6 Слияние ветвей

Для слияния определённых ветвей с текущей необходимо выполнить команду `git merge`. Перед слиянием в текущей ветви не должно быть незафиксированных изменений.

```
git merge <имена ветвей, которые требуется слить с текущей>
```

Если слияние закончилось конфликтом, следует исправить текущую ветвь и зафиксировать исправление командой `git commit`, либо отменить слияние, передав команде git merge параметр `--abort`.

2.9.7 Удаление ветви

Перед удалением ветвь должна быть слита с соответствующей ей удалённой ветвью. В случае локальной ветви в ней не должно быть незафиксированных изменений.

```
git branch -d <имя ветви>
```

Для удаления следящей ветви к ключу `-d` следует добавить ключ `-r`.

2.10 Синхронизация изменений

2.10.1 Синхронизация удалённой ветви с текущей локальной

```
git pull [<удалённый репозиторий>] [<именованные ссылки>]
```

Для получения изменений из всех удалённых ветвей команде `git pull` следует передать параметр `--all`.

Непосредственно перед получением изменений из удалённых ветвей необходимо убедиться в том, что в локальном репозитории отсутствуют незафиксированные состояния (см. п.п. 0, 2.6).

В некоторых случаях команда `git pull` может необратимо изменить локальную историю разработки. Более надёжный способ получения удалённых изменений в локальной ветви заключается в ручном извлечении изменений из удалённого репозитория и переносе их на заданное состояние локальной ветви:

```
git fetch <удалённый репозиторий>
git rebase –p <удалённый репозиторий>/<именованная ссылка>
```

2.10.2 Синхронизация локального репозитория с удалённым

```
git push [<удалённый репозиторий>] [<именованные ссылки>]
```

Если в удалённом репозитории отсутствует ветвь с именем текущей активной ветви в локальном репозитории, она будет создана автоматически.

Для загрузки изменений из всех локальных ветвей в удалённый репозиторий команде `git push` следует передать параметр `--all`.

Для удаления заданных именованных ссылок из удалённого репозитория команде следует передать параметр `--delete`.

Для удаления ветви из удалённого репозитория перед именем именованной ссылки в примере выше следует добавить символ «:» (двоеточие).

Непосредственно перед загрузкой изменений в удалённый репозиторий необходимо произвести синхронизацию удалённого репозитория с локальным и убедиться в отсутствии конфликтов состояний между репозиториями. При наличии конфликтов их необходимо разрешить, зафиксировать новое разрешающее конфликты состояние, и лишь после выполнения этих действий проводить синхронизацию локального репозитория с удалённым (см. п. 2.10.1).

2.11 Частичный перенос истории разработки

Посредством команды git rebase осуществляется перенос последовательности зафиксированных состояний на другое указанное состояние:

```
git rebase [--onto <имя нового базового состояния>] [<имя удалённой ветви>]
[<имя локальной ветви>]
```

Базовое состояние – зафиксированное состояние, поверх которого будет осуществляться перенос изменений.

Именем базового состояния, удалённой и локальной ветви может выступать любое существующее имя объекта.

Перенос истории может завершиться конфликтами слияния. После устранения конфликтов процесс должен быть перезапущен передачей команде параметра `--continue` или отменён передачей параметра `--abort`. Перенос также можно продолжить, отложив разрешение текущего конфликта передачей параметра `--skip`.

Так как выполнение команды `git rebase` приводит к изменению истории разработки, не следует использовать её при работе в ветви, доступной другим разработчикам.

2.12 Обмен историей разработки с внешней средой

2.12.1 Подготовка описания различий между двумя зафиксированными состояниями

Команда `git format-patch` извлекает из репозитория различия между указанными зафиксированными состояниями и оформляет их в виде файлов описания различий в заданном каталоге, которые затем можно применить к другому репозиторию.

Если вместо диапазона состояний в команде передан идентификатор одного состояния, СУВ формирует описания различий между этим состоянием и HEAD.

```
git format-patch [--output-directory <выходной каталог>] [<идентификатор
первого состояния в серии> | <диапазон зафиксированных состояний>]
```

2.12.2 Применение изменений из внешних файлов без фиксирования состояния

Команда `git apply` применяет к репозиторию различия из указанных файлов, но не производит фиксированию внесённых изменений.

2.12.3 Применение изменений из внешних файлов с фиксированием состояния

Команда `git am` применяет к репозиторию различия из указанных файлов и производит фиксирование внесённых изменений в текущей ветви. В случае возникновения конфликтов во время применения изменений необходимо разрешить их вручную и продолжить применение изменений, передав команде параметр

`--continue`; пропустить текущий файл описания изменений и продолжить со следующего, передав параметр `--skip`; или прервать процесс и отменить все внесённые изменения, передав параметр `--abort`.

2.13 Перенос отдельных изменений в пределах локального репозитория

Копирование отдельных зафиксированных изменений поверх HEAD производится командой `git cherry-pick`:

```
git cherry-pick <имена зафиксированных состояний>
```

Для редактирования текста комментария к переносимому изменению перед его применением команде следует передать параметр `--edit`.

Для вставки в текст комментария создаваемого командой нового зафиксированного состояния уведомления о том, откуда было перенесено изменение, команде должен быть передан параметр `-x`.

2.14 Отмена зафиксированных изменений

Отмена зафиксированных изменений производится командой `git revert`, принимающей список идентификаторов изменений, которые требуется откатить. Для каждого отменённого состояния в репозитории создаётся состояние, обратное ему, если команде не передан параметр `--no-commit`.

2.15 Удаление истории разработки

Удаление истории разработки – необратимая операция.

2.15.1 Возврат целых файлов к заданному зафиксированному состоянию

```
git reset [<имя состояния>] <файлы и каталоги с изменениями>
```

2.15.2 Возврат фрагментов файлов к заданному зафиксированному состоянию

```
git reset --patch [<имя состояния>] <файлы и каталоги с изменениями>
```

2.15.3 Возврат ветви к заданному зафиксированному состоянию с сохранением незафиксированных изменений

```
git reset --soft [<имя состояния>]
```

2.15.4 Возврат ветви к заданному зафиксированному состоянию

```
git reset --hard [<имя состояния>]
```

2.16 Выделение значимых этапов в истории разработки

2.16.1 Создание неподписанного этапа

```
git tag -a <имя этапа> [<имя зафиксированного состояния> | <имя объекта>]
```

2.16.2 Создание подписанного этапа

Этап разработки может быть подписан цифровой подписью. По умолчанию используется ключ, соответствующий указанному в параметрах репозитория адресу электронной почты разработчика.

```
git tag -s <имя этапа> [<имя зафиксированного состояния> | <имя объекта>]
```

2.16.3 Перечисление существующих этапов

```
git tag [-l]
```

2.16.4 Верификация цифровой подписи

```
git tag -v <имя этапа>
```

2.17 Поиск регрессионных изменений

Процесс поиска регрессионных изменений исчерпывающе описан и проиллюстрирован в статье [2].

2.17.1 Инициализация поиска регрессионных изменений

```
git bisect start [<имя неисправного состояния> [<перечень имён исправных состояний>]] [--] [<пути, в которых должен выполняться поиск>]
```

2.17.2 Выполнение поиска

1) указание заведомо неисправного состояния:

```
git bisect bad [<имя состояния>]
```

2) указание заведомо исправного состояния:

```
git bisect good [<имя состояния>]
```

3) автоматизированный поиск регрессионных изменений при наличии программы, способной самостоятельно определять результат тестирования:

```
git bisect run <команда тестирования> <аргументы команды>
```

2.17.3 Исключение состояния из поиска

```
git bisect skip <имя, перечень или диапазон имён состояний>
```

2.17.4 Сохранение и воспроизведение истории поиска

Если в процессе поиска очередное состояние было помечено неверно (например, исправное помечено как неисправное), возможно сохранение истории поиска, исправление ошибочно установленных меток состояний и перезапуск поиска с учётом просмотренных при прошлом запуске состояний.

1) сохранение истории поиска:

```
git bisect log > <имя файла>
```

2) возврат к исходному состоянию:

```
git bisect reset
```

3) воспроизведение выполненных при прошлом запуске действий:

```
git bisect replay <имя файла>
```

2.17.5 Завершение поиска

Возврат к состоянию, предшествовавшему процедуре поиска регрессионных изменений:

```
git bisect reset
```

2.18 Обеспечение целостности репозитория

2.18.1 Удаление неиспользуемых файлов и сжатие содержимого репозитория

Удаление объектов, на которые в репозитории больше нет ссылок, и сжатие промежуточных состояний хранимых файлов выполняется командой `git gc`. Рекомендуется выполнять эту операцию регулярно, например, один раз после каждых нескольких десятков операций фиксирования. Следует помнить, что при выполнении данной операции приоритетом для СУВ является время её выполнения, а не степень полноты обхода графа состояний и степень сжатия промежуточных данных.

При необходимости гарантированного полного обхода графа состояний и сжатия промежуточных данных с высокой степенью компрессии следует передавать команде параметр `--aggressive`. Делать это рекомендуется один раз после каждых нескольких сотен операций фиксирования.

2.18.2 Проверка целостности репозитория

Для проверки связности и целостности хранимых в репозитории данных используется команда `git fsck`. Для выявления всех случаев нарушения целостности и связности следует передать ей параметры `--unreachable --root --tags --strict --verbose`. Для извлечения объектов, на которые нет ссылок, следует передать параметр `--lost-found`.

2.19 Настройка СУВ и установка параметров репозитория

2.19.1 Настройка системы управления версиями

Установка и изменение параметров СУВ и репозитория производится командой `git config` либо ручной правкой конфигурационных файлов `.conig`, находящегося в служебном каталоге репозитория, и `.gitconfig`, находящегося в домашнем каталоге пользователя. Переменные и их значения, при инициализации нового репозитория требующие установки в первую очередь, приведены в Таблица 1.

2.19.2 Установка параметров обработки содержимого файлов и каталогов

Соответствие определённых параметров обработки содержимого путям в репозитории задаётся в файлах `.gitattributes`. Файлы, находящиеся выше в иерархии файловой системы, имеют больший приоритет. Наивысший приоритет имеет файл `info/attributes`, находящийся в служебном каталоге репозитория.

Правила состоят из шаблонов путей и применяемых к ним атрибутов и записываются по одному в строке. В шаблонах путей допускается использование стандартных символов подстановки «`?`» и «`*`». Одному шаблону может быть поставлено в соответствие несколько атрибутов, разделённых пробелами. Атрибут для каждого шаблона пути может иметь четыре состояния:

а) установлен – атрибут имеет значение «истина»;

б) снят – атрибут имеет значение «ложь», перед его именем ставится знак «-» («минус»);

в) инициализирован значением, отделённым от имени знаком «=»;

г) не задан.

Если для какого-либо шаблона пути требуется перевести атрибут, применённый к надмножеству путей, в состояние «не задан», перед именем атрибута ставится знак «!».

Таблица 1 – Основные переменные и их значения

Имя атрибута	Значение	Описание
core.autocrlf	true	Конвертирование символов концов строк в CLRF при импорте в локальный репозиторий, конвертирование в LF при экспорте в удалённый репозиторий
	input	Конвертирование символов концов строк в CLRF при импорте в локальный репозиторий; конвертирование в LF при экспорте не выполняется
core.safecrlf	true/false	Конвертирование символов концов строк выполняется, только если это не может привести к повреждению данных
	warn	СУВ предупреждает о повреждении данных при конвертировании символов концов строк, но не прерывает процесс
core.editor	<текстовая строка>	Текстовый редактор для редактирования комментариев при фиксировании состояний и выделении значимых этапов разработки
core.ignorecase	true/false	Игнорирование регистра имён файлов. Следует использовать с файловыми системами, не поддерживающими регистр имён файлов
core.pager	<текстовая строка>	Программа для постраничного отображения на экране результатов выполнения команд СУВ
core.whitespace	[-]blank-at-eol	Запрет на пробелы на концах строк в обрабатываемых файлах
	[-]space-before-tab	Запрет на пробелы перед табуляцией в начале строки
	[-]indent-with-non-tab	Запрет на tabwidth (см. ниже) или более пробелов в начале строки
	[-]tab-in-indent	Запрет на табуляцию в начале строки
	[-]blank-at-eof	Запрет на пустые строки в конце файла
	[-]trailing-space	Запрет на пробелы на концах строк и пустые строки в конце файла
	[-]cr-at-eol	Символ CR перед символом конца строки не считается пробелом
	tabwidth=[1-63]	Количество символов, занимаемое табуляцией
color.branch, color.diff, color.grep, color.showbranch, color.status	true, auto	Цветовая разметка вывода команды соответствующей команды включена при работе в эмуляторе терминала
	always	Цветовая разметка включена безусловно
	false, never	Цветовая разметка выключена
format.pretty	oneline/short/ medium/full/fuller/ email/raw	Оформление журнала разработки и отображаемых командой git show зафиксированных состояний
rerere.autoupdate	true/false	Обновление индекса после успешного автоматического разрешения конфликтов

Имя атрибута	Значение	Описание
rerere.enabled	true/false	Автоматическое разрешение конфликтов при многократном слиянии одних и тех же ветвей
user.email	<текстовая строка>	Адрес электронной почты пользователя
user.name	<текстовая строка>	Имя пользователя

В типовом репозитории, содержащем двоичные файлы и текстовые файлы, редактируемые в различных ОС, и не предполагающем интенсивного слияния ветвей, могут использоваться следующие атрибуты, приведённые в Таблица 2.

Таблица 2 – Возможные атрибуты файлов и их значения

Имя атрибута	Значение	Описание
diff	Установлен	Различия между состояниями файлов отображаются, как если бы файлы были текстовым
	Текстовая строка	Для отображения различий между состояниями файлов используется указанный обработчик
delta	Снят	К заданным файлам не будет применена компрессия
eol	“crlf”	Концы строк приводятся к формату LF при импорте и к формату CRLF при экспорте
	“lf”	Концы строк приводятся к формату LF при импорте и не приводятся к формату CRLF при экспорте
text	Установлен	Концы строк находящихся в репозитории файлов приводятся к формату LF, файлы считаются текстовыми
	“auto”	Решение о преобразовании концов строк принимается автоматически на основании типа файла
	Не задан	При выборе формата концов строк используется значение переменной core.autocrlf
whitespace	Установлен	Обработка всех распознаваемых ошибок расстановки пробелов
	Текстовая строка	Список ошибок расстановки пробелов в формате, принимаемом переменной core.whitespace
	Не задан	Обработка всех ошибок расстановки пробелов, определённых переменной core.whitespace
Примечание. «Установлен» – атрибут установлен в значение “true”. «Снят» – атрибут установлен в значение “false”. «Текстовая строка» – значением атрибута является указанная текстовая строка. «Не задан» – данный атрибут не поставлен в соответствие указанному файлу.		

2.19.3 Заполнение списка игнорирования

Правила игнорирования СУВ определённых файлов и каталогов перечисляются в файлах `.gitignore`. Файлы, находящиеся выше в иерархии файловой системы,

имеют больший приоритет. Наивысший приоритет имеет файл `info/exclude`, находящийся в служебном каталоге репозитория.

В правилах допускается использование стандартных символов подстановки «`?`» и «`*`». Правила записываются по одному в строке. Строки, содержащие комментарии, начинаются с символа «`#`». Префикс «`!`» приводит к инверсии правила.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 git(1) manual page – URL: http://git-scm.com/docs/git.html (дата обращения: 16.01.2015).

2 Couder, C. Fighting regressions with git bisect – URL: http://schacon.github.com/git/git-bisect-lk2009.html (дата обращения: 16.01.2015).