Системы управления

версиями

Инструментальные средства разработки ПО

**Система управления версиями** (Version Control System, VCS, Revision Control System) — программное обеспечение для облегчения работы с изменяющейся информацией

Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости

возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы

Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно

изменяющихся электронных документов

Традиционные системы управления версиями используют

централизованную модель, когда имеется единое хранилище документов, управляемое специальным сервером, который и выполняет большую часть функций по управлению версиями.

Пользователь, работающий с документами, должен сначала получить нужную ему версию документа из хранилища; обычно создаётся локальная копия документа, так называемая «рабочая копия». Может быть получена последняя версия или любая из предыдущих, которая может быть выбрана по номеру версии или дате создания, иногда и по другим признакам.

После того, как в документ внесены нужные изменения, новая версия помещается в хранилище. В отличие от простого сохранения файла,

предыдущая версия не стирается, а тоже остаётся в хранилище и может быть оттуда получена в любое время.

Сервер может использовать так называемую **дельта-компрессию** — такой способ хранения документов, при котором сохраняются только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных.

Поскольку обычно наиболее востребованной является последняя версия файла, система может при сохранении новой версии сохранять её целиком, заменяя в хранилище последнюю ранее сохранённую версию на разницу между этой и последней версией.

Некоторые системы поддерживают сохранение версий обоих видов: большинство версий сохраняется в виде дельт, но периодически (по

специальной команде администратора) выполняется сохранение версий всех файлов в полном виде; такой подход обеспечивает максимально

полное восстановление истории в случае повреждения репозитория.

Иногда создание новой версии выполняется незаметно для пользователя (прозрачно), либо прикладной программой,

имеющей встроенную поддержку такой функции, либо за счёт использования специальной файловой системы. В этом случае пользователь просто работает с файлом, как обычно, и при

сохранении файла автоматически создаётся новая версия.

Часто бывает, что над одним проектом одновременно работают несколько человек. Если два человека изменяют один и тот же файл, то один из них может случайно отменить изменения, сделанные другим. Системы

управления версиями отслеживают такие конфликты и предлагают средства их решения.

Большинство систем может автоматически объединить (слить) изменения,

сделанные разными разработчиками. Однако такое автоматическое

объединение изменений, обычно, возможно только для текстовых файлов и при условии, что изменялись разные (непересекающиеся) части этого файла.

Такое ограничение связано с тем, что большинство систем управления

версиями ориентированы на поддержку процесса разработки программного обеспечения, а исходные коды программ хранятся в текстовых файлах. Если автоматическое объединение выполнить не удалось, система может

предложить решить проблему вручную.

Часто выполнить слияние невозможно ни в автоматическом, ни в ручном режиме, например, если формат файла неизвестен или

слишком сложен. Некоторые системы управления версиями дают возможность заблокировать файл в хранилище. Блокировка не

позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла (например,

средствами файловой системы) и обеспечивает, таким образом, исключительный доступ только тому пользователю, который работает с документом.

Многие системы управления версиями предоставляют ряд других возможностей:

* Позволяют создавать разные варианты одного документа, т. н. ветки, с общей историей изменений до точки ветвления и с разными — после неё.
* Дают возможность узнать, кто и когда добавил или изменил конкретный набор строк в файле.
* Ведут журнал изменений, в который пользователи могут записывать пояснения о том, что и почему они изменили в данной версии.
* Контролируют права доступа пользователей, разрешая или запрещая

чтение или изменение данных, в зависимости от того, кто запрашивает это действие.

Каждая система управления версиями имеет свои специфические особенности в наборе команд, порядке работы пользователей и администрировании. Тем не менее, общий порядок работы для

большинства VCS совершенно стереотипен. Здесь предполагается,

что проект, каким бы он ни был, уже существует и на сервере размещён его репозиторий, к которому разработчик получает доступ.

Первым действием, которое должен выполнить разработчик, является извлечение рабочей копии проекта или той его части, с которой предстоит работать. Это действие выполняется с помощью команды извлечения версии (обычно checkout или clone).

Разработчик задаёт версию, которая должна быть скопирована, по

умолчанию обычно копируется последняя (или выбранная администратором в качестве основной) версия.

По команде извлечения устанавливается соединение с сервером, и проект (или его часть — один из каталогов с подкаталогами) в виде дерева каталогов и файлов копируется на компьютер разработчика. Обычной практикой является дублирование рабочей копии: помимо основного каталога с проектом на локальный диск (либо в отдельный, специально выбранный каталог, либо в

системные подкаталоги основного дерева проекта) дополнительно

записывается ещё одна его копия.

Работая с проектом, разработчик изменяет только файлы основной рабочей копии. Вторая локальная копия хранится в качестве эталона, позволяя в любой момент без обращения к серверу определить, какие изменения внесены в конкретный файл или

проект в целом и от какой версии была «отпочкована» рабочая

копия; как правило, любая попытка ручного изменения этой копии приводит к ошибкам в работе программного обеспечения VCS.

* Обновление рабочей копии
* Модификация проекта
* Фиксация изменений

Делать мелкие исправления в проекте можно путём

непосредственной правки рабочей копии и последующей

фиксации изменений прямо в главной ветви (в стволе) на сервере.

Однако при выполнении объёмных работ такой порядок

становится неудобным: отсутствие фиксации промежуточных изменений на сервере не позволяет работать над чем-либо в групповом режиме, кроме того, повышается риск потери

изменений при локальных авариях и теряется возможность анализа и возврата к предыдущим вариантам кода в пределах данной работы.

Поэтому для таких изменений обычной практикой является

создание ветвей (branch), то есть «отпочковывание» от ствола в какой-то версии нового варианта проекта или его части, разработка в котором ведётся параллельно с изменениями в основной версии. Ветвь создаётся специальной командой.

Рабочая копия ветви может быть создана заново обычным образом (командой извлечения рабочей копии, с указанием адреса или идентификатора ветви), либо путём переключения имеющейся рабочей копии на заданную ветвь.

Базовый рабочий цикл при использовании ветвей остаётся точно таким же, как и в общем случае: **разработчик** периодически

**обновляет рабочую копию** (если с ветвью работает более одного человека) и фиксирует в ней свою ежедневную работу.

Иногда ветвь разработки так и остаётся самостоятельной (когда изменения порождают новый вариант проекта, который далее

развивается отдельно от основного), но чаще всего, когда работа, для которой создана ветвь, выполнена, ветвь реинтегрируется в ствол (основную ветвь).

Это может делаться командой слияния (обычно merge), либо путём создания патча (patch), содержащего внесённые в ходе разработки ветви изменения и применения этого патча к текущей основной версии проекта.

Механизм автоматического слияния изменений работает, основываясь на следующих принципах:

* Изменения могут состоять в модификации содержимого файла, создании нового файла или каталога, удалении или

переименовании ранее существовавшего файла или каталога в проекте.

* Если два изменения относятся к разным и не связанным между собой файлам и/или каталогам, они всегда могут быть объединены автоматически. Их объединение состоит в том, что изменения, сделанные в каждой версии проекта, копируются в объединяемую версию.
* Создание, удаление и переименование файлов в каталогах

проекта могут быть объединены автоматически, если только они не конфликтуют между собой. В этом случае изменения,

сделанные в каждой версии проекта, копируются в объединяемую версию. Конфликтующими обычно являются:

* + Удаление и изменение одного и того же файла или каталога.
  + Удаление и переименование одного и того же файла или каталога (в случае, если система поддерживает операцию переименования).
  + Создание в разных версиях файла с одним и тем же именем и разным содержимым.
* Изменения в пределах одного текстового файла, сделанные в разных версиях, могут быть объединены, если они находятся в разных местах этого файла и не пересекаются. В этом случае в объединённую версию вносятся все сделанные изменения.
* Изменения в пределах одного файла, если он не является

текстовым, всегда являются конфликтующими и не могут быть объединены автоматически.

* Ситуацию, когда при слиянии нескольких версий сделанные в них изменения пересекаются между собой, называют конфликтом. При конфликте изменений система управления версиями не может автоматически создать объединённый проект и вынуждена обращаться к разработчику.
* Для разрешения конфликта система, в общем случае, предлагает разработчику три варианта конфликтующих файлов: базовый, локальный и серверный.
* Конфликты в файловой системе разрешаются проще: там может конфликтовать только удаление файла с одной из прочих операций, а порядок файлов в каталоге не имеет значения, так что разработчику остаётся лишь выбрать, какую операцию нужно сохранить в сливаемой версии.

Механизм блокировок:

* Файлы, для работы с которыми требуется блокировка,

помечаются специальным флагом «блокируемый». Такая пометка может ставиться автоматически при добавлении файла в проект, обычно для этого предварительно создаётся список масок имён файлов, которые при добавлении должны становиться блокируемыми.

* Если файл помечен как блокируемый, то при извлечении рабочей копии с сервера он получает в локальной файловой системе атрибут «только для чтения», что препятствует его случайному редактированию.
* Разработчик, желающий изменить файл, вызывает специальную команду блокировки (lock) с указанием имени этого файла. В результате работы этой команды происходит следующее:
  + сервер проверяет, не заблокирован ли уже файл другим разработчиком; если это так, то команда блокировки завершается с ошибкой «файл

заблокирован другим пользователем» и разработчик, вызывавший её, должен ожидать, пока другой пользователь не снимет свою блокировку;

* + файл на сервере помечается как «заблокированный», с сохранением идентификатора заблокировавшего его разработчика и времени блокировки;
  + если блокировка на сервере прошла удачно, на локальной файловой

системе с файла рабочей копии снимается атрибут «только для чтения», что позволяет начать его редактировать.

* Разработчик работает с заблокированным файлом. Если в процессе работы выясняется, что файл изменять не нужно, он может вызвать команду снятия блокировки (unlock, release lock). Все изменения файла будут отменены, локальный файл вернётся в состояние «только для

чтения», с файла на сервере будет снят атрибут «заблокирован» и

другие разработчики получат возможность изменять этот файл.

* По завершении работы с блокируемым файлом разработчик фиксирует изменения. Обычно блокировка при этом снимается автоматически, хотя в некоторых системах блокировку требуется

снимать вручную после фиксации, либо указывать в команде фиксации изменений соответствующий параметр. Так или иначе, при этом файл после изменений теряет флаг «заблокирован» и может быть изменён другими разработчиками.

Система управления версиями обеспечивает хранение всех существовавших вариантов файлов и, как следствие, всех

вариантов проекта в целом, имевших место с момента начала его разработки. Но само понятие «версии» в разных системах может трактоваться двояко.

Система управления версиями обеспечивает хранение всех существовавших вариантов файлов и, как следствие, всех

вариантов проекта в целом, имевших место с момента начала его разработки. Но само понятие «версии» в разных системах может трактоваться двояко.

* Одни системы поддерживают версионность файлов (например,

CVS)

* Для других систем понятие «версия» относится не к отдельному файлу, а к репозиторию целиком (например, Subversion)
* Для практических целей обычно имеет значение не отдельный файл, а весь проект целиком

**Тег** (tag) — это символическая метка, которая может быть связана с определённой версией файла и/или каталога в репозитории

С помощью соответствующей команды всем или части файлов проекта, отвечающим определённым условиям (например, входящим в головную версию главной ветви проекта на

определённый момент времени) может быть присвоена заданная метка. Таким образом можно идентифицировать версию проекта

(версия «XX.XXX.XXX» — это набор версий файлов репозитория, имеющих тег «XX.XXX.XXX»), зафиксировав таким образом его

состояние на некоторый желаемый момент

Как правило, система тегов достаточно гибкая и позволяет

пометить одним тегом и не одновременные версии файлов и каталогов. Это позволяет собрать «версию проекта» любым произвольным образом

С точки зрения пользователя системы пометка тегами может выглядеть по-разному. В некоторых системах она изображается именно как пометка (тег можно создать, применить к определённым версиям файлов и каталогов, снять).

В других системах (например, Subversion) тег представляет собой просто отдельный каталог на файловом дереве репозитория, куда из ствола и ветвей проекта с помощью команды копирования делаются копии нужных версий файлов.

Так что визуально тег — это просто вынесенная в отдельный каталог копия определённых версий файлов репозитория. По

соглашению в дерево каталогов, соответствующее тегу, запрещена фиксация изменений (то есть версия проекта, представляемая

тегом, является неизменной).

**Общие принципы** правильного **использования VCS** немногочисленны и едины для

любых разработок и систем управления версиями.

1. Любые рабочие, тестовые или демонстрационные версии проекта собираются только из репозитория системы. «Персональные» сборки, включающие ещё

незафиксированные изменения, могут делать только разработчики для целей

промежуточного тестирования.

1. Текущая версия главной ветви всегда корректна. Не допускается фиксация в главной ветви неполных или не прошедших хотя бы предварительное тестирование

изменений. В любой момент сборка проекта, проведённая из текущей версии, должна быть успешной.

1. Любое значимое изменение должно оформляться как отдельная ветвь.

Промежуточные результаты работы разработчика фиксируются в эту ветвь. После завершения работы над изменением ветвь объединяется со стволом. Исключения допускаются только для мелких изменений, работа над которыми ведётся одним разработчиком в течение не более чем одного рабочего дня.

1. Версии проекта помечаются тегами. Выделенная и помеченная тегом версия более никогда не изменяется.

# Распределённые системы управления версиями

**Распределённые системы управления версиями** (Distributed Version Control System, DVCS) – системы контроля версий, которые используют распределённую модель вместо традиционной клиент- серверной.

Они, в общем случае, **не нуждаются в централизованном хранилище**: вся история изменения документов хранится на каждом компьютере, в локальном хранилище, и при

необходимости отдельные фрагменты истории локального хранилища синхронизируются с аналогичным хранилищем на другом компьютере.

В некоторых таких системах локальное хранилище располагается

* + **amend** — Внести изменения, не создавая новой версии — обычно когда разработчик ошибочно зафиксировал (**commit**) версию, но не залил (**push**) её на сервер.
  + **blame** — Понять, кто внёс изменение.
  + **branch** — Ветвь — направление разработки, независимое от других. Ветвь

представляет собой копию части (как правило, одного каталога) хранилища, в которую можно вносить свои изменения, не влияющие на другие ветви. Документы в разных ветвях имеют одинаковую историю до точки ветвления и разные — после неё.

* + **changeset, changelist, activity** — Набор изменений. Представляет собой

поименованный набор правок, сделанных в локальной копии для какой-то общей

цели. В системах, поддерживающих наборы правок, разработчик может объединять локальные правки в группы и выполнять фиксацию логически

связанных изменений одной командой, указывая требуемый набор правок в качестве параметра. При этом прочие правки останутся незафиксированными.

* + **check-in, commit, submit** — Создание новой версии, фиксация изменений. В некоторых СУВ

(Subversion) — новая версия автоматически переносится в хранилище документов.

* + **check-out, clone** — Извлечение документа из хранилища и создание рабочей копии.
  + **conflict** — Конфликт — ситуация, когда несколько пользователей сделали изменения одного и того же участка документа. Конфликт обнаруживается, когда один пользователь

зафиксировал свои изменения, а второй пытается зафиксировать и система сама не может корректно слить конфликтующие изменения.

* + **graft, backport, cherry-picking, transplant** — Использовать встроенный в СУВ алгоритм

слияния, чтобы перенести отдельные изменения в другую ветвь, не сливая их. Например, исправили ошибку в экспериментальной ветви — вносим эти же изменения в стабильный ствол.

* + **head** — Основная версия — самая свежая версия для ветви/ствола, находящаяся в хранилище. Сколько ветвей, столько основных версий.
  + **merge, integration** — Слияние — объединение независимых изменений в единую версию документа. Осуществляется, когда два человека изменили один и тот же файл или при

переносе изменений из одной ветки в другую.

* + **pull, update** — Получить новые версии из хранилища. В некоторых СУВ (Subversion) — происходит и pull, и switch, то есть загружаются изменения, а потом рабочая копия доводится до последнего состояния. Будьте внимательны, понятие update двусмысленно и в Subversion и Mercurial значит разное.
  + **push** — Залить новые версии в хранилище. Многие распределённые СУВ (Git, Mercurial) предполагают, что commit надо давать каждый раз, когда программист выполнил какую-то законченную функцию. А залить — когда есть интернет и другие хотят ваши изменения. Commit обычно не требует ввода имени и пароля, а push — требует.
  + **rebase** — Перенос точки ветвления (версии, от которой начинается ветвь) на более позднюю версию основной ветви. Например, после выпуска версии 1.0 проекта в стволе продолжается доработка

(исправление ошибок, доработка имеющейся функциональности), одновременно начинается работа над новой функциональностью в новой ветви. Через какое-то время в основной ветви происходит выпуск версии 1.1 (с исправлениями); теперь желательно, чтобы ветвь разработки новой

функциональности включала изменения, произошедшие в стволе

* + **repository, depot** — Хранилище документов — место, где система управления версиями хранит все документы вместе с историей их изменения и другой служебной информацией.
  + **revision** — Версия документа. Системы управления версиями различают версии по номерам, которые назначаются автоматически.
  + **shelving** — Откладывание изменений. Предоставляемая некоторыми

системами возможность создать набор изменений (changeset) и сохранить его на сервере без фиксации (commit’а). Отложенный набор изменений

доступен на чтение другим участникам проекта, но до специальной команды не входит в основную ветвь. Поддержка откладывания изменений даёт возможность пользователям сохранять незавершённые работы на сервере, не создавая для этого отдельных ветвей.

* + **strip** — Удалить целую ветвь из хранилища.
  + **tag, label** — Метка, которую можно присвоить определённой версии документа. Метка представляет собой символическое имя для группы документов, причём метка описывает не только набор имён файлов, но и версию каждого файла. Версии включённых в метку документов могут принадлежать разным моментам времени.
  + **trunk, mainline, master** — Ствол — основная ветвь разработки проекта. Политика работы со стволом может отличаться от проекта к проекту, но в целом она такова: большинство изменений вносится в ствол; если требуется серьёзное изменение, способное привести к нестабильности, создаётся ветвь, которая сливается со

стволом, когда нововведение будет в достаточной мере испытано; перед выпуском очередной версии создаётся ветвь для последующего выпуска, в которую вносятся только исправления.

* + **update, sync, switch** — Синхронизация рабочей копии до некоторого заданного состояния хранилища. Чаще всего это действие означает обновление рабочей

копии до самого свежего состояния хранилища. Однако при необходимости можно синхронизировать рабочую копию и к более старому состоянию, чем текущее.

* + **working copy** — Рабочая (локальная) копия документов.

# Какие системы управления версиями вы знаете?

AccuRev SCM

GNU Bazaar BitKeeper

CVS CVSNT

darcs

IC Manage Mercurial MKS Integrity

Revision Control System

SCM

Anywhere

Synergy

Team Foundation Server

Dimensions Monotone

Source Code

Vault

CA Software

Change Manager

ClearCase Code Co-op Codeville

CM

Fossil Git

GNU arch

Perforce Helix Core

PVCS

Rational Team Concert

Control System StarTeam Subversion Surround SCM SVK

Veracity Vesta

Visual SourceSafe

Visual Studio Team Services

[List of version control software](https://en.wikipedia.org/wiki/List_of_version_control_software)

[Comparison of version control software](https://en.wikipedia.org/wiki/Comparison_of_version_control_software)

**Конфигурационное управление** (software configuration management, SCM) в программной инженерии — **комплекс методов**, направленных на систематический **учёт изменений,**

**вносимых разработчиками** в программный продукт **в процессе** его

**разработки** и **сопровождения**, сохранение целостности системы после изменений, предотвращение нежелательных и

непредсказуемых эффектов, формализацию процесса внесения изменений.

В связи с высокой динамичностью сферы разработки ПО, в ней конфигурационное управление особенно полезно

К процедурам можно отнести создание резервных копий, контроль исходного кода, требований проекта, документации и т. д

Степень формальности выполнения данных процедур зависит от размеров проекта, и при правильной её оценке данная концепция может быть очень полезна

* + **Контроль**: SCM позволяет отслеживать изменения в контролируемых объектах, обеспечивает соблюдение процесса разработки
  + **Управление**: SCM диктует процесс автоматической

идентификации в ходе всего жизненного цикла ПО, обеспечивает простоту модификации и сопровождения ПО

* + **Качество**
  + Идентификация конфигурации
  + Контроль конфигурации: контроль над изменениями материалов
  + Учёт текущего состояния: состояние документов, состояние кода, состояние отдельных задач и всего проекта в целом
  + Управление процессом разработки
  + Управление сборкой
  + Управление окружением
  + Отслеживание задач и проблем (в частности, отслеживание ошибок)

**Ревизия конфигурации** — процесс проверки соответствия документа нижнего уровня всем требованиям верхнего.

**Аудит конфигурации** — процесс проверки соответствия готового продукта или его части документации.

**Контроль конфигурации** — процесс, при котором все предлагаемые изменения продукта проходят одобрение специальной группы или отдельного человека. Одна из функций такой группы — контроль актуальности всех имеющихся документов, а также контроль того что

все изменения сначала вносятся в документацию, а уже затем в объект

изменения.

**Учет состояния конфигурации** — процесс подготовки отчетов о текущем состоянии продукта и состоянии утверждённых изменений.

**Infrastructure as Code** (IaC, Инфраструктура как код) – это процесс управления и предоставления компьютерных центров обработки данных с помощью машиночитаемых файлов описаний

инфраструктуры, а не физической аппаратной конфигурации или

инструментов интерактивной настройки.

Управляемая IaC ИТ-инфраструктура включает в себя физическое оборудование, виртуальные машины и связанные с ними конфигурационные ресурсы. Конфигурационные описания могут храниться в системе управления версиями. Они могут использовать скрипты или декларативные описания.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tool** | **Released by** | **Method** | **Approach** | **Written in** |
| [Ansible Tower](https://en.wikipedia.org/wiki/Ansible_(software)#Ansible_Tower) | [RedHat](https://en.wikipedia.org/wiki/RedHat) | Push | Declarative and  imperative | [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) |
| [CFEngine](https://en.wikipedia.org/wiki/CFEngine) | CFEngine | Pull | Declarative | - |
| [Chef](https://en.wikipedia.org/wiki/Chef_(software)) | Chef | Pull | Declarative and imperative | [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)) |
| [Otter](https://en.wikipedia.org/wiki/Otter_(software)) | [Inedo](https://en.wikipedia.org/wiki/Inedo) | Push | Declarative and  imperative | - |
| [Puppet](https://en.wikipedia.org/wiki/Puppet_(software)) | Puppet | Pull | Declarative | [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)) |
| [SaltStack](https://en.wikipedia.org/wiki/SaltStack) | SaltStack | Push and Pull | Declarative and imperative | [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) |
| [Terraform](https://en.wikipedia.org/wiki/Terraform_(software)) | [HashiCorp](https://en.wikipedia.org/wiki/HashiCorp) | Push | Declarative | [Go](https://en.wikipedia.org/wiki/Go_(programming_language)) |

**Git** — распределённая система управления версиями, созданная Линусом Торвальдсом для управления разработкой ядра Linux.

Первая версия выпущена 7 апреля 2005 года

*«I'm an egotistical bastard, so I name all my projects after myself. First Linux, now git»*

*Linus Torvalds*

git config git init git add git clone

git commit git status git push

git checkout git remote git branch

git pull git merge git diff git tag

git log git reset git rm git stash git show git fetch

git ls-tree git cat-file git grep

git instaweb git gc

git archive git prune git fsck

git rebase

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| add | config | get-tar-commit-id | merge-ours | remote | stage |
| add--interactive | count-objects | grep | merge-recursive | remote-ext | stash |
| am | credential | hash-object | merge-resolve | remote-fd | status |
| annotate | credential-cache | help | merge-subtree | remote-ftp | stripspace |
| apply | credential-cache--daemon | http-backend | merge-tree | remote-ftps | submodule |
| archive | credential-store | http-fetch | mergetool | remote-http | submodule--helper |
| bisect | daemon | http-push | mktag | remote-https | subtree |
| bisect--helper | describe | imap-send | mktree | remote-testsvn | symbolic-ref |
| blame | diff | index-pack | mv | repack | tag |
| branch | diff-files | init | name-rev | replace | unpack-file |
| bundle | diff-index | init-db | notes | request-pull | unpack-objects |
| cat-file | diff-tree | instaweb | pack-objects | rerere | update-index |
| check-attr | difftool | interpret-trailers | pack-redundant | reset | update-ref |
| check-ignore | difftool--helper | log | pack-refs | rev-list | update-server-info |
| check-mailmap | fast-export | ls-files | patch-id | rev-parse | upload-archive |
| check-ref-format | fast-import | ls-remote | prune | revert | upload-pack |
| checkout | fetch | ls-tree | prune-packed | rm | var |
| checkout-index | fetch-pack | mailinfo | pull | send-pack | verify-commit |
| cherry | filter-branch | mailsplit | push | sh-i18n--envsubst | verify-pack |
| cherry-pick | fmt-merge-msg | merge | quiltimport | shell | verify-tag |
| clean | for-each-ref | merge-base | read-tree | shortlog | web--browse |
| clone | format-patch | merge-file | rebase | show | whatchanged |
| column | fsck | merge-index | receive-pack | show-branch | worktree |
| commit | fsck-objects | merge-octopus | reflog | show-index | write-tree |
| commit-tree | gc | merge-one-file | relink | show-ref |  |