**2 СТРУКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СИСТЕМЫ**

Для проведения лабораторного практикума были выбраны следующие системы контроля версий:

а) Subversion.

б) Mercurial.

в) git.

В БрГТУ.100529-07 91 00 приведены схемы взаимодействия программ.

**2.1 Обзор Subversion**

Subversion, часто сокращаемая до SVN, из-за основной команды svn – система контроля версий, которая была выпущена для замены устаревшей CVS ещё в 2004 году. Subversion реализует все основные функции CVS и свободна от ряда недостатков последней.

Subversion — централизованная система (в отличие от распределённых систем, таких как Git или Mercurial), то есть данные хранятся в едином хранилище. Хранилище может располагаться на локальном диске или на сетевом сервере.

Диаграмма таких взаимоотношений представлена на схеме 1:



Схема 2.1 – Диаграмма взаимоотношений в VCS

Работа в Subversion построена следующим образом: клиенты копируют файлы из хранилища, создавая локальные рабочие копии, затем вносят изменения в рабочие копии и фиксируют эти изменения в хранилище. Несколько клиентов могут одновременно обращаться к хранилищу. Для совместной работы над файлами в Subversion преимущественно используется модель копирование — изменение — слияние. Кроме того, для файлов, не допускающих слияние (различные бинарные форматы файлов), можно использовать модель блокирование — изменение — разблокирование.

При сохранении новых версий используется дельта-компрессия: система находит отличия новой версии от предыдущей и записывает только их, избегая дублирования данных.

При использовании доступа с помощью WebDAV также поддерживается прозрачное управление версиями — если любой клиент WebDAV открывает для записи и затем сохраняет файл, хранящийся на сетевом ресурсе, то автоматически создаётся новая версия. WebDAV (Web Distributed Authoring and Versioning) или просто DAV — набор расширений и дополнений к протоколу HTTP, поддерживающих совместную работу пользователей над редактированием файлов и управление файлами на удаленных веб-серверах.

Общая схема работы с SVN выглядит следующим образом:

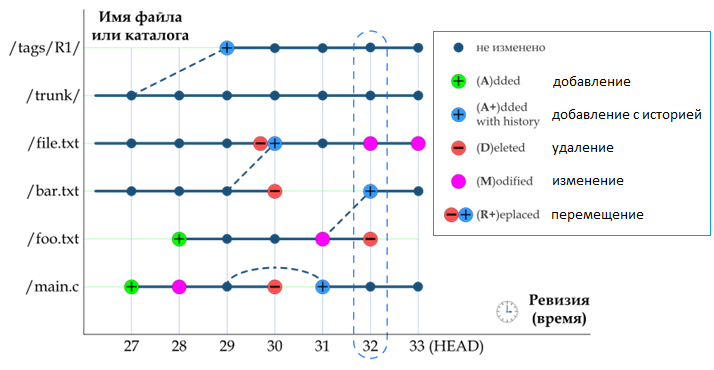


Схема 2.2 – Основные операции над файловой системой.

Над объектами файловой системы в хранилище Subversion (см. рис. 1) могут быть произведены перечисленные ниже операции. В скобках указано краткое именование операции в обозначениях команды svn status.

Добавление (A). Добавление объекта в файловую систему. Добавленный объект не имеет истории ревизий. Пример на рисунке:

файл /main.c был добавлен в ревизии 27.

Модификация (M). Модификация объекта, например, изменение содержимого файла или изменение свойств файла или директории. Пример на рисунке:

файл /main.c был модифицирован в ревизии 28.

Удаление (D). Удаление файла из головной и последующих ревизий. При этом файл остаётся в предыдущих ревизиях. Пример на рисунке:

файл /main.c был удалён в ревизии 30.

Добавление с историей (A+). Представляет собой копирование объекта внутри файловой системы хранилища, то есть объектимя\_источника@ревизия\_источника копируется в имя\_копии@HEAD. Скопированный объект наследует от источника историю ревизий до момента копирования (наследование истории показано на рисунке пунктирными связями). Примеры на рисунке:

в ревизии 29 директория /tags/R1 была скопирована с директории /trunk@27;

в ревизии 31 файл /main.c был скопирован с /main.c@29, то есть с более ранней ревизии самого себя, таким образом, произведено восстановление ранее удалённого (в ревизии 30) файла с сохранением истории ревизий.

Замена (R+). Имеет место в случае, когда в одной ревизии произведено и удаление объекта (D), и добавление с историей (A+) объекта с тем же самым именем. Хотя имя при операции замены остаётся неизменным, Subversion рассматривает объект до и после замены как два различных объекта с различными историями ревизий (история старого заканчивается в точке замены, история нового наследуется от источника копирования и продолжается далее). Пример на рисунке:

в ревизии 30 файл /file.txt был заменён: старый файл /file.txt удалён, а новый файл с тем же именем скопирован с файла /bar.txt@29

Для начала работы с SVN необходимо забрать данные из репозитория (в этой работе это должно быть сделано при помощи командной строки). Чаще всего, «репозиторий» на начальном этапе – это просто ссылка; в данной лабораторной работе ­– полученная от преподавателя.

Для того, чтобы забрать данные из репозитория, необходимо воспользоваться командой svn co (помощи по команде: snv help co), которая является сокращением от svn checkout. Например,

svn co http://example.com/svn/name

Скорее всего, первый раз вам будет ввести имя пользоватля и пароль, которые в дальнейшем можно сохранить у себя на компьютере, чтобы не вводить их при каждом запросе к репозиторию. После этого папка с директорией появится у вас на рабочей машине.

Далее вносятся необходимые изменения в требуемые файлы. После их сохранения и выполнения команды svn status требуемые файлы покажутся в консоли. Если всё верно и изменения правильные, то можно начинать процесс заливки этой версии репозитория на сервер. Если файлы по какой-то причине не отображаются – попробуйте команду svn add . (с точкой на конце – она является признаком того, что вы хотите добавить все файлы); если и это не сработало – попробуйте выполнить svn add . --force

Однако сначала необходимо получить все остальные изменения. Это делается при помощи команды svn up (сокращенно от svn update).

Для начала нужно ввести svn ci –m "здесь сообщение о том, что вы сделали", сокращение от svn commit –m "message". Сообщение нужно для того, чтобы при дальнейшей разработке знать, кто что сделал.

В дальнейшем, когда задание будет закончено, при помощи таких сообщений и команды svn log можно будет узнать, кем и когда были сделаны коммиты, а самое главное, что в них было (т.е. информация, указанная в сообщении). В svn log можно указать флаг -l , который подскажет Subversion,какое количество сообщений необходимо взять. Например,

svn log -l 10

покажет первые 10 сообщений.

Краткий справочник команд Subversion:

svn checkout http://repository.url/svn/name — извлекаем файлы проекта из репозитория, сокращение: svn co;

svn update — получаем обновления из репозитория, сокращение: svn up;

svn update -r rev\_num ./file\_name — извлекаем ревизию файла с номером rev\_num;

svn add ./file\_name — добавляем файл в репозиторий (не важно текстовый или бинарный);

svn commit ./file\_name — заливаем файл в репозиторий (не важно текстовый или бинарный);

svn rename ./old\_file\_name ./new\_file\_name — переименовываем файл в репозитории;

svn remove ./file\_name — удаляем файл/директорию из репозитория;

svn status — просматриваем локально измененные файлы, сокращение: svn st;

svn status -u — просматриваем локально измененные и изменившиеся в репозитории файлы, сокращение: svn st -u;

svn diff ./file\_name — показывает локальные изменения в файле построчно;

svn diff -r rev\_num1:rev\_num2 ./file\_name — показывает различия между ревизией rev\_num1 и rev\_num2 файла;

svn revert ./file\_name — откатывает локальные изменения файла (выгружает из репозитория последнюю закоммиченную ревизию);

svn revert -R ./ — откатывает все локальные изменения файлов;

svn log ./file\_name — список ревизий с комментариями;

svn blame ./file\_name — показывает авторов изменений файла построчно, синоним: svn annotate;

svn propset svn:ignore ./file\_name . — добавляем файл в список игнорируемых файлов;

svn propset svn:keywords "Id Author Date" ./file\_name — установка атрибутов файла;

svn cleanup — снимает блокировки с файлов;

svn unlock http://repository.url/svn/file\_name — снять блокировку файла (URL можно узнать с помощью команды svn info ./file\_name | grep URL, его и нужно передавать в svn unlock);

svnadmin setlog --bypass-hooks /path/to/repository -r rev\_num ./commit\_text\_file — заменяет текстовое описание коммита, где rev\_num — номер ревизии, commit\_text\_file — путь к файлу, содержащему новый комментарий к коммиту;

svn help command\_name — выводит помощь по команде command\_name, например, «svn help update»;

svn merge -r rev\_to\_rollback:rev\_good ./file\_name — откатываем ревизию номер rev\_to\_rollback до ревизии rev\_good, причем все изменения старше rev\_to\_rollback сохраняются (Например, у файла есть ревизии 11,12 и 13. Хотим откатить 12-ую ревизию, но так, что бы изменения 13-ой остались в силе. Делаем тогда так: svn merge -r 12:11 ./file\_name);

svn copy http://repository.url/svn/name/trunk/ http://repository.url/svn/name/branches/new\_branch\_name/ — создаем ветку с названием new\_branch\_name из главной линии разработки;

svn merge --dry-run -r rev\_num1:rev\_num2 http://repository.url/svn/name/trunk/ — проверяем, что будет изменено при объединении веток, где rev\_num1 — номер ревизии, когда ваша ветка была «открыта», или это м.б. номер предыдущего объединения (слияния), rev\_num2 — версия главной линии разработки, с которой производим объединение. Необходимо отметить, что все изменения будут применены для директории, в которой выполнялась эта команда;

svn merge -r rev\_num1:rev\_num2 http://repository.url/svn/name/trunk/ — синхронизирует вашу ветку с главной линией разработки с учетом ревизий: rev\_num1 — номер ревизии, когда ваша ветка была «открыта», или это м.б. номер предыдущего объединения (слияния), rev\_num2 — версия главной линии разработки, с которой производим объединение. Необходимо отметить, что все изменения будут применены для директории, в которой выполнялась эта команда.

**2.2 Обзор Mercuial**

Чтобы забыть о проблемах централизованных систем контроля версий, были созданы децентрализованные системы контроля версий (ДСКВ или DVCS – Distributed Version Control System).

Как следует из названия, в таких системах клиенты не просто забирают последнюю (или выбранную, но всегда одну) версию проекта: на самом деле они полностью копируют его содержание и историю. Таким образом, такой подход имеет два больших преимущества перед VCS:

В любой момент времени разработчик может посмотреть состояние проекта в какой-то момент времени даже без наличия доступа к центральному серверу;

Каждая копия проекта это, по сути дела, полная резервная копия проекта. Таким образом, если сервер выйдет из строя, то любой разработчик сможет восстановить его состояние на любой другой машине.



Схема 2 – Децентрализованная система контроля версий

Mercurial — это современная распределенная система контроля версий с открытым кодом. Эта система — замена для более ранних систем вроде Subversion. У нее два основных назначения:

Она хранит все предыдущие версии каждого файла

Она может объединить разные версии вашего кода, то есть сотрудники могут независимо работать над кодом и затем объединять свои изменения.

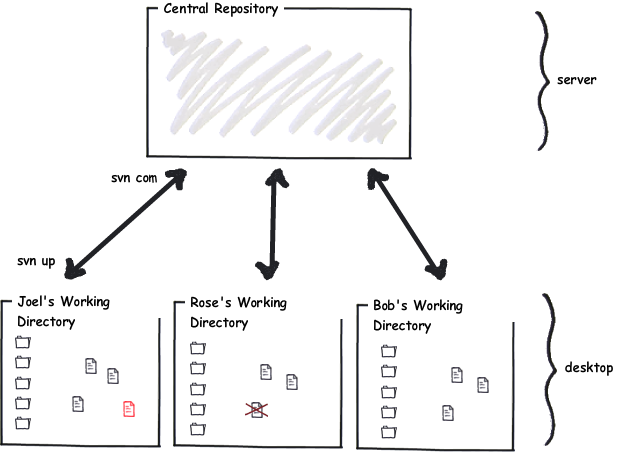
Система Mercurial написана на Python, хотя чувствительные к производительности части (например, своя реализация diff) выполнены в качестве модулей-расширений на C. Mercurial первоначально была написана для Linux, позже портирована под Windows, Mac OS X и большинство Unix-систем. Репозитории Mercurial управляются при помощи утилиты командной строки hg, но есть и GUI–интерфейсы.

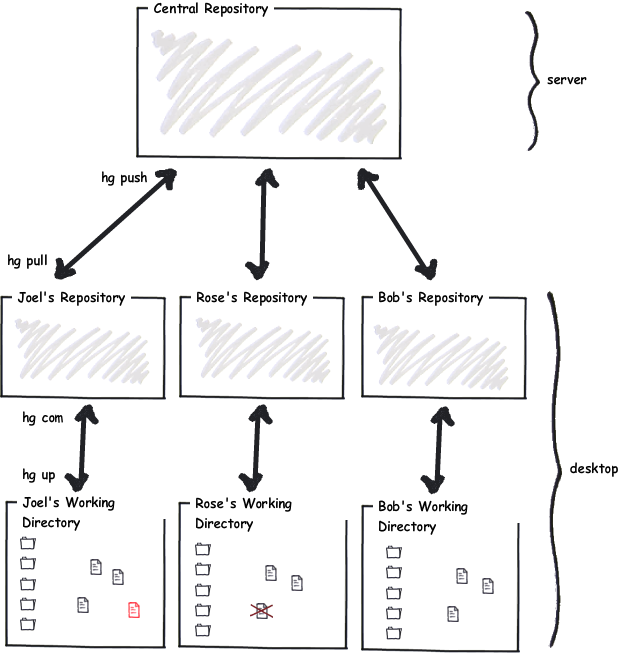
При работе с Subversion, когда вы вносите новый код в репозиторий, его получают все. Так как весь новый код, который пишет программист, содержит баги и/или недоделки, то разработчик встаёт перед следующим выбором:

Вносить неработающий (не до конца рабочий) код (невозможно при работе с реальным заказчиком);

Придерживать новый код до момента его окончательной отладки (всё время).

На практике, команда, использующая Subversion, часто днями или неделями ничего не добавляет в репозиторий. В таких командах новички боятся залить что-то в репозиторий из-за опасения поломать проект и возвращаются к старым методам передачи работы с проектом через архивы. Вот простая иллюстрация работы с Subversion:



При работе с Mercurial у каждого разработчика свой собственный репозиторий, живущий у него на компьютере:  


Mercurial разделяет момент внесения кода в репозиторий и момент получения этого кода всеми остальными.

И это означает, что вы можете коммитить (hg com) (hg commit), но все остальные не получат ваши изменения. Когда у вас накопятся изменения, которые вас устраивают, вы проталкиваете (hg push) их в главный репозиторий.

В Subversion мыслят ревизиями. Ревизия — это то, как выглядит вся файловая система в определенный момент времени. В Mercurial вы мыслите наборами изменений (changesets). Набор изменений — это четкий список изменений между двумя соседними ревизиями. Subversion, по сути, система контроля изменений для файлов, а в Mercurial контроль изменений применяется ко всему каталогу, включая все подкаталоги.

Когда нам нужно сделать слияние, Subversion смотрит на обе ревизии —измененный собственно разработчика и измененный код, который уже находится на сервере — и пытается угадать, как слепить их вместе в одно целое (слияние). Обычно Subversion это не удается, и получаются длинные списки конфликтов («merge conflicts»), которые на самом деле не конфликты, а просто места, в которых система не смогла разобраться в наших изменениях. Такие конфликты разрешаются вручную.

Большинство людей работают с Mercurial через интерфейс командной строки. Так можно работать в Windows, Unix, и Mac. Команда для Mercurial — это hg. Для того, чтобы выполнить эту команду, необходимо

sudo apt-get install mercurial

Если выполнить hg без параметров, то вы получите список наиболее часто используемых команд. Вы можете также попробовать выполнить hg help для получения полного списка команд.

Для того чтобы воспользоваться преимуществами контроля версий, вам нужен репозиторий. Репозиторий хранит все предыдущие версии всех ваших файлов. На самом деле, для экономии места на диске, все предыдущие версии не будут храниться — будет храниться только компактный список изменений.

hg init создает репозиторий. В результате выполнения этой команды появляется новый каталог .hg:

Это и есть репозиторий! Это каталог со всем, что нужно Mercurial для работы. Настройки, предыдущие версии файлов, теги и прочее.  Вам почти всегда не стоит возиться с этим каталогом самостоятельно. Если мы захотим добавить в него все исходные файлы, то нужно всего лишь выполнить hg add.

В данной лабораторной работе выполнять hg init нет необходимости, достаточно сделать hg clone.

hg clone делает полную копию всего репозитория.

hg add помечает файлы как запланированные для добавления в репозиторий. Файлы на самом деле не будут добавлены до тех пор, пока вы не зафиксируете изменения.

Остался еще один шаг… вам нужно зафиксировать ваши изменения. Какие изменения? Факт добавления всех этих файлов.

Почему вы вынуждены фиксировать изменения? При работе с Mercurial фиксирование изменений означает: «Эй, вот как файлы сейчас выглядят, так их и запомни, пожалуйста». Это как создание копии всего каталога… каждый раз, когда у вас есть что-то измененное, и оно вам типа нравится, то вы фиксируете изменения.

hg commit сохраняет текущее состояние всех файлов в репозиторий.

В случае такой ошибки, которая вам вероятно всего встретится первый раз при работе с репозиторием:

error: abort: no username supplied (see "hg help config")

вам необходимо будет добавить файл hgrc (именно с таким именем, без расширения) в .hg директорию. Содержание файла должно будет быть примерно таким:

[ui]

username = Your Name <your@mail>

После этого сохранение должно заработать. Тем ни менее, Mercurial ещё запустит редактор для того, чтобы вы могли напечатать комментарий к изменениям. Вам нужно просто написать что-нибудь, что напомнит вам о сделанных изменениях. Это делается для того, чтобы вы могли просмотреть историю изменений.

hg log отображает историю изменений, зафиксированных в репозитории.

Благодаря Mercurial, если вам не нравится то, что вы сделали, вы можете выполнить удобную команду hg revert, которая немедленно вернет ваш каталог к виду, в котором он был в момент последнего коммита.

hg revert возвращает измененные файлы к зафиксированному в репозитории виду.

Используйте аргумент --all, если вы хотите вернуть все файлы к прежнему состоянию.

Таким образом, когда вы работаете над исходным кодом, используя Mercurial, вы:

Делаете изменения

Оцениваете, подходят ли они

Если подходят, то выполняете commit

Если не подходят, то выполняете revert

GOTO 1

Со временем вы можете забыть, на чем остановились и что поменяли с момента прошлого коммита. Mercurial отслеживает все это для вас. Все что вам нужно, это выполнить hg status и Mercurial даст вам список измененных файлов.

hg status отображает список измененных файлов, добавляя значок в начале каждой строки. Этот значок сообщает о том, что же произошло. M означает «Modified» — файл был изменен. ! означает отсутствие — файл должен быть здесь, но куда-то делся. ? означает что, состояние не определено — Mercurial ничего не знает про этот файл.

Кроме hg status есть ещё команда hg diff, которая сообщит вам, что произошло с файлом с прошлого коммита.

hg diff показывает, что изменилось в файле.

Теперь, вы наверняка хотите положить ваши изменения на сервер. Это делается в пять этапов:

команда hg pull

команда hg merge

тестирование

команда hg commit

команда hg push

hg pull подтягивает (pull) все наборы изменений из другого хранилища в текущий. Заметьте, именно из другого хранилища.

hg merge делает объединение двух изменений в одно большое. В рамках этой лабораторной работы, если вы всё делали правильно, вам не нужно будет делать эту операцию.

hg push проталкивает свежие изменения из текущего репозитория в центральный.

Также есть некоторые другие популярные команды, которые вряд ли пригодятся вам для выполнения данной лабораторной работы, однако могут значительно облегчить вам жизнь в случае необходимости.

hg remove помечает файлы как запланированные для удаления из репозитория. Файлы на диске не будут удалены до тех пор, пока вы не зафиксируете изменения.

hg cat отображает содержимое любого файла для любой ревизии.

Для того чтобы увидеть как файл выглядел раньше, можно просто указать нужный номер набора изменений (changeset из лога) при помощи аргумента -r («revision», то есть ревизия). Если у файла сложное содержимое или большая длина, а изменился он лишь немного, то можно использовать команду hg diff с аргументом r для вывода разницы между двумя ревизиями.

hg update приводит рабочий каталог к состоянию, как у заданной ревизии.

Однако всё время смотреть изменения при помощи командной строки довольно неудобно. Кроме этого, иногда бывает необходимость в создании центрального репозитория (например, на сервере). Проще всего создать центральный репозиторий при помощи встроенного в Mercurial веб-сервера. Вам нужно лишь создать репозиторий при помощи hg init и открыть к нему доступ при помощи hg serve. По умолчанию репозиторий будет доступен на 8000-ом порту.

hg serve запускает веб-сервер и делает текущий репозиторий доступным в сети Интернет.

**2.3 Обзор git**

Терминология:

index — область зафиксированных изменений, т.е. всё то, что вы подготовили к сохранению в репозиторий.

commit — изменения, отправленные в репозиторий.

HEAD — указатель на commit, в котором мы находимся.

master — имя ветки по-умолчанию, это тоже указатель на определённый коммит

origin — имя удалённого репозитория по умолчанию (можно дать другое)

checkout — взять из репозитория какое-либо его состояние.

В качестве системы контроля версий для данной лабораторной работы будем использовать git. Git – распределённая система управления версиями файлов.

Как и в случае с Mercurial, по сути дела, git-репозиторий отличается от любого другого только папкой, в данном случае – .git . Все остальные файлы и папки – это просто файлы и папки, такие же, как и все остальные на рабочей машине.

Для того, чтобы получить git-репозиторий (под которым подразумеваются все файлы и папки + папка .git), необходимо выполнить git clone

После этого, в рамках данной лабораторной работы, вам необходимо будет сделать свою ветку. Это делается при помощи команды git checkout -b style

ПРИМЕЧАНИЕ: git checkout -b <имяветки> является шорткатом для git branch <имяветки> за которым идет git checkout <имяветки>.

Далее, вам необходимо будет сделать изменения в файлах, которые после нужно будет добавить себе в ветку. Команда, которая делает это – git commit –m "Здесь сообщение".

Теперь, когда у вас есть весь необходимый набор файлов, их можно запустить в общий репозиторий. Это делается командой git push origin <имя ветки>.

Комбинация git commit – git push продолжается до тех пор, пока в вашей ветке не закончена работа. Если она закончена, то теперь нужно как-то синхронизировать вашу ветку с общей (традиционное название – master).

Для этого вам нужно:

Перейти в ветку master

Забрать последние изменения ( git pull ( git pull = git fetch + git merge ))

Соеденить свою ветку с мастер-веткой: git merge <имя ветки>, например, git merge task1

Отослать изменения на сервер: git push origin master

Также следует понимать, что только команда git push меняет состояние другого репозитория; все остальные команды меняют только ваш репозиторий.

Итого, цикл работы с git выглядит следующим образом:

git clone – создаёт копию проекта на локальной машине разработчика;

работа непосредственно с файлами (в default или, в другой ветке – git checkout branch\_name)

git pull – забрать изменения, которые были сделаны разработчиками во время, прошедшее с момента git clone

git add – добавляем файлы в тот планируемый commit

git status – просмотр всех изменённых и добавленных файлов

git commit – создание пакета со всеми staged файлами

git push – отправка пакета на сервер

Если у вас возникают вопросы в процессе работы с git, то вы можете воспользоваться командой git --help, которая позволит вам узнать наиболее часто встречающиеся функции; также вы можете воспользоваться справочником по отдельным командам. Так, в частности, команда git checkout --help откроет браузер с информацией об использовании checkout, вместе с параметрами и примерами.

Для начала рассмотрим те вещи, которые вам нужно знать про git:

Git не хранит папки. Он хранит только файлы. Поэтому, например, невозможно положить в git пустую папку – можно только папку с файлами.

Git не хранит файлы. На самом деле, он хранит объекты, в которых указаны пути к файлам. Так что в git-репозитории действительно есть файлы, но вот сам git (т.е. папка .git, как мы узнали из предыдущей лабораторной) их не хранит.

Другими словами, git сохраняет в commit содержимое всех файлов (делает слепки содержимого каждого файла и сохраняет в objects). Если файл не менялся, то будет использован старый object. Таким образом, в commit в виде новых объектов попадут только изменённые файлы, что позволит хорошо экономить место на диске и даст возможность быстро переключиться на любой commit.

Ревизии (Revision) не имеют порядкового номера. Разные файлы входят в разные ревизии, поэтому в каждый момент времени каждый файл принадлежит своему времени.

Редакции (правки, commits) могут идти не по порядку.

В Git’е, по сути дела, нет веток. На самом деле, у нас есть много коммитов, которые образуют граф. Выбираем любой путь от parent-commit к любому child-commit и получаем состояние проекта на этот коммит. Чтобы коммит «запомнить» можно создать на него именованный указатель. Такой именованный указатель и есть ветка (branch). `HEAD` работает по такому же принципу — показывает, где мы есть сейчас. Новые коммиты являются продолжением текущей ветки (туда же куда и смотрит HEAD).

Git – достаточно сложная система. Чтобы облегчить работу программиста, в поставке git есть две вещи, которые должны быть у вас под рукой всегда:

git status — показывает состояние вашего репозитория (рабочей копии) и где вы находитесь.

gitk— графическая утилита, которая показывает наш граф. В качестве ключей передаём имена веток или --all, чтобы показать все.

Если вы сделали что-то не так, запутались, не знаете, что происходит — эти две команды вам помогут.

В список полезных команд git также входят:

git diff, а именно:

git diff HEAD~ – показывает разница между текущим и предыдущим коммитом

git diff 0da94be 59ff30c – показывает изменения между двумя коммитами.

git diff --name-only SHA1 SHA2 – показывает только имена файлов, которые изменились.

git diff --name-only HEAD~10 HEAD~5 – показывает только имена файлов, которые изменились, при этом, показывает только те файлы, которые изменились на расстоянии от 10 до 5 коммита до верхушки ветки. Т.е., если было 20 изменений, то это команда покажетизменения, которые были между десятым и пятнадцатым коммитом.

git show – покажет вам информацию о текущем коммите: как, когда, зачем (сообщение) он был сделан.