**МИНОБРНАУКИ РОССИИ**

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Гжельский государственный университет»** (ГГУ)

Колледж ГГУ

Специальность 09.02.07. Информационные системы и программирование.

**Реферат**

**по предмету «Моделирование и анализ ПО»**

**на тему «Системы контроля версий ПО»**

ВЫПОЛНИЛ:

Студент группы ИСП-О-17

Скрябин С.И.

ПРОВЕРИЛА:

Прокуронова А. Ю.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

п. Электроизолятор

1. г.

**Система управления версиями** (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Version Control System, VCS* или *Revision Control System*) — [программное обеспечение](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) 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,](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. В частности, системы управления версиями применяются в [САПР](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%BD%D1%8B%D1%85_%D1%80%D0%B0%D0%B1%D0%BE%D1%82), обычно в составе систем управления данными об изделии ([PDM](https://ru.wikipedia.org/wiki/PDM)). Управление версиями используется в инструментах [конфигурационного управления](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%84%D0%B8%D0%B3%D1%83%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5) (*Software Configuration Management Tools*).

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](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)) размещён его [репозиторий](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D0%B9" \o "Репозиторий), к которому разработчик получает доступ.

### *Начало работы с проектом*

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.

### *Ежедневный цикл работы*

При некоторых вариациях, определяемых особенностями системы и деталями принятого технологического процесса, обычный цикл работы разработчика в течение рабочего дня выглядит следующим образом.

***Обновление рабочей копии***

По мере внесения изменений в основную версию проекта рабочая копия на компьютере разработчика стареет: расхождение её с основной версией проекта увеличивается. Это повышает риск возникновения конфликтных изменений (см. [ниже](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8#%D0%9A%D0%BE%D0%BD%D1%84%D0%BB%D0%B8%D0%BA%D1%82_%D0%B8%D0%B7%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D0%B9)). Поэтому удобно поддерживать рабочую копию в состоянии, максимально близком к текущей основной версии, для чего разработчик выполняет операцию обновления рабочей копии (**update**) насколько возможно часто.

***Модификация проекта***

Разработчик модифицирует проект, изменяя входящие в него файлы в рабочей копии в соответствии с проектным заданием. Эта работа производится локально и не требует обращений к серверу VCS.

***Фиксация изменений***

Завершив очередной этап работы над заданием, разработчик фиксирует (**commit**) свои изменения, передавая их на сервер (либо в основную ветвь, если работа над заданием полностью завершена, либо в отдельную ветвь разработки данного задания). VCS может требовать от разработчика перед фиксацией обязательно выполнить обновление рабочей копии. При наличии в системе поддержки отложенных изменений (**shelving**) изменения могут быть переданы на сервер без фиксации.

### *Ветвления*

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, 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.

### *Слияние версий*

Три вида операций, выполняемых в системе управления версиями, могут приводить к необходимости объединения изменений. Это:

* Обновление рабочей копии (изменения, сделанные в основной версии, сливаются с локальными).
* Фиксация изменений (локальные изменения сливаются с изменениями, уже зафиксированными в основной версии).
* Слияние ветвей (изменения, сделанные в одной ветви разработки, сливаются с изменениями, сделанными в другой).

Во всех случаях ситуация принципиально одинакова и имеет следующие характерные черты:

1. Ранее была сделана копия дерева файлов и каталогов репозитория или его части.
2. Впоследствии и в оригинальное дерево, и в копию были независимо внесены некоторые изменения.
3. Требуется объединить изменения в оригинале и копии таким образом, чтобы не нарушить логическую связность проекта и не потерять данные.

### *Конфликты и их разрешение*

Для разрешения конфликта система, в общем случае, предлагает разработчику три варианта конфликтующих файлов: базовый, локальный и серверный. Конфликтующие изменения либо показываются разработчику в специальном программном модуле объединения изменений, либо просто помечаются специальной разметкой прямо в тексте объединённого файла. 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.

### *Блокировки*

Механизм блокировки позволяет одному из разработчиков захватить в монопольное использование файл или группу файлов для внесения в них изменений. На то время, пока файл заблокирован, он остаётся доступным всем остальным разработчикам только на чтение, и любая попытка внести в него изменения отвергается сервером. Технически блокировка может быть организована по-разному. Типичным для современных систем является следующий механизм.

В современных VCS предпочтительным является использование неблокирующих извлечений, блокировки же считаются скорее неизбежным злом, которое нужно по возможности ограничивать. Недостатки использования блокировок очевидны:

* Блокировки просто мешают продуктивной работе, поскольку вынуждают ожидать освобождения блокированных файлов.
* Частота возникновения конфликтов и сложность их разрешения в большинстве случаев не настолько велики, чтобы создать серьёзные затруднения.
* Блокировки создают административные проблемы. Типичный пример: разработчик может забыть снять блокировку с занятых им файлов, уходя в отпуск. Для разрешения подобных проблем приходится применять административные меры.

С другой стороны, в некоторых случаях использование блокировок вполне оправданно.

### *Версии проекта, теги*

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.

Для практических целей обычно имеет значение не отдельный файл, а весь проект целиком. В системах, поддерживающих версионность отдельных файлов, для идентификации определённой версии проекта можно использовать дату и время — тогда версия проекта будет состоять из тех версий входящих в него файлов, которые имелись в репозитории на указанный момент времени.

**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.

### Система одновременных версий (CVS)

Порядок использования системы управления версиями в каждом конкретном случае определяется техническими регламентами и правилами, принятыми в конкретной фирме или организации, разрабатывающей проект.

#### Преимущества:

* Испытанная временем технология, которая удерживается на рынке десятки лет.

#### Недостатки:

* Переименование или перемещение файлов не отражается в истории
* Риски безопасности, связанные с символическими ссылками на файлы
* Нет поддержки атомарных операций, что может привести к повреждению кода
* Операции с ветками программного кода дорогостоящие, так как эта система контроляне предназначена для долгосрочных проектов с ветками кода.

### Git

   Эта система была создана для управления разработкой ядра Linux и использует подход, который в корне отличается от CVS и SVN.

В основу **[Git](http://git-scm.com/)** закладывались концепции, призванные создать более быструю распределенную систему контроля версий, в противовес правилам и решениям, использованным в CVS. Так как Git разрабатывалась главным образом под Linux, то именно в этой ОС она работает быстрее всего.

Git также работает на Unix-подобных системах (как MacOS), а для работы на платформе Windows используется пакет mSysGit.

Программный код может быть недоступен, когда используется компьютер без репозитория. Для решения этой проблемы есть обходные пути и некоторые разработчики полагают, что быстродействие Git является справедливой платой за неудобства.

Кроме того, в Git есть множество инструментов для навигации по истории изменений. Каждая рабочая копия исходного кода содержит всю историю разработки, что крайне полезно, когда программируешь без Интернет-соединения.

#### Преимущества:

* Прекрасно подходит для тех, кто ненавидит CVS/SVN
* Значительное увеличение быстродействия
* Дешевые операции с ветками кода
* Полная история разработки доступная оффлайн
* Распределенная, пиринговая модель

#### Недостатки:

* Высокий порог вхождения (обучения) для тех, кто ранее использовал SVN
* Ограниченная поддержка Windows (по сравнению с Linux)

### Apache Subversion (SVN)

[**SVN**](http://subversion.apache.org/) создавалась как альтернатива CVS с целью исправить недостатки CVS и в то же время обеспечить высокую совместимость с ней.

Как и CVS, SVN это бесплатная система контроля версий с открытым исходным кодом.

Для сохранения целостности базы данных SVN использует так называемые атомарные операции. При появлении новой версии к финальному продукту применяются либо все исправления, либо ни одно из них. Таким образом, код защищают от хаотичных частичных правок, которые не согласуются между собой и вызывают ошибки.

В качестве недостатков SVN упоминаются сравнительно низкая скорость и нехватка распределенного управления версиями. Распределенный контроль версий использует пиринговую модель, а не  централизованный сервер для хранения обновлений программного кода. И хотя пиринговая модель работает лучше в open source проектах, она не идеальна в других случаях. Недостаток серверного подхода в том, что когда сервер падает, то у клиентов нет доступа к коду.

#### Преимущества:

* Система на основе CVS
* Допускает атомарные операции
* Операции с ветвлением кода менее затратны
* Широкий выбор плагинов IDE
* Не использует пиринговую модель

#### Недостатки:

* Все еще сохраняются ошибки, связанные с переименованием файлов и директорий
* Неудовлетворительный набор команд для работы с репозиторием
* Сравнительно небольшая скорость