|  |  |
| --- | --- |
| ДИСЦИПЛИНА | Создание программного обеспечения |
| ИНСТИТУТ | Институт перспективных технологий и индустриального программирования |
| КАФЕДРА | Кафедра индустриального программирования |
| ВИД УЧЕБНОГО МАТЕРИАЛА | Лекция |
| ПРЕПОДАВАТЕЛЬ | Макиевский Станислав Евгеньевич |
| СЕМЕСТР | 3 семестр, 2024-2025 гг. |

# 

## Лекция №1

## Структура семестра. Форма отчётности. Инструменты. Git

**Что такое проект?**

Проект – это временное предприятие с уникальным результатом, целью которого является создание нового продукта, услуги или результатов изменений. Проекты могут быть разнообразными – от строительства здания до разработки программного обеспечения или проведения маркетинговой кампании. Они всегда имеют четкую цель, определенные ограничения по времени и ресурсам.

**Основные характеристики проектов:**

Уникальность: Каждый проект является уникальным, поскольку его результаты отличаются от других проектов.

Ограничения: Проекты ограничены временем, бюджетом, ресурсами и объемом работы.

Цель: Каждый проект имеет определенную цель, которую необходимо достичь.

Проектная деятельность:

Проектная деятельность – это систематический подход к планированию, выполнению и контролю проектов.

Она включает в себя следующие основные этапы:

Инициация проекта: На этом этапе определяются цели, требования и ожидания заказчика, а также оценивается жизнеспособность проекта.

Планирование проекта: Здесь разрабатывается детальный план, определяются необходимые ресурсы, сроки и этапы выполнения проекта.

Выполнение проекта: Проект начинается и выполняется в соответствии с предварительным планом.

Мониторинг и контроль: Отслеживание прогресса проекта, выявление и решение проблем, корректировка плана при необходимости.

Завершение проекта: Оценка результатов, предоставление заключительного отчета, архивирование данных и опыта для будущих проектов.

Требования к форме отчетности:

Отчетность является важным аспектом проектной деятельности, поскольку она позволяет донести информацию о прогрессе и результатах проекта заинтересованным сторонам. Отчетность должна быть четкой, информативной и своевременной. Вот основные требования к форме отчетности:

Отчет о статусе проекта: Включает информацию о текущем состоянии проекта, выполненных работах, проблемах и рисках, а также планируемых мерах для их решения.

Финансовый отчет: Важен для контроля бюджета проекта. В нем указываются расходы, доходы, прогнозируемые затраты и планируемые расходы.

Отчет об использовании ресурсов: Данный отчет предоставляет информацию о том, какие ресурсы были использованы в процессе выполнения проекта, а также оценивает их эффективность.

Заключительный отчет: Представляет обобщенные результаты проекта, достигнутые цели, уроки и рекомендации для будущих проектов.

Отчет об оценке рисков: Включает информацию о выявленных рисках, их вероятности, возможных последствиях и принятых мерах по управлению рисками.

Коммуникационный отчет: Описывает информацию о коммуникации внутри команды проекта и с заинтересованными сторонами.

Визуализация данных: Использование графиков, диаграмм и других визуальных средств помогает наглядно представить информацию и облегчает понимание данных.

Правильная и своевременная отчетность позволяет управлять проектом эффективно, принимать правильные решения и достигать поставленных целей. Она также способствует повышению прозрачности и доверия со стороны заинтересованных сторон, что является важным фактором успеха проекта.

**Классификация проектов**

Методы управления проектами зависят от масштаба проекта, сроков реализации, качества, ограниченности ресурсов, места и условий реализации. все названные факторы являются основанием для выделения различных типов проектов, их классификации:

1) по масштабу — микропроект, малый, средний, мегапроект:

— микропроект — это чаще всего форма представления индивидуальной инициативы, получившей признание окружающих. Микропроект делается для себя и своих. он может не требовать внешнего финансирования, специального оборудования, может создаваться из подручных средств;

— малые проекты невелики по масштабу, просты и ограничены объемами. так, в американской практике малые проекты связаны с объемом капиталовложений в размере 10–15 млн долл., трудозатратами до 40–50 тыс. чел. типичный пример малого проекта — модернизация действующих производств. специфика малых проектов состоит в том, что они допускают некоторое упрощение в процедуре проектирования и реализации (простой график, руководитель — одно лицо, необязательно создание команды проекта и т. д.);

— средние проекты наиболее распространены в практике. они имеют сравнительно небольшую длительность — 2–5 лет, требуют более тщательной проработки всех подсистем проекта и предполагают более значительные затраты;

— мегапроекты — это целевые программы, содержащие множество взаимосвязанных проектов, объединенных общей целью, выделенными ресурсами, отпущенным временем. Мегапроекты обладают высокой стоимостью — до 1 млрд долл., трудоемкостью — до 2 млн чел., длительностью реализации — 5–7 лет;

2) по сложности — простой, организационно сложный, технически сложный, ресурсно сложный, комплексно сложный;

3) по срокам реализации — краткосрочный, средний и долгосрочный. краткосрочные проекты требуют для своей реализации примерно год, максимум два, краткосрочные проекты обычно реализуются на предприятиях по производству новинок различного рода, опытных установках, восстановительных работах. коммерческие проекты часто реализуются как краткосрочные. среднесрочные проекты осуществляются за 3–5 лет. длительность осуществления долгосрочных проектов 10–15 лет;

4) по требованиям к качеству и способам его обеспечения — бездефектный, модульный, стандартный. бездефектные проекты направлены на повышение качества продукции или услуг; модульные — на обеспечение качества по какому-либо определенному направлению;

5) по уровню участников — международный, отечественный, государственный, территориальный, местный;

6) по характеру проектируемых изменений проекты делятся на инновационные и поддерживающие (реанимационные, реставрационные). задача инновационных проектов — внедрение принципиально новых разработок. основная цель поддерживающих проектов — сохранить status quo. Поддерживающие проекты, в свою очередь, можно разделить на антикризисный, чрезвычайный, проект реформирования, проект реструктуризации;

7) по сферам и направлениям деятельности — строительный, инжениринговый, финансовый, исследовательский (маркетинговый), технический, технико-экономический, консалтинговый, научно-технический, экологический, социальный, политический и т. д.;

8) по целевым установкам — престиж-проекты и проекты влияния;

9) по особенностям финансирования — инвестиционные (основной мотив инвестора — получение прибыли), спонсорские (спонсор предоставляет средства на поддержку проекта, если это может стать формой его рекламы или презентации, сформировать образ фирмы), кредитные (получение финансовых средств возможно только при условии предоставления гарантий кредитному учреждению, поэтому кредитный проект предполагает развернутое финансово-экономическое обоснование), бюджетные (источники финансирования — бюджеты различных уровней), благотворительные (как правило, это бездоходные и затратные проекты, финансирование таких проектов имеет форму меценатства, грантовую форму);

10) по затрачиваемым ресурсам и получаемой прибыли — коммерческий (получение прибыли), социальный (достижение социальных целей). По признаку преобладающей направленности социальные проекты могут быть: информационно-просветительскими, обучающими, реабилитационными (психологическая, социально-психологическая, трудовая реабилитация), физкультурно-оздоровительными, художественно-творческими, культурными;

11) псевдопроекты — особая группа проектов, создающая видимость проекта, форма, которая прикрывает какое-то другое содержание, не представленное в самом проекте. выделяют две разновидности псевдопроектов — проекты-фикции и квазипроекты:

— проекты-фикции используют проектную форму как маскировку (от лат. Fictio — выдумка, замысел) — это могут быть псевдопроекты по типу фирмы «рога и копыта» остапа бендера, когда фикция имеется уже на стадии замысла. Это могут быть своего рода и декорации деятельности — изображение позитивных результатов проекта там, где они не были достигнуты. еще с XVIII в. для обозначения таких афер стало применяться название «потемкинская деревня»;

— квазипроекты (латинская приставка quasi означает «как будто») — это то, что обладает признаками настоящего проекта, но планирует нововведение, которое на самом деле таковым не является.

**Что такое системы контроля версий?**

Система управления версиями (от англ. Version Control System, VCS) — 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.

Такая система фиксирует изменения, что позволяет вам в случае чего откатиться к любой предыдущей версии файла. Кроме отката изменений, система контроля версий позволяет сравнивать версии одного и того же файла, чтобы найти в нем изменения, видеть, кто эти изменения внес, когда это было сделано и что могло вызвать проблему.

Чтобы сформировать у вас интуитивное понимание, приведу пример. Представьте себе, что вы разрабатываете голосового помощника для мобильных устройств. Ваше приложение уже выпущено в Play Market и в AppStore и пользуется успехом. Теперь вам захотелось добавить в него новую функцию: таймер. Вы пишите код, загружаете обновление в Play Market и AppStore, но внезапно вы узнаете от пользователей, что все сломалось. Теперь, чтобы исправить ошибку, вам нужно пересматривать весь код, чтобы вспомнить, что вы изменяли, и убирать это.

Такой процесс займет очень много сил и времени, а пользователей оставит недовольными. Вам бы очень помогла система контроля версий. Вы бы загрузили в нее файлы с кодом до внесения изменений, а после обнаружения ошибки, откатились бы к ним за считанные секунды. Функция отката изменений позволила бы вам оперативно загрузить в магазины старую версию приложения, которая работает без ошибок.

### **Зачем нужен контроль версий?**

Понятно, что проект для команды - это бесценные знания, накопленные за все время его существования, и никому бы не хотелось, чтобы с ними что-то случилось. Система контроля версий защищает эти накопления от человеческого фактора, непредвиденных последствий и разных форс-мажоров.  
  
 Обычно, работая в команде, каждый разработчик трудится над какой-то своей частью проекта: создает новые функции, оптимизирует уже существующие, исправляет ошибки в уже написанном коде. Сам проект обычно организован в виде дерева файлов. Контроль версий помогает сотрудникам одновременно работать над проектом, отслеживать, кто какие изменения внес, и избегать при этом различных конфликтов (например, когда два человека изменят один и тот же файл по-разному).  
  
Помимо этого, как уже было сказано выше, любое изменение кода может привести к непредвиденным последствиям и сломать весь проект вообще. Система контроля версий защищает и от этого.

### **Разновидности архитектур VCS**

Существует три основных разновидности архитектур систем контроля версий: локальная, централизованная и распределенная. Рассмотрим их все.

### Локальная система контроля версий

Вы спросите: почему нельзя просто сохранять файл под разными именами, а затем открывать нужный? Да, так делать можно, но это вызовет массу проблем. Вы можете забыть, какой файл содержит итоговую версию, можете запутаться в миллионе одинаковых файлов, отличающихся парой строк кода. Вы не сможете быстро сравнить две версии, чтобы посмотреть, какие изменения были внесены.  
  
Для решения этих проблем была придумана **локальная система контроля версий**. Схематически она представляет из себя следующее:

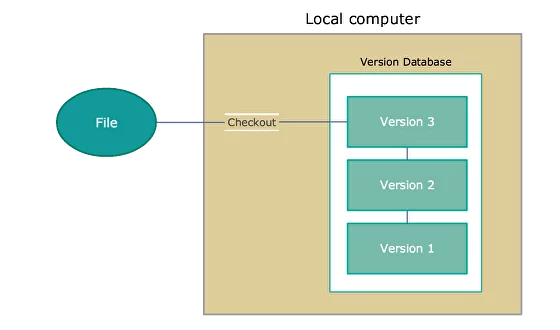


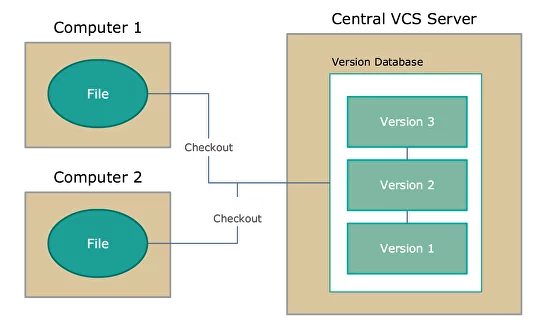
Схема работы локальной системы контроля версий

Из рисунка видно, что есть несколько версий одного и того же файла. А сам файл сейчас находится в состоянии третьей версии.  
  
Одной из самых популярных локальных систем контроля версий на сегодняшний день (не считая Git) остается система **RCS**. Она работает по принципу сохранения изменений в ваших файлах. То есть она хранит не целую новую версию, а только указания к изменению первоначального файла. Например, "добавить к предыдущей версии строку **import math**". Таким образом, последовательно изменяя файл, система воссоздает любую из его версий.

### Централизованная система контроля версий

Как было сказано выше, одна из проблем, с которой вы столкнетесь, работая в команде разработчиков - это то, что вам нужно **делиться файлами** (и их версиями) с другими разработчиками.

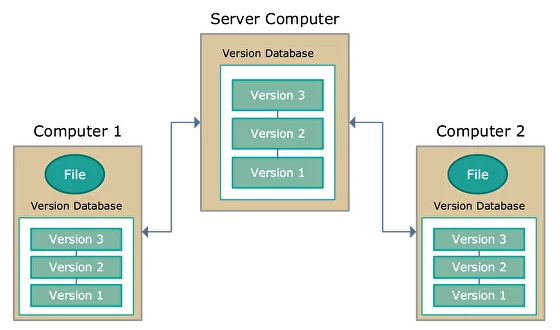
Например, чтобы старший разработчик проверил ваш код на ошибки, перед тем, как дать вам разрешение на внесение изменений в рабочий файл. Для решения этой проблемы была придумана **централизованная система контроля версий**. Схематически она устроена так:



Из рисунка видно, что есть один сервер с сохраненными версиями всех файлов, а разработчики обращаются к нему за файлами.  
  
Такими системами были очень популярные в свое время **CVS**, **Subversion** и **Perforce**. Долгое время такой тип системы контроля версий считался стандартом. Такая система довольно удобна с точки зрения руководства компании. Она позволяет им следить, кто и чем занимается в текущий момент, позволяет настроить: кому какие файлы можно редактировать, а кому - нельзя.  
  
Тем не менее, централизованная система контроля версий имеет множество **недостатков**. Если сервер отключится на несколько часов, то работа целой компании будет парализована: никто не сможет сохранять версии файлов с кодом.  
  
Если будет поврежден жесткий диск сервера, на котором хранятся данные, то восстановить их не удастся (если только не были сделаны копии), и вся работа будет потеряна.  
  
В этом плане такая система мало чем отличается от локальной: если вы храните все данные проекта только на одной машине, вы рискуете потерять все из-за непредвиденного сбоя в работе этой машины.  
  
Кроме того, поскольку вы не храните никаких данных локально, вам нужен бесперебойный высокоскоростной интернет на протяжении целого дня, чтобы скачивать и загружать данные с сервера. Это, конечно, не проблема, но однозначно минус.

### Распределенная система контроля версий

**Распределенная система контроля версий** решает все описанные выше проблемы. К этой группе систем относится **Git**, **Mercurial**, **Bazaar** и некоторые другие. На схеме распределенные системы контроля версий выглядят так:



Особенность этой архитектуры в том, что клиенты не хранят у себя отдельные файлы, они хранят **полную копию** всех версий проекта. Поэтому, если что-то случится с сервером, работа не остановится, а продолжится, как ни в чем не бывало. Работники будут сохранять версии у себя на компьютере, а как только сервер восстановится, они загрузят все эти версии на него.  
  
Кроме того, почему сервер обязательно должен быть один? **Правильный ответ: серверов может быть сколько угодно!**Это открывает безграничные возможности для коллаборации разработчиков. Ведь это значит, что если вы делаете какое-нибудь открытое программное обеспечение и используете систему контроля версий, любой человек сможет скопировать данные с вашего сервера на свой, улучшить это ПО, не боясь ошибок (ведь можно откатиться к предыдущей версии), а затем (с вашего согласия) записать улучшенную версию на ваш сервер. И так с миллионами людей по всему миру.

**Как появился Git?**

Git - это самая популярная на сегодня система контроля версий. Это развитый проект с открытым кодом, активно поддерживаемый и совершенствуемый. Как я уже говорил выше, Git устроен по принципу распределенной архитектуры, но поддерживает и локальную работу (в таком случае ваш компьютер будет единственным сервером, где хранятся все версии проекта).

Git был разработан командой Линуса Торвальдса в 2005 году, как open-source аналог уже существующим системам. Но разработка Git не была спонтанным решением. Дело в том, что с самого первого релиза в 1991 году разработка ядра Linux выполнялась по старинке: старая версия архивировалась, а новые патчи от разработчиков становились новой версией.

Но с ростом популярности рос и объем данных, поэтому в 2002 году было принято решение перевести ядро Linux на распределенную систему управления версиями BitKeeper от BitMover Inc. Однако между компаниями произошел разлад и BitMover Inc. отозвали лицензию на бесплатное использование своего ПО.

Этот инцидент и подстегнул Линуса Торвальдса с командой разработчиков создать свою открытую распределенную систему контроля версий. Ребята хотели разработать надежное решение, обладающее высокой скоростью работы и упрощающее командную разработку.

Основные требования к новой системе были следующими:

Скорость

Простота дизайна

Поддержка нелинейной разработки (тысячи параллельных веток)

Полная распределенность

Возможность эффективной работы с такими большими проектами, как ядро Linux (как по скорости, так и по размеру данных)

Разработка новой системы контроля версий началась 3 апреля 2005 года, а первая версия Git была готова к 7 апрелю того же года. Ядро Linux было переведено на Git 16 июня. 25 июня Линус отказался от должности главного разработчика, но несмотря на это, проект до сих пор поддерживается мировым сообществом под руководством Джунио Хамано.

**Почему именно Git?**

Чтобы в полной мере понять, в чем заключаются все преимущества Git, надо уметь работать в Git. Ну а пока мы не умеем, просто пробежимся по основным возможностям этой системы. Не пугайтесь новых непонятных слов, все термины будут подробно описаны позднее. Сейчас наша задача понять, что Git - это самая крутая VCS на сегодняшний день.

Итак, Git обладает следующими преимуществами.

1. Полная копия репозитория лежит у вас на машине.

Отсюда вытекает два серьезных преимущества: все работает очень быстро, и вы получаете полный контроль над репозиторием:

Чтобы создать репозиторий нужна всего одна команда - git init.

Все файлы VCS хранятся только в одной папке .git. Никаких .svn в каждой директории.

Вам не нужен постоянный и бесперебойный интернет. Утром - скачали данные с сервера к себе на машину, днем - поработали у себя, вечером - залили данные обратно на север. Проще, чем заливать каждый файл после изменения.

В локальном репозитории вы можете создавать дополнительные ветки, тестировать что-то новое и делать все, что угодно. Никто не увидит этого, ведь репозиторий только ваш.

2. Контроль

В Git можно делать что угодно с коммитами:

Удалить

Изменить

Поменять местами

Объединить несколько коммитов в один

Разделить один коммит на несколько

Перетаскивать коммиты между ветками

3. Ветки

Ветки в Git - это настолько мощный и функциональный инструмент, что все выполняется в них. От маленьких задач до релиза.

Создать ветку, переключиться между ветками, слить ветки, удалить ветку - рутинные операции.

За исключением релизных, ветки живут 1-3 дня. Создали ветку, написали новую функцию, протестировали, убедились, что все работает, слили с основной и удалили.

4. Коммиты

Чтобы сделать коммит (фиксацию изменений), нужно указать, какие изменения в него необходимо добавить (именно изменения, а не файлы). Преимущество этого в следующем.

Перед тем, как делать коммит, можно посмотреть и настроить, какие файлы в него попадут. Например, если вы хотите зафиксировать изменения только в одном файле из целого проекта - Git позволит вам это сделать.

Вы можете занести в коммит только часть изменения в файле. А остальные изменения откатить, или положить в другой коммит.

Просмотр истории коммитов и различий в файлах

Можно посмотреть, какие изменения были внесены в файл в разных коммитах.

Можно посмотреть историю коммитов всей ветки, чтобы проследить, как менялись файлы.

5. Stash

Stash - это очень удобная функция Git. Она позволяет заморозить текущие изменения и переключиться на другую ветку.

Например, коллега попросил вас срочно помочь ему с его работой, а у вас множество изменений в файлах, которые еще рано класть в коммит. Вы просто прописываете git stash, после чего переключаетесь на ветку коллеги и помогаете ему. Вам не придется создавать бесполезный коммит, только чтобы сохранить изменения в своих файлах, но при этом вы их и не потеряете при смене ветки.

6. Работа в команде

Вся работа выполняется атомарно в соответствующих ветках. После завершения работы, она отправляется на ревью, и только после этого ветка может быть слита с основной. Это позволяет не допустить присутствие непроверенного кода на основной ветке.

Очень удобно проводить ревью задачи, которая выполнена в отдельной ветке. Посмотреть различия коммитов, что-то исправить, прокомментировать, отправить обратно на доработку, а потом объединить отдельные коммиты и слить в основную ветку.

7. GitHub, BitBucket etc.

Гитхаб, битбакет и другие бесплатные удаленные репозитории очень удобны и часто используются для open-source проектов. Там делают форки этих проектов, обсуждают их и развивают.

## ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ

1. Л. Н. Боронина и З. В. Сенук - Основы управления проектами. — Екатеринбург.: Издательство уральского университета, 2018. — 109 с.: ил.
2. Управление и работа с Git / [Электронный ресурс] // smartiqa.ru : [сайт]. — URL: <https://smartiqa.ru/courses/git/lesson-1?ysclid=lkpvmilfug265319540> (дата обращения: 30.07.2023).