**Среды разработки и тестирования ПО**

На схеме показано, какие среды используются в процессе разработки ПО, кто эти среды поддерживает и кто использует.



Ниже приведено описание схемы:

* Среда разработка (Development Env) – это среда, в которой работаю программисты. Здесь они занимаются написанием и отладкой кода, а также выполняют модульное тестирование
* Среда тестирования (Test Env) – это окружение, в котором работает команда QA. Здесь выполняется проверки функциональности и регресс с использованием тестовых данных. Как правило эта среда не связана или частично связана с внешними системами (нет полноценной интеграционной схемы)
* Превью (интеграционная) среда (Preview Env) – это среда с настроенной интеграционной схемой между системами и продуктами, а также со структурой данных, приближенной к продуктивной
* Продуктивная среда (Production Env) – это окружение, в котором развернуто ПО, где продукт доступен пользователям

Служба технической поддержки условно делится на уровни. Принято считать, что существует четыре линии.

**Содержание**

* [Первая линия техподдержки](https://itssupport.ru/#1)
* [Вторая линия техподдержки](https://itssupport.ru/#2)
* [Третья линия техподдержки](https://itssupport.ru/#3)
* [Четвертая линия техподдержки](https://itssupport.ru/#4)

**Первая линия техподдержки**

Консультирует пользователей по функционалу устройств и программ. Решает не сложные технические задачи, часто путём удалённого подключения.  
Задачи:

* получить максимально точный запрос от клиента - специалисты должны четко понимать, какую именно задачу пользователя нужно решить
* осуществлять фильтрацию задач и перенаправлять в другой отдел, если задача не относится к технической поддержке
* оценить сложность задачи
* решить задачу самостоятельно или перенаправить запрос

Кроме того, в задачи первой линии техподдержки входит обеспечение психологического комфорта пользователя. То есть, специалист берет на себя функцию громоотвода, в случаях, когда клиент эмоционально реагирует на сложности в работе с продуктом. После разговора градус напряжения клиента спадает, появляется уверенность, что проблема будет решена, специалисты заинтересованы помочь, понятен алгоритм дальнейших действий.  
  
Первая линия техподдержки должна обладать минимальным набором знаний и навыков, необходимых для решения стандартных пользовательских задач.  
  
Кроме того, первая линия отвечает за организацию таких процессов, как, например, выезд специалиста или замена оборудования, в случаях, когда такая потребность очевидна.

**Вторая линия техподдержки**

Решает технические вопросы, требующие глубоких знаний, специальных доступов.

Специалисты обязаны в совершенстве знать, как работает продукт.

Задачи:

* осуществить диагностику работы устройства или программы
* определить причину возникшей проблемы
* помочь пользователю устранить ошибку
* самостоятельно провести комплекс мероприятий по решению задачи
* назначать выезды специалистов или замену оборудования

Данные специалисты также могут удалённо подключаться к пользователям, но кроме того, обладают широкими правами доступа в специализированные системы, могут вносить правки на уровне кода.

**Третья линия техподдержки**

Исправляет серьёзные программные ошибки, выявляет уязвимые места, информирует разработку о необходимых доработках.

Задачи:

* решить задачу, с которой не смогла справиться вторая линия
* выявить причину инцидента
* провести работу по устранению причин неполадок совместно с поставщиком оборудования, ПО, в случае, если причина в них

**Четвертая линия техподдержки**

Отвечает за доработку продукта, как самостоятельно, так и с привлечением сторонних ресурсов, например, поставщиков.

Задачи:

* осуществлять анализ работы продукта
* осуществлять анализ потребностей пользователей
* формировать проекты доработок
* взаимодействовать с поставщиками на предмет выработки совместных решений
* осуществлять доработки

Зачастую на практике такое разделение бывает условным. Например, первая и вторая линия могут быть слиты воедино, отвечая за работу с пользователями, а третья и четвёртая - за разработку.

Кроме того, третья и четвертая линии могут быть отделом разработки и находиться на стороне заказчика.

Тем не менее, независимо от того, разделена ли служба формально на четыре уровня, все они так или иначе существуют и работа по поддержке пользователей затрагивает их все.

В любом производственном процессе, к которому относится и разработка программ, есть различные этапы. Грубо говоря, их можно представить так:

* [Производство](https://ru.hexlet.io/blog/posts/environment#proizvodstvo)
* [Сборка](https://ru.hexlet.io/blog/posts/environment#sborka)
* [Контроль и испытания](https://ru.hexlet.io/blog/posts/environment#kontrol-i-ispytaniya)
* [Continuous Integration Server](https://ru.hexlet.io/blog/posts/environment#continuous-integration-server)
* [Доставка](https://ru.hexlet.io/blog/posts/environment#dostavka)

1. Производство (разработка)
2. Сборка
3. Контроль и испытания
4. Доставка

**Производство**

С этим пунктом всё более менее понятно: мы поставили себе любимый редактор, открыли в нем исходный код проекта и работаем, не покладая рук. Возможно, даже, пишем тесты и, самое главное, иногда их запускаем. Если это веб-сайт, то периодически запускаем сервер и смотрим в браузере, что получилось.

Место, в котором происходит этот процесс, **называется средой разработки,** которая, как правило, является локальной. Зачем нужно какое-то специальное название? Чтобы это понять, нужно рассматривать ситуацию в целом. У нас всегда, как минимум, есть две среды. Одна - это среда разработки (ее часто называют development environment), а другая - это среда эксплуатации (так говорят редко, обычно — боевая среда, production). И код должен вести себя по-разному в зависимости от среды.

Например:

* В среде разработки шире уровень логгирования, то есть мы видим все отладочные сообщения и они нам помогают разрабатывать. В продакшене этот уровень отключают, так как очень быстро улетает место на диске.
* В среде разработки мы не можем слать письма по-настоящему. Если это произойдет, то ваши пользователи не будут рады. Кстати, это часто бывает у тех, кто не настраивает разные среды.
* В среде разработки отключают кэширование (техника для ускорения доступа).
* Среда разработки может содержать нерабочий код и находиться в неконсистентном (несогласованном) состоянии. Это нормально, мы ведь разрабатываем.

Кроме этого, код в среде разработки обычно пишется не в основной ветке вашей системы контроля версий, а в ветке-фиче. Это важно, так как не блокирует возможность делать быстрые правки, если на сервере что-то поломалось, и нужно поправить только небольшой кусок, а новые наработки вы ещё не готовы выливать.

К сожалению, в среде PHP до сих пор часто встречается ситуация, при которой разработка ведется прямо в боевой среде. Что приводит к очень печальным последствиям.

**Сборка**

После того, как вы реализовали свою задачу (фичу) и она была протестирована, код задачи вливается в основную ветку — и происходит так называемая интеграция. Это название связано с тем, что, возможно, кроме этой фичи, параллельно велась разработка другой фичи, в другой ветке, и с высокой вероятностью ту задачу выполняли даже не вы. И вот теперь в основной ветке они встретились, а работают они вместе или нет — ещё предстоит выяснить.

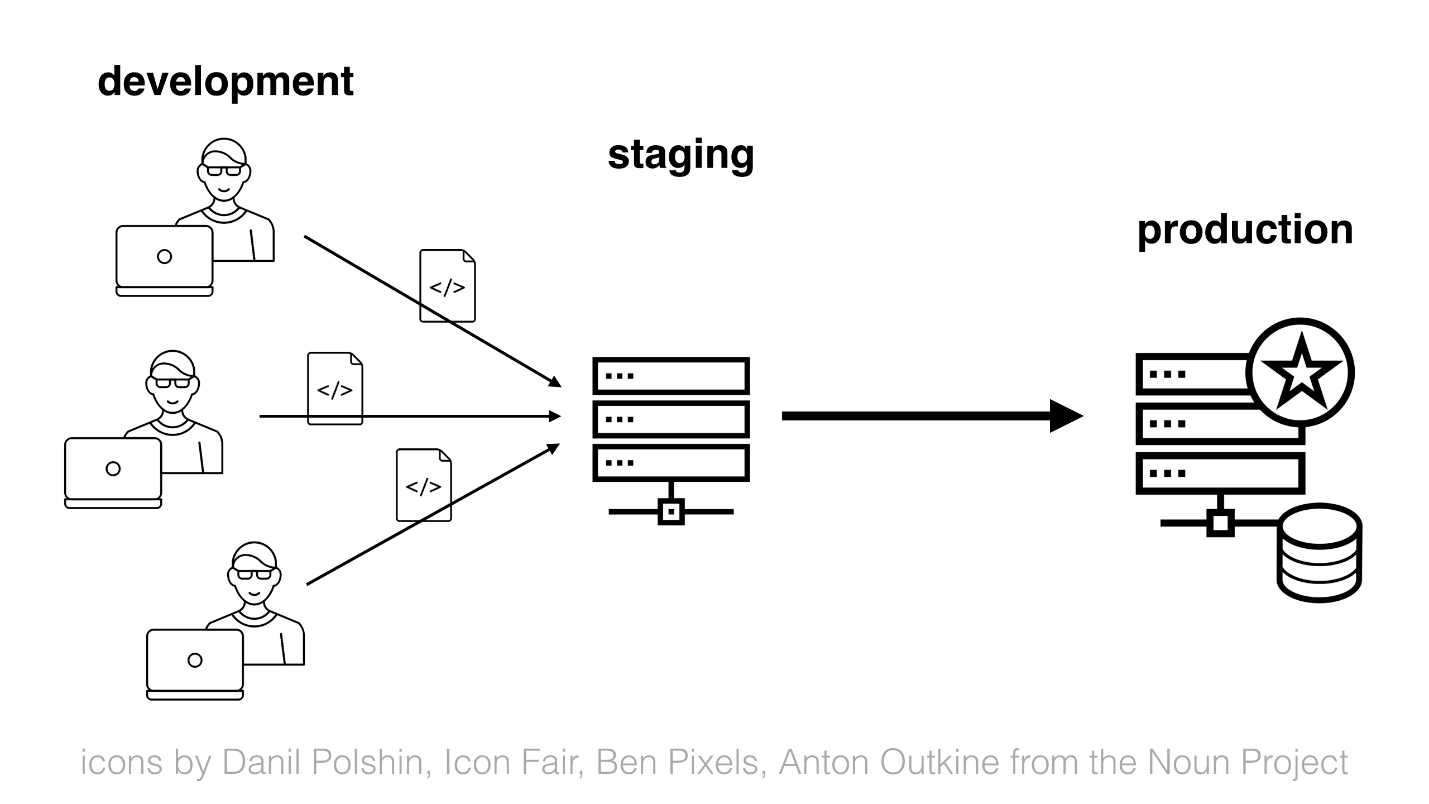
(Этот пункт сильно зависит от того, какой процесс выбран в конкретной команде).

**Контроль и испытания**

Обычно тестирование включает в себя несколько этапов. Первый, на котором происходит проверка конкретно вашей отдельной фичи, и второй, на котором проверяется всё то, что пойдет в следующий релиз.

Ведь даже собрав всё в одну ветку (все фичи) и проверив их локально, нельзя быть до конца уверенным, что в бою, на реальных данных, всё заработает хорошо. Кроме этого, скорее всего, у вас есть менеджер или даже тестировщики, которые тоже хотят посмотреть/проверить, всё ли хорошо. И тут на сцену врывается ещё одна производственная среда, которая называется средой интеграции (предпродакшен), или стейджинг (staging), как её все называют.

Стейджинг — это такая среда, в которой происходит проверка перед боем. Её особенностью является максимальное приближение к условиям боевой среды, что дает возможность полнее протестировать то, что происходит. Обычно это то место, куда идут менеджеры, тестировщики, заказчики. Часто стейджинг выполняет сразу две задачи — проверку конкретных фич от разработчиков и окончательный прогон приложения перед релизом.



Тут появляется ещё одно новое слово: «релиз». Релиз по-другому называют «выпуск». С одной стороны, это процесс выкатки в бой новой версии системы. С другой стороны, так иногда называют сборку, которая представляет из себя новую версию системы.

**Continuous Integration Server**

Одна из разновидностей сборочной среды называется «сервер непрерывной интеграции». Это такая отдельная машина (а может быть целый парк машин), на которую выливается код для проверки в автоматическом режиме. Обычно это происходит по какому-нибудь событию, например, на Github это пулреквест. В настроенных проектах каждый пулреквест отправляется в сервис, подобный <https://travis-ci.org>. Этот сервис прогоняет тестовый набор на нужной ветке (с фичей) и после этого прикрепляет отчет к пулреквесту, в котором пишет о результатах проверки.

Такая система позволяет очень сильно ускорить процесс интеграции. Сильно снижается нагрузка на разработчиков и автоматизируется рутина. Разработчику достаточно писать код и отправлять его в репозиторий, а система сама будет проводить необходимые проверки и выполнять слияние. Непрерывная интеграция является частью практик под названием «экстремальное программирование (XP)».

**Доставка**

Мы упустили один важный момент. Каким образом новый код попадает в предпродакшен и в продакшен-среду после того, как вы закончили разработку? Делает он это благодаря процессу, который в простонародье называют «деплой».

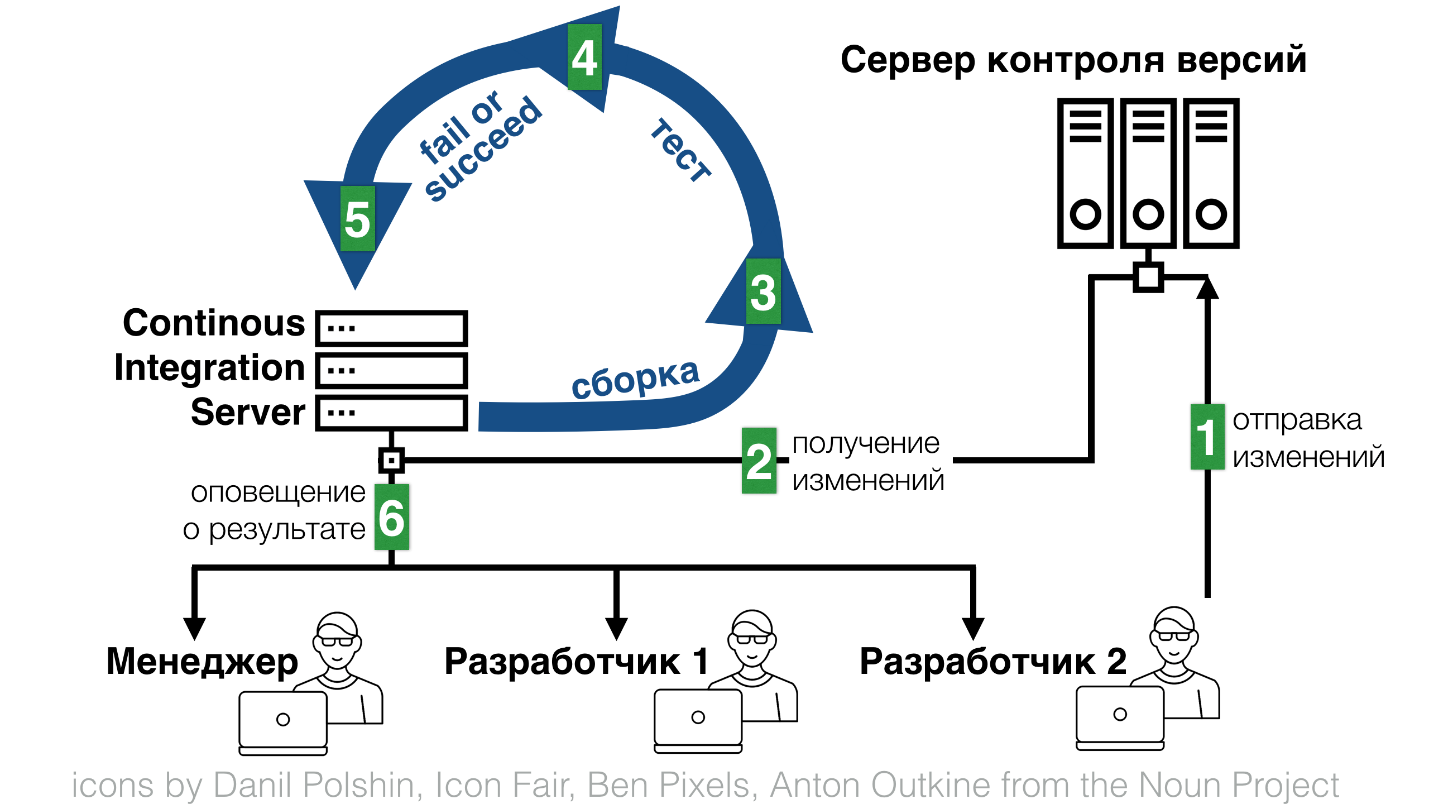
Как показывает практика, многие до сих пор делают деплой руками. Заходят на сервер (а если их много?) клонируют код, руками меняют базу и так далее.

Можно бесконечно обсуждать то, насколько это плохо. Начиная с того, что по сути отсутствует налаженный, повторяемый процесс, а значит всегда есть вероятность того, что ворвется человеческий фактор и случайно будет что-то забыто/потеряно/удалено. Заканчивая тем, что знания хранятся в одной голове, и сам процесс релиза становится вуду-процедурой, которую может делать только Вася, а иногда он болеет, ходит в отпуск и может уволиться. Часто в таких компаниях релиз — крайне болезненная процедура, которая занимает не один час, а может даже пару дней.

При хорошо отлаженном процессе, релиз занимает десяток минут, и может делаться любым разработчиком в любой момент (почти). Хекслет иногда деплоится по 5-10 раз в день.

Основные задачи, которые стоят перед вами во время деплоя:

* Взять новую версию кода из репозитория и залить его на сервер(а)
* Сделать все необходимые приготовления: накатить миграции, собрать frontend-скрипты и т.д.
* Переключить проект на новую версию
* Откатиться в случае ошибок



В среднем проекте количество действий, которое необходимо сделать при деплое, уже составляет десятки различных задач. Хорошая новость в том, что в современном мире это настолько отработанная процедура, что существует немало решений, позволяющих настроить деплой любого проекта. Одним из таких решений является набор скриптов поверх ansible: <https://github.com/ansistrano/deploy>

Для тестирования мобильного приложения Авито может быть использовано:

1. Отдельный контейнер, развернутый для тестируемой ОС с тестируемой версией мобильного приложения
2. Виртуальная машина, с установленной на нее средой и тестируемой версией мобильного приложения
3. Мобильное устройство с необходимой ОС и тестируемой версией мобильного приложения

Я использовала мобильный телефон Samsung Galaxy A12 SM-A127F 32GB, операционная система Android 10, приложение Авито из плеймаркет версии 133.5