

Лекция extra1.

Системы управления версиями

“Операционные системы”-2019, ИВК, УлГТУ.

Из чего состоит ИС для программиста?

- Исходный код;
- Документация;
- Развёртки системы (staging, rc, release, ...).



Типичная задача: внесение правок в код

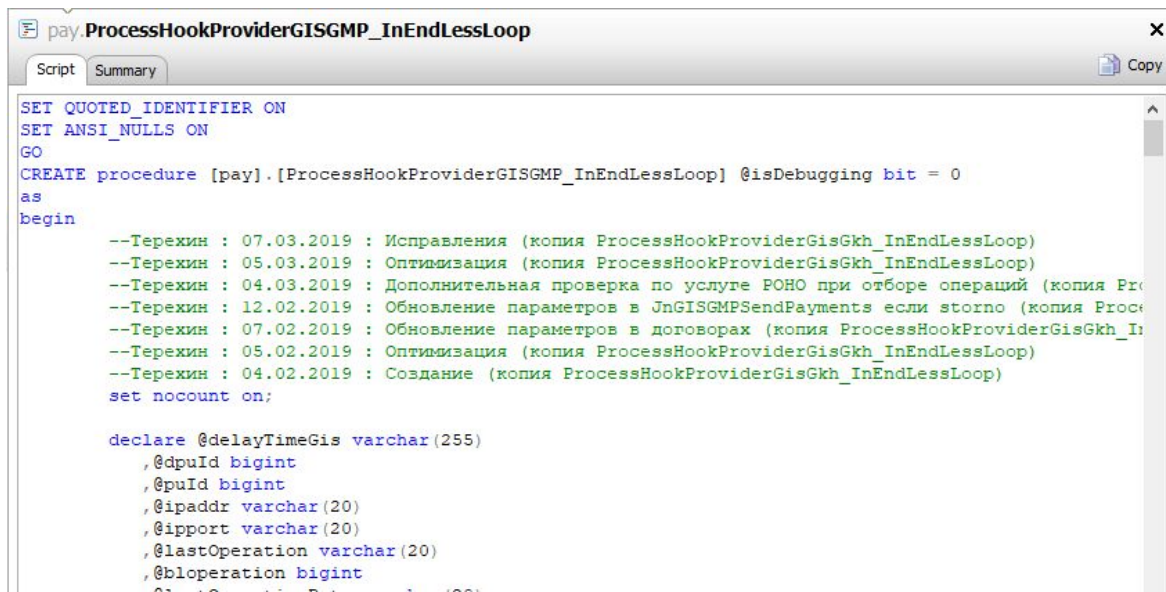
- Берётся файл с исходным кодом;
- Вносятся правки;
- (опционально) Программа компилируется;
- Новая версия программы разворачивается.

Но:

- Что делать, если правку нужно откатить?
- Что делать, если надо посмотреть, когда была исправлена та или иная часть кода?
- Что делать, если разработку ведут сразу несколько людей?

Возможные решения

- Для каждой задачи создавать копию файла (или всего проекта);
- В начале каждого файла писать, кто, когда и зачем его менял;
- Хранить исходные тексты на едином сервере (например, FTP).



```
SET QUOTED_IDENTIFIER ON
SET ANSI_NULLS ON
GO
CREATE procedure [pay].[ProcessHookProviderGISGMP_InEndLessLoop] @isDebugging bit = 0
as
begin
    --Терехин : 07.03.2019 : Исправления (копия ProcessHookProviderGisGkh_InEndLessLoop)
    --Терехин : 05.03.2019 : Оптимизация (копия ProcessHookProviderGisGkh_InEndLessLoop)
    --Терехин : 04.03.2019 : Дополнительная проверка по услуге ПОНО при отборе операций (копия ProcessHookProviderGisGkh_InEndLessLoop)
    --Терехин : 12.02.2019 : Обновление параметров в JnGISGMPsSendPayments если storno (копия ProcessHookProviderGisGkh_InEndLessLoop)
    --Терехин : 07.02.2019 : Обновление параметров в договорах (копия ProcessHookProviderGisGkh_InEndLessLoop)
    --Терехин : 05.02.2019 : Оптимизация (копия ProcessHookProviderGisGkh_InEndLessLoop)
    --Терехин : 04.02.2019 : Создание (копия ProcessHookProviderGisGkh_InEndLessLoop)
    set nocount on;

    declare @delayTimeGis varchar(255)
            ,@dpuId bigint
            ,@puId bigint
            ,@ipaddr varchar(20)
            ,@ipport varchar(20)
            ,@lastOperation varchar(20)
            ,@blopoperation bigint
```

Системы управления версиями

Для решения проблем, описанных выше, придумали системы управления версиями (VCS).

Версиями чего? Да всего. Например, текстовых и бинарных файлов.

Например, с помощью VCS можно очень легко узнать “а как выглядел исходный код программы 4 года назад?”

Репозиторий, версии файлов

Набор данных, для которого ведётся учёт версий, называется **репозиторием**.

Любой вариант файла в репозитории называется **версией** файла.
У файла может быть бесконечно много версий.

Например, есть репозиторий “КурсовойРИС”.

В нём 2 файла: README.md (3 версии файла) и TODO.md (1 версия файла).

Возникает вопрос: как в репозитории появляются версии файлов?

Локальная копия, коммит

Перед работой сотрудник (пусть будет разработчик) скачивает к себе на рабочий компьютер копию репозитория. Она называется **рабочей копией**.

Обычно, требуется скачать последнее состояние репозитория, хотя это и не обязательно.

Затем разработчик вносит правки в своей рабочей копии (на сервере его правок ещё нет).

После необходимых правок разработчик закрепляет изменения у себя в рабочей копии. Это называется **коммит** (*commit*). На этом этапе ответ на вопрос: “Есть правки на сервере или нет?” – зависит от конкретной VCS.

Описание коммита

В коммите обычно содержится:

- идентификатор (хэш или номер - зависит от VCS);
- информация об авторе изменений;
- дата и время изменений;
- произвольный комментарий;
- информация об изменениях файлов репозитория;
- идентификатор коммитов-родителей (в GIT).

Централизованные и распределённые VCS

В централизованных VCS репозиторий хранится на удалённом сервере, и для работы необходим доступ к нему. Пример: SVN.

Есть большой недостаток: если “упадёт” сервер с репозиторием, нельзя будет сделать коммит.

В децентрализованных VCS (DVCS) в рабочей копии разработчика создаётся свой репозиторий, с которым он и работает. Пример: git.

При этом к команде закрепления изменений (**commit**) добавляются команды получения данных из удалённого репозитория (**pull**) и команда отправки данных в удалённый репозиторий (**push**).

Версии репозитория

В самом простом варианте версии репозитория последовательны и выражаются в цепочке коммитов:

```
* e7b2df1... up to 0.3.0
* dd80544... deep merge hashes so as to not lose data
* 8477cb5... Adding Movable Type migration library to lib/jekyll/co
* a8ab6a0... Added wordpress converter
* 9fb1f6e... fix backup file ignore merge
```

Читается обычно снизу вверх. То есть сначала был коммит 9fb1, после него сделали коммит a8ab6, после него сделали коммит 8477cb5 и т.д.

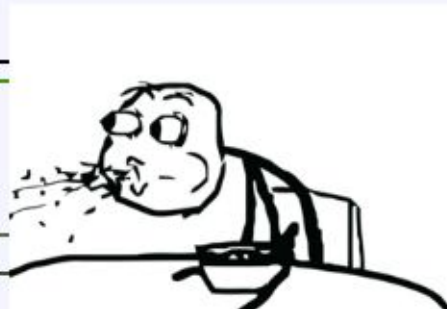
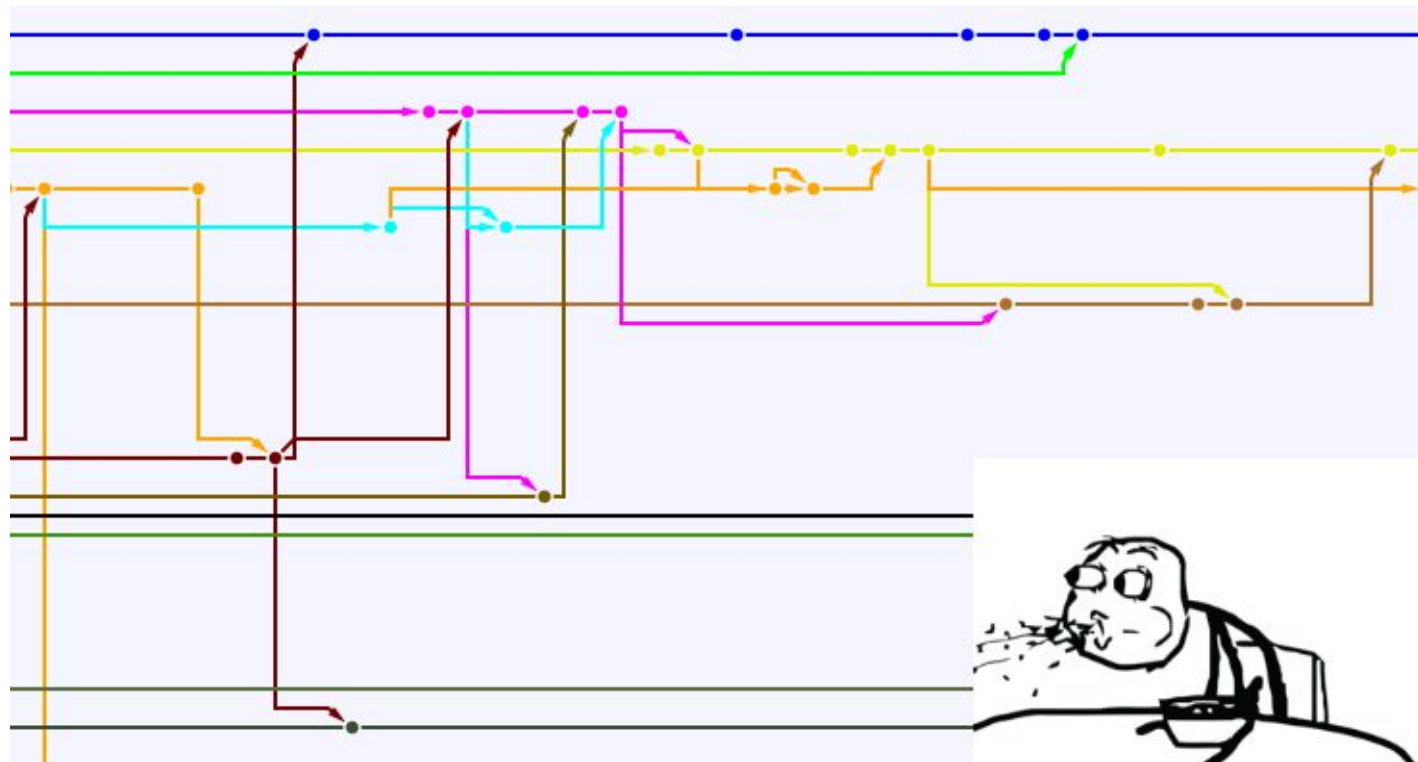
Точками (или звёздочками) обозначаются коммиты разработчиков.

Версии репозитория

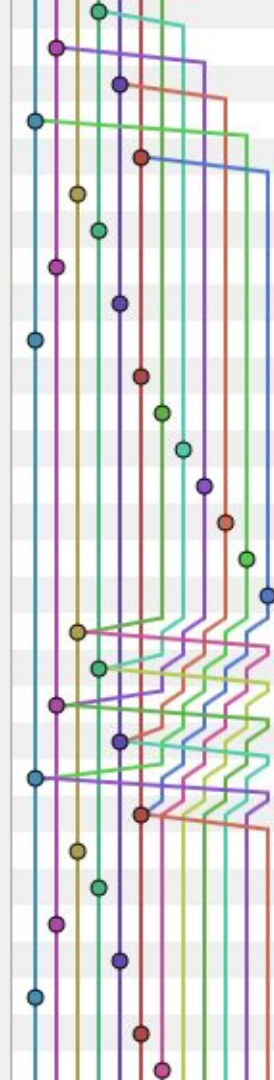
Сложности начинаются, когда два разработчика сделали свои коммиты от одного общего (назовём его базовым коммитом). И получается:

```
* 23e97c1... Merge commit '87a5e747f'
|\
| * 87a5e74... Added array_to_sentence_string filter
* | 9c52bcc... Merge commit '4f677f627ef569be8'
|\ \
| * | 4f677f6... apply Liquid templating to includes
| | /
* | 956eece... update history
* | 9aff9c8... use canonical documentation style
* | feff93c... If title is omitted from a Post's YAML, use the fil
* | 4db6961... Added new accessible Liquid attribute for Sites: .to
* | 4770882... added typo to manifest and Rakefile
* | 29313b0... fixing typo in lib/jekyll/converters/mt.rb
* | e0e7bf1... added Typo 4+ conversion module and docs
| /
* e7b2df1... up to 0.3.0
* dd80544... deep merge hashes so as to not lose data
* 8477cb5... Adding Movable Type migration library to lib/jekyll/co
* a8ab6a0... Added wordpress converter
* 9fb1f6e... fix backup file ignore merge
```

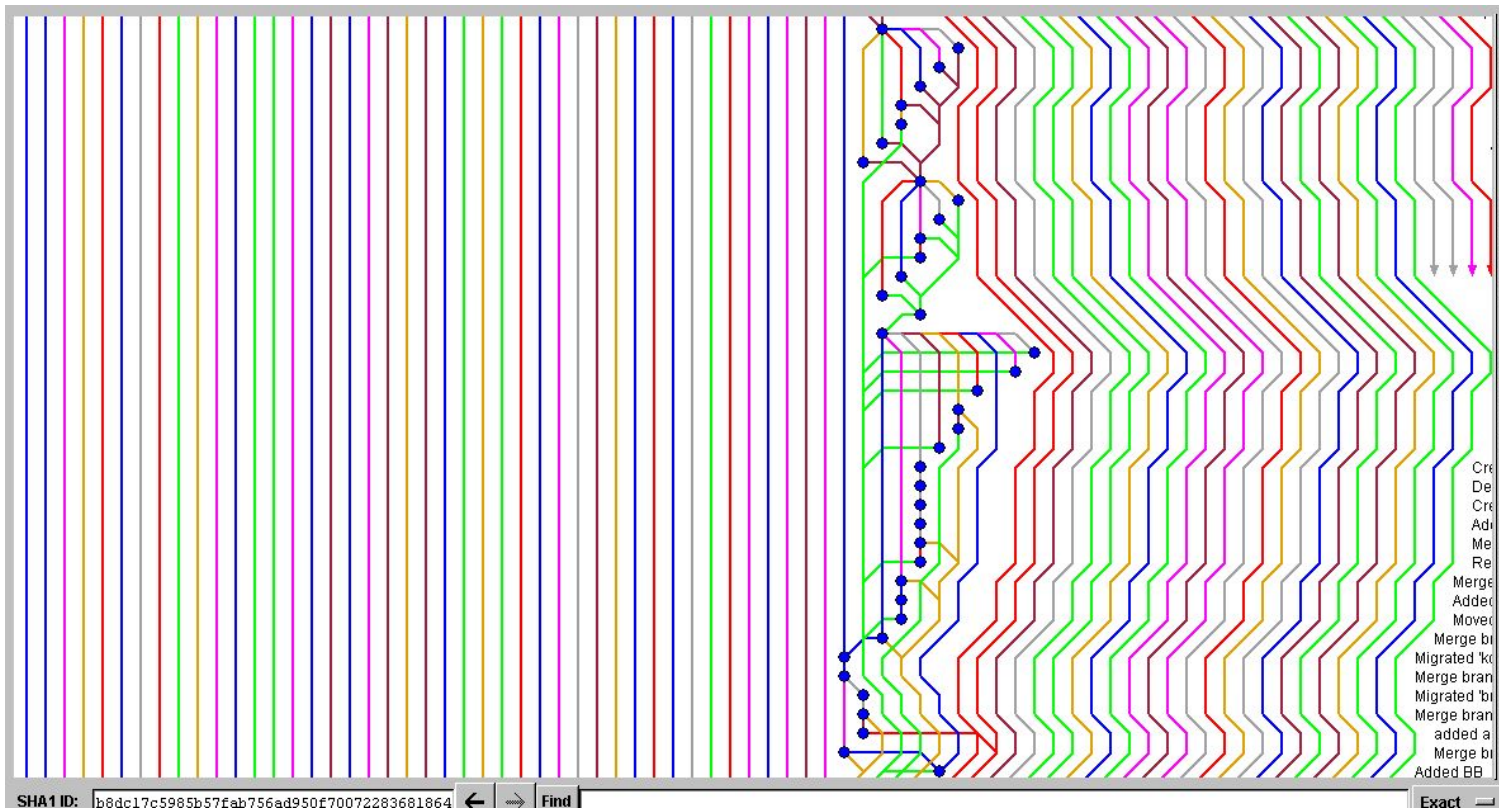
Или так



ns
ionWIP



Или даже так..



Ветки

Каждая цепочка коммитов называется **веткой** (*branch*).

Обычно выделяется главная ветка (*master* в git, *trunk* в svn).

Разработчики могут создавать свои ветки (“почковать” их от других веток).

Новые ветки обычно создаются для объединения коммитов по одной задаче в одном месте. Сделано это, прежде всего, для того, чтобы в “главной” ветке не было состояния, при котором половина коммитов по задаче уже есть, а вторая половина ещё не написана. То есть в главной ветке задача или сделана полностью, или не сделана вообще. Промежуточный результат выделяют в ветку.

Слияние веток

Если все изменения в ветке были сделаны, их необходимо “слить” в главную (или любую другую) ветку.

Этот процесс называется **слиянием** (*merge*).

В момент слияния информация о ветке обычно остаётся.

Однако, если ветка больше не нужна, то её можно удалить, чтобы не было “зависших веток”.

(тут, Моисеев, перелистни на 2 слайда назад)...

Отвлечёмся. Зачем вообще VCS нам?

- В любой команде разработчиков принято использовать системы управления версиями;
- О системах управления версиями спрашивают абсолютно на всех технических собеседованиях и очень расстраиваются, если кандидат не знает, что это такое;
- Чтобы хранить копии своих курсовых и лабораторных работ, причём со всеми версиями файлов. Так, на всякий случай.

Workflow (рабочий процесс)

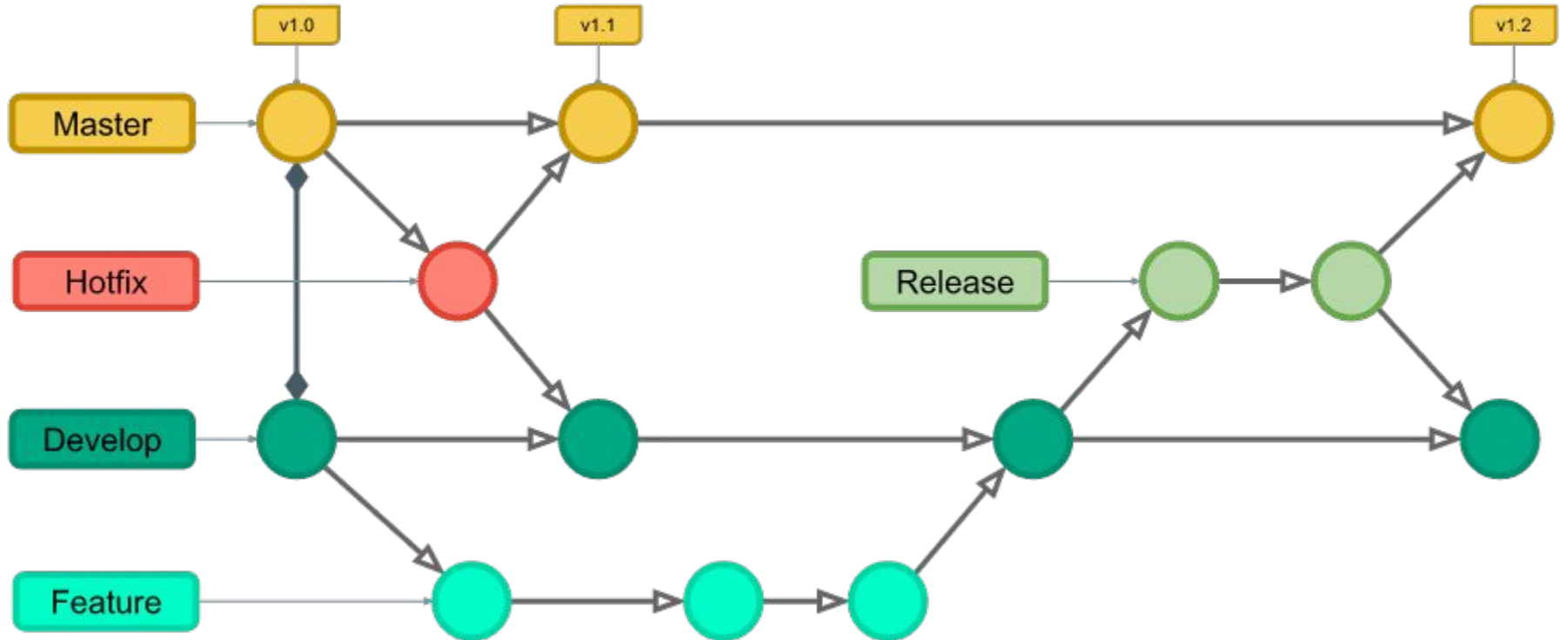
Во время использования той или иной системы управления версиями у людей складываются определённые успешные практики.

Они получили названия workflow (рабочий процесс).

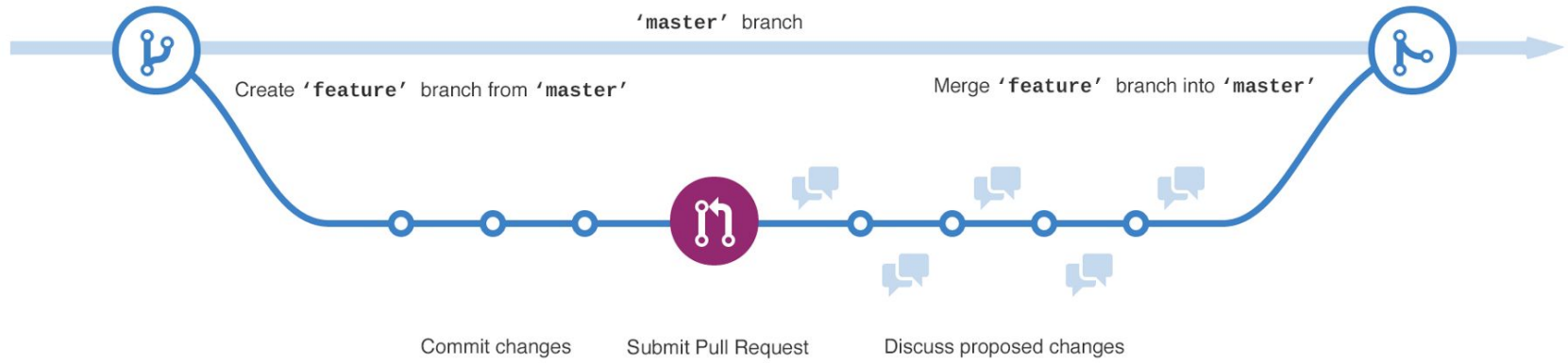
Для git существуют следующие популярные рабочие процессы:

- gitflow;
- github flow (для GitHub);
- gitlab flow (для GitLab).

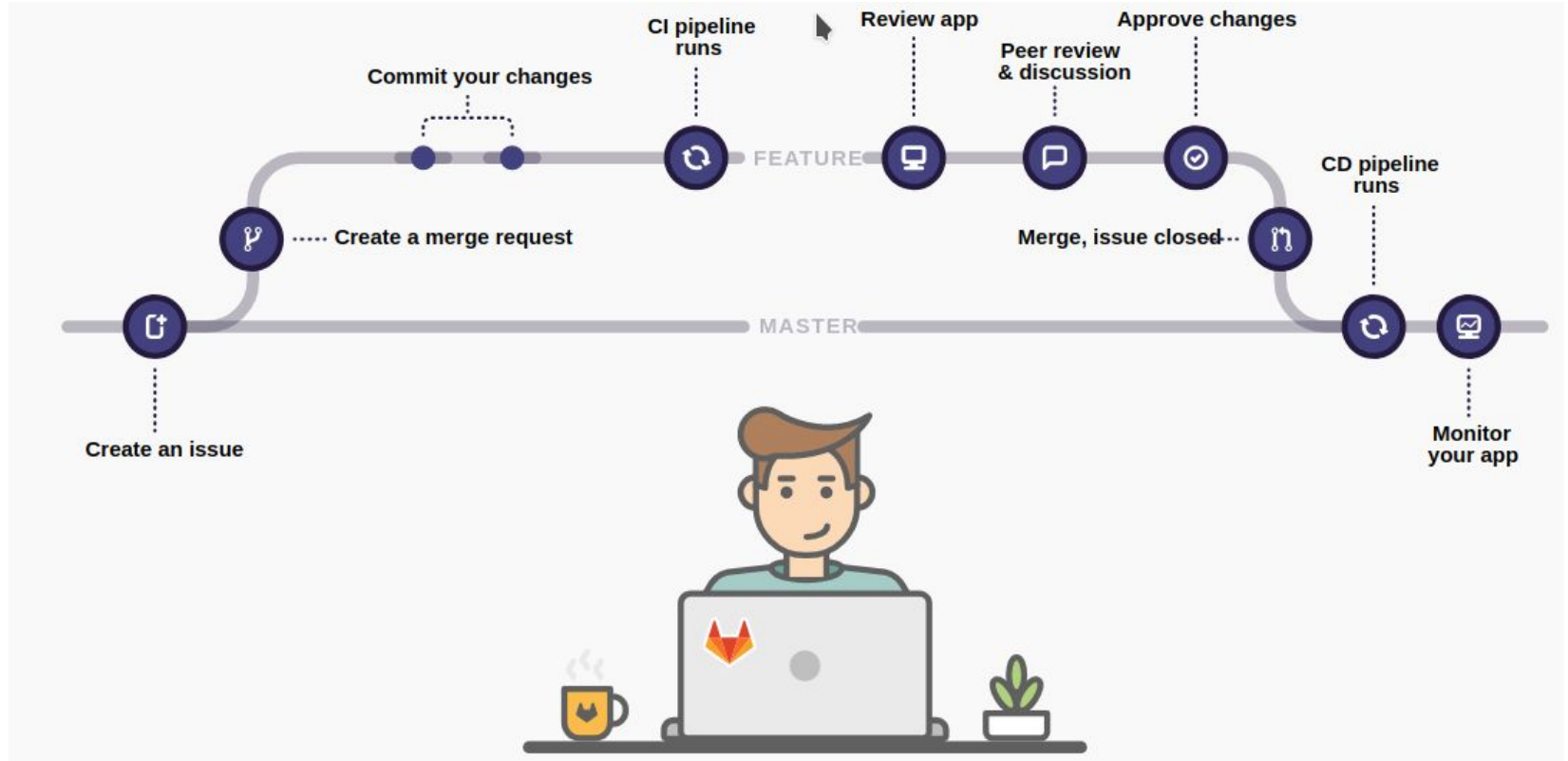
Git flow



Github flow



Gitlab flow



Репозиторий для л/р по ОС

<https://github.com/vladdy-moses/ulstu-ivk-os-2019>

Будем пользоваться github flow: для успешной отправки своего кода в репозиторий необходимо сделать fork репозитория себе (то есть создать копию своего репозитория из исходного), а затем делать pull requests в исходный репозиторий.

Что сделать? Домашнее задание?

<https://githowto.com/ru>