Для начала работы с центральным репозиторием, следует создать копию оригинального проекта со всей его историей локально.

переходим в нужную директорию (с помощью cd)

Клонируем репозиторий, используя протокол http:

git clone https://github.com/Interbellum/collective\_development.git

Теперь в нашей директории находится полная копия проекта.

Переходим в директорию /collective\_development

Вводим Git branch –(перечисляет существующие ветки и отмечает активную(на которой вы сейчас находитесь)

Теперь у нас есть локальная копия репозитория, теперь можно приступить непосредственно к разработке.

**Стандартная работа с гитом любого члена команды разработчиков.**

Допустим на нас завели тикет #150 с задачей “разработать Quality Requariments”(или выписываем его на себя сами)

Открывем консоль и Переходим в нашу дирректорию (в моем случае у меня находится в домашней папке)

cd /collective\_development

git branch (узнаю на какой нахожусь ветке, по умолчанию master, убеждаемся в этом )

(Если вдруг вы находитесь на другой ветке, переходим на master командой )

git checkout master

git pull (обновляем все локальные файлы с удаленного репозитория)

Теперь нам надо отрастить собственную ветку (Личную)

git checkout –b <name\_branch ( например PA\_QR что значит PA инициалы человека, и смысл ветки QR (quality requariments))

git push origin PA\_QR (расшариваем если нам нужно нашу ветку, то есть теперь она будет доступна для всех, делается это единажды)

git merge master (мерджим нашу ветку с master)

Теперь отращиваем ещё одну веточку только уже от нашей, собственной созданной ветки)

git checkout –b PA\_QR#150 ( называем ее по номеру тикета, который будем решать на нашей ветки)

Далее работаем. Изменяем файлы, создаем новые и тд…..

После работы нужно применить изменения, для этого командой add добавляем все измененные файлы

Git add file.py fil2.pe file3.py

Git commit –m “#150” (вводим в ковычках комментарии с кратким опсианием причиной коммита)

Git checkout PA\_QR (переходим на нашу «старшую» ветку)

Git merge PA\_QR#150 (применяем изменения с ветки PA\_int#150 на нашу PA\_QR)

Git push (отправляем изменения на удаленный репозиторий, теперь все изменения видны всем)

Git branch –d PA\_QR#150 (удаляем нашу временную ветку, где мы решали нашу задачу по данному тикету)

Закрываем наш тикет .

И заводим новый тикет на интеграцию нашего изменения Интегратору.

В тикете явно указываем на цель тикета. И указываем имя ветки на которой находятся исправленная

------------------------------------------------------------------------------------------------

Процесс интегрирования (работа интегратора)

(выше описана основная идея работы с гитом, здесь алгоритм будет в основном тот же)

Допустим на нас завели тикет на интеграцию номер #111. Проинтегрировать изменения члена команды AP с ветки AP\_QR.

Открывем консоль и Переходим в нашу дирректорию (в моем случае у меня находится в домашней папке)

cd /collective\_development

git checkout master

git pull (обновляем все локальные файлы с удаленного репозитория)

Теперь нам надо отрастить собственную ветку (Личную)

git checkout –b <name\_branch ( например PA\_int что значит PA инициалы человека, и смысл ветки int (integrated))

git merge master (мерджим нашу ветку с master)

Теперь отращиваем ещё одну веточку только уже от нашей, собственной созданной ветки)

git checkout –b PA\_int#150 ( называем ее по номеру тикета, который будем решать на нашей ветке)

Далее работаем. Изменяем файлы, создаем новые и тд…..

применяем изменения, для этого командой add добавляем все измененные файлы.

Git add file.py fil2.pe file3.py

Git commit –m “#111” (вводим в кавычках комментарии с кратким описанием причиной коммита)

Git checkout PA\_int (переходим на нашу «старшую» ветку)

Git merge PA\_int#111 (применяем изменения с ветки PA\_int#111 на нашу PA\_int)

Делаем Sanity тесты, проверяем что наш проект работает, основные функции.

Если вдруг наш проект «упал» , Sanity тесты не прошли, значит закрываем наш тикет на интеграцию, и заводим новый тикет на человека кто просил проинтегрировать его изменения, с описанием полной причины, какие тесты не прошли, и тд.

Если Sanity тесты прошли успешно, и наш проект работает как ожидается

Git push (отправляем изменения на удаленный репозиторий)

Git branch –d PA\_int#150 (удаляем нашу временную ветку, где мы решали вопрос интеграции по данному тикету)

Git checkout master

Git merge PA\_int

Git push

Закрываем наш тикет .

Возникновение конфликта.

Конфликт может возникнуть в следущей ситуации, если мы отрастили ветку «ветка1» от Master для решения какого то тикета.

Не закончив работу над этим срочно пришлось отрастить ещё одну ветку «ветку2» для решения второго тикета.

Сделали срочную работу над тикетом2, произвели слияние.

Начали работать над первым тикетом, на ветке1, но теперь мастер указывает на другой коммит, нежели до начала работы над данным тикетом.

 Если вы изменили одну и ту же часть файла по-разному в двух ветках, которые собираетесь слить, Git не сможет сделать это чисто. Если ваше решение проблемы одного тикета изменяет ту же часть файла, что и другой тикет , вы получите конфликт слияния.

Git не создаст новый коммит для слияния и выдаст ошибку.

git status – покажет вам информацию об конфликте.(укажет в каком файле произошел конфликт(ы)

Ваш файл в котором произошел конфликт содержит секцию, которая выглядит примерно так:

<<<<<<<Head:file.py -ниже до знаков ===== находится версия кода на котором вы были, так как именно на неё вы перешли перед выполнением команды merge

«Сдесь текст кода, который был в файле “file.py” на ветке head

==========

«сдесь текст кода, который был на второй ветке»

>>>>>>>>>>>PA\_#150 file.py название файла тот же файл только со второй ветки

Для того чтобы исправить нужно заменить весь текст что после «<» и до последнего знака «> « на какой то из этих двух вариантов кода, или сделать другой вариант кода.

*Но это уже дело не интегратора, так что заводим тикет с полным описанием и указанием проблемы на человека, кто работал над этим файлом.*

После того как решена проблема с конфликтами , отмечаем исправленный файл командой add fily.py и так для каждого конфликтного файла. Индексирование данных файлов будет означать что конфликты исправлены.

Можете выполнить git status ещё раз, чтобы убедиться, что все конфликты были разрешены.

Если вы довольны тем, что получили, и удостоверились, что всё, имевшее конфликты, было проиндексировано, можете выполнить git commit для завершения слияния