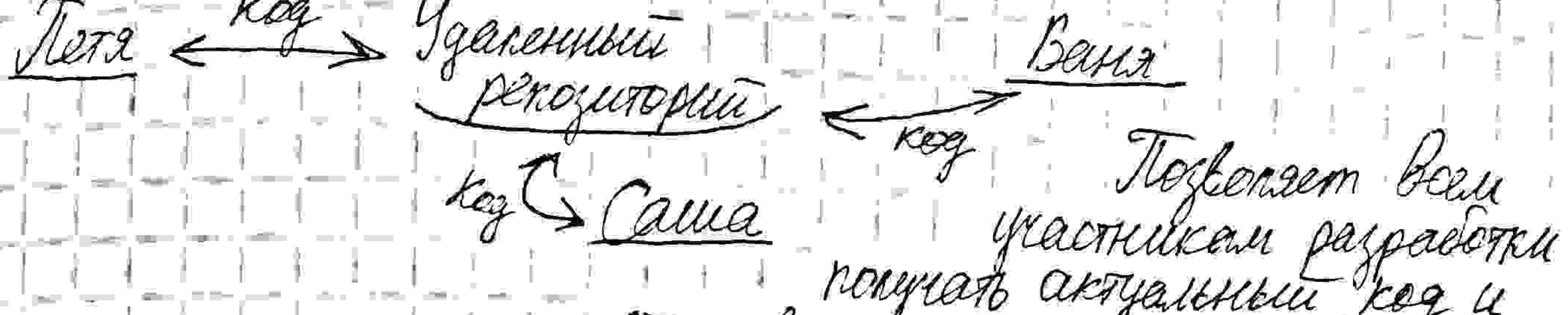
Система контроля версий Git

Назначение и возможности

- Имеется история разработки. (журнал изменений)
- Разграничение прав доступа к коду.
- контроль нескольких версий



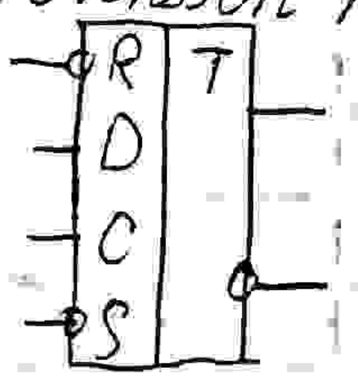
Позволяет всем участникам разработки получать актуальный код и откатываться к предыдущим версиям.

Базовые операции

Создадим аккаунт на Github.

логин Illuqi Svensson / wolfin1@mail.ru

пароль:



Рабочее название компонента.

Добавим новый репозиторий:

"+▾" → "New repository".

Зададим имя, тип лицензии и сделаем его публичным.

Скачаем графическое приложение Smartgit (syntevo.com)

Устанавливаем. Зададим некоммерческое использование

имя почта, встроенный SmartGit SSh client, далее, далее.

После установки нужно склонировать репозиторий в локальный компьютер.

- На сайте копируем адрес в колонке "Clone or download" и вставляем в программе "Repository" → "clone"

Указываем папку, в которой будет репозиторий.

Зайдем в папку и создадим файл с некоторым кодом.

SmartGit отметит новый файл как Untracked

Добавим его в репозиторий кнопкой Commit

кнопка commit добавляет файл в локальный репозиторий.

Любые изменения файла теперь будут фиксироваться с надписью modified.
Все коммиты хранятся в локальной папке .git (скрыта)
Теперь отправим все на сервер кнопкой Push
введем логин/пароль от Github.

Тонкости настройки SmartGit

- "Repository" → "Settings" → должен быть указан Name и Email.
- "Edit" → "Preferences" → "Executables" → должна быть ссылка на git.exe
  Если нет, то скачать и указать путь.

Откат изменений кода

Базовые операции: Clone, commit, Pull, Push.
Удалять можно в самом SmartGit перед commit, причем файл будет удален и в локальном репо.
Для отката кода до последнего commita есть кнопка „Discard" функция похожа на „Ctrl+Z". Уже закоммиченный код не откатывает и действует в локальном репо.
Чтобы откатить закоммиченные файлы нужно открыть „Log" выбираем до куда хотим откатить и жмем „Revert"
Так же можно откатиться до последнего пуша, при этом все изменения и файлы до и после коммитов исчезнут.
Так же выбираем степень отката и жмем „Reset", указав, как будем откатывать.

Discard - откат до commit'a
Revert - откат после commit'a
Reset hard - откат до Push'a

Работа с ветками

Лучше иметь под руки рабочую версию программы, а разра-

ботку вести параллельно.
Основная и главная ветка называется master.
Сделаем еще одну ветку:
"Local branches" → "Add branch" – создаем ветку.
" "Checkout" – переходим в нее.
Таким образом мы получаем копию содержимого ветки master

В ней можно увидеть изменения всех веток.
Все ветки используют одну и ту же папку (репо)

Когда изменения отдельной ветки нужно добавить к master нужно выбрать эту ветвь и нажать „merge"
Такое слияние коснется все в сторонюю ветку. Если все хорошо то переходим в ветку master и повторяем процедуру.
После слияния вторую ветку можно удалить.

Все ветки и их содержимое можно так же запушить на сервер и все увидят эти ветки.

Конфликты в Git

Конфликты могут произойти тогда, когда сливаются два одинаковые файла, которые редактировались параллельно друг другу. Git не может определить, какие изменения принять.
Есть несколько вариантов решения:
• „Abort merge" – отменяет слияние
• Conflict solver (жмем на конфликтный файл)
Открывается окно. По середине итоговая ветка, по бокам – сливаемые.
Редактируем код вручную на основе двух веток и отправляем в репозиторий.
Не забываем сами .master ветку со стороны и запушить все на сервер.
Если ветка не нужна, то её можно удалить как локально, так и на сервере сразу.

Рекомендации по Git

Нужно иметь мастер версию стабильную и ветку для разработок. Сливать в мастер только подтвержденные и протестированные фичи.

Не стоит добавлять в репо весь проект, особенно бинарники. Они занимают много места и охватывают множество файлов. При коммите и пуше их нужно игнорировать.

Жмем на нужный файл и выбираем „ignore" при этом появится файл .gitignore со списком игнорируемых файлов.

Если такие файлы уже есть в репо, то игнорировать нельзя, нужно предварительно все удалить.

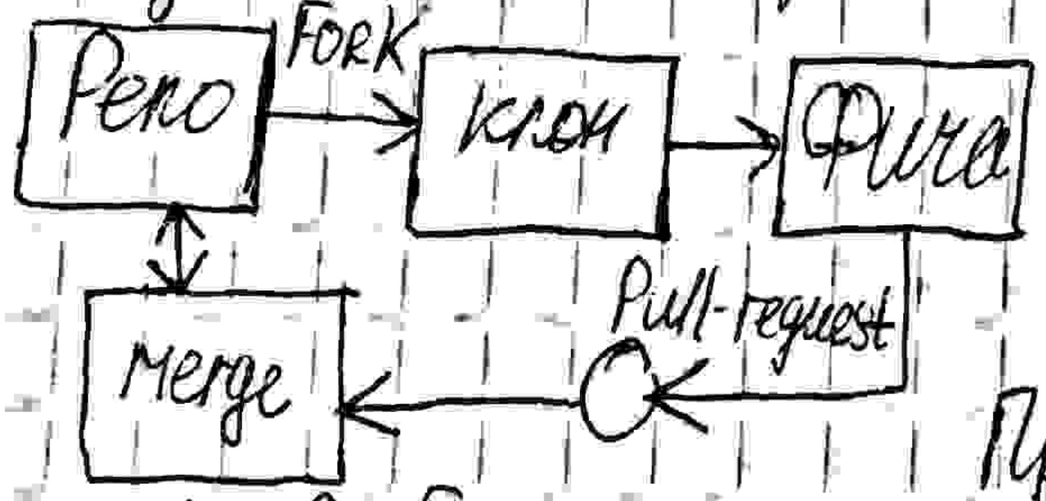
Нужно быть осторожнее при слиянии с мастер-веткой. Удаленные в feature ветке файлы могут пропасть и в мастере.

Лучше делать отдельные коммиты под каждую задачу. Так проще понять объем изменений и проще откатить.

Не стоит править участки кода в нескольких ветках сразу, так как это время-затратно и приведет к конфликту. Для этого можно выбрать в логе нужный коммит и слить его кнопкой „Cherry-pick".

Pull Request

Предложение по изменению репозитория.



Позволяет разработчикам предлагать свои варианты кода владельцу репозитория.

Процедура:

1. Выбираем интересующий репозиторий.
2. Получаем клон кнопкой fork.
3. Создаем новый файл и кодим.
4. Сохраняем файл коммитом.
5. Предлагаем свой вариант кнопкой New Pull request

Хозяин репозитория увидит request и сможет слить или отклонить предложение, а так же прокомментировать.

Консольный git-клиент

Консольный Git достаточно стандартизован. Даже графическое приложение использует консольные команды.

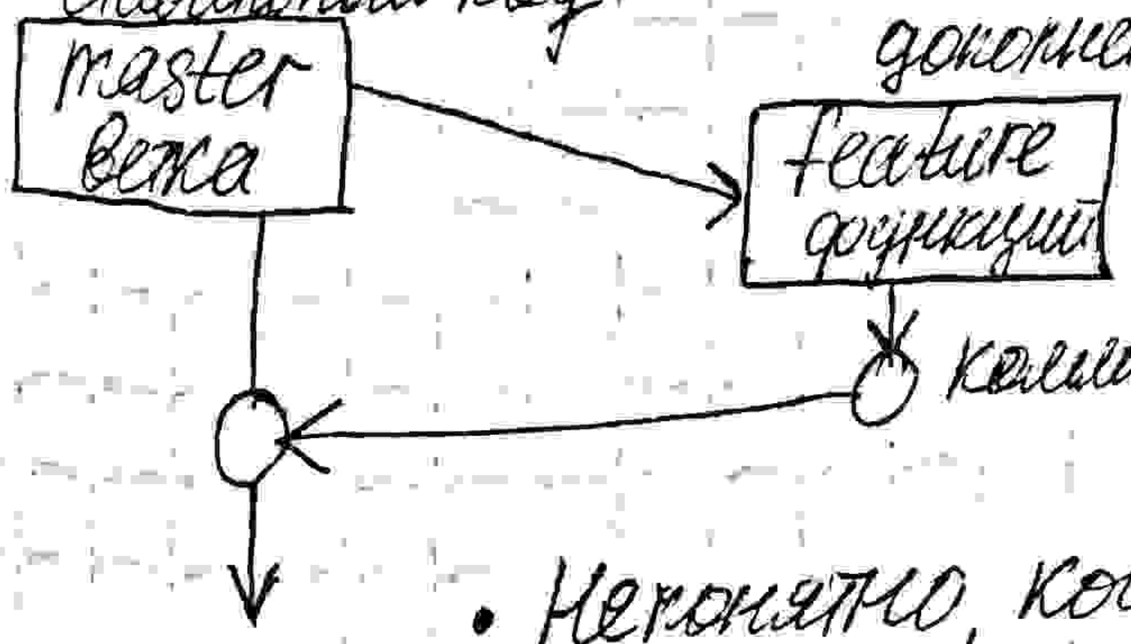
Скачаем любой консольный клиент.

Основные команды:

git clone "адрес репо с github"
git status - показать состояние свежих коммитов.
git add "имя файла" - добавить файл для коммита.
git commit - коммит добавленных.
ключ -a - коммит всех (даже недобавленных)
-m "текст" - добавить сообщение.
git push origin - закинуть на сервер в ветку origin
git log - история ветки.
git branch -a - посмотреть ветки
git branch "имя" - создать ветку
git checkout "имя" - перейти в ветку
git push "origin" "feature" - пушим ветки.
git merge "имя" - слить ветки
git branch -d "имя" - удалить ветку локально
git push origin --delete "имя" - удалить ветку на сервере.

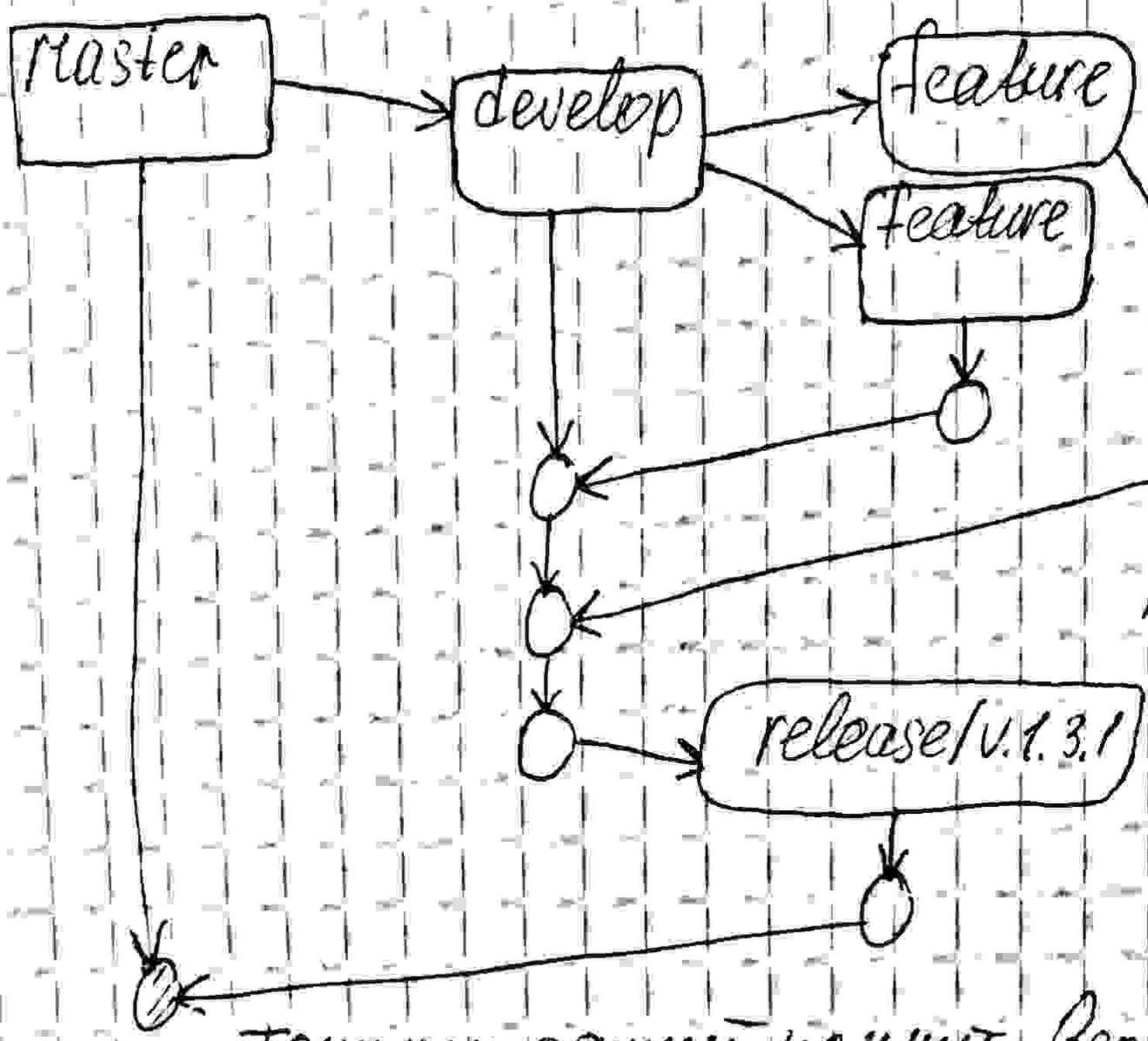
Работа с версиями программы

Простой вариант:



Проблемы:
- хорошо работающие версии в функциональных ветках не гарантируют работоспособность при совместном слиянии в мастер.
- Непонятно, когда нужно выпускать новый релиз.

Более правильный вариант:



Отвечающая от мастер ветки.
разрабатывают фичи в своих ветках и сливают с веткой develop.
При достаточном количестве изменений выпускают новый релиз.

Релиз тестируют и если все хорошо, то сливают с мастером.

Текущий данный коммит версии релиза.
Гит сохраняет состояние кроликов и в любой момент можно создать ветку с состоянием, как в текущей ветке.
Так можно делать с любыми коммитами.

Private версия Git отличается от Public, но команды и принцип действия одинаковые.
Публичная версия имеет ограничение в 1-100 Гб.
Для приватной версии свои условия.