**Лекция. Работа с распределенной системой контроля версий Git.**

**Введение**

Сегодня мы продолжаем освоение СКВ и рассмотрим наиболее популярную из них - git.

Git (произнoсится «гит») — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день его поддерживает Джунио Хамано. Git свободная система и распространяется она под лицензией GNU GPL 2.

Предназначена она в основном, для работы распределенной команды разработчиков. То есть разработчики могут находиться в разных концах света и работать над одним проектом.

Нет, Git используют не только разработчики, но и дизайнеры, писатели, редакторы, проектировщики, переводчики. GitHub часто используют hr специалисты и hr менеджеры для поиска успешных кандидатов на те, или иные вакансии в различных областях. Для математического анализа успешности проектов.

Система Git очень экономична и не требует рассылки большого количества файлов. Отслеживаются и пересылаются изменения в файлах и ссылки на эти изменения. То есть основная рассылка это рассылка разницы в ваших редактированиях.

На этом занятии Вы должны научиться устанавливать и настраивать git для работы с локальным репозиторием.

**Какие файл не должны попасть в репозиторий**

Первый и может быть наиболее важный вопрос, который следовало бы рассмотреть - это безопасность системы для разработчика. Если в организации вашу сеть защищает системный администратор. То обычному пользователю приходится эти вопросы решать самостоятельно.

Что не должно попасть в удаленный репозиторий: лишние файлы конфигурации, исполняемые файлы, временные файлы, ключи к репозиторию, платежные ключи (когда вы работаете над платежными сервисами и системами, например, Amazon). В ветки master и release не должны попасть изменения от начинающих, неопытных программистов и некачественный код, не прошедший тестирование.

**Локальный контроль версий**

Git можно установить локально. Для локальной работы над проектом Вам не обязательно даже будет подключение к интернету. Репозиторий который вы создаете локально - это фактически такой же репозиторий как на сервере, только без возможности работать над проектом распределенной команды разработчиков. Система контроля версий - по существу это реализация права программиста на ошибку в своем коде. Как вы помните «человеку свойственно ошибаться». Так вот, ошибаться программисту следует гораздо меньше. И, чтобы вернуться к своему коду и исправить ошибку, добавить функциональность и существует такая замечательная система, как система контроля версий. Кроме того использование системы контроля версий — это более эффективный менеджмент.

**Контроль версий для командной работы**

Ответьте на вопрос: В чем заключается работа в команде и вы поймете насколько управление проектом становится более эффективным с системой контроля версий. Еще одно преимущество: Система контроля версий облегчает процесс ревью (пересмотра, доработки) кода. Следующее преимущество: Автоматическое разрешение конфликтов. То есть объединение, например разных версий методов, сделанных разными разработчиками в одну ветку.

**Git-хостинг**

Git-хостинг на разных условиях предлагают многие компаний.  
Довольно известные из них: Github, Sourceforge, Google Code, GitLab, Codebase и тд.

Сервисы git в порядке популярности(по моему мнению):

1. Ваш локальный сервис, использование git только локально
2. GitHub
3. BitBucket
4. GitLab

**Предметная область и основные термины**

**Система контроля версий** - 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.

**Git** - одна из распределенных систем контроля версий.

**GitHub** - один из сервисов для использования системы контроля версий Git.

**repository** - некоторое хранилище файлов, ссылок на изменения в файлах

**commit** - отслеживание изменений, сохраняет разницу в изменениях

**HEAD** - (специальный указатель) символическая ссылка на последние изменения. Примечание: Не обязательно ссылается на commit. Может указывать на ветвь. HEAD используется репозиторием для определения того, что выбрано с помощью checkout. Обратите внимание на это различие: «head» (в нижнем регистре) относится к любому из названных заголовков в хранилище; «HEAD» (верхний регистр) относится исключительно к текущему активному заголовку (ссылке). Это различие часто используется в документации Git. HEAD может указывать на именованную вершину какой-либо ветки или на commit.

**Объекты Git**. Четыре типа объектов: Blob, Tree, Commit и References.  
Ветвь определяется не в самом Git, а наследуется от операционной и файловой систем. Более подробно об объектах Git вы можете прочитать в документации.  
**git сервисы** - сервисы предоставляющие услуги для пользователей git.

**working directory** - рабочий каталог на вашем компьютере

**staging area** - область подготовленных файлов или рабочая область

**branch** - ветка, состоит из набора коммитов, обычно ссылается на последний коммит

**merge** - слияние, слияние веток в одну

**pull** - втянуть, взять проект с сервера, получить изменения из удаленного репозитория

**push** - вытолкнуть, отправить изменения на сервер

**Практическая часть**

Для использования системы git вам нужно:  
1. Установить программу git на вашей системе.  
2. Настроить программу и проверить её работоспособность локально  
3. Зарегистрировать ваш аккаунт на GitHub  
4. Создать локальный репозиторий или копировать репозиторий существующего проекта  
5. Написать файл README.MD.  
6. В случае, если вы начинаете проект, создать удаленный репозиторий  
7. Фиксировать изменения локально  
8. Отправлять изменения на GitHub  
9. Зарегистрировать аккаунты разработчиков вашего проекта  
10. Выдать им ссылку на проект

1. **Установка git**

В Linux:  
sudo apt-get update && sudo apt-get upgrade *#обновление перед установкой*

sudo apt-get install git *#установка git*

В Windows (<https://gitforwindows.org/>):

После установки вы можете кликнуть правой кнопкой мышки на папке в проводнике Windows и выбрать открыть «Git Bash Here». Git Bash Here — означает отрыть терминал git здесь.

В терминале введите команду, чтобы проверить версию вашего git.  
git --version *#вывод версии*

В Linux,(Ctrl+Alt+T — терминал, если у вас не назначены другие горячие клавиши) откройте терминал и введите

git --version

В случае успешной установки на консоль выведется версия вашего git.

**2.Настройка программы Git**

*Примечание:*

Тут следует упомянуть, что настройку Git вы осуществляете на нескольких уровнях. То есть некоторые настройки вы делаете для определенного пользователя операционной системы(не системы git, а операционной системы). Другие настройки вы делаете для всех пользователей операционной системы. Далее вы можете делать настройки для определенной папки(локально). Вы делает настройки для репозитория находящегося на сервере. Эти настройки вы можете не делать, если работаете только со своим локальным репозиторием.

В этой лекции мы рассматриваем команды консоли в основном Windows git-bash.  
Более подробно об отличиях вы можете посмотреть в документации. Упомяну только что есть некоторые отличия. Например, в параметрах пути к файлам и папкам. В Windows используется / — слеш, а в Linux обратный слеш \. Хотя в Windows вы можете самостоятельно настроить на использование обратного слеша.

Настройка пользователя и емейл:

git config --global user.name "My Name"

git config --global user.email myEmail@example.com

Чтобы ввести настройки только одного репозитория, перейдите в его папку и сделайте то же без --global.

Настройка внешнего редактора для Linux:  
git config --global core.editor emacs *#подключить внешний редактор emacs*

Вы можете выбрать другой текстовый редактор. Например, не emacs, a vi или nano или другой на ваше усмотрение.

В Windows, такой командой вы можете задать текстовый редактор, например notepad++.

Для x64 Windows вы можете использовать команду (только пропишите свой путь):  
git config --global core.editor "'C:/Program Files (x86)/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"

Настройки git хранятся в файлах. Git проверяет 4 места для файла конфигурации (здесь в Linux):

* Файл вашего компьютера .gitconfig.
* Ваш пользовательский, файл вашего пользователя .gitconfig файл находится в ~/.gitconfig.
* Второй пользовательский файл конфигурации, расположенный в $ XDG\_CONFIG\_HOME/git/config или $HOME/.config/git/config.
* Конфигурационный файл локального репозитория: .git/config

cat .git/config *#просмотр конфигурации локального репозитория*

Каждый файл добавляет или переопределяет параметры git, определенные в файле над ним:

* Конфигурация системы.
* Конфигурация пользователя.
* Конфигурация, специфичная для репозитория.

Вы можете просмотреть файлы конфигурации:

Для системы и всех пользователей:

git config --system --list

git config --system –edit

Для пользователя:  
git config --global --list

git config --global --edit

Проверка настроек вашей конфигурации git:  
git config --list *#вывести на экран конфигурацию.*

Если список большой, вы можете пролистывать его с помощью стрелок клавиатуры или «pg up», «pg dn». Для выхода клавиша q.

Посмотреть какая конфигурация, где установлена можно так:  
git config --list --show-origin

**Для чего нужно рассмотреть консольные команды?**

На этот вопрос многие сразу скажут: «Зачем, ведь существуют UI».   
Часто в консоли вы можете сделать, что-то гораздо быстрее. С помощью набора консольных команд вы сами в будущем сможете автоматизировать процесс. Консольные команды более гибкий инструмент. Почему? Да потому что ваш UI может и «не знать» о существовании той или иной команды. UI может вообще отсутствовать как таковой, например, на сервере ввиду своей небезопасности. На первом этапе консольные команды во многом помогут в общем понимании того как работает система. Все их запоминать нет необходимости. Вы в любой момент сможете найти справку по той или иной команде. Теоретические знания, без которых никуда, лучше усваиваются с применением на практике.

**Команды Git(консольные)**

git опции команда аргументы

Пример:  
git branch -d <name> *# удалить локальную ветку с именем name*

git branch -d bugFix00 *#удалить локальную ветку с именем bugFix00.*

Опции:  
-C - использовать указанную папку репозитория вместо текущей папки;  
-c параметр=значение - использовать указанное значение параметра конфигурации;  
-p - прокручивать весь вывод с помощью less;

**Инициализация локального репозитория.**

1. Переходим в папку проекта.

cd ваша\_папка *#команда терминала, переход в папку с именем ваша\_папка*

1. Инициализация локального репозитория

git init *#инициализация локального репозитория*

1. Добавление всех файлов и папок начиная с текущей

git add . *#тут мы добавляем все.*

Можно добавить отдельный файл, например:

git add имя.расширение

Таким образом, мы говорим - отслеживать изменения нашего файла.  
Для добавления всего в папке рекомендуют использовать команду:

git add -A

1. Создание commit

git commit *#сохранить изменения в локальном репозитории*

-m «комментарий» #аргумент создания комментария коммиту. Ваши изменения будут уже с осмысленным комментарием.

Вы можете использовать полное имя ключа, вместо его сокращения. К примеру, вместо -m вы можете использовать --message=«комментарий»

git commit --message="$Ваш осмысленный комментарий"

Тут нужно сделать некоторое замечание. Чтобы использовать русские буквы в комментариях, нужно сделать предварительные настройки. Вам нужно настроить кодировку символов в системе, кодировку символов в текстовом редакторе или IDE, кодировку символов в терминале, кодировку символов в git. Также Windows «не любит» одиночные апострофы, поэтому коментарий заклечен в кавычки. На Linux есть возможность использовать апострофы при оформлении коментария к коммиту.

1. Показать изменения внесённые вашим коммитом

git show *#показать изменения внесенные вашим коммитом*

1. Просмотреть текущее состояние git:

git status *#просмотр текущего состояний git*

Показывает информацию - какая ветка текущая.  
Какие файлы изменены и тд. Команда показывает, что находится в рабочей области(в staging area).

**Источники**

1. Вячеслав Мищенко. [Работа с распределенной системой контроля версий Git на примере GitHub](https://habr.com/ru/post/451662/#nav1)
2. [Git - для новичков - #1 - основы](https://www.youtube.com/watch?v=PEKN8NtBDQ0)
3. [Изучение GitHub в одном видео уроке за 15 минут!](https://www.youtube.com/watch?v=JfpCicDUMKc)