Лабораторная работа № 1

Тема работы:

Реализация многопоточности.

Теоретическая часть:

Git — это бесплатная распределенная система контроля версий с открытым исходным кодом, предназначенная для быстрой и эффективной работы с любыми проектами - от небольших до очень крупных. Git прост в освоении, занимает мало места и обладает молниеносной производительностью. Он превосходит такие SCM (Supply Chain Management – пер. Управление цепями поставок)-инструменты, как Subversion, CVS, Perforce и ClearCase, благодаря таким возможностям, как дешевое локальное ветвление, удобные области хранения и множество рабочих процессов.

Ветвление и слияние

Особенностью Git, которая действительно выделяет его среди практически всех других SCM, является модель ветвления. Git позволяет и поощряет создание нескольких локальных веток, которые могут быть совершенно независимы друг от друга. Создание, объединение и удаление этих ветвей занимает считанные секунды.

Это означает, что вы можете делать такие вещи, как:

- **Бесконтактное переключение контекста.** Создайте ветку для опробования какой-либо идеи, сделайте несколько фиксаций, вернитесь к тому месту, откуда была сделана ветка, примените исправление, вернитесь к тому месту, где проводились эксперименты, и объедините его.
- **Ролевые коделайны.** Иметь ветку, которая всегда содержит только то, что отправляется в продакшн, другую, в которую сливается работа для тестирования, и несколько небольших веток для повседневной работы.
- **Рабочий процесс, основанный на функциях.** Создавайте новые ветки для каждой новой функции, над которой вы работаете, чтобы можно было плавно переключаться между ними, а затем удаляйте каждую ветку, когда функция будет объединена с основной линией.
- **Одноразовые эксперименты.** Создайте ветку для экспериментов, поймите, что она не работает, и просто удалите ее, бросив работу, и никто больше не увидит ее (даже если за это время вы продвинули другие ветки).

Распределенный

Одной из самых приятных особенностей любой распределенной SCM, в том числе и Git, является ее распределенность. Это означает, что вместо того, чтобы выполнять "выдачу" текущей вершины исходного кода, вы выполняете "клонирование" всего репозитория.

Множественное резервное копирование

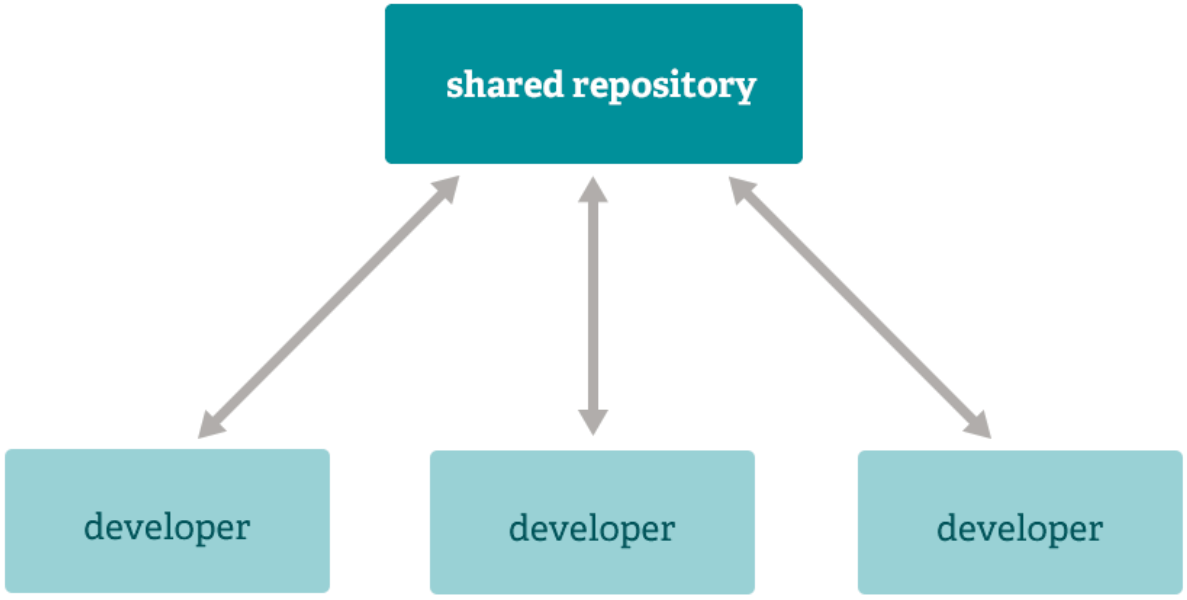
Это означает, что даже если вы используете централизованный рабочий процесс, каждый пользователь, по сути, имеет полную резервную копию главного сервера. Каждая из этих копий может быть поднята вверх, чтобы заменить основной сервер в случае сбоя или повреждения. По сути, в Git нет единой точки отказа, если только не существует единственной копии репозитория.

Любой рабочий процесс

Благодаря распределенной природе Git'a и превосходной системе ветвлений можно с относительной легкостью реализовать практически бесконечное число рабочих процессов.

Рабочий процесс в стиле Subversion

Централизованный рабочий процесс встречается очень часто, особенно у людей, переходящих с централизованной системы. Git не позволит вам выполнить push, если кто-то выполнил push с момента последнего извлечения, поэтому централизованная модель, при которой все разработчики выполняют push на одном сервере, работает просто замечательно.



Переведено с помощью www.DeepL.com/Translator (бесплатная версия)

Источник: https://git-scm.com/

Установка Git

Скачивайте установщик (https://git-scm.com/downloads), запускаете установщик, выбираете нужные для вас опции (либо оставляете все как есть, если не сильно разбираетесь), ожидаете завершения процесса установки и готово.

Использование Git

В рабочей папке где находятся исходные файлы вашего приложения/проекта, через терминал или командную строку используете команду **git init** (оф. документация - https://git-scm.com/docs/git-init) для инициализации локального репозитория, после чего Git будет отслеживать изменения исходных файлов и для фиксирования этих самых изменения - используйте команду **git commit** (оф. документация - https://git-scm.com/docs/git-commit). В дальнейшем если необходимо синхронизовать изменения в общем для всех репозитории – используйте команду **git push** (оф. документация - https://git-scm.com/docs/git-push)

Многопоточность

Многопото́чность (англ. Multithreading) — свойство платформы (например, операционной системы, виртуальной машины и т. д.) или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение может достичь более эффективного использования ресурсов вычислительной машины. Такие потоки называют также потоками выполнения (от англ. thread of execution); иногда называют «нитями» (буквальный перевод англ. thread) или неформально «тредами».

Сутью многопоточности является квазимногозадачность на уровне одного исполняемого процесса, то есть все потоки выполняются в адресном пространстве процесса. Кроме этого, все потоки процесса имеют не только общее адресное пространство, но и общие дескрипторы файлов. Выполняющийся процесс имеет как минимум один (главный) поток. Многопоточность (как доктрину программирования) не следует путать ни с многозадачностью, ни с многопроцессорностью, несмотря на то что операционные системы, реализующие многозадачность, как правило, реализуют и многопоточность.

К достоинствам многопоточной реализации той или иной системы перед многозадачной можно отнести следующее:

- Упрощение программы в некоторых случаях за счёт использования общего адресного пространства.
- Меньшие относительно процесса временны́е затраты на создание потока.

К достоинствам многопоточной реализации той или иной системы перед однопоточной можно отнести следующее:

- Упрощение программы в некоторых случаях, за счёт вынесения механизмов чередования выполнения различных слабо взаимосвязанных подзадач, требующих одновременного выполнения, в отдельную подсистему многопоточности.
- Повышение производительности процесса за счёт распараллеливания процессорных вычислений и операций ввода-вывода.

Источник: Википедия

Многопоточность в Java

Поток – это поток выполнения в программе. Виртуальная машина Java позволяет приложению иметь несколько потоков одновременного выполнения. Каждый поток имеет свой приоритет. Потоки с более высоким приоритетом выполняются предпочтительнее, чем потоки с более низким приоритетом. Каждый поток может быть отмечен или не отмечен как демон. Когда код, выполняющийся в некотором потоке, создает новый объект Thread, новый поток изначально имеет приоритет, равный приоритету создающего потока, и является демонским потоком тогда и только тогда, когда создающий поток является демоном.

При запуске виртуальной машины Java обычно существует единственный недемонский поток (который, как правило, вызывает метод main некоторого обозначенного класса). Виртуальная машина Java продолжает выполнять потоки до тех пор, пока не произойдет одно из следующих событий:

- Вызван метод exit класса Runtime, и менеджер безопасности разрешил операцию выхода.
- Все потоки, не являющиеся потоками-демонами, погибли, либо вернувшись из вызова метода run, либо выбросив исключение, распространяющееся за пределы метода run.

Существует два способа создания нового потока выполнения. Один из них заключается в объявлении класса подклассом Thread. Этот подкласс должен переопределить метод run класса Thread. После этого можно выделить и запустить экземпляр подкласса. Например, поток, вычисляющий простые числа, превышающие заданное значение, может быть написан следующим образом:

```
class SomeThread extends Thread {

	…

	@Override

	public void run() {

		// TODO: Здесь будет происходит магия

	}

}
```

Следующий код создаст поток и запустит его в работу:

```
SomeThread some_t = new SomeThread();

some_t.start();
```

Другой способ создания потока заключается в объявлении класса, реализующего интерфейс Runnable. Затем этот класс реализует метод run. После этого экземпляр класса может быть выделен, передан в качестве аргумента при создании Thread и запущен. Тот же пример в другом стиле выглядит следующим образом:

```
class SomeThread implements Runnable {

	…

	@Override

	public void run() {

		// TODO: Здесь будет происходит магия

	}

}
```

Следующий код создаст поток и запустит его в работу:

```
SomeThread some_t = new SomeThread();

new Thread(some_t).start();
```

Каждый поток имеет имя для идентификации. Несколько потоков могут иметь одно и то же имя. Если при создании потока имя не указано, то для него генерируется новое имя.

Переведено с помощью www.DeepL.com/Translator (бесплатная версия)

Источник: https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html

Основное задание:

1. Установить среду Git на ваше устройство, подготовьте папку для вашего приложения/проекта и проинициализируйте локальный репозиторий в этой папке (команда git init).
2. Реализуйте приложение с многопоточностью (как минимум два потока). В команде, распределите кто какую часть приложения будет реализовывать, что эта часть будет из себя представлять и какую(-ие) функцию(-и) будет(-ут) выполнять.
3. Сделайте commit-ы на изменения в коде приложения/проекта (команда git commit). Создайте аккаунт на GitHub (https://github.com/) (если нет такого) и отправьте преподавателю юзернейм или ссылку на профиль. После получения дальнейших указаний от преподавателя, push-ите изменения в вашу отдельную ветку в репозитории на GitHub - https://github.com/FCIM-PCD/Practice-Work-RU (либо если вы согласовали реализацию собственного проекта, то на отдельный для проекта репозиторий) (команда git push).

Варианты для реализации задания:

- Реализовать простой счетчик как отдельный поток. (классический пример варианта)
- Реализовать генератор случайных слов или словосочетаний (возможно даже и предложений) как отдельный поток.
- Реализовать обработчик информации как отдельный поток.
- Реализовать расчет по математической формуле как отдельный поток.

- Реализовать проверку предложения на грамматические или синтаксические ошибки как отдельный поток.
- Реализовать имитацию бросания игральных костей как отдельный поток.
- Реализовать конвертацию валют как отдельный поток.
- Реализовать обработку информации в стеке (каноничном его понимании) как отдельный поток.
- Реализовать обработку файла как отдельный поток.

По завершению работы, составьте отчет, в котором должно быть – Ваша фамилия, имя, группа, тема работы, Ваш вариант для реализации задания, краткое описание реализации задания, ссылку на исходный код на GitHub. Сохранить отчет в формате PDF и отправить на ELSE - https://else.fcim.utm.md/mod/assign/view.php?id=5825.