Разработка научных приложений
Локальная работа с Git

Выполнение заданий, предлагаемых ниже, поможет освоить основы работы с системой контроля версий Git. Крайне рекомендуется при работе над заданиями использовать консольный файловый менеджер Far или подобный ему.

1. **Начало работы**

   Чтобы проверить, какая версия Git установлена на машине, в командной строке Windows введите следующую команду.

   ```
   >> git --version
   ```

   Если Git не установлен, необходимо его установить. Для этого скачайте установочный файл по адресу http://git-scm.com/downloads . После этого запустите его и следуйте инструкциям. Установка требует прав администратора.

   После установки необходимо «представиться» Git: указать имя пользователя и его адрес электронной почты. Для этого используется команда **git config**:

   ```
   >> git config --global user.name 'John Doe'
   >> git config --global user.email johndoe@example.com
   ```

   Чтобы узнать, что означает опция –**global**, воспользуйтесь справкой Git:

   ```
   >> git config --help
   ```

2. **Фиксация изменений. Коммиты**

   Создайте новую директорию и инициализируйте git-репозиторий:

   ```
   >> mkdir mygit
   >> cd mygit/
   >> git init
   Initialized empty Git repository in .git/
   >>
   ```

   Просмотрите теперь содержимое директории **.git**. Она является скрытой и по умолчанию не будет видна в Проводнике Windows. Попробуйте разобраться в структуре поддиректорий и содержимом конфигурационного файла.

   Теперь немного заполните директорию **mygit**. Создайте файл **foo.txt** с текстом

   ```
   1
   2
   3
   4
   5
   ```

и скопируйте его в поддиректорию **./bar** (ее тоже вначале надо создать). Чтобы узнать, что по поводу ваших действий думает Git, используйте команду **git status**. Попробуйте разобраться в полученной информации.

Теперь заставьте Git контролировать изменения в репозитории:

```
>> git add .
>> git status
```

Сравните результаты предыдущей команды **git status** и последней. Объясните, что произошло.

Сделайте первый коммит:

```
>> git commit -a -m "Первый коммит"
Created initial commit 24479dd: init
 2 files changed, 12 insertions(+), 0 deletions(-)
  create mode 100644 bar/foo.txt
  create mode 100644 foo.txt
```

Обратите внимание на опцию **-m**. То, что следует за ней — комментарий к коммиту. Он обязателен и, если его не ввести, Git откроет стандартный текстовый редактор ОС (в случае Windows это будет notepad.exe) и заставит записать комментарий в нем. Проверьте это:

```
>> git commit -a
```

Команда **git show** покажет, что произошло в последний момент:

```
>> git show
```

Попытайтесь интерпретировать результат выполнения этой команды.

Теперь немного измените содержимое файла **foo.txt** и создайте новый файл **bar.txt**, введя в него тот же текст, что и был раньше в **foo.txt**.

Изучите результат выполнения следующих команд (подсказка: diff — сокращение от difference):

```
git diff
git diff --cached
```

Используйте встроенную справку, чтобы понять, в чем разница между командами.

Теперь сделайте новый коммит, чтобы зафиксировать изменения в репозитории.

3. Метки и gitk

Если в репозитории были сделаны важные изменения, то хотелось бы каким-то образом **пометить** соответствующий коммит, чтобы потом было просто к нему вернуться. Для этого в Git используется команда **git tag**, которая позволяет присвоить **метку** (tag) коммиту.

Проверьте при помощи **git status** состояние репозитория и, если все изменения сохранены, присвойте последнему коммиту метку «init». Как это сделать, узнайте при помощи встроенной справки по команде **git tag**.

После этого сделайте любые изменения в содержимом репозитория. Закоммитьте их и присвойте этому коммиту метку «second». Разберитесь в результате команды

```
>> git diff init..second
```

Теперь примените команду, которая осуществляет одну из главных функций любой системы контроля версий — откат изменений к заданному состоянию.

```
>> git checkout init
```

Проверьте файлы в рабочей директории. В каком они состоянии, какому коммиту это состояние соответствует? Что будет, если «откатиться» к коммиту second? Проверьте ваши предположения.

Разумеется, держать в голове всю информацию о коммитах (даже только важных) занятие бессмысленное. Git располагает утилитой **gitk**, которая в сравнительно удобной форме представляет эту информацию. Попробуйте разобраться в том, что показывает эта программа:

```
>> gitk
```

Совпадает ли то, что вы видите, с тем, как вы представляли себе собственные действия в репозитории до этого?

4. Ветки и операция слияния

Сначала создадим свежий репозиторий и наполним его содержимым:

```
>> rmdir /S mygit
>> mkdir mygit
>> cd mygit
>> git init
```

Создайте те же два файла и поддиректорию **bar**, что и раньше. Зафиксируйте изменения:

```
>> git add .
>> git commit -a -m "Первый коммит"
```

Теперь познакомимся с другим основным понятием Git — веткой. Для начала под веткой можно понимать последовательность состояний содержимого репозитория. Создадим ветку **fix** (как будто возникла необходимость что-то исправить в коде, «пофиксить»).

```
>> git branch fix
```

Если последнюю команду запустить без аргумента, она покажет список имеющихся веток.

Теперь переключимся на созданную ветку.

```
>> git checkout fix
>> git branch
```

Как изменился результат вывода команды **git branch**? Подумайте, в какую ветку «истории» кода пойдет следующий коммит?

Сделайте какие-нибудь изменения (например, создайте новый файл) в ветке **fix** и закоммитьте их. Представьте мысленно, как соотносится между собой содержимое двух веток.

Теперь вернемся к основной ветке **master**:

```
>> git checkout master
```

Просмотрите директорию. Видите ли вы те изменения, которые сделали в ветке **fix**?

Выполните еще одну основную операцию — **слияние**.

```
>> git merge fix
```

Просмотрите опять поддиректорию. В каком состоянии ее содержимое? Попробуйте объяснить себе, как происходит слияние. При работе с системами контроля версий всегда важно понимать, что и как происходит с различными цепочками состояний в разных ветках. Обычно при разработке приложений используется несколько веток: одна для стабильной версии программы, другая для внедрения новых возможностей, третья для исправления багов и т.п.

5. Домашнее задание

Установите и настройте Git на своей домашней машине. Создайте репозиторий, в котором под управление Git проделайте следующие действия:

5a. Напишите на Octave скрипт, реализующий метод деления отрезка пополам для нахождения точки минимума функции одной переменной. При этом постарайтесь за время написания кода сделать несколько коммитов, не скупясь на поясняющий комментарий к нему. Просматривайте, как изменяется «история кода» при помощи утилиты **gitk**.

5b. Создайте ветку, в которой добавьте новую функциональность к вашему коду — реализуйте отрисовку графика минимизируемой функции и точек на оси абсцисс, являющихся последовательными приближениями в методе деления пополам. Не забывайте коммитить!

5c. Вернитесь к главной ветке и выполните слияние. Зафиксируйте состояние репозитория, запакуйте его в архив **.zip** (обязательно zip !) с названием вида Фамилия_Имя_ДЗ1.zip (кириллицей). Отправьте его на адрес **ogulenko.a.p@onu.edu.ua** .