ИСР4.1. Примеры использования базовых команд Git

Создание нового репозитория

git init

Чтобы создать новый репозиторий, нам нужно открыть терминал, зайти в папку нашего проекта и выполнить команду init. Это включит приложение в этой конкретной папке и создаст скрытую директорию .git, где будет храниться история репозитория и настройки.

Команда git init используется для инициализации локального репозитория.

```
$ mkdir Desktop/git_exercise/
$ cd Desktop/git_exercise/
$ git init
```

Определение состояния

git status

Команда git status — это еще одна важнейшая команда, которая показывает информацию о текущем состоянии репозитория: актуальна ли информация на нём, нет ли чего-то нового, что поменялось, и так далее. Запуск git status на нашем свежесозданном репозитории должен выдать:

```
$ git status
On branch master
Initial commit
Untracked files:
(use "git add ..." to include in what will be committed)
hello.txt
```

Подготовка файлов

git add

Команда git add используется, чтобы добавить отслеживание изменений, вносимых в файлы.
Мы можем выполнить команду к какому-то конкретому файлу: git add file.c или же к группе файлов, используя маску: git add "*.c". А также если мы хотим добавить все, что находится в директории, мы можем использовать: git add -A

```
$ git add hello.txt
```

```
$ git add -A
```

Коммит(фиксация изменений)

git commit

Коммит представляет собой состояние репозитория в определенный момент времени. Это похоже на снапшот, к которому мы можем вернуться и увидеть состояние объектов на определенный момент времени. Чтобы зафиксировать изменения, нам нужно хотя бы одно изменение в области подготовки (мы только что создали его при помощи git add), после которого мы может коммитить: git commit

```
$ git commit -m "Initial commit."
```

Отправка изменений на сервер

git push

Сейчас самое время переслать наш локальный коммит на сервер. Этот процесс происходит каждый раз, когда мы хотим обновить данные в удаленном репозитории. Команда, предназначенная для этого - git push. Она принимает два параметра: имя удаленного репозитория (мы назвали наш origin) и ветку, в которую необходимо внести изменения (master — это ветка по умолчанию для всех репозиториев).
Пример: git push origin master

```
$ git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 212 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/tutorialzine/awesome-project.git
* [new branch] master -> master
```

Клонирование репозитория

git clone

Чтобы скачать данные из репозитория и получить полностью работоспособную копию проекта, необходимо воспользоваться командой - git clone.

```
$ git clone https://github.com/tutorialzine/awesome-project.git
```

Новый локальный репозиторий создается автоматически с GitHub в качестве удаленного репозитория.

Запрос изменений с сервера

git pull

Если мы сделали изменения в нашем удаленном репозитории, другие пользователи могут скачать изменения при помощи команды git pull.
Пример git pull origin mster

```
$ git pull origin master
From https://github.com/tutorialzine/awesome-project
* branch master -> FETCH_HEAD
Already up-to-date.
```

Ветвление

git branch

Основная ветка в каждом репозитории называется master. Чтобы создать еще одну ветку, используем команду branch - git branch name

```
$ git branch amazing_new_feature
```

Это создаст новую ветку, пока что точную копию ветки master.

git branch

Сейчас, если мы запустим branch, мы увидим две доступные опции:

```
$ git branch
amazing_new_feature
* master
```

master — это активная ветка, она помечена звездочкой. Но мы хотим работать с нашей “новой потрясающей фичей”, так что нам понадобится переключиться на другую ветку. Для этого воспользуемся командой checkout, она принимает один параметр — имя ветки, на которую необходимо переключиться - git checkout amazing_new_feature

```
 $ git checkout amazing_new_feature
```

Слияние веток. Алгоритм для слияния

Шаг 1.

Наша “потрясающая новая фича” будет еще одним текстовым файлом под названием feature.txt. Мы создадим его, добавим и закоммитим:

```
$ git add feature.txt
$ git commit -m "New feature complete."
```

Шаг 2.

Изменения завершены, теперь мы можем переключиться обратно на ветку master.

```
$ git checkout master
```

Шаг 3.

Теперь, если мы откроем наш проект в файловом менеджере, мы не увидим файла feature.txt, потому что мы переключились обратно на ветку master, в которой такого файла не существует. Чтобы он появился, нужно воспользоваться merge для объединения веток (применения изменений из ветки amazing_new_feature к основной версии проекта).

```
$ git merge amazing_new_feature
```

Шаг 4.

Теперь ветка master актуальна. Ветка amazing_new_feature больше не нужна, и ее можно удалить.

```
$ git branch -d awesome_new_feature
```