Введение в git и github

Д.Г. Штукенберг, ЛНМО

4 апреля 2019 г.

1 Системы контроля версий

Когда работа над проектом занимает длительное время и (или) требует многих участников, возникает вопрос об удобном и надёжном хранилище для файлов. Такие задачи как синхронизация изменений, хранение истории изменений, различных версий файлов — они требуют больших усилий и внимательности, и всё равно файлы периодически могут быть утеряны или перепутаны.

Для облегчения работы над такими проектами были разработы системы контроля версий. Один из самых ранних примеров — система CVS. Мы же будем пользоваться системой Git — пожалуй, самой широко распространённой системой контроля версий на данный момент. Помимо распространённости, в её пользу говорит наличие бесплатных серверов, на которых можно бесплатно разместить свои проекты — Github и Bitbucket.

2 Основные понятия

Главная идея систем контроля версий: они хранят не сами файлы проекта, а их изменения. Это позволяет с небольшими затратами восстановить любое состояние вашего проекта на любой момент времени. Если вы добавили, изменили или удалили какие-то файлы, вы информируете систему о вашем действии — и по этой информации можно восстановить как самое новое состояние проекта, так и предыдущее, можно выяснить, что изменилось между несколькими версиями.

Хранилище информации по проекту в git называется *репозиторием*. Репозиторий хранит последовательность *коммитов*: изменений проекта, состоящих из каких-то добавлений, изменений и удалений файлов. Коммит не разбивается на более мелкие: либо мы целиком применяем содержащиеся в нём изменения к проекту, либо мы целиком от них отказываемся. При этом сам по себе коммит может содержать множество изменений в различных файлах.

Поясним данную идею на примере переименования переменной: представим, что в проекте из 34 файлов мы переименовали функцию `weird_function`, назвав её `strange_function`. Если изменение будет произведено только в части файлов, а в ещё 12 файлах название функции осталось старым, мы получим некомпилирующуюся программу. Поэтому всё переименование, все изменения во все 34 файла имеет смысл помещать в один коммит.

Git — распределённая система, у неё нет центрального сервера, где лежат все файлы проекта, каждая локальная копия проекта содержит весь репозиторий.

3 Порядок работы

Ниже приведён перечень шагов, необходимый для создания и работы с проектом под управлением git. Если репозиторий создаётся только на локальном компьютере, и не предполагается, что он будет доступен извне, первый шаг (создание репозитория на хосте) можно пропустить.

3.1 Создание репозитория на хосте

Данные инструкции даны в предположении, что репозиторий будет храниться (*хоститься*) на сайте `https://github.com`. Подобным же образом его можно завести на сайте `https://bitbucket.com`.

1. Зарегистрируйтесь на сайте `https://github.com/`, либо войдите в свой аккаунт, если вы регистрировались ранее.
2. Выберите пункт `Start a project` (как вариант: `New repository` из меню вверху экрана).
3. Укажите имя для репозитория.
4. Укажите права доступа (рекомендация — оставьте `public`).
5. Нажмите кнопку `Create a repository`

3.2 Создание локального репозитория

Чтобы созданный на хосте репозиторий был бы доступен на локальном компьютере, используйте команду `git clone`. Ей аргументом передаётся https-адрес репозитория, который появляется на экране (на домашней странице проекта) сразу после его создания.

```
git clone https://github.com/shd/test-project.git
```

Эта команда создаст подкаталог для вашего проекта и создаст репозиторий, аналогичный тому, что лежит на хосте. Сразу после создания он будет, естественно, (практически) пустой.

Если репозиторий сразу создаётся локально, вместо клонирования нужно выполнить команду `git init <имя проекта>` — она создаст пустой репозиторий, не привязанный ни к какому хосту.

3.3 Добавление, изменение и удаление файлов

После создания локального репозитория заходите в каталог с ним, и работайте. В тот момент, когда вы решите сделать первый коммит — добавить к репозиторию первые файлы вашего проекта — вам потребуется команда `git add`.

А именно, вам нужно перечислить все файлы, которые вы добавляете, в аргументах этой команды. Если вы, например, создали файлы `alpha.ml` и `beta.ml`, выполните следующую команду:

```
git add alpha.ml beta.ml
```

Также, команда `add` применяется, если вы изменили какой-то файл, уже ранее добавленный к репозиторию.

Если вы хотите удалить какой-то файл из репозитория, используйте `git rm <имя файла>`.

3.4 Формирование коммита

Ваши изменения не будут добавлены к репозиторию, пока вы не выполните (не сформируете) коммит:

```
git commit -m "Commit message"
```

Данное действие запомнит в репозитории все изменения проекта, о которых git был уведомлён командами `add` и `rm`. Сообщение коммита (commit message) обязательно, но его содержание может быть любым. Сообщение предназначено для вас и ваших товарищей: желательно, чтобы из него была понятна цель вашего коммита.

Если вы не знаете, что в точности будет включено в коммит, напишите команду `git status` — она укажет, в чём отличие текущего состояния проекта от последнего состояния из репозитория.

Данная команда напечатает общий список отличий: какие файлы изменены, добавлены или удалены. Если вы хотели бы узнать конкретные отличия какого-то файла, используйте `git diff <имя файла>`.

3.5 Синхронизация с хостом

В тот момент, когда вы закончили работу и хотите передать её результаты на сервер, используйте команду `git push`. После неё все новые коммиты будут доступны на хосте (если точнее, на том сервере, на котором укажете — по умолчанию новые коммиты будут отправлены на тот сервер, с которого вы склонировали репозиторий). Например, эту команду можно вызвать после занятий в классе на уроке информатики для передачи наработанного кода на сервер.

Если же вы хотите скачать локальные изменения на свой компьютер, используйте `git pull`. Например, эта команда может вами быть вызвана дома перед выполнением домашнего задания для получения выполненных на уроке заданий.

Не забывайте эти две команды! Если не выполнить их, то окажется, что те изменения, которые вы сделали в своём проекте, например, в школе, останутся там на локальном компьютере — и не будут вам доступны дома.

4 Краткий перечень команд работы с git

1. Команды для работы с репозиторием в целом.

   (a) `git clone <репозиторий>` — клонировать проект, то есть создать полную локальную копию репозитория.

   (b) `git pull` — взять свежую версию из исходного репозитория (в нашем случае: версию с github) и объединить эту версию с текущей версией на локальном компьютере. В случае значительных расхождений git может отказаться от объединения — тогда надо пользоваться командой `git merge`.

   (c) `git push` — обновить исходный репозиторий (в нашем случае: версию на github), внести в него изменения, внесённые на локальном компьютере. Если на сервере есть новые изменения, ещё не учтённые на локальном компьютере, то операция

будет отклонена, и вам потребуется сперва объединить сторонние изменения с локальными при помощи команд `git pull` и `git merge`, и только потом выложить текущий локальный репозиторий на github с помощью команды `git push`.

2. Команды для работы с коммитами и отдельными файлами из репозитория

   (a) `git add <имя файла>` — добавляет файл к проекту, если его в проекте нет. Если же файл уже в проекте — уведомляет git о том, что изменения в файле надо запомнить.

   (b) `git rm <имя файла>` — удаляет файл из проекта.

   (c) `git commit -m "<сообщение>"` — создаёт коммит, учитывая изменения во всех файлах, указанных в командах `add` и `rm`. Изменения в остальных файлах будут проигнорированы.

   (d) `git status` — печатает информацию об отличиях файлов на диске от репозитория. То есть, о том, какие файлы не отслеживаются git, какие добавлены, какие удалены, какие изменены.

   (e) `git diff <имя файла>` — печатает информацию об отличии указанного файла от репозитория.

   (f) `git log [<имя файла>]` — печатает информацию коммитах, в которые входил данный файл. Если имя файла не указывает, печатает информацию о всех коммитах.

   (g) `git checkout <имя файла>` — вернуться к версии файла, находящейся в репозитории. Все локальные изменения файла, не добавленные в репозиторий, после выполнения данной команды будут утеряны.