GeekBrains

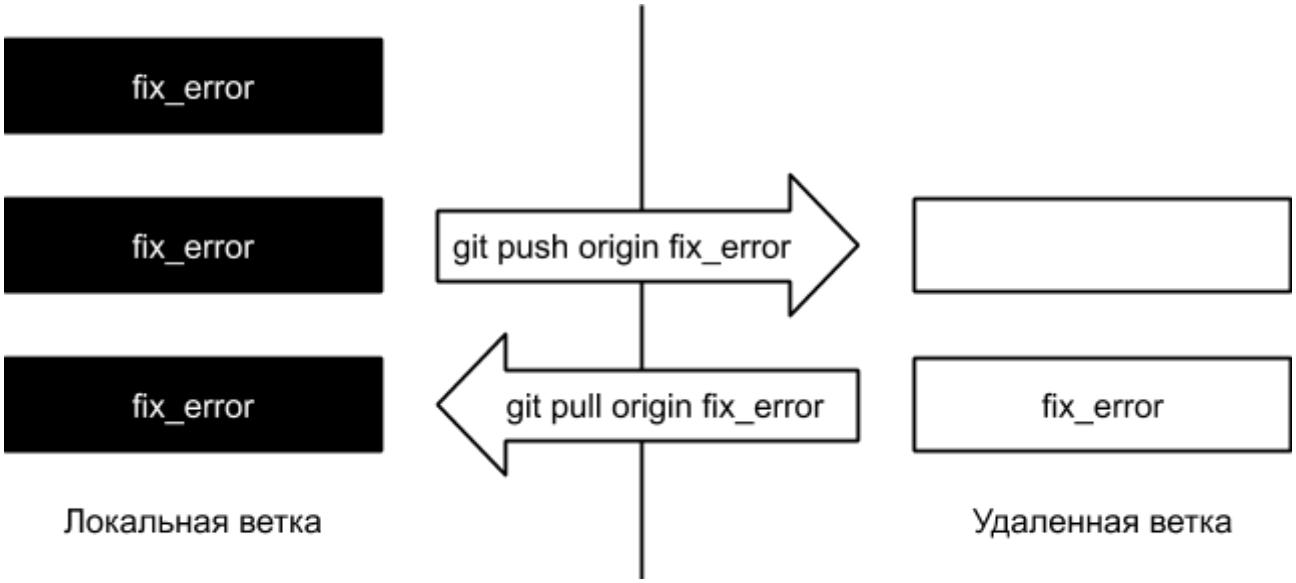
Git. Базовый курс

Урок 8

Удаленные ветки

Управление удаленными ветками

При работе с репозиторием ветка может располагаться локально и на удаленном репозитории. Пока вы не отправите коммиты ветки при помощи команды **git push**, они не появятся на удаленном сервере. Загрузить изменения из удаленного репозитория можно при помощи команды **git pull**.



Создадим локальную ветку fix_error:

```
$ git checkout -b fix_error
Switched to a new branch 'fix_error'
```

Запросим список текущих веток:

```
$ git branch
* fix_error
  master
```

Изменение локальной ветки не отражается на удаленной, пока мы не отправим изменения на сервер. Давайте в этом убедимся, перейдя в GitHub на страницу branches.

На удаленном сервере у нас находится лишь одна ветка master, о новой ветке fix_error GitHub не подозревает, так как она не была еще отправлена на сервер.

Список удаленных веток можно посмотреть и в консоли. Для этого команде **git branch** следует добавить параметр **-r**:

```
$ git branch -r
  origin/HEAD -> origin/master
```

```
  origin/master
```

Как видим, в отчете команды тоже только одна удаленная ветка — master. Обратите внимание, что она имеет суффикс в виде имени удаленного репозитория, у нас это origin.

Давайте отправим текущую ветку в удаленный репозиторий:

```
$ git push origin fix_error
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 346 bytes | 346.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'fix_error' on GitHub by visiting:
remote:      https://github.com/igorsimdyanov/hello/pull/new/fix_error
remote:
To github.com:igorsimdyanov/hello.git
 * [new branch]     fix_error -> fix_error
```

Для этого воспользуемся командой **git push**, после которой указываем имя репозитория origin и название ветки fix_error. Изменения отправлены, давайте посмотрим список удаленных веток.

```
$ git branch -r
  origin/HEAD -> origin/master
  origin/fix_error
  origin/master
```

Итак, новая ветка появилась на сервере.

Что будет, если мы переименуем или удалим ветку локально? Давайте переименуем ветку fix_error в to_delete:

```
$ git branch -m fix_error to_delete

$ git branch
  master
* to_delete

$ git branch -r
  origin/HEAD -> origin/master
  origin/fix_error
  origin/master
```

Как видно, ветка была переименована только локально, изменения не коснулись удаленного репозитория. Мы можем попробовать отправить ветку to_delete на удаленный сервер:

```
$ git push origin to_delete
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'to_delete' on GitHub by visiting:
remote:      https://github.com/igorsimdyanov/hello/pull/new/to_delete
remote:
To github.com:igorsimdyanov/hello.git
 * [new branch]     to_delete -> to_delete

$ git branch -r
  origin/HEAD -> origin/master
  origin/fix_error
  origin/master
  origin/to_delete
```

В результате на сервере появится сразу три ветки.

Более того, мы можем даже удалить локальную ветку to_delete:

```
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
```

Давайте переместимся на ветку master. У нас не получится удалить выбранную ветку, нужно это делать с какой-то другой:

```
$ git branch -D to_delete
Deleted branch to_delete (was 9e02299).

$ git branch
* master

$ git branch -r
  origin/HEAD -> origin/master
  origin/fix_error
  origin/master
  origin/to_delete
```

Опять же, изменения затрагивают только локальные ветки, ветки удаленного репозитория остаются незатронутыми. Для того, чтобы управлять удаленными ветками, следует воспользоваться отдельным набором команд. Например, для удаления ветки на сервере, следует воспользоваться командой git push:

```
$ git push origin :to_delete
To github.com:igorsimdyanov/hello.git
 - [deleted]         to_delete
```

Однако имя ветки следует предварить двоеточием.

Запрашиваем список удаленных веток, чтобы убедиться, что ветки to_delete больше не существует:

```
$ git branch -r
  origin/HEAD -> origin/master
  origin/fix_error
  origin/master
```

Если потребуется переименовать ветку, то сделать это одной командой довольно сложно. Проще всего создать на удаленном сервере копию ветки, а старую ветку удалить.

Давайте переименуем ветку fix_error в to_delete. Для этого создаем локально копию удаленной ветки с именем to_delete:

```
$ git branch to_delete origin/fix_error
Branch 'to_delete' set up to track remote branch 'fix_error' from 'origin'.

$ git branch
* master
  to_delete
```

Забрасываем локальную ветку to_delete на удаленный сервер:

```
$ git push origin to_delete
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'to_delete' on GitHub by visiting:
remote:      https://github.com/igorsimdyanov/hello/pull/new/to_delete
remote:
To github.com:igorsimdyanov/hello.git
 * [new branch]     to_delete -> to_delete

$ git branch -r
  origin/HEAD -> origin/master
```

```
  origin/fix_error
  origin/master
  origin/to_delete
```

Теперь старую ветку fix_error можно удалить:

```
$ git push origin :fix_error
To github.com:igorsimdyanov/hello.git
 - [deleted]         fix_error

$ git branch -r
  origin/HEAD -> origin/master
  origin/master
  origin/to_delete
```

Ветка переименована:

```
$ git branch
* master
  to_delete
```

Статус и навигация по git-проекту

Выяснить, на какой ветке мы сейчас находимся, можно при помощи команды **git branch** или **git status**. Как видно на протяжении текущего и предыдущих роликов, мы постоянно вынуждены набирать эту команду, чтобы понимать, на какой из веток сейчас находимся, чтобы случайно не выполнить команду не в той ветке.

Если в качестве командной оболочки используется bash, можно воспользоваться решением git-aware-promt (https://github.com/jimeh/git-aware-prompt), который встраивает название текущей ветки в подсказку командной строки. Обратите внимание, что проект тоже расположен на GitHub, чтобы им воспользоваться, нам придется его клонировать.

Давайте создадим в домашнем каталоге папку .bash, в которой будут располагаться скрипты. Здесь мы для этого воспользуемся консольной командой **mkdir**, однако вы можете создать папку любым удобным для вас способом:

```
$ mkdir .bash
```

Переходим в папку:

```
$ cd .bash
```

Теперь нам необходимо в нее клонировать репозиторий с проектом git-aware-promt. Возвращаемся в git-репозиторий проекта, копируем ссылку для клонирования.

Обратите внимание, что ссылка для клонирования может начинаться с префикса https или git@ — это разные протоколы общения с git-сервером.

Использование git@ потребует наличие на git-сервере открытого ключа. Этот способ удобен при работе с вашими собственными репозиториями. При использовании протокола https:// мы можем читать репозиторий без ограничений. Именно эту ссылку мы и копируем, чтобы клонировать проект.

Однако если мы захотим закинуть свои изменения в чужой проект, нам либо откажут, либо запросят логин и пароль.

Клонируем проект при помощи команды **git clone**:

```
$ git clone https://github.com/jimeh/git-aware-prompt.git
```

Чтобы проект заработал, нам потребуется внести изменения в файл .bash_profile домашнего каталога. Этот файл выполняется всякий раз при инициализации терминала.

```
$ source "~/.bash/git-aware-prompt/main.sh
```

При помощи команды source выполняется скрипт main.sh из проекта git-aware-prompt. Кроме того, модифицируем переменную окружения PS1 таким образом, чтобы в нее добавлялось название текущей ветки:

```
export PS1="\w \[\$txtcyn\]\$git_branch\[\$txtred\]\$git_dirty\[\$txtrst\]\$ "
```

Переменная окружения PS1 в Unix-мире несет ответственность за формирование подсказки командной строки. Открываем новый терминал и переходим внутрь проекта:

```
$ cd test
```

Как видно, подсказка командной строки изменяется, теперь она содержит название текущей ветки:

```
test (master)$
```

Если мы переключимся на ветку to_delete:

```
$ git checkout to_delete
```

В подсказке командной строки появится название ветки to_delete:

```
test (to_delete)$
```

Давайте внесем изменения в файл README.md (добавим строку Hello, world!):

```
test (to_delete) *$
```

Как видно, к подсказке добавляется красная звездочка, которая сообщает о том, что имеются неиндексированные изменения.

Как скрывать изменения

Если в ходе работы над веткой мы внесли какие-то изменения в файл, но вынуждены срочно прерваться, чтобы переключиться в другую ветку, в лучшем случае незафиксированные изменения перейдут в эту новую ветку.

```
$ git checkout master
```

Чтобы этого не происходило, следует либо выполнить команду **git commit**, либо отложить изменения:

```
$ git checkout to_delete
```

Для того, чтобы отложить изменения на потом, нам потребуется добавить их в индекс:

```
$ git add .
$ git status
```

После этого можно выполнить команду **git stash**:

```
$ git stash
```

Обратите внимание, что красная звездочка исчезла, мы очистили текущую ветку от изменений:

```
$ git status
```

Однако они не потеряны, а спрятаны в stash-списке. Давайте их посмотрим при помощи команды **git stash list**:

```
$ git stash list
```

Теперь мы можем переключиться на другую ветку и поработать с ней:

```
$ git checkout master
$ touch hello.txt
$ git add .
$ git commit -m 'New file hello.txt'
```

Чтобы вернуться к ветке to_delete и незавершенным изменениям, переключаемся на нее при помощи команды **checkout**:

```
$ git checkout to_delete
$ git stash list
```

Теперь можно вернуть данные из stash, для этого воспользуемся командой **git stash apply**:

```
$ git stash apply
```

Изменения, которые были спрятаны в stash-списке, снова доступны для работы, команду **git stash** можно выполнять несколько раз подряд в разных ветках.

Каждый раз в stash-список будет добавляться новая запись:

```
$ git add .
$ git stash
$ git stash list
```

Они пронумерованы цифрами 0, 1 и т. д.

По умолчанию git использует последнюю запись, если вы хотите воспользоваться более ранним вариантом, можно явно указать метку stash-записи после команды **git stash apply**:

```
$ git stash apply stash@{1}
```

Впрочем, держать много записей в стеше не стоит, удалить записи можно при помощи команды **git stash drop**:

```
$ git stash drop
$ git stash list
$ git stash drop
$ git stash list
```

Используемые источники

Для подготовки методического пособия мы использовали эти ресурсы:

- Официальная документация Git: **https://git-scm.com/book/ru/v2**.