Лабораторная работа 1.2 Исследование возможностей Git для работы с локальными репозиториями

Цель работы: *исследовать базовые возможности системы контроля версий Git для работы с локальными репозиториями.*

Теоретические сведения

Просмотр истории коммитов

После того, как вы создали несколько коммитов или же клонировали репозиторий с уже существующей историей коммитов, вероятно Вам понадобится возможность посмотреть что было сделано — историю коммитов. Одним из основных и наиболее мощных инструментов для этого является команда `git log`.

Следующие несколько примеров используют очень простой проект «simplegit». Чтобы клонировать проект, используйте команду:

```
$ git clone https://github.com/schacon/simplegit-progit
```

Если вы запустите команду `git log` в каталоге клонированного проекта, вы увидите следующий вывод:

```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    Remove unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    Initial commit
```

По умолчанию (без аргументов) `git log` перечисляет коммиты, сделанные в репозитории в обратном к хронологическому порядке — последние коммиты находятся вверху. Из примера можно увидеть, что данная команда перечисляет коммиты с их SHA-1 контрольными суммами, именем и электронной почтой автора, датой создания и сообщением коммита.

Команда `git log` имеет очень большое количество опций для поиска коммитов по разным критериям. Рассмотрим наиболее популярные из них.

Одним из самых полезных аргументов является `-p` или `--patch`, который показывает разницу (выводит *патч*), внесенную в каждый коммит. Так же вы можете ограничить количество записей в выводе команды; используйте параметр `-2` для вывода только двух записей:

```
$ git log -p -2
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number

diff --git a/Rakefile b/Rakefile
index a874b73..8f94139 100644
--- a/Rakefile
+++ b/Rakefile
@@ -5,7 +5,7 @@ require 'rake/gempackagetask'
 spec = Gem::Specification.new do |s|
     s.platform  =   Gem::Platform::RUBY
     s.name      =   "simplegit"
-    s.version   =   "0.1.0"
+    s.version   =   "0.1.1"
     s.author    =   "Scott Chacon"
     s.email     =   "schacon@gee-mail.com"
     s.summary   =   "A simple gem for using Git in Ruby code."

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    Remove unnecessary test

diff --git a/lib/simplegit.rb b/lib/simplegit.rb
index a0a60ae..47c6340 100644
--- a/lib/simplegit.rb
+++ b/lib/simplegit.rb
@@ -18,8 +18,3 @@ class SimpleGit
     end

 end
-
-if $0 == __FILE__
-  git = SimpleGit.new
-  puts git.show
-end
```

Эта опция отображает аналогичную информацию но содержит разницу для каждой записи. Очень удобно использовать данную опцию для код ревью или для быстрого просмотра серии внесенных изменений. Так же есть возможность использовать серию опций для обобщения. Например, если вы хотите увидеть сокращенную статистику для каждого коммита, вы можете использовать опцию `--stat`:

```
$ git log --stat
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700
```

```
    Change version number

 Rakefile | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    Remove unnecessary test

 lib/simplegit.rb | 5 -----
 1 file changed, 5 deletions(-)

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    Initial commit

 README           |  6 ++++++
 Rakefile         | 23 +++++++++++++++++++++++
 lib/simplegit.rb | 25 +++++++++++++++++++++++++
 3 files changed, 54 insertions(+)
```

Как вы видите, опция `--stat` печатает под каждым из коммитов список и количество измененных файлов, а также сколько строк в каждом из файлов было добавлено и удалено. В конце можно увидеть суммарную таблицу изменений.

Следующей действительно полезной опцией является `--pretty`. Эта опция меняет формат вывода. Существует несколько встроенных вариантов отображения. Опция `oneline` выводит каждый коммит в одну строку, что может быть очень удобным если вы просматриваете большое количество коммитов. К тому же, опции `short`, `full` и `fuller` делают вывод приблизительно в том же формате, но с меньшим или большим количеством информации соответственно:

```
$ git log --pretty=oneline
ca82a6dff817ec66f44342007202690a93763949 Change version number
085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7 Remove unnecessary test
a11bef06a3f659402fe7563abf99ad00de2209e6 Initial commit
```

Наиболее интересной опцией является `format`, которая позволяет указать формат для вывода информации. Особенно это может быть полезным когда вы хотите сгенерировать вывод для автоматического анализа — так как вы указываете формат явно, он не будет изменен даже после обновления Git:

```
$ git log --pretty=format:"%h - %an, %ar : %s"
ca82a6d - Scott Chacon, 6 years ago : Change version number
085bb3b - Scott Chacon, 6 years ago : Remove unnecessary test
a11bef0 - Scott Chacon, 6 years ago : Initial commit
```

Полезные опции для `git log --pretty=format` отображает наиболее полезные опции для изменения формата.

Опция	Описания вывода
`%H`	Хеш коммита
`%h`	Сокращенный хеш коммита
`%T`	Хеш дерева
`%t`	Сокращенный хеш дерева
`%P`	Хеш родителей
`%p`	Сокращенный хеш родителей
`%an`	Имя автора
`%ae`	Электронная почта автора
`%ad`	Дата автора (формат даты можно задать опцией --date=option)
`%ar`	Относительная дата автора
`%cn`	Имя коммитера
`%ce`	Электронная почта коммитера
`%cd`	Дата коммитера
`%cr`	Относительная дата коммитера
`%s`	Содержание

Вам наверное интересно, какая же разница между *автором* и *коммитером*. Автор — это человек, изначально сделавший работу, а коммитер — это человек, который последним применил эту работу. Другими словами, если вы создадите патч для какого-то проекта, а один из основных членов команды этого проекта применит этот патч, вы оба получите статус участника — вы как автор и основной член команды как коммитер.

Опции `oneline` и `format` являются особенно полезными с опцией `--graph` команды `log`. С этой опцией вы сможете увидеть небольшой граф в формате ASCII, который показывает текущую ветку и историю слияний:

```
$ git log --pretty=format:"%h %s" --graph
* 2d3acf9 Ignore errors from SIGCHLD on trap
*  5e3ee11 Merge branch 'master' of git://github.com/dustin/grit
|\
| * 420eac9 Add method for getting the current branch
* | 30e367c Timeout code and tests
* | 5a09431 Add timeout protection to grit
* | e1193f8 Support for heads with slashes in them
|/
* d6016bc Require time for xmlschema
*  11d191e Merge branch 'defunkt' into local
```

Данный вывод будет нам очень интересен в следующей лабораторной работе, где мы рассмотрим ветвление и слияние.

Мы рассмотрели только несколько простых опций для форматирования вывода с помощью команды `git log` — на самом деле их гораздо больше. Наиболее распространенные опции для команды `git log` содержит описание как уже рассмотренных, так и нескольких новых опций, которые могут быть полезными в зависимости от нужного формата вывода.

Опция	Описание
`-p`	Показывает патч для каждого коммита.
`--stat`	Показывает статистику измененных файлов для каждого коммита.
`--shortstat`	Отображает только строку с количеством изменений/вставок/удалений для команды --stat.
`--name-only`	Показывает список измененных файлов после информации о коммите.
`--name-status`	Показывает список файлов, которые добавлены/изменены/удалены.
`--abbrev-commit`	Показывает только несколько символов SHA-1 чек-суммы вместо всех 40.
`--relative-date`	Отображает дату в относительном формате (например, «2 weeks ago») вместо стандартного формата даты.
`--graph`	Отображает ASCII граф с ветвлениями и историей слияний.
`--pretty`	Показывает коммиты в альтернативном формате. Возможные варианты опций: oneline, short, full, fuller и format (с помощью последней можно указать свой формат).
`--oneline`	Сокращение для одновременного использования опций `--pretty=oneline --abbrev-commit`.

Ограничение вывода

В дополнение к опциям форматирования вывода, команда `git log` принимает несколько опций для ограничения вывода — опций, с помощью которых можно увидеть определенное подмножество коммитов. Вы уже видели одну из таких опций — это опция `-2`, которая показывает только последние два коммита. В действительности вы можете использовать `-<n>`, где `n` — это любое натуральное число и представляет собой `n` последних коммитов. На практике вы не будете часто использовать эту опцию, потому что Git по умолчанию использует постраничный вывод и вы будете видеть только одну страницу за раз.

Однако, опции для ограничения вывода по времени, такие как `--since` и `--until`, являются очень удобными. Например, следующая команда покажет список коммитов, сделанных за последние две недели:

```
$ git log --since=2.weeks
```

Это команда работает с большим количеством форматов — вы можете указать определенную дату вида `2008-01-15` или же относительную дату, например `2 years 1 day 3 minutes ago`.

Также вы можете фильтровать список коммитов по заданным параметрам. Опция `--author` дает возможность фильтровать по автору коммита, а опция `--grep` искать по ключевым словам в сообщении коммита.

Следующим действительно полезным фильтром является опция `-S`, которая принимает аргумент в виде строки и показывает только те коммиты, в которых изменение в коде повлекло за собой добавление или удаление этой строки. Например, если вы хотите найти последний коммит, который добавил или удалил вызов определенной функции, вы можете запустить команду:

```
$ git log -S function_name
```

Последней полезной опцией, которую принимает команда `git log` как фильтр, является путь. Если вы укажете каталог или имя файла, вы ограничите вывод только теми коммитами, в которых были изменения этих файлов. Эта опция всегда указывается последней после двойного тире (`--`), чтобы отделить пути от опций:

```
$ git log -- path/to/file
```

В нижеприведенной таблице Вы можете увидеть эти и другие распространенные опции.

Опция	Описание
`-(n)`	Показывает только последние n коммитов.
`--since`, `--after`	Показывает только те коммиты, которые были сделаны после указанной даты.
`--until`, `--before`	Показывает только те коммиты, которые были сделаны до указанной даты.
`--author`	Показывает только те коммиты, в которых запись author совпадает с указанной строкой.
`--committer`	Показывает только те коммиты, в которых запись committer совпадает с указанной строкой.
`--grep`	Показывает только коммиты, сообщение которых содержит указанную строку.
`-S`	Показывает только коммиты, в которых изменение в коде повлекло за собой добавление или удаление указанной строки.

Например, если вы хотите увидеть, в каких коммитах произошли изменения в тестовых файлах в исходном коде Git в октябре 2008 года, автором которых был Junio Hamano и которые не были коммитами слияния, вы можете запустить следующую команду:

```
$ git log --pretty="%h - %s" --author='Junio C Hamano' --since="2008-10-01" \
   --before="2008-11-01" --no-merges -- t/
5610e3b - Fix testcase failure when extended attributes are in use
acd3b9e - Enhance hold_lock_file_for_{update,append}() API
f563754 - demonstrate breakage of detached checkout with symbolic link HEAD
d1a43f2 - reset --hard/read-tree --reset -u: remove unmerged new paths
51a94af - Fix "checkout --track -b newbranch" on detached HEAD
b0ad11e - pull: allow "git pull origin $something:$current_branch" into an
unborn branch
```

Из почти 40 000 коммитов в истории исходного кода Git, эта команда показывает только 6, которые соответствуют этим критериям.

Операции отмены

В любой момент вам может потребоваться что-либо отменить. Здесь мы рассмотрим несколько основных способов отмены сделанных изменений. Будьте осторожны, не все операции отмены в свою очередь можно отменить! Это одна из редких областей Git, где неверными действиями можно необратимо удалить результаты своей работы.

Отмена может потребоваться, если вы сделали коммит слишком рано, например, забыв добавить какие-то файлы или комментарий к коммиту. Если вы хотите переделать коммит — внесите необходимые изменения, добавьте их в индекс и сделайте коммит ещё раз, указав параметр `--amend`:

```
$ git commit --amend
```

Эта команда использует область подготовки (индекс) для внесения правок в коммит. Если вы ничего не меняли с момента последнего коммита (например, команда запущена сразу после предыдущего коммита), то снимок состояния останется в точности таким же, а всё что вы сможете изменить — это ваше сообщение к коммиту.

Запустится тот же редактор, только он уже будет содержать сообщение предыдущего коммита. Вы можете редактировать сообщение как обычно, однако, оно заменит сообщение предыдущего коммита.

Например, если вы сделали коммит и поняли, что забыли проиндексировать изменения в файле, который хотели добавить в коммит, то можно сделать следующее:

```
$ git commit -m 'Initial commit'
$ git add forgotten_file
$ git commit --amend
```

В итоге получится единый коммит — второй коммит заменит результаты первого.

> Очень важно понимать, что когда вы вносите правки в последний коммит, вы не столько исправляете его, сколько *заменяете* новым, который полностью его перезаписывает. В результате всё выглядит так, будто первоначальный коммит никогда не существовал, а так же он больше не появится в истории вашего репозитория.
>
> Очевидно, смысл изменения коммитов в добавлении незначительных правок в последние коммиты и, при этом, в избежании засорения истории сообщениями вида «Ой, забыл добавить файл» или «Исправление грамматической ошибки».

Отмена индексации файла

Следующие два раздела демонстрируют как работать с индексом и изменениями в рабочем каталоге. Радует, что команда, которой вы определяете состояние этих областей, также подсказывает вам как отменять изменения в них. Например, вы изменили два файла и хотите добавить их в разные коммиты, но случайно выполнили команду `git add *` и добавили в индекс оба. Как исключить из индекса один из них? Команда `git status` напомнит вам:

```
$ git add *
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    README.md -> README
    modified:   CONTRIBUTING.md
```

Прямо под текстом «Changes to be committed» говорится: используйте `git reset HEAD <file>…` для исключения из индекса. Давайте последуем этому совету и отменим индексирование файла `CONTRIBUTING.md`:

```
$ git reset HEAD CONTRIBUTING.md
Unstaged changes after reset:
M   CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    README.md -> README

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   CONTRIBUTING.md
```

Команда выглядит несколько странно, но — работает! Файл `CONTRIBUTING.md` изменен, но больше не добавлен в индекс.

> Команда `git reset` *может* быть опасной если вызвать её с параметром `--hard`. В приведенном примере файл не был затронут, следовательно команда относительно безопасна.

Отмена изменений в файле

Что делать, если вы поняли, что не хотите сохранять свои изменения файла `CONTRIBUTING.md`? Как можно просто отменить изменения в нём — вернуть к тому состоянию, которое было в последнем коммите (или к начальному после клонирования, или еще как-то полученному)? Нам повезло, что `git status` подсказывает и это тоже.

В выводе команды из последнего примера список изменений выглядит примерно так:

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   CONTRIBUTING.md
```

Здесь явно сказано как отменить существующие изменения. Давайте так и сделаем:

```
$ git checkout -- CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    README.md -> README
```

Как видите, откат изменений выполнен.

> Важно понимать, что `git checkout -- <file>` — опасная команда. Все локальные изменения в файле пропадут — Git просто заменит его версией из последнего коммита. Ни в коем случае не используйте эту команду, если вы не уверены, что изменения в файле вам не нужны.

Помните, все что попало в *коммит* почти всегда Git может восстановить. Можно восстановить даже коммиты из веток, которые были удалены, или коммиты, перезаписанные параметром `--amend`. Но всё, что не было включено в коммит и потеряно — скорее всего, потеряно навсегда.

Работа с удалёнными репозиториями

Для того, чтобы внести вклад в какой-либо Git-проект, вам необходимо уметь работать с удалёнными репозиториями. Удалённые репозитории представляют собой версии вашего проекта, сохранённые в интернете или ещё где-то в сети. У вас может быть несколько удалённых репозиториев, каждый из которых может быть доступен для чтения или для чтения-записи. Взаимодействие с другими пользователями предполагает управление удалёнными репозиториями, а также отправку и получение данных из них. Управление репозиториями включает в себя как умение добавлять новые, так и умение удалять устаревшие репозитории, а также умение управлять различными удалёнными ветками, объявлять их отслеживаемыми или нет и так далее. В данном разделе мы рассмотрим некоторые из этих навыков.

Просмотр удалённых репозиториев

Для того, чтобы просмотреть список настроенных удалённых репозиториев, вы можете запустить команду `git remote`. Она выведет названия доступных удалённых репозиториев. Если вы клонировали репозиторий, то увидите как минимум `origin` — имя по умолчанию, которое Git даёт серверу, с которого производилось клонирование:

```
$ git clone https://github.com/schacon/ticgit
Cloning into 'ticgit'...
remote: Reusing existing pack: 1857, done.
remote: Total 1857 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (1857/1857), 374.35 KiB | 268.00 KiB/s, done.
Resolving deltas: 100% (772/772), done.
Checking connectivity... done.
$ cd ticgit
$ git remote
origin
```

Вы можете также указать ключ `-v`, чтобы просмотреть адреса для чтения и записи, привязанные к репозиторию:

```
$ git remote -v
origin  https://github.com/schacon/ticgit (fetch)
origin  https://github.com/schacon/ticgit (push)
```

Если у вас больше одного удалённого репозитория, команда выведет их все. Например, для репозитория с несколькими настроенными удалёнными репозиториями в случае совместной работы нескольких пользователей, вывод команды может выглядеть примерно так:

```
$ cd grit
$ git remote -v
bakkdoor  https://github.com/bakkdoor/grit (fetch)
bakkdoor  https://github.com/bakkdoor/grit (push)
cho45     https://github.com/cho45/grit (fetch)
cho45     https://github.com/cho45/grit (push)
defunkt   https://github.com/defunkt/grit (fetch)
defunkt   https://github.com/defunkt/grit (push)
koke      git://github.com/koke/grit.git (fetch)
koke      git://github.com/koke/grit.git (push)
origin    git@github.com:mojombo/grit.git (fetch)
origin    git@github.com:mojombo/grit.git (push)
```

Это означает, что мы можем легко получить изменения от любого из этих пользователей. Возможно, что некоторые из репозиториев доступны для записи и в них можно отправлять свои изменения, хотя вывод команды не даёт никакой информации о правах доступа.

Добавление удалённых репозиториев

В предыдущих разделах мы уже упоминали и приводили примеры добавления удалённых репозиториев, сейчас рассмотрим эту операцию подробнее. Для того, чтобы добавить удалённый репозиторий и присвоить ему имя (shortname), просто выполните команду `git remote add <shortname> <url>`:

```
$ git remote
origin
$ git remote add pb https://github.com/paulboone/ticgit
$ git remote -v
origin  https://github.com/schacon/ticgit (fetch)
origin  https://github.com/schacon/ticgit (push)
pb  https://github.com/paulboone/ticgit (fetch)
pb  https://github.com/paulboone/ticgit (push)
```

Теперь вместо указания полного пути вы можете использовать `pb`. Например, если вы хотите получить изменения, которые есть у Пола, но нету у вас, вы можете выполнить команду `git fetch pb`:

```
$ git fetch pb
remote: Counting objects: 43, done.
remote: Compressing objects: 100% (36/36), done.
remote: Total 43 (delta 10), reused 31 (delta 5)
Unpacking objects: 100% (43/43), done.
From https://github.com/paulboone/ticgit
 * [new branch]      master     -> pb/master
 * [new branch]      ticgit     -> pb/ticgit
```

Ветка `master` из репозитория Пола сейчас доступна вам под именем `pb/master`.

Получение изменений из удалённого репозитория — Fetch и Pull

Как вы только что узнали, для получения данных из удалённых проектов, следует выполнить:

```
$ git fetch [remote-name]
```

Данная команда связывается с указанным удалённым проектом и забирает все те данные проекта, которых у вас ещё нет. После того как вы выполнили команду, у вас должны появиться ссылки на все ветки из этого удалённого проекта, которые вы можете просмотреть или слить в любой момент.

Когда вы клонируете репозиторий, команда `clone` автоматически добавляет этот удалённый репозиторий под именем «origin». Таким образом, `git fetch origin` извлекает все наработки, отправленные на этот сервер после того, как вы его клонировали (или получили изменения с помощью fetch). Важно отметить, что команда `git fetch` забирает данные в ваш локальный репозиторий, но не сливает их с какими-либо вашими наработками и не модифицирует то, над чем вы работаете в данный момент. Вам необходимо вручную слить эти данные с вашими, когда вы будете готовы.

Если ветка настроена на отслеживание удалённой ветки (см. следующий раздел и главу Ветвление в Git чтобы получить больше информации), то вы можете использовать команду `git pull` чтобы автоматически получить изменения из удалённой ветки и слить их со своей текущей. Этот способ может для вас оказаться более простым или более удобным. К тому же, по умолчанию команда `git clone` автоматически настраивает вашу локальную ветку `master` на отслеживание удалённой ветки `master` на сервере, с которого вы клонировали репозиторий. Название веток может быть другим и зависит от ветки по умолчанию на сервере. Выполнение `git pull`, как правило, извлекает (fetch) данные с сервера, с которого вы изначально клонировали, и автоматически пытается слить (merge) их с кодом, над которым вы в данный момент работаете.

Отправка изменений в удаленный репозиторий (Push)

Когда вы хотите поделиться своими наработками, вам необходимо отправить их в удалённый репозиторий. Команда для этого действия простая: `git push <remote-name> <branch-name>`. Чтобы отправить вашу ветку master на сервер `origin` (повторимся, что клонирование обычно настраивает оба этих имени автоматически), вы можете выполнить следующую команду для отправки ваших коммитов:

```
$ git push origin master
```

Эта команда срабатывает только в случае, если вы клонировали с сервера, на котором у вас есть права на запись, и если никто другой с тех пор не выполнял команду `push`. Если вы и кто-то ещё одновременно клонируете, затем он выполняет команду `push`, а после него выполнить команду `push` попытаетесь вы, то ваш `push` точно будет отклонён. Вам придётся сначала получить изменения и объединить их с вашими и только после этого вам будет позволено выполнить `push`. Обратитесь к главе Ветвление в Git для более подробного описания, как отправлять изменения на удалённый сервер.

Просмотр удаленного репозитория

Если хотите получить побольше информации об одном из удалённых репозиториев, вы можете использовать команду `git remote show <remote>`. Выполнив эту команду с некоторым именем, например, `origin`, вы получите следующий результат:

```
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/schacon/ticgit
  Push  URL: https://github.com/schacon/ticgit
  HEAD branch: master
  Remote branches:
    master                               tracked
    dev-branch                           tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

Она выдаёт URL удалённого репозитория, а также информацию об отслеживаемых ветках. Эта команда любезно сообщает вам, что если вы, находясь на ветке master, выполните `git pull`, ветка master с удалённого сервера будет автоматически влита в вашу сразу после получения всех необходимых данных. Она также выдаёт список всех полученных ею ссылок.

Это был пример для простой ситуации и вы наверняка встречались с чем-то подобным. Однако, если вы используете Git более интенсивно, вы можете увидеть гораздо большее количество информации от `git remote show`:

```
$ git remote show origin
* remote origin
  URL: https://github.com/my-org/complex-project
  Fetch URL: https://github.com/my-org/complex-project
  Push  URL: https://github.com/my-org/complex-project
  HEAD branch: master
  Remote branches:
    master                           tracked
    dev-branch                       tracked
    markdown-strip                   tracked
    issue-43                         new (next fetch will store in
remotes/origin)
    issue-45                         new (next fetch will store in
remotes/origin)
    refs/remotes/origin/issue-11     stale (use 'git remote prune' to remove)
  Local branches configured for 'git pull':
    dev-branch merges with remote dev-branch
    master     merges with remote master
  Local refs configured for 'git push':
    dev-branch                     pushes to dev-branch                     (up
to date)
    markdown-strip                 pushes to markdown-strip                 (up
to date)
    master                         pushes to master                         (up
to date)
```

Данная команда показывает какая именно локальная ветка будет отправлена на удалённый сервер по умолчанию при выполнении `git push`. Она также показывает, каких веток с удалённого сервера у вас ещё нет, какие ветки всё ещё есть у вас, но уже удалены на сервере, и для нескольких веток показано, какие удалённые ветки будут в них влиты при выполнении `git pull`.

Удаление и переименование удалённых репозиториев

Для переименования удалённого репозитория можно выполнить `git remote rename`. Например, если вы хотите переименовать `pb` в `paul`, вы можете это сделать при помощи `git remote rename`:

```
$ git remote rename pb paul
$ git remote
origin
paul
```

Стоит упомянуть, что это также изменит имена удалённых веток в вашем репозитории. То, к чему вы обращались как `pb/master`, теперь стало `paul/master`.

Если по какой-то причине вы хотите удалить удаленный репозиторий — вы сменили сервер или больше не используете определённое зеркало, или кто-то перестал вносить изменения — вы можете использовать `git remote rm`:

```
$ git remote remove paul
$ git remote
origin
```

При удалении ссылки на удалённый репозиторий все отслеживаемые ветки и настройки, связанные с этим репозиторием, так же будут удалены.

Работа с тегами

Как и большинство СКВ, Git имеет возможность помечать определённые моменты в истории как важные. Как правило, эта функциональность используется для отметки моментов выпуска версий (v1.0, и т. п.). Такие пометки в Git называются тегами. В этом разделе вы узнаете, как посмотреть имеющиеся теги, как создать новые или удалить существующие, а также какие типы тегов существуют в Git.

Просмотр списка тегов

Просмотреть список имеющихся тегов в Git можно очень просто. Достаточно набрать команду `git tag` (параметры `-l` и `--list` опциональны):

```
$ git tag
v1.0
v2.0
```

Данная команда перечисляет теги в алфавитном порядке; порядок их отображения не имеет существенного значения.

Так же можно выполнить поиск тега по шаблону. Например, репозиторий Git содержит более 500 тегов. Если вы хотите посмотреть теги выпусков 1.8.5, то выполните следующую команду:

```
$ git tag -l "v1.8.5*"
v1.8.5
v1.8.5-rc0
v1.8.5-rc1
v1.8.5-rc2
v1.8.5-rc3
v1.8.5.1
v1.8.5.2
v1.8.5.3
v1.8.5.4
v1.8.5.5
```

Создание тегов

Git использует два основных типа тегов: легковесные и аннотированные.

Легковесный тег — это что-то очень похожее на ветку, которая не изменяется — просто указатель на определённый коммит.

А вот аннотированные теги хранятся в базе данных Git как полноценные объекты. Они имеют контрольную сумму, содержат имя автора, его e-mail и дату создания, имеют комментарий и могут быть подписаны и проверены с помощью GNU Privacy Guard (GPG). Обычно рекомендуется создавать аннотированные теги, чтобы иметь всю перечисленную информацию; но если вы хотите сделать временную метку или по какой-то причине не хотите сохранять остальную информацию, то для этого годятся и легковесные.

Аннотированные теги

Создание аннотированного тега в Git выполняется легко. Самый простой способ — это указать `-a` при выполнении команды `tag`:

```
$ git tag -a v1.4 -m "my version 1.4"
$ git tag
v0.1
v1.3
v1.4
```

Опция `-m` задаёт сообщение, которое будет храниться вместе с тегом. Если не указать сообщение, то Git запустит редактор, чтобы вы смогли его ввести.

С помощью команды `git show` вы можете посмотреть данные тега вместе с коммитом:

```
$ git show v1.4
tag v1.4
Tagger: Ben Straub <ben@straub.cc>
Date:   Sat May 3 20:19:12 2014 -0700

my version 1.4

commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number
```

Здесь приведена информация об авторе тега, дате его создания и аннотирующее сообщение перед информацией о коммите.

Обмен тегами

По умолчанию, команда `git push` не отправляет теги на удалённые сервера. После создания теги нужно отправлять явно на удалённый сервер. Процесс аналогичен отправке веток — достаточно выполнить команду `git push origin <tagname>`.

```
$ git push origin v1.5
Counting objects: 14, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (12/12), done.
Writing objects: 100% (14/14), 2.05 KiB | 0 bytes/s, done.
Total 14 (delta 3), reused 0 (delta 0)
To git@github.com:schacon/simplegit.git
 * [new tag]         v1.5 -> v1.5
```

Если у вас много тегов, и вам хотелось бы отправить все за один раз, то можно использовать опцию `--tags` для команды `git push`. В таком случае все ваши теги отправятся на удалённый сервер (если только их уже там нет).

```
$ git push origin --tags
Counting objects: 1, done.
Writing objects: 100% (1/1), 160 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To git@github.com:schacon/simplegit.git
 * [new tag]         v1.4 -> v1.4
 * [new tag]         v1.4-lw -> v1.4-lw
```

Теперь, если кто-то клонирует (clone) или выполнит `git pull` из вашего репозитория, то он получит вдобавок к остальному и ваши метки.

Удаление тегов

Для удаления тега в локальном репозитории достаточно выполнить команду `git tag -d <tagname>`. Например, удалить созданный ранее легковесный тег можно следующим образом:

```
$ git tag -d v1.4-lw
Deleted tag 'v1.4-lw' (was e7d5add)
```

Обратите внимание, что при удалении тега не происходит его удаления с внешних серверов. Существует два способа изъятия тега из внешнего репозитория.

Первый способ — это выполнить команду `git push <remote> :refs/tags/<tagname>`:

```
$ git push origin :refs/tags/v1.4-lw
To /git@github.com:schacon/simplegit.git
 - [deleted]         v1.4-lw
```

Это следует понимать как обновление внешнего тега пустым значением, что приводит к его удалению.

Второй способ убрать тег из внешнего репозитория более интуитивный:

```
$ git push origin --delete <tagname>
```

Переход на тег

Если вы хотите получить версии файлов, на которые указывает тег, то вы можете сделать `git checkout` для тега. Однако, это переведёт репозиторий в состояние «detached HEAD», которое имеет ряд неприятных побочных эффектов.

```
$ git checkout v2.0.0
Note: switching to 'v2.0.0'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 99ada87... Merge pull request #89 from schacon/appendix-final

$ git checkout v2.0-beta-0.1
Previous HEAD position was 99ada87... Merge pull request #89 from
schacon/appendix-final
HEAD is now at df3f601... Add atlas.json and cover image
```

Если в состоянии «detached HEAD» внести изменения и сделать коммит, то тег не изменится, при этом новый коммит не будет относиться ни к какой из веток, а доступ к нему можно будет получить только по его хешу. Поэтому, если вам нужно внести изменения — исправить ошибку в одной из старых версий — скорее всего вам следует создать ветку:

```
$ git checkout -b version2 v2.0.0
Switched to a new branch 'version2'
```

Если сделать коммит в ветке `version2`, то она сдвинется вперед и будет отличаться от тега `v2.0.0`, так что будьте с этим осторожны.

Аппаратура и материалы

1. Компьютерный класс общего назначения с конфигурацией ПК не хуже рекомендованной для ОС Windows 10 с подключением к глобальной сети Интернет.
2. Операционная система Windows 10.
3. Система контроля версий Git.
4. Браузер для доступа к web-сервису GitHub, рекомендован к использованию Google Chrome.

Указания по технике безопасности

При работе на ЭВМ без разрешения руководителя занятия запрещается:

- подавать (снимать) напряжение на ПЭВМ и электрические розетки с распределительного щита;
- включать и выключать блоки питания ПЭВМ и мониторы;
- извлекать ПЭВМ из защитного кожуха;
- устранять неисправности, возникшие в ходе выполнения лабораторной работы.

Методика и порядок выполнения работы

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и выбранный Вами язык программирования (выбор языка программирования будет доступен после установки флажка **Add .gitignore**).
3. Проработайте примеры лабораторной работы. Отразите вывод на консоли при выполнении команд git в отчете для лабораторной работе.
4. Выполните клонирование созданного репозитория на рабочий компьютер.
5. Дополните файл `.gitignore` необходимыми правилами для выбранного языка программирования и интегрированной среды разработки.
6. Добавьте в файл README.md информацию о дисциплине, группе и ФИО студента, выполняющего лабораторную работу.
7. Напишите небольшую программу на выбранном Вами языке программирования. Фиксируйте изменения при написании программы в локальном репозитории. Должно быть сделано не менее 7 коммитов, отмеченных не менее 3 тэгами.
8. Просмотреть историю (журнал) хранилища командой `git log`. Например, с помощью команды `git log --graph --pretty=oneline --abbrev-commit`. Добавить скриншот консоли с выводом в отчет по лабораторной работе.
9. Просмотреть содержимое коммитов командой `git show <ref>`, где `<ref>`:
   - `HEAD`: последний коммит;
   - `HEAD~1`: предпоследний коммит (и т. д.);
   - `b34a0e`: коммит с указанным хэшем.

   Отобразите результаты работы этих команд в отчете.
10. Освойте возможность отката к заданной версии.

    10.1. Удалите весь код из одного из файлов программы репозитория, например main.cpp, и сохраните этот файл.

    10.2. Удалите все несохраненные изменения в файле командой: `git checkout -- <имя_файла>`, например `git checkout -- main.cpp`.

    10.3. Повторите пункт 10.1 и сделайте коммит.

    10.4. Откатить состояние хранилища к предыдущей версии командой: `git reset --hard HEAD~1`.

    Сделайте выводы об изменении содержимого выбранного Вами файла программы после выполнения пунктов 10.1–10.4. Отразите эти выводы в отчете.
11. Зафиксируйте сделанные изменения.
12. Добавьте отчет по лабораторной работе в *формате PDF* в папку *doc* репозитория. Зафиксируйте изменения.
13. Отправьте изменения из локального репозитория в удаленный репозиторий GitHub.

14. Проконтролируйте изменения, произошедшие в репозитории GitHub.
15. Самостоятельно изутите работу с сервисами GitLab или BitBucket. Создайте репозиторий на одном из этих сервисов. Создайте зеркало Вашего репозитория на GitHub с использованием созданного Вами репозитория.

    > Для создания зеркала необходимо вначале выполнить команду получения всех изменений с репозитория GitHub
    >
    > ```
    > $ git fetch --prune
    > ```
    >
    > после чего выполнить отправку содержимого локального репозитория в репозиторий GitLab или BitBucket с помощью команды:
    >
    > ```
    > $ git push --prune git@example.com:/new-location.git
    > +refs/remotes/origin/*:refs/heads/* +refs/tags/*:refs/tags/*
    > ```
    >
    > соответственно, вместо `git@example.com:/new-location.git` необходимо подставить адрес репозитория, где будет создано зеркало.

16. Отправьте адрес репозитория GitHub и адрес репозитория зеркала на электронный адрес преподавателя: *rvoronkin@ncfu.ru*.

Содержание отчета и его форма

Отчет по лабораторной работе оформляется электронно в формате PDF, должен содержать ответы на контрольные вопросы, скриншоты терминала с командами Git, скриншоты окна браузера после соответствующих изменений репозитория.

Вопросы для защиты работы

1. Как выполнить историю коммитов в Git? Какие существуют дополнительные опции для просмотра истории киммитов?
2. Как ограничить вывод при просмотре истории коммитов?
3. Как внести изменения в уже сделанный коммит?
4. Как отменить индексацию файла в Git?
5. Как отменить изменения в файле?
6. Что такое удаленный репозиторий Git?
7. Как выполнить просмотр удаленных репозиториев данного локального репозитория?
8. Как добавить удаленный репозиторий для данного локального репозитория?
9. Как выполнить отправку/получение изменений с удаленного репозитория?
10. Как выполнить просмотр удаленного репозитория?
11. Каково назначение тэгов Git?
12. Как осуществляется работа с тэгами Git?
13. Самостоятельно изучите назначение флага `--prune` в командах `git fetch` и `git push`. Каково назначение этого флага?