Практические занятия по работе с распределённой системой контроля версий.

Git.

По материалам сайта https://githowto.com/ru

1. Подготовка

Цели

- Полная готовность к работе с Git.

01 Установка имени и электронной почты

Если вы никогда ранее не использовали git, для начала вам необходимо осуществить установку. Выполните следующие команды, чтобы git узнал ваше имя и электронную почту. Если git уже установлен, можете переходить к разделу окончания строк.

Выполнить:

```
git config --global user.name "Your Name"
git config --global user.email "your_email@whatever.com"
```

02 Параметры установки окончаний строк

Также, для пользователей Unix/Mac:

Выполнить:

```
git config --global core.autocrlf input
git config --global core.safecrlf true
```

Для пользователей Windows:

Выполнить:

```
git config --global core.autocrlf true
git config --global core.safecrlf true
```

2. Финальные приготовления

Цели

- Установить материалы учебника и подготовить их к работе.

01 Скачайте учебные материалы

Скачайте учебные материалы здесь:

- http://githowto.com/git_tutorial.zip

02 Распакуйте учебные материалы

Пакет учебных материалов должен иметь главную папку «git_tutorial» с двумя подпапками:

- work — пустой рабочий каталог. Здесь будут лежать ваши репозитории.
- files — заранее упакованные файлы для того, чтобы вы могли продолжить работать с учебными материалами на любом этапе. Если вы застрянете, просто скопируйте нужный урок в свою рабочую папку.

3. Создание проекта

Цели

- Научиться создавать git репозиторий с нуля.

01 Создайте страницу «Hello, World»

Начните работу в пустом рабочем каталоге (например Work, если вы скачали архив с предыдущего шага) с создания пустого каталога с именем «hello», затем войдите внего и создайте там файл с именем `hello.html` с таким содержанием.

Выполните:

```
mkdir hello
cd hello
touch hello.html
```

Файл: ***hello.html***

```
Hello, World
```

02 Создайте репозиторий

Теперь у вас есть каталог с одним файлом. Чтобы создать git репозиторий из этого каталога, выполните команду git init.

Выполните:

```
git init
```

Результат:

```
$ git init
Initialized empty Git repository in
/Users/alex/Documents/Presentations/githowto/auto/hello/.git/
```

03 Добавьте страницу в репозиторий

Теперь давайте добавим в репозиторий страницу «Hello, World».

Выполните:

```
git add hello.html
git commit -m "First Commit"
```

Вы увидите …

Результат:

```
$ git add hello.html
$ git commit -m "First Commit"
[master (root-commit) 911e8c9] First Commit
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 hello.html
```

4. Проверка состояния

Цели

- Научиться проверять состояние репозитория

01 Проверьте состояние репозитория

Используйте команду `git status`, чтобы проверить текущее состояние репозитория.

Выполните:

```
git status
```

Вы увидите

Результат:

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

Команда проверки состояния сообщит, что коммитить нечего. Это означает, что в репозитории хранится текущее состояние рабочего каталога, и нет никаких изменений, ожидающих записи.

Мы будем использовать команду `git status`, чтобы продолжать отслеживать состояние репозитория и рабочего каталога.

5. Внесение изменений

Цели

- Научиться отслеживать состояние рабочего каталога

01 Измените страницу «Hello, World»

Добавим кое-какие HTML-теги к нашему приветствию. Измените содержимое файла на:

Файл: *hello.html*

```
<h1>Hello, World!</h1>
```

02 Проверьте состояние

Теперь проверьте состояние рабочего каталога.

Выполните:

```
git status
```

Вы увидите …

Результат:

```
$ git status
# On branch master
# 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:   hello.html
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Первое, что нужно заметить, это то, что git знает, что файл `hello.html` был изменен, но при этом эти изменения еще не зафиксированы в репозитории.

Также обратите внимание на то, что сообщение о состоянии дает вам подсказку о том, что нужно делать дальше. Если вы хотите добавить эти изменения в репозиторий, используйте команду `git add`. В противном случае используйте команду `git checkout` для отмены изменений.

03 А далее...

Давайте проиндексируем изменения.

6. Индексация изменений

Цели

- Научиться индексировать изменения для последующих коммитов

01 Добавьте изменения

Теперь дайте команду git проиндексировать изменения. Проверьте состояние

Выполните:

```
git add hello.html
git status
```

Вы увидите…

Результат:

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

Изменения файла `hello.html` были проиндексированы. Это означает, что git теперь знает об изменении, но изменение пока не *перманентно* (читай, *навсегда*) записано в репозиторий. Следующий коммит будет включать в себя проиндексированные изменения.

Если вы решили, что *не* хотите коммитить изменения, команда состояния напомнит вам о том, что с помощью команды `git reset` можно снять индексацию этих изменений.

7. Индексация и коммит

Отдельный шаг индексации в git позволяет вам продолжать вносить изменения в рабочий каталог, а затем, в момент, когда вы захотите взаимодействовать с версионным контролем, git позволит записать изменения в малых коммитах, которые фиксируют то, что вы сделали.

Предположим, что вы отредактировали три файла (`a.html`, `b.html`, and `c.html`). Теперь вы хотите закоммитить все изменения, при этом чтобы изменения в `a.html` и `b.html` были одним коммитом, в то время как изменения в `c.html` логически не связаны с первыми двумя файлами и должны идти отдельным коммитом.

В теории, вы можете сделать следующее:

```
git add a.html
git add b.html
git commit -m "Changes for a and b"
git add c.html
git commit -m "Unrelated change to c"
```

Разделяя индексацию и коммит, вы имеете возможность с легкостью настроить, что идет в какой коммит.

8. Коммит изменений

Цели

- Научиться коммитить изменения в репозиторий

01 Закоммитьте изменения

Достаточно об индексации. Давайте сделаем коммит того, что мы проиндексировали, в репозиторий.

Когда вы ранее использовали `git commit` для коммита первоначальной версии файла `hello.html` в репозиторий, вы включили метку `-m`, которая делает комментарий в командной строке. Команда commit позволит вам интерактивно редактировать комментарии для коммита. Теперь давайте это проверим.

Если вы опустите метку `-m` из командной строки, git перенесет вас в редактор по вашему выбору. Редактор выбирается из следующего списка (в порядке приоритета):

- переменная среды GIT_EDITOR
- параметр конфигурации core.editor
- переменная среды VISUAL
- переменная среды EDITOR

У меня переменная EDITOR установлена в `emacsclient` (доступен для Linux и Mac).

Сделайте коммит сейчас и проверьте состояние.

Выполните:

```
git commit
```

Вы увидите в вашем редакторе:

Результат:

```
|
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.html
#
```

В первой строке введите комментарий: «Added h1 tag». Сохраните файл и выйдите из редактора (для этого в редакторе по-умолчанию (Vim) вам нужно нажать клавишу ESC, ввести `:wq` и нажать Enter). Вы увидите…

Результат:

```
git commit
Waiting for Emacs...
[master 569aa96] Added h1 tag
 1 files changed, 1 insertions(+), 1 deletions(-)
```

Строка «Waiting for Emacs…» получена из программы `emacsclient`, которая посылает файл в запущенную программу emacs и ждет его закрытия. Остальные выходные данные – стандартные коммит-сообщения.

02 Проверьте состояние

В конце давайте еще раз проверим состояние.

Выполните:

```
git status
```

Вы увидите…

Результат:

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

Рабочий каталог чистый, можете продолжить работу.

9. Изменения, а не файлы

Цели

- Понять, что git работает с изменениями, а не файлами.

Большинство систем версионного контроля работают с файлами. Вы добавляете файл в версионный контроль, а система отслеживает изменения файла с этого момента.

Git фокусируется на изменениях в файле, а не самом файле. Когда вы осуществляете команду `git add file`, вы не говорите git добавить файл в репозиторий. Скорее вы говорите, что git надо отметить текущее состояние файла, коммит которого будет произведен позже.

Мы попытаемся исследовать эту разницу в данном уроке.

01 Первое изменение: Добавьте стандартные теги страницы

Измените страницу «Hello, World», чтобы она содержала стандартные теги `<html>` и `<body>`.

Файл: ***hello.html***

```
<html>
  <body>
    <h1>Hello, World!</h1>
  </body>
```

```
</html>
```

02 Добавьте это изменение

Теперь добавьте это изменение в индекс git.

Выполните:

```
git add hello.html
```

03 Второе изменение: Добавьте заголовки HTML

Теперь добавьте заголовки HTML (секцию `<head>`) к странице «Hello, World».

Файл: *hello.html*

```
<html>
  <head>
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

04 Проверьте текущий статус

Выполните:

```
git status
```

Вы увидите…

Результат:

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.html
#
# 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:   hello.html
#
```

Обратите внимание на то, что `hello.html` указан дважды в состоянии. Первое изменение (добавление стандартных тегов) проиндексировано и готово к коммиту. Второе изменение (добавление заголовков HTML) является непроиндексированным. Если бы вы делали коммит сейчас, заголовки не были бы сохранены в репозиторий.

Давайте проверим.

05 Коммит

Произведите коммит проиндексированного изменения (значение по умолчанию), а затем еще раз проверьте состояние.

Выполните:

```
git commit -m "Added standard HTML page tags"
git status
```

Вы увидите…

Результат:

```
$ git commit -m "Added standard HTML page tags"
[master 8c32287] Added standard HTML page tags
 1 files changed, 3 insertions(+), 1 deletions(-)
$ git status
# On branch master
# 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:   hello.html
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Состояние команды говорит о том, что `hello.html` имеет незафиксированные изменения, но уже не в буферной зоне.

06 Добавьте второе изменение

Теперь добавьте второе изменение в индекс, а затем проверьте состояние с помощью команды `git status`.

Выполните:

```
git add .
git status
```

Примечание: В качестве файла для добавления, мы использовали текущий каталог («.»). Это самый краткий и удобный путь для добавления всех изменений в файлы текущего каталога и его подкаталоги. Но поскольку он добавляет все, *не лишним* будет проверить состояние перед запуском `add`, просто чтобы убедиться, что вы не добавили какой-то файл, который добавлять было не нужно.

Я хотел показать вам трюк с `add`, далее мы будем на всякий случай продолжать добавлять явные файлы.

Вы увидите…

Результат:

```
$ git status
```

```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.html
#
```

Второе изменение было проиндексировано и готово к коммиту.

07 Сделайте коммит второго изменения

Выполните:

```
git commit -m "Added HTML header"
```

10. История

Цели

- Научиться просматривать историю проекта.

Получение списка произведенных изменений — функция команды `git log`.

Выполните:

```
git log
```

Вы увидите…

Результат:

```
$ git log
commit fa3c1411aa09441695a9e645d4371e8d749da1dc
Author: Alexander Shvets <alex@githowto.com>
Date:   Wed Mar 9 10:27:54 2011 -0500

    Added HTML header

commit 8c3228730ed03116815a5cc682e8105e7d981928
Author: Alexander Shvets <alex@githowto.com>
Date:   Wed Mar 9 10:27:54 2011 -0500

    Added standard HTML page tags

commit 43628f779cb333dd30d78186499f93638107f70b
Author: Alexander Shvets <alex@githowto.com>
Date:   Wed Mar 9 10:27:54 2011 -0500

    Added h1 tag

commit 911e8c91caeab8d30ad16d56746cbd6eef72dc4c
Author: Alexander Shvets <alex@githowto.com>
Date:   Wed Mar 9 10:27:54 2011 -0500

    First Commit
```

Вот список всех четырех коммитов в репозиторий, которые мы успели совершить.

01 Однострочная история

Вы полностью контролируете то, что отображает `log`. Мне, например, нравится однострочный формат:

Выполните:

```
git log --pretty=oneline
```

Вы увидите…

Результат:

```
$ git log --pretty=oneline
fa3c1411aa09441695a9e645d4371e8d749da1dc Added HTML header
8c3228730ed03116815a5cc682e8105e7d981928 Added standard HTML page tags
43628f779cb333dd30d78186499f93638107f70b Added h1 tag
911e8c91caeab8d30ad16d56746cbd6eef72dc4c First Commit
```

02 Контроль отображения записей

Есть много вариантов выбора, какие элементы отображаются в логе. Поиграйте со следующими параметрами:

```
git log --pretty=oneline --max-count=2
git log --pretty=oneline --since='5 minutes ago'
git log --pretty=oneline --until='5 minutes ago'
git log --pretty=oneline --author=<your name>
git log --pretty=oneline --all
```

В unix-системах доступна справочная страница `man git log`.

03 Изощряемся

Вот что я использую для просмотра изменений, сделанных за последнюю неделю. Я добавлю `--author=alex`, если я хочу увидеть только изменения, которые сделала я.

```
git log --all --pretty=format:"%h %cd %s (%an)" --since='7 days ago'
```

04 Конечный формат лога

Со временем, я решила, что для большей части моей работы мне подходит следующий формат лога.

Выполните:

```
git log --pretty=format:"%h %ad | %s%d [%an]" --graph --date=short
```

Выглядит это примерно так:

Результат:

```
$ git log --pretty=format:"%h %ad | %s%d [%an]" --graph --date=short
* fa3c141 2011-03-09 | Added HTML header (HEAD, master) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags [Alexander Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

Давайте рассмотрим его в деталях:

- `--pretty="..."` — определяет формат вывода.
- `%h` — укороченный хэш коммита
- `%d` — дополнения коммита («головы» веток или теги)
- `%ad` — дата коммита
- `%s` — комментарий
- `%an` — имя автора
- `--graph` — отображает дерево коммитов в виде ASCII-графика
- `--date=short` — сохраняет формат даты коротким и симпатичным

Таким образом, каждый раз, когда вы захотите посмотреть лог, вам придется много печатать. К счастью, мы узнаем о git алиасах в следующем уроке.

05 Другие инструменты

Оба `gitx` (для Mac) и `gitk` (для любой платформы) полезны в изучении истории изменений.

11. Алиасы

Цели

- Научиться настраивать алиасы и шорткаты для команд git

01 Общие алиасы

Для пользователей Windows:

Выполнить:

```
git config --global alias.co checkout
git config --global alias.ci commit
git config --global alias.st status
git config --global alias.br branch
git config --global alias.hist "log --pretty=format:'%h %ad | %s%d [%an]' --
graph --date=short"
git config --global alias.type 'cat-file -t'
git config --global alias.dump 'cat-file -p'
```

Также, для пользователей Unix/Mac:

git status, git add, git commit, git checkout — общие команды, для которых полезно иметь сокращения.

Добавьте следующее в файл .gitconfig в вашем $HOME каталоге.

Файл: *.gitconfig*

```
[alias]
  co = checkout
  ci = commit
  st = status
  br = branch
  hist = log --pretty=format:\"%h %ad | %s%d [%an]\" --graph --date=short
  type = cat-file -t
  dump = cat-file -p
```

Мы уже успели рассмотреть команды `commit` и `status`, в предыдущем уроке рассмотрели команду `log` и совсем скоро познакомимся с `checkout`. Главное, что стоит запомнить из этого урока, так это то, что теперь вы можете вводить `git st` там, где раньше приходилось использовать `git status`. Аналогичным образом, пишем `git co` вместо `git checkout` и `git ci` вместо `git commit`. Что лучше всего, команда `git hist` позволит избежать ввода очень длинной команды `log`.

Попробуйте использовать новые команды.

02 Задайте алиас `hist` в файле `.gitconfig`

По большей части, я буду продолжать печатать полные команды в этом руководстве. Единственным исключением будет использование алиаса `hist`, указанного выше, когда мне понадобится посмотреть git лог. Если вы хотите повторять мои действия, убедитесь, что алиас `hist` установлен в вашем файле `.gitconfig`.

03 `Type` и `Dump`

Мы добавили несколько алиасов для команд, которых мы еще не рассматривали. С командой `git branch` разберемся чуть позже, а команда `git cat-file` используется для исследования git, в чем мы вскоре убедимся.

04 Алиасы команд (опционально)

Если ваша оболочка поддерживает алиасы или шорткаты, вы можете добавить алиасы и на этом уровне. Я использую:

Файл: *.profile*

```
alias gs='git status '
alias ga='git add '
alias gb='git branch '
alias gc='git commit'
alias gd='git diff'
alias go='git checkout '
alias gk='gitk --all&'
alias gx='gitx --all'

alias got='git '
alias get='git '
```

Сокращение `go` для команды `git checkout` особенно полезно. Оно позволяет мне вводить:

```
go <branch>
```

для переключения в отдельную ветку.

И да, я достаточно часто пишу вместо `git get` или `got`, поэтому создам алиасы и для них.

12. Получение старых версий

Цели

- Научиться возвращать рабочий каталог к любому предыдущему состоянию.

Возвращаться назад в историю очень просто. Команда checkout скопирует любой снимок из репозитория в рабочий каталог.

01 Получите хэши предыдущих версий

Выполните:

```
git hist
```

Примечание: Вы не забыли задать `hist` в вашем файле `.gitconfig`? Если забыли, посмотрите еще раз урок по алиасам.

Результат:

```
$ git hist
* fa3c141 2011-03-09 | Added HTML header (HEAD, master) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags [Alexander Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

Изучите данные лога и найдите хэш для первого коммита. Он должен быть в последней строке данных `git hist`. Используйте этот хэш-код (достаточно первых 7 знаков) в команде ниже. Затем проверьте содержимое файла `hello.html`.

Выполните:

```
git checkout <hash>
cat hello.html
```

Примечание: Многие команды зависят от хэшевых значений в репозитории. Поскольку ваши хеш-значения будут отличаться от моих, когда вы видите что-то вроде `<hash>` или `<treehash>` в команде, подставьте необходимое значение хэш для вашего репозитория.

Вы увидите…

Результат:

```
$ git checkout 911e8c9
```

```
Note: checking out '911e8c9'.

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 -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 911e8c9... First Commit
$ cat hello.html
Hello, World
```

Выходные данные команды `checkout` очень хорошо объясняют ситуацию. Старые версии git будут ругаться, что не расположены в локальной ветке. В любом случае, сейчас об этом не беспокойтесь.

Обратите внимание на то, что содержимое файла hello.html является значением по умолчанию.

02 Вернитесь к последней версии в ветке master

Выполните:

```
git checkout master
cat hello.html
```

Вы увидите…

Результат:

```
$ git checkout master
Previous HEAD position was 911e8c9... First Commit
Switched to branch 'master'
$ cat hello.html
<html>
  <head>
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

«master» — имя ветки по умолчанию. Переключая имена веток, вы попадаете на последнюю версию выбранной ветки.

13. Создание тегов версий

Цели

- Узнать, как создавать теги для коммитов для использования в будущем

Давайте назовем текущую версию страницы hello первой (*v1*).

01 Создайте тег первой версии

Выполните:

```
git tag v1
```

Теперь текущая версия страницы называется *v1*.

02 Теги для предыдущих версий

Давайте создадим тег для версии, которая идет перед текущей версией и назовем его *v1-beta*. В первую очередь нам надо переключиться на предыдущую версию. Вместо поиска до хэш, мы будем использовать ^, обозначающее «родитель v1».

Если обозначение `v1^` вызывает у вас какие-то проблемы, попробуйте также `v1~1`, указывающее на ту же версию. Это обозначение можно определить как «первую версию предшествующую v1».

Выполните:

```
git checkout v1^
cat hello.html
```

Результат:

```
$ git checkout v1^
Note: checking out 'v1^'.

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 -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 8c32287... Added standard HTML page tags
$ cat hello.html
<html>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

Это версия с тегами `<html>` и `<body>`, но еще *пока* без `<head>`. Давайте сделаем ее версией *v1-beta*.

Выполните:

```
git tag v1-beta
```

03 Переключение по имени тега

Теперь попробуйте попереключаться между двумя отмеченными версиями.

Выполните:

```
git checkout v1
git checkout v1-beta
```

Результат:

```
$ git checkout v1
Previous HEAD position was 8c32287... Added standard HTML page tags
HEAD is now at fa3c141... Added HTML header
$ git checkout v1-beta
Previous HEAD position was fa3c141... Added HTML header
HEAD is now at 8c32287... Added standard HTML page tags
```

04 Просмотр тегов с помощью команды `tag`

Вы можете увидеть, какие теги доступны, используя команду `git tag`.

Выполните:

```
git tag
```

Результат:

```
$ git tag
v1
v1-beta
```

05 Просмотр Тегов в логах

Вы также можете посмотреть теги в логе.

Выполните:

```
git hist master --all
```

Результат:

```
$ git hist master --all
* fa3c141 2011-03-09 | Added HTML header (v1, master) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (HEAD, v1-beta)
[Alexander Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

Вы можете видеть теги (`v1` и `v1-beta`) в логе вместе с именем ветки (`master`). Кроме того `HEAD` показывает коммит, на который вы переключились (на данный момент это `v1-beta`).

14. Отмена локальных изменений (до индексации)

Цели

- Научиться отменять изменения в рабочем каталоге

01 Переключитесь на ветку Master

Убедитесь, что вы находитесь на последнем коммите ветки master, прежде чем продолжить работу.

Выполните:

```
git checkout master
```

02 Измените hello.html

Иногда случается, что вы изменили файл в рабочем каталоге, и хотите отменить последние коммиты. С этим справится команда `checkout`.

Внесите изменение в файл hello.html в виде нежелательного комментария.

Файл: ***hello.html***

```
<html>
  <head>
  </head>
  <body>
    <h1>Hello, World!</h1>
    <!-- This is a bad comment.  We want to revert it. -->
  </body>
</html>
```

03 Проверьте состояние

Сначала проверьте состояние рабочего каталога.

Выполните:

```
git status
```

Результат:

```
$ git status
# On branch master
# 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:   hello.html
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Мы видим, что файл `hello.html` был изменен, но еще не проиндексирован.

04 Отмена изменений в рабочем каталоге

Используйте команду `checkout` для переключения в версию файла `hello.html` в репозитории.

Выполните:

```
git checkout hello.html
git status
cat hello.html
```

Результат:

```
$ git checkout hello.html
$ git status
# On branch master
nothing to commit (working directory clean)
$ cat hello.html
<html>
  <head>
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

Команда `status` показывает нам, что не было произведено никаких изменений, не зафиксированных в рабочем каталоге. И «нежелательный комментарий» больше не является частью содержимого файла.

15. Отмена проиндексированных изменений (перед коммитом)

Цели

- Научиться отменять изменения, которые были проиндексированы

01 Измените файл и проиндексируйте изменения

Внесите изменение в файл `hello.html` в виде нежелательного комментария

Файл: ***hello.html***

```
<html>
  <head>
    <!-- This is an unwanted but staged comment -->
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

Проиндексируйте это изменение.

Выполните:

```
git add hello.html
```

02 Проверьте состояние

Проверьте состояние нежелательного изменения.

Выполните:

```
git status
```

Результат:

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.html
#
```

Состояния показывает, что изменение было проиндексировано и готово к коммиту.

03 Выполните сброс буферной зоны

К счастью, вывод состояние показывает нам именно то, что мы должны сделать для отмены индексации изменения.

Выполните:

```
git reset HEAD hello.html
```

Результат:

```
$ git reset HEAD hello.html
Unstaged changes after reset:
M   hello.html
```

Команда `reset` сбрасывает буферную зону к HEAD. Это очищает буферную зону от изменений, которые мы только что проиндексировали.

Команда `reset` (по умолчанию) не изменяет рабочий каталог. Поэтому рабочий каталог все еще содержит нежелательный комментарий. Мы можем использовать команду `checkout` из предыдущего урока, чтобы удалить нежелательные изменения в рабочем каталоге.

04 Переключитесь на версию коммита

Выполните:

```
git checkout hello.html
git status
```

Результат:

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

Наш рабочий каталог опять чист.

16. Отмена коммитов

Цели

- Научиться отменять коммиты в локальный репозиторий.

01 Отмена коммитов

Иногда вы понимаете, что новые коммиты являются неверными, и хотите их отменить. Есть несколько способов решения этого вопроса, здесь мы будем использовать самый безопасный.

Мы отменим коммит путем создания нового коммита, отменяющего нежелательные изменения.

02 Измените файл и сделайте коммит

Измените файл `hello.html` на следующий.

Файл: *hello.html*

```
<html>
  <head>
  </head>
  <body>
    <h1>Hello, World!</h1>
    <!-- This is an unwanted but committed change -->
  </body>
</html>
```

Выполните:

```
git add hello.html
git commit -m "Oops, we didn't want this commit"
```

03 Сделайте коммит с новыми изменениями, отменяющими предыдущие

Чтобы отменить коммит, нам необходимо сделать коммит, который удаляет изменения, сохраненные нежелательным коммитом.

Выполните:

```
git revert HEAD
```

Перейдите в редактор, где вы можете отредактировать коммит-сообщение по умолчанию или оставить все как есть. Сохраните и закройте файл. Вы увидите…

Результат:

```
$ git revert HEAD --no-edit
[master 45fa96b] Revert "Oops, we didn't want this commit"
 1 files changed, 1 insertions(+), 1 deletions(-)
```

Так как мы отменили самый последний произведенный коммит, мы смогли использовать `HEAD` в качестве аргумента для отмены. Мы можем отменить любой произвольной коммит в истории, указав его хэш-значение.

Примечание: Команду `--no-edit` можно проигнорировать. Она была необходима для генерации выходных данных без открытия редактора.

04 Проверьте лог

Проверка лога показывает нежелательные и отмененные коммиты в наш репозиторий.

Выполните:

```
git hist
```

Результат:

```
$ git hist
* 45fa96b 2011-03-09 | Revert "Oops, we didn't want this commit" (HEAD,
master) [Alexander Shvets]
* 846b90c 2011-03-09 | Oops, we didn't want this commit [Alexander Shvets]
* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

Эта техника будет работать с любым коммитом (хотя, возможно, возникнут конфликты). Она безопасна в использовании даже в публичных ветках удаленных репозиториев.

05 Далее

Далее давайте посмотрим на технику, которая может быть использована для удаления последних коммитов из истории репозитория.

17. Удаление коммиттов из ветки

Цели

- Научиться удалять самые последние коммиты из ветки

`Revert` из предыдущего раздела является мощной командой, которая позволяет отменить любые коммиты в репозиторий. Однако, и оригинальный и «отмененный» коммиты видны в истории ветки (при использовании команды `git log`).

Часто мы делаем коммит, и сразу понимаем, что это была ошибка. Было бы неплохо иметь команду «возврата», которая позволила бы нам сделать вид, что неправильного коммита никогда и не было. Команда «возврата» даже предотвратила бы появление нежелательного коммита в истории `git log`.

01 Команда `reset`

Мы уже видели команду `reset` и использовали ее для согласования буферной зоны и выбранного коммита (мы использовали коммит HEAD в нашем предыдущем уроке).

При получении ссылки на коммит (т.е. хэш, ветка или имя тега), команда `reset`…

1. Перепишет текущую ветку, чтобы она указывала на нужный коммит
2. Опционально сбросит буферную зону для соответствия с указанным коммитом
3. Опционально сбросит рабочий каталог для соответствия с указанным коммитом

02 Проверьте нашу историю

Давайте сделаем быструю проверку нашей истории коммитов.

Выполните:

```
git hist
```

Результат:

```
$ git hist
* 45fa96b 2011-03-09 | Revert "Oops, we didn't want this commit" (HEAD,
master) [Alexander Shvets]
* 846b90c 2011-03-09 | Oops, we didn't want this commit [Alexander Shvets]
* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

Мы видим, что два последних коммита в этой ветке - «Oops» и «Revert Oops». Давайте удалим их с помощью сброса.

03 Для начала отметьте эту ветку

Но прежде чем удалить коммиты, давайте отметим последний коммит тегом, чтобы потом можно было его найти.

Выполните:

```
git tag oops
```

04 Сброс коммитов к предшествующим коммиту Oops

Глядя на историю лога (см. выше), мы видим, что коммит с тегом «v1» является коммитом, предшествующим ошибочному коммиту. Давайте сбросим ветку до этой точки. Поскольку ветка имеет тег, мы можем использовать имя тега в команде сброса (если она не имеет тега, мы можем использовать хэш-значение).

Выполните:

```
git reset --hard v1
git hist
```

Результат:

```
$ git reset --hard v1
HEAD is now at fa3c141 Added HTML header
$ git hist
* fa3c141 2011-03-09 | Added HTML header (HEAD, v1, master) [Alexander
Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

Наша ветка master теперь указывает на коммит v1, а коммитов Oops и Revert Oops в ветке уже нет. Параметр `--hard` указывает, что рабочий каталог должен быть обновлен в соответствии с новым head ветки.

05 Ничего никогда не теряется

Что же случается с ошибочными коммитами? Оказывается, что коммиты все еще находятся в репозитории. На самом деле, мы все еще можем на них ссылаться. Помните, в начале этого урока мы создали для отмененного коммита тег «oops». Давайте посмотрим на *все* коммиты.

Выполните:

```
git hist --all
```

Результат:

```
$ git hist --all
* 45fa96b 2011-03-09 | Revert "Oops, we didn't want this commit" (oops)
[Alexander Shvets]
* 846b90c 2011-03-09 | Oops, we didn't want this commit [Alexander Shvets]
* fa3c141 2011-03-09 | Added HTML header (HEAD, v1, master) [Alexander
Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

Мы видим, что ошибочные коммиты не исчезли. Они все еще находятся в репозитории. Просто они отсутствуют в ветке `master`. Если бы мы не отметили их тегами, они по-прежнему находились бы в репозитории, но не было бы никакой возможности ссылаться

на них, кроме как при помощи их хэш имен. Коммиты, на которые нет ссылок, остаются в репозитории до тех пор, пока не будет запущен сборщик мусора.

06 Опасность сброса

Сброс в локальных ветках, как правило, безопасен. Последствия любой «аварии» как правило, можно восстановить простым сбросом с помощью нужного коммита.

Однако, если ветка «расшарена» на удаленных репозиториях, сброс может сбить с толку других пользователей ветки.

18. Удаление тега oops

Цели

- Удаление тега oops (уборка)

01 Удаление тега oops

Тег oops свою функцию выполнил. Давайте удалим его и коммиты, на которые он ссылался, сборщиком мусора.

Выполните:

```
git tag -d oops
git hist --all
```

Результат:

```
$ git tag -d oops
Deleted tag 'oops' (was 45fa96b)
$ git hist --all
* fa3c141 2011-03-09 | Added HTML header (HEAD, v1, master) [Alexander
Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

Тег «oops» больше не будет отображаться в репозитории.

19. Внесение изменений в коммиты

Цели

- Научиться изменять существующие коммиты

01 Измените страницу, а затем сделайте коммит

Добавьте в страницу комментарий автора.

Файл: ***hello.html***

```
<!-- Author: Alexander Shvets -->
<html>
  <head>
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

Выполните:

```
git add hello.html
git commit -m "Add an author comment"
```

02 Ой... необходим email

После совершения коммита вы понимаете, что любой хороший комментарий должен включать электронную почту автора. Обновите страницу hello, включив в нее email.

Файл: ***hello.html***

```
<!-- Author: Alexander Shvets (alex@githowto.com) -->
<html>
  <head>
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

03 Измените предыдущий коммит

Мы действительно не хотим создавать отдельный коммит только ради электронной почты. Давайте изменим предыдущий коммит, включив в него адрес электронной почты.

Выполните:

```
git add hello.html
git commit --amend -m "Add an author/email comment"
```

Результат:

```
$ git add hello.html
$ git commit --amend -m "Add an author/email comment"
[master 6a78635] Add an author/email comment
 1 files changed, 2 insertions(+), 1 deletions(-)
```

04 Просмотр истории

Выполните:

```
git hist
```

Результат:

```
$ git hist
* 6a78635 2011-03-09 | Add an author/email comment (HEAD, master) [Alexander
Shvets]
* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

Мы можем увидеть, что оригинальный коммит «автор» заменен коммитом «автор/email». Этого же эффекта можно достичь путем сброса последнего коммита в ветке, и повторного коммита новых изменений.

20. Перемещение файлов

Цели

- Научиться перемещать файл в пределах репозитория.

01 Переместите файл hello.html в каталог lib

Сейчас мы собираемся создать структуру нашего репозитория. Давайте перенесем страницу в каталог lib.

Выполните:

```
mkdir lib
git mv hello.html lib
git status
```

Результат:

```
$ mkdir lib
$ git mv hello.html lib
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   renamed:    hello.html -> lib/hello.html
#
```

Перемещая файлы с помощью git, мы информируем git о 2 вещах

1. Что файл `hello.html` был удален.
2. Что файл `lib/hello.html` был создан.

Оба эти факта сразу же проиндексированы и готовы к коммиту. Команда git status сообщает, что файл был перемещен.

02 Второй способ перемещения файлов

Позитивной чертой git является то, что вы можете забыть о версионном контроле до того момента, когда вы готовы приступить к коммиту кода. Что бы случилось, если бы мы использовали командную строку операционной системы для перемещения файлов вместо команды git?

Оказывается, следующий набор команд идентичен нашим последним действиям. Работы здесь побольше, но результат тот же.

Мы могли бы выполнить:

```
mkdir lib
mv hello.html lib
git add lib/hello.html
git rm hello.html
```

03 Коммит в новый каталог

Давайте сделаем коммит этого перемещения.

Выполните:

```
git commit -m "Moved hello.html to lib"
```

21. Подробнее о структуре

Цели

- Добавить еще один файл в наш репозиторий

01 Добавление index.html

Давайте добавим файл index.html в наш репозиторий. Следующий файл отлично подойдет для этой цели.

Файл: *index.html*

```
<html>
  <body>
    <iframe src="lib/hello.html" width="200" height="200" />
  </body>
</html>
```

Добавьте файл и сделайте коммит.

Выполните:

```
git add index.html
git commit -m "Added index.html."
```

Теперь при открытии index.html, вы должны увидеть кусок страницы hello в маленьком окошке.

22. Git внутри: Каталог .git

Цели

- Узнать о структуре каталога `.git`

01 Каталог `.git`

Настало время провести небольшое исследование. Для начала, из корневого каталога вашего проекта…

Выполните:

```
ls -C .git
```

Результат:

```
$ ls -C .git
COMMIT_EDITMSG  MERGE_RR    config      hooks       info        objects
rr-cache
HEAD        ORIG_HEAD   description index       logs        refs
```

Это магический каталог, в котором хранятся все «материалы» git. Давайте заглянем в каталог объектов.

02 База данных объектов

Выполните:

```
ls -C .git/objects
```

Результат:

```
$ ls -C .git/objects
09  24  28  45  59  6a  77  80  8c  97  af  c4  e7  info
11  27  43  56  69  6b  78  84  91  9c  b5  e4  fa  pack
```

Вы должны увидеть кучу каталогов, имена которых состоят из 2 символов. Имена каталогов являются первыми двумя буквами хэша sha1 объекта, хранящегося в git.

03 Углубляемся в базу данных объектов

Выполните:

```
ls -C .git/objects/<dir>
```

Результат:

```
$ ls -C .git/objects/09
6b74c56bfc6b40e754fc0725b8c70b2038b91e
9fb6f9d3a104feb32fcac22354c4d0e8a182c1
```

Смотрим в один из каталогов с именем из 2 букв. Вы увидите файлы с именами из 38 символов. Это файлы, содержащие объекты, хранящиеся в git. Они сжаты и закодированы, поэтому просмотр их содержимого нам мало чем поможет. Рассмотрим далее каталог .git внимательно

04 Config File

Выполните:

```
cat .git/config
```

Результат:

```
$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
    ignorecase = true
[user]
    name = Alexander Shvets
    email = alex@githowto.com
```

Это файл конфигурации, создающийся для каждого конкретного проекта. Записи в этом файле будут перезаписывать записи в файле `.gitconfig` вашего главного каталога, по крайней мере в рамках этого проекта.

05 Ветки и теги

Выполните:

```
ls .git/refs
ls .git/refs/heads
ls .git/refs/tags
cat .git/refs/tags/v1
```

Результат:

```
$ ls .git/refs
heads
tags
$ ls .git/refs/heads
master
$ ls .git/refs/tags
v1
v1-beta
$ cat .git/refs/tags/v1
fa3c1411aa09441695a9e645d4371e8d749da1dc
```

Вы должны узнавать файлы в подкаталоге тегов. Каждый файл соответствует тегу, ранее созданному с помощью команды `git tag`. Его содержание – это всего лишь хэш коммита, привязанный к тегу.

Каталог *heads* практически аналогичен, но используется для веток, а не тегов. На данный момент у нас есть только одна ветка, так что все, что вы увидите в этом каталоге – это ветка *master*.

06 Файл HEAD

Выполните:

```
cat .git/HEAD
```

Результат:

```
$ cat .git/HEAD
ref: refs/heads/master
```

Файл HEAD содержит ссылку на текущую ветку, в данный момент это должна быть ветка master.

23. Git внутри: Работа непосредственно с объектами git

Цели

- Исследовать структуру базы данных объектов
- Научиться использовать SHA1 хэши для поиска содержимого в репозитории

Давайте исследуем объекты git с помощью некоторых инструментов.

01 Поиск последнего коммита

Выполните:

```
git hist --max-count=1
```

Эта команда должна показать последний коммит в репозиторий. SHA1 хэш в вашей системе, вероятно, отличается от моего, но вы увидите что-то наподобие этого.

Результат:

```
$ git hist --max-count=1
* 8029c07 2011-03-09 | Added index.html. (HEAD, master) [Alexander Shvets]
```

02 Вывод последнего коммита

С помощью SHA1 хэша из коммита, указанного выше…

Выполните:

```
git cat-file -t <hash>
```

```
git cat-file -p <hash>
```

Вот что выходит у меня…

Результат:

```
$ git cat-file -t 8029c07
commit
$ git cat-file -p 8029c07
tree 096b74c56bfc6b40e754fc0725b8c70b2038b91e
parent 567948ac55daa723807c0c16e34c76797efbcbed
author Alexander Shvets <alex@githowto.com> 1299684476 -0500
committer Alexander Shvets <alex@githowto.com> 1299684476 -0500

Added index.html.
```

Примечание: Если вы задали алиасы «type» и «dump», как описано в уроке об алиасах, можете вводить команды `git type` и `git dump` вместо длинных команд (которые я никогда не запоминаю).

Это вывод объекта коммита, который находится во главе ветки master.

03 Поиск дерева

Мы можем вывести дерево каталогов, ссылка на который идет в коммите. Это должно быть описание файлов (верхнего уровня) в нашем проекте (для конкретного коммита). Используйте SHA1 хэш из строки «дерева», из списка выше.

Выполните:

```
git cat-file -p <treehash>
```

Вот как выглядит мое дерево…

Результат:

```
$ git cat-file -p 096b74c
100644 blob 28e0e9d6ea7e25f35ec64a43f569b550e8386f90    index.html
040000 tree e46f374f5b36c6f02fb3e9e922b79044f754d795    lib
```

Да, я вижу index.html и каталог lib.

04 Вывод каталога lib

Выполните:

```
git cat-file -p <libhash>
```

Результат:

```
$ git cat-file -p e46f374
100644 blob c45f26b6fdc7db6ba779fc4c385d9d24fc12cf72    hello.html
```

Существует файл `hello.html`.

05 Вывод файла `hello.html`

Выполните:

```
git cat-file -p <hellohash>
```

Результат:

```
$ git cat-file -p c45f26b
<!-- Author: Alexander Shvets (alex@githowto.com) -->
<html>
  <head>
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

А вот и он. Мы вывели объекты коммитов, объекты деревьев и объекты блобов непосредственно из репозитория git. Это все, что есть – блобы, деревья и коммиты.

06 Исследуйте самостоятельно

Исследуйте git репозиторий вручную самостоятельно. Смотрите, удастся ли вам найти оригинальный файл hello.html с самого первого коммита вручную по ссылкам SHA1 хэша в последнем коммите.

24. Создание ветки

Цели

- Научиться создавать локальную ветку в репозитории

Пора сделать наш hello world более выразительным. Так как это может занять некоторое время, лучше переместить эти изменения в отдельную ветку, чтобы изолировать их от изменений в ветке master.

01 Создайте ветку

Давайте назовем нашу новую ветку «style».

Выполните:

```
git checkout -b style
git status
```

Примечание: `git checkout -b <имяветки>` является шорткатом для `git branch <имяветки>` за которым идет `git checkout <имяветки>`.

Обратите внимание, что команда `git status` сообщает о том, что вы находитесь в ветке «style».

*02*Добавьте файл стилей style.css

Выполните:

```
touch lib/style.css
```

Файл: ***lib/style.css***

```
h1 {
  color: red;
}
```

Выполните:

```
git add lib/style.css
git commit -m "Added css stylesheet"
```

*03*Измените основную страницу

Обновите файл hello.html, чтобы использовать стили style.css.

Файл: ***lib/hello.html***

```
<!-- Author: Alexander Shvets (alex@githowto.com) -->
<html>
  <head>
    <link type="text/css" rel="stylesheet" media="all" href="style.css" />
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

Выполните:

```
git add lib/hello.html
git commit -m "Hello uses style.css"
```

*04*Измените index.html

Обновите файл index.html, чтобы он тоже использовал style.css

Файл: ***index.html***

```
<html>
  <head>
    <link type="text/css" rel="stylesheet" media="all" href="lib/style.css"
/>
  </head>
  <body>
    <iframe src="lib/hello.html" width="200" height="200" />
  </body>
</html>
```

Выполните:

```
git add index.html
git commit -m "Updated index.html"
```

05 Далее

Теперь у нас есть новая ветка под названием **style** с 3 новыми коммитами. Далее мы узнаем, как осуществлять навигацию и переключаться между ветками.

25. Навигация по веткам

Цели

- Научиться перемещаться между ветками репозитория

Теперь в вашем проекте есть две ветки:

Выполните:

```
git hist --all
```

Результат:

```
$ git hist --all
* 07a2a46 2011-03-09 | Updated index.html (HEAD, style) [Alexander Shvets]
* 649d26c 2011-03-09 | Hello uses style.css [Alexander Shvets]
* 1f3cbd2 2011-03-09 | Added css stylesheet [Alexander Shvets]
* 8029c07 2011-03-09 | Added index.html. (master) [Alexander Shvets]
* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]
* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]
* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

01 Переключение на ветку Master

Просто используйте команду `git checkout` для переключения между ветками.

Выполните:

```
git checkout master
cat lib/hello.html
```

Результат:

```
$ git checkout master
Switched to branch 'master'
$ cat lib/hello.html
<!-- Author: Alexander Shvets (alex@githowto.com) -->
<html>
  <head>
  </head>
  <body>
```

```
    <h1>Hello, World!</h1>
  </body>
</html>
```

Сейчас мы находимся на ветке Master. Это заметно по тому, что файл hello.html не использует стили `style.css`.

02 Вернемся к ветке «style».

Выполните:

```
git checkout style
cat lib/hello.html
```

Результат:

```
$ git checkout style
Switched to branch 'style'
$ cat lib/hello.html
<!-- Author: Alexander Shvets (alex@githowto.com) -->
<html>
  <head>
    <link type="text/css" rel="stylesheet" media="all" href="style.css" />
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

Содержимое `lib/hello.html` подтверждает, что мы вернулись в ветку **style**.

26. Изменения в ветке master

Цели

- Научиться работать с несколькими ветками с различными (и, возможно, конфликтующими) изменениями.

Пока вы меняли ветку «style», кто-то решил обновить ветку master. Они добавили README.

01 Создайте файл README в ветке master.

Выполните:

```
git checkout master
```

Файл: ***README***

```
This is the Hello World example from the git tutorial.
```

02 Сделайте коммит изменений README в ветку master.

Выполните:

```
git add README
git commit -m "Added README"
```

27. Просмотр отличающихся веток

Цели

- Научиться просматривать отличающиеся ветки в репозитории.

01 Просмотрите текущие ветки

Теперь у нас в репозитории есть две отличающиеся ветки. Используйте следующую лог-команду для просмотра веток и их отличий.

Выполните:

```
git hist --all
```

Результат:

```
$ git hist --all
* 6c0f848 2011-03-09 | Added README (HEAD, master) [Alexander Shvets]
| * 07a2a46 2011-03-09 | Updated index.html (style) [Alexander Shvets]
| * 649d26c 2011-03-09 | Hello uses style.css [Alexander Shvets]
| * 1f3cbd2 2011-03-09 | Added css stylesheet [Alexander Shvets]
|/
* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]
* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]
* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]
* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

Это наша первая возможность увидеть в действии `--graph` в `git hist`. Добавление опции `--graph` в `git log` вызывает построение дерева коммитов с помощью простых ASCII символов. Мы видим обе ветки (style и master), и то, что ветка master является текущей HEAD. Общим предшественником обеих веток является коммит «Added index.html».

Метка `--all` гарантированно означает, что мы видим все ветки. По умолчанию показывается только текущая ветка.

28. Слияние

Цели

- Научиться сливать две отличающиеся ветки для переноса изменений обратно в одну ветку.

01 Слияние веток

Слияние переносит изменения из двух веток в одну. Давайте вернемся к ветке style и сольем master с style.

Выполните:

```
git checkout style
git merge master
git hist --all
```

Результат:

```
$ git checkout style
Switched to branch 'style'
$ git merge master
Merge made by recursive.
 README |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README
$ git hist --all
*   5813a3f 2011-03-09 | Merge branch 'master' into style (HEAD, style)
[Alexander Shvets]
|\
| * 6c0f848 2011-03-09 | Added README (master) [Alexander Shvets]
* | 07a2a46 2011-03-09 | Updated index.html [Alexander Shvets]
* | 649d26c 2011-03-09 | Hello uses style.css [Alexander Shvets]
* | 1f3cbd2 2011-03-09 | Added css stylesheet [Alexander Shvets]
|/
* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]
* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]
* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]
* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

Путем периодического слияния ветки master с веткой style вы можете переносить из master любые изменения и поддерживать совместимость изменений style с изменениями в основной ветке.

Однако, это делает графики коммитов действительно уродливыми. Позже мы рассмотрим возможность перебазирования, как альтернативы слиянию.

02 Далее

Но что если изменения в ветке master конфликтуют с изменениями в style?

29. Создание конфликта

Цели

- Создание конфликтующих изменений в ветке master.

01 Вернитесь в master и создайте конфликт

Вернитесь в ветку master и внесите следующие изменения:

```
git checkout master
```

Файл: ***lib/hello.html***

```
<!-- Author: Alexander Shvets (alex@githowto.com) -->
<html>
  <head>
    <!-- no style -->
  </head>
  <body>
    <h1>Hello, World! Life is great!</h1>
  </body>
</html>
```

Выполните:

```
git add lib/hello.html
git commit -m 'Life is great!'
```

Внимание: используйте для этого коммита одинарные кавычки, дабы избежать проблем с символом `!`. В bash он считается служебным.

02 Просмотр веток

Выполните:

```
git hist --all
```

Результат:

```
$ git hist --all
* 454ec68 2011-03-09 | Life is great! (HEAD, master) [Alexander Shvets]
| * 5813a3f 2011-03-09 | Merge branch 'master' into style (style) [Alexander
Shvets]
| |\
| |/
|/|
* | 6c0f848 2011-03-09 | Added README [Alexander Shvets]
| * 07a2a46 2011-03-09 | Updated index.html [Alexander Shvets]
| * 649d26c 2011-03-09 | Hello uses style.css [Alexander Shvets]
| * 1f3cbd2 2011-03-09 | Added css stylesheet [Alexander Shvets]
|/
* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]
* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]
* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]
* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
```

```
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

После коммита «Added README» ветка master была объединена с веткой style, но в настоящее время в master есть дополнительный коммит, который не был слит с style.

03 Далее

Последнее изменение в master конфликтует с некоторыми изменениями в style. В следующем шаге мы решим этот конфликт.

30. Разрешение конфликтов

Цели

- Научиться разрешать конфликты во время слияния

01 Слияние master с веткой style

Теперь вернемся к ветке style и попытаемся объединить ее с новой веткой master.

Выполните:

```
git checkout style
git merge master
```

Результат:

```
$ git checkout style
Switched to branch 'style'
$ git merge master
Auto-merging lib/hello.html
CONFLICT (content): Merge conflict in lib/hello.html
Automatic merge failed; fix conflicts and then commit the result.
```

Если вы откроете lib/hello.html, вы увидите:

Файл: *lib/hello.html*

```
<!-- Author: Alexander Shvets (alex@githowto.com) -->
<html>
  <head>
<<<<<<< HEAD
    <link type="text/css" rel="stylesheet" media="all" href="style.css" />
=======
    <!-- no style -->
>>>>>>> master
  </head>
  <body>
    <h1>Hello,World! Life is great!</h1>
  </body>
</html>
```

Первый раздел - версия во главе текущей ветки (style). Второй раздел - версия ветки master.

02 Решение конфликта

Вам необходимо вручную разрешить конфликт. Внесите изменения в `lib/hello.html` для достижения следующего результата.

Файл: ***lib/hello.html***

```
<!-- Author: Alexander Shvets (alex@githowto.com) -->
<html>
  <head>
    <link type="text/css" rel="stylesheet" media="all" href="style.css" />
  </head>
  <body>
    <h1>Hello, World! Life is great!</h1>
  </body>
</html>
```

03 Сделайте коммит решения конфликта

Выполните:

```
git add lib/hello.html
git commit -m "Merged master fixed conflict."
```

Результат:

```
$ git add lib/hello.html
$ git commit -m "Merged master fixed conflict."
Recorded resolution for 'lib/hello.html'.
[style 645c4e6] Merged master fixed conflict.
```

04 Расширенные возможности слияния

Git не предоставляет никаких графических инструментов слияния, но будет с удовольствием работать с любыми сторонними инструментами слияния, которые вы хотите использовать (обсуждение таких инструментов на StackOverflow).

31. Перебазирование как альтернатива слиянию

Цели

- Узнать различия между перебазированием и слиянием.

Обсуждение

Давайте рассмотрим различия между слиянием и перебазированием. Для того, чтобы это сделать, нам нужно вернуться в репозиторий в момент до первого слияния, а затем повторить те же действия, но с использованием перебазирования вместо слияния.

Мы будем использовать команду `reset` для возврата веток к предыдущему состоянию.

32. Сброс ветки style

Цели

- Сброс ветки style до точки перед первым слиянием.

01 Сброс ветки style

Давайте вернемся во времени на ветке style к точке *перед* тем, как мы слили ее с веткой master. Мы можем **сбросить** ветку к любому коммиту. По сути, это изменение указателя ветки на любую точку дерева коммитов.

В этом случае мы хотим вернуться в ветке style в точку перед слиянием с master. Нам необходимо найти последний коммит перед слиянием.

Выполните:

```
git checkout style
git hist
```

Результат:

```
$ git checkout style
Already on 'style'
$ git hist
*   645c4e6 2011-03-09 | Merged master fixed conflict. (HEAD, style)
[Alexander Shvets]
|\
| * 454ec68 2011-03-09 | Life is great! (master) [Alexander Shvets]
* |   5813a3f 2011-03-09 | Merge branch 'master' into style [Alexander
Shvets]
|\ \
| |/
| * 6c0f848 2011-03-09 | Added README [Alexander Shvets]
* | 07a2a46 2011-03-09 | Updated index.html [Alexander Shvets]
* | 649d26c 2011-03-09 | Hello uses style.css [Alexander Shvets]
* | 1f3cbd2 2011-03-09 | Added css stylesheet [Alexander Shvets]
|/
* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]
* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]
* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]
* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

Это немного трудно читать, но, глядя на данные, мы видим, что коммит «Updated index.html» был последним на ветке style перед слиянием. Давайте сбросим ветку style к этому коммиту.

Выполните:

```
git reset --hard <hash>
```

Результат:

```
$ git reset --hard 07a2a46
HEAD is now at 07a2a46 Updated index.html
```

02 Проверьте ветку.

Поищите лог ветки style. У нас в истории больше нет коммитов слияний.

Выполните:

```
git hist --all
```

Результат:

```
$ git hist --all
* 454ec68 2011-03-09 | Life is great! (master) [Alexander Shvets]
* 6c0f848 2011-03-09 | Added README [Alexander Shvets]
| * 07a2a46 2011-03-09 | Updated index.html (HEAD, style) [Alexander Shvets]
| * 649d26c 2011-03-09 | Hello uses style.css [Alexander Shvets]
| * 1f3cbd2 2011-03-09 | Added css stylesheet [Alexander Shvets]
|/
* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]
* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]
* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]
* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

33. Сброс ветки master

Цели

- Сбросить ветку master в точку до конфликтующего коммита.

01 Сброс ветки master

Добавив интерактивный режим в ветку master, мы внесли изменения, конфликтующие с изменениями в ветке style. Давайте вернемся в ветке master в точку перед внесением конфликтующих изменений. Это позволяет нам продемонстрировать работу команды rebase, не беспокоясь о конфликтах.

Выполните:

```
git checkout master
git hist
```

Результат:

```
$ git hist
* 454ec68 2011-03-09 | Life is great! (HEAD, master) [Alexander Shvets]
* 6c0f848 2011-03-09 | Added README [Alexander Shvets]
* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]
* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]
* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]
* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

Коммит «Added README» идет непосредственно перед коммитом конфликтующего интерактивного режима. Мы сбросим ветку master к коммиту «Added README».

Выполните:

```
git reset --hard <hash>
git hist --all
```

Просмотрите лог. Он должен выглядеть, как будто репозиторий был перемотан назад во времени к точке до какого-либо слияния.

Результат:

```
$ git reset --hard 6c0f848
$ git hist --all
* 6c0f848 2011-03-09 | Added README (HEAD, master) [Alexander Shvets]
| * 07a2a46 2011-03-09 | Updated index.html (style) [Alexander Shvets]
| * 649d26c 2011-03-09 | Hello uses style.css [Alexander Shvets]
| * 1f3cbd2 2011-03-09 | Added css stylesheet [Alexander Shvets]
|/
* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]
* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]
* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]
* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

34. Перебазирование

Цели

- Использовать команду rebase вместо команды merge.

Итак, мы вернулись в точку до первого слияния и хотим перенести изменения в master в нашу ветку style.

На этот раз для переноса изменений из ветки master мы будем использовать команду rebase вместо слияния.

Выполните:

```
git checkout style
git rebase master
git hist
```

Результат:

```
$ go style
Switched to branch 'style'
$
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: Added css stylesheet
Applying: Hello uses style.css
Applying: Updated index.html
$
$ git hist
* 6e6c76a 2011-03-09 | Updated index.html (HEAD, style) [Alexander Shvets]
* 1436f13 2011-03-09 | Hello uses style.css [Alexander Shvets]
* 59da9a7 2011-03-09 | Added css stylesheet [Alexander Shvets]
* 6c0f848 2011-03-09 | Added README (master) [Alexander Shvets]
* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]
* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]
* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]
* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

01 Слияние VS перебазирование

Конечный результат перебазирования очень похож на результат слияния. Ветка style в настоящее время содержит все свои изменения, а также все изменения ветки master. Однако, дерево коммитов значительно отличается. Дерево коммитов ветки style было переписано таким образом, что ветка master является частью истории коммитов. Это делает цепь коммитов линейной и гораздо более читабельной.

02 Когда использовать перебазирование, а когда слияние?

Не используйте перебазирование …

1. Если ветка является публичной и расшаренной. Переписывание общих веток будет мешать работе других членов команды.
2. Когда важна *точная* история коммитов ветки (так как команда rebase переписывает историю коммитов).

Учитывая приведенные выше рекомендации, я предпочитаю использовать rebase для кратковременных, локальных веток, а слияние для веток в публичном репозитории.

35. Слияние в ветку master

Цели

- Мы поддерживали соответствие ветки style с веткой master (с помощью rebase), теперь давайте сольем изменения style в ветку master.

01 Слияние style в master

Выполните:

```
git checkout master
git merge style
```

Результат:

```
$ git checkout master
Switched to branch 'master'
$
$ git merge style
Updating 6c0f848..6e6c76a
Fast-forward
 index.html       |    2 +-
 lib/style.css |    8 ++++++++
 lib/hello.html   |    6 ++++--
 3 files changed, 13 insertions(+), 3 deletions(-)
 create mode 100644 lib/style.css
```

Поскольку последний коммит ветки master прямо предшествует последнему коммиту ветки style, git может выполнить ускоренное слияние-перемотку. При быстрой перемотке вперед, git просто передвигает указатель вперед, таким образом указывая на тот же коммит, что и ветка style.

При быстрой перемотке конфликтов быть не может.

02 Просмотрите логи

Выполните:

```
git hist
```

Результат:

```
$ git hist
* 6e6c76a 2011-03-09 | Updated index.html (HEAD, master, style) [Alexander
Shvets]
* 1436f13 2011-03-09 | Hello uses style.css [Alexander Shvets]
* 59da9a7 2011-03-09 | Added css stylesheet [Alexander Shvets]
* 6c0f848 2011-03-09 | Added README [Alexander Shvets]
* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]
* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]
* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]
* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
```

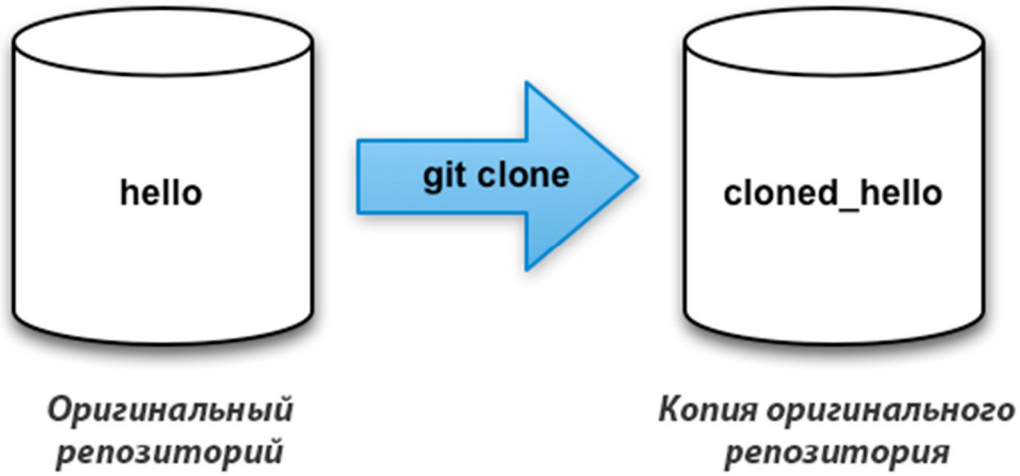
```
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

Теперь ветки style и master идентичны.

36. Несколько репозиториев

До сих пор мы работали с одним git репозиторием. Однако, git удается отлично работать с несколькими репозиториями. Эти дополнительные репозитории могут храниться локально, или доступ к ним может осуществляться через сетевое подключение.

В следующем разделе мы создадим новый репозиторий с именем «cloned_hello». Мы покажем, как перемещать изменения из одного репозитория в другой и как разрешать конфликты, возникающие в результате работы с двумя репозиториями.



А пока что поработаем с локальными репозиториями (т.е. репозиториями, хранящимися на вашем локальном жестком диске), однако практически все, что вы узнаете в этом разделе, будет применяться к нескольким репозиториям, несмотря на то, хранятся ли они локально или являются публичными.

Примечание: Мы будем вносить изменения в обе копии наших репозиториев. Обращайте внимание на то, в каком репозитории вы находитесь на каждом шаге следующих уроках.

37. Клонирование репозиториев

Цели

- Научиться делать копии репозиториев.

Если вы работаете в команде, последующие 12 глав довольно важны в понимании, т.к. вы почти всегда будете работать с клонированными репозиториями.

01 Перейдите в рабочий каталог

Перейдите в рабочий каталог и сделайте клон вашего репозитория hello.

Выполните:

```
cd ..
pwd
ls
```

Примечание: Сейчас мы находимся в рабочем каталоге.

Результат:

```
$ cd ..
$ pwd
/Users/alex/Documents/Presentations/githowto/auto
$ ls
hello
```

В этот момент вы должны находиться в «рабочем» каталоге. Здесь должен быть единственный репозиторий под названием «hello».

02 Создайте клон репозитория hello

Давайте создадим клон репозитория.

Выполните:

```
git clone hello cloned_hello
ls
```

Результат:

```
$ git clone hello cloned_hello
Cloning into cloned_hello...
done.
$ ls
cloned_hello
hello
```

В вашем рабочем каталоге теперь должно быть два репозитория: оригинальный репозиторий «hello» и клонированный репозиторий «cloned_hello»

38. Просмотр клонированного репозитория

Цели

- Узнать о ветках в удаленных репозиториях.

01 Посмотрите на клонированный репозиторий

Давайте взглянем на клонированный репозиторий.

Выполните:

```
cd cloned_hello
ls
```

Результат:

```
$ cd cloned_hello
$ ls
README
index.html
lib
```

Вы увидите список всех файлов на верхнем уровне оригинального репозитория `README`, `index.html` и `lib`).

02 Просмотрите историю репозитория

Выполните:

```
git hist --all
```

Результат:

```
$ git hist --all
* 6e6c76a 2011-03-09 | Updated index.html (HEAD, origin/master, origin/style,
origin/HEAD, master) [Alexander Shvets]
* 1436f13 2011-03-09 | Hello uses style.css [Alexander Shvets]
* 59da9a7 2011-03-09 | Added css stylesheet [Alexander Shvets]
* 6c0f848 2011-03-09 | Added README [Alexander Shvets]
* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]
* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]
* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]
* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

Вы увидите список всех коммитов в новый репозиторий, и он должен (более или менее) совпадать с историей коммитов в оригинальном репозитории. Единственная разница должна быть в названиях веток.

03 Удаленные ветки

Вы увидите ветку **master** (**HEAD**) в списке истории. Вы также увидите ветки со странными именами (**origin/master**, **origin/style** и **origin/HEAD**). Мы поговорим о них чуть позже.

39. Что такое origin?

Цели

- Узнать об именах удаленных репозиториев.

Выполните:

```
git remote
```

Результат:

```
$ git remote
origin
```

Мы видим, что клонированный репозиторий знает об имени по умолчанию удаленного репозитория. Давайте посмотрим, можем ли мы получить более подробную информацию об имени по умолчанию:

Выполните:

```
git remote show origin
```

Результат:

```
$ git remote show origin
* remote origin
  Fetch URL: /Users/alex/Documents/Presentations/githowto/auto/hello
  Push  URL: /Users/alex/Documents/Presentations/githowto/auto/hello
  HEAD branch (remote HEAD is ambiguous, may be one of the following):
    style
    master
  Remote branches:
    style  tracked
    master 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)
```

Мы видим, что «имя по умолчанию» удаленного репозитория – оригинальное **hello**. Удаленные репозитории обычно размещаются на отдельной машине, возможно, централизованном сервере. Однако, как мы видим здесь, они могут с тем же успехом указывать на репозиторий на той же машине. Нет ничего особенного в «имени по умолчанию», однако существует традиция использовать «имя по умолчанию» на первичном централизованном репозитории (если таковой имеется).

40. Удаленные ветки

Цели

- Узнать о локальных и удаленных ветках

Давайте посмотрим на ветки, доступные в нашем клонированном репозитории.

Выполните:

```
git branch
```

Результат:

```
$ git branch
* master
```

Как мы видим, в списке только ветка master. Где ветка style? Команда **git branch** выводит только список локальных веток по умолчанию.

01 Список удаленных веток

Для того, чтобы увидеть все ветки, попробуйте следующую команду:

Выполните:

```
git branch -a
```

Результат:

```
$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/style
  remotes/origin/master
```

Git выводит все коммиты в оригинальный репозиторий, но ветки в удаленном репозитории не рассматриваются как локальные. Если мы хотим собственную ветку **style**, мы должны сами ее создать. Через минуту вы увидите, как это делается.

41. Изменение оригинального репозитория

Цели

- Внести некоторые изменения в оригинальный репозиторий, чтобы затем попытаться извлечь и слить изменения из удаленной ветки в текущую

01 Внесите изменения в оригинальный репозиторий hello

Выполните:

```
cd ../hello
# (You should be in the original hello repository now)
```

Примечание: Сейчас мы находимся в репозитории *hello*

Внесите следующие изменения в файл README:

Файл: *README*

```
This is the Hello World example from the git tutorial.
(changed in original)
```

Теперь добавьте это изменение и сделайте коммит

Выполните:

```
git add README
git commit -m "Changed README in original repo"
```

02 Далее

Теперь в оригинальном репозитории есть более поздние изменения, которых нет в клонированной версии. Далее мы извлечем и сольем эти изменения в клонированный репозиторий.

42. Извлечение изменений

Цели

- Научиться извлекать изменения из удаленного репозитория.

Выполните:

```
cd ../cloned_hello
git fetch
git hist --all
```

Примечание: Сейчас мы находимся в репозитории *cloned_hello*

Результат:

```
$ git fetch
From /Users/alex/Documents/Presentations/githowto/auto/hello
   6e6c76a..2faa4ea  master     -> origin/master
$ git hist --all
* 2faa4ea 2011-03-09 | Changed README in original repo (origin/master,
origin/HEAD) [Alexander Shvets]
* 6e6c76a 2011-03-09 | Updated index.html (HEAD, origin/style, master)
[Alexander Shvets]
* 1436f13 2011-03-09 | Hello uses style.css [Alexander Shvets]
* 59da9a7 2011-03-09 | Added css stylesheet [Alexander Shvets]
* 6c0f848 2011-03-09 | Added README [Alexander Shvets]
* 8029c07 2011-03-09 | Added index.html. [Alexander Shvets]
* 567948a 2011-03-09 | Moved hello.html to lib [Alexander Shvets]
* 6a78635 2011-03-09 | Add an author/email comment [Alexander Shvets]
* fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
* 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander
Shvets]
* 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
* 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
```

На данный момент в репозитории есть все коммиты из оригинального репозитория, но они не интегрированы в локальные ветки клонированного репозитория.

В истории выше найдите коммит «Changed README in original repo». Обратите внимание, что коммит включает в себя коммиты «origin/master» и «origin/HEAD».

Теперь давайте посмотрим на коммит «Updated index.html». Вы увидите, что локальная ветка master указывает на этот коммит, а не на новый коммит, который мы только что извлекли.

Выводом является то, что команда «git fetch» будет извлекать новые коммиты из удаленного репозитория, но не будет сливать их с вашими наработками в локальных ветках.

01 Проверьте README

Мы можем продемонстрировать, что клонированный файл README не изменился.

Выполните:

```
cat README
```

Результат:

```
$ cat README
This is the Hello World example from the git tutorial.
```

Как видите, никаких изменений.

43. Слияние извлеченных изменений

Цели

- Научиться перемещать извлеченные изменения в текущую ветку и рабочий каталог.

01 Слейте извлеченные изменения в локальную ветку master

Выполните:

```
git merge origin/master
```

Результат:

```
$ git merge origin/master
Updating 6e6c76a..2faa4ea
Fast-forward
 README |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
```

02 Еще раз проверьте файл README

Сейчас мы должны увидеть изменения.

Выполните:

```
cat README
```

Результат:

```
$ cat README
This is the Hello World example from the git tutorial.
(changed in original)
```

Вот и изменения. Хотя команда «git fetch» не сливает изменения, мы можем вручную слить изменения из удаленного репозитория.

03 Далее

Теперь давайте рассмотрим объединение fetch & merge в одну команду.

44. Извлечение и слияние изменений

Цели

- Узнать о том, что команда `git pull` эквивалентна комбинации `git fetch` и `git merge`.

Обсуждение

Мы не собираемся опять проходить весь процесс создания нового изменения и его извлечения, но мы хотим, чтобы вы знали, что выполнение:

```
git pull
```

действительно эквивалентно двум следующим шагам:

```
git fetch
git merge origin/master
```

45. Добавление ветки наблюдения

Цели

- Научиться добавлять локальную ветку, которая отслеживает изменения удаленной ветки.

Ветки, которые начинаются с remotes/origin являются ветками оригинального репозитория. Обратите внимание, что у вас больше нет ветки под названием style, но он знает, что в оригинальном репозитории ветка style была.

01 Добавьте локальную ветку, которая отслеживает удаленную ветку.

Выполните:

```
git branch --track style origin/style
git branch -a
git hist --max-count=2
```

Результат:

```
$ git branch --track style origin/style
Branch style set up to track remote branch style from origin.
$ git branch -a
  style
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/style
  remotes/origin/master
$ git hist --max-count=2
* 2faa4ea 2011-03-09 | Changed README in original repo (HEAD, origin/master,
origin/HEAD, master) [Alexander Shvets]
* 6e6c76a 2011-03-09 | Updated index.html (origin/style, style) [Alexander
Shvets]
```

Теперь мы можем видеть ветку style в списке веток и логе.

46. Чистые репозитории

Цели

- Научиться создавать чистые репозитории.

Чистые репозитории (без рабочих каталогов) обычно используются для расшаривания.

Небольшое пояснение, что же все-таки означает «чистый репозиторий». Обычный git-репозиторий подразумевает, что вы будете использовать его как рабочую директорию, поэтому вместе с файлами проекта в актуальной версии, git хранит все служебные, «чисто-репозиториевские» файлы в поддиректории .git. В удаленных репозиториях нет смысла хранить рабочие файлы на диске (как это делается в рабочих копиях), а все что им действительно нужно — это дельты изменений и другие бинарные данные репозитория. Вот это и есть «чистый репозиторий».

01 Создайте чистый репозиторий

Выполните:

```
cd ..
git clone --bare hello hello.git
ls hello.git
```

Примечание: Сейчас мы находимся в рабочем каталоге

Результат:

```
$ git clone --bare hello hello.git
Cloning into bare repository hello.git...
done.
$ ls hello.git
HEAD
config
description
hooks
info
objects
packed-refs
refs
```

Как правило, репозитории, оканчивающиеся на «.git» являются чистыми репозиториями. Мы видим, что в репозитории hello.git нет рабочего каталога. По сути, это есть не что иное, как каталог .git нечистого репозитория.

47. Добавление удаленного репозитория

Цели

- Добавить чистый репозиторий в качестве удаленного репозитория к нашему оригинальному репозиторию.

Давайте добавим репозиторий hello.git к нашему оригинальному репозиторию.

Выполните:

```
cd hello
git remote add shared ../hello.git
```

Примечание: Сейчас мы находимся в репозитории hello.

48. Отправка изменений

Цели

- Научиться отправлять изменения в удаленный репозиторий.

Так как чистые репозитории, как правило, расшариваются на каком-нибудь сетевом сервере, нам необходимо отправить наши изменения в другие репозитории.

Начнем с создания изменения для отправки. Отредактируйте файл README и сделайте коммит

Файл: *README*

```
This is the Hello World example from the git tutorial.
(Changed in the original and pushed to shared)
```

Выполните:

```
git checkout master
git add README
git commit -m "Added shared comment to readme"
```

Теперь отправьте изменения в общий репозиторий.

Выполните:

```
git push shared master
```

Общим называется репозиторий, получающий отправленные нами изменения. (Помните, мы добавили его в качестве удаленного репозитория в предыдущем уроке.)

Результат:

```
$ git push shared master
To ../hello.git
   2faa4ea..79f507c  master -> master
```

Примечание: Мы должны были явно указать ветку master для отправки изменений. Это можно настроить автоматически, но я все время забываю нужные команды. Для более простого управления удаленными ветками переключитесь в «Git Remote Branch».

49. Извлечение общих изменений

Цели

- Научиться извлекать изменения из общего репозитория.

Быстро переключитесь в клонированный репозиторий и извлеките изменения, только что отправленные в общий репозиторий.

Выполните:

```
cd ../cloned_hello
```

Примечание: Сейчас мы находимся в репозитории *cloned_hello*.

Продолжите с…

Выполните:

```
git remote add shared ../hello.git
git branch --track shared master
git pull
cat README
```

50. Размещение ваших git репозиториев

Цели

- Научиться настраивать git сервер для совместного использования репозиториев.

Есть много способов расшаривать git репозитории по сети. Вот быстрый способ.

01 Запуск git сервера

Выполните:

```
# (From the work directory)
git daemon --verbose --export-all --base-path=.
```

Теперь в отдельном окне терминала перейдите в ваш рабочий каталог

Выполните:

```
# (From the work directory)
git clone git://localhost/hello.git network_hello
cd network_hello
ls
```

Вы увидите копию проекта hello.

02 Отправка в Git Daemon

Если вы хотите совершить отправку в репозиторий Git Daemon, добавьте метку `--enable=receive-pack` к команде git daemon. Будьте осторожны, этот сервер не производит аутентификацию, поэтому любой может отправлять изменения в ваш репозиторий.

51. Расшаривание репозиториев

Цели

- Научиться расшаривать репозитории по WIFI.

Посмотрите, запущен ли git daemon у вашего соседа. Обменяйтесь IP-адресами и проверьте, сможете ли вы извлекать изменения из репозиториев друг друга.