**Лабораторная работа 2**

# Алиасы

**Цели:** Научиться настраивать алиасы и шорткаты для команд git

**1.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.

**2.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.

**3. 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.

**4. 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'i

alias gco='git checkout '

alias gk='gitk --all&'

alias gx='gitx --all'

alias got='git '

alias get='git '

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

gco <branch>

для переключения в отдельную ветку.

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

# 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.

## 1. 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](https://githowto.com/ru/aliases).

#### 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.

#### 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.

## 2. Вернитесь к последней версии в ветке master

#### ВЫПОЛНИТЕ:

git checkout master

cat hello.html

#### 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.

# Создание тегов версий

### Цели: Узнать, как создавать теги для коммитов для использования в будущем

Давайте назовем текущую версию страницы hello первой (v1).

## 1. Создайте тег первой версии

#### ВЫПОЛНИТЕ:

git tag v1

Теперь текущая версия страницы называется v1.

## 2. 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 по хэшу, 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 c тегами <html> и <body>, но еще пока без <head>. Давайте сделаем ее версией v1-beta.

#### ВЫПОЛНИТЕ:

git tag v1-beta

## 3. 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

## 4.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

## 5.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).

# Отмена локальных изменений (до индексации)

### Цели: Научиться отменять изменения в рабочем каталоге

## 1. Переключитесь на ветку Master

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

#### ВЫПОЛНИТЕ:

git checkout master

## 2. 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>

## 3.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.

## 4. 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.

# Отмена проиндексированных изменений (перед коммитом)

### Цели: Научиться отменять изменения, которые были проиндексированы

## 1Измените 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

## 2Проверьте 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

#

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

## 3Выполните сброс буферной зоны

К счастью, вывод состояния 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.

## 4Переключитесь 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.

# Отмена коммитов

### Цели: Научиться отменять коммиты в локальный репозиторий.

## 1Отмена 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.

## 2Измените 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"

## 3.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.

## 4.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.

## 5Далее

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

# Удаление коммитов 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.

## 1Команда 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

## 2.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.

## 3.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

## 4.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.

## 5. 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.

## 6Опасность 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.

# Удаление тега oops

### Цели: Удаление тега oops (уборка)

## 1Удаление 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.

# Внесение изменений в коммиты

### Цели: Научиться изменять существующие коммиты

## 1.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"

## 2.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>

## 3.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(-)

## 4.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.

# Перемещение файлов

### Цели: Научиться перемещать файл в пределах репозитория.

## 1. 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.

## *2.* 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

## 3. Коммит в новый каталог

Давайте сделаем коммит этого перемещения.

#### ВЫПОЛНИТЕ:

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

# Подробнее о структуре

### Цели: Добавить еще один файл в наш репозиторий

## 1. 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.

# Git внутри: Каталог .git

### Цели: Узнать о структуре каталога .git

## 1. 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.

## 2. 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.

## 3. 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

## 4. 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.

## 5. 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.

## 6. 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.

# Git внутри: Работа непосредственно с объектами git

### Цели

1. Исследовать структуру базы данных объектов
2. Научиться использовать SHA1 хэши для поиска содержимого в репозитории

Давайте исследуем объекты git с помощью некоторых инструментов.

## 1. 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]

## 2. Вывод последнего коммита

С помощью SHA1 хэша из коммита, указанного выше…

#### ВЫПОЛНИТЕ:

git cat-file -t <hash>

git cat-file -p <hash>

#### 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](https://githowto.com/ru/aliases), 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.

## 3. 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.

## 4. Вывод каталога lib

#### ВЫПОЛНИТЕ:

git cat-file -p <libhash>

#### РЕЗУЛЬТАТ:

$ git cat-file -p e46f374

100644 blob c45f26b6fdc7db6ba779fc4c385d9d24fc12cf72 hello.html

Существует файл hello.html.

## 5. 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.

## 6. 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.