

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

## Факультет физико-математических и естественных наук

### Кафедра прикладной информатики и теории вероятностей

#### ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

Дисциплина: Математическое моделирование Выполнили: Нгуен Фыок Дат Номер студенческого билета : 1032195855 Группа: НФИбд-01-20

МОСКВА 2022 г.

**1. Цель работы** Работа с git. **1.1 Подготовка** **1.1.1 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. `git config --global user.name "Your Name"`

```
Administrator@DatZ MINGW64 ~/Desktop
$ git config --global user.name "PhuocDatNguyen"
```

`git config --global user.email "your\_email@whatever.com"`

```
Administrator@DatZ MINGW64 ~/Desktop
$ git config --global user.email "pdat2708@gmail.com"
```

#### 1.1.2 \*\*Параметры установки окончаний строк\*\*

-Настройка `core.autocrlf` с параметрами `true` и `input` делает все переводы строк текстовых файлов в главном репозитории одинаковыми. `core.autocrlf true` - git автоматически конвертирует `CRLF`→`LF` при коммите и обратно `LF`→`CRLF` при выгрузке кода из репозитория на файловую систему (используют в Windows). `core.autocrlf input` - конвертация `CRLF` в `LF` только при коммитах (используют в Mac/Linux). -Если `core.safecrlf` установлен в `true` или `warn`, git проверяет, если преобразование является обратимым для текущей настройки `core.autocrlf`. `core.safecrlf true` - отвержение необратимого преобразования `lf`↔`crlf`. -Полезно, когда специфические бинарники похожие на текстовые файлы. `core.safecrlf warn` - печать только предупреждение, но принимает необратимый переход. -Для пользователей Unix/Mac: `git config --global core.autocrlf input` `git config --global core.safecrlf true` Для пользователей Windows: `git config --global core.autocrlf true`

```
Administrator@DatZ MINGW64 ~/Desktop
$ git config --global core.autocrlf true
```

`git config --global core.safecrlf true`

```
Administrator@DatZ MINGW64 ~/Desktop
$ git config --global core.safecrlf true
```

#### 1.1.3 \*\*Установка отображения unicode\*\*

-По умолчанию, git будет печатать не-ASCII символов в именах файлов в виде восьмеричных последовательностей \nnn. Что бы избежать нечитаемых строк, установите соответствующий флаг. `git config --global core.quoteopath off`

```
Administrator@DatZ MINGW64 ~/Desktop
$ git config --global core.quoteopath off
```

**1.2 Создание проекта**  
**1.2.1 Создайте страницу «Hello, World»** Начните работу в пустом рабочем каталоге с создания пустого каталога с именем hello, затем войдите в него и создайте там файл с именем hello.html. `mkdir hello cd hello touch hello.html echo "Hello, World!" > hello.html`

```
Administrator@DatZ MINGW64 ~/Desktop
$ mkdir hello

Administrator@DatZ MINGW64 ~/Desktop
$ cd hello

Administrator@DatZ MINGW64 ~/Desktop/hello
$ touch hello.html

Administrator@DatZ MINGW64 ~/Desktop/hello
$ echo "Hello, World!" > hello.html
```

#### 1.2.2 \*\*Создание репозитория\*\*

Чтобы создать git репозиторий из этого каталога, выполните команду `git init`. `git init`

```
Administrator@DatZ MINGW64 ~/Desktop/hello
$ git init
Initialized empty Git repository in C:/Users/Administrator/Desktop/hello/.git/
```

#### 1.2.3 \*\*Добавление файла в репозиторий\*\*

Добавим файл в репозиторий. `git add hello.html git commit -m "Initial Commit"`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git add hello.html

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git commit -m "Initial Commit"
[master (root-commit) 091d4cf] Initial Commit
1 file changed, 1 insertion(+)
create mode 100644 hello.html
```

#### 1.2.4 \*\*Проверка состояние репозитория\*\*

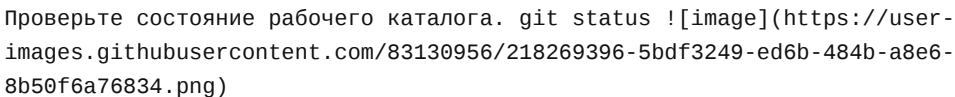
Используйте команду `git status`, чтобы проверить текущее состояние репозитория. `git status`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git status
On branch master
nothing to commit, working tree clean
```

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

**1.3 Внесение изменений** 1.3.1 **Измените страницу «Hello, World»** Добавим кое-какие HTML-теги к нашему приветствию. Измените содержимое файла `hello.html` на:

## Hello, World!

Проверьте состояние рабочего каталога. `git status` 

`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.

**1.4 Индексация изменений** Теперь выполните команду `git`, чтобы проиндексировать изменения. Проверьте состояние. `git add hello.html git status`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git add hello.html

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello.html
```

Изменения файла `hello.html` были проиндексированы. Это означает, что `git` теперь знает об изменении, но изменение пока не записано в репозиторий. Следующий коммит будет включать в себя проиндексированные изменения. Если вы решили, что не хотите коммитить изменения, команда состояния напомним вам о том, что с помощью команды `git reset` можно снять индексацию этих изменений. 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. Разделяя индексацию и коммит, вы имеете возможность с легкостью настроить, что идет в какой коммит.

#### 1.4.1 \*\*Коммит изменений\*\*

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`. Сделайте коммит и проверьте состояние. `git commit` Откроется редактор. 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). Теперь еще раз проверим состояние. `git status` Рабочий каталог чистый, можно продолжить работу.

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git commit
[master 5207fdd] Added h1 tag
1 file changed, 1 insertion(+), 1 deletion(-)

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git status
On branch master
nothing to commit, working tree clean
```

#### 1.4.2 \*\*Добавьте стандартные теги страницы\*\*

Измените страницу «Hello, World», чтобы она содержала стандартные теги и .

## Hello, World!

Теперь добавьте это изменение в индекс `git`. `git add hello.html` ![image](https://user-images.githubusercontent.com/83130956/218269570-999cd260-f2b1-420c-b331-b00d96c11ff7.png)

Теперь добавьте заголовки HTML (секцию ) к странице «Hello, World».

## Hello, World!

Проверьте текущий статус: `git status` ![image](https://user-images.githubusercontent.com/83130956/218269655-8d62596f-afe0-4f3b-a937-f99dbe494dae.png)

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. Произведите коммит проиндексированного изменения (значение по умолчанию), а затем еще раз проверьте состояние. `git commit -m "Added standard HTML page tags" git status`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git commit -m "Added standard HTML page tags"
[master f6d43e9] Added standard HTML page tags
1 file changed, 5 insertions(+), 1 deletion(-)

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <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. Теперь добавьте второе изменение в индекс, а затем проверьте состояние с помощью команды `git status`. `git add . git status`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git add .

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello.html

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git commit -m "Added HTML header"
[master 34693ed] Added HTML header
1 file changed, 2 insertions(+)
```

В качестве файла для добавления, мы использовали текущий каталог (.). Это 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 какойто файл, который добавлять было не нужно. Второе изменение было проиндексировано и готово к коммиту. Сделайте коммит второго изменения `git commit -m "Added HTML header"`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git commit -m "Added HTML header"
[master 34693ed] Added HTML header
1 file changed, 2 insertions(+)
```

Получим список произведенных изменений: `git log`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git log
commit 34693ed17c2e5d8ad58dbd58ec52b391ab74b78c (HEAD -> master)
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:06:48 2023 +0300

    Added HTML header

commit f6d43e9a1f22c53c214e80ff563dceb88385f3c6
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:06:05 2023 +0300

    Added standard HTML page tags

commit 5207fddb416e084a0053b8a3d6f530f1a502f6b8
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:03:30 2023 +0300

    Added h1 tag

commit 091d4cffeb163f0bcfb09d399956e0e1839cc440
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:01:53 2023 +0300

    Initial Commit
```

Однострочный формат истории: `git log --pretty=oneline`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git log --pretty=oneline
34693ed17c2e5d8ad58dbd58ec52b391ab74b78c (HEAD -> master) Added HTML header
f6d43e9a1f22c53c214e80ff563dceb88385f3c6 Added standard HTML page tags
5207fddb416e084a0053b8a3d6f530f1a502f6b8 Added h1 tag
091d4cffeb163f0bcfb09d399956e0e1839cc440 Initial Commit
```

#### 1.4.4 \*\*Получение старых версий\*\*

Возвращаться назад в историю очень просто. Команда `checkout` скопирует любой снимок из репозитория в рабочий каталог. Получите хэши предыдущих версий `git log`

```

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git log
commit 34693ed17c2e5d8ad58dbd58ec52b391ab74b78c (HEAD -> master)
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:06:48 2023 +0300

    Added HTML header

commit f6d43e9a1f22c53c214e80ff563dceb88385f3c6
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:06:05 2023 +0300

    Added standard HTML page tags

commit 5207fddb416e084a0053b8a3d6f530f1a502f6b8
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:03:30 2023 +0300

    Added h1 tag

commit 091d4cffeb163f0bcfb09d399956e0e1839cc440
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:01:53 2023 +0300

    Initial Commit

```

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

```

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git checkout 091d4cf
Note: switching to '091d4cf'.

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 switching back to a branch.

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

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

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

HEAD is now at 091d4cf Initial Commit

Administrator@DatZ MINGW64 ~/Desktop/hello ((091d4cf...))
$ cat hello.html
Hello, World!

```

Вернитесь к последней версии в ветке master `git checkout master` `cat hello.html`

```
Administrator@DatZ MINGW64 ~/Desktop/hello ((091d4cf...))
$ git checkout master
Previous HEAD position was 091d4cf Initial Commit
Switched to branch 'master'

Administrator@DatZ MINGW64 ~/Desktop/hello (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. 1.4.5 **Создание тегов версий** Давайте назовем текущую версию страницы hello первой (v1). Создайте тег первой версии `git tag v1`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git tag v1
```

Теперь текущая версия страницы называется v1. 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». Вместо обозначения v1^ можно использовать v1~1. Это обозначение можно определить как «первую версию предшествующую v1». `git checkout v1^` `cat hello.html`



```

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git checkout v1^
Note: switching to '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 switching back to a branch.

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

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

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

HEAD is now at f6d43e9 Added standard HTML page tags

Administrator@DatZ MINGW64 ~/Desktop/hello ((f6d43e9...))
$ cat hello.html
<html>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>

```

Это версия с тегами и , но еще пока без . Давайте сделаем ее версией v1-beta. `git tag v1-beta`

```

Administrator@DatZ MINGW64 ~/Desktop/hello ((f6d43e9...))
$ git tag v1-beta

```

#### 1.4.6 \*\*Переключение по имени тега\*\*

Теперь попробуйте попереключаться между двумя отмеченными версиями. `git checkout v1`  
`git checkout v1-beta`

```

Administrator@DatZ MINGW64 ~/Desktop/hello ((v1-beta))
$ git checkout v1
Previous HEAD position was f6d43e9 Added standard HTML page tags
HEAD is now at 34693ed Added HTML header

Administrator@DatZ MINGW64 ~/Desktop/hello ((v1))
$ git checkout v1-beta
Previous HEAD position was 34693ed Added HTML header
HEAD is now at f6d43e9 Added standard HTML page tags

```

#### 1.4.7 \*\*Просмотр тегов с помощью команды tag\*\*

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

```
Administrator@DatZ MINGW64 ~/Desktop/hello ((v1-beta))
$ git tag
v1
v1-beta
```

Вы также можете посмотреть теги в логе. `git log master --all`

```
Administrator@DatZ MINGW64 ~/Desktop/hello ((v1-beta))
$ git log master --all
commit 34693ed17c2e5d8ad58dbd58ec52b391ab74b78c (tag: v1, master)
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:06:48 2023 +0300

    Added HTML header

commit f6d43e9a1f22c53c214e80ff563dceb88385f3c6 (HEAD, tag: v1-beta)
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:06:05 2023 +0300

    Added standard HTML page tags

commit 5207fddb416e084a0053b8a3d6f530f1a502f6b8
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:03:30 2023 +0300

    Added h1 tag

commit 091d4cffeb163f0bcfb09d399956e0e1839cc440
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:01:53 2023 +0300

    Initial Commit
```

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).

#### 1.5 Отмена локальных изменений (до индексации) 1.5.1 Переключитесь на ветку master

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

```
Administrator@DatZ MINGW64 ~/Desktop/hello ((v1-beta))
$ git checkout master
Previous HEAD position was f6d43e9 Added standard HTML page tags
Switched to branch 'master'
```

##### 1.5.2 \*\*Измените hello.html\*\*

Иногда случается, что вы изменили файл в рабочем каталоге, и хотите отменить последние коммиты. С этим справится команда `git checkout`. Внесите изменение в файл `hello.html` в виде нежелательного комментария.

# Hello, World!

1.5.3 **\*\*Проверьте состояние\*\*** Сначала проверьте состояние рабочего каталога. `git status` ![image](https://user-images.githubusercontent.com/83130956/218270097-4ef0a26f-f2fb-4cc8-b8fa-14155be71838.png)

Мы видим, что файл `hello.html` был изменен, но еще не проиндексирован. 1.5.4 **Отмена изменений в рабочем каталоге** Используйте команду `git checkout` для переключения версии файла `hello.html` в репозитории. `git checkout hello.html` `git status` `cat hello.html`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git checkout hello.html
Updated 1 path from the index

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git status
On branch master
nothing to commit, working tree clean

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ cat hello.html
<html>
<head>
</head>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

Команда `git status` показывает нам, что не было произведено никаких изменений, не зафиксированных в рабочем каталоге.

1.6 **Отмена проиндексированных изменений (перед коммитом)** 1.6.1 **Измените файл и проиндексируйте изменения** Внесите изменение в файл `hello.html` в виде нежелательного комментария

## Hello, World!

Проиндексируйте это изменение. `git add hello.html` ![image](https://user-images.githubusercontent.com/83130956/218270169-d1b30435-db32-439d-834c-ee01fcf0b920.png)

1.6.2 **Проверьте состояние** Проверьте состояние нежелательного изменения. `git status`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   hello.html
```

Состояние показывает, что изменение было проиндексировано и готово к коммиту. 1.6.3 Выполните сброс буферной зоны К счастью, вывод состояния показывает нам именно то, что

мы должны сделать для отмены индексации изменения. `git reset HEAD hello.html`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git reset HEAD hello.html
Unstaged changes after reset:
M      hello.html
```

Команда `git reset` сбрасывает буферную зону к HEAD. Это очищает буферную зону от изменений, которые мы только что проиндексировали. Команда `git reset` (по умолчанию) не изменяет рабочий каталог. Поэтому рабочий каталог все еще содержит нежелательный комментарий. Мы можем использовать команду `git checkout`, чтобы удалить нежелательные изменения в рабочем каталоге. 1.6.4 Переключитесь на версию коммита `git checkout hello.html` `git status` Наш рабочий каталог опять чист.

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git checkout hello.html
Updated 1 path from the index

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git status
On branch master
nothing to commit, working tree clean
```

**1.7 Отмена коммитов** 1.7.1 **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. 1.7.2 **Измените файл и сделайте коммит** Измените файл `hello.html` на следующий.

## Hello, World!

Выполните: `git add hello.html` `git commit -m "Oops, we didn't want this commit" !`  
[image](https://user-images.githubusercontent.com/83130956/218270233-d98bc995-a028-4e53-8114-d3e9ffe2a216.png)

1.7.3 Сделайте коммит с новыми изменениями, отменяющими предыдущие Чтобы отменить коммит, нам необходимо сделать коммит, который удаляет изменения, сохраненные нежелательным коммитом. `git revert HEAD`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git revert HEAD
[master 2382aa5] Revert "Oops, we didn't want this commit"
1 file changed, 1 deletion(-)
```

Перейдите в редактор, где вы можете отредактировать коммит-сообщение по умолчанию или оставить все как есть. Сохраните и закройте файл. 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. 1.7.4 Проверьте лог Проверка лога показывает нежелательные и отмененные коммиты в наш

репозиторий. `git log`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git log
commit 2382aa522f0aa63956fa26342a46970baa3bf74d (HEAD -> master)
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:15:42 2023 +0300

    Revert "Oops, we didn't want this commit"

    This reverts commit 6711158089099390772ebc284c3e8e54bd1ee277.

commit 6711158089099390772ebc284c3e8e54bd1ee277
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:15:21 2023 +0300

    Oops, we didn't want this commit

commit 34693ed17c2e5d8ad58dbd58ec52b391ab74b78c (tag: v1)
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:06:48 2023 +0300

    Added HTML header

commit f6d43e9a1f22c53c214e80ff563dceb88385f3c6 (tag: v1-beta)
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:06:05 2023 +0300

    Added standard HTML page tags

commit 5207fddb416e084a0053b8a3d6f530f1a502f6b8
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:03:30 2023 +0300

    Added h1 tag

commit 091d4cffeb163f0bcfb09d399956e0e1839cc440
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:01:53 2023 +0300

    Initial Commit
```

Эта техника будет работать с любым коммитом. 1.8 **Удаление коммитов из ветки** `git revert` является мощной командой, которая позволяет отменить любые коммиты в репозитории. 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`. 1.8.1 **Команда `git reset`** При получении ссылки на коммит (т.е. хэш, ветка или имя тега), команда `git reset`: • переписывает текущую ветку, чтобы она указывала на нужный коммит; • опционально сбросит буферную зону для соответствия с указанным коммитом; • опционально сбросит рабочий каталог для соответствия с указанным коммитом. 1.8.2 **Проверьте нашу историю**

Давайте сделаем быструю проверку нашей истории коммитов. Выполните: `git log`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git log
commit 2382aa522f0aa63956fa26342a46970baa3bf74d (HEAD -> master)
Author: PhuocDatNguyen <pdatt2708@gmail.com>
Date: Sat Feb 11 18:15:42 2023 +0300

    Revert "Oops, we didn't want this commit"

    This reverts commit 6711158089099390772ebc284c3e8e54bd1ee277.

commit 6711158089099390772ebc284c3e8e54bd1ee277
Author: PhuocDatNguyen <pdatt2708@gmail.com>
Date: Sat Feb 11 18:15:21 2023 +0300

    Oops, we didn't want this commit

commit 34693ed17c2e5d8ad58dbd58ec52b391ab74b78c (tag: v1)
Author: PhuocDatNguyen <pdatt2708@gmail.com>
Date: Sat Feb 11 18:06:48 2023 +0300

    Added HTML header

commit f6d43e9a1f22c53c214e80ff563dceb88385f3c6 (tag: v1-beta)
Author: PhuocDatNguyen <pdatt2708@gmail.com>
Date: Sat Feb 11 18:06:05 2023 +0300

    Added standard HTML page tags

commit 5207fddb416e084a0053b8a3d6f530f1a502f6b8
Author: PhuocDatNguyen <pdatt2708@gmail.com>
Date: Sat Feb 11 18:03:30 2023 +0300

    Added h1 tag

commit 091d4cffeb163f0bcfb09d399956e0e1839cc440
Author: PhuocDatNguyen <pdatt2708@gmail.com>
Date: Sat Feb 11 18:01:53 2023 +0300

    Initial Commit
```

Мы видим, что два последних коммита в этой ветке – «Oops» и «Revert Oops». Давайте удалим их с помощью сброса. 1.8.3 **Для начала отметьте эту ветку** Но прежде чем удалить коммиты, давайте отметим последний коммит тегом, чтобы потом можно было его найти. `git`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git tag oops
```

`git tag oops`

1.8.4

**Сброс коммитов к предшествующим коммиту Oops** Глядя на историю лога, 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). `git`

```
reset --hard v1 git log
```

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git reset --hard v1
HEAD is now at 34693ed Added HTML header

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git log
commit 34693ed17c2e5d8ad58dbd58ec52b391ab74b78c (HEAD -> master, tag: v1)
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:06:48 2023 +0300

    Added HTML header

commit f6d43e9a1f22c53c214e80ff563dceb88385f3c6 (tag: v1-beta)
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:06:05 2023 +0300

    Added standard HTML page tags

commit 5207fddb416e084a0053b8a3d6f530f1a502f6b8
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:03:30 2023 +0300

    Added h1 tag

commit 091d4cffeb163f0bcfb09d399956e0e1839cc440
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:01:53 2023 +0300

    Initial Commit
```

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. 1.8.5 **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. git log

--all

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git log --all
commit 2382aa522f0aa63956fa26342a46970baa3bf74d (tag: oops)
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:15:42 2023 +0300

    Revert "Oops, we didn't want this commit"

    This reverts commit 6711158089099390772ebc284c3e8e54bd1ee277.

commit 6711158089099390772ebc284c3e8e54bd1ee277
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:15:21 2023 +0300

    Oops, we didn't want this commit

commit 34693ed17c2e5d8ad58dbd58ec52b391ab74b78c (HEAD -> master, tag: v1)
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:06:48 2023 +0300

    Added HTML header

commit f6d43e9a1f22c53c214e80ff563dceb88385f3c6 (tag: v1-beta)
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:06:05 2023 +0300

    Added standard HTML page tags

commit 5207fddb416e084a0053b8a3d6f530f1a502f6b8
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:03:30 2023 +0300

    Added h1 tag

commit 091d4cffeb163f0bcfb09d399956e0e1839cc440
Author: PhuocDatNguyen <pdat2708@gmail.com>
Date: Sat Feb 11 18:01:53 2023 +0300

    Initial Commit
```

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.

### 1.8.6 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.

### 1.9 Удаление тега oops

#### 1.9.1 Удаление тега oops

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



мысopa. git tag -d oops git log --all

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git tag -d oops
Deleted tag 'oops' (was 2382aa5)

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git log --all
commit 34693ed17c2e5d8ad58dbd58ec52b391ab74b78c (HEAD -> master, tag: v1)
Author: PhuocDatNguyen <pdatt2708@gmail.com>
Date: Sat Feb 11 18:06:48 2023 +0300

    Added HTML header

commit f6d43e9a1f22c53c214e80ff563dceb88385f3c6 (tag: v1-beta)
Author: PhuocDatNguyen <pdatt2708@gmail.com>
Date: Sat Feb 11 18:06:05 2023 +0300

    Added standard HTML page tags

commit 5207fddb416e084a0053b8a3d6f530f1a502f6b8
Author: PhuocDatNguyen <pdatt2708@gmail.com>
Date: Sat Feb 11 18:03:30 2023 +0300

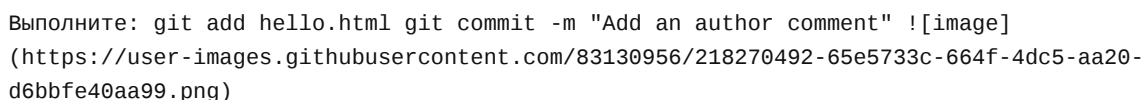
    Added h1 tag

commit 091d4cffeb163f0bcfb09d399956e0e1839cc440
Author: PhuocDatNguyen <pdatt2708@gmail.com>
Date: Sat Feb 11 18:01:53 2023 +0300

    Initial Commit
```

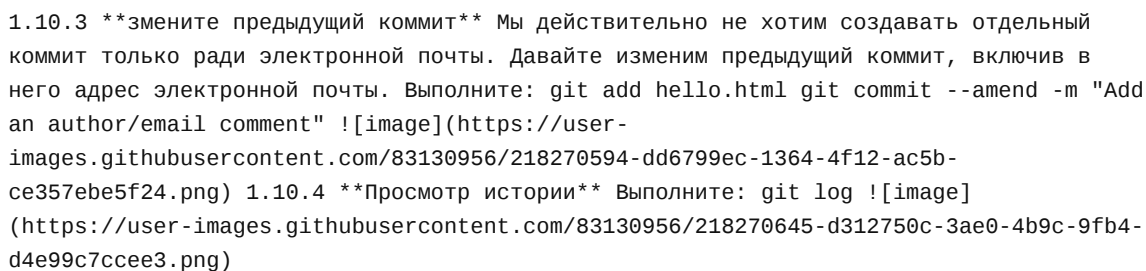
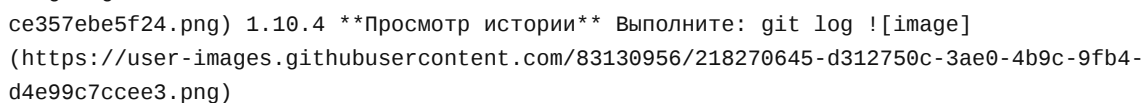
Тег «oops» больше не будет отображаться в репозитории. 1.10 **Внесение изменений в коммиты** 1.10.1 **Измените страницу, а затем сделайте коммит** Добавьте в страницу комментарий автора (вставьте свою фамилию).

## Hello, World!

Выполните: `git add hello.html git commit -m "Add an author comment" !`  
(<https://user-images.githubusercontent.com/83130956/218270492-65e5733c-664f-4dc5-aa20-d6bbfe40aa99.png>)

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

## Hello, World!

1.10.3 **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" !`  
(<https://user-images.githubusercontent.com/83130956/218270594-dd6799ec-1364-4f12-ac5b-ce357ebe5f24.png>) 1.10.4 **Просмотр истории** Выполните: `git log !`  
(<https://user-images.githubusercontent.com/83130956/218270645-d312750c-3ae0-4b9c-9fb4-d4e99c7ccee3.png>)

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.

### 1.11 Перемещение файлов

#### 1.11.1 Переместите файл `hello.html` в каталог `lib`

Сейчас мы собираемся создать структуру нашего репозитория. Давайте перенесем страницу в каталог `lib`. `mkdir lib git mv hello.html lib git status`

```
Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ mkdir lib

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git mv hello.html lib

Administrator@DatZ MINGW64 ~/Desktop/hello (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:    hello.html -> lib/hello.html
```

Перемещая файлы с помощью `git mv`, мы информируем `git` о 2 вещах: • Что файл `hello.html` был удален. • 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.

### 1.12 Второй способ перемещения файлов

Положительной 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`? 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`

#### 1.12.1 Коммит в новый каталог

Давайте сделаем коммит этого перемещения: `git commit -m "Moved hello.html to lib"`

### 1.13 Подробнее о структуре

#### 1.13.1 Добавление `index.html`

Добавим файл `index.html` в наш репозиторий

404

The requested path could not be found