

Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и  
радиоэлектроники»

Кафедра инженерной психологии и эргономики

**Пользовательские интерфейсы информационных систем**

Отчет по практическим занятиям на тему  
«Образовательный курс GitHowTo»

Выполнила:  
студентка гр.  
210901  
Маслова Ю.Ю.

Проверил:  
Давыдович К. И.

Минск 2024

# Часть 1. Основы git

## Финальные приготовления

Репозиторий Git — это хранилище, в котором расположен ваш проект и его история. Это может быть локальное хранилище где-то на вашем компьютере или удаленное хранилище на сервисе типа GitHub или другом хостинге в Интернете. Репозиторий служит для отслеживания изменений в проекте, координации работы между несколькими людьми и отслеживания истории проекта.

Вы можете думать о коммите как о снимке вашего проекта в определенный момент времени. Правда, коммит содержит только информацию об изменениях, которые были внесены в репозиторий с момента последнего коммита. Он не содержит все файлы репозитория (если только это не первый коммит). Таким образом, каждый коммит — это небольшой кусочек истории репозитория, основанный на предыдущем коммите. Все они связаны между собой в цепочку, формируя историю изменений вашего проекта.

Ветка — это параллельная версия репозитория. Ветки позволяют вам работать над отдельными функциями вашего проекта, не влияя на основную версию. Закончив работу над новой фичей, вы можете объединить эту ветку с основной версией проекта.

Если вы никогда ранее не использовали Git, для начала вам необходимо осуществить установку. Выполните следующие команды, чтобы Git узнал ваше имя и электронную почту. Эти данные используются для подписи изменений сделанных вами, что позволит отслеживать, кто и когда сделал изменения в файле.

```
glush@DESKTOP-8RBE01G MINGW64 ~  
$ git config --global user.name
```

```
glush@DESKTOP-8RBE01G MINGW64 ~  
$ git config --global user.email
```

Мы будем использовать main в качестве имени ветки по умолчанию. Чтобы установить его, выполните следующую команду:

```
glush@DESKTOP-8RBE01G MINGW64 ~  
$ git config --global init.defaultBranch main
```

Корректная обработка окончаний строк:

```
glush@DESKTOP-8RBE01G MINGW64 ~  
$  
git config --global core.autocrlf true  
git config --global core.safecrlf warn
```

## Создание проекта

Начните работу в пустой директории (например, repositories, если вы скачали архив с предыдущего шага) с создания пустой поддиректории work, затем войдите в неё и создайте там файл hello.html с таким содержанием:

```
glush@DESKTOP-8RBE01G MINGW64 ~  
$ cd D:/githowto/repositories  
  
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories  
$ mkdir work  
  
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories  
$ cd work  
  
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work  
$ touch hello.html
```

Теперь у вас есть директория с одним файлом. Чтобы создать Git-репозиторий из этой директории, выполните команду git init.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work  
$ git init  
Initialized empty Git repository in D:/githowto/repositories/work/.git/
```

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

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git add hello.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git commit -m "Initial Commit"
[main (root-commit) 664b7eb] Initial Commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 hello.html
```

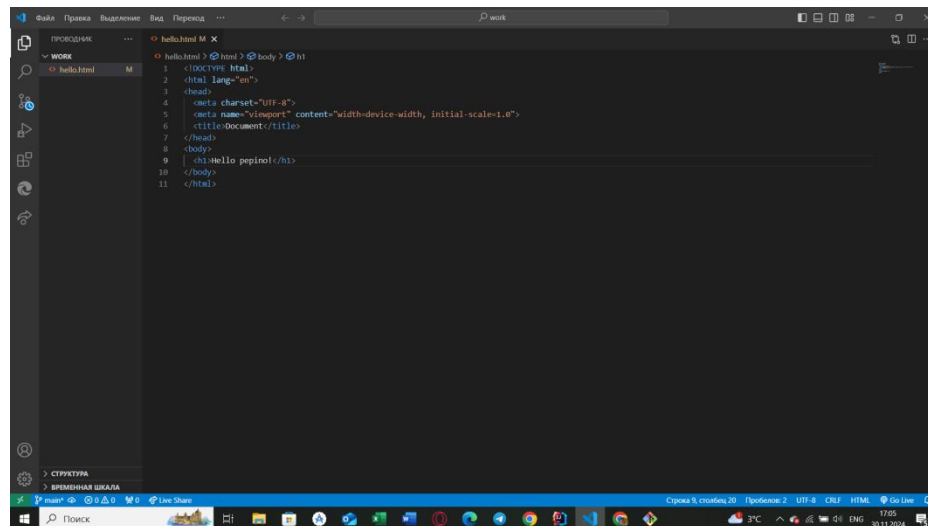
## Проверка состояния

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

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git status
On branch main
nothing to commit, working tree clean
```

## Внесение изменений

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



Теперь проверьте состояние рабочей директории.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git status
On branch main
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")
```

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.

## Индексация изменений

Теперь дайте команду Git проиндексировать изменения. Проверьте состояние:

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git add hello.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git status
On branch main
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 *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.

## Раздельная индексация

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ touch a.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ touch b.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ touch c.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git add a.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git add b.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git commit -m "Changes only for a and b"
[main 48f824a] Changes only for a and b
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a.html
 create mode 100644 b.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git add c.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git commit -m "Unrelated change for c"
[main dd8515d] Unrelated change for c
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 c.html
```

## 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`





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.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git commit -m "Added standard HTML page tags"
[main ed71770] Added standard HTML page tags
1 file changed, 2 insertions(+)

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git status
On branch main
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")
```

Теперь добавьте второе изменение в индекс, а затем проверьте состояние с помощью команды `git status`. Мы использовали текущую директорию (`.`) в качестве аргумента для добавления. Это самый короткий и удобный способ добавления всех изменений в текущей директории. Но поскольку Git добавляет в индекс *всё*, то *не лишним* будет проверить состояние репозитория перед запуском `add`, просто чтобы убедиться, что вы не добавили какой-то файл, который не следовало бы добавлять.



```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git add .

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git commit -m "Added HTML header"
[main 70fc9bc] Added HTML header
1 file changed, 1 deletion(-)
```

## История

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

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git log
commit 70fc9bcd240613d846d3edec0cfb5d20aae14829 (HEAD -> main)
Author: pepino <hanna.hlushko.2005@gmail.com>
Date:   Sat Nov 30 17:25:20 2024 +0300

    Added HTML header

commit ed717706df379557b3e46446b6a9696471a3ed98
Author: pepino <hanna.hlushko.2005@gmail.com>
Date:   Sat Nov 30 17:23:23 2024 +0300

    Added standard HTML page tags

commit c5208d5a917d5658c30bb11b857adf8c503c5473
Author: pepino <hanna.hlushko.2005@gmail.com>
Date:   Sat Nov 30 17:17:52 2024 +0300

    added h1 tag

commit dd8515d0b33bcc44bef49b7ea811ba414f7ff625
Author: pepino <hanna.hlushko.2005@gmail.com>
Date:   Sat Nov 30 17:14:03 2024 +0300

    Unrelated change for c
```

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

Давайте рассмотрим его в деталях:

- `--pretty="..."` — определяет формат вывода.
- `%h` — укороченный хеш коммита.
- `%ad` — дата коммита.
- `|` — просто визуальный разделитель.
- `%s` — комментарий.
- `%d` — дополнения коммита («головы» веток или теги).
- `%an` — имя автора.
- `--date=short` — сохраняет формат даты коротким и симпатичным.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git log --pretty=format:"%h %ad | %s%d [%an]" --date=short
70fc9bc 2024-11-30 | Added HTML header (HEAD -> main) [pepino]
ed71770 2024-11-30 | Added standard HTML page tags [pepino]
c5208d5 2024-11-30 | added h1 tag [pepino]
dd8515d 2024-11-30 | Unrelated change for c [pepino]
48f824a 2024-11-30 | Changes only for a and b [pepino]
de6367a 2024-11-30 | Changes for a and b [pepino]
664b7eb 2024-11-30 | Initial Commit [pepino]
```

Таким образом, каждый раз, когда вы захотите посмотреть лог, вам придется много печатать. К счастью, существует несколько опций конфигурации Git, позволяющих настроить формат вывода истории по умолчанию:

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git config --global format.pretty '%h %ad | %s%d [%an]'
git config --global log.date short
```

## Получение старых версий

Git позволяет очень просто путешествовать во времени, по крайней мере, для вашего проекта. Команда `checkout` обновит вашу рабочую директорию до любого предыдущего коммита.

Просмотрите историю изменений и найдите хеш первого коммита. Он должен быть в последней строке результата `git log`. Используйте этот хеш (достаточно первых 7 символов) в команде ниже. Затем проверьте содержимое файла `hello.html`.

```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work ((664b7eb...))
$ git checkout c5208d5
Previous HEAD position was 664b7eb Initial Commit
HEAD is now at c5208d5 added h1 tag

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work ((c5208d5...))
$ cat hello.html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Hello pepino!!</h1>
</body>
</html>
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work ((c5208d5...))
$

```

Команда `checkout` в течение длительного времени была своеобразным швейцарским ножом в мире Git. Она имеет множество различных опций, которые позволяют выполнять совершенно разные вещи: переключать ветки, сбрасывать код и так далее. В какой-то момент команда Git решила разделить ее на несколько команд. Команда `switch` является одной из них — ее единственным назначением является переключение между ветками. Команда `checkout` все еще доступна, но использовать ее для переключения веток больше не рекомендуется.

```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work ((701c9bc...))
$ git switch main
Switched to branch 'main'

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ cat hello.html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Hello pepino!!</h1>
  <h2>pupu</h2>
</body>
</html>

```

Думаю, вы согласитесь, что работать с хешами коммитов напрямую просто неудобно. Разве не было бы здорово, если бы вы могли обозначать конкретные коммиты понятными для человека



названиями? Таким образом, вы могли бы четко видеть важные вехи в истории проекта. Кроме того, вы могли бы легко перейти к определенной версии проекта по ее названию. Именно для этого в Git придумали теги.

## Создание тегов версий

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work ((c5208d5...))
$ git tag v1

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work ((v1))
$ git log
c5208d5 2024-11-30 | added h1 tag (HEAD, tag: v1) [pepino]
dd8515d 2024-11-30 | Unrelated change for c [pepino]
48f824a 2024-11-30 | Changes only for a and b [pepino]
de6367a 2024-11-30 | Changes for a and b [pepino]
564b7eb 2024-11-30 | Initial Commit [pepino]
```

Обозначим версию, предшествующую текущей, названием v1-beta. Прежде всего, мы переключимся на предыдущую версию. Вместо того чтобы искать хеш коммита, мы будем использовать обозначение ^, а именно v1^, указывающее на коммит, предшествующий v1.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work ((v1))
$ git checkout v1~1
Previous HEAD position was c5208d5 added h1 tag
HEAD is now at dd8515d Unrelated change for c

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work ((dd8515d...))
$ cat hello.html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Hello pepino!</h1>
</body>
</html>
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work ((dd8515d...))
```

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work ((dd8515d...))
$ git tag v1-beta

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work ((v1-beta))
$ git log
dd8515d 2024-11-30 | Unrelated change for c (HEAD, tag: v1-beta) [pepino]
48f824a 2024-11-30 | Changes only for a and b [pepino]
de6367a 2024-11-30 | Changes for a and b [pepino]
664b7eb 2024-11-30 | Initial Commit [pepino]
```

Теперь попробуйте переключаться между двумя отмеченными версиями.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work ((v1-beta))
$ git checkout v1
Previous HEAD position was dd8515d Unrelated change for c
HEAD is now at c5208d5 added h1 tag

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work ((v1))
$ git checkout v1-beta
Previous HEAD position was c5208d5 added h1 tag
HEAD is now at dd8515d Unrelated change for c
```

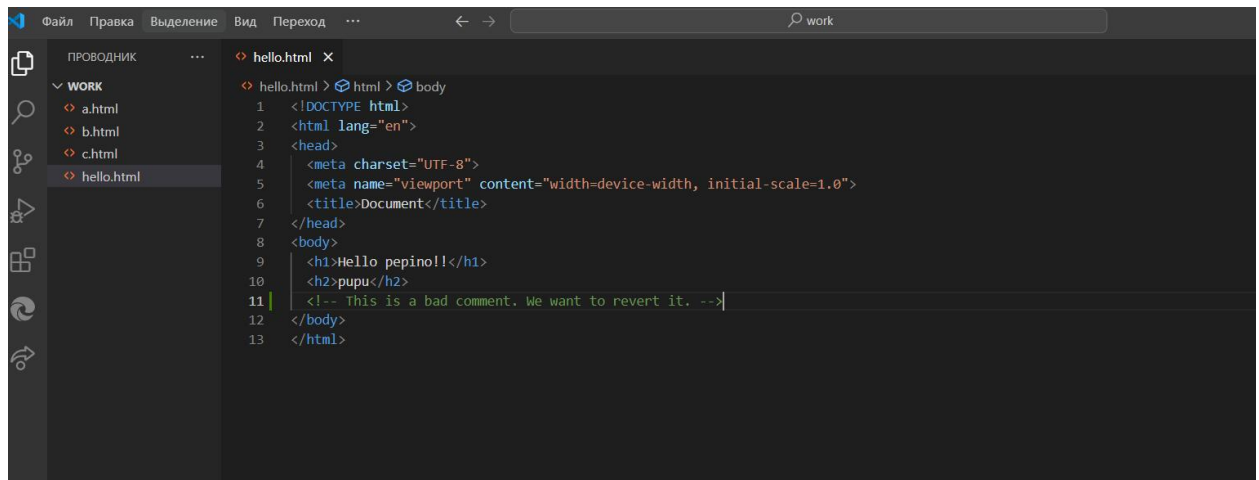
Вы можете увидеть, какие теги доступны, используя команду `git tag`.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work ((v1-beta))
$ git tag
v1
v1-beta
```

Вы также можете посмотреть теги в логе.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work ((v1-beta))
$ git log main --all
70fc9bc 2024-11-30 | Added HTML header (main) [pepino]
ed71770 2024-11-30 | Added standard HTML page tags [pepino]
c5208d5 2024-11-30 | added h1 tag (tag: v1) [pepino]
dd8515d 2024-11-30 | Unrelated change for c (HEAD, tag: v1-beta) [pepino]
48f824a 2024-11-30 | Changes only for a and b [pepino]
de6367a 2024-11-30 | Changes for a and b [pepino]
664b7eb 2024-11-30 | Initial Commit [pepino]
```

## Отмена локальных изменений (до индексации)



```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git checkout hello.html
Updated 1 path from the index

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git status
On branch main
nothing to commit, working tree clean

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ cat hello.html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Hello pepino!!</h1>
  <h2>pupu</h2>
</body>
</html>
```

Команда status показывает нам, что в рабочей директории не было сделано никаких незафиксированных изменений. И «нежелательный комментарий» больше не является частью содержимого файла.

### Отмена проиндексированных изменений (перед коммитом)

Внесите изменение в файл hello.html в виде нежелательного комментария



```
hello.html M X
hello.html > html > head
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 | <!-- This is an unwanted but staged comment -->
5 <meta charset="UTF-8">
6 <meta name="viewport" content="width=device-width, initial-scale=1.0">
7 <title>Document</title>
8 </head>
9 <body>
10 <h1>Hello pepino!!</h1>
11 <h2>pupu</h2>
12 </body>
13 </html>
```

Команда `reset` (по умолчанию) не изменяет рабочую директорию. Поэтому рабочая директория всё еще содержит нежелательный комментарий. Мы можем использовать команду `checkout` из предыдущего урока, чтобы убрать нежелательные изменения в рабочей директории.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git add hello.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   hello.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git reset HEAD hello.html
Unstaged changes after reset:
M    hello.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git checkout hello.html
Updated 1 path from the index

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git status
On branch main
nothing to commit, working tree clean
```

## 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.

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

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git add hello.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git commit -m "oops, we didn't want this commit"
[main 329e222] oops, we didn't want this commit
1 file changed, 1 insertion(+)

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git revert HEAD
[main d5940de] Revert "oops, we didn't want this commit"
1 file changed, 1 deletion(-)

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git log
d5940de 2024-11-30 | Revert "oops, we didn't want this commit" (HEAD -> main)
[pepino]
829e222 2024-11-30 | oops, we didn't want this commit [pepino]
70fc9bc 2024-11-30 | Added HTML header [pepino]
ed71770 2024-11-30 | Added standard HTML page tags [pepino]
c5208d5 2024-11-30 | added h1 tag (tag: v1) [pepino]
dd8515d 2024-11-30 | Unrelated change for c (tag: v1-beta) [pepino]
48f824a 2024-11-30 | Changes only for a and b [pepino]
de6367a 2024-11-30 | Changes for a and b [pepino]
564b7eb 2024-11-30 | Initial Commit [pepino]
```

## Удаление коммитов 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.

Мы уже видели команду `reset` и использовали ее для согласования области подготовки с выбранным коммитом (в предыдущем уроке мы использовали коммит `HEAD`).

Если выполнить команду `reset` с указанием ссылки на коммит (т.е. метки `HEAD`, имени ветки или тега, хеша коммита), то команда...

1. Изменит текущую ветку, чтобы она указывала на указанный коммит.
2. Опционально сбросит область подготовки до соответствия с указанным коммитом.
3. Опционально сбросит рабочую директорию до соответствия с указанным коммитом.

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.

Мы видим, что ошибочные коммиты не исчезли. Они все еще находятся в репозитории. Просто они отсутствуют в ветке `main`. 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 хешей этих коммитов. Коммиты, на которые нет ссылок, остаются в репозитории до тех пор, пока не будет запущен сборщик мусора.

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

Однако, если ветка уже стала общедоступной на удаленных репозиториях, сброс может сбить с толку других пользователей ветки.

```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git tag oops

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git reset --hard v1
HEAD is now at c5208d5 added h1 tag

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git log
c5208d5 2024-11-30 | added h1 tag (HEAD -> main, tag: v1) [pepino]
dd8515d 2024-11-30 | Unrelated change for c (tag: v1-beta) [pepino]
48f824a 2024-11-30 | Changes only for a and b [pepino]
de6367a 2024-11-30 | Changes for a and b [pepino]
664b7eb 2024-11-30 | Initial Commit [pepino]

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git log --all
d5940de 2024-11-30 | Revert "oops, we didn't want this commit" (tag: oops) [pepino]
329e222 2024-11-30 | oops, we didn't want this commit [pepino]
70fc9bc 2024-11-30 | Added HTML header [pepino]
ed71770 2024-11-30 | Added standard HTML page tags [pepino]
c5208d5 2024-11-30 | added h1 tag (HEAD -> main, tag: v1) [pepino]
dd8515d 2024-11-30 | Unrelated change for c (tag: v1-beta) [pepino]
48f824a 2024-11-30 | Changes only for a and b [pepino]
de6367a 2024-11-30 | Changes for a and b [pepino]
664b7eb 2024-11-30 | Initial Commit [pepino]

```

## Удаление тега oops

Тег oops свою функцию выполнил, давайте удалим его. Это позволит внутреннему механизму Git убрать остаточные коммиты, на которые теперь не ссылаются никакие ветки или теги.

```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git tag -d oops
Deleted tag 'oops' (was d5940de)

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git log -all
error: switch `l' expects a numerical value

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git log --all
c5208d5 2024-11-30 | added h1 tag (HEAD -> main, tag: v1) [pepino]
dd8515d 2024-11-30 | Unrelated change for c (tag: v1-beta) [pepino]
48f824a 2024-11-30 | Changes only for a and b [pepino]
de6367a 2024-11-30 | Changes for a and b [pepino]
664b7eb 2024-11-30 | Initial Commit [pepino]

```

## Внесение изменений в коммиты



Добавьте в страницу комментарий автора.

```
1 <!DOCTYPE html>
2 | <!-- Author: Alexander Shvets -->
3 <html lang="en">
4 <head>
5 |   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Document</title>
8 </head>
9 <body>
10 |   <h1>Hello pepino!!</h1>
11 </body>
12 </html>
```

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git add hello.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git commit -m "Added copyright statement"
[main d0a97f1] Added copyright statement
1 file changed, 1 insertion(+)

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git log
d0a97f1 2024-11-30 | Added copyright statement (HEAD -> main) [pepino]
c5208d5 2024-11-30 | added h1 tag (tag: v1) [pepino]
dd8515d 2024-11-30 | Unrelated change for c (tag: v1-beta) [pepino]
48f824a 2024-11-30 | Changes only for a and b [pepino]
de6367a 2024-11-30 | Changes for a and b [pepino]
664b7eb 2024-11-30 | Initial Commit [pepino]
```

Однако после создания коммита вы понимаете, что любой хороший комментарий должен включать электронную почту автора. Обновите страницу hello.html, включив в нее email.

```

<> hello.html > ...
1  <!DOCTYPE html>
2  <!-- Author: Alexander Shvets (pepino@gmail.com) -->
3  <html lang="en">
4  <head>
5  |   <meta charset="UTF-8">
6  |   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7  |   <title>Document</title>
8  </head>
9  <body>
10 |   <h1>Hello pepino!!</h1>
11 </body>
12 </html>

```

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

```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git add hello.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git commit --amend -m "Added copyright statement with email"
[main bd1b601] Added copyright statement with email
Date: Sat Nov 30 20:15:11 2024 +0300
1 file changed, 1 insertion(+)

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git log
bd1b601 2024-11-30 | Added copyright statement with email (HEAD -> main) [pepi
o]
c5208d5 2024-11-30 | added h1 tag (tag: v1) [pepino]
dd8515d 2024-11-30 | Unrelated change for c (tag: v1-beta) [pepino]
48f824a 2024-11-30 | Changes only for a and b [pepino]
de6367a 2024-11-30 | Changes for a and b [pepino]
664b7eb 2024-11-30 | Initial Commit [pepino]

```

## Создание ветки

Разработка новой функциональности всегда связана с риском: разработка может занять много времени, вы можете в конечном итоге отказаться от неё и т. д. По этой причине лучше всего изолировать разработку фичи в отдельной ветке. Когда фича будет готова, вы сможете слить эту ветку с веткой main. До того времени ветка main будет защищена от рискованного и непроверенного кода. Кроме того, вы можете работать над несколькими фичами параллельно, над каждой в собственной ветке. Вы также можете в любой момент вносить изменения в ветке main, например, чтобы



исправить ошибку в стабильном коде. Пришло время сделать нашу страницу более стильной с помощью CSS. Мы будем развивать эту возможность в новой ветке под названием style.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git switch -c style
Switched to a new branch 'style'

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git status
On branch style
nothing to commit, working tree clean
```

```
# style.css > h1
1  h1 {
2    color: red;
3  }
```

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ touch style.css

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git add style.css

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git commit -m "Added css stylesheet"
[style a42143a] Added css stylesheet
1 file changed, 3 insertions(+)
create mode 100644 style.css
```

```
<> hello.html > html > head > link
1  <!DOCTYPE html>
2  <!-- Author: Alexander Shvets (pepino@gmail.com) -->
3  <html lang="en">
4  <head>
5    <meta charset="UTF-8">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>Document</title>
8    <link type="text/css" rel="stylesheet" media="all" href="style.css" />
9  </head>
10 <body>
11   <h1>Hello pepino!!</h1>
12 </body>
13 </html>
```

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git add hello.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git commit -m "Included stylesheet into hello.html"
[style 24c5b60] Included stylesheet into hello.html
1 file changed, 1 insertion(+)
```

## Переключение веток

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

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git switch main
Switched to branch 'main'

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ cat hello.html
<!DOCTYPE html>
<!-- Author: Alexander Shvets (pepino@gmail.com) -->
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Hello pepino!!</h1>
</body>
</html>
```

## Перемещение файлов

Я доволен нашими CSS-изменениями, но есть только один момент, который я хотел бы решить до того, как мы объединим наши изменения с `main`. Давайте переименуем файл `hello.html` в `index.html`. Также давайте перенесем наш файл стилей в специально отведенную директорию `css`.

Git позволяет просматривать историю изменений конкретного файла. Давайте посмотрим историю изменений файла `hello.html` перед его переименованием.

```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git log hello.html
24c5b60 2024-11-30 | Included stylesheet into hello.html (HEAD -> style) [pepino]
bd1b601 2024-11-30 | Added copyright statement with email (main) [pepino]
c5208d5 2024-11-30 | added h1 tag (tag: v1) [pepino]
de6367a 2024-11-30 | Changes for a and b [pepino]
664b7eb 2024-11-30 | Initial Commit [pepino]

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git log style.css
a42143a 2024-11-30 | Added css stylesheet [pepino]

```

Возможность просмотра истории изменений для конкретного файла очень полезна. Она позволяет увидеть, что именно изменилось, а также кто и когда внес эти изменения. Кроме того, существует возможность увидеть изменения, связанные с конкретным коммитом. Я постоянно пользуюсь этим, чтобы разобраться, почему та или иная штука была реализована именно так в настоящей версии кода.

Команда `show` используется для просмотра изменений в конкретном коммите. Посмотрим изменения в файле `hello.html` в коммите, с тегом `v1` (можно использовать любую ссылку на коммит, например, метку `HEAD`, хеш коммита, имя ветки или тега и т.д.).

```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git show v1
c5208d5 2024-11-30 | added h1 tag (tag: v1) [pepino]

diff --git a/hello.html b/hello.html
index 5e67cb3..951b4ca 100644
--- a/hello.html
+++ b/hello.html
@@ -6,6 +6,6 @@
     <title>Document</title>
   </head>
   <body>
-    <h1>Hello pepino!</h1>
+    <h1>Hello pepino!!</h1>
   </body>
 </html>
\ No newline at end of file

```

Как видите, очень удобно иметь возможность видеть историю изменений конкретного файла. Но когда вы переименовываете или перемещаете какой-либо файл, есть риск потерять историю этого файла, если вы выполните эту процедуру неправильно.

Давайте переименуем наш файл `hello.html` в `index.html` с помощью стандартной команды `mv` и посмотрим, что из этого получится.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ mv hello.html index.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git status
On branch style
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    hello.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

Git воспринимает наше изменение так, будто файл был удален и создан заново. Это тревожный звоночек. Нам нужно сообщить Git, что мы именно переименовали файл, а не удалили его и сразу создали новый. В простейшем случае Git сам поймёт, что файл был переименован, как только мы добавим его в индекс:

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git add .

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git status
On branch style
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    hello.html -> index.html
```

Видите, файл указан как переименованный. Но это всего лишь Git пытается проявить смекалку. Это не всегда работает. Например, если вы переименовали, **а также** изменили кучу файлов, Git может оказаться не в состоянии понять, что именно было переименовано. В этом случае вы можете потерять информацию об истории файлов до их переименования, поскольку файлы будут восприняты как новые.

В большинстве операционных систем переименование и перемещение файлов — это одно и то же. Итак, давайте переместим



наш файл `style.css` в директорию `css`, но на этот раз сделаем это безопасно с помощью команды `git mv`. Эта команда гарантирует, что перемещение будет записано в истории Git как перемещение.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ mkdir css

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git mv style.css css/style.css

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git status
On branch style
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    style.css -> css/style.css
        renamed:    hello.html -> index.html
```

Давайте закоммитим наши изменения и проверим историю изменений в файле `css/styles.css`. Для просмотра истории файла до его перемещения нам потребуется добавить опцию `--follow`. Выполним оба варианта команды, чтобы понять разницу.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git commit -m "Renamed hello.html; moved style.css"
[style c6976da] Renamed hello.html; moved style.css
2 files changed, 0 insertions(+), 0 deletions(-)
rename style.css => css/style.css (100%)
rename hello.html => index.html (100%)

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git log css/style.css
c6976da 2024-11-30 | Renamed hello.html; moved style.css (HEAD -> style) [pepin
o]

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git log --follow css/style.css
c6976da 2024-11-30 | Renamed hello.html; moved style.css (HEAD -> style) [pepin
o]
a42143a 2024-11-30 | Added css stylesheet [pepino]
```

## Изменения в ветке main

Как я уже говорил, Git позволяет работать с несколькими ветками одновременно. Это очень удобно при работе в команде, поскольку люди могут параллельно работать над разными функциями. Это также полезно при работе в одиночку: разрабатывая

функции в отдельных ветках, вы можете исправлять ошибки и выпускать небольшие обновления, используя стабильный код в ветке main. Создадим файл README. В нем будет рассказано о сути нашего проекта.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ touch README.md

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git switch main
Switched to branch 'main'

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git add README
fatal: pathspec 'README' did not match any files

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git add README.md

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git commit -m "Added README"
[main 0aa2a16] Added README
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md
```

## Просмотр отличающихся веток

Теперь у нас есть две расходящиеся ветки в репозитории. Используйте следующую команду log для просмотра веток и их расхождения.

Опция --all гарантирует, что мы видим все ветки, так как по умолчанию в логе показывается только текущая ветка.

Опция --graph добавляет простое дерево коммитов, представленное в виде простых текстовых линий. Мы видим обе ветки (style и main) причём ветка main помечена как HEAD, что означает, что она является текущей. Общим предком для обеих веток является ветка, в которую был внесен коммит «Added copyright statement with email».



```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git log --all --graph
* 0aa2a16 2024-11-30 | Added README (HEAD -> main) [pepino]
| * c6976da 2024-11-30 | Renamed hello.html; moved style.css (style) [pepino]
| * 24c5b60 2024-11-30 | Included stylesheet into hello.html [pepino]
| * a42143a 2024-11-30 | Added css stylesheet [pepino]
|/
* bd1b601 2024-11-30 | Added copyright statement with email [pepino]
* c5208d5 2024-11-30 | added h1 tag (tag: v1) [pepino]
* dd8515d 2024-11-30 | Unrelated change for c (tag: v1-beta) [pepino]
* 48f824a 2024-11-30 | Changes only for a and b [pepino]
* de6367a 2024-11-30 | Changes for a and b [pepino]
* 664b7eb 2024-11-30 | Initial Commit [pepino]

```

## Слияние

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

```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git switch style
Switched to branch 'style'

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git merge main
Merge made by the 'ort' strategy.
 README.md | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git log --all --graph
* 3638e99 2024-11-30 | Merge branch 'main' into style (HEAD -> style) [pepino]
|
| \
|  * 0aa2a16 2024-11-30 | Added README (main) [pepino]
|  * c6976da 2024-11-30 | Renamed hello.html; moved style.css [pepino]
|  * 24c5b60 2024-11-30 | Included stylesheet into hello.html [pepino]
|  * a42143a 2024-11-30 | Added css stylesheet [pepino]
|/
* bd1b601 2024-11-30 | Added copyright statement with email [pepino]
* c5208d5 2024-11-30 | added h1 tag (tag: v1) [pepino]
* dd8515d 2024-11-30 | Unrelated change for c (tag: v1-beta) [pepino]
* 48f824a 2024-11-30 | Changes only for a and b [pepino]
* de6367a 2024-11-30 | Changes for a and b [pepino]
* 664b7eb 2024-11-30 | Initial Commit [pepino]

```

## Создание конфликта

При слиянии двух веток Git пытается перенести изменения из одной ветки в другую. Если в обеих ветках была изменена одна и та же часть файла, Git может не справиться с автоматическим слиянием изменений. В этом случае Git сообщит о конфликте и попросит

разрешить его вручную. В этом уроке мы смоделируем конфликт, а затем научимся его разрешать.

В реальной жизни конфликты слияния регулярно возникают при работе в команде. Например, вы и ваш коллега начали работать над двумя разными фичами, затрагивающими одни и те же файлы. Ваш коллега закончил работу первым и слил свои изменения в ветку main. Теперь вы хотите слить свои изменения в ветку main. Но ветка main теперь отличается от той, с которой вы начинали работать в начале — в ней появился новый код, присланный вашим коллегой. Вероятно, Git не сможет автоматически объединить ваши изменения и попросит помощи человека.

Помните, что в нашей ветке main страница по-прежнему называется hello.html? Переключитесь обратно на ветку main и внесите следующие изменения:

```
<> hello.html > html > body > p
1  <!DOCTYPE html>
2  <!-- Author: Alexander Shvets (pepino@gmail.com) -->
3  <html lang="en">
4  <head>
5    <meta charset="UTF-8">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>Hello World Page</title>
8  </head>
9  <body>
10   <h1>Hello pepino!!</h1>
11   <p>Let's learn Git together.</p>
12 </body>
13 </html>
```

[illegible]

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

## Разрешение конфликтов



```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git switch style
Switched to branch 'style'

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git merge main
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style|MERGING)
$ git status
On branch style
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")

```

```

index.html > html
1  <!DOCTYPE html>
2  <!-- Author: Alexander Shvets (pepino@gmail.com) -->
3  <html lang="en">
4  <head>
5    <meta charset="UTF-8">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <<<<<< HEAD:index.html (текущее изменение)
8    <title>Document</title>
9    <link type="text/css" rel="stylesheet" media="all" href="style.css" />
10   =====
11   <title>Hello World Page</title>
12   >>>>>> main:hello.html (входящее изменение)
13   </head>
14   <body>
15     <h1>Hello pepino!!</h1>
16     <p>let's learn Git together.</p>
17   </body>
18   </html>

```

Часть между <<<<<< >>>>>> является конфликтом. Верхняя часть соответствует ветке style, которая является текущей веткой (или HEAD) репозитория. Нижняя часть соответствует изменениям из ветки main. Git не может решить, какие изменения применить, поэтому он просит вас разрешить конфликт вручную. Вы можете оставить изменения из ветки style или из main, либо объединить их любым удобным способом. Вы также можете внести в файл любые другие изменения.

Кстати, вы заметили, что наше второе изменение, тег <p>, не является частью конфликта? Это потому, что Git сумел автоматически объединить ее.

Прежде чем мы приступим к разрешению нашего конфликта, хочу заметить, что сразу бросаться к разрешению конфликта не всегда оптимально. Конфликт может быть вызван изменениями, о которых вы не знаете. Или же изменения слишком велики, чтобы

разрешить конфликт сразу. По этой причине Git позволяет прервать слияние и вернуться к предыдущему состоянию. Для этого можно воспользоваться командой `git merge --abort`, как это было предложено командой `status`, которую мы выполнили ранее.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style|MERGING)
$ git merge --abort

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git status
On branch style
nothing to commit, working tree clean
```

Чтобы разрешить конфликт, нужно отредактировать файл до состояния, которое нас устраивает, и затем закоммитить его как обычно. В нашем случае мы объединим изменения из обеих веток. Поэтому мы отредактируем файл до следующего состояния:

```
<> hello.html > html > head > link
1  <!DOCTYPE html>
2  <!-- Author: Alexander Shvets (pepino@gmail.com) -->
3  <html lang="en">
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Document</title>
8      <link type="text/css" rel="stylesheet" media="all" href="style.css" />
9  </head>
10 <body>
11     <h1>Hello pepino!!</h1>
12     <p>Let's learn Git together.</p>
13 </body>
14 </html>
```

```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style|MERGING)
$ git commit -m "Resolved conflict"
[style 2b6c396] Resolved conflict

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git status
On branch style
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        hello.html

nothing added to commit but untracked files present (use "git add" to track)

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git log --all --graph
* 2b6c396 2024-11-30 | Resolved conflict (HEAD -> style) [pepino]
|
| * 67d3d9c 2024-11-30 | Added meta title (main) [pepino]
| * | 3638e99 2024-11-30 | Merge branch 'main' into style [pepino]
|/
| * 0aa2a16 2024-11-30 | Added README [pepino]
| * | c6976da 2024-11-30 | Renamed hello.html; moved style.css [pepino]
| * | 24c5b60 2024-11-30 | Included stylesheet into hello.html [pepino]
| * | a42143a 2024-11-30 | Added css stylesheet [pepino]
|/
* bd1b601 2024-11-30 | Added copyright statement with email [pepino]
* c5208d5 2024-11-30 | added h1 tag (tag: v1) [pepino]
* dd8515d 2024-11-30 | Unrelated change for c (tag: v1-beta) [pepino]
* 48f824a 2024-11-30 | Changes only for a and b [pepino]
* de6367a 2024-11-30 | Changes for a and b [pepino]
* 664b7eb 2024-11-30 | Initial Commit [pepino]

```

## rebase против merge

Давайте рассмотрим различия между слиянием и перебазируванием. 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.

## Сброс ветки style

Давайте вернемся во времени на ветке `style` к точке *перед* тем, как мы слили ее с веткой `main`. Мы можем сбросить ветку к любому коммиту при помощи команды `reset`. 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 `main`. Нам необходимо найти последний коммит перед слиянием.



```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git switch style
Already on 'style'

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git log --graph
* 2b6c396 2024-11-30 | Resolved conflict (HEAD -> style) [pepino]
|
| * 67d3d9c 2024-11-30 | Added meta title (main) [pepino]
| * 3638e99 2024-11-30 | Merge branch 'main' into style [pepino]
|
| * 0aa2a16 2024-11-30 | Added README [pepino]
| * c6976da 2024-11-30 | Renamed hello.html; moved style.css [pepino]
| * 24c5b60 2024-11-30 | Included stylesheet into hello.html [pepino]
| * a42143a 2024-11-30 | Added css stylesheet [pepino]
|
| * bd1b601 2024-11-30 | Added copyright statement with email [pepino]
| * c5208d5 2024-11-30 | added h1 tag (tag: v1) [pepino]
| * dd8515d 2024-11-30 | Unrelated change for c (tag: v1-beta) [pepino]
| * 48f824a 2024-11-30 | Changes only for a and b [pepino]
| * de6367a 2024-11-30 | Changes for a and b [pepino]
| * 664b7eb 2024-11-30 | Initial Commit [pepino]

```

Это немного трудно читать, но, глядя на данные, мы видим, что коммит «Renamed hello.html; moved style.css» был последним на ветке style перед слиянием. Давайте сбросим ветку style к этому коммиту. Чтобы сослаться на этот коммит, мы либо используем его хеш, либо посчитаем, что этот коммит находится за 2 коммита до HEAD, то есть HEAD~2 в нотации Git.

```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git reset --hard HEAD~2
HEAD is now at c6976da Renamed hello.html; moved style.css

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git log --graph
* c6976da 2024-11-30 | Renamed hello.html; moved style.css (HEAD -> style) [pepino]
|
| * 24c5b60 2024-11-30 | Included stylesheet into hello.html [pepino]
| * a42143a 2024-11-30 | Added css stylesheet [pepino]
| * bd1b601 2024-11-30 | Added copyright statement with email [pepino]
| * c5208d5 2024-11-30 | added h1 tag (tag: v1) [pepino]
| * dd8515d 2024-11-30 | Unrelated change for c (tag: v1-beta) [pepino]
| * 48f824a 2024-11-30 | Changes only for a and b [pepino]
| * de6367a 2024-11-30 | Changes for a and b [pepino]
| * 664b7eb 2024-11-30 | Initial Commit [pepino]

```

## Перебазирование

Мы вернули ветку style к точке перед первым слиянием. При этом в ветке main есть два коммита, которых нет в ветке style: новый файл README и конфликтующее изменение в файле index.html. На этот раз мы перенесем эти изменения в ветку style с помощью команды rebase, а не merge.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git rebase main
error: The following untracked working tree files would be overwritten by check
out:
    hello.html
Please move or remove them before you switch branches.
Aborting
error: could not detach HEAD

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git status
On branch style
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    hello.html

nothing added to commit but untracked files present (use "git add" to track)
```

Опять возник конфликт! Обратите внимание, что конфликт возник в файле hello.html, а не в файле index.html, как в прошлый раз. Это связано с тем, что rebase находился в процессе применения изменений style поверх ветки main. Файл hello.html в main еще не был переименован, поэтому он все еще имеет старое имя.

При слиянии возник бы «обратный» конфликт. При слиянии изменения ветки main были бы применены поверх ветки style. В ветке style файл переименован, поэтому конфликт возник бы в файле index.html.

```
<> index.html > html > body > p
1  <!DOCTYPE html>
2  <!-- Author: Alexander Shvets (pepino@gmail.com) -->
3  <html lang="en">
4  <head>
5    <meta charset="UTF-8">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>Hello World Page</title>
8  </head>
9  <body>
10   <h1>Hello pepino!!</h1>
11   <p>Let's learn Git together.</p>
12 </body>
13 </html>
```

Но после этого нам не нужно коммитить изменения. Мы можем просто добавить файл в индекс и продолжить процесс rebase. Вот почему я люблю rebase! Он позволяет мне устранять конфликты, не создавая кучу уродливых конфликтов слияния.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style|REBASE 3/4)
$ git add .

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style|REBASE 3/4)
$ git rebase --continue
[detached HEAD fe51ab8] Renamed hello.html; moved style.css
3 files changed, 4 insertions(+), 19 deletions(-)
rename style.css => css/style.css (100%)
delete mode 100644 hello.html
Successfully rebased and updated refs/heads/style.
```

Используйте команду rebase:

- Когда вы получаете изменения из удаленного репозитория и хотите применить их к своей локальной ветке.
- Если вы хотите, чтобы история коммитов была линейной и легко читаемой.

Не используйте команду rebase:

- Если текущая ветка является публичной и общей. Переписывание таких веток будет мешать работе других членов команды.
- Если важна *точная* история ветки коммитов (поскольку команда rebase переписывает историю коммитов).

Учитывая приведенные выше рекомендации, я предпочитаю использовать команду rebase для краткосрочных, локальных веток и команду merge для веток в публичном репозитории.

## Слияние в ветку main

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

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (style)
$ git switch main
Switched to branch 'main'

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git merge style
Updating 67d3d9c..281f98e
Fast-forward
 css/style.css | 3 +++
 hello.html    | 3 ++-
 index.html    | 16 ++++++
 3 files changed, 21 insertions(+), 1 deletion(-)
 create mode 100644 css/style.css
 create mode 100644 index.html
```

## Часть 2. Несколько репозиториев

До сих пор мы работали только с одним Git-репозиторием. Однако Git - это **распределенная** система управления версиями, а значит, она отлично подходит для работы с несколькими репозиториями. Эти дополнительные репозитории могут храниться локально, быть доступными через сеть или Интернет. Они также могут быть размещены на GitHub, GitLab, BitBucket или любом другом Git-хостинге.

В следующем разделе мы представим, что решили взять некоторую работу на дом. В былые времена можно было бы перенести этот репозиторий на флешку и взять его с собой домой. Сегодня мы, скорее всего, поделимся репозиторием через GitHub. На самом деле, не важно, как вы делитесь своей работой: Git будет работать одинаково. Большая часть информации, изложенной в этом разделе, может быть применена и для работы с несколькими репозиториями, независимо от того, хранятся ли они локально или передаются по сети.

Поэтому для простоты представим, что мы используем два независимых репозитория, располагая их локально в разных директориях: work и home.

### Клонирование репозиториев



```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories
$ git clone work home
Cloning into 'home'...
done.

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories
$ ды
bash: ды: command not found

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories
$ ls
home/  work/

```

## Просмотр клонированного репозитория

```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories
$ cd home

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ ls
README.md  a.html  b.html  c.html  css/  hello.html  index.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git log --all
281f98e 2024-11-30 | pupupu (HEAD -> main, origin/style, origin/main, origin/HEAD) [pepino]
fe51ab8 2024-11-30 | Renamed hello.html; moved style.css [pepino]
852da91 2024-11-30 | Included stylesheet into hello.html [pepino]
aeb37cc 2024-11-30 | Added css stylesheet [pepino]
67d3d9c 2024-11-30 | Added meta title [pepino]
0aa2a16 2024-11-30 | Added README [pepino]
bd1b601 2024-11-30 | Added copyright statement with email [pepino]
c5208d5 2024-11-30 | added h1 tag (tag: v1) [pepino]
dd8515d 2024-11-30 | Unrelated change for c (tag: v1-beta) [pepino]
48f824a 2024-11-30 | Changes only for a and b [pepino]
de6367a 2024-11-30 | Changes for a and b [pepino]
664b7eb 2024-11-30 | Initial Commit [pepino]

```

## Что такое 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:

```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git remote
origin

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git remote show origin
* remote origin
  Fetch URL: D:/githowto/repositories/work
  Push URL: D:/githowto/repositories/work
  HEAD branch: main
  Remote branches:
    main tracked
    style tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local ref configured for 'git push':
    main pushes to main (up to date)

```

Мы видим, что имя по умолчанию (origin) удаленного репозитория — изначальное work. 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 origin, однако существует традиция использовать origin в качестве имени первичного централизованного репозитория (если таковой имеется).

### Удаленные ветки

Давайте посмотрим на ветки, доступные в нашем клонированном репозитории.

```

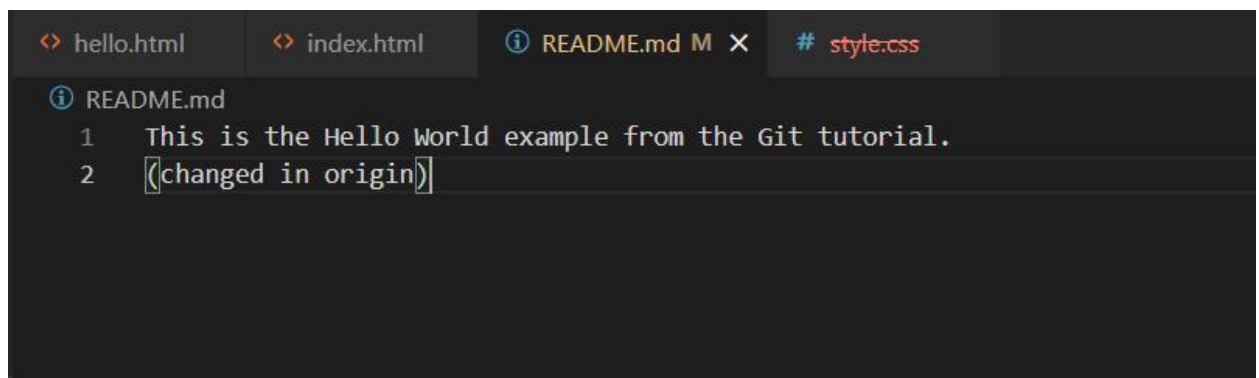
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git branch
* main

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git branch -a
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/main
  remotes/origin/style

```

### Изменение оригинального репозитория

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



```
<> hello.html <> index.html ⓘ README.md M ✕ # style.css
ⓘ README.md
1 This is the Hello World example from the Git tutorial.
2 [[changed in origin]]
```

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ cd ../work

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git add README.md

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git commit -m "Changed README in original repo"
[main aaad450] Changed README in original repo
1 file changed, 2 insertions(+)
```

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

### Подтягивание изменений

```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ cd ../home

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 349 bytes | 49.00 KiB/s, done.
From D:/githowto/repositories/work
  281f98e..aaad450  main       -> origin/main

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git log --all
aaad450 2024-11-30 | Changed README in original repo (origin/main, origin/HEAD) [pepino]
281f98e 2024-11-30 | pupupu (HEAD -> main, origin/style) [pepino]
fe51ab8 2024-11-30 | Renamed hello.html; moved style.css [pepino]
852da91 2024-11-30 | Included stylesheet into hello.html [pepino]
aeb37cc 2024-11-30 | Added css stylesheet [pepino]
67d3d9c 2024-11-30 | Added meta title [pepino]
0aa2a16 2024-11-30 | Added README [pepino]
bd1b601 2024-11-30 | Added copyright statement with email [pepino]
c5208d5 2024-11-30 | added h1 tag (tag: v1) [pepino]
dd8515d 2024-11-30 | Unrelated change for c (tag: v1-beta) [pepino]
48f824a 2024-11-30 | Changes only for a and b [pepino]
de6367a 2024-11-30 | Changes for a and b [pepino]
664b7eb 2024-11-30 | Initial Commit [pepino]

```

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/main и origin/HEAD.

Теперь давайте посмотрим на коммит «Renamed hello.html; moved style.css». Вы увидите, что локальная ветка main указывает на этот коммит, а не на новый коммит, который мы только что подтянули.

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

## Слияние подтянутых изменений



```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git merge origin/main
Updating 281f98e..aaad450
Fast-forward
 README.md | 2 ++
 1 file changed, 2 insertions(+)

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ cat README.md
This is the Hello World example from the Git tutorial.
(changed in origin)

```

Команда `fetch` позволяет контролировать то, что именно подтягивается и сливается в ваши локальные ветки, но для удобства существует также команда `pull`, которая подтягивает и сливает изменения из удаленной ветки в текущую одним вызовом.

### Добавление ветки наблюдения

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

```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git branch --track style origin/style
branch 'style' set up to track 'origin/style'.

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git branch -a
* main
  style
  remotes/origin/HEAD -> origin/main
  remotes/origin/main
  remotes/origin/style

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git log --max-count=2
aaad450 2024-11-30 | Changed README in original repo (HEAD -> main, origin/main, origin/HEAD) [pepino]
281f98e 2024-11-30 | pupupu (origin/style, style) [pepino]

```

### Чистые репозитории

Чистый репозиторий — это репозиторий, не имеющий рабочей директории. Он содержит только директорию `.git`, в которой Git хранит все свои внутренние данные. Основное предназначение таких репозиториях — быть центральным хранилищем, в которое

разработчики могут отправлять и из которого могут получать данные. Поэтому в них нет смысла создавать рабочие файлы, они будут только впустую занимать место на диске. Чистые репозитории также используются в сервисах Git-хостинга таких, как GitHub и GitLab. В следующих уроках мы узнаем, как создать чистый репозиторий и как отправлять в него изменения.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ cd ..

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories
$ git clone --bare work work.git
Cloning into bare repository 'work.git'...
done.

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories
$ ls work.git
HEAD  config  description  hooks/  info/  objects/  packed-refs  refs/
```

Принято считать, что репозитории, заканчивающиеся на `.git`, являются чистыми репозиториями. Мы видим, что в репозитории `work.git` нет рабочей директории. По сути, это просто директория `.git` из обычного репозитория.

### Добавление удаленного репозитория

Давайте добавим репозиторий `work.git` к нашему оригинальному репозиторию.

```
glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories
$ cd work

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git remote add shared ../work.git
```

### Отправка изменений

Поскольку чистые репозитории обычно располагаются на каком-либо удаленном сервере, вы не сможете туда просто зайти, дабы подтянуть изменения. Поэтому нам необходимо как-нибудь передать наши изменения в репозиторий.

Начнем с создания изменения, которое нужно передать в репозиторий. Отредактируйте README и закоммитьте его:

```
i README.md
1 This is the Hello World example from the Git tutorial.
2 [(changed in the origin and pushed to shared)]

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git switch main
Already on 'main'
M README.md

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git add README.md

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git commit -m "Added shared comment to readme"
[main 86b40ab] Added shared comment to readme
1 file changed, 1 insertion(+), 1 deletion(-)

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ git push shared main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 374 bytes | 374.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
To ../work.git
aaad450..86b40ab main -> main
```

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

### Подтягивание общих изменений

Быстро переключитесь в репозиторий `home` и подтяните изменения, только что отправленные в общий репозиторий.



```

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/work (main)
$ cd ../home

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git remote add shared ../work.git

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git branch --track shared main
branch 'shared' set up to track 'main'.

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git pull shared main
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 354 bytes | 50.00 KiB/s, done.
From ../work
 * branch                main          -> FETCH_HEAD
 * [new branch]          main          -> shared/main
Updating aaad450..86b40ab
Fast-forward
 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ cat README.md
This is the Hello World example from the Git tutorial.
(changed in the origin and pushed to shared)

```

## Размещение ваших Git-репозитория

Хотите создать свой собственный GitHub? Существует множество способов совместного использования репозитория Git по сети. Здесь приведен простой и быстрый (но ненадежный и опасный) способ.

The image shows two side-by-side screenshots of a Windows command prompt window titled 'MINGW64: d:/githowto/repositories'. The left screenshot shows the process of cloning a repository and listing files. The right screenshot shows the process of pulling updates and configuring the Git daemon.

```

glush@DESKTOP-8RBE01G MINGW64 ~
$ cd D:/githowto/repositories

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories
$ git clone git://localhost/work.git network_work
Cloning into 'network_work'...
remote: Enumerating objects: 40, done.
remote: Counting objects: 100% (40/40), done.
remote: Compressing objects: 100% (35/35), done.
remote: Total 40 (delta 17), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (40/40), done.
Resolving deltas: 100% (17/17), done.

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories
$ cd network_work

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/network_work (main)
$ ls
README.md a.html b.html c.html css/ hello.html index.html

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/network_work (main)
$ |

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git branch --track shared main
branch 'shared' set up to track 'main'.

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ git pull shared main
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 354 bytes | 50.00 KiB/s, done.
From ../work
 * branch                main          -> FETCH_HEAD
 * [new branch]          main          -> shared/main
Updating aaad450..86b40ab
Fast-forward
 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ cat README.md
This is the Hello World example from the Git tutorial.
(changed in the origin and pushed to shared)

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories/home (main)
$ cd ../

glush@DESKTOP-8RBE01G MINGW64 /d/githowto/repositories
$ git daemon --verbose --export-all --base-path=.
[1772] Ready to rumble
[4700] Connection from [::1]:58405
[4700] unable to set SO_KEEPALIVE on socket: Input/output error
[4700] Extended attribute "host": localhost
[4700] Extended attribute "protocol": version=2
[4700] Request upload-pack for '/work.git'

```



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

**Выводы:** Git — мощная и сложная распределенная система контроля версий, которая работает с изменениями, а не файлами. Git позволяет создавать репозитории с ветками, коммитить и отслеживать изменения в файлах, а также позволяет откатываться к любой интересующей версии проекта благодаря истории изменений.