5. Индексация изменений

Добавьте изменения

Теперь дайте команду Git проиндексировать изменения. Проверьте состояние:

Выполните

git add hello.html

git status

Вы увидите:

Результат

```
$ git add hello.html

$ git status

On branch 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.

6. Индексация и коммит

Отдельный шаг индексации в Git позволяет вам разделять большие изменения на маленькие коммиты. Аналогия: вы помыли машину и заодно залили жидкость для очистки стекла — эти два изменения по своей сути независимы, а потому лучше пометить их отдельно. В противном случае, в истории изменений бачка для жидкости очистки стекла будет запись "Помыл машину", что не соответствует сути изменения и может запутать того, кто потом будет разбираться в этой истории.

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

В теории, вы можете сделать следующее:

git add a.html

git add b.html

git commit -m "Changes for a and b"

git add c.html

git commit -m "Unrelated change to c"

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

7. Коммит изменений

Цели

Научиться коммитить изменения в репозиторий.

01Закоммитьте изменения

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

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

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

переменная среды GIT_EDITOR

параметр конфигурации core.editor

переменная среды VISUAL

переменная среды EDITOR

У меня переменная EDITOR установлена в vim. Если вы предпочитаете GUI-редактор, то теперь можно использовать VS Code в качестве Git-редактора.

Сделайте коммит сейчас и проверьте состояние.

Выполните

git commit

Вы увидите в вашем редакторе:

Результат

|

Please enter the commit message for your changes. Lines starting

with '#' will be ignored, and an empty message aborts the commit.

#

On branch main

Changes to be committed:

#	modified: hello.html

#

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

Результат

$ git commit

[main 78433de] Added h1 tag

 1 file changed, 1 insertion(+), 1 deletion(-)

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

02Проверьте состояние

В конце давайте еще раз проверим состояние.

Выполните

git status

Вы увидите:

Результат

$ git status

On branch main

nothing to commit, working tree clean

Рабочая директория чиста, можем продолжить работу.

8. Изменения, а не файлы

Цели

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

Большинство систем контроля версий работает с файлами: вы добавляете файл в систему, и она отслеживает изменения файла с этого момента.

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

Мы попытаемся исследовать эту разницу в данном уроке.

***01*Первое изменение: Добавьте стандартные теги страницы**

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

Файл: hello.html

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

***02*Добавьте это изменение**

Теперь добавьте это изменение в индекс Git.

Выполните

```
git add hello.html
```

***03*Второе изменение: Добавьте заголовки HTML**

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

Файл: hello.html

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

***04*Проверьте текущий статус**

Выполните

```
git status
```

Вы увидите:

Результат

$ git status

On branch main

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: hello.html

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

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

Давайте проверим.

***05*Коммит**

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

Выполните

git commit -m "Added standard HTML page tags"

git status

Вы увидите:

Результат

$ git commit -m "Added standard HTML page tags"

[main 46afaff] Added standard HTML page tags

1 file changed, 5 insertions(+), 1 deletion(-)

$ 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")

Команда status показывает, что в файле hello.html ещё есть незаписанные изменения, но область подготовки уже пуста.

***06*Добавьте второе изменение**

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

Выполните

git add .

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.

Я просто хотел показать вам трюк с add ., в дальнейшем мы будем добавлять все файлы явно.

Вы увидите:

Результат

$ git add .

$ git status

On branch main

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: hello.html

Второе изменение было проиндексировано и готово к коммиту.

***07*Сделайте коммит второго изменения**

Выполните

git commit -m "Added HTML header"

Результат

$ git commit -m "Added HTML header"

[main b7614c1] Added HTML header

1 file changed, 2 insertions(+)

9. История

Цели

- Научиться просматривать историю проекта.

Получение списка произведенных изменений — функция команды git log.

Выполните

git log

Вы увидите:

Результат

$ git log

commit b7614c1aea1ffbc46400fe1a163842d6ec620a43

Author: Alexander Shvets <alex@githowto.com>

Date: Tue Nov 28 05:51:38 2023 -0600

  Added HTML header

commit 46afaff2232fc3d564c40f65cb82e7e94839a1bb

Author: Alexander Shvets <alex@githowto.com>

Date: Tue Nov 28 05:51:38 2023 -0600

  Added standard HTML page tags

commit 78433de967102f2b59d0a8a60eb397b2663ed282

Author: Alexander Shvets <alex@githowto.com>

Date: Tue Nov 28 05:51:38 2023 -0600

  Added h1 tag

commit 58369706affbc1c27fa03a65fc7a05847278045f

Author: Alexander Shvets <alex@githowto.com>

Date: Tue Nov 28 05:51:38 2023 -0600

  Initial commit

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

***01*Однострочная история**

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

Выполните

git log --pretty=oneline

Вы увидите:

Результат

$ git log --oneline

b7614c1 Added HTML header

46afaff Added standard HTML page tags

78433de Added h1 tag

5836970 Initial commit

***02*Контроль отображения записей**

Вот еще интересные варианты просмотра истории:

git log --oneline --max-count=2

git log --oneline --since="5 minutes ago"

git log --oneline --until="5 minutes ago"

git log --oneline --author="Your Name"

git log --oneline --all

Существует огромное количество вариантов просмотра истории, вы можете порыться на странице руководства git-log, чтобы увидеть их все.

***03*Изощряемся**

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

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

***04*Конечный формат лога**

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

Выполните

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

Выглядит это примерно так:

Результат

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

b7614c1 2023-11-28 | Added HTML header (HEAD -> main) [Alexander Shvets]

46afaff 2023-11-28 | Added standard HTML page tags [Alexander Shvets]

78433de 2023-11-28 | Added h1 tag [Alexander Shvets]

5836970 2023-11-28 | Initial commit [Alexander Shvets]

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.

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:

Выполните

git config --global format.pretty '%h %ad | %s%d [%an]'

git config --global log.date short

***05*Другие инструменты**

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