DevOps
2022.09.06

Opracował: Mikołaj Ziółek
ziolek.mikosz@protonmail.com

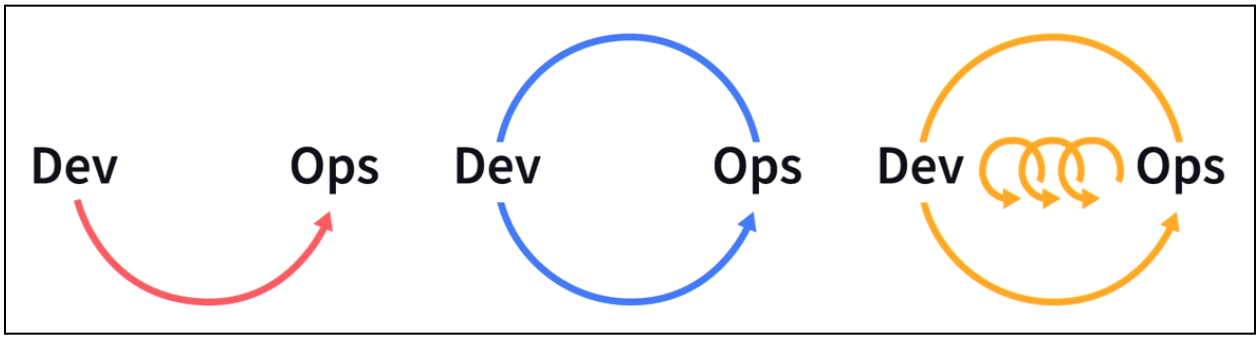
Czym jest DevOps

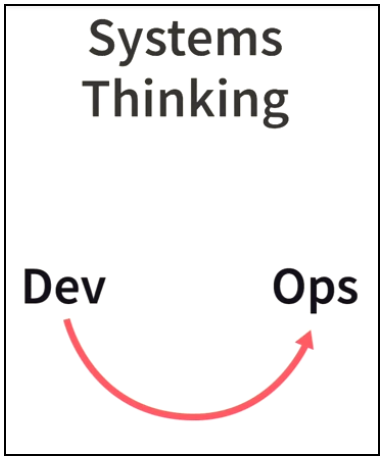
Albo raczej kim jest - to osoba wewnątrz projektu, która stara się ułatwić życie reszcie developerów poprzez m. in. **automatyzacje procesów** przez nich wykonywanych, **nadzorowanie oprogramowania** lub samej **infrastruktury**, tworzenie pipeline'ów **CI/CD**.

Model C.A.M.S.

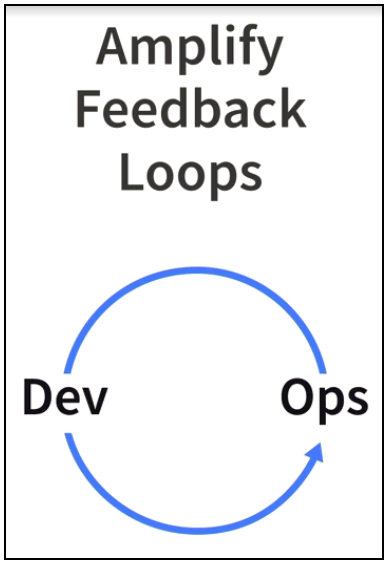
- **C**ulture
    - Sedno tego czym zajmuje się DevOps jest związane z ludźmi i biznesem. Pomijając wszelkie zaawansowane narzędzia, z którym pracują, głównym zadaniem jest stworzenie odpowiedniej kultury w organizacji aby zapewnić rozwiązania dobrej jakości.
- **A**utomation
    - Automatyzacją nazwiemy na przykład kawałek technologii, która wykonuje dane zadanie **bez** dodatkowej ingerencji człowieka. Celem tego zabiegu jest przyspieszenie i ułatwienie powtarzalnej pracy.
- **M**easurement
    - Metryka różnych aspektów będących częścią projektu może być pomocna, pomiary wartości t.j. *czas cyklu zadania, koszta, satysfakcja pracowników mogą ułatwić znalezienie lepszego punkt widzenia na potencjalne problemy i zaangażować zespół, i ustalić ogólny cel.*
- **S**haring
    - ***"Discrete continuous improvement"***
    - *Dzielenie się wiedzą, pomysłami i nawet problemami jest bardzo ważne w życiu DevOps'a. Dzięki temu ktoś może się nauczyć czegoś nowego, lub ten ktoś udzieli Ci rady odnośnie problemu, z którym się zmagasz,*
      lub zespołowi spodoba się Twój nowy pomysł i zostanie wdrożony.

'The Three Ways'

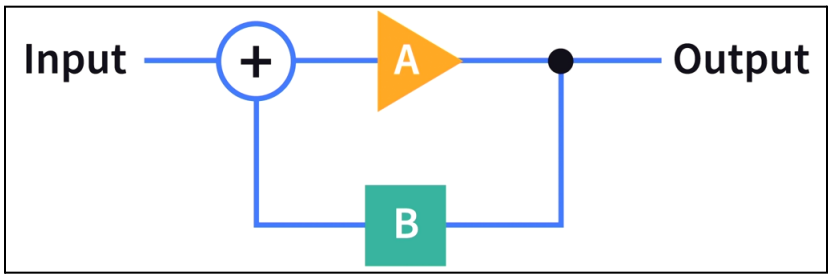




*Ścieżka pierwsza mówi o tym, że na początku powinniśmy się skupić na **ogólnym wyniku** projektowego łańcucha wartości. Musimy pamiętać o tym, że optymalizacja jednego z elementów systemu, może skutkować pogorszeniem drugiego, a w końcu pogorszeniem się ogólnego wyniku.*
Np. Dodanie więcej serwerów do gry komputerowej, nie uwzględniając mocy przerobowej naszej bazy danych, która nie poradzi sobie z większą ilością zapytań może skutkować zawieszeniem serwera lub jego upadkiem.



*Ścieżka druga nawiązuje do tworzenia, skracania i wzmacniania pętli pozyskiwania informacji zwrotnych (**feedback loop**) między częściami organizacji.*
Feedback loop'em nazywamy proces, który bierze pod uwagę własne wyniki przy podejmowaniu decyzji, co dalej.



Ścieżka trzecia przypomina nam o tworzeniu kultury pracy, która pozwala zarówno na ciągłe eksperymentowanie, jak i uczenie się.

W tym podejściu skupiamy się na pracy nad sobą, doskonalenie umiejętności poprzez powtarzanie ćwiczeń, szukanie wyzwań i próbowanie nowych rozwiązań.

Agile Development

„Agile to zestaw wartości i zasad, w których procesy pracy, metody, współpraca i dostarczanie są **stale ulepszane** i **dostosowywane** do zmieniającego się kontekstu. Zespół wykonuje pracę w **małych**, ale eksploatacyjnych przyrostach".

Manifesto for Agile Software

Osoby i interakcje ponad procesami i narzędziami

Działające oprogramowanie ponad obszerną dokumentacją

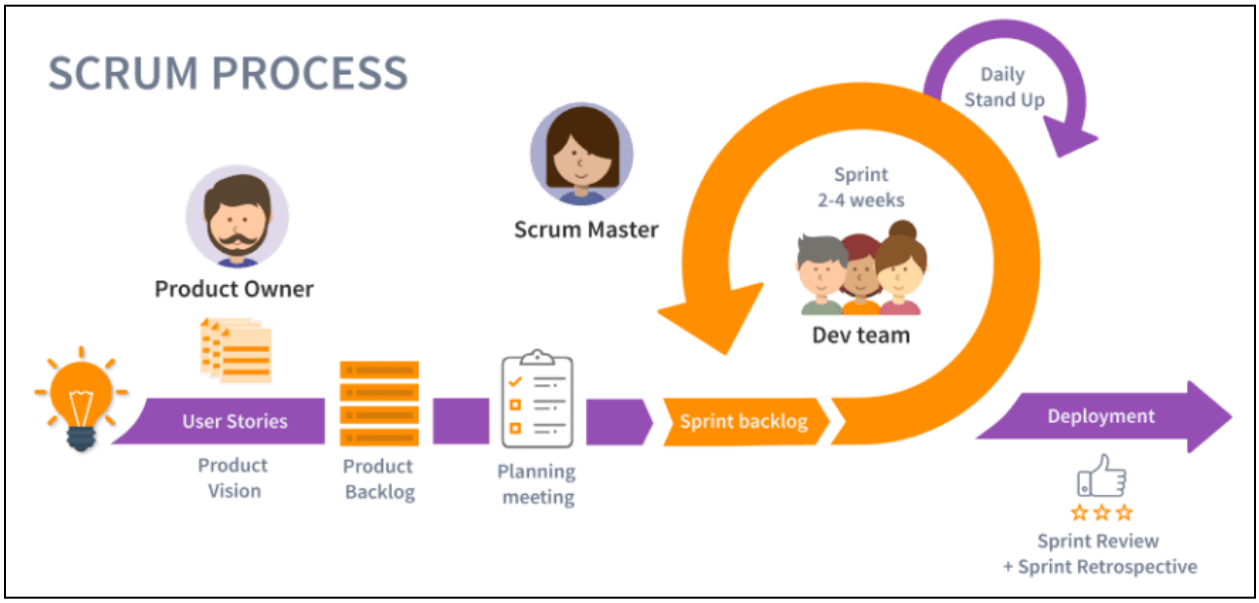
Współpraca z klientem ponad negocjacjami umowy

Reagowanie na zmianę ponad realizację planu

Oznacza to, że podczas gdy przedmioty po prawej stronie **mają wartość**, bardziej cenimy przedmioty po lewej stronie.

Scrum

Scrum to framework, który pomaga zespołom współpracować ze sobą. Często jest uważany za framework **Agile** do zarządzania projektami, który zawiera w sobie zestaw narzędzi, ról i spotkań (standups), aby pomóc zespołom w organizacji pracy i zarządzaniu nią.

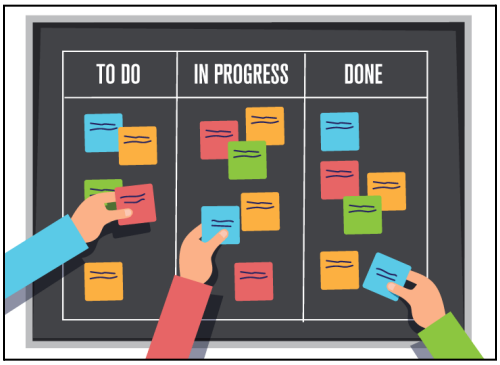


Dodatkowo Scrum zawiera w sobie **sprinty**.

Sprint to krótki, ograniczony czasowo okres, w którym zespół scrumowy pracuje nad wykonaniem określonej pracy

Kanban

Kanban jest to również framework stosowany przy podejściu Agile, wymaga komunikacji w czasie rzeczywistym i przejrzystości pracy. Elementy owej pracy są reprezentowane poprzez karteczki samoprzylepne, bądź wizualnie na tablicy Kanban w dedykowanym narzędziu. Dzięki takiemu podejściu, każdy z zespołu ma wgląd do aktualnego stanu każdego z elementów projektu.



Lean Development

Metodologią Lean nazwiemy zbiór wiedzy, który jest rozwijany w oparciu o produkcję i skupia się na dostarczaniu lepszych produktów poprzez ciągłe doskonalenie.

- „Zrób coś, gdy ktoś tego potrzebuje, zamiast mieć nadzieję, że ktoś tego chce. Oznacza to, że produkt powinien być wykonany tylko wtedy, gdy jest na niego zapotrzebowanie.
- Poświęć trochę czasu na podniesienie jakości w okresach braku popytu, zwiększa to również produkcję i wydajność.
- Zamiast wykonywać daną pracę wielokrotnie, skup się na poprawie jakości wykonywanej pracy

Lean Development skupia się na ciągłym **podnoszeniu jakości** i **satysfakcji klienta**.

Różnica pomiędzy podejściem Lean a Agile polega na tym, że przy metodologii **Lean** zespoły **zwiększają szybkość zarządzając przepływem** (flow) (zwykle poprzez ograniczanie pracy w toku), podczas gdy w **Agile** zespoły **kładą nacisk na małe rozmiary partii**, aby zapewnić szybką dostawę (często w sprintach).

ITSM, ITIL and SDLC

IT **S**ervice **M**anagement

ITSM jest ogólnym pojęciem dotyczącym zarządzania wszystkimi usługami IT w organizacjach.

Information **T**echnology **I**nfrastructure **L**ibrary

https://soflab.pl/itsm/co-to-jest-itsm-i-czym-sie-rozni-od-itil/

Git Essentials

Basic commands

```
$ git init
```

*Po zainstalowaniu Git'a i utworzeniu naszego folderu projektowego możemy zainicjalizować nasze repozytorium przy pomocy komendy **git init**.*

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn
$ git init
Initialized empty Git repository in D:/Projects/devop-learn/.git/
```

*Po wykonaniu komendy w folderze projektu zostanie utworzony folder '**.git**'. To w nim odbywa się cała magia **śledzenia zmian**. Tutaj Git przechowuje wszystko, co wie o naszym projekcie.*

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ ls -la
total 8
drwxr-xr-x 1 ziole 197609 0 Sep  6 23:58 ./
drwxr-xr-x 1 ziole 197609 0 Sep  6 23:57 ../
drwxr-xr-x 1 ziole 197609 0 Sep  6 23:58 .git/
```

```
$ git add
```

*Aby śledzić zmiany, należy najpierw jakichś dokonać. W tym przypadku w folderze projektu umieszczę tę właśnie notatkę i użyję komendy '**git add .**' aby dodać wszystkie (w tym przypadku ten jeden plik) pliki z aktualnego katalogu, które nie są jeszcze śledzone.*

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git add .
```

```
$ git commit
```

*Po dodaniu zmian z katalogu, Git jeszcze faktycznie ich nie śledzi, aby to osiągnąć, należy wywołać komendę **git commit** z dodatkową flagą **-m "i naszym komentarzem"**.*

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git commit -m "[Initial Commit] Added my learning notes"
[master (root-commit) 7dfde50] [Initial Commit] Added my learning notes
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 DevOps_Notes.pdf
```

git commit -- amend

W przypadku gdybyśmy chcieli dodać nowe zmiany do **istniejącego** commit'a, skorzystamy komendy **git commit --amend -m "nasz komentarz"**.

Ta opcja działa tylko dla **ostatniego** commitu w naszej hierarchii. Nie możemy dodać nowych zmian do innego, wcześniejszego, commitu.

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/explore_california (master)
$ git show 074168e922d5de01ce2a8b4372c18a57075590ec
commit 074168e922d5de01ce2a8b4372c18a57075590ec (HEAD -> master)
Author: mikosz08 <46966211+mikosz08@users.noreply.github.com>
Date:   Sat Oct 1 15:25:11 2022 +0200

    Adds file to commit    ZAWARTOŚĆ POPRZEDNIEGO COMMITA

diff --git a/explore_california/file_to_commit.txt b/explore_california/file_to_commit.txt
new file mode 100644
index 0000000..e69de29

ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/explore_california (master)
$ git status
On branch master
Your branch is ahead of 'DevOps/master' by 12 commits.
  (use "git push" to publish your local commits)

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

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

ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/explore_california (master)
$ git add file_to_amend.txt

ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/explore_california (master)
$ git commit --amend -m "Add file_to_commit & file_to_amend"
[master 23c3318] Add file_to_commit & file_to_amend
 Date: Sat Oct 1 15:25:11 2022 +0200
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 explore_california/file_to_amend.txt
 create mode 100644 explore_california/file_to_commit.txt
```

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/explore_california (master)
$ git show 23c331891ce55ba769409b96ba2cd47845c4b4b0
commit 23c331891ce55ba769409b96ba2cd47845c4b4b0 (HEAD -> master)
Author: mikosz08 <46966211+mikosz08@users.noreply.github.com>
Date:   Sat Oct 1 15:25:11 2022 +0200

    Add file_to_commit & file_to_amend

diff --git a/explore_california/file_to_amend.txt b/explore_california/file_to_amend.txt
new file mode 100644
index 0000000..e69de29    STARY COMMIT ZOSTAŁ ZASTĄPIONY NOWYM Z NOWYMI ZMIANAMI
diff --git a/explore_california/file_to_commit.txt b/explore_california/file_to_commit.txt
new file mode 100644
index 0000000..e69de29
```

*Wynikiem operacji jest zupełnie nowy commit (z innym **SHA**) zawierający wnętrze poprzedniego + nowo dodane zmiany.*

```
$ git log
```

*Komenda **git log** wylistuje wszystkie commit'y i informację o nich, które zostały utworzone w ramach naszego projektu.*

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git log
commit 7dfde5067f9e1c7d246e9eec1d35791054e238c3 (HEAD -> master)
Author: mikosz08 <46966211+mikosz08@users.noreply.github.com>
Date:   Wed Sep 7 00:34:20 2022 +0200

    [Initial Commit] Added my learning notes
```

Idąc od góry, widzimy:

commit - *Unikalny numer identyfikujący nasz commit.*

Author - *Autor commit, pobrany z pliku konfiguracyjnego .gitconfig.*

Date - *Data utworzenia commit.*

message - *Komentarz dodany przez autora w trakcie tworzenia commit komendą git commit -m "komentarz"*

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git log --since=2022-09-01 --until=2022-09-31 --author=mikosz --grep="Init"
commit 7dfde5067f9e1c7d246e9eec1d35791054e238c3
Author: mikosz08 <46966211+mikosz08@users.noreply.github.com>
Date:   Wed Sep 7 00:34:20 2022 +0200

    [Initial Commit] Added my learning notes
```

```
$ git status
```

*Zadaniem komendy **git status** jest wylistowanie wykrytych zmian wewnątrz projektu.*

Przykładowe przypadki użycia:

Brak zmian:

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Wykrycie nowego pliku:

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        new_file.txt.txt

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

Zarejestrowanie śledzenia nowego pliku:

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git add new_file.txt.txt

ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   new_file.txt.txt
```

Zarejestrowanie usunięcia nowego pliku:

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   new_file.txt.txt

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:    new_file.txt.txt
```

```
$ git diff
```

*Podczas wprowadzania zmian w wielu plikach (już istniejących w naszym repozytorium), czasem możemy zapomnieć, lub nawet nie spodziewać się, że jakiś plik zawiera w sobie nowe elementy. W takich chwilach dobrze jest posłużyć się komendą **git diff,** która wylistuje nam istniejące modyfikacje.*

*W tym przykładzie dodałem do repozytorium plik **file.txt**, po czym go zmodyfikowałem:*

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file.txt

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:   file.txt
```

*Komendą **git diff** mogę podejrzeć co zostało zmienione:*

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git diff
diff --git a/file.txt b/file.txt
index f925f18..0cd5375 100644
--- a/file.txt
+++ b/file.txt
@@ -1 +1 @@
-Hello, this is a file.
\ No newline at end of file
+Hello, this is a modified file.
\ No newline at end of file
```

Jak widać kolorem czerwonym został zaznaczony fragment, który uległ zmianie i kolorem zielonym jak wygląda teraz.

*Domyślnie **git diff** pokaże nam zmiany istniejące pomiędzy **working directory** a **staging area** (więcej o nich później), czyli pomiędzy naszym lokalnym miejscem roboczym a obszarem, w którym git przechowuje poprzednią wersję pliku. Czyli po użyciu komendy **git add file.txt** nie zobaczymy już zmian.*

*W przypadku gdybyśmy chcieli zobaczyć zmiany na już dodanych plikach skorzystamy z komendy **git diff --staged**.*

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git add file.txt

ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git diff --staged
diff --git a/file.txt b/file.txt
new file mode 100644
index 0000000..3aae3e9
--- /dev/null
+++ b/file.txt
@@ -0,0 +1 @@
+Hello, this is a modified and staged file.
\ No newline at end of file
```

*W tym przypadku ujrzymy zmiany pomiędzy **repository** a **staging area** (tak jakby poziom wyżej).*

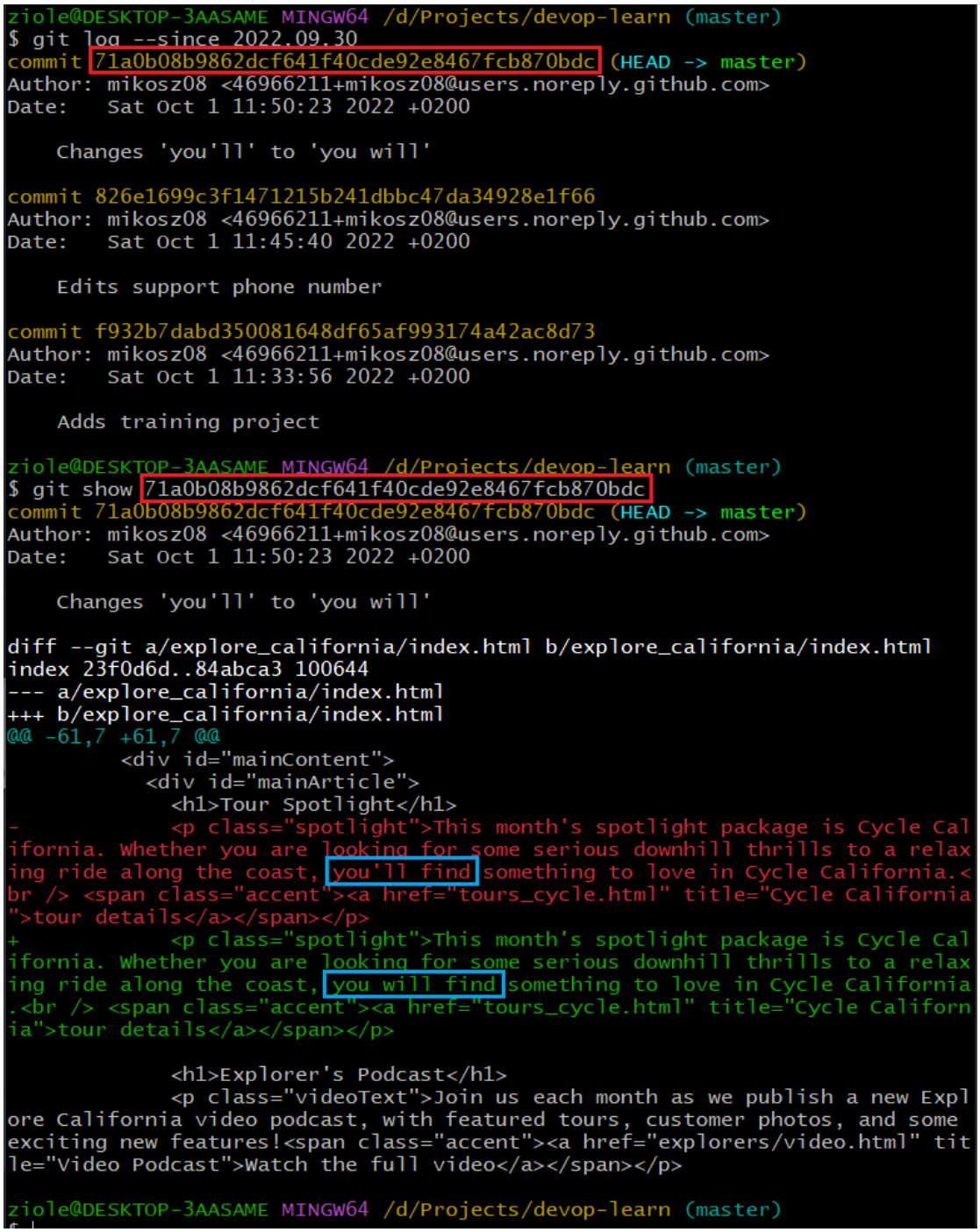
```
$ git rm --cached
```

*W przypadku gdy dodamy do **staging area** niechciany plik, możemy go usunąć przy pomocy **git rm –cached <nazwa_pliku>**. Ta operacja usunie go z **indexu** ale sam plik **pozostanie** w folderze roboczym.*

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git status
On branch master
Your branch is up to date with 'DevOps/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file_to_delete.txt


ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git rm --cached file_to_delete.txt
rm 'file_to_delete.txt'

ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git status
On branch master
Your branch is up to date with 'DevOps/master'.

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

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

```
$ git show
```

*Komenda **git show <sha>** powie nam co zawiera dany **commit**. (polecam flagę **- -color-words**)*

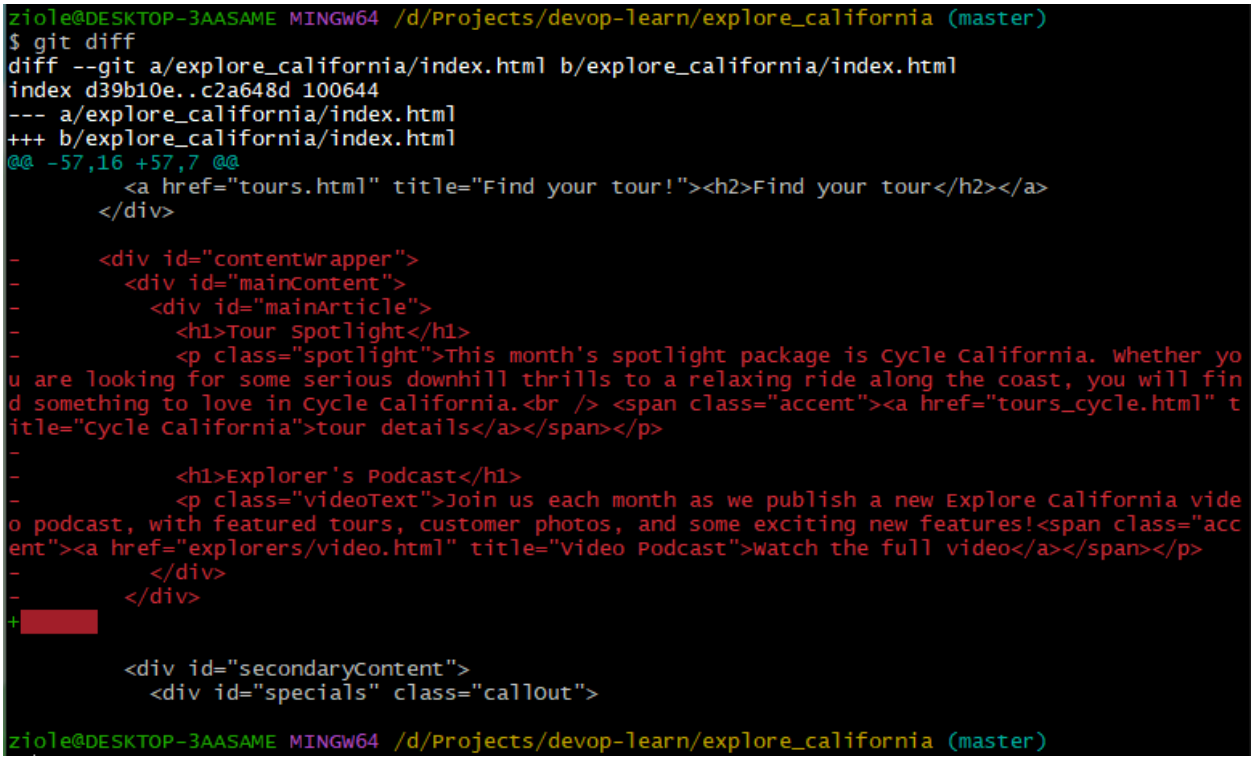


```
$ git checkout
```

`git reset` changes the staging index and `git checkout` changes the working directory.

```
$ git checkout -- <file_name>
```

Załóżmy, że wyłączyłem edytor i okazało się, że przez przypadek usunąłem kawałek istotnego kodu:



Nie mam już możliwości cofnięcia zmian z **ctr + z**. Na szczęście Git śledził poprzednie wersje tego pliku i teraz mogę w łatwy sposób polecić mu sprawdzenie danego pliku i **przywrócenie aktualnej wersji z repozytorium**, w tym przypadku użyję opcji **git checkout -- index.html**.

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/explore_california (master)
$ git checkout -- index.html

ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/explore_california (master)
$ git status
On branch master
Your branch is ahead of 'DevOps/master' by 10 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

```
$git checkout <SHA> -- <file_name>
```

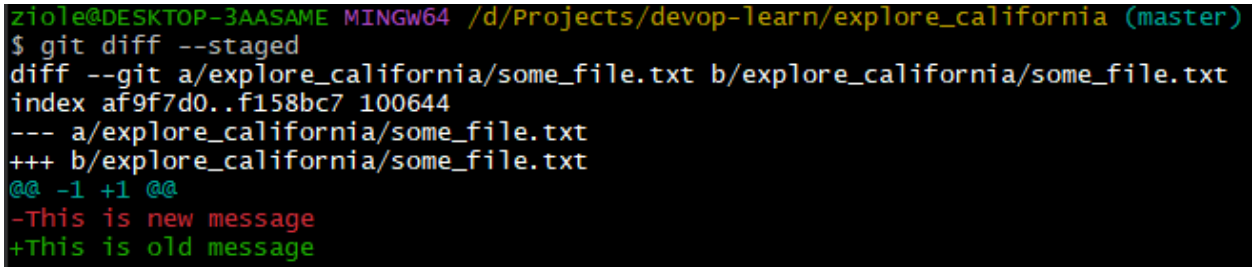
W przypadku jeśli chcemy przywrócić wersję pliku z konkretnego commitu, do komendy **git checkout** musimy jeszcze dorzucić **SHA** i nazwę pliku, którego wersję chcemy przywrócić.

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/explore_california (master)
$ git show dfa1107d3f2c70a5f91729f9a22bc9ac8c91b0c8
commit dfa1107d3f2c70a5f91729f9a22bc9ac8c91b0c8 (HEAD -> master)
Author: mikosz08 <46966211+mikosz08@users.noreply.github.com>
Date:   Sat Oct 1 15:48:19 2022 +0200

    Edits some_file

diff --git a/explore_california/some_file.txt b/explore_california/some_file.txt
index f158bc7..af9f7d0 100644
--- a/explore_california/some_file.txt
+++ b/explore_california/some_file.txt
@@ -1 +1 @@
-This is old message
+This is new message
```

Załóżmy, że chcemy przywrócić starą wersję pliku **some_file.txt**. Aktualnie w jego zawartości znajdziemy wiadomość “This is new message”.

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/explore_california (master)
$ git checkout 3bd693067110eb94158f15518f291f23525d6a41 -- some_file.txt

ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/explore_california (master)
$ git status
On branch master
Your branch is ahead of 'DevOps/master' by 14 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   some_file.txt
```

Po wprowadzeniu, wcześniej znalezionego (np.: komendą **git log**) SHA, Git automatycznie przywrócił poprzednią wersję pliku i umieścił ją w **staging area**.



```
$ git reset
```

Resetowanie przyda się w momencie gdy umieściliśmy już zmiany w **staging area**, tzn zrobiliśmy **git add .**, i chcielibyśmy cofnąć ten proces.

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/explore_california (master)
$ git status
On branch master
Your branch is ahead of 'DevOps/master' by 10 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   tours.html
        modified:   tours/tour_detail_backpack_cal.html
        modified:   tours/tour_detail_bigsur.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:   contact.html
        modified:   explorers.html
        modified:   index.html
        modified:   mission.html
        modified:   resources.html
        modified:   resources/faq.html
        modified:   support.html
```

Załóżmy, że jednak **tours.html** nie powinien się znaleźć w **staging area**, przywrócimy go za pomocą komendy **git reset HEAD tours.html** :

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/explore_california (master)
$ git reset HEAD tours.html
Unstaged changes after reset:
M       explore_california/contact.html
M       explore_california/explorers.html
M       explore_california/index.html
M       explore_california/mission.html
M       explore_california/resources.html
M       explore_california/resources/faq.html
M       explore_california/support.html
M       explore_california/tours.html

ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/explore_california (master)
$ git status
On branch master
Your branch is ahead of 'DevOps/master' by 10 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   tours/tour_detail_backpack_cal.html
        modified:   tours/tour_detail_bigsur.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:   contact.html
        modified:   explorers.html
        modified:   index.html
        modified:   mission.html
        modified:   resources.html
        modified:   resources/faq.html
        modified:   support.html
        modified:   tours.html
```

Ten zabieg nazywamy **unstaging**'iem.

```
$ git revert
```

W przypadku gdy, na przykład, okazało się, że nowy **commit** zawiera zmiany, które są całkowicie błędne i niepotrzebne, chcielibyśmy je cofnąć.

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git show 1960ec332ed18fb2b29ed20ff3954ba9eec29cff
commit 1960ec332ed18fb2b29ed20ff3954ba9eec29cff (HEAD -> master)
Author: mikosz08 <46966211+mikosz08@users.noreply.github.com>
Date:   Sat Oct 1 17:09:10 2022 +0200

    Adds working file

diff --git a/working_file.txt b/working_file.txt
new file mode 100644
index 0000000..68300b8
--- /dev/null
+++ b/working_file.txt
@@ -0,0 +1 @@
+It works!


ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$

ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git show a64d005661bf95855d6b3b5ab0b013fb4d65b378
commit a64d005661bf95855d6b3b5ab0b013fb4d65b378 (HEAD -> master)
Author: mikosz08 <46966211+mikosz08@users.noreply.github.com>
Date:   Sat Oct 1 17:10:11 2022 +0200

    Break file

diff --git a/working_file.txt b/working_file.txt
index 68300b8..b309c62 100644
--- a/working_file.txt
+++ b/working_file.txt
@@ -1 +1 @@
-It works!
+It does not work! :(
```

Możemy to osiągnąć komendą **git revert <SHA>**. Gdzie SHA pochodzi od commitu, który spowodował niechciane zmiany.

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git revert a64d005661bf95855d6b3b5ab0b013fb4d65b378
[master b4cf76f] Revert "Break file"
 1 file changed, 1 insertion(+), 1 deletion(-)

ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git show b4cf76ffb26bf963134b368088b5fc65e9ab8ac9
commit b4cf76ffb26bf963134b368088b5fc65e9ab8ac9 (HEAD -> master)
Author: mikosz08 <46966211+mikosz08@users.noreply.github.com>
Date:   Sat Oct 1 17:11:19 2022 +0200

    Revert "Break file"

    This reverts commit a64d005661bf95855d6b3b5ab0b013fb4d65b378.

diff --git a/working_file.txt b/working_file.txt
index b309c62..68300b8 100644
--- a/working_file.txt
+++ b/working_file.txt
@@ -1 +1 @@
-It does not work! :(
+It works!
```

```
$ git clean
```

Sprzątanie plików-śmieci, takich jak .log, .err, może być uciążliwe, zwłaszcza jak liczymy je w dziesiątkach lub setkach.

Aby wyczyścić naszą stację roboczą z nie-śledzonych plików (**untracked files**) użyjemy komendy **git clean -f**.

Dodatkowym krokiem bezpieczeństwa będzie użycie flagi **-n**, która pokaże nam co by się stało gdybyśmy rzeczywiście uruchomili ten skrypt. (Zalecane podejście ponieważ usuwanie plików skryptem często jest niebezpieczne).

```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git status
On branch master
Your branch is ahead of 'DevOps/master' by 35 commits.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        junk_01.txt
        junk_02.txt
        junk_03.txt

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

ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git clean -n
Would remove junk_01.txt
Would remove junk_02.txt
Would remove junk_03.txt
```

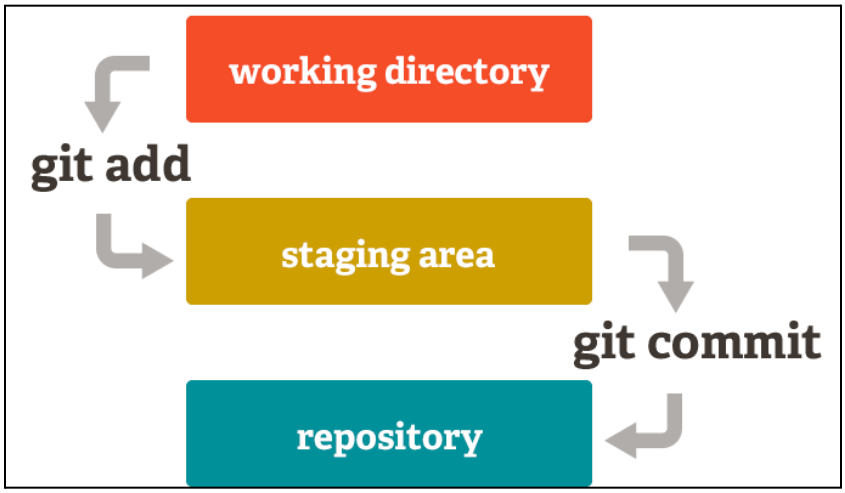
W momencie gdy jesteśmy pewni, że pliki, które mają zostać usunięte są właściwe, uruchamiamy komendę z flagą -f, force.

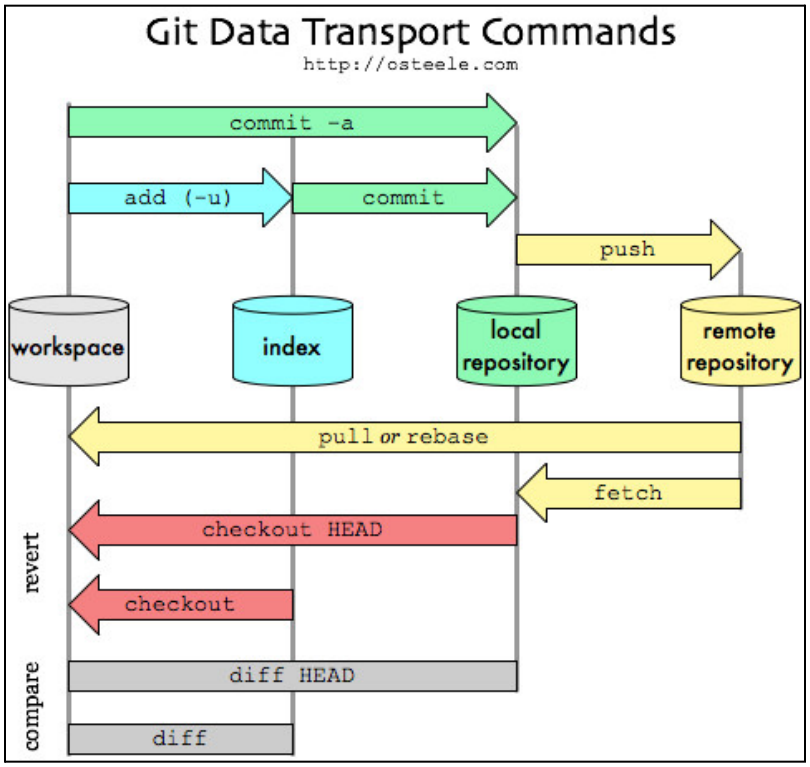
```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn (master)
$ git clean -f
Removing junk_01.txt
Removing junk_02.txt
Removing junk_03.txt
```

Git Architecture

*Git używa tak zwanej **architektury trzech drzew (three-tree architecture)**.*

*To oznacza, że (tak jak inne systemy kontroli) posiada repozytorium i strefę roboczą **i** dodatkowo **staging index** czyli miejsce, do którego odkładane są nasze **commit'y**.*





Dzięki temu możemy dokonać zmian w wielu różnych plikach w naszym obszarze roboczym, a następnie utworzyć zestaw z kilku niezależnych commitów.

Hash Values & HEAD

Hash Value

Git generuje sumę kontrolną (checksum) dla każdego zbioru zmian.
*Jest to wartość **HASH** - liczba generowana przez pobranie danych i wprowadzenie ich do algorytmu matematycznego.*

*W folderze .git wewnątrz katalogu naszego projektu znajdziemy plik **'HEAD'** - znajdziemy w nim informację o tym na jaką wartość hash, w tym momencie, wskazuje HEAD.*

Przy pomocy komendy 'git log' możemy sprawdzić czy faktycznie tak jest.

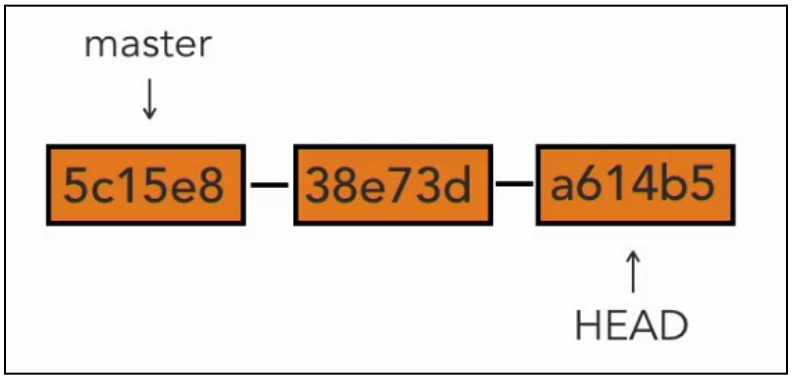
```
ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/.git (GIT_DIR!)
$ cat HEAD
ref: refs/heads/master

ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/.git (GIT_DIR!)
$ cat refs/heads/master
e4d974efe63ed09495b02a00a68643c8536f0d76

ziole@DESKTOP-3AASAME MINGW64 /d/Projects/devop-learn/.git (GIT_DIR!)
$ git log
commit e4d974efe63ed09495b02a00a68643c8536f0d76 (HEAD -> master)
Author: mikosz08 <46966211+mikosz08@users.noreply.github.com>
Date:   Thu Sep 8 02:23:55 2022 +0200

    Adds new topic - 'Git Architecture'
```

Jak widać wartość w pliku HEAD wskazuję na hash zawarty w moim ostatnim commit'cie.

H-E-A-D

*Jest to zmienna nazywana wskaźnikiem lub referencja do konkretnego **commit'a**.*

HEAD zawsze wskazuje na 'czubek' naszego branch'a (o branch'ach później) w naszym repozytorium. To miejsce w którym zakończyliśmy nasze ostatnie działania.



Bibliografia

Works Cited

"Git Essentials." *YouTube*, https://www.linkedin.com/learning/git-essential-training-the-basics/use-git-version-control-software-to-manage-project-code?autoplay=true&contextUrn=urn%3Ali%3AlearningCollection%3A6980271041537290240. Accessed 9 October 2022.

"What is Agile?" *Atlassian*, https://www.atlassian.com/agile. Accessed 9 October 2022.