Git - Undo changes

Spis treści

HEAD

Zanim przejdziemy do dalszych wyjaśnień, chciałbym dodać wyjaśnienie, czym jest **HEAD**. Termin ten będzie pojawiał się wielokrotnie i będziemy na niego patrzeć z różnej perspektywy. Stwierdzenie **HEAD** pojawiało się już wcześniej.

Jeszcze nie wyjaśnialiśmy sobie dokładnie, czym są branche (*gałęzie*), dlatego na ten moment przyjmijmy, że jesteśmy na gałęzi głównej **main** i wrócimy do tego, czym są gałęzie później.

HEAD można rozumieć jako wskaźnik aktualnej referencji gałęzi, na której się znajdujemy. Bardziej po polsku, **HEAD** to miejsce, gdzie się znajdujesz w repozytorium. Jest to wskazanie, gdzie obecnie w tym repozytorium jesteś. Typowo **HEAD** wskazuje na branch i wtedy jesteśmy na najnowszych zmianach na tym branchu, ale **HEAD** może też wskazywać na konkretny commit (wrócimy jeszcze do tego).

Jeżeli chcemy ugryźć **HEAD** w inny sposób, to można powiedzieć, że jest to odpowiedź na pytanie: *Gdzie jestem w tym momencie w repozytorium?*. Jesteś albo na konkretnym branchu, albo na konkretnym commicie. Jeszcze inaczej patrząc, **HEAD** można rozumieć jak taki wskaźnik do punktu w czasie, na który obecnie patrzymy w naszym repozytorium. Z reguły jesteśmy w najświeższym punkcie naszego repozytorium, na najnowszych zmianach.

Mówiąc jeszcze innymi słowami, **HEAD** to wskaźnik na aktualną pozycję w repozytorium. **HEAD** może wskazywać na gałąź (branch) lub na konkretny commit (wtedy będziemy mówić o stanie *Detached HEAD*, jeszcze do tego stanu wrócimy). Oznacza to, że **HEAD** wskazuje na ostatni commit w bieżącej

gałęzi lub w trybie *Detached HEAD* na konkretny commit (jeszcze raz zaznaczam, że wrócimy do *Detched HEAD*).

HEAD jest używany przez Git do określenia, *który commit jest aktualnie "zobaczony" i jakie zmiany zostały wprowadzone od tego momentu*. Przykładowo, gdy robimy checkout na inną gałąź, **HEAD** zostaje przestawiony na ten nowy branch i wskazuje na ostatni commit w tej gałęzi.

Gdy wcześniej wpisywaliśmy `git log`, to widzieliśmy taki zapis:

```
HEAD -> main
```

Oznaczało to, że jesteśmy aktualnie na branchu **main**. Do branchy jeszcze wrócimy, do bycia na konkretnym commicie również.

Cofanie dodanych zmian

Wiemy już, w jaki sposób dodać zmiany do repozytorium. Przydałoby się też dowiedzieć, w jaki sposób możemy przywrócić repozytorium do stanu poprzedniego. Poznamy dwa sposoby na cofanie zmian, będą to `git reset` i `git revert`. Trochę się między sobą te polecenia różnią, a właściwie to efekt ich działania się trochę różni. To kiedy można te polecenia stosować z powodzeniem, zależy też od tego, czy pracujemy tylko w obrębie repozytorium lokalnego, czy może nasze zmiany znalazły się już w repozytorium zdalnym. Przypominam, że jak dotychczas cały czas rozmawiamy o pracy w obrębie repozytorium lokalnego. Gdy włączymy do przykładów repozytorium zdalne - zostanie to wyraźnie zaznaczone. Przejdźmy do przykładów.

Git revert

Komenda `git revert` służy do wycofania zmian, ale jest ona o tyle charakterystyczna, że zmiany te są wycofywane przez dodanie nowego commita wycofującego zmiany. Commit wycofujący zmiany może być rozumiany jako takie lustrzane odbicie zmiany, którą chcemy wycofać. Mówiąc lustrzane odbicie, mam na myśli, że do historii w repozytorium zostaną dodane zmiany, które zniwelują efekt zmian wprowadzonych w jakimś konkretnym commicie, czyli:

- jeżeli jakaś linijka została usunięta - revert doda ją z powrotem,
- jeżeli jakaś linijka została dodana - revert ją usunie,
- jeżeli jakaś linijka została zmodyfikowana - revert przywróci ją do stanu poprzedniego, itp.

Jeszcze raz zaznaczę, że takie wycofanie zmian zostanie odłożone w historii jako **kolejny** commit.

Komenda `git revert` ma dostępnych dużo parametrów, żeby je zobaczyć, wystarczy wpisać komendę jak poniżej:

```
git revert
```

Jednakże najpopularniejszym zastosowaniem tej komendy będzie **revert** konkretnego **commit**a. Możemy wtedy zapisać takie polecenie:

```
git revert 4fc2671c
```

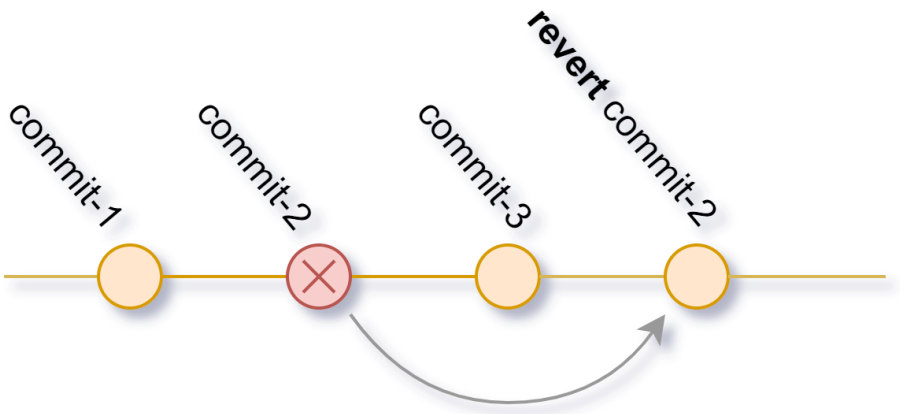
Do historii zostanie dodany wtedy commit, który będzie wyglądał w ten sposób (`git log`):

```
Author: fake_user <fakeemail@gmail.com>
Date:   Thu Apr 21 18:56:04 2022 +0200

    Revert "NewClass"

    This reverts commit 4fc2671c471413f65105191ef587e8428079f298.
```

Powyższy commit odkłada w historii kolejny wpis, który "odwraca" efekt konkretnego commita.



Obraz 1. Git revert

Commit zaznaczony na czerwono nie jest usuwany, zostaje nadal w historii. Na jego podstawie jest tworzony commit kompensujący zmiany wprowadzone w czerwonym commicie.

Pokazany przykład jest prosty, tzn. **Git** bezproblemowo może wykonać **revert**. W praktyce mogą wystąpić sytuacje, gdzie **Git** nie będzie w stanie wykonać automatycznego **revert**a i wtedy dostaniemy taką informację na ekranie:

```
error: could not revert b21422e... Dog.java
hint: after resolving the conflicts, mark the corrected paths
hint: with 'git add <paths>' or 'git rm <paths>'
hint: and commit the result with 'git commit'
```

Wiadomość ta oznacza, że **Git** nie był w stanie wykonać takiego wycofania automatycznie. Jeżeli po zobaczeniu takiej wiadomości otworzysz plik, w którym zmiany chcesz wycofać, to może on wyglądać analogicznie do tego poniżej:

```
public class Dog {

    public static void main(String[] args) {
        System.out.println("print");
<<<<<<< HEAD
=======
        System.out.println("print3");
```

```
>>>>>>> parent of 23b4206... print
        System.out.println("print2");
    }
}
```

Na razie tymi zapisami się nie przejmuj, wrócimy do nich, gdy będziemy rozmawiać o **merge** i o **merge conflicts**. Na ten moment wystarczy Ci informacja, że gdy coś takiego wystąpi podczas robienia `git revert`, to wystarczy napisać:

```
git revert --abort
```

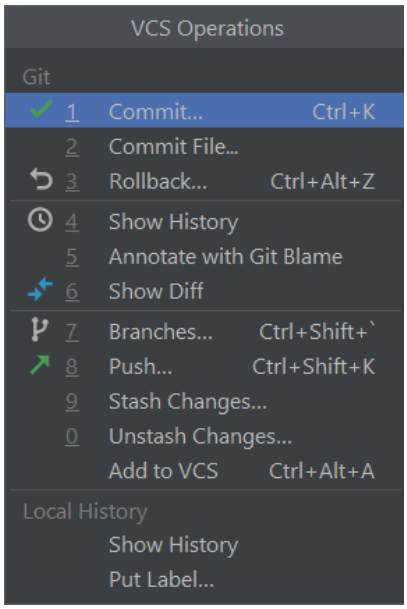
Co może być zrozumiane jako: "wycofaj się z tego reverta i wróć do stanu wyjściowego".

Samo polecenie `git revert` może być z powodzeniem używane, gdy nasze zmiany są już zsynchronizowane z repozytorium zdalnym. Wynika to z tego, że `git revert` nie zmienia istniejącej już historii, tylko dodaje nowe wpisy. Wrócimy do tego, gdy przejdziemy do przykładów z repozytorium zdalnym.

Intellij

To samo (`git revert`) możemy zrobić z poziomu Intellij. Mamy dwie możliwości.

1. Możemy przejść do zakładki **Git** > **Log**, znaleźć commit, który nas interesuje i kliknąć go prawym przyciskiem myszki. Następnie mamy dostępną opcję **Revert commit**.
2. Możemy dwukrotnie wcisnąć klawisz **Shift** i wpisać "VCS operations", gdy wejdziemy w tę opcję, pojawi nam się takie okienko:



Obraz 2. Intellij VCS Operations

To samo okienko może być otworzone przy wykorzystaniu skrótu `Alt + `` . Zapamiętaj ten skrót, bo będzie on pomocny w pracy z VCS w Intellij.

Jeżeli wybierzemy teraz pozycję czwartą, to pokaże nam się historia zmian tylko i wyłącznie pliku, który mamy aktualnie otwarty. Możemy wtedy kliknąć prawym przyciskiem myszki na dowolnej zmianie i

zrobić **revert** całego commita, który uwzględnia otwarty plik.

Git reset

Wprowadzone zmiany można również cofnąć przy wykorzystaniu komendy `git reset`. Ta komenda jednak różni się od poprzedniej, bo `git reset` zmienia historię repozytorium. Komenda `git revert` cofała zmiany i dodawała to cofnięcie jako kolejny wpis w historii, czyli nie zmienialiśmy wpisów w historii, które już istnieją. Polecenie `git reset` jest o tyle inne, że przy jego wykorzystaniu modyfikujemy istniejącą już historię, czyli można powiedzieć, że zmieniamy ją wstecz. Do dyspozycji mamy dostępnych pięć wariantów tego polecenia, ale poruszymy 3 najczęściej używane:

- **soft** - użyjemy wtedy flagi `--soft`,
- **mixed** - użyjemy wtedy flagi `--mixed`, należy tutaj zaznaczyć, że jest to domyślny wariant,
- **hard** - użyjemy wtedy flagi `--hard`.

Przejdźmy po kolei przez każdą z możliwości.

soft

Ten wariant `git reset` można rozumieć jako **undo commit**, czyli cofnięcie commita, który został dodany. Flaga `--soft` przesuwa **HEAD**, czyli zmienia położenie aktualnego najnowszego commita na branchu, na którym się znajdujemy. Nie są modyfikowane w żaden sposób pliki, nad którymi pracujemy. Stosując `--soft` przesuwamy się wstecz na historii zmian i cofamy konkretne commity. Zmiany, które zostały wprowadzone po cofniętym commicie, nadal będą widoczne w repozytorium, jedynie nie będą one zacommitowane. Czyli `--soft` nie zmienia zawartości plików, zmienia jedynie historię zmian w **Git** i przenosi pliki z commita z powrotem do stanu **staged**.

Kiedy jej używać? Wyobraź sobie, że pracujesz na jakimś branchu i dodałeś/-aś tam 10 commitów. Orientujesz się, że możesz przecież z tego zrobić jeden commit, bo będzie to czytelniejsze. Możesz zatem cofnąć wykonane commity przy pomocy `--soft` i zamiast 10 commitów zrobić jeden. Zawartość plików nie ulegnie zmianie, modyfikujesz w ten sposób jedynie historię zmian.

Drugim przykładem, kiedy ta komenda będzie przydatna to wtedy gdy przez przypadek zacommitujemy plik, którego nie chcieliśmy uwzględniać w commicie. Możemy wtedy taki commit cofnąć i zrobić go jeszcze raz. Wykorzystanie:

```
git reset --soft a35bfdb
```

Jeżeli teraz wykonamy komendę `git status`, to na ekranie zauważymy taki fragment:

```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Dog.java ①
```

① Czyli klasa `Dog.java`, która była zmodyfikowana w cofanym commicie, została przeniesiona w status **staged**. Zmiany wykonane w tym pliku nadal są dostępne, jedynie cofnęliśmy wykonany commit z

historii zmian w repozytorium.

Drugim sposobem na wykorzystanie flagi `--soft` jest użycie wskaźnika **HEAD**:

```
git reset --soft HEAD^ ①
git reset --soft HEAD~1 ②
git reset --soft HEAD~2 ③
git reset --soft HEAD~3 ④
```

① Cofnij ostatni commit.

② Cofnij ostatni commit.

③ Cofnij ostatnie 2 commity.

④ Cofnij ostatnie 3 commity.

Po wykonaniu tych komend zobaczysz, że wycofana została ilość commitów adekwatna do polecenia i pliki, których dotyczyły cofane commity, znajdują się teraz w stanie **staged**.

mixed

Wykorzystanie flagi `--mixed` będzie różniło się nieznacznie od flagi `--soft`. Przy wykorzystaniu tej flagi nadal będziemy cofali commity, tyle że tym razem pliki po cofnięciu nie będą w stanie **staged**. Pliki takie zostaną przeniesione do **working directory** i będziemy musieli je ponownie dodać przy pomocy komendy `git add`.

Flaga ta będzie przydatna w sytuacji, gdy będziemy chcieli cofnąć commit i ponownie wybrać pliki, które będą miały znaleźć się w następnym commicie. Przypomnę, że do commita wchodzą pliki w stanie **staged**, ale musimy te pliki do tego stanu dodać, wykorzystując komendę `git add`. Czyli jeżeli wykonamy teraz:

```
git reset --mixed a35bfdb
```

A następnie wykonamy komendę `git status`, to na ekranie zobaczymy taki fragment:

```
Changes not staged for commit: ①
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Dog.java
```

① Jeżeli chcemy, żeby w tym momencie plik `Dog.java` znalazł się w kolejnym commicie, musimy go dodać komendą `git add`.

Flaga `--mixed` jest flagą domyślną - nie musimy zatem jej podawać, ale możemy. Należy również zaznaczyć, że w tym przypadku również zmodyfikowaliśmy tylko historię zmian w repozytorium, nie wpłynęliśmy na zmiany dokonane w plikach, nadal są one widoczne.

hard

Kolejnym wariantem komendy `git reset` jest możliwość dodania flagi `--hard`. Ten wariant nie dość, że

przesunie wskaźnik **HEAD**, to jeszcze zaktualizuje stan plików **staged** i stan **working tree**. Oznacza to, że nie dość, że cofniemy wybrane commity, to jeszcze usuniemy wszystkie zmiany, które zostały w tych commitach wprowadzone.

To jest wersja komendy `git reset`, która powoduje utratę wprowadzonych zmian, należy jej zatem używać z rozsądkiem.

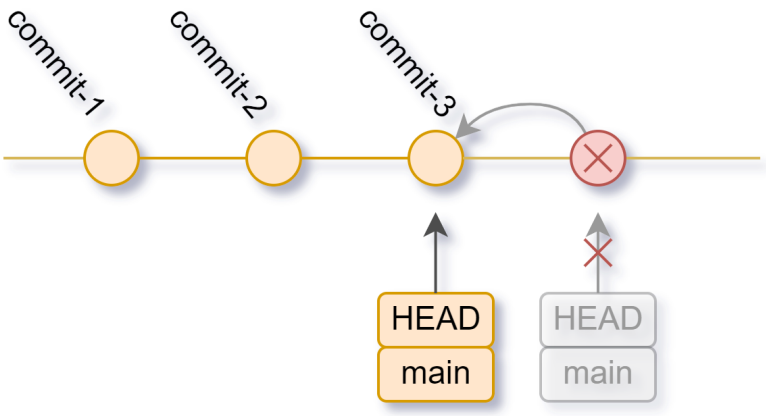
Wielu developerów przeszło w swojej karierze taką sytuację, że przez przypadek wywołali komendę `git reset` z flagą `--hard` i nieświadomie usunęli zmiany, nad którymi pracowali kilka godzin. No cóż - zdarza się. 😀

Ten wariant komendy będzie używany, gdy świadomie chcemy się pozbyć commitów razem ze zmianami, które zostały przez te commity wprowadzone. Należy tego polecenia używać z rozwagą, bo jeżeli usuniemy wprowadzone zmiany przez przypadek, to bardzo prawdopodobne jest, że będziemy musieli pisać ten sam kod od nowa.

Jednakże mogą wystąpić sytuacje, gdzie napiszemy jakiś kod, dodamy kilka commitów i uznamy, że nie jest nam to do niczego potrzebne i zwyczajnie będziemy chcieli się tego kodu pozbyć. Wtedy wykorzystamy `--hard`. Przykład:

```
git reset --hard HEAD^
HEAD is now at cb785f5 Initial commit ①
```

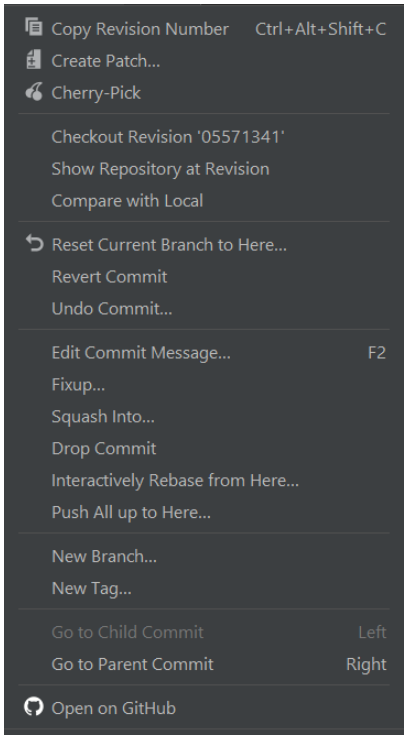
① **Git** informuje nas o przeniesieniu wskaźnika **HEAD**.



Obraz 3. git reset --hard

Intellij

Te same operacje możemy wykonać z poziomu Intellij. Wystarczy, że wejdziemy w zakładkę **Git** > **Log** i klikniemy w dowolny commit prawym przyciskiem myszki. Pokaże nam się wtedy to okno:



Obraz 4. Intellij i Git log

Możemy teraz wybrać opcje:

- **Reset Current Branch to Here** - tutaj będziemy mogli wybrać omawiane wcześniej flagi. Do tego Intellij wyjaśni ich znaczenie.
- **Undo Commit** - to zachowanie będzie analogiczne do `--soft`
- **Drop Commit** - ta opcja w prawdzie też spowoduje pozbycie się commita, natomiast Intellij nie wykorzysta tutaj pod spodem komendy `git reset`. Wykorzystane zostanie `git rebase`, o którym będziemy rozmawiać później.

Pobaw się każdą z opcji i zobacz jaką komendę wykonuje pod spodem Intellij. Możesz to sprawdzić w zakładce **Git** > **Console**.

Intellij pozwala nam na wykonanie `git reset` również z poziomu paska w górnej części ekranu - jest tam dostępna opcja **Git**. Jeżeli rozwiniesz tę opcję, możesz wtedy wybrać **Reset HEAD**. Intellij poprowadzi Cię wtedy przez kolejne kroki, żeby wykonać reset. Masz tam też możliwość kliknięcia znaku zapytania, który przekieruje Cię do pomocy.

Usunięcie pliku

Raczej jest to oczywiste, że plik możemy usunąć manualnie i zmiana taka będzie widoczna z poziomu **Git**. Możemy również wykorzystać **Git** do usunięcia pliku. Różnica jest taka, że jak wykonamy usunięcie pliku ręcznie (nie przez Intellij), to zmiana taka nie będzie dodana do **staging area**. Oznacza to, że będziemy musieli w takim przypadku wykonać jeszcze `git add`. Jeżeli natomiast wykonamy usunięcie przy wykorzystaniu komendy **Git**a - usunięcie takie będzie automatycznie oznaczone jako **staged**. Wystarczy wykonać komendę:

```
git rm Dog.java
```

Wycofanie zmian non-committed

Zwróć uwagę, na to, że cały czas rozmawialiśmy o wycofywaniu zmian, które są **commit**ted. A jak podejść do wycofania zmian, które nie zostały zacommitowane?

git reset

Możemy wykonać komendę:

```
git reset --hard
```

W takim przypadku wszystkie pliki, które są w stanie **non-staged** lub **staged** zostaną odrzucone.

Git checkout

W przypadku gdy dodaliśmy jakieś zmiany w plikach, które już były zacommitowane i chcemy przywrócić oryginalny stan tych plików, możemy wykonać również komendę:

```
git checkout .
```

Intellij

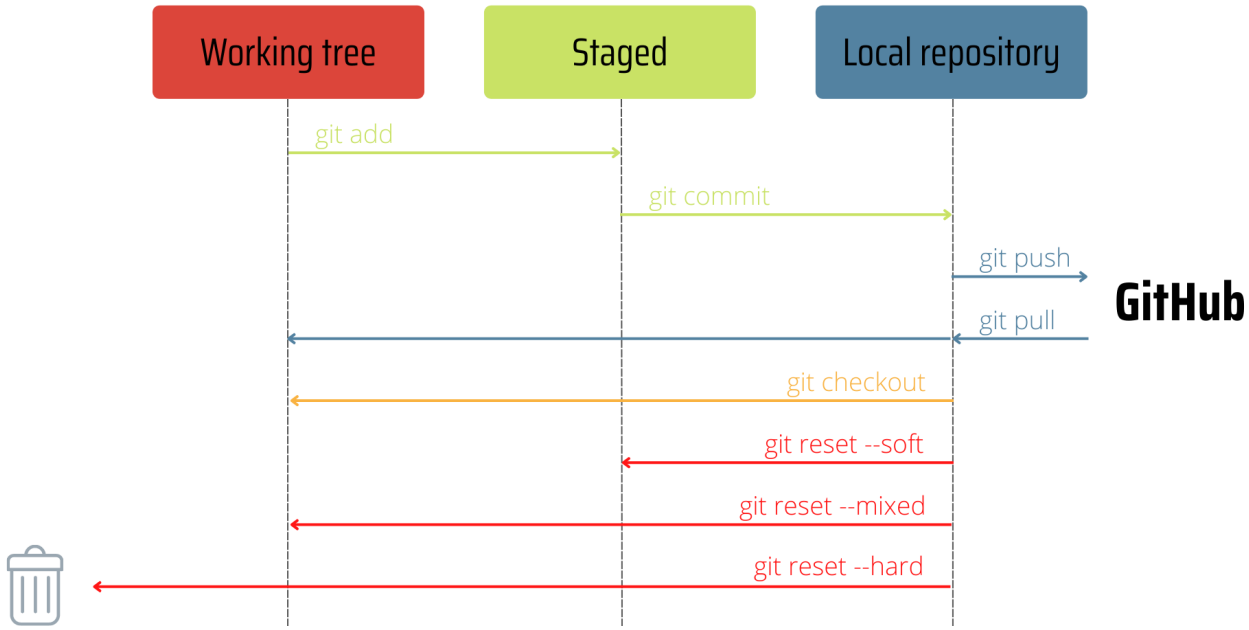
Troszkę inaczej będzie to wyglądało, jeżeli chcielibyśmy wycofać tego rodzaju zmiany przez Intellij. W tym przypadku możemy przejść do zakładki **Git** > **Local Changes**, zaznaczyć pliki, które nas interesują, kliknąć prawym przyciskiem myszy i wybrać **Rollback**.

Możemy również zrobić to samo z poziomu konkretnego pliku, wykorzystując skrót `Alt + `` i wybierając opcję **3**.

Podsumowanie

Widzisz na tym etapie, że mamy kilka możliwości, żeby wycofać dodawane przez nas zmiany. Każdy ze sposobów różni się w zależności od tego, czy plik jest zacommitowany, czy nie. Cały czas poruszamy się też w obrębie repozytorium lokalnego. Ma to o tyle znaczenie, że dopóki zmiany nie wyjdą poza repozytorium lokalne, możemy do nich podchodzić inaczej, niż do zmian, które znalazły się już w repozytorium zdalnym. Dopóki zmiany są tylko w repozytorium lokalnym - możemy spokojnie bawić się na nich wykorzystując `git reset`. Gdy zmiany znajdą się już w repozytorium zdalnym - będzie to bardziej problematyczne i łatwiej będzie wtedy zastosować `git revert`. Jedno i drugie polecenie różniło się też wpływem na historię zmian, `git reset` modyfikowało istniejącą historię, a `git revert` dodawało kolejne wpisy do historii.

Wrócimy jeszcze do tego tematu, gdy przejdziemy do pracy z repozytorium zdalnym, bo tam przy wykorzystaniu komendy `git reset` trzeba będzie uważać. W niektórych przypadkach użycie jej nie będzie nawet możliwe. Żeby podsumować przedstawione wcześniej możliwości, spójrz na poniższą grafikę.



Obraz 5. Git reset