Git - Stash

Spis treści

Odkładanie zmian na potem 1
Intellij 3
Podsumowanie terminów 5

Odkładanie zmian na potem

Wyobraź sobie, że pracujesz na jednym branchu nad jakąś częścią projektu, nad nowym featurem. Dodajesz linijki kodu, modyfikujesz istniejące klasy i przychodzi taki moment, że musisz przełączyć się na innych branch, a jednocześnie wypracowane przez Ciebie zmiany nie są dopracowane do tego stopnia, żeby móc je zacommitować. Często jest tak, że nie udało Ci się skończyć danej funkcjonalności, a potrzebujesz zwyczajnie przełączyć się na inny branch. Masz wtedy dwie możliwości:

- dodajesz commit do brancha, ze świadomością, że potem trzeba będzie zrobić `git reset --soft`,
- odkładasz wprowadzone zmiany na potem, czyli wykonujesz `git stash`.

Wykonanie `git stash` może być rozumiane jako odłożenie niedokończonych i niedopracowanych funkcjonalności na półkę, na potem. Często do określenia niegotowych zmian stosuje się określenie **dirty**. Zmiany takie, które zostały odłożone na potem, mogą zostać odtworzone na dowolnym branchu później. Trzeba jednak pamiętać, że jeżeli te zmiany leżały na półce, a my zaczęliśmy w tym czasie edytować te same pliki i zmieniać ich zawartość - może to doprowadzić do konfliktów, podczas gdy będziemy chcieli te zmiany z półki zabrać.

Przy stosowaniu tej komendy należy pamiętać, że nieskończone zmiany odkładane są na **stosie** nieskończonych zmian. Zobaczysz, o co w tym chodzi, jak przejdziemy do przykładów.

Wprowadź kilka zmian w projekcie, dodaj niektóre zmiany do **stejdża**, czyli zrób `git add`, ale ich nie commituj. Następnie wykonaj komendę:

```
git stash
# lub
git stash push
```

Na ekranie zostanie wtedy wydrukowana wiadomość:

```
Saved working directory and index state WIP on main: 9d184c2 Dog
```

W tym momencie zmiany, na których pracowałeś/-aś zostały odłożone na później i możesz zwrócić uwagę, że Twoje working tree jest puste - nie ma tutaj żadnych zmian. Jeżeli wykonasz teraz `git status`, to zobaczysz wiadomość:

```
nothing to commit, working directory clean
```

Możemy teraz ponowić powyższą czynność kilka razy. Dodaj zatem kolejne zmiany, dodaj parę commitów i wykonaj `git stash` ponownie. Powtórz czynność kilkukrotnie. Następnie wykonaj polecenie:

```
git stash list
```

Na ekranie zostanie wydrukowane coś analogicznego do:

```
stash@{0}: WIP on main: 9d184c2 commit-1-name
stash@{1}: WIP on main: 9k20al1 Revert commit 3
stash@{2}: WIP on main: al2ka01 Add data source
```

Zmiany odłożone jako najstarsze będą miały najwyższy numer i będą na samym dole. Najnowsze odłożone zmiany będą miały numer **0** i będą na samej górze. Stąd właśnie pojawiło się wcześniejsze porównanie do stosu.

W dowolnym momencie na dowolnej gałęzi możesz przywrócić zmiany odłożone na potem. Do tego celu możesz wykorzystać komendę:

```
git stash apply ①
# lub
git stash apply stash@{2} ②
```

① W tym przypadku zostanie przywrócona najnowsza odłożona pozycja, czyli ta z numerem **0**.

② W tym przypadku sami określamy, która pozycja ze stosu ma zostać przywrócona.

Po wykonaniu tego polecenia, na ekranie zostanie wydrukowane wiadomość analogiczna do:

```
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   Dog.java
    modified:   src/Cat.java

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

Git na podstawie informacji odłożonych przez `git stash` stara się przywrócić stan plików odłożonych na stosie na potem. Jeżeli mieliśmy puste working directory, czyli nie mieliśmy wykonanych żadnych zmian, "byliśmy na czysto", **Git** przywróci zmiany odłożone na stosie. Możemy to zrobić na tym samym albo na innym branchu. Możemy to zrobić również, gdy już edytowaliśmy jakieś pliki i nie zostały one jeszcze zacommitowane. Jeżeli **Git** nie będzie w stanie odtworzyć zmian automatycznie - dostaniemy **merge conflict**

Zwróć uwagę, że po wykonaniu `git stash apply` zawartość stosu nadal jest taka sama. W jaki sposób

możemy zatem usunąć wpis ze stosu? Możemy wykorzystać komendę:

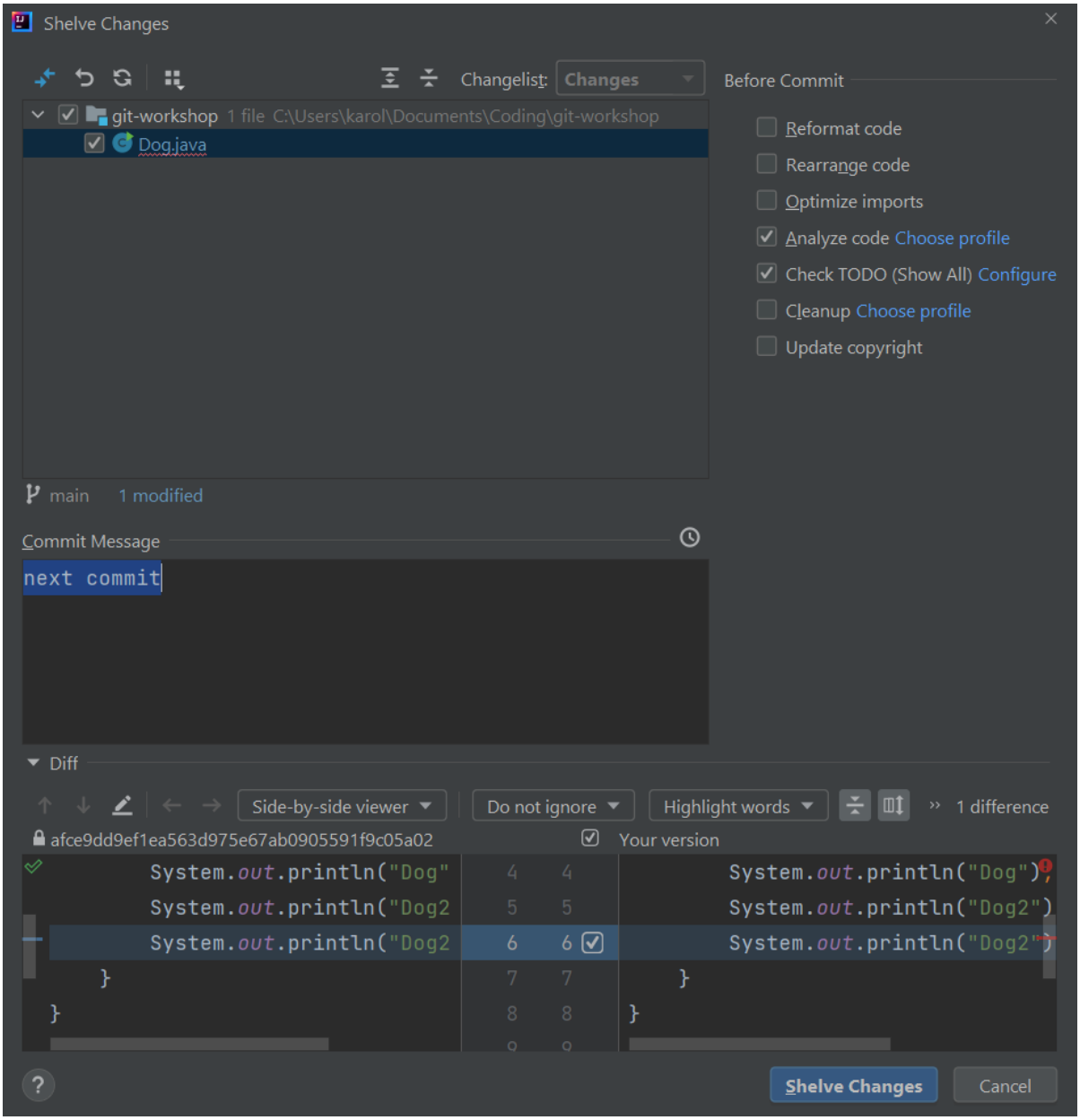
```
git stash pop
```

Komenda ta zadziała jak złączenie `git stash apply` i jednoczesnego usunięcia wpisu ze stosu. Możemy również taki wpis usunąć przy wykorzystaniu komendy:

```
git stash drop 'stash@{0}'
# lub jeżeli powyższe nie działa
git stash drop 'stash@{2}'
```

Intellij

Intellij również daje nam mechanizm odkładania zmian na półkę, natomiast nosi on lekko inną nazwę. Jeżeli przejdziesz do zakładki **Git** > **Local Changes**, zaznaczysz kilka plików i klikniesz prawy przycisk myszy, to pojawi Ci się opcja **Shelve Changes...**. Jeżeli ją wybierzesz, to otworzy Ci się okno podobne do okna **commit** w Intellij:



Obraz 1. Intellij Shelve Changes

Możesz tutaj zapisać wiadomość, pod którą następnie będziesz w stanie wyszukiwać zmian odłożonych na półkę. Reszta opcji jest analogiczna do opcji, które daje nam Intellij, gdy robimy **commit**. Zwróć uwagę, że przycisk zatwierdzający ma napisane **Shelve Changes**. Gdy go klikniesz i przejdziesz do zakładki **Git** > **Console** to zauważysz, że nie została wykonana komenda `git stash`. Dlatego napisałem, że jest to mechanizm analogiczny, ale jednak nie ten sam. Odłożone na półkę zmiany są teraz widoczne w zakładce **Git** > **Shelf** pod nazwą, która została przez Ciebie nadana przed chwilą.

Możesz teraz wybrać konkretną nazwę reprezentującą zmiany odłożone na półkę, kliknąć prawy przycisk myszy i wybrać opcję **Unshelve**. Dzięki temu Intellij będzie mógł przywrócić odłożone na półkę zmiany. W tym przypadku możemy ręcznie wybrać które zmiany przywracamy. Należy jednak pamiętać, że jeżeli Intellij nie będzie w stanie przywrócić zmian automatycznie - wyświetli nam się okno do rozwiązania **merge conflicts**. Gdy rozwiążemy je ręcznie, nasze zmiany zostaną przywrócone.

Intellij ma również dostępną opcję **Stash Changes**, którą możesz znaleźć, gdy wybierzesz skrót `Alt + `` i numerek **9**.

Podsumowanie terminów

Z racji, że omówiliśmy już dużo zagadnień, chciałbym, żebyśmy sobie podsumowali terminy, które się dotychczas pojawiły:

Słowo	Znaczenie
Repository	Repozytorium zawiera historię zmian projektu w czasie. W **Git** każda kopia repozytorium jest kompletnym repozytorium. Repozytorium pozwala na zapisywanie kolejnych zmian w postaci **commit**ów.
Working tree	Stwierdzenia **Working tree** lub **Working directory** używa się do określenia katalogu, w którym pracujemy nad plikami, które reprezentują projekt w danej wersji. W tym katalogu znajdziemy pliki na dysku, na których możemy pracować, czyli możemy je dodawać, usuwać albo modyfikować.
Staging area	Stwierdzenia tego używa się do określenia miejsca, do którego dodajemy zmiany, z których następnie ma zostać stworzony **commit**.
Index	Ta nazwa jest używana jako alternatywa do **Staging area**.
Commit	**Commit** identyfikuje zmianę treści projektu w sposób unikalny. Jeżeli chcemy przejrzeć konkretne zmiany w projekcie, które wystąpiły w czasie - będziemy przeglądać konkretne commity. Commit ma przypisanego autora, unikalny identyfikator, czas powstania oraz konkretne zmiany w plikach.
Revision	Stwierdzenia tego używa się do oznaczenia konkretnej wersji kodu źródłowego. Można powiedzieć, że **Git** implementuje rewizje przy wykorzystaniu commitów.
Branch	Branch reprezentuje niezależną linię rozwoju projektu. Gałęzie mogą być rozumiane jako warstwa abstrakcji służąca do zarządzania commitami. Branche umożliwiają rozwidlanie się historii projektu, dzieki czemu możemy mieć wiele przestrzeni roboczych niezależnych od siebie.
HEAD	To symboliczna nazwa używana do wskazania aktualnie wybranej gałęzi. HEAD może wskazywać na branch albo na konkretny commit - mówimy wtedy, że repozytorium znajduje się w stanie Detached HEAD.