



M07 PRACA ZESPOŁOWA

L04 Modyfikacja historii

(History was written by the winners)

Maciej Aniserowicz

W Gicie historia jest w pełni modyfikowalna

...co BARDZO przydaje się przy pracy lokalnej...

...ale **nie tylko!!**

Kiedy można modyfikować historię?

Przed PUSH - Lokalnie - zawsze bezpieczne

- Poprawka commit message
- Porządek w historii przed wysłaniem commitów do zdalnego repo

Po PUSH - Zdalnie - **powinno** być bezpieczne

- Praca na własnej "prywatnej" gałęzi (o tym za chwilę)

Kiedy nie można modyfikować historii?

Po PUSH - Zdalnie - poza własną dedykowaną gałęzią.

Niebezpieczne! UNIKAĆ!!!

Co się stanie, gdy wyślemy nadisaną historię?

Stan naszego repozytorium będzie zgodny ze stanem repozytorium zdalnego... ale wszyscy inni będą musieli ściągnąć nową wersję i się z nią zintegrować!

Kiedy trzeba modyfikować historię?

Czasami nie da się uniknąć. Przypadki:

- usunięcie **wrażliwych** danych z repozytorium
 - hasło / connection string / plik z kluczem
- **porządek** w historii (na przykład po nieudanym merge / przedwczesnym commit)

Co robić?

Zakomunikować w zespole! Umówić się, że o danej godzinie nastąpi "zepsucie" historii i później **wszyscy** pobiorą nową wersję.

BEST PRACTICE

Nie modyfikuj historii już wysłanej na serwer,
chyba że to **ABSOLUTNIE KONIECZNE**.

(albo pracujesz w pojedynkę)

(albo masz 100% pewności, że nikt jeszcze nie
pobrał Twoich poprzednich zmian)



Zabezpieczenia

Konfiguracja Gita (serwer):

- *receive.denyNonFastForwards* - nie przyjmuje modyfikacji historii

Konfiguracja dla całego repozytorium, więc bezpieczne, ale nieco ograniczające...

Uprawnienia per gałąź?

Usługi **hostingowe** (GH, GL, BB) dają kontrolę nad takimi rzeczami na poziomie gałęzi.

Ale **nie musimy** trzymać kodu na zewnątrz, żeby wyłączyć modyfikację historii na poziomie wybranej gałęzi!

Do tego dojdziemy wkrótce.



Różne sposoby

Kilka sposobów modyfikowania historii

- revert
- amend
- reset
- rebase
- filter-branch

Revert

Pamiętasz jak mówiłem przy "*interactive add*", że opcja "revert" jest niefortunnie nazwana? No właśnie... istnieje osobna komenda "*git revert*"! I robi coś zupełnie innego.

To nie jest pełna "modyfikacja historii".

Chodzi o **odwrócenie**, a **nie wycofanie** zmian.

Operacja w **100% bezpieczna**: tworzony jest **nowy commit**, odwrotny do odwracanego.

Revert DEMO

- 
1. edit index.html
 2. git commit . -m "added unwanted change"
 3. git revert @
 4. :q
 5. F5
 6. gitk

Revert - rekommendowany sposób

Efekt osiągnięty.

I zachowujemy **poprawny stan** repozytorium.

Amend

Modyfikacja ostatniego commita.

Zastosowania:

- modyfikacja **zawartości** ostatniego commita
 - dodanie nowego pliku, kolejnej zmiany itd
- modyfikacja **commit message**

Amend DEMO

- 
1. edit index.html
modify TITLE
 2. git commit . -m "changd title of website to something better"
literówka!
 3. git commit --amend
poprawka literówki
 4. :q

Amend modyfikuje historię

Commity stworzone przez Gita są **niemodyfikowalne**.

Zawsze w miejsce starego jest tworzony nowy.

Amend wstawia **nowy commit w miejsce starego**

- w historii zmienia się SHA1
- historia jest zmodyfikowana!

Nie używać po PUSH!

Amend - uwaga na tagi!

Amend tworzy nowy commit, a zaaplikowany **tag zostaje na starym!**

1. git tag tag_amend_test
2. gitk
3. git commit --am
4. gitk --all

Trzeba przenieść tag, usuwając stary i tworząc nowy.

„Wtrącenie

COMMIT DRIVEN DEVELOPMENT?

Najpierw commit, potem kod.

Najpierw commit, potem kod. Jak?

W **pustym** commicie piszemy, co będziemy robić.

A dopiero potem **dodajemy** do niego kolejne zmiany.

Zmienia sposób myślenia!

Niemalże gwarantuje spójne commity - widząc commit message, nie będziemy dodawali niepasujących zmian. Warto spróbować.

"Commit Driven Development" **DEMO**

- 
1. git commit --allow-empty -m "Changed links to polish"
 2. git commit . --amend

Reset

Komenda pojawiała się już wielokrotnie. **Bez argumentów**: wycofuje zmiany ze staging area...

Ale z **argumentem <commit>** "przewija" HEAD do podanego miejsca
(zmienia wierzchołek aktualnej gałęzi)

Nie jest to modyfikacja historii - bo **nie zmienia commitów** - ale zmienia wskaźnik HEAD.

Jeżeli zmiana jest niekompatybilna ze zdalnym repo: nie będzie można wykonać PUSH.



Rebase

Rebase

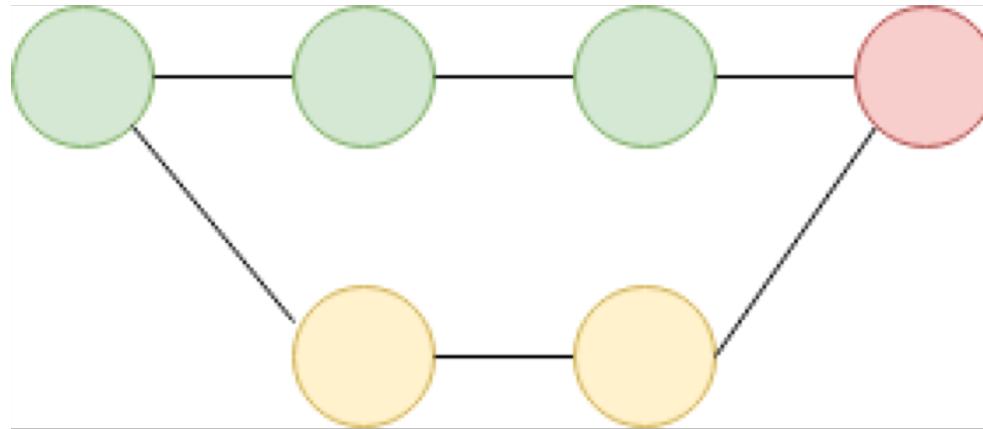
Świetna i bardzo **przydatna** komenda!

"**Nakłada**" jedną gałąź na drugą gałąź.

Jak to wygląda?

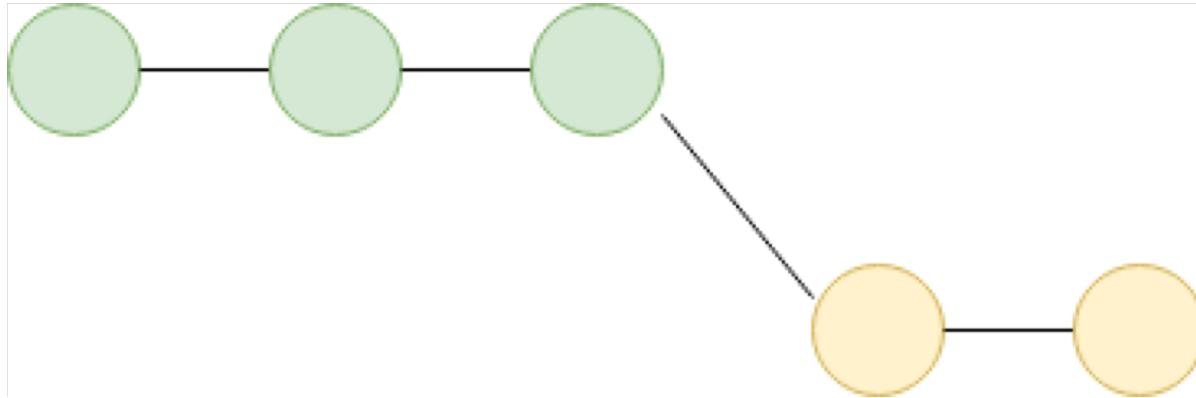
Rebase vs merge (1/2)

Merge - łączy dwie gałęzie



Rebase vs merge (2/2)

Rebase - przenosi zmiany, budując liniową historię



Rebase vs merge DEMO (1/3)

1. git branch

reading-list-subpage

2. gitk --all

ta gałąź powstała dawno temu! (w kursie nic nie jest przypadkowe ;))

Dołączenie jej do *mastera* spowoduje integrację ze wszystkimi commitami powstającymi dotychczas... oraz bałagan w historii!

Rebase vs merge **DEMO (2/3)**

Zobaczmy merge:

1. git merge reading-list-subpage
2. gitk --all

czytelnie? NIE!

więc...

3. git reset --hard ORIG_HEAD

Rebase vs merge **DEMO (3/3)**

Zobaczmy **rebase**:

1. git checkout reading-list-subpage
2. git rebase master
3. F5

zmiany nadal widoczne

4. gitk --all
- jaka piękna historia!

Dzięki **rebase**...

Możemy zbudować **czytelną, liniową historię**.

Jak dokładnie? O tym już za chwilę.

(a może pamiętasz z poprzednich lekcji?)

Wróćmy na chwilę do **PULL**

Co robi pull?

git pull = fetch + merge

Nieprzygotowany merge powoduje zagmatwaną historię. Alternatywa:

git pull --rebase = fetch + rebase

(choć przy rekomendowanym sposobie pracy - o czym zaraz - nie ma to znaczenia)

„ Ciekawostka

"Pod spodem" operacja rebase to po prostu zestaw **cherry-pick** na wybranych commitach.

. Jeden . Po . Drugim .

Rebase interactive

To powinna być osobna komenda :)

Pozwala na **wizualne** przeglądanie i modyfikowanie historii

git rebase -i <commit>

- modyfikacja commitów po podanym punkcie w historii

Rebase interactive DEMO

1. git rebase -i @~5

p - bez zmian

r - zmiana message

e - edycja commita

s - złączenie commitów

2. d "changed title of books page"

3. F5 - przywrócony stary tytuł

4. git reset --hard ORIG_HEAD

5. F5 - ponownie widoczny nowy tytuł

Rebase interactive - zastosowania

- porządek w historii przed wysłaniem jej na serwer
- rozbicie commita na kilka mniejszych
- poprawka w commit message w nie-ostatnim commicie

Rebase **ABORT?**

Tak jak w merge:

- *git rebase --abort*
- *git reset --hard ORIG_HEAD*

Rebase - uwaga na **tagi!**

Rebase (tak jak *commit --amend*) tworzy **nowe commity**.

Tagi zostają na starych commitach!

Potencjalne rozwiązanie:

- *git rebasetags* (zewnętrzny nieoficjalny skrypt, używać **z rozwagą**, przetestować!)
<http://kursgita.pl/rebasetags>

Rebase - uwaga na gałęzie!

Jeśli robimy *rebase mastera*, to musimy **ponownie wykonać *rebase* wszystkich pozostałych gałęzi!**

(choć rebase gałęzi master, przy odpowiednich praktykach, powinien zdarzać się bardzo rzadko... lub nigdy)



2 bonusy

Repo Cleaner

Narzędzie usuwające duże/problematyczne pliki z całej historii.

Przepisuje całą historię, kasując z każdego commita wybrane pliki.

<http://kursgita.pl/repocleaner>

filter-branch

Bardzo zaawansowana komenda, która może **WSZYSTKO**.

Poza zakresem tego kursu... ale teraz poradzisz sobie z dokumentacją.

Piszemy **skrypt** aplikowany do każdego commita z podanego zakresu.

filter-branch - zastosowania

- zmiana zawartości commita
 - usunięcie plików (jak Repo Cleaner)
 - dodanie plików
 - zmiana struktury katalogów
- zmiana daty (a nawet autora!) commita
- zmiana commit message (np dodanie wspólnego prefixu)
- ... bez żadnych ograniczeń



push po modyfikacjach

Wysłanie zmodyfikowanej historii

git push --force

Wysyła commity na zdalny serwer;
nawet jeśli **nie są kompatybilne** z docelowym repozytorium!

(wspominaliśmy, że można wyłączyć akceptację takich commitów na
serwerze: *receive.denyNonFastForwards*)

Force jest złe!

--force to **bardzo agresywna praktyka!**

Nadpisujemy zdalne repozytorium, do końca **nie wiedząc co nadpisujemy**.

Lepiej (jeśli już koniecznie trzeba):

git push --force-with-lease

Stara się upewnić, że mamy pobraną (*fetch*) najnowszą wersję zdalnego repozytorium. Więc **wiemy co nadpisujemy**.

BEST PRACTICE

Nigdy nie używaj

- git push **--force**

Zamiast tego zawsze (jeśli w ogóle):

- git push **--force-with-lease**



Podsumowanie

- git revert
- git commit --amend
- git rebase
- git pull --rebase
- git rebase -i
- git config receive.denyNonFastForwards
- git push --force
- git push --force-with-lease



ZADANIE: ROZBICIE COMMITA

Commit dodający zawartość do strony "books.html" na dwa:

- dodający nagłówek
- dodający listę

Przydatne komendy (kolejność **losowa**):

- git reset
- e
- git commit
- git rebase -i

