Gerrit

1. Co to jest Gerrit ?

   To jest SCM (Source Code Mangement platform) tak jak Github, BitBucket ect.

   Opiera się na VCS Git

2. Czemu używać gerrit ?

   Pozwala na opiniowanie commitów, zanim zostaną zaakceptowane do repozytorium git

   Pozwala na sprawdzenie, że każdy kod przechodzi przez kontrole jakości

3. Code review w Gerrit

   Składa się z:

   1. Comments
   2. Approvals/Rejections
   3. Code quality verification
   4. Additional patches to fix issues
   5. Diff between version
   6. Dependencies
   7. Change status

   Code review składa się z 2 etaptów:

   1. Subminitng code changes to gerrit (zmiany są przechowywane w staging area) gdzie następnie są opiniowane (przez innych, oraz tesotwane poprzez różne narzędzia)
   2. Jeżeli kod jest zatwierdzony, to zostaje no zmarge’owany

Terminologia:

1. Change - commit, któy jest upiniowany. Jest identyfikowany poprzez change-id

2. Patchset - kolejne commity, które są próbami poprawienia istniejącej zmiany

Administracja

1. Tworzenie nowego repozytorium → Browse → Repository → Create new
2. Branches and git tags
    a. Wchodzimy do repozytorium - prawa zakładka - branches - create new
    b. Wchodzimy w repozytorium - prawa zakładka - tags - create new
3. Acces controll
    a. References
    b. Access Categaory
    c. Exclusive
    d. User/Group and Acces Rule
4. Submit Types
    a. Opcja ta jest dostępna w zakłądce - Repozytorium - general - Submit type
    b. Definuje ona, metodę jaka jest używana do wdrażania zmiany do projektu, w przypadkuy gdy docelowy branch jest w stanie changed (head reference ozmienił się poprzez mergowanie lub commitowanie)

Dostępne opcje:

1. Fast Forward - Przed wdrożeniem należy zrobić rebase, czyli jeżeli będą jakieś konflikty, należy je roziwązać, żeby wypchać commit do review. Jest to dobra opcja w przypadku kodu, ktory nie ma ogromnych zmian - w tym przypadku, trzeba by było robić często rebase.
2. Merge if necessary - opcja lepsza, gdy jest dużo zmian. (niby jest domyślną i rekomendowaną strategią w gerrit)
3. Rebase if necessary
4. Rebase always
5. Merge always

6. Cherry pick

User settings

Ciekawsze:

1. Notification about repo change

Pushing code to Gerrit

1. Commit-msg hook

Jest to skrypu, który jest wywoływany podczas tworzenia git commit

Musi on być wyciągnięty (pull) z gerrit, i umieszczony w loklanym repozytorium git dla każdego clone. Ten hook automatycznie ustawia unikalny change-id w footerze commit message git.

Gerrit używa tego chage-id, do śledzenia commitów pomiędzy cherry pick i rebase.

Pozwala powiązać kolejne wersje tej samej zmiany (tzw. patchsety) w jednym miejscu, dzięki czemu Gerrit wie, że są to poprawki dotyczące tego samego zgłoszenia.

Jeśli Change-Id już istnieje w komunikacie, hook go nie nadpisuje.

Workflow

1. Pobranie repozytorium
2. Pobranie hook’ów (poprzez scp, curl ect.)
3. Upewnienie się, że hooki są executable dla user’a
4. Przechodzimy na odpowiedniego brancha
    a. Dodajemy pliki
    b. dodajemy commit’y → tutaj hook dodaje change-id automatycznie + nas komentarz
5. Git push do naszego brancha zdalnego (branch zdalny dev)

a. → git push orgin HEAD:dev

6. Git push for code review
    a. Powoduje to powstanie nowego okna change revision w Gerrit UI, pozwalając innym na zmiany, review, komentarze, opiniowanie wcelu zmergowani'a brancha. Komendy (pierwsza skrócona, druga z podaniem pełnej ścieżki). Jest to komenda do wysyłania patchsetu do opiniowania.
    b. → git push orgin HEAD:refs/for/<branch-name>
    c. → git push orgin HEAD:refs/for/refs/heads/<branch-name>

Wprowadzanie poprawek

git commit --amend --no-edit

git push origin HEAD:refs/for/<branch-name>

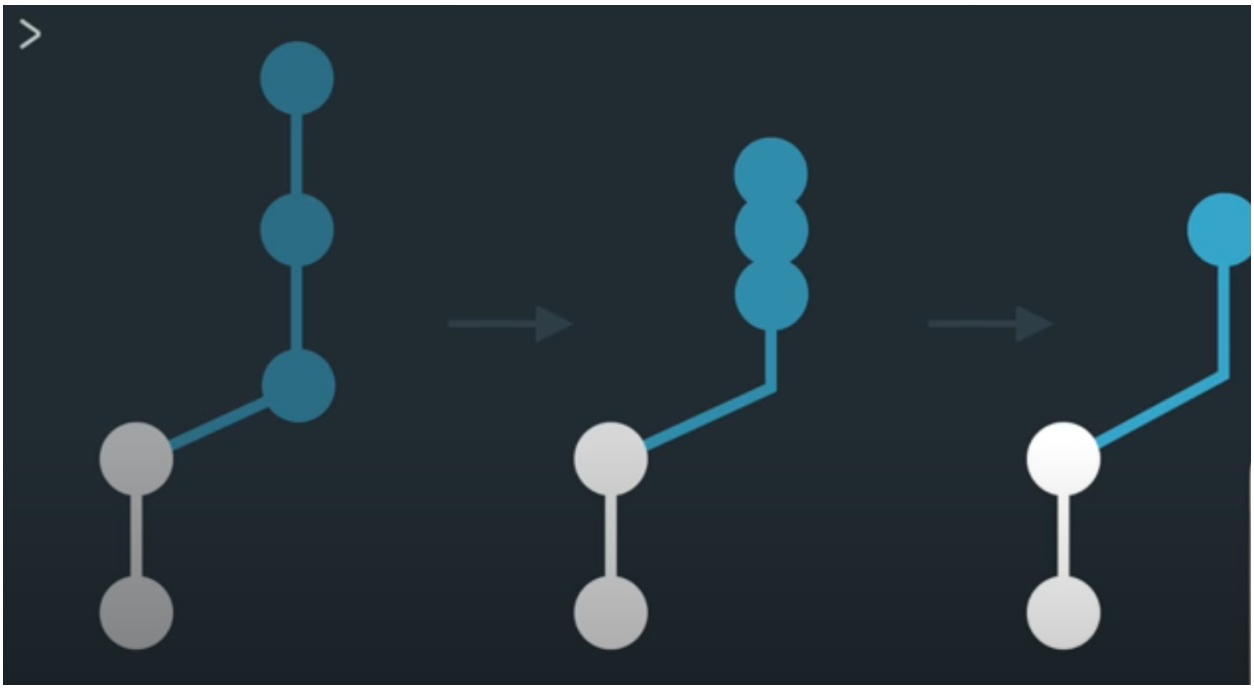
Czy widać pośrednie commit'y czy tylko efekt końcowy jest wypychany do recenzji ?

Domyślnie, każdy commit jest jako osobny patchset\

Stosowanie amend (niezalecane)

Jeżeli np. mamy 20 commitów, i chcemy je scalić w 1 do wypchania jako patchset, to możemy zrobić rebase -i ~<ilość commitów> (komenda, która scala lokalne commity w mniejszą ilość) z pomocą komendy squash (tam gdzie są commity pośrednie, zamieniamy pick na squash w edytorze tekstowym) po zapisaniu otrzymamy plik, w któym definujemy squash commit message (możemy symbolem # stare commity zignorować)

```
1 pick ee8ce13 Accessibility fix for frontpage bug
2 squash 4673264 Updated screenreader attributes
3 squash 5c4c543 Added comments & updated README
```



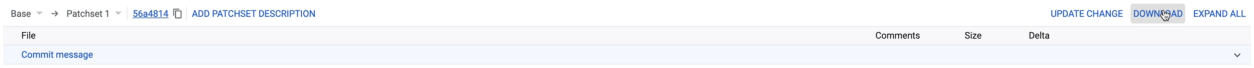
Wypychanie patchestu do istniejącego code review - wprowadzanie poprawek

W przypadku, gdy dostaliśmy już feedback, o istniejących problmach w pierwszym patchsecie, należy wprowadzić poprawki, i do istniejącego code review wprowadzić nowy patches. Czynność powtarzamy, aż do zadowolenia recenzentów.

```
git fetch https://<Gerrit-Server>/<project> refs/changes/12/34567/8
git checkout FETCH_HEAD
git add <path-of-reworked-file>
git commit --amend
git push origin HEAD:refs/for/<branch-name>
```

Postępowanie:

1. Wchodzimy na pasku w zakładkę download:



I wybieramy komendę do checkout:

Wprowadzamy zmiany, zawsze następnie wykonując ammend

Wypychanie zmian, tak jak ostatnio:

1. → git push orgin HEAD:refs/for/<branch-name>

WIP - Work in Progress Changes

1. WIP changes nie wymagają akcji z strony reviewer'a
2. Reviewerzy nie są informowani o o zmianie
3. Zmiania nie jest pokazywana w reviewers' dashboard

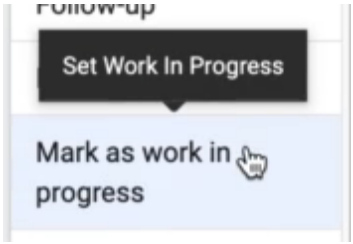
Kiedy używać:

1. Kiedy została zaimplementowana tylko część kodu
2. Jeżeli chcesz wypchać, ale w pierwszej kolejności chcesz sprawdzić czy przechodz CI testy (recenzenci nie muszą o tym wiedzieć)
3. Chcesz wprowadzić zmiany bez informowania o tym recenzentów

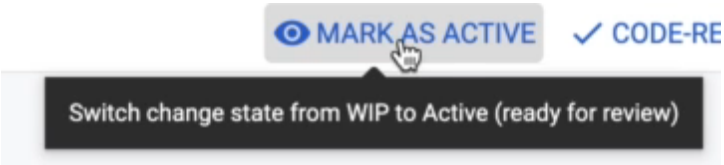
Jak to zrobić ? → Z poziomu GUI:

1. Symbol hamburgera

2. Set Work in Progress



3. Jeżeli chcemy wrócić do powiadamiania recenzentó (Mark as active)



Jak to zrobić ? → Dodając do git push: dodajemy na końcu %wip

1. → git push orgin HEAD:refs/for/<branch-name>%wip

następnie, aby zdjąć wip, należy dodać: %ready

1. → git push orgin HEAD:refs/for/<branch-name>%ready

Private changes

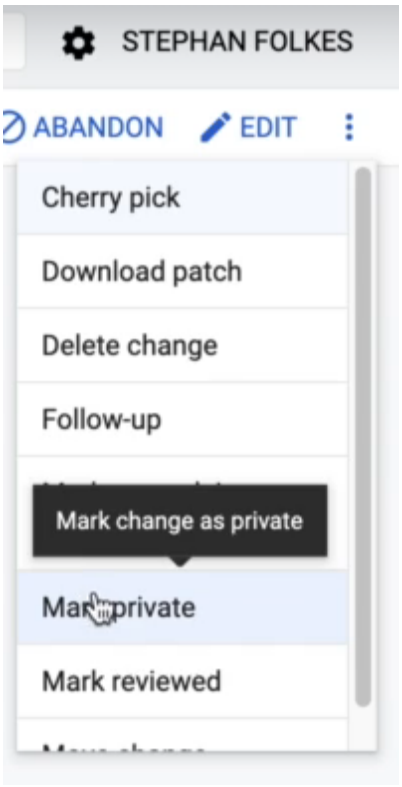
Private changes are only visible to the owner and reviewer assigned to the change

Users with global privileges as abe to view private changes (w zależności od konfiguracji)

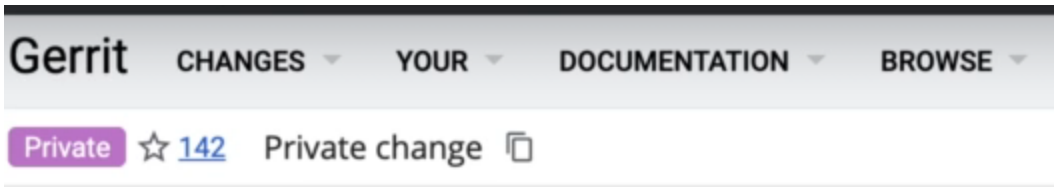
Przydatne gdy:

1. Chce się zobaczyć jak wygladałby push, bez widoczności dla innych
2. Jeżeli chce się pokazać określonym osobom, przed docelowymi oceniającymi

Opcja dostępna z poziomu gui:



Zmienia się wtedy napis na pasku:



W tym samym rozwiijanym menu, można zmienić na unprivate

Na etapie wypychania repo do review: %private

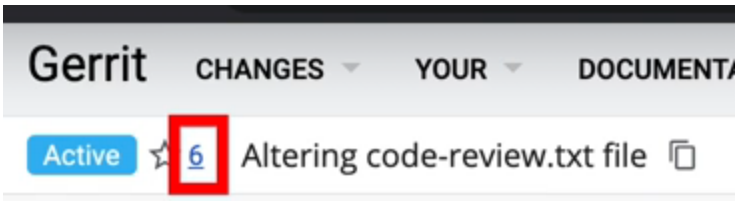
1. → git push orgin HEAD:refs/for/<branch-name>%ready

Aby zmienić, wystarczy dodać %remove-private

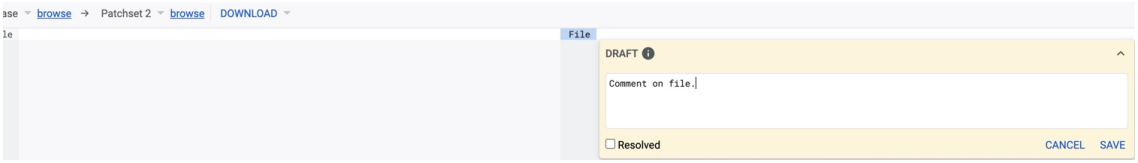
1. → git push orgin HEAD:refs/for/<branch-name>%remove-private

Using the change screen

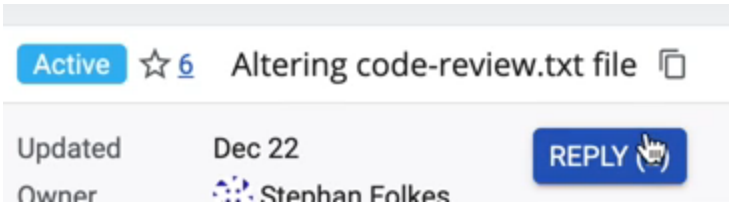
1. Identyfikator

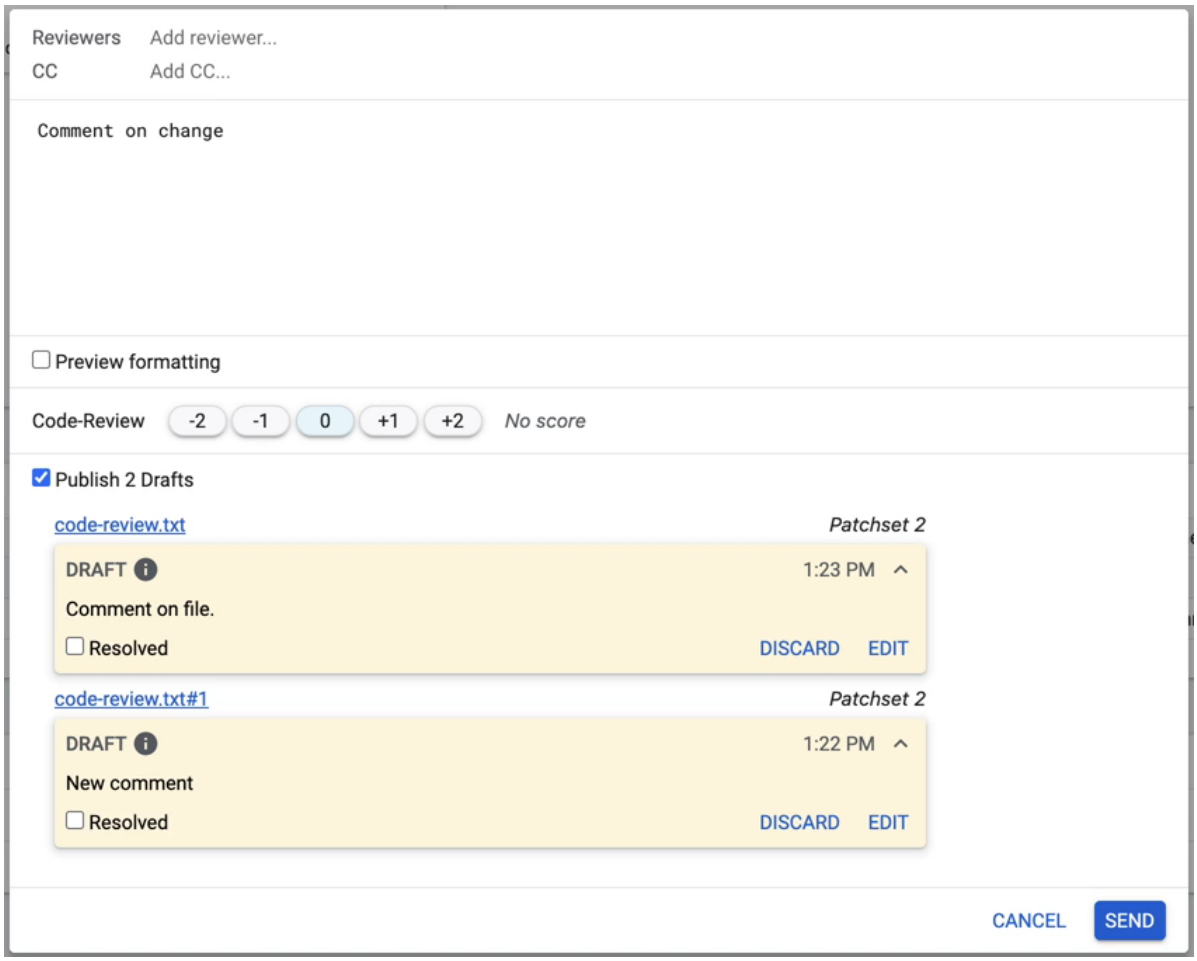


2. Komentarze:
    a. Inline w GUI - poprzez zaznaczenie kodu
    b. Dla całego dokumentu: Poprzez kliknięcie File na środku panelu:



    c. Poprzez przycisk review (wspiera md i ma możliwość wystawwiania score)

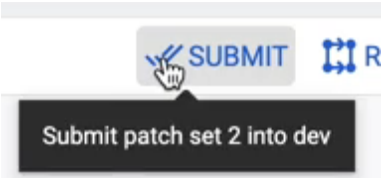




Submit

Oznacza wdrożenie swojej zmiany do wybranego brancha.

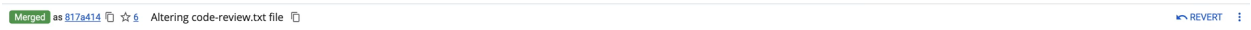
Wymagana jest permisja do tego, oraz odpowiednia ilość pozytywnych code-review (i nie może być ujemnych)



Jeżeli submitowanie powoduje konflikt, należy wykonać rebase, rozwiązać konflikty i wypuścić nowego patchset

Revert a submitted change

Mamy dostępną taką opcje, to udanym submicie, kiedy kod jest zmergowany

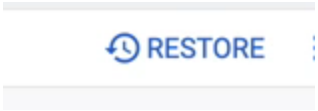


Następnie trzeba podać powód revert'a

Abandon :(

Wystarczy kliknąć w Gerrit GUI przycisk abandon

Zmiana abandon, może być przywrócona poprzez przycisk restore:



Trzeba mieć odpowiednie uprawinenia na to 😟

Grupowanie zmian

Grupowanie zmian poprzez tematy

Temat pozwala na grupowanie. Temat może być stosowany w kontekście:

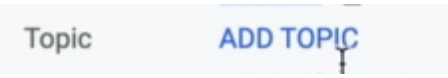
- user story
- feature
- component

- target version number
- targer build architecture

Tylko 1 temat może być przypisany do zmiany

Są 2 metody ustawienia tego:

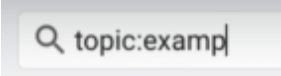
1. GUI - add topic



1. Podczas push na gerrit : dodanie flagi -o topic=topicName

np. → git push orgin HEAD:refs/for/<branch-name> -o topic=myTopic

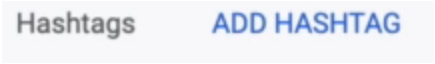
Możemy wyszukiwać następnie w gerrit, odowłując się do tematu (search bar):



Grupowanie poprzez hashtagi:

Róznica jest taka, że każda zmiana może mieć wiele hashtagów

Również możemy dodać to porzez gui:



oraz poprzez dodanie: -o t=hashTag1 -o t=hashTwo

np. → git push orgin HEAD:refs/for/<branch-name> -o t=hashTag1 -o t=hashTwo

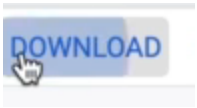
Conflict resolution → Rebase

Gerrit analizuje, czy podana zmiana będzie powodowała konflikt na granchu, i wtredy będzie o tym infomacja:

i nie będzie się dało wykonać submit

Postępowanie naprawcze:

1. Wykonujemy git pull
2. Dowload i wykonujemy komendę checkout :



Checkout

3. będziemy w stanie detached HEAD, i wykonujemt rebase na docelowy branch np. dev

```
git rebase orgin/dev
```

Wyświetli się komunikat o konflikcie, który należy rozwiązać

Poniważ ChangeId się nie zmieniło, gerrit rozpozna to.

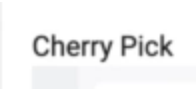
Teraz wypychamy zmiany, jako nowy patchset.

Cherry-pick

Nasz kod może być zależny od innych (nawet jeszcze nie zatwierdzonych) zmian.

Odnośnie zmiany, która jeszcze nie została jeszcze zatwiedzona:

Wtedy wchodzimy w download → Cherry Pick



Wchodzimy w terminal:

1. Musimy być na docelowym branchu
2. Robimy na nim git pull
3. wklejamy cherry pick
4. Rozwiązujem konflikt, poprzez zostawienie tego, co my chcemy .