GIT (2)

Aliasy

Używając aliasów możemy uprościć komendy gita lub zdefiniować własne, zawierające opcje, których najczęściej używamy.

git config --global alias.co checkout

git config --global alias.tree ‘log --oneline --decorate --graph’

git config --global alias.st status

git config --global

Git stash - schowek

Stash to rodzaj schowka, do którego możemy wrzucić zmiany - np. kiedy jesteśmy w środku implementacji jakiejś funkcjonalności, nie chcemy ich jeszcze commitować, a chcemy zająć się doraźnie czymś innym.

```
git stash

git stash show

git stash list

git stash apply

git stash apply stash@{1}

git stash clear
```

.gitignore (1)

W repozytorium git powinniśmy przechowywać tylko pliki źródłowe naszego projektu. Nie powinny być przechowywane:

- pliki konfiguracyjne środowiska programowania
- pliki tymczasowe
- generowane raporty
- skompilowane wersje projektu

Do wykluczenia plików z kontroli wersji możemy użyć pliku .gitignore

.gitignore (2)

Zawartość pliku .gitignore:

```
tmp/
*.log
*.swp
**/logs
debug[0-9]
!important/log
```

- plik .gitignore jest wersjonowany jak każdy inny plik w repozytorium
- podkatalogi mogą posiadać własne pliki .gitignore, jednak najczęściej stosuje się jeden wspólny dla projektu
- szablony plików, które mają być wykluczone, ale są zdefiniowane tylko dla lokalnego komputera, są przechowywane w pliku `.git/info/exclude`

Cofanie zmian i usuwanie

`Bardzo ostrożnie z poniższymi komendami!`

`git reset --hard` - usuwa zmiany dodane do stage area oraz nieśledzone

`git clean -f -d` - usuwa nieśledzone zmiany

`git clean -f -x -d` - usuwa również ignorowane pliki

`git clean -fxd :/` - usuwa nieśledzone i ignorowane pliki z całego repozytorium (bez :/ dotyczy tylko bieżącego katalogu)

Git hooks

Git hooks to mechanizm definiujący zachowanie gita podczas wykonania różnego rodzaju akcji. Mogą służyć np. do wymuszania określonego szablonu commit message, uruchomieniu testów przed wykonaniem push.

Git hooks to pliki wykonywalne w katalogu `.git/hooks`

- `applypatch-msg`
- `pre-applypatch`
- `post-applypatch`
- `pre-commit`
- `prepare-commit-msg`
- `commit-msg`
- `post-commit`
- `pre-rebase`
- `post-checkout`
- `post-merge`
- `pre-receive`
- `update`
- `post-receive`
- `post-update`
- `pre-auto-gc`
- `post-rewrite`
- `pre-push`

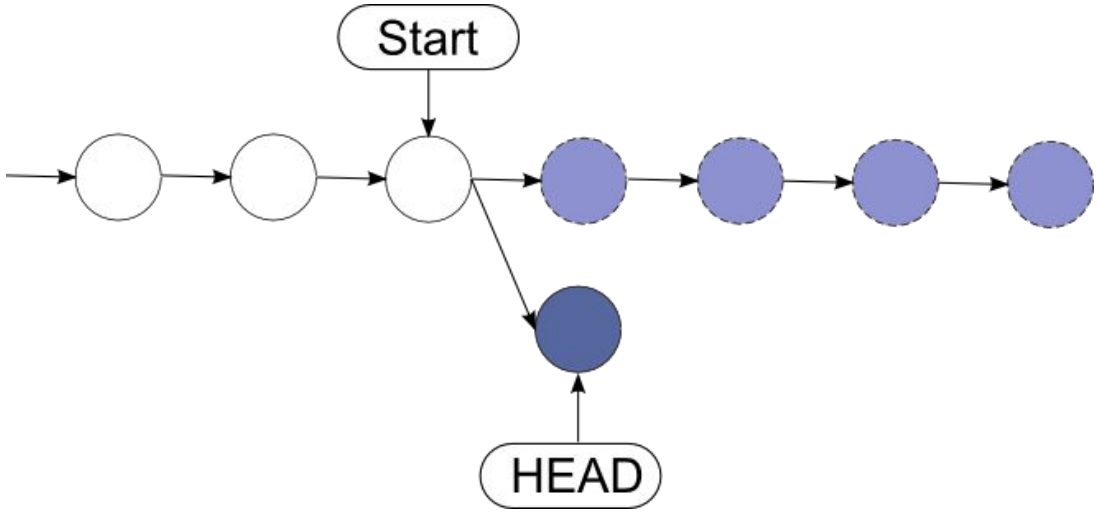
Squash commit

Korzystając z komendy

git rebase -i #nr_commitu

Możemy połączyć commity i “skrócić” historię zmian.

Może to być bardzo pomocne zwłaszcza, gdy praktyką łączenia branchy jest właśnie rebase.



Dobre praktyki

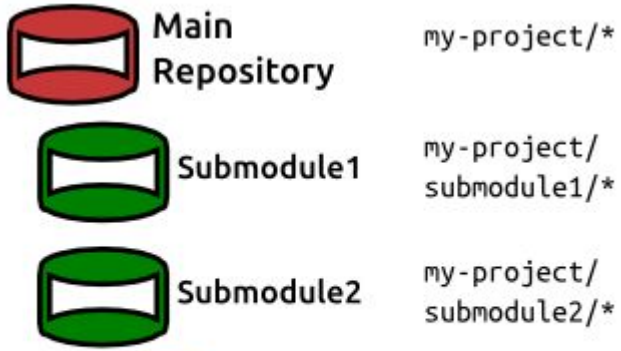
1. Twórz przejrzyste commity, z jedną odpowiedzialnością
2. Komentuj commity w sposób zrozumiały i wyjaśniający intencję
3. Commituj od samego początku i rób to często
4. Nie zmieniaj opublikowanej historii
5. Nie commituj plików, które są generowane
6. Przestrzegaj ustalonych standardów i workflow

Submoduły

Submoduły są sposobem na zarządzanie zależnościami. Zdarza się tak, że nasz projekt może działać tylko w połączeniu z innym projektem. Wtedy możemy dodać repozytorium z tym projektem jako podrzędne repozytorium i wykonywać na nim operacje git.

```
git submodule add -b <branch> --name <name> <repository>
git submodule add <repository-url> <sub-directory-name>
```

Nie jest to polecany schemat, niemniej można się z nim spotkać w niektórych projektach



GitOps

- GIT jest jedynym źródłem prawdy o systemie, miejscem przechowywania konfiguracji
- GIT jest jedynym miejscem, gdzie wykonujemy operacje (tworzenia, zmiany, usuwania) na wszystkich środowiskach,
- Wszystkie zmiany są udokumentowane.

Jednym z przykładów jest GitLab - CI/CD