System kontroli wersji Git

Laboratorium 1

Zadanie 1

1. Utworzyć konto na portalu gitlab.com - https://gitlab.com/users/sign_up.
2. Utworzyć publiczny projekt na GitLab o nazwie zgodnej ze wzorcem „**pba_lab1_numer albumu**” np. *pba_lab1_jk12890.*
3. Uruchomić terminal i dokonać konfiguracji użytkownika Git zgodnie z instrukcją pokazaną na portalu GitLab.

   Przykładowe polecenia w terminalu:

   ```
   git config --global user.name "Testowy User"
   git config --global user.email "pba.zut@gmail.com"
   ```

4. Utworzyć nowe repozytorium zgodnie z instrukcją wyświetlaną w stworzonym projekcie. W nowym, pustym repozytorium utworzyć plik README.md. W dowolnym edytorze wpisać krótki opis projektu. Następnie dodać plik README.md do plików, które podlegają zarządzaniu przez system wersjonowania, wykonać „commit” zmian oraz zsynchronizować zmiany z repozytorium zdalnym.

   Przykładowe polecenia w terminalu:

   ```
   git clone https://gitlab.com/xxx/pba_lab1_jk12890.git
   cd pba_lab1_jk12890
   git switch -c main
   touch README.md
   git add README.md
   git commit -m "add README"
   git push -u origin main
   ```

5. Poleceniem git log sprawdzić historię repozytorium.

Zadanie 2

6. W ustawieniach projektu na portalu GitLab w zakładce „***Settings/Repository***” w sekcji „**Protected branches/*Branch”*** ustawić opcje „***Allowed to push”*** na wartość „***No one”***.
7. Zmodyfikować zawartość pliku README.md. Podjąć próbę dodania zmiany do repozytorium zdalnego przy wykorzystaniu poleceń „***git add”***, „***git commit”***, „***git push”***. W sprawozdaniu załączyć rezultat polecenia push. Poprawne wykonanie instrukcji z punktu 6. i 7. powinno skutkować odmową wykonania polecenia push.

```
! [remote rejected] main -> main (pre-receive hook declined)
error: failed to push some refs to 'https://gitlab.com/xxx/pba_ lab1_ jk12890.git'
```

8. Kontynuowanie pracy w ramach przygotowanego laboratorium wymagać będzie przywrócenia repozytorium do stanu sprzed punktu 7. W rezultacie wynik polecenia „***git log”*** powinien być identyczny jak w punkcie 5.

```
git reset --hard HEAD^1
```

9. Utworzyć branch o nazwie „***dev***”. Branch może być utworzona przez interfejs graficzny portalu GitLab lub z poziomu wiersza poleceń. W pierwszym przypadku w celu synchronizacji informacji z repozytorium zdalnym należy skorzystać z polecenia **„*git fetch***”. W drugim przypadku należy pamiętać o konieczności synchronizacji repozytorium lokalnego z repozytorium zdalnym poprzez wywołanie polecenia „***git push”***. W sprawozdaniu załączyć rezultat wywołania polecenia „***git branch”***, na którym widać branch ***dev*** oraz ***main***.
10. Znajdując się na branchy ***dev*** utworzyć plik SECURITY.md

> *You should include a SECURITY.md file that highlights security related information for your project. This should contain:*
>
> ***Disclosure policy.***
> *Define the procedure for what a reporter who finds a security issue needs to do in order to fully disclose the problem safely, including who to contact and how. Consider HackerOne's community edition or simply a 'security@' email.*
>
> ***Security Update policy****.*
> *Define how you intend to update users about new security vulnerabilities as they are found.*
>
> ***Security related configuration****.*
> *Settings users should consider that would impact the security posture of deploying this project, such as HTTPS, authorization and many others.*
>
> ***Known security gaps & future enhancements.***
> *Security improvements you haven't gotten to yet. Inform users those security controls aren't in place, and perhaps suggest they contribute an implementation! For some great reference examples of SECURITY.md files, look at Apache Storm and TensorFlow.*
>
> *[źródło: www.snyk.io]*

11. Utworzony plik dodać do plików obserwowanych przez system git, wykonać commit zmian oraz zsynchronizować repozytorium lokalne i zdalne. Utworzona branch nie została oznaczona jako „protected”, dlatego polecenie **„git push”** tym razem powinno zakończyć się sukcesem.
12. Za pomocą interfejsu portalu GitLab utworzyć ***merge request*** za pomocą którego zmiany na branchy ***dev*** zostaną dograne do chronionej branchy ***main*** **(„Merge Requests/Create merge request”, „Submit merge request”). UWAGA!** Przed naciśnięciem na przycisk **„Submit Merge Request”** odznaczyć checkbox „*Delete source branch when merge request is accepted*.” Po przejrzeniu zmian, których dotyczy ***MR*** zaakceptować zmiany poprzez wybranie przycisku **„Merge”**

Zadanie 3

13. Celem zadania jest konfiguracja uwierzytelniania użytkownika przy wykorzystaniu kluczy SSH. Rozwiązanie to pozwoli na dostęp do repozytorium bez konieczności logowania się i zwiększy bezpieczeństwo projektu. Z poziomu terminala Linux wygenerować parę kluczy SSH np. RSA. Po wpisaniu odpowiedniego polecenia klucze pojawią się w katalogu domowym w ukrytym folderze ***.ssh***.

```
ssh-keygen -t rsa -b 2048 -C "pba.zut@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/mba/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/mba/.ssh/id_rsa
Your public key has been saved in /home/mba/.ssh/id_rsa.pub
```

14. Klucz publiczny należy skopiować do schowka. W interfejsie graficzny na portalu GitLab odnaleźć ustawienia użytkownika. Następnie w zakładce „SSH Keys" odszukać miejsce przygotowane do wklejenia klucza ssh. **UWAGA!** Należy skopiować cały rekord zaczynający się od ***ssh-rsa*** i kończący na nazwie wskazanej przy generowaniu klucza, przekazanej po parametrze ***-C.*** Dodatkowa dla każdego z dodawanych kluczy ustawić należy datę ważności klucza publicznego. Po uzupełnieniu wymaganych informacji zatwierdzamy klucz przyciskiem **„Add key".**
15. W celu przetestowania konfiguracji należy zmodyfikować dowolny plik oraz wykonać synchronizacje utworzonych zmian. Komenda ***push*** powinna zakończyć się bez konieczności wprowadzania nazwy użytkownika GitLab oraz hasła co potwierdza, że logowanie przy wykorzystaniu kluczy ssh zostały skonfigurowane poprawnie.

Zadanie 4

16. W zależności od dystrybucji Linux wywołać odpowiednią komendę do instalacji pakietu ***git-secrets***.

```
sudo apt-get install git-secrets
```

17. W folderze projektu utworzyć nowy plik o nazwie run.sh. Zawartość pliku przygotować w taki sposób, by do zmiennej „password" przypisać dowolna wartość hasła a następnie wyświetlić ustawione hasło.

18. W folderze projektu wywołać polecenie, które instaluje narzędzie w repozytorium.

```
cd /path/to/repo
git secrets --install
```

19. Skonfigurować wzorce, które będą wyszukiwane podczas synchronizacji repozytorium. Następnie przetestować wynik skanowania plików źródłowych.

```
git secrets --add 'password\s*=\s*.+'
git secrets --scan
```

20. Z poziomu terminala Linux ustawić zmienną środowiskową $PASS, która przypisywana będzie do zmiennej password w skrypcie ***run.sh***.
21. Do konfiguracji dodać wyrażenie regularne, które zezwalać będzie na przekazywanie w kodzie haseł uzupełnianych na podstawie zmiennych środowiskowych.

```
git secrets --add --allowed '\$.*'
```

22. Przeprowadzić próbę pokazującą, że hasło przypisane jako tekst powoduje zwrócenie błędu podczas skanowania. Analogicznie pokazać, że hasło wczytywane z poziomu zmiennej środowiskowej nie powoduje błędu podczas skanowania i przy dodawaniu zmian do repozytorium. Dokonać synchronizacji zmian w skrypcie ***run.sh*.**