Bezpieczeństwo w inżynierii oproramowania

dr Agnieszka Zbrzezny

1 Systemy kontroli wersji

Na potrzeby poniższego opisu zakładamy, że polecenia wykonuje studentka o nazwisku Ewa Kot z grupy laboratoryjnej nr. 1.

1.1 Rozproszony system kontroli wersji `git`

- Instalacja programu `git` w systemie Ubuntu i jego pochodnych:

```
$ sudo apt-get install git
```

- Podstawowa konfiguracja programu `git` po jego instalacji:

```
$ git config --global user.name "Ewa Kot"
$ git config --global user.email "ewa.kot@gmail.com"
$ git config --global core.editor "vim"
```

1.2 Serwer `bitbucket.org`

- Zakładamy konto o nazwie `ewakot` powiązane z adresem `ewakot@gmail.com`
- Tworzymy repozytorium o nazwie `bwip-gr1-KotEwa`
- Wykonujemy po kolei polecenia, wyświetlone na stronie (omówione także w następnym podrozdziale)

1.3 Tworzenie lokalnego repozytorium na własnym komputerze

- W wybranym katalogu tworzymy podkatalog o nazwie `bwip-gr1-KotEwa`:

```
$ mkdir -p studia/bwip-gr1-KotEwa
```

- Przechodzimy do utworzonego podkatalogu:

```
$ cd studia/bwip-gr1-KotEwa
```

- Tworzymy puste repozytorium

```
$ git init
```

- Wiążemy lokalne repozytorium z tym utworzonym na serwerze `bitbucket.org`:

```
$ git remote add origin https://ewakot@bitbucket.org/ewakot/bwip-gr1-kotewa.
```

- Tworzymy plik `README`

  ```
  $ echo "Ewa Kot - Bezpieczeństwo w inżynierii oprogramowania" >> README
  ```

- Spawdzamy status repozytorium

  ```
  $ git status
  ```

- Rozpoczynamy śledzenie pliku `README`

  ```
  $ git add README
  ```

- Ponownie sprawdzamy status repozytorium

  ```
  $ git status
  ```

- Zatwierdzamy zmiany w repozytorium

  ```
  $ git commit -m "Initial commit with README"
  ```

- Wysyłamy pierwszy raz zmiany na serwer:

  ```
  $ git push -u origin master
  ```

- Wysyłanie kolejnych zmian na serwer:

  ```
  $ git push
  ```

1.4 Udostępnienie repozytorium do czytania dla prowadzącego zajęcia

- W repozytorium klikamy `Settings` (po lewej stronie ekranu na dole), a następnie klikamy `User and group access`.
- Wyszukujemy użytkownika `Agnieszka Zbrzezny` (nazwa użytkownika: `agnieszka_m_zbrzezny`).
- Klikamy przycisk `Add`.

1.5 Klonowanie repozytorium na innym komputerze (np. w pracowni komputerowej

- Klonowanie wg adresu pobranego ze strony repozytorium

  ```
  $ git clone https://ewakot@bitbucket.org/kotewa/bwip-gr1-kotewa.git
  ```

- Lokalna konfiguracja programu `git`:

  ```
  $ cd sp-kot
  $ git config user.name "Ewa Kot"
  $ git config user.email "ewa.kot@gmail.com"
  $ git config core.editor "vim"
  ```

1.6 Pobieranie na własnym komputerze zmian wprowadzonych do repozytorium na innym komputerze

- Przechodzimy do właściwego podkatalogu:

  ```
  $ cd studia/bwip-gr1-kotewa
  ```

- Pobieramy zmiany z serwera

  ```
  $ git pull
  ```

1.7 Książki na temat git-a

- `https://git-scm.com/book/pl/v1`

2 Środowiska virtualne

Wirtualne środowiska Python umożliwiają skonfigurowanie Python sandbox za pomocą własnego zestawu pakietów oddzielnego od systemowych `site-packages`, w których będą działać. Istnieje wiele powodów, aby używać środowisk wirtualnych, na przykład jeśli masz wiele usług działających z tą samą instalacją Pythona, ale z różnymi pakietami i różnymi wymaganiami dotyczącymi wersji pakietu. Ponadto może się okazać przydatnym, aby zachować zależne wymagania pakietu dla każdego projektu Python, nad którym pracujesz. Wirtualne środowiska pozwalają to zrobić. Wersja `virtualenv` PyPI działa w większości środowisk. Od wersji Python 3.3 moduł środowiska wirtualnego `venv` jest częścią standardowej biblioteki. Jednak niektóre problemy z `venv` zostały zgłoszone w Ubuntu. Ponieważ `virtualenv` działa z Pythonem 3.6 i Ubuntu, będziemy używać `virtualenv`. Oto jak skonfigurować środowisko wirtualne w macOS i Linux:

```
# instalacja pip
$ sudo apt-get install python3-pip

# instalacja virtualenv u ywaj c pip3
$ pip3 install -U virtualenv

# instalacja virtualenv u ywaj c apt
$ sudo apt install virtualenv

# tworzenie wirtualnego  rodowiska
$ virtualenv -p /usr/bin/python3.6 /path/to/env_name/

#aktywowanie wirtualnego  rodowiska
$ source /path/to/env_name/bin/activate

#deaktywowanie

(env_name) $
... do your work ...
(env_name) $ deactivate
```

3 Zadania

1. Przećwicz aktywację i dezaktywację swojego wirtualnego środowiska kilka razy.
   - `$ source venv/bin/activate`
   - `$ deactivate`
2. Zainstaluj pytest na swoim wirtualnym środowisku za pomocą pip3.
3. Utwórz kilka plików testowych. Możesz również użyć przykładów z wykładu. Przetestuj je.
4. Zmień asserty. Nie używaj tylko porównań `==`, ale np. użyj `in`, `not in`, `>`.
5. Pobierz projekt do rozdziału 2, tasks_proj, ze strony internetowej książki `https://pragprog.com/titles/bopytest/source\_code` i upewnij się, że możesz zainstalować go lokalnie instalacją `pip /path/to/tasks_proj`.
6. Przejrzyj drzewo katalogów.
7. Uruchom program pytest z jednym plikiem.

8. Uruchom pakiet pytest w jednym katalogu, na przykład `tasks_proj/tests/func`. Użyj pytest aby uruchamiać testy indywidualnie, a także katalog testów na raz. Niektóre testy nie działają. Czy rozumiesz, dlaczego zawodzą?

9. Dodaj znaczniki `xfail` lub `skip` dla testów zakończonych niepowodzeniem, tak aby w końcu uruchomić pytest w katakogu testów bez niepowodzeń.

10. Nie mamy jeszcze żadnych testów dla `tasks.count()`, oprócz innych funkcji. Wybierz nieprzetestowaną funkcja API i pomyśl o tym, jakie przypadki testowe musimy mieć aby upewnić się, że działa poprawnie.

11. Co stanie się, jeśli spróbujesz dodać zadanie z już ustawionym identyfikatorem? Brakuje niektórych testóœ wyjątków w `test_api_exceptions.py`. Sprawdź, czy możesz wypełnić brakujące wyjątki. (W tym ćwiczeniu można zajrzeć do `api.py`).