Hogyan tovább?

Verziókezelés: Git

Mi az a verziókezelés?

- Verziókezelés és verziókezelő fogalma
- Elosztott és központosított verziókezelő rendszer
- Néhány konkrét verziókezelő technológia legalább megemlítés szinten: Git, SVN, Mercury
- Verziókezelő hosztolás, például GitHub, BitBucket, GitLab

Hogy működik a Git?

- Milyen verziókezelő rendszer?
- Telepítése, GitBash használata, `git` parancs ellenőrzése
- Alapfogalmak:
    - *Repository*
    - *Working directory*
    - *Staging index*
    - *Commit*
    - *Remote repository*
    - *Branch*
    - *Merge*
    - *Conflict*
    - *Blame*

    > Ezek a fogalmak nagyjából ugyanazok minden verziókezelő esetében, így egyszer megtanulni mindenképp érdemes őket. Bár a legtöbb helyen Git-et használnak, még elő-elő fordulhat SVN vagy Mercury.

- Parancssori Git használat:
    - `git init`
    - `git status`
    - `git clone`
    - `git config`
    - `git add`
    - `git commit`
    - `git push`
    - `git pull`

    - `git checkout`
    - `git branch`

    > Ennél sokkal többet is fel lehetne itt sorolni, de felesleges kezdésnek minden műveletet megtanulni, hiszen ezek egy részét inkább érdemes valamilyen grafikus programmal elvégezni.

- *.gitignore* fájl

Git Hosting és kliensek

- Eclipse vagy IntelliJ Git plugin

  > Érdemes kipróbálni őket, de mind a kettőben kicsit gyatra a Git plugin, érdemes lehet egy másik, Java IDE-től teljesen különálló szoftvert is kipróbálni.

- Grafikus Git kliens: SmartGit, GitKraken, SourceTree, TortoiseGit, GitHub Desktop...

  > A SmartGit-tel kapcsolatban nagyon pozitív tapasztalataim vannak, de próbáltam a GitKraken-t és GitHub Desktop-ot is. Az utóbbi talán az egyik legelhanyagolhatóbb.

- Git repository hoszt szolgáltatások: GitHub, GitLab, GitBucket

  > A GitHub talán a legismertebb és neki van talán a legtöbb szolgáltatása. Érdemes létrehozni egy felhasználót, majd egy *repository*-t és megnézni, hogy miket tud a GitHub azon kívül, hogy Git repo-t hosztol.

Hasznos anyagok a Git tanulásához

- Traversy Media - Git bevezető: **https://youtu.be/SWYqp7iY_Tc**
    - Egyszer érdemes végignézni, a fogalmakat érdemes angolul tanulgatni belőle
    - A youtube csatorna egyébként zseniálisan jó, bár nem kifejezetten *Java* a fő technológia, de akadnak nyelvtől, programozási paradigmától független jó témakörök, melyeket feldolgoznak a csatornán. Érdemes feliratkozni.
- SanFranciscobol Jottem - Git kurzus: **https://youtu.be/XDKZu9kuEn8**
    - Sajnos az a program, amivel bemutatják a Git működését már nem érhető el
- Git cheatsheet: **https://github.com/firith/git-kezdo**

- Git Markdown cheatsheet: https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet
- GitBucket (Atlassian termék, csakúgy mint a Jira, Confluence) tananyaga: https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud
- SourceTree tananyag (grafikus felületes): http://teleyah.com/cspro/EthiopiaSep2016/IntroGitSourceTree.pdf

Maven

Build automatizáló eszközök koncepciója

- Projekt csomagolása, exportálása, integrálása (jar, war, ejb stb...)
- Függőségek
- Automatizált tesztelés
- Néhány eszköz megemlítés szinten: *Apache Ant*, *Apache Maven*, *Gradle*

Maven alapfogalmak

- Maven telepítésekor előjövő fogalmak
  - Környezeti változók
  - A .m2 mappa
  - `mvn` parancs megismerése
- POM, azaz Project Object Model
- Életciklusok és fázisok végrehatjása: `mvn clean`, `mvn clean install`, `mvn compile` stb...

> 3 életciklus, életciklusonként viszont elég sok fázis van, de ezek mindegyikét teljesen felesleges ismerni. Inkább úgy fogalmazok, hogy ha a következő szavakat az `mvn` parancs után írjuk, akkor tudjuk azt, hogy mit várhatunk a végrehajtásakor: `clean`, `compile`, `package`, `install`, `test`

Maven projekt konfigurálása

- Java verzió beállítása
- Projekt koordináták és konvenciók
- Függőségek beállítása, függőségek *scope*-ja (elég csak a `provided` , `runtime` és `test` *scope*-okat ismerni)
- Maven Central Repository: függőségeket böngészhetünk vele könnyedén (https://mvnrepository.com/repos/central)

- Maven mappastruktúra: *main*, *test*, *java*, *resources*, *target* (bővebben nem nagyon kell)
- Modularitás: szülő projekt, multimodul projekt létrehozása

> A Maven-t a legkönnyebb úgy megtanulni véleményem szerint, hogy valahányszor létrehozunk egy projektet, az legyen mindig Maven projekt. Ezt lehet akár manuálisan, vagy az IDE-ben összekattintgatni. Eclipse és IntelliJ is támogatja szerencsére.

Gyakori függőségek

- Apache Commons (https://commons.apache.org), Google Guava (https://github.com/google/guava): Általános célú projektek, mely közös megoldásokat adnak a Java-ban való programozáshoz. Habár a fejlesztés minden aspektusára van valami megoldásuk, célszerű csak a legfontosabbakat ismerni belőlük.
- JUnit 4, JUnit 5: teszt keretrendszerek
- Slf4j: Naplózó szolgáltatás
- Továbbiak itt kereshetők: https://mvnrepository.com/popular

Hasznos anyagok a Maven tanulásához

- Elsősorban a Maven honlapja javasolt: https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

  > Sajnos nem túl barátságos, de minden lényeges információ megtalálható itt. Érdemes legalább egyszer átnézni, főleg a telepítési útmutatót.

- Másodsorban javasolt természetesen a Baeldung blogja (https://www.baeldung.com/maven és https://www.baeldung.com/tag/maven-basics/)

  > A leírás kicsit jobban belemegy a plugin fogalomba, de kezdésnek nem nagyon kell róluk sokat tudni, de egyszer elolvasni nem árt. A mutlimoduláris projekteket szintén nem kell nagyon erőltetni kezdésnek. Az `archetype` plugint viszont érdemes kipróbálni legalább egyszer, az IDE-k is ezzel dolgoznak a háttérben.

- Baeldung guide, de ez most egy összefoglaló link: https://www.baeldung.com/maven-guide

  > Érdemes témakörönként szemezgetni és kipróbálni.

Maven kapcsán érdemes gyakorlati szemszögből megközelíteni a tanulást, így a tanulás során készített projekteket érdemes Maven projektként létrehozni és nézegetni.

Webes alkalmazás fejlesztése

A webfejlesztés sok technológiát és paradigmát gyúr össze egy fogalomba, így igazából lehetetlen megállapítani, hogy mire lehet pontosan szüksége egy webfejlesztőnek.

De általában elmondható, hogy kell

- kliens oldali programozói ismeret, ami az esetek többségében **JavaScript**-es technológia
- szerver oldali programozói ismeret, ami igazából majdnem, hogy bármi lehet, de most maradjunk valamilyen **Java** technológiánál
- adatbázis ismeret: SQL + PL/SQL

  > Az SQL-lel kapcsolatban a `view` és `index` fogalmakkal érdemes még kibővíteni az akadémiás tudást, a PL/SQL kapcsán kezdésnek csak említés szinten érdemes ismerni a képességeit és hogy mire való.

HTML, CSS úgy általában mindig fog kelleni, kivéve ha nem kifejezetten böngészős alkalmazást csinálunk (ami manapság nem túl korszerű).

Hasznos anyagok

- http://webprogramozas.inf.elte.hu/tananyag/kliens/
- https://github.com/horvathgyozo/alkfejl-2017 - Spring-es, Angular-os, Typescript-es kurzus feladatokkal és megoldásokkal

  > Ez egy teljes ELTÉ-s kurzus anyaga, példákkal, feladatokkal óráról, órára minden rögzítve. Nagyon jó anyaggal rendelkezik, teljesen dokumentált! Kb minden van benne, ami ahhoz kell, hogy egy alap webalkalmazást összehozzunk. Figyelni kell azonban az előbukkanó fogalmakra, például JSON, REST Api, HTTP Method, Dependency Injection stb...

- https://spring.io/guides/gs/spring-boot/#scratch - Spring Boot bevező

> A Spring Boot egyike a nagy Java webfejlesztéses keretrendszereknek, így bevezető anyagból és videókból annyi van belőle, mint égen a csillag és egyik jobb, mint a másik. Így nem is igazán merek ennél többet belinkelni. Nekem a fenti 3 tananyag nagyon sokat segített. Persze azzal lehet a legtöbbet tanulni, ha ezt kipróbálja az ember.

Elmélet

Ide csak bedobok néhány témakört, amit érdemes lehet nézegetni, esetleg feltüntetni az önéletrajzban, LinkedIn profilon.

- UML

  > Az UML-ből csak az osztálydiagrammot néztük, de ennél sokkal többet is specifikál a szabvány. Nem érdemes mindegyiket ától cettig tudni, de a következő fogalmak hasznosak lehetnek:
  >
  > - osztálydiagram kapcsán az asszociációk típusai: kompozíció, aggregáció, asszociáció, számosság, navigálhatóság
  > - állapotgép
  > - use-case diagram
  > - szekvencia diagram

- Tervezési minták: **https://refactoring.guru/design-patterns**, **https://github.com/amilajack/reading/blob/master/Design/GOF%20Design%20Patterns.pdf** (régi könyv, főleg C++ nyelven vannak kódrészletek)

  > Van belőlük vagy 30, nem kell mindet, így néhány fontos darab: Factory Method, Strategy, Observer, Builder, Adapter, Singleton. Persze minél több, annál jobb. Fontos azonban tudni, hogy mikor érdemes használni adott tervezési mintát és azt, hogy ezeknek a mintáknak az alkalmazása mivel jár. Ezekre szeretnek rákérdezni interjún, de abszolút ki lehet vele tűnni a többi jelentkező közül.

- Clean Code és agilis szoftverfejlesztés bevezető jelleggel

  > Nagyon nagy a témakör, de szórakoztató módon tanulható, mivel Bob bácsi (Robert C. Martin, az agilis szoftverfejlesztés atyja) előadásai youtube-on elérhetők. És nagyon jó előadások! **https://youtu.be/7EmboKQH8lM**