

Web Programlama I

Ders 02 - Bash, Regex, Build Araçları ve Test

19.10.2021

Erciyes Üniversitesi
Bilgisayar Mühendisliği Bölümü

Eğitmen: Ömür ŞAHİN

1 Genel Bakış

2 Bash

3 Düzenli İfadeler (Regular Expression)

4 Build Araçları

5 Test

1-Genel Bakış

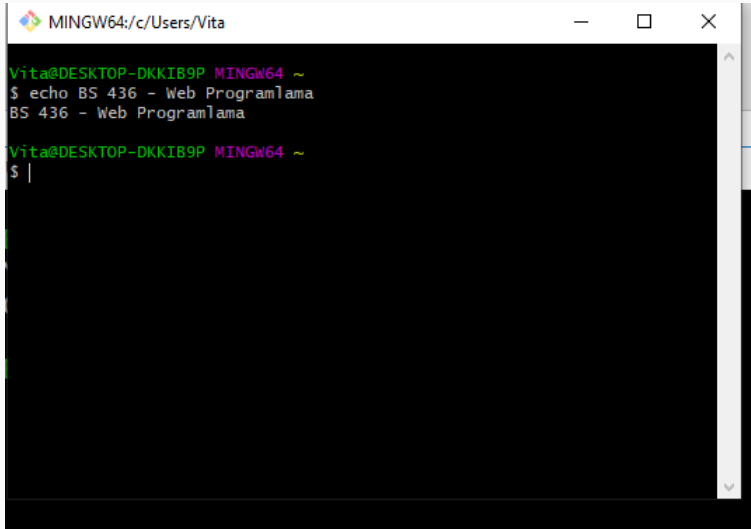
- Bash terminaline giriş
- Düzenli İfadelere (regular expression) giriş
- Build araçları: YARN, WebPack and Babel
- Test senaryosu nasıl yazılır?

2-Bash

- Bash, Linux/Mac/Unix sistemlerde kabuktur (shell) ve komut dilidir.
- Pek çok farklı shell bulunmaktadır.
 - PowerShell Windows'ta kullanılmaktadır.
- Shell ayrıca terminal, konsol (console), komut satırı (command-line) gibi isimlendirmelere sahiptir.
- Komut yazıp çalıştırmaya olanak tanır.

- Bash, Linux/Mac/Unix sistemlerde kabuktur (shell) ve komut dilidir.
- Pek çok farklı shell bulunmaktadır.
 - PowerShell Windows'ta kullanılmaktadır.
- Shell ayrıca terminal, konsol (console), komut satırı (command-line) gibi isimlendirmelere sahiptir.
- Komut yazıp çalıştırmaya olanak tanır.

Bash



```
MINGW64:/c/Users/Vita

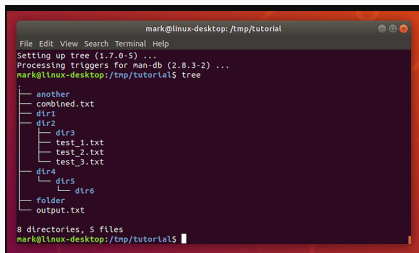
Vita@DESKTOP-DKKIB9P MINGW64 ~
$ echo BS 436 - Web Programlama
BS 436 - Web Programlama

Vita@DESKTOP-DKKIB9P MINGW64 ~
$ |
```


Neden?

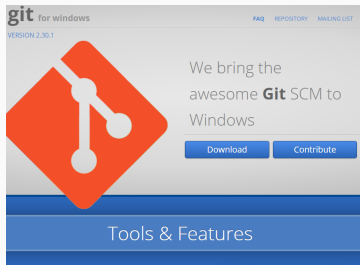
- Program yazıyorsanız oldukça kritik öneme sahiptir.
- Pek çok görevi otomatize etmeye fayda sağlar.
- Pek çok sunucu bir GUI sağlamaz.
 - Terminal kullanarak SSH üzerinden erişim sağlanır.
 - Gömülü ve IoT cihazlarına erişim için kullanılır.
- Temel komutları bilmeniz gerekir.
- Build araçlarını yürütmek için Bash komutlarına ihtiyaç duyulur.

- Linux/Mac kullanıyorsanız, kurulu olarak gelmektedir.
- Eğer Windows kullanıyorsanız GitBash kullanılması önerilmektedir.
 - <http://gitforwindows.org>



```
mark@linux-desktop: /tmp/tutorial
File Edit View Search Terminal Help
Setting up tree (1.7.0-5) ...
Processing triggers for man-db (2.8.3-2) ...
mark@linux-desktop: /tmp/tutorial$ tree
.
├── another
├── combined.txt
├── dir1
├── dir2
│   ├── dir3
│   │   ├── test_1.txt
│   │   ├── test_2.txt
│   │   └── test_3.txt
├── dir4
│   └── dir5
│       └── dir6
├── folder
└── output.txt

8 directories, 5 files
mark@linux-desktop: /tmp/tutorial$
```



- **..**: Mevcut klasör
- **...**: Üst klasör
- **~**: Home klasörü
- **pwd**: Mevcut klasörü yazdır (**P**ring **W**orking **D**irectory)
- **cd**: Klasör değiştir (**C**hange **D**irectory)
- **mkdir**: Klasör oluştur
- **ls**: Klasör içeriğini listele
- **cp**: Dosya kopyala
- **mv**: Dosya taşı
- **rm**: Sil ("-r" rekürsif olarak sil)
- **man**: Komutun kullanım kılavuzu

- **echo**: Girdi değerini yazdır
- **cat**: Dosya içeriğini yazdır
- **less**: Dosya içeriğini kaydırabilir (scroll) olarak yazdır.
- **>**: Yönlendirme
- **>>**: Ekleme
- **|**: Pipe komutu
- **which**: Programın konumu
- **\$**: Değişken yazdırırken kullanılır (örnek \$VAR)
- **wc**: Kelime saydır
- **find**: Arama
- **grep**: Regular Expressiona göre ayıkla
- **touch**: Erişim zamanını güncelle eğer dosya yoksa oluştur.

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler

\$ pwd

/d/Ders/Ornekler

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler

\$ ls

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler

\$ echo "Keyifler nasıl gençler" > test.txt

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler

\$ ls

test.txt

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler

\$ cat test.txt

Keyifler nasıl gençler

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler

\$ |



```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler
$ mkdir testKlasoru
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler
$ ls
test.txt  testKlasoru/
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler
$ cd testKlasoru/
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ ls
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ pwd
/d/Ders/Ornekler/testKlasoru
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ ls ..
test.txt  testKlasoru/
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ ls
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ |
```

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ cp ../test
test.txt      testKlasoru/

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ cp ../test.txt ./klasordeDosya.txt

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ cat klasordeDosya.txt
Keyifler nasıl gençler

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ mv ../test.txt .

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ ls
klasordeDosya.txt  test.txt

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru
$ |
```

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru

\$ echo \$PATH

/c/Users/Omur/bin:/mingw64/bin:/usr/local/bin:/usr/bin:/bin:/mingw64/bin:/usr/bin:/c/Users/Omur/bin:/e/Python/Python37/Scripts:/e/Python/Python37:/e/Program Files/doptOpenJDK/jdk-11.0.10.9-hotspot/bin:/c/WINDOWS/system32:/c/WINDOWS:/c/WINDOWS/system32/wbem:/c/WINDOWS/System32/WindowsPowerShell/v1.0:/cmd:/c/WINDOWS/System32/enSSH:/c/Users/Omur/AppData/Local/Microsoft/WindowsApps:/c/Users/Omur/AppData/Local/gitkraken/bin:/e/Program Files/Microsoft VS Code/bin:/e/texlive/2020/bin/win32/USERPROFILE%/AppData/Local/Microsoft/WindowsApps:/usr/bin/vendor_perl:/usr/bin/perl

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru

\$ which bash

/usr/bin/bash

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Ders/Ornekler/testKlasoru

\$ |

- Projedeki JavaScript dosya sayısını nasıl elde edersiniz?
- Bütün bu dosyalardaki satır sayısını nasıl elde edersiniz?

```
Omur@DESKTOP-LEBEUU3 MINGW64 /d/Calismalar/Github/futwebapp-tampermonkey (master)
$ find . -regex '^\.*\.jsx?' -not -path */node_modules/* | wc -l
38

Omur@DESKTOP-LEBEUU3 MINGW64 /d/Calismalar/Github/futwebapp-tampermonkey (master)
$ cat 'find . -regex '^\.*\.jsx?' -not -path */node_modules/*' | wc -l
5317
```

- "find" komutu rekürsif olarak bütün "." klasörünü arar.
 - Mac sistemlerde "-E" parametresini kullanmalısınız (find -E).
- JS/JSX dosyaları ile eşleşen regular expression:
 - `^` Dosya adının başlangıcı
 - `.*` Herhangi bir karakter (.) herhangi bir sayı (*)
 - `"\."` `"\"` karakteri (`"\"` escape karakteri olarak kullanılıyor)
 - `"\.jsx?"` x opsiyonel olarak şekilde dosya adının sonu.
 - `"-not -path */node_modules/*"` klasörü hariç
- `"| wc -l"`: dosya isimlerini satır saydırma programına verir.
- `cat 'x':` " içindeki komutu çalıştırır ve çıktısını terminale verir.
 - Böylelikle JS/JSX dosyaları içindeki her şeyi cat ile yazdırmış oluruz.

- Yukarı/Aşağı butonları ile geçmiş komutlara ulaşabilirsiniz (history ile de erişilebilir ve !komut numarası ile çağırılabilir.)
- "tab" tuşu ile yazılacak kelime tamamlanabilir.
- Bash komutları çalıştırılabilir script haline çevirilebilir.
 - ".sh" uzantısı kullanılabilir.
 - İlk satırın "#!<pathToBash>" olarak başlaması gerekmektedir. (Örnek, "#!/usr/bin/bash")
 - Diğer programlar gibi terminalde çalıştırılabilir olacaktır.

3-Düzenli İfadeler (Regular Expression)

- "erciyes.edu.tr" geçerli bir mail adresi midir?
 - @ sembolü içermemektedir. O yüzden hayır.
- "03ASD451ASD1245456ASDSAD45" geçerli bir telefon numarası mıdır?
 - Sayı olmayan ifadeler bulunmaktadır ve çok uzundur. O yüzden hayır.
- String ifade özel bir formatta kısıtlara sahip ise Regex kullanılabilir.
 - Geçerli bir mail belirten string ifadeler.
 - Genellikle HTML formlarının girdileri kontrol edilirken kullanılır.

Eşleşme Kuralları

- `"."` karakteri wildcard olarak kullanılır.
 - `"a.b"` ifadesi a ile başlayan ve b ile biten 3 harfli ifadeleri temsil eder.
- `"[]"` ifadesi set içerisindeki tek bir karakter ile eşleşir.
 - `"[abc]"` ifadesi "a", "b" ve "c" harfleri ile eşleşirken, "d", "ab" gibi karakterlerle eşleşmemektedir.
- `"[-]"` ifadesi aralığı belirtir.
 - `"[a - z]"` ifadesi bütün küçük harflerle eşleşirken, `[a - zA - Z]` ifadesi bütün harflerle eşleşir. `[0 - 8]` ise 9 hariç bütün sayılarla eşleşir.
- Özel ifadeler kullanmak için `\` kullanılmalıdır.
 - `"\\[\\.\\]"` ifadesi `"[.]"` ile eşleşir. a veya `[a]` ifadesi ile eşleşmemektedir.

- **()** regex'in sınırlarını belirlemede kullanılır.
- **|** or operatörü olarak çalışır.
- ***** önceki regex tanımını 0 veya daha fazla tekrarlar.
- **"ab*", "a", "ab" ve "abbbbb"** ile eşleşir.
- **"(ab)*", "",ab ve "abababababab"** ile eşleşir.
- **"(ab)|c" "ab" ve "c" ile eşleşir.**

- "+" en az 1 sefer demektir.
 - "x+" \rightarrow "xx*"
- "?" 0 veya 1 sefer demektir.
 - "x?" \rightarrow "boş karakter | x"
- "{" kaç kere tekrarlanacağını belirtir.
 - "x{5}" \rightarrow "xxxxx"
 - "x{2,4}" \rightarrow "(xx)|(xxx)|(xxxx)"

Örnek: Telefon Numarası

- 10 Haneli Numara:
 - Örnek: 3522076666
- + ile başlayan 2 haneli ülke kodu da opsiyonel olarak bulunmaktadır.
 - Türkiye kodu: +90

$((\backslash+)[0-9]\{2\})?[0-9]\{10\}$

- $(\backslash+)[0-9]\{2\})?$ → Opsiyonel olarak + ile başlayan 2 haneli sayı
- $[0-9]\{10\}$ → 10 adet 0-9 aralığında sayı

Regex'in kısıtları

- Regex kısıtları olan string ifadelerin geçerliliğini kontrol etmek için oldukça faydalıdır.
- Bütün kısıtları tanımlamada yeterli değildir.
- Örnek: Bir string ifadenin geçerli bir JavaScript kodu olup olmadığı Regex ile anlaşılamaz. Bunun için Context-Free-Grammar kullanılır.

4-Build Araçları

- JS geliştirmek için açık kaynak kütüphaneler oldukça önemlidir.
 - En önemli kütüphanelerden biri derste de kullanacağımız React kütüphanesidir.
- JS'te YARN ve NPM olmak üzere iki temel araç bulunmaktadır.
- YARN ve NPM aynı bağımlılık depolarına erişim sağlamaktadır.
- YARN'da yeni özellikler daha erken geldiği için bir adım öndedir.
- Terminal komutları ile kullanılmaktadır.

```
MINGW64:/f/Ders/BashExamples/InitExample
Vita@DESKTOP-DKKIB9P MINGW64 /f/Ders/BashExamples/InitExample
$ yarn init -y
yarn init v1.19.1
warning The yes flag has been set. This will automatically answer yes to all questions, which may have security implications.
success Saved package.json
Done in 0.13s.

Vita@DESKTOP-DKKIB9P MINGW64 /f/Ders/BashExamples/InitExample
$ yarn install
yarn install v1.19.1
info No lockfile found.
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

success Saved lockfile.
Done in 0.45s.

Vita@DESKTOP-DKKIB9P MINGW64 /f/Ders/BashExamples/InitExample
$ |
```

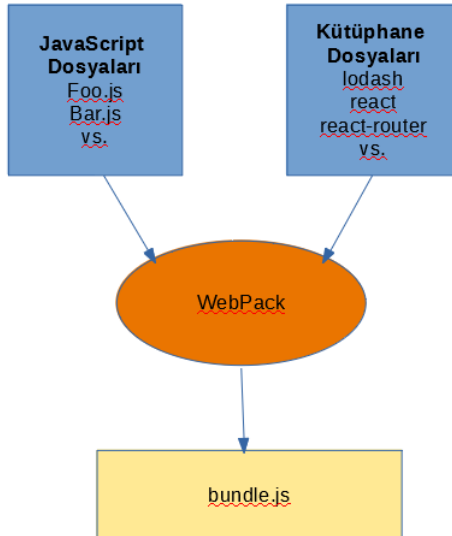
- yarn init -y
 - Mevcut klasörde yeni proje oluşturmak için gerekli package.json oluşturmak için kullanılır.
- yarn install
 - node_modules klasörüne package.json dosyasında tanımlı bağımlılıkları indirip kurmak için kullanılır.

- Projenin ana ayar dosyasıdır.
- Maven Java projelerindeki pom.xml dosyasına benzemektedir.
- Üç temel bileşeni bulunmaktadır.
 - **scripts**: YARN tarafından çalıştırılabilir komutlar bulunmaktadır. (Örnek: build, run vs.).
 - **dependencies**: Projede kullanılan bağımlılıklar belirtilir.
 - **devDependencies**: Yalnızca geliştirme ortamında kullanılan bağımlılıklar belirtilir. Uygulamanın son halinde kullanılmaz (WebPack gibi).

- Bağımlılıklar kurulduğunda, yarn.lock adında bir dosya oluşturulur.
- Bağımlılıklar tanımlandığında hangi sürüm olduğuna dair bir bilgi de içerir (1.1.13 gibi).
- \wedge karakteri en yakın minör versiyon ile eşleşmektedir.
 - Versiyonlama: majör.minör.patch
 - Örnek: \wedge 1.0.3 tanımı 1.5.7 ile eşleşirken, 2.0.0 ile eşleşmemektedir.
- yarn.lock bu dosya oluşturulduğunda kullanılan versiyonları barındırmaktadır.
 - Takım çalışmalarında oldukça önemlidir. Gerçekleştirilen güncellemeler projede bug oluşmasına sebep olabilir.

- Yalnızca bağımlılıkların indirilmesi yeterli değildir.
- Bu bağımlılıkların aynı zamanda HTML sayfaları tarafından erişilebilir olması gerekmektedir.
- Her bir bağımlılığı tek tek HTML dosyalarına eklemek oldukça zahmetli olabilir.
- Hatta büyük bir kütüphanenin yalnızca küçük bir işlevine ihtiyacımız olabilir.
- Bu problemlerin üstesinden gelmek için proje **WebPack** aracılığı ile paketlenabilir.

Paketleme (Bundling)



WebPack Ayarları

- Kurduktan sonra package.json dosyasında yarn tarafından çağırılması gerekmektedir.
- webpack-dev-server paketi değişiklikleri algılayarak HTTP server'ın yeniden başlamasını sağlayan kullanışlı bir araçtır.

```
1 "scripts": {  
2   "dev": "webpack-dev-server --open --mode  
    development",  
3   "build": "webpack --mode production"  
4 },  
5 "devDependencies": {  
6   "webpack": "^4.16.5",  
7   "webpack-cli": "^3.1.0",  
8   "webpack-dev-server": "^3.1.5"  
9 }
```

- WebPack'in kendi ayarları da bulunmaktadır.
 - Oluşturulacak dosya adı, klasör yolu gibi
- Ayarlar bir JavaScript dosyasında gerçekleştirilmektedir.

Kod Dönüştürme (Transformation)

- Paketleme yeterli olmamaktadır. Bazen kod dönüşümüne de ihtiyaç duyulmaktadır.
- TypeScript, JSX gibi diller desteklenmektedir.
 - Browser tarafından native olarak desteklenmeyen ve JS'e dönüştürülmesi gereken dillerdir.
 - React için JSX oldukça önemlidir.
- Eski browser desteği
 - Örnek: JS'in yeni özelliklerini eski eşdeğerlerine çevirmek için kullanılmaktadır.
- Boyut Küçültme
 - Yorum satırlarını ve boşluklar kaldırılarak daha hızlı indirilebilir küçük boyutta JS dosyaları oluşturulmaktadır.
- vs.

- JS dönüştürme için kullanılan temel araç Babel'dir.
- Kurduktan sonra package.json dosyasında ayarlama yapılmalıdır.

```
1 "babel": {  
2   "presets": [  
3     "@babel/env"  
4   ]  
5 },  
6 "devDependencies": {  
7   "@babel/cli": "7.7.4",  
8   "@babel/core": "7.7.4",  
9   "@babel/preset-env": "7.7.4",  
10  "babel-jest": "24.9.0"  
11 }
```

Kurulması Gereken Araçlar

- node kurulu olarak gelmeyen işletim sistemi kullanıyorsanız kurmanız gerekmektedir. <https://nodejs.org>
 - "node -version" komutu ile kontrol edebilirsiniz.
- <https://yarnpkg.com> adresinden YARN kurulmalıdır.
 - "yarn -version" komutu ile kontrol edebilirsiniz.
- WebPack/Babel/Jest: Manuel kuruluma gerek yoktur.
 - "yarn install" komutu kurulumu sizin için yapacak ve package.json içindeki devDependencies kısmına ekleyecektir.
 - Kurulan araçlar yerel olarak node_modules klasörüne eklenecektir. Böylelikle package.json içerisindeki scripts alanında kullanılabilir olacaktır.

5-Test

- Programın doğru çalışıp çalışmadığını kontrol etmek için test senaryolarının yazılması oldukça önemlidir.
- Dinamik yazımlı dillerde ise çok daha fazla önemlidir. Çünkü derleyicilerin yaptığı pek çok uyarıyı ve kontrolü yapmamaktadır.
- JS testi için pek çok kütüphane bulunmaktadır ancak bu ders kapsamında **Jest** kullanılacaktır.

Jest Ayarları

- Jest'in başlatılabilmesi için scripts içerisinde tanımlanmalıdır.
- Testlerin nerede bulunduğunu belirtmek için de ayrıca bir ayara ihtiyaç bulunmaktadır.

```
1 "scripts": {  
2   "test": "jest --coverage"  
3 },  
4 "jest": {  
5   "testRegex": "tests/.*-test\\.\\. (js|jsx)$",  
6   "collectCoverageFrom": [  
7     "src/**/*. (js|jsx) "  
8   ]  
9 }
```

Jest için Babel'e ihtiyaç bulunmaktadır.

- JS kodları tarayıcıda çalışmaktadır.
- Testler ise JS'i server üzerinde çalıştıran NodeJS'e ihtiyaç duymaktadır.
- Frontend kodları doğrudan NodeJS üzerinde çalıştırılamayabilir.
 - Örnek: JS modüllerinin kullanılması için farklı yöntemler gerekmektedir (import ve require() gibi).
- NodeJS üzerinde çalışması için gerekli bu türde bir dönüşümü Babel ile yapabiliriz.
- Jest'e Babel'i kullanacağını belirtmek için babel-jest kütüphanesine ihtiyaç vardır.