Web Development and API Design

Lesson 02: Bash, Regex, Build Tools and Testing

Prof. Andrea Arcuri

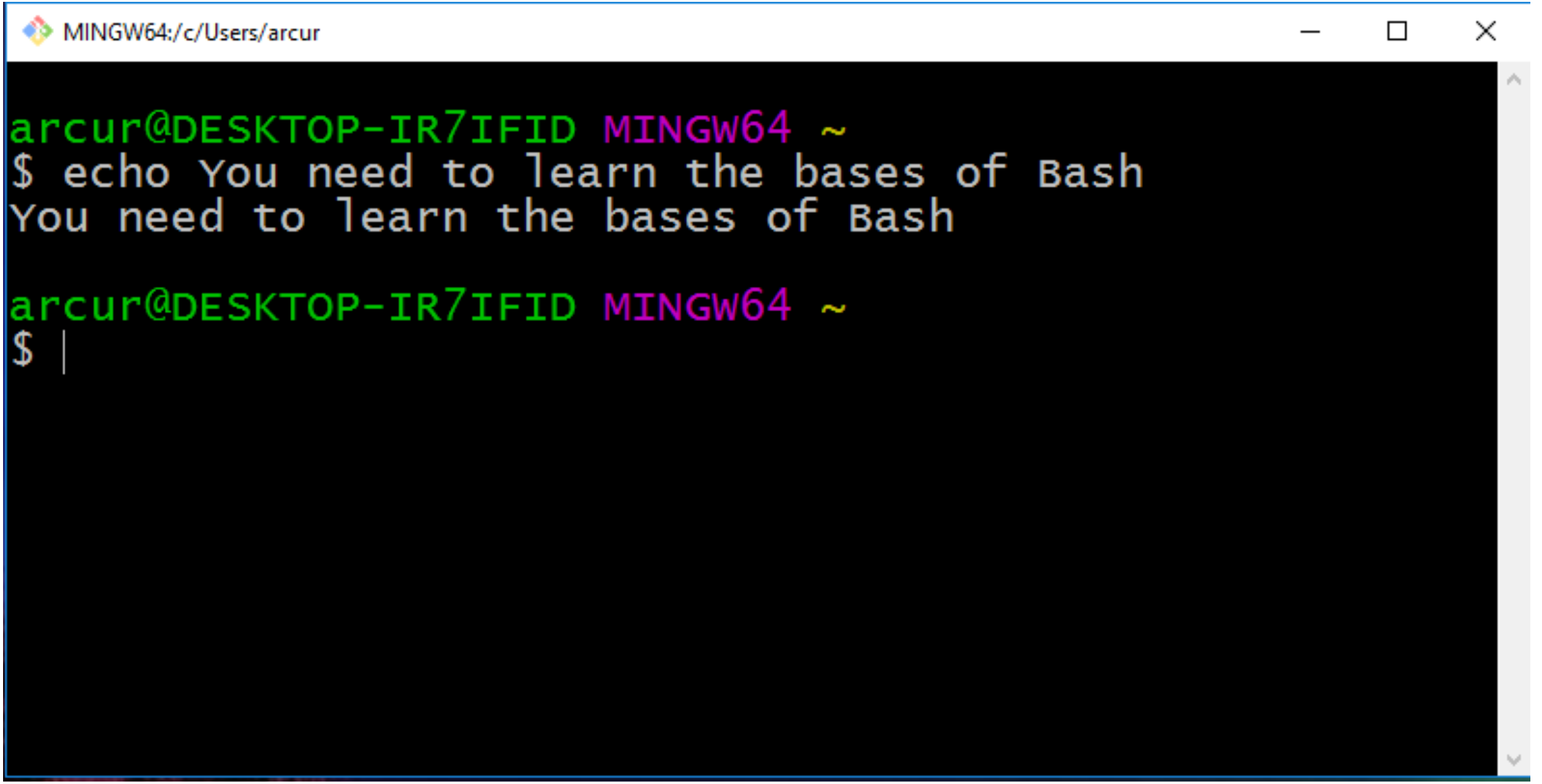
Goals

- Review/intro to Bash Terminal
- Review/intro to Regular Expressions
- Build tools: **YARN**, **WebPack** and **Babel**
- How to write test cases

Bash

Bash

- Bash is a Linux/Mac/Unix shell and command language
- There are also other kinds of shells
  - eg, PowerShell in Windows
- A *shell* is also called: *terminal*, *console*, *command-line*, etc.
- Enable to *type* commands (eg programs), and execute them

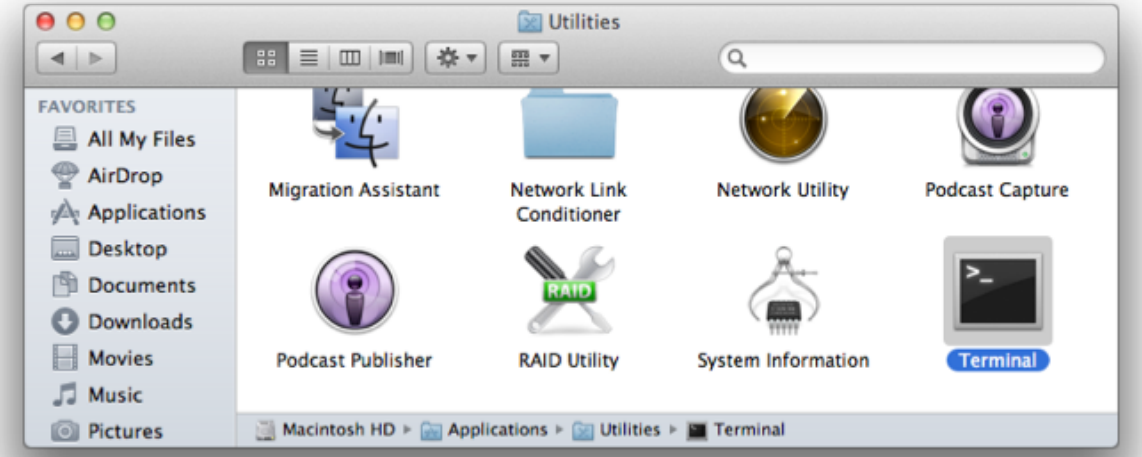

MINGW64:/c/Users/arcur
arcur@DESKTOP-IR7IFID MINGW64 ~
$ echo You need to learn the bases of Bash
You need to learn the bases of Bash
arcur@DESKTOP-IR7IFID MINGW64 ~
$

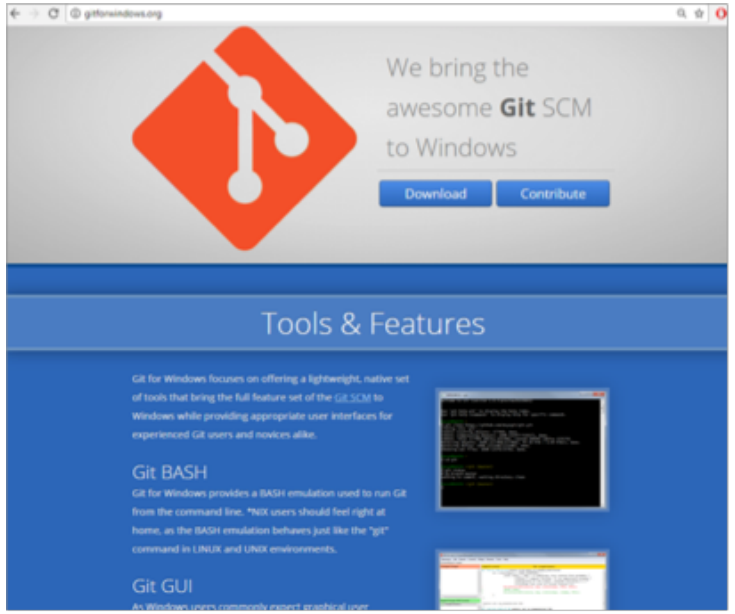
Why?

- Critical skill when you are a programmer
- Help automating several tasks
- When dealing with web/enterprise systems, many servers will NOT have a GUI…
  - … you will access them remotely via SSH using a terminal
  - … this also applies for embedded and IoT devices
- Helpful when commands with specific parameters (eg Git)
- You need to be able to do basic commands
- You will need *Bash* commands when using build tools

Installing Bash

- If you are using Linux/Mac, it is already installed
  - Mac: Utilities -> Terminal
- If using Windows, strongly recommended to install GitBash
  - which is part of “Git for Windows” at http://gitforwindows.org/





Basic Commands

- “**.**” the current directory
- “**..**” the parent directory
- “**~**” home directory
- “**pwd**” print working directory
- “**cd**” change directory
- “**mkdir**” make directory
- “**ls**” list directory content
- “**cp**” copy file
- “**mv**” move file
- “**rm**” remove (“-r” for recursive on directories)
- “**man**” manual for a specific command

Cont.

- “**echo**” print input text
- “**cat**” print content of file
- “**less**” scrollable print of file
- “**>**” redirect to
- “**>>**” append to
- “**|**” pipe commands
- “**which**” location of program
- “**$**” resolve variables
- “**wc**” word count
- “**find**” files
- “**grep**” extract based on regular expression
- “**touch**” modify access time of file, and create it if non-existent

MINGW64:/e/WORK/teaching/bash_examples

```
arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples
$ pwd
/e/WORK/teaching/bash_examples

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples
$ ls

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples
$ echo "ciao" > foo.txt

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples
$ ls
foo.txt

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples
$ cat foo.txt
ciao
```

MINGW64:/e/WORK/teaching/bash_examples/foo

```
arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples
$ mkdir foo

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples
$ ls
foo/  foo.txt

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples
$ cd foo

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples/foo
$ pwd
/e/WORK/teaching/bash_examples/foo

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples/foo
$ ls ..
foo/  foo.txt
```

MINGW64:/e/WORK/teaching/bash_examples/foo

```
arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples/foo
$ cp ../foo.txt ./bar.txt

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples/foo
$ cat bar.txt
ciao

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples/foo
$ ls
bar.txt

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples/foo
$ mv ../foo.txt  .

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples/foo
$ ls
bar.txt  foo.txt
```

MINGW64:/e/WORK/teaching/bash_examples/foo

```
arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples/foo
$ echo $PATH
/c/Users/arcur/bin:/mingw64/bin:/usr/local/bin:/usr/bin:/bin:/mingw64/bin:
/usr/bin:/c/Users/arcur/bin:/c/Program Files/Docker/Docker/Resources/bin:/
c/Users/arcur/bin:/c/Program Files/Java/jdk1.8.0_112/bin:/c/Users/arcur/ap
ache-maven-3.3.9-bin/apache-maven-3.3.9/bin:/c/ProgramData/Oracle/Java/jav
apath:/c/WINDOWS/system32:/c/WINDOWS:/c/WINDOWS/System32/Wbem:/c/WINDOWS/S
ystem32/WindowsPowerShell/v1.0:/cmd:/c/Program Files/MiKTeX 2.9/miktex/bin
/x64:/c/HashiCorp/Vagrant/bin:/c/Program Files/nodejs:/c/Program Files (x8
6)/Skype/Phone:/c/Program Files/PostgreSQL/9.6/bin:/c/Program Files/Micros
oft SQL Server/130/Tools/Binn:/c/Program Files/dotnet:/c/Program Files (x8
6)/GtkSharp/2.12/bin:/c/RailsInstaller/Ruby2.2.0/bin:/c/DevelopmentSuite/c
dk/bin:/c/HashiCorp/Vagrant/bin:/c/DevelopmentSuite/cygwin/bin:/c/Users/ar
cur/AppData/Local/Microsoft/WindowsApps:/c/Users/arcur/AppData/Roaming/npm
:/c/Program Files/Heroku/bin:/c/Users/arcur/AppData/Local/Microsoft/Window
sApps:/usr/bin/vendor_perl:/usr/bin/core_perl

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/teaching/bash_examples/foo
$ which bash
/usr/bin/bash
```

- What if you want to count the number of JavaScript files in your project?

- Or count the total number of lines in all those files?

```
arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/code/teaching/pg6300 (spring-2019)
$ find . -regex '^.*\.jsx?' -not -path */node_modules/* | wc -l
141

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/code/teaching/pg6300 (spring-2019)
$ cat `find . -regex '^.*\.jsx?' -not -path */node_modules/*` | wc -l
12272
```

- "*find*" all the files recursively in the current folder "."
  - WARNING: on Mac, you will need to add **–E**, ie "find -E"
- matching the regular expression for JS/JSX files
  - "**^**" beginning of file name
  - "**.***" any character (.), any number of times (*)
  - "**\.**" escaped any-character to represent the character "."
  - "**\.jsx?**" file ending, where the last "x" is optional (ie "?")
  - "**-not -path */node_modules/***" excludes files of imported dependencies
- "**| wc -l**": pipe file names to line count program
- **cat `x`**: the `` executes the command inside it, and then puts the output on the terminal
  - so, we print all content of all JS/JSX files with **cat**

Useful Tips

- User arrows (up/down) to go through history of commands
- Use “tab” key to complete words, ie commands / file names
- Bash commands can be put in executable scripts
  - Can use “*.sh” as file extension, eg “*foo.sh*”
  - First lines needs to be “*#!<pathToBash>*”, eg “*#!/usr/bin/bash*”
  - Then it can be executed from terminal like any other program

Regular Expressions

Constraints

- Is "*loreipsum.com*" a valid email address?
  - no, it does not have the symbol "@"
- Is "*3fenfrje35ddsre123345*" a valid telephone number?
  - no, it contains non-digits, and it is likely too long
- When a String represents a specify type of value (eg, emails), we can use a regex to specify its *constraints*
  - eg, the subset of all possible strings representing a valid email
  - eg, common when validating inputs in HTML forms

Definition, Regex is either:

1. An empty set
2. An empty string
3. A single character
4. A regex enclosed in parentheses **()**
5. Two or more concatenated regexs
6. Two or more regexs separated by *or* operator **|**
7. A regex followed by the the closure operator ***

Matching Characters

- Wildcard “.” matches any character
  - eg, “a.b” match any 3-letter word starting with “a” and ending with “b”
- Set **[]** matches any single character in the set
  - eg, [abc] does match “a”, “b” and “c”, but not “d”, nor “ab”
- Range **[-]** matches in the range
  - eg, [a-z] matches any lower case letter, [a-zA-Z] any letter, [0-8] any digit but 9
- Meta-characters need to be escaped with “\”
  - eg, “\[\.\]” would match “[.]”, but not “a” nor “[a]”

(), | and *

- **()** use to define boundary of a regex
- **|** is an or between two expressions
- ***** applies the previous regex 0 or more times
- Eg, "*ab**" does match "*a*", "*ab*" and "*abbbbbbbb*"
- Eg, "*(ab)**" does match "", "*ab*", "*abab*" and "*abababab*"
- Eg, "*(ab)|c*" does match "*ab*" and "*c*"

Shortcuts

- "**+**" at least once
  - "*x+*" equivalent to "*xx**"
- "**?**" zero or one time
  - "*x?*" equivalent to "*emptyString | x*"
- "**{}**" specific number of times
  - "*x{5}*" equivalent to "*xxxxx*"
  - "*x{2,4}*" equivalent to "*(xx)|(xxx)|(xxxx)*"

Example: Telephone Number

8 digit number

eg, 40012345

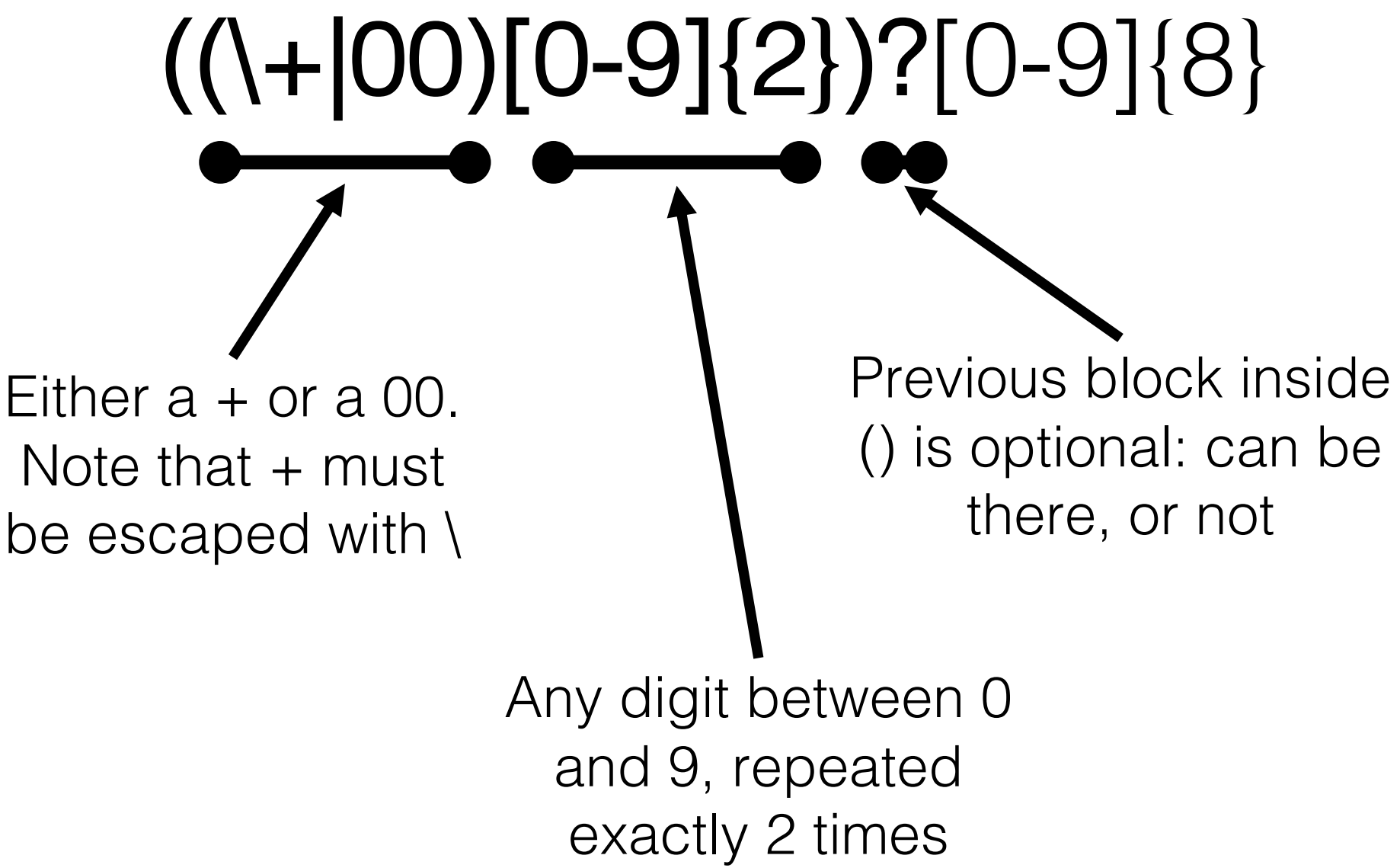
Might be preceded by a country code, which is either a + or 00 followed by 2 digits

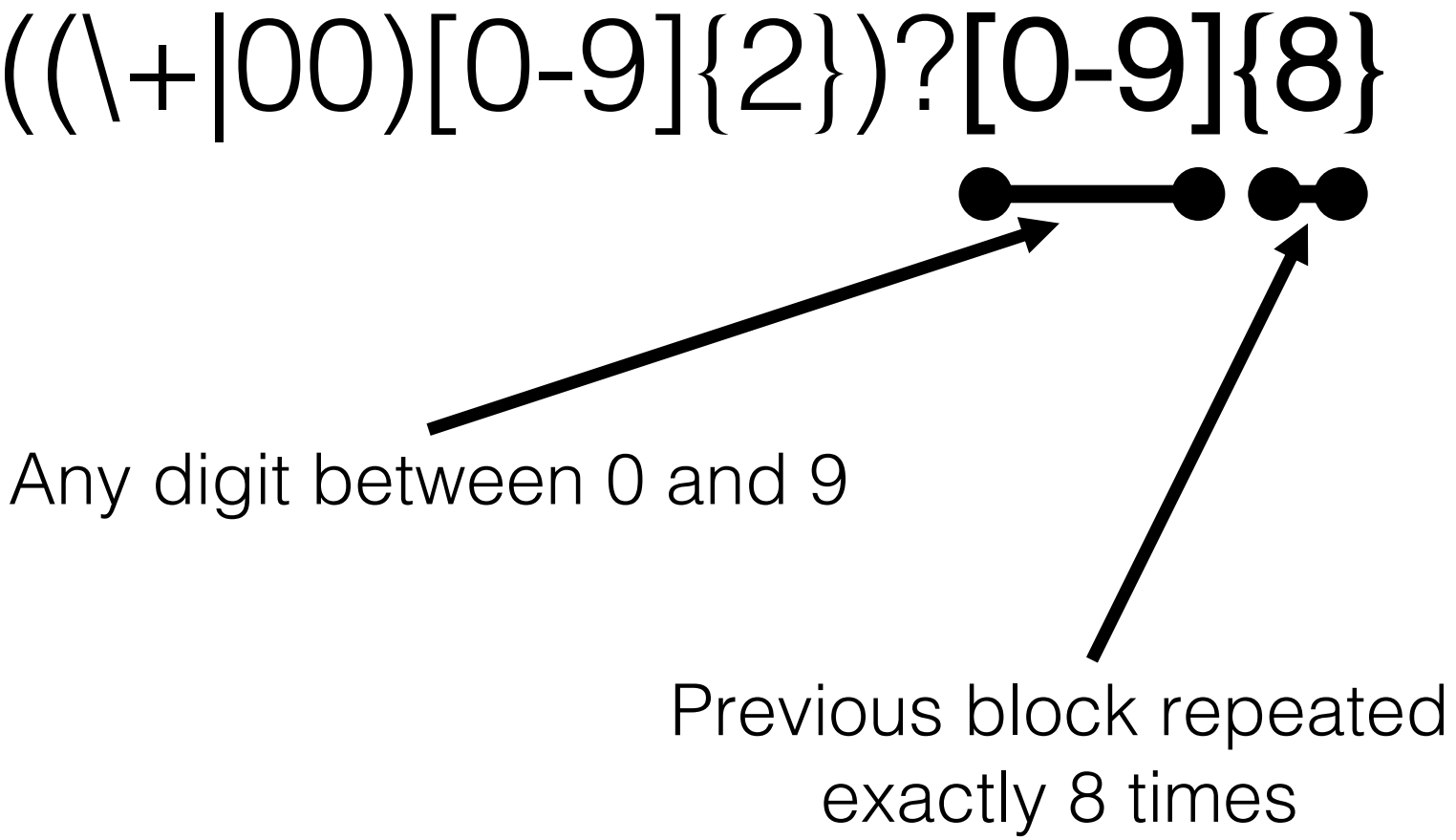
eg: +47 or 0047

Regex for Telephone Number

((\+|00)[0-9]{2})?[0-9]{8}

First time you see a regex…


((\+|00)[0-9]{2})?[0-9]{8}
Either a + or a 00. Note that + must be escaped with \
Any digit between 0 and 9, repeated exactly 2 times
Previous block inside () is optional: can be there, or not


((\+|00)[0-9]{2})?[0-9]{8}
Any digit between 0 and 9
Previous block repeated
exactly 8 times

Limitations of Regex

- Regex are very useful in many contexts to check the validity of strings that are supposed to have constraints
- However, they are not fully expressive, as there are constraints you cannot express with a regex
- Example: you cannot use a regex to check if a string is a valid JavaScript code
  - for that, you need a Context-Free Grammar, but we will not see them in this course

Do Not Use Regex for HTML!!!

https://stackoverflow.com/questions/1732348/regex-match-open-tags-except-xhtml-self-contained-tags

… Regex-based HTML parsers are the cancer that is killing StackOverflow it is too late it is too late we cannot be saved the trangession of a child ensures regex will consume all living tissue (except for HTML which it cannot, as previously prophesied) dear lord help us how can anyone survive this scourge using regex to parse HTML has doomed humanity to an eternity of dread torture and security holes using regex as a tool to process HTML establishes a breach between this world and the dread realm of corrupt entities (like SGML entities, but more corrupt) a mere glimpse of the world of regex parsers for HTML will instantly transport a programmer's consciousness into a world of ceaseless screaming, he comes, the pestilent slithy regex-infection will devour your HTML parser, application and existence for all time like Visual Basic only worse he comes he comes do not fight he comes, his unholy radiance destroying all enlightenment, HTML tags leaking from your eyes like liquid pain, the song of regular expression parsing will extinguish the voices of mortal man from the sphere I can see it can you see it it is beautiful the final snuffing of the lies of Man ALL IS LOST ALL IS LOST the pony he comes he comes he comes the ichor permeates all MY FACE MY FACE oh god no NO NOOOO NO stop the anales are not real ZALGO IS TONY THE PONY HE COMES

Build Tools

YARN/NPM

- We need to use *external libraries*, typically open-source
  - An important library we are going to use in the rest of the course is for example *React*
- Two main tools in JS: **YARN** and **NPM**
- Both **YARN** and **NPM** access the same dependency repository
- **YARN** tends to be better, with new features coming earlier
- We will use it from terminal
- As **YARN** executes JS code, we need a runtime for it: that is the reason why you also need to install **NodeJS**
  - not going to start a build tool inside a browser…

```
arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/code/teaching/foo/example
$ yarn init -y
yarn init v1.12.3
warning The yes flag has been set. This will automatically answer yes to all questions, which ma
ty implications.
success Saved package.json
Done in 0.05s.

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/code/teaching/foo/example
$ ls
package.json

arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/code/teaching/foo/example
$ yarn install
yarn install v1.12.3
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 0.12s.
```

- **yarn init –y**
  - create a new *package.json* in current folder, needed when starting new project
- **yarn install**
  - download and install in “*node_modules*” folder all the dependencies declared in *package.json*

package.json

- Main configuration file for the project
- Similar to *pom.xml* in Maven Java projects
- Three main parts you need to care about:
  - **scripts**: executable commands from YARN. Eg, to build or run the app
  - **dependencies**: dependencies used in the project
  - **devDependencies**: dependencies only used during development, but not being part of the final app (eg, we will see *WebPack*)

JSON for Configuration Files

<rant>

JSON as format for configuration files is simply **awful**.

For example, you cannot have comments…

NPM is not better, as uses exactly the same *package.json*

</rant>

yarn.lock

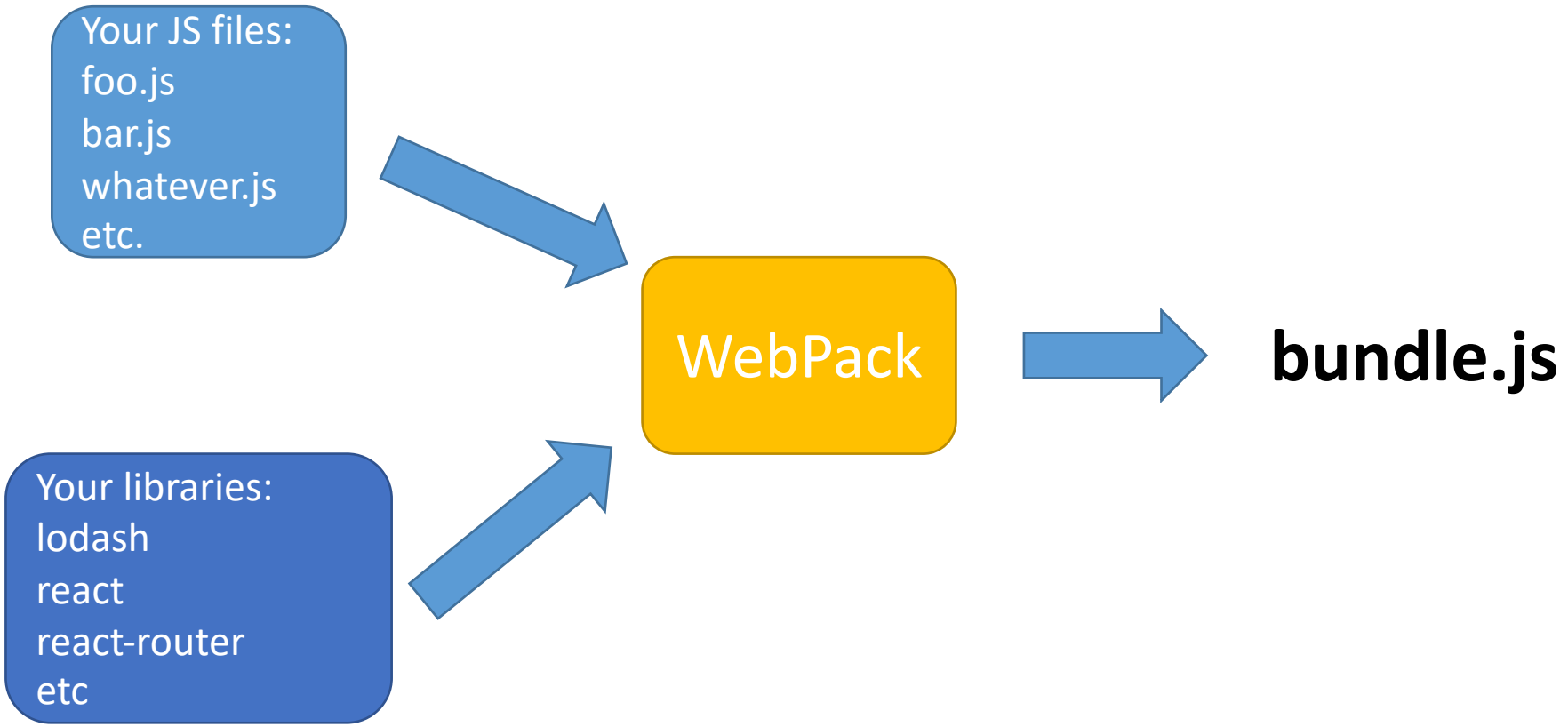
- Once you install the dependencies, you will see that YARN does create a *yarn.lock* file
- Dependencies need to define which version to use, eg 1.0.2
- You can use caret **^** to represent the most recent major version
  - e.g., **^1.0.2** will match **^1.4.1**, but not **2.0.0**
- *yarn.lock* just tells YARN to use the exact same versions of the libraries when such file was created
  - extremely important when working in a team, and new minor updates break backward compatibility or introduce new regression bugs
  - even if you fix a version X of a dependency, this could use other third-dependencies with **^**, and so *yarn.lock* becomes essential

WebPack

- Downloading dependencies is not enough
- Such dependencies need to be accessed by the HTML pages
- Might be cumbersome to update HTML files for each dependency, for each different version
- Furthermore, we might only need a small set of functions from a specific library
- Solution: *bundling* with **WebPack**

Bundling

In the end, we get a *single* JS file



Configuring *WebPack*

- Needs to be installed and called with YARN from *package.json*
- *webpack-dev-server* is a useful tool that starts a HTTP server with hot-reloading
  - ie, if you modify your JS files, it automatically re-bundle them

```
"scripts": {
    "dev": "webpack-dev-server --open --mode development",
    "build": "webpack --mode production"
},
"devDependencies": {
  "webpack": "^4.16.5",
  "webpack-cli": "^3.1.0",
  "webpack-dev-server": "^3.1.5"
}
```

webpack.config.js

- Besides being called from *package.json*, WP also needs its own configurations
  - Eg, name of the file to create, and which directory to save it into
- Configuration done in a JavaScript file

Code Transformation

- Bundling is not enough, might need to do *transformations*
- Support other languages: *TypeScript* and **JSX**
  - which are not natively supported by browsers, which only deal with JS
  - JSX will be essential when dealing with *React*
- Support old browsers:
  - eg, transform code using new JS features (eg, *async/await*) into equivalent, valid old JS
- Minification:
  - eg, remove comments and empty spaces from JS files to decrease their size, needed to make their download faster
- etc.

Babel

- **Babel** is the main tool to make JS transformations
- Need to be installed and configured from *package.json*

```
"babel": {
    "presets": [
      "@babel/env"
    ]
},
"devDependencies": {
    "@babel/cli": "7.7.4",
    "@babel/core": "7.7.4",
    "@babel/preset-env": "7.7.4",
    "babel-jest": "24.9.0",
```

Tools to Install

- If your OS does not come with "*node*", install it from *https://nodejs.org*
  - run "*node --version*" on terminal to check if installed and its version
- Install YARN from https://yarnpkg.com/
  - run "*yarn -version*" on terminal to check it
- *WebPack/Babel/Jest*: no need to do any manual installation
  - "*yarn install*" will do it for you, if those tools are added under *devDependencies* in the *package.json* file
  - tools will be added "locally" into the *node_modules* folder, and can be used from inside the "*scripts*" object in *package.json*

Testing

Testing

- To check the correctness of a program, writing test cases is very important
- For *dynamically typed* language, is even more important
  - as you lose a lot of warnings and checks from a compiler
- Different libraries in JS for testing, but we will use **Jest**
  - which is one of the most popular

Configuring *Jest*

- Need entry in **scripts** to start it
- Need extra configurations to find out where the tests are

```
"scripts": {
  "test": "jest --coverage"
},
"jest": {
  "testRegex": "tests/.*-test\\.(js|jsx)$",
  "collectCoverageFrom": [
    "src/**.(js|jsx)"
  ]
}
```

Need *Babel* for *Jest*

- JS code running in browser
- But where are the tests run? We need a JS runtime: *NodeJS*
- But *frontend* code might not be directly executed on *NodeJS*
  - eg, different ways to handle JS *modules,* eg, "*import*" vs "*require()*"
- Using *Babel* to make the required transformations to be able to run such code on *NodeJS*
- To tell *Jest* to use *Babel*, you need to add as well the library *babel-jest*