

front-end

prof. Lucas Ferreira



engenharia de
software



engenharia de
computação





Introdução ao Node.js + NPM, ferramentas e Git + Github



Node.js

- Roda em Linux, Mac e Windows
- Abordagem orientada a eventos
- Melhor preparado para trabalhar com interações de tempo real
- **Destaque:** SockJS, Socket.IO e Engine.IO
- Muita contribuição open-source
- Gerenciador de Pacotes oficial: NPM
- Acompanha a evolução do ECMAScript



Node.js

- Criado por **Ryan Dahl** em **2009**
- Baseado na **engine V8**, desenvolvida pela Google e usada no Google Chrome
- O **Node.js** se preocupa em oferecer um ambiente de execução
- Enquanto que a engine V8 se preocupa com a interpretação do código JavaScript
- Uso recomendado para o desenvolvimento de APIs e aplicações de tempo-real
- Caracterizado por sua arquitetura baseada em eventos (Event-Loop), bem como ao I/O não bloqueante que tem foco em tarefas assíncronas
- **Node.js** trabalha por meio de single-thread (o que pode ser um problema)



Instalação do Node.js

Acesse o site oficial <https://nodejs.org> e siga o guia!

PS.: Instale sempre uma versão estável e LTS



"Hello World"

Presume-se que após instalado seu executável do **Node.js** esteja disponível no **PATH** de seu sistema operacional. Abra seu *Terminal/Prompt/Powershell* e digite:

```
node
```

Neste momento você estará com o console interativo do **Node.js** aberto 🙌



"Hello World"

Crie um arquivo chamado *hello.js* e insira o seguinte código:

```
console.log("Hello World");
```

Logo após volte ao seu terminal/prompt e execute seu arquivo recém criado:

```
node hello.js
```



API Nativa

A API nativa de módulos do Node.js é muito extensa e poderosa, temos diversos módulos a disposição para criarmos recursos poderosos com o runtime.

Alguns destes módulos 🖐

- [http](#)
- [console](#)
- [fs](#)
- [net](#)
- [path](#)
- [os](#)
- [process](#)
- [stream](#)
- [url](#)
- ...e muito [mais aqui!](#)



HelloWorld.txt

Utilizando o *módulo* `fs` é possível criar, manipular e excluir arquivos físicos do sistema, conforme o exemplo abaixo:

```
const fs = require("fs");
fs.writeFile("HelloWorld.txt", "Hello World ;)", err => {
  if (err) throw err;
  console.log("o arquivo foi salvo com sucesso");
});
```



Hello World no Navegador

Utilizando o *módulo http* é possível criarmos um pequeno servidor web que responderá a solicitações http direto de seu navegador:

```
const http = require("http");
const port = 5000;
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader("Content-Type", "text/plain");
  res.end("Hello World de novo!\n");
});
server.listen(port, () => {
  console.log(`Server running at http://localhost:${port}/`);
});
```



Gerenciamento de pacotes com NPM

- NPM é um gerenciador de pacotes para código JavaScript
- Focado na reutilização de código compartilhado, onde pequenos pacotes são criados para solucionar problemas específicos
- É possível instalar um pacote de forma global (para toda a máquina):
`npm install -g express`
- Ou instalar um pacote de forma local (apenas para o projeto atual):
`npm install express`
- Gerenciamento de pacotes locais ocorre por meio do arquivo **`package.json`**
- Todos os pacotes públicos encontram-se em <https://www.npmjs.com>



Hello World no Navegador (*mais esperto*)

Neste exemplo iremos instalar um pacote chamado express, definido como "um framework para aplicativo da web do Node.js mínimo e flexível que fornece um conjunto robusto de recursos para aplicativos web e móvel".

```
const express = require("express");
const app = express();
const port = 5000;
app.get("/", function (req, res) {
  res.send("Hello World!");
});
app.get("/teste", function (req, res) {
  res.send("Teste");
});
app.listen(port, function () {
  console.log(`Running at http://localhost:${port}/`);
});
```



Pacotes interessantes

Pacote	Descrição
standardjs	Guia de estilo JavaScript, com analisador e fixador de código automático.
dotenv	Pacote para carregar variáveis de ambiente.
socket.io	Framework para criação de aplicações de tempo real.
swagger	Pacote composto por ferramentas para projetar e criar APIs.
express	Framework minimalista para desenvolvimento de aplicações web.
helmet	Pacote de apoio para proteção de aplicações web.
sequelize	ORM para Node.js (Banco de dados relacionais).
mongoose	ODM para Node.js (MongoDB).
winston	Pacote para armazenar log assíncrono.
morgan	Pacote para armazenar log de requisição HTTP.
supertest	Pacote para teste em alto nível de requisições HTTP.
mocha	Pacote para teste unitário.

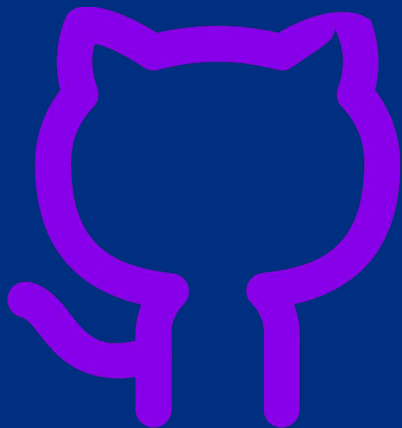


LET'S CODE!





GIT & GITHUB





GIT x GITHUB

Antes de tudo é preciso saber que Git não é GitHub.

GIT é um sistema distribuído de controle de versão de código, assim como o SVN e o clássico CVS.

GitHub é um dos locais onde você pode "hospedar" seus repositórios/arquivos organizados pelo sistema Git na web/nuvem. Atualmente é a principal plataforma do mercado, porém existem outros concorrentes igualmente úteis como *GitLab* e *Bitbucket*.



GITHUB

O **GitHub** iniciou suas atividades com o modelo de ser gratuito para projetos abertos e pago para projetos privados.

Dado seu modelo inicial de ser gratuito para projetos open-source, a comunidade de desenvolvedores web adotou com rapidez e garantiu o uso amplo da plataforma para diversos projetos e bibliotecas. Hoje todo projeto open-source "de respeito" possui um repositório para contribuição no GitHub.

Toda a plataforma do GitHub foi adquirida pela Microsoft no ano de 2018 por **\$7.5 bilhões!**



Preparando o terreno

Partindo do princípio que Git é o software/sistema utilizado para versionar códigos, e que GitHub é uma das plataformas possíveis para "armazenar" esses códigos na "nuvem", iremos na verdade instalar o GIT em nossas máquinas (*e não o github em si*).

Por tanto o GIT instalado na máquina irá cuidar do projeto local e quando estivermos seguros de que o projeto "avançou" (feature, bugfix, conclusão e etc) iremos usar o mesmo GIT para "enviar" nosso código para alguma plataforma on-line segura, como por exemplo o GitHub.

Somente com um servidor distribuído (*e on-line*) compatível com GIT é que iremos poder compartilhar nossos códigos com outros membros de um mesmo projeto, principalmente com membros que trabalham a distância.



Preparando o terreno

Existem duas maneiras usuais de termos o "GIT" presente em nossa máquina:

1. Instalando o Git "oficial" 🖱️ <https://git-scm.com/downloads>; ou
2. Instalando o GitHub Desktop que apesar de seu modo "visual" vem um cliente de Git embutido em sua instalação 🖱️ <https://desktop.github.com>.

Eu recomendo o primeiro método, pois nem todos os projetos que você irá trabalhar na vida serão hospedados no GitHub.



Uso básico

Não sou o maior especialista em GIT do mundo, o cliente que roda em linha de comando tem uma infinidade de comandos que na grande maioria dos projetos de pequeno/médio porte nunca serão usados. Tudo que eu sei são coisas que já precisei fazer e deu tudo "certo".

Na sequência alguns poucos e importantes comandos para auxiliar quem ainda não "manja" de Git/GitHub.



Uso básico

Logo após instalar o GIT a primeira coisa que temos que fazer é conectar nossa conta do GITHUB nele.

```
# definindo o "nome" do responsável pelas alterações  
git config --global user.name "lucasferreira"
```

```
# definindo o e-mail usado na conta do github para login  
git config --global user.email panchorf@gmail.com
```



Uso básico

Na sequência vale observar que não será apenas o e-mail definido na etapa anterior que nos dará autorização de mudar arquivos "hospedados" no GitHub.

Para nos identificarmos como oficialmente responsáveis por um determinado repositório de código precisaremos nos autenticar de outras formas.

Uma delas é trabalhando com repositórios baseados em HTTPS, assim no momento que formos enviar os arquivos para o GitHub teremos que digitar a senha usada na plataforma para nos autenticar.



Uso básico

Neste link temos um detalhamento melhor sobre o processo de HTTPS como conexão/autenticação do GitHub:

<https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls>



Uso básico

Outra opção seria nos autenticar usando chave pública/privada trabalhando com SSH ao invés de HTTPS. Apesar de "mais seguro" não é o método recomendado para quem está começando com Git / GitHub pois aumenta em muito a complexidade de configuração da máquina.

Caso queiram entender melhor como configurar sua máquina para conectar no GitHub via SSH, usem o link abaixo:

<https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-ssh-urls>



Uso básico

Uma vez que estejamos com o GIT instalado e devidamente configurado, temos algumas opções para chegar até o GitHub:

Iniciar uma pasta/projeto em nossa máquina a partir de um repositório já existente no GitHub

HTTPS

```
git clone https://github.com/lucasferreira/slides-sw-aula-05.git
```

ou SSH

```
git clone git@github.com:lucasferreira/slides-sw-aula-05.git
```



Uso básico

Uma vez que estejamos com o GIT instalado e devidamente configurado, temos algumas opções para chegar até o GitHub:

Iniciar uma pasta/projeto em nossa máquina e DEPOIS conectar ela em algum repositório já existente no GitHub

```
# após criar a pasta entrar dentro dela e rodar esse comando  
git init
```

```
# no momento que achar necessário conectar essa pasta a algum  
# repositório remoto, por exemplo GitHub  
git remote add origin https://github.com/lucasferreira/slides-sw-aula-05.git
```



Uso básico

Ao iniciar um repositório primeiro em nossa máquina para depois associar ao github/gitlab vale observar que dependendo da tua configuração de máquina podemos ter um probleminha em relação ao nome do branch principal, segue um bom artigo:

<https://melix.github.io/blog/2020/06/updating-git.html>



Uso básico

Enquanto nós não termos certeza que nossa pasta está associada a um repositório on-line (ex: GitHub) nossas alterações de código ficarão apenas em nossa máquina.

Dando continuidade, quando fizermos nossas primeiras mudanças de arquivos (novos ou alterados), devemos adicionar ao "registro" do git local os arquivos em questão para que dali em diante o git "cuide" deles:

```
# arquivo individual  
git add meu_arquivo.txt
```

```
# pasta inteira  
git add meu_diretorio
```

```
# tudo que entrou de novo por ultimo, sejam pastas ou arquivos  
git add .
```



Uso básico

Após adicionar os arquivos ao registro do Git, no momento que acharmos que avançamos o suficiente podemos criar um registro na linha do tempo de evolução do projeto, através do comando commit:

```
git commit -m "nova tela adicionada ao projeto"
```

O comando acima irá reunir todas as mudanças e novidades pendentes e registrar sobre um registro incremental identificado pela mensagem que escrevemos ali junto. Neste momento ainda estamos apenas em nossa máquina, e podemos "registrar" quantos commits quisermos.



Uso básico

Depois dos commits quando acharmos que é a hora de "distribuir" nossa evolução de código e enviar para plataforma GitHub todos os commits que fizemos até então (contendo os seus arquivos), precisaremos usar o seguinte comando:

```
# o primeiro push para o servidor deveremos indicar qual o branch de trabalho  
# principal repositórios das antigas `master` mais modernos serão `main`  
git push -u origin master
```

```
# depois nos próximos envios ao GitHub apenas  
git push
```



Uso básico

Por fim, se você estiver trabalhando em mais de uma máquina ou tiver mais de uma pessoa contribuindo com seu projeto, para receber as últimas atualizações de um repositório remoto, utilizei o comando *pull*:

```
git pull
```



Uso básico

Para outros comandos e ideias sugiro consultar essa lista de comandos aqui:

<https://gist.github.com/lucasferreira/89cef8ba30057994bf08d578d3075e89>


Por fim, vale observar que todos os comandos e dicas que eu dei usando o "*terminal / shell / powershell / CMD*" são todos possíveis de serem substituídos por algum uso equivalente em programas visuais com o VS Code (*que possui um cliente de git embutido*) e também o já falado GitHub Desktop.



Ferramentas de Desenvolvimento



Babel.js

Você  desenvolvedor moderno e antenado que entende que o desenvolvimento JavaScript ágil deve usar os últimos recursos da linguagem, sabe que algumas situações (*ou projetos*) o suporte a navegadores antigo pode ser importante?

Por exemplo as arrows functions `(() => { /* código */ })` até a versão 45 do Google Chrome não era suportada mesmo estando na especificação. E se você estiver desenvolvendo um projeto que precisa rodar no Internet Explorer ou em versões mais antigas do Chrome/Firefox?

Bom é aí que entra o **Babel.js** (<https://babeljs.io/>), um "compilador" de JavaScript que interpreta o seu código "moderno" e "compila/transforma" este mesmo código em uma versão mais "antiquada" com suporte a diversos navegadores antigos.



Babel.js

Por exemplo, o Babel.js pega o código abaixo:

```
const items = [1, 2, 3, 4];  
items.map(n => n ** 2);
```

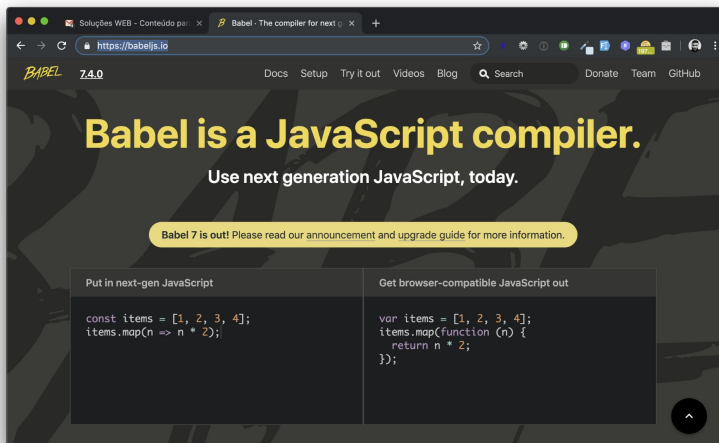
E cria uma versão compatível com qualquer navegador (antigo ou novo):

```
var items = [1, 2, 3, 4];  
items.map(function (n) {  
  return Math.pow(n, 2);  
});
```



Babel.js

Para uma pequena demonstração acessem o site <https://babeljs.io/>:





Webpack

Agora a brincadeira está ficando mais séria. Imagine que seu projeto cresceu, você agora possui muitos arquivos JavaScript e está complicado de organizar tudo isso em um único fluxo de carregamento na parte de front-end.

Você também deseja agilizar o Babel.js para que ele funcione em vários arquivos ao mesmo tempo e de forma mais automatizada possível.

Te apresento o "module bundler" **Webpack** (<https://webpack.js.org/>), uma das ferramentas de desenvolvimento front-end mais versáteis do mercado.

A mais uma dica, se você for investir no Webpack, vale a pena dar uma olhadinha nesse site aqui **Webpack Configurator** (<https://createapp.dev>)



Rollup, Parcel e Vite.js

Outras ferramentas na mesma linha do Webpack que também têm potencial de mercado são:

- Rollup: <https://rollupjs.org/>
- Parcel: <https://parceljs.org/>
- Vite.js: <https://vitejs.dev/>



—
obrigado 🚀

