Tuto NDI

Un document récapitulant tout ce que vous devez savoir pour pouvoir travailler le jour de la NDI

Sommaire

I - Notes

Pour toutes les libs, logiciels et langages qui seront nommé dans ce tuto auront **automatiquement un lien vers son site web respectif**. Exemple GIT.

Vous pourrez continuer à suivre **sans devoir remontrer tout en haut du tuto**.

À noté aussi que le mot `terminal` sur linux **est équivalent** à `Git bash` sur windows. Vous le verrez plus tard quand git sera installé

3 versions existe de ce document, une en noir et blanc classique, une autre (celle ci) dans un thème clair mais pas blanc, et un dernier dans un thème plus sombre

Ce tuto est réalisé avec le font `Fira Code` et avec l'outil nommé Typst.

Dans un code, quand vous voyez `<QuelqueChose>` vous devez **absolument remplacer** `<QuelqueChose>` par quelque chose. Il est très souvent expliqué soit par le nom à **l'intérieur** soit par le **contexte**.

Exemple:

```
1  echo "<EMAIL>" > bdd.csv
```

se transforme en

```
1  echo "jean@michel.com" > bdd.csv
```

La même chose avec les `[Truc]` sauf que vous pouvez très bien ne rien mettre.

Les 4 versions majeures qui seront traitées dans ce tuto sont:
- Windows sans WSL
- Ubuntu/Debian/WSL
- Fedora
- Arch Linux

Pour toute demande particulière ou fautes d'orthographes éventuelles, contactez moi par ce mail.

II - Installation du projet

Pour pouvoir programmer sur le projet, il faudra vous munir de plusieurs logiciels tel que:
1. Git
2. NodeJS

Le reste sera automatiquement installé grâce à ces logiciels

II.1 - Git

Première chose à avoir pour pouvoir travailler sur le projet est Git. Pour faire court, ça permet de travailler à plusieurs sur un projet sans se partager les fichiers et galérer comme des cons.

II.1.A - Windows sans WSL

Pour installer git sous windows, il suffit d'ailleurs sur cette page puis de prendre la version `Standalone Installer`. Principalement pendant son installation il suffit de presser le bouton `suivant` en boucle.

À moins que vous vouliez des options particulières, vous avez fini l'installation basique de git.

II.1.B - Ubuntu/Debian/WSL

Beaucoup plus simple que Windows, il suffit d'effectuer cette commande dans un terminal:

```
1  sudo apt install git
```

À noté que le `sudo` n'est pas forcément obligatoire pour votre système, cela dépend énormément du setup de votre système. Fin de l'installation.

II.1.C - Fedora

Très similaire à Debian, il suffit d'effecturer cette commande dans un terminal:

```
1  sudo dnf install git
```

À noté que le `sudo` n'est pas forcément obligatoire pour votre système, cela dépend énormément du setup de votre système. Fin de l'installation.

II.1.D - Arch Linux

Toujours aussi similaire, il suffit d'effectuer cette commande dans un terminal:

```
1  sudo pacman -S git
```

À noté que le `sudo` n'est pas forcément obligatoire pour votre système, cela dépend énormément du setup de votre système. Fin de l'installation.

II.2 - L'après Git

Une fois git installé, rendez vous sur un terminal. Vous aurez besoin de spécifier votre adresse mail et votre nom (vous pouvez très bien mettre un pseudo et une fausse email si vous le souhaitez).

Les commandes sont donc:

```
1    git config --global user.email
2    git config --global user.name "<NOM>"
```

Cela vous permettra de faire des `commits`. On reviendra sur le terme plus tard.

II.3 - NodeJS

Maintenant que git est setup, nous allons avoir besoin du protagoniste de ce projet, NodeJS. C'est le logiciel qui permet de faire du javascript sans devoir lancer un navigateur. C'est la version local du JS.

Il existe deux possibilité pour installer Node, soit directement par votre package manager (ou par lien direct avec windows), soit par Volta.sh, qui install un gestionnaire de version de

node et de ses outils. Je vous conseille cette manière en général mais les deux tutos vous seront proposé.

II.3.A - Méthode une, sans Volta.sh

II.3.A.a - Windows sans WSL

Très simple (pour une fois), il suffit d'aller sur la page de NodeJS puis de cliquer la version LTS. Au moment où j'écris le tuto c'est la 20.10.0.

Ensuite exécuter l'installeur comme un installeur classique.

II.3.A.b - Debian/Ubuntu/WSL

Beaucoup plus complexe, nous allons voir comment faire.

Premièrement, lancez ces commande:

```
sudo apt-get update
sudo apt-get install -y \
ca-certificates curl gnupg
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://deb.nodesource.com/
| sudo gpg --dearmor -o \
/etc/apt/keyrings/nodesource.gpg
```

Cette commande permet de télécharger les clefs nécessaires pour pouvoir télécharger NodeJS

Les backslashs (\) sont là pour mettre sur plusieurs lignes. C'est seulement décoratif dans ce cas précis.

Ensuite, il faut executer cette commande:

```
NODE_MAJOR=20
echo "deb [signed-by=/etc/apt/keyrings/
nodesource.gpg] https://deb.nodesource.
com/node_$NODE_MAJOR.x nodistro main" |
sudo tee /etc/apt/sources.list.d/
nodesource.list
```

Cette commande permet de rajouter NodeJS à la liste des paquets téléchargeable de votre pc.

Et enfin pour finir, executez ces commandes

```
1  sudo apt-get update
2  sudo apt-get install nodejs -y
```

Cette commande parle d'elle même, elle installe nodejs.

II.3.A.c - Fedora

C'est beaucoup plus simple que Debian. Il suffit d'effectuer cette commande:

```
1  dnf module install nodejs:20/common
```

II.3.A.d - Arch Linux

NodeJS sur Arch arrive toujours avec la dernière version, ce qui rend difficile le contrôle de version. Je vais donc utiliser une autre méthode pour l'installation, veuillez donc passer à la méthode 2 avec l'installation de volta qui sera beaucoup plus intéressant que d'installer une seule version.

II.3.B - Méthode 2, Volta.sh (Recommandé)

Nous allons nous baser sur Volta.sh, un gestionnaire de version de nodejs et de ses outils. La procédure reste très simple.

II.3.B.a - Windows

Malheureusement pour les utilisateur de windows, je ne **conseille** par de faire ce tuto. Cela est légèrement plus compliqué pour vous que pour les utilisateurs sous Unix mais si vous le souhaitez, je vous met à disposition le lien direct vers la page pour installer sous windows. Il suffit de télécharger l'installer puis grâce au lien direct depuis la

page, de mettre votre pc sous le mode `Developper`. Si vous l'avez déjà fait, libre à vous d'installer volta.

II.3.B.b - Pour tous les autres

Il suffit simplement de faire cette commande pour installer volta:

```
1  curl https://get.volta.sh | bash
```

Je vous recommande de relancer votre terminal après cette commande.

Puis pour installer node:

```
1  volta install node@20 npm@latest
```

NPM est le package manager pour toutes les librairies externe de node, un peu comme `pip` pour python. Nous en auront besoin quand nous installerons le site web

II.4 - Configurer git avec github

II.4.A - Générer une clef ssh

Nous allons commencer par générer une clef SSH qui vous suivra probablement très longtemps. Dans un premier temps, ouvrez votre terminal (Git bash pour windows) et tapez

```
1  ssh-keygen -t ed25519 -C "<EMAIL>"
```

L'email sera visible pour tout le monde, choisissez bien.

Ce message apparaitra `Enter a file in which to save the key`. Appuyez juste sur la touche entrée.

Quand vous êtes devant ce message `Enter passphrase (empty for no passphrase)`, soit vous choissiez de mettre un mot de passe ce qui est très utile sur les pc qui ne sont pas à vous mais très chiant car il faut retaper le mot de passe pour chaque action avec gitub, soit vous ne mettez rien.

Votre clef SSH est maintenant générée

II.4.B - Ajouter votre clef ssh

Après l'avoir générée, il faut biensûr l'ajouter à votre gestionnaire de clef ssh

II.4.B.a - Windows

Dans un terminal **PowerShell avec les permissions élevés**, tapez:

```
Set-Service ssh-agent -StartupType Automatic
```

Puis tapez cette commande en changeant bien le nom par le votre:

```
ssh-add /c/Users/<NAME>/.ssh/id_ed25519
```

II.4.B.b - Autres systèmes

Dans un terminal, tapez:

```
1  eval "$(ssh-agent -s)"
```

Vous aurez un message qui apparaitra.

puis ensuite, tapez:

```
1  ssh-add ~/.ssh/id_ed25519
```

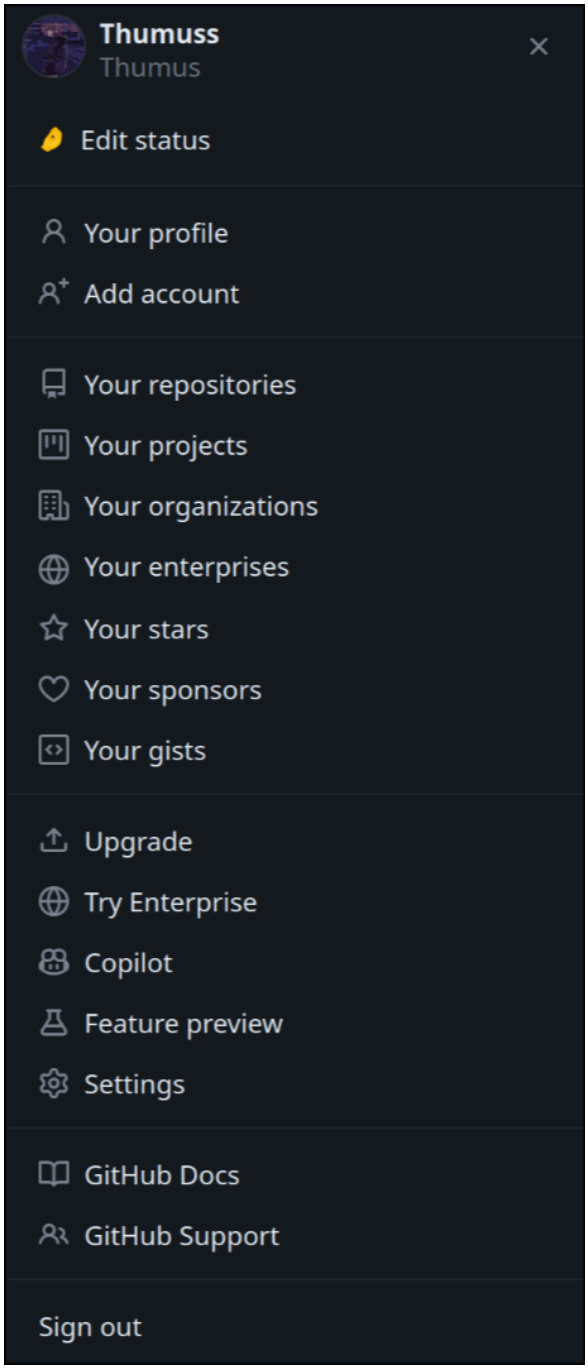
Si vous voulez que cette tâche soit automatique, ajoutez ces 2 commandes à la fin de votre fichier `~/.bashrc`.

Cela vous évitera de devoir refaire ces commandes.

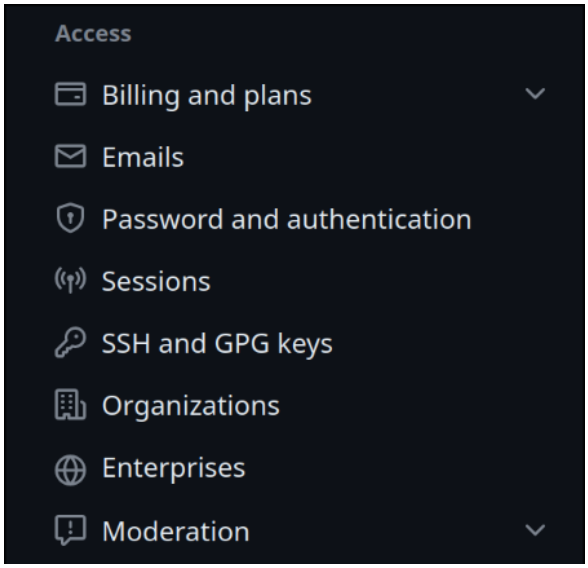
II.4.C - Ajouter votre clef ssh à github

Maitenant, allons sur github.

Quand vous êtes connecté, cliquez sur votre icone tout en haut à droite. Vous aurez un menu équivalent à ceci qui apparaitra



Cliquez sur `Settings`. À partir de là, vous aurez sur la droite un menu comme ceci:



Cliquez sur `SSH and GPG keys`.

Vous aurez un bouton `New SSH key` en haut à droite, cliquez dessus.

Appliquez un titre à votre clef ssh pour savoir sur quel pc elle est par exemple.

Windows: Ensuite, pour les pcs sous windows, il faut que vous alliez dans votre dossier utilisateur, puis que vous cherchiez votre dossier `.ssh` et que vous copiez le contenu du fichier `id_ed25519.pub`.

Autres: Pour tous les autres systèmes, il faut faire cette commande:

```
1  cat ~/.ssh/id_ed25519.pub
```

Un texte apparaitra, vous devez le copiez.

À partir de là il manque plus qu'à coller ce que vous avez copié dans le champ `Key` sur github.

Confirmer la création et ce sera parfait !

II.5 - Télécharger le projet

Maintenant que le plus gros est fait, il manque plus qu'à télécharger le projet et les librairies.

Dans un premier temps, choissiez un dossier où mettre votre projet. Un exemple, votre dossier document.

Ensuite, ouvrez un terminal dessus et tapez cette commande:

```
git clone git@github.com:ivelter/ndi2023.git
```

Cette commande va automatiquement télécharger le projet avec git initialisé. Cela vous évites de l'initialiser puis de relié le remote avec etc etc…

Il manque plus qu'à rentrer dans le projet! Il faut faire :

```
1  cd ndi2023
```

Et le tour est joué !

II.6 - Installer les libs et le projet

Si tout s'est bien passé, vous êtes actuellement dans le dossier du projet. Git étant initialisé, il manque plus qu'à installer les libs!

Dans un terminal donnant accès directement au projet, il faut faire:

```
1  npm i
```

Cela va installer toutes les libs ! Le projet est maintenant setup et vous pouvez commencer à travailler dessus !

III - Programmer le projet

Maintenant qu'on a vu comment installer le projet, il faut maintenant comprendre comment fonctionne le projet !

III.1 - Les librairies

Nous utilisons ces libraries suivantes:

Nextjs: Un vrai framework qui se base sur React et qui propose un système de route intéressante. Moins intéréssant pour les SPA (**S**ingle-**P**age **A**pplication) mais une fois que tu as compris la base, c'est simple.

Tailwindcss: Un framework CSS équivalent à bootstrap pour optimiser notre temps quand nous designerons le site. Marche principalement par class.

Sass: Sass est une version de css sous stéroides; permet d'éviter la redondance, permet de nest des classes et permet même de créer des fonctions (c'est excellent) . Nous utiliserons la version `SCSS` pour plus de simplicité.

III.2 - La structure du projet

Nous allons donc voir dans un second temps comment est structuré le projet en commençant par le haut.

III.2.A - .github/workflows/

Ces fichiers à l'intérieurs permettent d'indiquer à github de lancer des commandes directement sur github pour savoir si notre projet se lance bien, compile bien, est bien aligné avec nos principes de code…

Il y a très peu de chances que vous ayez besoin de changer quelque chose ici.

III.2.B - .next/

Ce dossier n'est pas forcément visible pour vous (pour l'instant) mais ce sont les artéfacts et le code généré par nextJS, la librairie principale pour notre site web.

III.2.C - out/

La même chose que .next mais il contiendra notre code compilé et exporté en html, css et js.

III.2.D - public/

Public est un dossier vide (pour le moment). Il contiendra tout ce qui concerne du code **static** tel que le favicon par exemple. Ce sont des données externe au site lui même mais important !

III.2.E - src/app/

Le protagoniste ! Ce dossier contient l'ensemble du code modifiable. Il permet de faire vivre le site web lui même. Il y a plusieurs fichier important à regarder à l'intérieur.

III.2.E.a - src/app/page.js

Pour qu'on puisse avoir une page qui s'affiche, nous devons toujours créer un fichier `page.js` à l'intérieur de nos dossiers.

Ces fichiers permettent de faire la page elle même. À ne pas confondre avec le `Layout`, cela permet de mettre du contenu principalement. On utilisera avec des fichiers scss (css) nommées `<NOM>.module.scss`.

III.2.E.b - src/app/layout.js

Le layout de la page, comment elle doit ressembler avec le

fichier `globals.scss` qui défini les règles de base du fichier.

Pour plus d'information sur la structure du projet lui même, visitez cette page !

III.2.F - tutos/

Le fichier source et l'output de ce tuto lui même. `config.json` permet de configurer selon vos choix (à noté que vous ne pouvez pas compiler ce tuto pour deux raisons, de 1 parce qu'il faut typst et de deux parce que la template est privée)

Le tuto en pdf se trouve dans `tutos/out/`

III.2.G - .dockerignore

Fichier de configuration pour docker; ignore les fichiers/dossiers à l'intérieur de ce fichier dans on copie le projet à l'intérieur de docker

III.2.H - .eslintrc.json

Un fichier de configuration pour `lint` notre code; permet d'avoir un code propre et suivant des conventions.

III.2.I - .gitignore

Fichier de configuration pour git; comme pour .dockerignore, git ignore les fichiers/dossiers à l'intérieur de ce fichier pour toutes les opérations qu'on effectue

III.2.J - Dockerfile

Fichier de configuration pour docker; permet de créer une image docker et d'expliquer comment devrait fonctionner notre code. Rend le partage beaucoup plus simple et consistent, plus de phrase du type “oUi MaIs Ça MaRcHe SuR mOn Pc”

III.2.K - LICENSE

Parle de lui même

III.2.L - next.config.mjs

Fichier de configuration pour next; permet la configuration du projet tel que pour l'exportation par exemple.

III.2.M - package-lock.json

Fichier généré automatiquement par npm; permet d'avoir exactement les mêmes librairies entre deux pc, ne **jamais** toucher à ce fichier.

III.2.N - package.json

Fichier de configuration de npm; permet d'indiquer les scripts et les libraries qu'on utilise. Automatise beaucoup la gestion des projets.

III.2.O - postcss.config.js

Fichier de configuration de postcss;

III.2.P - README.md

Assez évident mais l'entrée du projet, doit contenir la base pour toute personne pour lancer et utiliser le projet.

III.2.Q - tailwind.config.js

Fichier de configuration de tailwind;

Cette énumération peut faire peur (et mal à la tête) mais elle vous servira si vous êtes perdu !

Maintenant passons à comment bien utiliser git et comment build et lancer le projet!

IV - Utilisation de git

Git est un outil très puissant. Etant donné que nous seront 7 personnes à coder en même temps, il faut qu'on s'organise.

4 principes important à connaitre:

Les commits: Ce sont les points de sauvegarde de votre code, comme pour un jeu vidéo c'est important d'en faire plein avec des vrai noms pour ne pas se perdre

Les branches: C'est l'équivalent à plusieurs fichier de sauvegarde dans un jeu, vous pouvez avoir une partie en ayant fait une route génocide ou une autre partie en ayant fait une route pacifiste. Comme pour les commits, vous devez bien nommer ces branches pour pas qu'on se perdent.

Les merges: Pour le coup à la différence des jeux vidéos, on peut réunir ces fichiers de sauvegarde pour en faire une unie ayant tous nos points de sauvegardes. Cela permet qu'une fois qu'on sait que ce fichier de sauvegarde ne contient pas de bugs, de réunir le travail de tout le monde sur un endroit **stable et sûr**.

Les Pull Requests (alias PR): Pour être sûr que notre merge ne soit pas mauvais, on passe par une phase directement sur github qui permet aux gens de vérifier que notre code soit bon et en même temps cela permet à github de vérifier automatiquement que le code compile.

IV.1 - L'organisation

Nous allons donc nous organiser selon ce système:

IV.2 - Branch & checkout

Quand vous avez quelque chose à coder, genre rajouter un footer au site, il faut créer une branch comme ceci :

```
1  git branch footer-add
```

Puis ensuite il faut aller sur cette branche à l'aide de cette commande

```
1  git checkout footer-add
```

Vous pouvez ensuite coder dessus!

IV.3 - Commit

Quand vous voulez sauvegarder votre code, allez-y de façon intelligente. Ne faite pas un commit avec la moitié d'une ligne. Vous faire un commit, il faut faire :

```
git add .
git commit -m "Le message principale" \
-m "description qui n'est pas obligatoire"
```

`git add .` permet d'ajouter tout le code au commit, il faut changer le `.` par le chemin/fichier/dossier que vous voulez

Je rappelle que le backslash permet de sauter une ligne, ce n'est pas du tout obligatoire.

IV.4 - Push

Après que vous ayez fait tout vos commits, vous devez envoyer le code sur le repo du projet. Pour cela, il faut faire:

```
1  git push origin footer-add
```

Le nom de votre branch et de celle distante doit être la même ! `origin` est le nom donné automatiquement au repo distant.

IV.5 - PR

Après avoir push sur votre branch tout votre code, il faut que vous merge votre branche avec celle de `main`. Pour cela vous avez besoin d'allez directement sur la page `Pull requests` de github. Pour cette partie, on vous aidera directement le jour J.

IV.6 - Pull

Et enfin, il est possible qu'entre temps, quelqu'un ait avancé sur la branch ou merge directement sur le main. Dans ce cas là vous devez mettra à jour votre branch.

La commande est celle-ci :

```
1  git pull origin footer-add
```

Et si quelqu'un a merge:

```
1  git pull origin main
```

Vous connaissez maintenant la base pour utiliser git correctment,

V - Build le projet

Il ne manque plus qu'à voir le résultat du projet !

V.1 - Build pour tester le projet

Il suffit de faire dans le terminal du projet

```
1  npm run dev
```

Il suffit ensuite de se rendre sur la page indiquée!

V.2 - Build pour la production

La command est simple, il faut faire:

```
1  npm run lint && npm run build
```

Cela va vérifier si le projet suis bien les conventions de base puis va build le projet dans le dossier `out/` (voir au dessus)