Capitole du Libre 2026 (et un exposé sur les choix de conception de l'Internet)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 17 novembre 2026

https://www.bortzmeyer.org/capitole-du-libre-2026.html

Les 14 et 15 novembre s'est tenue à Toulouse Capitole du Libre <https://capitoledulibre.org/>, conférence consacrée aux logiciels libres mais pas que. Quelques notes suivent, entre autres sur l'exposé que j'ai fait sur les choix de conception de l'Internet (ou d'un éventuel remplaçant de ce réseau).

Capitole du Libre <https://capitoledulibre.org/>, c'est gros, avec beaucoup de monde, et un programme chargé <https://capitoledulibre.org/programme/> (112 exposés, 10 ateliers, 125 intervenants, 1[Caractère Unicode non montré [1]]400 participants, 10 kg de café, 500 repas aux camions de nourriture, 10 mètres carrés de pizzas pour les 110 bénévoles). Je ne vais donc pas parler de tout.

J'ai fait un exposé sur les choix de conception quand on veut réaliser un réseau informatique mondial (comme l'Internet). Voici les supports (en ligne sur https://www.bortzmeyer.org/files/capitoledulibre-2026-concevoir-reseau.pdf) et leur source (en ligne sur https://www.bortzmeyer.org/files/capitoledulibre-2026-concevoir-reseau.typ) (en Typst <https://www.bortzmeyer.org/typst.html>). Dès que la vidéo est publiée, je l'indiquerai ici (en attendant, vous pouvez regarder celles des éditions passées <https://videos.capitoledulibre.org/>).

Autres moments passionnants :

— Un formidable exposé de pdp_maker sur comment il a construit un PDP-11. C'est une machine qui a joué un rôle crucial dans l'histoire puisque c'est sur elle qu'on été conçus C et Unix (et c'était aussi une des machines les plus courantes d'Arpanet). Et certaines particularité du C (comme l'opérateur ++) viennent directement du PDP-11 (C n'est pas si indépendant du matériel que ça).

1. Car trop difficile à faire afficher par LaTeX

— L'exposé de Cyrielle Chatelain sur le rapport de la commission d'enquête parlementaire sur les dépendances numériques <https://www.assemblee-nationale.fr/dyn/17/organes/autres-commissions/commissions-enquete/ce-vulnerabilites-numeriques>, rapport dont elle nous a fait un excellent résumé.
— L'atelier de Guilhem Saurel sur Nix, système que je suivais de loin depuis longtemps, sans avoir jamais eu le courage de m'y mettre. Là, j'ai même modifié un paquetage Nix [Caractère Unicode non montré].
— Bertrand Boisvert s'est attelé à la tâche difficile de définir ce qu'est une IA générative libre.
— Et Jean-Michel Rogero a fait les travaux pratiques, avec des modèles (non-libres) qu'on peut faire tourner en local.

Le projet de pdp_maker était de construire un PDP-11 (ou plutôt un compatible), avec des composants modernes (il est trop trop dur et trop cher de trouver des pièces originales, les disques durs, par exemple, étaient remplacés par de la mémoire flash) mais acceptant les logiciels et périphériques (comme les cartes d'extension) du PDP 11. Le PDP est vraiment un vieux truc. Ainsi, la machine n'a pas d'horloge, le temps est mesuré par la fréquence de l'alimentation électrique (50 Hz, qu'il faut indiquer au noyau <https://mastodon.ie/@niallor/117632208132935377>). Lors de cette conférence de pdp_maker, j'ai découvert un site Web des passionnés des vieux Unix <https://www.tuhs.org/>. Je vous passe les innombrables difficultés qu'a rencontrées l'auteur. Plein de galères matérielles sont apparues, parfois intermittentes et non reproductibles. « Le debug des boots dégoute. » La température des cartes influait parfois sur leur bon fonctionnement. L'une des choses les plus dures a été de refaire la console que le PDP avait en face avant, avec ses jolies lumières.

Le premier démarrage a été fait avec le système d'exploitation RT-11 et, finalement, Unix v7 a pu démarrer sur cette machine.

La députée Cyrielle Chatelain était la rapporteure de la « Commission d'enquête sur les dépendances structurelles et les vulnérabilités systémiques dans le secteur du numérique et les risques pour l[Caractère Unicode non montré]indépendance de la France <https://www.assemblee-nationale.fr/dyn/17/organes/autres-commissions/commissions-enquete/ce-vulnerabilites-numeriques> » de 2026. Dans son exposé, consacrer à expliquer ce qu'était une commission d'enquête parlementaire et à résumer (le rapport fait 453 pages) les conclusions de celle-ci, elle a noté que, comme le savent sans doute déjà les lecteurices de ce blog, les dépendances sont fortes, et sont presque toujours vis-à-vis des USA. Et ce n'est pas une fatalité mais le résultat de choix. Par exemple, si le ministère de la justice emploie 666 ETP pour son informatique, il dépense l'équivalent de 1[Caractère Unicode non montré]500 ETP en prestations informatiques, en général vers des géants étatsuniens. Et le système des marchés publics encourage cette dépendance, puisqu'il rend presque obligatoie le passage par des intermdiaires comme l'UGAP, qui sont en pratique des revendeurs des GAFA. Plusieurs propositions de solutions ont été citées par la commission (évidemment, l'État n'en adoptera aucune) : zéro Microsoft dans les écoles, obligation de remboursement des aides en cas de vente de l'entreprise à un acteur extra-européen, actions spéciales pour l'État dans les boites comme Mistral, etc. (Elle a aussi une page Web sur ce sujet <https://www.cyriellechatelain.fr/vous-defendre/numerique/>.)

J'ai suivi l'atelier <https://cdl.stoemp.club/guix-nix-au-capitole-du-libre-2026/schedule/> Nix. Un des intérêts de Capitole du Libre est qu'il n'y a pas que des exposés magistraux mais aussi des discussions autour d'un café ou d'une bière. Non, pardon, je voulais dire, il y a aussi des ateliers pratiques, où on apporte son portable et on travaille. Bref, Nix est un système de gestion de paquetages pour installer des logiciels de manière reproductible, immuable (on ne peut pas modifier, juste réisntaller), et déclarative. On écrit ses souhaits dans des fichiers faits avec le langage Nix et c'est tout. On peut même avoir un système d'exploitation complet avec Nix et cela se nomme NixOS. Mais, pendant l'atelier, j'ai juste utilisé Nix sur une Debian classique. Utiliser Nix nécessite de changer pas mal d'habitudes, comme de passer d'un langage impératif à un langage fonctionnel, estime l'animateur de l'atelier. (Il y a aussi Guix, qui est comme Nix mais avec un autre langage, et est moins répandu.) L'un des intérêts de ces systèmes « immuables » est en sécurité : une machine piratée via un logiciel bogué ?

Le pirate ne pourra pas faire grand'chose et, même s'il le fait, on est sûr de pouvoir refaire la machine facilement et avec la garantie qu'elle sera identique.

L'atelier commence par la manipulation la plus simple : faire tourner un programme qui existe en paquetage Nix (`nixpkgs`, il s'agit ici du paquetage hello <`https://search.nixos.org/packages?channel=26.05&query=hello#show=hello`>), sur une Debian, donc :

```
% sudo apt install nix-setup-systemd

% sudo  nix run nixpkgs#hello
Hello, world!
```

(On peut faire tourner Nix sans être root mais je n'ai pas testé pendant l'atelier. Nix est très riche et je ne peux pas tout présenter.) Si la commande vous répond « *"error : experimental Nix feature 'nix-command' is disabled ; add '–extra-experimental-features nix-command' to enable it"* », le moyen le plus simple est d'éditer `/etc/nix/nix.conf` pour y mettre `experimental-features = nix-command flakes`. (Il y a d'innombrables autres paquetages <`https://search.nixos.org/packages`>.)

Si vous voulez modifier le paquetage, le plus simple est d'abord de créer un fichier `flake.nix` (comme cela, vous allez voir du langage Nix) :

```
% sudo nix flake init
wrote: "/home/stephane/tmp/nix/flake.nix"

% cat flake.nix
{
  description = "A very basic flake";

  inputs = {
    nixpkgs.url = "https://channels.nixos.org/nixpkgs-unstable/nixexprs.tar.zst";
  };

  outputs = inputs: {
    packages = builtins.mapAttrs (system: pkgs: {
      hello = pkgs.hello;

      default = inputs.self.packages.${system}.hello;
    }) inputs.nixpkgs.legacyPackages;
  };
}
```

Ce fichier déclare ce qu'il faut pour exécuter le programme (ou bien l'installer) :

```
% sudo nix run
Hello, world!
```

(Le vrai fichier décrivant ce paquetage est visible en ligne <`https://github.com/NixOS/nixpkgs/blob/nixos-26.05/pkgs/by-name/he/hello/package.nix`>.) Si on veut modifier le paquetage, alors, pour des raisons de repoductibilité et de traçabilité, il faut le faire dans le fichier Nix (on désactive les tests car le jeu de tests de GNU hello vérifie le message émis ; il serait évidemment plus propre de modifier le jeu de tests) :

```
  outputs = inputs: {
    packages = builtins.mapAttrs (system: pkgs: {
      hello = pkgs.hello.overrideAttrs {
              postPatch = ''
               substituteInPlace src/hello.c --replace "world" "capitole du libre"
              '';
              doCheck = false;
      };
```

Et le résultat :

```
% sudo nix run
warning: auto-disabling sandboxing because the prerequisite namespaces are not available and 'sandbox-fallba
Hello, capitole du libre!
```

Quand il installe un paquetage, Nix met tout sous `/nix` (au lieu des emplacements habituels), dans un répertoire nommé d'après un condensat du contenu du paquetage. Ainsi, deux versions différentes d'un même paquetage (ici, la version standard et notre modification) peuvent être présentes :

```
% ls -ltd /nix/store/*hello-2.12.3
dr-xr-xr-x 4 root root 4096 Jan  1  1970 /nix/store/lcz5s2cz1953fcyvjyhs8mm4c9h0dcpk-hello-2.12.3
dr-xr-xr-x 4 root root 4096 Jan  1  1970 /nix/store/sd779x86bjaqxj2lzh2jsir4h0zjz6d1-hello-2.12.3

% /nix/store/sd779x86bjaqxj2lzh2jsir4h0zjz6d1-hello-2.12.3/bin/hello
Hello, world!

% /nix/store/lcz5s2cz1953fcyvjyhs8mm4c9h0dcpk-hello-2.12.3/bin/hello
Hello, capitole du libre!
```

Pour que cela soit plus pratique, on utilise des liens symboliques allant de `/usr/local/bin` ou équivalent vers la version désirée. (La date au 1 janvier 1970, l'"*epoch*", est là aussi pour des raisons de reproductibilité ; on veut garantir la reproduction d'un système au bit près.)

NixOS est un système d'exploitation géré entièrement par Nix. Le système est immuable, la racine est montée en lecture seule. L'architecture typique est une machine de référence, bien sécurisée et sauvegardée, qui contrôle N machines immuables. Si vous voulez changer quelque chose, ou si une des machines a été piratée par la dernière faille de sécurité à la mode trouvée par un LLM, vous modifiez un ou plusieurs fichiers Nix sur la machine de réference et re-générez un système. On peut installer NixOS en partant de zéro ou, si vous êtes chez un hébergeur de VPS qui ne vous propose pas le choix de NixOS pour l'installation, NixOS infect `<https://github.com/elitak/nixos-infect>` (ou bien ses alternatives) permet d'installer un NixOS sur un système existant. (Lisez quand même bien la documentation `<https://nixos.wiki/wiki/NixOS_friendly_hosters>` avant, certains hébergeurs nécessitent un noyau Linux particulier.)

Passons maintenant au sujet à la mode, les IA génératives. Puisqu'on était à Capitole du Libre, il était normal de se pencher sur le fait de savoir si la définition classique du logiciel libre s'appliquait aux IA. L'exposé sur la définition de ce que peut être une « IA libre » était très intéressant, avec la définition des niveaux suivants :

— Niveau 0 : absolument pas libre du tout. C'est le cas du GPT d'OpenAI ou de Claude, qui ne peuvent être utilisés qu'en ligne, vous faisant dépendre de la bonne volonté et de la politique tarifaire du fournisseur, de sa politique de censure, et obligeant à leur confier toutes vos questions. À fuir pour tout travail sérieux.

— Niveau 1 : vous pouvez télécharger le modèle (par exemple à partir d'Hugging Face), le faire tourner en local (cf. l'exposé suivant), et même le modifier partiellement via le "*fine tuning*". C'est ce que le marketing appelle "*open weights*", voire "*open source*" quand le commercial ou le journaliste est suffisamment malhonnête et/ou ignorant. La plupart des autres modèles sont là (Mistral, Qwen, Llama, etc). C'est un peu l'équivalent du traditionnel "*freeware*".
— Niveau 2 : vous connaissez le mécanisme d'entrainement et le logiciel qui fait cet entrainement est libre.
— Niveau 3 : vous connaissez les données d'entrainement et elles sont récupérables. C'est le premier niveau où vous pouvez envisager, au moins en théorie, de re-créer le modèle vous-même, éventuellement après vos modifications. Bien sûr, c'est très théorique car il vous faudra beaucoup de GPU et d'électricité, donc beaucoup d'argent. Très peu de modèles sont à ce niveau, l'orateur citait Olmo et Apertus. Sur le fédivers, on m'a aussi cité Luciole <https://huggingface.co/collections/OpenLLM-France/luciole-llm>.
— Niveau 4 : les données d'entrainement sont libres. Aucun modèle n'est à ce niveau. Notez qu'il y a un débat ici : est-ce que le fait que les données d'entrainement soient libres est nécessaire pour que le LLM soit libre ? Le débat n'est pas réglé du point de vue juridique (je ne connais pas de décision judiciaire sur le fait que la licence des données d'entrainement influence celle du LLM), pas plus que du point de vue philosophique et politique (ce blog est sous GFDL donc sa licence - ou bien une équivalente - s'impose aux œuvres dérivées ; mais, si quelqu'un lit le blog puis écrit des articles, articles qui seront en partie influencés par ce que ce quelqu'un a lu, est-ce que la licence du blog comptera ?).

Si la question vous intéresse, vous pouvez aussi lire l'article que j'avais fait pour le Framablog <https://framablog.org/2023/07/31/que-veut-dire-libre-ou-open-source-pour-un-grand-modele-de-lan> (mais il est un peu vieux, les termes du débat ont changé).

Dans la même journée, il y avait un exposé beaucoup plus pratique, avec démonstrations en temps réel, de l'exécution de modèles en local, en général après les avoir récupéré sur Hugging Face, plutôt au format GGUF (d'autres formats sont dangereux, car contenant du code exécutable). (J'avais fait un article là-dessus <https://www.bortzmeyer.org/onprem-debut.html> il y a des années.) Selon la quantité de RAM que vous avez (64 Go dans la démo), vous pourrez faire tourner des modèles plus ou moins perfectionnés (l'indication du « nombre de paramètres » est une mesure très grossière de la « taille » d'un modèle ; OLMoE 7B <https://huggingface.co/allenai/OLMoE-1B-7B-0924> a 7 milliards de paramètres). Avec des petits modèles, saucissonnez : découpez le travail en petites tâches simples et des instructions courtes. Donnez des exemples de résultats attendus. L'orateur citait KImi K3 <https://huggingface.co/nvidia/Kimi-K3.1-Pro-NVFP4> et DeepSeek v4.1 flash <https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash-1017> comme exemple de très gros modèles (« il suffit d'avoir un centre de données et une petite centrale nucléaire ») ou bien Qwen 3.8 <https://huggingface.co/Qwen/Qwen3.8-27B> comme modèle plus « raisonnable ». La démonstration avec Qwen a d'ailleurs montré que Qwen répond désormais correctement à la question « quelle est la capitale de Taïwan ? » alors qu'avant, il disait qu'il n'y en avait pas, Taïwan n'étant « pas un pays ». Bon, au moins dans la version distribuée à l'international, la propagande chinoise s'est faite plus discrète. Ceci dit, Qwen pose certainement des problèmes de sécurité <https://www.hirundo.io/blog/qwen3-coder-vulnerabilities> mais il ne faut pas se faire d'illusions, tous les modèles incorporent une certaine vision du monde et, par exemple, les modèles étatsuniens ne sont pas plus « neutres » que les chinois.

Il existe plusieurs moteurs d'inférence pour faire tourner le modèle, comme Ollama ou llama.cpp (tous les deux sous une licence libre).

Photos et vidéos seront disponibles, les photos sur et les vidéos (via PeerTube, bien sûr) sur .

Bref, soutenez Capitole du Libre : .

Autres articles sur cette édition :
— Celui de Julien Del Rio <https://www.linkedin.com/posts/juliendelrio_capitoledulibre-ia-open>.