Utilisation d'ordinateurs en réseaux
Licence Informatique première année

Jean Méhat
jm@univ-paris8.fr
Université de Paris 8 Vincennes Saint Denis

23 janvier 2017

Table des matières

Chapitre 1

Introduction

Ceci est le chapitre d'introduction du cours de L1 de la licence informatique de l'IED intitulé *« Utilisation d'ordinateurs en réseaux »*.

1.1 Description sommaire du cours

Quand nous avons introduit le cours de première année dans la licence d'informatique (en 2015), c'était en partie pour répondre à un malentendu. D'un coté, nous avons construit la formation de licence sur notre conviction que c'est la programmation qui est à la base de notre discipline ; d'un autre coté, des étudiants arrivent chaque année dans notre formation convaincus qu'ils vont *"faire du web"*. Le cours est l'occasion de faire du web en insistant sur l'aspect programmation.

Le cours aborde de nombreux sujets distincts mais il ne fait que les effleurer. On ne peut pas traiter à fond tous les sujets nécessaires pour avoir une vision générale de l'organisation du réseau et de son utilisation, ce qui est un (autre) des objectifs du cours.

Le cours donne des repères sur la manière de concevoir et de réaliser des pages et/ou des sites pour le Web mais ce n'est pas son objet principal : il s'agit plutôt de montrer comment ça fonctionne pour qu'on comprenne ce qui se passe quand on ouvre le capot.

Cote langage de programmation, on en utilise principalement deux : PHP et Javascript. Il n'y a pas ici de cours de PHP ou de Javascript à proprement parler. J'ai traité ces sujets en sachant que les étudiants maîtrisent déjà un langage de script (en l'occurrence Python) et je n'ai donc fait que souligner ce que sont à mon sens les particularités de ces langages.

Le cours aborde aussi HTML, CSS et SQL que je rencontre souvent dans les CV sous la rubrique *langages de programmation* mais je ne pense pas que ce soient de véritables langages de programmation. (Comment réaliser ne serait-ce

qu'une récursion terminale en HTML ?)

En concevant ce support, j'ai essayé d'y introduire deux aspects des cours en présence qui me semble importants : éviter l'exhaustivité et assumer la subjectivité.

Dans un cours en présence, je ne pense pas qu'un enseignant puisse capter la pleine attention de l'étudiant pendant toute la durée de tous les cours. Dans un support pour l'enseignement a distance, les choses sont différentes parce que l'enseignant s'attend à ce que l'étudiant reprenne les choses difficiles jusqu'à les maîtriser complètement. Du coup, les études à distance sont sans doute plus difficiles que des études en présence. Si le support de cours tente de reprendre tous les sujets qui pourraient être abordés pendant un cours en présence (mais ne le seront donc pas tous, ou pas tous à fond, ou pas tous avec la pleine attention de l'étudiant), il devient trop massif pour être digestible. Pour garder le support supportable, j'ai essayé de trier les points traités. Par exemple, tous ceux qui connaissent un peu HTML penseront sans doute que j'ai passé sous silence une balise essentielle. Si j'avais traité de toutes les balises dont quelqu'un pourrait penser qu'elles sont essentielles, le support aurait été trop gros et pas bien utilisable comme support de cours mais plutôt comme manuel de référence. J'ai un peu essayé de suivre le modèle des livres de l'éditeur O'Reilly dont la première partie explique des concepts avec des exemples simples et la deuxième partie peut s'utiliser comme manuel de référence, en me limitant à la première partie : on trouve sur le web de nombreux manuels de référence, certains de qualité, sur les sujets traités ici.

Pour assumer la subjectivité, j'ai tenté dans le support d'indiquer mon opinion sur les technologies employées et précisant qu'il s'agissait d'opinions personnelles. Ce point de vue personnel qu'on partage avec les étudiants pour tenter de leur transmettre, c'est quelque chose qui se fait d'une façon naturelle dans un cours oral et qui à mon avis, y joue un rôle significatif. Dans un support, c'est plus difficile pour moi d'affirmer ces opinions, qui ne sont en général que la généralisation d'une expérience personnelle nécessairement limitée. Elles sont à prendre pour ce qu'elles sont (des opinions) ; vous avez parfaitement le droit de ne pas être d'accord.

1.2 Où trouver des informations complémentaires ?

Comme beaucoup de monde, quand je me pose une question précise, ma première démarche est d'aller la poser à un moteur de recherche. Quand la question porte sur les matières traitées dans le cours, le moteur propose souvent des réponses originaires de quelques sites, toujours les mêmes. Avoir navigué dans ces sites permet d'avoir une idée de la manière dont les sujets y sont traités ; de cette manière on peut trier les sites proposés en fonction du type de réponse dont on a besoin. Pour cette raison, je pense qu'un des aspects de la formation est d'avoir soi-même effectué suffisamment de recherches pour savoir quel genre de réponses on trouve le plus souvent sur un site donné.

On trouve sur Internet des quantités de pages et de tutoriels qui traitent à fond chacun des sujets abordés dans le cours. Attention, certains sont d'une qualité médiocre ou trop anciens pour contenir des informations utiles.

J'ai utilisé et/ou je recommande :

`https://developer.mozilla.org` est une excellente ressource pour explorer à fond un sujet. Tout (?) y est traduit en français.

`http://www.w3schools.com/` est bien à mon avis pour ses tutoriels qui couvrent légèrement toute la surface d'un sujet. En revanche ce n'est pas le bon endroit pour trouver une question traitée à fond ou trouver rapidement la réponse précise à une question précise.

Le livre *Eloquent JavaScript* de Marijn Haverbeke est excellent pour l'apprentissage du langage et de la programmation en général. La seconde édition décrit aussi l'utilisation de Javascript sur Internet. On peut l'étudier ou le télécharger depuis la page (`http://eloquentJavaScript.net/`). Au moment où j'écris, il y a des traductions de la seconde édition en Bulgare, en Portugais et en Russe mais pas en Français. La première édition est elle traduite à `http://fr.eloquentJavaScript.net/contents.html` mais n'aborde pas vraiment l'utilisation de Javascript pour le web.

1.3 Exercices et notation

Il n'y a pas d'exercice pour ce chapitre. En revanche, chaque chapitre suivant comprend un ou plusieurs exercices. Les étudiants doivent m'envoyer leurs réponses aux exercices d'un chapitre *dans un mail*, sous la forme d'un fichier attaché qui doit contenir du texte, soit sous la forme d'un PDF, soit sous la forme de HTML. (Je n'accepterai pas d'autres formats). Je répond normalement dans la semaine en vous demandant éventuellement de reprendre les aspects qui me semblent les plus importants si j'ai le sentiment que vous ne les avez pas maîtrisés.

Il faut que vous m'envoyiez vos exercices des que vous les avez réalisés. Une erreur que certains étudiants commettent parfois est d'attendre d'avoir suffisamment avancé dans le cours (selon leur estimation) pour m'envoyer plusieurs chapitres d'un coup : ce n'est pas une bonne idée : les corrigés des exercices sont utiles, à mon avis, pour répondre correctement aux exercices des chapitres suivants et j'ai besoin de suivre l'évolution de vos réponses pour mieux comprendre la manière dont vous recevez le cours. Vous devez attendre d'avoir reçu ma correction de vos exercices du chapitre avant de m'envoyer vos réponses aux exercices du chapitre suivant.

Attention, ces échanges imposent une limite à la durée minimale pour aller jusqu'au bout du cours. Même si vous dominez complètement le sujet, avec une demi-douzaine de chapitres et un délai d'une semaine entre les envois, il faut un mois et demi ; s'il y a des choses à reprendre, si je répond plus lentement (c'est souvent le cas pendant le mois d'août), la durée minimale augmente. Pour cette

raison, il y a une date limite pour commencer le cours.

Les corrigés que je vous envoie contiennent une appréciation à quatre niveaux principaux : *inacceptable et donc à reprendre*, *acceptable mais améliorable*, *correct* et *bien* (que j'abrège parfois avec *non*, *ok−*, *ok* et *ok+*).

La dernière appréciation que j'ai portée sur les exercices d'un chapitre rentre dans le calcul de la note finale, en combinaison avec la qualité de nos échanges. À titre d'information : l'an dernier, il y avait quatre points par chapitre (non = 0, ok− = 1, ok = 2, ok+ = 3) plus quatre points en fonction de mon ressenti de la qualité de nos échanges (tout particulièrement la prise en compte de mes remarques dans les corrections). Ça a donné une note (sur 22) que j'ai ramené à une note sur 20.

La chose importante pour vous est que ce n'est pas un problème d'envoyer une première version qui contient des erreurs si (1) ce sont des erreurs nouvelles, que vous n'aviez pas commises dans les chapitres précédents et (2) vous envoyez un corrigé quand c'est demandé et ne commettez plus cette erreur ni dans le corrigé, ni par la suite. Évidemment, c'est mieux pour vous comme pour moi si c'est bon du premier coup.

Comme dans tous les travaux universitaires, vous pouvez utiliser toute les sources d'information qui vous semblent utiles ; il est absolument indispensable d'indiquer clairement que vous avez utilisé ces sources, avec une citation suffisamment précise pour permettre au lecteur (moi en l'occurrence) de la retrouver facilement. Pour une page web, cela signifie indiquer son URL complet (ce qui apparaît dans la barre de navigation du navigateur) et pas simplement le nom du site web.

1.4 L'organisation de la suite du cours

Les chapitres 2 et 3 décrivent sommairement le HTML, pour structurer une page, et les CSS, pour contrôler la manière dont les structures sont affichées.

Le chapitre 4 présente le protocole HTTP et le fonctionnement d'un serveur Web. Dans le chapitre 5, je montre l'utilisation des formulaires du HTML, avec un peu de PHP pour utiliser les données envoyées et un peu de SQL pour les stocker.

Le chapitre 6 survole le langage Javascript, qui s'utilise principalement à l'heure actuelle pour rendre les pages Web réactives.

Le chapitre 7 parle aussi d'ordinateurs en réseaux mais pas par le biais du Web ; il contient une description sommaire de ssh (qui sert à accéder à un ordinateur à travers le réseau), du NAT, des systèmes de stockage de fichiers sur le réseau et de la façon dont les échanges de mails sont réalisés.

1.5 Remerciements

Corrections, suggestions d'améliorations de Adrien Revault d'Allones, Amine Chouikh, Farez Belhadj. François-Xavier Talgorn, Jean-Noël Vittaut, Marie-Solange Touzeau, Nicolas Jouandeau, Salah Bakir. Qu'ils en soient remerciés.

Je dois également exprimer mes remerciements aux étudiants de la licence informatique à l'IED qui ont, de gré ou de force, servi de cobayes pour l'élaboration du cours, particulièrement pendant la première année. Merci donc à Morgan Balderacchi, Ayala Boukris, Mathieu Fouquet, Yolaine Giovannini, Julien Lagarrigue, Rémi Montussac, Merwan Ropers, Manuel Touchefeu et Raphael Valentin qui ont essuyé les plâtres mais sont parvenus au bout du cours malgré ses déficiences.

Chapitre 2

Hyper-Text Markup Language : HTML

Ce chapitre présente HTML, une des briques sur lesquelles Internet s'est bâti. Il s'agit d'un langage de balisage conçu initialement pour marquer la *structure* d'un texte.

Le chapitre commence par expliquer l'origine et l'originalité des langages de balisages ; je montre ensuite un exemple élémentaire de page web (construite avec HTML) puis je présente ce qui me semble essentiel du HTML.

À la fin du chapitre, vous devriez être en mesure de concevoir une page HTML ordinaire et de lire et comprendre à peu près n'importe quelle page HTML.

2.1 Balises et mise en page

Une façon d'indiquer la structure d'un texte consiste à y placer des *balises* qui en définissent les éléments comme les phrases, les paragraphes, les chapitres etc. ou des indications typographiques (composer cette partie en italique par exemple).

Ceci s'oppose à ce qu'on trouve dans un éditeur de textes comme celui de Libre Office : là on est encouragé à décrire la façon dont on souhaite que le texte final apparaisse, en choisissant la police de caractère ou la taille de la fonte. (On les appelle couramment des éditeurs WYSIWYG, à prononcer plus ou moins comme *ouizi ouigue*, les initiales de *What You See Is What You Get*).

Les langages de balisage mélangent en général la structure du texte et des informations sur sa mise en page. Pour le HTML présenté dans ce chapitre, cependant, on est encouragé à déléguer la mise en pages aux *feuilles de styles en cascades* (*CSS : Cascaded Style Sheet*) présentées dans le chapitre suivant et

à se limiter à décrire la structure.

2.1.1 Des formateurs de texte

Avant la généralisation, dans les années 1980, des écrans graphiques qui ont permis d'afficher une approximation raisonnable de l'apparence de la feuille imprimée, le marquage de textes a été l'outil principal pour décrire les textes à imprimer depuis le début de l'utilisation des ordinateurs pour la mise en page

Le premier formateur fut `RUNOFF` (vers 1964) et d'autres programmes qui s'en inspirent (dont `nroff` et `troff` vers 1975 sous Unix ; ils sont encore utilisés pour les pages du manuel).

Un autre formateur important est TEX, écrit par D. E. Knuth ; TEX est célèbre pour la qualité de sa mise en page et tout particulièrement pour son rendu des équations mathématiques. Il est encore largement utilisé (particulièrement avec LATEX, qui rend son utilisation plus aisée) dans le monde des sciences exactes et notamment en informatique. Ce support est rédigé en utilisant LaTeX.

2.1.2 Des langages de balisage

Un langage de balisage est normalement utilisé purement pour décrire la structure d'un texte. (Cependant, la structure d'un texte peut désigner différentes choses : un nom, un prix unitaire, une quantité dans une ligne d'un devis par exemple aussi bien que le groupe nominal sujet le groupe verbal dans une phrase, par exemple.)

Le langage GML (comme *Generalized Markup Language*) est l'ancêtre des langages de balisages ; il en découle un langage standardisé SGML, avec deux descendants importants pour l'informatique : HTML et XML. Ces langages SGML, HTML et XML sont présentés dans les sections qui suivent. Les choses à retenir : HTML est beaucoup plus simple que SGML ; XML est beaucoup plus général que HTML ; il existe un intermédiaire entre eux nommé XHTML que j'évite quand je le peux.

Le langage SGML

Pour faciliter le traitement automatique des textes en langage naturel, des chercheurs ont défini dans les années 1970 un langage de balisage nommé SGML (comme *Standardized General Markup Language* : *langage de balisage général standardisé*). Le principe est de mélanger le texte avec des indications de structure ; ces indications sont appelées des *balises* en français.

Les balises de SGML sont de la forme `<struct>` au début d'un élément de la structure et `</struct>` à la fin, sauf exceptions. Ces balises fonctionnent comme des parenthèses ouvrantes et fermantes.

SGML permet d'étiqueter un texte, par exemple en indiquant les chapitres, les sections, les paragraphes etc. aussi bien que d'en identifier les structures grammaticales comme le verbe, le sujet etc.

Un document balisé avec SGML est souvent accompagné d'un DTD (comme *Document Type Definition*) qui décrit les balises employées et leurs caractéristiques.

SGML est standardisé depuis 1986. Ce standard est *très* complexe et de ce fait peu SGML est peu aisé à manipuler.

HTML

L'outil principal pour décrire des pages du Web est un langage de balisage qui s'appelle `HTML`. On peut le voir comme un sous-ensemble très restreint de `SGML` avec une caractéristique supplémentaire : la possibilité d'inclure des *liens* vers un autre document.

Les liens permettent de transformer un texte en *hyper-texte* (c'est ce que signifie `HTML`, pour *Hyper-Text Markup Language*). Un *hyper-texte* est un document qui n'est pas conçu pour être lu en séquence mais dans lequel le lecteur peut choisir son cheminement (comme dans un dictionnaire, la *Petite flore* de Gaston Bonnier ou *Un livre dont vous êtes le héros*).

La suite du chapitre traite principalement de HTML.

XML

`XML` est un autre dérivé de *SGML*, utilisé dans une version largement simplifiée, avec lequel on peut en principe représenter n'importe quelles données de façon portable. `XML` est l'acronyme de *eXtensible Markup Language*.

`XML` est une des façons courantes de représenter des données portables sur Internet ; une autre méthode est `JSON` présenté dans un autre chapitre. `JSON` est souvent préférable à `XML`, à mon avis, parce qu'il est plus simple à lire et à écrire, aussi bien pour une personne que pour un programme.

2.1.3 Une brève histoire de HTML

HTML a été conçu dans les années 1990 comme un moyen de partager des documents au CERN (le Centre Européen de Recherche Nucléaire) par Tim Berners-Lee. Le langage s'est répandu à travers Internet. (Il avait à l'origine des concurrents, par exemple Gopher de l'université du Minnesota mais il s'est imposé, sans doute grâce à son mélange de liens et de contenus et à sa facilité d'associer images et textes.)

La première version normalisée a été la version 2 en 1995 ; la version 4.1 a elle été normalisée en 1999 ; la version courante (en 2016) est la version 5, normalisée en 2014.

Au départ, les versions de HTML avaient des durées de vie assez courtes. Maintenant, la définition précise de HTML représente des enjeux industriels considérables et l'évolution est beaucoup plus lente. (Imaginez ce que coûterait la modification de toutes les pages Web qui existent à l'heure actuelle du fait d'une incompatibilité d'une nouvelle version de HTML !) C'est donc un standard stable.

2.2 Un exemple élémentaire

Exemple de départ : une page simple avec quelques titres de sections et quelques paragraphes.

2.2.1 Le source HTML

Le source HTML est placé dans un fichier texte ordinaire (le genre de fichier dont on peut consulter le contenu dans un terminal avec la commande `more`). Ici, le fichier s'appelle `fichier1.html` et voici son contenu :

```
<!DOCTYPE html>
<html>
  <!-- Commentaire -->
  <head>
    <title>Exemple</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Espaces et retours à la ligne</h1>
    <p>Si on saute plusieurs espaces comme ici :     (cinq
      espaces avant la parentèse), c'est affiché comme
      un seul espace.</p>
    <p>On remarque sans peine que les retours à la ligne
      du fichier texte sont traités de la même manière que
      des espaces.</p>
    <h1>L'apparence et le contenu</h1>
    <p>Quand on veut avoir du texte juste pour savoir
      à quoi ressemblerait l'affichage, on emploie
      fréquemment un texte en pseudo-latin qui commence
      par <em>lorem ipsum</em>.</p>
    <p>Un maquettiste qui veut juger de la qualité d'une
      mise en page peut aussi retourner la page : à
      l'envers, le contenu influe peu sur l'appréciation
      qu'on porte sur le document.</p>
  </body>
</html>
```

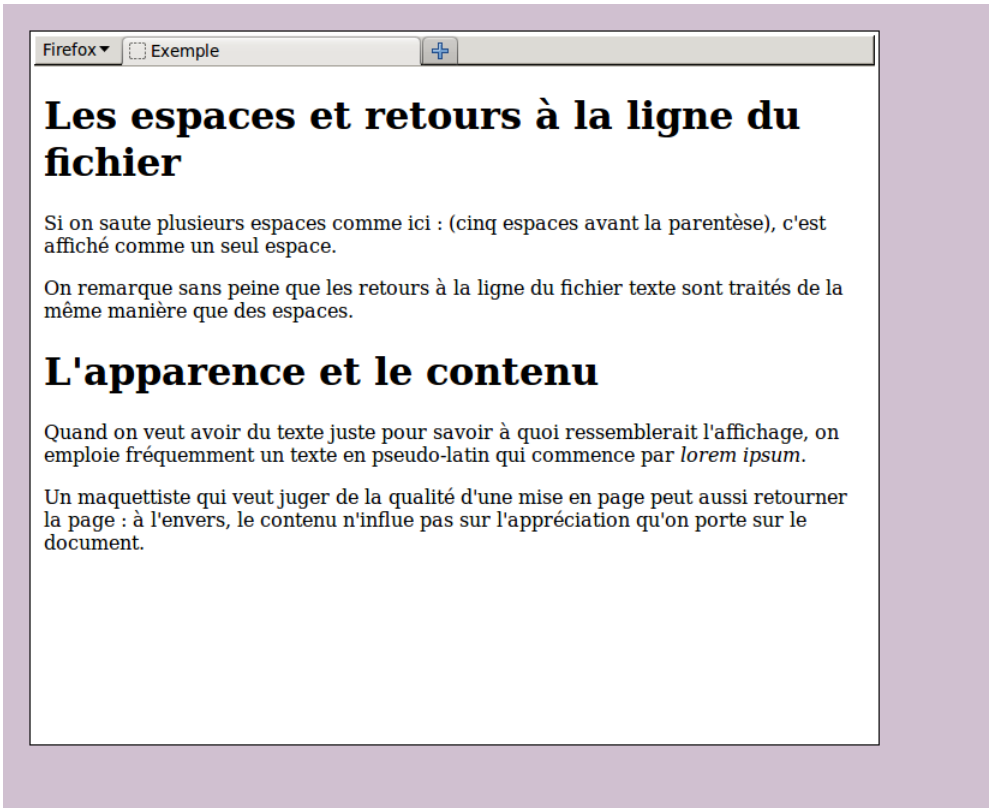


FIGURE 2.1 – Le rendu par le navigateur Firefox du premier exemple élémentaire.

2.2.2 Le rendu de la page

Le fichier porte le nom `fichier1.html` ; on demande son affichage à Firefox avec la ligne de commande `firefox fichier1.html &` ; on devrait voir quelque chose comme la figure 2.1 : le texte contenu dans le fichier est affiché, avec une mise en page qui dépend de la structure (indiquée par les balises présentées dans la section suivante). Certaines parties sont en caractères gras, d'autres en italique etc. sans que nous l'ayons spécifié.

2.2.3 Les balises et la structure d'arbre

Cet exemple élémentaire utilise les balises `HTML` : `head`, `title`, `meta`, `body`, `h1`, `p` et `em`. Elles servent à marquer le rôle de parties du fichier.

Le début d'une zone balisée est marqué par <*qqchose*> et la fin par </*qqchose*>. Ces balises fonctionnent comme des parenthèses ouvrantes et fermantes ; elles

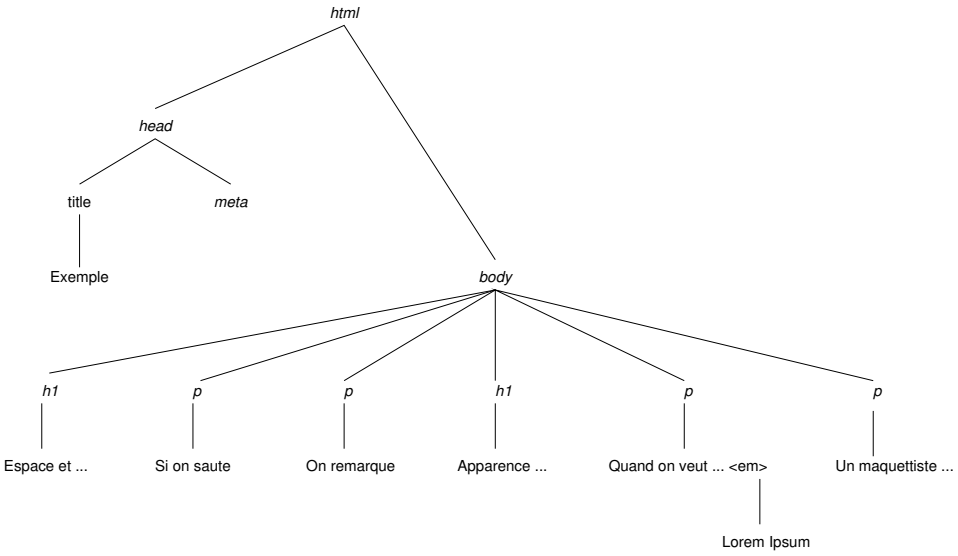


FIGURE 2.2 – Les balises HTML décrivent l'organisation du contenu de `fichier1.html` comme un arbre. Le contenu des éléments est seulement évoqué par ses premiers mots.)

décrivent un arbre, évoqué dans la figure 2.2.

Certaines balises n'ont pas de contenu du tout, comme ici la balise utilisée pour un commentaire `<!-- Commentaire -->`. Une autre balise qui s'utilise sans fermante est `<br>` qui sert à forcer un retour à la ligne (on ne l'utilise pas ici).

2.2.4 Organisation de la page et contenu

Au tout début du fichier, on trouve quelque chose qui indique son contenu : la phrase `<!DOCTYPE html>`. Cette phrase *n'est pas* du HTML mais annonce que ce qui suit en revanche doit être lu comme du HTML. Effectivement, juste après on trouve la balise ouvrante `<html>` et la fermante `</html>` à la fin du fichier : tout ce qui est entre les deux doit s'interpréter comme du HTML.

Le contenu du HTML est à son tour découpé en deux parties : d'une part du matériel d'en-tête (dans `<head>`) qui donne des indications générales sur le contenu du fichier et d'autre part ce qu'il faut afficher dans la page (dans `<body>`).

Dans l'en-tête, on trouve le nom sous lequel la page apparaîtra dans un onglet ou dans une icône, indiqué avec `<title>` ainsi que la manière dont y sont encodés les caractères accentués avec le `<meta charset="utf-8">`.

On pourrait trouver d'autres informations dans l'en-tête ; c'est notamment

un bon endroit pour placer des indications sur le style avec lequel afficher la page, avec les feuilles de styles décrites dans le chapitre suivant.

La suite du fichier est regroupée dans le `<body>`, c'est à dire le corps du fichier. Il contient ici une succession d'en-têtes de sections (indiqués avec la balise `<h1>`) et de paragraphes (indiqués avec la balise `<p>`. Un des paragraphes contient une partie de texte mis en valeur comme indiqué avec une balise `<em>`.

À RETENIR : le squelette d'une page HTML est donc

```
<!DOCTYPE html>
<html>
  <head>
    <title>TITRE</title>
    <meta charset="utf-8">
  </head>
  <body>
    LE CONTENU ICI
  </body>
</html>
```

2.3 Un peu plus de détails sur HTML

Dans cette section, je rentre un peu plus en détail dans la manière d'utiliser HTML : comment ça se passe avec les accents, comment ajouter des attributs aux balises, comment insérer des images dans le texte, comment ajouter des liens vers d'autres documents et finalement les balises élémentaires pour structurer les documents.

2.3.1 Les accents, les entités HTML

Il y a plusieurs systèmes concurrents d'affichage des caractères qui n'appartiennent pas à l'ASCII. Il faut utiliser le bon système pour que l'affichage des caractères accentués puisse se faire correctement. (C'est pareil en pire pour les langues qui n'utilisent pas notre alphabet romain.)

On vient de voir dans le premier exemple qu'on peut indiquer comment les accents sont codés dans le texte avec l'attribut `charset=` de la balise `<meta>`.

Une autre méthode consiste à indiquer le codage à employer pour la page avec *Unicode* ou *Western* par le menu qui sort en dessous de *View* > *Character Encoding* sous Firefox. Un inconvénient est que ça ne concerne que la session en cours. Chaque fois qu'on reviendra sur la page, il faudra répéter l'opération pour indiquer de nouveau au navigateur le codage à utiliser.

Finalement, une troisième manière consiste à employer les *entités* de HTML : une séquence de caractères (ASCII) qui indique le nom ou le code d'un autre caractère. Les entités commencent par une esperluette (`&`), se continuent soit

&	&	Et commercial (esperluette, ampersand)
>	>	Chevron fermant (supérieur, greater than)
<	<	Chevron ouvrant (inférieur, less than)
		Espace sans retour à la ligne (non breakable space)

FIGURE 2.3 – Les quatre *entités* (noms de caractères) HTML à connaître par cœur.

par un dièse plus un code, soit par un nom d'entité. Elles se terminent par un point virgule. Par exemple, le mot `élève` peut s'écrire `élève` en HTML.

Il existe une description HTML pour la plupart des caractères ; c'est facile d'en trouver des listes plus ou moins complètes sur Internet. Il faut en connaître quatre par cœur : le codage des chevrons ouvrants et fermants `&lt` et `&gt` (ils seraient interprétés comme des bouts de balises s'ils apparaissaient dans le texte), celui de l'esperluette `&` (elle serait interprété comme le début du codage d'une entité si elle aparaissait dans le texte) et celui de l'espace sans retour à la ligne ` ` qu'on peut utiliser en typographie française comme une espace fine devant les symboles de ponctuation en deux parties. (Ces noms proviennent de l'anglais : *Less Than* pour `lt`, *Greater Than* pour `gt`, *AMPersand* pour `amp` et *Non Breakable SPace* pour `nbsp`.)

2.3.2 Les attributs des balises

En plus de leurs noms, on peut ajouter des attributs aux balises, en général sous la forme `attribut=valeur`. Il faut parfois placer la valeur entre des guillemets (américains) ; comme c'est toujours possible, le plus simple est de les mettre systématiquement. Une balise avec un attribut va donc ressembler à quelque chose comme `<balise attribut="valeur">`

Les attributs jouent un peu le rôle des paramètres dans un appel de fonction.

On a utilisé l'attribut `charset=` de la balise `<meta>` avec la valeur `"utf-8"`.

Il existe une liste d'attributs qu'on peut utiliser avec à peu près toutes les balises. Les plus importants à mon sens sont `class=`, `id=` et `title=`.

L'attribut `class=` permet de diviser les éléments en différentes classes ; par exemple on peut décrire des paragraphes obligatoires ou facultatifs avec des balises+attributs `<p class="facultatif">` et `<p class="obligatoire">`. On verra dans le chapitre suivant que les CSS permettent d'afficher différemment des éléments du même type qui n'appartiennent pas à la même classe.

L'attribut `id=` permet de donner un nom (unique dans la page) à une instance d'un élément, ce qui facilitera le travail quand on voudra le retrouver (notamment pour le faire modifier par un programme).

L'attribut `title=` permet de donner des informations supplémentaires sur

un élément. C'est par exemple ce qui s'affiche quand on laisse la souris stationner sur cet élément. (Ne pas confondre la *balise* `<title>` qui est grosso-modo le nom à donner à la fenêtre avec les attributs `title=` qu'on peut mettre dans les autres balises et qui donne une sorte de nom à la structure.)

2.3.3 Les images

Un document décrit avec HTML peut contenir des images. Le plus courant est que l'image soit stockée dans un fichier séparé. Parmi les nombreux formats d'images, préférer le PNG en général et le JPEG pour les photos ; éviter le GIF sauf pour les images animées (mais on me conseille d'éviter les images animées...)

On indique la présence d'une image avec une balise `img` et usuellement au moins l'attribut `src` qui indique l'URL [1] qui contient l'image. Avec les images, on ajoute toujours un attribut `alt` qui contient une représentation alternative de l'image utilisée par les navigateurs qui n'affichent pas les images (par exemple pour les malvoyants). Un exemple pour une image de chien dans le fichier `chien.png` : `<img src="chien.png" alt="Une image de chien">`

L'attribut `title=` évoqué plus haut fonctionne comme sur n'importe quel élément.

De nombreux attributs permettent d'affiner la manière dont l'image doit être affichée (taille, alignement, bords etc.) mais il est préférable de s'en passer : ils décrivent la manière de mettre en page et donc il vaut mieux passer par les CSS présentés dans le chapitre suivant pour préciser ces points.

2.3.4 L'*Hyper*-texte : les ancres, les liens et les URLs

Le H et le premier T de HTTP sont les initiales de *Hyper-Texte*. La caractéristique d'un *Hyper-Texte* est d'être conçu pour être lu d'une façon non-linéaire avec des *liens* qui permettent de sauter d'une partie à l'autre du texte [2], Presque toutes les pages contiennent des liens, qui sont la façon la plus commune de naviguer sur le Web ; on peut voir tout le web comme un seul (immense) hyper-texte.

La balise `<a>` sert à placer des liens avec l'attribut `href=` qui indique où va le lien, dans un format qu'on appelle un URL.

Les URLs

Un URL (comme *Universal Resource Locator* est une chaîne de caractères qui sert à désigner à peu près n'importe quoi sur le Web. La forme la plus courante ressemble à `http://ietf.org/rfc.html`.

1. Ce qu'est un URL est détaillé un peu plus loin ; pour le moment on peut considérer que c'est juste le nom du fichier.
2. Le concept d'hyper-texte avait été popularisé dans les années 1980 par un système d'organisation des données sous cette forme nommé *Hypercard*.

Le `http:` est le nom de la méthode (du *protocole*) utilisé pour récupérer le contenu. Le protocole `http` est le principal utilisé sur le web (avec `https` qui est le même en sécurisé : les échanges sont cryptés pour empêcher l'interception du contenu des messages).

Le `//ietf.org` donne le nom d'un ordinateur sur le réseau Internet : celui de la machine qui contient la ressource.

Le `/rfc.html` correspond en pratique à un nom de fichier sur cette machine.

Les URLs peuvent être beaucoup plus complets ou plus courts.

Pour avoir quelque chose de complet, on peut :

— utiliser un autre protocole que `http` ou `https` ; le protocole `ftp` est aussi fréquemment utilisé et on peut spécifier un fichier sur le disque local avec le protocole `file:`.

— ajouter un nom d'utilisateur et un mot de passe, pour les protocoles comme `ftp` qui en ont besoin.

— ajouter un numéro de port. Le port par défaut pour `http` est le port numéro 80 mais on rencontre souvent aussi des serveurs expérimentaux sur les ports 8000 et 8080.

— ajouter des paramètres à la requête de page avec un point d'interrogation suivi des paramètres.

— spécifier un endroit dans la page HTML avec un dièse suivi de de l'identité d'un élément (spécifiée par son attribut `id=`).

Un URL complet serait donc quelque chose comme
`http://user:password@www.domaine.com:80/le/chemin#lapartie?params`.

Pour avoir quelque chose de plus court, on peut

— se passer du protocole : dans ce cas le navigateur utilisera le même protocole que celui de la page qui contient la référence (en général `http` ou `file`).

— se passer du nom de machine : dans ce cas, la référence sera sur la même machine que celle qui contient la référence.

— se passer du chemin d'accès à la ressource, en ne spécifiant qu'un nom *relatif* au lieu du nom *absolu* ; dans ce cas la ressource devra être dans le même répertoire que la page qui contient la référence.

Ainsi, dans le fichier `recto.html`, il vaut mieux mettre une référence vers un fichier voisin avec `<a href="verso.html">` ; de cette manière, on pourra suivre le lien aussi bien si on accède à `file:///tmp/recto.html` qu'à `http://www.example.com/exemple/recto.html`.

Les liens

Dans la page HTML, on spécifie le lien avec l'attribut `href=` de la balise `<a>`. Un exemple :

```
<a href="http://ietf.org/rfc.html">cliquer ici</a>
```

Le texte encadré par les `<a>` et `</a>` est composé d'une façon spéciale (en bleu et souligné) pour indiquer au lecteur qu'il s'agit d'un lien sur lequel il peut cliquer.

La balise `a` accepte aussi un attribut `target=` avec lequel on peut indiquer que la nouvelle page est à ouvrir dans une autre fenêtre ou dans un autre onglet.

2.3.5 Pour structurer un document

Il y a diverses balises pour structurer un document ; certaines sont incontournables et d'autres sont à éviter. On les regroupe usuellement entre celles qui forcent un retour à la ligne (les balises *à blocs*) et celles qui peuvent désigner un élément qui ne comprend qu'une partie d'une ligne (les balises *en ligne*).

Les balises à blocs

Certaines balises forcent un retour à la ligne. Ce sont les balises *à blocs*. Les plus importantes à mon avis sont `<p>`, `<h`*n*`>` et `<div>`. Je mentionne aussi `<pre>` pour mémoire.

Les balises `<h1>`, `<h2>`, ... `<h6>` délimitent des titres de parties de différents niveaux (avec les balises fermantes `</h`*n*`>` correspondantes). Par défaut elles sont rendues par du texte composé en caractères gras, avec un corps de police de plus en plus petit à mesure qu'on descend dans les niveaux les plus profonds.

La balise `<p>` sert à délimiter un paragraphe. Par défaut, elle est rendue sur mon navigateur par un saut d'une ligne, sans indentation.

Certains omettent la balise fermante `</p>` et font confiance au navigateur pour comprendre que le début d'un paragraphe est nécessairement précédé de la fin du paragraphe précédent ; je l'omettais moi-même il y a quelques années. Avec HTML 5, présenté plus loin, on peut imbriquer des équivalents de paragraphes : maintenant je place systématiquement ces balises en paires `<p>`$\cdots$`</p>`.

On dispose aussi d'une balise en blocs `<div>` qui permet de regrouper n'importe quels éléments. En effet, on peut parfaitement concevoir une page web qui ne contient que du texte organisé avec les balises `<h`*n*`>` et `<p>` mais le navigateur ne peut pas en déduire la relation qui relie les titres de paragraphes et les paragraphes eux-mêmes (figure 2.4).

Avec la balise `<pre>` (comme dans *préformatté*), on indique du texte qu'on souhaite que le navigateur affiche en respectant les espaces et les retours à la ligne. On l'utilise en général pour les programmes ou les commandes. La balise fait également passer en police à chasse fixe (tous les caractères ont la même largeur).

Les balises en lignes

Un autre classe de balises permet de délimiter des éléments sans forcer un retour à la ligne. Ce sont des balises *en ligne*. Les plus importantes à mon avis

(a)
```
<h1 id=hx>
<p id=pa>
<p id=pb>
<h1 id=hy>
<p id=pc>
<p id=pd>
```

(b)
```
<h1 id=hx>
     <p id=pa>
<p id=pb>
<h1 id=hy>
     <p id=pc>
<p id=pd>
```

(c)
```
<h1 id=hx>
     <p id=pa>
     <p id=pb>
<h1 id=hy>
     <p id=pc>
     <p id=pd>
```

FIGURE 2.4 – L'exemple de `fichier1.html` contient deux en-têtes de sections et quatre paragraphes qui se suivent sans indications supplémentaires (a). Cela ne permet pas de choisir entre les interprétations (b) et (c). En (b), l'en-tête `hx` s'applique au paragraphe `pa` alors que le paragraphe `pb` n'a pas d'en-tête (et idem pour `hy`, `pc` et `pd`). En (c), l'en-tête `hx` s'applique à la fois aux paragraphes `pa` et `pb`. La balise `<div>` permet de regrouper l'en-tête de paragraphe avec les paragraphes auxquels elle s'applique.

sont `<em>`, `<strong>` et `<span>`. Il faut aussi mentionner `<code>`.

La balise `<em>` sert à mettre du texte en valeur (*em* comme *emphasize* en anglais) sans introduire de rupture dans le gris typographique de la page. Ça se traduit normalement par du passage en italique. On l'utilise par exemple pour les citations ou pour traduire dans un texte mis en page ce qu'on aurait souligné dans un manuscrit.

La balise `<strong>` sert à attirer l'oeil sur une partie du texte. Ça se traduit normalement par un passage à une police grasse (avec des caractères dessinés avec un trait plus épais). Ça perturbe la lecture et donc on ne l'utilise traditionnellement que pour des titres.

Avec la balise `<code>`, on indique ce qui doit donner l'impression d'avoir été imprimé avec un ordinateur. Ça fait normalement passer en police à chasse fixe.

Il y a une balise en ligne générique pour définir des éléments. Elle s'appelle `<span>`. On peut l'utiliser pour construire des éléments, dont on précisera la nature avec l'attribut `class=` (ou l'identité avec l'attribut `id=`).

À retirer ou à expliciter

`<del>` et `<ins>` pour du texte détruit ou ajouté (inséré) ; normalement ça donne du texte barré ou souligné.

`<sub>` et `<sup>` pour les exposants et les indices.

Le reste

Ici, les choses importantes qui ne rentrent pas dans les classes ci-dessus.

Les tables : on les encadre avec `<table>`. Les tables sont constituées de lignes délimitées avec `<tr>` (*Table Row*) qui contiennent des cases délimitées avec `<td>`.

Les listes : on les encadre avec `<ul>` (comme *Unordered List* : des listes à puces) ou avec `<ol>` (comme *Ordered List* : les éléments sont numérotés). Les éléments des listes sont eux-mêmes encadrés avec des `<li>` (comme *List Item*). On peut imbriquer les listes.

Les listes de définitions délimitées avec `<dl>` ; `<dt>`, `<dd>` pour marquer le terme défini et sa définition. Ça donne en pratique à peu près la même chose qu'un simple tableau à deux colonnes.

`<hr>` comme *Horizontal Rule* pour un filet qui découpe la page. Est-il important ? Est-ce de la typographie ou de la structure ?

2.4 À éviter

Il y a des façons d'écrire du HTML qui se sont imposées à certaines époques et qu'on recommande maintenant d'éviter.

La balise `<br>` sert à forcer un retour à la ligne. Ne l'utilisez pas pour définir une sorte de sous-paragraphe (mais seulement pour éviter des problèmes avec les coupures de mots ; `<wbr>` peut aider aussi).

Les balises `<i>` et `<b>` ont longtemps servi pour obtenir de l'italique et du gras mais ne disent rien sur la structure du texte. Préférer `<em>` et `<strong>` présentés plus haut.

La balise `<font>` a servi autrefois à contrôler finement les polices de caractères utilisées, leurs tailles et leurs graisses. Elle a conduit à un chaos total parce que les polices du concepteur de la page ne sont pas toujours les mêmes que celles du navigateur qui l'affiche. Ne jamais l'utiliser ; se servir des CSS à la place. On peut à la limite utiliser `<smaller>` et `<larger>` pour changer de corps.

Les frames sont à éviter à tout prix.

À une époque, on a utilisé les tableaux pour forcer l'organisation des éléments dans la page. C'est passé de mode. Utiliser plutôt les *float*, *left*, *right* et *clear* des CSS.

2.5 HTML 5

La norme HTML 5 de 2014 a fixé toute une ménagerie de types de paragraphes de manière à permettre de décrire d'une façon commune les classes de paragraphes que chacun définissait auparavant pour lui-même dans ses définitions de style. On y trouve par exemple

— `<nav>` pour une barre de navigation, c.a.d. un paragraphe qui ne contient que des liens.
— `<section>` pour une section principale.
— `<article>` pour un article indépendant

— `<header>` et `<footer>` pour l'en-tête et le pied de page (de la page web ou de l'article, de la section).
— `<aside>` pour une information complémentaire indépendante du sujet principal.

La façon dont ces éléments s'imbriquent n'est pas claire : une section peut contenir des articles mais un article peut aussi contenir des sections par exemple.

La balise `<figcaption>` sert à indiquer la légende d'une figure délimitée avec `<figure>`. La figure peut contenir à peu près n'importe quoi (une ou plusieurs images, du texte, un tableau).

Les balises `<video>` et `<audio>` permettent de brancher sur des fichiers vidéos et audio (plutôt dans le format `ogg`) à la façon dont la balise `img` permet d'inclure des images. À la différence de cette dernière, les balises fermantes `</audio>` et `</video>` permettent d'encadrer du HTML qui sera utilisé si le navigateur de peut pas y accéder (ça joue le rôle du `alt=` de `<img>`).

```
<p>En français <abbr title=Monsieur>M.</abbr> est l'abbréviation
correcte de « Monsieur » ; <abbr title=Mister>Mr.</abbr> est un
anglicisme.</p>
```

2.6 Validation des pages

`http://validator.w3.org`

`http://tidy.sourceforge.net/`

`Firefox > Web Developer > Debugger`

2.7 Ce qui manque

Il manque énormément de détails ici, notamment sur les attributs qui peuvent (ou pas) accompagner les différentes balises.

Il faut que vous trouviez sur le Web une carte de référence succincte de HTML sur une ou deux pages, qui puisse vous servir de pense bête. Des mots clefs utiles pour en trouver sont *refcard* (pour *carte de référence*) et *cheat sheet* (comme anti-sèche en anglais).

Il faut à mon avis que vous ayez lu cette carte de référence de la première à la dernière ligne en comprenant tout de son contenu. De cette manière, vous pourrez l'utiliser rapidement si/quand vous aurez besoin d'un rappel sur HTML.

Trois points pas traités ici sont développés dans des chapitres ultérieurs : comment utiliser les CSS pour contrôler finement l'affichage ; comment saisir des données ; comment avoir des pages dynamiques avec des scripts.

2.8 L'exercice du chapitre

En guise d'échauffement, envoyez-moi l'URL de la carte de référence sur HTML que vous avez choisie, à poster en même temps que votre réponse à l'exercice principal.

L'exercice pri9ncipal pour ce chapitre consiste à réaliser directement en HTML une page Web personnelle, si ce n'est pas déjà fait. [Si vous avez une page Web réalisée autrement que directement en HTML, ça ne va pas : réalisez en une autre juste pour l'exercice ; si vous avez une page réalisée directement en HTML + CSS, retirez-en les CSS et faites le nécessaire pour qu'elle soit lisible.]

La page doit contenir au moins du texte (plusieurs paragraphes, avec de vraies phrases : sujet-verbe-complément a minima), des images, des liens (pas cassés). Des listes, des tableaux, du HTML5 sont facultatifs.

Essayez d'en soigner la mise en page et placez la sur le serveur d'hébergement de votre choix. Transmettez moi l'URL de la page avec éventuellement un commentaire sur ce que vous avez réalisé — ou pas réalisé — et/ou des constructions particulières du HTML que vous avez employées.

Dans les exercices des chapitres suivants, vous aurez à modifier cette page ; tentez donc de la garder simple et éditable pour partir sur des bases saines.

Attention : pour que l'exercice soit intéressant, il faut que la page contienne des choses qui vous semblent importantes. C'est nécessaire pour que se manifeste la tension entre contenu et apparence. Les pages qui contiennent des *Lorem ipsum* ne sont pas recevables. Si vous ne savez quel sujet y traiter, faites une page sur vous-même que vous pourrez utiliser pour chercher un stage (un CV, quoi).

Chapitre 3

Les Cascaded Style Sheets

Les Cascaded Style Sheets (très souvent abrégées en CSS) sont la bonne manière de décrire l'apparence que doit prendre un document décrit en HTML.

Dans un monde idéal, le HTML décrirait la structure du document et des CSS indiqueraient de quelle manière chaque type d'élément doit être affiché. Cela permettrait d'avoir un document unique et des CSS adaptées à chaque support ; un CSS pour un écran graphique, un CSS différent pour un écran de téléphone ou un troisième pour une impression sur du papier par exemple.

En pratique, l'imbrication entre les indications de styles des CSS et les indications de structure du HTML est assez forte. Il faut s'efforcer de séparer structure (dans le HTML) et présentation (dans les CSS) mais il est souvent nécessaire d'adapter la présentation à la structure. En revanche, il faut absolument éviter d'adapter la structure à la présentation recherchée : si on a envie de passer par là, c'est le symptome d'un problème plus fondamental (sans doute qu'on n'a mal compris quelque chose ou que la structure est mal adaptée).

Il y a un autre outil qui s'appelle XSL qui ressemble au CSS sans jouer exactement le même rôle. Il s'applique plutôt aux documents purement XML. Je vous recommande d'oublier son existence aussi longtemps que possible.

3.1 Le problème à résoudre

Avec certaines versions d'HTML on a introduit des directives pour avoir des balises qui indiquent comment mettre en page. C'est *très* facile d'en abuser et les concepteurs de pages ne s'en sont pas privé. [1]

1. Un exemple commun de mauvaise utilisation de la mise en page que tous les utilisateurs d'éditeurs WYSIWYG ont probablement commise quand ils étaient débutant : ajouter des espaces en début de ligne pour aligner verticalement un mot avec la ligne précédente. On peut s'imaginer que ça fonctionne jusqu'au moment où on se rend compte (1) que le nombre d'espaces dépend de la police utilisée (2) que quand on change la police, ça ne met pas à jour le nombre d'espaces (3) que le lecteur n'utilise pas nécessairement la même police de caractères

Les CSS résolvent *plus ou moins* le problème. C'est toujours possible — mais plus difficile — de faire du mauvais HTML qui ne s'affichera pas de la manière dont l'a choisi le lecteur.

Dans un fichier qui sert de page de style, ou dans un attribut `style=`, ou avec une balise `<style>`, on place les indications de mise en page en fonction des éléments d'un document. Pour ces éléments, on peut choisir leurs tailles, leurs couleurs, le fond sur lequel ils sont affichés ; la police de caractères utilisé et de façon générale un grand nombre de leurs paramètres d'affichage.

3.2 Une présentation moins brève des CSS

J'explique ici comment on fabrique une CSS et une toute petite partie (l'essentiel à mon avis) des manières dont on peut l'utiliser pour contrôler l'apparence des pages HTML.

3.2.1 Les CSS

L'idée principale est d'associer certains éléments de la page HTML avec la manière de les présenter ; par exemple avec `p { background-color:red; }` j'indique que les paragraphes (la balise `<p>`) doivent être écrits sur fond rouge (le fond est décrit avec la propriété *background-color* ; on lui donne la valeur *red* (rouge)).

Une CSS contient ainsi une liste de règles avec un *sélecteur* qui indique quel(s) élément(s) est(sont) concernée(s) et des déclarations entre accolades qui s'y appliquent. Chaque déclaration contient un nom de propriété, un deux-points, une valeur et un point-virgule qui la termine.

On peut avoir des CSS principalement de trois manières : au niveau du site Web, au niveau de la page, au niveau d'un élément.

---

que celle dont on s'est servi pour écrire et ne voit donc pas les caractères aligné. Ce sont des tabulations qu'il faut utiliser pour ça.

Un autre exemple est la mise en évidence de mots dans le texte ; normalement on le compose en italique et les premières versions d'HTML contenaient une balise (`<i>`) pour ce faire. Un problème surgit quand on a quelque chose à souligner dans un texte en italique : l'usage typographique veut qu'on revienne au romain parce que quelques mots en caractères romain dans un bloc de texte en italique sont mis en évidence. La balise `<i>` ne nous aide pas parce que si on l'emploie dans du texte qui est déja composé en italiques, le texte reste composé en italique. Avec la version courante d'HTML, on a une balise `<em>` (comme *emphasize* = mettre en évidence) qui permet facilement de mettre ça en œuvre : quand le texte environnant est composé en romain, la partie à souligner sera composée en romain et inversement : la balise `<em>` permet l'imbrication ; elle indique de la structure et pas de la mise en page.

Le problème est bien décrit par un article ancien qui s'élevait contre ce genre de dérives ; voir `http://home.olemiss.edu/~mudws/font.html` (1996). À lire et à comprendre. (Si vous avez du mal avec l'anglais, considérez cette lecture comme une bonne occasion de pratiquer.)

3.2.2 Au niveau du site web

On peut placer tout un ensemble de déclarations pour les CSS dans un fichier et le faire utiliser dans un ensemble de page en plaçant dans le HTML de la page quelque chose comme `<link rel="stylesheet" type="text/css" href="style.css">`.

Ça indique un document (qu'on trouvera à `style.css` comme indiqué par le `href=`) qui sert de feuille de style de la page (`rel="stylesheet"`). Il contient une description de CSS sous forme de texte brut (`type="text/css"`), ce qui signifie qu'on peut l'éditer avec un éditeur de texte ordinaire comme Emacs ou Vi et visualiser son contenu dans un terminal avec des commandes comme `cat`, `more` ou `less`.

On lit que le principal intérêt de ce type de CSS est qu'on peut modifier tout l'aspect d'un site web juste en changeant le contenu de cette CSS. En pratique, je crois que c'est rarement une bonne idée de changer tout le site comme ça. Il vaut mieux considérer à mon avis que le principal intérêt est de fixer des valeurs par défaut raisonnables qui permettent ordinairement d'oublier les CSS. [2]

Au niveau de la page

Dans un fichier HTML, on peut placer des bouts de CSS au début du fichier entre des balises `<style>···</style>` dans l'en-tête HTML (dans la partie `head`). Par exemple, quelque chose comme `<head> <style>body { background-color:#decede; }</style>` pour avoir un fond mauve pale.

Au niveau de l'élément

Dans le corps d'un fichier qui contient du HTML, on peut aussi rencontrer des indications de CSS qui ne s'appliquent qu'à un seul élément. Ils sont alors déclarés avec l'attribut `style` et ne contiennent que la déclaration (ce qu'on trouve entre les accolades quand on définit une CSS au niveau du site ou de la page). Par exemple `<p style="background-color:#efface;">` pour avoir un paragraphe qui a un fond d'une autre couleur que le reste. Ici, il n'y a pas de sélecteur : c'est clair que le style ne s'applique qu'à ce paragraphe.

C'est souvent assez mauvais signe quand on a besoin d'utiliser ce genre de spécifications ; on est probablement en train de manquer un élément significativ de la page qui mérite une déclaration pour lui-même ou bien on est en train de faire une horreur typographique. Si on répète une valeur d'attribut `style`, c'est presque certain qu'on est sur la mauvaise pente : il vaut sans doute mieux utiliser des classes d'éléments avec les sélecteurs décrits dans la section suivante.

2. Voir tout de même le site `http://csszengarden.com` à titre d'exemple de ce qu'on peut obtenir en mélangeant une bonne connaissance de HTML et des CSS avec de bonnes conceptions de pages Web.

```
<style>.fondrouge { background-color: red; }</style>
...
<h1 class="fondrouge">Exception</h1>
<p class="fondrouge">Ce paragraphe décrit l'exception.
```

```
<style>div.special { background-color: red; }</style>
...
<div class="special">
  <h1>Exception</h1>
  <p>Ce paragraphe décrit l'exception
</div>
```

FIGURE 3.1 – Deux manières de décrire un cas exceptionnel pour l'affichage par les CSS. Dans la première, on marque indépendamment le titre et le paragraphe comme exceptionnels (sur fond rouge). La deuxième manière est clairement meilleure parce que l'affichage exceptionnel concerne bien une structure constituée du titre et du paragraphe réunis, comme l'indique le `<div>`.

3.2.3 Les sélecteurs

Une déclaration dans une CSS indique à quelle structure elle s'applique.

Ce peut être une balise du HTML comme `<h1>` ou `<p>` pour désigner tous les titres ou tous les paragraphes. Dans ce cas, on met juste la (les) balise(s) avant les accolades, comme dans `h1, p { background-color:blue;}` pour avoir tous les titres de premier niveau et tous les paragraphes affichés sur fond bleu.

Ça peut désigner toutes les éléments qui appartiennent à une (ou plusieurs) classe(s) qu'on aura mentionnée(s) dans l'attribut `class` de la balise. Par exemple, si j'ai des paragraphes spéciaux, je peux les marquer dans le HTML avec `<p class="special">` et modifier leur affichage avec le nom de la classe comme dans `p.special {background-color:red;}`. On peut aussi sauter le nom de l'élément, ici avec `.special`, et tous les éléments appartenant à cette classe seront affectées.

Les balises `div` et `span` permettent de définir à peu près n'importe quelle structure dans une page HTML (avec la limite évoquée dans le chapitre précédent : on ne peut avoir qu'un arbre). Leurs sélections par classe pour les CSS permettent de les afficher d'à peu près n'importe quelle façon.

Ainsi, au lieu de noter qu'un paragraphe et son titre sont tous les deux de la classe `special`, il vaut mieux définir une `div` de la classe `special` qui contient le paragraphe et son titre (figure 3.1).

Il y a des sélections plus fines possibles, comme sélectionner un certain type d'éléments seulement quand on les utilise à l'intérieur d'un autre type d'éléments, sélectionner un élément par son numéro à l'intérieur d'un autre élément ou suivant le type des éléments voisins.

Un (pseudo)-sélecteur largement utilisé est `:hover` : il sert à modifier l'ap-

parence d'un élément quand la souris passe dessus. Essayer par exemple avec le fichier `hover.html` dont le style contient :

```
<style>
  p { color: black; background-color: white; }
  p:hover { color: white; background-color: black; }
 </style>
```

Quand la souris passe sur un paragraphe, il passe en vidéo inverse.

3.2.4 Les couleurs, les mesures, les boîtes, les polices

Il y a quelques choses qui méritent des explications particulières pour lesquelles je fournis quelques détails.

La désignation des couleurs

Pour les couleurs, on peut utiliser des noms de couleurs communes en anglais comme *black*, *white*, *red*, *blue* ou *yellow*. (Il y a plus de cent noms de couleurs comme ca, qui ne me semblent pas toujours clairs ; la différence entre *navy* et *darkblue* par exemple ne me saute pas aux yeux.)

On peut aussi nommer une couleur avec un nombre hexadécimal sur 3, 6 ou 9 chiffres qui indiquent un mélange de Rouge, Vert et Bleu (rgb) ; c'est de la synthèse additive, donc #000000 est noir et #ffffff est blanc. #c0ffee donne un vert pastel assez plaisant comme fond d'écran à mon avis. (En rgb, *navy* et *darkblue* se codent avec `#000080` et `#00008d` : on comprend qu'elles se ressemblent.)

Finalement, avec la version 3, on peut aussi spécifier un canal alpha (pour de la transparence) et du codage de couleur en HSL (Teinte = *Hue*, Saturation, Luminance).

Au lieu de spécifier une couleur, on peut désigner une image ; dans ce cas on indique son URL avec quelque chose comme `background-image:url("fond.gif");`. (Dans le cas où on utilise une image en guise de fond d'écran il y de nombreux paramètres pour positionner cette image et préciser comment elle se répète horizontalement et verticalement.

Les unités de mesures

Quand on spécifie des tailles dans les CSS, on peut utiliser différents systèmes de mesures. Les mesures peuvent être absolues ou relatives.

Mesure absolues : on peut spécifier les mesures en centimètres avec quelque chose comme `1cm`, en millimètres avec `1mm` ou en pouces (2.54 cm) avec `1in`. Les

typographes (et les pages HTML) utilisent aussi beaucoup le *point* (qui vaut 1/72 de pouce, un peu plus qu'un tiers de millimètre) avec `1pt`.[3]

On a aussi des mesures absolues qu'on ne connaît pas au moment où on décrit la CSS qui s'expriment en pixels avec `1px`. La taille que ça représentera dépendra de l'écran sur lequel la page sera affichée. À n'utiliser que pour les filets et les cadres.

Relatives à une police : avec une police de caractères (*font* en anglais), on a un autre type de taille : le *em* représente la hauteur des caractères de la police[4] avec `1em`.

Finalement on a une taille relative qui se définit avec le caractère pour-cent (`%`) par rapport à une autre taille ; par exemple avec `width: 50%`, on spécifiera quelque chose qui occupe la moitié de la largeur disponible.

Les boîtes

Autour d'un élément HTML, CSS place des marges, un cadre et du bourrage. Ça s'utilise couramment autour des images et des tableaux et c'est souvent spécifié en pixels.

Les marges, c'est ce qui entoure l'élément sans en faire partie. Ça aura la couleur de fond de l'élément du niveau supérieur (du `body` par exemple). On peut spécifier des marges différentes pour les quatre cotés *(top, bottom, left* et *right)*.

Les bords, c'est un cadre (facultatif) qui entoure l'élément ; on peut spécifier un style (comme `solid` ou `double`) et une couleur, voire une image en CSS 3.

Le bourrage (padding), c'est de l'espace entre le cadre et l'élément. Le bourrage partagera la couleur de fond de l'élément.

On peut positionner ces caractéristiques avec `margin`, `border` et `padding`. La version simple affecte tous les cotés avec par exemple `padding: 3px;` pour un bourrage de 3 pixels sur tous les cotés, On peut aussi spécifier des caractéristiques différentes sur les cotés en indiquant quatre tailles différentes. Par exemple `margin: 2px 4px 8px 16px;` demande une marge de 2 pixels en haut, 4 pixels en bas, 8 pixels à gauche et 16 pixels à droite. Il y a aussi des variantes à 3 et 2 mesures et des indications spécifiques comme `border-left` ou `margin-right`.

On peut aussi forcer des largeurs et des hauteurs avec `width` et `height`. Évitez de l'utiliser car c'est facile avec ça de produire du texte qui ne s'affiche correctement que sur son propre écran.

3. Il y a aussi le pica qui vaut 12 points mais il est moins utilisé.

4. En typographie traditionnelle, le *em* est la *largeur* du caractère m. La hauteur des caractères d'une police se mesure plutôt avec celle du caractère X ; les CSS supportent le *ex* comme unité mais elle est très peu employée.

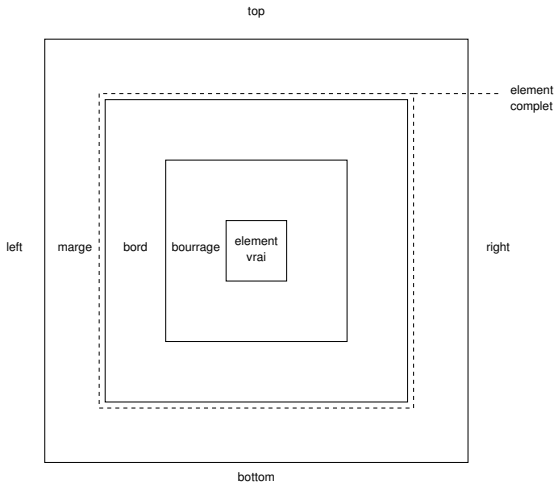


FIGURE 3.2 – Autour d'un élément, on trouve le bourrage et le bord, qui font partie de l'élément complet ; ils seront donc affectés par les propriétés qu'on aura spécifiées pour l'élément, comme sa couleur de fond. La marge est tout à l'extérieur ; elle fait partie de l'élément de niveau supérieur, donc les caractéristiques de l'élément ne l'affectent pas.

Les polices

On peut positionner tous les paramètres de la police à utiliser avec la propriété `font` comme dans `font: italic bold medium courier` ou bien positionner séparément certains paramètres. Les plus importants à mon avis :

- `font-style` comme `normal`, `italic` ou `oblique` (l'oblique est penché comme l'italique mais les 'a' et les 'f' ont la même forme qu'en romain.
- `font-weight` qui peut valoir `normal` ou `bold` (comme *gras*) ; on peut mesurer plus précisément jusqu'à quel point les caractères doivent être en gras ou pas). [5]
- `font-size` qui indique le corps ; on peut y mettre une taille absolue (`15pix` est commun) mais ça retire de la souplesse au navigateur et au lecteur : à éviter autant que possible. Utiliser plutôt une taille relative définie avec %, `smaller` ou `larger`. Sinon les noms de tailles vont de `xx-small` à `xx-large`.
- `font-family` qui peut être un nom générique comme `serif` ou `sans-serif` (avec ou sans empattement). [6]

5. La *graisse* de caractères est l'épaisseur du trait avec lequel ils sont tracés. En typographie classique, on évite de composer du texte justifié en gras car le résultat ressemble à une tache sur la page et rompt la lecture. On le réserve pour ce qui doit attirer l'oeil comme un titre.

6. En typographie classique, on utilise plutôt les polices sans empattement (comme Helvetica) pour les titres et les polices avec empattement (comme Times) pour le corps du texte ; pour affichage sur les écrans, l'usage est plutôt de tout faire avec des polices sans empattement.

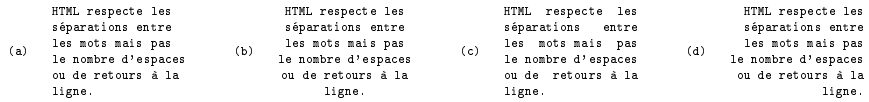


FIGURE 3.3 – Quatre façons de composer le texte : au fer à gauche (a), centré (b), justifié (c) et au fer à droite (d). On peut obtenir ça en spécifiant `left`, `center`, `justified` et `right` comme `text-align:`.

> En guise de nom de famille de police, on peut aussi nommer un nom de police comme `Times`, `Courrier` ou `Helvetica`. Un problème est que les polices de caractères viennent souvent avec des restrictions d'usages (copyright) et des différences subtiles d'implémentation : on ne sait pas trop ce que ça donnera (ni même si ça donnera quelque chose) sur l'écran du lecteur, aussi on tendait à n'utiliser que des polices de caractères courantes.
> Une solution avec les CSS 3 consiste à embarquer la police à utiliser avec le document ; la page spécifie où trouver la description des caractères de la police avec `@font-face` et le navigateur charge cette description si nécessaire.
> Dans tous les cas, éviter la multiplication des polices : sur une page, il faut sans doute n'utiliser qu'une seule police ou bien être très fort en graphisme pour ne pas faire quelque chose d'horrible.

Le texte

On peut centrer, mettre au fer à gauche ou à droite et justifier (aligner les marges gauches et droites en agrandissant les espaces entre les mots) avec la propriété `text-align` (figure 3.3).

On peut *décorer* le texte avec une ligne par dessus, par dessous ou en le barrant (avec `text-decoration`) ; c'est à éviter au maximum parce que ça perturbe la lecture.

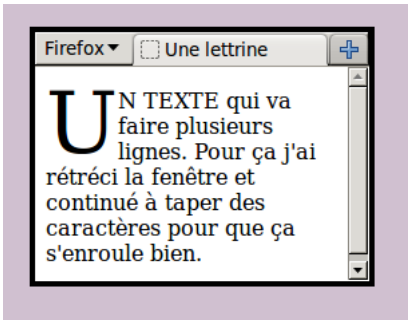
On peut faire bien d'autre choses encore (indenter, passer en majuscule ou en minuscules, modifier l'espace entre les caractères (le *crénage*).

On dirait que la balise `<blink>` qui permettait de faire des pages HTML vraiment hideuses n'a pas trouvé sa place dans les CSS. RIP.

3.2.5 float et clear avec div et span

La propriété `float` permet de laisser le navigateur choisir précisément l'endroit où placer quelque chose. La propriété `clear` permet de l'encourager à placer ce qui flotte à proximité d'un autre objet.

Il est courant d'utiliser `float` et `clear` avec `div` et `span` pour organiser complètement une page web. L'ancienne méthode qui consistait à utiliser des tableaux pour ce faire est parait-il passée de mode (mais encore utilisée dans les




```
<style>
  .lettrine { font-size: 400%;
              float: left;
              margin: -0.20em 0px -0.20em 0px; }
</style>
...
<body>
  <span class=lettrine>U</span>N TEXTE qui va
...
```

FIGURE 3.4 – Grâce au `float`, le texte s'enroule autour du grand U qui commence le chapitre : on a à peu près l'équivalent d'une lettrine.

mails que Facebook m'envoie).

Le but recherché

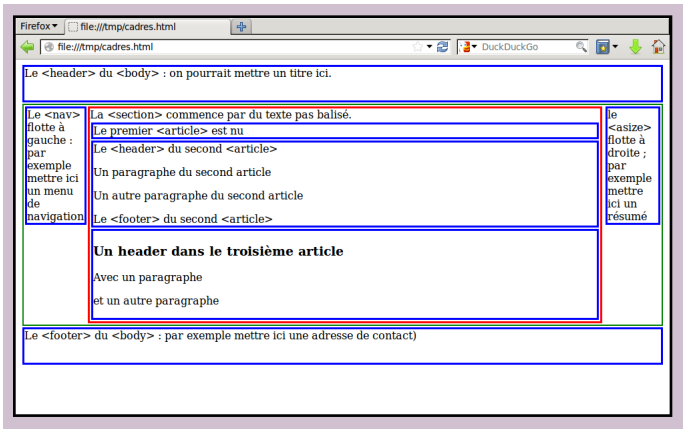
Le but est de donner des indications au navigateur qui lui permette de placer les éléments d'une façon harmonieuse à l'intérieur de la page ; dans un monde idéal, ces indications fonctionneraient pour n'importe quel support (aussi bien le grand écran d'un ordinateur que celui minuscule d'un téléphone). En pratique, c'est rarement le cas ; la plupart des sites donnent aux navigateurs des indications différentes suivant ce qu'ils savent de l'écran.

float pour enrouler du texte

On peut se servir de `float` pour que le texte s'enroule autour d'un élément, comme le texte d'un article s'enroule autour d'une image dans un article de Wikipedia. On peut par exemple l'utiliser pour réaliser une lettrine simple (figure ??).

Organiser une page

Avec `float` et `clear`, on peut procéder à l'imbrication de structures complexes pour contrôler leur affichage. Ainsi, un exemple rapide, combiné avec l'utilisation des éléments du HTML5 (décrits sommairement dans 2.5, page 23) donne la figure suivante :



La page est découpée en trois zones : la partie centrale est un `div`, encadré par un *header* et un *footer*. Le `div` se compose de deux zones flottantes à gauche et à droites (j'ai choisi d'utiliser les marqueurs `nav` et `aside` pour illustrer leur utilisation) et d'une partie centrale qui est une `section` qui contient des `articles`.

Si on remplace les marqueurs de blocs sémantiques de HTML 5 par des div, on obtiendra exactement le même résultat : c'est l'utilisation des `float` et des `clear` qui permettent d'obtenir ce résultat.

Framework

C'est délicat d'obtenir quelque chose de correct sur tous les affichages (notamment sur un écran d'ordinateur *et* un téléphone.

C'est important d'avoir quelque chose de cohérent entre les différentes pages.

Pour cette raison, presque tout le monde utilise un *framework*. Le découpage est déja fait et il ne reste qu'à remplir les cases.

Transformer une liste de liens en barre de navigation

http ://www.w3schools.com/css/css_navbar.asp

```
-----------  header --------
-             nav           -
-----------------------------
section
```

```
  header
  article
  article           aside
  article
  footer
section
  header
  article aside
  article aside
  footer
------------ footer ------
```

Il y a des centaines de tutoriaux sur le web qui expliquent comment faire ça. Un exercice du chapitre vous demande d'en trouver un qui vous convienne et de l'utiliser.

3.2.6 Je ne développe pas

On peut contrôler l'apparence des liens (selon qu'on a cliqué dessus ou pas, que la souris est dessus ou pas).

On peut faire des tas de choses formidables avec les listes (notamment des menus) et les tables. On peut aussi faire des pages très bien sans les utiliser.

3.3 Par l'autre bout de la lorgnette

background

background-color background-image background-repeat background-attachment background-position

border

The border-style property specifies what kind of border to display.

The following values are allowed :

dotted - Defines a dotted border dashed - Defines a dashed border solid - Defines a solid border double - Defines a double border groove - Defines a 3D grooved border. The effect depends on the border-color value ridge - Defines a 3D ridged border. The effect depends on the border-color value inset - Defines a 3D inset border. The effect depends on the border-color value outset - Defines a 3D outset border. The effect depends on the border-color value none - Defines no border hidden - Defines a hidden border

border-width : 5px ; border-color : red green blue yellow ; border : 5px solid red ; border-radius : 5px ;

border-top-style : dotted ; border-right-style : solid ; border-bottom-style : dotted ; border-left-style : solid ;

margin

absolute

box-shadow : 10px 10px 5px #888888

3.4 L'exercice du chapitre

L'exercice pour ce chapitre consiste à trouver une page sur le web qui vous explique d'une façon qui vous convienne comment utiliser `float`, `clear` avec `span` et `div` pour structurer une page, puis de modifier la home page que vous avez construite au chapitre précédent en utilisant les CSS, y compris `float`, `clear`, `span` et `div`.

Attention : il est important qu'il ne s'agisse pas d'une nouvelle page sur un nouveau sujet mais d'une *modification* de la page utilisée dans le chapitre précédent.

3.5 Ce qui manque

Ce serait bien d'avoir quelque chose sur les frameworks CSS ; sur le Responsive Web Design ; https ://en.wikipedia.org/wiki/Responsive_web_design D'un autre coté, ça commence à faire beaucoup.

Chapitre 4

Le protocole HTTP, clients et serveurs Web

Dans ce chapitre, on voit la manière la plus courante dont le contenu des pages Web est échangé : entre un navigateur et un *serveur Web* à travers un protocole d'Internet appelé HTTP.

4.1 TCP/IP, nom de machines et adresses IP, numéros de ports

Les messages du protocole HTTP sont transportés à travers internet dans un vaisseau qui s'appelle TCP/IP [1] ; les détails de son fonctionnement sortent du cadre du cours [2] mais il est nécessaire tout de suite d'en connaître quelques éléments que je présente ici : ce qu'est une connexion réseau et comment on l'identifie (avec l'identité des machines et des ports qu'elle connecte).

4.1.1 Connexion réseau

Du point de vue des programmes, une connexion réseau ressemble à un fichier : il est nécessaire d'établir la connexion comme on ouvre un fichier ; une fois la connexion établie on y lit et on y écrit des données comme on lit et qu'on écrit des données dans un fichier ou dans un pipe.

La différence principale avec les opérations sur un fichier ou un pipe, c'est qu'une connexion réseau est à double sens : les deux programmes qui tournent aux deux extrémités de la connexion peuvent chacun prendre l'initiative d'une

1. TCP/IP est en réalité un ensemble d'autres protocoles ; c'est la brique de base avec laquelle on construit un internet.
2. TCP/IP est étudié plus en détail dans le cours de réseaux de deuxième année.

écriture ; ce que l'un écrit est reçu par l'autre quand il lit et vice-versa. [3]

Le principal rôle de TCP/IP est de transporter les données d'un bout à l'autre de la connexion, à travers les différents relais qu'elle emprunte, sans erreurs de transmissions (avec des retransmissions en cas d'erreur). Quand la transmission échoue, il doit signaler une erreur d'écriture.

TCP/IP doit utiliser au mieux les capacités du réseau à transporter des données mais sans les dépasser, ce qui provoquerait un embouteillage qui ralentirait considérablement les choses.

4.1.2 Noms de machines et adresses IP

Chaque machine connectée à Internet possède une (ou plusieurs) adresses qui l'identifient de manière unique sous la forme d'une suite de nombres. Pour éviter de manipuler des nombres, les machines ont aussi presque toutes un (ou plusieurs) noms. Il y a un mécanisme un peu complexe qui permet de traduire l'un dans l'autre.

Noms de machines

Des noms de machine complets sont `www.exalead.com`, `en.wikipedia.org` ou `www.renater.fr`. Ils se composent du nom de la machine proprement dite (ici `www` ou `en`) et d'un nom de domaine (ici `exalead.com`, `wikipedia.org` et `renater.fr`).

Il y a un nom de machine spécial, `localhost`, qui désigne par convention la machine courante. On l'emploiera dans les exemples qui suivent pour éviter de compliquer les choses.

En pratique, les noms de machines complets sont organisés comme un arbre. Voir figure 4.1.

Adresses IP

Les noms de machines ne sont pas très pratiques pour les machines. Elles utilisent plutôt des adresses qui se présentent sous la forme de nombres avec de la ponctuation. Il y a deux systèmes d'adresses en cours d'utilisation sur Internet, les adresses IPv4 et les adresses IPv6. Par exemple les adresses de `en.wikipedia.org` sont 91.198.174.192 (pour IPv4) et `2620:0:862:ed1a::1` (pour IPv6). La machine `www.exalead.com` n'a qu'une adresse IPv4 (qui vaut 178.255.215.34) et pas d'adresse IPv6 au moment où j'écris.

3. Cette caractéristique contraste avec les fichiers où une lecture redonne les données qu'on y a écrites. Les pipes, on les utilise ordinairement d'une façon unidirectionnelle : l'un ne fait qu'y écrire et l'autre lit ce que le premier à écrit.

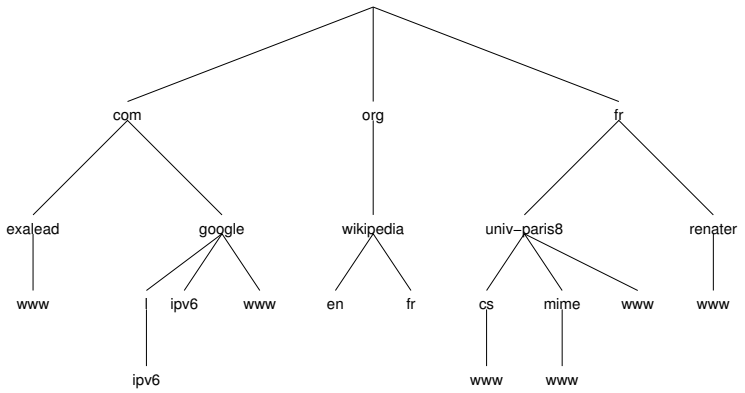


FIGURE 4.1 – L'ensemble des noms de machines sur Internet est organisé comme un arbre. Un nom complet décrit le chemin qui mène jusqu'à la racine, avec les différents noeuds séparés par des points.

La commande `host`

Il y a un mécanisme compliqué, appelé le *Domain Name System* pour mettre en correspondance noms de machines et adresses IP. Je ne le détaille pas dans ce cours. On peut l'interroger depuis la ligne de commande d'un terminal avec la commande `host`. Exemple d'utilisation

```
$ host www.exalead.com
www.exalead.com has address 178.255.215.34
$ host en.wikipedia.org
en.wikipedia.org has address 91.198.174.192
en.wikipedia.org has IPv6 address 2620:0:862:ed1a::1
$ host ipv6.google.com
ipv6.google.com is an alias for ipv6.l.google.com.
ipv6.l.google.com has IPv6 address 2a00:1450:4007:80c::200e
$ host localhost
localhost has address 127.0.0.1
localhost has IPv6 address ::1
```

La machine `Ipv6.google.com` est la seule que je connaisse qui dispose d'une adresse IPv6 sans avoir une adresse IPv4 (le contraire est courant). Cet exemple montre qu'on peut également avoir des *alias* en guise de nom de machine dans le système DNS : `ipv6.google.com` est un autre nom (un *alias*) pour la machine `ipv6.l.google.com`.

Les adresses de la machine courante `localhost` sont aussi des adresses spéciales.

4.1.3 Numéros de ports et noms de services

Pour distinguer les communications réseau différentes sur une même machine, il y a un nombre associé avec chaque extrémité d'une connexion réseau. C'est le numéro de port.

Certains de ces numéros sont réservés pour certains *services* (avec un enregistrement auprès de l'IANA, l'*Internet Assigned Number Authority*). Par exemple, le port numéro 80 est réservé pour le Web HTTP ordinaire, les ports numéros 8000 et 8080 pour le Web expérimental ; le port numéro 443 pour le Web sécurisé en cryptant les messages.

Sous Unix, les utilisateurs ordinaires n'ont pas le droit de se servir des numéros de ports inférieurs à 1024.

Une connexion est complètement identifiée de façon unique par les adresses et les numéros de ports qu'elle utilise *sur les deux machines qui se trouvent aux deux extrémités.* [4]

4.1.4 Une connexion TCP élémentaire avec `netcat`

Je présente ici la commande `netcat` (ou `nc`) qui permet de jouer facilement avec les connexions TCP au niveau de la ligne de commande. Il peut être nécessaire de l'installer spécifiquement sur votre machine.

Dans son usage le plus simple, on peut lancer avec `netcat` un processus qui attend une connexion réseau puis lui connecter, avec un autre `netcat`, un autre processus.

Pour rendre la manip plus facile à mettre en oeuvre, on va ici lancer les deux commandes `netcat` dans deux fenêtres de la même machine en utilisant `localhost` (= la machine courante). Les choses se passeraient de la même manière avec deux commandes lancées sur deux machines différentes.

Les commentaires sont en italiques et commentent la ligne qui précède.

4. Puisque les deux extrémités sont concernées, cela a la conséquence importante que plusieurs connexions peuvent utiliser le même numéro de port sur la même machine si les adresses ou les numéros de ports des autres extrémités sont différentes.

fenêtre 1 | *fenêtre 2*
`$ netcat -l -p 2016` |
attente de connexion sur le port 2016
| `$ netcat -p 20160 localhost 2016`
connexion au port 2016 de localhost (depuis le port 20160)
la connexion est établie (rien de spécial n'est affiché)
`Bonjour !` |
on tape une ligne
| `Bonjour !`
après le retour chariot, la ligne est envoyée et apparaît dans l'autre fenêtre
Ça fonctionne aussi dans l'autre sens
^C |
$ |
On termine une des commandes ; on a un nouveau prompt
| $
La connexion est coupée : l'autre commande sort aussi

Cette session montre l'ouverture de la connexion réseau : un côté attend sur le port numéro 2016, une connexion depuis n'importe quelle machine sur Internet ; l'autre côté s'y connecte depuis le port 20160.

Quand on tape une ligne d'un côté, elle est transmise telle quelle de l'autre côté (la ligne n'est transmise qu'une fois complètement tapée).

Quand un des côtés de la connexion sort, la connexion est coupée et l'autre côté sort à son tour.

4.2 HTTP

À travers une connexion internet, les échanges d'information doivent se faire dans un format compris par les programmes qui communiquent. Ce format s'appelle un *protocole*.

Le principal protocole utilisé sur le Web s'appelle HTTP comme *Hyper Text Transfer Protocol*. Il est d'un abord plutôt facile et ses grands principes sont présentés dans cette section. [5]

4.2.1 Le modèle client-serveur

Le protocole est fondé sur le modèle *client-serveur*.

Il y a un *serveur* Web qui gère un site Web (ou plusieurs). Il est en attente de demande de *clients* qui interrogent le site, comme le premier netcat lancé dans l'exemple de la section précédente attendait une connexion du deuxième netcat.

5. Les choses se compliquent sérieusement quand on rentre dans les détails qui permettent d'accélérer les échanges, notamment avec l'utilisation des caches.

Un navigateur web (Firefox, Opera ou autre) se connecte au serveur (comme le deuxième netcat de l'exemple de la section précédente) et transmet une requête au serveur Celui-ci répond à la question sur la même connexion.

Le principal intérêt du modèle client-serveur est sa simplicité. Les échanges de données servent à la fois à la transmission de l'information utile et à la synchronisation. Les choses sont beaucoup moins compliquées que quand les communications peuvent se produire à l'initiative de n'importe lequel des interlocuteurs.

4.2.2 Les méthodes de HTTP

Chaque requête du client au serveur contient un type de question ou de demande qu'on appelle une *méthode*. Chaque méthode correspond à une opération que le client demande au serveur d'effectuer. Les plus importantes à mon avis sont

- GET sert à récupérer une page (en pratique un fichier).
- HEAD sert à récupérer les paramètres d'une page (par exemple pour connaître sa taille) avant de la récupérer via GET.
  La spécification de HTTP demande que tous les serveurs sachent répondre aux requêtes GET et HEAD ; les autres sont facultatives.
- OPTIONS permet à un client d'interroger un serveur sur ce qu'il accepte de traiter.
- POST, qui sera détaillé dans un autre chapitre, permet au client d'envoyer des données au serveur ; c'est souvent de cette manière qu'on envoie les données d'un formulaire qu'on a rempli.

Il y a d'autre méthodes que je n'ai pas l'intention de développer, comme PUT (une autre manière d'envoyer des données du client vers le serveur) ou DELETE (le client demande au serveur de supprimer une page).

4.2.3 La récupération d'une page Web

Je vais montrer ici comment récupérer une page Web avec `netcat`. Cela va nous permettre d'examiner en détail et à notre rythme les opérations effectuées habituellement à toute vitesse par Firefox ou son équivalent. On va regarder ce qu'on trouve à l'URL `http://www.ai.univ-paris8.fr/~jm/page.txt`.

Établissement de la connexion

Tout d'abord, il faut établir une connexion entre le client et le serveur. On utilise netcat avec

```
netcat www.ai.univ-paris8.fr 80
```

c'est à dire le nom de machine `www.ai.univ-paris8.fr` et le port 80 (le port usuel pour le Web). On peut maintenant entrer la requête, que `netcat` trans-

mettra au serveur.

La requête GET

On tape à netcat :

```
GET /~jm/page.txt HTTP/1.1
Host: www.ai.univ-paris8.fr
```

Le client demande la page Web au serveur avec une requête GET. La première ligne commence par le mot GET, suivi du nom de la page demandée et se termine avec la version du protocole utilisé par le client (ici la version 1.1 du protocole HTTP).

Les lignes suivantes peuvent contenir des paramètres, chacun sur une ligne. Chaque ligne commence par le nom du paramètre, suivi du caractère ' :' puis sa valeur. Il y a ici le paramètre obligatoire qui est le nom du site Web sur lequel se trouve la page. [6]

La requête est terminée par une ligne vide (qui n'apparaît pas clairement sur le support mais si vous ne la tapez pas, le serveur va l'attendre indéfiniment).

La réponse du serveur

On reçoit en retour les lignes suivantes :

```
HTTP/1.1 200 OK
Date: Wed, 22 Jul 2015 17:51:09 GMT
Server: Apache/2.2.15 (Unix) mod_ssl/2.2.15 OpenSSL/0.9.8zc PHP/5.3.1
Last-Modified: Wed, 22 Jul 2015 17:50:32 GMT
ETag: ``6a803f-b1-52c87b3c82a00''
Accept-Ranges: bytes
Content-Length: 177
Content-Type: text/plain

Bonjour,

Ceci est une page de texte que j'ai placée dans ma page Web juste
pour le cours de pratique des ordinateurs. Il s'agit de texte ordinaire.

Jean Mehat, juillet 2015.
```

Ces lignes contiennent un compte-rendu, un en-tête et un contenu.

6. Le nom du site semble faire doublon avec le nom du serveur de la demande de connexion mais permet à un seul serveur de contenir facilement plusieurs sites.

Le compte-rendu Le compte-rendu est entièrement contenu dans la première ligne de la réponse. Le serveur commence pas la version du protocole qu'il utilise (c'est aussi `HTTP/1.1`) puis un compte-rendu sur 3 chiffres suivi d'une chaîne de caractère qui traduit ce compte-rendu de façon compréhensible pour les humains (ici `200 OK`).

Le premier chiffre du compte-rendu indique le sens général de la réponse. Il peut varier entre 100 et 500.

1xx jusqu'ici ça va bien mais il faut continuer.

2xx tout s'est bien passé

3xx il faut essayer autrement

4xx ça n'a pas marché (mais ça marchera peut-être plus tard)

5xx ça n'a pas marché (et ça ne marchera pas plus tard).

Tout le monde connaît le compte-rendu 404 : ça n'a pas marché parce que la page demandée n'existe pas pour le moment ; peut-être qu'elle existera plus tard.

L'en-tête Les lignes qui suivent sont dans le même format que les paramètres de la demande : un nom, suivi d'un deux points et une valeur.

Sont particulièrement importants ici la ligne `Content-Length` (qui indique le nombre d'octets du contenu) et `Content-Type` qui indique comment l'interpréter.

Le contenu La fin de l'en-tête est indiquée par une nouvelle ligne vide. À partir de là on trouve les octets du contenu. La manière dont le contenu sera affiché dépend bien sûr du navigateur et du type du contenu. Comme on peut le constater en consultant la page de la manière usuelle, le texte ordinaire (`text/plain`) est simplement affiché sans mise en page avec Firefox.

On peut avoir toute sorte de types de données différentes. La première partie indique le genre des données ; la seconde la manière dont elles sont encodées.

text/html

text/css

text/plain

image/png

image/jpeg

La liste officielle à `https://www.iana.org/assignments/media-types/media-types.xhtml` (elle est longue).

La suite Après la réponse initiale du serveur, le client peut envoyer une nouvelle requête sur la même connexion ou fermer la connexion. Le serveur ferme la connexion tout seul si on tarde trop (quelque secondes) à demander la suite.

4.2.4 Côté serveur

Côté serveur, les choses ne sont pas beaucoup plus compliquées pour faire tourner le protocole HTTP de base. On trouve toute une collection de serveurs HTTP élémentaires sur la page `https://gist.github.com/willurd/5720255`.

Quand Python version 2 est installé, on peut lancer un serveur élémentaire avec la commande « `python -m SimpleHTTPServer 8080` ». Le serveur écoute sur le port 8080 et répond en servant les pages de son répertoire courant. Depuis la même machine, on interrogera donc l'URL `http://localhost:8080/`.

Constater que quand l'URL désigne un répertoire, on voit le contenu du fichier `index.html` s'il existe. S'il n'existe pas, c'est la liste des fichiers qu'on obtient.

Constater ce qui se passe quand on interroge un fichier dont le nom ne se termine pas par `.html`.

Constater ce qui se passe quand on demande la lecture d'un fichier binaire avec netcat.

Étudier le source Python du module.

4.3 Les serveurs Web courants

Les serveurs Web courants
— Apache
— Microsoft Internet Server
— Nginx
— les autres (à peu près négligeables)

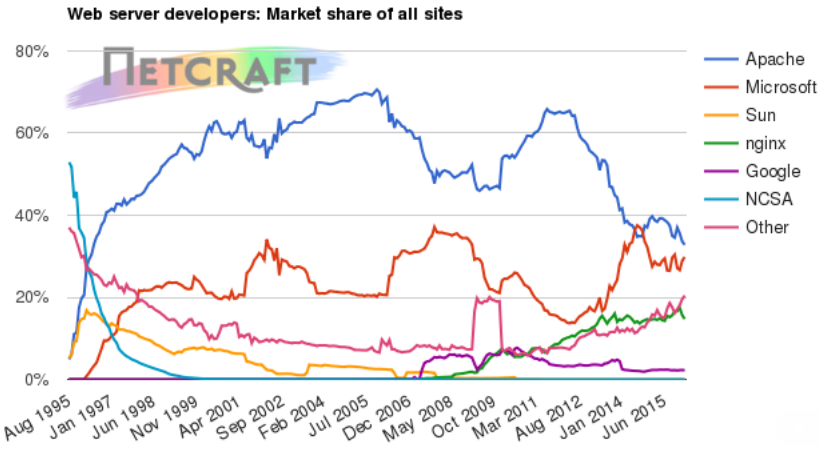
Netcraft est une société qui étudie depuis une vingtaine d'année le fonctionnement d'Internet, et du Web en particulier. Je recopie ici les graphiques qui montrent l'évolution des parts de marchés des divers serveurs Web.

Il y a deux graphiques. Le premier pour tous les sites découverts par Netcraft ; le second pour le million de sites les plus actifs. Le contraste entre les deux est intéressant à mon avis.

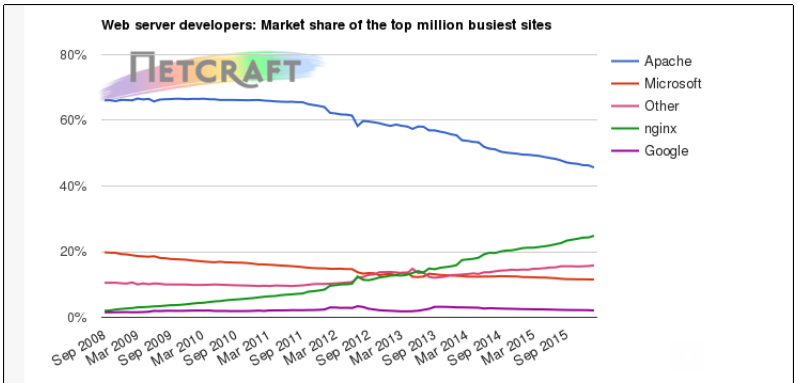
Quand on considère tous les sites, les choses évoluent en dents de scie, essentiellement parce que certains hébergeurs de sites Web ont une telle importance que quand ils changent leur infrastructure logicielle, ça a un effet net sur la courbe.

Si on ne considère que les sites les plus actifs, les tendances sont plus nettes : le serveur Apache est beaucoup plus important que tous les autres, depuis longtemps. Il perd peu pas peu du terrain face à une autre serveur libre : Nginx. Si les choses continuent au même rythme, Nginx devrait devenir aussi important qu'Apache vers 2019 ou 2020.

Dans le cours, on va utiliser Apache.



Developer	January 2016	Percent	February 2016	Percent	Change
Apache	304,271,061	33.56%	306,292,557	32.80%	-0.76
Microsoft	262,471,886	28.95%	278,593,041	29.83%	0.88
nginx	141,443,630	15.60%	137,459,391	14.72%	-0.88
Google	20,799,087	2.29%	20,640,058	2.21%	-0.08



Developer	January 2016	Percent	February 2016	Percent	Change
Apache	463,092	46.31%	456,483	45.65%	-0.66
nginx	243,340	24.33%	248,627	24.86%	0.53
Microsoft	115,358	11.54%	115,447	11.54%	0.01
Google	21,824	2.18%	21,008	2.10%	-0.08

4.4 Le serveur Apache

Avant l'installation, regarder le contenu de `/etc/apache2` (pour pouvoir comparer après).

4.4.1 Installation de Apache

Sur Ubuntu, l'installation simple se fait avec `sudo apt-get install apache2`. (On pourrait demander des extensions qui se trouvent dans des *modules* ; on peut aussi les installer plus tard.)

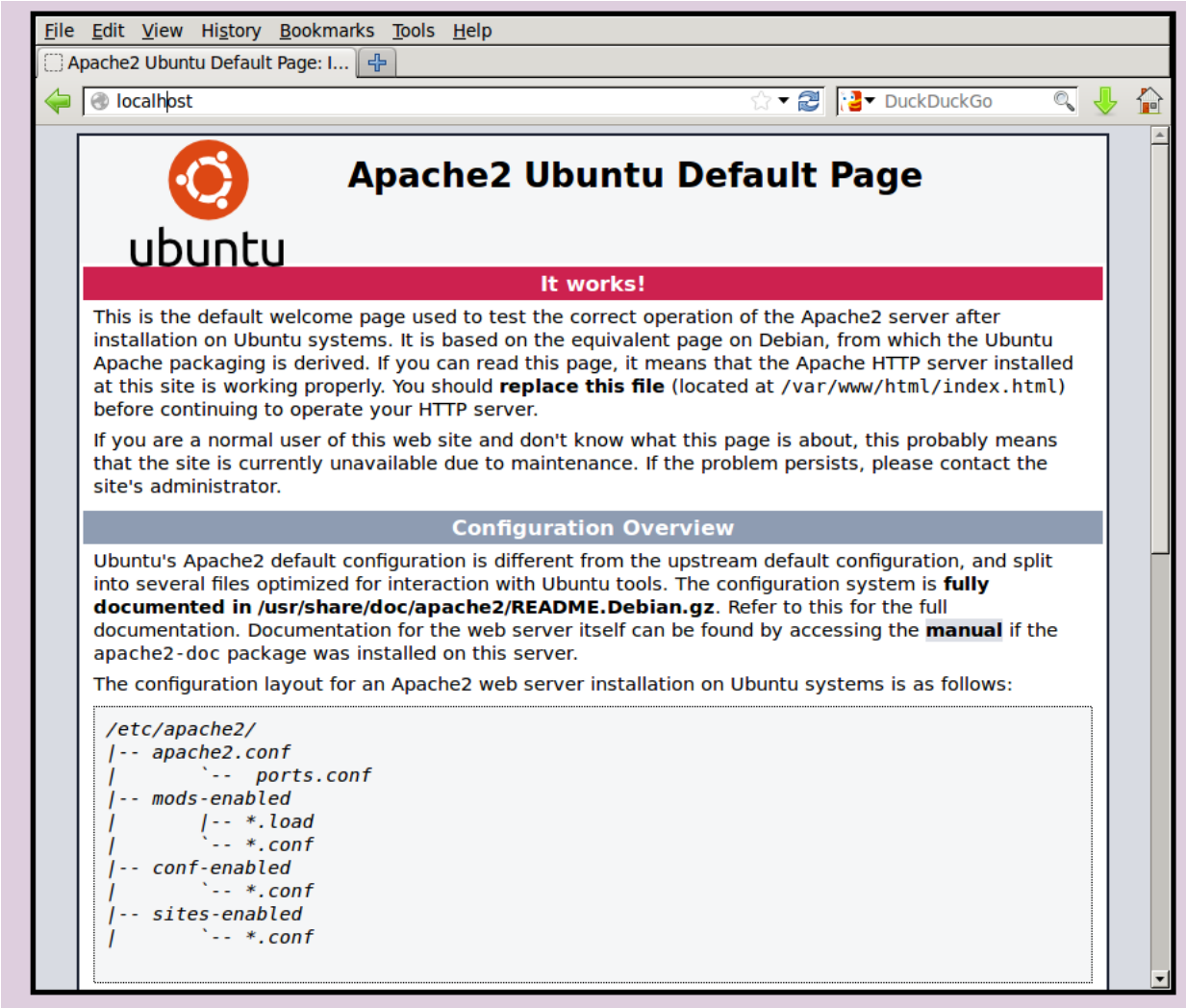
Sur ma machine, message d'erreur :

```
AH00558: apache2: Could not reliably determine the server's
fully qualified domain name, using 127.0.1.1. Set the
'ServerName' directive globally to suppress this message
```

Ce n'est pas bien grave (machine isolée). On peut y accéder (à la page par défaut) à l'adresse `http://localhost`.

4.4.2 Page de démarrage

La page par défaut sous Debian/Ubuntu. (Sur beaucoup d'autres systèmes ou distribution, on a juste une page qui contient *It works !*.)



La lecture de la page est intéressante : elle donne des éléments sur la configuration.

Le fichier qu'on voit par défaut s'appelle `index.html` et se trouve dans `/var/www/html/`. Le lire. Le sauvegarder. Le modifier.

4.4.3 Les fichiers de configuration

Tous les fichiers de configuration de Apache2 sont dans le dossier `/etc/apache2`. Il faut commencer la lecture avec `/etc/apache2/apache2.conf`.

Le principal format utilisé est le XML ; c'est un peu comme du HTML avec des balises mais il n'y a pas de vraiment de contenu : tout est dans la structure

de l'arbre et les attributs.

En dessous de `/etc/apache2`, il y a trois répertoires qui contiennent des paramètres, pour certains déjà prêts à l'emploi, pour la configuration, les modules et les sites. Leurs noms se terminent par `-available`.

Pour les activer, il faut créer un lien symbolique vers ces fichiers dans dans les répertoires `-enabled`. Il y a des commandes pour faire ça d'un coup sont le nom commence par `a2` (comme *Apache 2*) ; voir leurs pages de manuel. Un exemple sans ces commandes dans le prochain paragraphe.

4.4.4 Une modification simple de la configuration

Mon nom de login sur la machine est `jm` ; il faut adapter ce qui suit à votre propre nom de login. [7]

Fabriquer un répertoire `public_html` dans son répertoire de login et y placer un fichier `index.html`. (Vous n'êtes pas du tout obligé de fabriquer le fichier avec la commande `cat` pour établir le contenu du fichier ; je ne l'emploie ici que pour ôter toute ambiguïté aux commandes copiées.)

```
$ cd
$ mkdir public_html
$ cat > public_html/index.html <<FINIRICI
<!DOCTYPE html>
<html>
  <head>
    <title>Page Maizon</title>
    <meta charset=utf8>
  </head>
  <body>
    <p>Je suis un <em>acteur</em> du web !
  </body>
</html
FINIRICI
$
```

Vérifier que `http://localhost/~jm` n'est pas accessible (= erreur 404).

Aller dans `/etc/apache2` et y fabriquer des liens de `mods-enabled` vers `mods-available` pour `userdir.conf` et `userdir.load`.

```
$ cd /etc/apache2/mods-enabled
$ ln -s ../mods-available/userdir* .
ln: failed to create symbolic link ‘./userdir.conf’: Permission denied
ln: failed to create symbolic link ‘./userdir.load’: Permission denied
```

7. On peut retrouver facilement son nom de login avec la commande `who` ou la commande `id` ou en regardant les dernières lignes du fichier `/etc/passwd` et de quelques autres façons encore.

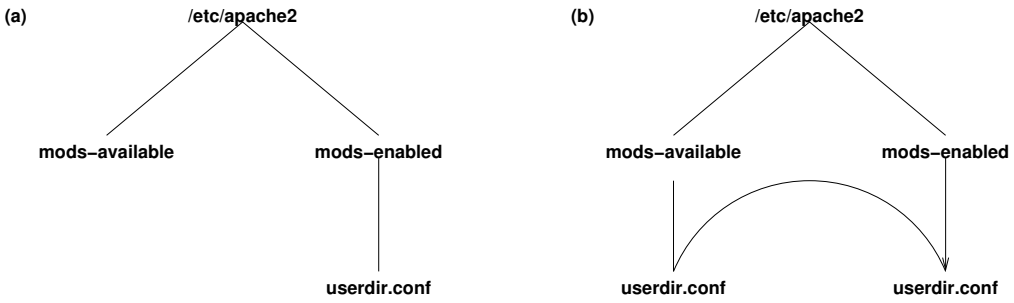


FIGURE 4.2 – (a) Faute de contenu du fichier `userdir.conf` dans le répertoire `/etc/apache2/mods-enabled`, le module n'est pas actif. (b) Pour l'activer, il suffit de créer un lien symbolique de `/etc/apache2/mods-enabled/userdir.conf` vers `/etc/apache2/mods-available/userdir.conf`. Je n'essaie pas d'expliquer ici pourquoi il vaut mieux utiliser un lien symbolique plutôt qu'une copie ou qu'un lien en dur. Vous devriez le comprendre seul après quelques mises à jour du serveur.

```
$ sudo ln -s ../mods-available/userdir* .
$ ls -l userdir*
lrwxrwxrwx 1 root root 30 févr. 25 16:48 userdir.conf -> ../mods-available/userdir.conf
lrwxrwxrwx 1 root root 30 févr. 25 16:48 userdir.load -> ../mods-available/userdir.load
```

Vérifier que `http://localhost/~jm` n'est toujours pas accessible.

Relancer le serveur Apache

```
$ sudo /etc/init.d/apache2 restart
 * Restarting web server apache2
AH00558: apache2: Could not reliably determine the server's etc.
```

Constater que `http://localhost/~jm` est maintenant accessible et qu'on y trouve le contenu du fichier `index.html` fabriqué au début de la section.

4.4.5 Sauter l'installation d'Apache

On peut trouver un serveur équivalent (avec php) avec un site Free. On peut en obtenir un en s'inscrivant (sans l'utiliser pour ne pas payer) pour l'accès via un téléphone ordinaire (qu'on trouve sous le nom *bas débit* sur le site). Ça permet d'avoir un site Web visible de partout sur Internet sans avoir à rendre sa machine accessible à travers le boîtier du fournisseur d'accès (pas facile à réaliser et un peu risqué : on verra ça dans un autre chapitre).

4.5 Les exercices

Recopier un gros fichier entre deux machines en utilisant netcat. Mesurer le temps que ça prend.

Effectuer une petite modification du module serveur HTTP de python (par exemple rajouter un type MIME).

Choisir et effectuer une modification intéressante sur la configuration d'un serveur Apache. M'envoyer l'adresse du serveur en question avec une indication de la modification que vous avez effectuée (de manière que je puisse tester qu'elle est bien prise en compte).

Chapitre 5

Lire et traiter des informations via un page Web : *formulaires*, PHP, MySQL

Ce chapitre présente une manière pour les pages Web d'interagir avec les personnes qui les consultent. Au lieu de se contenter d'afficher de l'information, la page peut en recevoir et modifier son comportement en conséquence. Elle introduit également le langage PHP qui est la façon la plus courante de traiter cette information coté serveur et très brièvement le système de gestion de bases de données MySQL.

5.1 Les formulaires

HTML contient un dispositif spécifique pour envoyer de l'information depuis un navigateur vers le serveur Web. Il s'agit des *formulaires*.

Le formulaire est encadré par une balise `<form>` et contient du HTML ordinaire mélangé avec des champs que le navigateur invite à remplir. Quand on envoie le formulaire, cela va se traduire par une requête HTTP.

5.1.1 Un exemple simple

La figure 5.1 contient un exemple de page avec un formulaire. Celui-ci permet juste d'entrer un nom et quelques goûts. Lire son contenu et observer ce que ça donne : cet exemple est utilisé dans la suite du chapitre.

```
<!DOCTYPE html>
<html>
  <meta charset="utf-8">
  <head>
    <title>Test d'un formulaire</title>
  </head>
  <body>
    <p>Ceci est une page qui soumet un formulaire (avec la méthode GET)
    sur une adresse pensée pour être écoutée avec netcat. On lance la
    commande netcat sur la même machine que celle où tourne le navigateur
    avec <code>netcat -l 8765</code>. L'action indiquée comme attribut
    de la forme est <code>http://localhost:8765/clic</code> ; elle va
    conduire le navigateur à ce connecter à ce netcat.

      <form method="get"
            action="http://localhost:8765/clic"
            style="background-color:pink;">
        <p>Ceci est le début du formulaire ; on peut y mettre
          le HTML usuel.

        <p>Le formulaire contient une zone où on pourra
          entrer une chaine de caractères et une autre où il
          faut effectuer un choix.

        <p>Nom : <input type="text" name="chaine1" />

        <p>Vous aimez
          <input type="checkbox" name="ail">l'ail
          <input type="checkbox" name="oignon">l'oignon
          <input type="checkbox" name="poireau">le poireau
          <input type="checkbox" name="ciboulette">la ciboulette

        <p>Vous préférez
          <input type="radio" name="pref" value="ail" />l'ail
          <input type="radio" name="pref" value="oignon" />l'oignon
          <input type="radio" name="pref" value="poireau" />le poireau
          <input type="radio" name="pref" value="ciboulette" />la ciboulette

        <p><input type="submit">

        <p>Il y a beaucoup d'autres types d'input !
      </form>
      Ce texte suit le formulaire.
  </body>
</html>
```

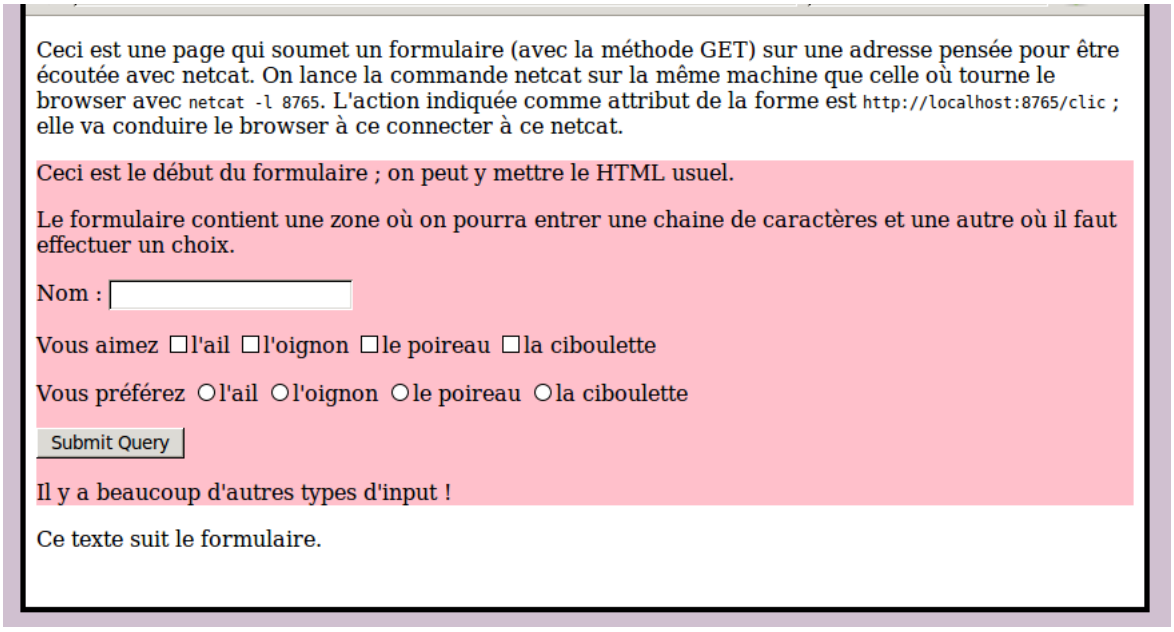


FIGURE 5.1 – Le HTML d'une page qui contient un formulaire et la manière dont le résultat est affiché par Firefox.

5.1.2 Les attributs de la balise `<form>`

La balise `<form>` contient deux attributs particuliers. L'un d'entre eux est l'adresse à laquelle envoyer le contenu du formulaire et l'autre la manière d'envoyer ces données.

On indique l'adresse à laquelle envoyer les données avec l'attribut `action`. [Si on ne met pas d'attribut action, c'est la même page qui est interrogée.]

On peut soumettre les données avec deux méthodes du protocole HTTP qui sont `GET` (la méthode ordinaire de récupération des pages Web) et `POST`. La différence entre les deux méthodes est explicitée dans la section suivante.

5.1.3 Les balises `<input>`

À l'intérieur du formulaire, les champs simples à remplir sont indiqués avec la balise `<input />`. La balise contient toujours un attribut `name=` qui permettra au serveur de savoir dans quel champs se trouve la valeur transmise. D'autres attributs indiquent de quelle genre d'entrée il s'agit.

`<input type="text">`

Les champs d'entrée de texte s'indiquent avec un `type="text"` pour les textes courts (par exemple un nom, une adresse ou un numéro de téléphone).

Par défaut, la zone affiche 20 caractères saisis. Vérifiez sur l'exemple (dans le champs *Nom*) qu'on peut saisir plus de 20 caractères ; le contenu défile dans ce cas.

Vérifier en modifiant l'exemple qu'on peut spécifier une valeur par défaut dans le champs avec l'attribut `value=` et une autre taille avec `width`. De nombreux autres attributs permettent de limiter les valeurs entrées et/ou de forcer ces valeurs à certains types de données.

`<input type="checkbox">` et `<input type="radio">`

Plusieurs sortes de cases à cocher sont possibles ; l'exemple utilise des cases du type *checkbox* et *radio*.

Les cases du type *checkbox* (indiquées avec `type="checkbox"`) peuvent être cochées ou pas individuellement, comme vous pouvez le vérifier sur l'exemple. Une case cochée sera transmise comme contenant la valeur `on` (par défaut) et pas transmise si elle n'est pas cochée.

Avec les cases du type *radio* (indiquées avec `type=radio`) un seul choix est possible. [1] Quand on coche un choix, cela efface le choix précédent. L'attribut

1. Les boutons *radio* s'appellent ainsi parce que (je crois) c'est de cette manière que fonctionnaient les boutons qui permettaient de choisir la bande de fréquence captée par les radios anciennes, comme les grandes ondes ou les ondes courtes.

`name=` permet aussi au navigateur de savoir quels boutons forment un groupe. L'attribut `value=` sert à déterminer la valeur à transmettre en fonction du bouton coché.

```
<input type="submit">
```

Le formulaire sera envoyé quand on cliquera sur le bouton.

D'autres types d'input

En vrac :
— le type `password` se comporte comme le `text` mais sans écho du texte tapé.
— le type `file` permet de choisir un fichier (un bouton permet de sélectionner le fichier à envoyer).
— le type `hidden` permet d'avoir quelque chose qu'on ne voit pas depuis le navigateur mais qui est transmis tout de même.

La version 5 de HTML a aussi ajouté des types de données usuels qui permettent au navigateur de compléter et de valider les données avant transmission. On peut spécifier une couleur, une date (ou des morceaux de date, comme un mois ou un jour de la semaine), un email, un nombre (entier ou flottant, éventuellement compris entre des valeurs limites), un numéro de téléphone etc. Le navigateur est en mesure de tester que les données correspondent bien au format attendu ; éventuellement de compléter ou de corriger les données.

5.1.4 D'autres façon d'entrer des données

La balise `<textarea>` pour avoir du texte qui s'affiche sur plusieurs lignes.

La balise `<select>` qui contient des balises `<option>` pour avoir des menus drop-down.

Les boutons peuvent avoir trois types :
— le type `submit` (utilisé dans l'exemple) sert à envoyer le formulaire
— le type `reset` sert à effacer les données déjà saisies dans le formulaire
— le type `button` donne un bouton qui ne fait rien ; (on verra plus tard comment on peut associer ce bouton apparemment inactif avec du code présent dans la page pour exécution par le navigateur).

5.2 `GET` et `POST`

Pour mémoire, l'attribut `action=` de la balise `<form>`, indique l'URL à laquelle soumettre le contenu du formulaire. (Si l'attribut est absent, c'est la page qui contient le formulaire qu'il utilise.) Dans notre exemple, j'ai indiqué l'adresse

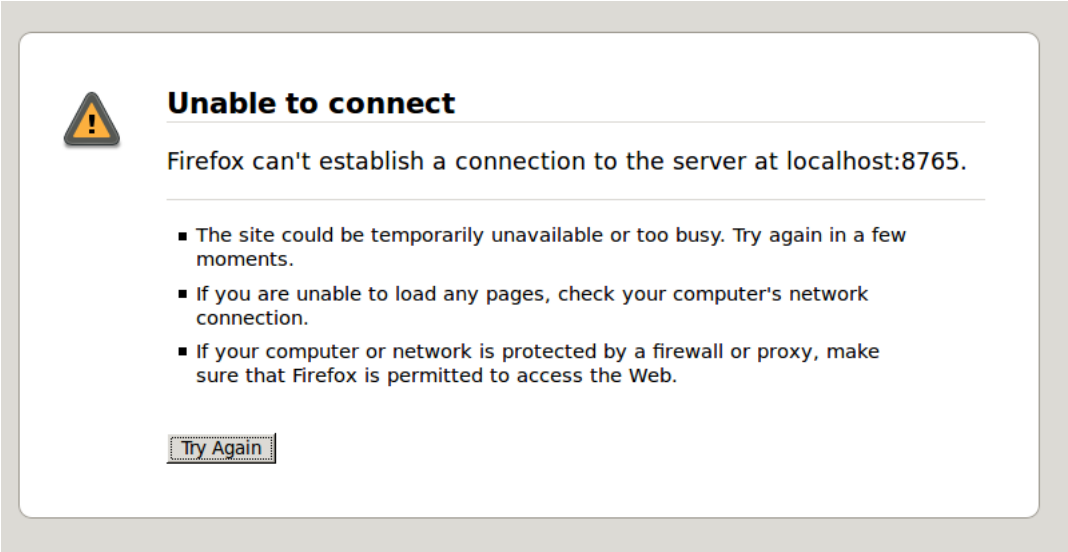


FIGURE 5.2 – Le message d'erreur avec lequel Firefox m'indique qu'il n'a pas réussi à joindre le serveur (sur la machine `localhost` et le port numéro 8765).

`http://localhost:8765/clic` qui indique donc le serveur de la machine courante qui attend les connexions sur le port 8765.

5.2.1 Soumission avec `GET`

L'exemple utilise la méthode `GET`, indiqué par l'attribut `method=` dans la balise `<form>`. Quand on clique sur le bouton `Submit query` après avoir rempli quelques champs, le navigateur essaie de transmettre les données.

Pas de connexion

Il n'y a (normalement) pas de processus en attente de connexions sur le port utilisé (le numéro 8765) et on obtient un message d'erreur (figure 5.2). Notez que le contenu du formulaire est visible dans la barre de navigation.

Réception via `netcat`

L'idée est d'utiliser la commande `netcat` pour lire les données envoyées sur ce port et ainsi observer l'échange des données entre le navigateur et le serveur. Dans un terminal, on entre donc la commande `netcat -l 8765` : on a maintenant un processus qui attend une connexion sur le port, prêt à lire et afficher les données qu'il y reçoit dans le terminal. (Le `netcat` attend sans rien afficher tant qu'il ne se passe rien.)

On valide de nouveau le contenu du formulaire. Au moment où on presse le bouton `submit`, le navigateur tente de nouveau de se connecter avec le serveur de la machine courante sur le port 8765 : cette fois-ci il trouve cette commande `netcat` qu'on a lancé ; il lui transmet les données et `netcat` les affiche sur le terminal :

```
$ netcat -l 8765
GET /clic?chaine1=Moi&ail=on&oignon=on&ciboulette=on&pref=ciboulette HTTP/1.1
Host: localhost:8765
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:28.0) Gecko/20100101 Firefox/28.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Noter la façon dont `Firefox` affiche, en bas à gauche de la fenêtre, le message *Waiting for localhost...* : il indique qu'il a réussi à se connecter et à envoyer des données ; il attend maintenant une réponse.

Pour ne pas faire attendre `Firefox`, on va lui répondre immédiatement en tapant par exemple à `netcat` :

```
Il n'y a pas de navigateur à l'URL que vous avez demandé.
Veuillez consulter un centre de renseignements.
```

Constater la manière dont le message dans la fenêtre Firefox change dès qu'on a terminé de taper la première ligne (et donc que `netcat` la reçoit et la transmet à `Firefox`) : c'est maintenant *Transferring data from localhost ...* et la page avec le formulaire est effacée.

Constater que quand on tape un contrôle-D, `netcat` (qui l'interprète comme une fin de fichier) coupe la connexion avec le navigateur. Celui-ci affiche, du mieux qu'il peut, le contenu de la page. Notre réponse via netcat n'est *pas* une façon valide de répondre à une requête `HTTP` ; `firefox` affiche les lignes qu'on a tapées comme du texte brut, faute de mieux.

Analyse de la requête

Le requête que `Firefox` a transmis quand on a validé le formulaire correspond à un message HTTP valide, elle : la première ligne contient l'objet demandé, les lignes suivantes ont des paramètres ou des précisions complémentaires.

La ligne `GET` La première ligne correspond à une demande ordinaire de page Web comme on l'a vu au chapitre précédent. La demande seule porterait sur la page nommée `clic` avec le protocole `HTTP`, version 1.1 :

```
GET /clic HTTP/1.1
```

Parce qu'on utilise la méthode `GET`, l'URL a été enrichi de toutes les valeurs du formulaire après un point d'interrogation :

— Le champs texte, qui porte le nom `chaine1`, contient la valeur `Moi`.
— Les boîtes qui portent les noms `ail`, `oignon` et `ciboulette` sont cochées, comme l'indique leur valeur `on`. (Celle qui porte le nom `poireau` n'est pas cochée et n'apparaît pas.)
— Parmi les boutons radios qui portent le nom `pref`, c'est celui qui vaut ciboulette qui est coché.

Il est intéressant de noter ce qui se passe quand les champs contiennent des valeurs qui n'appartiennent pas à l'ASCII ; par exemple, si dans la zone de texte on remplace *Moi* par *Moi-même* avec son accent circonflexe.

De la même manière : le point d'interrogation sert à introduire les valeurs des champs du formulaire ; l'esperluette sert à séparer les champs ; le égal sert à associer le nom d'un champs avec sa valeur. On peut (on doit ?) se demander ce qui se passe quand un champs texte contient ces caractères de ponctuation ? Et quand c'est le nom du champs qui le contient ? Et quand c'est l'URL elle-même ?

Finalement, on peut (on doit ?) se demander comment les choses se passent quand on a une grosse quantité de données à envoyer. Par exemple pour envoyer une image qui occupe quelques mégaoctets, Firefox envoie-t-il une première ligne de quelques mégaoctets ? Quels serveurs les acceptent ?

Dans la catégorie des questions stupides qu'il est sain de se poser : que se passe-t-il quand plusieurs zones portent le même nom ? par exemple, deux zones de texte différentes qui ont toutes les deux l'attribut `name="chaîne"`. [Hypothèses vraisemblables : une seule est affichée ; elles sont transmises toutes les deux (dans quel ordre ?) ; une seule est transmise (toujours la première ou toujours la deuxième ?).]

Les paramètres apparaissent dans la barre de navigation. Ça présente un problème de confidentialité puisque ce sera stocké dans les caches. Ça permet à l'utilisateur de les stocker dans un marque-page (un *bookmark*).

Les autres lignes Je décris sommairement toutes les autres lignes de la requête :

— `Host:` (obligatoire avec `HTTP 1.1`) indique sur quel site porte la requête (pour le cas où le serveur gérerait plusieurs sites).
— `User-Agent:` identifie le navigateur utilisé avec son numéro de version.
— `Accept:` indique les formats de données acceptés par firefox. Il préfère du texte en HTML ou du XHTML ou du XML mais il est prêt à accepter n'importe quoi (comme l'indique le `*/*`).
— `Accept-Language:` contient les langues à utiliser pour les essages. (C'est bien de demander des messages en anglais : ceux qui lisent bien l'anglais s'évitent les erreurs de traduction ; ça peut servir d'entraînement à la lecture de l'anglais pour les autres.)
— `Accept-Encoding:` indique comment les données peuvent être transmises (elles peuvent être compressées à la façon de la commande `gzip`).

— `Connection:` propose au client d'utiliser de nouveau la connexion pour transmettre une nouvelle requête quand celle-ci sera terminée.

5.2.2 Soumission avec `POST`

On peut modifier l'exemple pour placer dans la balise `<form>` l'attribut `method="post"` à la place de `method="get"`. Dans ce cas, c'est le verbe `POST` qui va être employé par le navigateur pour envoyer le contenu du formulaire au serveur.

Constater qu'en l'absence de `netcat`, le message d'erreur est identique mais on ne voit pas le contenu du formulaire dans la barre de navigation.

Constater qu'avec `netcat`, on reçoit quelque chose comme :

```
$ netcat -l 8765
POST /clic HTTP/1.1
Host: localhost:8765
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:28.0) Gecko/20100101 Firefox/28.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 58

chaine1=moi&ail=on&oignon=on&ciboulette=on&pref=ciboulette
```

Deux choses ont changé dans l'en-tête : la première ligne contient maintenant le verbe POST, sans le contenu du formulaire ; le contenu du formulaire est passé de la même manière que le serveur fournit des pages au navigateur.

Les compléments de la requête contiennent maintenant une ligne `Content-Type:` qui indique qu'il y a des données et la façon dont elles sont codées ; et une ligne `Content-Length:` qui indique que ces données occupent 58 octets.

Après la requête, il y a une ligne vide et le contenu des données, dans le même format que celui utilisé par la méthode `GET` : des coupes *nom=valeur* séparés par des esperluettes (sur 58 octets).

La méthode POST est à utiliser quand on ne veut pas que les valeurs apparaissent dans l'URL (leur présence dans les caches poserait un problème de confidentialité) ; quand elles sont trop volumineuses pour être transmises sur une seule ligne (quelques kilooctets). Elle ne permet pas de stocker les données transmises dans l'URL et on ne peut donc pas les conserver dans un bookmark.

5.3 Un peu de PHP

Maintenant que vous savez l'essentiel sur la construction et l'envoi des formulaires par le navigateur, on va s'intéresser un peu à la manière dont ces données peuvent être traitées du coté du serveur, essentiellement en survolant le langage de programmation PHP.

5.3.1 Fabriquer des pages au vol

On a vu qu'un serveur qui reçoit une requête `HTTP` peut y répondre avec le contenu d'un fichier (qui contient souvent du HTML) ; un inconvénient est que chaque URL correspond à une page : ça manque de variété où bien il y a beaucoup de pages.

Une autre possibilité est que l'URL fasse référence à un le fichier qui contient un *programme* ; le serveur fait tourner ce programme, qui produit du HTML avec lequel le serveur répond à la requête.

PHP joint les deux : à priori le fichier contient du HTML mais on peut placer des bouts de programmes au milieu.

5.3.2 Une très brève histoire de PHP

Quand il est apparu vers la fin des années 1990, PHP était un petit hack pratique pour faire des pages Web fabriquées au vol par le serveur. Son nom était l'acronyme de *Personal Home Page*. Il permettait d'éviter les complications des systèmes concurrents (notamment celles du langage Perl).

Depuis PHP a bien grandi ; il s'est enrichi de nombreuses bibliothèques ; de très nombreux sites Web très importants sont construits sur son utilisation (dont Facebook et WordPress) ; son nom a été repensé et signifie maintenant *PHP : Hypertext Processor*. [2]

Pour bien se servir de PHP, il faut avoir lu et compris des milliers de pages de documentation, notamment pour la description des nombreuses fonctions de bibliothèque. Pour s'en servir d'une façon utile, un petit peu d'expérience avec un langage de script (par exemple Python) suffit.

5.3.3 Installer PHP

La manière normale d'installer PHP est de l'activer sur un serveur Web. On va ici le faire sur le serveur Apache installé au chapitre 3.

2. Je crois me souvenir que quand PHP a commencé à être largement utilisé, le premier P a été interprété comme *Professional* ; maintenant on est arrivé à un acronyme récursif.

Vérifier l'absence de PHP

On prend un petit fichier et on lui donne un nom qui se termine par `.php`, par exemple `premier.php` ; on le place dans le répertoire `public_html` qu'on a activé en installant Apache ; on tente d'y accéder à l'URL `http://localhost/~jm/premier.php` (en adaptant bien sur le nom d'utilisateur : `jm` est à remplacer par votre nom de login).

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <p>Du HTML ordinaire avant.
    <?php echo 'Quelque chose d'imprimé par php.' ?>
    <p> Du HTML ordinaire après.
  </body>
</html>
```

Ce qui se trouve dans la balise `<?php ···?>` est ignoré et on voit :

```
Du html ordinaire avant.
Du html ordinaire après.
```

S'il n'y pas moyen de se connecter au serveur, vérifier qu'on a installé Apache comme dans le chapitre précédent.

S'il n'y a pas moyen d'afficher la page (1) vérifier le nom d'utilisateur (par exemple avec la commande `whoami`) et vérifier l'existence du fichier `~jm/public_html/premier.php`.

Si on voit *Quelque chose d'imprimé par php.* sur la page, c'est que PHP est déja activé sur le serveur.

Activer PHP sur le serveur

Pour Ubuntu, avec la commande `sudo apt-get install php5`. Après confirmation, cela va installer PHP et ce qui est nécessaire pour l'intégrer dans Apache. Cela relance aussi le serveur Apache.

Le fichier de configuration de PHP se trouve dans `/etc/php5/apache2/php.ini`. C'est un fichier texte avec de nombreux commentaires. Y jeter un coup d'œil pour se faire une idée de ce qu'on peut contrôler comme options. [voir la section suivante]

Activer PHP dans `public_html`

La configuration par défaut interdit l'exécution des scripts PHP qui se trouvent dans les répertoires `public_html`. Il faut donc l'autoriser explicitement en mo-

difiant le fichier de configuration de PHP pour Apache, qui lui se trouve dans `/etc/apache2/mods-enables/php5.conf`. Vers la ligne 20, on trouve les lignes

```
# Running PHP scripts in user directories is disabled by default
#
# To re-enable PHP in user directories comment the following lines
# (from <IfModule ...> to </IfModule>.) Do NOT set it to On as it
# prevents .htaccess files from disabling it.
<IfModule mod_userdir.c>
    <Directory /home/*/public_html>
        php_admin_flag engine Off
    </Directory>
</IfModule>
```

Comme l'indiquent les commentaires, il faut commenter les quatre lignes pour que les scripts PHP fonctionnent dans `public_html`. Je vous conseille aussi d'y ajouter un commentaire qui vous permettra de vous souvenir que vous avez modifié ces lignes, avec la date de la modification.

5.3.4 PHP à la hache

Comme dans le premier exemple : la façon naturelle de procéder est de parsemer le fichier qui contient du HTML avec du code PHP (entre `<?php` et `?>`). Dans les cas simples, le code PHP fabrique du HTML, souvent en appelant `echo` avec une chaîne de caractères comme argument.

PHP est un langage de script. Il emprunte sa syntaxe et une bonne partie de sa sémantique à de nombreux autres langages ; au C, au Shell, à Perl.

Mon ambition est de placer ce qui est nécessaire pour survivre dans un environnement qui utilise PHP : fabriquer des pages simples (notamment pour analyser les formulaires) ; lire des programmes en PHP écrits par d'autres. Le cadre du cours est trop restreint pour espérer devenir un(e) expert(e) PHP juste avec ce qui se trouve ici.

Les commentaires

Comme en C (avec `/* ... */`), en C++ (avec `//` jusqu'à la fin de la ligne) ou en shell (avec `#` jusqu'à la fin de la ligne).

Code global et fonctions

On peut avoir du code en ligne (des instructions exécutées quand elles sont lues) et des déclarations de fonction (le code n'est exécuté que quand la fonction est appelée). Une déclaration de fonction stupide pourrait être par exemple

```
function ajouter($a, $b){
```

```
  return $a + $b;
}
```

Les variables

Les variables ont toutes un nom qui commence par un dollar. On ne les déclare normalement pas. Elles ont un type, qui n'est normalement pas déclaré explicitement mais positionné à la première utilisation. PHP a des règles de conversion implicites qui permettent jusqu'à un certain point d'ignorer les questions de conversion ; par exemple `0.0 == "0"` compare un nombre en virgule flottante et une chaîne de caractère (ce qui est différent) mais les conversions permettent de répondre *vrai*.

Il y a des variables locales et des variables globales.

Une variable utilisée pour la première fois dans une fonction est une variable locale ; on ne la déclare pas. Elle n'est bien sûr pas accessible en dehors de la fonction. [3]

Une variable utilisée pour la première fois en dehors d'une fonction est une variable globale ; elle existe en un seul exemplaire et conserve sa valeur dans toute la page.

Une chose un peu bizarre : *on ne peut pas accéder normalement à une variable globale dans le code d'une fonction.* Il faut l'avoir déclarée dans la fonction avec `global`.

```
$x = 1;

function erreur(){
  global $x;
  $x += 1;
}
```

Il s'agit bien sûr de se prémunir contre la modification par inattention d'une variable globale dont on aurait ignoré ou oublié l'existence, en utilisant ce qu'on pense être une variable locale.

Le mot clef `static` joue le même rôle qu'en C. On peut voir ça comme une variable, locale à une fonction, qui conserve sa valeur entre les différents appels de la fonction. On peut aussi voir ça comme une variable globale à laquelle on ne peut accéder que dans la fonction où elle est déclarée.

Les différents types de données sont

— les nombres entiers et les nombres à virgules flottantes qui fonctionnent à peu près comme en C.

— les chaînes de caractères qui sont encadrées par des simples quotes (`'chaine'`) ou des doubles quotes (`"chaine"`). Comme en shell, les références à des

3. On peut fabriquer des *fermetures* qui permettent un accès aux variables locales en dehors de la fonction mais le sujet sort vraiment du cadre d'un cours de première année.

variables dans les chaînes de caractères sont remplacées par leurs valeurs. Par exemple après

```
$x = "l'un";
$y = "l'autre";
$paire = "$x ou $y";
```

la variable `$paire` vaut `"l'un ou l'autre"`.

Il y a des types un peu moins communs dont on peut sans doute ignorer les détails dans un premier temps :

- les booléens, qui ne peuvent prendre que les valeurs `true` et `false`.
- le type `NULL` (qui ne peut prendre que la valeur `NULL`).
- les tableaux qu'on peut fabriquer avec quelque chose comme `array(`el_0`, el`$_1$`, el`$_2$`)` pour les tableaux indexés et avec `array("one"=>"un", "two"=>"deux", "three"=>"trois")` pour les tableaux associatifs. On en référence les éléments de la manière usuelle des langages de programmation qui descendent du C : `$toto[0]` ou `$tata['two']`
- les objets pour lesquels on fabrique des classes avec `Class nom { ... };` et dont on fabrique de nouvelles instances avec `$var = new nom`.

Pour mettre les programmes au point, il est utile de connaître la fonction `var_dump` ; utilisée avec `var_dump($var)` elle permet de connaître la valeur *et le type* d'une variable.

Constantes, includes

On peut définir des constantes avec `define(nom, valeur);` et inclure des fichiers entiers avec `include 'nom-du-fichier';`. Ces constructions jouent le même rôle que les `#define` et `#include` du pré-processeur C.

Je souligne que les constantes définies de cette manière ont un nom qui ne commence *pas* par un dollar, à la différence des noms de variables.

Ceux qui veulent tout savoir apprendront avec plaisir que `define` peut prendre un troisième argument pour indiquer si majuscules et minuscules sont équivalentes ; qu'il existe un `require` qui fonctionne comme include, sauf en cas d'erreur.

Expressions

Les expressions se construisent de la façon usuelle des langages de programmation inspirés du C. À noter :

- la concaténation de chaînes de caractères s'effectue avec un opérateur spécifique `'.'` (un point tout seul) et non pas un +.
- il y a des opérateurs `===` et `!==` qui vérifient si deux objets ont même type et même valeur. Par exemple, `0 == "0"` est vrai puisqu'on peut convertir la valeur de l'un dans celle de l'autre mais `0 === 0.0` est faux puisque les deux expressions sont de types différents.
- les deux opérateurs `!=` et `<>` sont équivalents.

Les structures de contrôle

Les structures de contrôles sont les mêmes que celles qu'on trouve dans les langages qui descendent du C : if, if ... else, while, do...while, for(init ; test ; maj). Il est notable que le `switch` fonctionne d'une façon aussi étrange qu'en C.

Il y a une structure de contrôle supplémentaire qui permet d'itérer sur les éléments d'un tableau : `foreach ($array as $value){ ... }`. Le corps de la boucle sera exécuté pour chaque élément du tableau ; la variable `$value` vaudra la valeur de cet élément. Pour les tableaux associatifs, on peut aussi employer `foreach($array as $key => $value)`

Les fonctions

```
function foo($arg1, $arg2, $arg3){ ... }
function bar($arg = "valeur par défaut"){ ... }
```

Les super-globales

PHP dispose de quelques variables dites *super-globales* qui permettent d'accéder, à travers un tableau, à des données importantes.

La super-globale `$GLOBALS` permet d'accéder aux variables globales sans avoir besoin de prêter attention au contexte local ou global d'utilisation. `$GLOBALS['x']` permet de consulter et/ou de modifier la valeur de la variable globale `$x`.

Nous sommes particulièrement intéressé par les super-globales `$_GET` et `$_POST` avec lesquelles on peut récupérer les valeurs entrées dans un formulaire et transmis avec les méthodes `GET` et `POST`.

Ceux qui veulent tout savoir s'intéresseront aussi à
- `$_SERVER` qui contient des paramètres du serveur et du client.
- `$_REQUEST`
- `$_FILES`
- `$_ENV`
- `$_COOKIE`
- `$_SESSION`

Un exemple simple d'utilisation des données d'un formulaire

Avec PHP, on peut facilement récupérer les données du formulaire en utilisant les tableaux `$_POST` ou `$_GET` (suivant la méthode utilisée).

Un exemple simple de récupération des données :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemple de réception d'un formulaire</title>
```

```
    <meta charset="utf-8">
  </head>
  <body>

    <strong><large><center>Réception</center></large></strong>

    <p>C'est ici qu'arrivent les données de <code>form.php</code>.
    <?php
         foreach($_GET as $clef => $valeur)
           echo "<p>GET['$clef'] vaut '$valeur'";
     ?>
    <p>Fin des données du formulaire PHP.

  </body>
</html>
```

Modifier l'attribut `action=` de la balise `<form>` de la page `form.php` utilisée dans le chapitre pour le faire pointer sur cette page.

Constater que ce petit script imprime correctement les champs remplis.

Constater qu'on peut recharger la page sans problème.

Constater qu'on peut modifier la valeur des paramètres dans la barre de navigation avant de recharger la page : la modification est prise en compte.

Constater qu'on peut ajouter ou retirer des paramètres dans la barre de navigation. Les champs sont transmis comme édité, y compris quand on utilise un nom de champs qui n'apparaît pas dans le formulaire.

Après avoir modifié l'attribut `method=` pour y mettre `post`, constater qu'il n'y a plus rien dans le tableau `$_GET$` quand on soumet le formulaire.

Éventuellement, modifier le script pour afficher le contenu du tableau `$_POST$`

Constater que Firefox demande confirmation quand on tente de recharger la page, parce qu'il va renvoyer les données sans qu'on en soit nécessairement conscient.

Un autre exemple simple d'utilisation des données d'un formulaire

Histoire de voir un peu de code PHP en action, voici une autre façon d'utiliser les données du formulaire. J'y ai placé l'initialisation d'un tableau et une boucle `for` pour compter le nombre de choses cochées dans le formulaire.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemple de réception d'un formulaire</title>
    <meta charset="utf-8">
  </head>
```

```
  <body>

    <strong><large><center>Réception</center></large></strong>

    <p>C'est ici qu'arrivent les données du formulaire.

      <?php
         $nom = $_GET['chaine1'];
         $nom != NULL || $nom = 'anonyme';

         # Une façon de compter les choses cochées
         $compte = 0;
         for($i = 0; $i < count($liste); $i++)
           if ($_GET[$liste[$i]] == "on")
             $compte += 1;

         echo "<p>Bonjour, $nom";

         echo "<p>Compte vaut " . $compte . ".

         if ($compte == count($liste))
           echo "<p>Vous aimez tout !";
         else if ($compte == 0)
           echo "<p>Vous êtes vraiment difficile !";
         else
           echo "<p>Rien à dire.";
      ?>
  </body>
</html>
```

Nettoyer les données reçues

Quand on conçoit une page Web qui reçoit des données, il faut être conscient que ces données ne sont pas nécessairement celles qu'on a prévu de recevoir. Notamment, une page pour traiter un formulaire doit être prête à recevoir *n'importe quoi* comme valeur dans les champs du formulaire. En effet, on peut facilement choisir les données qui seront transmises quand un formulaire utilise la méthode `GET` (il suffit d'éditer le contenu de la barre de navigation). Avec la méthode `POST`, c'est un tout petit plus difficile mais à peine ; par exemple on peut aisément construire une requête dans un fichier avec son éditeur de texte usuel puis l'envoyer au serveur Web avec `netcat`.

Pour cette raison, les données transmises peuvent être problématiques : celui qui consulte la page peut parfois introduire du programme en guise de réponses dans le formulaire et ce programme pourra être exécuté par le serveur. Par exemple, on peut recharger la page qui dumpait les données du formu-

laire (section 5.3.4) mais en remplaçant dans la barre de navigation l'URL par `dumper.php?bidon=<script>alert('Boum !')</script>`

Constater ce qui se passe dans ce cas (apparition d'une fenêtre pop-up qui affiche le message *Boum !*) [4]. En envoyant des données bien choisies, on a conduit PHP à effectuer quelque chose d'imprévu par le concepteur de la page.

Il est raisonnable de *toujours* traiter les données reçues via les formulaires pour éviter ce genre de problème. Une façon de procéder est d'utiliser une fonction qui va ramollir ce qui peut poser problème en le codant comme du HTML ; la page de 5.3.4 devient

```
<?php
   # Inspiré de w3schools.net
   function nettoyer($x){
     if ($x){
        $x = trim($x);
        $x = stripslashes($x);
        $x = htmlspecialchars($x);
     }
     return $x;
   }

   foreach($_GET as $clef => $valeur){
     $sclef = nettoyer($clef);
     echo "<p>GET['$sclef'] vaut '" . nettoyer($valeur) . "'";
   }
?>
```

Ici, la fonction `nettoyer` encode les caractères spéciaux à la façon de HTML. Le `<script>` devient `<script\>` qui ne pose pas de problème.

Pour un exemple d'injection plus intéressant que notre petite explosion virtuelle : `https://xkcd.com/327/`.

Pas développés

Il existe un énorme corps de code déjà développé en PHP ; une bonne manière de faire des progrès en PHP est (1) d'apprendre le plus possible de documentation par cœur (savoir que quelque chose existe dans une bibliothèque est déjà fort utile) (2) de le mettre en œuvre dans du code (ce qui permet de vérifier que sa compréhension de la documentation est correcte) et (3) d'étudier la façon dont la bibliothèque est réalisée (c'est souvent une bonne idée de lire du code écrit par des experts).

4. L'état de la page au moment où la pop-up apparaît indique bien à quel moment le script est activé. Comparer avec `dumper.php?<script>alert('Boum !')</script>=bidon`

Creuser un gouffre de sécurité, nuire aux performances

Il y a moyen d'utiliser d'autres langages du coté du serveur à la place de PHP. Pour la plupart des choses vraisemblables, il existe un module pour le serveur Apache qui permet de l'utiliser d'une façon raisonnablement sécurisée avec des performances correctes.

Si on ne s'inquiète ni pour la sécurité, ni pour les performances (disons : pour un prototype qui ne tournera que sur un Intranet isolé sans données confidentielles), il peut être utile de savoir qu'il existe dans PHP une fonction de bibliothèque appelée `system` qui permet d'exécuter une commande shell. Elle permet de se servir de PHP comme d'une passerelle pour accéder à un autre environnement. Pour un script shell, `<? system("sh toto.sh"); ?>` est presque raisonnable. Pour un programme C `<? system("gcc toto.c"); system("./a.out"); ?>` fonctionnera au moins un peu (à condition qu'on ne s'inquiète pas des compilations inutiles et/ou des accès multiples à la même page).

5.4 Un peu de MySQL

Pour terminer le chapitre sur les formulaires, comment les construire et comment les traiter, nous allons parler un peu de bases de données. On va utiliser le SGBD (Système de Gestion de Bases de Données) MySQL. Il s'agit probablement du SGBD le plus largement utilisé dans l'univers du Web. Il y a une forme commune d'installation de serveurs web appelée LAMP comme *Linux, Apache, MySQL et PHP* qui en sont les constituants essentiels ; le cours aura permis de survoler tous ses éléments.

5.4.1 Installation de MySQL

On peut installer un serveur MySQL sous Ubuntu avec l'utilitaire usuel ; la commande doit être `apt-get install mysql-server` (avec un `sudo` si nécessaire). On peut aussi installer l'interface entre PHP et MySQL avec `apt-get install php5-mysql`.

Au moment de l'installation, il va être nécessaire de fournir un mot de passe pour le serveur MySQL. *Éviter de donner un mot de passe trop simple.*

La commande `apt-get` permet d'installer plusieurs packages à la fois et la façon la plus naturelle de procéder est d'installer d'un coup Apache, PHP et MySQL avec

```
apt-get install apache2 php5 mysql-server libapache2-mod-php5 php5-mysql
```

5.4.2 Le serveur MySQL

Le serveur MySQL est un processus qui gère des bases de données sur une machine (ici `localhost`). Il est lancé au démarrage de la machine et attend des

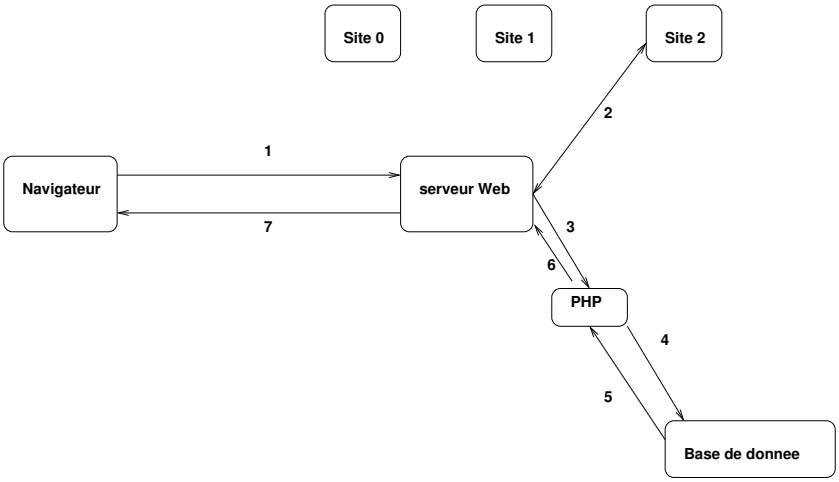


FIGURE 5.3 – Le navigateur client (Firefox) transmet une URL au serveur Web (1) ; le serveur Web (Apache) extrait la page des données du site concerné par la demande (2) et la transmet à l'interprète PHP (3). L'interprète PHP exécute le code contenu dans la page, le cas échéant en interaction avec un serveur de bases de données (MySQL) (4 et 5). L'exécution du code PHP produit en définitive une page HTML (6) que le serveur transmet au client (7).

connexions de clients (de la même manière que le serveur Apache). [5]

La façon ordinaire de consulter ou de modifier les données stockées dans les bases passe par un processus client qui se connecte au serveur et lui transmet des commandes. Le serveur envoie en retour les résultats au client. Comme la connexion entre les clients et le serveur peut se faire à travers une connexion TCP, rien n'oblige les clients à tourner sur la même machine que le serveur : ils peuvent se trouver n'importe où sur Internet. [6]

Une fois que ces logiciels sont installés, on a en première approximation quatre processus qui coopèrent quand on consulte une page Web (figure 5.3).

5. Pour vérifier la présence du serveur MySQL, utiliser la commande `ps` pour vérifier la présence d'un processus nommé `mysqld`. Par exemple `ps ax | grep mysql` donne quelque chose comme

```
14054 ? Ssl 1:22 /usr/sbin/mysqld
```

qui indique la présence d'un processus mysqld (qui a consommé une minute et 22 secondes de temps CPU) : c'est le serveur MySQL de la machine.

6. Parce que le serveur MySQL est accessible de tout l'Internet, il est recommandé de ne pas choisir un mot de passe trop simple pour le serveur. Le fait d'être protégé derrière une box qui fait du NAT n'est pas une protection suffisante, sauf à s'interdire tout usage du Wifi et d'IPv6. Quand j'écris ces lignes, le serveur MySQL n'accepte par défaut que les connexions depuis la machine locale mais il n'y a qu'une ligne à changer dans le fichier de configuration pour que tout soit ouvert.

5.4.3 Vérification de l'installation de MySQL

Pour vérifier que les choses fonctionnent, on va créer un trou de sécurité puis le reboucher.

Le shell MySQL

Il existe une commande en ligne qui permet de se connecter simplement avec le serveur. Elle s'appelle simplement `mysql`.

On peut se connecter au serveur avec la commande `mysql -u root -p`. L'option `-u root` indique qu'on se connecte en tant qu'utilisateur `root`. L'option `-p` indique qu'on est prêt à entrer un mot de passe.

Après un message d'accueil, `mysql` imprime un prompteur (une invite) `mysql>`; on peut y taper des commandes. Tapons

```
GRANT ALL PRIVILEGES ON *.* TO 'tata'@'localhost';
```

puis

```
FLUSH PRIVILEGES;
```

Il y a maintenant un `tata` qui peut faire à peu près n'importe quoi sur les bases de données de la machine quand il se connecte depuis `localhost`.

Quitter la commande (avec les touches Contrôle-D ou la commande `quit`) et se reconnecter en tant que `tata` avec `mysql -u tata`; noter qu'on n'a pas besoin de mot de passe. Créer une base de données avec `CREATE DATABASE 'bidon';` . Constater que tata n'a pas le droit de modifier les privilèges avec `REVOKE ALL PRIVILEGES ON *.* FROM 'tata'@'localhost';` et provoque une erreur `Access denied`.

Se connecter de nouveau en tant que root, révoquer les privilèges de tata. Se connecter en tant que tata, constater qu'on ne peut pas créer une seconde base de données.

Pour plus d'information `help Account Management`.

5.4.4 Description rapide de SQL

SQL (*Standard Query Language*), est un langage d'interrogation de bases de données normalisé. Il permet en principe d'accéder à n'importe quelle base de données qui respecte le standard; en pratique, chaque système de gestion de base de données présente des particularités qui font que ce qui fonctionne avec l'un ne marche pas nécessairement avec l'autre. Je présente ici ce qui me semble l'essentiel.

SGBD, serveur, base de données, tables

Un SGBD est l'abréviation standard pour *système de gestion de base de données*. Le SGBD le plus courant dans l'univers Linux est sans doute MySQL

(présenté ici) mais Oracle, SAS et Access sont d'autres systèmes importants.

Avec MySQL, les bases de données sont stockées sur une machine spécifique, le *serveur*. Pour accéder à la base, on utilise un programme *client* qui sert d'interface avec le processus serveur qui tourne sur ce serveur. [7] Le client peut tourner sur une autre machine et se connecter au serveur en utilisant les protocoles d'Internet.

Un serveur MySQL peut héberger plusieurs bases de données distinctes, de la même manière qu'un serveur web permet d'héberger plusieurs sites. Le client, en se connectant, indiquera la base concernée. En première approximation : on met dans une même base toutes les données qui seront utilisées par un client ; dans deux bases différentes, les données qui ne seront jamais utilisées en même temps par un même client.

Les données d'une base sont organisées sous forme de *tables*. Au moment de la création d'une table, on fixe la liste de ses champs, qui correspondent aux colonnes d'un tableau (avec un type de données affecté de façon permanente à chaque colonne). Par la suite, on y insère et on en détruit des *enregistrements*, les lignes du tableau. On extrait certains enregistrements d'une table. Le serveur peut aussi effectuer des calculs sommaires sur le contenu des colonnes numériques.

Une autre opération importante, appelée *join*, consiste à construire une table temporaire en extrayant les enregistrements de deux tables et en réunissant ceux qui possèdent deux champs identiques.

Les sections suivantes reviennent sur ces choses en montrant des commandes simples qu'on peut donner au shell `mysql`.

Création d'une base de données

```
CREATE DATABASE labase;
```

Pas grand chose à dire : normalement la base est créée.

Créer et remplir une table

On crée une table `id` avec deux colonnes et trois lignes dans laquelle on mettra les données :

clef	*nom*
1	toto
2	tata
3	joe

```
CREATE TABLE labase.id (
   clef INT(4) UNIQUE AUTO_INCREMENT PRIMARY KEY,
```

7. Le mot « *serveur* » peut en général désigner indifféremment le processus et la machine sur laquelle il tourne. L'un est logiciel, l'autre est du matériel.

```
   nom VARCHAR(26)
);
```

Dans la base `labase`, on crée donc une table `id` qui associe une clef (entière, sur 4 octets) avec un nom (une chaîne de caractères de longueur variable, normalement pas plus de 26 caractères).

On demande au SGBD de s'assurer que la clef est unique. On peut s'abstenir de la fournir et dans ce cas il incrémentera la dernière clef entrée. Cette clef sera utilisée comme *clef primaire* : le SGBD fera le nécessaire pour qu'on puisse retrouver rapidement les lignes qui contiennent une valeur donnée pour cette clef.

```
INSERT INTO labase.id VALUES (1, 'toto');
INSERT INTO labase.id VALUES (2, 'tata');
INSERT INTO labase.id VALUES (NULL, 'joe');
```

Des commandes pour insérer trois lignes dans la table. Noter que la clef vaut `NULL` dans la troisième ligne. Comme ce champs a été qualifié d'auto-incrémenté à la création de la table et que la dernière ligne portait le numéro 2, l prend la valeur $2 + 1$ pour cette nouvelle ligne.

Une autre table, d'autres données

Histoire de pouvoir effectuer quelques opérations, on fabrique une autre table et on y insère des données.

```
CREATE TABLE labase.aime (qui INT(4), quoi VARCHAR(10));
INSERT INTO labase.aime VALUES (1, 'ail');
INSERT INTO labase.aime VALUES (2, 'oignon');
INSERT INTO labase.aime VALUES (3, 'ail');
INSERT INTO labase.aime VALUES (3, 'oignon');
INSERT INTO labase.aime VALUES (3, 'chou');
```

Récupérer des données d'une table

```
SELECT * FROM labase.id;
```

La commande `select` permet de récupérer des lignes dans une table. Ça donne avec le client MySQL :

```
+------+------+
| clef | nom  |
+------+------+
|    1 | toto |
|    2 | tata |
|    3 | joe  |
+------+------+
```

`SELECT * FROM labase.aime;` fait pareil dans l'autre table de la base.

`SELECT qui FROM labase.aime WHERE quoi = 'ail';`

On peut choisir les lignes (comme le nom *select* le laisse penser) et choisir les champs qu'on voit (ici, seulement le champs `qui` et pas le champs `quoi`. On voit donc ici les noms de ceux qui aiment l'ail.

Détruire et modifier des lignes

Deux exemples simples :

```
DELETE FROM labase.aime WHERE quoi = 'chou';
UPDATE labase.id SET nom='tatou' WHERE id=2;
```

Croiser des tables

```
SELECT labase.id.nom, labase.aime.quoi
   FROM labase.id
   JOIN labase.aime
   ON labase.id.clef = labase.aime.qui;
```

Le croisement des tables est ce qui fait l'intérêt principal des bases de données du type de celles que gère MySQL. Cela permet d'éviter la duplication des données et facilite la mise à jour et le contrôle de leur cohérence.

Pour éviter de répéter sans cesse le nom de la base, on peut indiquer une base par défaut avec `USE labase;` et la même commande s'écrira (plus facilement) :

```
 USE labase;
 SELECT id.nom, aime.quoi
   FROM id JOIN aime
   ON id.clef = aime.qui;
```

Mot de passe facile

Pour la suite de la démonstration, je modifie le mot de passe de l'utilisateur `root` de MySQL.

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('peusûr');
```

5.4.5 Accès à la base de données depuis PHP

En première approximation, c'est vraiment facile : on connecte le programme PHP avec le serveur. Il envoie des commandes au serveur comme on l'a fait avec la commande `mysql`. Il reçoit la réponse.

Il y a plusieurs bibliothèques qui aident à faire ça ; la plus simple à mon avis est `MySLQi` que j'utilise ici. D'autres bibliothèques sont un peu plus compliquées

mais présentent certainement des avantages (notamment elles promettent d'accéder à d'autres systèmes de gestion de bases de données sans modification du code).

Utilisation élémentaire

La figure 5.4 contient un script élémentaire qui permet d'accéder à la base de données MySQL qu'on a construite au dessus à travers PHP. J'en détaille le contenu dans les paragraphes qui suivent.

La fonction nettoyer est la même que celle de 5.3.4 ; elle sert à sécuriser un peu les choses en évitant l'usage des caractères spéciaux dans les données soumises.

La connexion au serveur MySQL se fait avec un appel à la fonction `mysqli_connect` ; le quatrième argument (la base) est facultatif.

La commande est envoyée avec la fonction `mysqli_query`, comme une chaîne de caractères ordinaires. Vérifier qu'on peut y entrer n'importe quelle commande MySQL ; « `INSERT INTO aime VALUES (123, 'chou');` » aussi bien que « `DELETE FROM aime WHERE quoi = 'chou';` ».

On peut utiliser `SELECT` mais faute de code pour voir son contenu, ce n'est pas très utile. La section suivante contient un exemple d'utilisation des données renvoyées par `SELECT`.

Récupérer la sortie d'un `SELECT`

Le fichier `select.php` contient un exemple de code PHP qui utilise la sortie du select.

La requête (ici `SELECT id.nom, aime.quoi FROM id JOIN aime ON id.clef = aime.qui;`) est transmise avec un `mysqli_query` ordinaire.

Le code trouve le nombre de lignes de la réponse avec l'appel `mysqli_num_rows($res)` (ou `$res` est ce qu'a renvoyé l'interrogation).

Chaque ligne de données est récupérée avec un appel à `mysqli_fetch_assoc`. La fonction sert aussi à contrôler la boucle : quand il n'y a plus rien à lire, elle renvoie faux.

```
while($ligne = mysqli_fetch_assoc($res)) {
  ...
}
```

Le contenu de la ligne est renvoyé sous la forme d'un tableau (associatif) dont chaque élément correspond à une des cellules de la ligne. Le code la parcourt donc avec

```
<!DOCTYPE html>
<html>
    <meta charset="utf-8">
    <body>
        <?php
          # Tenter d'éviter les injections (inspiré de w3chools.net)
          function nettoyer($x){
            if ($x){
              $x = trim($x);
              $x = stripslashes($x);
              $x = htmlspecialchars($x);
            }
            return $x;
          }

           $commande = nettoyer($_GET['afaire']);
           if (! $commande)
             $adj = "";
           else {
             $adj = "autre";
             echo "<p>Vous avez entré la commande <code>$commande</code>";

             # Connexion
             $server = "localhost";
             $user = "root";
             $passe = "peusûr";
             $base = "labase";
             $con = mysqli_connect($server, $user, $passe, $base);
             if (! $con)
               echo "Pas de connexion : " . mysqli_connect_error();
             else {
               echo "<p>On est connecté à la base";

               # Envoi de la commande
               if ($res = mysqli_query($con, $commande))
                 echo "<p>La commande semble avoir fonctionné";
               else
                 echo "<p>Aie ! " . mysqli_error($con);
             }
           }
        ?>

        <form method="get"
              action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
            <p>Entrez une <?php echo $adj; ?> commande MySql pour LaBase : <input type="text" name="afaire" />
            <p><input type="submit">
        </form>
    </body>
 </html>
```

FIGURE 5.4 – Un petit script qui permet d'entrer des commandes MySQL via PHP.

```
foreach($ligne as $i)
```

5.4.6 Pas traité

Le cours ne fait qu'effleurer le vaste sujet des bases de données.

Il faudra comprendre la particularité des bases de données relationnelles (celles gérées par MySQL le sont).

Il faudra pratiquer pas mal parce que les bases de données sont intéressantes quand il y a beaucoup de données ; il y alors des problèmes de performances ; la compréhension des enjeux de performances vient mieux (à mon avis) avec de la pratique qu'avec des études théoriques.

Il faudra regarder les bases pas relationnelles, dites *No SQL*, qui jouent un rôle croissant sur le web (et dans le cloud).

5.5 Exercices

Comme d'hab : ajouter à votre page perso au moins un formulaire, traitez-le avec du PHP qui utilise aussi une base MySQL.

Chapitre 6

Le langage JavaScript

Encore un langage de script ! Celui-ci est bien intégré à HTML.

JavaScript est pensé pour être intégré dans les pages HTML transmises par le serveur et exécuté par le navigateur (figure 6.1). Cela permet d'avoir des pages dont le contenu se modifie *sans interaction avec un serveur Web* : une fois la page chargée, le JavaScript qu'elle contient s'exécute exclusivement dans le navigateur.

JavaScript peut être utilisé à l'extérieur d'un navigateur, notamment `node.js` (voir `nodejs.org`). Je ne suis pas certain que ce soit une très bonne idée.

JavaScript décoiffe un peu comme langage. On y trouve des choses habituelles pour un langage de programmation (variables, structures de contrôle, fonctions, objets). Il y a aussi des cotés moins habituels : on peut manipuler les fonctions comme des éléments de première classe, un peu à la manière du Lisp ; les objets y ont une implémentation inhabituelle et étonnamment puissante.

JavaScript n'a à peu près aucun rapport avec le langage Java (sinon que c'est un langage de script destiné initialement à ceux que la complexité de Java

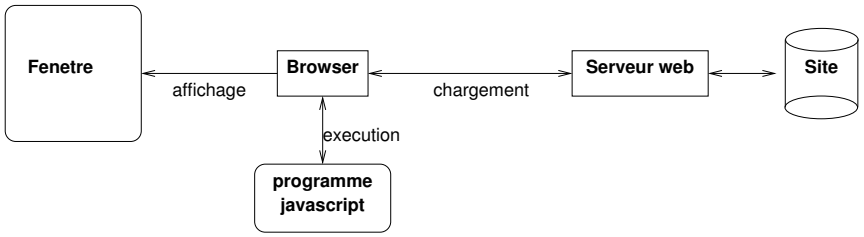


FIGURE 6.1 – Le navigateur (Firefox) charge une page Web depuis le serveur (Apache). En plus du HTML, la page contient des programmes en JavaScript que le navigateur exécute.

effrayait — à juste titre à mon avis).

Le langage vient à l'origine d'un des navigateurs à la mode au milieu des années 90 ; il est maintenant intégré dans à peu près tous les navigateurs d'une façon compatible. Il est aussi l'objet d'un norme formelle, sous le nom ECMAScript. [1]

6.1 Des ressources

Plein de tutoriels, de qualités variables, sont disponibles sur le net.

Eloquent JavaScript est excellent pour apprendre à programmer avec JavaScript. La seconde édition (pas encore traduite en français au printemps 2016) traite sérieusement et efficacement de l'intégration de JavaScript dans HTML.

`developer.mozilla.org/fr/docs/Web/JavaScript/`) contient plusieurs tutoriels en français. *Une réintroduction à JavaScript* est bien pour ceux qui savent déjà programmer (par exemple en Lisp et en Python ...).

6.2 Quelques exemples simples

On embarque le JavaScript entre des balises `<script>···</script>`, à n'importe quel endroit dans la page. [2]

6.2.1 Exemple : fabriquer une partie du document

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Un zéroième exemple de JavaScript</title>
  </head>
  <body style="background-color:#efface;">
    <script>
      var i;
      for(i = 0; i < 100; i++){
        document.write("<p id=par");
        document.write(i);
        document.write(">Ceci est le paragraphe numéro ");
        document.write(i);
        document.write("</p>\n");
```

1. ECMA est un organisme de normalisation, fondé par des constructeurs d'ordinateurs européens (le nom ECMA signifiait à l'origine *European Computer Manufacturer Association*).

2. On peut aussi faire charger du JavaScript depuis un fichier avec l'attribut `src=` de la balise `<script>`.

```
      }
    </script>
  </body>
</html>
```

Ca fabrique une page Web avec 100 paragraphes, comme on le ferait en PHP avec

```
<?php
  for($i = 0; $i < 100; $i++)
    echo "<p id=par" . $i . ">Ceci est le paragraphe " . $i ."</p>\n";
?>
```

Il y a une différence importante entre cet exemple et son équivalent en PHP : la boucle JavaScript est exécutée *par le navigateur* alors qu'en PHP elle est exécutée *par le serveur*. Quand on examine le source de la page (Firefox : clic droit > *View page source*) on voit le script JavaScript alors qu'on ne voit pas le PHP.

6.2.2 Exemple : alert

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Un premier exemple de JavaScript</title>
    <script>alert("dans la tete");</script>
  </head>
  <body style="background-color:#efface;">
    <p id="par1">Ceci est un premier paragraphe
    <p id="par2">Ceci est un deuxième paragraphe
    <p><button type="button">Taper ici</button>

    <script>alert("à la fin du corps");</script>
  </body>
</html>
```

L'exemple utilise juste la fonction `alert` qui fait surgir une fenêtre et demande une confirmation (deux fois) [3].

Noter que ça fonctionne : les fenêtres surgissent. Cela signifie que par défaut Firefox interprète le JavaScript présent dans les pages qu'il charge.

Noter l'état de la page quand la fenêtre d'alerte apparaît.

Déplacer le script dans le corps et noter la différence dans l'état de la page.

3. En réalité, `alert` est une méthode de l'objet `window` qu'on peut appeler avec `window.alert`.

Constater qu'il ne se passe pas grand chose quand on clique sur le bouton : il y a juste une sorte d'accusé de réception du bouton : il change de couleur pendant que le bouton de la souris est enfoncé.

6.2.3 Exemple : la console

Remplacer `alert` par `console.log`. Faire apparaître la console (avec `F12` sous Firefox ; un nouveau F12 la cache). Trouver le message.

Constater qu'au même endroit que la console, il y a des outils (dont un debugger).

6.2.4 Exemple : une boucle

Placer le `alert` dans une boucle, comme dans

```
<script>
  var i;
  for(i = 0; i < 100; i++){
    alert("message " + (i + 1) + " (sur 100)");
  }
</script>
```

C'est là qu'on est content que Firefox propose de fermer le clapet à la page en cochant une case dans la fenêtre d'alerte.

On déclare la variable avec `var` (sans type).

Visiblement, `(i + 1)` est converti en chaîne de caractères dans l'argument d'`alert`.

L'opérateur + sert à la fois pour l'arithmétique ($i + 1$) et pour la concaténation de chaînes de caractères. Regarder ce qui se passe quand on retire les parenthèses autour de `i + 1` !

```
    alert("message " + i + 1 + " (sur 100)");
```

6.2.5 Exemple : une fonction et un bouton

```
<button id="b1" type="button" onclick="clic()">Taper ici</button>

<script>
  var nfois = 0;
  function clic(){
    nfois += 1;
    alert("Un clic sur le bouton (" + nfois + " fois en tout)");
  }
</script>
```

Le script contient une définition de la fonction `clic`.

Dans l'attribut `onclick` du bouton, on peut mettre du JavaScript qui sera exécuté quand on y cliquera. Ici j'ai mis un appel à la fonction `clic`.

6.2.6 Exemple : modifier le contenu de la page

```
function clic(){
  nfois += 1;
  var str = "Taper ici " + nfois + " fois";
  document.getElementById("b1").innerHTML = str;
}
```

Ici, le JavaScript modifie ce qui est écrit sur le bouton. La fonction `document.getElementById` extrait de la page Web la balise dont l'attribut vaut `b1` ; elle modifie le texte contenu dans cette balise en accédant à `innerHTML`.

Vérifier qu'on peut modifier de la même manière le contenu des paragraphes `par1` et `par2` de la page.

6.3 Un survol de JavaScript

L'étendue du cours est telle qu'il n'est pas question de présenter le langage JavaScript en profondeur. (De toute façon, ce n'est pas dans un cours qu'on apprend à utiliser un langage de programmation : c'est en écrivant dans ce langage et en lisant des programmes écrits par d'autres.) Pour une présentation exhaustive de JavaScript, on trouve plein de tutoriels sur Internet ; certains sont d'excellente qualité ; voir notamment *Eloquent JavaScript* que j'ai mentionné au début du chapitre.

Ici, je me contente d'énumérer ce qui semble être les idiotismes les plus importants du langage.

6.3.1 Types et déclarations

On déclare les variables mais pas leurs types (qui peuvent changer en cours d'exécution). Le plus simple est de déclarer ses variables avant leur utilisation.

Les nombres ont la particularité d'être tous des nombres avec des virgules flottantes. De ce fait, (1) quelque chose comme 3/2 vaut 1.5 et (2) il y a parfois des surprises avec les arrondis[4]. Parmi les nombres en virgules flottantes, on trouve `Infinity` et `NaN` qui indiquent l'infini (en pratique, `Inf` : on a dépassé la plus grande valeur représentable ; `NaN` *(not a number)* on a essayé une opération avec un résultat pas défini).

4. Question : comment effectuer une division entière (« *euclidienne* ») en JavaScript ? À faire : écrire le résultat de 0.1 + 0.2.

Les chaînes de caractères se concatènent avec l'opérateur +.

Les conversions implicites entre chaînes et nombres tendent vers les chaînes ; par exemple `"14" + 3` et `3 + "14"` valent la même chose : la chaîne `"314"`. Attention quand même aux règles de conversions qui tiennent compte du contexte dans laquelle une expression est utilisée ; les résultats sont parfois étonnants pour ceux qui sont habitués aux régies de conversion qui ne tiennent compte que des constituants d'une expression (comme en C) :

`5 + 2`	`7`	nombres
`"5" + "2"`	`"52"`	chaînes
`"5" + 2`	`"52"`	conversion (normale) du nombre vers une chaîne
`5 + "2"`	`"52"`	conversion (plus étonnante) du nombre vers une chaîne
`"5" - "2"`	`3`	conversion (étonnante) des chaînes vers les nombres
`"5" * "2"`	`10`	pareil

Avec les conversions implicites, 0, "", False, Null et undefined sont égaux.

Il y a un type *Boolean* avec les valeurs *True* et *False*.

Il y a un type `undefined` (= pas initialisé) qui n'accepte qu'une valeur (`undefined`). Il y a une *valeur* `null` qui est du type `object`.

Il y a également des tableaux, des fonctions et des objets, évoqués plus loin.

6.3.2 Opérateurs et structures de contrôle

On trouve les mêmes opérateurs qu'en C et que dans les langages qui en descendent. Le + sert aussi pour la concaténation de chaînes de caractères.

Il y a (comme en PHP) des `===` et `!==` pour comparer des objets sans conversions : ont-ils même type *et* même valeur pour `===`.

JavaScript possède les opérateurs bit à bit du C : `~` pour la négation, `&`, `|` et `^` pour le *et*, le *ou* et le *ou exclusif*, `<<` et `>>` pour les décalages. Leur utilisation permet de forcer une conversion en entier. [5]

On retrouve les structures de contrôles issues du C (y compris le `switch`), avec `return`, `break` et `continue` qui fonctionnent de la même manière.

Il y a une forme de boucle `for` qui permet d'accéder aux différents champs d'un objet : **`for(i in xxx) ...`**.

On peut gérer les exceptions avec `try ... catch` et `throw`.

6.3.3 Les fonctions

On déclare une fonction avec le mot clef `function`, comme dans

```
function adder(a1, a2){ return a1 + a2; }
```

5. Si on force la conversion d'une expression pour qu'elle soit entière avec un opérateur bit à bit, est-elle signée ou pas ?

La valeur renvoyée par la fonction n'est pas typée, ses arguments non plus. Le nom de fonction est facultatif, ce qui permet d'avoir des fonctions anonymes ; on peut par exemple les stocker dans des variables :

```
var toto = function(a, b){ return a + b };

console.log(toto(12, 23));
console.log(toto("ceci", "cela"));
```

6.3.4 Les tableaux

Les tableaux sont indexés par des nombres (entiers) ; les index commencent toujours à 0, comme en C.

Tous les éléments d'un tableau ne sont pas nécessairement du même type.

La meilleure manière de les initialiser consiste à utiliser les crochets :

```
var tablo = ["zero", 1, "deux"];
```

Mesurer la longueur avec `length` ; la bonne manière de parcourir un tableau :

```
for(i = 0, lim = tablo.length; i < lim; i++)
  traiter(tablo[i]);
```

C'est un peu plus rapide que d'accéder à `tablo.length` à chaque tour de boucle. C'est beaucoup plus rapide que d'utiliser `for(i in tablo) ...`.

Le nombre d'éléments d'un tableau peut varier et on peut avoir des tableaux *creux* : la boucle suivante appellera `traiter` mille et une fois, dont 998 avec la valeur `undefined`.

```
var tablo = ["zero", "un" ];

tablo[1000] = "mille";
for(i = 0, lim = tablo.length; i < lim; i++)
  traiter(tablo[i]);
```

Ajouter et retirer des éléments à droite dans un tableau avec `push` et `pop` et à gauche avec `unshift` et `shift`.

Insérer ou retirer des éléments avec `splice`. Avec `tab.splice(ou, combien, lesquels)`, en démarrant à gauche de l'élément `ou`, ça en retire `combien` et ça insère `lesquels`.

Convertir en chaîne implicitement, avec `toString` ou avec `join`.

```
var tablo = [ "figues", "bananes", "noix" ];
console.log("des " + tablo.join(", des"));
```

Il y a plein d'autres méthodes sur les tableaux. Trouver et étudier une référence.

6.3.5 Les objets

Ça se déclare avec des accolades :

```
var coordA = { "x":10, "y":20 };
```

On peut voir un objet comme un tableau associatif : on peut accéder à un champs avec `coordA["x"]` aussi bien qu'avec `coordA.x`. C'est sérieusement restrictif mais ça permet d'utiliser
`for(i in coordA) console.log("Le champ "  i + " vaut " + coordA.i);+`

On peut ajouter des champs après la création d'un objet (`coordA.couleur = "blanc";`) et en retirer (`delete(coordA.couleur)`).

La meilleure manière de fabriquer un objet est cependant avec un constructeur (un prototype en fait) : une fonction qui initialise les champs :

```
function point(x, y){ this.x = x; this.y = y; }
var coordA = new point(10, 20);
var coordB = new point(20, 10);
```

Les méthodes sont des fonctions anonymes

```
coordA.manhattan = function(){ return this.x + this.y; }
coordB.manhattan = function(){ return this.x + this.y; }
```

ou mieux, dans le prototype pour ne pas avoir à l'ajouter à la fois à `coordA` et à `coordB` :

```
function point(x, y){
   this.x = x;
   this.y = y;
   this.manhattan = function(){ return this.x + this.y; }
}
var coordA = new(10, 20);
var coordB = new(20, 10);
```

Il y a beaucoup d'autres choses à savoir sur les objets JavaScript, notamment sur l'utilisation des prototypes. *Eloquent JavaScript* est bien sur ce point.

6.3.6 Évènements

Dans la page, on peut associer du code avec des actions de celui qui la consulte. Les plus importantes à mon avis :

- `onclick`, `ondblclick` quand il se passe quelque chose sur les boutons de la souris (`onmousedown` et `onmouseup` pour plus de détails).
- `onkeypress` quand il se passe quelque chose sur le clavier (`onkeydown` et `onkeyup` pour plus de détails).

— `onfocus` quand un élément d'un formulaire commence à être celui qui va recevoir les caractères tapés ; `onblur` quand il cesse de l'être ; `oninput` quand il reçoit des caractères.
— `onmouseover` quand la souris passe sur la structure.
— `onsubmit` quand un formulaire est envoyé (permet de contrôler le contenu et d'invalider l'envoi avec `return false;`).
— `onload` pour ce qui doit être effectué une fois (au chargement de la page).

Il y a bien d'autres évènements (environ 80) et avec chacun, un objet qui permet d'avoir des détails. Par exemple, avec un `keypress` on peut savoir si la touche *majuscule* était enfoncée.

6.3.7 D'autres choses

Au début du script ou d'une fonction, on peut modifier le comportement de JavaScript avec `"use strict";`. À utiliser systématiquement. Ça permet entre autres de vérifier que toutes les variables sont déclarées.

Le *hoisting* est la manière dont les déclarations de variables peuvent intervenir *après* leur utilisation. La bonne manière de procéder est de ne pas l'utiliser en déclarant toujours les variables au début.

L'instruction `debugger;` pose un point d'arrêt dans le code.

6.3.8 Les expressions régulières

Les expressions dites *régulières*[6] sont des éléments de première classe dans JavaScript. Voir le site `https://developer.mozilla.org`.

6.4 JSON

Le langage Javscript a débouché sur une représentation des données commune sur Internet appelée JSON (comme *JavaScript Object Notation*) : il s'agit grosso modo de représenter des données de la manière dont on initialise des variables JavaScript.

JSON est souvent en concurrence avec XML ; à mon avis, il est le plus souvent préférable de le choisir : les représentations de données sont plus faciles à lire et à écrire.

La principale différence entre JSON et du JavaScript ordinaire est que les clefs dans les objets doivent être mises entre guillemets en JSON (alors que les guillements sont en général facultatifs en JavaScript).

Un exemple , recopié de la page JSON de wikipedia :

```
{
```

6. Le terme *expressions régulières* est un anglicisme, pauvre traduction de *regular expressions* ; le terme correct en français est *expressions rationnelles*.

```
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

6.5 DOM et JavaScript

Du code JavaScript peut accéder au HTML de la page dans lequel il est embarqué, pour le consulter et *pour le modifier*. Pour cela, la page est représentée comme un objet à la structure un peu compliquée qu'on trouve dans la variable `document`.

Cet objet `document` est la racine d'un arbre qui représente les éléments du HTML imbriqués les uns dans les autres. Dans cet arbre, tout est représenté par des noeuds, qui sont également des objets JavaScript. Par exemple, la racine `document` a normalement deux enfants `head` et `body`.

On déjà utilisé ça dans les exemples du début du chapitre dans sa forme la plus simple : `el = document.getElementById("toto")` pour retrouver un élément en utilisant son `id` ; `el.innerHTML = "texte"` pour modifier son contenu.

On peut naviguer dans l'arbre, modifier sa forme en créant et ajoutant de nouveaux noeuds ou en détruisant certains, modifier les caractéristiques d'un élément en modifiant son contenu, ses attributs ou son style.

Une présentation détaillée du DOM occuperait facilement tout un chapitre. Sa présentation dans *Eloquent JavaScript* (deuxième édition) me semble très bien. On ne fait ici que survoler ce qui me semble essentiel.

6.5.1 Rechercher les éléments

On a déjà vu comment chercher un élément par `id` : `document.getElementById`

Par classe : `document.getElementsByClassName` , on obtient un tableau avec tous les éléments de la classe.

6.5.2 Modifier un élément

On a déjà vu comment modifier le texte contenu dans un élément avec `element.innerHTML`. On peut aussi modifier ses attributs avec `element.setAttribute` (ou en accédant directement à `element.`*attribute*). On peut aussi modifier son style via `element.style.property`.

6.5.3 Se déplacer dans l'arbre

Chaque noeud contient un lien vers son parent (`parentNode`) vers tous ses enfants (`childNodes`), vers son premier et son derniers enfants (`firstChild` et `lastChild`), vers son frère-ou-soeur précédent et suivant (`previousSibling` et `nextSibling`).

Avec ces liens, on peut parcourir l'arbre qui représente le document.

6.5.4 Modifier la structure du document

Fabriquer un noeud avec `createElement` ; l'ajouter avec `appendChild` ; le retirer avec `removeChild` ; le remplacer avec `replaceChild`.

Ajouter du code avec `document.getElementById(id).onclick = function(){code}`.

<script> document.getElementById("p2").style.color = "blue" ; </script>

6.6 Pas développés (et pourtant importants)

Les pages vraiment actives qui échangent des données avec des serveurs spécialisés : JavaScript + XML = AJAX. Il y aurait tout un cours à faire là-dessus.

Node.js permet l'exécution du JavaScript en dehors d'un navigateur. Je ne sais pas vraiment si c'est une bonne ou une mauvaise idée de l'utiliser. [À mon avis, JavaScript est un langage intéressant mais ce n'est *pas* la solution à *tous* les problèmes.]

Angular.js est le framework JavaScript commun en ce moment. Je ne le connais pas suffisamment pour avoir un avis autorisé dessus.

Emscripten : pour compiler du C en JavaScript.

JQuery est une bibliothèque JavaScript pour faciliter les opérations courantes ; ça débouche presque sur un nouveau langage ; ça sert de base au framework BootStrap (plus simple que `angular.js` m'a-t-on dit).

6.7 Exercice

Comme d'hab : ajouter du JavaScript à votre Home Page pour avoir une page qui réagit, même quand elle n'a pas accès au net.

Plus facile : imprimer la racine du DOM (`document`) en JSON, avec ses enfants (à grand coup de boucles `for(..in..)`) ; c'est une façon intéressante d'explorer le DOM en pratiquant JavaScript.

Chapitre 7

Internet, ce n'est pas que le Web !

Dans ce dernier chapitre, je vais tenter de parler d'Internet sous un autre aspect que le Web. Je montre comment utiliser la commande `ssh` pour lancer des commandes sur un ordinateur depuis un autre ordinateur ; comment conserver des fichiers sur plusieurs ordinateurs différents, les recopier et les synchroniser ; comment fonctionne le NAT qui isole jusqu'à un certain point nos machines derrière la box de notre fournisseur d'accès ; comment fonctionne le mail et notamment de quelle manière on transporte des données de types variés.

7.1 Créer un utilisateur

Pour pouvoir faire tourner les exemples du chapitre, on va créer un autre compte utilisateur.

On crée un utilisateur, avec un répertoire de login : [1]

```
$ sudo useradd toto
$ sudo passwd toto
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
$ sudo mkdir /home/toto
$ sudo chown toto /home/toto
```

Quand on tape le mot de passe de l'utilisateur *toto*, on n'a pas d'écho des caractères tapés ; pour avoir un niveau raisonnable de protection, éviter les mots de passe trop évidents. À la fin du chapitre, le compte sera à supprimer :

1. Il y a d'autres moyens plus sympathiques que la commande `useradd` pour ajouter un compte mais celle-ci présente l'avantage de la simplicité.

```
$ sudo userdel toto
$ sudo rm -fr /home/toto
```

7.2 Ssh

Les utilisateurs historiques d'Unix (et de Linux) sont attachés au travail dans un terminal, avec la ligne de commande. Une des raisons en est qu'on peut effectuer ce travail sur sa propre machine mais aussi sur une machine distante à travers laquelle on accède à Internet. Pour cela, une des commandes couramment utilisée est `Ssh`.

Le principal intérêt de `ssh` est qu'il est sécurisé : quand on l'utilise, on peut être assez raisonnablement confiant de l'identité de son interlocuteur et penser qu'il sera difficile à quelqu'un d'intercepter les échanges, parce qu'ils sont cryptés. [2]

Du coup, Ssh et son protocole sont largement utilisés dans de nombreux contextes de communication entre ordinateurs : on l'utilise dès qu'on a plusieurs ordinateurs à administrer ; c'est la manière normale d'accéder à certains équipements réseaux, par exemple pour les configurer.

7.2.1 Installer un serveur Ssh

Un client Ssh, qui permet de se connecter à un serveur Ssh, est installé par défaut. En revanche il faut ajouter le serveur. Sous Ubuntu, ça se fait normalement avec

```
# sudo apt-get install openssh-server
```

Ça installe des fichiers de configurations dans le répertoire `/etc/ssh` : `ssh_config` est le fichier de configuration du client et `sshd_config` celui du serveur. Il y a également dans le répertoire une clef d'encryption qui permet d'identifier la machine.

7.2.2 Le travail à distance

Dans la forme la plus courante, on lance une session. On peut aussi exécuter une seule commande.

Première connexion

Lors de la première connexion, Ssh demande confirmation de l'identité de la machine à laquelle on se connecte.

2. Les révélations d'Edgard Snowden laissent parait-il penser que la NSA, l'organisme des États Unis chargé d'intercepter les communications, est en mesure de décoder certaines communications via Ssh.

```
$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is 6b:e1:12:65:38:29:b1:59:25:c1:80:37:0b:63:04:91.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
jm@localhost's password:
```

La clef de cryptage du serveur est stockée de façon permanente sur la machine locale. Éventuellement, si on réinstalle le système sur le serveur, sa clef de cryptage peut changer. Dans ce cas, la clef ne correspond plus à celle qui est stockée et Ssh refuse de se connecter avec un message inquiétant :

```
$ ssh localhost
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
95:0f:78:e0:15:50:72:35:84:19:27:28:3f:37:5c:11.
Please contact your system administrator.
Add correct host key in /home/jm/.ssh/known_hosts to get rid of this message.
Offending RSA key in /home/jm/.ssh/known_hosts:65
  remove with: ssh-keygen -f "/home/jm/.ssh/known_hosts" -R localhost
RSA host key for localhost has changed and you have requested strict checking.
Host key verification failed.
```

La façon la plus simple de résoudre le problème est de détruire la ligne problématique du fichier `$HOME/.ssh/known_hosts` (ici la ligne numéro 65). La prochaine connexion demandera une confirmation comme une première connexion.

Une session

```
toto@machine1$ ssh machine2
toto@machine2$ hostname
machine2
toto@machine2$ exit
toto@machine1$
```

Quand on tape un caractère tilde (~) en début de ligne, le Ssh local (figure 7.1) interprète la ligne au lieu de la transmettre. Voir ce que donne `~?` (tilde suivi d'un point d'interrogation). *Tilde-Point* permet notamment de couper une connexion même si l'autre machine de répond pas.

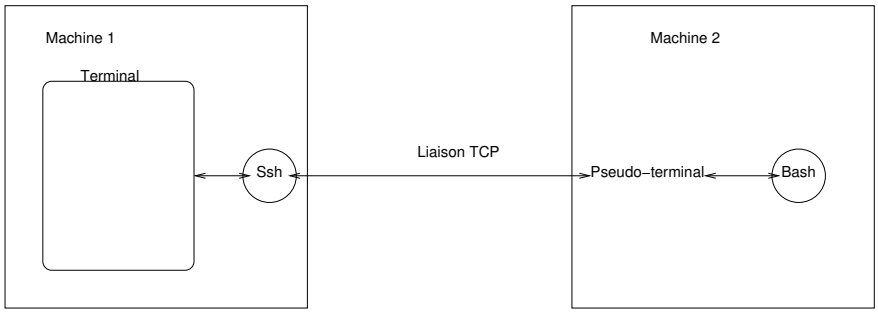


FIGURE 7.1 – Le programme ssh lancé sur la machine 1 transmet les caractères tapés au clavier à la machine 2 où ils sont transmis à un pseudo-terminal à travers une liaison TCP ; les caractères écrits par les programmes dans le pseudo-terminal sont transportés dans l'autre sens par la liaison TCP et affichés sur la sortie standard.

Une seule commande

```
toto@machine1$ hostname
machine1
toto@machine1$ ssh machine2 hostname
machine2
toto@machine1$
```

Se connecter comme un autre utilisateur

On se connecte en tant que `toto`.

```
$ ssh toto@localhost
toto@localhost's password:
  Message de bienvenue
$ who am i
toto     pts/12       2015-09-14 15:01 (localhost)
$
```

Se connecter sans mot de passe

C'est parfois utile de pouvoir se connecter à une machine sans avoir à entrer de mot de passe. (Ça permet notamment d'utiliser Ssh dans des scripts.)

Il faut fabriquer une clef d'encryption (en laissant la *pass phrase* vide) :

```
$ ssh-keygen
Generating public/private rsa key pair.
```

```
Enter file in which to save the key (/home/toto/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/toto/.ssh/id_rsa.
Your public key has been saved in /home/toto/.ssh/id_rsa.pub.
The key fingerprint is:
73:98:bf:3d:bd:6c:f6:7b:75:94:da:8a:cf:8b:d2:53 toto@yem
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|                 |
|                .|
|         o     ..|
|        S .   o. |
|         +   E .o|
|          o o.. o|
|         . *+o+ .|
|          o.+B+++|
+-----------------+
$
```

Ça crée deux fichiers : `id_rsa` contient la clef privée et doit être tenu secret ; `id_rsa.pub` contient la clef publique et peut être publié.

On place la clef publique dans le fichier `$HOME/.ssh/authorized_keys` *sur la machine sur laquelle on souhaite se connecter*; on peut maintenant y entrer sans avoir besoin d'entrer un mot de passe (depuis la machine qui contient le `id_rsa`) :

```
$ scp $HOME/.ssh/id_rsa.pub toto@localhost:.ssh/authorized_keys
toto@localhost's password:
$ scp /etc/issue.net toto@localhost:/tmp/fichier
$
```

Pour la première copie, il a été nécessaire d'entrer un mot de passe. Une fois la clef placée dans le fichier `authorized_keys`, on peut se connecter sans mot de passe, comme le montre la deuxième copie.

Supprimer le serveur Ssh

```
$ sudo apt-get remove openssh-server
$ ssh localhost
ssh: connect to host localhost port 22: Connection refused
$
```

Pour mémoire, supprimer aussi l'utilisateur toto et son répertoire de login.

```
$ sudo userdel toto
$ sudo rm -fr /home/toto
```

7.2.3 Scp et Sftp

Il y a deux commandes associées à Ssh qui permettent de copier des fichiers entre machines en utilisant le protocole de ssh : `scp` (qu'on a utilisé au-dessus) et `sftp`.

7.3 La recopie de fichiers

Une application importante sur Internet. Beaucoup de manière de procéder.

7.3.1 FTP

C'est un protocole, une commande, un serveur, un nom de machine.

File Transfer Protocol. Établir une connexion avec un serveur FTP, s'authentifier, rapatrier des fichiers avec `get`, en envoyer avec `put`.

À la différence de HTTP, il y a une connexion TCP qui reste ouverte en permanence (pour envoyer les commandes et recevoir des comptes rendus) et une nouvelle connexion pour chaque transfert de données. Il faut s'authentifier sur la connexion de commande.

Il y a du ftp *anonyme*, en utilisant le nom `anonymous` ou `ftp` si le serveur a été configuré comme ça. Le FTP anonyme est la façon ordinaire de trouver les distributions Linux.

On peut utiliser ftp dans le navigateur : `firefox ftp://ftp.ubuntu.com`, pour du FTP anonyme (ou pas).

Sur les ftp anonymes, le répertoire `mirrors` contient souvent des copies d'autres sites ftp. Si votre fournisseur d'accès est Free, c'est garanti que le contenu de `ftp.free.fr/mirrors/ftp.utuntu.com` arrivera rapidement sur votre machine parce que la durée de transport sera minime. Ce sera sans doute plus rapide que depuis `fr.ftp.ubuntu.com` et certainement *beaucoup* plus rapide que `ftp.ubuntu.com`.

J'utilise aussi la commande `ftp` pour envoyer les fichiers PDF aux imprimantes branchées directement sur le réseau (ça m'évite de configurer le logiciel de gestion des imprimantes).

Il y a une commande `filezilla` qui offre une interface graphique au protocole FTP.

7.3.2 Unison

Synchronise deux répertoires sur deux machines séparées.

Pratique quand il y a des fichiers modifiés dans deux endroits. (`Rsync` est suffisant quand ils ne sont modifiés que dans un seul endroit et qu'il suffit de recopier toutes les modifications dans l'autre endroit.)

7.3.3 supplément : CVS, GIT, SVN, Mercurial, Subversion etc.

Les logiciels de gestion de version ; ils ont souvent des protocoles associés pour recopier des arborescences. D'autres passent par Ssh. Certains font les deux.

C'est très utile mais ça sort du sujet du cours.

7.3.4 supplément : les disques dans les nuages

Les services d'hébergement de fichiers.

`https://en.wikipedia.org/wiki/Comparison_of_file_hosting_services`

Souvent gratuit quand la capacité est limitée (à quelques Gigaoctets quand même).

7.4 NAT

La plupart des connexions à Internet à domicile qui existent à l'heure actuelle font appel à une technique un peu délicate qu'il est utile de comprendre : le *Network Address Translation* (NAT), évoqué figure 7.2.

Chez soi, on dispose d'un boîtier du fournisseur d'accès qui assure la connexion avec Internet. Si votre Fournisseur d'Accès à Internet (FAI) s'appelle *Pasta*, la box s'appelle sans doute une *pastabox*. La façon dont cette box communique avec le reste du réseau varie (depuis le téléphone, lent, jusqu'à la fibre optique) et sort du cadre de ce cours. La chose importante est que la box dispose d'une adresse IPv4, visible depuis tout le réseau Internet (on parle d'une adresse *publique*) : cette adresse est même la seule chose visible du réseau du domicile depuis la reste du réseau Internet.

Du coté du domicile, les choses se passent d'une façon équivalente au reste du réseau, à une échelle réduite. Dans l'exemple de la figure 7.2, le réseau se compose de trois machines, qui disposent chacune d'une adresse IPv4 : deux ordinateurs ordinaires et la box. Ces machines communiquent entre elles en utilisant les mêmes protocoles que toutes les autres machines du réseau Internet. On peut considérer qu'elles forment un *Intranet* (minuscule).

Cependant, les adresses utilisées dans ce (tout petit) réseau privé sont particulières : ce ne sont pas des adresses ordinaires qui sont garanties uniques sur tout Internet. D'autres réseaux (d'autres Intranet) peuvent utiliser exactement les mêmes adresses. Ça simplifie considérablement l'attribution des adresses

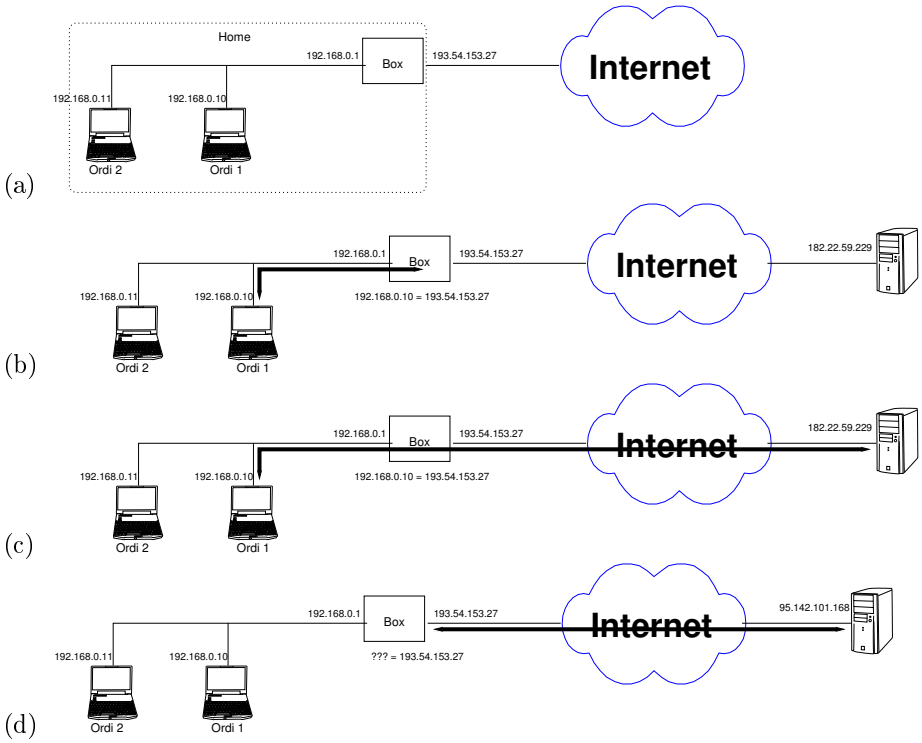


FIGURE 7.2 – (a) Le réseau du domicile contient les machines *Ordi 1* et *Ordi 2* reliées au réseau interne (chacune a son adresse). La liaison avec Internet est réalisée par la *box* du Fournisseur d'Accès qui a deux adresses : l'une à l'intérieur du domicile et l'autre visible de tout le reste du réseau Internet. (b) Quand *Ordi 1* établit une liaison TCP avec un serveur d'Internet, la *box* note dans une table que cette connexion concerne *Ordi 1* (c) Elle modifie en conséquence les messages pour cette connexion avant de les transmettre : le serveur a l'impression de dialoguer avec la box. (d) Quand une autre machine d'Internet tente d'établir une connexion avec *Ordi 2* en utilisant l'adresse visible de l'extérieur, la *box* ne sait pas si le paquet doit être transmis à *Ordi 1* ou à *Ordi 2* : la transmission échoue.

(chacun peut numéroter ses machines comme il veut au sein de son réseau) mais elles sont inutilisables sur le réseau Internet. Ce sont des adresses *Privées*.

Les machines avec des adresses privées peuvent tout de même communiquer avec Internet grâce à un mécanisme appelé le *NAT*[3] (comme *Network Address Translation*). Quand une des machines du domicile établit une connexion TCP avec un serveur qui se trouve ailleurs sur Internet, la demande de connexion passe par la box. La box remplace alors l'adresse (privée) de la machine par sa propre adresse publique et place cette traduction dans une table.

Quand le serveur répond à la demande de connexion du client, la box effectue la traduction inverse : elle remplace l'adresse du destinataire (sa propre adresse publique) par la véritable adresse du destinataire (l'adresse privée du destinataire) puis elle renvoie le message au destinataire. Tous les messages de cette connexion seront ensuite traduits de la même manière : l'adresse privée sera remplacée par l'adresse publique dans les messages qui sortent de l'intranet ; l'adresse publique sera remplacée par l'adresse privée sur les messages qui rentrent dans l'intranet.

Pour le serveur (coté Internet) tout se passe comme si la communication se faisait avec une machine qui possède l'adresse publique (avec la box), puisqu'elle envoie et reçoit des messages avec l'adresse publique. Pour le client (coté Intranet) la mécanique est transparente : il reçoit et envoie des messages avec sa propre adresse privée comme avec n'importe quelle autre adresse ordinaire. Seule la box est au courant : l'examen et le remplacement éventuel des adresses d'émission et de réception est suffisamment simple pour qu'elle l'effectue au vol sans retarder notablement les échanges.

7.4.1 Les deux problèmes principaux du NAT

Il y a tout de même des situations où le NAT pose des problèmes : quand on transmet des adresses dans le contenu des messages et quand on a des serveurs avec des adresses privées.

Quand le client envoie son adresse dans le contenu du message, la box risque de ne pas la traduire. (La box n'effectue le plus souvent que la traduction des adresses qui figurent sur *l'enveloppe* qui contient les messages, pas dans le message lui-même.) Or certains protocoles (notamment FTP dans certaines configurations) *exigent* que les adresses soient transmises dans le corps des messages. Cela oblige la box à examiner le *contenu* des messages et limite la transmission aux protocoles dont la box sait analyser le contenu.

Le NAT complique les choses quand on veut avoir un serveur sur son Intranet (sur une machine avec une adresse privée placée derrière sa box). Quand un client (sur Internet) souhaite s'y connecter, il doit adresser ses messages à l'adresse

3. Le terme correct devrait être NAPT comme *Network Address and Port Translation* car le mécanisme concerne aussi les numéros de ports, utilisés avec les adresses pour identifier les connexions réseaux. Le noyau du système Linux utilise aussi parfois le terme *IP Masquerade* pour désigner le même mécanisme.

publique de la box. Quand elle reçoit le message, il faut un mécanisme qui permette à la box de déterminer par quelle adresse privée elle doit remplacer son adresse publique sur l'enveloppe du message.

Il est en général possible de configurer la box pour lui indiquer une adresse privée vers laquelle faire suivre les messages qu'elle reçoit. Cela permet d'avoir à son domicile au moins un serveur visible depuis le reste d'Internet. Ces redirections par défaut sont souvent paramétrables en fonction du port auquel le client adresse le message, ce qui permet d'avoir un serveur par service.

7.4.2 Les deux avantages principaux du NAT et IPv6

Le principal avantage du NAT est d'avoir permis à Internet de continuer à grandir, même après que toutes les adresses IPv4 aient été utilisées.

L'autre avantage significatif est que la box se comporte de fait comme un *firewall* : elle ne laisse passer que les connexions TCP dont l'origine provient des machines de l'Intranet. Une machine extérieure échouera si elle tente de se connecter de sa propre initiative aux machines de l'Intranet (sauf en cas de redirection par défaut).

La nouvelle version du protocole principal d'Internet (qui s'appelle IPv6) dispose de beaucoup plus d'adresses que la version la plus courante (IPv4). De ce fait, on n'y utilise très peu le NAT. Du coup, à mesure que les FAI utilisent IPv6, les réseaux privés aux domiciles de leurs clients risquent de présenter des failles de sécurité : il ne faut activer IPv6 sur sa box que quand on comprend ces enjeux.

Exercice

Transférer les sources de votre page web sur un support auquel je puisse accéder depuis n'importe où sur le réseau ; je préfèrerais une page ftp dont je pourrais aspirer le contenu avec la commande `wget` mais d'autres méthodes sont acceptables aussi.

Installer et me rendre accessible un serveur sur votre réseau privé ou bien trouver un endroit où votre FAI indique que c'est impossible du fait du NAT. Le serveur n'a pas besoin d'être compliqué : il doit accepter une connexion, y lire le message `Bonjour`, répondre avec `Bonsoir` et c'est tout. [Il existe plusieurs manières de réaliser ce serveur ; la plus simple est sans doute d'utiliser la version de netcat qui permet d'exécuter une commande avec l'option `-e`.