Réalisation de pages de manuel

Jens Schweikhardt <jens@kssun3.rus.uni-stuttgart.de>, trad. Rene.Cougnenc@freenix.fr - Janvier 1996

Ce document explique ce que l'on doit connaître pour pouvoir rédiger de la documentation en ligne destinée à être exploitée par la commande `man(1)` : ce sont les fameuses *pages de manuel*.

Table des matières

1 Quelques évidences **2**

2 Organisation et principe du manuel en ligne **2**

2.1 Il n'y a pas que l'anglais au monde... 4

3 À quoi une page de manuel formatée doit-t-elle ressembler? **4**

4 Comment documenter plusieurs choses dans une seule page? **10**

5 Quel ensemble de macros choisir? **10**

6 Quels préprocesseurs utiliser? **11**

7 Distribution: code source et/ou documentation formatée? **12**

8 Fontes disponibles et conventions typographiques **13**

9 Conseils, trucs et astuces **14**

9.1 Peaufinez votre page de manuel 14

9.2 Comment obtenir une sortie pur ascii sans ces fichus caractères de contrôle? 15

9.3 Comment obtenir une sortie *PostScript* de haute qualité? 15

9.4 Comment faire fonctionner les commandes `apropos` et `whatis`? 16

9.5 La langue française 16

10 Copyright **17**

1 Quelques évidences

Pourquoi rédigeons-nous des documentations? À question stupide, réponse stupide : pour que tout le monde sache utiliser notre programme, notre fonction bibliothèque ou tout autre code de notre cru que nous diffusons dans ce but. Mais la rédaction n'est pas tout :

- la documentation doit être accessible. Si elle est placée en un endroit non standard tel que les outils destinés à son exploitation ne la trouvent pas, comment peut-elle remplir son rôle?
- la documentation doit être fiable et à jour. Il n'y a rien de plus irritant que de rencontrer un programme se comportant différemment de ce qui est décrit dans son manuel. Les utilisateurs vous maudiront, vous enverront des courriers d'insulte, puis rejetteront à jamais tout autre travail venant de vous.

La méthode traditionnelle pour accéder à la documentation sous UNIX fait appel à la commande `man(1)`. Ce document décrit les opérations nécessaires pour la rédaction d'une page de manuel qui sera correctement traitée par les outils de formatage prévus à cet effet. Parmi ces outils, les plus importants sont `man(1)`, `xman(1x)`, `apropos(1)`, `makewhatis(8)` et `catman(8)`.

La qualité et la véracité des informations sont de votre ressort, bien entendu. Malgré tout, vous trouverez dans ce guide quelques idées qui vous permettront d'éviter certains pièges courants.

2 Organisation et principe du manuel en ligne

Il vous vaut connaître avec précision le mécanisme d'accès aux pages de manuel afin de savoir donner un nom correct à vos documents, et d'être capable de les installer au bon endroit. Chaque page de manuel appartient à une *section* spécifique, dénotée par un simple chiffre. Les sections les plus courantes rencontrées sous **Linux** sont :

- **1** : commandes utilisateur pouvant être exécutées par tout le monde ;
- **2** : appels système, c'est-à-dire les fonctions fournies par le noyau ;
- **3** : fonctions des bibliothèques ;
- **4** : périphériques, c'est-à-dire les fichiers spéciaux que l'on trouve dans le répertoire `/dev` ;
- **5** : descriptions des formats de fichiers (comme par exemple `/etc/passwd`) ;
- **6** : les jeux, sans commentaire...
- **7** : divers (macros, conventions particulières, ...) ;
- **8** : outils d'administration système exécutables uniquement par le superutilisateur (`root`) ;
- **9** : un autre endroit (spécifique à **Linux**) destiné à la documentation des services offerts par le noyau ;
- **n** : nouvelle documentation, qui pourra être déplacée vers un endroit plus approprié ;
- **o** : ancienne documentation, qui peut être conservée encore un certain temps ;

– **l**: documentation locale, propre à ce système particulier.

Le nom du fichier source d'une page de manuel (le fichier d'entrée du système de formatage) est constitué du nom de la commande décrite (ou de la fonction, du fichier, etc.), suivi d'un point et du numéro de section. Si par exemple vous documentez le format du fichier « *passwd* », vous devrez appeler votre fichier source « *passwd.5* ». Vous voyez ici qu'il peut exister un fichier du même nom qu'une commande ; nous pourrions tout aussi bien avoir une fonction de bibliothèque appelée « *passwd* ». L'organisation en sections constitue la méthode habituelle pour résoudre ce type d'ambiguïté : la description de la commande se trouvera dans le fichier « *passwd.1* » et notre hypothétique fonction bibliothèque dans « *passwd.3* ».[1]

Attention aux éventuels conflits de noms avec des programmes, fonctions ou fichiers existants. Ce serait par exemple une très mauvaise idée que d'appeler votre nouvel éditeur de texte `ed`, `sed` (pour « super ed ») ou `red` (pour « Roger Editions ») ! En vous assurant que le nom de votre application est unique, vous éviterez qu'un utilisateur exécute votre programme par erreur ou lise sa page de manuel à la place de celle dont il désirait prendre connaissance, ou vice versa. Vous pouvez éventuellement vous aider de la base de données « lsm », qui recense beaucoup des programmes disponibles pour **Linux**.

Maintenant que nous savons déterminer le nom de notre fichier, il nous faut savoir dans quel répertoire l'installer (lorsque la personne qui installera votre application tapera la commande « `make install` » par exemple).

Sous **Linux**, toutes les pages de manuel sont situées dans des sous-répertoires à partir d'une (ou plusieurs) racine indiquée dans la variable d'environnement `MANPATH`. Les outils de traitement de la documentation utilisent cette variable de la même manière que le shell emploie `PATH` pour localiser les programmes exécutables. En fait, `MANPATH` a exactement le même format que `PATH`. Toutes deux contiennent une liste de répertoires, avec le caractère deux-points comme élément séparateur (mais `MANPATH` n'autorise pas de champs vides ou de chemins d'accès relatifs).

Si `MANPATH` n'existe pas ou n'est pas exportée, le répertoire */usr/man* sera alors utilisé par défaut. Dans le but d'accélérer les recherches et pour garder des répertoires de taille raisonnable, les répertoires pointés par `MANPATH` contiennent une série de sous-répertoires nommés « *manS* », où *S* désigne le caractère correspondant au numéro de section que nous avons présenté plus haut. Toutes les sections ne sont pas représentées, il n'y a par exemple pas de raison de conserver une entrée « *mano* ». Vous pourrez y trouver également des sous-répertoires appelés « *catS* », « *dviS* » et « *psS* », qui contiennent la documentation toute formatée, prête à être affichée ou imprimée : nous reviendrons sur ce sujet plus loin. Le seul fichier qui peut être présent à côté de ces sous-répertoires du répertoire de base s'appelle « *whatis* ». Nous décrirons son contenu et la manière de le créer plus tard.

La méthode la plus sûre pour installer au bon endroit une page de manuel de la section « S » est de placer le fichier dans le répertoire */usr/man/manS*. Toutefois un bon `Makefile` devra autoriser l'utilisateur à choisir un autre répertoire de base, disons par exemple par le biais d'une variable d'environnement que l'on pourrait nommer `MANDIR`. La plupart des distributions GNU peuvent être configurées à l'aide de l'option

1. Vous rencontrerez quelquefois des pages de manuel dont le nom ne se termine pas par un chiffre, mais par un chiffre et une ou plusieurs lettres comme par exemple « *xterm.1x* » ou « *wish.1tk* ». Le but de cette notation est d'indiquer que la documentation en question s'applique à un programme X Window ou à une application Tk, respectivement. Certains programmes d'affichage du manuel peuvent exploiter cette particularité ; `xman`, par exemple, affichera « *xterm(x)* » et « *wish(tk)* » dans la liste des documents disponibles.

`--prefix=/mon/option`. Les pages de manuel correspondantes seront alors installées à partir du répertoire de base */mon/option/man*. Nous vous conseillons d'employer une méthode similaire pour vos réalisations personnelles.

2.1 Il n'y a pas que l'anglais au monde...

Depuis l'avènement du « *Système de fichiers standard* » pour **Linux** (FSSTnd), les choses se sont un petit peu compliquées. Ce standard stipule que la structure de */usr/man* doit être prévue pour plusieurs jeux de pages de manuels, correspondant à différents langages, en raison de l'internationalisation du système. Il est alors introduit un niveau d'arborescence supplémentaire qui distingue chaque langue. Voici ce qui est dit dans la version 1.2 de ce standard :

> Le nom des sous-répertoires de langages de */usr/man* est basé sur l'annexe E du standard POSIX 1003.1, qui décrit la chaîne d'identification *locale* ; celle-ci constituant la méthode la mieux acceptée pour décrire un environnement culturel. Cette chaîne *locale* est formée ainsi :
>
> `<langage>[_<pays>][.<jeu-de-caracteres>][,<version>]`

Consultez le standard pour obtenir les chaînes les plus courantes dans nos contrées. En fonction de ces recommandations, nous aurons donc nos pages de manuel dans */usr/man/<locale>/man[1-9lno]*. Les versions formatées se trouveraient alors bien entendu dans */usr/man/<locale>/cat[1-9lno]* ; nous pourrions aussi ne les fournir que pour une seule langue.

Toutefois, je (l'auteur du document, pas le traducteur) ne peut pas recommander d'employer cette structure en l'état actuel des choses. Le standard **Linux** indique aussi que « les systèmes n'utilisant qu'un seul langage et qu'un seul jeu de caractères pour toutes les pages de manuel peuvent omettre le sous-répertoire *locale* et stocker les fichiers dans *mandir* ». Par exemple, sur les machines uniquement équipées pour la langue anglaise, qui n'utilise que le code ASCII, on peut placer les sous-répertoires *man[1-9]* directement dans */usr/man* (en fait, il s'agit de l'emplacement traditionnel et c'est le cas le plus fréquent ; c'est ainsi que sont organisés tous les systèmes par défaut).

Je (l'auteur du document, pas le traducteur) ne changerai pas ma configuration tant que tous les outils (comme `xman`, `info`, `tkman` et beaucoup d'autres) ne seront pas tous adaptés à cette nouvelle structure.

3 À quoi une page de manuel formatée doit-t-elle ressembler ?

Pour répondre à cette question, le mieux est de présenter un exemple pratique. En raison de la nature et du mode de réalisation ce ce document, nous ne pouvons pas reproduire les caractères accentués, ni les différents enrichissements du texte (gras et italiques principalement) ; consultez la section traitant des polices de caractères pour obtenir des détails sur ces possibilités.

Voici comment se présente la page de manuel de notre programme hypothétique « `prout` » :

```
PROUT(1)                Manuel utilisateur                PROUT(1)
```

```
NAME
       prout - proutibule la bibliotheque plaf

SYNOPSIS
       prout [-plaf] [-c fichier-config ] fichier ...

DESCRIPTION
       prout  proutibule  la  bibliotheque plaf en mouglifiant la
       table des symboles.  Par  defaut,  la  commande  recherche
       tous  les segments glurb et les trie par ordre betagonique
       decroissant afin que  le  gloupeur  gloup(1)  les  trouve.
       L'entree  symdef  est  alors  compactee selon l'algorithme
       NABOB.  Les  fichiers  sont  traites  dans  leur  ordre
       d'apparition sur la ligne de commandes.

OPTIONS
       -b     N'affiche  pas  'bidouille  en cours' sur la sortie
              standard pendant le traitement.

       -c fichier-config
              Utilise le fichier de configuration  fichier-config
              au  lieu  du  fichier global /etc/prout.conf.  Cela
              supprime   aussi   l'effet   de   la   variable
              d'environnement PROUTCONF.

       -a     Traite  egalement  les  en-tetes froutz en plus des
              segments glurb.

       -r     Mode  recursif.  Fonctionne  a  la  vitesse  de  la
              lumiere,  mais  necessite  plusieurs megaoctets de
              memoire virtuelle.

FICHIERS
       /etc/prout.conf
              Fichier de  configuration  general,  pour  tout  le
              systeme. Voir prout(5) pour plus de details.
       ~/.proutrc
              Fichier  de  configuration propre a chaque utilisa
              teur. Voir prout(5) pour plus de details.

ENVIRONNEMENT
       PROUTCONF
              Si elle existe, cette  variable  peut  contenir  le
              chemin  d'acces  complet a un autre fichier de con
              figuration global  prout.conf.  L'option  -c  rend
              cette variable inoperante.
```

```
DIAGNOSTICS
       Les  messages suivants peuvent etre affiches sur la sortie
       standard d'erreurs :

       Mauvais nombre magique.
              Le fichier d'entree ne semble pas etre  un  fichier
              archive.

       Segments glurb ancien style.
              prout  ne peut traiter que le nouveau style de seg
              ments glurb. Les bibliotheques GROBOL ne  sont  pas
              supportees dans cette version.

BOGUES
       Le  nom de cette commande aurait du etre choisi de maniere
       a mieux refleter sa fonction.

AUTEUR
       Marcel Dugenou    <dugenou@renux.freenix.fr>

VOIR AUSSI
       gloup(1), plaf(1), prout(5).

Linux                       JANVIER 1996                        1
```

Et voici les explications promises.

La section NAME

C'est la seule section indispensable. Les pages de manuel sans section **NAME** sont aussi inutiles qu'un réfrigérateur au pôle nord. Cette section possède aussi un format standardisé consistant en une liste de noms de programmes ou de fonctions suivis d'un tiret ou d'une courte description (une ligne en général) des fonctionnalités, l'élément séparateur de cette liste étant la virgule. C'est le format exploité par l'utilitaire **makewhatis(8)**, qui analyse ces noms de section pour les intégrer dans la base de données **whatis** : Voilà pourquoi il est important de respecter cette organisation. Dans le code source **groff**, cela donne quelque chose comme ceci :

```
.SH NAME
prout \- proutibule la bibliotheque plaf
```

Le « *backslash* » est très important : cette contre-oblique est nécessaire pour que le tiret qui suit soit distingué d'une marque de césure, qui peut apparaître aussi bien dans le nom de commande que dans la ligne de description.

Attention : en l'état actuel des choses, vous ne pouvez pas traduire **NAME** par **NOM** en français, à moins de modifier la plupart des programmes **makewhatis** existants. C'est bien dommage...

La section `SYNOPSIS`

Elle permet de donner un court aperçu des options du programme. Dans le cas des fonctions, cette section contient les fichiers d'en-têtes qu'il est nécessaire d'inclure, ainsi que le prototype afin que le programmeur connaisse immédiatement le type et le nombre des arguments demandés ainsi que le type de la valeur de retour.

La section `DESCRIPTION`

C'est elle qui explique en détail pourquoi votre séquence de 0 et de 1 est meilleure que toutes les autres. C'est ici que vous étalez tout votre savoir ! Gagnez l'estime des autres développeurs et l'admiration des utilisateurs en faisant de cette section la référence absolue, contenant toutes les informations possibles sur votre application, avec force détails. Expliquez clairement le rôle de chaque argument, décrivez le format des fichiers utilisés, les algorithmes qui effectuent le plus dur du travail, etc.

La section `OPTIONS`

Elle donne une description de chaque option modifiant le comportement du programme. Vous deviez vous en douter...

La section `FICHIERS`

Elle indique quels sont les fichiers utilisés par le programme ou la fonction. Ce sont par exemple les fichiers de configuration et d'initialisation auxquels il est directement fait appel. Il est toujours préférable de donner leur chemin d'accès complet et de se débrouiller pour que la procédure d'installation modifie la partie répertoire de ces chemins dans la page de manuel, en fonction des préférences de l'utilisateur. Pour citer un exemple, l'application GNU **`groff`** s'installe par défaut dans */usr/local*, les pages de manuel référencent donc */usr/local/lib/groff/**. Mais si vous l'installez par « **`make prefix=/opt/gnu`** », toutes les pages de manuel seront modifiées pour indiquer */opt/gnu/lib/groff/**.

La section `ENVIRONNEMENT`

Indique toutes les variables d'environnement prises en compte par votre programme ou votre fonction, et leur action. La plupart du temps, ces variables contiennent des chemins d'accès, des noms de fichiers ou des options par défaut.

La section `DIAGNOSTICS`

Elle doit donner un aperçu des messages d'erreur les plus courants susceptibles d'être affichés par le programme, et les éventuelles solutions à ces problèmes. Il n'est pas nécessaire de citer les messages d'erreur du système (issus de **`perror(3)`**) ou fatals (issus de **`psignal(3)`**) car ils peuvent apparaître durant l'exécution de n'importe quel programme.

La section `BOGUES`

Dans un monde idéal, cette section n'aurait pas lieu d'être ! Si vous êtes brave, vous pourrez décrire ici les limites et les inconvénients de votre application, voire les caractéristiques que certains pourraient considérer comme des défauts. Sinon, faites la même liste mais renommez cette section « **`A FAIRE`** »...

La section `AUTEUR`

Elle est très utile pour savoir qui a fait quoi, et pour trouver l'adresse à laquelle envoyer un rapport de bogue en cas d'erreur grossière dans la documentation (ce qui ne devrait pas arriver !) ou d'un comportement anormal du programme.

La section `VOIR AUSSI`

C'est une liste de pages de manuel relatives à l'application, citées par ordre alphabétique. Par convention, c'est la dernière section.

Vous êtes libre d'inventer toute autre section, si le cas à traiter ne peut se classer dans aucune de celles que nous venons d'énumérer. Nous avons volontairement décrit une version francisée de page de manuel, puisque ce document est destiné aux pays francophones. Néanmoins, vous devez avoir conscience que si vous devez diffuser une application dans le monde entier, il vous faudra fournir un manuel en langue anglaise (ce qui est la version standard, traditionnelle), et que les noms « officiels » de ces sections sont en réalité, dans l'ordre : `NAME`, `SYNOPSIS`, `DESCRIPTION`, `OPTIONS`, `FILES`, `ENVIRONMENT`, `DIAGNOSTICS`, `BUGS`, `AUTHOR` et `SEE ALSO`.

Tout cela est très bien, mais comment générer cette belle page de manuel ? Nous nous attendions à cette question ! Alors, voici le code source complet :

```
.\" Formater ce fichier par la commande :
.\" groff -man -Tlatin1 prout.1  (si vous avez saisi des accents Iso-8859-1)
.\" groff -man -Tascii  prout.1  (cas general )
.\"
.TH PROUT 1 "JANVIER 1996" Linux "Manuel utilisateur"
.SH NAME
prout \- proutibule la bibliotheque plaf
.SH SYNOPSIS
.B prout [-plaf] [-c
.I fichier-config
.B ]
.I fichier
.B ...
.SH DESCRIPTION
.B prout
proutibule la bibliotheque plaf en mouglifiant la table des symboles.
Par defaut, la commande recherche tous les segments glurb et les trie
par ordre betagonique decroissant afin que le gloupeur
.BR gloup (1)
les trouve. L'entree symdef est alors compactee selon l'algorithme NABOB.
Les fichiers sont traites dans leur ordre d'apparition sur la ligne
de commandes.
.SH OPTIONS
.IP -b
N'affiche pas 'bidouille en cours' sur la sortie standard pendant
le traitement.
.IP "-c fichier-config"
Utilise le fichier de configuration
.I fichier-config
au lieu du fichier global
```

```
.IR /etc/prout.conf .
Cela supprime aussi l'effet de la variable d'environnement
.B PROUTCONF.
.IP -a
Traite egalement les en-tetes froutz en plus des segments glurb.
.IP -r
Mode recursif. Fonctionne a la vitesse de la lumiere, mais necessite
plusieurs megaoctets de memoire virtuelle.
.SH FICHIERS
.I /etc/prout.conf
.RS
Fichier de configuration general, pour tout le systeme. Voir
.BR prout (5)
pour plus de details.
.RE
.I ~/.proutrc
.RS
Fichier de configuration propre a chaque utilisateur. Voir
.BR prout (5)
pour plus de details.
.SH ENVIRONNEMENT
.IP PROUTCONF
Si elle existe, cette variable peut contenir le chemin d'acces complet
a un autre fichier de configuration global
.IR prout.conf .
L'option
.B -c
rend cette variable inoperante.
.SH DIAGNOSTICS
Les messages suivants peuvent etre affiches sur la sortie standard d'erreurs :

Mauvais nombre magique.
.RS
Le fichier d'entree ne semble pas etre un fichier archive.
.RE
Segments glurb ancien style.
.RS
.B prout
ne peut traiter que le nouveau style de segments glurb. Les bibliotheques
GROBOL ne sont pas supportees dans cette version.
.SH BOGUES
Le nom de cette commande aurait du etre choisi de maniere a mieux
refleter sa fonction.
.SH AUTEUR
```

```
Marcel Dugenou    <dugenou@renux.freenix.fr>
.SH "VOIR AUSSI"
.BR gloup (1),
.BR plaf (1),
.BR prout (5).
```

4 Comment documenter plusieurs choses dans une seule page?

De nombreux programmes (**grep**, **egrep**) et fonctions (**printf**, **fprintf**, ...) sont décrits dans un seul document. Toutefois, cette documentation serait quasiment inutile si elle n'était accessible que sous un seul nom. Nous ne pouvons pas demander aux utilisateurs de mémoriser que la page de manuel de **egrep** est en réalité celle de **grep**. Il est par conséquent indispensable que la page soit accessible sous plusieurs noms différents. Il existe plusieurs possibilités pour obtenir cela :

1. réaliser plusieurs copies du document, une par nom ;
2. connecter toutes ces pages par des liens physiques ;
3. plusieurs liens symboliques pointant vers un unique document ;
4. faire appel au mécanisme de « source » de **groff**, en employant la macro-instruction « **.so** ».

La première solution est une perte de place, il faut l'éviter. La seconde n'est pas recommandée, car les versions « intelligentes » de l'utilitaire **catman** peuvent gagner beaucoup de temps en détectant le type ou le contenu du fichier. Les liens matériels réduiront l'efficacité de cet outil (dont le but est de formater à l'avance toutes les pages de manuel existantes afin qu'elles s'affichent toutes rapidement, sans que l'utilisateur doive attendre le temps nécessaire à ce traitement éventuel). La troisième solution comporte un piège : si vous êtes concerné par la portabilité, vous devez savoir qu'il existe des systèmes de fichiers ne connaissant pas les liens symboliques. En conclusion, la meilleure chose à faire consiste à employer la macro « **.so** ».

Voici comment opérer : si vous désirez que votre page de manuel soit accessible par les noms « truc » et « bidule » dans la section 1, mettez le document dans **truc.1** et réalisez un fichier **bidule.1** contenant ceci :

```
.so man1/truc.1
```

Il est très important de ne pas oublier le répertoire *man1*, car lors de l'exécution de **groff**, celui-ci aura comme répertoire courant le répertoire de base des pages de manuel, et il interprètera les arguments à **.so** comme étant relatifs à cet emplacement.

5 Quel ensemble de macros choisir?

Il existe un certain nombre de jeux de macros étudiés spécialement pour la confection de pages de manuel. On les trouve généralement dans le répertoire */usr/lib/groff/tmac*. Les noms des fichiers sont du genre *tmac.*<*quelque-chose*>, où <*quelque-chose*> est l'argument passé à l'option **-m** de **groff**. Souvent, l'espace

entre `-m` et <*quelque-chose*> est omis, de sorte que l'on peut écrire `groff -man` lorsque l'on désire formater des pages utilisant le jeu de macros *tmac.an*. Voilà la raison de ce nom bizarre, « *tmac.an* ».

En plus de ce *tmac.an*, il existe un autre ensemble de macros très populaire, *tmac.doc*, qui provient de l'université de Berkeley, Californie (UCB). De nombreuses pages de manuel BSD l'utilisent et il semble que l'UCB en ait fait un standard pour la documentation. Les macros *tmac.doc* sont bien plus souples mais malheureusement, beaucoup d'outils de lecture du manuel n'y font pas appel et utilisent toujours la commande `groff -man`. Par exemple, tous les programmes `xman` que nous avons rencontrés affichent n'importe quoi lorsqu'ils rencontrent des pages prévues pour ces macros *tmac.doc*. Alors, vous devez vous résigner : n'utilisez que *tmac.an* (l'emploi de tout autre ensemble de macros est très risqué). Le jeu *tmac.andoc* est un pseudo ensemble de macros qui analyse le code source et charge soit *tmac.an*, soit *tmac.doc* selon son contenu. En fait, tous les programmes d'exploitation du manuel devraient l'utiliser, mais jusqu'à présent peu le font, aussi il vaut mieux assurer le coup en se cantonnant au bon vieux *tmac.an*. À partir de maintenant, tout ce que nous décrirons concernant les macros ne sera valable que pour *tmac.an*.[2]

6 Quels préprocesseurs utiliser ?

`Groff` est fourni avec au moins trois préprocesseurs : `tbl`, `eqn` et `pic` (sur certains systèmes ils se nomment `gtbl`, `geqn` et `gpic`). Ils sont destinés à traduire des macro-instructions de pré-traitement et leurs arguments en code source `troff` standard. Le programme `tbl` est un préprocesseur de tableaux, `eqn` gère les équations et les mathématiques en général, et enfin `pic` est un préprocesseur d'images. Consultez leurs pages de manuel respectives pour découvrir les fonctionnalités qu'ils proposent.

Mais autant être clair : n'écrivez jamais de pages de manuel nécessitant quelque préprocesseur que ce soit.

`Eqn` produira généralement une sortie catastrophique pour les périphériques du genre télétype ; malheureusement 99% des terminaux sur lesquels sont lues les pages de manuel en font partie. Par exemple, *XAllocColor.3x* contient quelques expressions mathématiques avec des exposants. En raison de la nature des périphériques d'affichage, ces exposants se retrouveront sur la même ligne que le texte de base : « *N puissance deux* » s'affichera « `N2` ».

Il faut éviter `tbl` car tous les programmes `xman` que nous avons rencontrés ne fonctionnent pas avec lui. `Xman 3.1.6` utilise la ligne de commande suivante pour formater les pages de manuel, par exemple `signal(7)` :

```
gtbl /usr/man/man7/signal.7 | geqn | gtbl | groff -Tascii -man \
     > /tmp/xmana01760 2> /dev/null
```

Elle coince sur toutes les sources utilisant `gtbl`, car sa sortie est renvoyée une fois de plus dans son entrée. Le résultat donne une page de manuel sans le tableau voulu. Nous ne savons pas s'il s'agit d'un bogue ou d'une particularité de `gtbl`, qui n'aime pas sa propre sortie, ou si `xman` devrait être un peu plus gentil et ne pas l'appeler deux fois... Quoi qu'il en soit, si vous voulez un tableau, formatez-le vous-même et insérez-le entre deux lignes `.nf` et `.ni` de sorte qu'il ne soit pas traité plus tard. Vous ne pourrez pas obtenir de gras

2. Si vous voulez malgré tout employer les macros *tmac.andoc*, voici un pointeur vers leur documentation et un mode d'emploi très détaillé : (`http://www.bsdi.com/bsdi-man`). Vous trouverez un formulaire de recherche sur cette page. Entrez `mdoc` et il vous trouvera `mdoc(7)` et `mdoc.samples(7)`, un didacticiel sur la réalisation des pages de manuel BSD.

ou d'italiques par cette méthode, mais cela vous évitera de voir votre beau tableau disparaître au hasard des systèmes d'exploitation employés.

Nous n'avons encore jamais rencontré de page de manuel nécessitant un traitement par `pic`. Mais nous n'aimerions pas ça ! Comme vous pouvez le voir dans l'exemple ci-dessus, `xman` ne l'appelle jamais et les données lui étant destinées donneraient sans doute le vertige à ce pauvre `groff`.

7 Distribution : code source et/ou documentation formatée ?

Voyons les avantages `(+)` et les inconvénients `(-)` de quelques combinaisons possibles :

1. code source uniquement :
   - `(+)` taille de la distribution très réduite ;
   - `(-)` inutilisable sur les systèmes démunis de `groff`.
2. version formatée non compactée uniquement :
   - `(+)` utilisable même sur les systèmes dépourvus de `groff` ;
   - `(-)` l'utilisateur ne peut pas générer de fichier `DVI` ou *PostScript* ;
   - `(-)` gâchis de place disque sur les systèmes sachant aussi gérer les pages compactées.
3. version formatée et compactée uniquement :
   - `(+)` utilisable même sur les systèmes dépourvus de `groff` ;
   - `(-)` l'utilisateur ne peut pas générer de fichier `DVI` ou *PostScript* ;
   - `(-)` quel format de compactage utiliser? `.Z`? `.z`? `.gz`? Tous les trois?
4. code source plus version formatée non compactée :
   - `(+)` accessible même sur les systèmes démunis de `groff` ;
   - `(-)` taille de la distribution plus importante ;
   - `(-)` certains systèmes peuvent nécessiter des pages formatées compressées ;
   - `(-)` informations redondantes sur les systèmes équipés de `groff`.

À notre avis, la meilleure solution est de distribuer uniquement le code source. Le fait que cette documentation ne pourra pas être exploitée sur une machine sur laquelle `groff` n'est pas installé n'a aucune importance. Plus de 500 pages de manuel du *projet de documentation Linux* ne sont fournies que sous forme de code source. Les pages de manuel de `XFree86` ne sont disponibles que sous forme de code source. Les pages de manuel de la `FSF` n'existent que sous forme de code source. En fait, nous n'avons pratiquement jamais rencontré de logiciels distribués avec leurs pages de manuel préformatées. Si un administrateur a vraiment besoin d'accéder aux pages de manuel de son système, il aura forcément installé `groff`.

8 Fontes disponibles et conventions typographiques

N'utilisez jamais des opérateurs directs comme `\fB \fP`, etc. Employez des macros qui prennent des arguments, vous éviterez ainsi l'erreur classique : l'oubli du changement de fonte à la fin d'un mot, qui provoque la continuation du gras ou de l'italique jusqu'au changement suivant. Croyez-nous, c'est une erreur bien plus fréquente que vous ne pouvez l'imaginez !

Les macros *tmac.an* offrent les possibilités suivantes :

```
.B    caracteres gras
.BI   gras et italiques en alternance
.BR   gras et romain en alternance
.I    italiques
.IB   italiques et gras en alternance
.IR   italiques et romain en alternance
.RB   romain et gras en alternance
.RI   romain et italiques en alternance
.SM   taille reduite (9/10 du corps normal)
.SB   gras, taille reduite (tout le texte, pas d'alternance)
```

X en alternance avec Y signifie que les arguments impairs seront imprimés en X et les pairs en Y. Par exemple :

```
.BI "Arg 1 est gras, " "arg2 est en italiques, " "arg3 est en gras."
```

Les caractères de protection (`" "`) sont nécessaires pour placer des espaces dans ces arguments.

Voilà donc pour ce qui est disponible. Voyons maintenant comment il faut utiliser ces possibilités (nous avons honteusement copié ce qui est décrit dans `man(7)`) :

Bien qu'il existe dans le monde UNIX de nombreuses conventions typographiques arbitraires pour la réalisation de pages de manuel, plusieurs centaines de pages spécifiques à Linux se conforment au standard suivant :

Les fonctions et les arguments sont toujours indiqués en italiques, même dans la section `SYNOPSYS`, le reste de la fonction étant imprimé en corps gras :

```
.BI "mafonction(int " argc ", char **" argv );
```

Les noms de fichiers sont toujours en italiques, sauf dans la section `SYNOPSYS`, où les fichiers à inclure sont en gras. Vous écrirez donc :

```
.I /usr/include/stdio.h
.B #include <stdio.h>
```

Les noms de macros, qui sont la plupart du temps en majuscules, sont typographiés en gras :

```
.B MAXINT
```

Dans les énumérations de codes d'erreur, les codes sont en gras. Cette liste fait généralement appel à la macro `.TP` (paragraphe avec titre) comme ci-dessous :

```
.TP
.B EBADF
.I fd
n'est pas un descripteur de fichier valide.
.TP
.B EINVAL
.I fd
a une valeur incorrecte.
```

Toute référence à une autre page de manuel (ou au sujet de la page courante) est en gras. Si la section correspondante du manuel est indiquée, elle s'écrit en roman, sans espace :

```
.BR man (7)
```

Les acronymes sont plus élégants lorsqu'ils apparaissent dans un corps plus petit, nous vous recommandons par conséquent d'employer :

```
.SM UNIX     .SM ASCII     .SM TAB      .SM NFS      .SM LALR(1)
```

9 Conseils, trucs et astuces

9.1 Peaufinez votre page de manuel

Voici quelques conseils pour améliorer la lisibilité, la validité des informations fournies et la facilité de formatage de votre documentation :

- vos exemples doivent fonctionner, testez-les (utilisez le couper/coller pour passer à votre shell le code exact indiqué dans la page de manuel), redirigez l'affichage dans votre fichier source (ne tapez pas vous-même ce que vous pensez que votre programme affichera) ;
- relisez-vous, faites relire votre document par des tiers, corrigez toutes les éventuelles fautes de frappe ou d'orthographe (surtout si vous ne rédigez pas le texte dans votre langue natale) ;
- testez votre page : est-ce que **groff** trouve des erreurs lors du formatage ? Il est agréable de trouver dans un commentaire la commande nécessaire pour formater le document. Est-ce que **man(1)** affiche des avertissement ou des erreurs lorsque vous tapez « **man votre-programme** » ? Est-ce que la façon dont **man(1)** utilise les outils de formatage produit les résultats escomptés ? Est-ce que cela fonctionne aussi bien avec **xman(1x)** et **tkman(1tk)** ? **XFree86 3.1** contient la version **3.1.6** de **xman**, qui décompacte les pages par ces commandes :

  ```
  gzip -c -d < %s > %s
  zcat < %s > %s
  ```

- est-ce que **makewhatis(8)** arrive a extraire les descriptions sur une ligne de la section **NAME** ?

9.2 Comment obtenir une sortie pur ascii sans ces fichus caractères de contrôle ?

Jetez un coup d'oeil à `col(1)` ; il s'agit d'un filtre qui peut éliminer ces séquences de surimpression. Au cas où vous seriez trop paresseux pour réfléchir vous-même au problème, voici la commande magique :

```
$ groff -t -e -mandoc -Tascii manpage.1 | col -bx > manpage.txt
```

Les options `-t` et `-e` indique à `groff` de faire appel aux préprocesseurs `tbl` et `eqn`. C'est inutile pour les pages de manuel ne nécessitant aucun pré-traitement, mais cela ne gêne pas, si ce n'est par la petite charge supplémentaire induite. En revanche, ne pas utiliser `-t` lorsque ce serait nécessaire est une très mauvaise chose : le tableau sera extrêmement mal formaté. Vous pourrez même trouver (deviner serait un terme plus exact) la commande nécessaire pour traiter tel ou tel document `groff` (pas uniquement des pages de manuel) par le biais de `grog` :

```
$ grog /usr/man/man7/signal.7
groff -t -man /usr/man/man7/signal.7
```

En fait, « `grog` » signifie « *GROff Guess* », et cet outil fait bien ce qu'il dit (en anglais...) : il tente de deviner la commande nécessaire pour formater un document `groff` en fonction de son contenu. S'il était parfait, nous n'aurions jamais plus besoin d'options. Mais il arrive qu'il détermine un mauvais jeu de macros ; nous ne l'avons par contre jamais vu se tromper sur les préprocesseurs à employer.

Voici un petit script `perl` réalisé par l'auteur de ce document, qui peut supprimer les en-têtes et les pieds de page, ce qui permet de gagner quelques feuilles de papier lorsque l'on imprime de longues et complexes pages de manuel. Sauvez-le dans un fichier nommé *strip-header* et donnez-lui le mode 755.

```
#!/usr/bin/perl -n
#  make it slurp the whole file at once:
undef $/;
#  delete page breaks:
s/\n{4}\S.{50,}\n{6}\S.{50,}\n{3}/\n/g;
#  delete first header & last footer:
s/\n\S.{50,}\n//g;
#  collapse two or more blank lines into a single one:
s/\n{3,}/\n\n/g;
#  see what's left...
print;
```

Il faut appeler ce programme en tant que premier filtre après la comande `man`, car il se base sur le nombre de sauts de ligne issus de `groff`. Par exemple:

```
$ man bash | strip-headers | col -bx > bash.txt
```

9.3 Comment obtenir une sortie *PostScript* de haute qualité ?

C'est très simple :

```
$  groff -t -e -mandoc -Tps manpage.1 > manpage.ps
```

Imprimez *manpage.ps* avec votre imprimante PostScript préférée. Consultez la section précédente pour obtenir des explications sur les options.

9.4 Comment faire fonctionner les commandes `apropos` et `whatis`?

Supposons que vous recherchiez quels sont les compilateurs installés sur votre système, et comment les invoquer(nous considérons le cas courant, ou tout le manuel est en langue anglaise). Pour répondre à cette question (fréquemment posée), il faut faire :

```
$ apropos compiler
f77 (1) - Fortran 77 compiler
gcc (1) - GNU C and C++ compiler
pc (1) - Pascal compiler
```

Les commandes **apropos** et **whatis** sont destinées à obtenir rapidement le nom de la page de manuel traitant de tel ou tel sujet. Tous deux font appel à un certain nombre de fichiers nommés « *whatis* », que l'on peut trouver dans chaque répertoire de base du manuel. Comme nous l'avons dit auparavant, cette base de données *whatis* contient une entrée d'une ligne pour chaque page de manuel présente dans l'arborescence correspondante. En fait, cette ligne est exactement celle de la section **NAME** (pour être précis, tout est réduit à une seule ligne et le tiret est supprimé, la section étant placée entre parenthèses). Les fichiers constituant cette base de données sont créés par le programme **makewhatis(8)**. Il en existe plusieurs versions, par conséquent nous ne pouvons que vous renvoyer à leur propre documentation pour leur utilisation. Afin que cet outil fonctionne correctement, il est nécessaire que vous, le rédacteur des pages de manuel, vous conformiez au format de la section **NAME** décrit plus haut dans ce document.

La différence entre **apropos** et **whatis** est le mode de recherche et l'affichage obtenu : **apropos** (qui est équivalent à **man -k**) recherche la chaîne passée en argument à n'importe quel endroit de la ligne, alors que **whatis** (équivalent à **man -f**) recherche un nom de commande complet, uniquement sur la partie située avant le tiret. Par conséquent, « **whatis cc** » indiquera s'il existe une page de manuel pour **cc**, mais ne dira rien à propos de **gcc**.

9.5 La langue française

C'est bien sûr à vous de décider si vous allez rédiger votre manuel en français, en anglais ou dans ces deux langues. Le français possède des règles typographiques très différentes de l'anglais: n'espérez pas pouvoir les respecter avec les outils de formatage du manuel. Consultez la documentation de **groff** si vous désirez lui faire prendre en compte les motifs de césure de la langue de Molière, mais en ayant conscience que ce ne sera sans doute pas possible sur tous les systèmes sur lesquels votre documentation est susceptible d'être exploitée.

Vous pouvez utiliser les caractères accentués, pourvu qu'ils soient saisis selon la norme ISO-8859-1 (standard sous Linux). Les pages devront alors être formatées avec l'option **-Tlatin1** . Mais il faudra que toute la

chaîne de visualisation soit capable de gérer les caractères ISO sur 8 bits, ce qui est rarement le cas sans une configuration particulière des utilitaires « `more` » ou « `less` » généralement employés.

Vous voilà prévenu !

10 Copyright

Ce document est copyright (C) 1995 Jens Schweikhardt <`jens@kssun3.rus.uni-stuttgart.de`>. Il peut être reproduit et distribué en tout ou partie, sur tout support physique ou électronique, à condition que cette notice soit incluse dans chacune des copies. Sa diffusion commerciale est autorisée et encouragée ; toutefois l'auteur désire être prévenu dans ce cas.

Toute traduction, travail dérivé, ou compilation de travaux incluant quelque document Linux *HOWTO* que ce soit doit être couvert par ce présent copyright. C'est-à-dire que vous ne pouvez pas réaliser un ouvrage dérivé d'un *HOWTO* et imposer des restrictions supplémentaires sur sa distribution. Des dérogations à ces règles peuvent être accordées sous certaines conditions ; contactez le coordinateur à l'adresse donnée ci-après pour plus de précisions.

En résumé, nous avons le désir de promouvoir la diffusion de ces informations par le plus de canaux possibles. Malgré tout, nous voulons conserver la propriété des documents *HOWTO*, et aimerions être tenu au courant de tout projet de redistribution.

Si vous avez des questions, contactez Greg Hankins, le coordinateur, par courrier électronique à l'adresse `gregh@sunsite.unc.edu`, ou par téléphone au +1 404 853 9989.