Logiciel

Le Guide du Hacker Noweb (The noweb Hacker’s Guide)

Norman Ramsey

Department of Computer Science

Princeton University

September 1992

(Revised August 1994, December 1997)

Introduction

Ramsey (1994) décrit noweb du point de vue d'un utilisateur, montrant sa simplicité et des exemples d'utilisation.

Les outils de noweb sont implémentés en tant que *tubes*.

Chaque tube commence par le fichier source noweb.

Les étapes successives du tube mettent en œuvre des transformations simples de la source, jusqu'à ce que le résultat souhaité émerge de la fin du tube.

Les figures 1 et 2 montrent les tubes pour notangle et noweave.

Les tubes permettent d'étendre Noweb, ce qui permet de créer de nouvelles fonctionnalités de programmation sans avoir à écrire leurs propres outils.

Ce document explique comment modifier ou étendre noweb en insérant ou en supprimant des étapes de tube.

Les lecteurs doivent se familiariser avec les pages man de noweb, qui décrivent la structure des fichiers sources de noweb.

Le balisage, qui est la première étape de chaque tube, convertit la source de noweb en une représentation facilement manipulable par des outils Unix communs comme sed et awk, simplifiant la construction des étapes ultérieures du tube.

Les étapes intermédiaires ajoutent des informations à la représentation.

La dernière étape de notangle convertit en code ; Les dernières étapes de noweave convertissent en TeX, LaTeX ou HTML.

Les étapes intermédiaires sont appelées filtres, par analogie avec les filtres Unix.

Les étapes finales sont appelées back ends, par analogie avec les compilateurs - elles ne transforment pas la représentation intermédiaire de noweb ; elles émettent autre chose.

La représentation en tube

Dans le tube, chaque ligne commence par un signe arobase et l'un des mots-clés du tableau 1.

Les **mots-clés structurels** représentent directement la syntaxe source Noweb. Ils doivent apparaître dans un ordre qui reflète la structure de la source.

Les **mots-clés de marquage** peuvent être insérés n'importe où (dans des limites raisonnables) et, à quelques exceptions près, ils ne sont pas générés par le balisage.

Les **mots-clés wrapper** marquent le début et la fin du fichier, et contiennent des informations sur le formatage à faire en début et en fin de bloc. Ils sont utilisés par noweave mais pas par notangle et ils sont insérés directement par le script shell noweave, et non par le balisage.

Mots-clés structurels

Les mots-clés structurels représentent les morceaux de la source Noweb.

Chaque morceau est entouré d'une paire **@begin . . . @end** et le type de morceau est soit docs soit code. Les @begin et @end sont numérotés ; dans un fichier unique, les numéros doivent être croissants mais n'ont pas besoin d'être consécutifs. Les filtres peuvent changer les numéros de bloc à volonté.

Selon son genre, un morceau peut contenir de la documentation ou du code.

La documentation peut contenir du texte et des retours à la ligne, représentés par @text et @nl.

Elle peut aussi contenir du code cité, encadré par @quote . . . @endquote. Chaque @quote doit être terminée par @endquote dans le même bloc. Le code cité correspond à la construction [[. . .]] du source de noweb.

Mots-clés structurels	
@begin kind n	Début d'un morceau
@end kind n	Fin d'un morceau
@text string	chaîne string dans un morceau
@nl	Une nouvelle ligne dans un morceau
@defn name	Définition du bloc de code nommé name
@use name	Une référence au bloc de code nommé name
@quote	Début du code cité dans un bloc de documentation

Mots-clés structurels	
@endquote	Fin du code cité dans un bloc de documentation
Mots-clés de marquage	
@file filename	Nom du fichier d'où proviennent les morceaux
@line n	La ligne de texte suivante provient de la ligne source n du fichier en cours
@language language	Langage de programmation dans lequel est écrit le code
@index ...	Informations d'index.
@xref ...	Informations de référence croisée
Mots-clés wrapper	
@header formatter options	Première ligne, identifiant le formateur et les options
@trailer formatter	Dernière ligne, identifiant le formateur.
Mot clé d'erreur	
@fatal stagename message	Une erreur fatale s'est produite.
Lying, cheating, stealing keyword	
@literal text	Copier le texte sur la sortie.

Tableau 1: Mots clés utilisés dans la représentation tube de Noweb

Le code, qu'il apparaisse dans le code comme code cité ou dans un bloc de code, peut contenir du texte et des retours à la ligne, ainsi que des définitions et des utilisations de fragments de code, marqués avec @defn et @use.

Le premier mot-clé structurel d'un bloc de code doit être @defn.

@defn peut être précédé ou suivi par des mots clé étiquettes, mais le mot clé structurel suivant doit être **@nl** ; ensemble, **@defn** et **@nl** représentent la ligne **<<chunk name>>=** qui commence le morceau (y compris le retour chariot final).

Il découle de ce qui vient d'être dit que :

- Le code cité peut ne pas apparaître dans le code, ni dans **@defn** ni **@use**. Il est recommandé que noweave traite spécialement les back ends **[[. . .]]** quand il apparaît dans **defn** ou **use**, pour que le texte qu'il contient soit traité comme s'il s'agissait d'un code cité.
- Le texte dans les morceaux peut être distribué en autant de mots-clés **@text** que désiré. Un nombre quelconque de mots-clés **@text** vides sont autorisés. En particulier, il n'est pas réaliste d'espérer qu'une seule ligne sera représentée dans un seul **@text** (voir la discussion de finduses à la page 14).
- **markup** va parfois émettre **@use** dans @**@quote… @endquote**, par exemple à partir d'une

source comme **[[<<chunk name>>]]**.

- deux morceaux ne peuvent pas avoir le même numéro.
- comme les filtres suivants peuvent changer les numéros de bloc, aucun filtre ne doit placer des références aux numéros de bloc dans le tube.

Mots-clés de marquage

Les mots-clés structurels portent tout le code et la documentation qui apparaît dans un fichier source Noweb.

Les mots-clés de marquage contiennent des informations sur ce code ou cette documentation.

Le mot clé **@file** porte le nom du fichier source d'où viennent les lignes suivantes.

Le mot clé **@line** donne le numéro de ligne de la prochaine ligne **@text** suivante dans le fichier en cours (comme déterminé par le mot clé **@file** le plus récent).

La seule garantie de l'endroit où ils apparaissent est que le balisage introduit chaque nouveau fichier source par un **@file** qui apparaît entre les morceaux.

La plupart des filtres ignorent **@file** et **@line** mais **nt** les respecte, de sorte que **notangle** peut correctement numéroter les ligne si un filtre **noweb** déplace des lignes.

Langages de programmation

Pour prendre en charge l'indexation automatique ou l'impression de jolis caractères, il est possible d'indiquer le langage de programmation dans lequel un bloc est écrit. Le mot clé **@language** peut apparaître au plus une fois entre chaque paire **@begin code** et **@end code**. Voici les valeurs standard de **@language** et leurs significations :

awk	awk
c	C
c++	C++
caml	CAML
html	HTML
icon	Icon
latex	source LATEX
lisp	Lisp ou Scheme
make	un Makefile
m3	Modula-3
ocaml	Objective CAML
perl	un script perl
python	Python
sh	un script shell
sml	Standard ML

tex	plain TEX
tcl	tcl

Si le mot-clé **@language** est activé, il peut être utile de créer un registre automatique sur le World-Wide Web.

Il est impossible de placer une information **@language** directement dans un fichier source noweb.

Indexation et référence croisée

Les commandes **index** et **cross-reference** utilisent des *étiquettes*, des *identifiants* et des *balises*.

Une étiquette est une chaîne unique désignant un élément en programmation lettrée.
Ils servent d'étiquettes ou de « points d'ancrage » pour les étapes finales qui peuvent mettre en œuvre leur propre référence croisée.
Par exemple, l'étape finale LATEX utilise des labels comme arguments pour **\label** et **\ref** ; l'étape finale HTML utilise des labels pour nommer des ancres et s'y référer.
Les étiquettes ne contiennent jamais d'espace, ce qui simplifie l'analyse.

Les filtres standards font des références croisées au niveau fragment de code : chaque étiquette se réfère à un fragment de code particulier et toutes les références à ce fragment utilisent la même étiquette.

Un *identifiant* fait référence à un identifiant de langage source.
Dans Noweb, le concept d'identifiant est général.
Un identifiant est une chaîne arbitraire qui peut même contenir des espaces.
Les identifiants sont utilisés comme clés dans l'index ; les références à la même chaîne sont supposées désigner le même identifiant.

Les balises (Tags) sont des chaînes identifiant les composants de référence croisée dans le document final.
Par exemple, Classic WEB utilise des *numéros de section* consécutifs pour désigner des segments.
Par défaut, Noweb utilise des *références de sous-page*, par exemple, *24b* pour le deuxième segment apparaissant à la page 24.
Le backend HTML n'utilise aucune balise ; au lieu de cela, il implémente des références croisées en utilisant le mécanisme du *lien direct*.

La dernière étape de la référence croisée consiste à générer des balises et à associer une balise à chaque étiquette.
Tous les back-ends existants reposent sur un formateur de document pour faire ce travail, mais cette stratégie pourrait être modifiée.
Le calcul des balises dans un filtre Noweb peut être beaucoup plus facile que dans un formateur.
Par exemple, un filtre qui calcule des numéros de sous-pages en cherchant dans des fichiers **.aux** serait assez facile à écrire, et éliminerait beaucoup de code LATEX.

Informations d'index

J'ai divisé les mots-clés d'index en plusieurs groupes. Il semble y avoir pléthore de mots-clés, mais la

plupart sont des représentations simples de parties d'un document produit par noweave.

Les lecteurs voudront peut-être avoir un échantillon de la sortie de noweave à portée de main lors de l'étude de ceci et de la section suivante.

Definitions, uses, et @ %def	
@index defn ident	Le bloc actuel contient une définition d'**ident**
@index localdefn ident	Le bloc actuel contient une définition de ident, qui ne doit pas être visible en dehors de ce fichier
@index use ident	Le bloc actuel contient une utilisation d'ident
@index nl ident	Un retour à la ligne faisant partie du balisage, sans faire partie du bloc
Identificateurs définis dans un bloc	
@index begindefs	Début de la liste des identifiants définis dans ce bloc
@index isused label	L'identifiant nommé dans le **@index defitem** suivant est utilisé dans le bloc étiqueté par label
@index defitem ident	ident est défini dans ce fragment et utilisé dans tous les morceaux nommés dans le @index immédiatement précédent.
@index enddefs	Fin de la liste des identifiants définis dans ce bloc
Identifiants utilisés dans un bloc	
@index beginuses	Début de la liste des identifiants utilisés dans ce bloc
@index isdefined label	The identifier named in the following @index useitem is defined in the chunk labelled by label
@index useitem ident	ident is used in this chunk, and it is defined in each of the chunks named in the immediately preceding @index isdefined.
@index enduses	Fin de la liste des identifiants utilisés dans ce bloc
L'index des identifiants	
@index beginindex	Début de l'index des identifiants
@index entrybegin label ident	Début de l'entrée pour ident, dont la première définition se trouve à label

Definitions, uses, et @ %def	
@index entryuse label	Une utilisation de l'identificateur nommé dans le dernier @index entrybegin apparu au niveau du segment étiqueté label.
@index entrydefn label	Une définition de l'identificateur nommé dans le dernier @index entrybegin apparu au niveau du segment étiqueté label.
@index entryend	Fin de l'entrée commencée par la dernière @index entrybegin
@index endindex	Fin de l'index des identifiants

Definitions, uses, et @ %def

%@index defn, **@index use**, et **@index nl** sont les seuls mots-clés d'index qui apparaissent dans la sortie du balisage et qui peuvent donc apparaître dans n'importe quel programme.

Ils ne peuvent apparaître que dans les limites d'un bloc de code (@begin code . . . @end code).

@index defn et **index use** indiquent simplement que le bloc actuel contient une définition ou l'utilisation de l'identifiant ident qui suit le mot-clé.

L'emplacement de **@index defn** n'a pas besoin d'être en relation avec le texte de la définition, mais **@index use** est normalement suivie par un **@text** qui contient le texte du code source identifié comme étant utilisé.

Les instances de **@index defn** proviennent normalement de l'une de ces deux sources : soit un identificateur de définitions dépendant du langage, soit une ligne **@ %def** manuscrite.

Dans ce dernier cas, la ligne se termine par un retour chariot qui ne fait partie ni d'un bloc de code ni d'un bloc de documentation. Pour que les numéros de ligne restent précis, ce retour chariot ne peut pas être simplement abandonnée, mais elle ne peut pas non plus être représentée par **@nl** dans un bloc de documentation ou de code.
La solution est le mot-clé **@index nl**, qui n'a d'autre but que de garder une trace de ces retours chariot, pour que le back-end produise des numéros de ligne précis.

@index localdefn indique une définition qui ne doit pas

être visible en dehors du fichier courant.
Il peut être produit par un dispositif de reconnaissance dépendant du langage ou un autre filtre.

Identificateurs définis dans un bloc

Les mots-clés de **@index begindefs** jusqu'à **@index enddefs** sont utilisés pour représenter une structure de données plus complexe donnant la liste des identifiants définis dans un bloc de code.

La constellation représente une liste d'identifiants ; un **@index defitem** apparaît pour chaque identifiant.

Le groupe indique également dans quels autres morceaux chaque identifiant est utilisé ; ces morceaux sont listés par **@index isused keywords** qui apparaissent juste avant **@index defitem**.

Les étiquettes de ces mots-clés apparaissent dans l'ordre des blocs de code correspondants et il n'y a pas de doublons.

Ces mots clés peuvent apparaître n'importe où dans un bloc de code, mais il vaut mieux, dans les filtres, conserver ces mots clés ensemble.

Les filtres standards garantissent que seulement **@index isused** et **@index defitem** apparaît entre @index begindefs et @index enddefs.

Les filtres standards les placent à la fin du bloc de code, ce qui simplifie la traduction par le backend LATEX, mais cette stratégie pourrait changer à l'avenir.

Cela devrait aller de soi, mais les mots-clés dans ces groupes et tous les groupes similaires (y compris certains groupes @xref doivent être correctement structurés. C'est-à-dire :

Chaque @index begindefs doit avoir un @index enddefs correspondant dans le même bloc de code.

@index isused et @index defitem ne peuvent apparaître que entre @index begindefs et @index enddefs correspondants.

Les fichues choses ne peuvent pas être imbriquées.

Identifiants utilisés dans un bloc

Les mots-clés de @index beginuses à @index enduses sont le dual de @index begindef à @index enddef ; la structure répertorie les identifiants utilisés dans le bloc de code actuel, avec des références croisées aux définitions.

Des interprétations et restrictions similaires s'appliquent. Notez qu'un identificateur peut être défini dans plus d'un segment, bien que ce soit inhabituel.

L'index des identificateurs

Les mots-clés @index beginindex à @index endindex représentent l'index complet de tous les identifiants utilisés dans le document.

Chaque entrée de l'index est encadrée par @index entrybegin . . . @index entryend.

Une entrée fournit le nom de l'identifiant, plus les étiquettes de tous les morceaux dans lesquels l'identifiant est défini ou utilisé.

L'étiquette du premier bloc de définition est donnée au début de l'entrée de sorte que les back ends n'ont pas besoin de la rechercher.

Il est recommandé que les filtres gardent ces mots-clés ensemble.

Les filtres standards les placent presque à la toute fin du fichier noweb, juste avant le @trailer optionnel.

Informations de référence croisée

La fonction la plus fondamentale des mots-clés de références croisées est d'associer des étiquettes et des pointeurs (références croisées) avec des éléments du document, ce qui est fait avec les mots-clés @xref ref et @xref label. Les autres mots-clés @xref expriment des informations de références croisées de segments qui sont émises directement par un ou plusieurs back-ends.

La référence croisée entre blocs introduit l'idée d'une ancre, qui est une étiquette qui fait référence à un « point intéressant » que nous identifions au début d'un fragment de code.

L'ancre est l'endroit où nous nous attendons à aller lorsque nous voulons connaître un fragment de code ; sa valeur exacte et son interprétation dépendent du back-end utilisé. Le backend LATEX standard utilise le numéro de sous-page du fragment comme point d'ancrage, mais le backend HTML standard utilise un certain @text du bloc de documentation précédant le bloc de code.

Référence croisée de base	
@xref label label	Associe l'étiquette à l'élément étiqueté.
@xref ref label	Référence croisée de l'article étiqueté à l'article associé à l'étiquette.
Lier les définitions précédentes et suivantes d'un bloc de code	
@xref prevdef label	Le @defn de la définition précédente de ce bloc est associé à label.
@xref nextdef label	Le @defn de la définition suivante de ce bloc est associé à label.
Continuation des définitions du bloc actuel	
@xref begindefs	Démarre « Cette définition se continue en. . . »
@xref defitem label	Donne l'étiquette d'un bloc dans lequel la définition du bloc en cours est continuée.
@xref enddefs	Termine la liste des blocs où la définition est continuée.
Blocs où ce code est utilisé	
@xref beginuses	Démarre « Ce code est utilisé dans. . . »
@xref useitem label	Donne l'étiquette d'un bloc dans lequel ce bloc est utilisé.
@xref enduses	Termine la liste des blocs dans lesquels ce code est utilisé.
@xref notused name	Indique que ce segment n'est utilisé nulle part dans ce document.
La liste des blocs	
@xref beginchunks	Début de la liste des blocs
@xref chunkbegin label name	Début de l'entrée pour le nom du bloc, dont l'ancre se trouve à label.
@xref chunkuse label	Le bloc est utilisé dans le morceau étiqueté à label.
@xref chunkdefn label	Le bloc est défini dans le bloc étiqueté avec label.
@xref chunkend	Fin de l'entrée commencée par le dernier

Référence croisée de base	
@xref endchunks	Fin de la liste des blocs
Conversion d'étiquettes en mots-clés	
@xref tag label tag	Associe label avec tag.

Référence croisée de base

@xref label et @xref ref sont nommés par analogie avec les commandes LATEX \label et \ref. L'étiquette @xref est utilisée pour associer une étiquette à un élément suivant.

Les articles qui peuvent être ainsi étiquetés comprennent

@defn	Étiquette le fragment de code qui commence ainsi
@use	Étiquette cet usage particulier.
@index defn	Étiquette cette définition d'un identifiant.
@index use	Étiquette cette utilisation d'un identifiant.
@text	Étiquette généralement une partie d'un bloc de documentation.
@end docs	Étiquette généralement un bloc de documentation vide.

La plupart des back ends utilisent le bloc comme unité de référence croisée, donc les labels de @defn sont les plus souvent utilisés.

Le backend HTML, cependant, fait quelque chose d'un peu différent. Il utilise des étiquettes qui se réfèrent à la documentation précédant un bloc car que le navigateur HTML typique (Mosaic) place l'étiquette (La terminologie HTML appelle un label « anchor »). en haut de l'écran, et utiliser l'étiquette de @defn perdrait la documentation précédant immédiatement un bloc.

Les étiquettes utilisées par ce back-end pointent généralement vers @text, mais elles peuvent pointer vers @end docs lorsque aucun texte n'est disponible.

@xref ref est utilisé pour associer une référence à un élément suivant. Ces articles comprennent

@defn, @use	Fait référence à l'étiquette utilisée comme ancre pour ce bloc.
@index defn, @index use	Fait référence à l'étiquette utilisée comme ancre pour le premier bloc dans lequel cet identifiant est défini.

Lier les définitions précédentes et suivantes d'un bloc de code

@xref prevdef et @xref nextdef peuvent apparaître n'importe où dans un bloc de code, et ils donnent, le cas échéant, les étiquettes des définitions précédentes et suivantes de ce bloc de code.

Les filtres standards les placent actuellement au début du bloc de code, suivant le @defn initial, de sorte que l'information peut être utilisée sur la ligne @defn “a la Fraser et Hanson” (1995).

Continuation des définitions du segment actuel

Les mots clés allant de @xref begindefs à @xref enddefs apparaissent dans la première définition de chaque fragment de code. Ils fournissent les informations requises par le 'Cette définition est continuée en. . . 'Message imprimé par le backend LATEX standard. Ils peuvent apparaître n'importe où dans un bloc de code, mais les filtres standard les mettent après tous les @text et @nls, de sorte que les back-ends peuvent simplement imprimer du texte.

Blocs où ce code est utilisé

Les mots-clés de @xref beginuses à @xref enduses sont le dual de @xref begindefs à @xref enddefs ; ils montrent où est utilisé le bloc actuel.

Comme avec @xref begindefs . . . @xref enddefs, ils n'apparaissent que dans la première définition de n'importe quel morceau de code, et ils viennent à la fin.

Parfois, comme pour les segments racine, le code n'est utilisé nulle part, auquel cas @xref notused apparaît au lieu de @xref beginuses . . . @xref enduses.

Le nom du bloc en cours apparaît comme un argument de @xref notused car certains back ends peuvent souhaiter imprimer un message spécial pour les blocs inutilisés - ils peuvent par exemple être écrits dans des fichiers.

La liste des blocs

La liste des blocs, qui est définie par les mots-clés @xref beginchunks . . . @xref endchunks, est l'analogue de l'index des identifiants, mais elle liste tous les blocs de code dans le document et non les identifiants.

Il est mieux que les filtres gardent ces mots-clés ensemble.

Les filtres standards les placent à la fin du fichier noweb, juste avant l'index des identifiants.

Conversion des étiquettes en tags

Aucun des back-ends ne calcule réellement les balises ; ils utilisent tous des moteurs de formatage pour faire le travail.

Le back-end LATEX utilise un package de macros élaboré pour calculer les numéros de sous-pages, et le backend HTML fait en sorte que des « liens chauds » soient utilisés à la place des balises textuelles.

Mots-clés Wrapper

Les mots-clés du wrapper, @header et @trailer, sont spéciaux en ce sens qu'ils ne sont pas générés par le balisage ou par l'un des filtres standard ; à la place, ils sont insérés par le script shell noweave au tout début et à la fin du fichier.

Les back ends standard TEX, LATEX et HTML les utilisent pour fournir un balisage de préambule et de postambule, c'est-à-dire un texte standard qui doit généralement entourer un document.

Ils ne sont pas obligatoires (parfois, vous ne voulez pas ce type de message), mais lorsqu'ils apparaissent, ils doivent être les toutes premières et dernières lignes du fichier, et les noms des formateurs doivent correspondre.

Mot clé d'erreur

Le mot clé d'erreur @fatal indique qu'une erreur fatale s'est produite.

L'étape de tube à l'origine d'une telle erreur donne son nom et un message, et affiche également un message d'erreur standard.

Les filtres qui reçoivent @fatal doivent le copier dans leur sortie et se terminer avec le statut d'erreur.

Le back-end recevant @fatal doit se terminer avec le statut d'erreur. (Ils ne devraient rien écrire sur l'erreur standard puisque cela aura été fait.)

L'utilisation de @fatal permet aux scripts shell de détecter que quelque chose a mal tourné même si le seul statut de sortie auquel ils ont accès est le statut de sortie de la dernière étape d'un tube.

Mot-clé pour mentir, tricher, voler

Le mot-clé @literal est utilisé pour pirater la sortie directement dans les back-end, comme totex et tohtml. Ces back ends se contentent de copier le texte dans leur sortie.

Tangle appliqué au back-end ignore @literal.

Le mot-clé @literal est utilisé par les Maîtres pirates trop paresseux pour écrire de nouveaux back ends. Son utilisation est obsolète.

Filtres standards

Tous les filtres standard, sauf indication contraire, lisent le format de mot clé noweb sur l'entrée standard et l'écrivent sur la sortie standard.

Certains filtres peuvent également utiliser des fichiers auxiliaires.

markup

Strictement parlant, markup est un frontal, pas un filtre, mais j'en discute avec des filtres car il génère la sortie qui est traitée par tous les filtres.

La sortie de markup représente une séquence de fichiers.

Chaque fichier est représenté par une ligne “@file filename”, suivie d'une séquence de blocs.

markup numérote les blocs consécutivement à partir de 0.

Il reconnaît et annule également la séquence d'échappement pour les doubles crochets, par ex. convertir %%“@<<”%% to %%“<<”%%.

Les seuls mots-clés d'étiquetage trouvés dans sa sortie sont @index defn ou @index nl ; malgré ce qui est écrit à ce sujet, @index use n'apparaît jamais.

autodefs.*

J'ai écrit une demi-douzaine de filtres dépendants du langage qui utilisent des heuristiques simples (« analyse fuzzy » si vous préférez) pour essayer d'identifier des définitions intéressantes d'identifiants.

finduses

En utilisant le code fourni par Preston Briggs, ce filtre fait deux passes sur son entrée.

La première passe lit toutes les lignes @index defn et @index localdefn et construit un module de reconnaissance Aho-Corasick pour les identificateurs qui y sont nommés.

La seconde passe copie l'entrée, en recherchant ces identifiants dans chaque ligne @text qui est du code.

Lorsqu'il trouve un identifiant, finduses coupe la ligne @text, insérant @index use immédiatement avant la partie @text qui contient l'identifiant que l'on vient de trouver.

Le comportement décrit dupliquerait les éléments @text chaque fois qu'un identificateur serait un préfixe d'un autre.

L'option de ligne de commande **-noquote** empêche les finduses de rechercher des uses dans du code entre guillemets.

Si finduses reçoit des arguments, il prend ces arguments pour des noms de fichiers, et il lit des listes d'identifiants (un par ligne) à partir des fichiers ainsi nommés, plutôt que sur son entrée.

Cette technique permet à finduses de ne faire qu'un seul passage sur son entrée ; noweave utilise cela pour implémenter l'option -indexfrom.

finduses ne doit pas être exécuté avant les filtres qui, comme les filtres autodefs, s'attendent à ce qu'une ligne soit représentée dans un seul @text.

Les filtres (ou back ends) qui doivent être lancés plus tard, comme les prettyprinters, doivent être préparés pour traiter les lignes fragmentées et en intercalant des tags @index et @xref.

noidx

noidx calcule toutes les informations d'index et de références croisées représentées par les mots-clés @index et @xref

L'option de ligne de commande **-delay** retarde le contenu d'en-tête jusqu'à la fin du premier bloc et amène le contenu final avant le dernier tronçon.

En particulier, il provoque l'émission de la liste des morceaux et de l'index des identifiants avant le dernier bloc.

L'option **-docanchor n** définit l'ancre d'un fragment de code comme suit :

Si un bloc de documentation précède le bloc de code et comporte n lignes ou plus, n lignes à la fin de ce bloc de documentation.

Si un bloc de documentation précède le bloc de code et comporte moins de n lignes, au début de ce bloc de documentation.

Si aucun bloc de documentation ne précède le bloc de code, au début du bloc de code, comme si -docanchor n'avait pas été utilisé.

Back-ends standards

nt

Le back-end nt implémente notangle. Il extrait le programme défini par un bloc de code unique (en développant toutes les utilisations pour former leurs définitions) et écrit ce programme sur la sortie standard. Ses options en ligne de commande sont :

-t	Désactiver l'expansion des tabulations
-tn	Développer les tabulations sur n colonnes.
-R name	Développer le bloc de code nommé name.
-L format	Utiliser format comme chaîne de format pour les numéros de ligne.

Voir la page man de notangle pour plus de détails sur le fonctionnement de nt.

mnt

mnt (Multiple NoTangle) est un back-end qui peut extraire plusieurs fragments de code d'un même document en un seul passage.

Il rend le shell Noweb plus efficace.

En plus des options -t et -L reconnues par nt, il reconnaît -all comme une instruction pour extraire et écrire dans les fichiers tous les fragments de code conformes aux règles définies dans la page de manuel noweb.

Il accepte également les arguments, ainsi que les options ; chaque argument est considéré comme le nom d'un fragment de code qui devrait être émis dans le fichier du même nom.

Contrairement à nt, mnt a la fonction de cpif intégrée.

Il écrit dans un fichier temporaire, puis n'écrase un fichier existant que si le fichier temporaire est différent.

tohtml

Ce back-end émet du code HTML.

Il utilise le formateur html avec @header et @trailer pour émettre un langage HTML approprié.

Pour les autres formateurs (comme none), il n'émet aucun en-tête ni trailer. Ses options de ligne de commande sont :

Figure 1: Étapes de tube pour notangle

-delay	Accepted, for compatibility with other back ends, but ignored.
-localindex	Produces local identifier cross-reference after each code chunk.
-raw	Wraps text generated for code chunks in a LATEX rawhtml environment, making the whole document suitable for processing with latex2html
markup:	Convert to tube representation
nt:	Extract desired chunk to standard output

Étapes de tube pour noweave -index -autodefs c

```
markup: Convert to tube representation
    autodefs.c: Find definitions in C code
        finduses -noquote: Find uses of
defined identifiers
            noidx: Add index and cross-
reference information
                totex: Convert to LATEX
```

totex

Totex implémente les back-ends plain TEX et LATEX, avec @header tex et @header latex pour les distinguer.

Lorsque vous utilisez un en-tête LATEX, totex place le texte optionnel après l'en-tête à l'intérieur d'une commande \noweboptions.

Sur la ligne de commande, l'option -delay fait que totex retarde filename markup jusqu'après le premier bloc de documentation ; ce comportement transforme le premier bloc de documentation en un bloc « limbo », qui peut utilement contenir des commandes comme \documentclass.

L'option -noindex supprime la sortie relative à l'index des identificateurs ; Elle est utilisée pour implémenter noweave -x.

unmarkup

unmarkup tente d'être l'inverse de balisage : un document déjà dans le tube est reconverti en forme source noweb.

Ce back-end est utile pour essayer de convertir les autres programmes lettrés en forme source noweb.

Il peut également être utilisé pour capturer et éditer la sortie d'un module de reconnaissance automatique de définition.

Commandes standards

Les commandes standard sont toutes écrites en scripts de shell de Bourne (Kernighan et Pike, 1984).

Elles assemblent des tubes d'Unix à l'aide du balisage et des filtres et dos fins décrits ci-dessus.

Elles sont documentées dans les pages de manuel. Je montre deux exemples de tubes dans les Figures 1 et 2. Le code source est disponible dans le répertoire shell pour ceux qui veulent explorer plus loin.

Figure 3: awk command used to transform documentation to comments :

```
awk 'BEGIN { line = 0; capture = 0
             format =
sprintf("'"$format"'",'"$width"')
           }

function comment(s) {
    '"$subst"'
    return sprintf(format,s)
}

function grab(s) {
    if (capture==0) print
    else holding[line] = holding[line] s
}

/^@end doc/ { capture = 0; holding[++line] =
"" ; next }
/^@begin doc/ { capture = 1; next }

/^@text /     { grab(substr($0,7)); next}
/^@quote$/    { grab("[[") ; next}
/^@endquote$/ { grab("]]") ; next}

/^@nl$/ { if (capture !=0 ) {
            holding[++line] = ""
        } else if (defn_pending != 0) {
          print "@nl"
          for (i=0; i<=line && holding[i] ~
/^ *$/; i++) i=i
          for (; i<=line; i++)
            printf "@text %s\n@nl\n",
```

```
comment(holding[i])
          line = 0; holding[0] = ""
          defn_pending = 0
        } else print
        next
     }

/^@defn / { holding[line] = holding[line]
"<"substr($0,7)">="
            print ; defn_pending = 1 ; next }
{ print }’
```

$subst, $format et $width sont des variables du shell utilisées pour adapter le script pour différentes langages.

L’exécution de $subst élimine les marqueurs de fin de commentaire- (le cas échéant) de la documentation et le sprintf initial qui crée le format variable awk donne le format utilisé pour imprimer une ligne de la documentation comme un commentaire.

Exemples

Je ne donne pas d'exemple de la représentation de tube ; nous allons juste jouer avec les filtres existants. En particulier,

```
$ noweave -v options inputs >/dev/null
```

imprime (sur le terminal standard d'erreur) le tube utilisé par noweave pour implémenter un ensemble d’options.

Dans cette section, je donne des exemples de quelques filtres non standard, que j’ai lancés ensemble pour un but ou une autre.

Cette commande sed d'une ligne fait que noweb traite deux noms de fragments comme identiques s'ils ne diffèrent que par leur représentation des espaces :

```
$ sed -e ’/^@use /s/[ \t][ \t]*/ /g’ -e ’/^@defn /s/[ \t][ \t]*/ /g’
```

Ce petit filtre, un script shell Bourne écrit en awk (Aho, Kernighan et Weinberger 1988), crée la définition d'un bloc vide (<<>>=) qui est une suite de la définition de bloc précédente.

```
$ awk ’BEGIN { lastdefn = "@defn " }
/^@defn $/ { print lastdefn; next }
/^@defn /  { lastdefn = $0 }
{ print }’ "$@"
```

Pour partager des programmes avec des collègues qui ne font pas de programmation lettrée, j'utilise un filtre, illustré à la figure 3, qui place chaque ligne de documentation dans un commentaire et le déplace vers le bloc de code suivant. Avec ce filtre, notangle transforme un programme lettré en un

programme commenté traditionnel, sans perte d'information et avec une légère pénalité en termes de lisibilité.

Voir aussi

- **(en)** http://
- **(fr)** http://

Basé sur « Article » par Auteur.