



## Faire des Todo-Lists avec gedit

Le 9 septembre 2008 à 09:13, par advaya.

Déjà pourquoi utiliser gedit plutôt que gTodo ou évolution ? Principalement pour deux raisons : la première est que n'étant pas un directeur de projet, je n'ai à traiter que des "petites" tâches du genre liste de courses, personnes à contacter dans la semaine, etc. La seconde est que gedit est en passe de devenir ma plate-forme de travail privilégiée m'offrant un environnement presque équivalent à [kile](#) [1], de faire des scripts, de travailler sur du html/php ...

Le but de ce tutoriel est donc de réaliser cela en apprenant au passage à fabriquer soi-même une coloration syntaxique adaptée.

### Les Todo-lists

Je ne vais pas faire une méta-théorie sur les todo-list, j'imagine que cela existe déjà 😊. Dans mon utilisation personnelle, je retiens en gros quatre propriétés importantes pour les tâches d'une todo-list :

Le Statut : les tâches à faire (non encore en cours d'exécution) ; les tâches en cours ; les tâches terminées.

La Priorité : essentiellement en trois catégories : haute, moyenne ou basse ; ou encore un chiffre entre 1 et 5

La Réalisation : le degré d'achèvement de la tâche, que l'on peut représenter par un chiffre entre 1 et 5 ici aussi.

L'Ultimatum : la date à laquelle la tâche doit être accomplie entièrement.

Bien entendu, on pourrait ajouter des tas d'autres propriétés comme la liste des personnes concernées par la tâche, le "coût" de celle-ci, bref une gestion des ressources etc ... mais là on rentre dans de la «vraie» gestion de projet ce qui n'est pas l'objet de ce billet.

En revanche, j'ajoute personnellement deux autres concepts de tâche. Le premier est la tâche mise en attente, qui dépend d'autres tâches ou d'autres informations pour rentrer dans un processus d'exécution. La seconde que j'appelle «tâche floue». Ce type de tâche n'a ni priorité, ni ultimatum, ni statut bien défini, mais elle reste présente à l'esprit en tâche de fond, un peu comme la "pensée de la semaine".

Le but est donc de trouver un système de représentation compatible avec gedit permettant de faire des listes de tâches incluant toutes ces caractéristiques de façon lisible. En suivant l'exemple donné dans [ce billet](#) [2], on va pousser un peu plus loin la logique et utiliser deux choses : d'une part une notation adaptée aux différents status des tâches ; d'autre part une coloration syntaxique ad-hoc, correspondant à ce système de notation.

### Le système de notation

Le principe de départ est que la liste se lira ligne par ligne (cliquez sur l'image ci-dessus). En début de ligne de chaque tâche sera mentionné son statut à l'aide d'un code entre parenthèses : (!) pour une tâche importante ; (o) pour les tâches à prévoir ; (-) pour les tâches en cours d'exécution ; (x) pour les tâches terminées ; ( : ) pour les tâches "suspendues" ou "mises en attente" et (~) pour les tâches floues. Comme on le voit, le statut est ici mélangé avec la priorité (tâche importante ou urgente). C'est ce qui personnellement me convient, mais libre à vous d'adapter selon votre conception de la chose.

On va ensuite écrire sur la ligne correspondante un descriptif sommaire de la tâche, ainsi qu'éventuellement son ultimatum, précédé de "<<", et enfin un code représentant son niveau de réalisation, entre crochets. On ajoute un descriptif plus complet (optionnel) et d'autres renseignements sous cette ligne à l'aide de la barre verticale "|". L'utilisation de la touche TAB est vivement conseillée afin d'aligner les colonnes pour les différentes tâches. Dans l'exemple ci-dessous, on voit directement que la tâche qui doit être accomplie avant le 28/07, l'est pour le moment au 3/5 :

```
(-) résumé de la tâche en cours d'exécution    << 28/07/09    [xxx--]
| autres renseignements et informations
| relative à cette tâche
```

### gtksourceview

Nous avons maintenant besoin de faire une coloration syntaxique adaptée à notre définition de todo-list.

Pour cela, il faut d'abord savoir que les différentes colorations syntaxiques de gedit sont définies par des fichiers de type xml contenus dans /usr/share/gtksourceview-2.0. On trouve d'abord dans le sous-répertoire language-specs des fichiers .lang, qui décrivent la coloration syntaxique pour chaque type de fichier (css, script python, html etc ...). Le fichier css.lang, par exemple, décrit une suite de contextes (mot-clefs, propriétés etc ...) en adjoignant à chacun un style de présentation adapté. Par exemple,

```
<context id="comment" style-ref="comment">
  <start>/\*</start>
  <end>\*/</end>
```

```
</context>
```

définit le contexte "comment" pour les commentaires, qui commence par /\* et se termine par \*/ , auquel est associé le style "comment". Il faut alors remonter un peu plus haut dans le fichier pour voir la ligne suivante :

```
<style id="comment" _name="Comment" map-to="def:comment"/>
```

qui nous permet de savoir que le style "comment" pour les fichiers de type .css est en fait mappé vers "def:comment".

Nous voilà bien avancé, mais concrètement, où est défini "def:comment" ? C'est là qu'intervient le dossier /usr/share/gtksourceview-2.0/styles . On y trouve quelques fichiers .xml, dont le "classic.xml" qui est celui utilisé par défaut par gedit (pour choisir d'autres palettes de couleur, allez dans préférences->police et couleurs).

Un listing du fichier classic.xml nous donne alors, entre autres choses :

```
<!-- Comments -->
<style name="def:comment" foreground="blue"/>
<style name="def:shebang" foreground="blue" bold="true"/>
<style name="def:doc-comment-element" italic="true"/>
```

Nous y sommes : les commentaires entre /\* et \*/ seront ainsi représentés en bleu ( foreground="blue" ) dans gedit si on choisit la coloration syntaxique de type css.

## Mise en place

L'intérêt d'utiliser un système de notation en début de ligne pour chaque tâche est qu'il permet de déclencher une coloration syntaxique adaptée selon le statut. Idem pour la barre verticale, qui nous permettra d'assigner un style aux informations relatives à la tâche.

Pour cela, il va falloir d'une part créer un fichier todo.lang qui liste nos "contextes", et d'autre part définir concrètement les attributs de style que nous voulons voir appliqués à chaque contexte. Pour ce dont nous avons besoin, c'est assez simple, voici le fichier todo.lang que j'utilise :

```
<?xml version="1.0" encoding="UTF-8"?>
<language id="todo" _name="TODO" version="2.0" _section="Others">
  <metadata>
    <property name="mimetypes">text/plain</property>
    <property name="globs">*.todo</property>
  </metadata>

  <styles>
    <style id="comment" _name="Comment" map-to="todo:comment"/>
    <style id="important" _name="Important" map-to="todo:important"/>
    <style id="todos" _name="Todo" map-to="todo:todos"/>
    <style id="pending" _name="Pending" map-to="todo:pending"/>
    <style id="closed" _name="Closed" map-to="todo:closed"/>
    <style id="title" _name="Title" map-to="todo:title"/>
    <style id="suspend" _name="Suspend" map-to="todo:suspend"/>
    <style id="descrip" _name="Description" map-to="todo:descrip"/>
    <style id="complement" _name="Complement" map-to="todo:complement"/>
    <style id="realisation" _name="Realisation" map-to="todo:realisation"/>
    <style id="before" _name="Before" map-to="todo:before"/>
    <style id="fuzzy" _name="Fuzzy" map-to="todo:fuzzy"/>
  </styles>

  <definitions>

    <context id="todo">
      <include>

        <context id="fuzzy" style-ref="fuzzy">
          <start>\(~\)</start>
          <end>\.</end>
        </context>

        <context id="descrip" style-ref="descrip">
          <start>\*</start>
          <end>${</end>
        </context>
```

```

<context id="comment" style-ref="comment">
  <start>\#</start>
  <end>$</end>
</context>

<context id="complement" style-ref="complement">
  <start>\</start>
  <end>$</end>
</context>

<context id="important" style-ref="important">
  <start>\(!\)\</start>
  <end>$</end>
</context>

<context id="pending" style-ref="pending">
  <start>\(\-\)\</start>
  <end>$</end>

  <include>

    <context id="realisation" style-ref="realisation">
      <start>\[</start>
      <end>\]</end>
    </context>

  </include>
</context>

<context id="todos" style-ref="todos">
  <start>\(o\)\</start>
  <end>$</end>
</context>

<context id="closed" style-ref="closed">
  <start>\(x\)\</start>
  <end>$</end>
</context>

<context id="suspend" style-ref="suspend">
  <start>\(\:\)\</start>
  <end>$</end>
</context>

<context id="title" style-ref="title">
  <start>\[</start>
  <end>\]</end>
</context>

</include>
</context>

</definitions>
</language>

```

On voit donc au début du fichier une série de contextes : fuzzy, descrip, comment, etc ... auquel on associe les styles todo:fuzzy, todo:descrip ... (c'est plutôt explicite). Concrètement, ces styles seront définis dans le fichier .xml que nous allons créer ensuite. Mais revenons à notre todo.lang : on a un contexte général identifié par l'id "todo", puis la balise include permet de mettre des sous-contextes à l'intérieur. Ça commence par le contexte "fuzzy" qui débute par (~) (attention aux caractères spéciaux) et se finit en bout de ligne (c'est le \$ qui le précise). En pratique, dès qu'une ligne commencera par (~), on lui appliquera le style "todo:fuzzy".

Le reste du code est similaire, on définit pour chaque statut un style qui sera appliqué à toute la ligne. Il faut juste préciser le cas particulier du contexte "realisation" qui est lui-même inclus dans le contexte "pending", puisque le degré de réalisation de la tâche sera affiché sur la même ligne, mais avec un style différent. Sauvez le fichier sous le nom todo.lang dans le

dossier language-specs.

Venons-en maintenant à notre fichier .xml : en gros il suffit de reprendre le classic.xml et de lui adjoindre quelques définitions de style supplémentaires. Voici le fichier ClassicwithTodo.xml que j'utilise, seules quelques définitions de couleur et la dernière section (todo-list) sont ajoutées. Je crois que le code est suffisamment explicite.

```
<?xml version="1.0" encoding="UTF-8"?>
<style-scheme id="todo" _name="ClassicTodo" version="1.0">
  <author>Advaya</author>
  <_description>Classic with todo</_description>

  <!-- Palette -->
  <color name="blue" value="#0000FF"/>
  <color name="magenta" value="#FF00FF"/>
  <color name="violet" value="#6A5ACD"/>
  <color name="cyan" value="#008A8C"/>
  <color name="green" value="#2E8B57"/>
  <color name="bordeaux" value="#A52A2A"/>
  <color name="red" value="#FF0000"/>
  <color name="yellow" value="#FFFF00"/>
  <color name="purple" value="#A020F0"/>
  <color name="grey" value="#777777"/>
  <color name="darkgreen" value="#0C6836"/>
  <color name="mauve" value="#A020F0"/>

  <!-- Bracket Matching -->
  <style name="bracket-match" foreground="#white" background="#grey"
bold="true"/>
  <style name="bracket-mismatch" foreground="#white" background="#red"
bold="true"/>

  <!-- Search Matching -->
  <style name="search-match" background="yellow"/>

  <!-- Comments -->
  <style name="def:comment" foreground="blue"/>
  <style name="def:shebang" foreground="blue" bold="true"/>
  <style name="def:doc-comment-element" italic="true"/>

  <!-- Constants -->
  <style name="def:constant" foreground="magenta"/>
  <style name="def:special-char" foreground="violet"/>

  <!-- Identifiers -->
  <style name="def:identifrier" foreground="cyan"/>

  <!-- Statements -->
  <style name="def:statement" foreground="bordeaux" bold="true"/>

  <!-- Types -->
  <style name="def:type" foreground="green" bold="true"/>

  <!-- Others -->
  <style name="def:preprocessor" foreground="purple"/>
  <style name="def:error" background="red" bold="true"/>
  <style name="def:note" foreground="blue" background="yellow"
bold="true"/>
  <style name="def:underlined" italic="true" underline="true"/>

  <!-- Language specific styles -->
  <style name="diff:added-line" foreground="#008B8B"/>
  <style name="diff:removed-line" foreground="#6A5ACD"/>
  <style name="diff:changed-line" use-style="def:preprocessor"/>
  <style name="diff:special-case" use-style="def:constant"/>
  <style name="diff:location" use-style="def:statement"/>
  <style name="diff:diff-file" use-style="def:type"/>
```

```

<style name="xml:tags"                foreground="cyan"/>
<style name="xml:attribute-name"      foreground="violet"/>
<style name="xml:namespace"          foreground="green" bold="true"/>

<style name="js:object"                foreground="#2E8B57" bold="true"/>
<style name="js:constructors"         foreground="#008B8B"/>

<style name="latex:display-math"      foreground="#6A5ACD"/>
<style name="latex:command"          foreground="#2E8B57" bold="true"/>
<style name="latex:include"          use-style="def:preprocessor"/>

<style name="sh:variable"            foreground="#6A5ACD"/>

<!-- legacy styles for old lang files -->
<style name="Others"                 foreground="#2E8B57" bold="true"/>
<style name="Others 2"               foreground="#008B8B"/>
<style name="Others 3"               foreground="#6A5ACD"/>

<!-- toto lists -->
<style name="todo:title"              bold="true"/>
<style name="todo:important"          foreground="red"/>
<style name="todo:pending"           foreground="darkgreen"/>
<style name="todo:comment"           foreground="grey" italic="true"/>
<style name="todo:todos"             foreground="blue"/>
<style name="todo:suspend"           foreground="bordeaux"/>
<style name="todo:closed"            foreground="bordeaux"
strikethrough="true"/>
<style name="todo:complement"        foreground="grey"/>
<style name="todo:descrip"           italic="true"/>
<style name="todo:realisation"       foreground="mauve"/>
<style name="todo:fuzzy"             foreground="mauve"/>
<style name="todo:before"           />

</style-scheme>

```

Sauvez le fichier dans `/usr/share/gtksourceview-2.0/styles` en l'appelant par exemple `classictodo.xml`. Quittez gedit, relancez-le et allez sélectionner "ClassicTodo" dans préférences->police et couleurs. Toutes les colorations syntaxiques habituelles doivent être inchangées. Prenez un fichier vide, vérifiez que dans affichage->mode de coloration->others on retrouve bien l'entrée "TODO" et sélectionnez-la.

Nous sommes alors prêts à écrire notre première liste 😊

## Ecrire une todo-list

Dans les contextes, notre fichier a défini d'abord un titre, obtenu entre crochet et affiché en gras : entrez [Titre] pour le vérifier. Si cela ne s'affiche pas en gras, c'est que quelque chose s'est mal passé ...

Notre `todo.lang` définit plusieurs types de commentaires, qui commencent par le traditionnel #, la barre verticale | ou l'étoile \*. Les différents status sont ensuite définis avec des couleurs spécifique pour chacun. Copier le code ci-dessous dans gedit, cela vous donnera un aperçu des possibilités

```

[TITRE]

* descriptif complet de la liste
* si besoin est
* et informations complémentaires

(~) Fuzzy note.

(!) important                << 18/08
  * Attention à ces notes

(o) a faire, trucs pas encore en cours << 28/02
  complément d'info

```

(-) tâche en cours d'exécution << 19/05 [xx---]  
autre complément

(🕒) suspendue (en attente)  
mis en attente  
en attendant d'autres infos

(x) terminé



A noter que les fuzzy notes, repérées par (~), sont en général constituées de phrases, pas seulement d'un descriptif court de note. Elles doivent nécessairement se terminer par un point selon notre définition de contexte.

Noter également que dès qu'une tâche change de statut, le simple fait de changer le code de début de ligne change la coloration immédiatement, évidemment. En revanche, cela ne change pas l'ordre des lignes les unes par rapport aux autres en cas de changement de priorité ou statut, il faut le faire à la main. On peut faire faire cela par un script (par exemple, purger les tâches "terminées"), ce n'est pas très difficile, mais dans ce cas je pense qu'il vaut mieux carrément utiliser un utilitaire spécifique gérant les

todo-lists.

Pour gérer plusieurs todo-lists, je conseillerais d'utiliser des onglets plutôt que de faire un unique fichier dans lequel il faudra se déplacer. Par exemple : un onglet sur "work.todo", un autre sur "perso.todo" et un sur "courses.todo", en faisant en sorte que la liste tienne à peu près sur une page, et utilisez une session d'onglets regroupant tout cela.

Cliquez sur l'image en haut pour avoir un exemple d'utilisation.

<http://artisan.karma-lab.net/node/1625>  
(C) artisan numerique - CC BY-SA

#### Liens:

[1] <http://kile.sourceforge.net/>

[2] <http://crunchbang.org/archives/2007/09/25/gedit-todo-hack>