MTOOL - Steps

version 0.5 – juin 2007

E. Sevault CNRM/GMAP/ALGO

Proposition d'un système de directives pour le multistep et toute cette sorte de choses

Révisions :

Version 0.1 – 2006/12/19 - E. Sevault : version beta

Version 0.2 – 2007/01/15 - E. S. : version complète

Version 0.3 – 2007/01/16 - E. S. : broadcast / ftdir & Cie / maj doc

Version 0.4 – 2007/01/26 - E. S. & Ryad El Khatib : common / join / not / clean
+ autolog / autoclean / maj doc

Version 0.5 – 2007/06/17 - E. S. : maj clean / broadcast / doc
+ track / import / commandes en ligne
+ sync version alpha

Sommaire

1. Généralités.....4
Syntaxe des directives.....6
Pseudos-variables.....7
Inclusion de fichiers externes.....8
Catalogue de fichiers inclus.....9
Définition de profiles de job.....10
Insertion automatique en début de job.....12
Catalogue des directives.....13
Catalogue des variables d'environnement.....18
Utilisation avancée.....19
Le nettoyage automatique.....19
Proposition pour la gestion des anomalies.....21
Syntaxe des commandes en ligne.....22

1. Généralités

Motivation :

Distinguer au sein d'une seule exécution logique (le job) différentes étapes physiques (les steps) pour profiter au mieux d'un environnement hétérogène ou de contraintes d'exploitation particulières.

Principe :

Des directives (commentaires pour le shell de type sh / ksh / bash ou bien perl) sont insérées dans le corps du job initial qui passe par un « filtre » logiciel afin de générer différents steps physiques exécutés de façon chaînée par le gestionnaire de travaux (e.g. NQS).

Implémentation :

Un ensemble de programmes est disponible sur les machines cibles par renseignement de la variable **PATH**. Il est dans ce cas pratique de renseigner simultanément la variable **MTOOL_ROOT** et quelques alias de confort si on le souhaite... :

```
% MTOOL_ROOT=~mrpm631/public/mtool
% PATH=$PATH:$MTOOL_ROOT/bin
% cd $MTOOL_ROOT/bin
% ls -1
mtool_banner.pl
mtool_clean.pl
mtool_filter.pl
mtool_fsync.pl
mtool_import.pl
mtool_submit.pl
mtool_track.pl
mtool_wait.pl
% alias qf='mtool_filter.pl'
% alias banner='mtool_banner.pl'
```

Développement logiciel :

Un utilisateur avancé pourra directement constituer ses propres utilitaires en utilisant ou surchargeant le ou les modules « perl » suivant :

```
% cd $MTOOL_ROOT/src
% ls
Mtool  Mtool.pm  Text
% ls -1 */*
Mtool/Bin.pm
Mtool/Date.pm
Mtool/Exception.pm
Mtool/Filter.pm
Mtool/Kit.pm
Mtool/Script.pm
Mtool/Shell.pm
Mtool/Util.pm
Text/Banner.pm

Mtool/Bin:
Fsync.pm
Import.pm
Track.pm

Mtool/Script:
Perl.pm
Sh.pm

Mtool/Util:
Counter.pm
Nice.pm
Storable.pm
```

Syntaxe des directives

Les directives sont des commentaires shell (**ksh** pour l'instant, potentiellement **perl** dans l'avenir) en une seule ligne de la forme :

```
#MTOOL commande [arg-1=value] ... [arg-n=value]
```

Exemple : un job multi-step peut se réduire à un jeu très simple de directives :

```
#MTOOL autolog
#MTOOL set jobname=toto
set -ax

#MTOOL step id=premier
echo cool job

#MTOOL step id=second target=torisx
echo pas trop dur non plus

#MTOOL common
echo fini
```

Premiers commentaires : de ce job très simple il est déjà possible de déduire quelques règles de fonctionnement du multistep « mtool » :

- Avant la première commande **common** ou **step**, les instructions sont considérées comme communes à toutes les étapes à venir. C'est commode pour définir un certain nombre d'instructions ou de paramétrisations par défaut ;
- Il n'y a pas de fin de section (**common** ou **step**) explicite. Celle-ci est donnée par la fin du job ou par le début d'une autre section ;
- Un job peut être rigoureusement identique (tout au moins fonctionnellement) avec le job mono-step sous réserve d'instrumentaliser correctement le script ;
- Une étape peut être une section inactive (instruction **common**) dont la seule vocation est d'être partagée avec un ou plusieurs autres steps (voir tous dans le cas par défaut **join=all**) ;
- Par défaut une étape (**step**) dont l'argument **target** n'est pas spécifié s'exécute sur la machine courante en utilisant une carte de soumission par défaut ;
- La commande **set** permet de positionner une pseudo-variable globale au pré-processing de « mtool_filter.pl » (ce n'est pas une variable shell). Un certain nombre de pseudo-variables sont disponibles par défaut (voir plus loin) ;
- La pseudo-variable **jobname** est indispensable si vous utilisez les profils de job par défaut de la machine courante (voir ci-dessous).

Pseudos-variables

Il est possible de renseigner des variables de deux façons, selon qu'elles concernent la configuration propre à l'étape en cours (**configure**) ou qu'elles soient globales au traitement du job (**set**). Voici un court exemple :

```
#mtool set bigboss=torisx

#MTOOL step target=toritx
#MTOOL configure submitcmd=qsubperso
echo hello [this:coucou] from [next:id]

#MTOOL step target=[this:bigboss]
#MTOOL configure id=toto
#MTOOL set coucou=world
echo real world
```

Le premier **configure** (ligne 3) permet de définir une méthode de soumission pour cette étape particulière uniquement tandis que la ligne 1 rend disponible dans toute la suite du script que **bigboss** sera équivalent à « torisx ». Moins intuitif mais tout aussi valide, le second **set** qui positionne « coucou » à « world » permet à la première étape de résoudre complètement la ligne d'echo qu'elle s'apprête à exécuter, et qui dans le script réel du shell n°1 sera en fait :

```
echo hello world from toto
```

La seconde étape sera soumise sur la cible d'exécution (**target**) torisx en vertu du même mécanisme de résolution de variable. Il est donc temps de décrire complètement ce mécanisme. Les items à substituer doivent être de la forme suivante :

```
... [scope:var-name] ...
```

avec **scope** qui désigne le step depuis lequel on essaiera de déduire la valeur de la variable **var-name**. Les valeurs possible de **scope** sont :

```
this     : le step courant
first    : le premier step actif
prev     : le step actif précédent
next     : le step actif suivant
last     : le dernier step actif
```

Les quelques pseudo-variables pré-définies sont présentées dans le descriptif de la commande **set** du catalogue des directives (cf. *infra*).

Inclusion de fichiers externes

Avant de voir d'autres commandes permettant une utilisation plus fine du multi-step mtool, il est nécessaire de s'arrêter à une commande simple mais pratique pour garder un caractère générique à vos scripts : il s'agit de l'inclusion de fichiers externes :

```
#MTOOL include files=file1[,file2]...
```

La recherche de fichiers inclus se fait dans l'ordre suivant :

1. le fichier est disponible à partir du répertoire courant (c'est le cas pour un fichier spécifié de façon absolue) ;
2. le fichier est disponible dans le répertoire d'includes de la configuration utilisateur (dans l'ordre $HOME/.mtoolrc/include/*shell* ou $HOME/.mtoolrc/include ou $HOME/.mtoolrc) ;
3. le fichier est disponible dans le répertoire d'includes de la configuration mtool ;
4. le fichier est disponible dans le répertoire racine de la configuration mtool ;

Les fichiers inclus peuvent eux-mêmes contenir des directives MTOOL... dont des inclusions ! Les directives sans effet sont le « **set** » d'une pseudo-variable globale et **autolog** / **autoclean**.

Dans la mesure où les règles de substitution de pseudo-variables présentées précédemment s'appliquent bien entendu à ces commandes, il est donc possible de faire des inclusions différentes avec la même ligne de commande mais dépendant du step actif. Ainsi dans l'exemple ci-dessous, la paramétrisation de l'environnement se fait en fonction de fichiers pré-définis dans les répertoires de configuration de mtool :

```
set -ax
#MTOOL set jobname=toto
#MTOOL set data=toritx
#MTOOL set run=torisx
#MTOOL include files=path.basic.[this:target]
#MTOOL include files=arch.fortran.[this:target]

#MTOOL step target=[this:data]
echo fetch data env
env
#MTOOL step target=[this:run]
echo run data env
env
#MTOOL common
#MTOOL include files=gotosleep
```

Catalogue de fichiers inclus

Les répertoires **$MTOOL_ROOT/include** et **$MTOOL_ROOT/include/ksh** contiennent un certains nombre (réduit pour l'instant) de fichiers pouvant être utilisés pour les cibles d'exécutions **torisx** (SX8R) et **toritx** (TX7) :

```
% cat arch.fortran.torisx
F_PROGINF=DETAIL
F_FTRACE=FMT2
F_RECLUNIT=BYTE
F_SYSLEN=1024
F_FMTBUF=32768
F_SETBUF=32768
F_SETBUF6=0
F_SETBUF0=0

% cat arch.fortran.toritx
# empty configuration file

% cat arch.mpi.torisx
MPISUSPEND="ON"
MPISUSPENDCNTL="SLEEP"
MPIPROGINF="ON"
MPIDEBUG="OFF"
MPIEXPORT="MPIPROGINF,MPIDEBUG"
MPIEXPORT_FILTER="^F_"
SWAPP_MPI_LAUNCH="mpirun"
SWAPP_MPI_OPTIONS="nn,nnp"
SWAPP_MPI_NNP=8

% cat path.basic.torisx
PATH=/bin:/usr/bin:/usr/ucb

% cat path.basic.toritx
PATH=/bin:/usr/bin:/usr/ucb:/usr/X11R6/bin
```

Il est évident qu'à terme cette liste pourrait être étendue. Par ailleurs, en vertu des règles d'inclusion énoncées à la section précédente, un utilisateur peut tout aussi bien créer un fichier inclus par défaut dans sa propre configuration :

```
% ls -1 $HOME/.mtoolrc/include/ksh
% path.basic.torisx
```

Définition de profiles de job

Dans la mesure où les différentes étapes d'un job peuvent s'exécuter sur des cibles différentes ou demander des ressources variées, il doit être possible de spécifier le « profile » de ces étapes. Ces profiles peuvent être définis dans le corps du fichier job ou référencés dans des fichiers inclus.

La définition dans le corps de job est le seul de commande bloc dans la synthaxe des directives MTOOL :

```
#MTOOL set jobname=hello
#MTOOL profile target=torisx
#PBS -N [this:jobname]
#PBS -S /bin/ksh
#PBS -T mpisx
#PBS -b 1
#PBS -l cpunum_job=4
#PBS -l cputim_job=2000
#PBS -l elapstim_req=600
#PBS -l memsz_job=8000mb
#PBS -j o
#PBS -q express
#MTOOL end
```

Par la suite, toutes les étapes ayant **`torisx`** comme cible d'exécution pourront utiliser ce profile.

Cette définition peut se faire par un fichier inclus :

```
#MTOOL set jobname=hello
#MTOOL profile target=x1
#MTOOL include files=profile.x1
#MTOOL end
#MTOOL profile target=x2
#MTOOL include files=profile.x2
#MTOOL end
#MTOOL step target=x2
echo using profile.x2 found somewhere in my conf or defaults
```

Enfin, en l'absence de toute définition explicite il est toujours possible de bénéficier du « ramasse-miettes » des profiles par défaut de la configuration mtool qui prennent la forme suivante :

```
#MTOOL include files=profile/submitmethod.default.target
```

Avec ***submitmethod*** désignant la méthode de soumission du step courant (e.g.: **qsub**) et ***target*** la cible d'exécution du step actif. De tels fichiers sont pré-définis :

```
% cd $MTOOL_ROOT/profile
% cat qsub.default.toritx
#PBS -N [this:jobname]
#PBS -S /usr/bin/ksh
#PBS -j o
#PBS -l elapstim_req=00:30:00
#PBS -l memsz_job=500mb
#PBS -q ft

% cat qsub.default.torisx
#PBS -N [this:jobname]
#PBS -S /usr/bin/ksh
#PBS -j o
#PBS -b 1
#PBS -l cpunum_job=1
#PBS -l cputim_job=2700
#PBS -l elapstim_req=01:00:00
#PBS -l memsz_job=1000mb
#PBS -q mono

% ls
qsub.default.torisx -> qsub.mono.torisx
qsub.default.toritx -> qsub.ft.toritx
qsub.ft.toritx
qsub.mono.torisx
```

A mesure que les classes d'exploitations sont fixées, des profiles par défaut spécifiques sont définis pour le SX8R. Néanmoins il demeure plus recommandable de spécifier dans le corps du job le profile des sections de calcul au plus près des ressources réellement nécessaires.

Insertion automatique en début de job

Une séquence minimale définissant l'environnement de l'étape courante est insérée en début de shell :

```
#MTOOL banner string=STEP-indexnumber size=2 line=80
#MTOOL export MTOOL_STEP=stepnumber
#MTOOL export MTOOL_STEP_ID=[this:id]
#MTOOL export MTOOL_STEP_TARGET=[this:target]
#MTOOL export MTOOL_STEP_STORE=[this:store]
#MTOOL export MTOOL_STEP_WORKSPACE=[this:workspaceroot]/
    [this:workspace]
```

Toutes ces variables sont donc disponibles pour l'utilisateur dans la suite du job et pour tous les steps.

2. Catalogue des directives

- **autoclean**
  active le nettoyage automatique des anciens espaces de travail de **mtool** qui pour une raison ou une autre n'auraient pas été éliminés lors de précédentes soumissions. Pour en savoir plus, voir la section « utilisation avancée » en fin de documentation. Le comportement de ce nettoyage automatique peut être influencé par la pseudo-variable **cleanmode** que l'on peut définir en ligne de commande de **mtool_filter.pl** ou par la commande set. Dans l'immédiat, une recommandation simple est d'insérer la directive telle quelle :

```
#MTOOL autoclean
```

- **autolog**
  active la création d'une étape complémentaire qui assurera la collecte des sorties standard d'exécution de toutes les étapes en une sortie unique. Une variable d'environnement est alors insérée en début de séquence de chaque étape, **MTOOL_STEP_LOGFILE**, identique pour toutes les étapes et qui contient le chemin absolu de la logfile. En l'absence de spécification complémentaire, celle-ci est constituée par la concaténation du chemin de soumission, du nom du fichier du job et « .o » suivi du numéro de process. Un path alternatif peut être donné soit par surcharge de la variable **MTOOL_STEP_LOGFILE**, soit par le positionnement de la pseudo-variable **logfile**. Enfin, la machine cible de l'étape de collecte (dont l'identifiant par défaut est « autolog ») peut être spécifié par la pseudo-variable **logtarget** :

```
#MTOOL autolog
#MTOOL set logtarget=toritx
echo la logfile devrait être $MTOOL_STEP_LOGFILE
```

- **banner**
  permet d'afficher une bannière de configuration variable, en fonction des attributs suivants : **string** (la chaîne de caractères à transformer en bannière), **size** (le nombre de caractères constituant l'épaisseur du « trait », défaut = 1), **line** (la longueur de la ligne encadrant le texte), **char** (le caractère servant de brique élémentaire, défaut « # »), **lchar** (caractère utilisé pour constituer la ligne, défaut « = »), **rotate** (orientation de la bannière, défaut « H », sinon « V ») ; exemple :

```
#MTOOL banner string=coucou size=2 line=132
echo doit se voir de loin
```

- **broadcast**
  mécanisme pour diffuser de l'information (variables d'environnement ou systèmes

de fichiers) vers d'autres steps. L'usage le plus simple consiste à exporter une liste explicite de variables indiquées par l'attribut **vars** vers un autre step nommé par son indicateur **id** :

```
#MTOOL step id=compute
TITI=123
TOTO=$(($TITI+5))
#MTOOL broadcast id=[next:id] vars=titi,toto
#MTOOL step id=display
echo $TITI + 5 = $TOTO according to [prev:id]
```

- **clean**
  provoque le nettoyage conditionnel d'un répertoire en fonction de sa dernière modification exprimée en jours. L'activation de l'option **recurse** (on|off) permet de prendre en compte non plus la date de modification du répertoire lui-même mais celle du plus récent de ses éléments, et ce, de façon récursive. L'option **max** permet de spécifier une limite maximum au nombre de répertoires effectivement éliminés par cette procédure (défaut : 999) :

```
#MTOOL clean path=[this:workspaceroot] old=2
#MTOOL clean path=/work/mrpm631/xp/A* old=5 recurse=on max=3
```

- **common**
  ouvre une section non exécutable destinée à s'agglomérer (ou pas) avec une ou plusieurs étapes actives : si *a priori* tous les paramètres qui s'appliquent à la commande **step** sont valides, il faut savoir que l'attribut **active** est forcé à la valeur « no ». Par ailleurs, deux autres attributs prennent une valeur par défaut (mais peuvent être modifiés explicitement par cette directive) : **join** est positionné à « all » et **not** est positionné à la valeur de la pseudo-variable **notdefault** qui elle-même vaut « autolog » par défaut. Exemples :

```
#MTOOL common join=compute
echo ok pour steps avec id = compute, compute_2, etc.

#MTOOL common not=fetch
echo pour tous les steps sauf fetch, fetch_data, etc.

#MTOOL common join=fetch not=fetch_big
echo ok pour steps avec id = fetch, fetch_data, etc.
echo mais ko pour step id = fetch_big

#MTOOL common
echo ok pour tous sauf le collecteur (autolog par défaut)
```

- **configure**
  positionne un paramètre pour l'étape courante ; les paramètres possibles sont : **active**, **id**, **join,** **not**, **shell**, **submitcmd**, **submitmethod**, **target** ; exemple :

```
#MTOOL step
#MTOOL configure join=all active=no shell=bash
echo no chance to do the job
```

- **export**
  positionne dans le shell une variable d'environnement (cette commande sert uniquement à maintenir la possibilité de shell rigoureusement identiques avec ou sans mtool d'activé) :

```
TOTO=monostep
#MTOOL export TOTO=[prev:id]
echo le toto avant est... $TOTO !
```

- **import**
  met à disposition du step courant les éléments rendus publics par d'autres steps via la directive **broadcast**, elle-même éventuellement alimentée par un mode **track** actif. Par défaut l'attribut **tag** est positionné à l'identifiant du step courant.

```
#MTOOL import tag=[this:id] store=[this:store] type=fs
```

- **include**
  inclut un ou plusieurs fichiers externes :

```
#MTOOL include files=/tmp/bof,path.default.[this:target]
```

- **on|off**
  active / désactive le traitement du filtre mtool :

```
set -x
#MTOOL step id=toto
echo step [this:id]
#MTOOL off
#MTOOL configure submitcmd=myqsub
echo la ligne précédente est ignorée !
#MTOOL on
```

- **profile**
  ouvre la définition d'un profil de tâche pour la cible d'exécution spécifiée par l'attribut **target** (sinon, la cible par défaut est **auto**) ; le bloc se termine par la directive **end** :

```
#MTOOL profile target=toritx
#PBS -N [this:jobname]
#PBS -S /usr/bin/ksh
#PBS -j o
#PBS -l elapstim_req=00:30:00
#PBS -l memsz_job=500mb
#PBS -q ft
#MTOOL end
```

- **set**
  positionne une pseudo-variable pour le traitement du filtre mtool ; les pseudo-variables pré-définies sont : **storeroot**, **workspaceroot**, **workspace**, **ftdir**, **tmpdir**, **tmploc**, **workdir, logfile, logtarget, jobtag, cleanmode, notdefault**. En particulier jobtag permet de spécifier une extension au nom de répertoire automatique créé pour le stocake des step et de leurs sorties (à savoir, le « store » donné par la variable d'environnement **MTOOL_STEP_STORE** ou la balise dynamique **[this:store]**) ; exemple :

```
#MTOOL step id=toto
#MTOOL set myvar=2 workspaceroot=$HOME/virtualdisk
```

- **step**
  ouvre une nouvelle étape exécutable dans la tâche courante ; il est possible de positionner en argument de cette directive les même attributs que par la commande **configure** :

```
#MTOOL step id=toto join=titi target=brouette
echo encore quelques pelletées et c'est fini
```

- **submit**
  force la soumission d'une autre étape ; *a priori* l'insertion en fin d'étape pour lancer l'étape suivante est implicite (sauf bien entendu pour la toute dernière étape), ce qu'illustre le premier exemple ci-dessous. Néanmoins il est tout à fait possible d'utiliser cette directive pour lancer un job générique (qui ne soit pas multistep). Dans ce cas il est possible également de « marquer » ce job par un **tag** permettant d'attendre sa fin d'exécution via la directive **wait** :

```
# exemple 1 : fin de step standard
```

```
#MTOOL submit step=[next:number] out=step.[next:number].out
echo parti a la grâce de qsub

#exemple 2 : soumission explicitement
#MTOOL submit job=cooljob tag=cool output=[this:store]/
cool.log
echo une petite sieste ?
sleep 10
#MTOOL wait tag=cool
```

- **track**
  active le traçage des modifications d'environnement ou du système de fichier suivant l'attribut **type**. Le positionnement d'un attribut **tag** permet de différencier plusieurs opérations de suivi au sein d'un même step. Cette directive est généralement utilisée en conjonction avec celle de **broadcast**.

```
#MTOOL track tag=fun type=env,fs
TOTO=22
echo nouvelles du monde > bidon/news
#MTOOL broadcast id=[next:id] track=fun type=env,fs
# le prochain step connaîtra la valeur de TOTO
# et disposera localement d'une arborescence bidon/news
```

- **wait**
  positionne le shell courant en attente de fin d'exécution d'un job précédemment soumis (dans ce step ou dans un autre) et reconnaissable par son **tag** :

```
#MTOOL step id=debut
#MTOOL submit job=cooljob tag=cool output=[this:store]/
cool.log
echo on fait le boulot
echo puis on quitte sans se soucier de cooljob

#MTOOL step id=fin
echo autre chose
echo avant de quitter on vérifie que cooljob a fini aussi
#MTOOL wait tag=cool
```

Catalogue des variables d'environnement

- **MTOOL_STOREROOT**
  répertoire de stockage des steps physiques et de leurs sorties (logs) ;
  défaut : **$FTDIR/mtool/submit** ;
- **MTOOL_NODE**
  nom du noeud principal de soumission ;
  défaut : valeur renvoyée par la commande unix **hostname** ;
- **MTOOL_RC**
  répertoire de configuration personnelle ;
  défaut : **$HOME/.mtoolrc** ;
- **MTOOL_ROOT**
  répertoire racine d'installation de la boîte à outils ;
  défaut : **~mrpm631/public/mtool** ;
- **MTOOL_SHELL**
  référence du shell utilisé
  défaut : **ksh** ;
- **MTOOL_VERBOSE**
  dégré de verbosité ;
  défaut = 1 ;
- **MTOOL_WORKSPACEROOT**
  répertoire racine des espaces partagés entre steps ;
  défaut : **$FTDIR/mtool/spool** ;

Utilisation avancée

Le nettoyage automatique

De façon systématique le répertoire d'espace partagé (workspace) est automatiquement nettoyé par la dernière étape active. En cas de fin anormale d'exécution, il n'est pas nettoyé de manière à permettre un éventuel débuggage.

Le répertoire de soumission (store), contenant par défaut les fichiers effectivement soumis pour chaque step ainsi que les logs d'exécution associées, sont conservées indéfiniment si l'utilisateur conserve le mode d'attribution de répertoire de travail par défaut (incrément de compteur universel).

Pour ces deux espaces, l'activation de la directive **autoclean** permet un nettoyage ciblé. En fait le positionnement de la directive :

```
#MTOOL autoclean
```

est équivalente à l'ajout de la ligne suivante dans la dernière étape active, en fonction de la définition ou non de la pseudo-variable **cleanmode** :

```
#MTOOL include files=auto.clean.[this:target]
  ou
#MTOOL include files=autoclean.[this:cleanmode].[this:target]
```

Actuellement, les fichiers **auto.clean.toritx** et **auto.clean.torisx** du répertoire d'inclusions par défaut pointent tous deux sur le fichier **auto.clean.tori** dont voici le contenu :

```
#MTOOL clean path=[this:storeroot] old=5
#MTOOL clean path=[this:workspaceroot] old=2
```

Rappel : dans la mesure où l'utilisateur peut lui-même fournir un fichier auto.clean.* il est tout à fait possible de proposer une stratégie de nettoyage alternative totalement personnalisée, comme dans l'exemple ci-dessous :

```
# soyons prudents !
ls -l [this:workspaceroot]
#MTOOL clean path=[this:workspaceroot] old=5
#MTOOL clean path=[this:storeroot] old=1
```

Pour finir, notons que si par ailleurs le mécanisme de collecte automatique des sorties est activé (directive **autolog**), la procédure de nettoyage elle-même sera effectuée dans la tâche de collecte qui se trouve, par la force des choses, être la dernières des étapes.

Proposition pour la gestion des anomalies

Dans le cadre du multi-step, la gestion des « abort » intempestifs est de première importance, en particulier pour s'assurer de la collecte des sorties. Une proposition est d'attraper les erreurs et d'inclure un fichier qui varie en fonction de l'activation ou non de la fonctionnalité **autolog** :

```
function ERROR {
 set +e
 echo au secours !
#MTOOL include files=epilog.clean.step
#MTOOL include files=epilog.emergency.[this:autolog]
 trap "" 0
 exit 0
}

trap ERROR 0
```

Avec par exemple le fichier **epilog.clean.step** :

```
if [ -d "$TMPDIR" ]; then
  cd $HOME/tmp
  \rm -rf $TMPDIR
fi

if [ -d "$TMP_LOC" ]; then
  cd $HOME/tmp
  \rm -rf $TMP_LOC
fi
```

Et le fichier **epilog.emergency.on** :

```
# sur une seule ligne...
#MTOOL submit step=[last:number] out=[this:store]/step.
[last:number].out
```

Tandis que le fichier **epilog.emergency.off** pourra rester vide par exemple. On voit ici l'usage avantageux Syntaxe des commandes en ligneque l'on peut faire des préfixes du type **last** ! Encore une fois, dans la mesure où tous ces fichiers peuvent être personnalisés, les possibilités sont très vastes.

Syntaxe des commandes en ligne

La plupart des options de **mtool_filter.pl** se déduisent très facilement des pseudos variables et variables d'environnement décrites plus haut. Voici la liste alphabétique pour la version courante :

```
    autoclean=s
    autolog=s
    cleanmode=s
    export=s
    ftdir=s
    host=s
    id=s
    jobname=s
    jobtag=s
    logfile=s
    logtarget=s
    mtoolrc=s
    node=s
    notdefault=s
    root=s
    shell=s
    storeroot=s
    submit!
    submitcmd=s
    submitmethod=s
    tmpdir=s
    tmploc=s
    user=s
    verbose!
    workdir=s
    workspaceroot=s
    workspace=s
```

Exemple :

```
% alias qj='mtool_filter.pl'
% qj --autolog=on --autoclean=off --storeroot=$PWD job1 job2
```