**DUAS Dependency Checker (Synapps SAS)**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Qui | Quand | Description |
| V0 | Eric DELAWOEVRE | 11 Octobre 2019 | Initial |
| V1 | Eric DELAWOEVRE | 26 Novembre 2019 | Fonctionnement |
| V2 | Eric DELAWOEVRE | 1 Décembre 2019 | Analyse |
| V3 | Eric DELAWOEVRE | 02 Décembre 2020 | GraphViz |
| V4 | Eric DELAWOEVRE | 25 Janvier 2021 | ID Appli |
| V5 | Eric DELAWOEVRE | 17 Avril 2021 | Arguments en + |

Sommaire

[1 Introduction 3](#_Toc69586937)

[2 La distribution 3](#_Toc69586938)

[3 Flux 3](#_Toc69586939)

[4 Fonctionnement 4](#_Toc69586940)

[4.1 Mode connecté ou non-connecté 4](#_Toc69586941)

[4.2 Connaitre une chaine Dollar Universe 6](#_Toc69586942)

[4.2.1 Un seul environnement à la fois 6](#_Toc69586943)

[4.2.2 Un seul espace à la fois 7](#_Toc69586944)

[4.2.3 Conditionnement et chainage 7](#_Toc69586945)

[4.2.4 Tache sur Uproc 7](#_Toc69586946)

[4.2.5 Session mono-Uproc 7](#_Toc69586947)

[4.2.6 Entête et pied de session 7](#_Toc69586948)

[4.3 Mode commande, batch, conf, etc… 8](#_Toc69586949)

[4.4 Les entrées de depCheck 8](#_Toc69586950)

[4.5 Les sorties de depCheck 9](#_Toc69586951)

[4.5.1 Détection des applications 10](#_Toc69586952)

[5 Les vérifications de depCheck 11](#_Toc69586953)

[6 Argumentation 12](#_Toc69586954)

[6.1 Vérification d’exécution 12](#_Toc69586955)

[6.2 Arguments 13](#_Toc69586956)

[7 Exécution dans le contexte 13](#_Toc69586957)

[7.1 Synchronisation des tables de nœuds 13](#_Toc69586958)

[7.2 Génération des uxtraces en mode connecté 14](#_Toc69586959)

[7.3 Exécution 14](#_Toc69586960)

[7.4 Analyse 15](#_Toc69586961)

[7.4.1 Consistance 15](#_Toc69586962)

[7.4.2 Conception 15](#_Toc69586963)

[7.4.3 Obsolescence 16](#_Toc69586964)

[7.4.4 Les commandes 16](#_Toc69586965)

[7.4.5 Les batches externes 16](#_Toc69586966)

[7.4.6 Les objets 16](#_Toc69586967)

[7.5 GraphViz 16](#_Toc69586968)

[8 Beginner 17](#_Toc69586969)

[8.1 Unix 17](#_Toc69586970)

# Introduction

Le dependency checker est un outil développé par Averroes-Group (ex Synapps) permettant de repérer des erreurs fonctionnelles au sein d’un parc de nœud Dollar Universe.

Le programme est développé en perl. Il a la capacité de fonctionner sur Unix ou Windows pour des versions Dollar Universe 5.6 et 6

L’outil est mis à disposition sur un parc machine pour usage illimité. Il servira ponctuellement à qualifier les chaines de production afin d’en détecter les failles et les rattachements par les conditionnements.

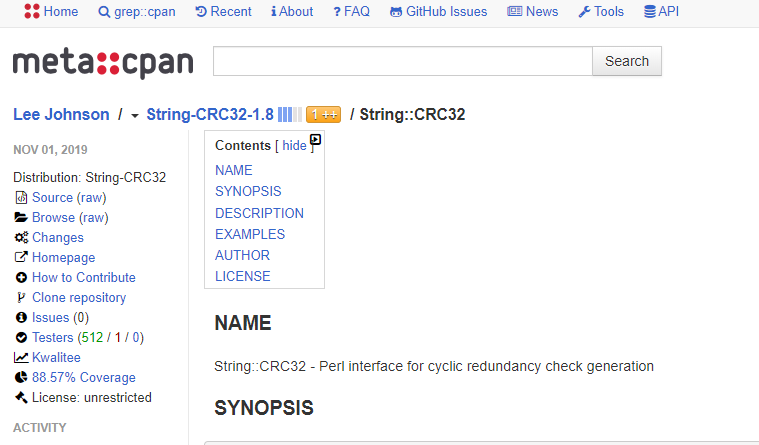
Ce document s’adresse à des exploitants Dollar Universe Niveau 2 ou supérieur. Plusieurs termes d’ordonnancement et Dollar Universe, employés, ne sont donc pas rappelés.

# La distribution

Un seul fichier, le script perl : depCheck.pl

Il est fortement conseillé de posséder un perl dont la distribution délivre d’une librairie officielle bien garnie. ActivePerl type Strawberry est adéquat

Il est parfois indispensable d’installer des nouvelles librairies. Il est alors tout à fait possible de le faire par l’outillage CPAN ou bien de les compiler facilement une fois récupérées et mises en place sur le serveur.



Exécuter perl –c depCheck.pl permet de vérifier l’intégrité du script et erreur de librairies manquantes.

# Flux

Selon son usage, depCheck peut être disposé à des endroits différents :

* Sur un serveur d’outillages
  + Connecté aux nœuds Dollar Universe par un nœud local de même société
  + Non connecté
* Sur un poste de travail

Le depCheck repose sur des uxtraces (dump d’un nœud Dollar Universe). Il peut être en mode « connecté » (Direct Command) et produire ces uxtraces pour les traiter par la suite, ou bien travailler sur des uxtraces rapatriés par un tiers.

Le mode connecté utilise un nœud Dollar Universe local au script. Il sera alors necessaire de :

* Charger l’environnement du nœud au préalable
* Disposer des droits à la consultation des objets du parc Dollar Universe interrogeable par ce nœud local.

# Fonctionnement

## Mode connecté ou non-connecté

Bien que son argumentation soit identique, depCheck se comporte sensiblement différemment selon qu’il soit en mode connecté ou pas.

Le mode connecté consiste à :

* Charger l’environnement du DUAS installé (IO tournant) : uxsetenv (v5.6) ou unienv (V6)
* Exécuter depCheck

depCheck réalise une analyse de tout le plan batch à partir de l’ensemble des nœuds DUAS. Cet ensemble provient d’une liste de nœuds.

Le mode connecté, cette liste de nœuds :

* Provient de la liste connus du nœud chargé (table de nœuds)
* Est affinée par argument (--nodes) et permet de réduire la liste afin d’écarter certains nœuds
* Est réalisée en uxtraces nécessaires s’ils n’existent pas. Si les uxtraces sont déjà présents, depCheck les utilisera plutôt que de réaliser un uxtrace sur le nœud distant.

Afin de forcer la réalisation des uxtraces, même s’ils existent, l’argument --force peut être indiqué.

En réalité, le mode connecté, seuls les commandes utiles d’extraction des objets propres à l’analyse sont effectuées. Les fichiers extraits sont en revanche nommés de la même manière.

Le mode non-connecté se base sur des uxtraces. Et le principe est le même :

* La liste de nœuds est imposée par l’argument --nodes.
* Les uxtraces se trouvent tous sous une forme zippée ou non

Les uxtraces doivent posséder le dump des CLINT par la variable U\_UPR\_SCRIPT\_NODISP=N afin que depCheck analyse également le contenu des CLINT

En revanche les queues ne sont pas présentes dans des uxtraces (en raison de l’aspect modulaire de DQM en v5.x). Ceci n’a peu d’incidence.

Egalement, pour réaliser des uxtraces à l’équivalent d’un mode connecté, il est possible de consulter la sortie de depCheck dans le fichier depCheck\_PARSECMD.log. Il contiendra les commandes. Elles sont au format Unix et facilement adaptables en Windows

Enfin, le mode connecté est détecté par la présence de la variable d’environnement UXEXE (v5.6) ou UNI\_DIR\_EXEC (v6). Même si l’environnement est chargé par défaut, il est possible de désolidariser le mode connecté du fonctionnement du depCheck en retirant la variable dans la session de commande.

## Connaitre une chaine Dollar Universe

Même si depCheck vérifie scrupuleusement les erreurs fonctionnelles d’intégration Dollar Universe, il est important de comprendre les idéaux de chainage de l’ordonnanceur.

Cette partie peut donc être considérée comme un prérequis à l’exécution de depCheck dans son utilisation nominale.

Le principe est universel :

* depCheck énumère toutes les taches actives, modèles ou non
* Tout le chainage Session/Uproc/UG est validé
  + Test de présence de chaque objet sur chacun des nœuds mis en jeu
  + Test de structure (cas des sessions par exemple) devant être identique sur chaque nœud mis en jeu
  + Test de contenu

NB : Une vérification de contenu d’uproc (CLINT, CLEXT, SYS,…) est effectuée. Tout ce qui sort du contexte Dollar Universe (dont le contenu du script exécuté par une uproc) sort également du contexte de vérification.

NB2 : Les uxtraces de taches contiennent par défaut les modèles et non-modèles. Il est possible de focaliser uniquement le contexte « exécutif » et ne générant que des uxtraces de tache avec les non-modèles par l’argument –nomodel. Attention donc au mode non-connecté ou la main sur les extractions ou les uxtraces par un autre tiers, apporte des taches aussi modèles.

NB3 : la notion de version d’objet (uproc, session) est hors cadre et non supporté. La version par défaut prise en charge par le nœud, est tenue en compte.

### Un seul environnement à la fois

Un nœud de production doit communiquer avec les autres nœuds de production. Ce n’est pas le cas pour les nœuds de non-production.

Ainsi :

* Le nom de l’unité de gestion (UG ou MU) est unique quel que soit l’environnement. Ce n’est pas forcément le cas pour son contenu (cas v6 pour les variables d’UG)
* Une tache, Une session et Une uproc sont uniques dans leur structure et leur contenu

L’utilisation de depCheck peut donc, dans un cas exceptionnel, être utiliser pour comparer une production d’une non-production. Les erreurs d’unicité ou de structure indiqueront les différences notables entre un développement et une production.

Dans tout le reste du document, il sera donc indispensable de noter que la mesure soit effectuée dans un seul environnement à la fois.

En mode connecté en se basant sur un nœud dont la liste contient plusieurs environnements, il faut utiliser l’argument --nodes.

### Un seul espace à la fois

L’espace (ou Area) est par défaut EXP (X). Il est possible de focaliser les extractions (uxtraces) sur un espace désigné en utilisant l’argument --area=<area>

Il est cependant important de noter que cette utilisation doit se porter dans un contexte ou l’espace EXP n’est pas utilisé comme porteur au test. Beaucoup de ces objets sont transverses (UG, TIH, Ressources, Queue)

### Conditionnement et chainage

Le conditionnement et les liens session permettent de relier les Uprocs.

Par la clé Session/Uproc/UG, l’unicité est vérifiée.

En intra nœud, c’est-à-dire lorsque l’UG traduite (par l’UG désigné, type UG ou TIH) est intra-nœud, l’unicité n’a pas de sens car on retombe sur le même objet. Sa présence est cependant vérifiée.

Si l’UG traduite, désigne un autre nœud, depCheck continue le chainage en partant sur les données connues du nœud désigné. C’est la raison pour laquelle les UG mises en jeu dans une chaine doivent être toutes identiques et en place sur les nœuds mis en jeu.

### Tache sur Uproc

Techniquement, il est possible d’exécuter une uproc directement sur une tache. On les identifie clairement lorsqu’il n’y a pas de session dans la définition de la tâche. On en retrouve dans les tache d’exploitation internes du nœud (purge, reporter, extraction).

Fonctionnellement, cette notion est typiquement considérée comme une faute importante de conception.

Cette situation sera ressortie comme une erreur pour le depCheck.

### Session mono-Uproc

Techniquement, il est possible de disposer d’une session ne contenant qu’une uproc. Cette notion est, comme pour le point précédent, une faute grave de conception. Elle sera considérée en tant que telle pour le depCheck

### Entête et pied de session

Il est de grande rigueur de disposer de sessions avec une uproc « entête » (Header) et une uproc « pied » (Trailer).

Ces uprocs sont des éléments clés pour le conditionnement et le fenêtrage des taches. Elles doivent donc sortir les aspects fonctionnels à la chaine. Très épurées, les uprocs sont reconnaissables par la règle de nommage imposée par les normes et standards.

Les entêtes peuvent posséder des éléments propres au fonctionnement de la session (donc de la chaine), des variables héritées par exemple.

Les pieds peuvent posséder des éléments propres à leur exploitation comme la supervision par exemple, par des variables.

depCheck comprend cette notion. Il reste cependant très ouvert au non-usage de cette norme.

## Mode commande, batch, conf, etc…

Outre l’aspect fonctionnel, il est important d’identifier les attaches extra-Dollar Universe. Comme l’exécutif se trouve au sein de l’uproc, cette dernière est décortiquée pour l’analyse.

On en dénombre plusieurs :

* TMPCMD : Détection de chaines de caractère uxexe/ux… ou uni\_dir\_exec/ux… ou uxmgr/ux…
* TMPBATCH : Détection de chaines de caractère portant .cmd, .bat, .jar, .sh et .ksh
* TMPFNC : Détection déclaration de fonctions de type F\_<myFunc>()
* TMPFNCC : Détection d’appel de fonctions de type F\_<myFunc>
* TMPCONF : Détection de chaines de caractère portant .conf
* TMPCLEXT : Contenu de l’uproc de type CLEXT

Le contenu script (contenu uproc) participant au calcul des checksum et parsing :

* SYSCMD <Ligne Uproc type CMD>
* SAPJOB <JOBNAME Uproc type SAP>
* RUNFILE <Ligne Uproc type CLEXT>
* <Contenu CLINT>

Les sorties en fichiers sont énumérées ainsi :

* « DUAS » : les type TMPCMD
* « BATCH » : les types TMPBATCH
* « FUNC » : les types TMPFNC
* « FUNC\_CALL » : les type TMPFNCC
* « CONF » : les types TMPCONF
* « CLEXT » : les types TMPCLEXT

## Les entrées de depCheck

Voici la liste des fichiers en entrée de depCheck

|  |  |  |
| --- | --- | --- |
| Appli\_ID\_Appli.txt | Optionnel | Format <ID> - <Application Name>  ID se trouve dans le label de la session ou de l’uproc |
| Serveur\_ID\_Appli.txt | Optionnel | Format <Node> - <Application Name>  Priorité à l’Appli\_ID |
| <Area>\_upr\_shw.txt | Obligatoire | Uxtrace ou direct command |
| <Area>\_ses\_shw.txt | Obligatoire | Uxtrace ou direct command |
| <Area>\_tsk\_shw.txt | Obligatoire | Uxtrace ou direct command |
| admin\_hdp\_lst.txt | Obligatoire | Uxtrace ou direct command |
| admin\_mu\_shw.txt | Obligatoire | Uxtrace ou direct command |
| <Area>\_res\_shw.txt | Obligatoire | Uxtrace ou direct command |
| admin\_user\_shw.txt | Obligatoire | Uxtrace ou direct command |
| admin\_rule\_shw.txt | Obligatoire | Uxtrace ou direct command |
| uxshw\_queue\_all.txt | Optionnel | Direct command |
| admin\_node\_lst.txt | Obligatoire | Uxtrace ou direct command |
| Uxtraces :  TMP\_<node>\_uxtrace.\*.tar.gz  uxtrace\_<node>\_\*\_DUAS\_\*.tar.gz  uxtrace\_Dollar Universe 6\_\*\_<node>\_\*.zip | | |

## Les sorties de depCheck

Selon les arguments, plus ou moins de fichiers de sortie sont générés

|  |  |  |
| --- | --- | --- |
| depCheck\_Errors.log |  | Liste l’ensemble des erreurs de vérification de depCheck. Cf « Vérification de depCheck » |
| depCheck\_PARSECMD.log |  | Liste des commandes à effectuer pour générer les fichiers d’uxtrace (ou direct command) |
| depCheck\_MemorizedNotNeed.log |  | Liste des UPR qui disposent d’une mémorisation inutile (car non conditionnées) |
| depCheck\_ObjHDP.log | --obj | Liste des objets de type HDP, tout confondu |
| depCheck\_ObjMU.log | --obj | Liste des objets de type MU, tout confondu |
| depCheck\_ObjTSK.log | --obj | Liste des objets de type TSK, tout confondu |
| depCheck\_ObjSESS.log | --obj | Liste des objets de type SESS, tout confondu |
| depCheck\_ObjUPR.log | --obj | Liste des objets de type UPR, tout confondu |
| depCheck\_ObjRES.log | --obj | Liste des objets de type RES, tout confondu |
| depCheck\_ObjUSER.log | --obj | Liste des objets de type USER, tout confondu |
| depCheck\_ObjRULE.log | --obj | Liste des objets de type RULE, tout confondu |
| depCheck\_ObjQUEUE.log | --obj | Liste des objets de type QUEUE, tout confondu |
| depCheck\_ObjNODE.log | --obj | Liste des objets de type NODE, tout confondu |
| depCheck\_Chains.log | --chain | Chainage complet. Il peut être volumineux |
| depCheck\_UnusedSESS.log | --unused | Objets de type SESS inutilisés |
| depCheck\_UnusedUPR.log | --unused | Objets de type UPR inutilisés |
| depCheck\_UnusedMU.log | --unused | Objets de type MU inutilisés |
| depCheck\_UnusedRES.log | --unused | Objets de type RES inutilisés |
| depCheck\_UnusedUSER.log | --unused | Objets de type USER inutilisés |
| depCheck\_UnusedRULE.log | --unused | Objets de type RULE inutilisés |
| depCheck\_UnusedNODE.log | --unused | Objets de type NODE inutilisés |
| depCheck\_UnusedQUEUE.log | --unused | Objets de type QUEUE inutilisés |
| depCheck\_CMDs.log | --cmd | Répartition par topologie des commandes DUAS (ux\*) |
| depCheck\_BATCH.csv | --cmd | Liste des éléments susceptibles de ressembler à des scripts, des fichiers de configuration, des commandes DUAS, etc… |
| depCheck\_FUNCs.csv | --csv | En cas d’utilisation de system de fonction au format F\_<myFun>(), le fichier contient les références dans la Uprocs. |
| ChaineTraitement.csv | --csv | Traitement niveau tâches Format #Node;Task;ses;uprHeader;mu;label;Appli;Rule;planification |
| PlanBatch.csv | --csv | Plan Batch  #Node;Task;Ses;sesLabel;Appli;index;upr;uprLabel;Mu;TargetNode;uprType;Queue;User;Prty;Descr;launchStart |
| Traitement.csv | --csv | Traitements niveau Uproc Format #Node;Task;ses;upr;script;TargetNode;label;Appli;description |
| DependanceT2T.csv | --csv | Format #Node;Task;ses;upr;condno;condSes;TargetNode;condDep;description |
| FormuleDependanceT.csv | --csv | Format #Node;tsk;ses;upr;formula |
| DependanceT2RES.csv | --csv | Format #Node;tsk;ses;upr;condno;nature;condDep;description |
| ParametreT.csv | --csv | Format #Node;upr;varname;value  NotifyRecipient est un mot réservé et correspond aux récipient email dans le paramètre de notification de type email de l’uproc |
| ParametreR.csv | --csv | Format #Node;condDep;label;nature;dir;fname |
| DependanceGroupe.csv | --csv | Format #groupOrder;Node;ses;sources |
| AttributionMU.csv | --csv | Format #Node;MU;TNode |
| Objects.csv | --csv | Format #Node;Task;ses;upr;mu;Appli |
| depCheck\_ExtCmdMU.sh | --ext | Liste des Commandes uxext de MU |
| depCheck\_ExtCmdAddHDP.sh | --ext | Liste des Commandes uxext de HDP |
| depCheck\_ExtCmdTSK.sh | --ext | Liste des Commandes uxext de TSK |
| depCheck\_ExtCmdSESS.sh | --ext | Liste des Commandes uxext de SESS |
| depCheck\_ExtCmdUPR.sh | --ext | Liste des Commandes uxext de UPR |
| depCheck\_ExtCmdRES.sh | --ext | Liste des Commandes uxext de RES |
| depCheck\_ExtCmdAddQUEUE.sh | --ext | Extractions sous forme de commande d’ajout |
| depCheck.gv | --gv | Fichier de définition source GraphViz. |

Attention : Les sorties de depCheck sont principalement dépendantes des interprétations effectuées. Par exemple, une absence de MU peut impliquer son statut « non-utilisé » (unsued) alors que ce n’est pas le cas. L’absence du MU est une erreur qui doit être corrigé avant. Une gestion de ressource en mode commande implique son statut « non-utilisé » aussi.

D’ordre général, il faut tout d’abord assimiler les erreurs pour écarter les vérifications caduques.

### Détection des applications

Une règle de nommage est adoptée. Si elle est respectée, depCheck peut reporter dans les fichiers de sortie csv les applications concernées.

Le fichier de graphèmes comporte un graphème suivi du nom de l’application. Il en existe de 2 types qui respecte exactement le même format :

* Fichier Appli\_ID\_Appli.txt
* Fichier Serveur\_ID\_Appli.txt

Format :

<GRAPHEME> - <Application >

Exemple :

CFIN – Clôture financière

CCTP – Clôture comptable

…

Le graphème doit se trouver dans le label de la session.

Format :

<GRAPHEME> - description de la session

La priorité est faite au niveau du nom du nœud (Serveur\_ID\_Appli.txt), puis Fichier Appli\_ID\_Appli.txt

Il ne faut oublier de positionner l’argument --iddir si le fichier d’application se trouve ailleurs du script depCheck.

# Les vérifications de depCheck

Lors de la lecture des fichiers d’entrée (uxtrace ou sortie des direct Command), depCheck commence à analyser et sortir tout problème sous forme d’erreur par famille extrait dans un fichier log : depCheck\_Errors.log

Voici les familles et analyses effectuées

|  |  |  |  |
| --- | --- | --- | --- |
| Famille | portée | Groupe | Désignation |
| Donnée | Par noeud | MU\_Parse  RES\_Parse  USER\_Parse  RULE\_Parse  QUEUE\_Parse  NODE\_Parse  TSK\_Parse | ERROR parsing DATA  Erreur de lecture de fichiers d’uxtrace ou de direct command. Il arrive que l’utilisateur n’ai pas les droits d’exécution ou que le nœud source (ou s’exécute depCheck) ne peut atteindre le nœud cible. |
| Par noeud | UPR\_Parse | En plus de parsing DATA …  ERROR retreiving uproc script  depCheck vérifie que l’uproc contient bien le bloc script (CLINT) |
| Par noeud | SESS\_Parse | En plus de parsing DATA …  ERROR Session with only one uproc  La session doit contenir plus d’une Uproc |
| Par noeud | HDP\_Parse | En plus de parsing DATA …  ERROR This Child HDP is absent into MU structure  ERROR This parent HDP is absent into MU structure  Si Un HDP existe, il faut que les UG (parent et enfant) existent dans la table des UG |
| Unicité | Global | UPR\_Unicity | Différence observée entre 2 nœuds sur la structure et contenu d’une uproc de même nom |
| Global | SES\_Unicity | Différence observée entre 2 nœuds sur la structure et contenu d’une session de même nom |
| Global | RULE\_Unicity | Différence observée entre 2 nœuds sur la structure et contenu d’une règle de même nom |
| Global | RULE\_Clone | De type information, indication de 2 règles de nom différent mais dont structure et contenu est la même |
| Cohérence | Par noeud | HDP\_Coherency | Test de différence et d’existence TIH en particulier sur le nœud cible |
| Par noeud | MU\_Coherency | Test d’absence de structure d’UG d’ordre général sur un nœud cible |
| Par nœud + MU | MU\_Coherency | Test de différence et d’existence ou de définition d’une UG en particulier sur le nœud cible |
| Chainage | Par noeud | UPR\_Compute | Absence de l’uproc sur la cible Absence de l’uproc cible conditionnée Mémorisation manquante pour le conditionnement |
| Par noeud | MU\_Compute | Absence de l’UG cible via la session (UG, type d’UG, TIH) |
| Par noeud | TSK\_Compute | Vérification de conception :   * Absence de l’Uproc d’entête * Absence de la session * Warning – Absence de la règle |
| Par noeud | tnode\_Compute | UG déclarée sans connaissance du nœud désigné |

# Argumentation

depCheck peut s’exécuter entièrement sans arguent. Le mode connecté est utilisé et les répertoires de transit sont les plus usités.

La sortie est alors un fichier d’erreurs rencontrées lors du processus de compréhension des chaines.

Seulement, il est important que l’utilisateur maitrise les arguments car le fonctionnement est différent selon la situation propre aux uxtraces générés ou non ou absents.

Voici la syntaxe :

depCheck.pl [--force] [--nodes=<node1>[,...]] [--l] [--chain] [--obj] [--cmd] [--unused] [--csv] [--gv] [--nomodel|--model] [--active] [--rulechk] [--dir=<Uxtraces directory>] [--fr]...

## Vérification d’exécution

Une Selon les cas, perl n’est pas connu explicitement du système, il faudra alors préfixer l’exécutable perl avant le script

Vérifier le script avant son exécution permet de contrôler que l’ensemble des librairies sont présentes et connues de perl pour que le script se lance correctement.

perl -c depCheck.pl

En cas d’erreur de vérification il faudra alors installer les librairies via CPAN ou par téléchargement et compilation.

## Arguments

Voici les arguments et leur portée

|  |  |  |
| --- | --- | --- |
| Argument | Par défaut | désignation |
| --dir=<dir> | /tmp/uxtraces  %TEMP% | Répertoire ou se trouvent les uxtraces, soit au format .zip officiel, soit auto généré dans des sous répertoires correspondant au nom du nœud |
| --iddir=<dir> | <Script directory> | Répertoire ou se trouvent les fichiers de transposition   * Graphème 🡪 application * Nœud 🡪 application |
| --csv |  | Active la génération de csv |
| --csvdir=<dir> | <Script directory> | Répertoire ou vont être déposées les sorties csv |
| --obj |  | Génère la synthèse des objets en fichiers log dans le répertorie du script. |
| --chain |  | Génère tout le chainage dans un fichier, dans le répertoire du script. |
| --unused |  | Génère tous les fichiers « unused » contenants les objets non rencontrés dans les chainages |
| --cmd |  | Génère un fichier contenant l’ensemble des commandes DUAS rencontré dans les uprocs. |
| --nodes=node1[,node2,…] |  | Filtre la liste des nœuds à tenir compte dans l’ensemble |
| --force |  | Permet de générer les Direct Command (similaire à l’uxtrace), même si les fichiers d’extraction existent |
| -l |  | Les calculs de vérification se basent sur des dépendances entre nœuds. L’option permet de calculer les dépendances aussi en local. Cette option est très peu utilisée. |
| --nomodel |  | Toutes les taches modèle et non modèles sont traitées. L’argument permet de ne traiter que les non modèles, le réel. |
| --model |  | Toutes les taches modèle et non modèles sont traitées. L’argument permet de ne traiter que les modèles, le design. |
| --active |  | Toutes les tache sont traitées actives et inactives. L’argument permet de ne traiter que les actives, le réel. |
| --rulechk |  | Les règles peuvent être analysées pour vérifier les similitudes |
| --gv |  | Export d’un fichier GraphViz (.gv). Ce fichier peut entrer dans l’outil dot |
| --area APP|INT|SIM | EXP | Seul EXP est traité. L’argument permet de spécifier l’espace à traiter |
| --ext |  | Ressort dans un fichier les commande d’extraction par objet. Peu utile. |
| --fr |  | Tentative de traduire certains champs en français |
| --h |  | … ou n’importe quel argument. Ressort la syntaxe |
| --force |  | Les nœuds sont extraits en uxtraces avant traitement quel que soit leur existence. C’est un mode connecté obligatoirement |

# Exécution dans le contexte

Il est important de tenir compte de la version. EN effet la version 5 et 6 sont incompatible en mode de commande. Il est donc indispensable de reprendre le besoin dans son contexte.

Une migration v5 vers v6, il sera idéal d’avoir un nœud v5 pour les extractions du parc (un référentiel par exemple). Le depCheck utilisera celui-ci.

En nominal, si la v6 est utilisée pour concevoir les plans de production, il est idéal d’avoir une version 6 de Dollar Universe.

## Synchronisation des tables de nœuds

Pour que le mode connecté fonctionne, le nœud instancié dans la SOCIETE concernée, doit être à jour. Pour cela il faut bien s’assurer que les 2 éléments suivants soit synchrones :

* Table des nœuds (uxlst node)
* Fichier de socket (uxsrsrv.sck) (v5)

Pour cela :

* Officialiser la liste cible. Nominalement c’est Univiewer. Filtrer sur Dollar Universe et la SOCIETE
* Aller en mode développement sur le nœud de référence si ce nœud sera le candidat pour faire les extractions
* Mettre à jour la tables de nœud si le nœud est en v5
* Mettre à jour le fichier socket uxsrsrv.sck si le nœud est en v5
* En version 6 la liste est celle représentée dans uniViewer et synchronisée automatiquement

## Génération des uxtraces en mode connecté

depCheck et le nœud permettant d’être en mode connecté est à jour. depCheck peut être exécuté.

Le répertoire des uxtrace sera en sous-jacence du script. Un argument sera alors mis en place. Egalement, pas d’uxtrace (zip ou tar.gz). L’argument --force sera également mis en place.

L’opération peut durer très longtemps. Elle dépend du nombre de nœud, du Traffic réseau et de la charge machine des serveurs distants.

Une fois générés, les fichiers sont analysés par depCheck et des fichiers de rapport sont construits. Ne pas en tenir compte, en procédant de la même manière pour l’ensemble des SOCIETE.

Cette étape permet de vérifier que l’ensemble des nœuds sont bien interconnectés aux 4 nœuds du serveur sur lequel les commande et le depCheck sont exécutés.

Lorsqu’une mise à jour ponctuel d’un uxtrace (ou plusieurs) doit être faite, il sera facile d’ajouter l’argument --nodes et d’exécuter depCheck.

## Exécution

Les uxtrace sont présents avec une fraicheur de donnée suffisante.

depCheck peut alors être exécuté en mode analyse. Comme des instances de nœud Dollar Universe existe, il faudra s’assurer de ne pas être en mode connecté. L’utilisation des fichiers uxtrace ne fait pas apparaitre la SOCIETE et l’analyse depCheck peut être effectuée sur des nœuds de toute nature.

Avant toute chose, il est intéressant de lancer depCheck dans un environnement (une SOCIETE) et filtrant sur les nœuds propre à l’usage : tous les nœuds de production à l’exception du nœud de référence par exemple s’il y en a un.

--nodes=nodeP1,nodeP2,nodeP3,…

Un fichier DependanceGroupe.csv indique les lotissements par nœud. Une nouvelle exécution de depCheck en filtrant sur un lot permet d’avoir une vision plus affinée sur le chainage, avec les erreurs associées.

--nodes=nodeP2,nodeP4,nodeP8

Enfin une comparaison peut être faite entre 2 nœuds d’environnement différent (exemple production vs Validation)

--nodes=nodeP1,nodeV1

Pour les arguments, même si plusieurs ne seront pas utiles, les arguments suivant peuvent être mis en place :

--rulechk

--csv et --csvdir

--obj

--chain

--unused

--cmd

## Analyse

### Consistance

Avant d’analyser les erreurs de conception ou fonctionnelles, il est impératif de vérifier que l’ensemble des données sont bien transmises au dependency Checker : la consistance.

Cette consistance se vérifie par

* La vérification des UG et des nœuds de résidence. Une UG qui fait référence à un nœud absent (donc de la connaissance de depCheck) posera inévitablement un problème dans la vérification effectuée par depCheck. Ceci peut même se traduire par un Direct Command ou uxtrace absent dans le référentiel de depCheck avant analyse (voire inconsistant, en erreur, …). L’action consiste alors à rapatrier ou réaliser une nouvelle fois les uxtrace manquants.
* La vérification des présences de Sessions et Uproc. Ceci commence à approcher une erreur de type « déploiement ». L’action est partagée :
  + Soit le résultat est cohérent au besoin et l’erreur est conceptuelle
  + Soit le résultat est incohérent à la correction doit être corrigée au niveau du plan

### Conception

La notion de conception touche la ou les manières dont sont « accrochées » les uprocs, les conditions attenantes, ainsi que l’unicité des uprocs et sessions.

L’unicité repose sur la vérification de la présence des uprocs du même nom sur le parc. Il est tout à fait irréalisable de positionner des uprocs de même nom dont les propriétés sont différentes. L’uproc est ainsi déployée par une source (référentiel, package, BusinessView) vers sa cible selon la structure des sessions et leurs hiérarchies d’UG.

Le raisonnement est le même pour l’unicité des sessions, qui elle apporte le séquencement et la hiérarchie logique (UG) ou la gravité va apparaitre lorsque la session est partagée sur plusieurs nœuds.

Si un objet est modifié sur la ou les cibles mais pas sur la source, l’intégrité s’en retrouve cassée.

L’action correctrice consiste à :

* Immédiatement rattraper l’erreur de conception pour les sessions en reprenant la cible réelle et la réinjecter sur la source, pour enfin la déployer à nouveau.
* Comprendre le changement directement sur la cible et non les cibles. Il est possible qu’une uproc sur une cible à un rôle différent d’un autre nœud. Il faut dans ce cas changer de nom d’uproc pour que les rôles soient partagés lors du déploiement.
* Renouveler les Direct Commandes ou uxtraces des nœuds impactés puis relancer le depCheck. Tenir compte également de la gestion de l’obsolescence (cf. plus bas)
* Valider la conception par le plan batch

### Obsolescence

L’obsolescence apparait lors de démontage applicatif et lors de modification de chaine. depCheck suit la chaine de bout en bout et procède, par l’absurde, au listage des objets inutilisés.

Il faut cependant être très vigilent sur le résultat d’objets orphelins. Ces derniers sont :

* Soit le fruit de modifications anciennes ou de dépôt par erreur sur un nœud. C’est historique et l’objet peu dans la majorité des cas être supprimé.
* Soit une modification récente de chaine impliquant une non-distribution vers la cible de l’objet mais distribué ailleurs. Ainsi l’objet peut être supprimé sur la cible mais pas au niveau de la source
* La source d’une analyse sur une source pour une non-production. Dans ce cas il est essentiel de comprendre le sort de l’objet qui est soit une obsolescence, soit un futur objet utile.

### Les commandes

depCheck tente de ressortir le mode commande Dollar Universe dans un fichier. Ce mode commande est important car il est un langage qui n’est plus à la portée du produit mais comme si la commande suffit à elle-même dans une console de commande.

En cas de migration ou changement de chaine, ces commandes peuvent/doivent être modifiées.

### Les batches externes

depCheck sort également par reconnaissance de ligne de commande sur des batch/exécutables externes. Ces batch peuvent contenir eux même des commandes Dollar Universe. Il est donc utile que l’exploitant se repose sur ce rapport pour garantir que les potentielles commandes dans ces scripts externes soient suivis/adaptées.

### Les objets

Outre une énumération des objets par type dans des rapports, il y a également un fichier qui recense toutes les clés propres à la supervision.

En effet, la supervision générique est désormais de type « objet » avec des classes instanciées. Il en découle une liste exhaustive des potentielles alertes de tout le parc mis en analyse par depCheck

Selon la modification des chaines de traitement, cette liste « fluctue ». Il peut servir de point d’entrée pour les outillages de supervision.

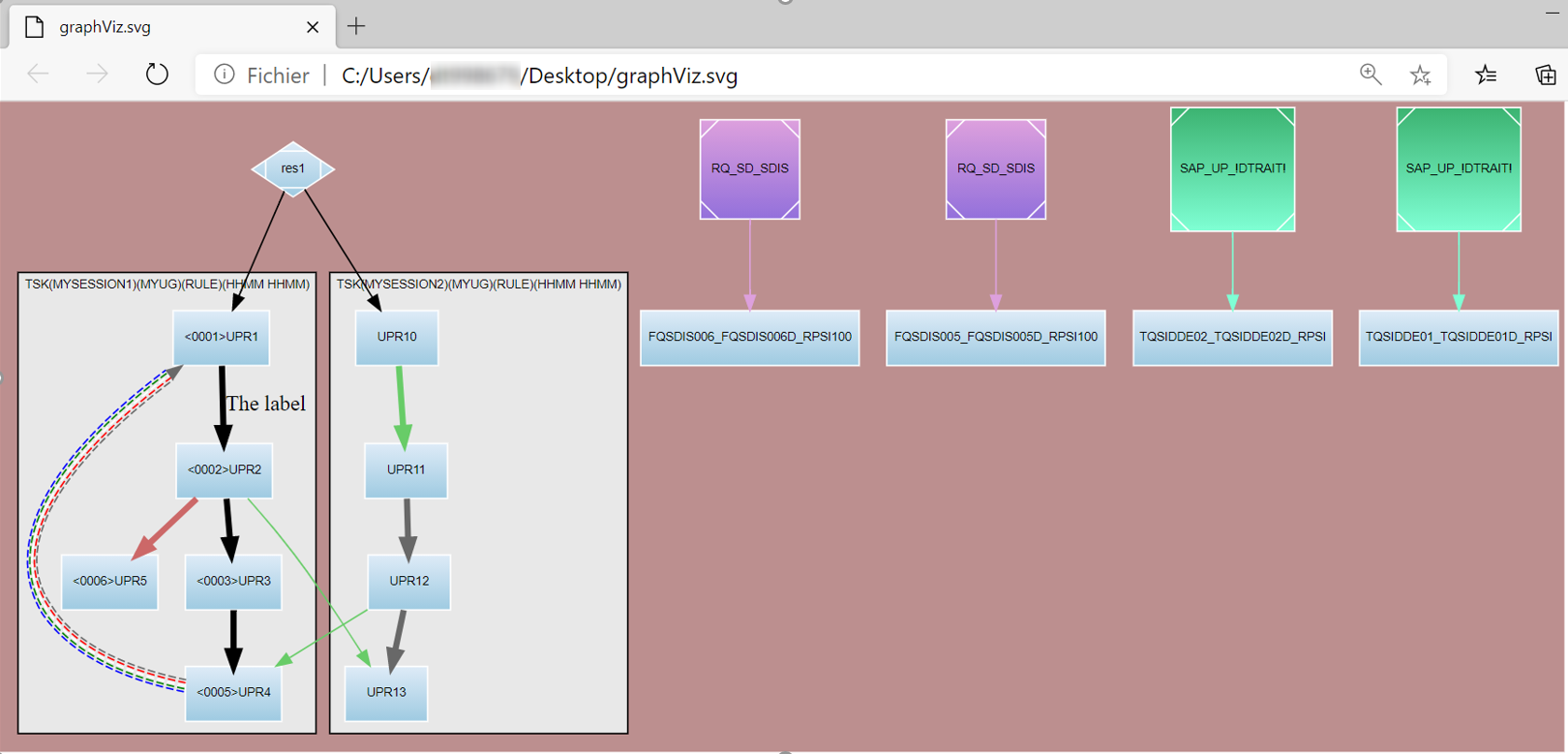
## GraphViz

Avec l’argument --gv, depCheck génère un schéma au format graphViz

graphViz peut ainsi exploiter ce fichier pour représenter sous différents formats le schéma du plan de production.

Exemple de génération par graphViz :

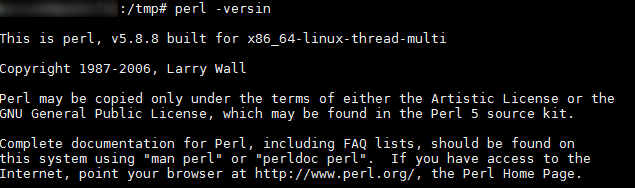
./dot –Tsvg ./depCheck.gv –o depCheck.svg



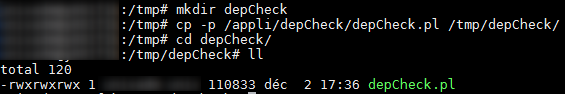
# Beginner

## Unix

Vérifier le perl



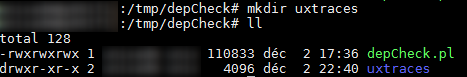
Préparer votre structure depCheck



Vérifier l’intégrité du script



Préparer le lieu où se trouveront tous les uxtraces

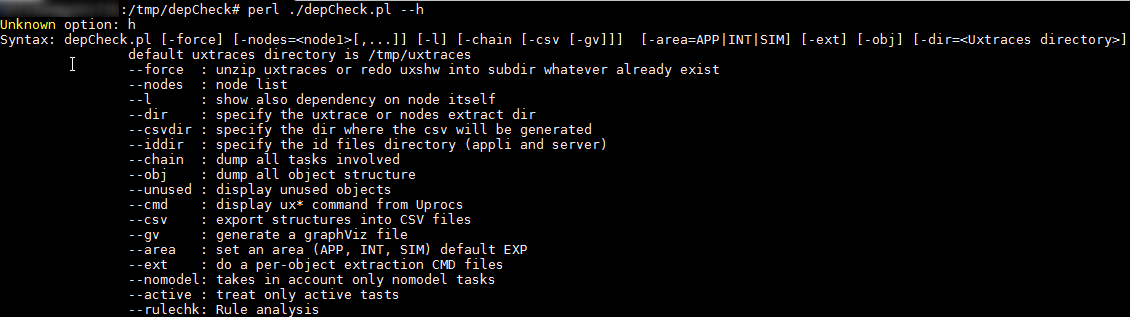


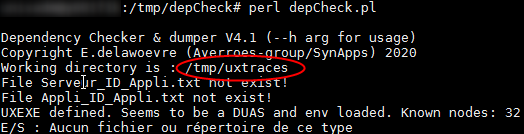
Notons qu’ici l’argument --dir sera /tmp/depCheck/uxtraces

**Mode connecté** : Vérifier votre nœud local en chargeant son environnement

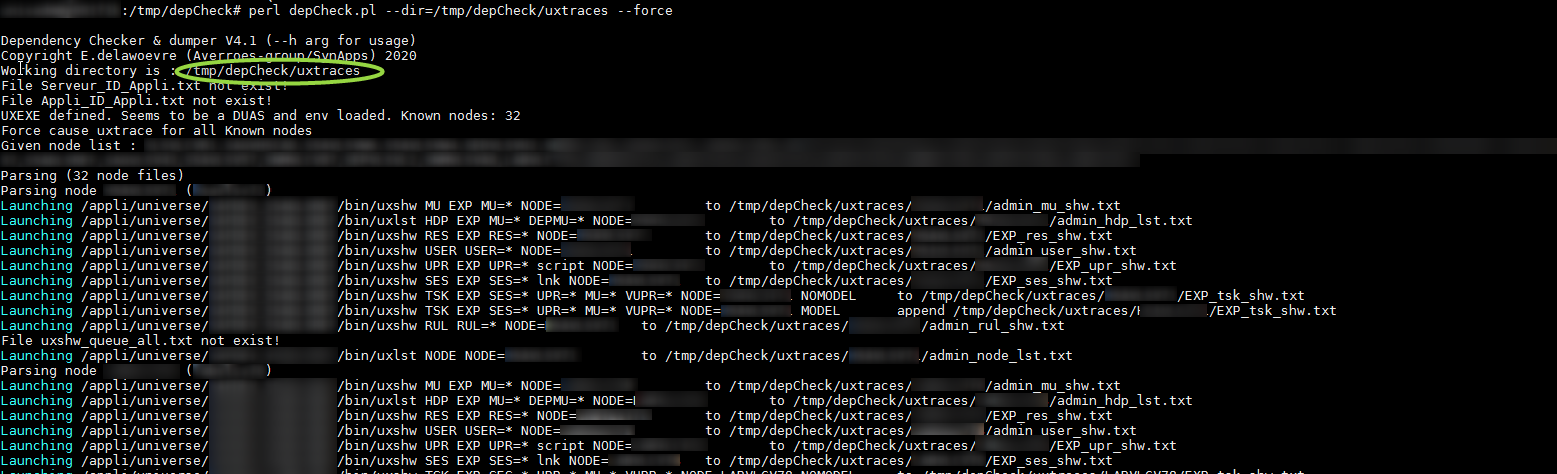


Vérifier la version et utilisation



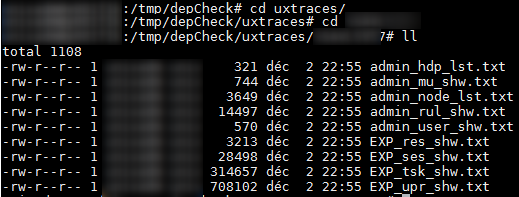


C’est parti avec notre argument --dir



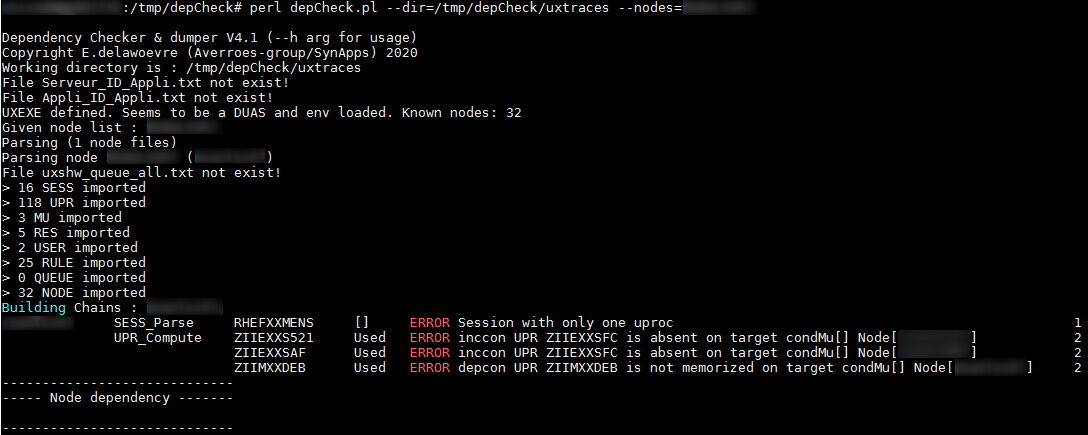
Ici, aucun control des nœuds à traiter, tous les nœuds connus du nœud chargé seront exportés. Noter que les nœuds doivent être de la même version dans ce mode connecté.

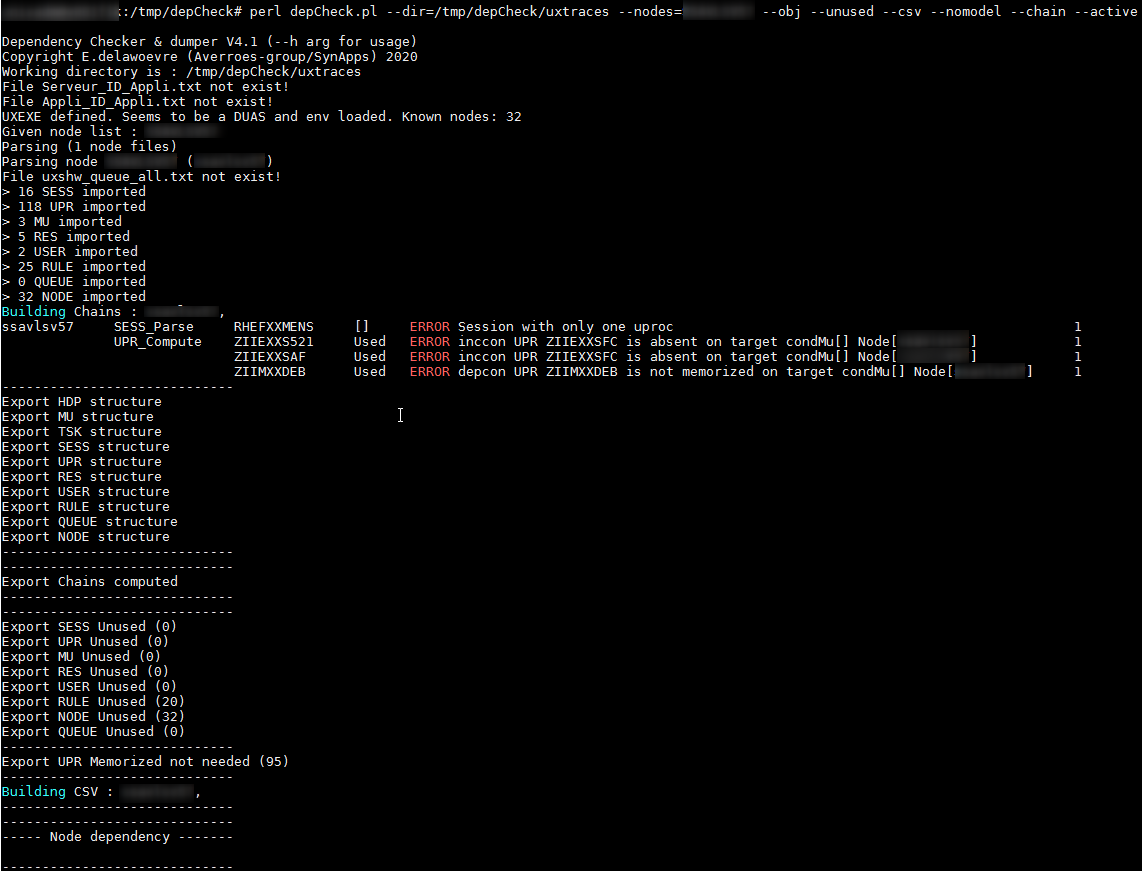
Vérifions un uxtrace



Tout semble bon, plaçons un seul nœud en argument et mettons quelques arguments. Inutile de rajouter l’argument --force car le nœud a déjà été extrait.

Ceci revient d’ailleurs à un mode Non-connecté





**Mode Non-Connecté**

Ici le principe est le même.

Noter tout de même que les uxtraces peuvent être placés dans le répertoire uxtraces (les zip et/ou tar.gz). Il faudra alors placer dans l’argument nodes le nom du nœud avec la bonne casse.





On voit assez clairement que le nœud en paramètre montre des nœuds dépendants. Leurs absences en argument entrainent de nombreuse erreur d’absence.

Dans tous les cas, les fichiers d’analyse sont tous rafraichis dans le répertoire du script :

