| Référence : DIT-MEP-DOC-NT-0177 | Indice : 03 |  | Définitif : |
| --- | --- | --- | --- |
| **Environnement d’hébergement des scripts et Règles de développement sous Unix** | | | |
| Date : 2013-03-13 | |  | 62 pages |

| Résumé |
| --- |
| Ce document a pour objet de fournir les règles et recommandations de développement de shell scripts sous Unix. |

|  |  |  |
| --- | --- | --- |
| Rédacteur | Vérificateur | Approbateur |
| Guillaume Ferrari | Guillaume Ferrari | Guillaume Ferrari (oui je suis partout) |

Documents associés

| Référence | Titre |
| --- | --- |
|  |  |
|  |  |
|  |  |

Documents annulés par le présent document

| Référence | Titre |
| --- | --- |
|  |  |
|  |  |
|  |  |

Historique

| Indice | Modifications |
| --- | --- |
|  |  |
|  |  |
|  |  |

Destinataires

| Entité | Nom |
| --- | --- |
|  |  |
|  |  |
|  |  |

Lieu de stockage

|  |
| --- |
| N/A |

Conventions typographiques

Les conventions suivantes sont utilisées dans la suite du document :

les commandes à taper dans un terminal sont indiquées dans un cadre :

# uname -a

Linux nom.domaine 2.6.18-194.26.1.el5 #1 SMP Fri Oct 29 14:21:16 EDT 2010 x86\_64 x86\_64 x86\_64 GNU/Linux

les extraits de fichiers utilisent la même convention :

UsePrivilegeSeparation yes

une note informative utilise le formalisme suivant :

|  |  |
| --- | --- |
|  | L’invite $ devant une commande indique qu’elle peut être exécutée par un utilisateur non privilégié |

un avertissement indique un risque, et est présenté ainsi :

|  |  |
| --- | --- |
|  | L’invite # devant une commande indique qu’elle doit être exécutée en tant qu’utilisateur privilégié[[1]](#footnote-1). |

les points d’attention sont en surbrillance dans le texte ;

les termes techniques tels que cat sont signalés par une fonte à chasse fixe.

Assistance

Le support du système de référence est assuré par :

Merci d’utiliser exclusivement ce circuit pour vos demandes d’assistance.

Remarques & suggestions

Si vous constatez une erreur, ambiguïté, manque de clarté, ou si vous avez des améliorations à suggérer sur ce document, merci d’écrire à :

Guillaume.ferrari@econocom.com

Table des matières

[1 INTRODUCTION ET OBJECTIFS DU DOCUMENT 6](#_Toc71863564)

[1.1 Contexte 6](#_Toc71863565)

[1.2 A qui est destiné ce document ? 6](#_Toc71863566)

[1.3 Objectifs du document 6](#_Toc71863567)

[1.4 Applicabilité des règles et recommandations décrites 6](#_Toc71863568)

[1.5 Formalisme et éléments présentés 6](#_Toc71863569)

[1.5.1 Eléments présentés 6](#_Toc71863570)

[1.5.2 Recommandations et règles 6](#_Toc71863571)

[1.6 Gains apportés par le respect de ce document 7](#_Toc71863572)

[1.6.1 Cohérence du parc exploité 7](#_Toc71863573)

[1.6.2 Possibilité de mutualisation 7](#_Toc71863574)

[1.6.3 Pourquoi respecter ces éléments dès la phase projet ? 7](#_Toc71863575)

[2 Les composants d’un serveur Unix 8](#_Toc71863576)

[2.1 Schéma logiciel d’un serveur 8](#_Toc71863577)

[2.2 Les systèmes de référence 8](#_Toc71863578)

[2.3 Les objets d’infrastructure 8](#_Toc71863579)

[2.4 Les objets applicatifs 9](#_Toc71863580)

[2.4.1 Les instances d’objets d’infrastructure 10](#_Toc71863581)

[2.4.2 Les applicatifs « purs » 10](#_Toc71863582)

[3 IDENTIFICATION DES APPLICATIONS 11](#_Toc71863583)

[3.1 Introduction 11](#_Toc71863584)

[3.2 Les NNA 11](#_Toc71863585)

[3.2.1 Généralités 11](#_Toc71863586)

[3.2.2 Attribution 11](#_Toc71863587)

[3.2.3 Généralités 11](#_Toc71863588)

[3.2.4 Dans quel cas un projet doit-il utiliser un UID/GID et un compte propre ? 12](#_Toc71863589)

[3.2.5 Bénéfices du respect de ces consignes 12](#_Toc71863590)

[3.2.6 Règles d’affectation des UID et GID 13](#_Toc71863591)

[4 NORMES PACKAGES ET SIGNATURES 14](#_Toc71863592)

[4.1 Normes de nommage des Packages 14](#_Toc71863593)

[4.2 Normes de nommage des Signatures 14](#_Toc71863594)

[4.2.1 Champ <nna> 15](#_Toc71863595)

[4.2.2 Champ <nomapplication.version> 15](#_Toc71863596)

[4.2.3 Champ <os> 15](#_Toc71863597)

[4.2.4 Champ <vos> 15](#_Toc71863598)

[4.2.5 Exemple 16](#_Toc71863599)

[5 ARBORESCENCE DES APPLICATIONS 17](#_Toc71863600)

[5.1 Objectifs et principes de l’arborescence proposée 17](#_Toc71863601)

[5.1.1 Objectifs et principes généraux 17](#_Toc71863602)

[5.1.2 Nommage de l’application 17](#_Toc71863603)

[5.2 PRESENTATION DE L’ARBORESCENCE APPLICATIVE SOUS UNIX 17](#_Toc71863604)

[5.3 Détail des répertoires 17](#_Toc71863605)

[5.3.1 Espace de livraison 17](#_Toc71863606)

[5.4 Produits Web 18](#_Toc71863607)

[5.4.1 Les différents produits web 18](#_Toc71863608)

[5.4.2 Nom d’instance 18](#_Toc71863609)

[5.4.3 Implantation 18](#_Toc71863610)

# INTRODUCTION ET OBJECTIFS DU DOCUMENT

## Contexte

Les opérateurs informatiques de l’entreprise XXX exploitent la plupart des applications métiers de l’entreprise. Afin de répondre aux exigences des maîtrises d’ouvrage, en particulier pour améliorer la disponibilité des applications sans augmenter le coût d’exploitation, les opérateurs ont besoin d’industrialiser les environnements dont ils ont la responsabilité. Un des critères d’industrialisation est l’homogénéité du parc exploité.

## A qui est destiné ce document ?

Ce document est à destination de l’ensemble des intervenants pouvant être amenés à manipuler des objets techniques sur des serveurs UNIX utilisés par les opérateurs d’exploitation :

* en amont, aux équipes de développement, qui devront tenir compte de l’ensemble des contraintes indiquées au cours de la phase de développement ;
* pendant la phase d’Intégration / Qualification, aux intégrateurs, qui valideront la norme des développements livrés, et qui pourront réaliser des développements complémentaires ;
* aux exploitants, qui manipuleront les objets mis en production.

## Objectifs du document

Ce document a pour objectif de définir des règles et recommandations communes, à l'attention de tous les contributeurs des objets techniques installés en production.

Ces règles et recommandations découlent pour la plupart du simple bon sens ou de principes généraux d'exploitation. Certaines sont également le fruit de l'expérience acquise au cours de l'exploitation de différentes applications. Ce document est donc susceptible de s'enrichir de nouveaux éléments.

## Applicabilité des règles et recommandations décrites

Les règles et recommandations décrites sont applicables à tous les nouveaux objets développés ou mis en place à partir de la date de validation du document.

Par conséquent, l’ensemble des développements existants ne devra pas impérativement être revu, sauf si l’état actuel a un impact sur l’exploitabilité.

Toutes les dérogations seront traitées au cas par cas.

## Formalisme et éléments présentés

### Eléments présentés

Le document présente en détail les éléments techniques les plus structurants pour les équipes d’exploitation :

* règles générales de gestion de l’espace disque sur les serveurs ;
* nommage et identification des applications et de leurs objets techniques ;
* implantation des objets techniques ;

### Recommandations et règles

La présente norme est composée de deux types de principes :

* des recommandations, qui sont des conseils de programmation visant une plus grande homogénéité des scripts réalisés au sein de la DIT,
* des règles fondamentales, dont le respect est impératif afin d’obtenir le niveau de cohérence attendu.

Les recommandations sont présentées dans une zone encadrée comme ci-dessous :

Ceci est une recommandation.

Les règles sont mises en évidence par une zone encadrée comme ci-dessous :

Ceci est une règle.

## Gains apportés par le respect de ce document

Le respect de cette implantation procure plusieurs avantages.

### Cohérence du parc exploité

Le respect de cette implantation assure une cohérence des applications exploitées et par conséquent facilite le travail de mise en production et d’exploitation.

Lors des phases de mise en production, un outillage mutualisé peut être constitué plus facilement.

En exploitation, l’utilisation de règles d’implantation communes permet aux exploitants de mieux connaître les applications

### Possibilité de mutualisation

Le respect de cette implantation permet la mutualisation de plusieurs applications sur un même serveur sans risque de chevauchement des arborescences des différentes applications. Cela permet en particulier de réaliser les phases de développement ou d’Intégration / Qualification sur des serveurs mutualisés, et de réaliser ainsi des économies.

Les différents cas pris en compte par ce document sont les suivants :

* une application unique sur un serveur unique ;
* une application unique sur un serveur unique avec plusieurs « instances d’unité » ;
* plusieurs applications sur un unique serveur.

### Pourquoi respecter ces éléments dès la phase projet ?

Lors de la phase de développement, l’attention est portée principalement sur le contenu fonctionnel des développements, au dépend de l’architecture technique.

Malheureusement, l’ensemble des points techniques présentés sont très structurants pour les phases de mise en production puis pour l’exploitation. Leur « non-respect » ne bloquera pas la mise en production ou l’exploitation, mais risque de les rendre plus complexes. L’impact pourra être un retard, un défaut de qualité ou un surcout.

Il est souhaitable de respecter les contraintes présentées dès le développement.

# Les composants d’un serveur Unix

Les serveurs UNIX utilisés par les opérateurs sont composés des éléments suivants :

* un système de référence ou souche système : c’est le système d’exploitation ;
* un certain nombre d’objets d’infrastructure : ce sont les produits et logiciels divers utilisés ;
* des objets applicatifs, qui sont soit des instances de produits ou logiciels, soit des objets applicatifs « purs » soit une combinaison des 2.

Les sous-chapitres qui suivent détaillent ces éléments.

## Schéma logiciel d’un serveur

On peut décomposer les différentes couches logicielles d’un serveur de la façon suivante :

Les différents composants sont décrits dans la suite.

Ils sont classés en 3 catégories :

* systèmes de référence (bleu foncé)
* objets d’infrastructure (bleu clair)
* objets applicatifs (orange et rouge)

## Les systèmes de référence

Ils sont représentés en **bleu** sur le schéma précédent.

Les systèmes de référence sont des systèmes d’exploitation

.

## Les objets d’infrastructure

Ils sont représentés en **bleu** sur le schéma précédent.

Les objets d’infrastructure sont tous les composants complémentaires au système, qu’une application peut être amenée à utiliser, y compris pour son exploitation. Les objets d’infrastructure comprennent en particulier l’ensemble des produits au référentiel technique.

Exemples : logiciel Apache, Nginx

**Remarque :** comme indiqué, il s’agit des logiciels et non pas des instances associées.

Exemples : pour oracle, l’objet d’infrastructure est limité au moteur oracle, et non pas aux bases créées ensuite..

## Les objets applicatifs

Les objets applicatifs sont fournis par les « groupes projets ».

Tout composant n’étant pas un système de référence ou un objet d’infrastructure est considéré comme applicatif.

Il existe 2 types d’objets applicatifs, sachant qu’on peut combiner les 2 types. Les 2 sous-chapitres suivants présentent ces 2 types.

### Les instances d’objets d’infrastructure

Les instances d’objets d’infrastructure sont aussi appelées « instances techniques ».

Ils figurent en **orange** sur le schéma ci-dessus.

L’instance technique n’est pas le logiciel, mais l’utilisation qui en est faite.

**Exemple 1 : oracle**

* Le moteur oracle (binaires oracle, librairies.. installés dans $ORACLE\_HOME) est un objet d’infrastructure.
* Les différentes bases créées et qui utilisent ce moteur constituent des instances techniques.

**Exemple 2 : apache**

* Le moteur apache (constitué des binaires et librairies du produit) est un objet d’infrastructure.
* Chaque instance apache, constituée en particulier d’une arborescence contenant les répertoires « conf » et « htdocs », est une instance technique.

Les instances techniques doivent respecter la normalisation associée à chaque produit.

### Les applicatifs « purs »

Les « applicatifs purs » sont des développements ou des modules qui ne sont pas des instances techniques, ou des développements ou modules réalisés autour des instances techniques.

Ils figurent en **rouge** sur le schéma ci-dessus.

**Exemple 1 : applicatif pur**

Un applicatif pur peut être constitué de binaires, de librairies et de fichiers annexes fournis par l’équipe projet.

**Exemple 2 : modules annexes à une instance technique**

Des traitements annexes à une instance weblogic peuvent être développés pour modifier les données d’un site. Ces développements sont considérés comme un applicatif pur.

Les équipes projets disposent donc d’une grande autonomie pour l’implantation de ce type de composant (par rapport aux instances techniques dont l’implantation est structurée par l’outillage fourni avec les produits associés).

Les équipes projet doivent donc porter une attention particulière à toutes les règles et recommandations présentées dans ce document lors des développements de modules applicatifs « purs ».

**Exemple 3 : modules complémentaires à un objet d’infrastructure**

Des librairies php non présentes dans la souche sont considérées comme des objets applicatifs purs.

**Exemple 4 : produit non référencé**

Tout produit non référencé par la DIT est considéré comme étant un applicatif pur.

Exemples : SAS, Sybase.

# IDENTIFICATION DES APPLICATIONS

## Introduction

L’identification des applications a plusieurs objectifs :

* disposer d’un référentiel applicatif unique et fiable des applications ;
* permettre la mutualisation d’un certain nombre de ressources (serveurs, infrastructures, outils…) ;
* permettre un inventaire des logiciels utilisés pour les différentes applications.

Elle se traduit :

* par l’attribution d’un identifiant unique à chaque application : le Numéro National d’Application ou « NNA » ;
* dans les cas où c’est utile, par l’attribution d’un nom d’utilisateur et d’un nom de groupe et des identifiants associés, qui peuvent être utilisés sur les serveurs UNIX. On parle dans la suite des « UID et GID sous UNIX ».

## Les NNA

### Généralités

Les Numéro Nationaux d’Application ou NNA ont remplacé les « trigrammes applicatifs ».

Les NNA sont constitués de 3 caractères alphanumériques.

**Exemple**: « acd » pour l’application « Accueil Distributeur ».

Les NNA peuvent être utilisés dans les chemins des arborescences applicatives. Ils permettent principalement la mutualisation d’instances sur des serveurs mutualisés.

### Attribution

La gestion des NNA est géré par Guillaume Ferrari (encore lui), voici les attributions :

* Groupe A: MWA
* Groupe B: MWB
* Groupe C: MWC

### Généralités

Sur un serveur UNIX, aucun utilisateur ou groupe ne doit être créé en laissant le système d’exploitation affecter un UID ou GID par défaut : chaque utilisateur ou groupe existant doit être déclaré dans « l’annuaire national des UID/GID ».

Si une application a besoin d’utiliser des utilisateurs et groupes dédiés, en complément du NNA, on peut aussi lui attribuer les éléments suivants :

* un nom d’utilisateur UNIX dédié et l’identifiant associé (UID) ;
* un groupe d’utilisateur UNIX dédié et l’identifiant associé (GID).

### Dans quel cas un projet doit-il utiliser un UID/GID et un compte propre ?

Toutes les applications n’ont pas besoin d’un UID/GID et de comptes propres. Par exemple, si un projet de type « site internet » est constitué d’une base oracle, d’une instance weblogic et d’une instance apache, les 3 instances utiliseront les comptes déjà définis pour ces produits (« webadm » et « webgrp »).

Par contre, si une application utilise un progiciel nécessitant la création de comptes spécifiques et que ce progiciel n’est pas utilisé par la DIT, alors la création d’un compte spécifique devra être réalisée.

Les applications utilisant du code autonome (c’est à dire non lié à un produit) doivent :

* soit utiliser un compte spécifique dédié ;
* soit partager utiliser un compte déjà existant. Par exemple, dans le cas d’une application constituée d’une base oracle et de scripts « sql » manipulant les données de la base, les scripts « sql » pourront avoir pour propriétaire « oracle » comme la base.

### Bénéfices du respect de ces consignes

L’utilisation des UID/GID référencés permet :

* de simplifier tous les transferts de données entre environnements. En effet, un certain nombre d’outils d’exploitation, de la commande « tar » jusqu’à l’infrastructure de sauvegarde / restauration Networker conservent les UID/GID. En cas de non-respect, un réajustement délicat peut être nécessaire.
* Les UID/GID utilisés par le projet peuvent être en conflit avec ceux d’un autre produit. Attention, la vérification sur les serveurs de production ne suffit pas à contourner cette règle : un nouveau produit peut être installé sur un serveur.

### Règles d’affectation des UID et GID

Tout système Unix accepte des UID et GID dont la valeur est comprise en 0 et 65535 au minimum. Certains, dans les premières valeurs, sont réservés au système et ne peuvent être utilisés à d’autres fins.

Pour faciliter l’affectation de ces UID et GID, le champ de valeurs utilisables a été découpé en quatre tranches :

* 0 à 999 : réservé au système et aux outils associés au système.
* 1000 à 3999 : « produits standards », il s’agit des logiciels du référentiel technique utilisés largement sur les serveurs (Apache, CFT, Oracle, …).

Voici les UID/GID recommandés pour les produits utilisés dans l’exercice :

|  |  |  |
| --- | --- | --- |
| Compte | UID | GID |
| apache:apache | 1111 | 1110 |
| www-data:www-data | 1101 | 1100 |
| nginx:nginx | 1121 | 1120 |

* 4000 à 9999 : Applications >10000 : réservés aux usages divers, identifiants de personnes physiques
* L’usage de cette tranche est libre et n’apparaît pas dans le répertoire national.

# NORMES PACKAGES ET SIGNATURES

L’architecture du stockage est très structurante pour les environnements de production.

En effet, une architecture disque non adaptée peut rendre plus difficile l’exploitation de l’application, ou rendre impossible la mise en place de certaines fonctionnalités (par exemple la mise en place d’un PRA).

Une attention particulière doit être portée à l’élaboration de cette architecture, avant l’installation des serveurs de production.

L’architecture des disques et leur configuration détaillée fait partie des prérogatives des équipes projet.

## Normes de nommage des Packages

* Le nom doit être :

<NNA>\_<VEM>\_<TYPE>\_<COMP>.<extension>

* **NNA** = 3 caractères
* **VEM** de l'appli = xx.yy.zz
* **TYPE** =

• dans le cas des packages applicatifs : **optionnel** ; jusqu'à 15 caractères ; permet de spécifier les packages particuliers : OUT, RAPPORT…

* **COMP** =

• dans le cas des packages applicatifs : jusqu'à 8 caractères, permet de préciser le rôle du package (Exemple : DATA, WEB)

* **extension** :

• .tar.gz pour les packages Unix

• .zip pour les packages Windows

Caractères autorisés : **alphanumérique,** « **\_** », « **-** », «  **.** »

**Tous les autres caractère sont interdits.**

## Normes de nommage des Signatures

***Le fichier de signature doit être un fichier vide situé dans le répertoire /etc/conf\_machine/signature***

***Ce fichier est utile pour identifier quels sont applications installées sur le serveur.***

Le nom est en **minuscules** uniquement.

Le nom est toujours constitué de **4 champs.**

Les champs sont séparés par des underscores ‘\_’

**Attention ! Les quatres champs sont obligatoires. Le caractère underscore « \_ » est réservé à la séparation des champs. Il ne doit pas être utilisé à l’intérieur d’un champ.**

**A l’intérieur d’un champ (entre deux underscores), les caractères « . » ou « - » permettent de faire des séparations.**

Le nom du fichier, du package, de l’objet métier et de sa version se déclinent comme suit :

**<nna>\_<nomapplication.version>\_<os>\_<vos>**

Les règles à suivre pour le contenu de chaque champ sont décrites ci-dessous.

### Champ <nna>

Ce champ doit contenir le NNA de l’application (Numéro National d’Application).

### Champ <nomapplication.version>

Le champ **<nomapplication.version>** se décline de la façon suivante :

Nomapplication.versionapplication

**Exemples** :

Pour une application nommé TATAYOYO, le champ <nomapplication.version> peut valoir :

tatayoyo.3.2

### Champ <os>

**<os>** doit contenir le nom de l’OS.

Les valeurs possibles sont obligatoirement les suivantes :

aix

solaris

linux

windows

### Champ <vos>

Ce champ contient la version de l’OS. La version peut aller jusqu’à la version du noyau si le package le nécessite.

Les règles suivantes sont définies pour les versions  :

* **Pour AIX :** **x.y** où x et y sont obligatoires. Exemples : 5.2, 5.3 (attention ! AIX 4 est une exception, on précise les 3 niveaux : 4.3.3)
* **Pour SOLARIS :** **x.** Exemples : 8, 10
* **Pour LINUX (UBUNTU) :** **UBUx.** Exemples : ubu18, ubu19, ubu20
* **Pour LINUX (CENTOS) :** **CENx.** Exemples : cen6, cen7
* **Pour Windows :** **[version]**. Exemples : 2000, 2003

### Exemple

L’application tatayoyo ayant le NNA tyo en version 01.00.00 installée sur CentOS7 aura pour signature :

tyo\_tatayoyo.1.0\_cen7

# ARBORESCENCE DES APPLICATIONS

Les applications sont les modules livrés par les équipes-projet. Comme vu précédemment, l’application est constituée « d’instances techniques » et « d’objets applicatifs purs ».

## Objectifs et principes de l’arborescence proposée

### Objectifs et principes généraux

Le principe général est de séparer les fichiers par catégories :

* code applicatif ;
* fichiers de configuration et paramétrage ;
* fichiers de logs ;
* fichiers de données persistantes ;
* fichiers de données temporaires.

Chacune de ces catégories peut être séparée en fonction de sous-ensembles. Par exemple, les différentes versions du code applicatif seront séparées.

Un système de liens symboliques permettra de faciliter la gestion des montées de version.

L’application pourra être séparée en « modules » ou « composants » qui pourront évoluer indépendamment en production.

La mise en place d’une nouvelle version de l’application fera l’objet d’une livraison complète d’un ou plusieurs modules.

### Nommage de l’application

Dans la suite du document, **<application>** fait référence au nom de l’application concernée (ou éventuellement à son trigramme applicatif).

Les noms des répertoires composant la sous-arborescence spécifique à l’application doivent exclusivement être constitués de lettres de a à z et de chiffres de 0 à 9.

**<application>** doit être constitué exclusivement de lettres de a à z en minuscule, et de chiffres de 0 à 9. L’utilisation du « \_ » est à proscrire car ce caractère peut être utilisé comme séparation dans le nom d’instances (Informatica ou Weblogic par exemple).

## PRESENTATION DE L’ARBORESCENCE APPLICATIVE SOUS UNIX

## Détail des répertoires

### Espace de livraison

L’espace de livraison est l’espace où les objets applicatifs seront déposés avant leur mise en place ou activation en production.

Implantation :

/appli/<nom\_appli>/LIVR/<version>

La version pourra être une version ou une date de livraison.

## Produits Web

### Les différents produits web

Les différents produits « web » du référentiel technique sont principalement :

* apache ;
* nginx;
* mysql ;
* mariadb

### Nom d’instance

Pour tous ces produits, il faut définir un nom “d’instance”.

Si l’application n’utilise qu’une unique instance, le nom de l’instance peut soit être :

* le NNA de l’application (obligatoire sur une plate-forme mutualisée) ;
* dérivé du nom de l’application et choisi par le projet.

Si l’application utilise plusieurs instances, on peut accoler au nom choisi ou au NNA un suffixe permettant de distinguer les instances. Les noms et suffixes ne doivent pas être séparés par un caractère quelconque.

### Implantation

**Instances applicatives :**

Les instances applicatives sont implantées dans :

/appli/projects/<instance>/<produit>

Exemple :

/appli/projects/regal/apache2

A la différence de l’implantation générale proposée, l’implantation des applications web ne permet pas de gérer la cohabitation de plusieurs versions en parallèle.

***A minima, /appli/projects/<instance> doit être un filesystem.***

Si le projet le souhaite, les différents répertoires /appli/projects/<instance>/<produit> peuvent être des filesystems. Ce choix peut être fait si les différents produits évoluent de façon très différente en terme d’occupation d’espace disque.

**Logs :**

Les logs sont implantés dans :

/var/projects/<instance>/<produit>

***A minima, /var/projects/<instance> doit être un filesystem.***

Si le projet le souhaite, les différents répertoires /var/projects/<instance>/<produit> peuvent être des filesystems. Ce choix peut être fait si les différents produits évoluent de façon très différente en terme d’occupation d’espace disque

✦ Fin du document ✦

1. Par exemple, par l’utilisateur root via un sudo, etc [↑](#footnote-ref-1)