Introduction aux Systèmes d'Exploitations

Unix

–

GNU / Linux

Jean-Michel Bruneau

1. Objectifs, Temps & Evaluation

Objectifs :

Instalation d'un système d'exploitation GNU / Linux.

Prise en main les principales commandes des systèmes Unix, manipulation de fichier et la manipulation de leur contenu, environnement et processus.

Programmation en Shell Unix.

Temps :

- 1,5 H CM : introduction aux systèmes Unix, commandes de base
- 2 x 3H de TD / TP : instalation d'un système GNU / Linux, commandes de base.
- 2 x 3H de TD / TP : programation en Shell Unix.

Evaluation (prévisionnelle) :

Contrôle continu : Programmation BASH.

Contrôle final : QCM.

2. Unix – « Le Commencement »

Qu'est-ce que Unix ?

Le système d'exploitation Unix est un ensemble de programmes qui agissent comme un lien entre l'ordinateur et l'utilisateur.

Les programmes informatiques qui allouent les ressources système et coordonner tous les détails des composants internes de l'ordinateur est appelé le système d'exploitation ou le noyau.

Les utilisateurs communiquent avec le noyau par un programme connu sous le nom **shell**. La « **coquille** » est un interpréteur de ligne de commande ; il traduit les commandes saisies par l'utilisateur et les convertit en un language compris par le noyau.

- **Unix** a été développé en **1969** par un groupe d'employés **d'AT&T**, Ken Thompson, Dennis Ritchie, Douglas McIlroy, et Joe Ossanna chez **Bell Labs**.
- Il existe différentes variantes Unix disponibles sur le marché. **Unix Solaris, AIX, HP Unix** et **BSD** sont quelques exemples. **GNU/Linux** est également un système de type « Unix » qui est disponible gratuitement.
- Plusieurs personnes peuvent utiliser un ordinateur Unix en même temps :
  Unix est un système **multi-utilisateurs**.
- Un utilisateur peut également exécuter plusieurs programmes en même temps :
  Unix est un environnement **multi-tâches**.

Qu'est-ce que GNU ?

GNU est un système d'exploitation libre créé en **1983** par **Richard Stallman**, maintenu par le projet GNU. Son nom est un acronyme récursif qui signifie en anglais « **GNU's Not UNIX** » (littéralement, « **GNU n'est pas UNIX** »). Il reprend les concepts et le fonctionnement d'UNIX.

Les logiciels qui composent GNU sont généralement utilisés en association avec des logiciels libres issus d'autres projets tels que le noyau Linux.

Son symbole est un « gnou », bovidé vivant en Afrique :

source : wikipedia

Qu'est-ce que Linux ?

Linux est, au sens restreint, le noyau d'un système d'exploitation Linux, et au sens large, tout système d'exploitation fondé sur le noyau Linux. Nous couvrirons ici le sens large.

À l'origine, le noyau Linux a été développé pour les ordinateurs personnels compatibles PC, et devait être accompagné des logiciels GNU pour constituer un système d'exploitation. Les partisans du projet GNU promeuvent depuis le nom combiné GNU/Linux. Depuis les années 2000, le noyau Linux est utilisé sur du matériel informatique allant des téléphones portables aux super-ordinateurs, et n'est pas toujours accompagné de logiciels GNU. C'est notamment le cas d'Android, qui équipe plus de 80 % des smartphones.

Le noyau Linux a été créé en 1991 par Linus Torvalds. C'est un logiciel libre. Les distributions Linux ont été, et restent, un important vecteur de popularisation du mouvement open source.

source : wikipedia

Qu'est-ce que GNU / Linux ?

Le débat « Linux ou GNU/Linux » est une controverse divisant les partisants du logiciel libre sur le nom à donner aux systèmes d'exploitation fondés sur le système GNU et un noyau Linux. Les plus nombreux (avec le grand public) l'appellent simplement « Linux », les autres (peut-être plus proches du projet GNU) l'appellent « GNU/Linux ».

source : Wikipedia

Architecture Unix

Voici un schéma fonctionnel de base d'un système Unix :

<table>
<tr><td colspan="10">Matériels - Hardware</td></tr>
<tr><td colspan="10">Noyau - Kernel</td></tr>
<tr><td colspan="3">Schell</td><td colspan="3">Compilateur</td><td colspan="4">Commandes & Utilitaires</td></tr>
<tr><td>sh</td><td>csh</td><td>…</td><td>cpp</td><td>as</td><td>…</td><td>ps</td><td>env</td><td>more</td><td>…</td></tr>
<tr><td colspan="3">Base de Données</td><td colspan="3">Services</td><td colspan="4">Programmes & Applications</td></tr>
<tr><td>MySQL</td><td>PostgreSQL</td><td>…</td><td>apache</td><td>cups</td><td>…</td><td>ssh</td><td>ftp</td><td>mail</td><td>…</td></tr>
</table>

Les concepts principaux qui unissent toutes les versions d'Unix sont les suivants :

- **Noyau :** Le noyau est le cœur du système d'exploitation. Il interagit avec le matériel et la plupart des tâches telles que la gestion de la mémoire, la planification des tâches et la gestion des fichiers.
- **Shell :** Le shell est l'utilitaire qui traite vos demandes. Lorsque vous tapez dans une commande à votre terminal, le shell interprète la commande et appelle le programme que vous voulez. La « coquille » utilise la syntaxe standard pour toutes les commandes.
  Les C Shell, Bourne Shell et Korn Shell sont les « coquilles » les plus célèbres qui sont disponibles avec la plupart des variantes Unix.
- **Commandes et utilitaires :** Il existe différentes commandes et utilitaires que vous pouvez faire usage de vos activités quotidiennes. cp, mv, chat et grep, etc. sont quelques exemples de commandes et utilitaires. Il y a plus de 250 commandes standard ainsi que de nombreux autres fournis par des logiciels tiers. Toutes les commandes viennent avec différentes options.
- **Fichiers et répertoires :** Toutes les données d'Unix sont organisées dans des fichiers. Tous les fichiers sont ensuite organisés dans des répertoires. Ces répertoires sont encore organisés en une structure arborescente appelé le système de fichiers.

Démarrage Système

Si vous avez un ordinateur qui a le système d'exploitation Unix installé, alors vous devez simplement activer le système pour le faire vivre.

Dès que vous allumez le système, il démarre le démarrage et enfin il vous invite à vous connecter dans le système, qui est une activité pour se connecter au système et de l'utiliser pour vos activités au jour le jour.

Connexion Unix

Lorsque vous connectez à un système Unix, vous voyez habituellement une invite par exemple les éléments suivants :

```
Debian GNU/Linux 10 my-server tty1

My-server login:
```

Ouvrir une session

- Demandez à votre ID utilisateur (identification de l'utilisateur) et mot de passe prêt. Contactez votre administrateur système si vous n'avez pas encore ces.
- Tapez votre code d'utilisateur à l'invite de connexion, puis appuyez sur ENTRER. Votre code d'utilisateur est sensible à la casse, alors assurez-vous que vous tapez exactement comme votre administrateur système a demandé.
- Tapez votre mot de passe à l'invite de mot de passe, puis appuyez sur ENTRER. Votre mot de passe est également sensible à la casse.
- Si vous fournissez le code d'utilisateur et mot de passe, vous serez autorisé à entrer dans le système. Lisez les informations et les messages qui apparaît à l'écran, qui est la suivante.

```
login : jmbruneau
jmbruneau's password:
Last login: Sun Jun 18 09:43:32 2018 from 162.161.64.173
$
```

Vous recevrez une invite de commande ($ ou #) où vous tapez toutes vos commandes. Par exemple, pour vérifier le calendrier, vous devez taper la commande **cal** comme suit :

```
$ cal
    Octobre 2018
di lu ma me je ve sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

Changer le mot de passe

Tous les systèmes Unix exigent des mots de passe pour vous assurer que vos fichiers et les données restent vos propres et que le système lui-même est sécurisé contre les pirates et les craquelins. Voici les étapes pour changer votre mot de passe :

Étape 1 : Pour commencer, le mot de passe de type à l'invite de commande comme indiqué ci-dessous.

Étape 2 : Entrez votre ancien mot de passe, celui que vous utilisez actuellement.

Étape 3 : Tapez votre nouveau mot de passe. Gardez toujours votre assez complexe de mot de passe afin que personne ne puisse le deviner. Mais assurez-vous, vous vous en souvenez.

Étape 4 : Vous devez vérifier le mot de passe en le saisissant à nouveau.

```
$ passwd
Changing password for jmbruneau
(current) Unix password: ******
New Unix  password: *******
Retype new Unix password: *******
passwd: all authentication tokens updated successfully
$
```

Remarque : Nous avons ajouté astérisque (*) ici juste pour montrer l'endroit où vous devez entrer les mots de passe actuels et nouveaux sinon à votre système. Il ne vous montre pas de caractère lorsque vous tapez.

Liste des répertoires et des fichiers

Toutes les **données sous Unix** sont organisées dans des **fichiers**. Tous les fichiers sont organisés dans des répertoires. Ces **répertoires** sont organisés dans une **structure arborescente** appelé le **système de fichiers**.

Vous pouvez utiliser la commande **ls** pour lister tous les fichiers ou répertoires disponibles dans un répertoire.

Voici un exemple d'utilisation de la commande **ls** avec option **-l.**

```
# ls -l
total 74
drwxr-x---  4 root root   100 mars  14  2019 bin
drwxr-xr-x  4 root root     8 avril 25  2018 cnam
-rw-------  1 root root     1 avril 21 09:44 dead.letter
drwxr-xr-x  2 root root     2 févr. 26  2016 Desktop
drwxr-xr-x 91 root root   179 mars  14  2019 etc
drwxr-xr-x 98 root root   187 sept.  4 23:06 etc.bck
drwxr-xr-x  3 root root     3 sept. 13  2018 ispace-syscom
drwxr-xr-x  5 root root     5 sept.  1  2018 macosx
drwxr-xr-x  3 root root     3 janv. 26  2018 maj
drwxr-xr-x  4 root root     4 janv.  9  2019 migrations
drwxr-x---  8 root root    11 juil. 12 15:26 mysql
drwxr-xr-x  3 root root     3 déc.   5  2017 Netspace
-rw-------  1 root root 30539 déc.  21  2018 nohup.out
drwxr-x---  5 root root     7 janv. 23  2019 pgsql
drwxr-xr-x  2 root root     3 févr. 26  2019 PrestaShop
drwxr-xr-x  2 root root    10 juil. 12 15:06 private
-rw-r--r--  1 root root   270 mars  26  2018 README.git
-rw-r--r--  1 root root   367 févr. 22  2019 README.goaccess
```

Les entrées commençant par **"d"** représentent des répertoires.

Par exemple, **bin**, **cnam**, **Desktop, ... (en gras)** sont des répertoires et les autres sont des fichiers.

Qui es-tu ?

Pendant que vous êtes connecté au système, vous pourriez vouloir savoir « Qui suis-je ? »

La meilleure façon de savoir « qui vous êtes » est d'entrer dans la commande **whoami** :

```
$ whoami
jmbruneau
$
```

Essayez-le sur votre système.

Cette commande indique le nom du compte associé à la connexion en cours. Vous pouvez essayer qui suis-je prescris aussi bien pour obtenir des informations sur vous-même.

Qui est connecté ?

Parfois, vous pourriez être intéressé de savoir qui est connecté à l'ordinateur en même temps.

Il y a trois commandes disponibles pour obtenir cette information : **users**, **who**, et **w**.

```
$ users
jmbruneau root manager

$ who
jmbruneau console  Oct 16 20:33
jmbruneau ttys000  Oct 16 20:33
jmbruneau ttys001  Oct 16 20:33
jmbruneau ttys002  Oct 16 20:33
jmbruneau ttys003  Oct 16 20:33
jmbruneau ttys004  Oct 16 20:33

$ w
18:10  up 9 days, 23:59, 22 users, load averages: 3,60 2,88 2,65
USER     TTY      FROM              LOGIN@  IDLE WHAT
jmbruneau console  -                22aoû2 10days -
jmbruneau s000     -                22aoû2  3:32 -sh
jmbruneau s001     -                22aoû2  3:32 -sh
jmbruneau s002     -                22aoû2 2days -sh
jmbruneau s003     -                22aoû2 2days -sh
jmbruneau s004     -                22aoû2 2days -sh
jmbruneau s005     -                Lun11    3:24 -sh
```

Déconnecter

Lorsque vous avez terminé votre session, vous devez vous déconnecter du système et faire en sorte que personne d'autre n'accède à vos fichiers.

Pour vous déconnecter

Il suffit de taper la commande **logout** ou **exit** de fermeture de session et le système va tout nettoyer et couper la connexion.

Arrêt du système

La façon la plus cohérente pour arrêter un système Unix correctement via la ligne de commande est d'utiliser l'une des commandes suivantes :

Commander	La description
halt	Apporte le système immédiatement
init 0	Pouvoirs le système hors tension à l'aide de scripts prédéfinis pour synchroniser et nettoyer le système avant d'arrêter
init 6	Redémarrages du système en le fermant complètement, puis le redémarrer

Commander	La description
poweroff	Arrête le système de mise hors tension
reboot	Le système redémarrages
shutdown	Arrête le système

Vous devez généralement être le **super-utilisateur** ou **root** (le compte privilégié sur un système Unix) pour arrêter le système.

Cependant, sur certaines Unix autonomes ou personnel, un administrateur et parfois les utilisateurs réguliers peuvent le faire **(attention : danger).**

3. Unix - Gestion des fichiers

Dans ce chapitre, nous aborderons en détail la gestion des fichiers sous Unix. Toutes les données sous Unix sont organisées dans des **fichiers**. Tous les fichiers sont organisés dans des **répertoires**. Ces répertoires sont organisés dans une structure arborescente appelé le **système de fichiers**.

Lorsque vous travaillez avec Unix, d'une façon ou d'une autre, vous passez la plupart de votre temps de travail avec les fichiers. Ce tutoriel va vous aider à comprendre comment créer et supprimer des fichiers, les copier et les renommer, créer des liens vers eux, etc.

Dans Unix, il existe trois types de fichiers :

- **fichiers ordinaires** : Un fichier ordinaire est un fichier sur le système qui contient des données, du texte ou des instructions du programme. Dans ce tutoriel, vous regardez travailler avec des fichiers ordinaires.
- **répertoires :** Les répertoires stockent les fichiers spéciaux et ordinaires. Pour les utilisateurs familiers avec Windows ou Mac OS, Unix répertoires sont équivalents aux dossiers.
- **fichiers spéciaux** : Certains fichiers spéciaux permettent d'accéder au matériel tels que les disques durs, les lecteurs de CD-ROM, des modems et des adaptateurs Ethernet. D'autres fichiers spéciaux sont similaires à des alias ou des raccourcis et vous permettent d'accéder à un fichier unique en utilisant des noms différents.

Liste des fichiers

Pour lister les fichiers et répertoires stockés dans le répertoire courant, utilisez la commande suivante :

```
$ ls
```

Voici un exemple de sortie de la commande ci-dessus avec :

```
$ ls -l
total 1236
drwxr-xr-x 3 root root      4096 déc.  11  2017 acpi
-rw-r--r-- 1 root root      2981 août  14  2016 adduser.conf
-rw-r--r-- 1 root root        45 août  17  2016 adjtime
-rw-r--r-- 1 root root       197 août  17  2016 aliases
-rw-r--r-- 1 root root     12288 mai   26 09:59 aliases.db
drwxr-xr-x 2 root root     12288 oct.  16 21:52 alternatives
-rw-r--r-- 1 root root      4185 oct.  15  2014 analog.cfg
...
drwxr-xr-x 2 root root      4096 déc.  11  2017 sgml
-rw-r----- 1 root shadow    1316 févr. 27  2018 shadow
-rw-r--r-- 1 root root        97 déc.  11  2017 shells
drwxr-xr-x 2 root root      4096 déc.  11  2017 skel
-rw-r--r-- 1 root root      7221 août  22  2016 smartd.conf
drwxr-xr-x 4 root root      4096 août  17  2016 smartmontools
drwxr-xr-x 2 root root      4096 oct.  16 21:54 snmp
drwxr-xr-x 2 root root      4096 oct.  16 21:55 ssh
```

La commande ls prend en charge l'option **-l** qui vous aidera à obtenir plus d'informations sur les fichiers de la liste.

Voici les informations sur toutes les colonnes de cette liste :

- **première colonne**: Représente le type de fichier et l'autorisation donnée au dossier. Ci-dessous la description de tous les types de fichiers.
- **deuxième colonne**: Représente le nombre de blocs de mémoire prises par le fichier ou le répertoire.
- **troisième colonne**: Représente le propriétaire du fichier. Ceci est l'utilisateur Unix qui a créé ce fichier.
- **quatrième colonne**: Représente le groupe du propriétaire. Chaque utilisateur Unix aura un groupe associé.
- **cinquième colonne**: Représente la taille du fichier en octets.
- **sixième colonne**: Représente la date et le moment où ce fichier a été créé ou modifié pour la dernière fois.
- **septième colonne**: Représente le fichier ou le nom du répertoire.

Dans la liste exemple de **ls**, chaque ligne de fichier commence par annonce, **-** ou **l**. Ces caractères indiquent le type du fichier qui est répertorié.

Préfixe	Description
-	fichier régulier, comme un fichier texte ASCII, binaire exécutable ou un lien dur
b	Bloquer fichier spécial. Bloquer fichier de périphérique d'entrée / sortie, tel qu'un disque dur physique
c	Spécial caractère. Raw entrée / fichier de dispositif de sortie tel qu'un disque dur physique
d	fichier répertoire qui contient une liste d'autres fichiers et répertoires
l	fichier lien symbolique. Liens sur un fichier régulier
p	tube nommé. Un mécanisme de communication interprocessus
s	Socket utilisé pour la communication interprocessus

Méta-caractères

Les méta-caractères ont une signification particulière dans Unix. Par exemple, ***** et **?** sont méta-caractères.

Nous utilisons ***** pour **0** ou plusieurs caractères,

un point d'interrogation **?** correspondant à un seul caractère.

Par exemple :

```
$ ls ch*.doc
```

Affiche tous les fichiers, dont les noms commencent par **ch** et se terminent par **.doc** :

Ici, * fonctionne comme un méta-caractère qui correspond à un caractère.

Si vous souhaitez afficher tous les fichiers se terminant par juste **.doc**, vous pouvez utiliser la commande suivante :

```
$ ls *.doc
```

Fichiers cachés

Un fichier invisible est un fichier dont le premier caractère est un point « **.** ». Les programmes Unix (y compris le shell) utilisent la plupart de ces fichiers pour stocker des informations de configuration.

Quelques exemples communs des fichiers cachés incluent les fichiers :

- **.profil** : Le Bourne shell (sh) script d'initialisation
- **.kshrc** : Le shell Korn (ksh) script d'initialisation
- **.cshrc** : Le C shell (csh) script d'initialisation
- **.rhosts** : Le fichier de configuration shell distant

Pour lister les fichiers invisibles, spécifiez l'option **-a** à **ls**

```
$ ls -a
.                                       master.passwd
..                                      master.passwd~orig
AFP.conf                                moduli~previous
DIR_COLORS                              nanorc
DIR_COLORS.xterm                        networks
DIR_COLORS.xterm-color                  networks~orig
afpovertcp.cfg                          newsyslog.conf
afpovertcp.cfg~orig                     newsyslog.d
aliases                                 nfs.conf
aliases.db                              nfs.conf~orig
apache2                                 notify.conf
apache2.mbp15                           ntp-restrict.conf
asl                                     ntp.conf
asl.conf                                ntp.conf~orig
authorization.deprecated                ntp_opendirectory.conf
auto_home                               openldap
```

- **point unique (.)** : Ceci représente le répertoire courant.
- **double point (..)** : Ceci représente le répertoire parent.

Création de fichiers

Vous pouvez utiliser l'éditeur **vi** ou **nano** pour créer des fichiers ordinaires sur tout système Unix. Il vous suffit de taper la commande suivante :

```
$ vi nom_fichier
```

La commande ci-dessus ouvre un fichier avec le nom de fichier donné. Maintenant, appuyez sur la touche i pour entrer en mode d'édition. Une fois que vous êtes en mode d'édition, vous pouvez commencer à écrire votre contenu dans le fichier comme dans le programme suivant :

Une fois que vous avez terminé avec le programme, procédez comme suit :

```
Ce fichier est unix .... Je l'ai créé pour la première fois .....
Je vais enregistrer ce contenu dans ce fichier.
```

- Appuyez sur la touche **Esc** pour sortir du mode d'édition.
- Appuyez sur **ZZ** pour écrire et sortir complètement du fichier.

Vous avez maintenant un fichier créé avec le nom de fichier « nom_fichier » dans le répertoire courant.

Modification de fichiers

Vous pouvez modifier un fichier existant à l'aide de l'éditeur vi. Nous discuterons en bref comment ouvrir un fichier existant :

```
$ vi nom_fichier
```

Une fois que le fichier est ouvert, vous pouvez entrer dans le mode d'édition en appuyant sur la touche **i** et vous pouvez ainsi éditer le fichier.

Si vous voulez vous déplacer dans un fichier, vous devez d'abord sortir du mode d'édition en appuyant sur la touche **Esc**.

Après cela, vous pouvez utiliser les touches suivantes pour vous déplacer dans un fichier :

- **l** pour déplacer vers le côté droit.
- **h** pour déplacer sur le côté gauche.
- **k** pour déplacer la tête dans le fichier.
- **j** pour déplacer la baisse dans le fichier.

Donc, en utilisant les touches ci-dessus, vous pouvez placer votre curseur où vous souhaitez modifier. Une fois que vous êtes positionné, vous pouvez utiliser la touche **i** pour venir en mode d'édition. Une fois que vous avez terminé avec l'édition dans votre fichier, appuyez sur **Esc** et puis sur **ZZ** pour sortir complètement du fichier.

Afficher le contenu d'un fichier

Vous pouvez utiliser la commande de « **cat** » pour voir le contenu d'un fichier (sans pagination). Voici un exemple simple pour voir le contenu du fichier créé ci-dessus :

```
$ cat nom_fichier
Ce fichier est unix .... Je l'ai créé pour la première fois .....
Je vais enregistrer ce contenu dans ce fichier.
$
```

Vous pouvez afficher les numéros de ligne en utilisant l'option **-b** avec la commande cat comme suit :

```
$ cat -b nom_fichier
1     Ce fichier est unix .... Je l'ai créé pour la première fois .....
2     Je vais enregistrer ce contenu dans ce fichier.
$
```

Compter les mots dans un fichier

Vous pouvez utiliser la commande **wc** pour obtenir un décompte du nombre total de lignes, des mots et des caractères contenus dans un fichier. Voici un exemple simple pour voir les informations sur le fichier créé ci-dessus :

```
$ wc /etc/passwd
      86     243    5253 /etc/passwd
```

Voici le détail de toutes les quatre colonnes :

- **première colonne**: Représente le nombre total de lignes dans le fichier.
- **deuxième colonne**: Représente le nombre total de mots dans le fichier.
- **troisième colonne**: Représente le nombre total d'octets dans le fichier.
  Ceci est la taille réelle du fichier.
- **La quatrième colonne**: Représente le nom du fichier.

Vous pouvez donner plusieurs fichiers et obtenir des informations sur ces fichiers à la fois. Voici syntaxe simple :

```
$ wc fichier1 fichier2 fichier3 …
```

Copie de fichiers

Pour faire une copie d'un fichier, utilisez la commande **cp**. La syntaxe de base de la commande est :

```
$ cp source_file destination_file
$
```

Voici l'exemple pour créer une copie du fichier file_name existant.

Vous allez maintenant trouver un fichier « file_name_copy » dans votre répertoire courant.

```
$ cp file_name file_name_copy
$
```

Ce fichier sera exactement le même que celui d'origine.

Renommage de fichiers

Pour modifier le nom d'un fichier, utilisez la commande **mv**. Voici la syntaxe de base :

```
$ mv old_file new_file
$
```

Le programme suivant renommer le nom de fichier existant en new_file_name.

La commande **mv** déplacera le fichier existant complètement dans le nouveau fichier. Dans ce cas, vous ne trouverez que newfile dans votre

```
$ mv file_name new_file_name
$
```

répertoire courant.

Suppression de fichiers

Pour supprimer un fichier existant, utilisez la commande **rm**. Voici la syntaxe de base :

```
$ rm file_name
```

Mise en garde : Un fichier peut contenir des informations utiles. Il est toujours recommandé d'être prudent lorsque vous utilisez cette commande Supprimer. Il est préférable d'utiliser l'option **-i** avec commande **rm**.

Voici l'exemple qui montre comment supprimer complètement le nom de fichier existant.

```
$ rm file_name
$
```

```
$ rm file_name1 file_name2 file_name3
$
```

Vous pouvez supprimer plusieurs fichiers à la fois avec la commande donnée ci-dessous :

Les flux Unix standards:

Dans des circonstances normales, chaque programme Unix comporte trois flux (i.e. fichiers) ouverts lorsqu'il démarre :

- **stdin :** est appelé l'entrée standard,
  le descripteur de fichier associé est **0**.
  Elle est également appellée **STDIN**.
  Un programme Unix va lire sur l'entrée par défaut **STDIN**.
- **stdout :** est appelé la sortie standard,
  le descripteur de fichier associé **est 1**.
  Elle est également appellée STDOUT.
  Un programme Unix va écrire dans la sortie par défaut **STDOUT**.
- **stderr** : est appelé l'erreur-type,
  le descripteur de fichier associé est **2**.
  Elle est également appellée **STDERR**.
  Un programme Unix va écrire tous les messages d'erreur dans **STDERR**.

4. Unix - Gestion des Répertoires

Ce chapitre porte sur la gestion des répertoires sous Unix.

Un répertoire est un fichier le travail qui est en charge de stocker les noms de fichiers et leurs informations connexes. Tous les fichiers, qu'ils soient de type ordinaire, spécial ou répertoire, sont contenus dans des répertoires.

Unix utilise une structure hiérarchique pour l'organisation des fichiers et des répertoires. Cette structure est souvent appelée arborescence. L'arbre a un seul nœud racine, le caractère slash (i.e. **/**), et tous les autres répertoires sont situés en-dessous.

Le répertoire d'accueil (home directory)

Le répertoire dans lequel vous vous trouvez lorsque vous vous loger est appelé votre répertoire personnel.

Vous allez faire une grande partie de votre travail dans votre répertoire personnel et les sous-répertoires que vous allez créer pour organiser vos fichiers.

Vous pouvez aller dans votre répertoire en utilisant à tout moment la commande suivante :

```
$ cd ~
$
```

Ici **~** indique votre répertoire personnel. Supposons que vous devez aller dans le répertoire d'un autre utilisateur, utilisez la commande suivante :

Pour aller dans votre dernier répertoire, vous pouvez utiliser la commande suivante :

```
$ cd ~user_name
$
```

```
$ cd -
$
```

Chemin (path) absolu ou relatif

Les répertoires sont organisés selon une hiérarchie à racine (/) dans la partie supérieure. La position d'un fichier dans la hiérarchie est décrite par son chemin.

Les éléments d'un chemin d'accès sont séparés par un /. Un chemin est absolu, si elle est décrite par rapport à la racine, donc absolu commencent les chemins toujours avec un /.

Voici quelques exemples de noms de fichiers absolus.

```
/etc/passwd
/users/jmbruneau/notes
/dev/rdsk0
```

Un chemin peut aussi être également relatif à votre répertoire de travail courant. Les chemins relatifs ne commencent jamais avec /.

Par rapport au répertoire de l'utilisateur jmbruneau, certains noms de fichiers pourrait ressembler à ceci :

```
Netspace/Data
Perso/UCA/IUT
```

Pour déterminer où vous êtes dans la hiérarchie du système de fichiers à tout moment, entrez la commande passwd pour imprimer le répertoire de travail en cours :

```
$ pwd
/Volumes/Users/jmbruneau/Perso/UCA
$
```

Lister le contenu d'un répertoire

Pour lister les fichiers dans un répertoire, vous pouvez utiliser la syntaxe suivante :

```
$ ls dir_name
```

Voici l'exemple de lister tous les fichiers contenus dans le répertoire **/usr/local** :

```
# ls /usr/local/
bin  etc  games  include  lib  man  sbin  share  src
#
```

Création des répertoires

Nous allons maintenant comprendre comment créer des répertoires. Les répertoires sont créés par la commande suivante :

```
$ mkdir dir_name
```

Ici, le répertoire est le chemin absolu ou relatif du répertoire que vous voulez créer. Par exemple, la commande

```
$ mkdir mydir
$
```

crée le répertoire « **mydir** » dans le répertoire courant. Voici un autre exemple :

```
$ mkdir /tmp/test-dir
$
```

Cette commande crée le répertoire **test-dir** dans le répertoire **/tmp**. La commande **mkdir** ne produit aucune sortie si elle crée avec succès le répertoire demandé.

Si vous donnez plus d'un répertoire sur la ligne de commande, **mkdir** crée chacun des répertoires. Par exemple, :

Crée les répertoires docs et pub dans le répertoire en cours.

```
$ mkdir pub docs
$
```

Création de répertoires parents

Nous allons maintenant comprendre comment créer les répertoires parents. Parfois, quand vous voulez créer un répertoire, son répertoire parent ou répertoires peuvent ne pas exister. Dans ce cas, mkdir émet un message d'erreur :

Dans ce cas, vous pouvez spécifier l'option **-p** à la commande **mkdir** qui crée tous les répertoires nécessaires pour vous. Par exemple :

```
$ mkdir /tmp/jmbruneau/test
mkdir: cannot create directory '/tmp/jmbruneau/test': No such file or directory
$
```

La commande ci-dessus crée tous les répertoires parents nécessaires.

```
$ mkdir -p /tmp/jmbruneau/test
$
```

Suppression d'un dossier

Les répertoires peuvent être supprimés à l'aide de la commande **rmdir** comme suit :

```
$ rmdir dir_name
$
```

Remarque : Pour supprimer un répertoire, assurez-vous qu'il est vide, ce qui signifie qu'il ne devrait pas être un fichier ou un sous-répertoire dans ce répertoire.

Vous pouvez supprimer plusieurs répertoires à la fois comme suit :

```
$ rmdir dir_name1 dir_name2 dir_name3
$
```

La commande ci-dessus supprime le dir_name1 des répertoires, dir_name2 et dir_name3, si elles sont vides. La commande rmdir ne produit aucune sortie si elle est couronnée de succès.

Changement de répertoire

Vous pouvez utiliser la commande cd pour faire plus que simplement passer à un répertoire. Vous pouvez l'utiliser pour modifier dans un répertoire en spécifiant un chemin absolu ou relatif valide. La syntaxe est donnée ci-dessous :

```
$ cd dir_name
$
```

Ici, dir_name est le nom du répertoire que vous voulez changer. Par exemple, la commande :

Les changements dans le répertoire **/usr/local/bin**. À partir de ce répertoire, vous pouvez vous placer dans le répertoire **/home/jmbruneau** en

```
$ cd /usr/local/bin
$
```

utilisant le chemin relatif suivant :

```
$ cd ../../../home/jmbruneau
$
```

Changement de Nom d'un Répertoire

La commande **mv** (move) peut également être utilisée pour renommer un répertoire. La syntaxe est la suivante :

```
$ mv olddir newdir
$
```

Vous pouvez donc renommer un répertoire **my_dir** en **your_dir** comme suit :

```
$ mv my_dir your_dir
$
```

Les répertoires « . » et « .. »

Le nom de fichier « **.** » représente le **répertoire de travail courant**; et le nom du fichier « **..** » représente le répertoire du niveau au-dessus du répertoire courant, souvent appelé le **répertoire parent**.

Si nous entrons la commande pour afficher une liste des répertoires et fichiers du répertoire de travail actuels et utilisons l'option **-a** pour lister tous

les fichiers et l'option **-l** pour donner la longue liste, nous obtenons le résultat suivant :

```
# ls -lsa
total 1980
 12 drwxr-xr-x 165 root     root      12288 Oct 17 03:05 .
  4 drwxr-xr-x  26 root     root       4096 Aug  7 16:27 ..
  4 drwxr-xr-x   3 root     root       4096 Jul  4  2016 acpi
  4 -rw-r--r--   1 root     root       2981 Jul  4  2016 adduser.conf
  4 -rw-r--r--   1 root     root         45 Dec 26  2016 adjtime
  0 lrwxrwxrwx   1 root     root         15 Dec 16  2016 aliases -> postfix/aliases
 12 -rw-r--r--   1 root     root      12288 Jun 16  2017 aliases.db
 20 drwxr-xr-x   2 root     root      20480 Jul  9 21:30 alternatives
```

5. Unix – Autorisation & Mode d'Accès aux Fichiers

Dans ce chapitre, nous verrrons en détail des modes d'autorisation des fichiers et l'accès à Unix.

La propriété des fichiers est un élément important d'Unix qui fournit une méthode sécurisée pour stocker des fichiers. Chaque fichier sous Unix possède les attributs suivants :

- **autorisations de propriétaire** : Les autorisations du propriétaire déterminent quelles actions le propriétaire du fichier peut effectuer sur le fichier.
- **autorisations de groupe :** Les autorisations déterminent les actions du groupe d'un utilisateur, qui est un membre du groupe qui appartient un fichier, peut effectuer sur le fichier.
- **Autres permissions (des autres)** : Les autorisations pour d'autres indiquent que l'action de tous les autres utilisateurs peuvent effectuer sur le fichier.

Les indicateurs d'autorisation

Tout en utilisant la commande **ls -l**, il affiche diverses informations relatives aux autorisations des fichiers, dossiers et autres comme :

```
$ ls -l /home/jmbruneau
-rwxr-xr--  1 jmbruneau users 1024  nov 2 00h10 mon_fichier
drwxr-xr--  1 jmbruneau users 1024  nov 2 00h10 mon_repertoire
```

Ici, la première colonne représente différents modes d'accès, à savoir l'autorisation associée à un fichier ou un répertoire.

Les autorisations sont divisées en groupes de trois, et chaque position dans le groupe représente une autorisation spécifique, dans cet ordre :

lecture (r), écriture (w), exécution (x)

- Les trois premiers caractères (2-4) représentent les autorisations pour le propriétaire du fichier.
  Par exemple, pour **-rwxr-xr--**, le propriétaire a la permission de lire (r), d'écrire (w) et d'exécuter (x)
- Le deuxième groupe de trois caractères (5-7) comprend les autorisations pour le groupe auquel appartient le fichier.
  Par exemple, pour **-rwxr-xr--**, le groupe a la permission de lire (r) et d'exécuter (x), mais aucune autorisation d'écriture.
- Le dernier groupe de trois caractères (8-10) représente les autorisations pour tout le monde.
  Par exemple, pour **-rwxr-xr--** le fichier est lisible (r) par tout le monde.

Modes d'accès aux fichiers

Les permissions d'un fichier sont la première ligne de défense dans la sécurité d'un système Unix. Les autorisations de base des Unix sont des autorisations de lecture, écriture et exécution :

Lire : r

Accorde la capacité de lire, par exemple, afficher le contenu du fichier.

Écrire : w

Accorde la possibilité de modifier ou supprimer le contenu du fichier.

Exécuter : x

Utilisateur avec permissions d'exécution pour exécuter un fichier en tant que programme.

Modes d'accès aux répertoires

Les modes d'accès aux répertoires sont organisés de la même manière que pour tout autre fichier. Quelques différences doivent être cependant mentionnées :

Lire : r

L'accès à un répertoire signifie que l'utilisateur peut lire le contenu. L'utilisateur peut accéder **aux noms des fichiers** à l'intérieur du répertoire.

Écrire : w

L'accès signifie que l'utilisateur peut ajouter ou supprimer des fichiers du répertoire.

Exécuter : x

L'exécution d'un répertoire n'a pas vraiment de sens, c'est en fait une autorisation de pouvoir traverser.

Un utilisateur doit avoir l'accès d'exécution au répertoire /bin afin de pouvoir exécuter la commande **ls** ou **cd**

Modification des autorisations

Pour modifier les autorisations des fichiers ou des répertoires, vous utiliserez la commande **chmod** (changement de mode). Il y a deux façons d'utiliser **chmod** :

- le mode **symbolique**
- le mode **absolu**.

chmod en mode symbolique

La meilleure façon pour un débutant pour modifier les autorisations de fichier ou de répertoire est d'utiliser le mode symbolique. Avec des autorisations symboliques, vous pouvez ajouter, supprimer ou spécifier l'ensemble d'autorisation que vous souhaitez à l'aide des opérateurs dans le tableau suivant :

Opérateur chmod	La description
+	Ajoute l'autorisation désignée (s) dans un fichier ou un répertoire.
-	Supprime l'autorisation désignée (s) à partir d'un fichier ou un répertoire.
=	Définit l'autorisation désigné (s).

Voici un exemple d'utilisation testfile. Exécution de **ls -l** sur la testfile montre que les autorisations du fichier sont les suivants :

```
$ ls -l test_file
-rw-r--r-- 1 jmbruneau staff 0 oct 17 10:52 test_file
```

Ensuite, chaque exemple de commande chmod du tableau précédent est exécuté sur le fcihier test_file, suivi par **ls -l**, afin que vous puissiez voir les changements d'autorisation.

Voici comment vous pouvez combiner ces commandes sur une seule ligne :

```
$ chmod o+wx test_file
$ ls -lsa test_file
0 -rw-r--rwx 1 jmbruneau staff 0 oct 17 10:59 test_file
$ chmod u-x test_file
$ ls -lsa test_file
0 -rw-r--rwx 1 jmbruneau staff 0 oct 17 10:59 test_file
$ chmod g=rx test_file
$ ls -lsa test_file
0 -rw-r-xrwx 1 jmbruneau staff 0 oct 17 10:59 test_file
```

chmod avec des autorisations « absolues »

```
$ chmod o +wx,u-x,g=rx test_file
$ ls -l test_file
-rw-r-xrwx 1 jmbruneau staff 0 oct 17 10:52 test_file
```

« La deuxième façon de modifier les autorisations avec la commande **chmod** consiste à utiliser un certain nombre de spécifier chaque ensemble d'autorisations pour le fichier.

Chaque autorisation est attribué une valeur, comme le montre le tableau ci-dessous, et le total de chaque ensemble d'autorisations fournit un certain nombre pour cet ensemble.

Nombre	Représentation octal des permissions	Ref
0	Aucune autorisation	---
1	l'autorisation d'exécution	--x
2	le droit d'écriture	-w-
3	Exécuter et d'écriture: 1 (exécution) + 2 (écriture) = 3	-wx
4	autorisation de lecture	r--
5	Lecture et d'exécution: 4 (lecture) + 1 (exécution) = 5	r-x
6	Lecture et d'écriture: 4 (lecture) + 2 (écriture) = 6	rw-
7	Toutes les permissions: 4 (lecture) + 2 (écriture) + 1 (d'exécution) = 7	rwx

Voici un exemple en utilisant le **test_file**. Exécution de **ls -l** sur la testfile montre que les autorisations du fichier sont les suivants :

```
$ ls -l test_file
0 -rwxrwxr-- 1 jmbruneau staff 0 oct 17 11:04 test_file
```

Ensuite, chaque exemple de commande **chmod** du tableau précédent est exécutée sur le testfile, suivi par **ls -l**, afin que vous puissiez visualiser les changements d'autorisation :

```
$ chmod 755 test_file
$ ls -lsa test_file
0 -rwxr-xr-x 1 jmbruneau staff 0 oct 17 11:04 test_file
$ chmod 743 test_file; ls -lsa test_file
0 -rwxr---wx 1 jmbruneau staff 0 oct 17 11:04 test_file
$ chmod 043 test_file; ls -lsa test_file
0 ----r---wx 1 jmbruneau staff 0 oct 17 11:04 test_file
```

Modification des propriétaires et des groupes

Tout en créant un compte sur Unix, il attribue un ID propriétaire (**UID**) et un ID de groupe (**GID**) à chaque utilisateur. Toutes les autorisations mentionnées ci-dessus sont également attribuées selon le propriétaire et les groupes.

Deux commandes sont disponibles pour changer le propriétaire et le groupe de fichiers :

- **chown**- La commande chown signifie « changement propriétaire » et est utilisé pour changer le propriétaire d'un fichier.
- **chgrp :** La commande chgrp signifie « groupe de changement » et est utilisé pour changer le groupe d'un fichier.

Changer de propriétaire

La commande **chown** change la propriété d'un fichier. La syntaxe de base est la suivante :

```
$ chown utilisateur "liste de fichier(s)"
```

La valeur de l'utilisateur peut être soit le nom d'un utilisateur sur le système ou l'identificateur d'utilisateur (**UID**) d'un utilisateur sur le système.

L'exemple suivant vous aidera à comprendre le concept :

```
$ chown jmbruneau test_file
$
```

Change le propriétaire du fichier donné à l'utilisateur **jmbruneau**.

Remarque : le super-utilisateur, la racine, a la capacité illimitée de changer la propriété d'un fichier, mais les utilisateurs normaux peuvent changer la propriété uniquement pour les fichiers dont ils sont propriétaires.

Changement de groupe de propriété

La commande **chgrp** modifie le groupe d'un fichier. La syntaxe de base est la suivante :

```
$ chgrp groupe "liste de fichier(s)"
```

La valeur du groupe peut être le nom d'un groupe sur le système ou l'ID de groupe (**GID**)

d'un groupe sur le système.

L'exemple suivant vous aide à comprendre le concept :

```
$ chgrp test_file staff
$
```

Change le groupe du fichier donné vers le groupe « staff ».

Permissions SUID et SGID

Souvent, quand une commande est exécutée, il devra être exécuté avec des privilèges spéciaux pour accomplir sa tâche.

À titre d'exemple, lorsque vous modifiez votre mot de passe avec la commande **passwd**, votre nouveau mot de passe est enregistré dans le fichier **/etc/shadow**.

En tant qu'utilisateur régulier, vous n'avez pas lire ou écrire un accès à ce fichier pour des raisons de sécurité, mais lorsque vous modifiez votre mot de passe, vous devez avoir l'autorisation d'écriture à ce fichier. Cela signifie que le programme **passwd** doit vous donner des autorisations supplémentaires afin que vous puissiez **écrire** dans le fichier **/etc/shadow**.

Des autorisations supplémentaires sont données aux programmes via un mécanisme connu sous le nom Set ID utilisateur (suid) et les bits Set ID de groupe (SGID).

Lorsque vous exécutez un programme qui a le bit SUID est activé, vous héritez des autorisations du propriétaire de ce programme. Les programmes qui n'ont pas le bit SUID sont exécutés avec les autorisations de l'utilisateur qui a démarré le programme.

Tel est également le cas avec SGID. Normalement, les programmes sont exécutés avec les autorisations de votre groupe, mais pour ceux qui on le bit SGID votre groupe sera changé juste pour ce programme par celui du propriétaire du groupe du programme.

Les bits SUID et SGID apparaîssent avec la lettre « s » si l'autorisation est disponible. Le bit « s » SUID sont situés dans les bits d'autorisation relatifs aux droits d'exécution des propriétaires.

Par exemple, la commande :

```
# ls -lsa /usr/bin/passwd
56 -rwsr-xr-x 1 root root 54192 May 17  2017 /usr/bin/passwd
#
```

Indique que le bit **SUID** est réglé et que la commande appartient à **root**.

Une lettre majuscule **S** au lieu d'une minuscule **s** indique que le bit d'exécution n'est pas actif.

Si le « **sticky bit** » est activé sur un répertoire, les fichiers ne peuvent être supprimés que si vous êtes l'un des utilisateurs suivants :

- Le propriétaire du répertoire « sticky bit »
- Le propriétaire du fichier en cours de suppression
- Le super-utilisateur, **root**

Pour définir les SUID et les bits SGID pour un répertoire essayer la commande suivante :

```
# chmod ug+s dir_name
# ls -l
drwsr-sr-x 2 root root  4096 19 juin 06h45 dir_name
#
```

6. Unix – Environnement

L'un des concepts important d'Unix est l'environnement qui est défini par des variables d'environnement. Certaines sont fixées par le système, d'autres par vous, d'autres encore par le shell, ou tout autre programme qui charge un autre programme.

Une variable est une chaîne de caractères à laquelle nous attribuons une valeur. La valeur attribuée pourrait être un nombre, texte, nom de fichier, un périphérique ou tout autre type de données.

Par exemple, nous avons d'abord mis une variable test, puis nous avons accès à sa valeur à l'aide de la commande **echo** :

Notez que les variables d'environnement sont définies sans utiliser le signe **$**, mais que pour leurs accès nous utilisons le signe **$** comme préfixe.

```
$ TEST="Unix Programmation"
$ echo $TEST
Unix Programmation
$
```

Ces variables conservent leurs valeurs jusqu'à ce que nous sortions du **shell**.

Lorsque vous vous connectez au système, le shell subit une phase d'initialisation pour mettre en place l'environnement. C'est généralement un processus en deux étapes qui implique la lecture des fichiers suivants :

- **/etc/profile**
- **.profile**

Le processus est le suivant :

- Les **shell** vérifie pour voir si le fichier **/etc/profile** existe.
- S'il existe, le shell le lit. Dans le cas contraire, ce fichier est ignoré. Aucun message d'erreur est affiché.
- Le shell vérifie également si le fichier « **.profile** » existe dans votre répertoire personnel. Votre répertoire personnel est celui où vous vous trouvez après une connection.
- S'il existe, le shell le lit. Aucun message d'erreur n'est affiché.

Dès que ces deux fichiers ont été lus, le shell affiche un « **invite de commande** » :

```
$
```

Ceci est l'invite de commande où vous pouvez entrer des commandes afin qu'elles soient exécutées.

Remarque : Le processus d'initialisation du **shell** détaillée ici s'applique à toutes les **shell** de type « **Bourne** ». Toutefois, certains fichiers supplémentaires sont utilisés par **bash** (Bourne Again) et **ksh**.

Le fichier « .profile »

Le fichier **/etc/profile** est maintenu par l'administrateur système de votre machine Unix et contient des informations d'initialisation du shell requise par tous les utilisateurs d'un système.

Le fichier « **.profile** » est sous votre contrôle. Vous pouvez ajouter autant d'informations de personnalisation du shell que vous voulez dans ce fichier. Le minimum d'informations que vous devez configurer comprend :

- Le type de terminal que vous utilisez
- Une liste des répertoires dans lesquels pour localiser les commandes
- Une liste de variables affectant l'apparence de votre terminal

Vous pouvez vérifier votre « .profile » dans votre répertoire personnel.

Ouvrez-le à l'aide de l'éditeur vi et vérifier toutes les variables définies pour votre environnement.

Réglage du type de terminal

En règle générale, le type de terminal que vous utilisez est configuré automatiquement soit par la connexion ou les programmes **getty**. Parfois, le processus de configuration automatique de votre terminal peut être incorrecte.

Si votre terminal est mal réglé, la sortie des commandes peut sembler étrange, ou vous pourriez ne pas être en mesure d'interagir avec le shell correctement.

Pour vous assurer que ce n'est pas le cas, la plupart des utilisateurs définisent leur terminal au plus petit dénominateur commun de la manière suivante :

```
$ TERM=”vt100”
$
```

Définition du chemin (PATH)

Lorsque vous tapez une commande sur l'invite de commande, le shell doit localiser la commande avant qu'elle ne puisse être exécutée.

La variable **PATH** spécifie les endroits où le shell doit chercher les commandes.

```
$ PATH=”/bin:/usr/bin”
$
```

Habituellement, la variable Path est définie comme suit :

Ici, chacune des entrées individuelles est séparées par le caractère deux-points (**:**) sont des répertoires. Si vous demandez au shell d'exécuter une commande et il ne peut pas la trouver dans l'un des répertoires donnés dans la variable **PATH**, un message de ce type apparaît :

```
$ bonjour
bonjour: command not found
$
```

Variables PS1 et PS2

Les caractères que le shell affiche que l'invite de commande sont stockés dans la PS1 variables. Vous pouvez modifier cette variable pour être tout ce que vous voulez. Dès que vous le changiez, il sera utilisé par le shell à partir de ce moment-là.

Par exemple, si vous avez émis la commande :

```
$ PS1='=>'
=>
=>
=>
```

Votre « prompt » deviendra =>. Pour définir la valeur de PS1 pour qu'il montre le répertoire de travail, exécutez la commande :

```
=> PS1="\u@\h\w$\"
jmbruneau@mbp15-jmb~/Perso/UCA$
```

Le résultat de cette commande est que l'invite affiche le nom de l'utilisateur, @, le nom de la machine, le répertoire de travail et **$**.

Il y a bien quelques séquences d'échappement qui peuvent être utilisés comme arguments de valeur pour PS1 ; essayez de vous limiter à la plus critique pour que l'invite ne vous accable pas d'informations.

Séquence d'échappement	La description
\t	Heure actuelle, exprimée sous la forme HH: MM: SS
\d	La date
\n	Nouvelle ligne
\s	environnement shell actuel
\W	Le dossier de travail
\w	Chemin complet du répertoire de travail
\u	Nom de l'utilisateur actuel

Séquence d'échappement	La description
\h	Nom de la machine actuelle
\#	Numéro de commande de la commande en cours. Augmente quand une nouvelle commande est entrée
\$	Si l'UID est 0 (i.e. si vous êtes connecté en tant que root), mettre à la fin le caractère # sinon utiliser $

Vous pouvez faire le changement vous-même chaque fois que vous vous connectez, ou vous pouvez avoir la modification apportée automatiquement PS1 en l'ajoutant à votre fichier .profile.

Lorsque vous exécutez une commande incomplète, le shell affiche une invite secondaire et attendez pour vous de compléter la commande et appuyez sur Entrée à nouveau.

La valeur par défaut est rapide secondaire> (le signe supérieur), mais peut être modifiée en re- définissant la variable shell PS2 :

Voici l'exemple qui utilise l'invite secondaire par défaut :

```
$ echo «c'est
> Test » ceci est un test
$
```

L'exemple ci-dessous définit à nouveau PS2 avec un message personnalisé :

```
$ PS2 = "prompt- secondaire>"
echo $ «c'est un prompt- secondaire> test » c'est
tester
$
```

Variables d'environnement

Voici la liste partielle des variables d'environnement importantes.

Ces variables sont définies et accessibles comme mentionné ci-dessous :

Variable	La description
DISPLAY	Contient l'identifiant de l'écran que les programmes X11 doivent utiliser par défaut.

Variable	La description
HOME	Indique le répertoire de l'utilisateur actuel: l'argument par défaut pour le CD intégré commande.
IFS	Indique le séparateur de champ interne qui est utilisé par l'analyseur pour la séparation de mots après l'expansion.
LANG	LANG se dilate à l'environnement local du système par défaut; LC_ALL peut être utilisé pour remplacer ce. Par exemple, si sa valeur est pt_BR, la langue est réglée sur (du Brésil) et les paramètres régionaux au Brésil.
LD_LIBRARY_PATH	Un système Unix avec un éditeur de liens dynamique, contient une liste séparée colon- des répertoires que l'éditeur de liens dynamique doit rechercher des objets partagés lors de la construction d'une image de processus après exec, avant de rechercher dans tous les autres répertoires.
PATH	Indique le chemin de recherche des commandes. Il est une liste séparée des deux-points répertoires dans lesquels le shell recherche les commandes.
PWD	Indique le répertoire de travail tel que défini par la commande cd.
RANDOM	Génère un nombre entier aléatoire entre 0 et 32767 chaque fois qu'il est fait référence.
SHLVL	Incrémente d'une à chaque fois une instance de bash est lancé. Cette variable est utile pour déterminer si la commande de sortie intégrée se termine la session en cours.
TERM	Fait référence au type d'affichage.
TZ	Fait référence à Fuseau horaire. Il peut prendre des valeurs comme GMT, AST, etc.

Variable	La description
UID	ID numérique de l'utilisateur, initialisé au démarrage du shell.

Voici un exemple montrant quelques variables d'environnement :

```
$ echo $HOME
/Volumes/Users/jmbruneau
$ echo $DISPLAY
/private/tmp/com.apple.launchd.4I2k4pzP92/org.macosforge.xquartz:0
$ echo $TERM
ansi
$ echo $PATH
/usr/local/opt/sqlite/bin:/usr/local/opt/openssl/bin:/usr/local/opt/e2fsprogs/bin:/usr/local/o
pt/e2fsprogs/sbin:/Volumes/Users/jmbruneau/.jenv/shims:/usr/local/opt/icu4c/sbin:/usr/local/op
t/icu4c/bin:/usr/local/opt/qt/bin:/usr/local/opt/mysql/bin:/usr/local/opt/postgresql@9.6/bin:/
usr/local/bin:/usr/local/sbin:/Volumes/Users/jmbruneau/bin:/Volumes/Users/jmbruneau/sbin:/usr/
bin:/bin:/usr/sbin:/sbin:/usr/local/bin:/opt/X11/bin:/usr/local/MacGPG2/bin:/Library/TeX/texbi
n:/Applications/Wireshark.app/Contents/MacOS:/usr/local/opt/payara/libexec/glassfish/bin:/usr/
local/opt/wildfly-as/libexec/bin:/usr/local/share/android-
sdk/emulator:/usr/local/share/android-sdk/tools:/usr/local/share/android-
sdk/tools/bin:/usr/local/share/android-sdk/platform-tools
```

Unix – Tubes (pipe) et filtres

Dans ce chapitre, nous discuterons en détail des « **tuyaux** » et les filtres Unix. Vous pouvez connecter deux commandes ensemble de sorte que la sortie d'un programme devient l'entrée du programme suivant. Deux commandes ou plus connectés de cette manière former un **tuyau**.

Pour faire un **tube**, placer une barre verticale (**|**) sur la ligne de commande entre deux commandes.

Lorsqu'un programme prend son entrée d'un autre programme, il effectue une opération sur cette entrée, et écrit le résultat sur la sortie standard. Il est considéré comme un filtre.

La commande « grep »

La commande **grep** recherche un ou plusieurs fichiers pour les lignes qui contiennent un certain motif. La syntaxe est :

```
$ grep motif fichier
```

Le nom « **grep** » vient de l'éditeur **ed** (un éditeur de ligne Unix) commande **:g/re/p** qui signifie « rechercher **g**lobalement les correspondances avec l'expression rationnelle (en anglais, ***r****egular* ***e****xpression*), et imprimer (***p****rint*) les lignes dans lesquelles elle correspond ».

Une **expression régulière** est soit un texte ordinaire (un mot, par exemple) et / ou des caractères spéciaux utilisés pour le filtrage.

L'utilisation la plus simple de **grep** est de chercher un modèle constitué d'un seul mot. Il peut être utilisé dans un tuyau de sorte que seules les lignes des fichiers d'entrée contenant une chaîne donnée sont envoyés à la sortie standard. Si vous ne donnez pas grep un nom de fichier à lire, il

lit son entrée standard : c'est la façon de travailler tous les programmes de filtrage :

Il existe diverses options que vous pouvez utiliser avec la commande **grep** :

Option	La description
-v	Imprime toutes les lignes qui ne correspondent pas. Modèle
-n	Imprime la ligne associée et son numéro de ligne.
-l	Imprime uniquement les noms des fichiers avec des lignes correspondant (lettre « l »)
-c	Imprime seulement le comptage des lignes correspondant.
-i	Matches soit en majuscules ou minuscules.

Recherchons dans /etc les fichiers contenant « **ip** » :

```
$ cd /etc
$ ls -lsa | grep "ip"
  1 drwxr-xr-x  3 root root        3 janv.  5  2019 ghostscr***ip***t
  9 drwxr-xr-x  2 root root        9 mars  16  2017 ***ip***route2
```

La commande « sort »

La commande de tri organise des lignes de texte par ordre alphabétique ou numériquement. L'exemple suivant trie les lignes dans le fichier alimentaire :

```
$ cat << _EOS_ | sort
> aaaa
> xxx
> eee
> _EOS_
aaaa
eee
xxx
$
```

La commande de tri organise des lignes de texte par ordre alphabétique par défaut.

Il existe de nombreuses options qui contrôlent le tri :

Option	La description
-n	Trie par ordre numérique (par exemple: 10 triera après 2), ne tient pas compte des blancs et des onglets.
-r	Renverse l'ordre de tri.
-F	Trie ensemble majuscules et minuscules.
+x	Ignore les premiers champs de x lors du tri.

Plus de deux commandes peuvent être reliés dans un « **pipe** ».

Pour prendre un exemple précédent de « **tuyau** » à l'aide grep, nous pouvons encore trier les fichiers modifiés en octobre par l'ordre de grandeur.

Le tuyau suivant comprend les commandes **ls**, grep et **sort** :

```
$ ls -l | grep 'oct' | sort +4n
-rwxr-xr-x 1 jmbruneau staff    675 oct 18  2019 arguments.sh
-rwxr-xr-x 1 jmbruneau staff    778 oct 18  2019 loops.sh
-rwx------ 1 jmbruneau staff   1384 oct 17  2019 mon_script.sh
-rw-r--r-- 1 jmbruneau staff  97989 oct 17  2019 TP_GNU-Linux-bash-1.1.pdf
-rw-r--r-- 1 jmbruneau staff 103010 oct 17  2019 TP_GNU-Linux-bash-1.2.pdf
-rw-r--r-- 1 jmbruneau staff 103246 oct 17  2019 TP_GNU-Linux-bash-1.3.pdf
-rw-r--r-- 1 jmbruneau staff 105728 oct 18  2019 TP_GNU-Linux-bash-1.4.pdf
-rw-r--r-- 1 jmbruneau staff 109658 oct 18  2019 TP1 n°4 : Shell GNU:Linux.pdf
-rw-r--r-- 1 jmbruneau staff 109685 oct 18  2019 TP_GNU-Linux-bash-1.5.pdf
-rw-r--r-- 1 jmbruneau staff 109766 oct 18  2019 TP_GNU-Linux-bash-1.6.pdf
-rw-r--r-- 1 jmbruneau staff 110528 oct 18  2019 TP_GNU-Linux-bash-1.7.pdf
-rw-r--r-- 1 jmbruneau staff 112397 oct 18  2019 TP_GNU-Linux-bash-1.8.pdf
-rw-r--r-- 1 jmbruneau staff 112636 oct 18  2019 TP_GNU-Linux-bash-1.9.pdf
$
```

Ce « **tuyau** » trie tous les fichiers dans votre répertoire modifié en octobre par l'ordre de la taille, et les imprime sur l'écran du terminal. L'option de tri +4n saute quatre champs (champs sont séparés par des blancs) puis trie les lignes dans l'ordre numérique.

La commande « more »

Supposons que vous avez une longue liste de répertoire. Pour faciliter la lecture de la liste triée, redirigez la sortie à travers plus comme suit:

L'écran se remplira une fois que l'écran est plein de texte composé de lignes triées par ordre de la taille du fichier. Au bas de l'écran s'affiche un « : », où vous pouvez taper une commande pour vous déplacer dans le texte trié.

```
$ ls -l | grep 'oct' | sort +4n | more
-rwxr-xr-x 1 jmbruneau staff     35 oct 17  2019 script1.sh
-rwxr-xr-x 1 jmbruneau staff     35 oct 17  2019 script2.sh
-rwxr-xr-x 1 jmbruneau staff     35 oct 17  2019 script3.sh
-rwxr-xr-x 1 jmbruneau staff     35 oct 17  2019 script4.sh
-rwxr-xr-x 1 jmbruneau staff     56 oct 17  2019 scp-www21.sh
-rwxr-xr-x 1 jmbruneau staff     77 oct 17  2019 sync-www21.sh
-rwxr-xr-x 1 jmbruneau staff    240 oct 18  2019 ma_variable.sh
-rwxr-xr-x 1 jmbruneau staff    396 oct 18  2019 hello.sh
-rwxr-xr-x 1 jmbruneau staff    626 oct 18  2019 controls.sh
-rwxr-xr-x 1 jmbruneau staff    675 oct 18  2019 arguments.sh
-rwxr-xr-x 1 jmbruneau staff    778 oct 18  2019 loops.sh
-rwx------ 1 jmbruneau staff   1384 oct 17  2019 mon_script.sh
-rw-r--r-- 1 jmbruneau staff  97989 oct 17  2019 TP_GNU-Linux-bash-1.1.pdf
:
```

7. Unix – Gestion des processus

Dans ce chapitre, nous abordons en détail sur la gestion des processus Unix.

Lorsque vous exécutez un programme sur votre système Unix, le système crée un environnement spécial pour ce programme. Cet environnement contient tout le nécessaire pour le système pour exécuter le programme comme si aucun autre programme en cours d'exécution étaient sur le système.

Chaque fois que vous exécutez une commande sous Unix, il créé, ou commence, un nouveau processus. Lorsque vous essayé la commande « ls » pour afficher le contenu du répertoire, vous avez commencé un processus. Un processus, en termes simples, est une instance d'un programme en cours d'exécution.

Le système d'exploitation écoutes processus par un numéro d'identification à cinq chiffres connu sous le nom **pid** ou l'**ID** de processus. Chaque processus dans le système a un **pid unique**.

Les pid peuvent éventuellement se répéter parce que tous les numéros possibles sont utilisés de manière cyclyque par le système. A tout moment, deux processus ne peuvent pas avoir le même pid car Unix utilise le pid pour suivre chaque processus.

Démarage d'un processus

Lorsque vous démarrez un processus (exécuter une commande), il y a deux manières dont vous pouvez l'exécuter -

- Processus de premier plan
- Processus d'arrière-plan

Processus de premier plan

Par défaut, tous les processus que vous commencez s'exécute au premier plan. Il reçoit son entrée à partir du clavier et envoie sa sortie à l'écran.

Vous pouvez voir cela se produire avec la commande ls. Si vous souhaitez lister tous les fichiers dans votre répertoire en cours, vous pouvez utiliser la commande suivante :

```
$ ls TP*.pdf
'TP1 n°4 : Shell GNU:Linux.pdf'   TP_GNU-Linux-bash-1.4.pdf   TP_GNU-Linux-bash-1.8.pdf
 TP_GNU-Linux-bash-1.1.pdf        TP_GNU-Linux-bash-1.5.pdf   TP_GNU-Linux-bash-1.9.pdf
 TP_GNU-Linux-bash-1.2.pdf        TP_GNU-Linux-bash-1.6.pdf
 TP_GNU-Linux-bash-1.3.pdf        TP_GNU-Linux-bash-1.7.pdf
$
```

Cela affiche tous les fichiers, dont les noms commencent par **TP** et se terminent par **.pdf**.

Le processus se déroule au premier plan, la sortie est dirigée vers mon écran.

Même si un programme est en cours au premier plan et prend du temps, aucunes autres commandes ne peuvent être exécutées (démarage de tout

autre processus), car l'invite de commande n’est disponible que lorsque la commande se termine.

Processus d'arrière-plan

Un processus d'arrière-plan fonctionne sans être connecté à votre clavier. Si le processus d'arrière-plan nécessite une entrée du clavier, il attend.

L'avantage de l'exécution d'un processus en arrière-plan est que vous pouvez exécuter d'autres commandes : vous ne devez pas attendre qu'il se termine pour lancer une autre commande.

La façon la plus simple de commencer un processus d'arrière-plan est d'ajouter une « **esperluette** » (**&**) à la fin de la commande.

```
$ ls TP*.pdf &
[1] 14392
mbp15-jmb:shells jmbruneau$ 'TP1 n°4 : Shell GNU:Linux.pdf'   TP_GNU-Linux-bash-1.4.pdf   TP_GNU-
Linux-bash-1.8.pdf
 TP_GNU-Linux-bash-1.1.pdf        TP_GNU-Linux-bash-1.5.pdf   TP_GNU-Linux-bash-1.9.pdf
 TP_GNU-Linux-bash-1.2.pdf        TP_GNU-Linux-bash-1.6.pdf
 TP_GNU-Linux-bash-1.3.pdf        TP_GNU-Linux-bash-1.7.pdf
```

Ici, si la commande **ls** veut une entrée (ce qui ne fonctionne pas), il entre dans un état d'arrêt jusqu'à ce que nous le déplacions au premier plan et qu nous lui donnions les données à partir du clavier.

Cette première ligne contient des informations sur le processus d'arrière-plan : le nombre d'emplois et l'ID de processus. Vous devez connaître le nombre d'emplois pour le manipuler entre l'arrière-plan et le premier plan.

Appuyez sur la touche Entrée et vous verrez les éléments suivants :

```
$ ls TP*.pdf &
[1] 14392
mbp15-jmb:shells jmbruneau$ 'TP1 n°4 : Shell GNU:Linux.pdf'   TP_GNU-Linux-bash-1.4.pdf   TP_GNU-
Linux-bash-1.8.pdf
 TP_GNU-Linux-bash-1.1.pdf        TP_GNU-Linux-bash-1.5.pdf   TP_GNU-Linux-bash-1.9.pdf
 TP_GNU-Linux-bash-1.2.pdf        TP_GNU-Linux-bash-1.6.pdf
 TP_GNU-Linux-bash-1.3.pdf        TP_GNU-Linux-bash-1.7.pdf

[1]+  Done                    gls --color=auto TP*.pdf
$
```

La première ligne vous indique que le processus d'arrière-plan de commande ls se termine avec succès. La dernière est une invitation pour une autre commande.

Liste des processus en cours

Il est facile de voir vos propres processus en exécutant la commande ps (état du processus) comme suit :

```
$ ps
  PID TTY          TIME CMD
  494 ttys000    0:00.02 -sh
  824 ttys010    0:00.02 -sh
  895 ttys011    0:00.02 -sh
  944 ttys012    0:00.04 -sh
10996 ttys012    0:00.03 ssh manager@51.210.215.44
  965 ttys013    0:00.04 -sh
10990 ttys015    0:00.39 ssh manager@51.210.215.44
 1024 ttys016    0:00.02 -sh
$
```

L'un des drapeaux les plus couramment utilisés pour **ps** est l'option **-f** (f pour « plein »), qui fournit plus d'informations, comme indiqué dans l'exemple suivant :

```
$ ps -f
  UID   PID  PPID   C STIME   TTY          TIME CMD
 1001   494   491   0  1:56   ttys000    0:00.02 -sh
 1001   824   811   0  1:56   ttys010    0:00.02 -sh
 1001   895   870   0  1:56   ttys011    0:00.02 -sh
 1001   944   943   0  1:56   ttys012    0:00.04 -sh
 1001 10996   944   0  8:22   ttys012    0:00.03 ssh manager@51.210.215.44
 1001   965   951   0  1:56   ttys013    0:00.04 -sh
 1001 10990  1023   0  8:22   ttys015    0:00.39 ssh manager@51.210.215.44
 1001  1024  1019   0  1:56   ttys016    0:00.02 -sh
$
```

Voici la description de tous les champs affichés par **ps** commande **–f** :

Colonne	Description
UID	ID utilisateur que ce processus appartient à (la personne exécutant)
PID	ID de processus
PPID	processus parent ID (l'ID du processus qui a commencé)
C	l'utilisation du processeur du processus
STIME	temps de démarrage du processus
TTY	type de terminal associé au processus

Colonne	Description
TIME	temps CPU pris par le processus
CMD	La commande qui a commencé ce processus

Il y a d'autres options qui peuvent être utilisées avec la commande ps :

Option	Description
-a	Affiche des informations sur tous les utilisateurs
-x	Affiche des informations sur les processus sans bornes
-u	Affiche des informations supplémentaires comme option -f
-e	Affiche des informations étendues

Arrêt d'un processus

Mettre fin à un processus peut être fait de plusieurs façons différentes. Souvent, à partir d'une commande basée sur la console, l'envoi d'une combinaison de touches **CTRL + C** (le caractère d'interruption par défaut) va quitter la commande. Cela fonctionne lorsque le processus est en cours d'exécution en mode d'avant-plan.

Si un processus est en cours en arrière-plan, vous devriez obtenir son **Job ID** en utilisant la commande « **ps** ». Après cela, vous pouvez utiliser la commande « **kill** » pour tuer le processus comme suit :

```
$ ps -f
  UID   PID  PPID   C STIME   TTY          TIME CMD
 1001   552   551   0 Sam11   ttys000    0:00.08 -sh
 1001 11570 11569   0  8:42   ttys001    0:00.05 -sh
 1001   636   634   0 Sam11   ttys002    0:00.02 -sh
 1001   646   645   0 Sam11   ttys003    0:00.02 -sh
 1001   694   689   0 Sam11   ttys004    0:00.19 -sh
$ kill 552
```

Ici, la commande kill termine le premier processus de « -**sh** ». Si un processus ne tient pas compte d'une commande régulière kill, vous pouvez utiliser « **kill -9** » suivi de l'ID de processus comme suit :

```
$ kill -9 11570
```

Processus parents et enfants

Vous pouvez modifier le piège précédent pour supprimer également les deux

Chaque processus unix possède deux numéros d'identification qui lui sont attribuées : l'ID de processus (PID) et l'identifiant de processus parent (PPID). Chaque processus utilisateur dans le système a un processus parent.

La plupart des commandes que vous exécutez ont le « shell » comme parent. Vous pouvez le vérifier le avec la commande « **ps -f** » qui liste à la fois l'ID du processus et l'ID du processus parent.

Processus « zombie » et orphelins

Normalement, quand un processus enfant est tué, le processus parent est mis à jour via un signal **SIGCHLD**. Ensuite, le parent peut faire une autre tâche ou redémarrer un nouvel enfant au besoin. Cependant, parfois le processus parent est tué avant que son enfant soit tué. Dans ce cas, le « père de tous les processus » le processus d'initialisation, devient le nouveau **PPID** (processus parent ID). Dans certains cas, ces processus sont appelés processus orphelins.

Quand un processus est tué, « **ps** » peut montrer encore le processus avec un **état Z**. C'est un **zombie** ou un **processus défunt**. Le processus est mort et ne pas être utilisé. Ces processus sont différents des processus orphelins. Ils ont terminé l'exécution, mais toujours trouver une entrée dans la table des processus.

Les processus « démons » (daemon)

Les « démons » sont des processus d'arrière-plan liés au système qui fonctionnent souvent avec les autorisations de demandes de racine et de services d'autres processus.

Un « **démon** » n'a pas de terminal de contrôle. Il ne peut pas ouvrir **/dev/TTY**. Si vous faites un « **ps -ef** » et regardez le champ **TTY**, tous les daemons auront **?** pour le téléscripteur.

Pour être précis, un démon est un processus qui fonctionne en arrière-plan, en attendant généralement que quelque chose arrive avec laquelle il soit capable de travailler. Par exemple, un démon d'impression en attente de commandes d'impression.

Si vous avez un programme qui appelle à un traitement long, il vaut la peine de faire un démon et l'exécuter en arrière-plan.

La commande « top »

La commande supérieure est un outil très utile pour montrer rapidement les processus triés par différents critères.

Il est un outil de diagnostic interactif qui se met à jour périodiquement et affiche des informations sur la mémoire physique et virtuelle, l'utilisation du processeur, la charge moyenne, et vos processus occupés, etc.

Voici la syntaxe simple pour exécuter la commande haut et de voir les statistiques de l'utilisation du processeur par différents processus :

```
$ top
```

```
Processes: 514 total, 2 running, 512 sleeping, 3370 threads                              11:16:31
Load Avg: 3.31, 2.84, 2.81  CPU usage: 4.52% user, 4.6% sys, 91.40% idle
SharedLibs: 232M resident, 62M data, 59M linkedit.
MemRegions: 176327 total, 5255M resident, 96M private, 3710M shared.
PhysMem: 16G used (2445M wired), 30M unused.
VM: 2725G vsize, 1117M framework vsize, 83757(63) swapins, 300331(0) swapouts.
Networks: packets: 499763/456M in, 324495/53M out.
Disks: 5254819/41G read, 1383896/14G written.

PID    COMMAND      %CPU TIME     #TH   #WQ  #PORT MEM    PURG   CMPRS  PGRP  PPID  STATE
15171  sshd         0.0  00:00.01 1     0    13    1796K  0B     0B     15170 15170 sleeping
15170  sshd         0.6  00:00.05 4     3    37    1652K+ 0B     0B     15170 1     sleeping
15166  top          6.5  00:01.65 1/1   0    25    6148K  0B     0B     15166 13398 running
15165  mdworker     0.0  00:00.05 3     1    51    3256K  0B     0B     15165 1     sleeping
15164  mdworker     0.0  00:00.05 3     1    51    3288K  0B     0B     15164 1     sleeping
15163  mdworker     0.0  00:00.05 3     1    51    3260K  0B     0B     15163 1     sleeping
15160  mdworker     0.0  00:00.04 3     1    45    3020K  0B     0B     15160 1     sleeping
15114  mdworker     0.0  00:00.26 4     1    59    28M    0B     0B     15114 1     sleeping
14605  QuickLookSat 0.0  00:00.33 4     1    79    4292K  0B     6760K  14605 1     sleeping
14353  Google Chrom 0.0  00:02.86 16    1    189   28M    0B     18M    403   403   sleeping
```

Job ID versus Processus ID

Les processus en arrière-plan ou en suspension sont habituellement manipulés par leur numéro de tâche (ID d'emploi).

Ce numéro est différent de l'ID de processus et il est utilisé parce qu'il est plus court.

En outre, un travail peut être constitué de plusieurs processus en cours d'exécution en série ou en même temps (en parallèle).

L'utilisation de l'ID de travail est alors plus facile que cellui des processus individuels.